



# AP 7.00

## Pricing Userexit Manual

OVERVIEW

DEVELOPMENT

UP-GRADE

VERSION 1.04 - 02 APRIL 2008 - © SAP AG 2008

## CHANGE LOG

Version	Date	Modification
1.00	01-DEC-2005	First release for AP7.00 SP03 (SP02 with notes 902329 and 902038)
1.01	19-JAN-2006	Additional examples in Chapter 4.2 and list of incompatible interface changes in chapter 7.2
1.02	06-APR-2006	Enhanced delivered Eclipse Project Template and revised chapter 4.2
1.03	25-JUL-2006	Enhanced chapter 2.5 and 2.10. Added appendixes.
1.04	02-APR-2008	Refined chapter 2, included remarks regarding right JDK version and IDEs.

© Copyright 2008 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®, NT®, EXCEL®, Word®, PowerPoint® and SQL Server® are registered trademarks of Microsoft Corporation.

IBM®, DB2®, DB2 Universal Database, OS/2®, Parallel Sysplex®, MVS/ESA, AIX®, S/390®, AS/400®, OS/390®, OS/400®, iSeries, pSeries, xSeries, zSeries, z/OS, AFP, Intelligent Miner, WebSphere®, Netfinity®, Tivoli®, Informix and Informix® Dynamic Server™ are trademarks of IBM Corporation in USA 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®, the Citrix logo, ICA®, Program Neighborhood®, MetaFrame®, WinFrame®, VideoFrame®, MultiWin® and other Citrix product names referenced herein are trademarks of Citrix Systems, Inc.

HTML, DHTML, XML, XHTML 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.

MarketSet and Enterprise Buyer are jointly owned trademarks of SAP AG and Commerce One.

SAP, R/3, mySAP, mySAP.com, xApps, xApp, 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 in several other countries all over the world. All other product and service names mentioned are the trademarks of their respective companies.



# CONTENTS

<b>Change Log .....</b>	<b>2</b>
<b>Contents .....</b>	<b>4</b>
<b>1 Overview.....</b>	<b>7</b>
1.1 Motivation .....	7
1.2 What changed?.....	7
1.3 Walk through .....	7
1.4 Prerequisites.....	8
1.5 Further Information .....	9
1.6 SAP E-Commerce for ERP using VMC .....	9
1.7 SAP Support for User Exits Implementations.....	9
<b>2 Development Environment .....</b>	<b>10</b>
2.1 Setting up Java .....	10
2.2 Setting up Eclipse .....	10
2.3 The Prepared Eclipse Project .....	10
2.4 AP Pricing Userexits .....	12
2.5 Restriction on the Java Implementation .....	12
2.6 Create the PRC_UE_CUSTOMER.jar .....	13
2.7 Uploading the Developed Userexits .....	13
2.8 Transport of Java Coding.....	14
2.9 Check for Uploaded Java Userexits .....	14
2.10 Deletion of Userexits.....	14
<b>3 Administration .....</b>	<b>15</b>
3.1 Overview about different Userexit Types.....	15
3.2 Register an Implementation .....	16
3.3 Assign Implementations to Formula .....	18
<b>4 Available Exits and APIs .....</b>	<b>20</b>
4.1 Logging capabilities .....	20
4.2 Userexit Types.....	20
4.2.1 Number Dependent Userexits .....	20
4.2.2 Userexits with Multiple Active Implementations .....	26
4.2.3 Userexits with one unique implementation .....	27
<b>5 Debugging.....</b>	<b>30</b>
5.1 Set VMC in Debug Mode .....	30
5.2 Attach Eclipse Java Debugger .....	30
5.3 View VMC Log Files .....	30
<b>6 Upgrade Guide.....</b>	<b>32</b>

6.1	Preparation .....	34
6.2	Implementation .....	34
6.3	Register and Assign the Userexit .....	36
6.4	Uploading and Testing .....	37
<b>7</b>	<b>Interfaces and Incompatible Changes .....</b>	<b>38</b>
7.1	Interface Packages .....	38
7.1.1	Condition Finding Interfaces .....	38
7.1.2	Pricing Interfaces .....	38
7.1.3	Document (Sales/Purchase Order) Interfaces .....	38
7.2	Incompatible Interface Changes to earlier releases .....	41
7.2.1	IConditionFindingManagerUserExit .....	41
7.2.2	IPricingDocumentUserExit .....	41
7.2.3	IPricingItemUserExit .....	42
7.2.4	IPricingConditionUserExit and IGroupConditionUserExit .....	43
7.2.5	ILastPrice .....	44
7.2.6	IDocumentUserExitAccess .....	45
7.2.7	IItemUserExitAccess .....	46
7.2.8	ISPCDocumentUserExitAccess .....	47
7.2.9	ISPCItemUserExitAccess .....	48
7.2.10	IPricingUserExits .....	48
7.2.11	IDocumentUserExit .....	50
7.2.12	IItemUserExit .....	51
7.2.13	ISPCItemUserExit .....	52
<b>A</b>	<b>Background on Java Restrictions .....</b>	<b>53</b>
A.1	Shared Memory caused restrictions .....	53
A.2	Reuse of Java VM caused restrictions .....	54
<b>B</b>	<b>Other ways to influence Pricing .....</b>	<b>55</b>
B.1	Passing additional information to pricing .....	55
B.2	External Data Source for Conditions .....	55
<b>C</b>	<b>FAQ .....</b>	<b>56</b>
C.1	Implementation .....	56
C.2	Troubleshooting .....	57



# 1 OVERVIEW

## 1.1 Motivation

SAP developed a very stable application server and also organized one of the first reliable development systems for large business applications. It includes everything a developer need. Additionally it includes a versioning and transport system, so even large application landscapes can be provided with changes and supplementary developed parts.

The server contains many preventive measures to stabilize it and improve the performance of the running applications on top. Some of these measures are separation of user sessions and OS processes, improved DB transaction handling, better memory footprint and much more.

Now SAP also goes that direction and applies nearly all server features not only to ABAP but also to Java. The Virtual Machine Container (VMC) runs Java, reads its code from the database and offers the same stability as the kernel to ABAP. All parts of the Internet Pricing Configurator (IPC) use now the new server infrastructure with Application Platform (AP) 2005 (e.g. delivered with CRM 5.0) and are called Pricing, Tax and Configuration Engines.

## 1.2 What changed?

With the move from IPCs own server technology to the VMC, SAP also changed the extension technique of allowing customer to add special coding to the existing functionality in pricing and condition technique.

Chapter 6 contains some helpful comments, how to adapt CRM 3.0, 3.1 and 4.0 user exits to the new way in AP 2005.

## 1.3 Walk through

An end-to-end example will be used in this document. It is a very short description and refers to a lot of details given in the rest of the document.

A customer would like to overturn the condition value calculation in pricing and include a special rounding algorithm. His requirement is that the given condition value is rounded and the value of 0.01 per given quantity is subtracted from it.

Example:

Item	Condition Value	Value Rounded	Expected
3 PC	99.70 EUR	100 EUR	99.97 EUR
8 BOX	40.50 EUR	41 EUR	40.92 EUR
2 KG	1.10 EUR	1 EUR	0.98 EUR

The customer would first setup his development environment (see first part of chapter 2), understand which user exit type to use (chapter 4, in this case 4.2.1.4), administer the meta data for that user exit (chapter 3) and then upload his development to the system (see last part of chapter 2).

The coding needed will be a condition value formula as it changes the automatically calculated condition value and will look like this:

```
ZSpecialRoundingValueFormula
package your.company.pricing.userexits;

import java.math.BigDecimal;

import com.sap.spe.base.logging.UserexitLogger;
import com.sap.spe.conversion.ICurrencyValue;
import com.sap.spe.pricing.transactiondata.userexit.IGroupConditionUserExit;
import com.sap.spe.pricing.transactiondata.userexit.IPricingConditionUserExit;
import com.sap.spe.pricing.transactiondata.userexit.IPricingDocumentUserExit;
import com.sap.spe.pricing.transactiondata.userexit.IPricingItemUserExit;
```

```

import com.sap.spe.pricing.transactiondata.userexit.ValueFormulaAdapter;

public class ZSpecialRoundingValueFormula extends ValueFormulaAdapter {

    private static UserexitLogger userexitlogger =
        new UserexitLogger(ZSpecialRoundingValueFormula.class);

    public BigDecimal overwriteConditionValue(IPricingItemUserExit item,
        IPricingConditionUserExit condition) {
        BigDecimal result;

        ICurrencyValue val = condition.getConditionValue();
        userexitlogger.writeLogDebug("old cond value: "
            + val.getValueAsString());

        result = val.getValue().setScale(0, BigDecimal.ROUND_HALF_UP);

        BigDecimal qnt = item.getProductQuantity().getValue();
        qnt = qnt.divide(new BigDecimal("100"), 2, BigDecimal.ROUND_HALF_UP);

        userexitlogger.writeLogDebug("new cond value: " + result.subtract(qnt));

        return result.subtract(qnt);
    }

    public BigDecimal overwriteGroupConditionValue(
        IPricingDocumentUserExit item, IGroupConditionUserExit condition) {
        // do nothing
        return null;
    }
}

```

This coding must be written, compiled and uploaded with help of the eclipse environment and the transaction /SAPCND/UE\_DEV (enter a leading /n while using the direct transaction field).

The metadata entries are maintained using the transaction /SAPCND/UEASS with the usage PR (for pricing). In the implementations part selected with the userexit type VAL, the customer adds the new userexit name ZSPECROUND and goes to details. Here he adds a description and the class name `your.company.pricing.userexits.ZSpecialRoundingValueFormula`. Now the coding is known to pricing and a formula number must be attached to that userexit. In the section formulas with the application CRM (or other) the customer adds the number 600 and selects his userexit name to it.

The only thing left for the customer is to use that formula and put it in a pricing procedure on the right line and restart the transaction using pricing functionality. Due to the buffering mechanism it may be necessary to restart the VMC or a buffer refresh takes place (see note 867428).

## 1.4 Prerequisites

The reader of this manual is expected to have a good understanding of the java language. Additional knowledge about pricing is required if new userexits should be implemented. It is also expected that the user has basic knowledge about the eclipse development environment for java.

Depending on the installed Support Package of AP 7.00 following table shows the SAP notes which must be implemented.

SP	Note	Description
<02		Various issues have been fixed with SP02. We strongly recommend not to use SP00 and SP01
<03	902038	Error messages with unchanged userexit entries in /SAPCND/UEASS
<03	902329	Module and Package checks in /SAPCND/UE_DEV
<03	915882	Attribute mapping in Userexit maintenance
<05	928865	Userexit Development concerning external taxes



## 1.5 Further Information

There are several additional information available elaborating different topics more detailed.

Source	Description
Note 809820	Note containing this document and the prepared eclipse project.
Note 880687	Logging and Tracing
Note 844817	Technical Information
Note 844816	Information for Upgrade
Note 867428	Adapt buffer refresh times
SAP Help	SAP Virtual Machine Container (HELP.BCVMC)

## 1.6 SAP E-Commerce for ERP using VMC

There are several additional notes needed to implement userexits for the VMC in the ERP E-Commerce scenarios.

Source	Description
Note 878865	How to fill userexit tables for the IPC ERP scenario
Note 885415	Error in the maintenance of the user exit Customizing tables
Note 937044	Charactaristic Value Surcharge in ERP ISA scenario
Note 978434	Field assignment for user exits, the 'A' usage not possible
Note 1025553	Missing number ranges for Pricing userexit in ISA R/3 scenario
Note 1035413	Feldzuordnung bei Userexits der Verwendung 'A' nicht möglich

## 1.7 SAP Support for User Exits Implementations

Support of issues with customer user exits are not covered by a normal SAP support contract. This document only helps to implement or let implement user exits and should not be regarded as part of the standard delivery. In case of troubles SAP support may request you to use the (remote) consulting service.

## 2 DEVELOPMENT ENVIRONMENT

The following section describes one way of developing pricing userexits. Only freely available tools which can be downloaded from the internet are used. Other integrated development environments (IDE) do also work but additional effort has to be spent because SAP already delivers with this note an example project for eclipse. Other eclipse based IDEs like NetWeaver Developer Studio may also work with that prepared project template.

### 2.1 Setting up Java

The pricing userexits shall be compiled with the J2SE 1.4.x or a compatible java compiler of version 1.4.x. Also the used libraries must be compatible with J2SE 1.4.x. The java development kit is available for download using <http://java.sun.com/downloads>. Old versions are also available at <http://java.sun.com/javase/downloads/previous.jsp>.

A standard installation of the JDK is sufficient.

It is important that the compiled class files are compatible to a JDK 1.4 version as well as the standard library used is only JDK 1.4. The VMC java environment of SAP BASIS 7.00 does only support 1.4 class files and libraries.

### 2.2 Setting up Eclipse

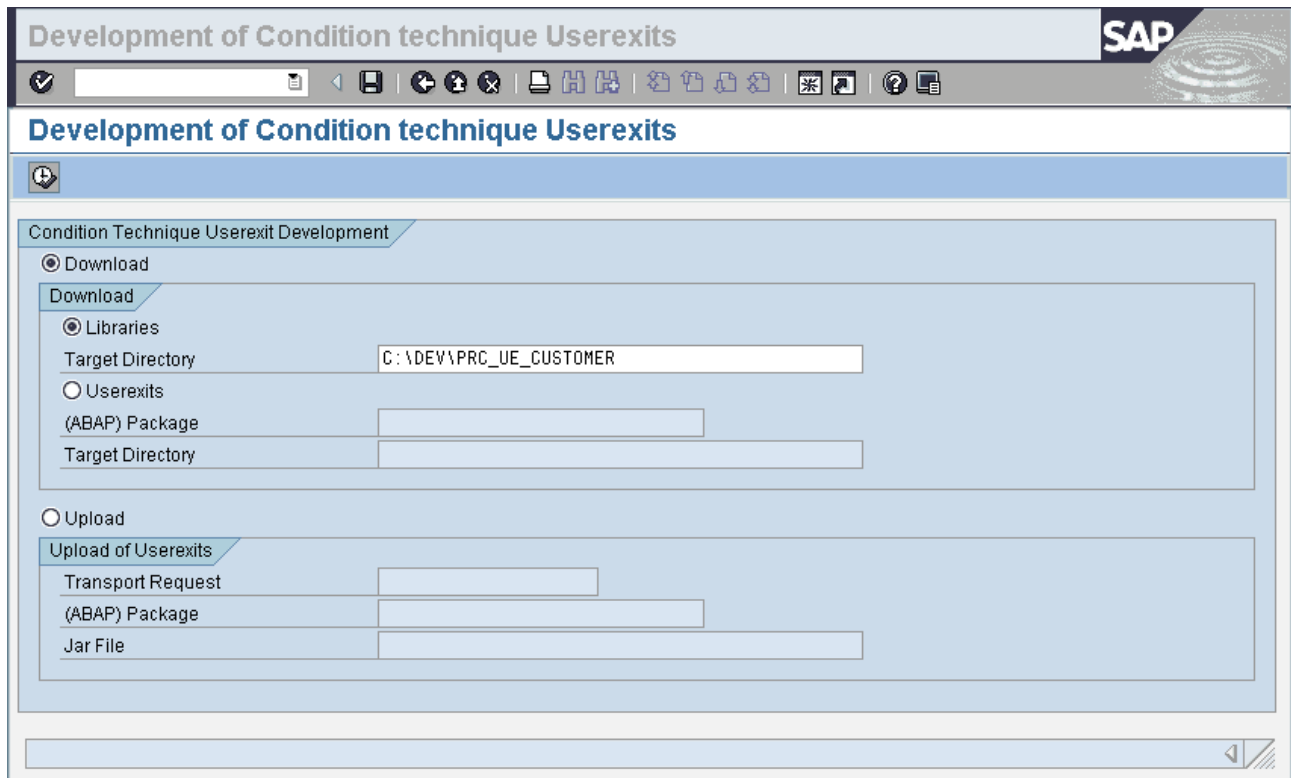
Eclipse 3.1 is recommended and can be downloaded using <http://www.eclipse.org/downloads>. Newer versions may also work. A simple unpacking of the downloaded file is all that is needed to install eclipse.

