

# Getting Improved Performances in BW-BPS / BI-IP When Using Enhancements Trough the ABAP Technology (Exits and Classes)

## Applies to:

BW-BPS 3.5

BI-IP 7.0

## Summary

In BW-BPS and BI-IP, the usage of the ABAP enhancements can be dramatically optimized. This document offers some golden rules for a selective technical public, but it also concerns the business team.

**Author(s):** Laurent Thibert

**Company:** SAP Switzerland

**Created on:** 10<sup>th</sup> July 2007

## Author Bio



Laurent Thibert is a Senior Business Intelligence Consultant at SAP Switzerland. He has been involved on different BI-SEM projects.

## Table of Contents

Situation .....	3
How does that work? .....	4
Organization of the code .....	4
Content of the code .....	4
Business Example .....	4
What's wrong with that? .....	5
What to do? .....	6
How to do in BW 3.5: ABAP function group and ABAP functions .....	7
The ABAP workbench: transaction SE80 .....	8
Furthermore in BW-BPS 3.5 .....	9
How to do in BI 7.0: Classes and methods .....	10
Example of object class .....	10
A new issue!!! What's wrong with that? (another topic) .....	11
The methodology .....	12
Another new issue!!! What's wrong with that? .....	14
Related Content .....	15
Copyright .....	16

## Situation

In some cases, we need to enhance the functionalities of BW-BPS / BI-IP help to the ABAP technology. SAP delivers some templates that we can copy and adapt for a usage at different moments:

- for determining the value(s) of a variable
- for determining some additional information when updating a cube (derivation)
- for modifying the content of the planning buffer (by adding/deleting some rows)
- ...

The performances of the SAP server can be degraded in the usages of those extensions are not optimal. This document offers some advanced information on how to build flexible and powerful extensions of BW-BPS and BI-IP.

## How does that work?

The different ABAP codes are called by the planning application at the right moment (variables values determination, derivation of additional information ...), or by a direct call (planning function, planning sequence ...).

The ABAP instructions are contained between two main instructions:

BW 3.5	BI 7.0
Function XXX.  Your code here.  Endfunction.	Method XXX.  Your code here.  Endmethod.

## Organization of the code

Most of the time, the ABAP code will contain two main blocks:

- declaration block

Here we can create some work areas which will temporally contain the data while the ABAP exit is applied.

- algorithm block

Here we can use the previous work areas to apply the business logic.

## Content of the code

Most of the time, such enhancement will need some external data. These data will condition the application of the ABAP coding by supplying the business requirements (customizing, business rules, other business data ...).

These data can be accessed from:

- a database table
- a file
- a transactional / real time cube
- others ...

The data are stored in some work areas that are declared in the first block (declaration) and used by the ABAP algorithm.

## Business Example

An end user is expecting to determine the financial depreciations of some investments. There are several strategies for the calculation of the depreciations. One of the strategies is a linear depreciation (20% by year for the next 5 year). Another could be accelerated depreciation (50 % by year for the next 4 years and the residual value for the fifth year).

The customizing must indicate the association of the investments to some cost elements. The cost elements will be associated to a depreciation strategy.

The end users will have a BW-BPS / BI-IP screen to input the values of the investments themselves. Then they will execute the SAP enhancement to calculate the depreciations for the next years.

The two ABAP blocks of the enhancement are themselves divided in two blocks.

Declaration block:

- technical work areas to store the content of the planning buffer and add some rows
- technical work areas to read the customizing

Algorithm block:

- instructions for filling the work areas associated to the customizing (getting the right information of the depreciation strategy)
- instructions for filling the planning buffer with new rows determined help to the customizing (application of the depreciation strategy)

### What's wrong with that?

The functions or methods are called several times by the planning application for a same screen, a same function, a single input ... And we do not master the different calls of the exits. The reasons for multiple calls are:

- navigation of the end user
- execution of an enhancement through several data packages
- execution of a planning sequence with several planning functions
- custom design of the web applications or excel workbooks
- standard design of the BW-BPS / BI-IP solutions themselves

This means that the same external data are identically loaded several times, which is definitively not useful.

### **What to do?**

Basically these data should be loaded only once, at the first call of the corresponding enhancement. Then, as long as the end user stays connected to the application, the enhancements should only focus on the business data in the planning buffer.

