This article explores some fundamental best practices when using IBM® Integration Designer, IBM Process Designer, and IBM Process Center together to create applications.

Introduction
IBM Business Process Manager Advanced offers two authoring environments. IBM Process Designer (hereafter called Process Designer) is used to model and execute high-level business processes, which often have human interactions. On the other hand, IBM Integration Designer (hereafter called Integration Designer) is leveraged to build and implement services that are completely automated or that invoke other services such as web services, enterprise resource applications, or applications running in CICS and IMS, which already exist in the enterprise. These authoring environments both interact with the Process Center, which is a shared repository and runtime environment.

There are two roles and skill sets to consider when developing business process management (BPM) applications using these environments:

- The **business author** is responsible for authoring all business processes. He or she is able to use services, but is not interested in the implementation details or how they work. The business author uses Process Designer to create business process diagrams (BPDs), and advanced integration services (AISs) to collaborate with the integration programmer.
- The **integration programmer** is responsible for doing all of the integration work necessary to support the processes the business author creates. For example, he or she will implement all the AISs, and will produce mappings between backend formats and the requirements of current applications. The integration programmer uses Integration Designer.

Sample part 1: Creating the travel approval process application

Repository Details
A major difference between the two products is that Process Designer works live from the Process Center repository artifacts, whereas Integration Designer has local workspaces with physical artifacts and files, permitting the ability to work offline. Integration Designer contributes modules, libraries and other projects to process applications or toolkits, which contain BPDs, AISs, and other artifacts provided by the business author.
We'll use a sample to illustrate some of the common best practices that should be followed when using Process Designer and Integration Designer to create an application. The travel approval sample used in this article mimics the HR department of a company that approves travel requests made by the company's employees. This sample shows the collaboration between the business author and the integration programmer and, along the way, touches upon best practices of the collaboration.

The sample starts with the business author using Process Designer to create a process application, called Travel App. The business author opens Travel App in the Designer, as shown in Figure 1.

**Figure 1. Open Travel App in Designer**

Once the process application is opened in the Designer, the business author creates a business process definition called Travel Approval and lays out the process flow. In this example, it is a simple flow where an employee enters the request, then the request is processed by an automated service for pre-approval. If the requester has the necessary clearance, the request is automatically approved. If not, the request goes through manual approval, which is a human interaction. The user is notified about the result at the end of all decision-making.

The automated processing of the request is beyond the business author's expertise, and will be provided by an integration programmer. Figure 2 shows the sample process.

**Figure 2. Sample Travel Approval process**

**Canonical models**

Sometimes the integration programmer needs to deal with a canonical data model in the enterprise. In this case, the business author must take this into consideration in order to reduce the likelihood of rework later in the iterative development process.
The business author knows the data that he or she wants to flow between the activities in the process and creates a data type to capture it, as shown in Figure 3. The business author continues to set the activity types and set the input/output data on each activity. The business author creates coaches for the human activities and defines an AIS for the automated processing that will be provided by the integration programmer. The process application in this state (Travel_Appv3.0.twx) is provided for you to download and import.

**Figure 3. The TravelRequest data type**

![TravelRequest Data Type](image)

**Best practice: Collaborate before defining an AIS**

Advanced integration services (AISs) may exist in both process applications and toolkits. The implementation of an AIS must coexist in the same process application or toolkit as its definition. Thus, where such a definition is made is very important for reuse and organization considerations. For example, if the business author is simply thinking that he or she needs a particular service implemented by an integration programmer, it may be tempting to just define the AIS in the process application or toolkit in which he or she is working. However, more care should be taken. The business author should contact the integration programmer to confirm where the AIS should be defined, and thus implemented. It is likely that the integration programmer wants to have the AIS shared among many process applications, in which case the business author should define the AIS in a toolkit. The integration programmer may also want to group similar AISs in a specific toolkit. For these reasons, the integration programmer should be consulted on the structure before the business author creates and defines a new AIS.

**Sample part 2: Bring the process application into the workspace**

After the business author has finished authoring his or her parts of Travel App, he or she notifies the integration programmer that an AIS is ready to be implemented. The integration programmer
Best practices when using IBM Integration Designer and IBM Process Designer together

opens Integration Designer, and goes to the Process Center console if it’s not already open. The integration programmer opens the Travel App process application as shown in Figure 4.

