

Implementation Guide for the Connection of External Data Sources to SAP Business Suite Applications



Plant Connectivity 15.0



Copyright

© Copyright 2014 SAP AG. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP AG. The information contained herein may be changed without prior notice. Some software products marketed by SAP AG and its distributors contain proprietary software components of other software vendors.

Microsoft, Windows, Excel, Outlook, and PowerPoint are registered trademarks of Microsoft Corporation. IBM, DB2, DB2 Universal Database, System i, System i5, System p, System p5, System x, System z, System z10, System z9, z10, z9, iSeries, pSeries, xSeries, zSeries, eServer, z/VM, z/OS, i5/OS, S/390, OS/390, OS/400, AS/400, S/390 Parallel Enterprise Server, PowerVM, Power Architecture, POWER6+, POWER6, POWER5+, POWER5, POWER, OpenPower, PowerPC, BatchPipes, BladeCenter, System Storage, GPFS, HACMP, RETAIN, DB2 Connect, RACF, Redbooks, OS/2, Parallel Sysplex, MVS/ESA, AIX, Intelligent Miner, WebSphere, Netfinity, Tivoli and Informix are trademarks or registered trademarks of IBM Corporation.

Linux is the registered trademark of Linus Torvalds in the U.S. and other countries.

Adobe, the Adobe logo, Acrobat, PostScript, and Reader are either trademarks or registered trademarks of Adobe Systems Incorporated in the United States and/or other countries.

Oracle is a registered trademark of Oracle Corporation.

UNIX, X/Open, OSF/1, and Motif are registered trademarks of the Open Group.

Citrix, ICA, Program Neighborhood, MetaFrame, WinFrame, VideoFrame, and MultiWin are trademarks or registered trademarks of Citrix Systems, Inc.

HTML, XML, XHTML and W3C are trademarks or registered trademarks of W3C®, World Wide Web Consortium, Massachusetts Institute of Technology.

Java is a registered trademark of Sun Microsystems, Inc. JavaScript is a registered trademark of Sun Microsystems, Inc., used under license for technology invented and implemented by Netscape.

SAP, R/3, SAP NetWeaver, Duet, PartnerEdge, ByDesign, SAP BusinessObjects Explorer, and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and other countries.

Business Objects and the Business Objects logo, BusinessObjects, Crystal Reports, Crystal Decisions, Web Intelligence, Xcelsius, and other Business Objects products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of Business Objects Software Ltd. in the United States and in other countries.

Sybase and Adaptive Server, iAnywhere, Sybase 365, SQL Anywhere, and other Sybase products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of Sybase, Inc. Sybase is an SAP company.

Data contained in this document serves informational purposes only. National product specifications may vary. These materials are subject to change without notice. These materials are provided by SAP AG and its affiliated companies ("SAP Group") for informational purposes only, without representation or warranty of any kind, and SAP Group shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP Group products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

SAP disclaimer

Document classification for SAP Library: PUBLIC

Coding samples

Any software coding and/or code lines/strings ("Code") included in this documentation are only examples and are not intended to be used in a productive system environment. The Code is only intended better explain and visualize the syntax and phrasing rules of certain coding. SAP does not warrant the correctness and completeness of the Code given herein, and SAP shall not be liable for errors or damages caused by the usage of the Code, except if such damages were caused by SAP intentionally or due to gross negligence.

Internet hyperlinks

The SAP documentation may contain hyperlinks to the Internet. These hyperlinks are intended to serve as a hint where to find supplementary documentation. SAP does not warrant the availability and correctness of such supplementary documentation or the ability to serve for a particular purpose. SAP shall not be liable for any damages caused by the use of such documentation unless such damages have been caused by SAP's gross negligence or willful misconduct.

Accessibility

The information contained in the SAP Library documentation represents SAP's current view of accessibility criteria as of the date of publication; it is in no way intended to be a binding guideline on how to ensure accessibility of software products. SAP specifically disclaims any liability with respect to this document and no contractual obligations or commitments are formed either directly or indirectly by this document.

Symbols







Symbol	Meaning
	Caution
	Example
	Recommendation
	Note
	Syntax
	Tip

Table of Contents

Copyright.....	2
SAP disclaimer.....	3
Coding samples	3
Internet hyperlinks.....	3
Accessibility.....	3
Symbols.....	3
Version overview	6
Overview	6
System prerequisites.....	6
Installation of PCo 2.3	7
Implementation options	7
1) Data exchange via PCo queries.....	7
Process steps for the data exchange using queries	7
2) Generating and processing notifications	8
Process steps for the generation and processing of notifications.....	9
Generating a PCo agent instance for an external data source	11
Application handle	11
Test program RPCO_BS_INT_TEST.....	12
Function keys on initial screen.....	13
Selection options on initial screen.....	13
Parameters for the communication between SAP Business Suite and PCo agent	13
Parameters for tag processing	14
Parameters for the ALV result display	15
Result list.....	15
Displayed tag information.....	16
Function keys in result list.....	16
Input help for the browsing of namespaces	17
Dealing with errors	18
ABAP Sample Implementations	19
Example for the data exchange using queries	19
Default values for order confirmation	19
Optimization possibilities.....	23
Examples for the generation and processing of notifications	23

1) Conversion of notifications to application logs	24
Implementation of class CL_PCO_IM_SAPTESTING_NOTIF for BAdI implementation	27
2) Automatic processing of time ticket confirmations	28
Implementation of class ZUD_RK_BADI_KONF for BAdI implementation.....	30
Appendix: Tips and Tricks	32
1) Conversion of timestamp formats.....	32
2) Linking of individual quotation marks with text string	32
3) Linking of space/blank with a text string.....	32
4) Conversion and formatting of floating point numbers.....	33
Transfer of floating point numbers from the Business Suite to PCo	33
Display of floating point numbers that are transferred from PCo to the Business Suite	33

Version overview

Version 1.0 (April 2014): First version of the implementation guide for PCo 15.0

Overview

SAP Plant Connectivity (PCo) enables you to easily set up a connection between applications of the SAP Business Suite with external data sources such as weighing systems, production machines, OPC servers or process control systems. The data exchange between the external data sources and PCo is carried out using PCo agents and agent instances.

- An **agent** is a NET-DLL Assembly component that can set up a connection between a data source and PCo.
- An **agent instance** is a user-defined configuration of an agent that enables the data flow.

Until recently, only the SAP applications *SAP Manufacturing Intelligence and Integration (SAP MII)*, *SAP Manufacturing Execution (SAP ME)* and *SAP Extended Warehouse Management (SAP EWM)* were able to use the options provided by SAP Plant Connectivity (PCo). For applications of the SAP Business Suite, the corresponding PCo functionality was not available until now.

With the introduction of an ABAP-based integration layer, which forms part of the *SAP NetWeaver* system, the range of applications for *PCo* has now also been extended to *SAP Business Suite* applications.

The data transfer between systems of the *Business Suite* and the PCo agents is based on RFC function calls (TCP/IP). Every PCo agent is first assigned the corresponding RFC destination in the *Business Suite* system. Business Suite applications that exchange data with PCo agents execute the corresponding methods of SAP NetWeaver.

