How to... Build a Fast and Flexible Comment Solution for BEx Web Applications

Applicable Releases:
SAP NetWeaver BW 7.01 and higher, SPS9 for BI Java, see SAP note 1531740,
Note 1775737

Topic Area:
Business Information Management

Version 2.7
November 2015
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.
SAP NetWeaver “How-to” Guides are intended to simplify the product implementation. While specific product features and procedures typically are explained in a practical business context, it is not implied that those features and procedures are the only approach in solving a specific business problem using SAP NetWeaver. Should you wish to receive additional information, clarification or support, please refer to SAP Consulting.
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.

Disclaimer
Some components of this product are based on Java™. Any code change in these components may cause unpredictable and severe malfunctions and is therefore expressly prohibited, as is any decompilation of these components. Any Java™ Source Code delivered with this product is only to be used by SAP’s Support Services and may not be modified or altered in any way.
### 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>
| 1.10             | Some remarks added regarding the use in non-planning scenarios  
                     Code additions in BSP (OnInitialization, OnInputProcessing) |
| 2.00             | Strongly improved capabilities of the solution:  
                     Nearly no restrictions on query design anymore (no naming convention for key figures/structures necessary, restricted key figures are supported, structures in rows are supported, key figures in rows are supported, etc.)  
                     Handling of MultiProviders has been improved  
                     New Java Script coding for refresh (due to changed refresh behavior)  
                     Correction for conversion errors when using time characteristics  
                     Remark added regarding ‘State-free Navigation’ (stateless option for Web Template) |

#### Changed/new coding:
- **Report** `Z_CREATE_COMMENTS_TABLE`
- **Include** `LZ_COMMENTSTOP`
- **Function Module** `Z_RETURN_COMMENTS`
- **Function Module** `Z_COMNTS_FIELDS_CONVERT`
- **Form** `CONVERT_VALUE` (include `LZ_COMMENTSF01`)
- **Form** `REPLACE_STRUCTURES_VAR` (include `LZ_COMMENTSF01`)
- **Method** `IF_RS_BBS_BADI_HANDLER~CALL_URL`
- **BSP Page** `OnInitialization`
- **BSP Page** `OnInputProcessing`
- **XHTML for the Web Template**
- **Java script for refreshing the Web Template**
2.10 Some changes in coding to correct errors with offsets in variables and with compound characteristics in the drill down
Changes in handling of comment buffer to solve problems when using multiple servers (see chapter 3.2 below)
BPS Page for entering Comments is now called as a ‘modal popup’ (see chapter 4.6.5)
Change of default behavior for comments on totals – by default they are not allowed anymore but can be switched on: see chapters 2, 3.5, and 4.6.3, node 1744421 should be implemented for switching off totals in the web template

Changed/new objects:
Data elements ZGUID, ZDATE, ZRAWSTRING
Data base table ZCOM_BUF
Type Group ZCO
Form CONVERT_VALUE (include LZ_COMMENTSF01)
Form CORRECT_COMPOUND_CHARS (include LZ_COMMENTSF01)
Form COB_PRO_CMP_GET (include LZ_COMMENTSF01)
Form REPLACE_STRUCTURES_VAR (include LZ_COMMENTSF01)
Form GET_BUFFER_INSTANCE (include LZ_COMMENTSF01)
Form SET_BUFFER_INSTANCE (include LZ_COMMENTSF01)
Function Module Z_SAVE_COMMENTS
Function Module Z_COB_PRO_CMP_GET
Method IF_RS_BBS_BADI_HANDLER~CALL_URL
BSP page (Tab ‘Layout’)
XHTML for the Web Template (two new buttons)
Java script in Web Template

2.20 Implementation of Note 1775737 necessary in order to solve errors with compound characters
Minor code corrections (indepednt of the above note):
Form COB_PRO_CMP_GET (include LZ_COMMENTSF01)
Form CONVERT_VALUE (include LZ_COMMENTSF01)
Function Module Z_RETURN_COMMENTS

2.30 Error when using characteristics with blanks (in key), corrections in:
Form REPLACE_STRUCTURES_VAR (include LZ_COMMENTSF01)

2.40 Incompatible system changes with NW 7.40: new version of Function Module Z_RETURN_PARTPROV

2.50 Incompatible system change with NW 7.40: new command for screen refresh necessary in Web Template – see Java Script

2.60 Corrections in:
Function Module Z_RETURN_COMMENTS
Form CONVERT_VALUE (include LZ_COMMENTSF01)
2.70

Corrections in:
Method IF_RS_BBS_BADI_HANDLER~CALL_URL
Form CONVERT_VALUE (include LZ_COMMENTSF01)
Function Module Z_RETURN_COMMENTS
## 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><code>&lt;Example text&gt;</code></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 .................................................................................................................................................. 2
2. General Description of the Solution ........................................................................................................ 2
3. Create the Foundations in the System ..................................................................................................... 5
   3.1 Create the Mapping Table .................................................................................................................. 5
   3.2 Create the Dictionary Objects for the Buffer Handling ...................................................................... 5
   3.3 Create the Report for Generating the Comments Table ..................................................................... 7
   3.4 Create the Function Modules .............................................................................................................. 8
   3.5 Create the Type Group ....................................................................................................................... 8
   3.6 Create a BSP Application for Editing the Documents ......................................................................... 9
   3.7 Create a New RRI Type and Receiver ................................................................................................. 12
4. Implement a Working Example ............................................................................................................... 12
   4.1 Create the Comment Table ............................................................................................................... 12
   4.2 Create the Planning Function .......................................................................................................... 13
   4.3 Register the Planning Function Exit .................................................................................................. 13
   4.4 Create a Query ................................................................................................................................ 13
   4.5 Set up an RRI .................................................................................................................................... 14
   4.6 Create a Web Application .................................................................................................................. 16
      4.6.1 Data Provider and Analysis Item .................................................................................................. 16
      4.6.2 Parameters Comment Module: .................................................................................................. 16
      4.6.3 Parameters Command Module: ................................................................................................ 18
      4.6.4 Java Script for Refreshing and for Disabling Web Template when the Popup is shown ... .......... 19
      4.6.5 Buttons for Comment Services and for Refresh and Save ......................................................... 20
5. Testing and Trouble Shooting ................................................................................................................. 21
6. Appendix .................................................................................................................................................. 22
   6.1 Report Z_CREATE_COMMENTS_TABLE ...................................................................................... 22
   6.2 Top Include for Function Group Z_COMMENTS ............................................................................. 30
   6.3 The Function Modules ....................................................................................................................... 30
   6.4 Include LZ_COMMENTSF01 ............................................................................................................ 59
   6.5 Type Group ZCO ............................................................................................................................. 76
   6.6 BSP Page Layout Tab ......................................................................................................................... 76
   6.7 BSP Page OnInitInitialization ............................................................................................................ 79
   6.8 BSP Page ‘OnInputProcessing’ ......................................................................................................... 82
   6.9 BSP Page ‘OnDestroy’. ..................................................................................................................... 85
   6.10 IF_RS_BBS_BADI_HANDLER~CALL_URL .................................................................................... 85
   6.11 IF_RS_BBS_BADI_HANDLER~GET_TARGETS ................................................................................. 88
   6.12 XHTML for Web Application .......................................................................................................... 89
   6.13 Java Script for Web Application ..................................................................................................... 92
1. Scenario

When analyzing BW data or especially when planning new key figure values using BW Integrated Planning/Planning Applications Kit there is a need to enter some comments to explain the entered values or add some text information to key figures. In this paper we will describe a flexible yet simple to use solution for using comments in BEx Web Applications.

