SAP Exchange Infrastructure: Guide for Customer Developments and Modifications in the Integration Repository

Release SAP XI 3.0

Copyright

Icons in Body Text

Icon	Meaning
	Caution
	Example
	Note
	Recommendation
	Syntax

Additional icons are used in SAP Library documentation to help you identify different types of information at a glance. For more information, see *Help on Help → General Information Classes and Information Classes for Business Information Warehouse* on the first page of any version of *SAP Library*.

Typographic Conventions

Type Style	Description
Example text	Words or characters quoted from the screen. These include field names, screen titles, pushbuttons labe ls, menu names, menu paths, and menu options. Cross-references to other documentation.
Example text	Emphasized words or phrases in body text, graphic titles, and table titles.
EXAMPLE TEXT	Technical names of system objects. These include report names, program names, transaction codes, table names, and key concepts of a programming language when they are surrounded by body text, for example, SELECT and INCLUDE.
`Example text`	Output on the screen. This includes file and directory names and their paths, messages, names of variables and parameters, source text, and names of installation, upgrade and database tools.
`Example text`	Exact user entry. These are words or characters that you enter in the system exactly as they appear in the documentation.
`<Example text>`	Variable user entry. Angle brackets indicate that you replace these words and characters with appropriate entries to make entries in the system.
`EXAMPLE TEXT`	Keys on the keyboard, for example, `F2` or `ENTER`.

1 Introduction

This guide contains recommendations for customers and consultants on how to implement their own developments in the Integration Repository and by using ABAP proxy generation (Release SAP XI 3.0). These developments can also include enhancements and modifications to SAP objects. This guide also explains the enhancement concept introduced in SAP XI 3.0 for enhancing data types without making modifications. You should make enhancements without modifications rather than modifications as far as is possible.

This guide covers the following areas:

- Software component versions and namespaces (structure recommendations, and so on)
- Interface development and proxy generation (new customer developments, reusing SAP definitions, enhancing SAP definitions, modifying SAP definitions)
- Developing integration scenarios, integration processes, and mappings
- Steps required following a customer development before configuration in the Integration Directory

Note that advanced knowledge of SAP XI 3.0 is necessary to understand this document. For more information about SAP Exchange Infrastructure, see SAP Service Marketplace at service.sap.com/xi.

2 Development Process in the Integration Builder

The structure of this document is based around the steps required during the development process to create objects in software components in the Integration Builder.

The most important steps of a development process are listed below. The steps are described in detail in the following sections.

1. **Creating software component versions in the System Landscape Directory**

   Before you can create objects in the Integration Repository, you must define the software component versions to which they will be assigned in the System Landscape Directory (SLD). The software component versions are assigned to product versions in the SLD. In the SLD, you can also define a software component version as being based on other software component versions. This enables you to define dependencies between software components.

2. **Importing software component versions to the Integration Repository**

   You import software component versions from the SLD to the Integration Repository. The “based-on” relationships are also transferred and displayed in the Integration Repository.

3. **Defining namespaces**

   Namespaces are assigned to software component versions in the Integration Repository. Customers and partners should use their own software component versions and their own namespaces.

4. **Defining objects in the Integration Repository**

   Once you have defined software component versions and namespaces, you can create objects in the Integration Repository: integration scenarios and integration processes, interface objects (including data types and data type enhancements), and mapping objects.

5. **Activating objects**

   When you save new or modified objects, these are added to change lists in the Integration Repository. Activating these change lists activates all the objects they contain.

6. **Generating proxy objects**

   An ABAP application also requires corresponding ABAP proxies for the interface objects in the Integration Repository, so that it can use the interfaces in its implementation. ABAP proxy generation is initiated in the corresponding ABAP system. The resulting ABAP development objects (ABAP-OO-classes/-interfaces, ABAP Dictionary objects) are transferred in ABAP transport requests.