The eclipse-IDE manages the files in projects and those in workspaces. A workspace contains all administrative data of eclipse. Project files, like java files, may lie also in that workspace but can also be outside. In this document the first approach has been chosen.

### 2.3 The Prepared Eclipse Project

Setting up the prepared project keeps the effort low and helps in fastened results. An experienced user can easily adapt the project layout and location to their needs (e.g. using a source versioning system).

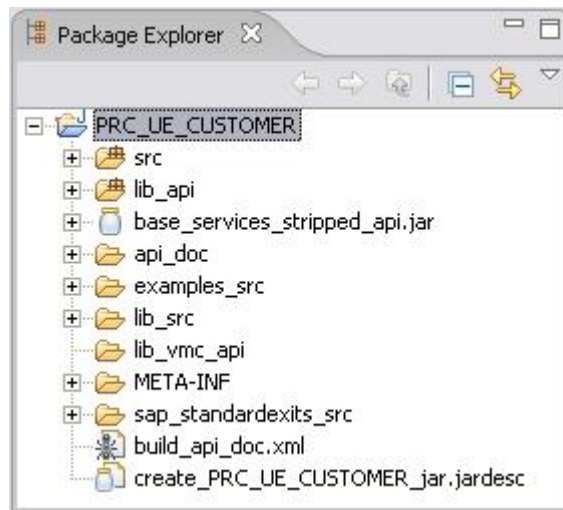
1. Create an empty folder where all relevant data is stored (e.g. C:\DEV). This folder will be the workspace folder for eclipse.
2. Unpack the ZIP file attached to the SAP note into that empty directory (e.g. C:\DEV). Then a subdirectory with the name PRC\_UE\_CUSTOMER should appear (C:\DEV\PRC\_UE\_CUSTOMER), containing the project files.
3. Start your SAP GUI on the same machine, login into the system and start transaction /SAPCND/UE\_DEV (using the command field enter /n/SAPCND/UE\_DEV).



4. Select the Download option and activate the option Libraries. Downloading requires a minimum authorization of exporting data from SAPGUI into the local file system (Object S\_GUI, ACTVT 61)
5. Enter as path the newly created subfolder PRC\_UE\_CUSTOMER (e.g. C:\DEV\ PRC\_UE\_CUSTOMER) and hit the execute button. The downloading will take a few minutes and create some subfolder in the PRC\_UE\_CUSTOMER directory containing some API jars and some source jars
6. Start eclipse and select as workspace the manual created folder (C:\DEV). Eclipse will create the meta data for an empty workspace
7. Select File→Import... and Import an Existing Project into Workspace. Select the PRC\_UE\_CUSTOMER folder as root directory. Eclipse should show the PRC\_UE\_CUSTOMER as project name
8. Now the JRE System Libraries have to be added to the build path of eclipse. Activate the context menu on the project (right mouse click) and select Build Path → Add Libraries. Select JRE System Library and use a 1.4.x version
9. Prepare the API source and build the HTML documentation. Activate the context menu (right mouse click) on build\_api\_doc.xml and select Run As→Ant Build. Please ignore the error message thrown during that process
10. Refresh the eclipse Project by right mouse clicking on the project and selecting Refresh (F5) in the context menu

Now the default project is ready to be used for custom developments. Java Documentation for all API classes is available and can be viewed by eclipse auto completion help or by opening the file PRC\_UE\_CUSTOMER/api\_doc/index.html.

The project has following structure (the screenshot can look different):



The different folders are:

- `src`: should contain the customer development. It is recommended to use also in java some naming convention to better distinguish standard SAP coding and own development. The java packages should e.g. start with `your.company`.
- `lib_api`: contains all relevant APIs needed to compile the userexits
- `api_doc`: contains the generated java API documentation (see `index.html`)
- `examples_src`: contains for each userexit type an example
- `lib_src`: contains jar files with sources to some API classes that can serve as reference documentation too
- `sap_standardexits_src`: contains the source code of the AP 2005 delivered standard SAP pricing userexits only for your reference

## 2.4 AP Pricing Userexits

SAP Application Platform 7.00 delivers also some SAP pricing userexits which includes most of the standard userexits for pricing in ERP (R/3) and previous CRM versions. These standard exits can be found in the project subfolder `sap_standardexits_src` and can be used as a reference for own implementations.

## 2.5 Restriction on the Java Implementation

The java coding of userexits are executed in VMC which restrict and discourage usage of special java functionality. Additionally some restriction concerning database access and usage of SAP JCo applies also.

- Don't use Property-Files (`java.util.Properties`)
- Don't use direct JCo calls or make use of OpenSQL/JDBC/Database/File/OS access (use external datasources for condition types or determine additional attributes before pricing is triggered)
- Don't use non static non-primitive variables (use containers available on the item or document with `set/getObjectForUserExits`). However simple static final members like `final static java.lang.String` or `java.math.BigDecimal` constants can be used.
- Don't use own classloaders
- Don't use Thread and other related functionality (e.g. `synchronized` keyword)
- Don't use own Garbage Collection code or `java.lang.ref.Reference`
- All classes must implement `java.io.Serializable` without custom serialization code and transient fields

Further explanations and background information is found in Appendix A. Some workarounds needed to overcome the restrictions can be found in the Appendix B and the FAQ.

## 2.6 Create the PRC\_UE\_CUSTOMER.jar

After implementing the customer userexits, it is time to upload the userexits classes into the system. However, it is required first to archive all the classes (preferably together with the sources) into a JAR file. Before creating the JAR file, make sure the java sources and compiled classes are up-to-date and error free.

To generate a JAR file eclipse provides the 'JAR Packager' feature that guides you through easy steps to configure and create such a JAR file.

1. From Eclipse's FILE menu choose the EXPORT option.
2. From the pop up list choose JAR FILE and click NEXT.
3. From the resources tree of PRC\_UE\_CUSTOMER choose only the SRC node making sure that the other nodes LIB\_API and LIB\_SRC are not chosen.
4. Make sure that the option 'export generated class files and resources' is selected.
5. Enter a name and path for the JAR file e.g. 'PRC\_UE\_CUSTOMER.jar'.
6. Click on FINISH to generate the JAR file.

To make it even easier, SAP delivers a configuration file called 'create\_PRC\_UE\_CUSTOMER\_jar.jardesc' which contains pre-configuration data that spares the above steps.

Simply right mouse click on the jardesc file within eclipse and select CREATE JAR from the context menu. This will export the JAR file PRC\_UE\_CUSTOMER.jar into Eclipse's workspace folder. That's the default destination and can be changed by using the option "Open JAR Packager ..." in the context menu of the .jardesc file.

Important is that the JAR file may only contain the class files, possibly with its java files along with a META-INF/MANIFEST.MF file. All files must not to use the SAP namespace (com.sap\*).

## 2.7 Uploading the Developed Userexits

Uploading the PRC\_UE\_CUSTOMER.jar file to the database is done via transaction /SAPCND/UE\_DEV (direct start of the transaction possible with a leading /n). This task requires advanced authority settings. Following authorization objects are concerned:

- S\_CTS\_ADMI Administration Functions in Change and Transport System
- S\_DATASET Authorization for file access
- S\_DEVELOP ABAP Workbench
- S\_GUI Authorization for GUI activities
- S\_TCODE Transaction Code Check at Transaction Start
- S\_VMC\_TRAN Transport of Java Development Objects

To upload a library it is required to have a modifiable workbench request and an ABAP package (ABAP development class) which exist beforehand. With selecting the upload option in transaction /SAPCND/UE\_DEV, the transaction request and the path to the jar file is necessary. The uploading is triggered by the execute button.

The jar file shall only contain customer coding. Uploading any other class will lead to an unexpected behavior of VMC and all applications running on VMC. **For each ABAP package only one jar file can be uploaded.**

An already existing version of the userexits is replaced by the new uploaded version (when using the same ABAP package). Each transaction using the VMC which requests classes from the uploaded user exits will load the new version if not already cached (statically buffered). If you upload a new version over an existing JAR file, the new coding is not taken into account automatically. You should reset the VMC (use transaction sm52->Virtual Machine Container->Reset).

After the upload the jar file itself is not required anymore and can be deleted.

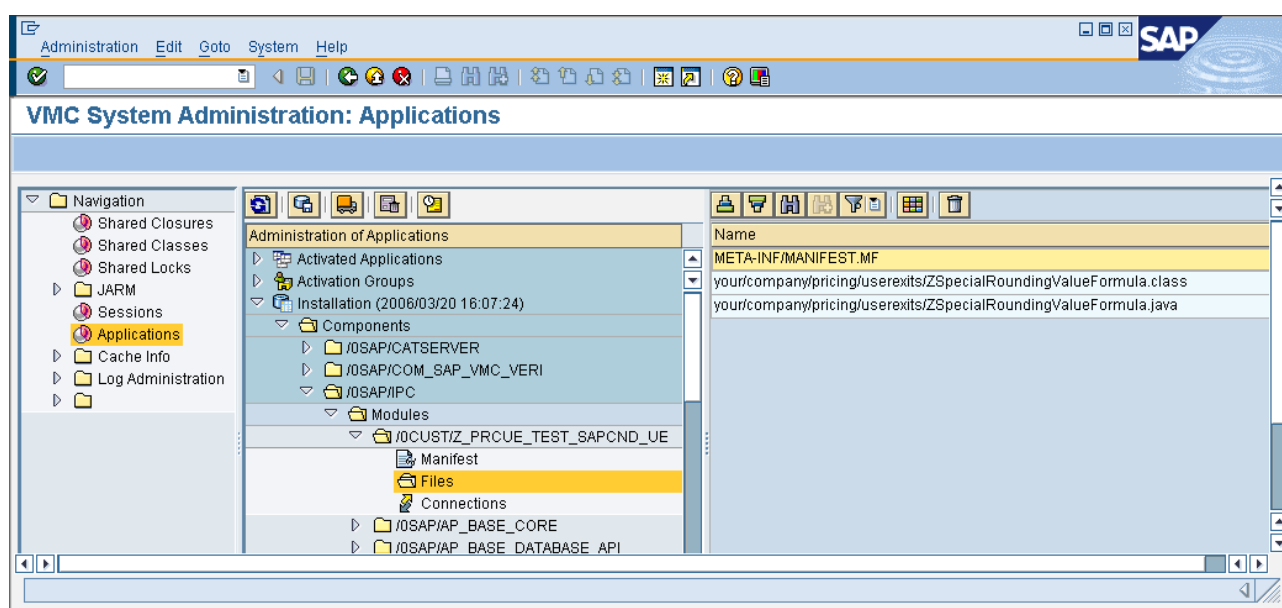
## 2.8 Transport of Java Coding

The transport of java coding is the same for ABAP as for Java. Both work just the same way.

## 2.9 Check for Uploaded Java Userexits

The transaction sm53 contains also a browser to see the installed and uploaded java modules along with the userexit files.

1. Select in the Navigation tree the element Application
2. Browse the Installation tree down to the shown level 0/SAP/IPC→Modules
3. All modules ending with \_SAPCND\_UE are customer uploaded modules equals jar files.



## 2.10 Deletion of Userexits

The deletion of user exits is supported by uploading an empty JAR file to the same ABAP development class/package.

As of NW2004S (7.00) SAP\_BASIS Support Package 09 you can also use the function module (transaction se37) SVMCRT\_TRANS\_JARC\_DELETE to delete the uploaded user exits. The parameters are following:

- PF\_COMPONENTNAME is 'IPC'
- PF\_MODULENAME is the ABAP development class name (the ABAP package) with the suffix '\_SAPCND\_UE'. E.g. if the ABAP package for the upload was '/CUST/PRICING' the module name would be '/CUST/PRICING\_SAPCND\_UE'.
- PF\_TRKORR is the ABAP correction transport request name.

For older SAP\_BASIS SPs the deletion is not supported and an empty jar file must be uploaded into the same ABAP package.

## 3 ADMINISTRATION

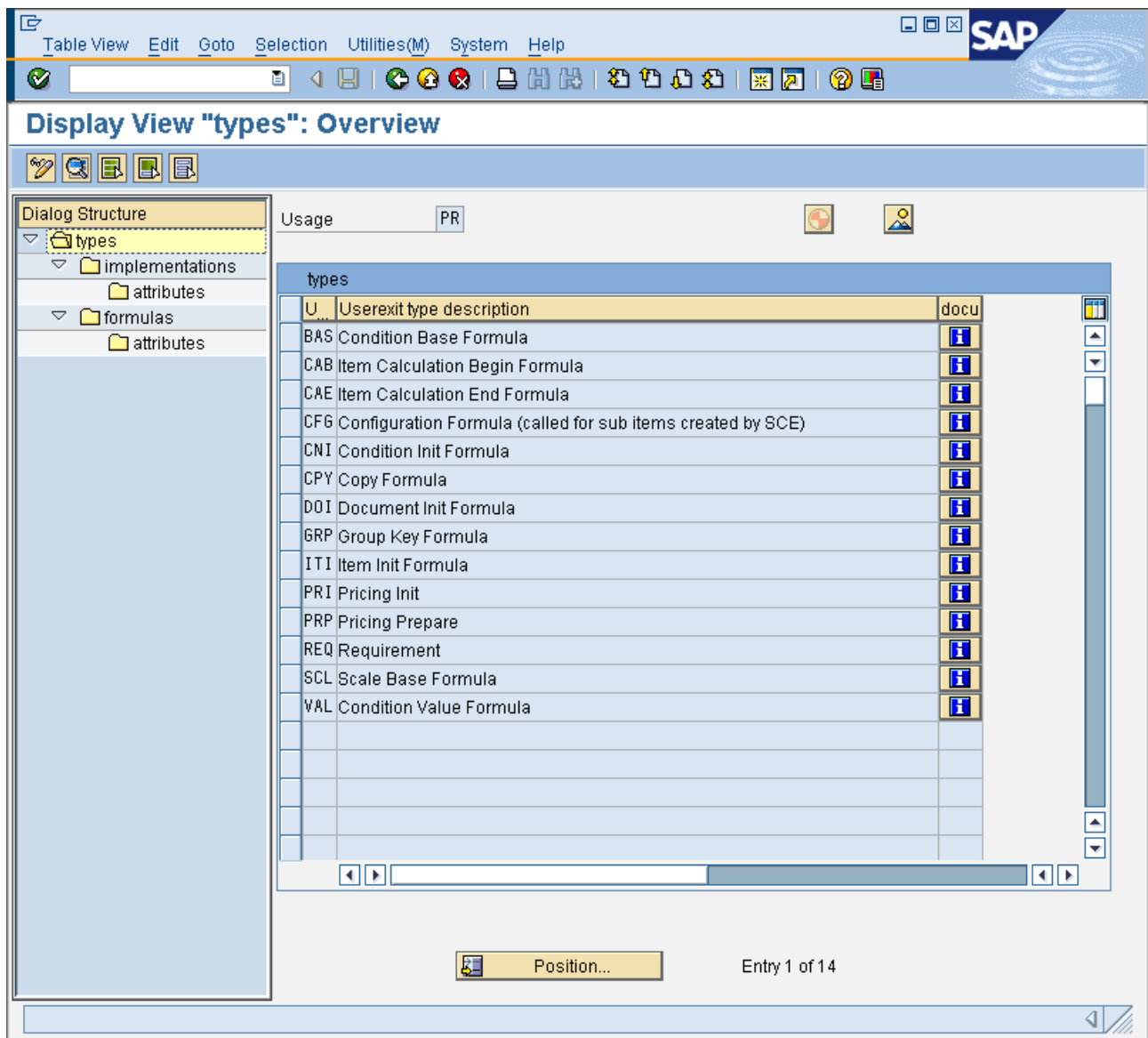
Every pricing userexit (including standard SAP exits) has to be known to the runtime. Each implementation is loaded during runtime and controlled by configuration available through the transaction /SAPCND/UEASS. The work is split into two parts.

