How to... Log Changes in Plan Data Using DataStore Object

Applicable Releases:
SAP NetWeaver 7.01, SP6 and higher

IT Practice:
Business Information Management

IT Scenario:
Business Planning and Analytical Services

Version 1.0
Mai 2010
## Document History

<table>
<thead>
<tr>
<th>Document Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>First official release of this guide</td>
</tr>
</tbody>
</table>
### Typographic Conventions

<table>
<thead>
<tr>
<th>Type Style</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Example Text</strong></td>
<td>Words or characters quoted from the screen. These include field names, screen titles, pushbuttons labels, menu names, menu paths, and menu options. Cross-references to other documentation.</td>
</tr>
<tr>
<td><strong>Example text</strong></td>
<td>Emphasized words or phrases in body text, graphic titles, and table titles.</td>
</tr>
<tr>
<td><strong>Example text</strong></td>
<td>File and directory names and their paths, messages, names of variables and parameters, source text, and names of installation, upgrade and database tools.</td>
</tr>
<tr>
<td><strong>Example text</strong></td>
<td>User entry texts. These are words or characters that you enter in the system exactly as they appear in the documentation.</td>
</tr>
<tr>
<td><strong>&lt;Example text&gt;</strong></td>
<td>Variable user entry. Angle brackets indicate that you replace these words and characters with appropriate entries to make entries in the system.</td>
</tr>
<tr>
<td><strong>EXAMPLE TEXT</strong></td>
<td>Keys on the keyboard, for example, F2 or ENTER.</td>
</tr>
</tbody>
</table>

### Icons

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>🔴</td>
<td>Caution</td>
</tr>
<tr>
<td>🌐</td>
<td>Note or Important</td>
</tr>
<tr>
<td>🌐</td>
<td>Example</td>
</tr>
<tr>
<td>🗣️</td>
<td>Recommendation or Tip</td>
</tr>
</tbody>
</table>
Table of Contents

1. Scenario ........................................................................................................................................... 1
2. Background Information .................................................................................................................. 1
3. Prerequisites ..................................................................................................................................... 2
4. Step-by-Step Procedure ................................................................................................................... 3
   4.1 Create the Characteristics and the DataStore Object ................................................................. 3
   4.2 Create the Class Implementing the BAdI Logic ........................................................................... 6
   4.3 Create and Test the Enhancement Spot Implementation .............................................................. 7
   4.4 Test your Scenario ......................................................................................................................... 10
5. Appendix ........................................................................................................................................ 12
1. Scenario

In many planning project it is not only necessary to create new plan data but also to keep track how this new data originated. Thus a mechanism is necessary that logs all changes done in the plans data – either when new data is created or existing data is changed. Usually the logging information contains information about the user who has done the changes and the date and time when these changes where performed. In an existing How to Paper we have described and approach how such a log can be achieved by the use of derivations. Unfortunately there are some problems with this approach:

1. The derivation is called any time when data is written into the delta buffer. Thus if a record is changed several times in one planning session then several records will be created all bearing different time stamps. But usually a log entry should rather only be written when data is saved.

2. As the tracking information is kept in the InfoCube it is not possible anymore to compress the InfoCube as records with identical characteristic values (which could be compressed) have different values in the user/time stamp. Thus the read performance can be influenced very negatively.

With the new Logging BAdI introduced with SAP NetWeaver 7.0 EHP 1, SP6, it is possible to create log information when data is saved to the InfoCube. In the BAdI all necessary information about the changes is provided and it is up to the BAdI implementation to write the logging information to a suitable data target. This data target could be a normal data base table or a DataStore Object. Thus the logging information can be kept separately from the plan data in the InfoCube.

In this how to paper we will give a very generic example how this logging information can be written into a DataStore Objects. Except for the name of the DataStore Object and of the characteristics carrying the user name, the date, the time, and the so-called save-id all further necessary information is retrieved dynamically. Thus the coding can be easily adapted for a given customer situation.

By using a DataStore Object reporting the logging information is made very easy and it is also possible to use the logging information to erase wrong records using data warehousing methods.