7. **Transporting objects**

   You must synchronize the transport of the generated ABAP proxy objects with the transport of the application objects (for example, programs, function modules, classes) used by the proxies. You execute the transport in the system landscape (development, test, productive system) using the ABAP transport infrastructure (Change and Transport System (CTS)).
   If the customer or partner has a similar landscape for the Integration Repository with a development, (test), and productive repository, you must also transport the objects of the Integration Repository through this landscape. To do this, you export the change lists from the development repository and import them to the target repositories.
   Make sure that you synchronize the Integration Repository transports and the ABAP transports to ensure that proxies are only generated in the Integration Repository using consistent end statuses of object definitions.

3 Software Component Versions and Namespaces

3.1 Basics

A development in the Integration Repository always takes place within a software component version. You define software components and their versions and products and product versions in the System Landscape Directory (SLD). The steps required for this definition are described in 3.2.

A **product** in the SLD represents a collection of all versions of a product. A product is a unit that can be shipped, is visible to the customer, and can be installed and upgraded.
A **product version** represents a particular version of a product.
A **software component** represents a collection of all versions of a software component. Software components represent the reusable modules of a product. They can be individually upgraded or have patches installed.
A **software component version** represents a particular version of a software component.

Customer developments must not be executed in an SAP component but in a customer software component version. This avoids conflicts with subsequent SAP software updates. You can also make modifications to SAP objects in the customer software component version by using a set procedure. You must also make any enhancements to SAP data types without modifications in the customer software component version.

Do not make modifications in the SAP software component version, since they will be immediately overwritten when SAP software updates are imported.

3.1.1 Software Component Versions

You use a software component version to group together objects in the Integration Repository that are shipped or installed together. At configuration time, the Integration Builder can then, together with information from the System Landscape Directory, determine which interfaces a service, in particular a business system, supports. This prevents incorrect or unnecessary routing rules.

To do this, you define which product versions (and thus software component versions) are available in a business system in the System Landscape Directory. Together with the information from the Integration Repository, this makes it possible to determine which interfaces this system supports. For example, by using this information the Integration Builder only shows the relevant interfaces in the search help for interfaces in the Integration Directory.

If you want to develop interfaces in the ABAP environment in one development project and in the Java environment in another development project, we recommend that you create two different software component versions. This is because the ABAP components do not support the Java interfaces and the other way around.

3.1.2 The “Based-On” Relationship

An existing software component version is often a prerequisite for a software component version because, for example, part of the software has to be reused. In this context we refer to *underlying* software components, that is, components that have to be available so that the component based on them is complete.

You define the “based-on” relationship in the System Landscape Directory. In the Integration Repository, the underlying software components are displayed in the navigation tree in the *Basis Objects* subtree of the corresponding “based-on” component.

ABAP component version SAP_ABA 6.20 is a prerequisite for ABAP component version BBPCRM 4.0 because classes, function modules, and types from the ABAP Dictionary of these basis components are used in CRM. The underlying component is therefore SAP_ABA 6.20 and the component based on it is BBPCRM 4.0. You can also say that BBPCRM 4.0 *is based on* SAP_ABA 6.20.

In the navigation tree in the Integration Repository the software component version SAP EBP SRM Server 4.0 references the objects from ABA from the namespaces http://sap.com/xi/ABA and http://sap.com/xi/ABA/Global as *Basis Objects*.

This means that objects in the Integration Repository can reference other objects from the same and also from underlying software component versions.

Data type definitions in BBPCRM 4.0 can reference data types not only from BBPCRM 4.0 but also from SAP_ABA 6.20. Data type enhancements in BBPCRM 4.0 can enhance data types from BBPCRM 4.0 and also from SAP_ABA 6.20.

You can define multilevel “based-on” relationships.
The objects of all the underlying software component versions are always visible in a software component version.

In a customer development, the “based-on” relationship is relevant in the following cases:

- The customer development is based on a particular SAP component from which objects are to be reused.
- The customer development involves enhancement or modification of SAP objects.

