**Integration Framework**

**Scenario Development**

**Last Modified: June 14, 2023**

[1 Concepts of Scenario Development 5](#_Toc137659532)

[1.1 Scenario Package Concept 5](#_Toc137659533)

[1.2 Scenario Step Concept 5](#_Toc137659534)

[1.3 From Package Design to Active Scenario Packages 7](#_Toc137659535)

[2 Creating a Scenario Package 9](#_Toc137659536)

[2.1 Scenario Package User Interface and Parameters 9](#_Toc137659537)

[2.2 Scenario Package Functions 12](#_Toc137659538)

[2.3 Using Scenario Package Definitions 14](#_Toc137659539)

[2.4 Using Scenario Package Tools 15](#_Toc137659540)

[3 Developing a Scenario Step 17](#_Toc137659541)

[3.1 Step Development Parameters, Functions and Tools 17](#_Toc137659542)

[3.1.1 Step Development User Interface and Parameters 17](#_Toc137659543)

[3.1.2 Scenario Step Functions 23](#_Toc137659544)

[3.1.3 Using Scenario Step Tools 25](#_Toc137659545)

[3.2 Inbound User Interfaces 27](#_Toc137659546)

[3.2.1 Inbound Main User Interface 27](#_Toc137659547)

[3.2.2 Inbound Channel User Interface 28](#_Toc137659548)

[3.2.3 Inbound Retrieval User Interface 34](#_Toc137659549)

[3.2.4 Inbound Formatting for Flat Files and Files with Field-Offset Definitions 40](#_Toc137659550)

[3.3 Inbound Types 41](#_Toc137659551)

[3.4 Processing 43](#_Toc137659552)

[3.4.1 The Process Flow 43](#_Toc137659553)

[3.4.2 The Integration Framework Message Format 44](#_Toc137659554)

[3.4.3 The Graphical Flow Designer 46](#_Toc137659555)

[3.4.4 The BizFlow Language 48](#_Toc137659556)

[3.4.4.1 Using the Control Icons of a Processing Atom 48](#_Toc137659557)

[3.4.4.2 Using BizFlow Control Structures 50](#_Toc137659558)

[3.4.4.3 General Atoms 50](#_Toc137659559)

[3.4.4.3.1 Using the Start and End Structure 50](#_Toc137659560)

[3.4.4.3.2 Using the Sequence Control Structure 52](#_Toc137659561)

[3.4.4.3.3 Using the Conditional Processing Control Structure 52](#_Toc137659562)

[3.4.4.3.4 Using the Iteration Control Structure 54](#_Toc137659563)

[3.4.4.4 Using Functional Processing Atoms 56](#_Toc137659564)

[3.4.5 Very Basics about Developing in XSL and XPath 59](#_Toc137659565)

[3.4.6 Using the Integration Framework XSL Library 62](#_Toc137659566)

[3.4.7 Variables, Properties and Tables 67](#_Toc137659567)

[3.4.7.1 Variables, Properties and Tables General Information 67](#_Toc137659568)

[3.4.7.2 System Variables 79](#_Toc137659569)

[3.4.7.3 Local Variables 80](#_Toc137659570)

[3.4.7.4 Global Variables 83](#_Toc137659571)

[3.4.7.5 In Memory Variables (Memory and Session) 86](#_Toc137659572)

[3.4.7.6 Message Values 89](#_Toc137659573)

[3.4.7.7 SLD Properties 91](#_Toc137659574)

[3.4.7.8 Config Properties 94](#_Toc137659575)

[3.4.7.9 Global Properties 96](#_Toc137659576)

[3.4.7.10 Local Properties 100](#_Toc137659577)

[3.4.7.11 Global Tables 103](#_Toc137659578)

[3.4.7.12 SysType Properties 108](#_Toc137659579)

[3.4.7.13 Criteria Fields 111](#_Toc137659580)

[3.4.8 Testing and Debugging the Process Flow 114](#_Toc137659581)

[3.5 Outbound 117](#_Toc137659582)

[3.5.1 Outbound User Interfaces 117](#_Toc137659583)

[4 Obfuscating Development Content 125](#_Toc137659584)

[5 Developing an Individual Error Handling Step 128](#_Toc137659585)

[5.1 Introduction 128](#_Toc137659586)

[5.2 Error Handling for Synchronous Scenario Steps 129](#_Toc137659587)

[5.3 Error Handling for Asynchronous Scenario Steps 129](#_Toc137659588)

[5.3.1 Asynchronous Processing Overview 129](#_Toc137659589)

[5.3.2 Inbound Dispatcher 131](#_Toc137659590)

[5.3.3 Inbound Database (DB) Trigger 133](#_Toc137659591)

[5.3.4 Inbound Trigger 133](#_Toc137659592)

[5.3.5 Inbound Data Retriever in Processing 133](#_Toc137659593)

[5.3.6 Processing 134](#_Toc137659594)

[5.3.7 Distributor 137](#_Toc137659595)

[5.3.8 Outbound 137](#_Toc137659596)

[6 Using Job Lists 139](#_Toc137659597)

[7 Defining Processes 142](#_Toc137659598)

[8 Designing and Using an Individual Message Log 143](#_Toc137659599)

[8.1 Designing an Individual Message Log Step 143](#_Toc137659600)

[8.2 Defining the Individual Message Log Step for the Scenario Package 145](#_Toc137659601)

[9 Defining Value Mappings 146](#_Toc137659602)

[10 Selecting an XML Processor 147](#_Toc137659603)

[11 Versioning of Scenario Packages and Steps 148](#_Toc137659604)

[12 Providing Documentation for Scenario Packages 150](#_Toc137659605)

[13 Authentication for Scenario Packages 150](#_Toc137659606)

[13.1 Authentication User Interface 150](#_Toc137659607)

[13.2 Using the SAP Business One Standard Authentication 153](#_Toc137659608)

[13.3 Defining a Package-Specific Authentication with an Individual User List 155](#_Toc137659609)

[14 Adding Datasets to Scenario Import and Export 156](#_Toc137659610)

[15 Creating an Administration User Interface for a Scenario Package 156](#_Toc137659611)

[15.1 Requesting a Password for a Scenario-Specific User List 156](#_Toc137659612)

[Appendix A. Table of Inbound Channels 158](#_Toc137659613)

[Appendix B. Table of Outbound Channels 159](#_Toc137659614)

[Appendix C. Table of Process Steps 160](#_Toc137659615)

[Copyrights, Trademarks, and Disclaimers 163](#_Toc137659616)

# 1 Concepts of Scenario Development

The integration framework is an integration and collaboration server that allows you to develop and run integration scenarios. Integration scenarios let systems automatically exchange data with each other.

## 1.1 Scenario Package Concept

Integration scenarios that run in the integration framework are called **scenario packages**. Scenario packages consist of all that is required to exchange data between systems consistently.

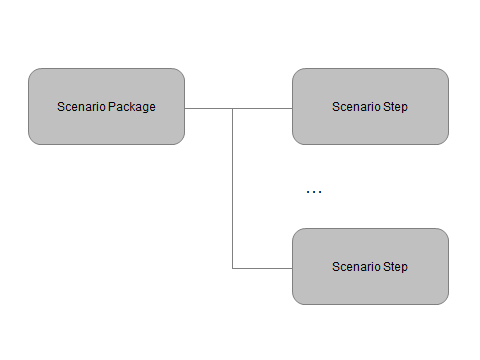
example.gif EXAMPLE

A company has a headquarters and subsidiaries. For daily business in the subsidiaries, employees need the customer and item master data from the headquarters in their subsidiary systems. In the integration framework, you provide a scenario package that initially loads the master data from the headquarters system to all subsidiary systems and then, you provide a mechanism that also provides master data changes and additions to the subsidiary systems afterwards.

SAP and SAP partners deliver integration scenarios that run in the integration framework, or customers develop them based on individual requirements for a specific project. The technical term for a scenario package in the integration framework BizStore is vPac (virtual package).

## 1.2 Scenario Step Concept

A scenario package consists of one or multiple scenario steps. A scenario step is a specific integration flow. The technical term for a scenario step in the integration framework BizStore is vBIU. BIU is the abbreviation for Business Integration Unit.



**Asynchronous and Synchronous Scenario Steps**

Scenario steps can be asynchronous flows between sender and receiver systems, or synchronous flows, triggered by a caller, and the response is returned to the caller.

**Inbound, Processing and Outbound (IPO)**

A scenario step has an inbound, a process, and an outbound phase.

In the **inbound** phase, the integration framework receives the incoming message and transforms it to the internal XML format. In the **process** phase, the integration framework transforms and enriches the message, determines the receivers and maps message values. In the **outbound** phase, the integration framework transforms the message into the technical format required by the receiver system or systems and finally sends out the message to the receiver system or systems.

An internal queuing mechanism is available to hand over data internally from one scenario step to another. Thus you can combine scenario steps in a scenario package.

example.gif Scenario Step Examples

* Sending a sales order from SAP Business One to the file system in the delimiter-separated values (DSV) format.   
  SAP Business One is the sender, the file system the receiver of the scenario step. An event from SAP Business One triggers or starts the scenario step. This is an asynchronous scenario step.
* A Web service returns a list of business partners from an SAP ERP system. The Web service system is the sender of this scenario step. This is a synchronous scenario step, because after receiving the request and providing the list of business partners, the integration framework sends the response back to the Web service system.
* A timer-triggered process picks up data from a database, calls a Web service for calculation, and hands over the data to a File Transfer Protocol (FTP) server. This is an asynchronous scenario step. The sender system is the database system; the receiver system is the FTP server.

In a scenario package,you can combine synchronous and asynchronous scenario steps. However, one scenario step exactly belongs to one scenario package. You cannot assign a scenario step to several scenario packages. If you want to use the same or a similar scenario step in different packages, copy the scenario step and adjust it accordingly.

## 1.3 From Package Design to Active Scenario Packages

The integration framework provides user interfaces for all scenario packages phases.

**Design**

In the design phase, you develop the scenario package content. You decide, for example, which system types communicate with each other, and how the integration framework must transform messages from the sender system to the format required by the receiver system. You decide whether you have specific security-related requirements, or whether you must fulfill specific monitoring requirements, for example.

When you create a scenario package, it has the design status by default, and you can develop content. You can, depending on your authorization rights, change all documents assigned to the scenario package as well as all scenario steps belonging to the scenario package. The scenario stepsare not active, that means, you cannot yet run them. Even, if a sender system triggers the appropriate events to start the integration, nothing happens. By default, new and imported scenario packages are in design mode.

**Setup**

Setting up a scenario package means that you prepare it for runtime in the specific system landscape. During setup, you select scenario steps you want to run, assign the systems from System Landscape Directory (SLD), define optional publishing and subscriber filters, define scheduler settings, if required, set values for properties, variables, global tables, and so on. The selected scenario steps are still not active. Therefore, you can easily switch the processes on or off.

In the integration framework, you have the following options to set up a scenario package:

* The integration framework provides a setup wizard that guides you through the setup steps. Select *Scenarios* → *Setup*, and click the  button.
* Select *Scenarios* → *Setup*, and click the *Sender*, *Receiver*, *Timer*, *Data Management* buttons.
* You can also call the scenario setup using the scenario control user interface, *Scenarios* → *Control* → *Setup*.

There is an option available for a predefined automatic setup during scenario package import.

For more information about defining sender and receiver systems in SLD, see the *Operations Guide Part 1*, section *System Landscape Directory*

**Active**

If a scenario package is active, it processes messages from sender to receiver systems and writes messages for monitoring purposes, if monitoring is switched on. After activation, you can no longer make changes to the scenario package. The scenario steps are active and run.

To activate a scenario package, you have the following options:

* In the integration framework, select *Scenarios* → *Setup*.
* Select *Scenarios* → *Control*, and click the activate checkbox in front of the related row.

There is also an option for predefined automatic activation during import of a scenario package. The scenario package must have the design & setup status for activation.

**Changing Active Scenario Packages**

In your development environment, you can allow changes for active scenario packages. This function is available to make changes easier at design time only. Do not use the option in a productive environment.

To allow changes, select *Maintenance* → *Cfg Dev Environment*, and click the *Allow Active Step Modification* checkbox.

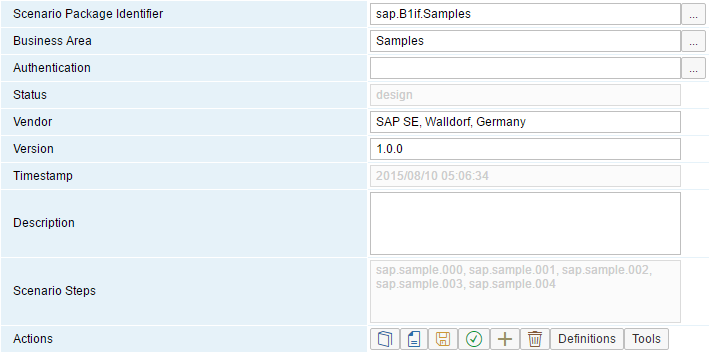
You can programmatically activate a scenario package using an integration framework API.

For more information, see the *API* Guide, section *Scenario APIs*

# 2 Creating a Scenario Package

## 2.1 Scenario Package User Interface and Parameters

To create a scenario package, logon to the integration framework and select *Scenarios* → *Package Design*.



Scenario Package Identifier

To create a scenario package, enter the identifier. The integration framework adds the namespace abbreviation. The abbreviation depends on the settings for the development environment. To set or change the prefix, select *Maintenance* → *Dev. Environment*, select the *Mode* and enter the development prefix in the *Development Prefix* field.

The maximum length of the identifier is 20 characters. You can use all letters, numbers, point and hyphen.

The scenario package identifier has the following pattern: <ns>.<id>.<steptask>

* <ns> is the vendor namespace abbreviation, for example, sap for SAP

Each scenario package belongs to a vendor. The vendor can be SAP, an SAP partner, or a customer.

* <id> is the name of the scenario package

Using an abbreviation for the scenario package in the step name enables you to display the steps together in the BizStore.

* <steptask> is the task of the step

example.gif EXAMPLE

sap.B1if.Samples is the identifier for the integration framework examples delivered by SAP.

To change a scenario package, click the […] button and select it from the list.

Business Area

You can assign your scenario package to a business area. Use the assignment in the *Scenarios* → *Control* section to sort scenario packages by business area.

The following options are available:

* Master Data
* Management Reporting
* Financial Accounting
* HR – Payroll
* Marketing
* Planning and Reporting
* Procurement
* Production
* CRM
* SRM
* FSCM
* Test

Authentication

If a scenario package contains at least one scenario step triggered by an incoming HTTP or Web service call, define the authentication method. The following options are available:

* No Authentication

The integration framework does not require credentials in incoming HTTP or Web service calls.

* Basic Authentication (user name, password)

The integration framework requires providing a user name and password of an integration framework runtime user in incoming HTTP or Web service calls.

* Basic Secure Authentication (enforce HTTPS, user name, password)

The integration framework requires HTTPS and providing a user name and password of an integration framework runtime user in incoming HTTP or Web service calls.

* B1 Authentication

The integration framework requires providing a user name and password in incoming HTTP or Web service calls. The user name and the password must be available in a SAP Business One company database.

* B1 Secure Authentication

The integration framework requires HTTPS and providing a user name and password in incoming HTTP or Web service calls. The user name and the password must be available in a SAP Business One company database.

* Optionally a special user-defined authentication

For more information about user-defined authentication, see section [*13 Authentication for Scenario Packages*](#a13)

Status

The read-only field displays the current status of the scenario package, for example, design or active.

Vendor

The information depends on the settings of the development environment. By default, the integration framework inserts the value of the field *Development Prefix Description* of the *Maintenance* → *Cfg Dev Environment* function. You can overwrite it.

Version

Enter the version number of your scenario package. If you create a scenario package, the integration framework sets the version to 1.0.0.

For more information, see section [*11 Versioning of Scenario Packages and Steps*](#a11)

Timestamp

The read-only field displays the timestamp of the last change of the scenario package. The integration framework does not reflect here changes in assigned scenario steps.

Description

Enter a description for the scenario package.

Scenario Steps

The read-only field displays the scenario steps that are assigned to the scenario package. You assign scenario steps to a scenario package in the *Scenario Step Design* user interface. In this way you cannot assign scenario steps to different scenario packages at the same time.

## 2.2 Scenario Package Functions



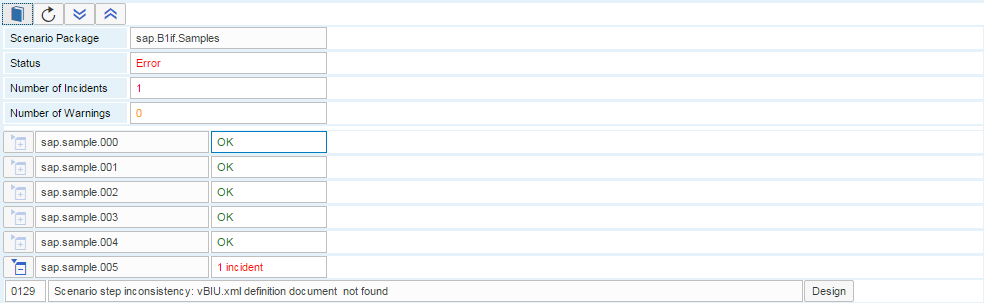
The integration framework provides context-based documentation. If documentation is available, the integration framework displays the documentation icon. The grey icon indicates not yet available documentation. To open documentation, click the icon.

, , 

An inbuilt consistency check validates your definitions against the model specification of the integration framework. The integration framework triggers the check when you open the user interface.

* The green icon indicates that all definitions are correct and complete.
* The yellow icon indicates that some warnings exist.
* The red icon indicates inconsistencies due to errors or incompleteness.

If the integration framework displays a yellow or red icon, click the icon. The integration framework displays details of the consistency check result and provides an incident number and a description for each warning and inconsistency. To check and solve the indicated issue in the related design user interface, click the [Design] button.





The integration framework supports providing documentation for a scenario package. Write documentation and save it in Portable Document Format (PDF) format in the base directory of the scenario package.

For more information, see section [*12 Creating Documentation for a Scenario Package*](#a12)

Optionally, you can generate a document based on the definitions. Select *Scenarios* → *Package Design* → [*Tools*] → *Generate documentation*. If both documents exist, a dropdown list allows you to select the documentation.



To create a scenario package, click the button.



To delete a scenario package including all assigned scenario steps, click the button.

recommendation.gif RECOMMENDATION

We recommend archiving scenario packages before deletion. The integration framework requests you to trigger archiving before deletion.



To save settings and definitions, click the button.

## 2.3 Using Scenario Package Definitions

[Definitions]

The integration framework provides you with various options that support you in programming scenario steps belonging to a scenario package.

To create definitions, click the [Definitions] button. You have the following options:

* Criteria Fields

Criteria fields represent XPath expressions in a message. You can define criteria fields at scenario package level. During scenario setup, the administrator sets filters for the criteria fields, for example, to filter messages for specific countries, regions or specific markets.

For more information, see section [*3.3.7.13 Criteria Fields*](#a3_3_7_13)

* Global Variables

Use global variables as representations of values in the design of scenario steps. Global variables are valid for all scenario steps of a scenario package.

For more information, see section [*3.3.7.4 Global Variables*](#a3_3_7_4)

* Global Properties

Global properties replace fixed values in scenario package development. The administrator selects individual values in scenario setup.

For more information, see section [*3.3.7.9 Global Properties*](#a3_3_7_9)

* Global Tables

Use global tables, if using global properties it not sufficient. With global tables, you can provide a list of values depending on combinations or other settings.

For more information, see section [*3.3.7.11 Global Tables*](#a3_3_7_11)

* Value Mappings

Use value mappings to externalize mappings between sender and receiver systems instead of hardcoding them in the development phase.

For more information, see section [*9 Defining Value Mappings*](#a9)

* Job List

Use the job list to subscribe your scenario package to changes in the SAP Business One landscape.

For more information, see section [*6 Using Job Lists*](#a6)

* Obfuscation

To make your development content unreadable, use the obfuscation function.

For more information, see section [*4 Obfuscating Development Content*](#a4)

* Error Handling

Provide individual error handling for your scenario package.

For more information, see section [*5 Developing an Individual Error Handling*](#a5)

* Message Log

Use the message log function to provide an individual message log for the scenario package

For more information, see section [*8 Designing and Using an Individual Message Log*](#a8)

* XML Processor

Select an XML processor for your scenario package processing.

For more information, see section [*10 Selecting XML Processors*](#a10)

* Processes

If a scenario package contains scenario steps that run in both directions between sender and receiver systems, you can use processes to assign scenario steps to a direction. It simplifies the scenario setup.

For more information, see [*7 Defining Processes*](#a7)

## 2.4 Using Scenario Package Tools

Scenario package tools provide the following functions:

* Save current version to archive
* Restore package from archive
* Delete package from archive

For more information, see [*11 Versioning of Scenario Packages and Steps*](#a11)

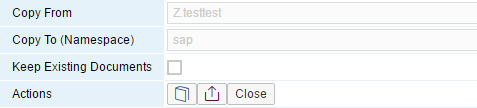
* Rename package

This function allows you to rename a scenario package. Enter a new identifier following the naming convention. The maximum length of the identifier is 20 characters. You can use all letters, numbers, point and hyphen. The integration framework adds the namespace abbreviation. The abbreviation depends on the settings of the development environment.

The integration framework adjusts the scenario package and references in the assigned scenario steps. Note that archived versions still have the old name.

* Copy package

The function allows you to copy a scenario package from another to your namespace.



Keep Existing Documents

If documents are already available in the namespace you copy the package to and you do not want the integration framework to overwrite documents, select the option.



Copy the scenario package

* Generate documentation

To generate a PDF document with information about the scenario package, and the assigned scenario steps, select the function. The integration framework saves the document in the base directory of the scenario package. To open the document, click the  icon.

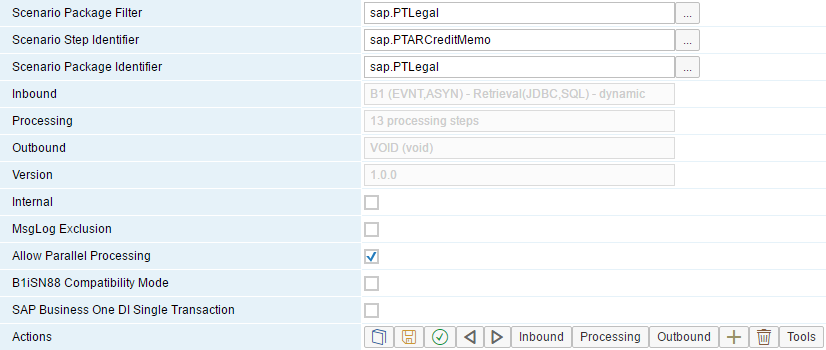
# 3 Developing a Scenario Step

## 3.1 Step Development Parameters, Functions and Tools

A scenario step is an integration flow. The integration framework support asynchronous flows between a sender and a receiver system type and synchronous flows, triggered by a caller. The short form for a scenario step in the integration framework BizStore is vBIU. BIU is the abbreviation for Business Integration Unit.

### 3.1.1 Step Development User Interface and Parameters

To start designing a scenario step, logon to the integration framework and select *Scenarios* → *Step Design*.



Scenario Package Filter

To reduce the selection list for the *Scenario Step Identifier* field to steps that belong to the scenario package, select a scenario package name.

Scenario Step Identifier

This parameter is the identifier of the scenario step. The identifier has a prefix and the step name separated by a dot (.). The prefix represents the integration framework namespace and denotes the company that provides the scenario step. Each SAP partner has an individual unique abbreviation. It guarantees a worldwide unique identifier and is important to support concurrent scenario steps from different vendors at the same time. Scenario steps starting with the Z prefix are steps developed in the customer namespace. If you create a scenario step, the integration framework adds the prefix based on the settings in the configuration of the development environment. In vendor mode, the integration framework uses the defined development prefix; in customer mode the integration framework uses the Z prefix. If you want to develop generic scenario steps you want to make available to many customers, develop them under your vendor settings. To display the configuration, select *Maintenance* → *Cfg Dev Environment*.

The prefix controls the authorization for changes. A customer can only change scenario steps in the Z namespace. A partner can change scenario steps in his or her own namespace and in the customer namespace.

The maximum length of the identifier is 20 characters. You can use all letters, numbers, point and hyphen.

recommendation.gif RECOMMENDATION

To easily associate the step by name with the corresponding scenario package, we recommend reflecting a package abbreviation at the beginning of the name, similar to the example above.

To select a step, click the […] button to open a list with all scenario steps, sorted by scenario packages.

Scenario Package Identifier

This is the identifier of the assigned scenario package. Each scenario step belongs to exactly one scenario package. You can develop a scenario step that is not assigned to any package. However, for step activation, assign the step to a scenario package. To do so, click the […] button and select the package. The character in brackets in front of the name displays the scenario package mode. You can only assign a step to a package, if the package is in design mode (D). It is not possible to assign a step to an active package (A). You can change the assignment of a step from one package to another. Both packages must be in design mode. If you change the assignment, the integration framework removes the no longer valid assignment and sets the new assignment.