2. Background Information

As already described above the new logging BAdI allows you to write into a data base table of your choice. This data base has most likely a different structure from the InfoCube itself. Thus the information from the changed data records must be filled into the structure of the target data base table (be it a standard data base table or a table underlying a DataStore Object). As the system needs to put the record information from one data structure to another anyway the BAdI is designed in such a way, that you provide the system with information about your target structure and the system will return the logging information already filled into your structure. This makes it very easy for you to update your data base table and is also better for performance.

The BAdI will return you the logging data filled with the DELTA values created by the user in the plan session. Using the BAdI you do not track the absolute value the user has entered or created by a planning function but the changes done to the data. Thus it is also possible to roll back changes done by the user by creating a suitable DTP (with reversed signs for the deltas) writing into the InfoCube.

It is important to note that the BAdI only tracks changes done in BW Integrated Planning. Data changes done via a DTP will not be tracked.

The BAdI gives you the opportunity to also write the request number, under which the changed record will be stored in the InfoCube, into the log as well as a so-called save-id. The save-id allows you to track changes in a consistent way even when you are writing back to several InfoCubes simultaneously: as usually the InfoCubes will have differing data structures you will have a DataStore Object for tracing the changes done to each individual InfoCube. Thus when you store data on multiple InfoCubes you will also distribute the log information over several DataStore Objects. The
system will provide you with an identification number that is unique in each save event. Thus all log entries created in one save event will receive the same save-id and it is possible to consolidate the log information even if it is kept in several DataStore Objects. Note that you do not need the save-id when you are saving to one InfoCube only.

Please note that in the BAdI you use the InfoCube name as a filter. That means that you can have several BAdI implementations at the same time. The system will decide by the InfoCube name which implementation (and thus which underlying class) will be used. The sample implementation that we give here is very generic. Just at the beginning of the methods in the implementing class we use a number of constants for the additional characteristics necessary for the tracking and set the name of the DataStore Object depending on the InfoCube name. You can easily adopt this coding in such a way that the same implementing class can be used for all InfoCubes where you want to log change information in a corresponding DataStore Object.

### 3. Prerequisites

The BAdI is delivered with EHP1, SP6. You can also have a look at SAP Note 1382767. If you need more information about the BAdI technology in general please have a look at the [documentation](#).
4. Step-by-Step Procedure

In our scenario we will use the full InfoCube information for the tracking of the changes. In addition we need fields in our DataStore Object that carry the information about the user, the data, the time, and the save-id. We only use one InfoCube and one DataStore Object in our scenario but want to show how to properly use the save-id in our example as well.

4.1 Create the Characteristics and the DataStore Object

1. Create a Save-ID

   Go to the transaction RSA1 and create a new characteristic. Have a look at the screen shot for the basic settings.

   ![Characteristics Screen Shot]

   The characteristic can be set to ‘with texts’ on the master data tab strip.

2. Create a Characteristic for the User

   Still in transaction RSA1 create a new characteristic for the user. Alternatively you can use a characteristic from the content.
3. Create a DataStore Object

Still in RSA1 go to the ‘InfoProvider’ section and create a new DataStore Object. As the key of the DataStore Object include all characteristics of the InfoCube for which you want to track the changes. If you like you can also choose a subset of the characteristics. Also add the characteristic for the user and the save-id as well as a characteristic for time (we use 0TIME) and the date (we use 0DATE).

As data field in the DataStore Object use the key figures of your InfoCube.
Now go to the entry ‘Settings’ and expand the node. Click on the button to change the type of the DataStore Object and select the option ‘Direct Update’.

You can now save and activate the DataStore Object.
4.2 Create the Class Implementing the BAdI Logic

1. Create the Implementing Class

   Go to the transaction se80 and create a new class. Choose an appropriate development class. Go to the tab 'Interfaces'. Enter the following interface: IF_BADI_INTERFACE and IF_RSPLS_LOGGING_ON_SAVE.