1. The implementation (the class) must be registered and a userexit name (a technical name) has to be assigned to it and
2. The assignment of the defined userexit name to a formula number (like the condition value formula number) which can be used in the other configuration parts (e.g. pricing procedure).

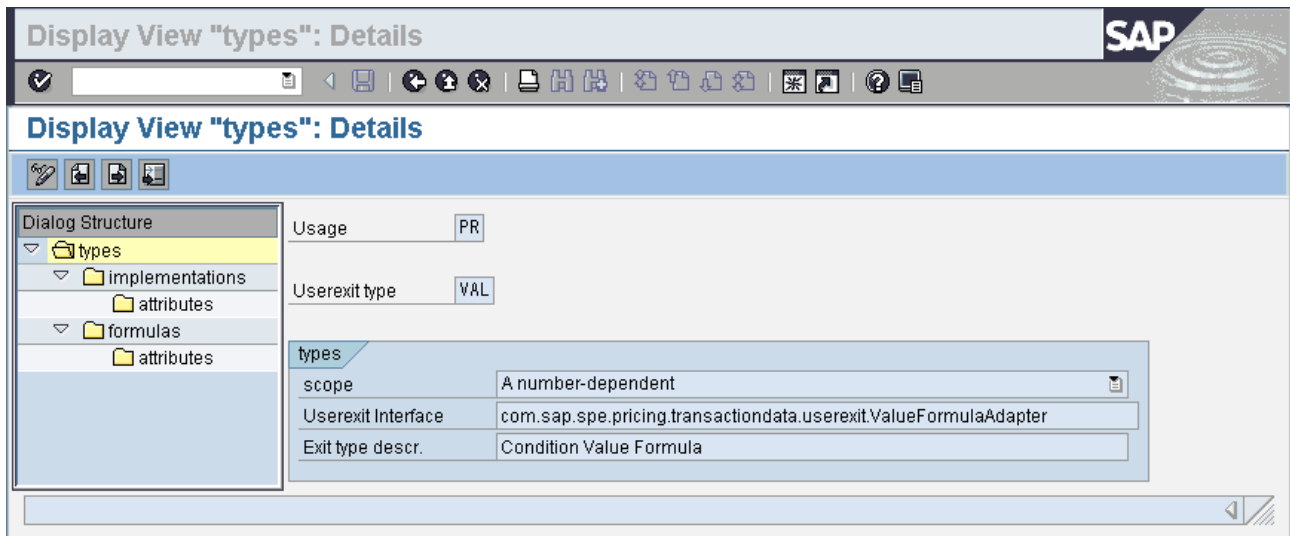
But first a little overview is presented.

### 3.1 Overview about different Userexit Types

After starting the transaction /SAPCND/UEASS and entering the right usage (here PR for Pricing) the following screen will come up.



It shows all different types of available userexits for the selected usage. A short comment is provided on each line by clicking the docu-information-icon. Each type of userexit for the usages PR (pricing), FG (freegoods) and PD (Product substitution) is described in chapter 4.2.



On the detail screen of each userexit type the scope and the userexit interface is given. The scope is one of A, B or C.

- A - number-dependent (like the requirements): The userexit is referenced via a formula number from other configuration like pricing procedure
- B - one unique implementation (like the document init formulas). The userexit with the number 0 will be executed
- C - multiple implementations (like pricing init). Each userexit with an attached formula number will be executed

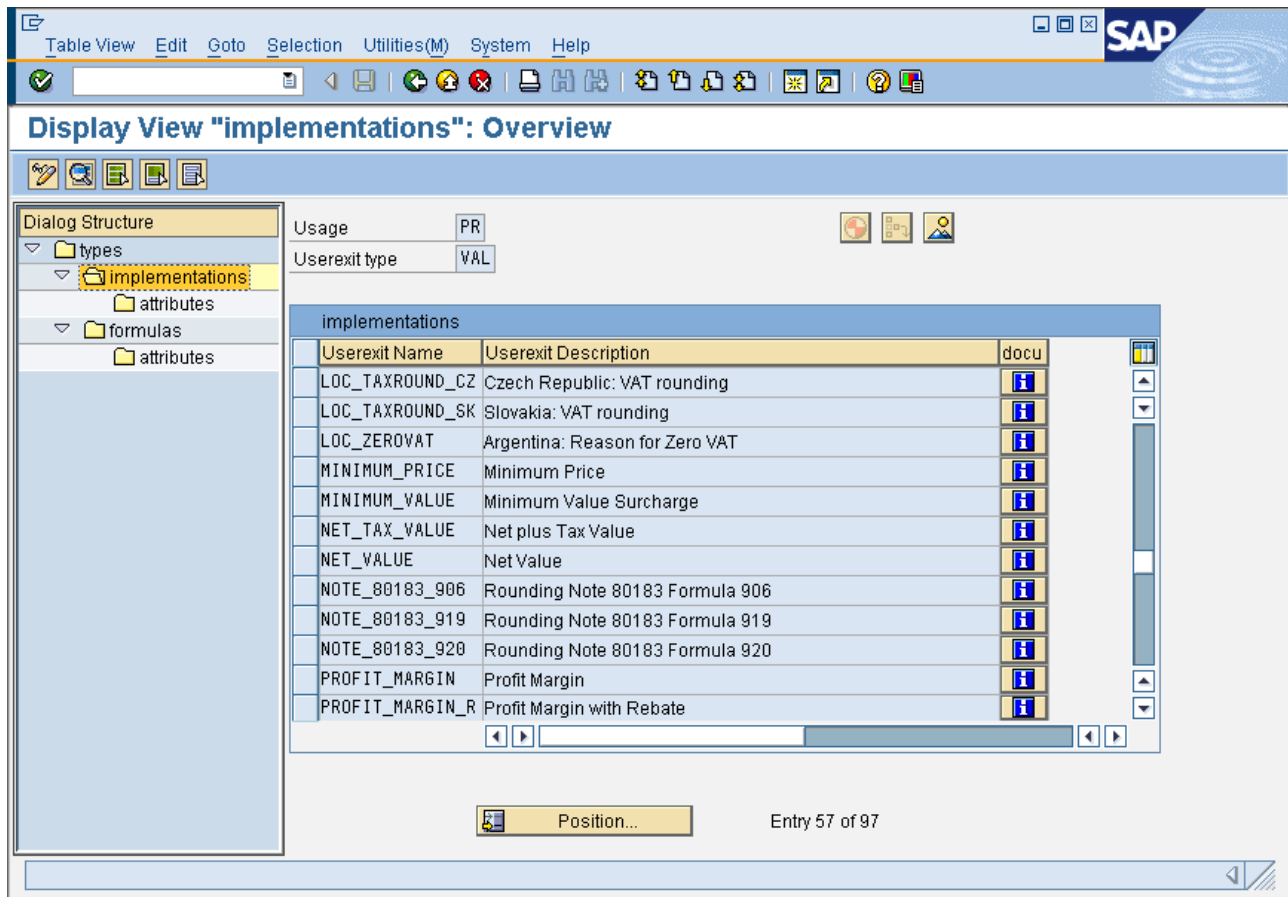
The formula number assignment is explained in chapter 3.3. Also userexit of scope B and C must get a number assigned.

The userexit interface entry describes the adapter class which a userexit must inherit (details in chapter 4.2).

## 3.2 Register an Implementation

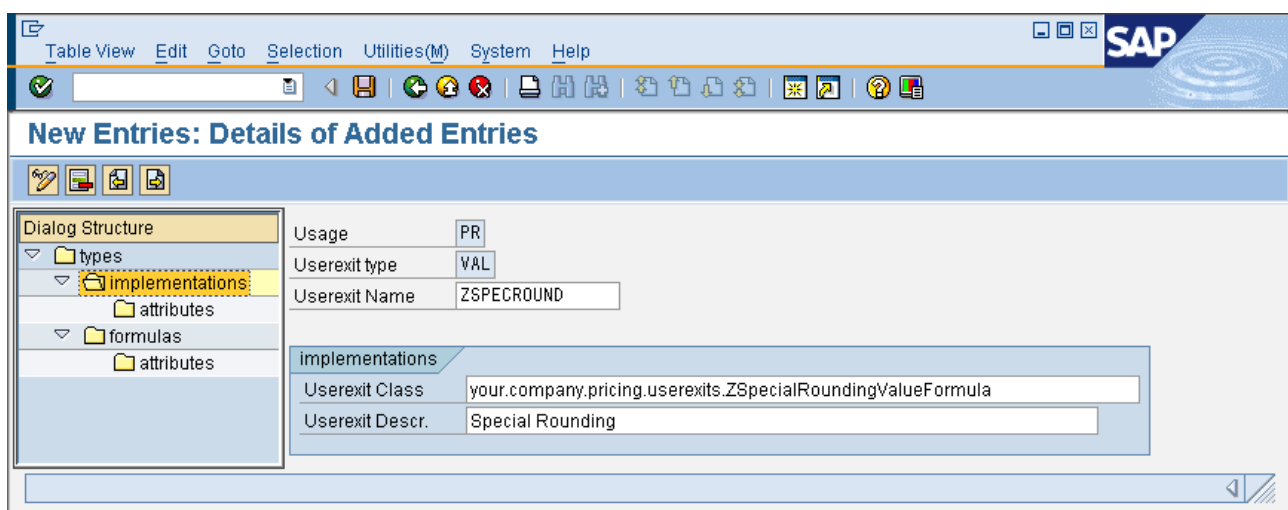
On the view level of implementations the different available implementations of a userexit are given. Here the customer classes must be registered.



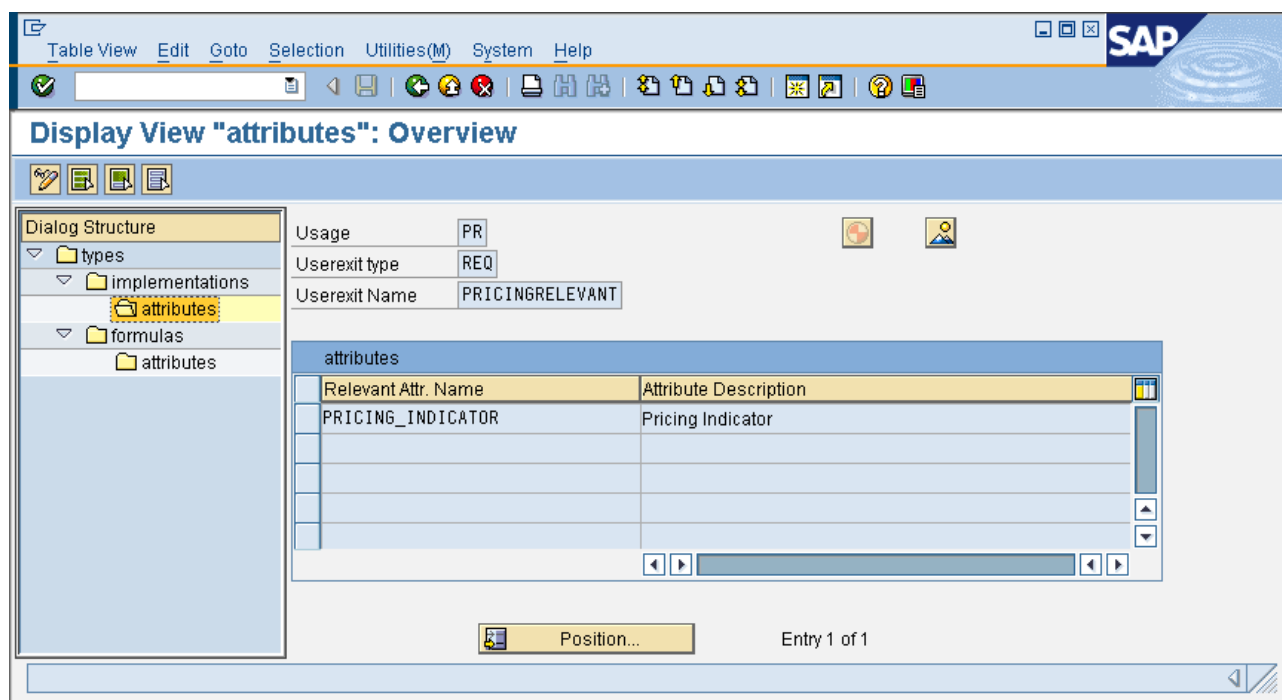


The registering of a new implementation can be done (cross client customizing) by creating a new entry. These data should be entered:

- Userexit name which is a symbolic or short description of the functionality. The customer namespace starts with Y or Z.
- Name of the implementation class (e.g. your.company.pricing.userexits.SpecialBaseFormula). There is no restriction on the name but it should be different from com.sap\*.
- And additional a long description.



If a userexit depends on attributes passed by the calling application, the attributes names (used in the implementation) must be entered on the sub screen attributes along with a description.



**Display View "attributes": Overview**

Dialog Structure: types, implementations, **attributes**, formulas, attributes

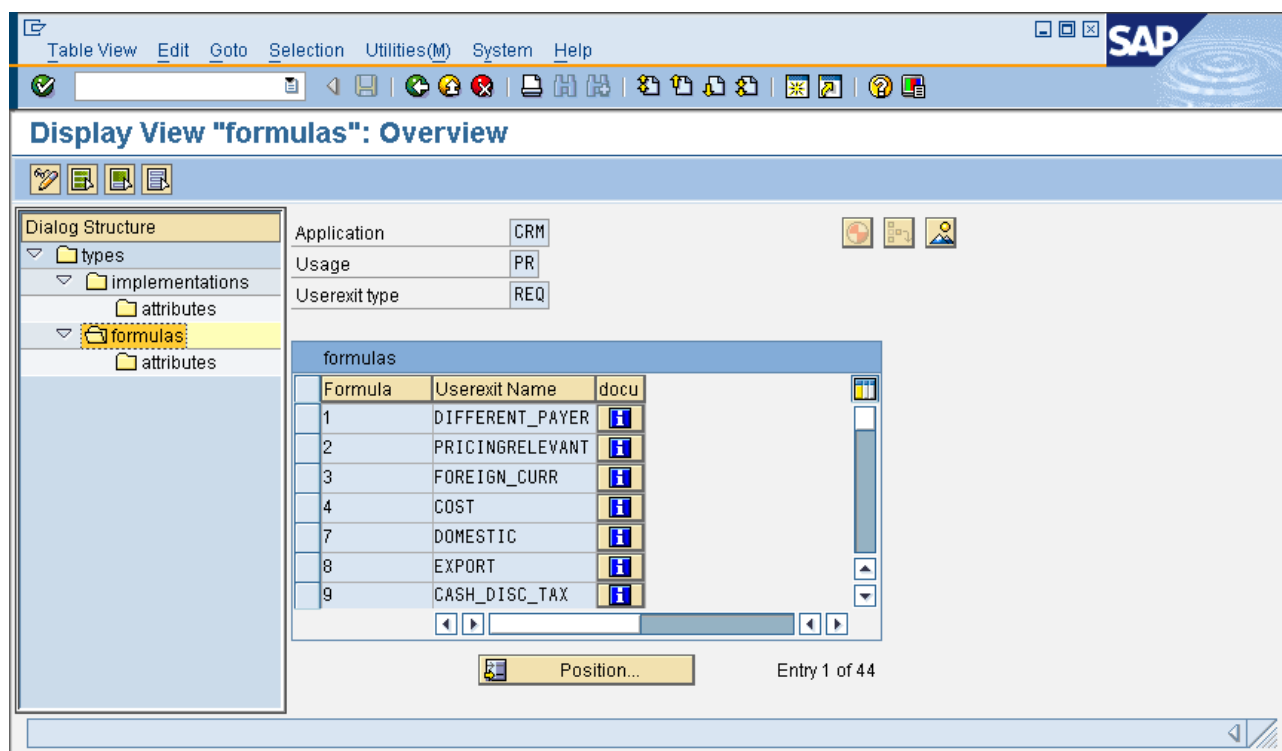
Usage: PR  
 Userexit type: REQ  
 Userexit Name: PRICINGRELEVANT

Relevant Attr. Name	Attribute Description
PRICING_INDICATOR	Pricing Indicator

Position... Entry 1 of 1

### 3.3 Assign Implementations to Formula

After the registering of a java userexit implementation the symbolic name of a userexit must be mapped to formula numbers used in e.g. a pricing procedure or access type.