This is a sample screen shot of the solution we will build up (the button ‘refresh comments’ mentioned later in the text has been hidden):

<table>
<thead>
<tr>
<th>Product group</th>
<th>Product</th>
<th>Billed Quantity Plan</th>
<th>Sales Plan</th>
<th>Comment Sales Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accessories</td>
<td>Bottle fastener</td>
<td>121,255.314</td>
<td>343,664</td>
<td>Comment for bottle fastener</td>
</tr>
<tr>
<td></td>
<td>Business card case</td>
<td>12,121.000</td>
<td>400</td>
<td>Comment business card case</td>
</tr>
<tr>
<td></td>
<td>Candy tin</td>
<td>22,112.000</td>
<td>498</td>
<td>New comment for Candy</td>
</tr>
<tr>
<td></td>
<td>Coffee mug</td>
<td>5,971,993.868</td>
<td>11,153.420</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Result</td>
<td>6,127,482.162</td>
<td>11,497.582</td>
<td></td>
</tr>
<tr>
<td>Beg &amp; Outdoor</td>
<td>A4 Writing case</td>
<td>12,052.896.917</td>
<td>5,000</td>
<td>Comment for writing case</td>
</tr>
<tr>
<td></td>
<td>Automatic umbrella</td>
<td>13,753,535.790</td>
<td>1,000</td>
<td>second comment</td>
</tr>
<tr>
<td></td>
<td>Matchbook</td>
<td>15,517,386.942</td>
<td>2,000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Result</td>
<td>41,333,821.649</td>
<td>8,000</td>
<td></td>
</tr>
<tr>
<td>Office</td>
<td>Lamy pen</td>
<td>31,810,393.588</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mousepad</td>
<td>64,440,699.525</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Multifunctional pen</td>
<td>16,141,037.922</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Post-it Set</td>
<td>96,457,898.426</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Result</td>
<td>208,850,029.461</td>
<td>300</td>
<td>300,000</td>
</tr>
</tbody>
</table>

2. General Description of the Solution

The solution follows closely the ideas realized with the standard BW documents. There a document is stored against a characteristic/key figure combination. The documents information can be displayed as icons or – provided the document just contains text – in a column of an analysis item (Web application) by using the so-called document module.

In order to enable customers to build their own comment solution we provide a so-called comment module that calls an ABAP function module that receives the selection of the visible cells in the chosen column and that returns a list of comments. The comment module will then display the received comments in a specified column. In the ABAP function module a customer can create a comment solution tailored to the specific needs.

In this paper we describe one possible solution which is generic in such respect that it should be applicable to almost all BW standard scenarios. Please keep in mind that the solution is not a standard SAP solution but has to be seen as an example of a customer implementation.

Our solution replaces the standard documents by pure text comments that are stored in flat database table. The comments are also stored against a characteristic/key figure combination and it is also possible to decide, which characteristics of the InfoCube should be used for the comments as well. As
the comments are not stored against the query name the comments are available in all queries accessing data from the used InfoCube.

**Important**

In our solution we store comments directly on Basic/Realtime InfoCubes.

If a query on a MultiProvider is used then the system tries to access the comments from the underlying single InfoProviders. Thus you have to fill the field 0INFOPROV in the field assignments of the report report interface. The InfoProvider has to be restricted in such a way that it is unique for the entire comment column.

If it is not possible to restrict the InfoProvider (for example you are using physical partitioned InfoCubes) you also have to modify the coding and fill in the name of the proper basic InfoCube when you receive the cell selection in the method IF_RS_BBS_BADI_HANDLER~CALL_URL and in function module Z_RETURN_COMMENTS.

With our solution it is also possible to create comments on totals. In previous versions of this how to paper the creation of comments on totals was possible by default. Nevertheless comments on totals can cause some problems. If there are no restrictions in the given query for a totaled characteristic then a comment on the total can is equivalent to a comment on a (existing) record on another level of aggregation (not using the totaled characteristic). If there are restrictions on that characteristic the comment does not correspond to a record on any aggregation level. As the comment solution also does not use the aggregation level to full extend there might be problems to distinguish comments on totals and comments on the characteristic value ‘#’.

**Important**

With version 2.10 of this how to guide we have changed the default behavior and have switched off the possibility to create comments on (sub-) totals. This is the recommended settings. If you do want to use comments on totals please refer to chapter 3.5 (Create the Type Group) and chapter 4.6.3 (Parameters of Command Module).

If you use comments on totals please note that we have decided against doing some sort of aggregation (like concatenation) of comments. If say the query shows product groups and products in the drill down and a comment is entered on a subtotal over all products in that product group then the comment will be stored against this product group and the product will be left empty.

As comments are often used in plan scenarios we have realized a comment buffer. When a comment is created then it is not stored in the data base at once but in a user specific buffer (more correctly spoken there is one comment buffer per log on and user). In order to store the comments a save has to be triggered. Thus we follow the logic of the delta buffer in BW Integrated Planning/Planning Applications Kit.

The comment solution can also be used in a pure reporting scenario. In this case no planning function for saving the comments to the data base table is necessary – comments can be directly saved on the popup where comments are entered or changed. In order to implement this behavior the logic of the BSP application has to be changed. Please see chapter 3.6 and 4.6.3 for details.

We do not have realized a specific authorization or user locking concept. A comment is always displayed in a query table in a column defined by a key figure. Thus we can use the BW reporting authorizations as authorizations for the comments – if the user has the authorization to display/change a certain key figure than he can also display/change the comment. Similarly we can switch off the link to the application (pop up) used to change comments if the underlying (input enabled) query is locked due to some user locking conflict (see below).
Technically the solution consists of an ABAP layer that is providing the storage location, the comments buffer and services to read, create, change, or delete comments. This ABAP service layer is called from different consumers: the comments are requested and displayed by the comment module. Comments are created, changed, and deleted via a BSP application that is called once a cell is clicked. The link between the BSP application and the Web application is realized via the so-called command module and the report-report interface. The saving of comments is triggered by a planning function (more precisely the Exit for modifying the selection of a planning function) which in its turn is started by a command from the web application.

In this paper we provide you the coding necessary to realize the described solution. We also explain how the objects, exit, or enhancements are to be implemented. Further we explain how a concrete example using the solution can be built up and also include the XHTML for such an example. As we using a flat data base table it is quite easy to create additional services like delete or copy functions for comments. Please be aware that we are using a special key for the table which has to be adapted accordingly (see chapter Create Comment Table).

In our solution we try to be generic and do not use any characteristic names. If you want to call our function modules you can create specific ABAP structures with your characteristics and use those structures instead of our generic approach.

**Important**

In our solution we can only have comment columns but not comment rows. You can have more than one comment column per query. If you are using MultiProviders then the underlying basic InfoProvider (InfoCube or Aggregation Level) must be unique within each comment column. If you are using restricted key figures the characteristics have to be restricted to single values.

In older versions of this how to paper we could not handle restricted key figures or structures in the rows. We also required some fixed names/prefixes for the structure containing the key figure and for the key figures themselves. In the current version of this how to paper all these restrictions have been removed.
removed. Thus nearly any BW query can be used in our comment solution. The new version of this how to paper is downward compatible – web applications built according to older versions of this how to paper will also run with this version.