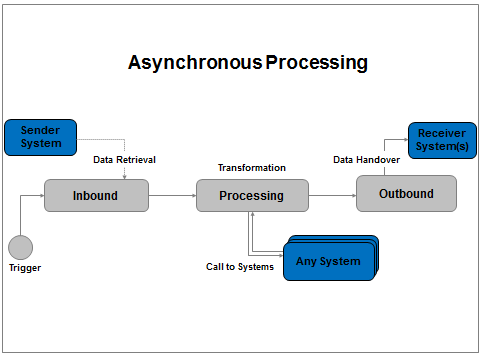
Inbound, Processing, Outbound

The *Inbound*, *Processing* and *Outbound* fields are read-only fields. They display basic information about the definitions in the inbound, processing and outbound phases. To enter definitions, click the [Inbound], [Processing] and [Outbound] buttons. For more information, refer to the following sections. Inbound, processing and outbound are the main processing phases, running in the integration patterns synchronous request-response and asynchronous, sender-receiver message processing.

**Asynchronous Processing**

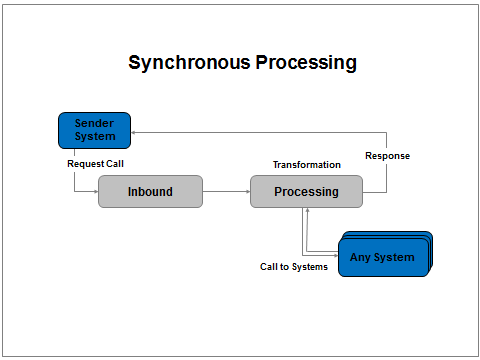
In asynchronous processing, a sender system sends a message or trigger, for example a SAP Business One event, to the integration framework that starts the scenario step processing. Alternatively, you have defined a timer for the scenario step that induces the integration framework to retrieve data from a sender system.

The inbound phase hands over the message in XML format to processing. The process flow transforms the message so that the integration framework can hand it over to the outbound phase. In the process flow, you have the option to define calls to various systems to enrich the message. The outbound phase transforms the message to the protocol the receiver system understands and hands over the message to the receiver system or systems.



**Synchronous Processing**

In synchronous processing, the inbound phase gets a request call from a sender system. The inbound phase hands over the message in XML format to processing. The process flow transforms the message so that the integration framework can hand it back to the original sender system. In the process flow, you have the option to define calls to various systems to enrich the message.



Version

The integration framework supports a versioning concept for scenario steps. The field displays the current version of the scenario step. The integration framework initializes a new scenario step with the 1.0.0 version.

For more information, see [*11 Versioning of Scenario Packages and Steps*](#a11)

Internal

By default, a scenario step is a public step that you can set up and activate by selecting *Scenarios* → *Setup* → *[Steps]*.

If you want to use a scenario step internally to be only called by other scenario steps in the process flow, select *Internal*.

The following applies for internal scenario steps:

* The integration framework displays but disables internal scenario steps for setup. This ensures that no one activates such a step accidentally.
* Only the processing phase definition is relevant for internal scenario steps. The inbound and outbound phase definitions are not relevant. You can ignore them.

MsgLog Exclusion (Message Log Exclusion)

To avoid the creation of message log entries for a scenario step at runtime, select *MsgLog Exclusion*. Use it especially for steps that constantly run, such as for example, a timer-triggered step that runs every minute. The setting avoids overloading the message log. If you select the checkbox, the integration framework does not display logs in the message log, choosing *Monitoring* → *Message Log*.

If you select the checkbox, the exclusion is valid independent of settings in the runtime settings *Maintenance* → *Cfg MsgLog*.

Allow Parallel Processing

To let scenario steps participate in parallel processing using scenario step instances for the processing and outbound phases at runtime, select the checkbox. By default, parallel processing is enabled for new scenario steps.

For more information about parallel processing, see the *Operations Guide 2*, in the *Configuring the Runtime* section

B1iSN88 Compatibility Mode

To generate SysType properties for runtime that have the SAP Business One 8.8 integration for SAP NetWeaver format, select the checkbox.

SAP Business One DI Single Transaction

The integration framework participates in the two-phase commit protocol with SAP Business One DI API. This leads to the following results:

* SAP Business One provides a mechanism for receiving notifications of data-driven events. Customers or partners add some code to the SBO\_SP\_TransactionNotification SAP Business One stored procedure. If you add code to SBO\_SP\_TransactionNotification to receive notifications for SAP Business One object changes and act on them accordingly, the notification does not work in conjunction with the integration framework processing.

The integration framework only commits or rolls back calls at the end of the processing phase, and not after each call to SAP Business One. If you design your notification in such a way that it rejects changes for a certain object, it is too late to do so at the end of the processing phase. This leads to an error that cannot be resolved and complex manual steps are required.

The DI transaction stays in in-doubt status, and the integration framework deactivates the IPO step. The integration framework recovery function tries to perform the transaction again after some time, but cannot succeed. If other subsequent steps try to perform their functions, but cannot do so, the integration framework deactivates them, too. If the integration framework has to handle many deactivated steps, other internal IPO steps are also affected and overall performance deteriorates.

* As long as a scenario step processing communicates with SAP Business One using the DI API, the related records in SAP Business One are locked for other users and processes, and the connection to the DI stays open. The related resources are occupied. This can have a negative impact on overall performance.

To change the integration framework behavior in connection with the DI API, select the SAP Business One DI Single Transaction checkbox. By default, the integration framework continues supporting the two-phase commit protocol and is, therefore, backward compatible.

If you select the new behavior for your scenario step, the following applies:

For the scenario step, the changed behavior affects SAP Business One inbound, the SAP Business One object, service, function, and the message call atoms, and SAP Business One outbound.

* Calls to SAP Business One are self-contained and the effects are immediately persistent after the return of the call. The behavior is similar to HTTP or synchronous Web service calls.
* Self-contained calls to the SAP Business One company database lock the database for only a very short time.
* Since the adapter no longer participates in the two-phase commit protocol, lengthy retry and rollback mechanisms no longer happen.
* The deactivation of internal integration framework IPO steps in conjunction with the DI API happens less often.

recommendation.gif RECOMMENDATION

* If you enable the function, the integration framework does not perform the scenario step as an atomic and consistent transaction.
* If an error occurs in the step, for overall data consistency, provide suitable compensation to undo changes in SAP Business One that have happened already. Note that for some objects, it is not possible to undo changes in SAP Business One. You cannot change objects that create journal entries. This is, for example, the case for invoices.
* We recommend using the function if you access SAP Business One only for reading data.
* Use the function if you write data to SAP Business One **only once**, and the call to SAP Business One is the last in processing.
* To make several calls to SAP Business One for writing data, distribute the calls to several scenario steps that you connect using the internal queue mechanism of the integration framework.
* **Do not use** the function if you perform several calls writing to SAP Business One, if the write operations depend on each other.

### 3.1.2 Scenario Step Functions



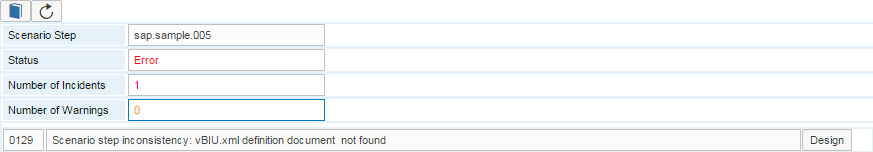
The integration framework provides context-based documentation. If documentation is available, the integration framework displays the documentation icon. The grey icon indicates not yet available documentation. To open documentation, click the icon.

, , 

The integration framework consistency check validates the definitions you have entered against the framework model specification. The integration framework checks definitions when you open the user interface, after selecting another scenario step or when you close a nested user interface, for example, *Inbound* to return to the current user interface.

* The green icon indicates that all definitions are correct and complete.
* The yellow icon indicates that some warnings exist.
* The red icon indicates inconsistencies due to errors or incompleteness.

If the integration framework displays a yellow or red icon, click the icon. The integration framework displays details of the consistency check result and provides an incident number and a description for each warning and inconsistency. To check and solve the indicated issue in the related design user interface, click the [Design] button.





Use the arrow icons to navigate between the steps. The integration framework sorts steps by the scenario step identifier.

[Inbound]

To define the inbound phase of a scenario step, click the [Inbound] button. No development is required for the phase. For the inbound phase, you define the inbound channel including the following:

* The system type that initiates the scenario step
* The trigger to start the scenario step, for example, a SAP Business One event
* The processing mode (synchronous, asynchronous)
* The identification method of incoming data, for example, a file name
* Optionally, you also define data retrieval from the sender system and data formatting during the inbound phase.

[Processing]

After the inbound phase, the integration framework hands over the inbound message to processing. You define processing using the BizFlow language in the graphical flow designer. The BizFlow language is a problem domain-specific language, especially focusing on the tasks required for integrating business applications. The functional components of the flow are sequences, conditions and iterations. Atoms as elements of the language offer transformations, conversions, calls, and so on. To open the graphical flow designer, click the [Processing] button. Here you define your processing logic.

[Outbound]

For asynchronous scenario steps, define outbound processing to specify the destinations the integration framework sends data to. The outbound definition is fully declarative, supported by multiple nested user interfaces that provide lists with appropriate options. Define the receiver system type and optionally some call details to allow the integration framework to handle the data handover.



To create a scenario step, click the button. Alternatively, enter a new name in the *Scenario Step Identifier* field.



To save parameter definitions, click the button.



To delete a scenario step, click the button.

### 3.1.3 Using Scenario Step Tools

The tools section provides the following functions:

* Rename Scenario Step

With this function you can rename a scenario step. This function is only available for scenario steps in your own and in the customer namespace. Enter the new name for the scenario step following the naming conventions. The integration framework generates the new name following the rules described for the *Scenario Step Identifier* attribute.

* Copy Scenario Step

This function allows copying the selected scenario step to a new scenario step. Use this function, if you develop a new scenario step similar to an existing one. Enter the name of the new scenario step and follow the naming conventions. The integration framework generates the new name following the rules described for the *Scenario Step Identifier* which even allows you to copy a scenario step from another namespace into your namespace. To prevent inadvertent copying, you must confirm the copy step first.

* Version Control

For more information, see [*11 Versioning of Scenario Packages and Steps*](#a11)

* Undo last modification in processing flow

Changes in the graphical designer can lead to fatal errors, typically caused by special characters in XML definitions. It can happen that the graphical designer cannot open anymore. In this case, use the function to undo the last change.

* Remove all not used XSL files

In the process flow, you use multiple XSL transformation (xform) atoms, each assigned to an XSL file, containing the code that performs the transformation. If you delete an XSL transformation atom from the flow, the integration framework does not delete the corresponding XSL file to avoid inadvertent deletion of developed code. If you are sure that you no longer need the XSL files no longer linked to an XSL transformation atom, use the function to clean up your development environment.

* Scenario Step Information

Use the function to display all settings of the scenario step. Optionally, you can display the information as an XML document.

[Copy] in partner mode

This function allows copying the selected scenario step to a new scenario step. Use the function to develop a new scenario step that is similar to an existing one. Enter the name of the new scenario step. Follow the naming conventions. The integration framework generates the new name following the rules described for the *Scenario Step Identifier* which even allows you to copy a scenario step from another namespace into your namespace. To prevent inadvertent copying, confirm the copy of the step.

## 3.2 Inbound User Interfaces

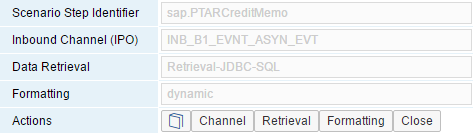
The integration framework supports the following inbound channels:

* SAP Business One
* SAP ERP
* Web Service Call
* HTTP Call
* Flat File
* Database
* Timer-based database retrieval
* Internal Queue
* Void (timer-based inbound)
* Predecessor

For more information about configuration details of the inbound channels, see the documentation assigned to the channel in the integration framework graphical flow designer user interface.

### 3.2.1 Inbound Main User Interface

To open the *Scenario Step Definition - INBOUND* user interface, click the [Inbound] button in the *Scenario Step Definition* user interface.



Inbound processing covers the following aspects of the scenario step:

* Inbound channel definition
* Data retrieval definition
* Formatting definition

Inbound Channel (IPO)

The read-only field displays the inbound channel. The channel covers the initiator system type, the trigger, the processing type and the identification method. The integration framework generates the identifier of the inbound channel based on definitions. In the example above, the inbound channel is asynchronous, of SAP Business One, and triggered by an incoming event. For more information, see *Appendix A: Table of Inbound Channels*

Data Retrieval

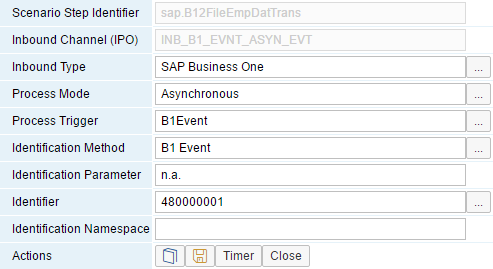
The read-only field displays how inbound data reaches the scenario step. It depends on the inbound channel type, whether data retrieval is required. Data retrieval is, for example, not required for all incoming calls that hand over data. Define data retrieval, if the incoming trigger, for example, an SAP Business One event, does not hand over the relevant business data.

Formatting

The read-only field is usually set to dynamic. If you must process a special formatting based on external instructions during the inbound phase, define the format control document. You need formatting information, for example, for an incoming text file, to control the conversion to XML by offset definitions or regular expressions.

### 3.2.2 Inbound Channel User Interface

To define inbound processing parameters, click the [*Channel*] button on the *Scenario Step Definition - INBOUND* user interface.



Inbound Channel (IPO)

The integration framework supports process types by channels. A channel consists of a sender system type, the process mode, the trigger and the identification method. A channel comprises all relevant combinations of these aspects. The integration framework generates the channel name reflecting the aspects.

For example, the channel INB\_HT\_CALL\_SYNC\_XPT indicates an inbound (INB) channel for HTTP (HT), a call triggers the process (CALL), processing is synchronous (SYNC), and an XPath statement defines the identification of the call (XPT). [Appendix A](#aA) (Table of Inbound Channels) displays channels that the integration framework supports.

Inbound Type

The inbound type defines the initiator system or system type. The following inbound types are available:

* Business Process
* SAP Business One
* SAP ERP
* Web Service Call
* HTTP Call
* Flat File
* Database
* Timer-based
* Internal Queue (internal explicit triggering by another scenario step)
* Void
* Predecessor (result process of another scenario step)

Process Mode

The process mode defines how the integration framework performs the processing. The supported mode depends on the inbound type. The mode can be Synchronous or Asynchronous:

* If a sender system calls the integration framework and waits for a response, select Synchronous.
* If you want to send a message from a sender system to a receiver system, select the Asynchronous mode for message-based integration.

Process Trigger

Process triggers depend on the selected inbound type and process mode. The following triggers are available:

* Call

An incoming call triggers the step.

* B1Event

An event coming from SAP Business One triggers the step.

* Timer

Scheduler-based settings trigger the step.

* File Exist

The existence of a file triggers the step.

* Queue

A message in an internal queue triggers the step.

* IDOC

An incoming IDoc from an ABAP-based system triggers the step.

Identification Method

The integration framework supports inbound channels in a generic way. In contrast to other integration solutions, you do not need to setup inbound processes per business object or method using extensive proxy generation. The integration framework can support, for example, all incoming Web service calls with a generic channel or all incoming events from a SAP Business One system with one generic channel. Define the business object or the required method based on the inbound data.

example.gif EXAMPLE

* You define a scenario step that retrieves all business partners from an SAP Business One system. The identification method is B1Event.
* An incoming HTTP call hands over an XML message with a specific name of the root tag. The identification method is Root Tag.

The supported methods depend on the selected inbound type, the process mode, and the process trigger. The following identification methods are available:

* B1 Logic

Incoming XML file, following the syntax of SAP Business One XML

* B1Event

Incoming event from an SAP Business One system

* File Name

Name of an incoming flat file

* First Line

First line of an incoming text file

* Fix Value

The integration framework looks for a fixed value.

* Queue/Stream

Name of internal queue and stream

* Root Tag

Name of the root tag of the inbound message

* vBIU (scenario step) Name

Name of the predecessor scenario step

* xPath

Any XPath statement that the integration framework processes on the inbound message

Identification Parameter

This parameter is optional. It depends on the selected identification method. If you select File Name as the identification method, you can optionally define the start position and the length of a substring of the file name that is relevant for identification. The format is x,y calculated as substring(filename,x,y). x is the start position; y is the length of the substring to be considered. If your incoming file name is always different because the timestamp is added to the file name, you can use this function to identify the file by only comparing the fixed parts of the file name.

example.gif EXAMPLE

The identification parameter is set to 3,2.

The identifier is myfile.

The settings let the integration framework retrieve all files that contain fi file name, for example, myfile.xml, myfile12.xml, myfilexyz.txt, and so on

If you select xPath as the identification method, define the XPath expression. Use the $msg variable for the incoming message.

Identifier

This is the identifier of the incoming business object or called method that the scenario step subscribes to. The integration framework compares the inserted string using the selected identification method against the value based on the incoming message.

Identification Namespace

This parameter is optional. If the processing of the incoming message requires the declaration of XML namespaces, define the XML namespaces. The integration framework generates the XML namespaces into all documents that are relevant for processing. Use the xmlns:ns=”nsdef” xmlns:ns=”nsdef” … syntax.

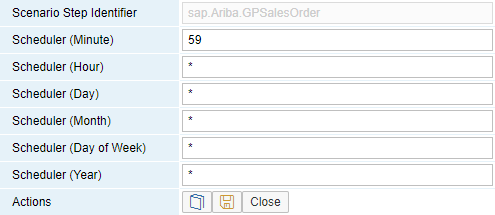


To save settings, click the button.

[Timer]

If the process trigger is a timer, define the default scheduler settings for the scenario step. During setup of the scenario package, the customer can overwrite the settings.

To define timer settings, click the button.



The timer definition is based on CRON syntax. CRON is time-based job schedule in Unix-like computer operating systems. A CRON entry has six fields to define day, date and time. The integration framework triggers the scenario step, if all entries are true. The integration framework compares the entries with the system time zone settings of the machine where the integration framework runs.

Scheduler (Minute)

Define the value for the *minute* part of the time. Use \* or a list of elements separated by comma. An element is either a number in the range 0 to 59 or two numbers in the range separated by a hyphen (meaning an inclusive range).

Scheduler (Hour)

Define the value for the *hour* part of the time. Use \* or a list of elements separated by comma. An element is either a number in the range 0 to 23 or two numbers in the range separated by a hyphen (meaning an inclusive range).

Scheduler (Day)

Define the value for the *day* part of the date. Use \* or a list of elements separated by commas. An element is either a number in the range 1 to 31 or two numbers in the range separated by a hyphen (meaning an inclusive range).

Scheduler (Month)

Define the value for the *month* part of the date. Use \* or a list of elements separated by commas. An element is either a number in the range 1 to 12 or two numbers in the range separated by a hyphen (meaning an inclusive range).

Scheduler (Day of Week)

Define the value for the *day of the week*. Use \* or a list of elements separated by commas. An element is either a number in the range 0 to 6 (Sunday = 0) or two numbers in the range separated by a hyphen (meaning an inclusive range).

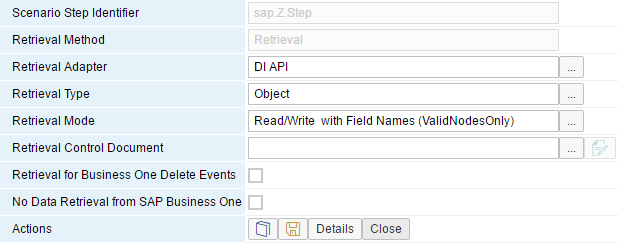
Scheduler (Year)

Define the value for the *year* part of the date. Use \* or a list of elements separated by comma. An element is either a number, specifying a year, for example, 2010, or two numbers in the range, separated by a hyphen (meaning an inclusive range).

The above settings trigger the step at the 59th minute every hour.

### 3.2.3 Inbound Retrieval User Interface

For sender systems that do not hand over data to the integration framework, you need to define the data retrieval from the sender systems. Click the [Retrieval] button. The integration framework sets the retrieval method and indicates, whether a retrieval definition is required or not. If the inbound trigger is an incoming call, you do not need to define a retrieval method. Then the retrieval method is Handover. If the integration framework displays Retrieval, define a method.



Retrieval Method

If the integration framework displays Retrieval, define a method.

Retrieval Adapter

Select the adapter the integration framework uses to retrieve the inbound data. The available options depend on the selected inbound channel. To display the adapters, click the […] button. The following adapters are available:

* DI API

Data interface of SAP Business One

* JDBC

Database request

* WSAS

Web Services Solicit Response

* HTTA

HTTP call

* FILI

File inbound

Retrieval Type

Select the protocol type for data retrieval. The values depend on the selected retrieval adapter. To select a value, click the […] button. The following values are available:

* Object

Object call, retrieval adapter is DI API

Object call, retrieval adapter is service layer for SAP Business One

* Service

Service call, retrieval adapter is DI API)

* SQL

SQL statement, retrieval adapter is JDBC

* Call

Retrieval adapter is FILI, WSAS, HTTA

Retrieval Control Document

This field is currently not relevant.

Retrieval for Business One Delete Events

This option is only relevant for events coming for SAP Business One DI API objects and services. If you also want to receive the delete events, select the option.

No Data Retrieval from SAP Business One

This option is only relevant for data retrieval from SAP Business One. If you only want to obtain the SAP Business One events without retrieving the data, select the checkbox. The integration framework disables and ignores all retrieval settings and leaves the @Role=”S” section of the incoming integration framework messages empty.

[Details]

Depending on the selected retrieval adapter and retrieval type, click the button to enter more details. The integration framework requires details for the following retrieval types:

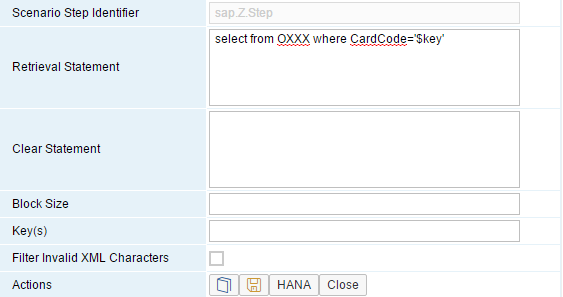
* Service
* JDBC SQL
* Incoming RFC Call

The integration framework needs details to perform a call to retrieve inbound data from the sender system. Note that no details are necessary for a DI API object. For a DI API service call, enter the service call details.

This data retrieval performs only the retrieval of the inbound data. If you want to retrieve additional data or information from the sender system or from another external system, do not perform such calls during the inbound phase. Define calls in the scenario step processing phase instead.

For more information, see section 3.3.

JDBC Call Details



Retrieval Statement

Define the SQL statement for data retrieval. You can use $key. The integration framework replaces the variable with the value handed over by the B1 event.

The integration framework supports the following placeholders in your SQL statement:

* Use $Top as a placeholder for the top value.
* Use $BlockNo as a placeholder for number of the block.
* Use $StartTime as a placeholder for an identifier, if you want to retrieve data using a timestamp. The integration framework sets the initial timestamp to 2006-10-01 00:00:00. In any further data retrieval the integration framework sets the $StartTime to the last retrieved time.
* Use $EndTime as a placeholder. The integration framework sets it to the current timestamp.

Clear Statement

Enter the SQL statement to clear the data after retrieval in the database table. This is optional. Set, for example, the flag from N for new to P for processed.

The integration framework supports following the placeholder in your SQL statement:

Use $where as a placeholder for the where clause calculation.

Block Size

If you want to retrieve data blocks from the database table, enter the block size. Enter a number greater than zero. If you use the $Top placeholder in your SQL statement, the integration framework replaces it with this value.

Key(s)

The field is optional. You need it, if you use the $where parameter in the *Clear Statement* field.

If you enter more than one key field, separate the entries by comma.

For more information, see the *Timer-Triggered Database Inbound* document

Filter Invalid XML Characters

To filter invalid XML characters from database fields, select the checkbox.

To define settings valid for all SQL calls performed in a scenario package, you can do the following:

1. Create the sqlcfg.xml document in the base directory of the scenario package.

For the xxx.<package>, provide the document in the following place in the BizStore: /com.sap.b1i.vplatform.scenarios.design/vPac.xxx.<package>/sqlcfg.xml

1. In the XML document, enter the following:

<sqlconfig xmlns="urn:com.sap.b1i.vplatform:entity">

<filterInvalXMLChar>true</filterInvalXMLChar>

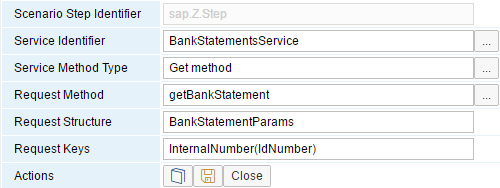
</sqlconfig>

If you are using explicit settings in XSL transformation atoms, you overlay the general setting.