Figure 4. Open Travel App application in workspace

Best practice: Connect to only one Process Center from an Integration Designer workspace

Connections to Process Center servers from Integration Designer are automatically established according to the projects contained in the workspace. It’s possible to have process applications and toolkits from different Process Centers in the same Integration Designer workspace. This can happen when you:

- Import project interchange files
- Import projects from a source control repository, such as CVS or Rational Team Concert
- Change the Process Center URL and credentials in the Process Center preference page after opening a process application or toolkit

Integration Designer will continue to operate normally when the workspace is connected to multiple Process Centers. However, you should avoid this because there is only one primary Process Center for an Integration Designer workspace. This Process Center is the one captured in the Process Center preference page and is the one that is shown in the Process Center perspective.

Some actions on process applications and toolkits are only possible in the Process Center perspective, including:

- Opening process applications and toolkits in Integration Designer workspace
- Creating new tracks
- Managing snapshots
- Exporting process applications and toolkits as .twx files
- Managing permissions

Using one workspace to connect to many Process Centers results in cumbersome user interactions, because the integration programmer will need to go to the preference page and change the URL and credentials in order to use the previously stated actions in the Process Center perspective. Having process applications and toolkits in the workspace from different
Process Centers is also likely to lead to confusion, as dependencies can't be made across Process Center servers. For example, a process application in one server can't depend on a toolkit in a different server.

**Best practice: Have only what you need in Integration Designer**

A process application or toolkit that can be opened by Integration Designer needs to be initialized by contributing a default module and a default library. Initialization is done automatically only once, by the first integration programmer that opens that process application or toolkit in his or her workspace. However, it's important not to initialize the dependent toolkits unless you know they will have some Integration Designer contributions (for example, call a BPD, or provide implementation for an AIS or attached monitor model). It's important not to initialize unnecessary toolkits because, upon publishing a newly initialized toolkit, the toolkit is considered changed and thus a snapshot will need to be created, which means that dependencies from process applications and toolkits on this toolkit will have to be updated to the new snapshot level. You can avoid initializing a toolkit by deselecting it from in the dialog that appears after opening the process application or toolkit in the workspace. Note that you can uninitialize a toolkit by disassociating all projects. For more information about disassociating projects, see Disassociating a module or library from a process application or toolkit in the IBM Business Process Manager V7.5 Information Center.

**Renaming projects**

To rename a project that is contributed to a process application or toolkit, you must first disassociate the project, then refactor the project name and re-associate it with the Process Center.

Another consideration is that as a process application or toolkit grows over time, it is likely that it will contain multiple modules and libraries. When a process application or toolkit is opened in Integration Designer, you can select to only bring in the subset of modules and libraries that are relevant to your current work. This reduces build time and prevents unnecessary network operations between the Integration Designer and the Process Center.

**Sample part 3: Adding an existing service**

The integration programmer opens the Travel App process application in Integration Designer, sees the AIS defined in the Business Integration view, and notices that it is marked as unimplemented, as shown in Figure 5.

**Figure 5. AutoApprove AIS unimplemented**
When thinking about the possible implementation, the integration programmer remembers that he or she has an existing module that contains a service that can be used, with some additional logic. The automatic approval will be partially based off of whether the employee is an executive or not. The employee number needs to be looked up in a backend system, which is exactly what the existing module does. There are lots of other services in this module too, but the integration programmer doesn't want the business authors to see them.

The integration programmer starts by adding the module and library to Process Center by using a new toolkit to contain them. The integration programmer does this for reuse and to limit the visibility of the artifacts to the business authors. The toolkit will be set up such that business authors have no read or write access, leaving the toolkit under the control of the integration programmer. In addition, the integration programmer decides that exposing the service in its current form is not a good idea because the interfaces are not understandable to the business author, and may be too complex.

The integration programmer brings the existing module and corresponding library into Integration Designer via a project interchange file (ExistingIIDService.zip, provided for download). The integration programmer switches to the Process Center console, creates a new toolkit called EmployeeServImpl and opens it in Integration Designer. He or she then proceeds to add both the module and library to the toolkit as shown in Figure 6.
Once the associations are made, the integration programmer creates a snapshot of the toolkit in Process Center and also changes the permissions on the EmployeeServImpl toolkit so that only people in his or her role can view and edit the toolkit.