Important
Comments can only be created on existing records. No comments are possible on new lines.

3. Create the Foundations in the System

In this chapter we describe how the necessary underlying objects are created in the backend. The source code can be found in the appendix.

Important
In some objects manual adjustments might be necessary. In the following chapters we explain the functionality of the different objects. If a manual adjustment might be required we will mark it.

3.1 Create the Mapping Table

You can store comments against any basic InfoCube. If you are working in a Multi Provider than the comments will be stored against the basic data provider the key figure cell is based on. We follow here closely the concept used in Integrated Planning/Planning Applications Kit – also in planning a key figure can only be stored in a single (Realtime-) InfoCube. If the single InfoCube cannot be determined from the cell selection a comment cannot be stored.

In our solution we will use one comment table per InfoCube. We need a table the holds the information which comment table belongs to which InfoCube:

Create a database table called ZMAP_IPROV_COM. The table should contain the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Key</th>
<th>Initial</th>
<th>Data element</th>
<th>Data Type</th>
<th>Length</th>
<th>Decimal</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLIENT</td>
<td>☑</td>
<td></td>
<td>Mandt</td>
<td>CLNT</td>
<td>3</td>
<td>0</td>
<td>0 Client</td>
</tr>
<tr>
<td>INFO_PROV</td>
<td>☑</td>
<td></td>
<td>RSINFOPROV</td>
<td>CHAR</td>
<td>36</td>
<td>0</td>
<td>0 infoProvider</td>
</tr>
<tr>
<td>COMMENTS_TAB</td>
<td></td>
<td>☑</td>
<td>TABLENAME16</td>
<td>CHAR</td>
<td>16</td>
<td>0</td>
<td>0 Table name, 16 characters</td>
</tr>
</tbody>
</table>

In case you want to check which comments table is used for a specific InfoCube please have a look at this database table.

3.2 Create the Dictionary Objects for the Buffer Handling

In older Versions of this How to Paper we used the shared buffer/memory to hold the comment buffer. This leads to errors when using multiple servers as the shared buffer is not shared across application servers. Thus we temporarily store the comment buffer in the database. As we do not have a ‘clean up event’ in the application the buffer stays on the database after closing the application. As it is session specific this will cause no problems in the application. In order to keep the used space on the database low we automatically delete buffer instances that are older than two days from the database whenever comments are stored.

First create the necessary data elements:

<table>
<thead>
<tr>
<th>Data Element</th>
<th>Description</th>
<th>Data Type</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZGUID</td>
<td>SYSUUUID_C</td>
<td>CHAR</td>
<td>32</td>
</tr>
</tbody>
</table>
For all three data elements we use a predefined data type as shown in the screen shot:

Now create the data base table ZCOM_BUF for holding the comment buffer. Follow the information given in the screen shots:
As delivery class choose ‘A’. If you are asked for an enhancement category then choose ‘cannot be enhanced’.

### 3.3 Create the Report for Generating the Comments Table

The comments table mirrors the structure of the corresponding InfoCube. As creating the comments table may be a bit time consuming we have included a report that generates the comments table from the InfoCube for you.

Go to se38 and create a new report (without top include) and call it Z_CREATE_COMMENTS_TABLE (see Appendix for coding). When you start the report you are prompted for the name of an InfoCube and the name of the comment table. The report:

- Checks whether for the chosen InfoCube there is already a comment table maintained. If so it stops the processing with an error message.
- Checks whether the chosen InfoCube is a basic InfoCube.
- Reads the definition of the InfoCube and creates a comment table containing:
  - A technical table key,
  - Every characteristic contained in the InfoCube (we cannot use the characteristics as the key to the comments table as a data base table cannot have more than 16 key fields),
  - One field for the key figure,
  - A field of type string for holding the comment.
- Fills the mapping table ZMAP_IPROV_COM.
3.4 Create the Function Modules

Create a new function group (Z_COMMENTS). The function group has to have a top include. Please copy the coding for the top include from the appendix.

Within the function group create all function modules from the appendix. All function modules can be created a standard function module with one exception: the function module Z_RETURN_COMMENTS has to be created as a remote-enabled function module (see attributes tab).

<table>
<thead>
<tr>
<th>Function Group</th>
<th>Z_COMMENTS</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short Text</td>
<td>Returns a list of comments to the Java module</td>
<td></td>
</tr>
</tbody>
</table>

The function module Z_SEL_EXIT is used for calling the save function module. It is thus not an integral part of the comments layer but a service control the comments solution. The function module is an exit that is called when a planning function is executed. We will create a dummy planning function to trigger the exit that then triggers the saving of the comments.

**Important**

If you do not use the same name for the planning function as we will (Z_DUMMY_SAVE) you will have to adapt the coding of the function module.

In the function group create a new include (e.g. LZ_COMMENTSF01) and add the coding from the appendix.

3.5 Create the Type Group

Create the type group ZCO from the coding in the appendix. In version 2.10 of this paper we have introduced two switches necessary for setting a correct behavior for comments on totals: zco_use_hash and zco_no_totals. With the default settings used below no comments on totals are possible.

If you have used the solution before and want to have the same behavior then you have to set default values of the variables to:

```text
Zco_use_hash value ' '.
Zco_no_totals value ' '.
```

Unfortunately this setting cannot distinguish between comments stored against a total and comments stored on a characteristic value '#'. Both will be stored against an empty value on the data base. This is why we had to change the solution and we do NOT recommend these settings.

If you want to use comments of totals then you should set the following values:

```text
Zco_use_hash value 'X'.
Zco_no_totals value ' '.
```

In order to distinguish between a comment on a total and a comment on '#' we actually store comments on the value '#' on the data base. Please be advised that two problems can arise: for time characteristics we cannot set the value '#' on the data base so also with this setting you cannot distinguish comments on totals and on '#' for time characteristics.
Also a comment on a total is not represented by an actual data record. In our new paper ‘How to… Build a Planning Function that Deletes Comments in BW’ ([https://scn.sap.com/docs/DOC-30197](https://scn.sap.com/docs/DOC-30197)) we delete records and the corresponding comments. If there is no record for a comment then this solution does not delete the comment.

Please also have a look at chapter 4.6.3 for new settings for totals in the web template.

### 3.6 Create a BSP Application for Editing the Documents

In order to create, change, or delete comments we use a BSP application. In [SE80](https://scn.sap.com/docs/DOC-30197) create a new BSP application called Z_COMMENTS. Within the BSP application you create a BSP page ‘with flow logic’ and call it z_edit_comments.htm.