**Display View "formulas": Overview**

Dialog Structure: types, implementations, attributes, **formulas**, attributes

Application: CRM  
 Usage: PR  
 Userexit type: REQ

Formula	Userexit Name	docu
1	DIFFERENT_PAYER	f
2	PRICINGRELEVANT	f
3	FOREIGN_CURR	f
4	COST	f
7	DOMESTIC	f
8	EXPORT	f
9	CASH_DISC_TAX	f

Position... Entry 1 of 44

In the formula screen the formula number gets assigned to a userexit name. All types of userexits must have a formula number assigned. The range reserved for customer formula number can be reviewed with transaction /SAPCND/UERNG and is depending on the application and usage (e.g. CRM and PR – Pricing). For userexits of scope B (one unique implementation) the formula number 0 must be used and therefore no customer range is defined. Don't change the range entries on your own.

In the sub screen all needed attributes (those used in the userexit implementation) must be mapped with fieldnames of the communication structure. Please mark the difference between the symbolic (internal) name of an attribute and the field name which is not visible to the userexit implementation.

**Display View "attributes": Overview**

Application: CRM  
Usage: PR  
Userexit type: REQ  
Formula Number: 2

Relevant Attr. Name	Field name
PRICING_INDICATOR	PRC_INDICATOR

Position... Entry 1 of 1

After registration and assignment the userexit formula must be uploaded before it can be assigned to any pricing procedure or other configuration. As the configuration is buffered for one day (default setting) the changes will only become immediately active with a restart of the VMC or the application server. While testing different configuration in a test or development system also the function module `IPC_DET_CLEAR_CUST_BUFFER` can be executed.

## 4 AVAILABLE EXITS AND APIS

This chapter focuses on the available userexits APIs, their types and functions.

### 4.1 Logging capabilities

For customer pricing userexits there is an easy way of including fast logging. The `com.sap.spe.base.logging.UserexitLogger` class implements two methods for logging debug messages or error messages. The logging is quite fast and logging is only done if the appropriate log level is reached which can be defined on run-time. For additional information about setting log levels and viewing the log files see chapter 5.3.

	ZSpecialRoundingValueFormula (shorten)
1	<code>package your.company.pricing.userexits;</code>
2	
3	<code>import com.sap.spe.base.logging.UserexitLogger;</code>
4	<code>[...]</code>
5	<code>public class ZSpecialRoundingValueFormula extends ValueFormulaAdapter {</code>
6	
7	<code>    private static UserexitLogger userexitlogger =</code>
8	<code>        new UserexitLogger(ZSpecialRoundingValueFormula.class);</code>
9	
10	<code>    public BigDecimal overwriteConditionValue(IPricingItemUserExit item,</code>
11	<code>        IPricingConditionUserExit condition) {</code>
12	<code>[...]</code>
13	<code>        userexitlogger.writeLogDebug("old cond value: "</code>
14	<code>            + val.getValueAsString());</code>
15	<code>[...]</code>
16	<code>    }</code>
17	<code>}</code>

Line 8: Create a static instance of the `UserexitLogger` class. As constructor parameter pass the actual class.

Line 13: Use `writeLogDebug(String s)` or `writeLogError(String s)` to log the string `s` into the log.

### 4.2 Userexit Types

This section provides a detailed description illustrated via a code example showing the APIs with their corresponding parameters, and a business use case example in the Javadoc part of the code.

The userexits types are categorized under three scopes:

- A - number-dependent: This type of userexits can have multiple active implementations and requires formula numbers to decide when to call this userexits implementation. This type of userexits is called according to the assignment of the formula number in the customizing, for example in the customizing of the pricing procedure. This is the most commonly used type of userexit
- B - one unique implementation: This type of userexits can have only one unique implementation. The userexit with the formula number 0 assigned is called.
- C - multiple active implementations: The userexits of this type are always called when a number is assigned to it. In case multiple active implementations of this type of userexits are required, assign dummy formula numbers to decide the sequence in which the userexits implementations should be called.

#### 4.2.1 Number Dependent Userexits

##### 4.2.1.1 Requirement (REQ)

This kind of userexits are used during condition finding on pricing procedure step/counter level and on condition access step level. They must be assigned, in the customizing, to the userexits type REQ – Requirement.

The userexits class must be inherited from `RequirementAdapter` and overwrite the method `checkRequirement`. If this method returns false, the current access is not made.

	ZSpecialRequirement
1	package your.company.pricing.userexits;
2	
3	import com.sap.spe.condmgnt.customizing.IAccess;
4	import com.sap.spe.condmgnt.customizing.IStep;
5	import com.sap.spe.condmgnt.finding.userexit.IConditionFindingManagerUserExit;
6	import com.sap.spe.condmgnt.finding.userexit.RequirementAdapter;
7	
8	public class ZSpecialRequirement extends RequirementAdapter {
9	/**
10	* Purpose: This is an example of a pricing requirement. This requirement is
11	* met if the document's item category is relevant for pricing and no
12	* previous condition in the pricing procedure has set the condition
13	* exclusion flag. This requirement can be assigned to condition types in
14	* the pricing procedure to avoid unnecessary accesses to the database when
15	* an item is not relevant for pricing or a condition exclusion indicator
16	* has been set.
17	*
18	* Example: A sales order is placed in the system. Some of the items in the
19	* order will be free to the customer and the customer service
20	* representative indicates this with the item category TANN. In the
21	* customizing, item category TANN has been configured as not relevant for
22	* pricing. Within the pricing procedure, the user assigns this requirement
23	* to all condition types. Using this requirement, the system does not
24	* access any pricing condition records for the free line item. In addition
25	* to offering free items, some of the prices for products in the sales
26	* order are defined as net prices. When a net price is found, no subsequent
27	* discounts or surcharges should be assigned to the item. This pricing
28	* requirement also ensures that further condition records are not accessed
29	* when a net price has already been found for the item (condition exclusion
30	* has been set).
31	*/
32	public boolean checkRequirement(IConditionFindingManagerUserExit item,
33	IStep step, IAccess access) {
34	String pricingIndicator = item.getAttributeValue("PRICING_INDICATOR");
35	return pricingIndicator.equals("X");
36	}
37	}

Line 8: Inherit from the API class `RequirementAdapter`.

Line 32: Overwrite the implementation of the `checkRequirement` method.

Line 34: Retrieve an attribute value to be used for the check. The attribute name is the symbolic name assigned to the userexit implementation, not the communication structure field.

Line 35: Return the check result, 'true' to make the access, 'false' to avoid the access.

#### 4.2.1.2 Condition Base Formula (BAS)

The condition base formula can be used to change the automatically calculated base value of a condition. This kind of userexits can be assigned, in the customizing, to the userexits type BAS – Condition base formula.

This userexit is called after the calculation of the condition base value for each pricing condition. The userexit class must be inherited from the `BaseFormulaAdapter` and overwrite the method `overwriteConditionBase`. The `overwriteConditionBase` method has the parameters `pricingItem` and `pricingCondition` which represents the item and the current condition.

If this method returns a null object reference, pricing will keep the automatically called base value.

	ZSpecialBaseFormula
1	package your.company.pricing.userexits;
2	
3	import java.math.BigDecimal;
4	
5	import com.sap.spe.pricing.transactiondata.userexit.BaseFormulaAdapter;
6	import com.sap.spe.pricing.transactiondata.userexit.IPricingConditionUserExit;
7	import com.sap.spe.pricing.transactiondata.userexit.IPricingItemUserExit;
8	
9	public class ZSpecialBaseFormula extends BaseFormulaAdapter {
10	

```

11      /**
12       * Purpose: This is an example of a condition base value formula. A
13       * condition base value formula can be used to influence the basis the
14       * system uses when computing a pricing value. A condition base formula is
15       * assigned to a step (line) in the pricing procedure.
16       *
17       * This example formula is used to convert the basis to an integer number.
18       * For example, a basis of 300.153 would be converted to 300. This formula
19       * can be used to compute pallet discounts.
20       *
21       * Example: A company sells their products in cases. Each of their materials
22       * has a conversion factor to pallets. When an order is placed by a
23       * customer, the user would like the system to calculate the number of full
24       * pallets for each line and to offer a 5 USD discount per full pallet
25       * ordered. The user sets up a discount condition type in the pricing
26       * procedure and assigns this condition base value formula to it. Within the
27       * condition records for this condition type, the user maintains the 5 USD
28       * per pallet discount rate. If an order line item is placed that contains
29       * 5.5 pallets, the system will adjust the base value to 5 and compute a
30       * discount of 25 USD for the sales line item.
31       *
32       */
33
34      public BigDecimal overwriteConditionBase(IPricingItemUserExit pricingItem,
35          IPricingConditionUserExit pricingCondition) {
36
37          return pricingCondition.getConditionBase()
38              .getValue().setScale(0, BigDecimal.ROUND_FLOOR);
39      }
40  }

```

Line 9: Inherit from the API class `BaseFormulaAdapter`.

Line 34: Overwrite the implementation of the `overwriteConditionBase` method.

Line 37: Returns the new value for the condition base.

### 4.2.1.3 Scale Base Formula (SCL)

This kind of userexit can be used to replace the automatically determined scale base. This kind of userexits must be assigned, in the customizing, to the userexits type SCL – Scale Base Formula.

This userexit is called after the calculation of the condition scale base value for a pricing condition. The userexit class must be inherited from the class `ScaleBaseFormulaAdapter` and overwrite at least `overwriteScaleBase`. If group condition processing for that condition type is enabled, also the overwriting of method `overwriteGroupScaleBase` is possible. Both methods can return `null` to indicate that the original value shall not be changed.

#### ZSpecialScaleBaseFormula

```

1  package your.company.pricing.userexits;
2
3  import java.math.BigDecimal;
4
5  import com.sap.spe.pricing.transactiondata.userexit.IGroupConditionUserExit;
6  import com.sap.spe.pricing.transactiondata.userexit.IPricingConditionUserExit;
7  import com.sap.spe.pricing.transactiondata.userexit.IPricingDocumentUserExit;
8  import com.sap.spe.pricing.transactiondata.userexit.IPricingItemUserExit;
9  import com.sap.spe.pricing.transactiondata.userexit.ScaleBaseFormulaAdapter;
10
11  public class ZSpecialScaleBaseFormula extends ScaleBaseFormulaAdapter {
12
13      /**
14       * Purpose: This is an example of a scale basis formula. A scale basis
15       * formula is assigned to a condition type in the customizing and alters the
16       * value that the system uses to read the scales in a condition record. This
17       * formula sets the integer number part of the value to zero. For example, the
18       * value 203.559 would be changed to 0.559.
19       *
20       * Example: A company sells their products in cases. Each of their materials
21       * has a conversion factor to pallets. When an order is placed by a
22       * customer, the user would like the system to add up the quantities across
23       * items and compute the number of full pallets. If the customer does not
24       * order in full pallets, the user would like to charge a fixed surcharge of
25       * 20 USD. The user sets up a condition type in the pricing procedure. In
26       * customizing for this condition type , this scale base formula is assigned

```

```

27      * as well as the group condition flag so that the quantities across order
28      * items can be considered. Within the condition records for this condition
29      * type, the user maintains a rate of "from 0.001 PAL" a fixed charge of 20
30      * USD. If an order is placed, for example, that is equal to 10.35 pallets,
31      * this formula will alter the value to 0.35 and then read the condition
32      * record scale. A surcharge of 20 USD would then be applied to the overall
33      * sales order.
34      *
35      */
36      public BigDecimal overwriteScaleBase(IPricingItemUserExit pricingItem,
37          IPricingConditionUserExit pricingCondition,
38          IGroupConditionUserExit groupCondition) {
39
40          BigDecimal roundedScaleBaseValue = pricingCondition.getConditionScale()
41              .getValue().setScale(0, BigDecimal.ROUND_FLOOR);
42          return pricingCondition.getConditionScale().getValue().subtract(
43              roundedScaleBaseValue);
44      }
45
46      public BigDecimal overwriteGroupScaleBase(
47          IPricingDocumentUserExit pricingDocument,
48          IGroupConditionUserExit groupCondition) {
49
50          BigDecimal roundedScaleBaseValue = groupCondition.getConditionScale()
51              .getValue().setScale(0, BigDecimal.ROUND_FLOOR);
52          return groupCondition.getConditionScale().getValue().subtract(
53              roundedScaleBaseValue);
54      }
55
56      }
57  }

```

Line 11: Inherit from the API class `ScaleBaseFormulaAdapter`.

Line 36: Overwrite the implementation of the `overwriteScaleBase` method.

Line 42: Return the changed scale base value.

Line 47: Overwrite the implementation of the `overwriteGroupScaleBase` method.

Line 53: Return the changed scale base value.

#### 4.2.1.4 Condition Value Formula (VAL)

This userexit can be used to replace the automatically determined condition value. This kind of userexits must be assigned, in the customizing, to the userexits type VAL – Condition Value Formula.

This userexit is called after the calculation of the condition value for each pricing condition. The userexit class must be inherited from the class `ValueFormulaAdapter` and overwrites at least `overwriteConditionValue`. If group condition processing for that condition type is enabled, also the overwriting of method `overwriteGroupConditionValue` is possible. Both methods can return `null` to indicate that the original value shall not be changed.

##### ZSpecialRoundingValueFormula

```

1 package your.company.pricing.userexits;
2
3 import java.math.BigDecimal;
4
5 import com.sap.spe.base.logging.UserexitLogger;
6 import com.sap.spe.conversion.ICurrencyValue;
7 import com.sap.spe.pricing.transactiondata.userexit.IGroupConditionUserExit;
8 import com.sap.spe.pricing.transactiondata.userexit.IPricingConditionUserExit;
9 import com.sap.spe.pricing.transactiondata.userexit.IPricingDocumentUserExit;
10 import com.sap.spe.pricing.transactiondata.userexit.IPricingItemUserExit;
11 import com.sap.spe.pricing.transactiondata.userexit.ValueFormulaAdapter;
12
13 public class ZSpecialRoundingValueFormula extends ValueFormulaAdapter {
14
15     private static UserexitLogger userexitlogger =
16         new UserexitLogger(ZSpecialRoundingValueFormula.class);
17
18     public BigDecimal overwriteConditionValue(IPricingItemUserExit item,
19         IPricingConditionUserExit condition) {
20         BigDecimal result;
21
22         ICurrencyValue val = condition.getConditionValue();

```

```

23         userexitlogger.writeLogDebug("old cond value: "
24             + val.getValueAsString());
25
26         result = val.getValue().setScale(0, BigDecimal.ROUND_HALF_UP);
27
28         BigDecimal qnt = item.getProductQuantity().getValue();
29         qnt = qnt.divide(new BigDecimal("100"), 2, BigDecimal.ROUND_HALF_UP);
30
31         userexitlogger.writeLogDebug("new cond value: " + result.subtract(qnt));
32
33         return result.subtract(qnt);
34     }
35
36     public BigDecimal overwriteGroupConditionValue(
37         IPricingDocumentUserExit item, IGroupConditionUserExit condition) {
38         // do nothing
39         return null;
40     }
41 }

```

Line 13: Inherit from the API class `ValueFormulaAdapter`.

Line 18: Overwrite the implementation of the `overwriteConditionValue` method.

Line 33: Return the changed condition value.

Line 36: Overwrite the implementation of the `overwriteGroupConditionValue` method.

Line 39: Return `null` to keep the automatically calculated value.

#### 4.2.1.5 Copy Formula (CPY)

While copying a document the pricing condition can be fixed or other changes can take place if needed. This kind of userexits must be assigned, in the customizing, to the userexits type CPY – Copy Formula.