System prerequisites

The following technical prerequisites are necessary in order to connect external data sources using *SAP Plant Connectivity*:

- *Plant Connectivity* starting with Version 2.3
 - Microsoft .Net-Framework as of version 4.0
- An SAP *Business Suite* system with SAP NetWeaver Application Server ABAP as follows:
 - NW ABAP Release 7.31
 - NW ABAP Release 7.02, starting with Support Package SAPKB70024
 - NW ABAP Release 7.01, starting with Support Package SAPKB70109
 - NW ABAP Release 7.00, starting with Support Package SAPKB70208
- TCP/IP Communication between PCo and a system of the SAP Business Suite

Depending on the type of external data sources used, OPC data servers must also be installed and configured.

Installation of PCo 15.0

Detailed information on the installation of PCo 15.0 can be found in the PCo Installation Guide on the Service Marketplace under <http://www.service.sap.com/instguides>.

Implementation options

PCo provides two basic ways for *Business Suite* Applications and external systems to exchange data:

- Via *PCo* queries that can access and change data of external sources through the *Business Suite* applications
- Via the sending of *PCo* notification messages, triggered by external data sources, to *Business Suite* applications for further processing

1) Data exchange via PCo queries

Using the *PCo* functions, *Business Suite* applications can address queries to external data sources, for example, in order to read or change tag values of a data source. The *PCo* agent instance that is connected to an external data source receives the query and interprets the instructions contained in it. The contents of the query are created using *the SAP MII* protocol language.

PCo itself has three types of queries:

- Queries relating to tags
- Queries relating to database contents
- Text-based queries with user-defined textual contents

Process steps for the data exchange using queries

The following steps are necessary to carry out and evaluate *PCo* queries:

1. *Business Suite*: Create an RFC connection (transaction *SM59*):
 - a. Maintain the Gateway options
 - b. Provide a name for the registered server program
 - c. Optional: Maintain the data for a secure network communication (*SNC*)
2. *Business Suite*: Maintenance of authorizations (transaction *PFCG*, authorization object **S_PCO_INT**) for the authorization profile of the user to be utilized for the RFC data exchange between the *Business Suite* and *PCo*. You can use the following *PFCG* profiles as templates:
 - **SAP_BC_SRV_PCO_BS_INT_ADMIN** and
 - **SAP_BC_SRV_PCO_BS_INT_USER**
3. *PCo*: Generating of a *PCo* agent instance for the external data source which is to communicate with the *Business Suite* application. First you must add a source and a destination system (see next steps).
4. Adding of the *PCo* source system
 - Selection of a source system type (OPC DA agent, OLE DB agent, etc.)
 - Maintenance of the RFC server connection data
5. Adding of the *PCo* destination systems

- Selection of the *PCo* destination system type *RFC Destination*
- Maintenance of the RFC Client data
 - Selection of *SAP NetWeaver* for the *PCo* Client type
 - Maintenance of the application server and additional connection data for the corresponding *SAP Business Suite* system
- 6. Creation of the *PCo* agent instance for the above-mentioned *PCo* source and destination systems
 - Maintenance of the service user name and password for the execution of the *PCo* agent instance as an *MS Windows* service
 - Maintenance of the *PCo Query Ports* for the source system type *SAP NW RFC Server*
 - Maintenance of the RFC server settings
 - Entering the name of the registered server program that was previously specified in the SAP RFC connection data
 - Assignment of the *PCo* destination system providing the information for the Repository structure of the *SAP DotNet Connector (SAP NCo)*
- 7. *Business Suite*: Implementation of method calls for the ABAP Wrapper classes of package *S_PCO*.
 - Assignment of a name for the [application handle](#) to be passed to *PCo*
 - Creation of the method calls in the appropriate coding segments (User Exit, BAdI, etc.) of the *Business Suite* applications
 - [Sample implementations](#) can be found at the end of this Implementation Guide
- 8. *PCo*: Starting of the *PCo* agent instance
- 9. *Business Suite*: Execution of the *Business Suite* application

2) Generating and processing notifications

For some business processes, it makes sense to trigger activities of the *SAP Business Suite* if certain values or parameters of external data sources have changed. The following are examples of this:

Business process	Activities in the SAP Business Suite
The status of a machine changes from “processing” to “malfunction”	<ul style="list-style-type: none"> • Automatic generation of a plant maintenance message • Status change in the equipment master data record
The value of a temperature sensor exceeds the maximum temperature previously specified of 120°C	<ul style="list-style-type: none"> • Alarm/alert notification (Email, SMS) to the shift manager
The weight of a semifinished product is calculated by means of individual weighing	<ul style="list-style-type: none"> • Automatic generation of a time ticket confirmation • Release of the subsequent order • Creation of a production inspection lot

In the above-mentioned examples, specific *Business Suite* applications must be automatically addressed. They receive the required tag data and information via notifications. These notifications are generated and sent by PCo as soon as the specified trigger conditions are fulfilled. The following applies here:

- The selection of the relevant tags is carried out using subscriptions on the part of the *Business Suite* applications.
- The processing of *PCo* notifications to be sent back to the *Business Suite* application can be controlled using BAdI implementations, for each specific case and in great detail.

Process steps for the generation and processing of notifications

The following steps are necessary to be able to generate and process *PCo* notifications:

Step	System	Activity
1	SAP Business Suite	Creation of an RFC connection (transaction <i>SM59</i>): <ul style="list-style-type: none"> • Maintenance of the Gateway options • Name assignment for a registered server program • Optional: Maintenance of the data for a secure network communication (SNC)
2	SAP Business Suite	Maintenance of authorizations (transaction <i>PFCG</i>) for authorization object S_PCO_INT in the authorization profile of the RFC user master record (see above). <ul style="list-style-type: none"> • Use of the new PFCG profiles SAP_BC_SRV_PCO_BS_INT_ADMIN and SAP_BC_SRV_PCO_BS_INT_USER as templates
3	SAP Plant Connectivity (PCo)	Generating of a <i>PCo</i> agent instance for the external data source that is to communicate with the <i>Business Suite</i> application.
4	SAP Plant Connectivity (PCo)	Creation of notification templates for a <i>PCo</i> agent instance. You can get further information under Generating a PCo agent instance for an external data source .
5	SAP Business Suite	<ul style="list-style-type: none"> • Creation of classes for the processing of notifications (transaction <i>SE24</i>). Classes inherit data from the higher-level class CL_PCO_NOTIF_HANDLER and overwrite the EXECUTE method with suitable implementations. • Implementation of the BAdI BADI_S_PCO_HANDLE_NOTIF (transaction <i>SE19</i> or <i>SE80</i> [package S_PCO, enhancement spot S_PCO_ES_BADI_HANDLE_NOTIF]): Depending on the BAdI filter value in the application of the corresponding application handle, the name of the class previously created is returned, which then is to process the notification.
6	SAP Business Suite	If the tag subscription is carried out by the Business Suite application itself, the following steps for the

		<p>implementation of method calls for the ABAP wrapper class of package S_PCO must also be carried out:</p> <ol style="list-style-type: none"> 1. Assignment of a name for the application handle that is to be passed to <i>PCo</i> 2. Creation of the method calls at the appropriate coding sections (User Exit, BAdI or similar) of the <i>Business Suite</i> applications. Sample implementations can be found at the end of this implementation guide.
7	SAP Plant Connectivity (PCo)	Start the <i>PCo</i> agent instance
8	SAP Business Suite	Subscription to tags for which notifications are to be sent if the trigger conditions are fulfilled
9	SAP Business Suite	<ol style="list-style-type: none"> 1. Specification of an application handle to which notifications are to be assigned. Via the application handle, it is subsequently possible to control the further processing of notifications when they are received by the <i>Business Suite</i> applications. <ul style="list-style-type: none"> • The sample program RPCO_BS_INT_TEST uses SAPTESTING as the default value for the application, as well as the log values of the application handle. • For the filter value of the application SAPTESTING, there is an active BAdI implementation that can be used to process and save the contents of the notification as application log messages. 2. Assignment of a notification template of the corresponding <i>PCo</i> agent instance
10	SAP Plant Connectivity (PCo)	<p>If the specified trigger conditions are fulfilled, <i>PCo</i> sends the notification – per RFC call as an XML document – to the corresponding recipients in the <i>Business Suite</i>.</p> <ul style="list-style-type: none"> • In this case, the function module S_PCO_RFC_EXECUTE_RECEIVE_EVNT is executed, in which the processing of the notification is to take place: • First the system checks whether a class for notification processing exists for the application specified in the corresponding application handle. • If this is the case, an instance of this class is generated and the EXECUTE instance method of the class is executed.