   ![Class Interface Table]

<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF_BADI_INTERFACE</td>
<td>Tag Interface for BAdIs</td>
</tr>
<tr>
<td>IF_RSPLS_LOGGING_ON_SAVE</td>
<td>Interface for BAdI: BADI_RSPLS_LOGGING_ON_SAVE</td>
</tr>
</tbody>
</table>

2. Implement the Method ‘IF_RSPLS_LOGGING_ON_SAVE~LOG_DEFINED’

   Go to the tab 'Methods'. You will see three methods here.
   The first method ‘IF_RSPLS_LOGGING_ON_SAVE~LOG_DEFINED’ is used to define whether for a given situation the logging should be switched on. Please note that in the definition of the enhancement implementation (see below) you have to specify whether for which InfoCube a given BAdI implementation should be called. Thus a BAdI implementation is called if and only if the InfoCube name is in the specified selection.

   In the given method you can specify whether say at the given moment or for the actual user a logging should be done. If the logging should be executed the method has to return a value for the exporting parameter r_log_defined. See the Appendix for a demo implementation of the method. Copy the demo implementation into your class and adapt it according to your needs.

3. Implement Method ‘IF_RSPLS_LOGGING_ON_SAVE~LOG_STRUCTURE’

   Now implement the second method ‘IF_RSPLS_LOGGING_ON_SAVE~LOG_STRUCTURE’. As already described above the idea of this method is that you tell the system how your target structure looks like at later the system will return you the data in this structure. In our case we want to write the data to our DataStore Object and will return information about its structure.

   Note that the method will be called for each InfoCube. In the method implementation we just rely on the name of the DataStore Object and of some of the used characteristics and retrieve all further necessary information dynamically.

   In order to adapt the coding to your needs you might have to replace the names of the characteristics at the beginning of the coding (see screen shot). Also you will have to adapt the case statement at the beginning of the coding to your InfoCube and DataStore Object name and then can reuse the rest of the coding as it is.
4. Implement Method ‘IF_RSPLS_LOGGING_ON_SAVE~LOG_WRITE’

In the last method we implement the actual writing to the DataStore Object. As most of the work is done by the system this method is quite simple to implement. Again just use the copy from the Appendix. Again you will have to adapt the name of the InfoCube(s) and the DataStore Object(s).

### 4.3 Create and Test the Enhancement Spot Implementation

1. Create a New Enhancement Implementation

   Go to the transaction se18. Enter the name of the enhancement spot (RSPLS_LOGGING_ON_SAVE) and press the button ‘Change’.

   ![Enhancement Spot](image)

   In the next screen choose the tab ‘Enhancement Implementations’. If there are already some implementations for the BAdI you will find them here.
From the menu choose the button for creating a new enhancement spot implementation.

On the next screen enter a name and a description of your BAdI implementation.

Now enter the name of the implementation and the name of the class containing your implementation.
2. Set the properties of the implementation

   In the next screen you can set the properties of your implementation. Please check that your implementation is set to active.

   Now expand the node with for your BAdI implementation and click on 'Filter Values'.
Now you can specify for which InfoCube your current BAdI implementation is called. For a new combination please press the button ‘Combination’.

By double clicking on the ‘????’ in the line you will get a popup where you can enter the selection for the InfoCube (use the name of your InfoCube).

Now save and activate your new implementation.

### 4.4 Test your Scenario

You can now test your scenario. Open a Web Layout or a BEx Analyzer Workbook.

Enter some new data. In our example we change the amount from 100 € to 300 €.
We press save and see a change with a delta of 200 € being tracked in our DataStore Object.
This appendix contains a sample implementation of the three methods necessary for implementing the BAdI.