This userexit is called during the copy process. The userexit class must be inherited from the class `PricingCopyFormulaAdapter` and overwrite the method `pricingCopy`. The parameters `pricingDocument`, `pricingItem` and `pricingCondition` are references to the target document, item and condition. For each condition this method is called. The pricing type describes what should happen with the pricing result when a new pricing takes place. The parameter `copyType` is a reference to the used customizing for that copy process and the `sourceSalesQuantity` contains the old quantity of the source item.

##### ZSpecialCopyFormula

```

1 package your.company.pricing.userexits;
2
3 import com.sap.spe.conversion.IQuantityValue;
4 import com.sap.spe.pricing.customizing.ICopyType;
5 import com.sap.spe.pricing.customizing.IPricingType;
6 import com.sap.spe.pricing.customizing.PricingCustomizingConstants;
7 import com.sap.spe.pricing.transactiondata.PricingTransactiondataConstants;
8 import com.sap.spe.pricing.transactiondata.userexit.IPricingConditionUserExit;
9 import com.sap.spe.pricing.transactiondata.userexit.IPricingDocumentUserExit;
10 import com.sap.spe.pricing.transactiondata.userexit.IPricingItemUserExit;
11 import com.sap.spe.pricing.transactiondata.userexit.PricingCopyFormulaAdapter;
12
13 public class ZSpecialCopyFormula extends PricingCopyFormulaAdapter {
14
15     /**
16      * Purpose: This is an example of a pricing copy formula. A pricing copy
17      * formula is assigned to a condition type in the customizing and alters
18      * the value that the system uses to read the scales in a condition
19      * record.
20      *
21      *
22      * Example: A customer sends back one of the ordered items of an already
23      * paid-for sales order. during the copy process the conditions should be
24      * fixed and their sign should be inverted. Except for the freight costs.
25      *
26      */
27     public void pricingCopy(IPricingDocumentUserExit pricingDocument,
28         IPricingItemUserExit pricingItem,
29         IPricingConditionUserExit pricingCondition,
30         IPricingType pricingType, ICopyType copyType,
31         IQuantityValue sourceSalesQuantity) {

```



```

32
33         // set condition control (KSTEU) to 'H'
34         pricingCondition.setConditionControl(
35             PricingCustomizingConstants.Control.VALUE_FIXED_FOR_COST_PRICE);
36
37         // invert condition value
38         pricingCondition
39             .setConditionValue(pricingCondition.getConditionValue()
40                 .getValue().multiply(
41                     PricingTransactiondataConstants.MINUS_ONE));
42
43         // invert condition base for percentage conditions that it looks
44         // consistent on the condition screen
45         if (PricingCustomizingConstants.CalculationType
46             .isFixedAmountOrPercentage(pricingCondition
47                 .getCalculationType())) {
48             pricingCondition.setConditionBaseValue(pricingCondition
49                 .getConditionBase().getValue().multiply(
50                     PricingTransactiondataConstants.MINUS_ONE));
51         }
52     }
53 }
54
55
56
57
58

```

Line 13: Inherit from the API class `PricingCopyFormulaAdapter`.

Line 15: Overwrite the implementation of the `pricingCopy` method.

#### 4.2.1.6 Group Key Formula (GRP)

This seldom used userexit influences the grouping of group conditions (conditions that will be processed together over more than one item). This kind of userexits must be assigned, in the customizing, to the userexits type GRP – Group Key Formula.

This userexit is called when the key of a group condition is determined. The userexit class must be inherited from the class `GroupKeyFormulaAdapter` and overwrite the method `setGroupKey`. The method determines from the different passed object references a String which will be used for the grouping rule of group conditions. Different string values for two conditions will result that the two conditions will never form one group.

##### ZSpecialGroupKeyFormula

```

1 package your.company.pricing.userexits;
2
3 import com.sap.spe.pricing.transactiondata.userexit.GroupKeyFormulaAdapter;
4 import com.sap.spe.pricing.transactiondata.userexit.IGroupConditionUserExit;
5 import com.sap.spe.pricing.transactiondata.userexit.IPricingConditionUserExit;
6 import com.sap.spe.pricing.transactiondata.userexit.IPricingDocumentUserExit;
7 import com.sap.spe.pricing.transactiondata.userexit.IPricingItemUserExit;
8
9 public class ZSpecialGroupKeyFormula extends GroupKeyFormulaAdapter {
10     /**
11      * Purpose: This is an example of a group key formula. A group key formula
12      * can be used to influence the basis the system uses when reading the scale
13      * of a group condition. The formula is assigned to a group condition type
14      * in customizing.
15      *
16      * This Formula adds up the quantities / values of all of the line items in
17      * the sales document independent of which condition types have been
18      * applied.
19      *
20      * Example: A company defines their prices with scales based on weight. When
21      * a sales order line item is priced, the user would like the system to read
22      * the scale with not just the weight of the current line item, but the
23      * combined weight of all items in the sales document. To accomplish this,
24      * the user defines their price condition types as group conditions and
25      * assigns this group key formula to them in customizing.
26      *
27      */
28     public String setGroupKey(IPricingDocumentUserExit document,
29                             IPricingItemUserExit item, IPricingConditionUserExit condition,
30                             IGroupConditionUserExit groupCondition) {

```

```

31         groupCondition.setConditionTypeName("++++");
32         return "002";
33     }
34 }

```

Line 9: Inherit from the API class `GroupKeyFormulaAdapter`.

Line 28: Overwrite the implementation of the `setGroupKey` method.

Line 31: Set all group conditions using this formula the condition type name to “++++” to avoid a splitting because of different condition type names.

Line 32: Set the conditions group key to the group key “002”.

## 4.2.2 Userexits with Multiple Active Implementations

### 4.2.2.1 Pricing Init (PRI)

In previous releases this userexit was called `CRMDocumentStandardExit`. There it was mainly used to pass header attributes to be used in the method `initializeDocument`. As of 5.0 such attributes can be customized as described in chapter 3.1. Pricing Init userexit can now only be used to set the rounding unit to the smallest unit of the document currency. This kind of userexit must be assigned, in the customizing, to the userexit type PRI – Pricing Init.

This userexit is called when a new pricing document is created. The userexit class must be inherited from the class `PricingInitFormulaAdapter` and must overwrite the method `initializeDocument`. This method has the parameter `documentUserExitAccess` which represents the pricing document.

#### ZPricingInit

```

1 package your.company.pricing.userexits;
2
3 import com.sap.spe.document.userexit.IDocumentUserExitAccess;
4 import com.sap.spe.document.userexit.PricingInitFormulaAdapter;
5
6 public class ZPricingInit extends PricingInitFormulaAdapter {
7     /**
8      * Example: In case the document currency is Swiss Frank,
9      * set the rounding unit to 5 (so called "rappen rounding")
10    */
11    public void initializeDocument(
12        IDocumentUserExitAccess documentUserExitAccess) {
13
14        if (documentUserExitAccess.getDocumentCurrency()
15            .getUnitName().equals("CHF")) {
16            documentUserExitAccess.setUnitToBeRoundedTo(5);
17        }
18    }
19 }

```

Line 6: Inherit from the API class `PricingInitFormulaAdapter`.

Line 11: Overwrite the implementation of the `initializeDocument` method.

Line 16: Set the rounding unit of the CHF currency to 0.05 (there always a factor of 100).

### 4.2.2.2 Pricing Prepare (PRP)

In previous releases this userexits was called `CRMItemStandardExit`. Pricing Prepare userexits can be used to add header and/or item attributes to be used during the pricing process. Such attributes can be customized as described in chapter 3.1. This kind of userexits must be assigned, in the customizing, to the userexits type PRP – Pricing Prepare.

Pricing Prepare userexit is called when creating a new pricing item and when a new pricing takes place. The userexit class must be inherited from the class `PricingPrepareFormulaAdapter` and must overwrite the method `addAttributeBindings`. This method has the parameter `itemUserExitAccess` which represents the pricing item.

#### ZPricingPrepare

```

1 package your.company.pricing.userexits;
2
3 import com.sap.spe.document.userexit.IItemUserExitAccess;
4 import com.sap.spe.document.userexit.PricingPrepareFormulaAdapter;
5

```

```

6 public class ZPricingPrepare extends PricingPrepareFormulaAdapter {
7
8     public void addAttributeBindings(IItemUserExitAccess itemUserExitAccess) {
9         // bound attribute ZLAND to value DE
10        itemUserExitAccess.addAttributeBinding("ZLAND", "DE");
11    }
12 }

```

Line 6: Inherit from the API class `PricingPrepareFormulaAdapter`.

Line 8: Overwrite the implementation of the `addAttributeBindings` method.

Line 10: Set the attribute ZLAND to the value “DE”. ZLAND is a symbolic (internal) attribute name, not to be mixed up with a field from the communication structure (field catalogue).

## 4.2.3 Userexits with one unique implementation

### 4.2.3.1 Item Calculation Begin Formula (CAB)

This seldom used userexit is available to change the document and item if necessary before the item pricing takes place. It corresponds to the R/3 userexit `userexit_xkomv_bewerten_init` (include:RV61AFZB). This kind of userexits must be assigned, in the customizing, to the userexits type CAB – Item Calculation Begin formula.

The userexit class must be inherited from `PricingItemCalculateBeginFormulaAdapter`. It gets a reference to the pricing document (`prDocument`) and the item (`prItem`).

```

1 package your.company.pricing.userexits;
2
3 import com.sap.spe.pricing.transactiondata.userexit.IPricingDocumentUserExit;
4 import com.sap.spe.pricing.transactiondata.userexit.IPricingItemUserExit;
5 import com.sap.spe.pricing.transactiondata.userexit.PricingItemCalculateBeginFormulaAdapter;
6
7 public class ZSpecialCalculationBeginFormula extends PricingItemCalculateBeginFormulaAdapter
8 {
9
10    public void calculationBegin(IPricingDocumentUserExit prDocument,
11                               IPricingItemUserExit prItem) {
12
13        [your coding]
14
15    }
16 }

```

Line 7: Inherit from the API class `PricingItemCalculateBeginFormulaAdapter`.

Line 10: Overwrite the implementation of the `calculationBegin` method.

### 4.2.3.2 Item Calculation End Formula (CAE)

This seldom used userexit is available to change the document and item if necessary after the item pricing took place. It corresponds to the R/3 userexit `userexit_xkomv_bewerten_end` (include: RV61AFZB). This kind of userexits must be assigned, in the customizing, to the userexits type CAE – Item Calculation End formula. The userexit class must be inherited from `PricingItemCalculateEndFormulaAdapter`. It gets a reference to the pricing document (`prDocument`) and the item (`prItem`).

```

1 package your.company.pricing.userexits;
2
3 import com.sap.spe.pricing.transactiondata.userexit.IPricingDocumentUserExit;
4 import com.sap.spe.pricing.transactiondata.userexit.IPricingItemUserExit;
5 import com.sap.spe.pricing.transactiondata.userexit.PricingItemCalculateEndFormulaAdapter;
6
7 public class ZSpecialCalculationEndFormula extends PricingItemCalculateEndFormulaAdapter {
8
9     private int stepNumber, counter;
10
11    public void calculationEnd(IPricingDocumentUserExit prDocument,
12                             IPricingItemUserExit prItem) {
13
14        [your coding]

```

```

15     }
16 }
17 }

```

Line 7: Inherit from the API class `PricingItemCalculateEndFormulaAdapter`.

Line 11: Overwrite the implementation of the `calculationEnd` method.

### 4.2.3.3 Condition Init Formula (CNI)

After initializing a pricing condition, the pricing condition can be changed using this userexit which is called whenever an internal condition (a transactional object or business entity) is created. This kind of userexits must be assigned, in the customizing, to the userexits type CNI – Condition Init Formula.

The userexit class must be inherited from `PricingConditionInitFormulaAdapter` and must overwrite the method `init`. It allows changing the newly create condition (parameter `prCondition`).

#### ZSpecialConditionInitFormula

```

1 package your.company.pricing.userexits;
2
3 import java.math.BigDecimal;
4
5 import com.sap.spe.pricing.transactiondata.userexit.IPricingConditionUserExit;
6 import com.sap.spe.pricing.transactiondata.userexit.IPricingDocumentUserExit;
7 import com.sap.spe.pricing.transactiondata.userexit.IPricingItemUserExit;
8 import com.sap.spe.pricing.transactiondata.userexit.PricingConditionInitFormulaAdapter;
9
10 public class ZSpecialConditionInitFormula extends PricingConditionInitFormulaAdapter {
11
12     public void init(IPricingDocumentUserExit prDocument, IPricingItemUserExit prItem,
13                     IPricingConditionUserExit prCondition) {
14
15 [your coding]
16
17     }
18 }

```

Line 10: Inherit from the API class `PricingConditionInitFormulaAdapter`.

Line 12: Overwrite the `init` method.

### 4.2.3.4 Item Init Formula (ITI)

After the pricing Item is initialized, the pricing item can be changed using this userexit which is called when a new pricing item is created. This kind of userexits must be assigned, in the customizing, to the userexits type ITI – Item Init Formula.

The userexit class must be inherited from the class `PricingItemInitFormulaAdapter` and implement the method `init`. A reference to the document and to the newly created item is passed.

#### ZSpecialItemInitFomula

```

1 package your.company.pricing.userexits;
2
3 import com.sap.spe.pricing.transactiondata.userexit.IPricingDocumentUserExit;
4 import com.sap.spe.pricing.transactiondata.userexit.IPricingItemUserExit;
5 import com.sap.spe.pricing.transactiondata.userexit.PricingItemInitFormulaAdapter;
6
7 public class ZSpecialItemInitFomula extends PricingItemInitFormulaAdapter {
8
9     public void init(IPricingDocumentUserExit prDocument, IPricingItemUserExit prItem) {
10
11 [your coding]
12
13     }
14 }

```

Line 7: Inherit from the API class `PricingItemInitFormulaAdapter`.

Line 9: Overwrite the implementation of the `init` method.

### 4.2.3.5 Document Init Formula (DOI)

After initializing a pricing document, the document can be changed using this userexit which is called when a new pricing document is created. This kind of userexits must be assigned, in the customizing, to the userexits type DOI – Document Init Formula.

The userexit class must be inherited from the class `PricingDocumentInitFormulaAdapter` and implement the method `init`. A reference to the newly created document is passed

	ZSpecialDocumentInitFormula
1	<code>package your.company.pricing.userexits;</code>
2	
3	<code>import com.sap.spe.pricing.transactiondata.userexit.IPricingDocumentUserExit;</code>
4	<code>import com.sap.spe.pricing.transactiondata.userexit.PricingDocumentInitFormulaAdapter;</code>
5	
6	<code>public class ZSpecialDocumentInitFormula extends PricingDocumentInitFormulaAdapter {</code>
7	
8	<code>    public void init(IPricingDocumentUserExit prDocument) {</code>
9	
10	<code>[your coding]</code>
11	
12	<code>    }</code>
13	<code>}</code>

Line 6: Inherit from the API class `DocumentInitFormula`.

Line 8: Overwrite the implementation of the `init` method.

#### 4.2.3.6 Configuration Formula (CFG)

This seldom used userexit will be called when the process of product configuration creates subitems. This kind of userexits must be assigned, in the customizing, to the userexits type CFG – Configuration Formula (called for sub items created by SCE).

The userexit class must be inherited from `SPCSubItemCreatedByConfigurationFormulaAdapter`. For each subitem the method `isRelevantForPricing` is called and a reference to the newly created subitem and the configuration instance is passed in.