You want to reuse objects from the SAP component SCM 4.0 in the customer component CUSTOMER_SCM 1.0. However, in the customer component CUSTOMER_**CRM** 1.0 you want to modify the BBPCRM 4.0 component. You must **not** just define a single component CUSTOMER_SCR_CRM based on SCM and CRM, as there is no ABAP component in which both SCM and CRM are available. Such definition would cause problems if you wanted to generate a proxy, since this would require objects from the underlying components to be already available.

3.1.3 Releases of a Customer Development

Although it may not be relevant at the start of development, it may become necessary in the life cycle of a customer development to introduce releases (so far we have only looked at CUSTOMER_SCM 1.0 and CUSTOMER_CRM 1.0).

Reasons for a new release may include:

- Extensive new developments
- Shipment of a new release of the underlying SAP component

In both cases you have the following options:

- Update the existing software component version (that is, make new developments in the current release or adapt the “based-on” relationship). Choose this option if you do not need to keep a version of the current development status in the system group or in the Integration Repository.
- Create a new software component version incorporating the new development or the new “based-on” relationship. This solution is advisable if you have different system releases in the system group (these have to be able to be described correctly).

If you choose the second option, give your customer development a new release ID when you create the new software component version (keep the component name the same, but change the release ID, for example `2.0` instead of `1.0`). The initial content transfer into the new release is implemented by way of a *release transfer* (in the Integration Builder under *Tools*). This enables you to transfer all objects from the old software component version into the new version.

3.1.4 Namespaces

(Technical) namespaces are used to identify objects within a software component version. They allow you to define namespaces for subdivisions within a software component version, for example to differentiate between subprojects.

You should define a namespace so that it is as unique as possible worldwide. Just as the company name is used in the namespace in SAP interfaces (for example, `http://sap.com/xi/CRM`), you should likewise include the company name in the namespace in customer developments. An additional restriction is that the same namespace must not appear simultaneously in several independent software component versions.

The namespace `http://customer.com/SCM` is assigned to the software component version `CUSTOMER_SCM 1.0`, whereas `http://customer.com/CRM` is used for `CUSTOMER_CRM 1.0`. **Do not** use the namespace `http://customer.com` in both components. This can result in objects with the same namespace `http://customer.com` having conflicting definitions (for example, material data types with different definitions or message interfaces with the same name but different functions). If definitions are not unique, this will cause problems in proxy generation and routing. On the other hand, if a namespace is used by only one software component, there can be no ambiguity, since an object name in a namespace of a software component can only be assigned once.

An intentional exception to this restriction is that a namespace can by all means be used for different versions of a software component. Thus, `http://sap.com/xi/CRM` can be used in `CUSTOMER_CRM 1.0` and in `CUSTOMER_CRM 2.0`, because version 2.0 results from the release transfer of the “old content” from release 1.0. In `CUSTOMER_CRM 2.0` the transferred objects will be developed further or new objects will be added in the same namespace. Therefore, the versions are closely related semantically and are evolutions of each other (that are usually compatible).

Comment:

The technical namespaces that you use to assign objects in the navigation tree in the Integration Repository and to name objects uniquely are not the same as the semantic XML namespaces, which you can also specify when defining message types and data type enhancements. The semantic XML namespaces are exclusively for use as namespaces in the XML payload.

The technical namespaces are also used as default namespaces for the XML payload (in message types and data type enhancements) if they are not expressly changed in the ‘XML Namespace’ input field.

It is advisable to change the XML namespaces if, for example, you want to qualify enhancement fields of data types of two different components (such as CRM and SCM) that are to exchange messages with each other with the same namespace prefix on both sides (to avoid unnecessary mapping or validation problems). In this case, both sides must agree on the same XML namespace. The technical namespace is usually different from component to component. In CRM and SCM, for example, the technical namespaces are `http://sap.com/xi/CRM` and `http://sap.com/xi/SCM` respectively.

3.1.5 Software Component Versions in the Integration Directory

A business system is often used as a *Service Without Party* as the starting point for limiting the amount of data to be selected when accessing information from the Integration Repository in the Integration Directory.