**IF_RSPLS_LOGGING_ON_SAVE~LOG_DEFINED**

```plaintext
method IF_RSPLS_LOGGING_ON_SAVE~LOG_DEFINED.
  check
  sy-uname = 'TESTUSER'.

  r_log_defined = rs_c_true.
endmethod.
```

**IF_RSPLS_LOGGING_ON_SAVE~LOG_STRUCTURE**

```plaintext
method IF_RSPLS_LOGGING_ON_SAVE~LOG_STRUCTURE.
  data: l_s_map like LINE OF e_t_map,
    l_s_map_proposal type IF_RSPLS_LOGGING_ON_SAVE=>TN_S_MAP_PROPOSAL.
  data: l_dsonm type rsdodsobject,
    l_t_iobj TYPE STANDARD TABLE OF BAPI6116IO,
    l_s_iobj like line of l_t_iobj,
    l_struc_name type TABNAME,
    l_s_details type BAPI6108,
    l_s_return type BAPIRET2.

  * Please adapt the names of the characteristics to your situation
    constants: c_iobjnm_user type RSIOBJNM value 'ZUSER',
    c_iobjnm_time type RSIOBJNM value '0TIME',
    c_iobjnm_date type RSIOBJNM value '0DATE',
    c_iobjnm_saveid type RSIOBJNM value 'ZSAVEID'.

  * please enter the name of your InfoCube(s) and DataStore Object(s)
    case i_infocube_name.
      when 'ZTSC01003'.
        l_dsonm = 'ZHOW2LOG'.
      endcase.

  * get the name of the structure for the DataStore object
    CALL METHOD CL_RSD_ODSO=>GET_TABLNM
    EXPORTING
      I_ODSOBJECT = l_dsonm
    IMPORTING
      E_TABLNM = l_struc_name
    EXCEPTIONS
      NAME_ERROR = 1
      INPUT_INVALID = 2
      others = 3.
    IF SY-SUBRC <> 0.
      MESSAGE ID SY-MSGID TYPE SY-MSGTY NUMBER SY-MSGNO
      WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4.
```
ENDIF.

e_structure_name = l_struc_name.
clear e_t_map.

* get the structure from the dso
* get the list of info objects in the DataStore object
CALL FUNCTION 'BAPI_ODSO_GETDETAIL'
  EXPORTING
    OBJVERS = RS_C_OBJVERS-ACTIVE
    ODSOBJECT = l_dsonm
    TABLES
      INFOOBJECTS = l_t_iobj.

* we do not need the record mode (usually 0RECORDMODE)...
delete l_t_iobj where iobjtp = 'DPA'.
* also name, date, time, saveid, and requid come from the proposal
delete l_t_iobj where infoobject = c_iobjnm_user.
delete l_t_iobj where infoobject = c_iobjnm_date.
delete l_t_iobj where infoobject = c_iobjnm_time.
delete l_t_iobj where infoobject = c_iobjnm_saveid.

loop at l_t_iobj into l_s_iobj.
  l_s_map-iobj_name = l_s_iobj-infoobject.

* get the field name in the dso structure
  CALL FUNCTION 'BAPI_IOBJ_GETDETAIL'
    EXPORTING
      VERSION = RS_C_OBJVERS-ACTIVE
      INFOOBJECT = l_s_iobj-infoobject
    IMPORTING
      DETAILS = l_s_details
      RETURN = l_s_return.
  if l_s_return is initial.
    l_s_map-field_name = l_s_details-fieldnm.
    append l_s_map to e_t_map.
  endif.
endloop.

* get the user name
  read table i_t_map_proposal into l_s_map_proposal
    with key field_type = IF_RSPLS_LOGGING_ON_SAVE=>N_C_FIELD_TYPE-user.
  if sy-subrc is initial.
    l_s_map-iobj_name = l_s_map_proposal-iobj_name.

  CALL FUNCTION 'BAPI_IOBJ_GETDETAIL'
    EXPORTING
      VERSION = RS_C_OBJVERS-ACTIVE
      INFOOBJECT = c_iobjnm_user
IMPORTING
   DETAILS   = l_s_details
   RETURN    = l_s_return.

if l_s_return is initial.
   l_s_map-field_name = l_s_details-fieldnm.
   append l_s_map to e_t_map.
endif.
endif.

* get the date
read table i_t_map_proposal into l_s_map_proposal
   with key field_type = IF_RSPLS_LOGGING_ON_SAVE=>N_C_FIELD_TYPE-date.
if sy-subrc is initial.
   l_s_map-iobj_name = l_s_map_proposal-iobj_name.