Generating a PCo agent instance for an external data source

In order to generate an agent instance for an external data source in the *PCo* system, which is to communicate with the *Business Suite* application, proceed as follows:

1. Add a *PCo* source system. For this:
 - Select the source system type (OPC DA Agent, OLE DB Agent, etc.)
 - Maintain the RFC server connection data.
2. Add a *PCo* destination system. For this:
 - Select the *PCo* destination system type *RFC Destination*.
 - Maintain the RFC client data.
 - Select the entry *SAP NW* for the *PCo* client type.
 - Maintain the application server and additional connection data for the corresponding *Business Suite* system.
3. Generation of the *PCo* agent instance for the above source and destination systems, as follows:
 - Maintain the service user name and password for the starting of the *PCo* agent instance as an *MS Windows* service.
 - Maintain the *PCo Query Ports* for the source system type *SAP NW RFC Server* as follows:
 - Maintain the RFC server settings.
 - Enter the name of the registered server program that was previously specified in the RFC connection data.
 - Assign the *PCo* destination system that provides the information for the Repository structure of the *SAP DotNet Connectors (SAP NCo)*.
4. Create notification templates for the *PCo* agent instance as follows:
 - Specify a trigger type (preferred trigger type: *Always*).
 - Specify the lifetime for the corresponding tag subscriptions. This refers to the duration for which a notification message retains its validity for the destination system. *PCo* retains the message for this period of time in the event of a transmission error.
 - Assign the *PCo* target systems to which the notifications are to be sent.

Application handle

To facilitate communication between the *Business Suite* applications and *PCo*, every *Business Suite* application creates a so-called “application handle”. This term refers to a system-wide unique application ID. The application handle is passed to *PCo* when requests are processed. It consists of two different parts:

- **APPLICATION:** User-defined name for the application that triggers the request and later processes the notification of a value change, for example:
 - **PRODORDCON** (production order confirmation)
 - **MAINTMSGCR** (generation of plant maintenance notifications)
 - **WEIGHINFO1** (information on the weighing process)
- **HANDLE:** User-defined content (max. of 80 characters), which can contain further information on the application context, for example:
 - Order confirmation data: plant, order type, confirmation number
 - Plant maintenance notification: plant, equipment ID
 - Weighing process: location, equipment ID, scale type

When you subscribe to tag value changes, the application handle is also passed. Please note the following:

- PCo groups the subscribed tabs using application handles.
- If messages are generated for these subscriptions, the messages also contain the application handle of the respective subscriptions.
- Information on the application handle can later be precisely evaluated when the notification is processed.
- An application handle containing the unique number of the production confirmation enables the reading of the respective order and operation data.

Test program **RPCO_BS_INT_TEST**

The new test program **RPCO_BS_INT_TEST** provides a basic understanding of the communication between *PCo* and the *Business Suite* applications. This program shows how easy it is to carry out the necessary implementations on the side of the *Business Suite* applications, so that external data sources can be connected to the *Business Suite* via *PCo*.

The focus of the test program is on the data exchange between the *Business Suite* application and external data sources can get read/write access to the *PCo* tags. The test program contains the following default values for the application handle:

- APPLICATION: SAPTESTING
- HANDLE: LOG

You can overwrite the default values of the test program with your own values if necessary.

If you want to carry out the sample implementation for the processing of notifications using the test program, you must pass the value SAPTESTING to the application handle for the application (APPLICATION).

The test program contains all functions that are provided in the SAP standard for the integration of external data sources using tag access options:

- Display of the supported agent features of *PCo* agent instances
- Browsing of namespaces for the tag hierarchies
- Reading of tags
- Writing of tag value changes
- Subscription to tags to be used for notifications
- Display of information on subscribed tags
- Extension of subscriptions
- Deletion of subscriptions
- Display of notification templates
- Display of application logs generated for the log object **S_PCO**.

Function keys on initial screen

On the initial screen of the test program, the following function keys are displayed:

Function key	Meaning
<i>Execute</i>	The test report is executed according to the selection conditions maintained on the initial screen.
<i>Get Variant</i>	Previously created report variants can be loaded. The selection conditions of the initial screen must previously have been saved as a report variant (<i>Save</i> pushbutton).
<i>Log Display</i>	Branches to the selection of application logs that were generated and saved for the integration of <i>Business Suite</i> applications.
<i>Features/Properties of PCoAgent</i>	Display of properties (reading, writing, subscription, support of native filters, etc.) of the <i>PCo</i> agent instance connected with the <i>Business Suite</i> system via a selected RFC connection.
<i>Properties of RFC Destination</i>	Enables the branching to the detail view of the RFC connection data maintenance for the selected RFC connection (transaction <i>SM59</i>).

Selection options on initial screen

Parameters for the communication between SAP Business Suite and PCo agent

For the communication between the Business Suite and the PCo agent, make entries for the following parameters:

Parameter	Meaning
RFC connection of the <i>PCo</i> agent instance	The F4 input help on a field lists all the RFC connections of the type TCP/IP that exist in the system.
Checkbox <i>RFC Connection Test</i>	If you select this checkbox, the system carries out an RFC connection test for the PCo agent that is to be accessed via the RFC connection when the object instance is generated. The results of the connection test are displayed in the status line.
Input field <i>Application</i>	The contents of the field are interpreted as the application name of the application handle and must have 10 characters.
Input field <i>Application handle</i>	The contents of the field are interpreted as the "handle content" of the application handle. <ul style="list-style-type: none"> The field content must contain at least one character. A maximum of 80 characters can be

	maintained as handle information.
Checkbox <i>Logging active</i>	If the checkmark is set, application logs are generated and saved automatically when functions of the test report are executed.
Checkbox <i>Automatic Load Features</i>	When the object instances are generated, the features of the corresponding <i>PCo</i> agent instances are loaded and buffered if you set this checkmark.
Checkbox <i>Test Mode (without PCo)</i>	<p>If you set this checkmark, the test report runs in test mode and uses hard-coded data for the results display.</p> <ul style="list-style-type: none"> • There is no communication with <i>PCo</i>. • The test mode serves primarily to simulate the interface.