[HANA]

To enter the SQL statement for SAP HANA, click [HANA].

SAP Business One DI API Service Call Details



Service Identifier

Define the service identifier of the B1 service, you want to call.

For more information, see the DI API help.

Service Method Type

Data retrieval supports the Get method and the GetList method. Select the service type method you need.

Request Method

Define the name of the request method.

For more information, see the DI API help.

Request Structure

Define the name of the request structure.

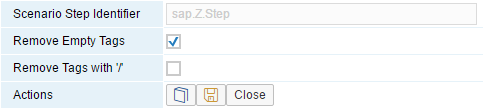
For more information, see the DI API help.

Request Keys

Define the key names of the request method. Usually, the key name in the request structure differs from the key name in the event, both values are necessary. Define the keys in format requestkey(eventkey).

For more information, see the SAP Business One DI API help.

RFC Call Details



Remove Empty Tags

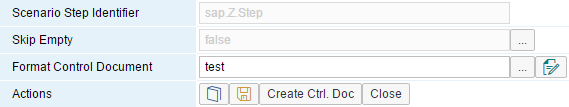
To remove empty tags from incoming IDoc, select the checkbox.

Remove Tags with /

To remove tags with / from incoming IDoc, select the checkbox.

### 3.2.4 Inbound Formatting for Flat Files and Files with Field-Offset Definitions

If the integration framework calculates the formatting based on an external control document, click the [Formatting] button to enter the required information. Formatting is relevant for incoming flat files that the integration framework transforms to XML files, based on regular expressions. Formatting is also relevant for a file with field-offset definitions. For delimiter-separated files, you can select to consider or ignore empty fields.



Skip Empty

If you have defined the csv payload type to handle delimiter-separated files, use the parameter to define the handling of empty fields.

* If you set the value to true, the integration framework ignores empty fields when converting the file to XML format.
* If you set the value to false, the integration framework creates tags for empty fields in the XML file. This is the default.

Format Control Document

The format control document contains XML tag names and the structure. Click the Create Ctrl. Doc button to provide the document in the base directory of the scenario step.



If you have enabled the embedded XML editor in the *Configuration of Development Environment* function, click the button and edit the format control document.

For incoming flat files, enter the technical inbound formatting settings, such as encoding, delimiter, wrapchar and payload type in SLD. For setting up a file system, see the *Operations Guide*, section *File System*

## 3.3 Inbound Types

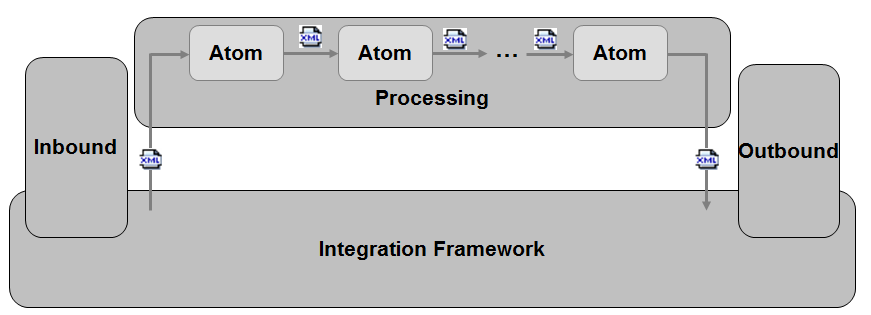
For more information about the inbound types, refer to the documentation about:

* Business Process inbound and the *Business Process Management Guide*
* SAP Business One inbound
* SAP ERP inbound
* File inbound
* Web Service inbound
* HTTP inbound
* Timer-based inbound
* Timer-based database inbound
* Predecessor inbound

## 3.4 Processing

### 3.4.1 The Process Flow

After the inbound phase, the integration framework hands over the incoming message to the process flow or BizFlow. The message runs through multiple steps and the process flow finally hands the message back to the integration framework for further processing. The complete process flow is **one transaction**.



In the process flow, you can define one or multiple flow atoms. Each atom hands over the outbound message as an inbound message to the next atom. The next atom works based on this inbound message and again creates an outbound message. The last atom in the flow creates the correct message for the receiver system for asynchronous processing, or for the caller for synchronous processing.

The integration framework offers many different flow atom types. The last flow atom in processing is always the **final transformation** (atom0, final).

When opening the process user interface for a scenario step for the first time, the integration framework generates a simple flow, consisting of the final transformation step. For asynchronous processing between a sender and a receiver system, this is often sufficient. The only required development is the adjustment of the generated XSL document (atom0.xsl). By default, the generated XSL document copies the message one-to-one.



### 3.4.2 The Integration Framework Message Format

The XML message that the integration framework processes from one flow atom to the next, is always a well-formed integration framework message. It consists of a header and a body section. The header contains process control information, the body contains the payload. Inside the body, different payload sections are available. The main payload sections are the following:

|  |  |
| --- | --- |
| **Payload Section** | **Description** |
| Payload/@Role=’T’ | The section contains the trigger information section, for example, the B1 event. |
| Payload/@Role=’S’ | The section contains the sender message. |
| Payload/@Role=’R’ | The section contains the receiver message. |

The T (trigger) and S (sender) payload sections are available at the beginning of the process flow. The final transformation creates the R (receiver) payload section at the end of the process flow.

At runtime, each flow atom adds outbound information as an additional payload section to the body of the integration framework message. Each flow atom has a unique identifier, starting with atom, followed by a sequential number, for example, atom1, atom2, and so on. Access each payload section using the absolute path or by using the identifier.

The integration framework is defined in the following XML namespace: xmlns:vpf="urn:com.sap.b1i.vplatform:entity"

<vpf:Msg MessageId="100518083505971270420A140FBBB876" …>

<vpf:Header>

<vpf:IPO Id="..."/>

<vpf:Sender Id="0010000101" ObjId="infile"/>

<vpf:Receiver Id="0010000102"/>

<vpf:vBIU Id="sap.Tutor-FileToDB" SId="sap.B1iFW-Test" filter=""/>

<vpf:Identification Ident="File Name" IdPar=""/>

<vpf:nsList/>

<vpf:Variables> ... </vpf:Variables>

<vpf:VarProperties> ... </vpf:VarProperties>

</vpf:Header>

<vpf:Body>

<vpf:Payload **Role="T"** Type="xxx">

**Trigger Message**

</vpf:Payload>

<vpf:Payload **Role="S"**>

**Sender Message**

</vpf:Payload>

<vpf:Payload id="atom1">

Result from step with id=atom1

</vpf:Payload>

…

<vpf:Payload id="atomn">

Result from step with id=atomn

</vpf:Payload>

<vpf:Payload **Role="R"**>

**Receiver Message**

</vpf:Payload>

</vpf:Body>

</vpf:Msg>

### 3.4.3 The Graphical Flow Designer

The integration framework provides a graphical flow designer that supports the integration development. To open the graphical flow designer, select *Scenarios* → *Step Design* → *[Processing]* or *Scenarios* → *Control* → *[Overview]* → *[Processing]*.

In the upper right corner, the tool bar provides the following functions:



[Test]

Opens the test environment

For more information, see the *Testing and* *Debugging the Process Flow* section of this guide.

 Refresh

Refreshes the user interface



Undo last flow change



Remove XSL documents that are no longer part of the flow. If you create a new XSL transformation atom (xform), the integration framework creates an XSL document in the BizStore. If you delete the transformation atom in the flow definition, the integration framework does not delete the assigned XSL document. In this way, the integration framework prevents deleting your work. The function allows deleting XSL documents without assignments to XSL transformation atoms.

[VarL]

Create and change local variables for the scenario step

[VarG]

Create and change global variables for the scenario package the scenario step is assigned to.

[PropL]

Create and change local properties for the scenario step.

[PropG]

Create and change global properties for the scenario package the scenario step is assigned to.



Save the current window size. To save the window size, select *Maintenance* → *Cfg Dev Environment* → *Store Resized Window*. This is only possible, if your browser supports the function.



The integration framework displays the icon of the inbound or sender system type that triggers the scenario step. To display documentation for the inbound channel, click the icon.



To display process flow documentation, click the icon.



If the scenario package contains an individual step for error handling, the integration framework displays the red icon.



The integration framework displays the icon for the outbound system type or receiver of the message. If you have defined a synchronous scenario step, the integration framework does not display the icon. To display documentation for the outbound channel, click the icon.

### 3.4.4 The BizFlow Language

Define the process flow using the BizFlow graphical language. The language supports you in defining model-driven integration of business applications. In the BizFlow, you assemble atoms into a process flow. The BizFlow language and the process atoms (short atoms) cover the typically required integration patterns for integrating business applications in an easy, declarative way.

The following atom types are available:

* Control structures
* Transformations
* Type conversions
* Calls

To display available atoms in the integration framework, select *Maintenance* → *System Info* → *[Functions]* → *Available Process Atoms*.

#### 3.4.4.1 Using the Control Icons of a Processing Atom

Each atom is available as a graphical shape with a short text in the center, displaying the atom type.



The green and red icons display the result of the plausibility check for the transformation atom. If values are missing or inconsistent, the integration framework displays the red icon, otherwise the integration framework displays the green icon.

**Moving Atoms**

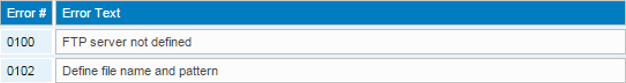
To select an atom, click the icon. The integration framework displays the selected atom in a red frame.



If you click a branch or for-each atom, you select a complete sequence of atoms. You can move the selected atom or sequence to another place in the flow. Another click on the icon deselects the atom or sequence of atoms. To move the atom to another position in the process flow, click the grey arrow in the atom that is the new predecessor of the selected atom.

**Atom Error Information**

The integration framework displays the  icon, if it also displays the red icon. To display the result of the plausibility check for the atom, click the icon. The integration framework displays the error information:





To display atom attributes, click the icon.



To edit an atom, click the icon. Each atom type has an individual user interface with specific attributes.



To delete an atom, click the button. If you click the button in a branch of a for-each atom, you delete the complete sequence of atoms.

**Adding an Atom**

To add an atom on the right side, click the  [Add] button.

With the  [Add] button that is only available in a path for conditional processing, you add a new path to the branch atom. If you have selected another atom or sequence and the integration framework displays it or them in a red frame, you move this atom or sequence with the  [Add] button to this place.

C:\_WorkingArea\Publications around B1i\vPlatform\backup\20110404\com.sap.b1i.vplatform.ide\bin\checkbox_sel.gif

If the integration framework displays the C:\_WorkingArea\Publications around B1i\vPlatform\backup\20110404\com.sap.b1i.vplatform.ide\bin\checkbox_sel.gif [Activate] checkbox in an atom or a complete sequence of atoms for a branch or a for-each atom, the atom or sequence is active. To deactivate the atom or sequence, deselect the checkbox for the atom or sequence. At design time, this allows you to temporarily focus on a particular implementation step although the overall flow structure is already designed.

#### 3.4.4.2 Using BizFlow Control Structures

In the BizFlow you have the following control structures available:

* Start and end
* Sequence
* Conditional processing
* Iteration
* Functional atoms

#### 3.4.4.3 General Atoms

##### 3.4.4.3.1 Using the Start and End Structure

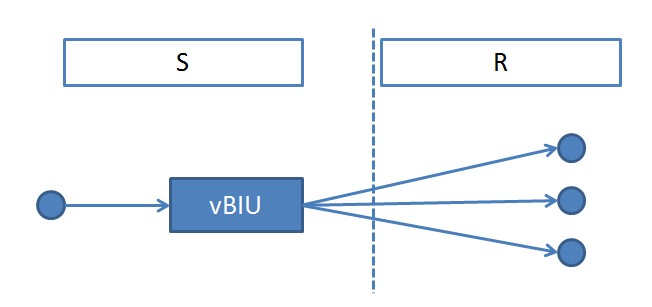
The integration framework generates the start and end atom. The atoms indicate the start and the end point of a process flow of the scenario step.

 … 

Above the start atom, the integration framework displays the scenario step and the dataset name where the integration framework has saved the scenario step in the BizStore. The  [Add] button allows you to add a new atom in this place. To display the BizFlow in XML format, click the  [View] icon.

**Running in the Sender Context**

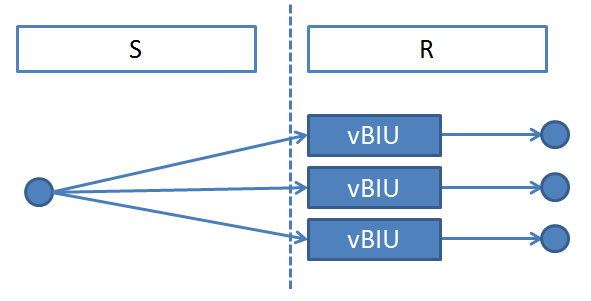
The  [S] icon indicates that the integration framework processes the BizFlow in the context of the sender system. Only after processing, the integration framework determines the receiver systems. Keep in mind, that the receiver information is not available during scenario step processing. This default setting makes the processing very efficient. If there are multiple receiver systems, the integration framework processes the BizFlow only once.



If it is necessary to call the receiver system before handing over the data, for example, for data enrichment, you cannot use the option.

**Running in the Receiver Context**

The C:\_WorkingArea\Publications around B1i\vPlatform\pics\Screen Elements\procr.png [R] icon indicates that the integration framework processes the BizFlow in the context of the receiver system after receiver determination. Consider that if you process a message to multiple receiver systems, the integration framework processes the BizFlow multiple times.



This option is necessary, if you must call the receiver system before handing over the data. Click the [S] button to change the setting to [R] or vice versa.

##### 3.4.4.3.2 Using the Sequence Control Structure

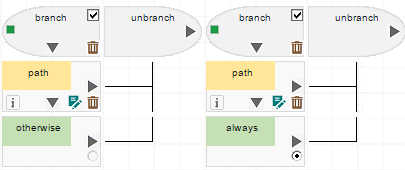
By default, the integration framework processes all atoms one after the other. If you design steps in the process flow, the integration framework ensures that it processes the steps in exactly this order.

note.gif NOTE

Note that one scenario step is one transaction. That means, that you cannot save information at runtime and no subsequent atom can rely on this information. However, you do not need saved information. Use the message or memory variables to hand over information between atoms.

##### 3.4.4.3.3 Using the Conditional Processing Control Structure

Conditional processing allows you to process information based on conditions in the message. Define the conditions as XPath expressions.

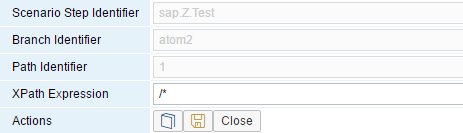


You can use multiple path atoms and one otherwise atom. The integration framework processes all path atoms for which the condition is true. The unbranch atom continues processing when the integration framework has processed all path atoms.

note.gif NOTE

Note that this is different from xsl:choose. It only processes the first true condition. The integration framework works in parallel mode; the processing order of different paths is random.

The integration framework processes the otherwise path, if none of the path conditions is true. If you click the radio button in the lower right corner of the otherwise atom, the integration framework always processes the otherwise path. This can be very useful to keep the original message, in case the original message is lost in the processing path.



Scenario Step Identifier

The read-only field displays the name of the scenario step.

Branch Identifier

The read-only field displays the name of the corresponding branch atom. The integration framework generates the name.

Path Identifier

The read-only field displays the number of the current path. Note that the number does not indicate the processing order of multiple path constructs. The integration framework generates the number.

XPath Expression

Define which part of the message the integration framework hands over to the path processing and the condition to process this path. You cannot use variables or properties for this XPath expression. You must declare the full path. For example, the /\*[/vpf:Msg/vpf:Body/vpf:Payload[./@Role='S']/inbound/flag='true'] XPath hands over the complete message of the sender system under the condition that /inbound/flag=’true’ in the message. If the expression is too complex, run a transformation beforehand. Add a transformation atom before the branch, which calculates a simple flag that you can check instead.

The unbranch atom consolidates the results of all path processing. To guarantee a well-formed XML document, the unbranch atom adds an artificial root tag <bfa:unbranch> to the XML document to consolidate all results from the different path sequences. Take this into account in the first subsequent transformation atom. To correct the unbranch atom, open the edit user interface for the subsequent transformation atom, click the [Generate] button and select the *Correct after Branch* option.

<bfa:unbranch xmlns:bfa="urn:com.sap.b1i.bizprocessor:bizatoms" schemaversion="1.0">

...

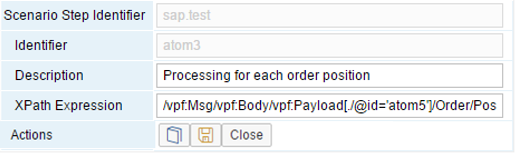
</bfa:unbranch>

##### 3.4.4.3.4 Using the Iteration Control Structure

The for-each atom allows you to run through iterations in the process flow, typically for all elements of a particular type or name, depending on a condition. You can use it, for example, in a transformation of an order to run through a processing for each order position.



The for-each construct consists of the for-each start atom and the join end atom. The join atom continues processing when the integration framework has finished all iterations. The integration framework usually works in parallel mode; the processing order of different iterations not necessarily follows the order of the positions and is therefore random.



Scenario Step Identifier

The read-only field displays the name of the scenario step.

Identifier

The read-only field displays the identifier of the atom. The integration framework generates the identifier.

Description

Enter a description for the atom.

XPath Expression

Define, which parts of the message the integration framework hands over to for-each processing. You cannot use variables for the XPath expression, declare the complete path. If the expression is too complex, run a transformation atom prior to the iteration. Add a transformation atom in front of the for-each atom. It calculates a simple flag that you can check instead.

The message in processing has the last element of the defined XPath expression as a root tag. Following the example above, where the split expression for the for-each atom is /vpf:Msg/vpf:Body/vpf:Payload[./@id='atom5']/Order/Position[./@status='ok'], each message inside the for-each processing has the root tag <Position>.

The join atom consolidates the results of all iterations. To guarantee a well-formed XML document, the join adds the artificial <bfa:join> root tag to cover all results. Take this into account in the first following transformation atom.

<bfa:join xmlns:bfa="urn:com.sap.b1i.bizprocessor:bizatoms" schemaversion="1.0">

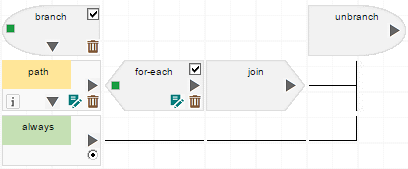
...

</bfa:join>

note.gif NOTE

Note that for-each processing invalidates the integration framework message. After for-each processing, the original integration framework message is no longer available. This is not a problem inside a path of a conditional processing, because the otherwise path (always=true) keeps the message.

If you create a for-each loop, choose the complete for-each processing (branch, split – join, xform). If you need another for-each loop inside an already existing path, use the simple for-each processing (split - join).



The complete sequence guarantees that the integration framework consolidates all results and the integration framework corrects the message to the integration framework message format.

#### 3.4.4.4 Using Functional Processing Atoms

All other processing atoms are functional atoms. For more information about functional atoms, open the edit user interface and click the book icon.

The following functional atoms types are available:

* Transformation atoms
* Simple call atoms
* Complex call or conversion atoms

The xform and final **transformation atoms** possess a corresponding XSL document. The integration framework saves the XSL document in the base directory of the scenario step. To display the XSL document, click the yellow area of the atom. To define the transformation, use an XML editor. All transformation atoms are yellow.



The **simple call atoms** work with the values you define. The atoms are blue. An example for a simple atom is the *Call URL* atom.



The **complex call** or **conversion atoms** send data to a conversion function or to a specific interface. You need a predecessor transformation, which prepares the data that you send to the function or interface. The complex call and conversion atoms are blue. An example is the *Send Email* atom.



If you create a complex call or conversion atom, the integration framework asks you whether you want to create a predecessor transformation atom. If you agree, the integration framework creates two atoms. In the call atom, the integration framework inserts the link to the predecessor transformation. The integration framework generates the correct schema for data handover to the following call or conversion atom into the XSL document of the predecessor transformation atom.

If you want to generate the request schema later into the XSL document, open the edit user interface of the transformation atom and click the [Generate] button.

The integration framework supports the following functional processing atoms:

* XSL Transformation (xform)
* Call B1 Object (SAP Business One Object)
* Call B1 Batch (SAP Business One Batch)
* Call B1 Service (SAP Business One)
* Call B1 Function (SAP Business One)
* Call Service Layer Object
* Call Service Layer Script
* Call URL
* Call HTTP
* Call Web Service
* Call Java Class
* Call .NET
* Call RFC
* Call SQL
* Conversion XML – TXT
* Conversion XML – JSON
* Conversion XML – BIN
* Conversion by Regex
* Conversion Value Mapping
* Generate Crystal Report
* Call Scenario Step
* DIR File System Info
* Key Expansion
* Upload File to File System (write)
* Download File from File System (read)
* Upload File to FTP Server (write)
* Download File from FTP Server (read)
* Send B1 Message (SAP Business One)
* Send email
* Receive email
* Persist Documents/Groups
* Put to Internal Queue
* Include B1iP BizFlow
* Copy File

For configuration details of the processing atoms, see the documentation of the specific atom.

### 3.4.5 Very Basics about Developing in XSL and XPath

XSLT (eXtensible Stylesheet Language Transformations) is a powerful, declarative language that you can use to manipulate XML documents in a very efficient way. Although it requires some experience to use the full power behind XSLT, with some basic knowledge, you can start immediately. Usually, you only need a small subset of XSLT for your tasks.

As a working principle, you can assume that you are in the role of a printer and your task is to print the correct outbound message for the receiver. The information you have is the current inbound message, which the sender system or the predecessor atom provides. Like a printer you cannot jump from the end to the beginning to add some data. Your development flow is strictly sequential.

Your outbound is always an XML document, which means it has to follow the XML guidelines. An XML document must have exactly one root tag. So you start your task by defining the root tag of your outbound message. Each tag consists of a start and an end tag, which you provide in one line or in two lines. Between the start and the end tag, provide a value or more nested start and end tags.

Let us assume you have the following sender message and your task is to create the following receiver message:

**Sender Message**

<SNDMsg>

<Header>

<Field1>headervalue01</Field1>

<Field2>headervalue02</Field2>

</Header>

<Lines>

<Line>

<LField1>value01</LField1>

<LField2>value02</LField2>

</Line>

<Line>

<LField1>value03</LField1>

<LField2>value04</LField2>

</Line>

</Lines>

</SNDMsg>

**Receiver Message**

<RCVMsg>

<RcvHeaderFieldA>headervalue01</RcvHeaderFieldA>

<RcvHeaderFieldB>headervalue02</RcvHeaderFieldB>

<LineCount>2</LineCount>

<RcvLine pos="1">

<First>value01</First>

<Second>value02</Second>

</RcvLine>

<RcvLine pos="2">

<First>value03</First>

<Second>value04</Second>

</RcvLine>

</RCVMsg>

The XSL (eXtensible Stylesheet Language) below transforms the sender to the receiver message:

<RCVMsg>

<RcvHeaderFieldA><xsl:value-of select="SNDMsg/Header/Field1"/></RcvHeaderFieldA>

<RcvHeaderFieldB><xsl:value-of select="SNDMsg/Header/Field2"/></RcvHeaderFieldB>

<xsl:variable name="linecount" select="count(SNDMsg/Lines/Line)"/>

<LineCount><xsl:value-of select="$linecount"/></LineCount>

<xsl:if test="$linecount&gt;0">

<xsl:for-each select="SNDMsg/Lines/Line">

<RcvLine>

<xsl:attribute name="pos"><xsl:value-of select="position()"/></xsl:attribute>

<First><xsl:value-of select="./LField1"/></First>

<Second><xsl:value-of select="./LField2"/></Second>

</RcvLine>

</xsl:for-each>

</xsl:if>

</RCVMsg>

**Procedure**

1. Create the <RCVMsg></RCVMsg> root tag.
2. To start defining the message, create the <RcvHeaderFieldA> and <RcvHeaderFieldB> header tags.
3. To insert the value of the sender message in between, use the <xsl:value-of> XSL command.

This is the most important command.

1. In the select section of the command, enter the XPath statement that leads to the exact location in the sender message from where you want to retrieve the value.   
   In our example, the XPath is SNDMsg/Header/Fieldx.
2. To introduce the linecount variable, use the <xsl:variable> command.
3. Assign the number of lines in the sender message that are equal to the number of tags with <Line> name between the <Lines> tags to the variable.
4. Again, describe the location using the count(SNDMsg/Lines/Line) XPath statement.
5. To complete the header information, create the <LineCount> tag and place there the value gathered by the variable.
6. Define a variable with $, followed by the name of the variable.
7. Provide the lines of the message. Provide a line only, if there is at least one line in the sender message. The condition in XSL is the <xsl:if> command.

note.gif NOTE

Be careful when using special characters, such as <, >, &.   
Do not use >, but the &gt; sequence instead.