  CALL FUNCTION 'BAPI_IOBJ_GETDETAIL'
     EXPORTING
       VERSION   = RS_C_OBJVERS-ACTIVE
       INFOOBJECT = c_iobjnm_date
     IMPORTING
       DETAILS   = l_s_details
       RETURN    = l_s_return.

if l_s_return is initial.
   l_s_map-field_name = l_s_details-fieldnm.
   append l_s_map to e_t_map.
endif.
endif.

* get the time
read table i_t_map_proposal into l_s_map_proposal
   with key field_type = IF_RSPLS_LOGGING_ON_SAVE=>N_C_FIELD_TYPE-time.
if sy-subrc is initial.
   l_s_map-iobj_name = l_s_map_proposal-iobj_name.

  CALL FUNCTION 'BAPI_IOBJ_GETDETAIL'
     EXPORTING
       VERSION   = RS_C_OBJVERS-ACTIVE
       INFOOBJECT = c_iobjnm_time
     IMPORTING
       DETAILS   = l_s_details
       RETURN    = l_s_return.

if l_s_return is initial.
   l_s_map-field_name = l_s_details-fieldnm.
   append l_s_map to e_t_map.
endif.
endif.

* The save id is important if you are using MultiCubes. All records sa
ved in
* one step (in different InfoCubes) will get the same saveid. If you are using
* only one InfoCube you do not need the saveid
* get the saveid
read table i_t_map_proposal into l_s_map_proposal
   with key field_type = IF_RSPLS_LOGGING_ON_SAVE=>N_C_FIELD_TYPE-
   saveid.
if sy-subrc is initial.
   l_s_map-iobj_name = l_s_map_proposal-iobj_name.
   CALL FUNCTION 'BAPI_IOBJ_GETDETAIL'
      EXPORTING
         VERSION = RS_C_OBJVERS-ACTIVE
         INFOOBJECT = c_iobjnm_saveid
      IMPORTING
         DETAILS = l_s_details
         RETURN = l_s_return.
   if l_s_return is initial.
      l_s_map-field_name = l_s_details-fieldnm.
      append l_s_map to e_t_map.
   endif.
endif.
endmethod.

IF_RSPLS_LOGGING_ON_SAVE~LOG_WRITE
method IF_RSPLS_LOGGING_ON_SAVE~LOG_WRITE.
* the log table has the same structure as the dso - insert
* it directly
   data: l_dsonm type RSDODSOBJECT.
   case i_infocube_name.
      when 'ZTSC0T003'.
         l_dsonm = 'ZHOW2LOG'.
   endcase.
   CALL FUNCTION 'RSDRI_ODSO_INSERT'
      EXPORTING
         I_ODSOBJECT = l_dsonm
         I_T_INSERT = i_t_logging_data
      * IMPORTING
      * E_RECORDS
      EXCEPTIONS
         DATA_TARGET_NOT_ODS = 1
         ODS_TYPE_NOT_TRANSACTIONAL = 2
         ACTIVE_TABLE_NAME_NOT_FOUND = 3
         RECORD_KEY_ALREADY_EXISTS = 4
         ARRAY_INSERT_FAILED = 5
         INTERNAL_ERROR = 6
         OTHERS = 7
IF SY-SUBRC <> 0.
  MESSAGE ID SY-MSGID TYPE sy-msgty NUMBER SY-MSGNO
      WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4.
ENDIF.

* The request ID is not written into the log table. If
* the request ID should be saved as well we recommend to
* do that in an additional data base table. The table
* should be built up with the following structure:
* request id (type rsrequnr)
* save id (char 32)
* infocube name (type tn_infocube_name).
* When using MultiProviders with more than one Realtime
* InfoCube then all records that are saved simultaneously
* hold the same save id. Via the table, the names of
* the corresponding InfoCubes can be found as well as
* the request into which the data was written.
endmethod.
www.sdn.sap.com/irj/sdn/howtoguides