Of course, if some changes would occur in these customizing data, the end user would have to sign off and sign on again so the most recent customizing values are contained in the buffer as well.

## How to do in BW 3.5: ABAP function group and ABAP functions

In BW 3.5, the enhancements are supported by the function modules technology. The functions are ABAP programs beginning with **Function** and ending with **Endfunction** as instructions. Such programs are not executable online, but are called by another ABAP program help to the instruction **Call function 'XXX'**.

This is how SAP offers to enhance its planning solution, by supplying the different calls of some functions that we can adapt.

However, the functions are not independents in the ABAP workbench. They are created within a **Function group**. The goal of a function group is to 'group' different functions which will do different tasks, but which will also share some common data.

### Technical example

A function group for the sales orders will contain 3 functions:

- add an item
- delete an item
- modify an item

The functions will share the same information of the order, but will act differently according to their role.

**Here we are:** the function group has got its own ABAP coding. This ABAP coding permits to declare some work areas (like a table to store the data of the sales order, the data of the customizing ...). This ABAP coding permits as well **to feed** these work areas **at the first call** of **any function belonging to that function group** and **before that function is itself applied**.

This means that we have the occasion to read only once all the external data that we need, and that we can avoid having some recurrent calls and performances degradations.

## The ABAP workbench: transaction SE80

The screenshot shows the SAP ABAP Editor interface. The left-hand pane displays the object browser for the function group Z\_BPS\_DEMO. The 'Includes' folder is expanded, showing the include LZ\_BPS\_DEMOTOP. The main editor area displays the ABAP code for the include, which includes type declarations, internal table declarations, work areas for reading BPS data, and a call to a function 'API\_SEMBPS\_GETDATA' with various parameters. The code is highlighted with red boxes to indicate key sections.

```

FUNCTION-POOL Z_BPS_DEMO.                "MESSAGE-ID ..."

* Types declaration
TYPES : BEGIN OF s_tab_cub1, "pour stocker les méthodes d'amortissement
        comp_code TYPE /b10/oicomp_code,
        area      TYPE /b10/oico_area,
        asset_class TYPE /b10/oiasset_clas,
        dep_area  TYPE /b10/oideprearea,
        cost_elem TYPE /b10/oicostelmnt,
        amount    TYPE /b10/oiamount,
      END OF s_tab_cub1.

* Internal tables and areas declaration
DATA: tab_cub1 TYPE STANDARD TABLE OF s_tab_cub1,
      wa_cub1  TYPE s_tab_cub1,
      index   TYPE i,

* Work areas for reading the BPS data
etk_row TYPE STANDARD TABLE OF upc_ys_api_row,
wa_row  TYPE upc_ys_api_row,
etk_data TYPE STANDARD TABLE OF upc_ys_api_data,
wa_data TYPE upc_ys_api_data.

FIELD-SYMBOLS : <field_ref> TYPE ANY.

LOAD-OF-PROGRAM.

* Lecture du cube de paramétrage Cube1
CALL FUNCTION 'API_SEMBPS_GETDATA'
  EXPORTING
    i_area      = 'ZFIADMS1'
    i_plevel   = 'PLIMMOS'
    i_package  = '0-ADHOC'
    i_layout   = 'LA001'
    i_read_only = 'X'
  TABLES
    etk_row    = etk_row
  
```

A function is organized with folders. One folder will contain the function modules. Another folder will contain the 'includes' programs necessary for the function group. The include known as the TOP include is the one that we can adapt to:

- declare the work areas to store the customizing data
- feed those work areas with the right data

It is necessary to use the right ABAP instruction so SAP can understand our goal at the execution. The ABAP coding will use the instruction **LOAD-OF-PROGRAM**. This instruction is considered by SAP as an **event**. It is executed at **the loading of the function group**. The ABAP coding written below this instruction is applied at this event.

### **Furthermore in BW-BPS 3.5**

As the function group can have a significant role in the performances, we should avoid having a single one for all the function modules. Indeed most of the time, a single function group exists per business planning area like Z\_BPS\_SD for Commercial Forecast enhancements, Z\_BPS\_FI for Financials enhancements ...

So from now we should consider having as much groups as necessary Z\_BPS\_SD1, Z\_BPS\_SD2 ... in order that all the functions do not load some data that they do not need.