1. Another important XSL command is the <xsl:for-each> command. It allows you running through multiple sections in a message. In the example, we run through all lines, again the sections are defined by an XPath statement. For each of the sections, create the <RcvLine> tag and with the pos attribute to which you assign the current position number, using the position() function.
2. Create the <First> and <Second> tags and assign the values of the sender message. Use the <xsl:value-of> XSL command. You currently define inside the for-each section. Address the location of the value relative to the section you are in. Define the XPath using ./LFieldx addresses the current tag.

### 3.4.6 Using the Integration Framework XSL Library

For frequently used tasks in the transformation area, the integration framework provides a library that you can use in the XSL stylesheets. The following library functions are available:

* String operations

String operations provide some functions to manipulate strings, respectively to retrieve some information from them. You can use string operations to delete leading zeros in a string, for example.

* Date/Time functions
* The date/time operations provide functions around date and time. You can retrieve the current date and time. You can calculate some special dates or provide formatted output.
* System functions

The system functions are more technical, for example, to generate a globally unique identifier (GUID).

* SAP Business One functions

You can, for example, retrieve an SAP Business One object name by providing the object identifier, or retrieve the name by providing the identifier.

To display available functions, select *Help* → *XSLT Library*.



XSLT Library

Select the library you want to use

[Load Docu]

To open the documentation, click the [Load Docu] button. The documentation displays available templates with a short description. The templates names start with the b1ilib*.* abbreviation followed by the name.

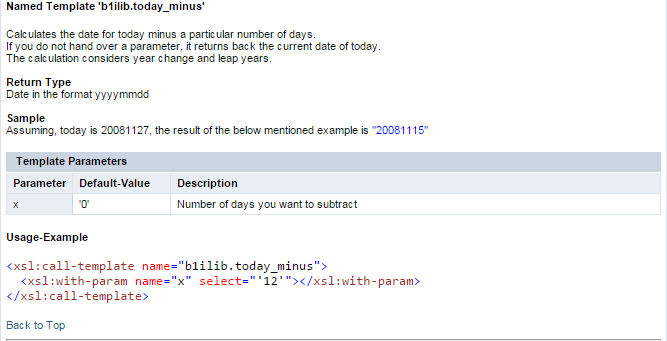


In the beginning, the integration framework displays the include command. Use the statement from the documentation and write it to the appropriate place in your XSL stylesheet.

note.gif NOTE

Without the include instruction, the template is not accessible at runtime and the integration framework throws a runtime error.

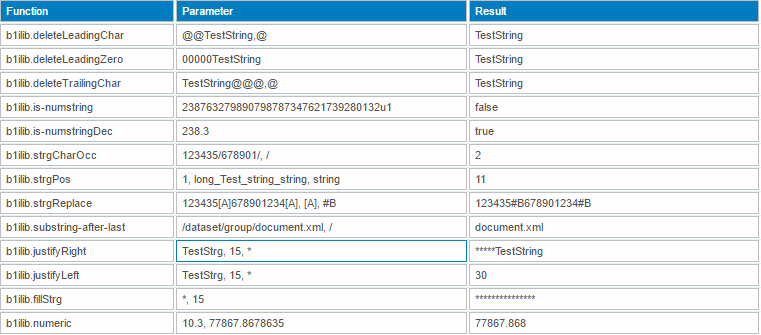
To display a function, click the name, and the integration framework displays the information. For example, to display detailed information about the b1ilib.today\_minus template, click the name. it provides information about how to call the function, a detailed description, the parameters, the default settings and an example how to use it. You can copy the example and paste it to your XSL stylesheet.



To navigate back to the overall list, click the [Back to Top] link. To use a template, copy the example from the documentation and paste it to your XSL stylesheet.

[Test]

To test the library, click the button. The integration framework displays the test result for the selected library.



**Including Library Functions in the XSL Transformation**

XSL library functions are called templates. To have the templates available, include the library document using the include command at the end of the XSL, before closing the <xsl:stylesheet> tag.

<xsl:template name="transform">

...

</xsl:template>

<xsl:include href="../../com.sap.b1i.system.lib/xsl/string.xsl"/>

### 3.4.7 Variables, Properties and Tables

This section provides you with information about variables, properties that you can use for scenario development

**Variables, Properties and Tables**

|  |  |
| --- | --- |
| Variables | [System variables](#a3_3_7_2) |
| [Local variables](#a3_3_7_3) |
| [Global variables](#a3_3_7_4) |
| In memory variables | [Memory variables](#a3_3_7_5) |
| [Session variables](#a3_3_7_5) |
| Message values | [Element values](#a3_3_7_6) |
| Properties | [SLD properties](#a3_3_7_7) |
| [Config properties](#a3_3_7_8) |
| [Global properties](#a3_3_7_9) |
| [Local properties](#a3_3_7_10) |
| [System Type Properties](#a3_3_7_12) | |
| [Global tables](#a3_3_7_11) | |
| [Criteria fields](#a3_3_7_13) | |

#### 3.4.7.1 Variables, Properties and Tables General Information

**Using Variables, Properties and Tables in Parameter Fields of Atom User Interfaces**

| **Variable** | **Description** |
| --- | --- |
| $string | Placeholder for the value of a system variable, local variable, global variable or local or global property. |
| $string(name) | Placeholder for the value of a system variable, local variable, global variable or local or global property.  The notation is only relevant in the context of an SQL call atom.  It provides explicit type casting to string to bypass the cross-side scripting check for an SQL call. |
| $[name] | Placeholder for the value of the first element with this name in the integration framework message. |
| $[atomx/name] | Placeholder for the value of the first element with this name in the section, created by atomx in the integration framework message. |
| $(name) | Placeholder for the value of a memory variable |
| $((name)) | Placeholder for the value of a session variable |
| $\*sysid.adapter.prop\* | Placeholder for the value of a property related to an SLD entry, for example, $\*0010000101.JDBC.url\* |
| ${tbl[row,col]} | Placeholder for the value of a global table type 1. row can be a number (for example, 2) or a condition (for example, 2=’233’ or 2=233) |
| ${tbl[row,row,col]} | Placeholder for the value of a global table type 2. row can be a number (for example, 2) or a condition (for example, 2=’233’ or 2=233) |
| $?cfgpar? | Placeholder for the value of an integration framework connectivity or runtime parameter |

For all notations in brackets or parenthesis, you can use a local or global variable in the notation to define the values (name, atomx, tbl, row, col).

example.gif EXAMPLE

$(($name))

**Fixed and Dynamic Values**

In the atom user interfaces, you can define fixed or dynamic values.

**Fixed Values**

* A fixed value always starts with #.

example.gif EXAMPLE

If you enter #0010000105, the integration framework hands over the 0010000105 value to the atom call.

* For parameter field names followed by the asterisk sign (\*) in the user interface, you can use variables and properties.

example.gif EXAMPLE

The $mySysId variable defines an XPath.

If you enter #$mySysId, the integration framework uses the XPath statement that is defined for the mySysId variable and runs this XPath statement against the integration framework message. The integration framework replaces $mySysId with the result value of the XPath statement.

The $var variable has the value 8 ($var=8).

If you enter $var, the integration framework replaces the variable with the number 8 at runtime.

To assign a value of string type to a variable, use the apostrophe (‘) character. For example, $var=’mystring’

You can combine fixed text and variables in one field.

example.gif EXAMPLE

For a property, you define the following value: #123$Var1DEF

Var1=/vpf:Msg/vpf:Body/vpf:Payload[./@id=’atom2’]/root

This is the incoming message:

<Msg>

<Body>

<Payload id=”atom2”>

<root>ABC</root>

The result is: 123ABCDEF

**Dynamic Values**

Dynamic values **do not** start with #.

The integration framework interprets values that do not start with # as an XPath statement. The integration framework runs the XPath statement against the integration framework message and replaces it with the value of the XPath.

If you want to mix fixed values and XPath statements, use the concat XPath function.

example.gif EXAMPLE

concat(’ABC’,/vpf:Msg/vpf:Body/vpf:Payload[./@id=’atom5’]/root/@att1,’DEF’)

In dynamic values, you cannot use variables and properties.

**Atom References**

In many atoms, you can use an atom reference. An atom reference links to the result of a predecessor transformation atom to pick up the request document that the current atom needs to send to the call or to define call properties.

Different atoms support different ways to define the property.

* The **simple atom definition** only requires the atom identifier.

For example, #atom5, #$myAtom, or an XPath statement, such as /vpf:Msg/vpf:Body/vpf:Payload[./@id=’atom5’]/myAtom

The following atoms support the simple atom definition:

|  |  |
| --- | --- |
| **Atom Name** | **Property Name** |
| B1 Object Call | Payload |
| B1 Service Call | Payload |
| Web Service | Payload and Settings |
| Crystal Report Call | Payload |
| HTTP Call | Payload and Settings |
| RFC Call | Payload |
| Regex Conversion | Input |
| Put to Internal Queue | Input |
| Send Email | Input |

* The **complex atom definition** provides the following options:
* Simple atom definition, for example, #atom5
* Sub atom definition, for example, #$atom5/root/mysection
* XPath statement, for example, #/vpf:Msg/vpf:Body/vpf:Payload[./@id=’atom5’]/root/mysection

The integration framework interprets the above statement as a string.

The following atoms support the complex atom definition:

| **Atom Name** | **Property Name** |
| --- | --- |
| .NET Call | Input |
| Java Class Call | Input |
| Call Scenario Step | Input |
| Bin2XML Conversion | Input |
| XML2TXT Conversion | Input |
| JSON Conversion | Input |
| Store File | Input |
| FTP Upload | Input |

* Special atom definition cases

|  |  |
| --- | --- |
| **Atom Name** | **Property Name** |
| Web Service Call | Reference for Connectivity |
| HTTP Call | Reference for Connectivity |

You can define an atom name, the sender system string, or a valid SysId.

**Inbuilt System Variables**

|  |  |
| --- | --- |
| **Variable** | **Description** |
| $b1task | Task (insert/update/delete) of the incoming transaction |
| $msg | This is the variable addressing the root tag of the message coming from the sender system |
| $now | This is the current time in format HH:MM:SS |
| $sender | Identifier of the sender system |
| $today | This is the current date in format YY-MM-DD |
| $userid | This is the identifier of the user that has logged. This is only relevant for session-based scenarios. |
| $username | This is the name of the user that has logged in. This is only relevant for session-based scenarios. |

**Inbuilt Configuration Properties (Connectivity and Runtime Parameters)**

|  |  |
| --- | --- |
| **Variable** | **Description** |
| $?Runtime.B1i HTTP Port? | Integration framework HTTP port. |
| $?Runtime.B1i HTTPS Port? | Integration framework HTTPs port. |
| $?Runtime.B1i Server? | Integration framework server name. |
| $?Connectivity.Internet: Proxy Host? | Proxy host name. |
| $?Connectivity.Internet: Proxy Port? | Proxy port number. |
| $?Connectivity.Send Email: SMTP Server? | SMTP server name. |
| $?Connectivity.Send Email: SMTP Port? | SMTP port number. |
| $?Connectivity.Send Email: SMTP User? | SMTP user name. |
| $?Connectivity.Send Email: SMTP Password? | SMTP password. |

**Inbuilt SLD Properties (SAP Business One)**

For more information about the SLD properties for SAP Business one, see the *Operations Guide*, section *SAP Business One*

**Variables**

$\*xxxxxxxxxx.B1DI.b1Server\*

$\*xxxxxxxxxx.B1DI.licenseServer\*

$\*xxxxxxxxxx.B1DI.company\*

$\*xxxxxxxxxx.B1DI.dbType\*

$\*xxxxxxxxxx.B1DI.dbUser\*

$\*xxxxxxxxxx.B1DI.dbPassword\*

$\*xxxxxxxxxx.B1DI.userName\*

$\*xxxxxxxxxx.B1DI.password\*

$\*xxxxxxxxxx.B1DI.language\*

$\*xxxxxxxxxx.B1DI.isTrust\*

$\*xxxxxxxxxx.B1DI.jcoPath\*

$\*xxxxxxxxxx.B1DI.diProxyHost\*

$\*xxxxxxxxxx.B1DI.diProxyPort\*

$\*xxxxxxxxxx.B1DI.proxyHost\*

$\*xxxxxxxxxx.B1DI.proxyPort\*

**Inbuilt SLD Properties (File System)**

For more information about the SLD properties for SAP Business one, see the *Operations Guide*, section *File System*

**Variables**

$\*xxxxxxxxxx.FILI.filePattern\*

$\*xxxxxxxxxx.FILI.Encoding\*

$\*xxxxxxxxxx.FILI.Delimiter\*

$\*xxxxxxxxxx.FILI.WrapChar\*

$\*xxxxxxxxxx.FILI.PayloadType\*

$\*xxxxxxxxxx.FILO.filePattern\*

**Inbuilt SLD Properties (Database)**

For more information about the SLD properties for SAP Business one, see the *Operations Guide*, section *Database System*

**Variables**

$\*xxxxxxxxxx.JDBC.driver\*

$\*xxxxxxxxxx.JDBC.url\*

$\*xxxxxxxxxx.JDBC.username\*

$\*xxxxxxxxxx.JDBC.password\*

**Inbuilt SLD Properties (HTTP)**

For more information about the SLD properties for SAP Business one, see the *Operations Guide*, section *HTTP System*

**Variables**

$\*xxxxxxxxxx.HTTA.destProtocol\*

$\*xxxxxxxxxx.HTTA.destHost\*

$\*xxxxxxxxxx.HTTA.destPort\*

$\*xxxxxxxxxx.HTTA.destPath\*

$\*xxxxxxxxxx.HTTA.query\*

$\*xxxxxxxxxx.HTTA.proxyHost\*

$\*xxxxxxxxxx.HTTA.proxyPort\*

$\*xxxxxxxxxx.HTTA.method\*

$\*xxxxxxxxxx.HTTA.authentication\*

$\*xxxxxxxxxx.HTTA.user\*

$\*xxxxxxxxxx.HTTA.password\*

$\*xxxxxxxxxx.HTTA.user2query\*

$\*xxxxxxxxxx.HTTA.password2query\*

$\*xxxxxxxxxx.HTTA.trustStoreURI\*

$\*xxxxxxxxxx.HTTA.keyStoreURI\*

$\*xxxxxxxxxx.HTTP.associatedSrvIP\*

**Inbuilt SLD Properties (SAP ERP)**

For more information about the SLD properties for SAP Business one, see the *Operations Guide*, section *SAP ERP*

**Variables**

$\*xxxxxxxxxx.RFCA.applicationSever\*

$\*xxxxxxxxxx.RFCA.client\*

$\*xxxxxxxxxx.RFCA.user\*

$\*xxxxxxxxxx.RFCA.password\*

$\*xxxxxxxxxx.RFCA.language\*

$\*xxxxxxxxxx.RFCA.systemNumber\*

$\*xxxxxxxxxx.RFCA.maxConnections\*

$\*xxxxxxxxxx.RFCA.gatewayServiceNumber\*

$\*xxxxxxxxxx.RFCA.gatewayHost\*

$\*xxxxxxxxxx.RFCA.senderPartner\*

$\*xxxxxxxxxx.RFCA.senderPort\*

$\*xxxxxxxxxx.RFCA.receiverPartner\*

$\*xxxxxxxxxx.RFCA.receiverPort\*

$\*xxxxxxxxxx.RFCP.applicationSever\*

$\*xxxxxxxxxx.RFCP.client\*

$\*xxxxxxxxxx.RFCP.user\*

$\*xxxxxxxxxx.RFCP.password\*

$\*xxxxxxxxxx.RFCP.language\*

$\*xxxxxxxxxx.RFCP.systemNumber\*

$\*xxxxxxxxxx.RFCP.maxConnections\*

$\*xxxxxxxxxx.RFCP.gatewayServiceNumber\*

$\*xxxxxxxxxx.RFCP.gatewayHost\*

$\*xxxxxxxxxx.RFCP.programID\*

$\*xxxxxxxxxx.RFCP.unicode\*

**Inbuilt SLD Properties (Web Services)**

For more information about the SLD properties for SAP Business one, see the *Operations Guide*, section *Web Service*

**Variables**

$\*xxxxxxxxxx.WSAN.destProtocol\*

$\*xxxxxxxxxx.WSAN.destHost\*

$\*xxxxxxxxxx.WSAN.destPort\*

$\*xxxxxxxxxx.WSAN.destPath\*

$\*xxxxxxxxxx.WSAN.query\*

$\*xxxxxxxxxx.WSAN.proxyHost\*

$\*xxxxxxxxxx.WSAN.proxyPort\*

$\*xxxxxxxxxx.WSAN.authentication\*

$\*xxxxxxxxxx.WSAN.user\*

$\*xxxxxxxxxx.WSAN.sslTrustStorePassword\*

$\*xxxxxxxxxx.WSAN.password\*

$\*xxxxxxxxxx.WSAN.sslTrustStorePath\*

$\*xxxxxxxxxx.WSAR.associatedSrvIP\*

$\*xxxxxxxxxx.WSAS.destProtocol\*

$\*xxxxxxxxxx.WSAS.destHost\*

$\*xxxxxxxxxx.WSAS.destPort\*

$\*xxxxxxxxxx.WSAS.destPath\*

$\*xxxxxxxxxx.WSAS.query\*

$\*xxxxxxxxxx.WSAS.proxyHost\*

$\*xxxxxxxxxx.WSAS.proxyPort\*

$\*xxxxxxxxxx.WSAS.authentication\*

$\*xxxxxxxxxx.WSAS.user\*

$\*xxxxxxxxxx.WSAS.password\*

$\*xxxxxxxxxx.HTTA.trustStoreURI\*

$\*xxxxxxxxxx.HTTA.keyStoreURI\*

The following table gives you an overview about variables, values, properties and tables. It describes, where you can define them, how you can assign values, where they are valid and where you can use them.

|  | **Definition** | | | **Assign Values** | **Scope** | **Usage** |
| --- | --- | --- | --- | --- | --- | --- |
| System variables | Predefined by the integration framework | | | Internally during processing | Everywhere | Atom user interfaces |
| Local variables | Scenarios, Step Design Processing [VLocal] | | Atom user interface [VarL] | Fixed by design or dynamically based on the inbound message | Scenario step | Access in XSL coding or in the atom user interface to define the atom parameters |
| Global variables | Scenario Package Design - Definitions | Scenarios – Step Design - Processing [VGlobal] | Atom user interface [VarG] | All scenario steps of a scenario package |
| Memory variables | In XSL <xsl:variable name=”mem1”  select0”document(‘/com.sap.b1i.internal.xc/xml/ipobag.xml?mem1=ABC’)”/> | | | Programmatically during process flow | Scenario step |
| Session variables | In XSL coding (authentication and/or process flow)  <xsl:variable name=”mem1”  select0”document(‘/com.sap.b1i.internal.xc/xml/sessioninfo.xml?sess1=ABC’)”/> | | | Existing with the first incoming HTTP call of a scenario package, accessible for all scenario steps, triggered by synchronous HTTP. Expires with the session. |
| Message value | Tags inside the XSL coding | | | Scenario step |
| Global properties | Scenario package design - Definitions | Scenario package design – Processing [PGlobal] | | Default from design or customized based on customer definitions during scenario setup | All scenario steps of a scenario package |
| Local properties | Scenarios, Step Design Processing [PLocal] | Scenario package design – Processing [PLocal] | | Default from design or customized based on customer definitions during scenario setup | Scenario step |
| SLD properties | Predefined in the integration framework | | | SLD maintenance user interface | Everywhere |  |
| Config properties | Maintenance – Cfg Runtime and Maintenance – Cfg Connectivity | | | Customer definition |
| Global tables | Scenarios – Package Design - Definitions | | | Customer definition during setup | All scenario steps of a scenario package |

#### 3.4.7.2 System Variables

System variables are predefined. The following system variables are available:

|  |  |
| --- | --- |
| **Variable** | **Description** |
| $msg | Addressing the root tag of the message provided by the sender system |
| $now | Current time in HH:MM:SS format |
| $today | Current date in YY-MM-DD format |

The following variables are only relevant for session-based login using HTTP to access user information. You can only access the variables at runtime; they are not available in the test environment. The values depend on the settings in authentication processing.

|  |  |
| --- | --- |
| **Variable** | **Description** |
| $userid | Identifier of the user that is logged in |
| $username | Name of the user that is logged in |

**Scope**

System variables always exist.

**Value Assignment**

The integration framework assigns values to the variables at runtime directly before it triggers the process flow. The integration framework sets the values for the system variables at the beginning of the transaction based on system timestamp or based on settings in the authentication.

**Using Variables in the Process Flow**

You can use the variables in the atom user interface in the parameter definition. All parameters that support variables are marked with \* (asterisk).

#### 3.4.7.3 Local Variables

Use local variables in the design and setup of a scenario step to address variable values in the inbound message. Local variables are valid for the scenario step for which you define them.

**Examples**

* In scenario step processing, you define the data retrieval using an SQL statement and in the where-clause you define the value of the primary key for which entry you want to retrieve the data. The data can be found in a specific place of the inbound message.
* During setup, you define a filter criterion for the receiver system based on a value in the incoming message. You use the variables in XPath statements or other processing instructions, such as SQL statements with a leading $, similar to using variables in XSLT.

To define local variables:

* Select Scenarios → Step Design →Processing → VLocal
* In the atom user interface, click the [VarL] button.

You can add and delete local variables, and you can define values for them. The values can be explicit values (literals) or XPath expressions. The integration framework sets the values at runtime before it triggers the process flow. The XPath statement can therefore access all data in the message, which is available at this point in time.

|  |  |
| --- | --- |
| **Variable** | **Value** |
| gvar1 | #any text |
| gvar2 | #001 |
| gvar3 | ’001’ |
| gvar4 | 001 |
| gvar5 | concat($gvar2,$gvar3,string($gvar4)) |
| gvar6 | number($gvar2)+number($gvar3)+$gvar4 |
| gvar7 | $msg/root/@flag |
| gvar8 | /vpf:Msg/vpf:Header/vpf:Sender/@ObjId |
| gvar9 | concat($gvar7,$gvar8,$gvar5) |

To provide values for local variables, you have the following options

|  |  |
| --- | --- |
| **Variable** | **Providing Values** |
| gvar1 | Start the value with the hash (#) character. |
| gvar2 | Start the value with the hash (#) character. |
| gvar3 | Wrap the value with the apostrophe (‘) character |
| gvar4 | If the value is a number, for example, 1234, enter it without hash and without wrapping it. The integration framework interprets the string as a number. Note that the integration framework deletes leading zeroes. gvar4 finally has value 1. |
| gvar5 | You can use XPath functions, such as, for example, number(), string(). |
| gvar6 | You can calculate new values by using other variables. Make sure that the new variable is in alphabetical order behind the variables you use. |
| gvar7 | To address the root tag of the message from the sender system, use the $msg system variable |
| gvar8 | You can enter XPath expressions. |
| gvar9 | You can use XPath functions and combine them. |

The example above generates the following values:

|  |  |
| --- | --- |
| **Variable** | **Value** |
| gvar1 | any text |
| gvar2 | 001 |
| gvar3 | 001 |
| gvar4 | 1 |
| gvar5 | 0010011 |
| gvar6 | 3 |
| gvar7 | true |
| gvar8 | myobject |
| gvar9 | truemyobject0010011 |

**Scope**

Local Variables are only valid for the scenario step for which you define them.

If you define a variable as a local and as a global variable, only the local variable is valid. You can use all variables in the complete process flow. Local variables are available in the header of the integration framework message.

<Msg ... >

<Header>

...

<Variables>

<var id="mystring" value="value"/>

...

By default, the integration framework generates local variables into the XSL stylesheet of a transformation atom. This allows you to directly access them in the XSL coding.

To optionally deactivate the generation, select *Maintenance* → *Cfg Dev Environment* → *Generate Variables to XSL*).

**Value Assignment**

The integration framework assigns the values to the variables at runtime, directly before triggering the process flow.

The values for local variables can be fixed literals defined at design time, or the integration framework dynamically sets them based on the incoming message.

**Using Variables in the Process Flow**

You can use local variables in the XSL coding. You can also use all variables in the atom user interface in the parameter definition. All parameters that support variables are marked with \* (asterisk).

In the XSL coding you can directly access the value of a local variable using $, the leading vp abbreviation, and the name of the variable, for example, <xsl:value-of select="$vpmystring"/>. It is a prerequisite that you have activated the generation into the XSL document. You can also use the incoming message in the following way: <xsl:value-of select="/vpf:Msg/vpf:Header/vpf:Variables/vpf:var[./@id='mystring']/@value"/>.

In the atom user interface, you can enter local variables similar to the XPath notation with $name. In a parameter field, you can use the same variable multiple times and you can combine them with other variables, properties, and so on.

**Example**

#select \* from TABLE where key1=’$myVar1’ and key2=’$myVar2’

note.gif NOTE

To avoid cross-site scripting, the integration framework checks variables in an SQL statement of an SQL call atom at runtime. If the variable is set in quotations marks (apostrophe), the integration framework handles it as a string, otherwise as a number. If the variable is not in quotation marks and not a number, the integration framework replaces the value with NaN. If you use a variable, for example, for a table name, you can avoid this behavior by explicitly type-casting the variable to a string using $string(variable-name).