Example

You are creating a receiver determination in the Integration Directory. First of all, specify a business system as the service. The corresponding input help should list all the (outbound) interfaces available in this business system. To do this, the Integration Builder determines which product versions (and thus software component versions) are installed in this business system. The available interfaces for these software component versions can then be determined in the Integration Repository.

To ensure that the interfaces from the customer component also appear in this input help, you need to specify in the System Landscape Directory that the customer component is installed in this business system.

For more information, see section 8.

3.1.6 Software Component Versions in ABAP Proxy Generation

In ABAP proxy generation (transaction SPROXY), the navigation tree displays the software component versions from the Integration Repository that are relevant for the development system. The software component versions that are relevant for this system are determined as follows:

1. The software component versions in the Integration Repository are identified.
2. The system determines which SAP software component versions (where `sap.com` is defined as the vendor) are installed in this system. To determine the installed software component versions in the SAP system, choose *System* → *Status* in the SAP menu. In the *System: Status* dialog box, choose *Component Information* under *Component Version*. These entries correspond to the entries in the CVERS table.
3. Since it is not possible to determine which customer components are active in this system in the SAP system, the system displays all components where the vendor is not `sap.com`.

Proxy generation then generates a corresponding counterpart in ABAP for an object definition from the Integration Repository.

Generated Proxy Objects

Interface Object in the Integration Repository	Proxy Object in the System
Message interface (outbound)	ABAP object class
Message interface (inbound)	ABAP object interface
Message Type	ABAP Dictionary structure
Fault message type	ABAP object exception class
Data type	ABAP Dictionary structure(s)/ ABAP Dictionary data element
Data type enhancement	ABAP Dictionary append structure

The generated objects are now reused, that is, there is an exact counterpart in the system (ABAP) for the object in the Integration Repository.

A SAP system only represents a particular version status of a software component version, and never more that one at one time.

`BBPCRM 4.0` is available in one business system, `BBPCRM 3.0` is available in another. However, there is no system where both `BBPCRM 3.0` and `BBPCRM 4.0` are installed.

Proxy objects are therefore only available in a single version status in an SAP system. Therefore, a proxy object only notes that it is the counterpart to an object, which is identified by name and namespace (the version is implicit from the system context). This behavior conforms to the syntax of WSDL and XSD (an object is identified by name and namespace) and also corresponds to the procedures of other proxy generation tools.

This has the following consequence:

If you want to use a namespace in more than one software component version, you can create objects with the same name and namespace in the different software component versions. If these objects are made available for proxy generation one after the other, they are mapped to the same ABAP object using these navigation paths (since a proxy object only notes the name and namespace).

The `Address` object is defined in the namespace `http://abc` in software component version `X` and a corresponding proxy object is generated in proxy generation (transaction `SPROXY`). This creates an ABAP Dictionary structure with the name `ADDRESS`. A software component version `Y` with the same namespace and the same object now also appears in the `SPROXY` transaction. If you navigate to the proxy object by this path and generate the corresponding proxy object, the definition from software component version `Y` overwrites the existing ABAP Dictionary structure `ADDRESS`.

Note that the behavior in the example is a desired effect in the case of a modification. The modified definition from the Integration Repository is used to overwrite the original SAP object on the proxy side (see also section 4, *Customer Development of Interfaces*). However, if there is more than one version of a customer component, all versions in the navigation tree are offered to proxy generation and you have to navigate to the correct one.

If a software component version contains an underlying software component version defined by the “based-on” relationship (see3.2.2), the navigation tree in proxy generation only displays the namespaces of the underlying software component version that contain modified objects.

3.1.7 Prerequisites for Packages in ABAP Proxy Generation

In ABAP proxy generation, you must specify the package in which the generated ABAP proxy objects are to be saved.

The packages used by the proxy objects must have a use access for the package interface `SAI_TOOLS`. The corresponding structure package must also contain a use access for the package interface `SAPPINT`.