On the properties tab of the page use the following settings:
Now change to the tab ‘layout’ and paste the coding from the appendix.

```javascript
<%@ page language="abap" %>
<%@extension name="htmlb" prefix="htmlb" %>
<head>
<script language="Javascript">
    function changeScreenSize(w,h)
    {
        window.resizeTo( w,h );
    }
    function delete_comment()
    {
        if (window.confirm("Are you sure you want to delete the comment?"))
        {
            window.open("z_edit_comments.htm?delete_flag=X","edit_comment");
            id = "BUTTON_GROUP_ITEM_1_btn0_acButton";
            window.parent.opener.document.getElementById(id).click();
            window.popup=window.open("","edit_comment");
            window.popup.close();
        }
    }
</script>
</head>
</body>
</html>
```

Now continue to the ‘Event Handler’ tab. Use the drop down box to navigate through the different events and implement the event ‘OnInitialization’, ‘OnInputProcessing’, and ‘OnDestroy’ from the appendix.
If you want to use the solution in a pure reporting scenario the comments should be stored or deleted once the BSP application is left. We would recommend the following two changes:

1. In the BSP application navigate to the page 'z_edit_comments.htm' On the ‘Layout’ tab please change the definition of the ‘Accept’ button from

```
<htmlb:button id = "ACCEPT"
text = "Accept Changes"
onClick = "ACCEPT"
onClientClick = "javascript:accept()"
tooltip = "Accept changes and close the window" />
```

to:

```
<htmlb:button id = "ACCEPT"
text = "Save Changes"
onClick = "ACCEPT"
onClientClick = "javascript:accept()"
tooltip = "Save changes and close the window" />
```

Also change the ‘Cancel’ button from

```
<htmlb:button id = "CANCEL"
text = "Cancel"
onClientClick = "javascript:cancel()"
tooltip = "Cancel without accepting the changes" />
```

to:

```
<htmlb:button id = "CANCEL"
text = "Cancel"
onClientClick = "javascript:cancel()"
tooltip = "Cancel without saving the changes" />
```

2. In the event handlers for the events ‘OnInitialization’ and ‘OnInputProcessing’ we already have placed some sample coding where the save function module is called right after writing the comment to the buffer or deleting the comment. You just have to un-comment the ABAP coding in order to use it.

On the ‘Page Attributes’ tab create some attributes as shown in the screen shot below.

Finally on the ‘Type Definitions’ tab enter a reference to the type group created above:
3.7 Create a New RRI Type and Receiver

In our solution we will start the BSP for editing the comment by clicking the cell. We use the Analysis Item Modification called ‘command module’ to trigger the url as a receiver in the report-report interface. The cell selection is also transmitted to the BSP application via RRI. Therefore we need an own report type as a receiver. Please follow the instructions from the documentation to create the new type.

http://help.sap.com/saphelp_nw70ehp1/helpdata/en/45/e5ab04810473b1e10000000a11466f/frameset.htm

The class implementing the interface IF_RS_BBS_BADI_HANDLER (implemented in the first step in the documentation) contains the information about the jump target and handles the transmission of the information. We will use the url of the BSP application as a jump target. As we do not need a batch target or a SAPGUI target in this interface type we only have to implement the methods IF_RS_BBS_BADI_HANDLER~CALL_URL and IF_RS_BBS_BADI_HANDLER~GET_TARGETS. You will find the coding again in the appendix.

Important
Please make sure that the url of the BSP is correct. You will not have to care about the protocol, server name, and port in the url as this will be filled by the system. You will find the url of the BSP application at the bottom of the properties tab of the BSP page.

Important
In this implementation we call the BSP application with the protocol ‘http’ (NOT ‘https’). We need the BSP application to trigger a refresh of the Web application once a comment has been changed and this is only working correctly when the Web template and the BSP application are using the same protocol. If you are using ‘https’ then you have to adapt the coding in the method IF_RS_BBS_BADI_HANDLER~CALL_URL.

4. Implement a Working Example

We now can create a working example in the system.

4.1 Create the Comment Table

As a first step we create the comment table itself. Call the report Z_CREATE_COMMENTS_TABLE, enter the name of your basic InfoCube and a name for the comment table. The system will create the comment table and will register the comment table in the mapping table ZMAP_IPROV_COM automatically. If you want to use a new (different) comments table for an InfoCube simply delete the corresponding entry from the mapping table and create the new comment table. The mapping table will again be updated automatically.

If you do not want to use all characteristics contained in the InfoCube for the comments as well you can simply change the comments table and delete those characteristics from the table. This concept is similar to the standard documents in BW where you can set a characteristic as document relevant or not relevant. The system will detect dynamically which characteristics are used in the comments table and thus the set of InfoObject fields in the mapping table can be a subset of the InfoObject fields in the InfoCube.

You should modify the comments table BEFORE any data is contained in the comments table. The reason therefore is the use of the technical table key.

The technical table key is a hashed value of the characteristic values in the comments table. By using this technique we obtain a unique key for every comment entry and we can use this hash value for an optimized access to the single comment entries.
Important

If you change the structure of the comment table when there are already entries in the table you have to adapt the hash key!

You have to run a program that goes through every record, calculates the hash key, and replaces the table entry with the one with the new hash key. Also if you create a program/planning function that is inserting comments into the comments table (copy function, repost etc.) you have to make sure that the hash key is filled correctly. You can use the function module Z_CREATE_HASHKEY to create the hash key. Make sure that all characteristic values are converted to the INTERNAL format before creating the hash key. You can use the function module RSPLSI_INPUT_CONVERT to do the conversion.

The function module Z_CREATE_HASHKEY has to be called with a structure containing all characteristic values the comment is based on, a field for the hash key, and a field for the key figure. If you know which comment table you are working on you can simply use the underlying structure of your comments table. If you want to create some generic function you have to create a structure and move the characteristic value into the structure using field symbols and assign statement. The structure information can be obtained from the function module Z_CREATE_DATAREF with the structure name zco_ty_comments_buf. Please have a look at function module ZUPDATE_COMMENTS in order to see how the function modules are used.

4.2 Create the Planning Function

Note

If you want to use our solution in a pure reporting scenario and have implemented the recommendations in 3.5 then you do not need to perform step 4.2 and 4.3. You can continue directly with 4.4

Create a dummy planning function (Z_DUMMY_SAVE) which does not change any data (e.g. revaluation by 0 percent). We will use a set of data already selected in the query such that no new data is selected from the database. It is also possible to create a package that returns an empty selection. It has to be taken into account that the selection must be chosen in such a way that no two users lock each other when executing the dummy function. We will also execute the planning function in delta mode to further reduce the data selection time.

4.3 Register the Planning Function Exit.

As already mentioned above we use the exit for changing the selection for a planning function for triggering the saving of the comments. The exit is only called if it has been registered in the system. Please go ahead and follow the instructions given in the SAP Note 1101726 and register the function module Z_SEL_EXIT by using the report SAP_RSADMIN_MAINTAIN as described in the note. Please note that this function module now will be called for all selections. Thus the check statement that is checking the name of the function module in the beginning is very important.

4.4 Create a Query

We now create a query that displays some key figures as well as the dummy column for the comments in the columns. The query needs to comply with certain restrictions.

Restriction 1

If you use a restricted key figure for the comment column(s) then each characteristic used in the key figure must be restricted to a single value. Variables can be used for the restriction.
Restriction 2
Within a comment column the InfoCube must be restricted to a single basic InfoCube or an aggregation level on a Realtime InfoCube. This InfoCube must be the same for all cells in the comment column.

Restriction 3
Currently you cannot have a characteristic drilled down in the columns but you can only use a fixed number of columns. [We try to eliminate this restriction in the future.]

Important
In older versions of this paper a certain name for the structure containing the key figures was required. These restrictions do not hold anymore. Also the dummy key figure does not have to comply to any naming conventions any more.

Create a key figure that should be commented (in our case: 0D_INV_PLAN/Sales Plan).

Next to the column create another column using the same key figure or the same selection. This column will be used as a place holder for the comments. When using the query in the web application the selection from this column will be used to select the comments and the column itself overwritten by the comments. The 'comment key figure' does not have to be input enabled so it is easier to use it as a standard non input enabled key figure.