**Best practice: Protect mirrored artifacts in toolkits**

Data types are one kind of artifact shared between Process Designer and Integration Designer. Business authors and integration programmers can make changes to these artifacts, and those changes will be visible to both of them. In order to support this capability, data types are stored in two forms in Process Center: a Process Designer format (proprietary) and an Integration Designer format (XSD files). After a change is made to a data type, the artifact is updated in both formats using a translation mechanism that keeps them synchronized. When an integration programmer publishes resources to Process Center, the translation occurs and modifies, creates, and deletes the artifacts in the Process Designer format. The business author sees the changes after the publish operation is complete. Conversely, if a business author makes a change to a data type, the translation occurs immediately after the creation, deletion, or the editor being saved. An integration programmer sees these changes the next time he or she refreshes the workspace.

It's important to note that the formats have different characteristics and different fidelity for capturing the details that might be present in the other format. The Integration Designer format
(XSD) contains structures that can't be represented in the Process Designer format. Hence, if a data type is authored in Integration Designer, published to Process Center, and modified in Process Designer, the original data type format in Integration Designer may be significantly different, with a possible loss of modeled information.

For data types that originate in Integration Designer, either completely authored or imported, it is desirable to protect them from the loss of modeled information. Process Center provides the ability to set user permissions on process applications and toolkits. Therefore, in order to protect the Integration Designer data types, you can place them inside a library, and associate the library with a toolkit with permissions that prevent business authors from making changes. In the Process Center perspective, the permissions can be changed such that business authors are given only read access, not write access. With this technique, integration programmers are free to modify the data types, and business authors will see the updated data types in Process Designer. However, the business authors won't be able to change the data types in Process Designer, avoiding any translation that may result in loss of modeled information.

Another way of protecting data types is to not mirror them at all. If the data types are not referenced directly or indirectly by an AIS implementation, then the library associated with the process application or toolkit can be left as unmarked for mirroring. The default library is always mirrored, meaning that the artifacts are translated to the Process Designer artifact domain. All other libraries are not marked as mirrored by default, as seen in the properties dialog for the library, meaning that the artifacts will never be seen in Process Designer. A library should only be marked as mirrored when necessary. With this technique, the business author won't see the data types at all, so this method can't be used if the data types are to be used by the business author.

Using either approach greatly reduces the likelihood of data types being modified by a business author in Process Designer, causing a lower fidelity model in Integration Designer.

**Best practice: Facade from "edge" interfaces and data types into the Integration Designer domain**

As discussed previously, data types are one of the main kinds of shared artifacts between Process Designer and Integration Designer. After one tool changes a data type, the definition of the data type in the other format is updated with the changes.

Because data types are likely to be used by many other artifacts in both Integration Design and Process Designer, changes made to the types through one tool are likely to break artifacts defined in the other tool. A good pattern to avoid excessive or accidental breakages is to "facade" the data types and the interfaces, isolating the models from changes introduced through the other tool. For example, suppose a given data type is referenced by an input variable to a business process definition in Process Designer. Further, suppose that the same data type is used in a BPEL process, some Java™ snippets, and in a state machine in Integration Designer. If the business author decides to delete the type, or move it to another toolkit, then the BPEL process, Java snippets, and state machine in Integration Designer will all be broken. However, if a facade pattern was used, these artifacts would not be broken at all. Instead, only the facade logic, such as a data map, would be broken. Since the breakage does not occur within the business logic of the
BPEL, the snippets, or the state machine, it is easier and less error prone to adjust the facade. The facade pattern is shown in detail in part four of the sample.

Sample part 4: Creating a facade service

Now that the integration programmer has the existing service associated with a toolkit, he or she could decide to either add a dependency directly from Travel App to the EmployeeServlmpl toolkit, or he or she may decide to create a more reusable service for business authors to use in the future. The integration programmer decides to do the latter, so he or she creates another toolkit, EmployeeServices, to contain the public service interface he or she will surface for the business authors. A dependency is made from EmployeeServices to the snapshot of EmployeeServlmpl and the integration programmer begins to implement the facade pattern by creating interfaces and data types in EmployeeServices that represent a subset or simplification of those in EmployeeServlmpl. For example, there are three operations on the ExecutiveServices interface, yet the integration programmer only wants to surface one of them. Likewise, the integration programmer doesn't want business authors to see the AddressServices or PayrollQueries services that also exist. To complete the facade pattern, the integration programmer implements a simple mediation flow in EmployeeServices to mediate from the public interfaces and data types to the existing service. Figures 7 and 8 show the assembly diagram and mediation flow implementation.