#### 3.4.7.4 Global Variables

Use global variables in the design and setup of a scenario package to address variable values in the inbound message. Global variables are valid for the scenario package and all steps assigned to the package.

To define global variables, you have the following options:

* Select Scenarios → Package Design → Definitions
* Select Scenarios → Step Design → Processing → VGlobal
* In the atom user interface, click the [VarG] button.

You can add and delete global variables. You can enter explicit values (literals) or XPath expressions. At runtime, the integration framework sets the variables before triggering the process flow. The XPath statement can access all data in the integration framework message that is available at this point in time.

| **Variable** | **Value** |
| --- | --- |
| gvar1 | #any text |
| gvar2 | #001 |
| gvar3 | ’001’ |
| gvar4 | 001 |
| gvar5 | concat($gvar2,$gvar3,string($gvar4)) |
| gvar6 | number($gvar2)+number($gvar3)+$gvar4 |
| gvar7 | $msg/root/@flag |
| gvar8 | /vpf:Msg/vpf:Header/vpf:Sender/@ObjId |
| gvar9 | concat($gvar7,$gvar8,$gvar5) |

You have the following options to provide values for global variables:

| **Variable** | **Providing Values** |
| --- | --- |
| gvar1 | Start the value with the hash (#) character. |
| gvar2 | Start the value with the hash (#) character. |
| gvar3 | Wrap the value with the apostrophe (‘) character |
| gvar4 | If the value is a number (for example, 1234), enter it without any hash or wrapped by the apostrophe. The integration framework interprets the string as a number. Note that the integration framework deletes leading zeroes. gvar4 finally has value 1. |
| gvar5 | You can use XPath functions like for example, number(), string(). |
| gvar6 | You can calculate new values by using other variables. |
| gvar7 | You can use the $msg system variable to address the root tag of the message from the sender system. |
| gvar8 | You can enter XPath expressions. |
| gvar9 | You can use XPath functions and combine them. |

The example above generates to the following values:

| **Variable** | **Value** |
| --- | --- |
| gvar1 | any text |
| gvar2 | 001 |
| gvar3 | 001 |
| gvar4 | 1 |
| gvar5 | 0010011 |
| gvar6 | 3 |
| gvar7 | true |
| gvar8 | myobject |
| gvar9 | truemyobject0010011 |

**Scope**

Global variables are valid for all scenario steps of a scenario package. Local Variables are valid for the Scenario Step in which you define them.

If you define a variable as local and as global variable, the local variable wins. You can use all variables in the complete process flow. Global variables are available in the header of the integration framework message.

<Msg ... >

<Header>

...

<Variables>

<var id="mystring" value="value"/>

...

By default, the integration framework generates global variables into the XSL stylesheet of a transformation atom. This allows you to directly access them in the XSL coding. To deactivate the generation, select *Maintenance* → *Cfg Dev Environment* → *Generate Variables to XSL*.

**Value Assignment**

The integration framework assigns values to the variables at runtime directly before it triggers the process flow. The integration framework assigns values for global variables using literals during at design time or dynamically at runtime based on the incoming message.

**Using Variables in the Process Flow**

You can use global variables in your XSL coding. You can also use the variables in the parameter definition in the atom user interface. All parameters that support variables are marked with \* (asterisk).

In the XSL coding, access the global variable value using $, the leading vp abbreviation and the name of the variable in the following way: <xsl:value-of select="$vpmystring"/>. You can use it, if the generation into the XSL stylesheet is active or by using the incoming message (<xsl:value-of select="/vpf:Msg/vpf:Header/vpf:Variables/vpf:var[./@id='mystring']/@value"/>).

In the atom user interfaces you can note the variables similar to the XPath notation using $name. In a parameter field, you can use one variable multiple times and you can combine the variables with other variables, properties, and so on.

**Example**

#select \* from TABLE where key1=’$myVar1’ and key2=’$myVar2’

note.gif NOTE

To avoid cross-site scripting, the integration framework checks variables in an SQL statement of an SQL call atom at runtime. If the variable is set in quotations marks (apostrophe), the integration framework handles it as a string, otherwise as a number. If the variable is not in quotation marks and not a number, the integration framework replaces the value with NaN. If you use a variable, for example, for a table name, you can avoid this behavior by explicitly type-casting the variable to a string using $string(variable-name).

#### 3.4.7.5 In Memory Variables (Memory and Session)

The following in memory variables types are available in the integration framework:

* Memory variables
* Session variables

**Memory Variables**

Developers use memory variables at design time to hand over messages between multiple XSL transformations. In the process flow, all transformation stylesheets have read and write access. You can also use memory variables as placeholders for parameter definitions in an atom user interface.

**Session Variables**

Session variables are only available for scenario steps that the integration framework processes in a synchronous HTTP call where the scenario step runs in a session. The integration framework typically sets the session variables in the authentication phase and in the process flow, you can access the values. All transformation stylesheets have read and write access in the process flow. You can also use session variables as placeholders for parameter definitions in an atom user interface.

**Definition**

In the integration framework a container is available for memory variables and for session variables. The name of the session container is sessioninfo, the name for the memory container is ipobag.

The definition of a in memory variable happens by adding a variable to the container. You can do this in the stylesheet of a transformation atom or during authentication specifying a variable using the xsl:variable command. Choose any variable name for this.

<!-- set memory variable mem1 = ABC -->

<xsl:variable

name="memVar"

select="document('/com.sap.b1i.internal.xc/xml/ipobag.xml?mem1=ABC')"/>

<!-- set session variable sess1 = 123 -->

<xsl:variable

name="sessVar"

select="document('/com.sap.b1i.internal.xc/xml/sessioninfo.xml?sess1=123')"/>

**Scope**

Memory variables exist for the scenario step in which you define them. You can use memory variables in the process flow.

Session variables exist with the first incoming HTTP call of a scenario package and you can access them in all scenario steps triggered by synchronous HTTP calls. Session variables expire with the session expiration. They are the only variables you can use to communicate between different scenario steps inside one session.

**Value Assignment**

You assign memory variables and session variables with the same statement as the definition.

<!-- change value of memory variable mem1 to XYZ -->

<xsl:variable

name="memVar"

select="document('/com.sap.b1i.internal.xc/xml/ipobag.xml?mem1=XYZ')"/>

<!-- change value of session variable sess1 = 456 -->

<xsl:variable

name="sessVar"

select="document('/com.sap.b1i.internal.xc/xml/sessioninfo.xml?sess1=456')"/>

**Using In Memory and Session Variables in the Process Flow**

You can use in memory variables and session variables in XSL coding and as placeholders in the atom user interfaces in the parameter definition. All parameters supporting variables are marked with \*.

In XSL coding, access the value of a variable using the following XPath statements:

<xsl:value-of select="document('/com.sap.b1i.internal.xc/xml/ipobag.xml')/xci:ipo-bag  
 /xci:key[./@name='mem1']/@value)"/>

<xsl:value-of select="document('/com.sap.b1i.internal.xc/xml/sessioninfo.xml')/xci:session-  
 info/xci:key[./@name='sess1']/@value)"/>

In the atom user interfaces, use memory variables in parenthesis $(name), session variables in double parenthesis $((name)). In a parameter field, you can use a variable multiple times. Combine a variable with other variables, properties, and so on.

**Example**

#select \* from TABLE where key=’$(myMemVar)’

**Example**

#select \* from TABLE where key=’$((sess1))’

#### 3.4.7.6 Message Values

Developers use message values at design time to hand over messages between multiple XSL transformations. In the process flow, all transformation stylesheets have read and write access. You can also use message values as placeholders for parameter definitions in an atom user interface.

**Definition**

In the process flow, each flow atom adds a payload section to the integration framework message. The integration framework processes the message from one flow atom to the next. In the XSL stylesheet that is assigned to a transformation atom, you can define your individual XML output. Each atom has a unique name (atom1, atom2, …).

<Msg…>

<Header>

…

</Header>

<Body>

<Payload Role=”T” Type=”Handover”>…</Payload>

<Payload Role=”S”> …</Payload>

<Payload Role=”X” id=”atom1>

…

</Payload>

…

<Payload Role=”X” id=”atomn>

…

</Payload>

<Payload Role=”R” id=”atom0”>

<cache.list xmlns=””>

<cache.del count=”0”/>

<cache.deactive count=”0”/>

<cache.active count=”0”/>

<jobs.count=”0”/>

<cache.list/>

</Payload>

</Body>

</Msg…>

Any transformation atom can create any kind of XML structure, for example, the following, which is then immediately available as a payload section after atom processing:

<root>

<myheader>123</myheader>

<positions>

<posdef>definitions</posdef>

</positions>

</root>

**Scope**

Once the integration framework has created the section, it is part of the message until the end of the process flow. After processing, you can always access the information using the message log, if the message log is switched on.

**Value Assignment**

The integration framework assigns values in the transformation atom that creates the data section. No subsequent transformation can change it.

**Using Message Values in the Process Flow**

You can access the data section of any predecessor atom in your XSL coding and as a placeholder in the atom user interface in the parameter definition. All parameters supporting the message values are marked with \*.

In the XSL coding, access any value of a predecessor data section using XPath statements. The following XPath statement shows how to access data, produced by atomx, based on the example above.

<xsl:value-of select="/vpf:Msg/vpf:Body/vpf:Payload[./@id=’atomx’]/root/myheader

In the atom user interfaces, you can access element values from such a data section using the message values with a leading $ and the name in brackets. You can define the name of an existing element or filtered to a particular atom output section. In a parameter field, you can use a message value multiple times and you can combine it with other variables, properties, and so on.

**Example**

#select \* from TABLE where key1=’$[myheader]’ or key2=’$[atomx/posdef]’

#### 3.4.7.7 SLD Properties

To access connectivity parameters of a system, use SLD properties.

**Definition**

The SLD properties are predefined in the integration framework repository. The integration framework provides so called *SysTypes* that are linked to active and passive adapters. You define the connectivity with the corresponding properties.

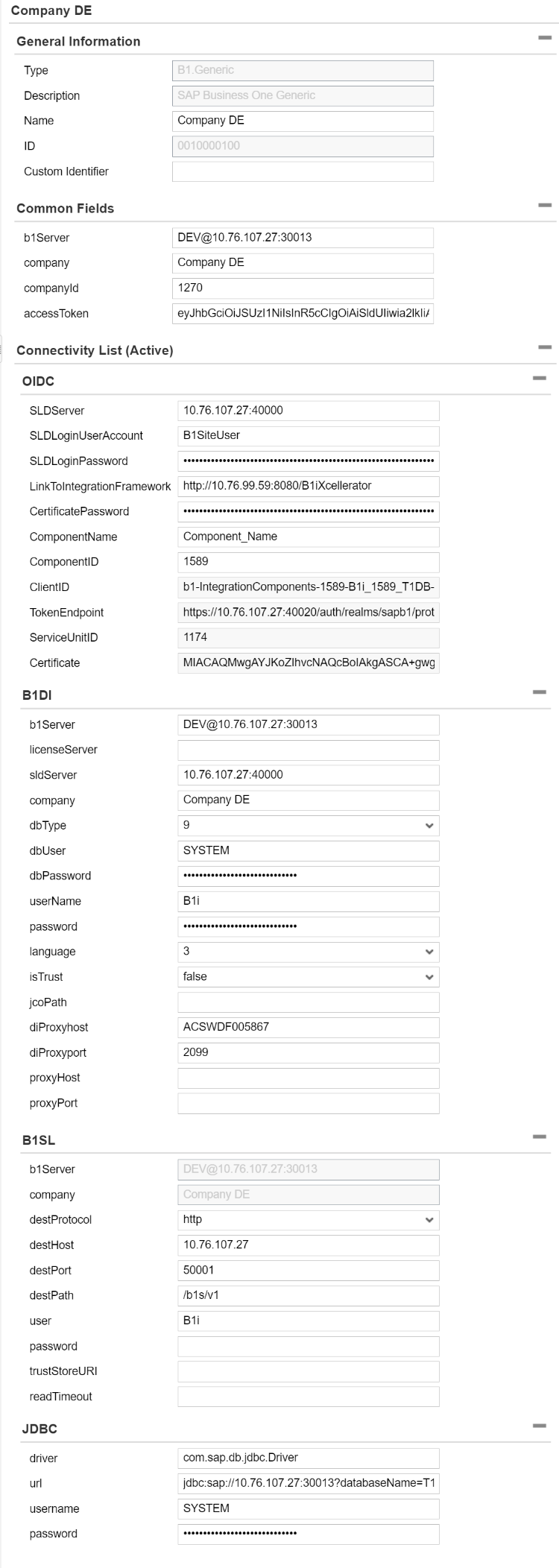
| **SysType (System Type)** | **Active Adapter** | **Passive Adapter** |
| --- | --- | --- |
| B1.2004 | B1DI |  |
| B1.2005 | B1DI, JDBC |  |
| B1.2007 | B1DI, JDBC |  |
| B1.8.8 | B1DI, JDBC |  |
| As of B1 9.0 | B1DI, JDBC, B1SL (HTTA for service layer) |  |
| B1 Generic | B1DI, JDBC, B1SL (HTTA for service layer) |  |
| BI.3.5.3 | HTTA |  |
| BI.7.0.3 | HTTA |  |
| ECC6.0 | RFCA, WSAN, WSAS | RFCP, WSAO, WSAR |
| S/4HANA on Premise | RFCA, WSAN, WSAS | RFCP, WSAO, WSAR |
| F.AnySystem | FILI | FILO |
| H.AnySystem | HTTA | HTTP |
| J.AnySystem | JDBC |  |
| R3.46C | RFCA | RFCP |
| R3.47.100 | RFCA | RFCP |
| R3.47.200 | RFCA | RFCP |
| W.AnySystem | WSAN, WSAS | WSAO, WSAR |
| P.AnySystem | FTPI, FTPP |  |
| T.AnySystem | CRON |  |
| E.AnySystem | SMTP, MAIR |  |

**Scope**

SLD properties are always available. They are saved in an XML document in the BizStore repository. At runtime, the values are available in the processed integration framework message for all scenario steps.

**Value Assignment**

When you set up the customer environment, you create the required entries in SLD selecting *SLD* → Create System. When you select a SysType, the integration framework requests the required properties to access the system using the linked adapters.



**Using SLD Properties in the Process Flow**

You can use SLD properties in your XSL coding. You can also use SLD properties in the atom user interfaces in the parameter definition. All parameters that support SLD properties are marked with \*.

In XSL coding, you can access the value of an SLD property. Use the XSL document function, which the integration framework technology overlays, to open the table document and use XPath statements to select the required data.

<xsl:variable name="sysiddoc"  
 select="document('/com.sap.b1i.system.sld.directory/SysId.xml/0010000101(Id)')"/>

<xsl:variable name="url"  
 select="$sysiddoc/sim:SysId/sim:ConnectivityList/sim:Connectivity  
 [./@ConnectivityTypeId='JDBC']/sim:Parameter[./@Key='url']/@Value"/>

In the atom user interfaces, enter an SLD property with $\*sysid.adapter.property\*

**Example**

#select \* from TABLE where key=’$\*0010000101.JDBC.url\*’

#### 3.4.7.8 Config Properties

**Definition**

To define general environment settings, use config properties. There are two types of config properties, the connectivity parameters and the runtime parameters.

**Scope**

Config properties always exist. They are saved in an XML document. At runtime, the values are available in the processed integration framework message for all scenario step.

**Value Assignment**

When you set up the customer environment, define the parameters by selecting *Maintenance* → Cfg Connectivity and *Maintenance* → Cfg Runtime.

**Using Config Properties in the Process Flow**

You can use config properties in the XSL coding. You can also use all properties in the atom user interfaces in the parameter definition. All parameters supporting the properties are marked with \*.

In XSL coding you can access the config property values with the following XSL coding:

<!-- Connectivity.Internet: Proxy Host-->

<xsl:value-of select="document('/com.sap.b1i.vplatform.ide/mode/mode.xml')/vpf:vPlatform/vpf:webhost"/>

<!-- Connectivity.Internet: Proxy Port-->

<xsl:value-of select="document('/com.sap.b1i.vplatform.ide/mode/mode.xml')/vpf:vPlatform/vpf:webport"/>

<!-- Connectivity.Send Email: SMTP Server -->

<xsl:value-of select="document('/com.sap.b1i.vplatform.ide/mode/mode.xml')/vpf:vPlatform/vpf:smtpserver"/>

<!-- Connectivity.Send Email: SMTP Port -->

<xsl:value-of select="document('/com.sap.b1i.vplatform.ide/mode/mode.xml')/vpf:vPlatform/vpf:smtpport"/>

<!-- Connectivity.Send Email: SMTP User -->

<xsl:value-of select="document('/com.sap.b1i.vplatform.ide/mode/mode.xml')/vpf:vPlatform/vpf:smtpuser"/>

<!-- Connectivity.Send Email: SMTP Password -->

<xsl:value-of select="document('/com.sap.b1i.vplatform.ide/mode/mode.xml')/vpf:vPlatform/vpf:smtppwd"/>

<!-- Runtime.B1i HTTP Port -->

<xsl:variable name="cdoc" select="document('/com.sap.b1i.vplatform.ide/mode/mode.xml')"/>

<xsl:choose>

<xsl:when test="string-length($cdoc/vpf:vPlatform/http)=0">8080</xsl:when>

<xsl:otherwise><xsl:value-of select="$cdoc/vpf:vPlatform/http"/></xsl:otherwise>

</xsl:choose>

<!-- Runtime.B1i HTTPS Port -->

<xsl:choose>

<xsl:when test="string-length($cdoc/vpf:vPlatform/https)=0">8443</xsl:when>

<xsl:otherwise><xsl:value-of select="$cdoc/vpf:vPlatform/https"/></xsl:otherwise>

</xsl:choose>

<!-- Runtime.B1i Server -->

<xsl:choose>

<xsl:when test="string-length($cdoc/vpf:vPlatform/server)=0">

<xsl:value-of select="document('/com.sap.b1i.internal.xc/xml/hostname.xml')/\*"/>

</xsl:when>

<xsl:otherwise><xsl:value-of select="$cdoc/vpf:vPlatform/server"/></xsl:otherwise>

</xsl:choose>

In the atom user interfaces, add a config property using $ and the name of the property wrapped with the ? character.

**Example**

#select \* from TABLE where key=’$?Connectivity.SendEmail:SMTPServer’ or key=’$?Runtime.B1iHTTPPort?’

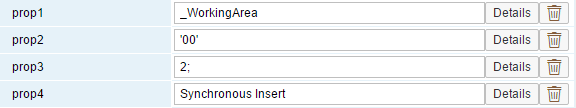
#### 3.4.7.9 Global Properties

Use global properties, if you cannot code a fixed value. This is the case, if you provide integration development to many customers and each of them needs a different setting, or you want to allow customers to frequently change the settings. At design time, introduce the properties, optionally with a default value and enumerations. During setup, the customer sets his or her values and the scenario works accordingly.

**Definition**

To define global properties, select *Scenarios* → *Step Design* → *Processing* → *PGlobal*or *Scenarios* → *Package Design* → *Definitions*.

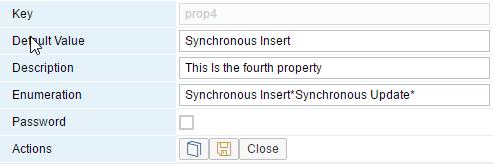
You can add and delete global properties and you can maintain the default values and enumerations for the properties. Default values are explicit values (literals) only.



To define a literal, enter the value. The integration framework interprets numeric values as numbers, it deletes leading zeroes. Explicitly mark the string as a string, such as for prop2.

[Details]

To add a description, enumerations, or to define a property as a password, click the [Details] button.



Description

Add a description. The integration framework displays it as quick info text on the setup user interface when the customer enters his or her values for the properties.

Enumeration

You can define an enumeration. It is a list of allowed values. Separate the values by \* (asterisk). Add an asterisk after the last value. In the setup user interface, the integration framework provides a dropdown list for the customer. The customer can select one of the defined values.

Password

To define a property that is a password, select *Password*. If you have already entered a default value, or you have defined enumerations, the integration framework removes the values. There are no default values for passwords. In scenario setup, the partner or customer enters a valid password. When the user enters the password, the integration framework conceals the value and displays dots instead.

**Using the XML Document**

Alternatively, you can provide descriptions, enumerations in an XML document. Open the XML document in your XML editor and enter the values.

/com.sap.b1i.vplatform.scenarios.design/vPac.<name>/vProp.xml

<vProp>

<prop id="prop1" value="\_WorkingArea" enum="" desc=""/>

<prop id="prop2" value="&apos;00&apos;" enum="" desc=""/>

<prop id="prop3" value="2;" enum="" desc=""/>

<prop id="prop4" value="Synchronous Insert" enum="Synchronous Insert\*Synchronous   
 Update\*" desc="This Is the fourth property" pwd="false"/>

</vProp>

**Scope**

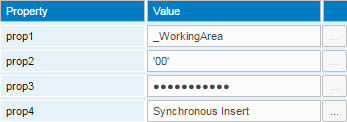
Global property definitions are available in the vProp.xml document in the scenario design area of the BizStore. At runtime, the values are available in the processed integration framework message for all scenario steps that belong to the scenario package.

By default, the integration framework generates the global properties into the XSL stylesheet of a transformation atom. It allows you to access the properties from your XSL coding.

To deactivate the generation, choose *Maintenance* → *Cfg Dev Environment* → *Generate Variables to XSL*.

**Value Assignment**

To define property default values at design time, choose *Scenarios* → *Step Design* → *Processing* → *PGlobal*or *Scenarios* → *Package Design* → *Definitions*. When you set up a scenario package for a customer, you can overwrite the default values in scenario setup. To open the scenario setup user interface, choose *Scenarios* → *Setup* → *[Data Mgt.]* → *Global Properties*.



If enumerations exist for the property, the integration framework enables the selection button […] to open a list with enumeration definitions, for example, for prop4.

All changes have immediate effect on running scenarios.

**Using Global Properties in the Process Flow**

You can use global properties in your XSL coding. You can also use all properties in the atom user interfaces in the parameter definition. All parameters supporting the properties are marked with \*.

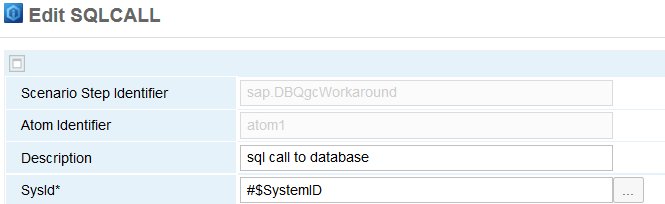
In XSL coding, access the value of a global property directly by entering $, the vp abbreviation and the name of the property (<xsl:value-of select="$vpprop1"/>) if the generation into the XSL is active, or by using the incoming message (<xsl:value-of select="/vpf:Msg/vpf:Header/vpf:Properties/vpf:prop[./@id='prop1']/@value"/>).

In the atom user interfaces you can note the global property similar to the XPath notation with $name. In a parameter field you can use a global property multiple times and you can combine the property with other variables, properties, and so on.

**Example**

#select \* from TABLE where key=‘$prop1’

**Example (Global Property in Atom)**



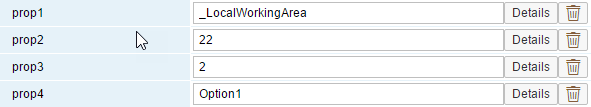
#### 3.4.7.10 Local Properties

Use local properties, if you cannot code a fixed value. This can, for example, be the case, if you provide your integration development to many customers and each of them needs a different setting or you want to allow your customer to frequently change the settings. At design time, you introduce the properties, optionally with a default value and enumerations. During setup the customer sets his or her values and the scenario works accordingly. Use local properties instead of global properties, if they are only valid for a scenario step of a scenario package.

**Definition**

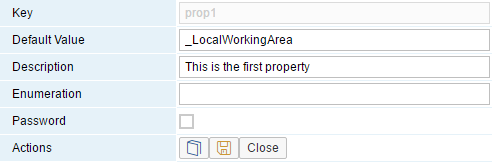
To define local properties choose *Scenarios* → *Step Design* → *Processing* → *PLocal*.

You can add and delete local properties and you can maintain the default values and enumerations for the properties. Default values are explicit values (literals) only.



[Details]

To define a description and enumerations, click the button.



Description

Add a description. The integration framework displays it as quick info text in the user interface when the customer maintains his or her values for the properties.

Enumeration

You can define an enumeration. It is a list of allowed values. Separate the values by \* (asterisk). Add an asterisk after the last entry. In the setup user interface, the integration framework provides a drop-down list for the customer. The customer can select one of the defined values.

Password

If you want to define a property that is a password, select *Password*. If you have entered a default value, or you have defined enumerations, the integration framework removes the values. Only in scenario setup, the partner or customer enters a valid password. When the user enters the password, the integration framework conceals the value and displays dots instead.

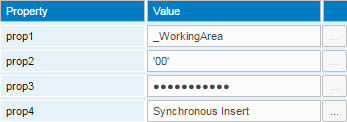
**Scope**

Local properties are valid for the scenario step. The integration framework saves them in an XML document. At runtime, the values are available in the processed integration framework message for the scenario step.