You can also use restricted key figures (even with variables), use a drill down for further characteristics in the columns or even use the key figures in the rows.

4.5 Set up an RRI
Call the transaction RSBBS. Enter the name of the query you just created. Create a new sender/receiver assignment using the special type of RRI you have created in your enhancement. In our case we have called the new type 'Modify Comment', our new receiver type is called 'COMMENT'. Please make sure you fill the receiver object by using the value help on the field 'Receiver Object'.
No choose ‘Assignment Details’. For each characteristic used in the query and used in the comments set up a field assignment. Technically we only need the assignment for the characteristics used in the comments table (and in the case of MultiProvider also the field 0INFOPROV) but it is safer to just map all characteristics contained in the query. Unnecessary characteristics will be ignored when reading the comments. The sender information is filled by the system automatically. Choose ‘URL Parameter’ as type. In the field ‘Field Name’ enter the technical name of the characteristic (see below).
Save the new sender/receiver assignment.
In the case you are using a MultiProvider please map the sender object ‘InfoProvider’ to the field name ‘0INFOPROV’.

### 4.6 Create a Web Application

Start the Web Application Designer and create a new web template. In the appendix you can find an example of the described web application. You can change to the XHTML tab and paste the coding. As the Java Script is not contained directly in the XHTML you will have to open the Java Script item and enter the Java Script from the appendix. In order to use the example you have to adapt the names of the query, planning function etc.

We now describe step by step what is contained in the Web application.

#### Restriction

The Web Template cannot run in a stateless mode. Thus the Web template parameter ‘Status-free Navigation’ has to be set to ‘off’ (which is the default value).

### 4.6.1 Data Provider and Analysis Item

Create a data provider using your query. Create an analysis item using this data provider. In the analysis item use two modules – the command module in order to trigger the RRI (and thus the BSP to create/change/delete comments) and the comment module that displays the comments. For the correct settings of the modules please copy the settings from the sample web application. We will have a look at some of the parameter settings that you might have to change.

### 4.6.2 Parameters Comment Module:
COLUMN_5: With this parameter we specify that we display the comments in the fixed column 5. Here also the key columns are taken into account – in our example we have two key columns and three data columns. Thus we display the comments in the last column. Instead of using a fixed column you can use:
- a specific index (as done here)
- specific characteristic
- specific characteristic value
- specific attribute
- specific structure member (key figure or characteristic value)

For an example how these options can be used please see the documentation for modification module ColumnWidth
http://help.sap.com/saphelp_nw70ehp1/helpdata/en/47/a0623a12753377e10000000a421937/frameset.htm

If you want to display the comment column in every column containing a certain key figure (structure member) then you just use the parameter ‘COLUMN’ and enter the name of the structure member. You will find this name in the query designer on the tab ‘Extended’.

In each column containing the specified structure member we will display a comment column. If you now place additional characteristics in the columns (say the year) then you would see a comment column for every year in the columns. The module is automatically generating several comment columns.

Hint

If you want to see several comment columns in FIXED positions you can use several comment modules (and command modules, see below) with each having a fixed column. If you want to comment several (different) structure members you also have to use a comment module for each of these structure members.

ABAP_FUNCTION_MODULE: This parameter contains the name of the ABAP function module returning the comments. In our case it is ‘Z_RETURN_COMMENTS’. You do not have to change this setting unless you want to create your own function modules.
How To... Build a Fast and Flexible Comment Solution for BEx Web Applications

ALIGMENT: You can specify how the text of the comment is to be aligned horizontally. Possible values are: LEFT, RIGHT, and CENTER.

VERTICAL_ALIGNMENT_OTHER: You can specify the vertical alignment. Possible values are: TOP, BOTTOM, and CENTER.

MAX_ROW_LENGTH: Setting for the maximal row length.

TOOLTIP: By default the tooltip is switched on. Please do not switch off the tooltip if you use the parameter MAX_ROW_NUMBER (see below).

WRAPPING: This parameter specifies whether a word wrapping is to be done if a maximum row length is set.

MAX_ROW_NUMBER: You can use this parameter to specify how many rows of a comment will be shown in the Analysis Item. If the comment has more rows than specified here then the entire comment will be visible in the tooltip and in the BSP for editing the comment. Thus you can make sure that the table does not grow too large if you use multi lines comments. In our example we have set the number of rows to 4.

4.6.3 Parameters Command Module:

ACTION_ON_CLICK: with the current setting the command module will call the RRI upon cell click. In the parameters of the command you have to set the correct data provider (in our case it is set to DP_1, the only Data Provider contained in the Web Template) and the RRI receiver. You will find the name of the RRI receiver in the transaction RSBBS in the settings for your query:
Customizing command:

**ACTION_IN_CONTENT_VISIBLE**: This parameter specifies whether the command can only be triggered from a cell in the data area (as in our case) or also from a column heading or row entry.

**COLUMN_5**: With this parameter we specify that a click in column 5 (the column we use for the documents) we trigger the RRI and thus the BSP to edit the comments (see above for additional options).

**ONLY_DATA_ENTRY_ENABLED**: This setting specifies whether the command is triggered in any case or only if the underlying data provider is enabled for data entry. In our case the setting is switched on. The setting mirrors the user lock behavior of the key figures: if you are using a plan queries and another user is currently working on the same selection then (the data provider for) your plan query will be switched input off automatically. If the parameter is set in the module then in this case the RRI and thus the BSP to change the comments cannot be triggered. Thus we can change comments if and only if we can change the corresponding key figures and we have realized a user locking concept for the comments. If you are not working with plan queries this parameter is less important.

**Important**

In the attached Web template the setting 'ONLY_DATA_ENTRY_ENABLED' is set. If you use our example as a template for copying and you want to use comments in reporting queries in your template then please change the setting.

**ACTION_IN_RESULT_VISIBLE**: This setting is introduced with note 1744421. It controls whether the command module triggers an event to create a new comment on total lines. By default the command is switched off on totals. This complies with the default settings we have introduced with version 2.10 of this paper where comments on totals are switched off by default (see chapter 2 and 3.5).

**Important**

If you implement note 1744421 and version 2.10 of this guide and still want to use comments on totals then you have to change the default behavior of this setting and set it to on.

### 4.6.4 Java Script for Refreshing and for Disabling Web Template when the Popup is shown

If not yet done (by copying our solution) create a Java Script item and enter the Java Script from the appendix. We use this Java Script for updating the web template with the new comment. Within the Java Script the Web Template is forced to wait for half a second. This is necessary for the correct update after entering or changing a comment: when you press ‘ok’ on the BSP application the comment is written back to the comment buffer. Then the comment module has to read the comment.
and to display it. At the same time the BSP forces the Web Application to refresh so the command module is triggered. As we have to make sure that the update does not happen before the comment is correctly written to the buffer we force the Web Application to wait for one second.

**Important**

In older versions (pre BW 7.40) of this how to paper we used the command ‘set_item_parameters’ to trigger a refresh. The behavior of this command has slightly changed in BW 7.40 and it does not always repaint the screen (and invoke the comment module) if no data has been changed.

Thus we have changed the Java script in such a way that we use the command ‘Apply State (TRANSFER_STATE). We have used the option ‘All Data Providers’ in our Web Template. You can accommodate the solution using the name of your data provider carrying the comments or using the name of the corresponding analysis item.