Parameters for tag processing

For the processing tags, the following parameters are significant:

- **Field *Mode Namespace Browsing*.** This field enables one of three different modes:
 - **Mode *Preselection by Tag Hierarchy*:** Here the tag hierarchy of the connected external data source is displayed in a tree structure. If you select individual tags in the hierarchy, then the respective tag values, together with the fully qualified path and further details will be displayed when the test report is executed. If you select a group folder, the system will determine all tags (child nodes) having the group folder as a superior node. When you then execute the program, the system determines further detail information on the tags and displays them in the results list.
 - **Mode *Preselection by Specifying Qualified Path ID (F4 help)*:** When you select this mode, an additional input field for the fully qualified path, as well as a checkbox for this. Proceed as follows:
 1. After pressing the F4 pushbutton on this field, the tag hierarchy is displayed in a tree structure.
 2. You can now select an entry (tag or tag group). The system returns you to the initial screen, where you can see your selection.
 3. The system returns you to the initial screen, where you can see your selection with the fully qualified path of the tag or tag group.
 4. Now you can run the program. The system reads further tag data from the external data source and displays this in the results list. If you select a tag group, the display of the respective detail information is displayed in the results list.
 - **Mode *Preselection via Filter Value for Tags*.** When you select this mode, two further fields are displayed for entry:
 - **Input field *Masking*:** Via the F4 help you can select the type of filter to be used for tag searches. Depending on the *PCo* settings made for the corresponding agent instance, the search may be supported by the selected filter type. The following filter types are available:
 - Filter type **NATIVE**: The filter uses the filter options of the connected data source (this filter access usually has very high performance)
 - Filter type **REGEX**: The filter uses regular expressions

- Filter type LEGACY: The filter uses the filter logic of the former PCo function *UDS (Universal Data Sources)*, which is still in use by SAP MII for the connection of external data sources.
- Input field *Filter-String for Tag Alias*: Here you can search for the name or alias of a tag using wildcards (, *'). If tags are found, they are displayed in the results list when you run the test program.

For tag determination, there are the following functional restrictions:

- No tags are found for which the search string is only a part of the fully qualified path, for example:
 - Searching for *,*Tag_A'* returns the tag *,Root/Temperature/TempTag_A'*, but not the tag *,Root/TagGroup_A/Temp_A'*
- The fully qualified path information cannot be determined for these tags. For the tag subscription however, the fully qualified path of the tags is required. This means that subscriptions are not possible. In this test program, therefore, the function keys for the subscription, extension of subscriptions and deletion of subscriptions is hidden in the ALV results display.
- Input field *Maximal value (MaxRows)*
 - Using this input parameter, you can limit the number of tags and tag groups found. If a tag group is selected, and the maximal value is set at "6", the system reads the first six sub-groups as well as the first six tags, which are child nodes for the selected tag group. If more tags or tag groups exist than were selected, the system will output the corresponding message.
- Input field *Number of Decimals (Floating Point)*
 - For floating point figures, the decimal display is usually preferred. A value of "0" leads to the display of floating point numbers without limiting the number of decimals. Other values limit the number of decimals and lead to rounded floating point numbers.
 - Due to the internal display of the decimal places for a floating point number of the type "f" using dual fractions, there is no exact number corresponding to every number in the decimal system. This means that assignments and interim results of calculations can contain rounding errors, which can only be avoided by using a two-level rounding procedure.

Parameters for the ALV result display

Input field *ALV Display Variant*:

For the display of the results list, you can maintain application-specific ALV display variants. The input help for this field contains the variants maintained.

Result list

If tags were found for the selection conditions defined, the system subsequently reads additional tag data and displays the results in a table overview (ALV grid).

Displayed tag information

The following data is displayed in the results list:

Information	Meaning
Name or alias of the tag ("Tag alias")	The alias can be changed when subscribing. As long as a subscription exists, the alias name will always be displayed. Otherwise, the system will output the tag name.
Fully qualified path information ("Qualified Tag ID")	Display of the fully qualified path in the internal PCo format (with "/" as a separator): <ul style="list-style-type: none"> For the communication with PCo, fully qualified paths must always be passed to PCo from tags in the PCo-internal format.
Subscription (icon)	Indicates whether subscriptions exist for a tag or not
Shortened display of the tag value (max. 40 characters)	<ul style="list-style-type: none"> The field is ready for input Invalid tag values are indicated by the string "___".
Value change	In order to carry out changes in values for the specified tags, you next have to carry out the function to write the value change.
Short description	Short text on the SQL data type of the tag
Timestamp	Timestamp in UTC format
Date and Time fields	Day and time of day

Function keys in result list

The following function keys are available:

Function key	Meaning
<i>Display Agent Features</i>	Dialog box display of all functions available for the PCo agent
<i>Application Logs</i>	Display of application logs that are used for the application log object S_PCO
<i>Display Subscriptions</i>	Dialog box display of information on subscriptions that exist for the specified application handle: <ul style="list-style-type: none"> Name of the subscription Name or alias of the tab Fully qualified path of the tag in native display
<i>Notification Templates</i>	Dialog box with the name and description of notification templates maintained for the selected PCo agent instance
<i>Subscribe</i>	Selection of tags from the results list for which notifications are to be created if the tag values change. <ul style="list-style-type: none"> After pressing this pushbutton, the system displays the existing notification templates maintained for the PCo

	<p>agent.</p> <ul style="list-style-type: none"> • After selecting a notification request, the subscription is executed. • Tags that were successfully executed will display an icon in the "Subscr." column of the results list.
<i>Delete Subscriptions</i>	<p>Removal of subscriptions:</p> <ul style="list-style-type: none"> • If no entries are marked in the results list, the system deletes all subscriptions that exist for the specified application handle. • If individual entries of the results list are selected for which subscriptions exist, the system deletes these tags from the subscriptions.
<i>Extend Subscription</i>	<p>Extension of the validity period for all subscriptions of the application handle. The validity period is extended by changing the validity start to the value of the current date and time.</p>
<i>Write ValuesChange</i>	<p>Passing of changed tag values (in column <i>Value Chg.</i>) to the external data source that is linked to the Business Suite via a PCo agent instance</p>
<i>Update Display</i>	<p>Renewed loading of detail information for the selected tags</p>

Input help for the browsing of namespaces

The SAP Business Suite applications require an easy way to process tags and tag groups of an external data source. The elementary search help **S_PCO_ELM_BROWSE_TAG** (package **S_PCO**) serves as a sample template for the selection of a tag or a tag group. The namespace of the tags is displayed in a dialog box. After you select a tag or tag group, the search help returns the fully qualified path of the selected object.

The following search parameters are available:

Search help field	Meaning
MAX_ENTRIES	Entry of the maximum value for the selection of tags and tag groups. The entered value must be > 0.
RFC_DEST	Entry of the RFC connection for the selected <i>PCo</i> agent instance
SIMUL_MODE	Execution of the search help in simulation mode ("X"). Note that there is no communication with the <i>PCo</i> agent instance, but that instead, hard-coded data is accessed.
TAG_ID	Output field for the first 255 characters of the fully qualified path for the selected object, in the <i>PCo</i> -internal format (with "/" as separator)
TAG_DESCR	<p>Output field for the description of the tag:</p> <ul style="list-style-type: none"> • This only receives values in simulation mode. • In productive mode, the tag description can currently not be determined: <p>This <i>PCo</i> metadata is currently not returned to the calling <i>Business Suite</i> application when a <i>PCo</i> request is executed.</p>
NODE_TEXT	Output field for the name of the tag or tag group
IS_GROUP	Output field for the indicator specifying whether the selected object is a tag group or a tag

Due to the width of 464 characters, it is not possible to create a personal search input help. The output of the fully qualified path must also be limited to a maximum of 255 characters when the character string is output in search help screens.

If you want to avoid the above restriction, then you can also use the function module **S_PCO_CALL_POPUP_NAMESP_BROWS** (package **S_PCO**, function group **S_PCO**) instead of the search help.

