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: 10th 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

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 deprecations 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:

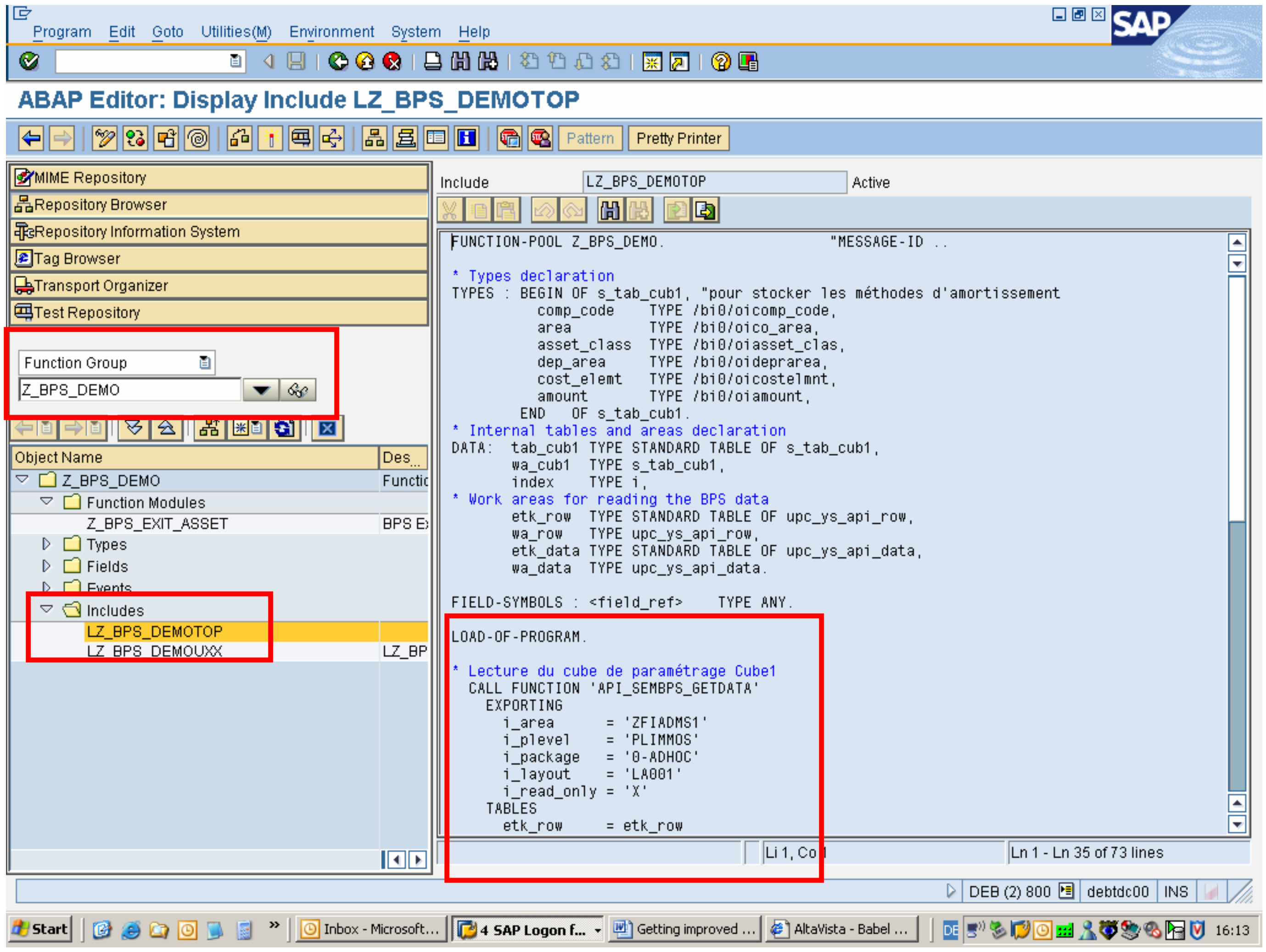
- add an item
- delete an item
- modify an item

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



Function Group
Z_BPS_DEMO

Object Name	Des...
Z_BPS_DEMO	Functio
Function Modules	
Z_BPS_EXIT_ASSET	BPS E
Types	
Fields	
Events	
Includes	
LZ_BPS_DEMOTOP	
LZ_BPS_DEMOUXX	LZ_BP