Figure 7. Assembly diagram

![Assembly diagram](image)

Figure 8. Mediation flow

![Mediation flow](image)

Next the integration programmer creates a snapshot of EmployeeServices and creates a dependency from Travel App to EmployeeServices. The integration programmer implements the AIS, using the facade service. In the sample, a BPEL process is used for the AIS implementation. Note that the business author is free to modify the artifacts in EmployeeServices and the underlying service in EmployeeServlmpl will not be affected. The twx file for the two new toolkits is EmployeeServicesv1.0.twx (provided for download). At this point, the entire application is as shown in Figure 9. The completed application is available in Travel_Appv4.0.twx (provided for download).
Best practice: Work on a process application and its dependent toolkit at the same time

Tips, tracks, and snapshots
A track in Process Center is similar to branches or streams in other repository systems. One or more tracks may exist for a given process application or toolkit. The latest content of a track is called the tip, and all changes must be made on the tip. At any point in time, a snapshot can be made to capture the current state of a track. Snapshots are immutable, and new tracks can only be made from snapshots.

Unlike Process Designer, Integration Designer allows working on multiple process applications and toolkits at the same time. The integration programmer can open multiple process applications or toolkits in the workspace. Dependent toolkits may also be brought into a workspace automatically when a process application is opened. Since a process application can only depend on a particular snapshot of a toolkit and the snapshots are read only, the integration programmer who intends to make changes to both should ensure he or she is working with the "tip" of each track in the workspace. Integration Designer monitors when a change is being made to the snapshot of a process application or toolkit. If a process application or toolkit is changed, but the tip of the track is not in the workspace, a dialog is presented informing the integration programmer that the changes cannot be saved to the Process Center. This dialog is a hint that the tip may need to be brought in if the changes are intended to be eventually committed.

Private changes
Process applications and toolkits are shown as grayed out in the business integration view if you only have read only access to them, they are snapshots, or they are marked as immutable in Process Center. Even though you can change the projects in Integration Designer, you won't be able to publish the changes to Process Center.

When the integration programmer intends to make changes to a process application and its dependent toolkits, he or she has to open the tips of the tracks in the workspace. If a snapshot of a dependent toolkit is in the workspace, it can be replaced with the latest from the tip using the context menu. Only toolkits that are being modified should have the track tip loaded in the
workspace. When changes are published to Process Center, Integration Designer recognizes that a dependent toolkit is in the workspace and that the tip of its track is present. As a result, a new snapshot of the toolkit is created and all dependencies are updated with the new snapshot value. Even if no changes have been made to the tip, a dialog still appears. This step is necessary to ensure consistency of the dependency model across Process Center and the Integration Designer workspace. In Process Center, a dependency on a toolkit is always specified using a snapshot of the toolkit. It’s not possible to have a dependency on the tip of a toolkit track. An example of the dialog presented is shown in Illustration 10, which would be the result of publishing Travel App with the branch tips loaded in the workspace of the other two toolkits in the sample.

**Figure 10. Create snapshots and update dependencies dialog**

Another situation to consider is that, during the creation of AISs, more than one integration programmer might be collaborating to implement the complete set of AISs required by the business author. In this case, creating a new toolkit snapshot for each publish is unnecessary and will create unneeded snapshots. In this case, a snapshot is needed only after the integration programmers have completed their work. To continue iterative development without creating new snapshots and updating dependencies, you can uncheck **Create snapshots and update dependencies** in the dialog shown in Figure 10.

**Best practice: Check for breakages in both tools**

If a process application or toolkit contains content authored in both Process Designer and Integration Designer, you must take extra caution to avoid unintended or unnoticed breakages.
The set of artifacts visible to a business author is different from that seen by the integration programmer. For example, a mediation flow authored in Integration Designer will never be seen by a business author. Conversely, an undercover agent authored in Process Designer will not be seen by an integration programmer. However, some artifacts are visible to both roles, such as data types, business process definitions, and advanced integration services. In addition, there are dependencies between process applications and toolkits that enable artifact visibility across project dependencies. For example, a project dependency in Integration Designer must be resolvable in the scope of the process application or toolkit. If a module in a process application depends on a library, the library must be in the same process application or in a dependent toolkit. Integration programmers and business authors can change these artifacts or dependencies, causing an adverse impact to artifact visibility that is seen only by the other role.