## How to do in BI 7.0: Classes and methods

In BI 7.0, the enhancements are supported by the ABAP Object Oriented technology. The methods are still ABAP programs beginning with **Method** and ending with **Endmethod** as instructions. Such programs are not executable online, but are called by another ABAP program help to the instruction **Call method 'XXX'**.

SAP delivers from now some templates help to what is called **interfaces**. The interfaces are empty ABAP programs templates, which must be associated to a customer object class. The BI-IP planning functions will call the methods within the object classes, help a **function type** that targets the object class and delivers some others features (detailed later).

The organization of the coding is quite similar to the function groups as there is a container called that time **object class**. The different interfaces deliver several methods that the customers may enhance.

### Example of object class

The following object class has got two interfaces supplied by SAP for enhancing the planning functions.

The screenshot shows the SAP Class Builder interface for the class Z\_IP\_EXIT\_SD\_FORECAST. The 'Interfaces' tab is active, displaying a table of implemented interfaces.

Interface	Abstract	Final	Model...	Description
IF_RSPLFA_SRVTYPE_IMP_CHECK	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Planning Function Type: Checks
IF_RSPLFA_SRVTYPE_IMP_EXEC	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Planning Function Type: Execution (Without Reference Data)
	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

The descriptions of the methods are explicit enough to understand which one is to be used depending of the needs.

The screenshot shows the SAP Class Builder interface for Class Z\_IP\_EXIT\_SD\_FORECAST. The 'Methods' tab is selected, displaying a list of methods with their levels, visibility, and descriptions.

Method	Level	Visibility	M...	Description
IF_RSPLFA_SRVTYPE_IMP_EXEC~INIT_EXECUTION	Instance Method	Public		Initialization for Execution
IF_RSPLFA_SRVTYPE_IMP_EXEC~EXECUTE	Instance Method	Public		Execution
IF_RSPLFA_SRVTYPE_IMP_EXEC~FINISH_EXECUTION	Instance Method	Public		Actions at End of Execution
IF_RSPLFA_SRVTYPE_IMP_CHECK~INIT_CHECK	Instance Method	Public		Initialization for Checks
IF_RSPLFA_SRVTYPE_IMP_CHECK~CHECK_CHAR_USAGE	Instance Method	Public		Characteristic Usage Check
IF_RSPLFA_SRVTYPE_IMP_CHECK~CHECK_PARAM_SET	Instance Method	Public		Parameter Record Check

By double clicking on a method, we can access the ABAP editor for applying the enhancement.

The screenshot shows the SAP ABAP editor for the method IF\_RSPLFA\_SRVTYPE\_IMP\_EXEC~INIT\_EXECUTION. The code is as follows:

```
method IF_RSPLFA_SRVTYPE_IMP_EXEC~INIT_EXECUTION.
  * YOUR CODE HERE ...
endmethod.
```

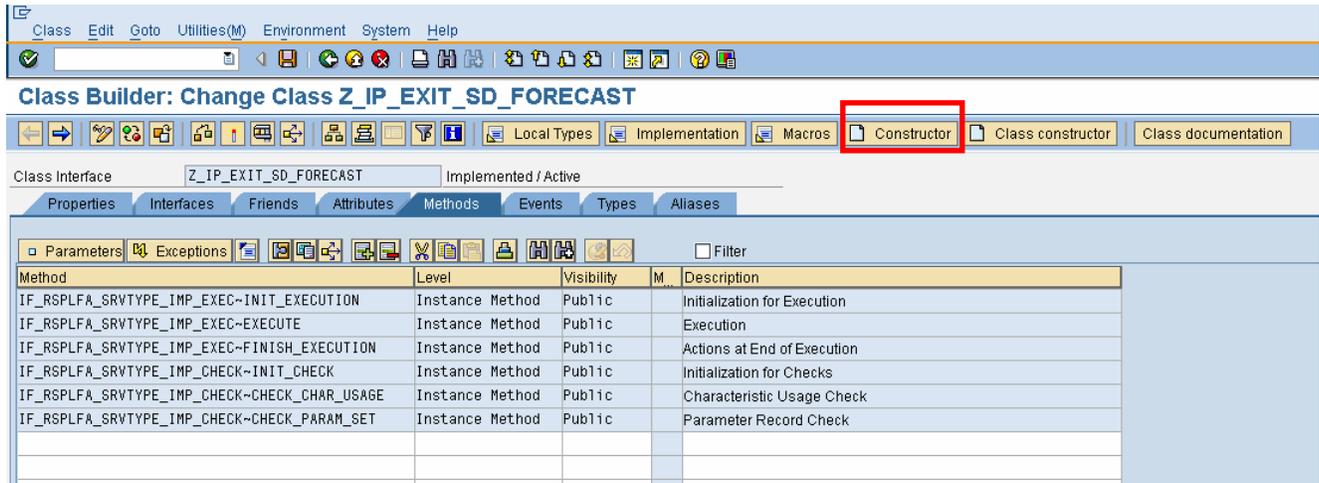
### A new issue!!! What's wrong with that? (another topic)