Include LZ_BPS_DEMOTOP Active

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

* Types declaration
TYPES : BEGIN OF s_tab_cub1, "pour stocker les méthodes d'amortissement
          comp_code    TYPE /bi0/oicomp_code,
          area         TYPE /bi0/oico_area,
          asset_class  TYPE /bi0/oiasset_clas,
          dep_area     TYPE /bi0/oideprarea,
          cost_elemt   TYPE /bi0/oicostelmnt,
          amount       TYPE /bi0/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
```

Li 1, Co 1 | Ln 1 - Ln 35 of 73 lines | DEB (2) 800 | debtdc00 | INS

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

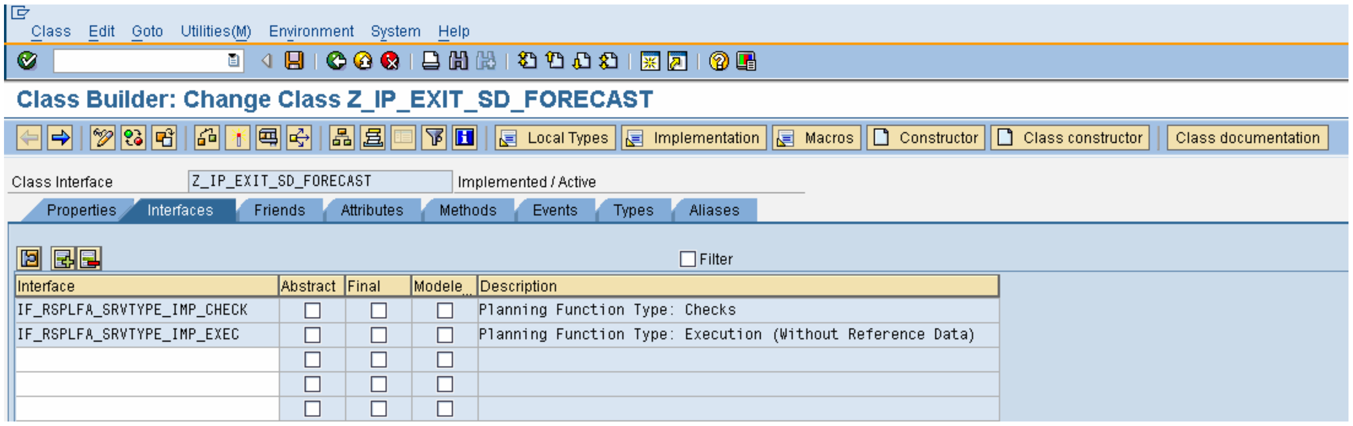
In BI 7.0, the enhancements are supported by the ABAP Object Oriented technology. The methods are still ABAP programs beginning with **Method** and ending with **Endmethod** as instructions. Such programs are not executable online, but are called by another ABAP program help to the instruction **Call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