	ZSpecialConfigurationFormula
1	<code>package your.company.pricing.userexits;</code>
2	
3	<code>import com.sap.spc.document.userexit.ISPCItemUserExitAccess;</code>
4	<code>import com.sap.sce.front.base.Instance;</code>
5	<code>import com.sap.spc.document.userexit.SPCSubItemCreatedByConfigurationFormulaAdapter;</code>
6	
7	<code>public class ZspecialConfigurationFormula</code>
8	<code>    extends SPCSubItemCreatedByConfigurationFormulaAdapter {</code>
9	
10	<code>    /**</code>
11	<code>     * All configuration sub items are pricing relevant</code>
12	<code>     */</code>
13	<code>    public boolean isRelevantForPricing(ISPCItemUserExitAccess subItem,</code>
14	<code>        Instance instance) {</code>
15	<code>        return true;</code>
16	<code>    }</code>
17	<code>}</code>

Line 8: Inherit from the API class `SPCSubItemCreatedByConfigurationFormulaAdapter`.

Line 10: Overwrite the `isRelevantForPricing` method.

Line 15: Set the configuration subItem to pricing relevant by returning true.

## 5 DEBUGGING

### 5.1 Set VMC in Debug Mode

The logical application server is usually hosted on different servers and load balanced via the message server. To debug the VMC the user has to know on which host his work process run. In the SAPGUI menu System -> Status... the Server name (first part) is displayed. On the next page (3<sup>rd</sup> button) also the corresponding IP address is given. Switching between hosts of one server can be done via transaction sm51.

Run transaction sm52 to display the available VMCs. This window will later show the servers running virtual machines and also the debug state and port of each one.

A VMC work process switches into debug mode when a program flow from ABAP to Java (RFC function module) is passed and ABAP was in debug mode (with /h). To help debugging each call of such a pricing RFC module a conditional breakpoint triggered. To active the breakpoint the user parameter PRC\_RFC has to be set to X (transaction su3).

While running the transaction (e.g. order processing) each call to pricing will hold and with a step-into debugging into the RFC module (F5) a VMC debug port will be activated. This VMC will stay attached as long the ABAP transaction runs.

### 5.2 Attach Eclipse Java Debugger

After a VMC work process has an open debug port, which can be viewed with transaction sm52, the eclipse debugger can be attached to it.

Because attaching the debugger to the VMC will run directly the code, first interesting java breakpoint may be set. Attaching eclipse works then as follow:

1. Open menu Run->Debug...
2. Create a new Remote Java Application Configuration
3. Enter as host the server where the VMC work process is running on
4. Enter the debug port as the one VMC work process debug port shown in transaction sm52
5. Click on Debug

The debugging view will normally popup automatically and a successful attachment will show all working threads of that VMC work process.

### 5.3 View VMC Log Files

First the logging must be switched on for a specific java class or package. Transaction sm53 starts the VMC System Administration. On the left side the branch Log Administration the node Log Configuration is shown. The displayed view allows the entering of a Location and Category. In case of the UserexitLogger the Location is the full name of the userexit class name, e.g. `your.company.pricing.userexits.ZSpecialRoundingValueFormula`. The Severity is depending on which UserexitLogger method is used. ALL includes logging independent of their severity levels; DEBUG contains also logging done for ERROR severity.

With Copy button the logging level is directly set for all VMC work processes on that application server. The Reset button set the default for the selected classes or packages.

To include all logging done in the userexits the logging can also be switched on for packages, e.g. `your.company.pricing`. If entered in the Location or Category field, the severity level is set for all classes of that package and all classes of sub packages.

Now the log files are activated and traces depending on the settings.

The log can now be checked with the same transaction sm53. The log viewer is started by clicking the navigation menu entry Log Display onto Log Administration. The root node in log selection (lower left box) has to be switched on. On the right side all messages are shown and details in the lower right box. The log entries can also be filtered by different criteria such as Severity, Log Name (which is the class name or package) or User.

## 6 UPGRADE GUIDE

This chapter is intended to help upgrading IPC 3.0 and IPC 4.0 pricing userexits to the new AP 7.00 architecture. It refers to all chapters before. As an example, an old IPC 4.0 PricingUserExits.java file is taken and converted.

This is an excerpt of the pricing userexit.

### (IPC 4.0) PricingUserExits.java

```

1 package userexits;
2
3 [...]
4 import java.math.BigDecimal;
5
6 public class PricingUserExits
7     implements IPricingUserExits
8 {
9
10 [...]
11     public boolean checkRequirement(IConditionFindingManagerUserExit item,
12         IStep step, int reqNo)
13         throws SXERuntimeException
14     {
15         switch(reqNo)
16         {
17             case 700: {
18                 String zland = null;
19                 try {
20                     zland = item.getHeaderAttributeValue("ZLAND").getValue();
21                 } catch(Exception ex) {
22                     zland = new String();
23                 }
24                 return zland != null && !zland.equals("US") && !zland.equals("CA");
25             }
26             case 701: {
27                 String itemCategory = null;
28                 boolean priceRelevant = false;
29                 try {
30                     itemCategory = item.getItemAttributeValue("ZITMCAT").getValue();
31                     String priceIndicator = item.getItemAttributeValue(
32                         AttributeNames.pricingIndicator).getValue();
33                     priceRelevant = priceIndicator != null && priceIndicator.equals("X");
34                 }
35                 catch(Exception ex) {
36                     itemCategory = new String();
37                 }
38                 return priceRelevant && itemCategory != null &&
39                     (itemCategory.equals("TANN") || itemCategory.equals("ZTNN"));
40             }
41             [...]
42             default: {
43                 throw new SXERuntimeException("Error in user exit: no condition " +
44                     "found for requirement number" + reqNo + ".");
45             }
46         }
47     }
48
49     public BigDecimal overwriteConditionBase(IPricingItemUserExit prItem,
50         ILastPrice lastPrice, IPricingConditionUserExit prCondition, int baseFormNo)
51         throws SXERuntimeException
52     {
53         switch(baseFormNo)
54         {
55             case 710:
56                 return prItem.getNetPrice();
57             default:
58                 throw new SXERuntimeException("Error in user exit: no coding " +
59                     "found for base formula number" + baseFormNo + ".");
60         }
61     }
62
63     public BigDecimal overwriteConditionValue(IPricingItemUserExit prItem,

```



```

64      ILastPrice lastPrice, IPricingConditionUserExit prCondition, int valueFormNo)
65      {
66          switch(valueFormNo)
67          {
68              case 720:
69                  try {
70                      boolean isCanada = prItem.getHeaderAttributeValue(
71                          AttributeNames.taxDepartCty).getValue().startsWith("CA");
72                      if(!isCanada)
73                          return new BigDecimal(0.0d);
74                  }
75                  catch(UnsuppliedAttributeException e) {
76                      return new BigDecimal(0.0D);
77                  }
78              default:
79                  throw new SXERuntimeException("Error in user exit: no coding "+
80                      "found for value formula number" + valueFormNo + ".");
81          }
82      }
83
84      [...]
85
86      public String[] determineRelevantAttributesForRequirement(
87          boolean headerAttributes, int reqNo)
88      {
89          String relevantAttributes[] = new String[0];
90          if(headerAttributes)
91              switch(reqNo)
92              {
93                  case 700:
94                      relevantAttributes = (new String[] { "ZLAND" });
95                      break;
96              }
97          else
98              switch(reqNo)
99              {
100                 case 701:
101                     relevantAttributes = (new String[] {
102                         "ZITMCAT", AttributeNames.pricingIndicator });
103                     break;
104                 }
105             return relevantAttributes;
106         }
107
108         public String[] determineRelevantAttributesForBaseFormula(
109             boolean headerAttributes, int baseFormulaNo)
110         {
111             return null;
112         }
113
114         public String[] determineRelevantAttributesForValueFormula(
115             boolean headerAttributes, int valueFormulaNo)
116         {
117             String relevantAttributes[] = new String[0];
118             if(headerAttributes)
119                 switch(valueFormulaNo)
120                 {
121                     case 720:
122                         relevantAttributes = (new String[] {
123                             AttributeNames.taxDepartCty });
124                         break;
125                 }
126             return relevantAttributes;
127         }
128
129         [...]
130
131         public int[] getRequirementNumbers()
132         {
133             return (new int[] { 700, 701 });
134         }
135
136         public int[] getConditionBaseFormulaNumbers()
137         {
138             return (new int[] { 710 });
139         }
140     }

```

```

141     public int[] getConditionValueFormulaNumbers()
142     {
143         return (new int[] { 720 });
144     }
145 }

```

The above pricing userexits shows a common customer implementation. Implemented are:

- 700. Requirement: Checks if the header (document) attribute ZLAND is neither US nor CA.
- 701. Requirement: Checks if the item attribute ZITMCAT is either “ZTAN” or “ZTNN” and the item attribute PRICING\_INDICATOR is “X”.
- 710. Base value formula: Sets the base and also the condition value to the item’s net price (net price calculated at the time the base value formula is called).
- 720. Condition value formula: Sets the condition value to 0 if the header (document) attribute TAX\_DEPART\_CTY is not CA.

All required attributes with the corresponding determineRelevantAttributes-methods and all implemented formula number are shown.

## 6.1 Preparation

As of AP700 each userexit formula is implemented in an own java class which must be registered in the system as described in chapter 3. Some preparation steps make it easier to implement and adopt the userexits.

- Setup the development environment as described in chapter 2 and read the explanations provided there.
- Find a descriptive name for the functionality achieved with each formula.

Formula	Name	Description
700 Req	ZLANDUSCA	Checks if ZLAND is neither “US” nor “CA”
701 Req	ZITMCAT	Checks if ZITMCAT is “ZTAN” or “ZTNN” and for active pricing indicator
710 Base	ZSETNETP	Sets the base of the condition to the item’s net price
720 Value	ZTAXDEPCA	Sets the value of the condition to 0 if the departure country is not Canada

- Collect for each formula the used header (document) and item attributes.

Formula	Name	H/I	Description
700 Req	ZLAND	Header	Country of the last depot
701 Req	ZITMCAT	Item	Item Category
701 Req	PRICING_INDICATOR	Item	Pricing Indicator
720 Value	TAX_DEPART_CTY	Header	Tax Departure Country

- Define a java package structure or a simple package name along with the class name for the implementation.

Formula	Name	Class
700 Req	ZLANDUSCA	your.company.pricing.req.Zland
701 Req	ZITMCAT	your.company.pricing.req.Zitmcat
710 Base	ZSETNETP	your.company.pricing.base.Zsetnetp
720 Value	ZTAXDEPCA	your.company.pricing.value.Ztaxdepca

## 6.2 Implementation

Next step is to create each userexit and adapt the coding to implement the adapter’s methods. For each kind of userexit an adapter must be inherited, more information is provided in chapter 4.

- Create new empty classes which inherit from the appropriate adapter class (depending on the type of the userexit).

- Override necessary methods and fill in the coding parts which were formally (and usually) in the switch-case of the old implementation.
- Some adaptations have to be made because parameters names and other references might have changed.

The four resulting userexits may look like these:

	<b>your.company.pricing.req.Zland</b>
1	package your.company.pricing.req;
2	
3	import com.sap.spe.base.logging.UserexitLogger;
4	import com.sap.spe.condmgnt.customizing.IAccess;
5	import com.sap.spe.condmgnt.customizing.IStep;
6	import com.sap.spe.condmgnt.finding.userexit.IConditionFindingManagerUserExit;
7	import com.sap.spe.condmgnt.finding.userexit.RequirementAdapter;
8	
9	public class Zland extends RequirementAdapter {
10	
11	private static UserexitLogger uellogger =
12	new UserexitLogger(Zland.class);
13	
14	public boolean checkRequirement(IConditionFindingManagerUserExit item,
15	IStep step, IAccess access)
16	{
17	String zland = item.getAttributeValue("ZLAND");
18	if (zland.equals("")) {
19	uellogger.writeLogError("ZLAND attribute missing");
20	return false;
21	} else {
22	return !zland.equals("US") && !zland.equals("CA");
23	}
24	}
25	}

	<b>your.company.pricing.req.Zitmcat</b>
1	package your.company.pricing.req;
2	
3	import com.sap.spe.base.logging.UserexitLogger;
4	import com.sap.spe.condmgnt.customizing.IAccess;
5	import com.sap.spe.condmgnt.customizing.IStep;
6	import com.sap.spe.condmgnt.finding.userexit.IConditionFindingManagerUserExit;
7	import com.sap.spe.condmgnt.finding.userexit.RequirementAdapter;
8	
9	public class Zitmcat extends RequirementAdapter {
10	
11	private static UserexitLogger uellogger =
12	new UserexitLogger(Zitmcat.class);
13	
14	public boolean checkRequirement(IConditionFindingManagerUserExit item,
15	IStep step, IAccess access)
16	{
17	String itemCategory = item.getAttributeValue("ZITMCAT");
18	String priceIndicator = item.getAttributeValue("PRICING_INDICATOR");
19	boolean priceRelevant = priceIndicator.equals("X");
20	
21	if (itemCategory.equals("")) {
22	uellogger.writeLogError("ZITMCAT attribute missing");
23	return false;
24	}
25	
26	return priceRelevant &&
27	(itemCategory.equals("TANN")    itemCategory.equals("ZTNN"));
28	}
29	}

	<b>your.company.pricing.base.Zsetnetp</b>
1	package your.company.pricing.base;
2	
3	import java.math.BigDecimal;
4	
5	import com.sap.spe.pricing.transactiondata.userexit.BaseFormulaAdapter;
6	import com.sap.spe.pricing.transactiondata.userexit.IPricingConditionUserExit;
7	import com.sap.spe.pricing.transactiondata.userexit.IPricingItemUserExit;

```

8
9 public class Zsetnetp extends BaseFormulaAdapter {
10
11     public BigDecimal overwriteConditionBase(IPricingItemUserExit pricingItem,
12         IPricingConditionUserExit pricingCondition) {
13         return pricingItem.getNetPrice().getValue();
14     }
15 }

```

#### your.company.pricing.value.Ztaxdepca

```

1 package your.company.pricing.value;
2
3 import java.math.BigDecimal;
4
5 import com.sap.spe.pricing.customizing.PricingCustomizingConstants;
6 import com.sap.spe.pricing.transactiondata.PricingTransactiondataConstants;
7 import com.sap.spe.pricing.transactiondata.userexit.IPricingConditionUserExit;
8 import com.sap.spe.pricing.transactiondata.userexit.IPricingItemUserExit;
9 import com.sap.spe.pricing.transactiondata.userexit.ValueFormulaAdapter;
10
11 public class Ztaxdepca extends ValueFormulaAdapter {
12
13     public BigDecimal overwriteConditionValue(IPricingItemUserExit pricingItem,
14         IPricingConditionUserExit pricingCondition) {
15         String taxdepcty = pricingItem.getAttributeValue("TAX_DEPART_CTY");
16         if (taxdepcty.equals("")) {
17             // INACTIVE_DUE_TO_ERROR = X
18             pricingCondition
19
20         .setInactive(PricingCustomizingConstants.InactiveFlag.INACTIVE_DUE_TO_ERROR);
21         }
22         boolean isCanada = taxdepcty.equals("CA");
23         if (!isCanada)
24             return PricingTransactiondataConstants.ZERO;
25         // keep the automatically calculated Condition Value
26         return null;
27     }
28 }

```

- Compile the java sources and create the jar package per the description in chapter 2 and chapter 2.6.

## 6.3 Register and Assign the Userexit

Use the following steps to register your classes.

- Start transaction /n/SAPCND/UEASS.
- Enter the right usage for your implementation (in this case PR – pricing).
- For the Requirement formula select the userexit type REQ and click on implementations.
- Create a new entry and enter the following details:
  - In the field “Userexit Name” type in the chosen userexit name (e.g. ZLANDUSCA)
  - In the field “Userexit Class” type in the fully qualified class name (e.g. your.company.pricing.req.zland)
  - In the field “Userexit Descr.” type in a meaningful description.
- Assign the attributes used in the formula implementation by
  - selecting the newly entered userexit and by going to implementations → attributes
  - and entering the required attributes along with a meaningful description (e.g. ZLAND Country of the last depot).
- Assign formula numbers to the userexit implementation. E.g. 700 for ZLANDUSCA.
- Map the relevant attributes to the catalogue field names. E.g. ZLAND with ZLAND.