Dealing with errors

If there are errors on the *PCo* side during the processing of *PCo* requests, the *PCo* agent instance passes the corresponding messages (*PCo* user messages) to the calling *Business Suite* application and raises a classed-based exception (class **CX_PCO_BS_INT**). The calling application intercepts this exception.

PCo messages are saved in the table attribute **PCO_MSG_OBJ** of the exception object and can be evaluated by the *Business Suite* application. The auxiliary class **CL_PCO_UTILITY** provides further methods to convert *PCo* messages in application log records and to save them in the application log.

Please note that when you are carrying out the F4 search help, the raised exceptions may NOT be output as error or termination messages (message type "Error" [E] or "Abend" [A]). This would lead to a program termination (short dump).

Instead, please use a more suitable message type ("Status" [S], "Information" [I] or "Warning" [W]) and set the display format *Error* or *Abend* via the ABAP command *DISPLAY LIKE*:

```
MESSAGE lv_err_txt TYPE cl_pco_utility=>gc_msgty_stat  
DISPLAY LIKE cl_pco_utility=>gc_msgty_error.
```

ABAP Sample Implementations

The following sections contain ABAP sample implementations showing how easily the exchange of information between external data sources and Business Suite applications can be implemented on the ABAP side.

For the communication with the external data source, it is necessary in addition to create and start the corresponding *PCo* agent instances.

Example for the data exchange using queries

Default values for order confirmation

Scenario

In this scenario, production progress of a discrete manufacturing facility is monitored via confirmations. For this, the user creates a time ticket confirmation in transaction CO11N. After pressing the pushbutton *Propose actual data*, the input fields are prefilled with the corresponding data of the operation. The machine used for production provides the actual data for yield, scrap and manual rework. This data is to be transferred when the actual data is automatically proposed for the corresponding input fields.

Implementation

A reading access to the data of the external data source is required. A suitable coding section for the implementation of the corresponding PCo query would be the Include **ZXCOFU06**, which belongs to the user exit **CONFPP01**. The coding of the user exit is processed when the user presses the pushbutton *Propose actual data*.

Simplification

For reasons of simplification, the fully qualified paths are specified in hard-coded form for the tags of the data source. The entry of a fully qualified path must always be specified in the *PCo*-internal format (with *"/* as separator). Please note the following:

- Use the test program **RPCO_BS_INT_TEST** or the search help **S_PCO_ELM_BROWSE_TAG** to determine the fully qualified path.
- If the fully qualified path of the tag is too long, you can use function module **S_PCO_CALL_POPUP_NAMESP_BROWS** and evaluate the output parameter **ET_SEL_NODES**. You can get the fully qualified path as a string variable from the component **TAG_ID**.

Implementation of a user exit

1. Start transaction CMOD and create a new project.
2. Assign the enhancement **CONFPP01**.
3. Switch to the component view and position the cursor on the entry **EXIT_SAPLCORF_101** of the function module exit.
4. When you double-click, the system switches to the display of the function module **EXIT_SAPLCORF_101**.
5. Create the implementation for the user exit by putting the cursor on **ZXCOFU11** and double-clicking it.
6. Insert the following source code.
7. Save and activate your changes and finally, activate the user exit.

Sample coding

The following sections of the corresponding ABAP coding are provided below:

- Generation of an application handle
- Instantiation of a wrapper class for the integration of PCo (**CL_PCO_PAC**)
- Creation of a buffer table with information on the tags whose values are to be read
- Calling of the wrapper method for the reading of tag information
- Conversion of query results (data reference) into floating point values and assignment to the corresponding fields of the confirmation structure.
- Error handling or success message

ABAP Coding

```

*&-----*
*&  Include                ZXCOFU1
*&-----*

DATA:
  lt_tag_data      TYPE pco_t_tag_data,
  lt_tag_results  TYPE pco_t_query_result_tag_data.
DATA:
  ls_appl_handle  TYPE pco_s_appl_handle,
  ls_tag_data     TYPE pco_s_tag_data,
  ls_tag_result   TYPE pco_s_query_result_tag_data.
DATA:
  lv_log_handle   TYPE balloghndl,
  lv_msg_txt     TYPE string.
DATA:
  lo_pco_exc      TYPE REF TO cx_pco_bs_int,
  lo_pco_msg      TYPE REF TO cl_pco_query_message,
  lo_pco_pac      TYPE REF TO cl_pco_pac,
  lo_pco_util     TYPE REF TO cl_pco_utility,
  lo_tag_query    TYPE REF TO if_pco_tag_query.

CONSTANTS:
  lc_pco_rfc_dest TYPE rfcdst VALUE 'PCO_BS_INT_UD_Q63',
  lc_tag_alias_quant TYPE string VALUE 'UD_CONF_QUANT',
  lc_tag_alias_rework TYPE string VALUE 'UD_CONF_REWORK',
  lc_tag_alias_scrap TYPE string VALUE 'UD_CONF_SCRAP',
  lc_tag_id_quant     TYPE string
    VALUE 'Channel_0_User_Defined/A_Uwes tags/UD_CONF_QUANT',
  lc_tag_id_rework    TYPE string
    VALUE 'Channel_0_User_Defined/A_Uwes tags/UD_CONF_REWORK',
  lc_tag_id_scrap     TYPE string
    VALUE 'Channel_0_User_Defined/A_Uwes tags/UD_CONF_SCRAP'.

FIELD-SYMBOLS:
  <ld_value> TYPE any.

```

```

CLEAR: lt_tag_data, lt_tag_results.

* General part: Map default values for AFRUD_EXP
MOVE-CORRESPONDING afrud_imp TO afrud_exp.
* UoM
afrud_exp-meinh = caufvd_imp-gmein.

* PCo integration part
ls_appl_handle-appl = 'PRODORCONF'.

* Create application handle: Use order number, plant, sequence,
* operation, and ID of confirmation
CONCATENATE caufvd_imp-aufnr caufvd_imp-werks afrud_imp-aplfl
afrud_imp-vornr afrud_imp-rueck INTO ls_appl_handle-handle.

TRY.
    CREATE OBJECT lo_pco_pac
        EXPORTING
            is_appl_handle = ls_appl_handle
            iv_check_dest  = abap_false
            iv_load_feat   = abap_false
            iv_log_active  = abap_true
            iv_rfc_dest    = lc_pco_rfc_dest.

    lo_tag_query = lo_pco_pac->get_tag_query_obj( ).
* Fill buffer table with tag information to be read for confirmation
* (produced quantity, rework, scrap)
ls_tag_data-tag_id      = lc_tag_id_quant.
ls_tag_data-tag_alias  = lc_tag_alias_quant.
INSERT ls_tag_data INTO TABLE lt_tag_data.
CLEAR ls_tag_data.
ls_tag_data-tag_id      = lc_tag_id_rework.
ls_tag_data-tag_alias  = lc_tag_alias_rework.
INSERT ls_tag_data INTO TABLE lt_tag_data.
CLEAR ls_tag_data.
ls_tag_data-tag_id      = lc_tag_id_scrap.
ls_tag_data-tag_alias  = lc_tag_alias_scrap.
INSERT ls_tag_data INTO TABLE lt_tag_data.
CLEAR ls_tag_data.
* Read tag data via PCo agent
lo_tag_query->read_tag(
    EXPORTING
        it_tag_data = lt_tag_data
    IMPORTING
        et_tag_result = lt_tag_results ).
* Convert data references into confirmation values
LOOP AT lt_tag_results INTO ls_tag_result.
    IF sy-subrc = 0.
        CASE ls_tag_result-datatype.
* Float point values