The organization of the object class does not deliver by default a method that is called uniquely at its instantiation as opposed to the function groups. However, this is only a default behaviour that can be modified.

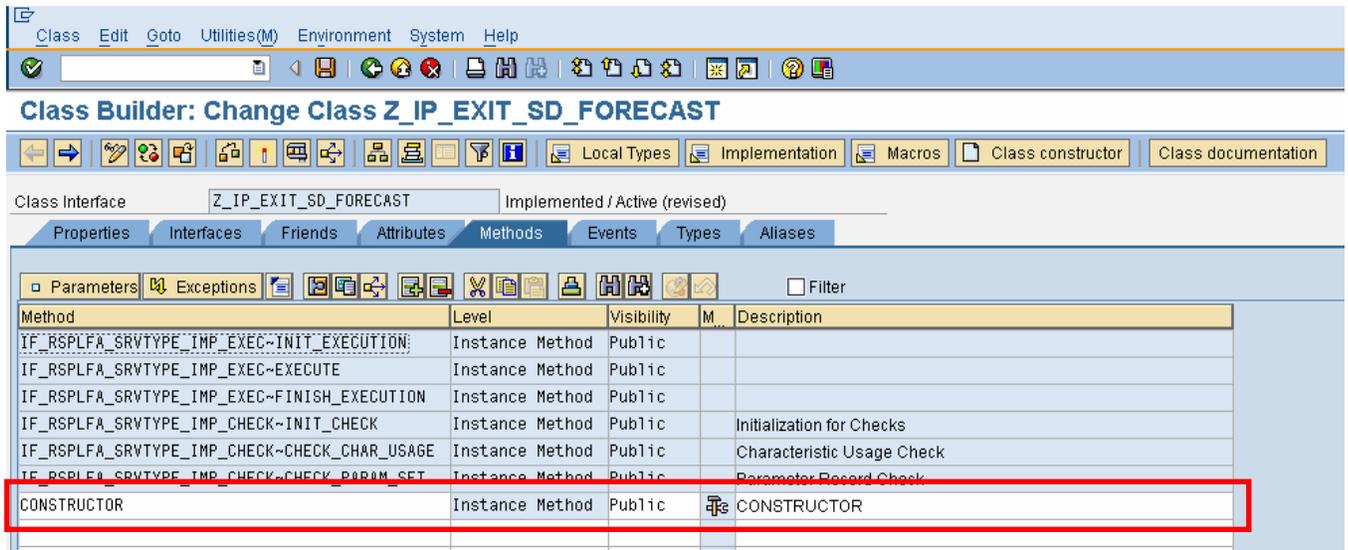
Therefore it is necessary to add one standard method called **CONSTRUCTOR**. This method has been designed by SAP as the one executed only once when calling the object at the first time.