By default, the integration framework generates the local properties into the XSL stylesheet of a transformation atom. It allows you to access the properties from your XSL coding as a variable. To deactivate the generation, choose *Maintenance* → *Cfg Dev Environment* → *Generate Variables to XSL*.

**Value Assignment**

At design time, you define property default values. To open the scenario step design user interface, select *Scenarios* → *Step Design* → *Processing* → *PLocal*. When you set up a scenario package for a customer, you can overwrite the default values. To open the scenario setup user interface, select *Scenarios* → *Setup* → *[Data Mgt.]* → *Local Properties*.



The integration framework generates the user interface based on your definition at design time. The integration framework displays the selection buttons […] to open a list with enumeration definitions and displays the description text in the explanation area of the user interface.

All changes have immediate effect on running scenarios.

**Using Local Properties in the Process Flow**

You can use local properties in your XSL coding for the scenario step. You can also use properties in the atom user interfaces in the parameter definition. All parameters supporting the properties are marked with \*.

In XSL coding, access the value of a local property with $, the leading vp abbreviation and the name of the property (<xsl:value-of select="$vpprop1"/>) if you have activated the generation into the XSL. To retrieve the property value from the incoming message, use the following xPath:

<xsl:value-of select="/vpf:Msg/vpf:Header/vpf:Properties/vpf:prop[./@id='prop1']/@value"/>.

In the atom user interfaces, you can note the local property similar to the XPath notation with $name. In a parameter field you can use a local property multiple times and you can combine it with other variables, properties, and so on.

**Example**

#select \* from TABLE where key=’$prop1’

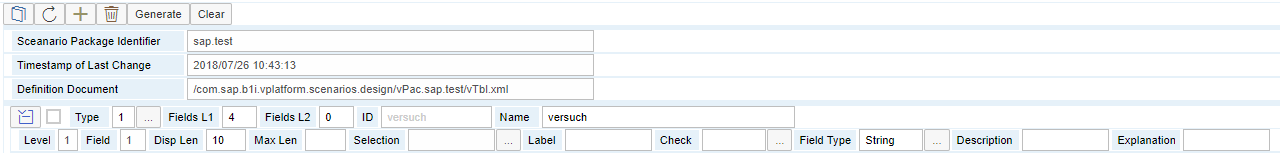
#### 3.4.7.11 Global Tables

Use the global table function, if defining properties does not cover what you need. Whenever you need multiple entries, each with a set of values instead of a single value, introduce a **global table of level 1**. **Global tables of level 2** support a structure of a 1:n relationship. You can define a set of values for parent and child segments. An example is a list of values, depending on combinations of sender and receiver systems.

**Definition**

* To define global tables, select Scenarios → Package Design → Definitions → Global Tables and click the  button.
* In the input field, enter the name of the global table and click *Select.*

The integration framework opens the following user interface:



* In the *Type* field, select the global table type.
* In the *Fields L1* field, enter the number of fields you provide in the global table.

The integration framework adds the fields to the user interface.

* If you have selected the table type 2, in the *Fields L2* field, enter the number of fields on level two.
* For each field, define the display length, the maximum length, a selection for the possible entries, the field label, checks for user entries, a description that is a tooltip in scenario setup and an explanation that becomes part of the Explanations area for the global table setup..

You can define multiple tables. When the integration framework displays the user interface, it displays a table with one header entry for each global table definition for the scenario package. In the header entry, you define the structure of the global table. To expand the definition, use the  (*Expand*) button to display all field definitions for the table. To collapse the entry, click the  (*Collapse*) button.

You can compare global tables to database tables. The integration framework allows you, like a database system, defining the structure of the table or tables. Additionally, you can define the user interface for editing the table. Later, it allows the customer to enter his or her values in scenario setup.

You define the structure definition and the definition of the setup user interfaces in this definition user interface. The definition of the setup user interface supports labels, selections, enumerations, plausibility checks and defining field lengths. The integration framework generates the user interface for editing the table based on the definitions. You can always change the definitions and regenerate the tables. The integration framework does not overwrite existing values.

**Buttons**



To create a new definition for a global table, click the button.



To delete definitions for selected global tables, click the button.

[Generate]

To create new tables and update existing tables, click the button.

[Clear]

To delete tables for which definitions no longer exist, click the button.

**Fields of the Header Section**

|  |  |
| --- | --- |
| **Variable** | **Description** |
| Type | Defines the table of type 1 or 2.  You can compare global tables of type 1 to database tables. Define the structure, and then add multiple records.  You can compare global tables of type 2 to two database tables having a parent child relationship. |
| Fields L1 | Define the number of fields on the first level |
| Field L2 | Define the number of fields on the second level |
| ID | Displays the table identifier |
| Name | Displays the table name |

**Position Section**

| **Variable** | **Description** |
| --- | --- |
| Level | Defines, if the field belongs to the header or the child segment. Possible values are 1 and 2. The integration framework generates the value. |
| Field | Field number  The integration framework generates the value. |
| Disp Len | Input field length |
| Max Len (maximum length) | With the maximum length, cut the field entry to the defined length. |
| Selection | The integration framework provides inbuilt selection functions, for example, systems in SLD.  Use them to provide users with a selection list. You can also define enumerations. Enter values, separated by \*. |
| Label | Enter a text. The integration framework displays it as the field label. |
| Check | Use plausibility checks, for example, isNumber to verify the field entries. |
| Description | Enter a description. The framework displays the description as a tooltip for the field in scenario setup. |
| Explanation | Enter an explanation for the field. The framework displays the entry in the Explanations area below the global table. |

**Scope**

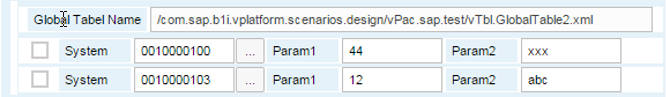
Global tables are valid for a scenario package. You can access a global table in all scenario steps of the package. The integration framework saves global table definitions to an XML documents in the scenario package design path in the BizStore:

/com.sap.b1i.vplatform.scenarios.design/ vPac.name/vTbl.name.xml

**Value Assignment**

To assign vales to global tables, select *Scenarios* → *Setup* → *[Data Mgt].* → *Table:vTbl.name*.

The customer can maintain his or her values in scenario setup either in the setup wizard or in the Data Management section.



The picture above displays the edit function for a global table of type 1.

Compare the generated user interface for editing with the definitions user interfaces. You find that the generated user interfaces are based on the definitions above.



To add a record to the table, click the button. If it is an edit user interface for a global table of type 2, the button is available on both levels.



To delete selected entries, click the button.

Global tables are saved as XML documents in the scenario package design path in the integration framework database:

/com.sap.b1i.vplatform.scenarios.design/ vPac.name/vTbl.name.xml

The XML structure shows an example for a global table of type 2 definition:

<table xmlns="" len="10,10,5,5,15" L1="2" L2="3" level="2" name="myTable3">

<?com.sap.b1i.system\_import protect?>

<sel id="1" label="Header 1" plaus="" max="10">Systems B1</sel>

<sel id="2" label="Header 2" plaus="" max="10">Systems R3</sel>

<sel id="3" label="Pos 1" plaus="" max=""/>

<sel id="4" label="Pos 2" plaus="IsAlpha" max=""/>

<sel id="5" label="Pos 3" plaus="" max=""/>

<row open="true">

<col>0010000103</col>

<col>0010000109</col>

<row>

<col>111</col>

<col>a</col>

<col>122</col>

</row>

<row>

<col>222</col>

<col>b</col>

<col>244</col>

</row>

<row>

<col>333</col>

<col>c</col>

<col>67</col>

</row>

</row>

...

**Using Global Tables in the Process Flow**

You can access global tables in the process flow from each XSL transformation using the XSL document function, which is overlaid by the integration framework technology. Open the table document and use XPath statements for data selection.

<xsl:variable name="mytbl1"   
 select="document('/com.sap.b1i.vplatform.scenarios.design/vPac.name/vTbl.myTable1.xml')"/>

<xsl:variable name="mytbl3"  
 select="document('/com.sap.b1i.vplatform.scenarios.design/vPac.name/vTbl.myTable3.xml')"/>

...

<xsl:value-of select="$mytbl1/table/row[2]/col[3]"/> <!-- A -->

<xsl:value-of select="$mytbl3/table/row[1]/row[2]/col[3]"/> <!-- 244 -->

You can also use global tables in the parameter definition of an atom. All parameters supporting global tables are marked with \*.

In the atom user interfaces, you can note the global tables following the supported syntax. In a parameter field, you can use one global table entry multiple times and you can combine entries with other variables, properties, and so on.

| **Expression** | **Description** |
| --- | --- |
| ${tbl[row,col]} | Placeholder for the value of a global table of type 1.  row can be a number (for example, 2) or a condition (for example, 2=’233’ or 2=233) |
| ${tbl[row,row,col]} | Placeholder for the value of a global table of type 2.  row can be a number (for example, 2) or a condition (for example, 2=’233’ or 2=233) |

**Example**

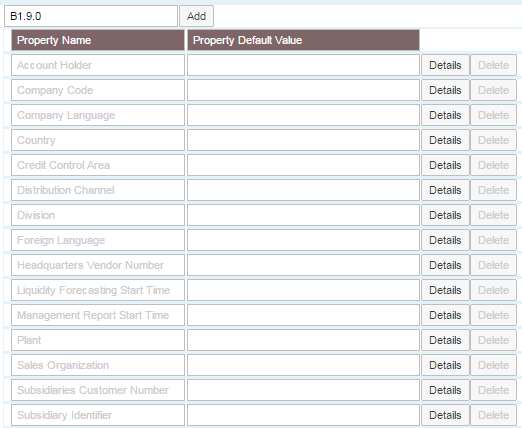
#select \* from slsp where key1=’${myTable3[1=0010000115,1=555,2] or key2=’${myTable1[1=0010000112,2]}

#### 3.4.7.12 SysType Properties

SAP delivers properties for SLD system types. You can add your own properties to the system types.

**Definition**

To display properties delivered by SAP and define properties for system types, select *SLD* → *SysType Properties*.



The integration framework displays property names and default values for system type properties that SAP delivers. You cannot change properties delivered by SAP.

[Add]

If you want to add system type properties for a system type, scroll down to the system type and click the [Add] button for the system type.

The integration framework adds a new entry at the end of the property list.

[Details]

To provide property details, click the [Details] button.

The integration framework opens a user interface. You can edit the following:

* Name

Enter the property name.

* Default Value

If you want to provide a default value for the property, enter the default value.

* Description

Enter a description for the property.

* Enumeration

If you want to define a list of values, where the user selects one, enter the values in the following way:

Value1\*Value2\*Value3\*



Save the property details and close the user interface.

[Delete]

To delete a system type property, click the button. You can only delete properties that are not delivered by SAP.

**Scope**

SysType properties always exist. They are saved in an XML document. At runtime their values are available in the processed integration framework message for all scenario steps.

**Value Assignment**

At design time you can define default values.

To define property default values at design time, select *SLD* → *SysType Properties*. When you set up a scenario package for a customer, you can overwrite the default values. To open the SLD, select *SLD*, select the system for which you want to define values and select the [Properties] button.

**Using SysType Properties in the Process Flow**

SysType properties are part of the integration framework message at runtime. You can access the properties in the scenario package. They are available in the <SysTypeProperties> tag. The integration framework provides the key, value, direction, the property source and the value source.

If you have developed a scenario in SAP Business One integration for SAP NetWeaver prior to the 9.0 release, you can change the representation of SysType properties. Select *Scenarios* → *Step Design* and select the *B1iSN88 Compatibility Mode* checkbox. In this way, you do not have to change how you access the properties in the process flow.

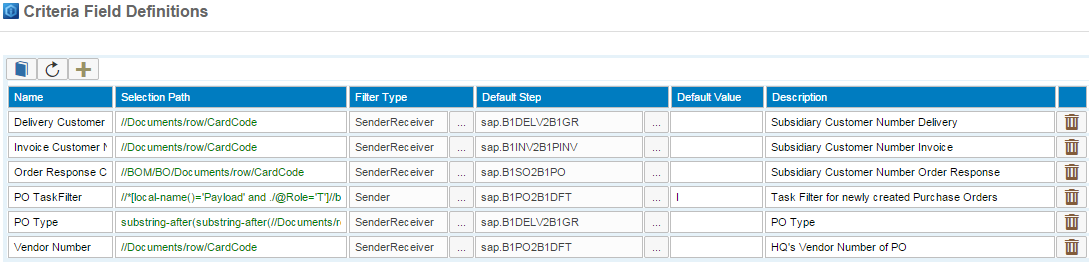
#### 3.4.7.13 Criteria Fields

Criteria fields represent XPath expressions in a message. Define criteria fields for XPath expressions you want to use for filtering or routing messages from certain senders or receivers.

**Definition**

To define criteria fields, select *Scenarios* → *Package Design* → *Definitions* → *Criteria Fields*.

You can add or delete criteria fields.





To add a line to the table, click the button.

Name

Enter the criteria field name.

Selection Path

Enter the selection path (XPath) to the field. Use $msg for the incoming message.

Filter Type

You can define the following:

* Sender: In scenario package setup, the integration framework only displays this criterion in the sender filter section. You can only set a value in this section.
* SenderReceiver: In scenario package setup, the integration framework only displays this criterion in the *SenderReceiver* filter section. You can only set a value in this section.
* Both: In scenario package setup, the integration framework displays this criterion in the *Sender* filter and in the *SenderReceiver* filter section. You can set a value in both sections.

Default Step

You can define the following:

* By default, the integration framework sets the value to All. In scenario setup, the integration framework displays the criteria field for all scenario steps in scenario package setup and you can set values.
* If you select a scenario step, the integration framework displays the criteria field only for the scenario step and you can set a value.

Default Value

Enter a default value. The integration framework provides this value in the scenario package setup as a default value for the criteria field. You can overwrite the default value in scenario package setup.

Description

Enter a description for the criteria field.



To delete a criteria field definition, click the button.

**Scope**

Criteria fields are valid for all scenario steps of the scenario package for which you have defined them.

**Value Assignment**

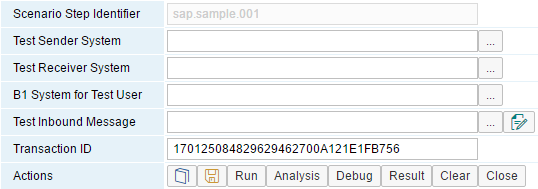
Assign values to the criteria fields in the setup configuration wizard in the third *Filter Definitions* step.

For more information, see the *Operations Guide Part 1*, section *Setting Up a Scenario Package*

At runtime, the integration framework only processes messages that fulfill the filter definitions you have defined. You can find messages that do not fulfill the filter criteria in the *Filtered* section of the message log.

### 3.4.8 Testing and Debugging the Process Flow

The integration framework provides a test and debugging environment for the process flow. To access the test environment click the [Processing] button in scenario step design and click the [Test] button in the graphical flow designer. The integration framework opens the following user interface:



Test Sender System

If you have already defined the inbound phase, select a test sender system. The test environment needs the information, if you use, for example, a call B1 object atom in the process flow, and you have defined the *SysId* of the called system as the Sender System, this information gives the test environment the sender context.

Test Receiver System

If you have already defined the outbound phase, you can select a receiver system.

B1 System for Test User

If you define a call to a SAP Business One system in the process flow, this field provides the information about the user the integration framework uses for the call.

Test Inbound Message

Create or select the inbound test message. Use the internal XML editor to provide the message content.

If you have activated and run the scenario step and you have selected the *Record Test Message at Runtime* checkbox in the *Scenario Development Configuration* user interface (Maintenance → Cfg Dev. Env.), the rectest.xml test message is available for selection.

Transaction ID

The integration framework displays a transaction ID after you have clicked the [Run] button.

[Run]

To start the test, click the button.

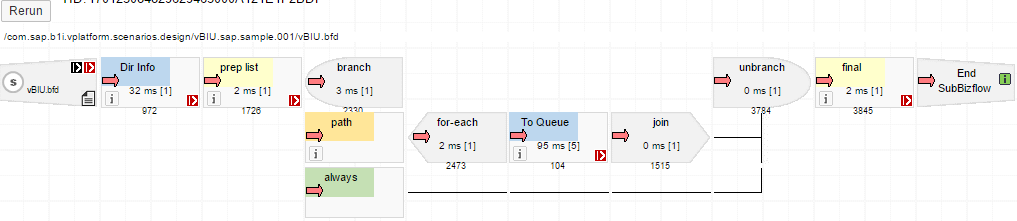
[Analysis]

To analyze the scenario step on B1iP level, click the button. The integration framework displays the following:



[Debug]

To display the graphical flow designer in debug mode, click the button. The integration framework displays, for example, the following:



The process flow in the illustration above belongs to the sap.sample.000 scenario step of the sap.B1if.Samples scenario package.



For each atom, click the red arrow to display the inbound message for the atom. This enables you checking how the message changes running through the process flow. The step picks up files from a folder. In the folder, there are five files available. In for-each processing, select one of the five calls, you want to display.

To display technical inbound parameters of processing, click the button.

Additionally, the integration framework displays the processing time in milliseconds and the size of the message.



To display the configuration details of the atom, click the icon.



The integration framework displays the test result with status and more detailed information.

[Result]

To display the processing result, click the button.

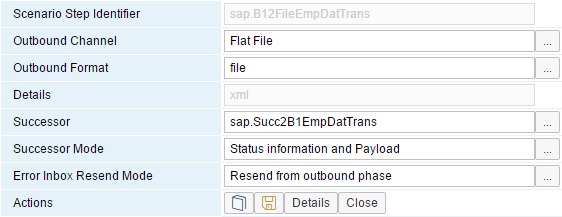
[Clear]

To clear integration framework processing from transactions that have ended in INCOMMIT status, click the button.

## 3.5 Outbound

### 3.5.1 Outbound User Interfaces

To open the outbound definition, in the scenario step definition user interface, click the [Outbound] button.



Outbound Channel

To select the technical adapter for the outbound phase, click the […] button. The following channels are available:

* Business Process

The integration framework hands over data to a business process

* SAP Business One

The integration framework inserts data using the SAP Business One DI API, the SAP Business One service layer or SQL statements

* SAP ERP

The integration framework hands over data using a remote function call.

* Flat File

The integration framework saves data in a flat file in the file system.

* Web Service

The integration framework hands over data using a Web service call

* HTTP Call

The integration framework hands over data with an HTTP call.

* Database

The integration framework hands over data using SQL.

* Void

The integration framework does not use the outbound phase and does not hand over any data.

Outbound Format

To select the outbound format, click the […] button. The values depend on the selected outbound channel. The following values are available:

* DI Object, DI Service, SQL, B1i SQL, service layer for SAP Business One,
* rfc for the SAP ERP outbound channel
* file for the Flat File outbound channel
* ws for the Web Service outbound channel
* http for the HTTP outbound channel
* jdbc for the Database outbound channel
* queue for business process
* void for the Void outbound channel

The selected format corresponds to the message that the scenario step processing creates. For more information about the correct outbound schema, see the integration framework reference guide 05 about the available APIs.

Details

The read-only field displays detailed settings. To enter detailed settings, click the [Details] button.

Successor

The integration framework allows you to define another scenario step which the integration framework calls after it finishes the processing of the current scenario step. You can use this mechanism to send back result information to the sender system, for example. Select a scenario step from the list. The list displays only scenario steps, assigned to the same scenario package and that have the Predecessor inbound type.

Successor Mode

This field is only relevant if you trigger another scenario step when the integration framework has finished the processing of the current scenario step. Enter the type of information you are interested in:

* Status information only

The integration framework hands over the summarized execution status of the predecessor scenario step to the successor step.

**Example**

<?xml version="1.0"?>

<vPStatus offline.id="" succ="sap.ScenarioStep2" ReceiverObjectTypeId="" SenderObjectTypeId="sap.ScenarioStep1" ReceiverSysId="0010000100" SenderSysId="0010000000" EndTimeStamp="20140423142735" BeginTimeStamp="20140423142734" SubMessageId="1" B1IMessageId="140423142734739111450A3B26D56049" Phase="final" Message="Processing completed" no="0000" status="success" xmlns=""/>

* Status information and payload

The integration framework hands over the execution status and the integration framework message payload content to the successor step.

* Status information and complete message

The integration framework hands over the execution status and the complete integration framework message to the successor step.

Error Inbox Resend Mode

If you process your scenario step on the receiver side, you can determine, from which place in processing the integration framework resends messages in the error inbox. You have the following options:

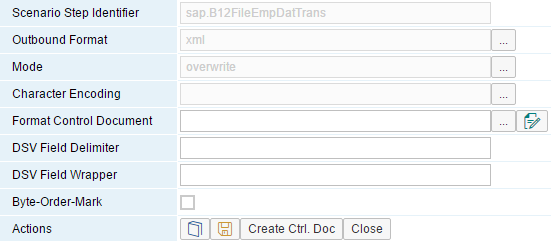
* Resend from outbound phase
* Resend including mapping

[Details]

Depending on the selected outbound channel and outbound format, enter some more details. To define the details, click the [Details] button. The integration framework requires details for the following outbound channels and formats:

* Outbound channel is Flat File
* Outbound channel is Web Service
* Outbound format is DI object
* Outbound format is DI service
* Outbound format is service layer (SL) object
* Outbound format is SQL
* Outbound channel is Database

Flat File Details



Outbound Format

To define the outbound format of the flat file, click the […] button. The following values are available:

* xml (XML representation)
* dsv (delimiter-separated value)
* txt (offset-defined values)

Character Encoding

Optional character encoding is necessary to apply the technical representation of characters according to the country or system-specific needs. The following entries are available: ISO 8859-1, ISO 8859-2, ISO 8859-3, ISO 8859-4, ISO 8859-5, ISO 8859-6, ISO 8859-7, ISO 8859-8, ISO 8859-9, ISO 8859-11, ISO 8859-13, ISO 8859-14, ISO 8859-15, ISO 8859-16, US-ASCII, EBCDIC, Shift-JIS, EUC-JP, ISO-2022, GB2312, EUC-KR, Big5, Unicode UTF-7, Unicode UTF-8, Unicode UTF-16, ISO-10646-UCS2, ISO-10646-UCS4. The default is ISO-8859-1.

Format Control Document

This parameter is only relevant for the txt outbound format. Enter the name of the control document. Make the document available in the BizStore base directory of the scenario step.

For information about how to import the control document, see the integration framework reference guide 01. For more information about the format of the control document, see the reference guide 05.

DSV Field Delimiter

This parameter is only relevant for the outbound format dsv (delimiter-separated values). Enter the delimiter character. The default is a comma.

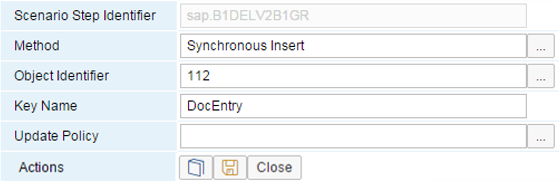
DSV Field Wrapper

This parameter is only relevant for the outbound format dsv. Enter the wrapper character. The default is empty. The field wrapper is optional. Use it to avoid that the integration framework misinterprets a field delimiter character inside a value as a field delimiter.

Byte-Order-Mark

If you select the field, the integration framework creates a byte order mark at the beginning of the text stream that signals the order of the text stream.

SAP Business One DI Object Details



Method

To define the method for handing over data to B1, click the […] button. The following values are available:

* Synchronous insert
* Synchronous insert with fallback to update
* Synchronous update
* Synchronous update with fallback to Insert
* Synchronous delete

Object Identifier

To define the identifier of the SAP Business One object, click the […] button. For more information, see the SAP Business One DI documentation.

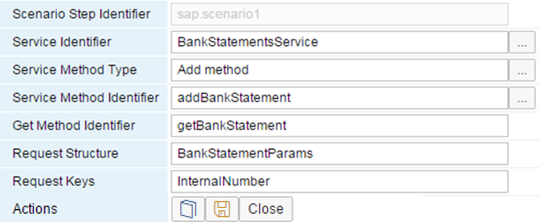
Key Name

Define the name of the primary key. If there are multiple keys, enter the keys separated by comma. For more information, see the SAP Business One DI documentation.

Update Policy

For update, you can either delete and insert a record, or use the default behavior.

SAP Business One DI Service Details



Service Identifier

To define the service identifier of the B1 service, you want to call, click the […] button. For more information, see the SAP Business One DI API documentation.

Service Method Type

To select the method you want to call to hand over data to B1, click the […] button. The following method types are available:

* Add method
* Update method
* Remove method

Service Method Identifier

Define the name of the method call. For more information, see the SAP Business One DI API documentation.

Get Method Identifier

Define the name of the get method call. This is necessary because the integration framework performs each data hand over using a select with a subsequent data merge and storage. For more information, see the SAP Business One DI API documentation.

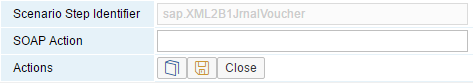
Request Structure

Define the name of the request structure. For more information, see the SAP Business One DI API documentation.