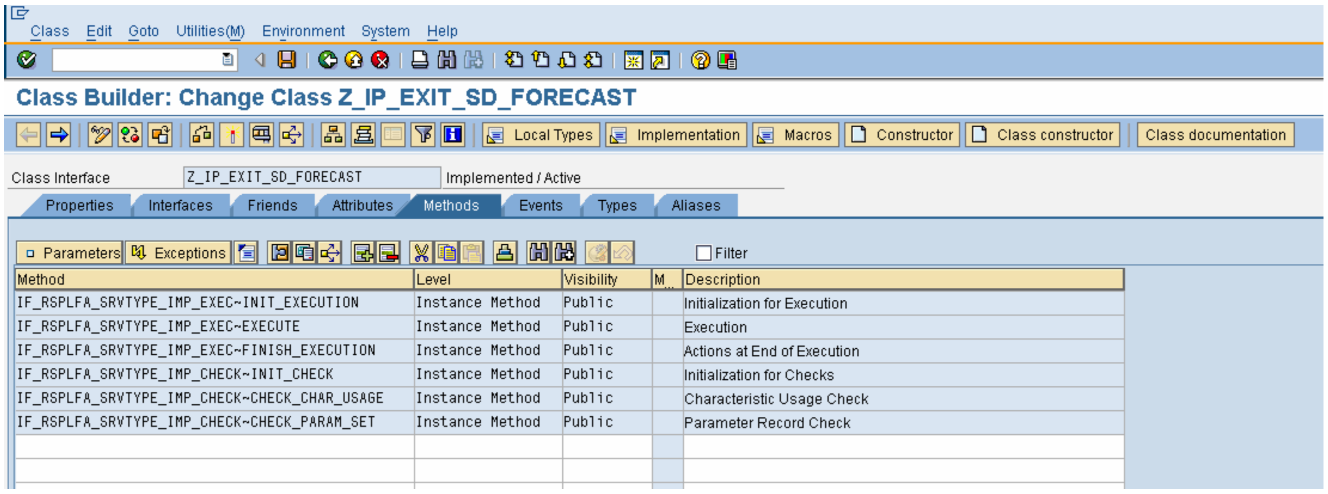


Class Builder: Change Class Z_IP_EXIT_SD_FORECAST

Class Interface Z_IP_EXIT_SD_FORECAST Implemented / Active

Interface	Abstract	Final	Modele...	Description
IF_RSPLFA_SRVTYPE_IMP_CHECK	☐	☐	☐	Planning Function Type: Checks
IF_RSPLFA_SRVTYPE_IMP_EXEC	☐	☐	☐	Planning Function Type: Execution (Without Reference Data)
	☐	☐	☐	
	☐	☐	☐	
	☐	☐	☐	

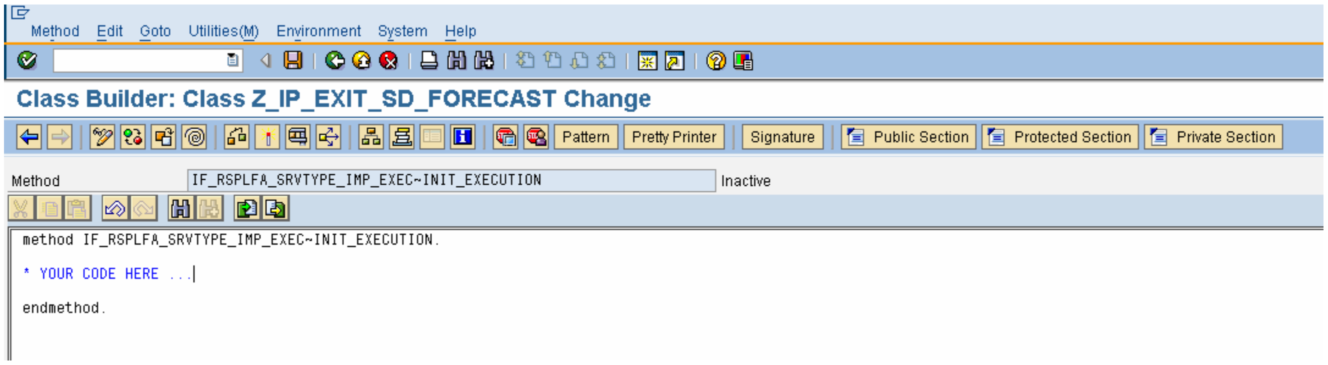
The descriptions of the methods are explicit enough to understand which one is to be used depending of the needs.



Class Interface Z_IP_EXIT_SD_FORECAST Implemented / Active

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.


Method Edit Goto Utilities(M) Environment System Help
Class Builder: Class Z_IP_EXIT_SD_FORECAST Change
Pattern
Pretty Printer
Signature
Public Section
Protected Section
Private Section

Method IF_RSPLFA_SRVTYPE_IMP_EXEC~INIT_EXECUTION Inactive