A use access must exist for the package interface `SAI_SXMS` for the proxy runtime.

You create packages and define their properties in the Package Builder (transaction `SE21` or `SPACKAGE`).

3.2 Software Components in the System Landscape Directory

You have to create software component versions in the System Landscape Directory before you can work with them in SAP Exchange Infrastructure. You must assign them to a product in the SLD. Customers have to create their own product and their own software component version for their own development.

3.2.1 Creating Products and Software Components

1. Call the System Landscape Directory and choose *Software Catalog*.
2. Choose *New Product…* and enter a product definition:
   - Vendor: Enter an address other than `sap.com`, for example `CustXY.com`.
   - Name: Product name, for example `CUSTXY`.

- Version: Version number, for example `1`.

3. Choose *Create* and add a software component to your product:
   - Product: `CUSTXY`, `1` of `CustXY.com`
   - Vendor: `CustXY.com`
   - Name: for example `CUSTXY`
   - Version: for example `1`
4. Choose *Create* again.

You have created a product with a software component (to be more exact, a product version with a software component version).

3.2.2 Defining the “Based-On” Relationship

If you want to modify or use objects from an SAP component, you also have to define the fact that the customer component is based on the SAP component.

1. On the System Landscape Directory initial screen, choose *Software Catalog*.
2. In the *Versions* column of the displayed list, select the corresponding customer software component version (for example, `CUSTXY, 1 of CustXY.com`) by navigating to the customer component version and clicking it.

   

   If you cannot find your software component version in the list (because the list is too long, for example), use a *Filter* for your selection.
3. Click the *Usage Dependencies* link.
4. Under *Dependency Context*, choose *InstallationTime* and choose *Define Dependencies*.
5. If necessary, select a filter to get a list of candidates for the underlying software component version. In the list, select the SAP software component version(s) upon which you want to base your customer component.
6. Choose *Create*.

3.3 Software Component Versions in the Integration Repository

3.3.1 Importing a Software Component Version into the Integration Repository

The System Landscape Directory contains a number of software component versions, but they are not all relevant to XI, so the first step is to transfer the XI-relevant software component versions to the Integration Repository.

1. Call the Integration Builder (Design).
2. Choose *Tools* → *Transfer from System Landscape Directory* → *Import Software Component Versions*.
3. Select the new customer component from the list and choose *Import*.

3.3.2 Assigning Namespaces

The navigation tree in the Integration Builder displays imported software component versions as well as the underlying software component versions.
The objects of the underlying software component versions are displayed in the navigation tree in the *Basis Objects* subtree.

To create new objects in the Integration Builder, you must assign namespaces to the software component versions (see also 3.1.4).
Open the software component version and assign one or more customer namespaces (for example, `http://CustXY.com`).

Activate your changes.

4 Customer Development of Interfaces

4.1 Customer Interfaces

4.1.1 Integration Repository

Using the Integration Builder (Design), create new interface objects in customer namespaces. If the “based-on” relationship is maintained, then you can also reuse SAP objects (the input help offers them automatically).

4.1.2 ABAP Proxy Generation

Generate the proxies for the message interfaces you have defined in the relevant customer development system and transport them to the subsequent system. The customer development system must be connected to the Integration Repository. For more information, see the relevant section in the *Configuration Guide*.

If you want to use objects from other namespaces (for example, from SAP) in your namespace, then the objects must already exist. They are **not** regenerated or generated during proxy generation.

Proxy generation is a task for developers, just like creating programs or structures in the ABAP Dictionary. Therefore, you require developer authorization for your user.
Furthermore, we recommend that you create customer objects in the system in the customer namespace (for example, beginning with `Z` or with `/.../`). Developers can specify this prefix, together with the package in which the objects are to be created, before proxy generation.
Using the customer namespace is particularly important for the append structures (proxy objects of data type enhancements). Both the append structure name and the field names in the append structure are qualified with the customer prefix.
This prevents potential naming conflicts when parallel enhancements are made to the same data type by different partners or customers.