Request Keys

Define the key names of the request method. For more information, see the SAP Business One DI API documentation.

Web Service Details



SOAP Action

By default, the integration framework hands over an empty SOAP action. Some older Web services require the definition of the called method using the SOAP action. For this purpose, you can define the SOAP action in the outbound definition.

SQL or Database Details



SQL Processing Mode

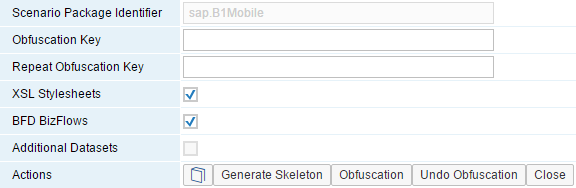
This setting is relevant for multiple SQL calls only. The default setting is multiple. The following options are available:

* Single concatenates all SQL statements with a semicolon to one SQL statement. The integration framework only hands over one SQL statement to the database system. Some relational database management systems, for example, DB2 do not support this feature.
* Multiple processes each SQL statement one after the other.

# 4 Obfuscating Development Content

To make your code difficult to read for human beings, use the obfuscation function.The software is unintelligible but still functionally equivalent to the original code.

The integration framework allows you to obfuscate your individual scenario package content and datasets, you have developed using your development prefix.



If you obfuscate scenario package content, the following happens:

* You cannot debug the scenario package.

In your WebDAV client, you cannot access the content of the XSL and the BFD documents. The content is unreadable.

* In the *Scenarios* → *Control* function, you cannot access the processing section. The integration framework displays flow encrypted.
* In the *Scenarios* → *Step Design* function, you cannot open the processing.

**Prerequisites**

Archive or export your scenario package content before obfuscating it. If you forget the obfuscation key, you cannot undo this function.

* To archive your scenario package content, select Package Design → Tools → Save current version to archive.
* To export your scenario package, select Scenarios → Export.

Obfuscation Key

Enter an obfuscation key. Use digits, upper-case, and lower-case characters for the key. The key can have any length. A longer key improves the protection.

Repeat Obfuscation Key

Repeat the obfuscation key.

XSL Stylesheets

To obfuscate all XSL stylesheets of the scenario package, select this option.

After obfuscation, you cannot read the xform atoms any more.

BFD BizFlows

To obfuscate the bfd BizFlows of the scenario package, select the option.

Additional Datasets

To obfuscate additional datasets, select the checkbox. The integration framework identifies the datasets by the development prefix, you have defined in the configuration function of the development environment (Maintenance → Cfg. Dev. Env.). It searches the BizStore and includes datasets that contain .<your\_development\_prefix>..

[Generate Skeleton]

Before obfuscating XSL stylesheets or bfd BizFlow documents, generate skeletons of all scenario steps. The skeletons enable debugging of obfuscated content. If you generate skeletons, the integration framework generates the vBIU.skeleton.bfd document in the basic directory of the scenario package.

* To indicate obfuscated content in the scenario step, the start atom is orange:

….

* You can click the debug arrows to displays the message.
* You can display detailed information for all atoms delivered by SAP.
* Partner content in xform atoms, the path condition in the path atom, or the statements in an SQL call remain unreadable.

[Obfuscation]

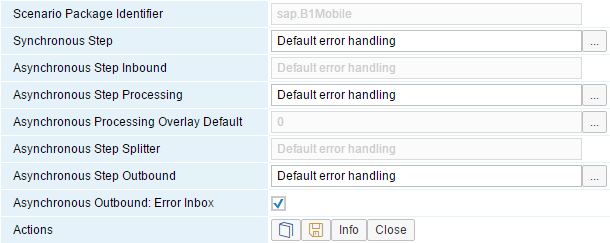
To obfuscate your scenario package content, click the button.

[Undo Obfuscation]

To undo obfuscation, click the button.

# 5 Developing an Individual Error Handling Step

## 5.1 Introduction



The integration framework provides a default error handling. To define settings for the default error handling, select Maintenance → Cfg Error Handling.

On scenario package level, you can additionally define individual error handling steps that replace the default error handling. If you define individual error handling steps, the integration framework hands over the message to individual steps and deletes the message.

**Designing an Error Handling Step**

For an individual error handling step, use the internal queue inbound type.

In scenario step processing, define what you require for your scenario package error handling.

The message that the integration framework sends to the queue of your error handling step contains the following information:

* All processing information until the error has happened.
* In the payload role S section, find all header attributes.
* In the ErrorInfos additional payload, find detailed error descriptions.

Use the information in the message to provide your individual error handling.

## 5.2 Error Handling for Synchronous Scenario Steps

Synchronous scenario steps consist of an IPO (inbound, processing, outbound) step for inbound, and a processing (vBIU) IPO step. The processing IPO step returns the result to the original sender.

**Default Error Handling**

If an error occurs, the following default error handling is available:

* The integration framework sends an HTTP error or SOAP fault to the sender system.
* The integration framework writes an entry to the *Failure* section of the message log.

**Scenario Error Handling**

Additionally, you can define and call an individual error handling scenario step.

* To call an individual scenario step, select Scenarios → Package Design → [Definitions] → Error Handling.

The integration framework displays scenario steps of the package with the internal queue inbound type.

* Select the error handling step.

The integration framework writes the message for which an error occurs to the internal queue. The message contains all information the integration framework has gathered until the error has occurred.

## 5.3 Error Handling for Asynchronous Scenario Steps

### 5.3.1 Asynchronous Processing Overview

The processing of an asynchronous scenario step consists of seven internal IPO (inbound processing outbound) steps that are connected with each other by internal queues. Each IPO is one transactional unit and each has an individual error handling concept.

The inbound type, the outbound type and the number of receivers determine, which IPO steps are involved in asynchronous message processing:

The following illustration gives an overview of asynchronous message processing.

**Inbound**

The sender system triggering the inbound dispatcher, the inbound DB trigger, the inbound trigger and the successor steps trigger the integration framework processing.

**Processing**

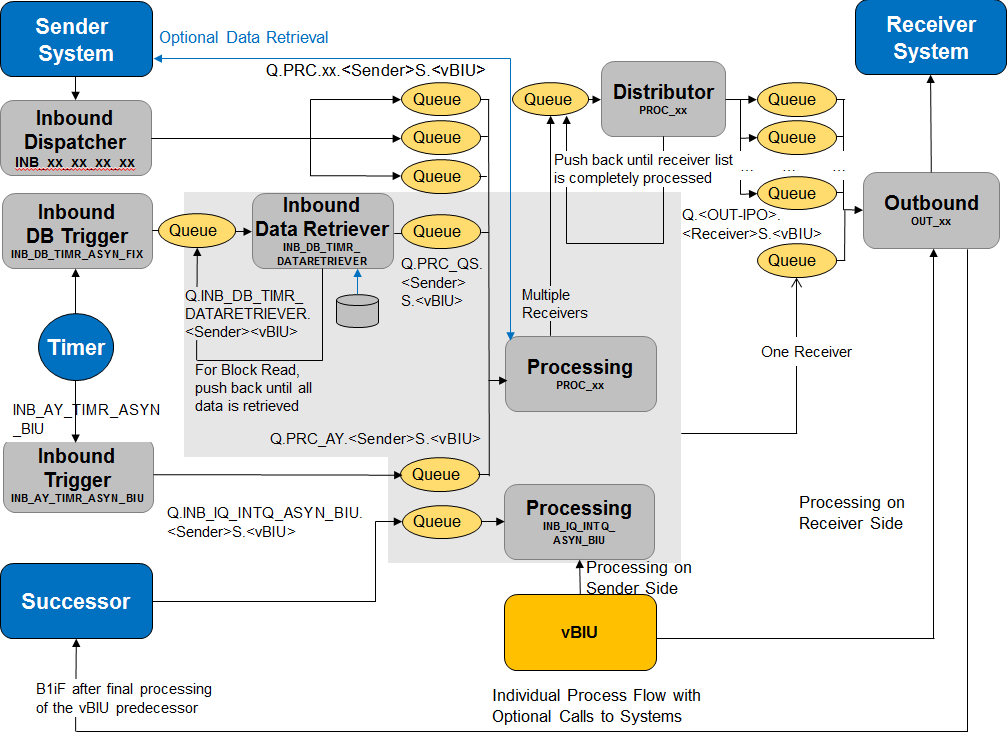
In processing, the inbound data retriever retrieves data from a database. The processing step retrieves data from the sender system, for example, for SAP Business One inbound, and calls the vBIU.

**Distributor**

The distributor step handles message processing for multiple receivers.

**Outbound**

The outbound step hands over message to the receiver system and to a successor step.



There is a difference between an IPO step and the IPO instance. Each IPO step can run multiple times in parallel. A running IPO is an IPO instance. The IPO is typically set up per sender system and in some cases per sender system and scenario step. Find information about the instance granularity in the tables of each chapter below.

### 5.3.2 Inbound Dispatcher

The inbound dispatcher IPO step dispatches calls coming into the integration framework. You do not have the option to define an individual error handling.

The step has the following tasks:

* It assembles the basic integration framework message.
* It detects scenario steps that have subscribed to the incoming call.

The inbound definition of the scenario step contains the according identification information.

* It triggers each subscribed scenario steps using the Q.PRC\_xx.<Sender>, S.<vBIU> internal queue.

The inbound dispatcher IPO step supports the following inbound types:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **System Type** | **Inbound Dispatcher** | | **Outbound Queue** | |
| **IPO ID** | **IPO Instance** | **Queue** | **Stream** |
| SAP Business One | INB\_B1\_EVNT\_ASYN\_EVT | vP.<Sender>.in\_BEAE | Q.PRC\_B1.<Sender> | S.<vBIU> |
| SAP ERP | INB\_R3\_IDOC\_ASYN\_XPT | vP.<Sender>.in\_RIAX | Q.PRC\_R3.<Sender> | S.<vBIU> |
| File System | INB\_FI\_EXST\_ASYN\_XPT | vP.<Sender>.in\_FEAX | Q.PRC\_Fix.<Sender> | S.<vBIU> |
| INB\_FI\_EXST\_ASYN\_NAM | vP.<Sender>.in\_FEAN | Q.PRC\_Fin.<Sender> | S.<vBIU> |
| INB\_FI\_EXST\_ASYN\_FIX | vP.<Sender>.in\_FEAF | Q.PRC\_FIi.<Sender> | S.<vBIU> |
| INB\_FI\_EXST\_ASYN\_FSL | vP.<Sender>.in\_FEAL | Q.PRC\_FIf.<Sender> | S.<vBIU> |
| Internal Queue | INB\_IQ\_INTQ\_ASYN\_QS | vP.<Sender>.in\_IQ.<vBIU> | Q.PRC\_QS.<Sender> | S.<vBIU> |

The table above displays the system types that trigger the inbound dispatcher, the related IPO step identifiers, the IPO instances at runtime and the queues and streams the inbound dispatcher writes to.

**Default Error Handling**

If an error occurs, the integration framework triggers the following default error handling:

* The framework deactivates the inbound dispatcher instance for one minute.

The framework triggers the deactivation and activates the instance again after one minute. This avoids high load on the integration framework server, if problems occur in the inbound channel, for example, if the integration framework tries to pick up a corrupt file from the file system.

* The integration framework writes an entry to the *Failure* section of the message log.

### 5.3.3 Inbound Database (DB) Trigger

A timer triggers the inbound database (DB) trigger IPO step. No default and individual error handling is available for the IPO step, because it does not handle critical data processing.

The inbound database trigger IPO step triggers the inbound data retriever IPO step, unless the step still runs, because the inbound database trigger IPO step has triggered it before.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **System Type** | **Inbound Dispatcher** | | **Outbound Queue** | |
| **IPO ID** | **IPO Instance** | **Queue** | **Stream** |
| Sche-duler | INB\_DB\_TIMR\_ASYN\_FIX | vP.<Sender>.in\_DB | Q.INB\_DB\_TIMR\_DATARETRIEVER.<Sender> | <vBIU> |

### 5.3.4 Inbound Trigger

To trigger timer-based inbound processes, a timer triggers the inbound trigger IPO step. This step triggers the processing IPO step.

No default and individual error handling is available for the IPO step, because it does not handle critical data processing.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **System Type** | **Inbound Dispatcher** | | **Outbound Queue** | |
| **IPO ID** | **IPO Instance** | **Queue** | **Stream** |
| Sche-duler | INB\_AY\_TIMR\_ASYN\_ BIU | vP.<Sender>.in\_TI.  <vBIU> | Q.PRC\_AY.<Sender> | <vBIU> |

### 5.3.5 Inbound Data Retriever in Processing

To retrieve data from a database, the inbound database trigger IPO step calls the inbound data retriever IPO step.

The inbound data retriever IPO step has the following tasks:

* It retrieves the data from the database.
* It assembles the basic integration framework message with the database data in the S payload section.
* It puts the integration framework message into an internal queue to trigger data processing.
* If the integration framework retrieves data from the database block by block, the IPO step calls itself using an internal queue until it has retrieved all data.

### 5.3.6 Processing

The processing IPO step has the following tasks:

* It retrieves data from the sender, if applicable, for example, for SAP Business One inbound.
* If processing happens on the sender side, the processing IPO step calls the vBIU (processing phase of the scenario step.)



| **System Type** | **Inbound Dispatcher** | | **Outbound Queue** | |
| --- | --- | --- | --- | --- |
| **IPO ID** | **IPO Instance** | **Queue** | **Stream** |
| Sche-duler | INB\_DB\_TIMR\_DATA RETRIEVER | vP.<Sender>.dr\_DB.<vBIU> | Q.PRC\_QS.<Sender> | S. <vBIU> |
| Queue | PRC\_xx | vP.<Sender>.prc\_xx | Q.PRQ\_xx.<Sender> or Q.OUT\_xx.<Receiver> |
| Queue | INB\_IQ\_INTQ\_ASYN\_ BIU | vP.<Sender>.in\_BIU.<vBIU> |

**Default Error Handling**

If an error occurs, the integration framework triggers the following default error handling:

* The integration framework deactivates the inbound dispatcher instance for one minute.

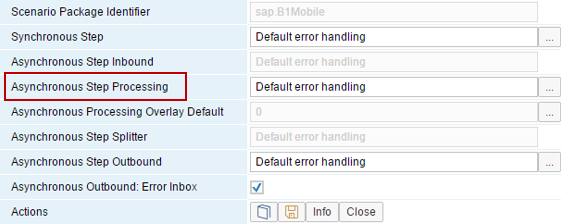
This action avoids high load on the server due to not resolvable problems in an inbound channel, for example, because of a wrong SQL statement.

* The integration framework writes an entry to the *Failure* section of the message log.

**Scenario Package Error Handling**

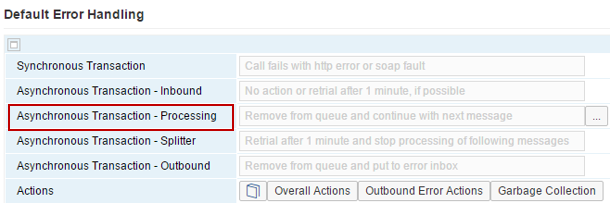
Additionally, you can do the following:

* Call an individual scenario step.
* To call an individual error handling scenario step, select *Scenarios* → *Package Design* → *[Definitions]* → *Error Handling*.
* For the *Async. Processing: Step* field, select a scenario step of the scenario package that has the internal queue inbound type and that performs the individual error handling.



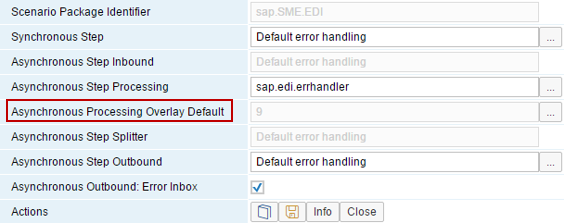
* The integration framework optionally deletes the message from the inbound queue.

It depends on the settings for default error handling, whether the integration framework deletes the message. To define the default error handling, select *Maintenance* → *Cfg Error Handling*.

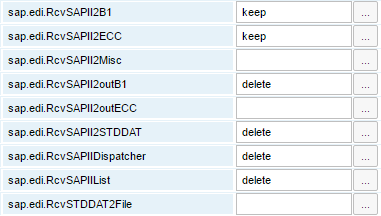


If you select an individual scenario step to handle errors, the integration framework deletes the message from the queue, regardless of the default setting. It is the general concept that the individual error handling step takes over the responsibility for erroneous messages. The default error handling hands over the message to individual error handling, deletes the message from the queue and processing continues. The individual error handling step is responsible for correct processing and takes care that the information in the message is not lost.

However, to overlay the above described behavior, define settings for the *Async. Processing: Overlay Default* field.



For each scenario step of the package, you can select either to keep messages in the queue or to delete messages from the queue.



The value of the *Async. Processing: Overlay Default* field displays for how many steps you have selected individual settings.

### 5.3.7 Distributor

The integration framework only calls the distributor IPO step, if the message has more than one receiver. The integration framework hands over the message to the processing IPO step including the receiver list. The distributor IPO step performs the value mapping and hands over the message to the first receiver system. Then, the integration framework puts the message back into the queue triggering itself again for the next receiver.

| **System Type** | **Inbound Dispatcher** | | **Outbound Queue** | |
| --- | --- | --- | --- | --- |
| **IPO ID** | **IPO Instance** | **Queue** | **Stream** |
| Queue | PRQ\_xx | vP.<Sender>.prq\_xx | Q.PRQ\_xx.<Sender> | S.<vBIU> |

**Default Error Handling**

If an error occurs, the integration framework triggers the following default error handling:

* The integration framework deactivates the distributor instance for one minute.

It avoids high load on the server.

* The integration framework writes an entry to the *Failure* section of the message log.

No individual error handling is available.

### 5.3.8 Outbound

The outbound IPO step hands over the generated payload to the receiver system. If the integration framework performs processing on the receiver side, it calls the vBIU processing.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **System Type** | **Inbound Dispatcher** | | **Outbound Queue** | |
| **IPO ID** | **IPO Instance** | **Queue** | **Stream** |
| Queue | OUT\_xx | vP.<Receiver>.out\_xx |  |  |

**Default Error Handling**

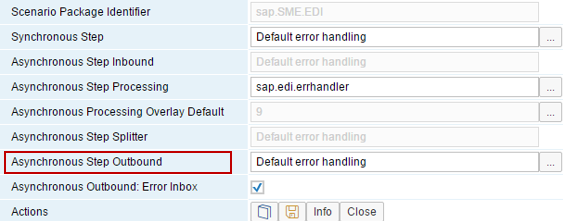
If an error occurs, the integration framework triggers the following default error handling:

* The integration framework writes an entry to the *Failure* section of the message log.
* The integration framework deletes the message from the inbound queue.
* The integration framework puts the message into the error inbox.

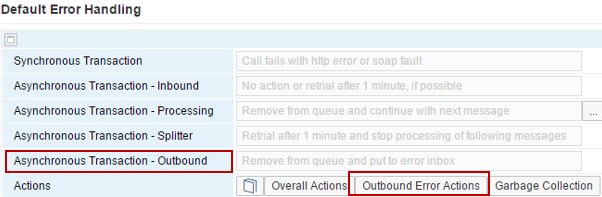
**Scenario Error Handling**

To change the default behavior, you have the following options:

* Let the integration framework trigger an individual scenario step.
* To trigger an individual error handling scenario step, select *Scenarios* → *Package Design* → *[Definitions]* → *Error Handling*.
* For the *Async. Outbound: Step* field, select a scenario step of the scenario package that has the internal queue inbound type and that performs the individual error handling.



* In default error handling, the integration framework deletes the message from the inbound queue.



To overlay the default behavior, create an error action for the scenario package, the scenario step or the outbound phase, optionally in combination with sender and receiver definitions.

To define the error action, select Maintenance → Cfg Error Handling → [Outbound Error Actions], and select Keep in Inbound Queue.

* By default, the integration framework hands over the message to the error inbox.

The integration framework provides an error inbox for each receiver system. In the error inbox, it collects messages that the receiver system has rejected.

* To change the default behavior, define an error action for a scenario package, a scenario step or the outbound phase, optionally in combination with sender and receiver system definitions.

To define the error action, select Maintenance → Cfg Error Handling → [Outbound Error Actions], and select Don’t Send to Error Inbox.

* Alternatively, for your scenario package, deselect the *Asynchronous Outbound: Error Inbox*.
* The integration framework runs a successor scenario step.

Each scenario step can have a successor scenario step that is triggered by an internal queue. The integration framework triggers the successor step after it has finished the outbound phase. If an error occurs in the outbound phase, it still triggers the successor step.

To define a successor step, select *Scenarios* → *Step Design* → *[Outbound)*.

# 6 Using Job Lists

If you develop a scenario package that is always relevant for all company databases of SAP Business One, you can use the function to subscribe your scenario package to changes in the SAP Business One landscape. Subscribe to the following jobs:

* New company in SAP Business One

If you create a job for this event, the integration framework adds the company that has been created in SAP Business One and in SLD to all scenario steps with B1 inbound and B1 outbound channels.

* Add new company to scenario package configuration
* Delete company in SAP Business One

If you create a job for this event, the integration framework removes the company that has been deleted in SAP Business One and in SLD from all scenario steps with B1 inbound and B1 outbound channels.

example.gif EXAMPLE

The sap.Xcelsius and sap.DATEV-HR scenario packages use this function. This way, the integration with dashboards and Datev-HR is always available for all SAP Business One company databases.

To create jobs, select *Scenarios* → *Package Design* → *[Definitions]* and select *Job List*.

The integration framework provides the following options:

* New company in SAP Business One
* Delete company in SAP Business One

Select an option, and the integration framework opens the user interface for job list definition, either for adding a new SAP Business One company, or for removing a deleted SAP Business One company:

**New Company in SAP Business One**

The integration framework opens the following user interface:





To add a job, click the button.

In the input field, enter a number for the position of the job, and click [Select].

The integration framework adds Job 1 to the list:



To select the task, click the […] button, and select the Add new company to Scenario Package configuration option. The other option is obsolete.



To delete a job, click the button.

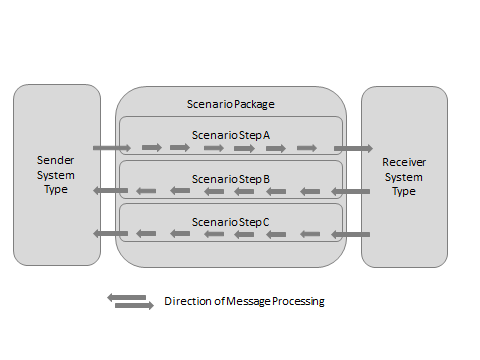
**Delete Company in SAP Business One**

Proceed as described above to remove a deleted SAP Business One company from the scenario package configuration.

# 7 Defining Processes

If a scenario package contains scenario steps that run in both directions between sender and receiver systems, you can use processes to assign scenario steps to a direction.

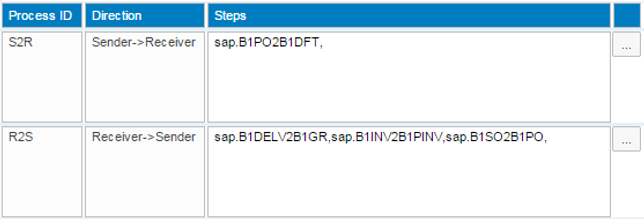
example.gif EXAMPLE



In the illustration above, the scenario package has three scenario steps. An incoming message of the sender system type triggers message processing in scenario step A. The message that processing hands over to the receiver system type leads to subsequent steps in the receiver system type that trigger first scenario step B and then scenario step C.

If you have, for example, defined criteria fields for the scenario package, the setting of directions simplifies the entry of filter definitions in the scenario setup wizard. The wizard only displays filter definitions for the valid combinations of sender and receiver systems.

To define processes, select *Scenarios* → *Package Design* → *[Definitions]* and select *Processes*.



To select scenario steps for the *Sender -> Receiver* direction, click the […] browse button. Select one or more scenario steps.

To select scenario steps for the *Receiver* -> *Sender* direction, click the […] browse button. Select one or more scenario steps.

[Clear]

To remove all entries, click the button.

**Result**

If you define processes, the scenario setup wizard provides the option to apply the processes to simplify the input of filter definitions.

# 8 Designing and Using an Individual Message Log

The message log in the Monitoring section of the integration framework displays information about messages that the integration framework has processed. For a scenario package, you can develop an individual message log scenario step.

## 8.1 Designing an Individual Message Log Step

For an individual message log step, use the internal queue inbound type.

In scenario step processing, define what you need for your scenario package monitoring. You can, for example, write the message log information to a database table that you access with an individual user interface, or you send e-mails to an administrator depending on message log status information, and so on.

The message that the integration framework sends to the queue of your message log step, contains all information that the default message receives.