## 6.4 Uploading and Testing

Now it's time to upload the compiled and packaged implementation into the system's DB. Use the steps described in chapter 2.7.

After the implementation has been uploaded and the customizing has been completed the userexits are now ready for use.

## 7 INTERFACES AND INCOMPATIBLE CHANGES

### 7.1 Interface Packages

This chapter describes the interfaces that can be used in 5.0 userexits. Each package that ends with “userexit” is an interface package that can be used for userexits implementations.

#### 7.1.1 Condition Finding Interfaces

`com.sap.spe.condmngt.finding.userexit:`

- `IConditionFindingManagerUserExit`: it provides information required for the search of condition records such as Attributes. This interface is relevant for implementing requirements.

#### 7.1.2 Pricing Interfaces

`com.sap.spe.pricing.transactiondata.userexit:` All the pricing related userexits APIs are part of this package

- `IPricingDocumentUserExit`: The pricing document contains accumulated pricing information of all pricing items and provides methods which are processing all pricing items. The methods of this interface can be used in the pricing userexits.
- `IPricingItemUserExit`: The pricing item contains accumulated pricing information of all pricing conditions and provides methods which are working on all pricing condition of this pricing item. The methods of this interface can be used in the pricing userexits.
- `IPricingConditionUserExit`: A pricing condition contains information copied from the pricing knowledge base (from the customizing of the condition type and the pricing procedure), from the condition record, and from the result of the calculation. The methods of this interface can be used in the pricing userexits.
- `ILastPriceUserExit`: Interface for last price object. The last price is determined for each pricing item and available in some pricing userexits. It corresponds to the pricing condition which holds the last price on the pricing item.

`com.sap.spe.pricing.customizing:`

- `PricingCustomizingConstants`: contains useful constants for the pricing condition properties such as calculation type ...etc.

`com.sap.spe.pricing.transactiondata:`

- `PricingTransactiondataConstants`: contains useful constants for often used `BigDecimal` values such as zero.

#### 7.1.3 Document (Sales/Purchase Order) Interfaces

`com.sap.spe.document.userexit:`

- `IDocumentUserExitAccess`: Provides basic functionalities on document level (Sales Order) that are required to perform pricing.
- `IItemUserExitAccess`: Provides basic functionalities on Item level (Sales Order Item) that are required to perform pricing.

`com.sap.spc.document.userexit:`

- `ISPCDocumentUserExitAccess`: Similar to `IDocumentUserExitAccess`.
- `ISPCItemUserExitAccess`: Similar to `IItemUserExitAccess`. Additionally a method is provided to access the product configuration.





## 7.2 Incompatible Interface Changes to earlier releases

### 7.2.1 IConditionFindingManagerUserExit

Earlier Release	Changes in AP 7.00	Comments
Was part of package <code>com.sap.spe.condmgnt.finding.application</code>	Now is part of <code>com.sap.spe.condmgnt.finding.userexit</code>	
<code>getClient()</code>	Removed	Alternatively use <code>com.sap.vmc.runtime.RuntimeInformation.getInstance().getUserInfo().getClient();</code>
<code>getHeaderAttributeBinding(IAttributeClass)</code> <code>getHeaderAttributeBinding(String)</code> <code>getHeaderAttributeValue(String)</code> <code>getItemAttributeBinding(IAttributeClass)</code> <code>getItemAttributeBinding(String)</code> <code>getItemAttributeValue(String)</code>	Removed	Alternatively use <code>getAttributeValue(String)</code> which now returns a <code>String</code> instead of an <code>IAttributeBinding</code> or <code>IAttributeValue</code> .  Additionally, there is no differentiation between Header and Item Attributes in the Userexits.  Consider that <code>getAttributeValue(String)</code> no longer throws an <code>UnsuppliedAttributeException</code> . In case the Attribute is not supplied, an empty string is returned.  Earlier it was possible to return a <code>NULL</code> value. In AP700 it is guaranteed that no <code>NULL</code> value is returned.
<code>getConditionAccessTimestamp(String)</code> <code>getDefaultConditionAccessTimestamp()</code>	Removed	Alternatively use an <code>Attribute</code> in the method <code>getAttributeValue(String)</code>

### 7.2.2 IPricingDocumentUserExit

Earlier Release	Changes in AP 7.00	Comments
Was part of package <code>com.sap.spe.pricing.transactiondata.application</code>	Now is part of <code>com.sap.spe.pricing.transactiondata.userexit</code>	

getLocale() getLanguage()	Removed	Alternatively, to get the language use <code>com.sap.vmc.runtime.RuntimeInformation.getInstance().getUserInfo().getLocaleSettings().getLanguage();</code>
getMessageManager()	Removed	
getStandardExits() getStandardExitsForIBUs()	Removed	
getUserExitConditions()	The return type changed from Vector to IPricingConditionUserExit[]	
getUserExitItems()	The return type changed from Vector to IPricingItemUserExit[]	
getAlwaysPerformGroupConditionProcessing()	isAlwaysPerformingGroupConditionProcessing()	
getKeepZeroPricesActive()	isZeroPriceActive()	
getPartialProcessing()	isPartialProcessing()	

### 7.2.3 IPricingItemUserExit

Earlier Release	Changes in AP 7.00	Comments
Was part of package <code>com.sap.spe.pricing.transactiondata.application</code>	Now is part of <code>com.sap.spe.pricing.transactiondata.userexit</code>	
getConditionsForCumulation()	The return type changed from SortedSet to IPricingConditionUserExit[]	
getUserExitConditions()	The return type changed from Vector to IPricingConditionUserExit[]	

getAccumulatedValuesForConditionsWithPurpose() getDynamicReturnValues() getSubtotals()	The return type changed from HashTable to Map	
getCashDiscount() getCashDiscountBasis() getFreight() getNetPrice() getNetValueWithoutFreight() getSubtotal(char)	The return type changed from BigDecimal to ICurrencyValue	
getConditionsWithoutInvisible()	Removed	Alternatively use getUserExitConditions()
getDocument()	Removed	Alternatively use getUserExitDocument()
getErrorMessage()	Removed	Functionality was not implemented even in earlier releases.
getIsReturn()	isReturn()	
getIsStatistical()	isStatistical()	
getItemId()	getId()	
getNumberOfVisibleConditions()	Removed	Alternatively use getUserExitConditions().length
getPricingTimestamp()	Removed	Alternatively use getConditionFindingTimestamp() on the IPricingConditionUserExit object
getSalesQuantity()	getProductQuantity()	
isHighLevelItem()	Removed	Alternatively check if getHighLevelItemUserExit() == null

## 7.2.4 IPricingConditionUserExit and IGroupConditionUserExit

Earlier Release	Changes in AP 7.00	Comments
Was part of package com.sap.spe.pricing.transactiondata.application	Now is part of com.sap.spe.pricing.transactiondata.userexit	
getConditionUpdate()	isConditionUpdate()	

<code>getCurrencyConversion()</code>	<code>isCurrencyConversion()</code>	
<code>getDeletionAllowed()</code>	<code>isDeletionAllowed()</code>	
<code>getHeaderConditionFlag()</code>	<code>isHeaderCondition()</code>	
<code>getIndicatorStructureCondition()</code>	<code>getStructureConditionFlag()</code>	
<code>getInterCompanyFlag()</code>	<code>isInterCompanyBilling()</code>	
<code>getInvoiceListFlag()</code>	<code>isInvoiceList()</code>	
<code>getItemConditionFlag()</code>	<code>isItemCondition()</code>	
<code>getPricingUnit()</code>	The return type changed from <code>IDimensionalValue</code> to <code>IPhysicalValue</code>	
<code>getPurpose()</code>	The return type changed from <code>String</code> to <code>IConditionPurpose</code>	
<code>getRoundingDifference_Locale()</code>	Removed	Alternatively use <code>getRoundingDifference().toString()</code>
<code>getSalesTaxCode()</code>	<code>getTaxCode()</code>	
<code>getStatistical()</code>	<code>isStatistical()</code>	
<code>getStepNo()</code>	<code>getStepNumber()</code>	
<code>getVariantFlag()</code>	<code>isVariantCondition()</code>	
<code>getVarnumh()</code>	<code>getConditionRecordId()</code>	
<code>hasBeenChangedManually()</code>	<code>isManuallyChanged()</code>	

## 7.2.5 ILastPrice

Earlier Release	Changes in AP 7.00	Comments
Was part of package <code>com.sap.spe.pricing.transactiondata.application</code>	Now is part of package <code>com.sap.spe.pricing.transactiondata.userexit</code>	The Interface is renamed to <code>ILastPriceUserExit</code>

<code>getVarnumh()</code>	<code>getConditionRecordId()</code>	
<code>getStepNo()</code>	<code>getStepNumber()</code>	

## 7.2.6 IDocumentUserExitAccess

Earlier Release	Changes in AP 7.00	Comments
Was part of package <code>com.sap.spe.document</code>	Now is part of <code>com.sap.spe.document.userexit</code>	
<code>addHeaderAttributeBinding(IAttributeClass)</code> <code>addHeaderAttributeBinding(IAttributeClass, IAttributeValue)</code> <code>addHeaderAttributeBinding(IAttributeClass, IAttributeValue[])</code> <code>addHeaderAttributeBinding(String)</code> <code>addHeaderAttributeBinding(String, String)</code> <code>addHeaderAttributeBinding(String, String[])</code>	Removed	Alternatively use <code>addAttributeBinding(String, String[])</code>
<code>getHeaderAttributeBinding(IAttributeClass)</code> <code>getHeaderAttributeBinding(String)</code>	Removed	Alternatively use <code>getAttributeBinding(String)</code>
<code>getHeaderAttributeEnvironment()</code>	Removed	Alternatively use <code>getAttributeEnvironment()</code>
<code>getLocale()</code>	Removed	Alternatively, to get the language use <code>com.sap.vmc.runtime.RuntimeInformation.getInstance().getUserInfo().getLocaleSettings().getLanguage();</code>

<code>getRelevantHeaderAttributes()</code> <code>getRelevantItemAttributes()</code>	Removed	Alternatively use <pre>com.sap.spe.document.DocumentEngineFactory. getFactory().getDocumentEngine().getRelevantAttribute s( documentUserExitAccess.getPricingProcedure().getAppli cation(), documentUserExitAccess.getPricingProcedure().getUsage ( ), documentUserExitAccess.getPricingProcedure().getName( ) );</pre>
<code>isHeaderAttributeRelevantForPricing(String)</code> <code>isItemAttributeRelevantForPricing(String)</code>	<code>isAttributeRelevantForPricing(String)</code>	
<code>setHeaderAttributeEnvironment(IAttributeBinding[])</code>	Removed	Alternatively use <code>addAttributeBinding(String, String[])</code> in a loop

## 7.2.7 ItemUserExitAccess

Earlier Release	Changes in AP 7.00	Comments
Was part of package <code>com.sap.spe.document</code>	Now is part of <code>com.sap.spe.document.userexit</code>	
<code>addHeaderAttributeBinding(IAttributeClass)</code> <code>addHeaderAttributeBinding(IAttributeClass, IAttributeValue)</code> <code>addHeaderAttributeBinding(String)</code> <code>addHeaderAttributeBinding(String, String)</code> <code>addItemAttributeBinding(IAttributeClass)</code> <code>addItemAttributeBinding(IAttributeClass, IAttributeValue)</code> <code>addItemAttributeBinding(String)</code> <code>addItemAttributeBinding(String, String)</code>	Removed	Alternatively use <code>addAttributeBinding(String, String)</code> <code>addAttributeBinding(String, String[])</code>

<code>addVariantCondition(String, String, String)</code>	<code>addVariantCondition(String, double, String)</code>	Convert the String to double before calling the method
<code>getHeaderAttributeBinding(IAttributeClass)</code> <code>getHeaderAttributeBinding(String)</code> <code>getItemAttributeBinding(IAttributeClass)</code> <code>getItemAttributeBinding(String)</code>	Removed	Alternatively use <code>getAttributeBinding(String)</code>
<code>getPricingTimestamp()</code>	Removed	
<code>getSalesQuantity()</code> <code>getSalesQuantityUnit()</code>	Removed	Alternatively use <code>getProductQuantity()</code> <code>getProductQuantity().getUnit()</code>
<code>isHeaderAttributeRelevantForPricing(String)</code> <code>isItemAttributeRelevantForPricing(String)</code>	<code>isAttributeRelevantForPricing(String)</code>	
<code>isHeaderAttributeSetExplicitly(String)</code> <code>isItemAttributeSetExplicitly(String)</code>	<code>isAttributeSetExplicitly(String)</code>	
<code>removeVariantCondition(IVariantCondition)</code>	<code>removeVariantCondition(String)</code>	

## 7.2.8 ISPCDocumentUserExitAccess

Earlier Release	Changes in AP 7.00	Comments
Was part of package <code>com.sap.spc.base</code>	Now is part of <code>com.sap.spc.document.userexit</code>	
<code>getDistributionChannel()</code> <code>getSalesOrganization()</code> <code>getSessionType()</code>	Removed	Earlier those methods were required to read the product sales data which is no longer required in this release.

## 7.2.9 ISPCItemUserExitAccess

Earlier Release	Changes in AP 7.00	Comments
Was part of package <code>com.sap.spc.base</code>	Now is part of <code>com.sap.spc.document.userexit</code>	

## 7.2.10 IPricingUserExits

Earlier Release	Changes in AP 7.00	Comments
Was part of package <code>com.sap.spe.pricing.transactiondata.application</code>	Now is part of package <code>com.sap.spe.condmgnt.finding.userexit</code>  <code>com.sap.spe.pricing.transactiondata.userexit</code>	
<code>determineRelevantAttributesForFormulaIndependentUserExits(boolean)</code>  <code>determineRelevantAttributesForRequirement(boolean, int)</code>  <code>determineRelevantAttributesForBaseFormula(boolean, int)</code>  <code>determineRelevantAttributesForValueFormula(boolean, int)</code>  <code>determineRelevantAttributesForScaleBaseFormula(boolean, int)</code>  <code>determineRelevantAttributesForGroupConditionKey(boolean, int)</code>  <code>determineRelevantAttributesForPricingCopyFormula(boolean, int)</code>	Removed	Alternatively use transaction <code>/n/SAPCND/UEASS</code> to maintain the relevant attributes for the formulas. Refer to Ch. 3.



<code>getRequirementNumbers()</code> <code>getConditionBaseFormulaNumbers()</code> <code>getConditionValueFormulaNumbers()</code> <code>getConditionScaleBaseFormulaNumbers()</code> <code>getGroupConditionKeyFormulaNumbers()</code> <code>getPricingCopyFormulaNumbers()</code>	Removed	Alternatively use transaction /n/SAPCND/UEASS to maintain the formula numbers. Refer to Ch. 3.
<code>pricingConditionInit(IPricingDocumentUserExit, IPricingItemUserExit, IPricingConditionUserExit)</code>	Removed	Alternatively extend the PricingConditionInitFormulaAdapter and implement the method <code>init(IPricingDocumentUserExit, IPricingItemUserExit, IPricingConditionUserExit)</code> as described in Ch. 4.2.
<code>pricingDocumentInit(IPricingDocumentUserExit)</code>	Removed	Alternatively extend the PricingDocumentInitFormulaAdapter and implement the method <code>init(IPricingDocumentUserExit)</code> as described in Ch. 4.2.
<code>pricingItemCalculateBegin(IPricingDocumentUserExit, IPricingItemUserExit)</code>	Removed	Alternatively extend the PricingItemCalculateBeginFormulaAdapter and implement the method <code>calculationBegin(IPricingDocumentUserExit, IPricingItemUserExit)</code> as described in Ch. 4.2.
<code>pricingItemCalculateEnd(IPricingDocumentUserExit, IPricingItemUserExit)</code>	Removed	Alternatively extend the PricingItemCalculateEndFormulaAdapter and implement the method <code>calculationEnd(IPricingDocumentUserExit, IPricingItemUserExit)</code> as described in Ch. 4.2.
<code>pricingItemInit(IPricingDocumentUserExit, IPricingItemUserExit)</code>	Removed	Alternatively extend the PricingItemInitFormulaAdapter and implement the method <code>init(IPricingDocumentUserExit, IPricingItemUserExit)</code> as described in Ch. 4.2.
<code>init(IEngineUserExit)</code>	Removed	Alternatively  to get the Client use: <code>com.sap.vmc.runtime.RuntimeInformation.getInstance().getUserInfo().getClient();</code>  to get the database connection: <code>com.sap.sxe.db.db.getDb()</code> . This is not recommended nor supported by SAP. Refer to section "2.5 Restriction on the Java Implementation"
<code>checkRequirement(IConditionFindingManagerUserExit, IStep, int)</code>	Removed	Alternatively extend the RequirementAdapter and implement the method <code>checkRequirement(IConditionFindingManagerUserExit, IStep, IAccess)</code> as described in Ch. 4.2.