## The methodology

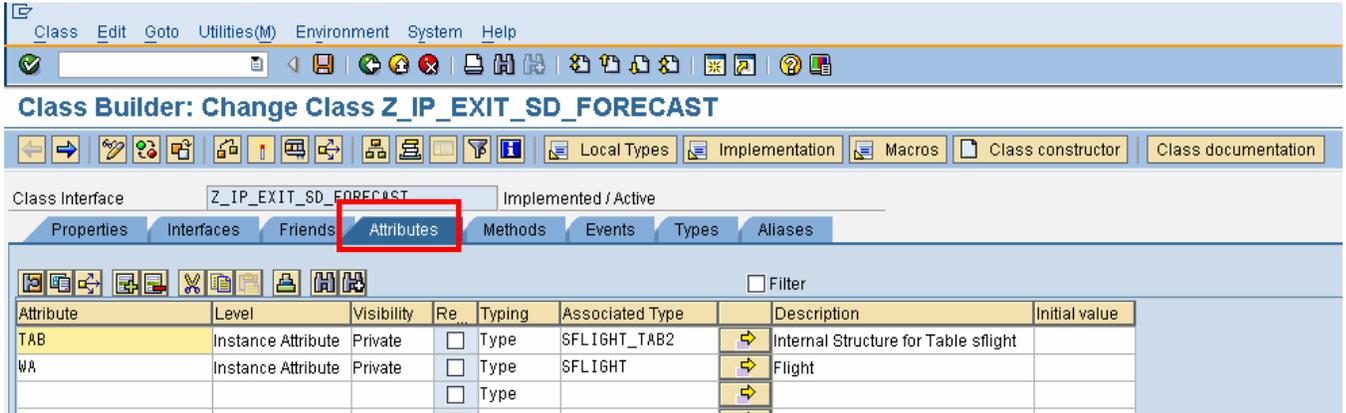
From the initial screen you can create the CONSTRUCTOR by clicking on the corresponding button.



You get then a new method that can be coded for unique ABAP statements.



This method is able to load some data like the function groups. However, the declaration of the data can not be done in ABAP. This must be done in the tab called **Attributes**. Here there is a declaration of an internal table **TAB** which will be loaded help to the CONSTRUCTOR method. And there is as well a work area **WA** that will help reading the content of the table TAB.



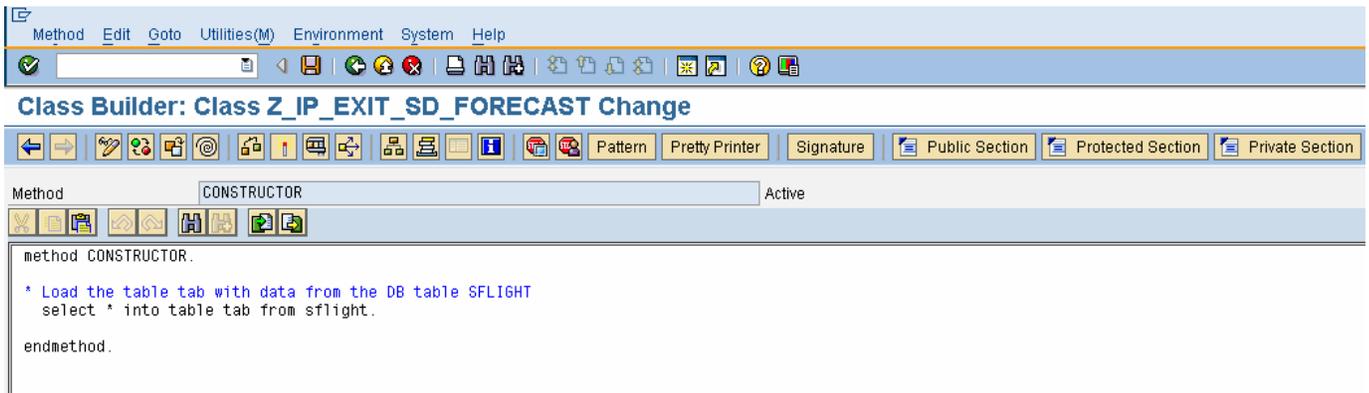
Class Builder: Change Class Z\_IP\_EXIT\_SD\_FORECAST

Class Interface: Z\_IP\_EXIT\_SD\_FORECAST Implemented / Active

Properties Interfaces Friends **Attributes** Methods Events Types Aliases

Attribute	Level	Visibility	Re	Typing	Associated Type	Description	Initial value
TAB	Instance Attribute	Private	<input type="checkbox"/>	Type	SFLIGHT_TAB2	Internal Structure for Table sflight	
WA	Instance Attribute	Private	<input type="checkbox"/>	Type	SFLIGHT	Flight	

Now the method CONSTRUCTOR can contain the ABAP instructions to fill in the table TAB.