4.2 Enhancing SAP Interface Objects

4.2.1 Integration Repository

Message interfaces can be defined using message types or external definitions.

If message interfaces reference message types, you can enhance them without modification by creating a data type enhancement for the data type that is referenced in the message type.

You must create the data type enhancement in the customer software component in the customer namespace.
Using the customer namespace prevents naming conflicts between elements or attributes of the enhancement and the names of the data type and other data type enhancements that are also valid in the customer context for this data type (for example, partner enhancements). Elements and attributes of enhancements are qualified with the namespace prefix in XML instances of interfaces.

4.2.2 ABAP Proxy Generation

If an ABAP application for SAP interfaces requires the corresponding ABAP proxies, these are generally shipped with the interfaces. You can generate the proxy objects corresponding to the data type enhancement on the ABAP side by using the transaction SPROXY. To do this, you must first navigate to the customer component and customer namespace in which the data type enhancement is defined in the navigation tree of this transaction. If you call the function for creating the proxy for the data type enhancement, an append structure for the structure corresponding to the data type is generated.

During generation of the proxy append structure for a data type enhancement, the system asks you for the prefix for the names. Enter the customer namespace prefix. This prefix for the names of the ABAP objects is used for the append names and for the append fields. This prevents naming conflicts in the proxy objects in the case of multiple enhancements of the same data type by different customers/partners.

4.3 Modifying SAP Interface Objects

4.3.1 Integration Repository

If you want to modify SAP interface objects at the customer, a „based-on“ relationship must be defined between the customer software component and the SAP software component (see 3.2.2).

The prerequisite for changing objects in the underlying software components is that the `Objects are Modifiable` property is set in the customer software component version. To do this, open the customer software component version and select the *Objects are Modifiable* checkbox under *Object Attributes*.

You can then modify an SAP object as follows:

1. In the navigation tree of the customer software component, select the object to be modified in the *Basis Objects* subtree.
2. In the object editor, select change mode.

   The following message is displayed:

   `Object is defined in a subordinate software component version. Do you want to add the object to the software component and modify it?`
3. Choose *Modify*.

   You can edit the interface objects in the customer component, in other words make modifications to interfaces, (fault) message types, or data types.
4. Activate the change list containing this object.

   SAP upgrades do not overwrite the modified version of the object, but the system displays a message that a conflict resolution is necessary.

You can then use the conflict editor to compare the new SAP object with its modified version in an integration step.

If an object is changed as a result of a new SAP version being imported and the customer has made his or her own changes to the object but not yet activated them (see step 4 above), the conflict is not reported until the change list is activated.

Version conflicts occur when you attempt to activate an object version whose predecessor does not correspond to the last active version.

You can display any open conflicts in the Integration Repository on the *Conflicts* tab page in the navigation area of the Integration Builder.

4.3.2 ABAP Proxy Generation

Regenerate the SAP proxies in your modification system and then transport the modifications to the subsequent systems.

In the transaction SPROXY, a modified object now occurs twice: once in the SAP original component and once in the customer component. To regenerate the proxy to reflect the modified status of the object, navigate to the modified object in the customer component.

Note that a modification of this kind in the ABAP system will be overwritten when SAP sends a software update (for example, if the proxy object concerned has been changed as part of a Support Package). Unlike in the Integration Repository, the modification is lost but it can be easily reinstated by regenerating the proxies.

4.3.3 Modifying External Definitions

The concept of making enhancements without modifications does not apply to data types that are defined in external definitions (in DTDs, XSDs, or WSDLs). The modification process for external definitions in the Integration Repository is the same as the process for all objects in the Integration Repository described in 3.3.1.

As a rule, applications can provide special support for intended enhancements by providing import or include statements in the main document. If this is the case, the customer does not have to modify the main document, but merely has to enhance the parts that are necessary in the corresponding imported or included definitions. However, there should be separate guides available for such enhancements of external definitions for these applications.