<pre>overwriteConditionBase(IPricingItemUserExit, ILastPrice, IPricingConditionUserExit, int)</pre>	Removed	<p>Alternatively extend the BaseFormulaAdapter and implement the method <code>overwriteConditionBase(IPricingItemUserExit, IPricingConditionUserExit)</code> as described in Ch. 4.2.</p> <p>Note: to get the <code>ILastPrice</code> use method <code>IPricingItemUserExit.getUserExitLastPrice()</code>.</p>
<pre>overwriteConditionValue(IPricingItemUserExit, ILastPrice, IPricingConditionUserExit, int)  overwriteGroupConditionValue(IPricingDocumentU serExit, IGroupConditionUserExit, int)</pre>	Removed	<p>Alternatively extend the ValueFormulaAdapter and implement the method <code>overwriteConditionValue(IPricingItemUserExit, IPricingConditionUserExit)</code> or/and <code>overwriteConditionValue(IPricingItemUserExit, IPricingConditionUserExit)</code> as described in Ch. 4.2.</p> <p>Note: to get the <code>ILastPrice</code> use method <code>IPricingItemUserExit.getUserExitLastPrice()</code>.</p>
<pre>overwriteGroupConditionScaleBase(IPricingDocum entUserExit, IGroupConditionUserExit, int)  overwriteScaleBase(IPricingItemUserExit, ILastPrice, IPricingConditionUserExit, IGroupConditionUserExit, int)</pre>	Removed	<p>Alternatively extend the ScaleBaseFormulaAdapter and implement the method <code>overwritesScaleBase(IPricingItemUserExit, IPricingConditionUserExit, IGroupConditionUserExit)</code> or/and <code>overwriteGroupScaleBase(IPricingDocumentUserExit, IGroupConditionUserExit)</code> as described in Ch. 4.2.</p> <p>Note: to get the <code>ILastPrice</code> use method <code>IPricingItemUserExit.getUserExitLastPrice()</code>.</p>
<pre>setGroupConditionKey(IPricingDocumentUserExit, IPricingItemUserExit, IPricingConditionUserExit, IGroupConditionUserExit, int)</pre>	Removed	<p>Alternatively extend the GroupKeyFormulaAdapter and implement the <code>setGroupKey(IPricingDocumentUserExit, IPricingItemUserExit, IPricingConditionUserExit, IGroupConditionUserExit)</code> as described in Ch. 4.2.</p>
<pre>pricingCopy(IPricingDocumentUserExit, IPricingItemUserExit, IPricingConditionUserExit, IPricingType, ICopyType, IQuantityValue, int)</pre>	Removed	<p>Alternatively extend the PricingCopyFormulaAdapter and implement the method <code>pricingCopy(IPricingDocumentUserExit, IPricingItemUserExit, IPricingConditionUserExit, IPricingType, ICopyType, IQuantityValue)</code> as described in Ch. 4.2.</p>

## 7.2.11 IDocumentUserExit

Earlier Release	Changes in AP 7.00	Comments
-----------------	--------------------	----------

Was part of package <code>com.sap.spe.document</code>	Now is part of <code>com.sap.spe.document.userexit</code>	
<code>init(IEngineUserExit)</code>	Removed	Alternatively to get the Client use: <code>com.sap.vmc.runtime.RuntimeInformation.getInstance().getUserInfo().getClient();</code>  To get the database connection: <code>com.sap.sxe.db.db.getDb()</code> . This is not recommended nor supported by SAP. Refer to section “2.5 Restriction on the Java Implementation”
<code>determineRelevantAttributes()</code>	Removed	Alternatively use transaction <code>/n/SAPCND/UEASS</code> to maintain the relevant attributes for the formulas. Refer to Ch. 3.
<code>initializeDocument(IDocumentUserExitAccess)</code>	Removed	Alternatively extend the <code>PricingInitFormulaAdapter</code> and implement the method <code>initializeDocument(IDocumentUserExitAccess)</code> as described in Ch. 4.2.

## 7.2.12 ItemUserExit

Earlier Release	Changes in AP 7.00	Comments
Was part of package <code>com.sap.spe.document</code>	Now is part of <code>com.sap.spe.document.userexit</code>	
<code>init(IEngineUserExit)</code>	Removed	Alternatively to get the Client use: <code>com.sap.vmc.runtime.RuntimeInformation.getInstance().getUserInfo().getClient();</code>  To get the database connection: <code>com.sap.sxe.db.db.getDb()</code> . This is not recommended nor supported by SAP. Refer to section “2.5 Restriction on the Java Implementation”
<code>determineRelevantAttributes(boolean)</code>	Removed	Alternatively use transaction <code>/n/SAPCND/UEASS</code> to maintain the relevant attributes for the formulas. Refer to Ch. 3.

<code>addAttributeBindings(IItemUserExitAccess)</code>	Removed	Alternatively extend the <code>PricingPrepareFormulaAdapter</code> and implement the method <code>addAttributeBindings(IItemUserExitAccess)</code> as described in Ch. 4.2.
--	---------	---

### 7.2.13 ISPCItemUserExit

Earlier Release	Changes in AP 7.00	Comments
Was part of package <code>com.sap.spc.base</code>	Now is part of <code>com.sap.spc.document.userexit</code>	
<code>init(IEngineUserExit)</code>	Removed	Alternatively to get the Client use: <code>com.sap.vmc.runtime.RuntimeInformation.getInstance().getUserInfo().getClient();</code>  To get the database connection: <code>com.sap.sxe.db.db.getDb()</code> . This is not recommended nor supported by SAP. Refer to section “Restriction on the Java Implementation”
<code>determineRelevantAttributes(boolean)</code>	Removed	Alternatively use transaction <code>/n/SAPCND/UEASS</code> to maintain the relevant attributes for the formulas. Refer to Ch. 3.
<code>isSubItemCreatedBySceRelevantForPricing(ISPCItemUserExitAccess, Instance)</code>	Removed	Alternatively extend the <code>SPCSubItemCreatedByConfigurationFormulaAdapter</code> and implement the method <code>isRelevantForPricing(ISPCItemUserExitAccess, Instance)</code> as described in Ch. 4.2.

## A BACKGROUND ON JAVA RESTRICTIONS

Chapter 2.5 provides a short list of restrictions on the java implementation of user exits in pricing. In this appendix additional explanations and background information are provided.

The Virtual Machine Container (VMC) is part of the ABAP Web Application Server's Kernel which enables Java Bytecode to be executed in the same environment as ABAP. This means that the user exits are executed in a multi-server and multi-VM environment.

The VMC creates dynamically several Java VMs processing different requests. The user's session data is dynamically attached to a free VM to process a request and is possibly detached afterwards so the Java VM can process another user request.

Shared Memory is the memory which is used and managed by the VMC (once per Application Server Instance). This Shared Memory contains

- a pool of loaded and preprocessed/precompiled Java classes
- all loaded user sessions
- a pool of application data caches, like configuration (pricing procedures) or master data (e.g. condition records)

The Java VM itself adheres to the normal memory concept of a standard Java Runtime Environment. As a part of the Java VM process separation (fail safety) e.g. the static members of each class are stored in that Java VM only. Also native references (like JCo, File, Database, JDBC, ...) are kept and managed by that single Java VM. The normal class-instances created in that Java VM are also managed by that VM only.

During the processing of a request some user's session specific data might be created. This data (instances of classes) is copied per request to the shared memory. At the next request the (new) Java VM can get those user instances again by copying (or mapping – then read-only) them back from the shared memory. This sharing of classes and sharing of instances through the shared memory is the main reason for the listed restrictions.

### A.1 Shared Memory caused restrictions

For each registered user exit an instance is created in the heap memory of each Java VM. Static variables are kept local to that Java VM. Later this implementation detail may change and the instance of this user exit is kept in the shared memory.

All instances of user exits and objects referenced by user exits must be shareable. Shareability is a restriction on allowed instances and class constellations to allow the VMC to do a fast copy of the data from the Java VM into the shared memory and back.

The important rules to follow for customer classes are:

- Implement `java.io.Serializable`
- Do not use any custom code during serialization by implementing methods like `readObject`, `writeObject`, `readExternal`, `writeExternal`, `readResolve`, `writeResolve`
- Do not have transient fields
- Do not use `java.lang.Properties` (not shareable)
- Do not use own classloaders
- Do not influence the Garbage Collector by implementing `finalize()` or derive from `java.lang.Reference`

Static members in customer code are not appreciated. The referenced objects stay in the Java VM and can pollute the Java VM memory. These references are local to that Java VM and the user context may not run next time on the same Java VM. This restriction can be relaxed for simple static final members, e.g. to `java.lang.String`, `java.math.BigDecimal` or primitives like `double`, but using any container class (e.g. from class tree `java.lang.Map`) is strongly discouraged.

As the user exit instance is kept per Java VM (current pricing implementation), all instance members are shared and therefore are accessible by each request running on the same Java VM. Any use of instance members, which are not used within the same single method run, can lead to problems. That's setting the member in one call and using the member in another. In later Support Packages of the pricing engine the use of members in any user exit class (also helper classes) will be causing issues, because the instance will be kept in shared memory and will only have read-only accessibility.

## A.2 Reuse of Java VM caused restrictions

The VMC technology does not support application code concurrency. That means that the code must not use

- `java.lang.Threads`
- `sleep()` / `wait()`
- `synchronize`

The use of the above mentioned functionalities could lead to a break up of the transaction integrity or lock a Java VM.

This rule above is the main reason to avoid the use of DB, File, Socket, JCo functionality. Each of those functionalities creates native references to the Operating System. A Java VM referencing native code can not be shared and the request running might be mapped to another Java VM.

## B OTHER WAYS TO INFLUENCE PRICING

Pricing is a highly customizable and configurable engine, but sometimes the normal possibilities provided by the pricing engine are not sufficient. With the user exits nearly all special business requirements can be fulfilled. The VMC restricts some of the used techniques so that adaptations need to be developed. One example is the use of DB selects to retrieve additional information in e.g. ItemUserExit or requirement. Most performance issues other than group conditions and exclusion logic are caused by user exit implementations.

Different ways of passing additional information or passing back extra results can be achieved by using the techniques described in this chapter.

Please be reminded that using any of these techniques described in this document is not supported with a standard SAP support contract (same as for the whole userexits concept).

### B.1 Passing additional information to pricing

A good understanding of the condition technique (e.g. field catalogue) is a prerequisite to understand this technique.

Pricing Attributes are used to access condition table during condition finding. E.g. the sold-to party is such an attribute with a name and a value. These attributes can not only be used in access sequences but also in many user exits. The retrieval of information for these attributes is best done outside the pricing engine and its user exits. This means that e.g. the requirement (which is executed a many times) only takes that piece of information and uses it, but does not deal with the retrieval.

To summarize, if the information used in a requirement is not depending on the current calculation result, then the information can be passed to the pricing engine. This is the best way to avoid e.g. DB or JCo calls from the user exits.

Nearly all application scenarios using the pricing engine provide a way of filling additional attributes.

1. Create a field catalogue entry for the additional attribute
2. Program the filling of that attribute value

The application calling the pricing engine and the way it is allowing customer code to fill or influence attributes is not part of this document. Here is only a short list of documentation to start with (also various notes exist):

Application	Description
CRM Online Order Processing	Implement BADI CRM_COND_COM_BADI. Note 850077
ISA R/3 and CRM	Different BADIs available (first choice) or enhance backend objects. Service Market Place "Development and Extension Guide – SAP Internet Sales"
Mobile Solutions	Implement VBA user exit methods on SPCDOCHANDLER or SPCITEMHANDLER. Note 677314 and Note 934216

### B.2 External Data Source for Conditions

Some pricing conditions types can not be retrieved by normal condition finding. That is the configuration of access sequences and condition tables is not flexible enough. In such a case an additional attribute, as shown in B.1, can be used to fill the correct value of a condition.

An equivalent approach is supported in ABAP based scenarios (e.g. not MSA or ISA catalogue) by using the External Data Source feature. To use this feature the pricing condition type must have the data source field (DTASRC) configured to X/Y/Z and the BADI PRC\_DATA\_SOURCES implemented.

## C FAQ

### C.1 Implementation

#### 1. How do I pass information from one user exit to another?

It is possible to transfer object references to the pricing document as well as to each item. Many user exits allow access to the interface `IPricingItemUserExit` which provides the following methods for storing and retrieving data:

- `IPricingItemUserExit.setObjectForUserExits(String key, Object obj)`
- `Object IPricingItemUserExit.getObjectForUserExits(String key)`

Use the parameter `key` as a description of the information you want to store. `obj` is the stored object, containing your data, which is shared and can be accessed later across the same item.

The same technique can be used to store data, unique within the entire document, by using the interface `IPricingDocumentUserExit`. This can be accessed (if not passed in as a parameter) by calling `IPricingItemUserExit.getUserExitDocument()`.

Cautions: 1. All objects passed must be shareable. 2. Your coding should not depend on the sequence in which the user exits are called. The pricing procedure is normally processed from top to the bottom i.e. a formula should only depend on data calculated or set in the preceding lines of the pricing procedure. The sequence of processing of the pricing items is not guaranteed.

#### 2. How can I add or alter attribute values?

The attributes can be altered before the actual pricing starts (e.g. `PricingInitFormulaAdapter`). In this class you have access to the method `IDocumentUserExitAccess.addAttributeBinding(String attributeName, String[] attributeValues)`. This call will replace the attribute values with the passed ones and returns the old attribute values.

Important: The attribute name must be registered.

#### 3. How can I show error or warning messages to application users?

A message can be raised in the application log using the following methods. Depending on the scenario the message is then displayed along with the other application messages. This is not supported in every scenario, e.g. Mobile Applications.

- `IPricingDocumentUserExit.setStatusMessage(StatusEvent event)`
- `IPricingItemUserExit.setStatusMessage(StatusEvent event)`

As `StatusEvent` you should use one of its subclasses `ErrorStatusEvent`, `InfoStatusEvent` or `WarningStatusEvent` (all from package `com.sap.spe.base.util.event`. Here is the signature of the method to use:

```
/** Creates a new instance with specified context
 * @param source the source of this event
 * @param messageArea the message area of the status information
 * @param messageNumber the message number of the status information
 * @param messageArguments the arguments of the status information
 * @param message the current status information
 * @param context the context of this event
 * @param popup open message in PopUp
 */
public StatusEvent(Object source, String messageArea, int messageNumber,
    String[] messageArguments, String message, String context, boolean popup);
```

The boolean `popup` should be false. True is not supported, source should be the local object `this`, `messageArea` is the ABAP message class.



## C.2 Troubleshooting

### 4. My user exits are not processed. What can I do?

You can check with the transaction `sm53` if your jar package has been uploaded and the files are listed. As a second step the registration must be checked. Be careful, the entries for the class name are case sensitive, with full specified java package name and must not end with `.class`. You can further check the vmc log files for some indications, e.g. class could not be found or instantiated or doesn't extend the right class. Make also sure that you have assigned a formula number to it (even if the userexit type has the scope B or C).