```

```

    WHEN cl_pco_utility=>gc_sdt_float OR
         cl_pco_utility=>gc_sdt_real OR
         cl_pco_utility=>gc_sdt_double.
*   Get value from data reference
    ASSIGN ls_tag_result-value->* TO <ld_value>.
    IF <ld_value> IS ASSIGNED.
        CASE ls_tag_result-tag_alias.
            WHEN lc_tag_alias_quant.
                afrud_exp-lmnga = <ld_value>.
            WHEN lc_tag_alias_rework.
                afrud_exp-rmnga = <ld_value>.
            WHEN lc_tag_alias_scrap.
                afrud_exp-xmnga = <ld_value>.
        ENDCASE.
    ENDIF.
    WHEN OTHERS.
*   Unsupported data format
    ENDCASE.
ELSE.
    RAISE EXCEPTION TYPE cx_pco_bs_int
    EXPORTING
        textid      = cx_pco_bs_int=>error_query_exec
        error_cause = text-ert.
    ENDIF.
ENDLOOP.
*   Send success message
CONCATENATE text-suc ls_appl_handle-appl ls_appl_handle-handle
    INTO lv_msg_txt SEPARATED BY space.
CREATE OBJECT lo_pco_msg
    EXPORTING
        iv_msg_sev = lo_pco_msg->gc_msg_sev_info
        iv_msg_text = lv_msg_txt.

CATCH cx_pco_bs_int INTO lo_pco_exc.
*   Error occurred
lo_pco_util->add_exc_obj_to_log(
    EXPORTING
        is_appl_handle = ls_appl_handle
        iv_activity     = lo_pco_util->gc_act_read
        iv_bal_subobj   = lo_pco_util->gc_bal_subobj_tag
        iv_msg_severity = cl_pco_query_message=>gc_msg_sev_error
        iv_req_type     = cl_pco_query=>gc_tag_req_type
        io_exc_obj      = lo_pco_exc
    CHANGING
        cv_log_handle  = lv_log_handle ).

lv_msg_txt = lo_pco_exc->get_longtext( ).
MESSAGE lv_msg_txt TYPE cl_pco_utility=>gc_msgty_error.
ENDTRY.

```

Optimization possibilities

There are the following ways of optimizing the performance of the system:

- Dynamic determination of path information via the work center of the operation to be confirmed:
 - Use of characteristics classification for work centers or equipment, in order to store parts of the path as characteristic values
- Evaluation of *PCo* user messages:
 - Table **PCO_MSG_OBJ** of the exception object **LO_PCO_EXC**
 - Evaluation of the exporting parameter **ET_PCO_MSG_OBJ** of method **READ_TAG**

Examples for the generation and processing of notifications

Notifications are generated if the trigger conditions of a subscription are fulfilled by tags. *PCo* sends the notification to the target systems that are assigned in the notification.

In the case of a subscription on the part of *Business Suite* applications, *PCo* first creates a notification for the passed application handle. For this, the *Business Suite* application must select an existing notification template of the corresponding *PCo* agent instance when subscribing to tags. The notification template is created in the *PCo* Management Console for an agent instance. The following information is stored in the notification template:

Information in notification template	Meaning
Trigger type and trigger expression	The trigger type <i>Always</i> is the preferred type for Business Suite integration scenarios. The other trigger types cannot be used fully for Business Suite integration scenarios.
The lifetime of the subscription (in seconds)	The default value for the lifetime is one hour (3,600 seconds).
Settings for reliable connections and reliable delivery of messages	<ul style="list-style-type: none"> • Number of connection tries, if the connection of <i>PCo</i> to an external data source is interrupted • Lifetime of notification messages that could not be delivered. The default value for the lifetime is one day.
<i>PCo</i> target systems to which the notification is to be sent	Here you assign the <i>PCo</i> destination system for your <i>Business Suite</i> system, which is to receive and process the notification message. The connection data to the <i>Business Suite</i> system must previously have been maintained when the corresponding <i>PCo</i> destination system is defined in the <i>PCo</i> Management Console.

1) Conversion of notifications to application logs

Scenario

Using the test program **RPCO_BS_INT_TEST** you can create subscriptions for tag value changes. If the values of these tags change, *PCo* generates a notification and sends it to the *Business Suite* systems that were assigned as PCo destination systems in the notification templates used. From the contents of the notification (header data, error messages, expressions), the system generates application log entries. These application logs can be displayed using the test program.

Implementation

A new class for the processing of notifications is necessary. This class inherits data from class **CL_PCO_NOTIF_HANDLER** and redefines the instance method **EXECUTE**. In addition, the BAdI **BADI_S_PCO_HANDLE_NOTIF** for the filter value APPLICATION = SAPTESTING must also be implemented.

Note:

In package **S_PCO** the above-mentioned ABAP objects are already implemented.

Implementation of a class for the processing of notifications (class **CL_PCO_SAPTESTING_NOTIF**)

Program blocks

- Setting the severity to *Error*, if error messages are part of the notification
- Generation of an application log entry containing the name and the description of the notification
- Evaluation of the passed expressions and generation of the corresponding application log entries
 - Conversion of the expressions into instances of the exception class **CX_PCO_BS_INT**, since this can easily be implemented in application logs
- Creation of error messages for the application log
- Saving of the application logs and posting of the changes

ABAP Coding of the method EXECUTE (class **CL_PCO_SAPTESTING_NOTIF**)

1. First maintain the following method parameters:

Parameter	Type	Data type	Optional	Description
IT_ERROR_MSG	Importing	Type PCO_T_QUERY_MESSAGE_OBJ	X	Error messages for notifications (Object inst.)
IT_EXPR_DATA	Importing	Type PCO_T_EXPR_DATA	X	PCo expression data
IS_APPL_HANDLE	Importing	Type PCO_S_APPL_HANDLE	X	PCo: Data structure for application handle
IS_NOTIF_HEADER	Importing	Type PCO_S_NOTIF_HEADER	X	PCo: Header data for notification
IV_TEST_MODE	Importing	Type BOOLE_D	X	'X': Test mode

2. Next, assign the exception class **CX_PCO_BS_INT** (PCo Suite Integration: Exception Class)
3. After this, copy the following ABAP coding into the method:

```
METHOD execute.
* This redefined method writes data of sent notification into
* application log
```



```

DATA:
  lt_exc_obj      TYPE pco_t_exc_obj.
DATA:
  ls_bal_context TYPE bal_s_cont,
  ls_expr_data   TYPE pco_s_expr_data.
DATA:
  lv_log_handle  TYPE balloghndl,
  lv_msg_severity TYPE s_pco_query_message_severity,
  lv_msg_stext   TYPE text40,
  lv_msg_txt     TYPE string.
DATA:
  lo_exc_obj     TYPE REF TO cx_pco_bs_int,
  lo_pco_util    TYPE REF TO cl_pco_utility.

* Determine message severity for application log messages
IF it_error_msg[] IS INITIAL.
  lv_msg_severity = cl_pco_query_message=>gc_msg_sev_info.
ELSE.
  lv_msg_severity = cl_pco_query_message=>gc_msg_sev_error.
ENDIF.

* Notification header: Add message containing notification name and
* description
CLEAR: lv_msg_txt, lv_msg_stext, lo_exc_obj.
CONCATENATE text-not is_notif_header-name text-des
  is_notif_header-descr INTO lv_msg_txt SEPARATED BY space.
WRITE lv_msg_txt TO lv_msg_stext.
CREATE OBJECT lo_exc_obj
  EXPORTING
    msg_stxt = lv_msg_stext
    msg_ltxt = lv_msg_txt.
INSERT lo_exc_obj INTO TABLE lt_exc_obj.

* Notification header: Add message containing notification destination,
* ID and status
CLEAR: lv_msg_txt, lv_msg_stext, lo_exc_obj.
CONCATENATE text-dst is_notif_header-dest text-nid
  is_notif_header-id text-sta is_notif_header-status
  INTO lv_msg_txt SEPARATED BY space.
WRITE lv_msg_txt TO lv_msg_stext.
CREATE OBJECT lo_exc_obj
  EXPORTING
    msg_stxt = lv_msg_stext
    msg_ltxt = lv_msg_txt.
INSERT lo_exc_obj INTO TABLE lt_exc_obj.

* Add expression data of PCo notification message
LOOP AT it_expr_data INTO ls_expr_data.
  CLEAR: lv_msg_txt, lv_msg_stext, lo_exc_obj.
  CONCATENATE text-not text-nam ls_expr_data-name
    text-val ls_expr_data-value INTO lv_msg_txt SEPARATED BY space.