4.3.4 Context Objects

There is generally no point in modifying SAP context objects at the customer because, other than the description, the context objects only contain information about their reference type. If the type is changed in a field of a request message that the context object is assigned to, then it is usually recommended to assign a new context object with an appropriate type rather than modifying the context object, because a context object can be assigned in multiple request messages.

It is also possible to adapt an SAP interface at the customer so that a context object that is required but missing at the customer is assigned.

5 Customer Development of Integration Scenarios

5.1 Without Modifying SAP Integration Scenario Objects

Integration scenarios developed by customers must be assigned to a customer software component version (see also section 3). If you are developing a customer integration scenario, you can reuse SAP objects from the Integration Repository and create new customer objects there.

5.1.1 Application Components of an Integration Scenario

You insert application components in your customer integration scenario as colored lanes (integration scenario editor). These application components must be assigned to a product or to a template

(representing a product). These are the prerequisites if you want to define new customer actions or reuse SAP actions. You use actions to define connections in the integration scenario.

The way in which you define the application component depends on the product that the application component is to represent and which actions you want to use. For more information, see the following sections.

Non-SAP Products in a Customer Integration Scenario

Create a customer product version and a customer software component version for a non-SAP product in the System Landscape Directory (see section 3.2.1). You can now create an application component for this product in the integration scenario. You can use all the actions that you have defined in the corresponding customer software component version in the application component.

SAP Products Without Customer Actions

It is possible to integrate an SAP product directly in the customer integration scenario as an application component. However, this means that you can only use SAP actions from the integrated SAP product in this particular application component and cannot use customer actions.

SAP Products With Customer Actions

To be able to use customer actions in application components that reference SAP products, you must first perform the following steps in the System Landscape Directory:

- You have created a customer product version, for example `CUST_CRM`, as described in the last section.
- The customer product version comprises software component versions, for example `CUST_A` and `CUST_B`, which are based on the software component versions of the SAP product. Note that SAP software component versions can also have underlying SAP software component versions that have possibly not been imported into the Integration Repository. If you need to use actions from such software component versions, then you must also define customer software component versions based on those from SAP for them.

By meeting these prerequisites, it is possible to use the following actions in application components that are assigned to the `CUST_CRM` product:

- Customer actions defined in one of the customer software component versions `CUST_A` or `CUST_B`.
- SAP actions from the underlying SAP software component version for `CUST_A` and `CUST_B`.

Product Templates in a Customer Integration Scenario

The alternative to the above three variants is to use a product template. A product template represents an arbitrary product. You do not have to make any changes in the System Landscape Directory. New customer actions must be assigned to the customer software component version in which the integration scenario was created and the option *Use in Product Templates* must be selected.

Restrictions:

- You cannot use any SAP actions that belong to a product in customer product templates.
- Navigation to customer interfaces may also be restricted.

5.1.2 Actions and Interfaces of an Integration Scenario

To define connections in the integration scenario, you require actions that can be used in the respective application component. You must create customer actions in customer software component versions. The actions can then be used in all customer products that this software component version has been assigned to.

Actions enable you to reference customer inbound and outbound interfaces or SAP interfaces by using the input help. You can also enter interface names in actions manually, but the following restrictions apply:

- The Integration Builder cannot check whether the interface name is correct. In other words, it cannot check whether the interface is located in the same software component version as the action (see below), or whether the interface object has been created at all.
- You cannot navigate from the action to the interface.

Whether you can reference an interface or not depends on the interface:

- **Customer Interfaces in Actions**

  The customer interface must be created in the same customer software component version in which the action is located. Only then can the interface be connected to the action.

- **SAP Interfaces in Actions**

  SAP interfaces are available in an action if the customer software component version is based on the SAP software component version in which the interface is defined.

- **SAP Interfaces Modified by Customers in Actions**

  Same as described above in Customer Interfaces in Actions.

You can use interfaces in actions to define connections in an integration scenario.