```
method IF_RSPLFA_SRVTYPE_IMP_EXEC~INIT_EXECUTION.

* YOUR CODE HERE ...

endmethod.
```

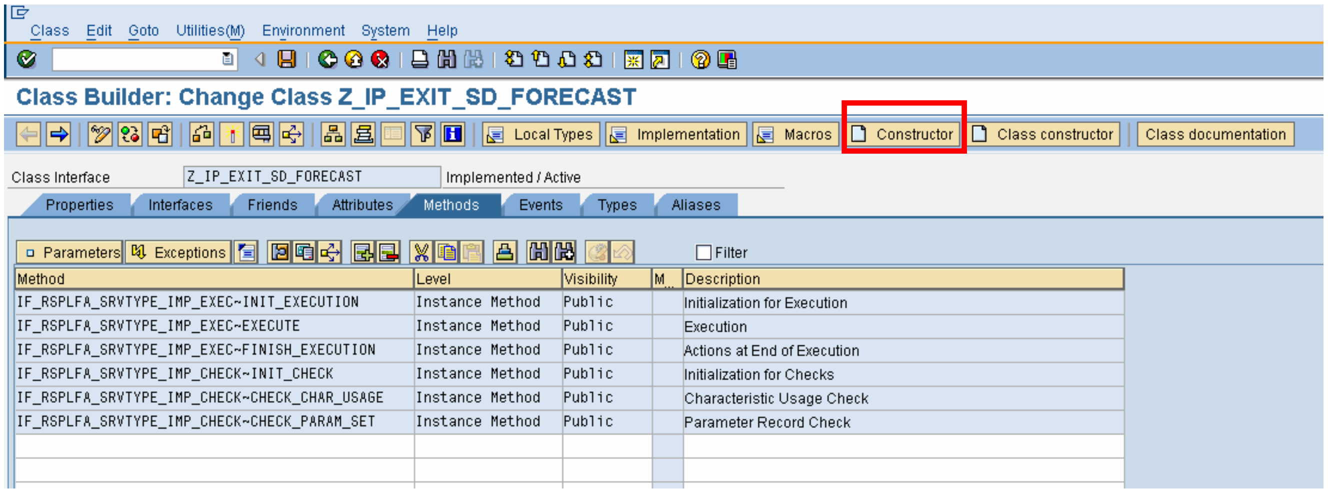
A new issue!!! What's wrong with that? (another topic)

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

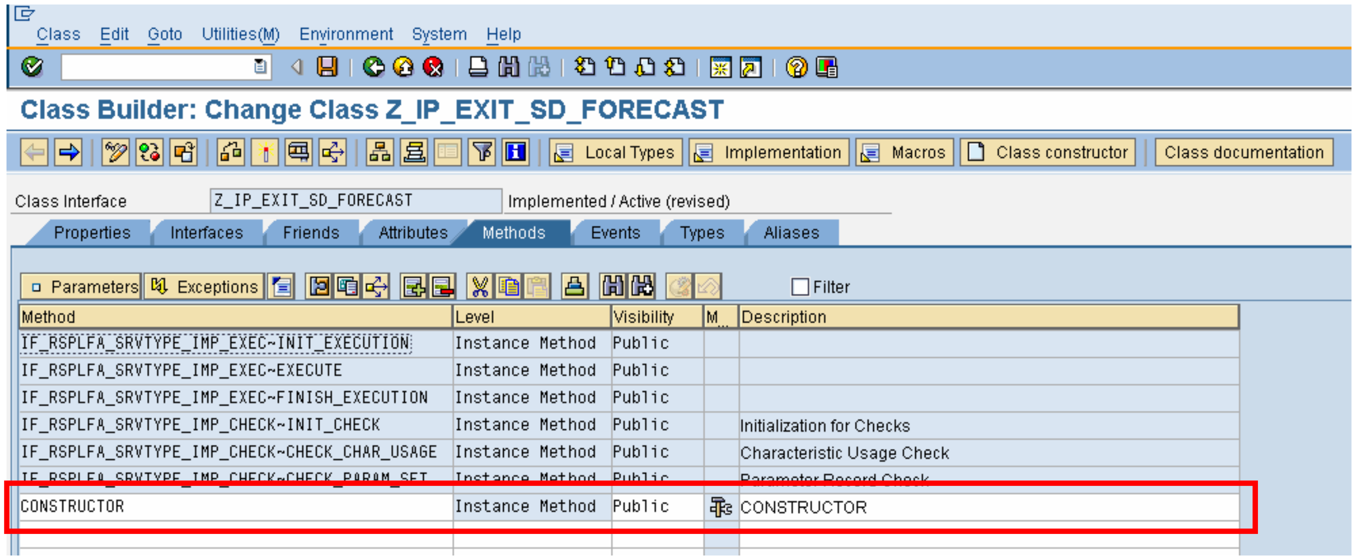
From the initial screen you can create the CONSTRUCTOR by clicking on the corresponding button.


Class Edit Goto Utilities(M) Environment System Help
Class Builder: Change Class Z_IP_EXIT_SD_FORECAST
Local Types
Implementation
Macros
Constructor
Class constructor
Class documentation
Properties Interfaces Friends Attributes Methods Events Types Aliases
Parameters
Exceptions
Filter

Class Interface Z_IP_EXIT_SD_FORECAST Implemented / Active

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

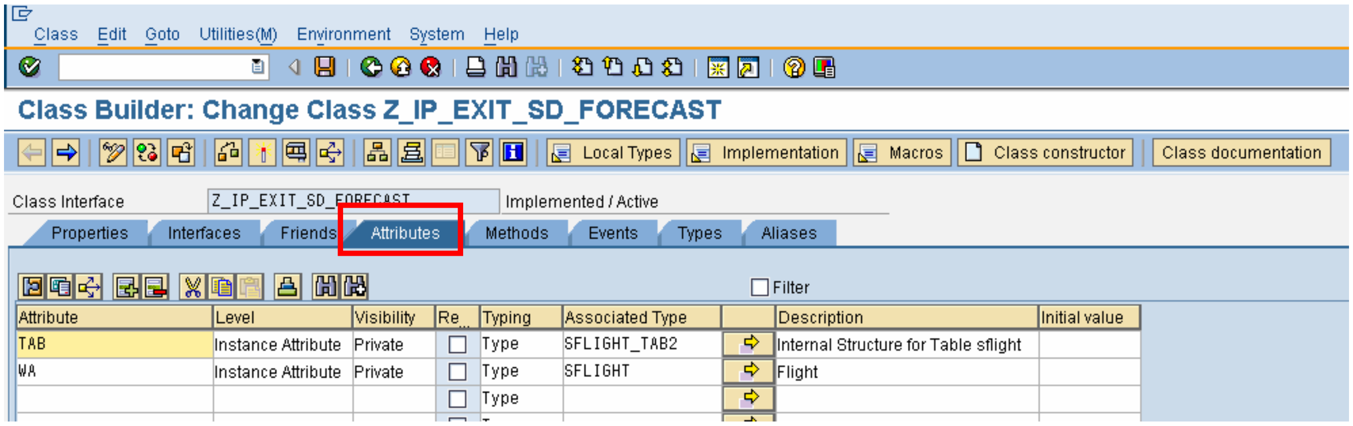
You get then a new method that can be coded for unique ABAP statements.