```

```

WRITE lv_msg_txt TO lv_msg_stext.
CREATE OBJECT lo_exc_obj
  EXPORTING
    msg_stxt = lv_msg_stext
    msg_ltxt = lv_msg_txt.
INSERT lo_exc_obj INTO TABLE lt_exc_obj.
ENDLOOP.

CREATE OBJECT lo_pco_util.
lo_pco_util->add_exc_obj_to_log(
  EXPORTING
    it_exc_obj      = lt_exc_obj
    is_appl_handle  = is_appl_handle
    iv_activity     = cl_pco_utility=>gc_act_not_cbck
    iv_bal_subobj   = cl_pco_utility=>gc_bal_subobj_tag
    iv_msg_severity = lv_msg_severity
    iv_req_type     = cl_pco_query_xml_builder=>gc_req_type_tag
  CHANGING
    cv_log_handle  = lv_log_handle ).

* Add error messages to application log
IF NOT it_error_msg[] IS INITIAL.
  lo_pco_util->create_msg_context(
    EXPORTING
      is_appl_handle = is_appl_handle
      iv_activity     = cl_pco_utility=>gc_act_not_cbck
      iv_req_type     = cl_pco_query=>gc_tag_req_type
    IMPORTING
      es_msg_context = ls_bal_context ).

  lo_pco_util->add_msg_obj_to_log(
    EXPORTING
      it_msg_obj      = it_error_msg
      is_msg_context  = ls_bal_context
      iv_log_handle   = lv_log_handle ).

ENDIF.

* Save all application logs
lo_pco_util->save_logs(
  EXPORTING
    iv_execute_commit = abap_true
    iv_in_update_task = abap_false
    iv_log_handle     = lv_log_handle
    iv_save_all       = abap_false ).

* Execute COMMIT WORK to store application log entries on database
CALL FUNCTION 'DB_COMMIT'.

ENDMETHOD.

```

Implementation of class CL_PCO_IM_SAPTESTING_NOTIF for BAdI implementation

Program block

- Return of the class name of the class processing notifications (**CL_PCO_SAPTESTING_NOTIF**), if the application name for the application handle has the value **SAPTESTING**.

ABAP Coding of the method **IF_EX_S_PCO_HANDLE_NOTIF~GET_CLASS_NAME**

- First create the following parameters for the method:

Parameter	Type	Data type	Optional	Description
IS_APPL_HANDLE	Importing	Type PCO_S_APPL_HANDLE		PCo: Data structure for application handle
EV_CLASS_NAME	Exporting	Type SEOCLSNAM		Name of the class processing notifications

- Then assign the exception class **CX_PCO_BS_INT** (PCo Suite Integration: Exception Class).
- Finally, copy the following ABAP coding into the method:

```

METHOD if_ex_s_pco_handle_notif~get_class_name.
* Processing of notifications is executed with the help of classes that
* inherit super class CL_PCO_NOTIF_HANDLER.
* Based on entered application handle information this BAdI
* implementation returns the name of handler class

CONSTANTS:
  lc_appl_name_saptesting  TYPE pco_s_appl_handle-appl
    VALUE 'SAPTESTING',
  lc_class_name_saptesting TYPE seoclsname
    VALUE 'CL_PCO_SAPTESTING_NOTIF'.

CLEAR: ev_class_name.

IF is_appl_handle-appl = lc_appl_name_saptesting.
  ev_class_name = lc_class_name_saptesting.
ENDIF.

ENDMETHOD.

```

2) Automatic processing of time ticket confirmations

Scenario

Time ticket confirmations are to be carried out automatically as soon as values for manufactured quantities, scrap and rework change. These values are assigned to tags for which a subscription exists.

Implementation

A new class for the processing of notifications is required. From class **CL_PCO_NOTIF_HANDLER**, this class inherits data and redefines the instance method **EXECUTE**. ABAP Objects. A redefinition can be used to reimplement an instance method in a subclass without changing the interface.

In addition, the BAdI **BADI_S_PCO_HANDLE_NOTIF** must be implemented using the filter value **APPLICATION = PRODORDCON**. The application handle contains the unique confirmation number as part of the handle; the other confirmation data, such as the order and operation number, is completed via the confirmation number.

Simplified procedure

- No evaluation of error messages that can be part of a notification.
- Use of hard-coded names for tags (evaluation of expressions in the notification)
- Reading of order number and operation data from the database (per SELECT)
- No rework of confirmations for which the corresponding goods movement could not be carried out

Implementation of class for the processing of notifications (class **ZUD_RK_CONF**)

Program sections

- Determination of the confirmation number from the application handle of the confirmation.
- Read operation for confirmation number
- Read order number of the corresponding production order from the database
- Determination of sequence and operation number
- Generation of confirmation using the BAPI **BAPI_PRODORDCONF_CREATE_TT**.
- Execution of posting by calling the function module **BAPI_TRANSACTION_COMMIT**

ABAP Coding of the method **EXECUTE** (class **ZUD_RK_CONF**)

1. First create the following parameters for the method:

Parameter	Type	Data type	Optional	Description
IT_ERROR_MSG	Importing	Type PCO_T_QUERY_MESSAGE_OBJ	X	Error messages for notifications (Object inst.)
IT_EXPR_DATA	Importing	Type PCO_T_EXPR_DATA	X	PCo expression data
IS_APPL_HANDLE	Importing	Type PCO_S_APPL_HANDLE	X	PCo: Data structure for application handle
IS_NOTIF_HEADER	Importing	Type PCO_S_NOTIF_HEADER	X	PCo: Header data for notification
IV_TEST_MODE	Importing	Type BOOLE_D	X	'X': Test mode