* In the <Payload Role="**S**"> section, in the <B1ifMsgLog> tag, find information that the integration framework uses to display the default message log:
* SenderSysId
* SenderSysIdName
* SenderObjectTypeId
* ReceiverSysId
* ReceiverSysIdName
* BeginTimeStamp
* EndTimeStamp
* Status
* In the <ResultMessage> tag, the integration framework displays the result of message processing.
* If the integration framework runs a synchronous scenario step, it creates one message log entry.
* If the integration framework runs an asynchronous scenario step, it calls the message log twice. The default message log only displays the message log for which Status **is not** processing. Take this also into consideration when you create your individual message log information.

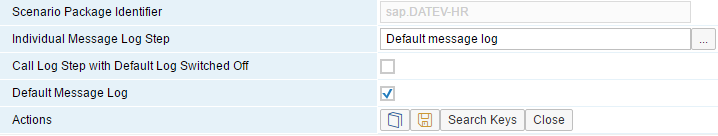
Use an XSL transformation (xform) atom, to retrieve the information you want to use in your individual message log.

Retrieve the information in the following way, for example for the sender system name:

<xsl:value-of select="/vpf:Msg/vpf:Body/vpf:Payload[@Role='S']/B1ifMsgLog/@SenderSysIdName"></xsl:value-of>

## 8.2 Defining the Individual Message Log Step for the Scenario Package

To select a message log step and define settings for the individual message log, select *Scenarios*⏵*Package Design* ⏵*[Definitions]* and select *Message Log*.



Scenario Package Identifier

The integration framework displays the identifier of the scenario package.

Individual Message Log Step

By default, each scenario package uses the default message log (*Monitoring* → *Message Log*). To select an individual message log step, click the […] browse button. The integration framework displays all scenario steps of the package that have the internal queue inbound type. Select the message log step.

Call Log Step with Default Log Switched Off

To enable the individual log step to receive message log information, although the default message log is switched off (*Maintenance* → *Cfg MsgLog*, *Message Log* not selected) select the option.

Default Message Log

To additionally display message log information for the individual message log step in the default message log, select the checkbox. This is the default.

[Search Key]

To define individual search keys that you can use in the message log selection, click the button.

For each scenario step of the scenario package, you can define two search keys. The search keys are XPath expressions. In the XPath, you can use the following variables:

* $S for the sender message section
* $R for the receiver message section
* $T for the trigger message section

You can only use XPath statements using the variables, XPath statements without using the variables are not supported.

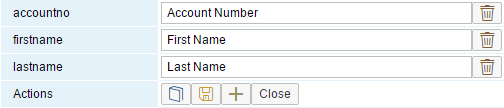
# 9 Defining Value Mappings

The integration framework supports value mapping. It allows externalizing mappings between values in sender and receiver systems instead of hardcoding them in the development phase.

At runtime, value mapping takes place after scenario step processing and before handover to the receiver system or systems. If the integration framework finds tags in the **receiver** XML message with one of the defined names, and value mapping definitions exist for the fields, the integration framework maps the values according to the definitions.

If you use VOID outbound, use the Value Mapping atom to perform the value mapping in the process flow.

To define fields in the outbound message for which the integration framework performs a value mapping, select *Scenarios* → *Package Design* → *[Definitions]* →*Value Mappings*.



Adding a Value Mapping Field]

* To define a field as a value mapping field, click  [Add].
* In the window, enter the name of the field and click *Select*.

The integration framework displays the new field in the *Value Mapping Fields* user interface.

* Enter the description for the mapping field in the input field and click the *Save* button.

[Delete]

To delete a value mapping field, click the button. The integration framework deletes the field from the list.

**Result**

During scenario setup, the administrator can define value mappings for the fields for sender and receiver systems.

# 10 Selecting an XML Processor

The SAP XML Toolkit XML processor and the Java Runtime Environment (JRE) XML processor are part of the integration framework.

note.gif NOTE

As of integration framework version 1.22.9, the Java Runtime Environment (JRE) XML processor is the default processor. If you create scenario packages, the option to select an XML processor is no longer available.

**For scenario packages created prior to integration framework version 1.22.9:**

Select an XML processor for your scenario package processing. You have the following options:

* jrexml

The integration framework uses the JRE XML processor. As of integration framework version 1.22.9, this is the default.

* sapxml or empty

The integration framework uses the SAP XML Toolkit.

**Procedure**

To change the processor, perform the following steps:

1. Select Scenarios → Setup, select and deactivate the scenario package.
2. Select Scenarios → Package Design, select the scenario package and click the *Definitions* button, select *XML Processor*, and select the jrexml option.
3. Activate and test the scenario package.

The integration framework displays the XML processor it uses in the XSL transformation atom in the *XML Processor* field.

# 11 Versioning of Scenario Packages and Steps

The integration framework provides versioning of scenario packages and scenario steps. This gives you the opportunity to enhance or correct, for example, scenario packages and steps and provide a new version, while an older version is running in the customer landscape.

**Scenario Package**

* On scenario package level, the integration framework provides a function to save the design definitions of the current version to archive. The integration framework creates the new com.sap.b1i.vplatform.scenarios.archive dataset in the BizStore and a subfolder with the name of the scenario package identifier. The integration framework adds all scenario package design content to a ZIP file. The ZIP file has the name of the scenario package plus the version of the scenario package.
* You can set the scenario package to a higher version and continue with development.
* If you need to go back to a previous scenario package version, use the Restore from archive function. To retrieve an older version from archive, archive and delete the current version in the integration framework.
* The integration framework provides a function to delete versions from archive.

To work with scenario package versions, select Scenarios → Package Design, click the *Tools* button and select one of the following options:

* Save current version to archive
* Restore version from archive
* Delete version from archive.

**Scenario Step**

The integration framework supports a powerful version control for scenario steps. It allows keeping multiple versions of the same scenario step in the framework environment. You always have an active version relevant for setup and activation. The *Version* parameter displays the version number. All other versions of the same scenario step are in the archive folder.

* *Save current version to archive* saves the current version to the archive folder. The integration framework collects all documents of the scenario step and archives them in a <step\_name>.<version>.zip ZIP file. The integration framework system overwrites an existing ZIP file with an identical name.
* *Create new version* allows you to create a new version for a scenario step. The version number consists of three parts, separated by dot. The first part reflects the main version, the second part the sub version and the last part reflects minor extensions in the step. The integration framework calculates the version number. Before creating a new version, consider saving the current version to the archive. You have the following options:
* Create a new minor version, for example, 1.0.1
* Create a new subversion, for example, 1.1.0
* Create a new main version, for example, 2.0.0
* *Restore archived version* restores a version from the archive. If you need to go back to a previous version, the integration framework stores the current version to archive and then let you select the version from archive.

note.gif NOTE

Note that each scenario step has a link to a scenario package. The link is part of the restored version. If the link of an older, restored step is no longer valid, the integration framework indicates it with a red light during the consistency check.

* *Delete version from archive* provides you with the function to clean up the archive

To work with scenario step versions, select Scenarios → Step Design, click the *Tools* button, select *Version Control* and select one of the following options:

* Save current version to archive
* Create new version with the additional option to create a new minor, sub, or main version
* Restore version from archive
* Delete version from archive.

# 12 Providing Documentation for Scenario Packages

If you develop scenario packages and deliver them to customers, provide documentation for the package that contains the following:

* A solution overview that describes what the scenario package does
* A short description for each scenario step of the scenario package
* Configuration steps required in the sender system
* Configuration steps required in the receiver system or systems
* A detailed description of the scenario setup that is required in the integration framework.

**Documentation Delivery**

To deliver the documentation with the scenario package, create a PDF document called vPac.pdf and load it to the following place in the BizStore:

/com.sap.b1i.vplatform.scenarios.design/vPac.*<name>*/vPac.pdf

<name> is the name of your scenario step.

To upload the documentation, select *Tools* → *Control Center* → *Maintenance* → *BizStore Upload*

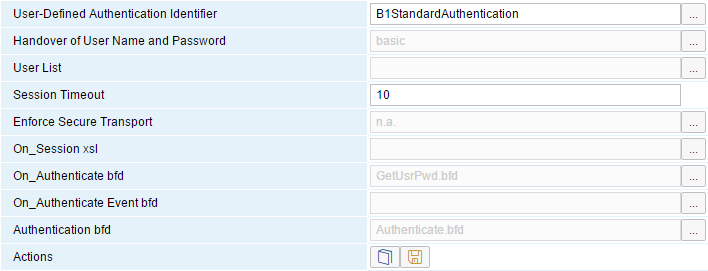
For more information, see the guide *Development Environment*, *Document Handling*, *Uploading a Document*

# 13 Authentication for Scenario Packages

## 13.1 Authentication User Interface

To define or display an authentication procedure for incoming HTTP or Web services calls of a scenario package, select Scenarios → Authentication.

The integration framework displays the following user interface:



User-Defined Authentication Identifier

To define an authentication procedure for a scenario package, enter an authentication identifier. The integration framework creates a folder with the authentication identifier name in the BizStore in the following place: com.sap.b1i.vplatform.scenarios.authen

Create all documents required for your authentication procedure in this folder.

To display an authentication, select the identifier.

Handover of User Name and Password

Select how the incoming call hands over the user name and password to the integration framework.

You have the following options:

* basic

The user name and password information is provided as basic authentication in the HTTP header of the incoming call.

* userdef

If the incoming call hands over the user name and password in the payload of the incoming call, select this option.

User List

Select the user list for authentication from the BizStore. If you leave the field empty, the integration framework uses the global user list for IPO step authentication of B1iP by default. Find the global list in the following place in the BizStore: com.sap.b1i.system.xc/xml/users\_ipo.xml. The document for the user list must conform to the com.sap.b1i.system.xc/xsd/users.xsd schema.

Session Timeout

Define the timeout of a pending HTTP session in minutes. The integration framework does not establish a session for an empty or zero value.

Enforce Secure Transport

Select whether you want to secure the technical connection to the integration framework. You have the following options:

* true

To enforce a secure connection for incoming call, select true. If the caller does not use HTTPS, the integration framework rejects incoming calls.

* false

If the incoming call can use HTTP, select false.

On\_Session.xsl

Select an XSL document that describes where the session identifier is provided. This is only relevant, if you do not use cookies.

On\_Authenticate.bfd

If you have selected the user-defined (userdef) option for handing over the user name and password, select a BizFlow that retrieves the authentication credentials. The execution profile is SAPXML.

If you leave the field empty, the integration framework calls the stylesheet for any authentication mode. The default retrieval mechanism searches for the first user and password tag in the input message. The stylesheet gets the input message conforming to the com.sap.b1i.system.xc/xsd/on\_authenticate.xsd schema. The stylesheet returns the message with credentials.

On\_Authenticate Event.bfd

Select a BizFlow for authentication events, such as logon, logoff, timeout, and so on. The BizFlow gets an input message conforming to the com.sap.b1i.system.xc/xsd/on\_authevent.xsd schema. The integration framework performs the BizFlow asynchronously to the incoming action.

If you do not provide a BizFlow, refer to the authentication events provided in the user administration.

Authentication.bfd

Select a BizFlow for the authentication validation. The integration framework calls the BizFlow for the basic and the userdef authentication modes. If you leave the field empty, the integration framework validates against the user list relevant for the IPO step. The BizFlow input message must conform to the input message tag of the com.sap.b1i.system.xc/xsd/on\_authcheck.xsd schema. If the integration framework rejects a user, it returns a negative or empty result. It returns 0 for a positive result. Cache the result for later use. The integration framework checks again after N minutes to confirm the check.

## 13.2 Using the SAP Business One Standard Authentication

When selecting the authentication for a scenario package, the integration framework provides the B1 Authentication and the B1 Secure Authentication options.

* The B1 Authentication uses the integration framework port for HTTP.
* The B1 Secure authentication uses the integration framework port for HTTPS.

Use the SAP Business One standard authentications in a scenario package to verify credentials of an incoming HTTP or Web service call against users of an SAP Business One company database. The standard authentications check whether the user name and password are available in the SAP Business One company database and performs the call, if the check was successfully performed.

**Prerequisites**

* In SLD, a system of H.AnySystem type or W.AnySystem type must be available for the SAP Business One company database that you use for the authentication check. Leave all entries of the H.AnySystem type or W.AnySystem type empty, except for the associatedSrvIP field. Enter here the information that is available in the b1server field of the SAP Business One company database SLD entry.
* In the incoming call, provide the user information in the following way:
* User: language/username/company or language\user\company

For example: en-US/manager/SBODemoDE

* Password: password
* language

Provide the language abbreviation. The parameter is mandatory. The integration framework removes leading and trailing spaces. If the parameter value does not fit the enumeration, the integration framework uses the en-US default value.

cs-CZ, da-DK, de-DE, el-GR, en-CY, en-GB, en-US, en-SG, es-AR, es-CO, es-ES, es-PA, fi-FI, fr-FR,he-IL, hu-HU, it-IT, ja-JP, ko-KR, nl-NL, no-NO, pl-PL, pt-BR, pt-PT, ru-RU,sk-SK, sr-YU, sv-SE, xx-XX, zh-CN, zh-TW.

* username

This is the user code of an SAP Business One user with the B1i license attached to access SAP Business One remotely through the integration framework. The integration framework removes leading and trailing spaces.

* company

This is the name of an SAP Business One company database. The company database must have an entry in the integration framework SLD. The name you provide must be identical with the value of the company field of the SLD entry. The integration framework uses the company database to verify the authentication for the user who requests access. The parameter is case-insensitive. The integration framework removes leading and trailing spaces.

* password

Provide the password for the username.

The integration framework performs the following checks for the incoming call using the technical B1i user:

* With an SQL call to the company database, the integration framework checks whether the user defined in the incoming call is available.
* Using the AuthenticateUser method, the integration framework performs a DI call and checks the user name and password in the company database.

If the authentication is successful, the integration framework returns an HTTP 200 response and the response matching the request.

If the authentication is not successful, the integration framework returns an HTTP error code, for example, 401 and an error message:

* b1i user wrong username or password
* wrong username or password
* not able to assign the associated B1 server
* name of B1 company db is empty
* user is empty
* password is empty
* not able to find the assigned user id in SAP Business One
* not able to find the system identifier of the associated B1 company db
* error in B1 authentication check (DI call)
* unknown error in authentication

## 13.3 Defining a Package-Specific Authentication with an Individual User List

For your scenario package that uses synchronous scenario steps, either HTTP or Web service calls, you want to define a specific authentication to access the integration framework. To enable authentication for specific users, create a user list in the BizStore, create an authentication for the scenario package that points to the user list and link the defined authentication to the scenario package.

**Procedure**

1. In the integration framework, select Scenarios → Authentication and enter the following information:

* In the User-Defined Authentication Identifier field, enter the identifier of your scenario package.
* In the Handover of User Name and Password field, select basic.
* In the User List field, enter users.xml.
* Save your settings.

The integration framework creates a folder for your scenario package in the com.sap.b1i.vplatform.scenarios.authen dataset in the BizStore.

1. Open the BizStore and find the schema definition for users.xml in the following place:

com.sap.b1i.system.xc/xsd/users.xsd

1. In the BizStore, create the users.xml document in the com.sap.b1i.vplatform.scenarios.authen.vPac.<your namespace>.<your package name> folder.

You can find an example users.xml file as part of the sap.B1if.Samples scenario package.

1. To let an administrator of the scenario package set the password for the user, create an administration user interface for the scenario package.

For more information, see section 15.2 Requesting a Password for a Scenario-Specific User List

1. In the integration framework, select Scenarios → Package Design, and in the Scenario Package Identifier field, select your package.
2. In the Authentication field, select the authentication for your package and save your settings.

# 14 Adding Datasets to Scenario Import and Export

If you have created additional datasets in the BizStore that belong to a scenario package and perform tasks that are relevant for the scenario package, select the datasets. The integration framework includes the datasets into scenario import and export.

The integration framework displays datasets for selection that belong to the namespace you have defined in the *Configuration Development Environment* user interface.

# 15 Creating an Administration User Interface for a Scenario Package

## 15.1 Requesting a Password for a Scenario-Specific User List

You want to use an individual user list for a scenario-specific authentication. The sap.B1if.Samples scenario package contains an example how to request a password for a user through an administration user interface. To display the authentication, choose Scenarios Authentication and in the User-Defined Authentication Identifier field, select sap.B1if.Samples.

To use the authentication in your scenario package, do the following:

* In the BizStore, go to the vPac folder of the sap.B1if.Samples package, copy the admin.xml document and paste it to the vPac of your package. The document contains the user interface definitions.
* In the BizStore, in the folder of you package in com.sap.b1i.vplatform.scenarios.authen, copy the following sap.B1if.Samples documents and paste them to the folder:
* getpwd.xsl

The XSL document contains instruction how to get the password.

* setpwd.bfd

The BizFlow definition document sets the password.

* setpwd.xsl

The XSL document contains instructions how to set the password in the user interface.

* Open the setpwd.bfd document, search for sap.B1if.Samples and replace it with the name of your scenario package.

# Appendix A. Table of Inbound Channels

| **Channel (IPO)** | **System Type** | **Inbound Mode** | | |
| --- | --- | --- | --- | --- |
| **Trigger** | **Mode** | **Identification** |
| **INB\_WS\_CALL\_SYNC\_XPT**  Inbound for incoming WSAR call, object identification by XPath or root tag | ws | WSAR Call | SYNC | xPath |
| ws | WSAR Call | SYNC | Root Tag |
| **INB\_HT\_CALL\_SYNC\_XPT**  Inbound for incoming HTTP call, object identification by XPath, URL parameter or root tag | http | HTTP Call | SYNC | xPath |
| http | HTTP Call | SYNC | URL Parameter |
| http | HTTP Call | SYNC | Root Tag |
| **INB\_B1\_EVNT\_ASYN\_EVT**  SAP Business One inbound triggered by B1 event | B1 | B1 Event | ASYNC | B1 Event |
| **INB\_FI\_EXST\_ASYN\_XPT**  Inbound for data from file system, triggered by existance of file in inbound directory, identification by XPath statement, root tag or B1 logic | file | File exists | ASYNC | xPath |
| file | File exists | ASYNC | Root Tag |
| file | File exists | ASYNC | B1 Logic |
| **INB\_FI\_EXST\_ASYN\_NAM**  Inbound for data from file system, triggered by existance of file in inbound directory, identification by file name | file | File exists | ASYNC | File Name |
| **INB\_FI\_EXST\_ASYN\_FIX**  Inbound for data from file system, triggered by existance of file in inbound directory, fixed identification | file | File exists | ASYNC | Fix Value |
| **INB\_FI\_EXST\_ASYN\_FSL**  Inbound for data from file system, triggered by existance of file in inbound directory, identification by first line | file | File exists | ASYNC | First Line |
| **INB\_R3\_IDOC\_ASYN\_XPT**  Inbound for data from SAP ERP, triggered by incoming IDoc, identification by XPath statement | R/3 | IDOC | ASYNC | xPath |
| R/3 | IDOC | ASYNC | Root Tag |
| **INB\_IQ\_INTQ\_ASYN\_BIU**  Inbound for successor scenario step, triggered by internal queue | vBIU | Queue | ASYNC | vBIU Name |
| **INB\_IQ\_INTQ\_ASYN\_QS**  Inbound for vBIU, triggered by internal queue | Queue | Queue | ASYNC | Fix Value |
| **INB\_DB\_TIMR\_ASYN\_FIX**  Inbound for database retrieval, triggered by timer | DB | Timer | ASYNC | Fix Value |
| **INB\_AY\_TIMR\_ASYN\_BIU**  Timer-triggered inbound | VOID | Timer | ASYNC | VOID |
| **INB\_DB\_TIMR\_DATARETRIEVER**  Inbound for SQL data retrieval for a scenario step, triggered via internal queue | DB | Timer | ASYNC | Fix Value |

# Appendix B. Table of Outbound Channels

| **IPO** | **System** | **Outbound Mode** | | | **Details** | | **Formatting** | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Adapter** | **Format** | | **Method** | **Ctrl.Doc** | **Frmt** | **Doc** | **Enc.** |
| **OUT\_DB** | DB | JDBC | SQL |  | single |  | xml | - | - |
| DB | JDBC | SQL |  | multiple |  | xml | - | - |
| DB | JDBC | B1i |  | single |  | xml | - | - |
| DB | JDBC | B1i |  | multiple |  | xml | - | - |
| **OUT\_FILE** | file | FILO | XML |  |  |  | xml | - | - |
| file | FILO | XML |  |  |  | dsv | yes | yes |
| file | FILO | XML |  |  | yes | txt | - | yes |
| **OUT\_B1** | B1 | DIAPI | Object | DI reduced | pars |  | xml | - | - |
| B1 | DIAPI | Object | DI full |  |  | xml | - | - |
| B1 | DIAPI | Service | DI reduced | pars | yes | xml | - | - |
| B1 | DIAPI | Service | DI full |  |  | xml | - | - |
| B1 | JDBC | JDBC | sql reduced | single | yes | xml | - | - |
| B1 | JDBC | JDBC | sql reduced | multiple |  | xml | - | - |
| B1 | JDBC | JDBC | sql full | single |  | xml | - | - |
| B1 | JDBC | JDBC | sql full | multiple |  | xml | - | - |
| B1 | JDBC | JDBC | B1isql | single |  | xml | - | - |
| B1 | JDBC | JDBC | B1isql | multiple |  | xml | - | - |
| **OUT\_R3** | SAP ERP | R3 | IDoc |  |  |  |  |  |  |
| **OUT\_WS** | Web Service | WS | WS |  |  |  |  |  |  |
| **OUT\_HT** | HTTP | HTTP |  |  |  |  |  |  |  |
| **OUT\_BP** | BP |  |  |  |  |  |  |  |  |

# Appendix C. Table of Process Steps

| **vBIU** | **System Type** | **Process Step ID** | **Phase** | **Task** | **Processing** |
| --- | --- | --- | --- | --- | --- |
|  | SAP Business One | INB\_B1\_EVNT\_ASYN\_EVT | Inbound | Event triggered | asynchronous |
|  | PRC\_B1 | Processing | Data Retrieval |  |
|  | PRQ\_B1 | Processing | Data Distributor |  |
|  | OUT\_B1 | Outbound | Outbound |  |
|  | SAP ERP | INB\_R3\_IDOC\_ASYN\_XPT | Inbound | IDOC triggered | asynchronous |
|  | PRC\_R3 | Processing | Data Retrieval |  |
|  | PRQ\_R3 | Processing | Data Distributor |  |
|  | OUT\_R3 | Outbound | Outbound |  |
|  | Database | INB\_DB\_TIMR\_DATA RETRIEVER | Inbound | Timer-trigggered database inbound | asynchronous |
|  | INB\_DB\_TIMR\_ASYN\_FIX | Inbound | Timer triggered | asynchronous |
|  | OUT\_DB | Outbound | Outbound |  |
|  | File System | INB\_FI\_EXST\_ASYN\_XPT | Inbound | file exist triggered | asynchronous |
|  | PRC\_FIx | Processing | Data Retrieval |  |
|  | PRQ\_FIx | Processing | Data Distributor |  |
|  | INB\_FI\_EXST\_ASYN\_NAM | Inbound | file exist triggered | asynchronous |
|  | PRC\_FIn | Processing | Data Retrieval |  |
|  | PRQ\_FIn | Processing | Data Distributor |  |
|  | INB\_FI\_EXST\_ASYN\_FS | Inbound | file exist triggered | asynchronous |
|  | PRC\_FIf | Processing | Data Retrieval |  |
|  | PRQ\_FIf | Processing | Data Distributor |  |
|  | INB\_FI\_EXST\_ASYN\_FIX | Inbound | file exist triggered | asynchronous |
|  | PRC\_FIi | Processing | Data Retrieval |  |
|  | PRQ\_FIi | Processing | Data Distributor |  |
|  | INB\_FI\_TIMR\_ASYN\_XPT | Inbound | timer triggered | asynchronous |
|  | PRC\_FTx | Processing | Data Retrieval |  |
|  | PRQ\_FTx | Processing | Data Distributor |  |
|  | INB\_FI\_TIMR\_ASYN\_FIX | Inbound | timer triggered | asynchronous |
|  | PRC\_FTf | Processing | Data Retrieval |  |
|  | PRQ\_FTf | Processing | Data Distributor |  |
|  | OUT\_FILE | Outbound | Outbound |  |
|  | HTTP | INB\_HT\_CALL\_SYNC\_XPT | Inbound | Incoming HTTP call | synchronous |
|  | Web Service | INB\_WS\_CALL\_SYNC\_XPT | Inbound | Incoming HTTP call | synchronous |
|  | Predecessor | INB\_IQ\_INTQ\_ASYN\_BIU | Inbound | vBIU triggered | asynchronous |
|  | PRQ\_IQ | Processing | Data Processing |  |
|  | Internal Queue | INB\_WS\_CALL\_ASYN\_XPT | Inbound | Internal Queue triggered | asynchronous |
|  | PRC\_QS | Processing | Data Retrieval |  |
|  | PRQ\_QS | Processing | Data Distributor |  |
|  | Void | VOID | Outbound | Outbound |  |

# Copyrights, Trademarks, and Disclaimers

© Copyright 2020 SAP SE. All rights reserved.

The current version of the copyrights, trademarks, and disclaimers at <https://www.sap.com/about/legal/copyright.html> is valid for this document.