Class Edit Goto Utilities(M) Environment System Help
Class Builder: Change Class Z_IP_EXIT_SD_FORECAST
Local Types
Implementation
Macros
Class constructor
Class documentation
Properties Interfaces Friends Attributes Methods Events Types Aliases
Parameters
Exceptions
Filter

Class Interface Z_IP_EXIT_SD_FORECAST Implemented / Active (revised)

Method	Level	Visibility	M...	Description
IF_RSPLFA_SRVTYPE_IMP_EXEC~INIT_EXECUTION	Instance Method	Public		
IF_RSPLFA_SRVTYPE_IMP_EXEC~EXECUTE	Instance Method	Public		
IF_RSPLFA_SRVTYPE_IMP_EXEC~FINISH_EXECUTION	Instance Method	Public		
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
CONSTRUCTOR	Instance Method	Public		CONSTRUCTOR

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.

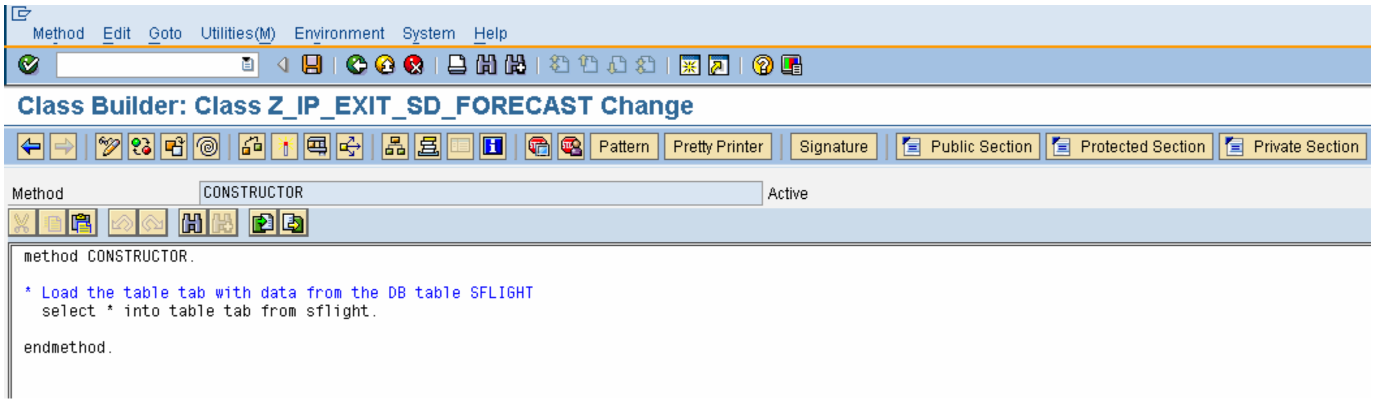


Class Builder: Change Class Z_IP_EXIT_SD_FORECAST

Class Interface Z_IP_EXIT_SD_FORECAST Implemented / Active