In order to ensure that an entire process application or toolkit is error-free, you should open and analyze it in both tools. Using a new workspace in Integration Designer, you should select all modules and libraries when opening the process application or toolkit. Once the build is complete, you can see any errors in the Problems view.

**Sample part 5: Dealing with a breakage in the process application**

The business author decides that he or she wants to change the interface contract slightly. He or she goes back into Process Designer and modifies the data type by deleting one of the fields. The business author lets the integration programmer know that changes have been made in the GeneralEmployee data type in the toolkit with the public services, EmployeeServices. At this point, the integration programmer opens Integration Designer and opens the EmployeeServices toolkit. After the build is complete, the integration programmer can see there is a problem. There is a field that was being used in a map that is now incomplete, so an adjustment needs to be made. Figure 11 shows the broken map and problem marker. One course of action, for example, would be for the integration programmer to delete the bottom Move node in the map, as shown in Figure 11.

**Figure 11. Broken data map**

---

**Best practice: Manage dependencies in Integration Developer**

Many mirrored libraries in Integration Designer may be associated with one process application or toolkit. The shared data types within these mirrored libraries are flattened in one common area in
Process Designer within the given process application or toolkit. Because of this difference, you must take special care when multiple mirrored libraries are being used in one process application or toolkit.

The first issue that arises is that Integration Designer manages the dependencies between the default libraries according to toolkit dependencies, but dependencies to and from libraries explicitly associated by integration programmers are not automatically managed. For example, suppose a process application has two mirrored libraries with a data type defined in each one. There are no dependencies between the XSDs or libraries yet. With the above artifacts in the Integration Designer domain, the business author would see a flattened structure, with the data types under the process application. The artifacts are shown in Figure 12.

**Figure 12. No references or dependencies**

![Diagram showing no references or dependencies between libraries in Integration Designer](image)

Suppose the business author adds a parameter (or field in Integration Designer jargon) in the Customer type, which has the type AccountDetails. However in Integration Designer, these artifacts are contained in separate libraries. When the integration programmer invokes **Refresh from Process Center**, validation error markers will exist if MyProcessApp.Library does not depend on OtherMirroredLibrary. You need to open the dependency editor of MyProcessApp.Library, and manually add a dependency to OtherMirroredLibrary. Figure 13 shows the state of the artifacts after this change.
The second issue is the ease with which circular dependencies between libraries can be introduced. Continuing with the example, suppose the business author wants to make a small change to AccountDetails. He or she creates a new type, Address, and adds a parameter in AccountDetails that uses it as a type. The problem is that back in the Integration Designer domain, there would need to be a cycle in the project dependencies to handle the dependency chain. Types created in Process Designer are always mirrored to the default library of a process application or toolkit. Hence, Address.xsd is created in MyProcessApp_Library. MyProcessApp_Library already has a dependency on OtherMirroredLibrary because Customer uses AccountDetails, but the new parameter added to AccountDetails requires a dependency from OtherMirroredLibrary (AccountDetails) to MyProcessApp_Library (Address). Circular dependencies are not allowed within Integration Designer projects, so the Process Application will be in a broken state. Figure 14 shows the artifacts and necessary dependencies.

To work around this issue, the integration programmer should move one of the XSD files, such as moving AccountDetails.xsd to MyProcessApp_Library.

**Conclusion**

Using the best practices described in this article, you can avoid many pitfalls when developing applications that span both Integration Designer and Process Designer. By understanding how
each user role leverages the shared Process Center repository and the effects of having a shared intersection of the domain models, you can create better development processes. Breakages in models can be caught earlier, and patterns can be used to minimize the breakages in those models.

**Acknowledgments**

The author would like to thank David Booz and John Green for their contributions as reviewers of this article.
## Downloadable resources

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Files for importing to Process Center</td>
<td>Travel apps.zip</td>
<td>2209KB</td>
</tr>
<tr>
<td>File for importing to Integration Designer</td>
<td>ExistingIIDService.zip</td>
<td>21KB</td>
</tr>
</tbody>
</table>
Related topics

- Disassociating a module or library from a process application or toolkit in the IBM Business Process Manager V7.5 Information Center.
- developerWorks BPM zone: Get the latest technical resources on IBM BPM solutions, including downloads, demos, articles, tutorials, events, webcasts, and more.
- IBM BPM Journal: Get the latest articles and columns on BPM solutions in this quarterly journal, also available in both Kindle and PDF versions.

© Copyright IBM Corporation 2011
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)