5.2 Modifying SAP Integration Scenarios

SAP does not support the modification of SAP XI integration scenarios. We recommend that you recreate the integration scenario in a customer software component version instead. If you only want to change one interface in a connection, for example because you have added a customer parameter, you can modify the interface as described in chapter 4.2. The name of the interface in the connection remains unchanged. Since the modified interface is then located in a customer software component version, it is no longer possible to navigate to the modified interface.

6 Customer Development of Integration Processes

The process for developing integration processes at the customer is similar to that for interface objects.
You must use the “based-on” relationship when modifying SAP integration processes.

You can only make enhancements without modifications to data types that are referenced using the message types of the abstract interfaces assigned to the container elements.

7 Customer Development of Mappings

When developing mappings at the customer in SAP XI 3.0, you can use the “based-on” relationship, as in interfaces.
Interface mappings can only reference message mappings and imported archives from the same or an underlying software component version.

Process Flow

1. Define a software component version with the relevant namespaces in which you want to develop your mappings.
2. If you want to reuse SAP mappings, define the “based-on” relationship for the corresponding underlying SAP software component.
3. Develop the mapping programs as follows:
   - Use the graphical mapping editor in the Integration Builder to develop message mappings.
   - Use the customer’s local development environment to develop Java programs and then import them into the Integration Builder at a later date.
   - Develop XSLT in the same way with the relevant tool on your PC.

   In Java programs and in XSLT, the name+path (or name+package) of the objects in the imported archives must be unique for all archives within a namespace, otherwise it is unclear at runtime which object is to be used.

   

   If you import the archive A1 with the XSLT mapping `Mappings/CRM/SalesOrder/MapSalesOrder.xsl` to the Integration Repository, then the same namespace cannot contain another archive A2 with an XSLT mapping `Mappings/CRM/SalesOrder/MapSalesOrder.xsl`.
4. If you also want to use SAP mapping programs, you must copy them to your software component version. Since the name+path (or name+package) must again be unique, you must save your own developments in a customer-specific path. This avoids naming conflicts at a later stage.

   

   If there is a large number of SAP objects but just a small number of customer objects, you can do the following:

   - Insert the SAP namespaces in the customer component

   - Perform a release transfer for the SAP mapping objects to the customer component

   - Add the customer objects to the SAP namespace (remember to use a naming convention so that no conflicts arise with SAP objects)

5. You can use SAP or customer mapping templates to create or change message mappings at the customer.
   When the mapping templates are loaded to the mapping editor, the appropriate templates of all software components of the Integration Repository are offered for selection.

8 Input Helps for Interfaces in the Integration Directory

The input help for interfaces in the Integration Builder (Configuration) identifies which product versions (and therefore which software component versions) are available for a given business system as the default. The input help therefore contains the interfaces from the Integration Repository for this particular software component version.

Since the customer software component version is not yet officially installed in this business system, the input help does not display the customer interfaces at first. However, it is still possible to display these interfaces. To do so, you can choose *All* in the input help dialog box. Note however that the Integration Builder would take a long time to select all interfaces and display them in the input help. Therefore, we recommend that you specify in the System Landscape Directory that the customer component is installed in the business system. The next section describes this procedure in detail.

8.1 Assigning Customer Components to a Business System

1. In the System Landscape Directory (SLD), navigate to *Business Landscape*.
2. In the list, select the business system that is to contain the customer component.
3. In this view, navigate to the system that is specified in *Technical System*.
4. Under *Installed Products*, choose *Add…* and add the customer product:
    a. Select the product in the list. The system now displays the software component versions that are assigned to this product under *Installed Software Components*.
    b. Select all software component versions.
    c. Choose *Add Product*. The system returns to the screen for the technical system.
5. Navigate back to the business system and under *Installed Products*, select your product again and save.
6. In the Integration Builder (Configuration) menu, choose *Environment* → *Clear SLD Data Cache*. This updates the information about the business systems that is buffered in the Integration Builder.