2. Next, assign the exception class **CX_PCO_BS_INT** (PCo Suite Integration: Exception Class)
3. After this, copy the following ABAP coding into the method:

```
METHOD execute.
DATA:
  lt_conf          TYPE STANDARD TABLE OF bapi_pp_timeticket.
DATA:
  ls_appl_handle  TYPE pco_s_appl_handle,
  ls_cafko_ru     TYPE cafko_ru,
  ls_cafvc_ru     TYPE cafvc_ru,
  ls_afvc         TYPE afvc,
  ls_conf         TYPE bapi_pp_timeticket,
  ls_expr_data    TYPE pco_s_expr_data.

* Get confirmation number from handle information
ls_conf-conf_no = is_appl_handle-handle.

CALL FUNCTION 'CO_DB_CAFVC_RU_READ'
  EXPORTING
    rueck_imp      = ls_conf-conf_no
  IMPORTING
    cafvc_ru_exp  = ls_cafvc_ru
  EXCEPTIONS
    not_found     = 1
    OTHERS        = 2.
IF sy-subrc <> 0.
  RAISE EXCEPTION TYPE cx_pco_bs_int
  EXPORTING
    textid        = cx_pco_bs_int=>error_query_exec
    error_cause   = 'Error reading order data'.
ENDIF.

* Determine order number
SELECT SINGLE aufnr FROM afko INTO ls_conf-orderid
  WHERE
    aufpl = ls_cafvc_ru-aufpl.

* Determine order operation data for which confirmation shall
* be executed
SELECT SINGLE * FROM afvc INTO ls_afvc
  WHERE
    aplz1 = ls_cafvc_ru-aplz1 AND
    aufpl = ls_cafvc_ru-aufpl.

ls_conf-sequence = ls_afvc-plnfl.
ls_conf-operation = ls_afvc-vornr.

LOOP AT it_expr_data INTO ls_expr_data.
  CASE ls_expr_data-name.
    WHEN 'UD_CONF_QUANT'.
      ls_conf-yield = ls_expr_data-value.
    WHEN 'UD_CONF_SCRAP'.
      ls_conf-scrap = ls_expr_data-value.
    WHEN 'UD_CONF_REWORK'.
```

```

        ls_conf-rework = ls_expr_data-value.
    WHEN 'UD_STATUS'.
        ls_conf-fin_conf = ls_expr_data-value.
    ENDCASE.
ENDLOOP.

```

```

INSERT ls_conf INTO TABLE lt_conf.

```

```

* Use BAPI to create confirmation
CALL FUNCTION 'BAPI_PRODORDCONF_CREATE_TT'
    TABLES
        timetickets = lt_conf.

```

```

* Execute COMMIT WORK
CALL FUNCTION 'BAPI_TRANSACTION_COMMIT'.
ENDMETHOD.

```

Implementation of class ZUD_RK_BADI_KONF for BAdI implementation

Program block

- Return of the class name of the handler class (**ZUD_RK_CONF**), if the application name of the application handle has the value "SAPTESTING".

ABAP Coding of the method IF_EX_S_PCO_HANDLE_NOTIF~GET_CLASS_NAME

- First create the following parameters for the method:

Parameter	Type	Data type	Optional	Description
IS_APPL_HANDLE	Importing	Type PCO_S_APPL_HANDLE		PCo: Data structure for application handle
EV_CLASS_NAME	Exporting	Type SEOCLSNAME		Name of the class processing the notification

- Next assign exception class **CX_PCO_BS_INT** (PCo Suite Integration: exception class).
- Finally, copy the following ABAP Coding into the method:

```

METHOD if_ex_s_pco_handle_notif~get_class_name.

CONSTANTS:
    lc_appl_name_prodordcon  TYPE pco_s_appl_handle-appl
        VALUE 'PRODORDCON',
    lc_class_name_zud_rk_conf TYPE seoclsname
        VALUE 'ZUD_RK_CONF'.

CLEAR ev_class_name.

IF is_appl_handle-appl = lc_appl_name_prodordcon.

```

```
    ev_class_name = lc_class_name_zud_rk_conf.  
ENDIF.  
ENDMETHOD.
```

Appendix: Tips and Tricks

1) Conversion of timestamp formats

Timestamps generated and processed in ABAP do not exist in a timestamp format which can be used in JAVA or other programming languages. In order to convert the ABAP timestamp value into the ISO format `yyyy-mm-ddThh:mm:ss` you can use the following ABAP expression starting with Basis release 7.02:

```
ev_conv_tstmp = |{ iv_time_stamp TIMESTAMP = ISO TIMEZONE = 'UTC ' }|
```

2) Linking of individual quotation marks with text string

Use case: If string variables are linked, they should be set in single quotes.

Example: Output of text strings 'Templates **text**-Module'

```
CONCATENATE 'Templates' 'text' '-Module' INTO lv_result.
```

Result: `lv_result = 'Templatetext-Module'`

Solution: Use of double quotes, in order to bracket a text string with single quotes. See the constant `GC_TXT_QM` of the class `CL_PCO_QUERY`:

```
CONCATENATE 'Templates' cl_pco_query=>gc_txt_qm 'text' cl_pco_query=>gc_txt_qm  
  '-Module' INTO lv_result.
```

Result: `lv_result = 'Template' text '-Module'`

3) Linking of space/blank with a text string

Use case: The linking of text string variables containing a blank (SPACE) leads to a text string in which the blanks no longer exist.

Example: Output of the text string 'ABCD EFGH'

```
lv_string_1 = 'ABCD '      lv_string_2 = 'EFGH'
```

```
CONCATENATE lv_string_1 lv_string_2 INTO lv_string_3.
```

Result: `lv_string_3 = 'ABCDEFGH'`

Solution: Use of single quotes to mask blanks. See the constant `GC_TXT_SP` of class `CL_PCO_QUERY`.

```
CONCATENATE lv_string_1 cl_pco_query=>gc_txt_sp lv_string_2 INTO lv_string_3.
```

Result: `lv_string_3 = 'ABCD EFGH'`

4) Conversion and formatting of floating point numbers

Floating point numbers can exist in two formats: the scientific format (1.2345E01) or the decimal format (12.345). If floating point numbers are to be displayed in the ABAP system, one of three decimal point formats must be used. Via the user master record (table USR01, field DCPFM), the system determines the formatting settings of the user who is logged on. The following format types exist in the system:

- Decimal format: 1,234,567.89 (USR01-DCPFM = 'X')
- Decimal format: 1 234 567,89 (USR01-DCPFM = 'Y')
- Decimal format: 1.234.567,89 (USR01-DCPFM = '')

With respect to the integration of the Business Suite applications, there are two different cases:

- Floating point numbers are converted to string variables and added to the CDATA segment of the PCo Query. The Business Suite application then transfers the contents of the PCo query to PCo per RFC call.
- The Business Suite contains floating point numbers as the result of PCo queries in the form of data references. The data references are converted to string variables and output on the *Results* screen of the test program.

If you are working with the “thousand” separator, the correct formatting of floating point numbers for the display of values is difficult to achieve: No SAP standard solution exists for this problem. Other approaches to this problem, also discussed on the internet, might be [SDN thread 1449990](#) and [SDN thread115017](#), however, these may lead to inadequate results.

Transfer of floating point numbers from the Business Suite to PCo

In order to transfer floating point numbers from Business Suite applications to PCo, the conversion to a string value must be carried out. The string value is then a part of the query data string that is subsequently passed to PCo. At the same time, there is a standardization of decimal separators and removal of the thousand separator (see class **CL_PCO_UTILITY**, method **CONVERT_STRVAL_TO_SDT_DREF**). PCo can then process the string values formatted in this way without further conversion steps.

Display of floating point numbers that are transferred from PCo to the Business Suite

In the test program **RPCO_BS_INT_TEST** the display of floating point numbers is in the decimal format. Using the instance method **FORMAT_FLOAT_CHAR** (class **CL_PCO_UTILITY**), the system sets the thousand and decimal separators according to the formatting settings made in the master record of the current user. In this method, a special algorithm has been defined, which ensures the correct positioning of the thousand and decimal separators.