If you are using a version of BW pre 7.40 you can also use the command ‘set_item_parameters’. You will have to accommodate the Java Script coding accordingly by using the name of your data providers and will have to call the function executeJS_SET_ITEM_PARAMETERS_EAA in the function executeJS_DELAY_REFRESH of our Java Script coding below.

**Hint**

If you are using Tab Strips and several Analysis Items you might have to use the same technique for at least one Analysis Item for each tab where you have Analysis Items with comments.

We also use the Java script to disable the web template when the comment popup is show. Thus we reach a modal behavior of the popup. If you do not want this behavior you can either just clear the Java script inside the functions hide_popup and show_popup or remove the calls of the second and the third button from the BSP layout tab. Please note that the button id starts with ‘0’ so the second and third buttons have the ids BUTTON_GROUP_ITEM_1_btn1_acButton and BUTTON_GROUP_ITEM_1_btn2_acButton.

### 4.6.5 Buttons for Comment Services and for Refresh and Save

In your web application create a button group. We will create some buttons inside this button group to refresh the screen after a comment has been entered and to disable and enable the Web template once the comment popup is called and closed again. Please make sure that the sort order of the buttons is as in the sample template attached to this guide. Also make sure that the button group is called BUTTON_GROUP_ITEM_1. As explained above the BSP application for editing comments has to trigger a refresh/update of the Web application after changes have been done to the comment. We realize this by calling a (refresh) button with a specific technical name. If you want to rename the button the you will have to adopt the BPS application accordingly. The same holds true for the buttons for the disabling and enabling of the Web template.

In order to create the necessary buttons just copy the XHTML coding for the button group 1 from the sample template. Please note that we have made the buttons invisible by using a span around them. Do not use the property ‘Visibility’ of the button group for hiding as this will also disable the Java script. You can place a span or a div around the button group and set it invisible (see sample template).

Now also create a (data) refresh button and a save button (in a new button group). The save button has to call the dummy planning function you have created above. The planning function should be called in delta mode to make sure that the runtime is optimized. When the planning function is
triggered our exit for changing the selection will be called and will save the comments. After that a normal save command should be triggered. In our example we have created two save buttons: one that is called ‘Save Comments Only’ that just triggers the planning function and one that is called ‘Save All’ that triggers the planning function AND a normal save event.

Note
Instead of creating a save button you can as well trigger the planning function for saving the comments from the automatic planning sequence that is executed on save. Thus you can be sure the function is called any time data is saved to the InfoCube from a planning application. For further information see:

http://help.sap.com/saphelp_nw70ehp1/helpdata/en/e3/e60138fede083de10000009b38f8cf/frameset.htm

5. Testing and Trouble Shooting
You can now test your application. For the event that you run into some problems or want to trace the performance of the application we have included the RFC function module to retrieve the comments into the RSTT trace. The RSTT trace is enabled by the following coding at the beginning of the function module Z_RETURN_COMMENTS:

```abap
INCLUDE rsbolap_trace.*
* Begin of the rfc function module
trace_start.
rfc_function_module_start 'Z_RETURN_COMMENTS' 0.
```

The trace is ended by the statement at the end of the function module:

```abap
* End of the rfc function module
rfc_function_module_end 0.
```

If you create your own comment solution please make sure that you use the same statements at the beginning and at the end you your function module returning the comments. Note that only the RFC function module returning the comments can be traced.

You can record a trace either by switching it on in the transaction RSTT or by adding the parameter ‘&TRACE=X’ to the url of your web application and restarting it. The system will now record an ABAP trace that can be replayed from transaction RSTT. In the display of the trace you can see the runtime of the function module returning the comments. If you want to debug the retrieval of the comments you
can simply set a break point (session break point) in the ABAP coding of the function module and replay the trace.

6. Appendix

6.1 Report Z_CREATE_COMMENTS_TABLE

REPORT z_create_comments_table.
* This report creates a table with the name entered by the user that stores the comments for an info provider.
* It also creates an entry in table zmap_iprov_com which maps the table name to the info provider.
* Prerequisite: Table zmap_iprov_com must exist.

TYPES: BEGIN OF l_ty_infoprov_commenttable,
  client TYPE mandt,
  info_prov TYPE rsinfoprov,
  comments_tab TYPE tabname16,
END OF l_ty_infoprov_commenttable.

DATA: l_t_infoobj TYPE STANDARD TABLE OF bapi6112io,
  l_r_iobj TYPE REF TO bapi6112io,
  l_t_table_fields TYPE TABLE OF dd03p,
  l_s_table_field TYPE dd03p,
l_s_table_header TYPE dd02v,
l_s_techn_set TYPE dd09v,
l_s_iobj_properties TYPE rsd_s_cob_pro,
l_position TYPE i,
l_s_infoprov_commenttable TYPE l_ty_infoprov_commenttable,
l_errorOccurred TYPE bool_d.

PARAMETERS: p_iprov TYPE rsinfoprov OBLIGATORY,
p_tabnam TYPE tablename16 OBLIGATORY,
p_tabtxt TYPE as4text OBLIGATORY.

CONSTANTS: lco_map_table_name TYPE ddobjname VALUE 'ZMAP_IPROV_COM',
lco_char_compare(38) TYPE c value 'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789'.

* Do some basic checks for entered table name:
* only in A-Z, 0-9 and _
*IF NOT p_tabnam CO lco_char_compare.
* WRITE: '/', 'Table name may only consist of characters 0-9, A-Z, _'.
* RETURN.
*ENDIF.
* no spaces
*IF p_tabnam CN ' '.
* WRITE: '/', 'Table name may not contain blanks.'.
* RETURN.
*ENDIF.
* length
IF STRLEN( p_tabnam ) <= 3 .
  WRITE: '/', 'Please enter a longer table name.'.
  RETURN.
ENDIF.
* first character in A-Z
IF p_tabnam(1) CN 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'.
  WRITE: '/', 'First character of table name must be in A-Z'.
  RETURN.
ENDIF.
* no _ at second or third position or last position
IF p_tabnam(3) CA '_'.
  WRITE: '/', 'Second and third character of table name may not be _'.
  RETURN.
ENDIF.

* 1)
* Check whether table zmap_iprov_com exists. If not, create it.
CALL FUNCTION 'DDIF_TABL_GET'
  EXPORTING
    name = lco_map_table_name
    state = 'A'
  IMPORTING
    dd02v_wa = l_s_table_header
  EXCEPTIONS
    illegal_input = 1
OTHERS = 2.
IF sy-subrc <> 0 OR l_s_table_header IS INITIAL.
* Table ZMAP_IPROV_COM does not exist. Create it!
   PERFORM create_mapping_table CHANGING l_error_occurred.
   IF l_error_occurred = 'X'.
      RETURN.
   ENDIF.
ENDIF.
CLEAR l_s_table_header.

* 2) Check whether there is already a table for comments for this info provi
ider
SELECT SINGLE * FROM zmap_iprov_com INTO l_s_infoprov_commenttable WHERE in
fo_prov = p_iprov.
IF sy-subrc = 0.
   WRITE: /, 'There is already a table for info provider ', p_iprov, ' which
contains the comments: ', l_s_infoprov_commenttable-comments_tab.
   RETURN.
ENDIF.

* 3) Get Characteristics of InfoProvider
*---------------------------------------------------------------

   data: l_iprov type rsinfoprov.