Attribute	Level	Visibility	Re...	Typing	Associated Type		Description	Initial value
TAB	Instance Attribute	Private	☐	Type	SFLIGHT_TAB2	⇨	Internal Structure for Table sflight	
WA	Instance Attribute	Private	☐	Type	SFLIGHT	⇨	Flight	
			☐	Type		⇨		

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



Class Builder: Class Z_IP_EXIT_SD_FORECAST Change

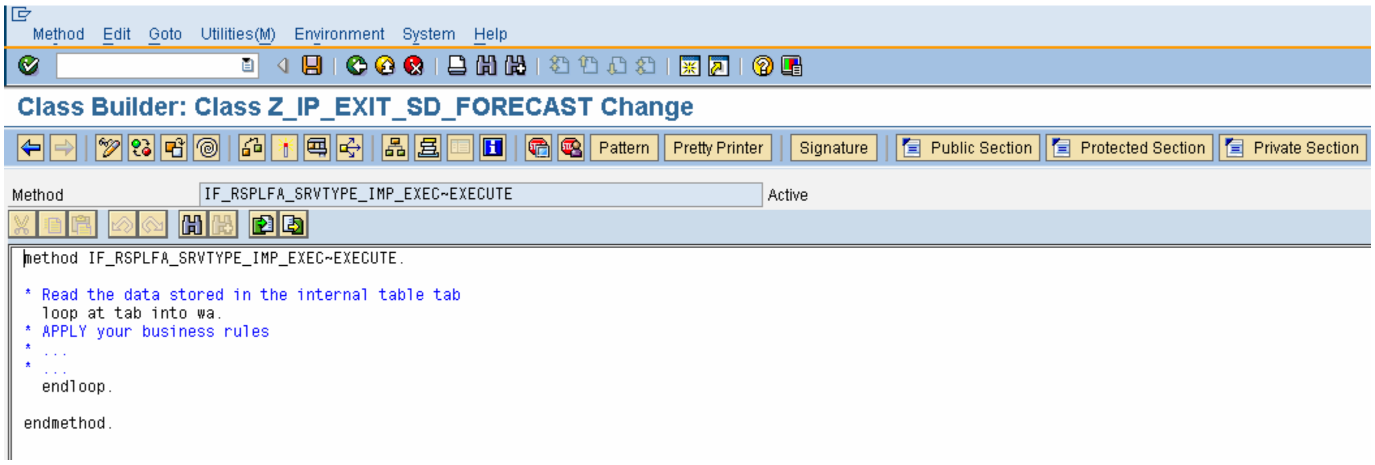
Method CONSTRUCTOR Active

```
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.


Method Edit Goto Utilities(M) Environment System Help
Class Builder: Class Z_IP_EXIT_SD_FORECAST Change
Pattern
Pretty Printer
Signature
Public Section
Protected Section
Private Section
Method
IF_RSPLFA_SRVTYPE_IMP_EXEC~EXECUTE
Active

```
method IF_RSPLFA_SRVTYPE_IMP_EXEC~EXECUTE.

* Read the data stored in the internal table tab
  loop at tab into wa.
* APPLY your business rules
* ...
* ...
  endloop.

endmethod.
```

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