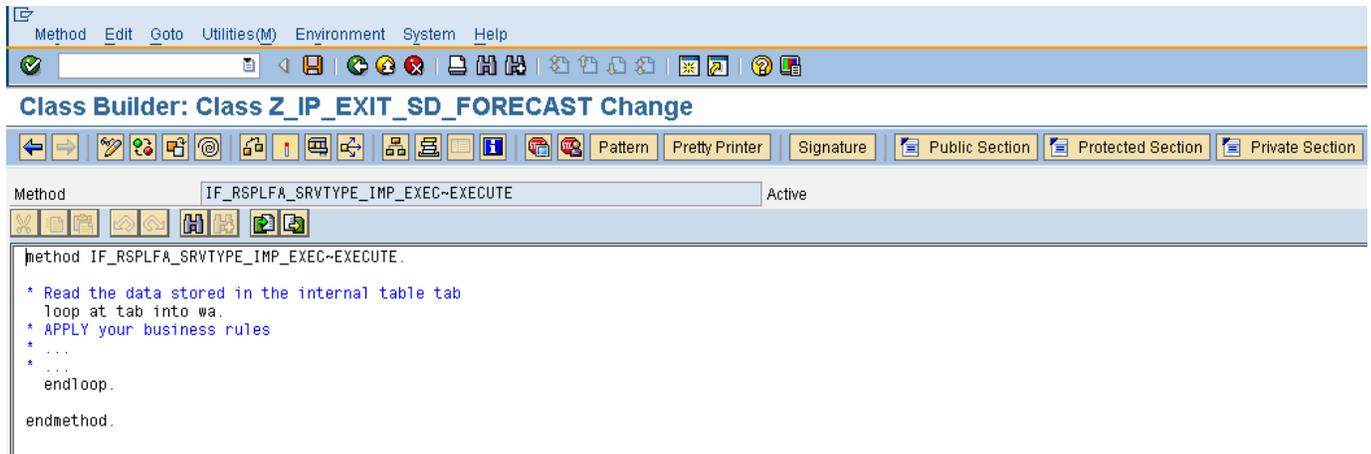
Method: CONSTRUCTOR Active

```

method CONSTRUCTOR.
* Load the table tab with data from the DB table SFLIGHT
  select * into table tab from sflight.
endmethod.

```

And finally other methods like the following can reach those data.



### Another new issue!!! What's wrong with that?

In BI-IP 7.0 you can not add as many methods as needed to a single object class, as opposed to BW-BPS in 3.5 where a single function group could contain as many functions as needed.

This means that the implementation of each method of an object class is unique ... by default once again luckily!

If we want to implement several times the same method, for different business rules sharing the same data to be loaded by the CONSTRUCTOR method, we need to differentiate them within the code itself.

The customizing of BI-IP offers us an easy solution as it is possible to add some parameters to the planning functions. These parameters will provide their values to the enhancements. Therefore we can add a key figure as a parameter. The value will be checked so the same method could apply different ABAP statements depending on the value of the parameter.

Example of the suggested code:

Case param\_value.

When '001'.

\* Apply the rules for SD Forecast Business Rule 1 Order based

When '002'.

\* Apply the rules for SD Forecast Business Rule 2 Invoices based

Endcase.

## **Related Content**

The [How to Load a File into BI-Integrated Planning](#) paper explains clearly the methodology for transferring the parameters from BI-IP customizing to the corresponding function types, and as well how to read the values of the parameters with the correct ABAP statements.

## Copyright

© Copyright 2007 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, Outlook, and PowerPoint 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, i5/OS, POWER, POWER5, OpenPower and PowerPC are trademarks or registered trademarks of IBM Corporation.

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

Oracle is a registered trademark of Oracle Corporation.

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

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

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

Java is a registered trademark of Sun Microsystems, Inc.

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

MaxDB is a trademark of MySQL AB, Sweden.

SAP, R/3, mySAP, mySAP.com, xApps, xApp, SAP NetWeaver, 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. Data contained in this document serves informational purposes only. National product specifications may vary.

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

These materials are provided "as is" without a warranty of any kind, either express or implied, including but not limited to, the implied warranties of merchantability, fitness for a particular purpose, or non-infringement.

SAP shall not be liable for damages of any kind including without limitation direct, special, indirect, or consequential damages that may result from the use of these materials.

SAP does not warrant the accuracy or completeness of the information, text, graphics, links or other items contained within these materials. SAP has no control over the information that you may access through the use of hot links contained in these materials and does not endorse your use of third party web pages nor provide any warranty whatsoever relating to third party web pages.

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