   CALL FUNCTION 'Z_RETURN_PARTPROV'
      EXPORTING
         I_INFOPROV     = p_iprov
      IMPORTING
         E_PARTPROV     = l_iprov
      EXCEPTIONS
         NOT_UNIQUE     = 1
         OTHERS         = 2
   .
   IF SY-SUBRC <> 0.
      WRITE:/, 'Please check name of InfoProvider. Only Basic InfoCubes are all
owed here.'.
      RETURN.
   ENDIF.
   p_iprov = l_iprov.

   *CALL FUNCTION 'RSD_CUBE_IOBJ_GET'
   CALL FUNCTION 'BAPI_CUBE_GETDETAIL'
      EXPORTING
         infocube       = p_iprov
      TABLES
         infoobjects   = l_t_infoobj.

   IF l_t_infoobj IS INITIAL .
      WRITE:/, 'Please check name of InfoProvider. No Info Objects could be det
ermined for entered InfoProvider.'.
      RETURN.
ENDIF.

* 4) Create a DDIC Table for InfoProvider in which the comments are stored
*---------------------------------------------------------------

* a) Specify Table Header
l_s_table_header-tabname = p_tabnam.
l_s_table_header-ddtext = p_tabtxt.
l_s_table_header-ddlanguage = sy-langu.
l_s_table_header-tabclass = 'TRANSP'.
l_s_table_header-as4user = sy-uname.
l_s_table_header-contflag = 'A'.
l_s_table_header-mainflag = 'X'.

* b) Fill technical settings
*******************************
l_s_techn_set-tabname = p_tabnam.
l_s_techn_set-tabkat = 0.
l_s_techn_set-tabart = 'APPL1'.
l_s_techn_set-bufallow = 'X'.
l_s_techn_set-pufferung = 'X'. "Fully buffered ( G = Generically, P = Single X = Fully)
*i_s_techn_set-schfeldanz = 5. " up to 5 key fields

* c) Specify Table Fields
l_s_table_field-tabname = p_tabnam.
l_s_table_field-ddlanguage = sy-langu.
l_s_table_field-notnull = 'X'.

* ID | Characteristic 1 | ... | Characteristic M | KYFNM | CMT
* _
Column ID (Key Field as tables may only have up to 16 key fields, will be
filled by Hash Key)
l_s_table_field-keyflag = 'X'.
l_s_table_field-fieldname = 'ID'.
l_position = l_position + 1.
l_s_table_field-position = l_position.
l_s_table_field-rollname = 'HASH160'.
APPEND l_s_table_field TO l_t_table_fields.

* _
Specify Table Fields for Characteristics of InfoProvider (technical info o
bects and key figures can be ignored)
LOOP AT l_t_infoobj REFERENCE INTO l_r_iobj WHERE iobjtp <> 'KYF' AND iobjt
p <> 'DPA'.
CLEAR l_s_iobj_properties.
CALL FUNCTION 'Z_COMNTS_FIELDS_CONVERT' EXPORTING
  i_iobjnm = l_r_iobj->infoobject
IMPORTING
  e_fieldname = l_s_table_field-fieldname.
CALL FUNCTION 'RSD_IOBJ_GET'
EXPORTING
  i_iobjnm = l_r_iobj->infoobject
IMPORTING
  e_s_cob_pro = l_s_iobj_properties
EXCEPTIONS
  iobj_not_found = 1
  illegal_input = 2
  bct_comp_invalid = 3
  OTHERS = 4.
IF sy-subrc = 0.
  l_position = l_position + 1.
  l_s_table_field-position = l_position.
  l_s_table_field-keyflag = ' '.
  l_s_table_field-rollname = l_s_iobj_properties-dtelnm.
  APPEND l_s_table_field TO l_t_table_fields.
ENDIF.
ENDLOOP.

* - Specify Table Fields for Key Figure Names
CLEAR l_s_table_field.
  l_s_table_field-tabname = p_tabnam.
  l_s_table_field-fieldname = 'KYFNM'.
  l_s_table_field-notnull = 'X'.
  l_s_table_field-ddlanguage = sy-langu.
  l_s_table_field-keyflag = ' '. "? Shall Key Figure be a key field together with ID ?
  l_position = l_position + 1.
  l_s_table_field-position = l_position.
  l_s_table_field-rollname = 'RSIOBJNM'.
  APPEND l_s_table_field TO l_t_table_fields.

* - Specify Table Field for Comment
CLEAR l_s_table_field.
  l_s_table_field-tabname = p_tabnam.
  l_s_table_field-fieldname = 'CMT'.
  l_s_table_field-notnull = 'X'.
  l_s_table_field-ddlanguage = sy-langu.
  l_s_table_field-keyflag = ' '.
  l_position = l_position + 1.
  l_s_table_field-position = l_position.
  l_s_table_field-ddtype = 'STRG'.
  l_s_table_field-ddtext = 'Comment'.
  APPEND l_s_table_field TO l_t_table_fields.

* - Create Transparent Table
CALL FUNCTION 'DDIF_TABL_PUT'
  EXPORTING
    name = l_s_table_header-
           tabname " Name of the Table to be Written
    dd02v_wa = l_s_table_header
    dd09l_wa = l_s_techn_set
  TABLES
dd03p_tab = l_t_table_fields " Table fields

EXCEPTIONS
  tabl_not_found = 1
  name_inconsistent = 2
  tabl_inconsistent = 3
  put_failure = 4
  put_refused = 5
  OTHERS = 6.

IF sy-subrc = 0.
  WRITE: '/,'Transparent Table ', p_tabnam, ' has been created.'.
ELSE.
  WRITE: '/,'Error occurred'.
  RETURN.
ENDIF.

* - Try to activate table (Enhancement category is missing)
CALL FUNCTION 'DDIF_TABL_ACTIVATE'
  EXPORTING
    name = l_s_table_header-tabname
  * IMPORTING
  * RC
  * EXCEPTIOns
    not_found = 1
    put_failure = 2
    OTHERS = 3.

IF sy-subrc <> 0.
  WRITE: '/,'Transparent Table ', p_tabnam, ' could not be activated.'.
  RETURN.
ELSE.
  WRITE: '/,'Transparent Table ', p_tabnam, ' has been activated.'.
ENDIF.

* 5) Create entry in table ZMAP_IPROV_COM for this new created comments tab
le and given info provider
l_s_infoprov_commenttable-client = sy-mandt.
l_s_infoprov_commenttable-info_prov = p_iprov.
l_s_infoprov_commenttable-comments_tab = p_tabnam.
INSERT INTO zmap_iprov_com VALUES l_s_infoprov_commenttable.
IF sy-subrc <> 0.
  WRITE: '/,'An entry for Info Provider ', p_iprov, ' already exists in table
ZMAP_IPROV_COM. Please check this.'.
ELSE.
  COMMIT WORK AND WAIT.
  IF sy-subrc = 0.
    WRITE: '/,'An entry for Info Provider ', p_iprov,' in table ZMAP_IPROV_CO
M was created successfully.'.
  ELSE.
    WRITE: '/,'An error occurred during COMMIT WORK.'.
  ENDIF.
ENDIF.

*---------------------------------------------------------------------*
* Form create_mapping_table
FORM create_mapping_table CHANGING x_error_occurred TYPE boole_d.

* a) Specify Table Header
  l_s_table_header-tabname = lco_map_table_name.
  l_s_table_header-ddtext = 'Mapping Table for Info Providers and their Comments Table'.
  l_s_table_header-ddlanguage = sy-langu.
  l_s_table_header-tabclass = 'TRANS'.
  l_s_table_header-as4user = sy-uname.
  l_s_table_header-contflag = 'A'.
  l_s_table_header-mainflag = 'X'.

* b) Fill technical settings
  **********************************
  l_s_techn_set-tabname = lco_map_table_name.
  l_s_techn_set-tabkat = 0.
  l_s_techn_set-tabart = 'APPL0'.
  l_s_techn_set-bufallow = 'X'.
  l_s_techn_set-pufferung = 'P'. "( G = Generically, P = Single X = Fully)

* c) Specify Table Fields
  l_s_table_field-tabname = lco_map_table_name.
  l_s_table_field-ddlanguage = sy-langu.
  l_s_table_field-notnull = 'X'.

* CLIENT | INFOPROV | COMMENTS_TAB
* - CLIENT
  l_s_table_field-keyflag = 'X'.
  l_s_table_field-fieldname = 'CLIENT'.
  l_s_table_field-position = l_s_table_field-position + 1.
  l_s_table_field-rollname = 'MANDT'.
  APPEND l_s_table_field TO l_t_table_fields.

* - INFOPROV
  l_s_table_field-keyflag = 'X'.
  l_s_table_field-fieldname = 'INFO_PROV'.
  l_s_table_field-position = l_s_table_field-position + 1.
  l_s_table_field-rollname = 'RSINFOPROV'.
  APPEND l_s_table_field TO l_t_table_fields.

* - COMMENTS_TAB
  l_s_table_field-keyflag = '.
  l_s_table_field-fieldname = 'COMMENTS_TAB'.
  l_s_table_field-position = l_s_table_field-position + 1.
  l_s_table_field-rollname = 'TABNAME16'.
  APPEND l_s_table_field TO l_t_table_fields.
* - Create Transparent Table

CALL FUNCTION 'DDIF_TABL_PUT'
EXPORTING
  name = l_s_table_header-
tabname " Name of the Table to be Written
  dd02v_wa = l_s_table_header
  dd09l_wa = l_s_techn_set
TABLES
  dd03p_tab = l_t_table_fields " Table fields
EXCEPTIONS
  tabl_not_found = 1
  name_inconsistent = 2
  tabl_inconsistent = 3
  put_failure = 4
  put_refused = 5
  OTHERS = 6.
IF sy-subrc = 0.
  WRITE: /, 'Transparent Table ', lco_map_table_name, ' has been created.
  .
ELSE.
  WRITE: /, 'Error occurred'.
  x_error_occurred = 'X'.
  RETURN.
ENDIF.

* - Try to activate table (Enhancement category is missing)

CALL FUNCTION 'DDIF_TABL_ACTIVATE'
EXPORTING
  name = l_s_table_header-
tabname
* IMPORTING
  RC =
* EXCEPTIONS
  not_found = 1
  put_failure = 2
  OTHERS = 3.
IF sy-subrc <> 0.
  WRITE: /, 'Transparent Table ', lco_map_table_name, ', could not be acti
  vated.'.
  x_error_occurred = 'X'.
  RETURN.
ELSE.
  WRITE: /, 'Transparent Table ', lco_map_table_name, ', has been activ
  ed.'.
  ENDIF.

CLEAR: l_s_table_header, l_s_techn_set, l_s_table_field, l_t_table_fields
, l_position.
ENDFORM.  " create_mapping_table
6.2 Top Include for Function Group Z_COMMENTS

FUNCTION-POOL Z_COMMENTS.

"MESSAGE-ID ..

TYPE-POOLS: rsdm, ZCO, RZX0.

data: g_handle type guid_32.

constants: c_new type c value 'N',
           c_del type c value 'D'.

constants: C_KYFNM_iobjnm(10) type c value '1KYFNM',
           c_kyfnm_fieldnm(10) type c value 'KYFNM',
           c_field_infoprov(20) type c value '_INFO_PROVIDER_',
           c_field_partprov(20) type c value '0INFOPROV',
           c_field_query(20) type c value '_QUERY_VIEW_NAME_'.

6.3 The Function Modules

FUNCTION z_comnts_fields_convert.

**"# Local Interface:
**"# IMPORTING
**"# REFERENCE(I_IOBJNM) TYPE RSIOBJNM
**"# EXPORTING
**"# REFERENCE(E_FIELDNAME) TYPE FIELDNAME

DATA: l_s_details TYPE bapi6108,
      l_s_return TYPE bapiret2.

CLEAR e_fieldname.

if i_iobjnm cs zco_c_kyfstruc_query.
   e_fieldname = c_kyfnm_fieldnm.
elseif i_iobjnm = c_kyfnm_iobjnm.
   e_fieldname = c_kyfnm_fieldnm.
else.
*GET the FIELD name in the dso structure
   CALL FUNCTION 'BAPI_IOBJ_GETDETAIL'
      EXPORTING
         version = rs_c_objvers-active
         infoobject = i_iobjnm
      IMPORTING
         details = l_s_details
         return = l_s_return.
      IF l_s_return IS INITIAL.
         e_fieldname = l_s_details-fieldnm.
      ENDIF.
endif.

* CASE i_iobjnm.
  * WHEN c_kyfnm_iobjnm.
    * e_fieldname = c_kyfnm_fieldnm.
  *
  * WHEN zco_c_kyfstruc_query.
    * e_fieldname = c_kyfnm_fieldnm.
  *
  * WHEN OTHERS.
  **GET the FIELD name in the dso structure
  * CALL FUNCTION 'BAPI_IOBJ_GETDETAIL'
  * EXPORTING
    * version = rs_c_objvers-active
    * infoobject = i_iobjnm
  * IMPORTING
    * details = l_s_details
    * return = l_s_return.
  *
  * IF l_s_return IS INITIAL.
    * e_fieldname = l_s_details-fieldnm.
  *
  * ENDCASE.
ENDFUNCTION.

FUNCTION z_create_dataref.
**"-----------------------------------------------
**"Local Interface:
**" IMPORTING
  ** Reference(I_INFOPROV) TYPE RSINFOPROV
  ** Reference(I_TYPE) TYPE STRING
  ** EXPORTING
    ** Reference(E_O_DATADESC) TYPE REF TO CL_ABAP_DATADESCR
**"-----------------------------------------------

TYPES: BEGIN OF ty_types,
  infoprov TYPE rsinfoprov,
  ty_comments_db TYPE REF TO cl_abap_structdescr,"
  ty_comments_line TYPE REF TO cl_abap_structdescr,"
  ty_comments_action TYPE REF TO cl_abap_structdescr,"
  tt_comments_db TYPE REF TO cl_abap_tabledscr,"
  tt_comments_line TYPE REF TO cl_abap_tabledscr,"
  tt_comments_action TYPE REF TO cl_abap_tabledscr,"
  ty_call_buf TYPE REF TO cl_abap_structdescr,
  tt_call_buf TYPE REF TO cl_abap_tabledscr,
END OF ty_types.

STATICS: l_t_types TYPE SORTED TABLE OF ty_types WITH UNIQUE KEY infoprov.

DATA:
l_s_types TYPE ty_types,
l_comments_table TYPE tabname16,
l_o_structdesc TYPE REF TO cl_abap_structdescr,
l_o_tabledesc TYPE REF TO cl_abap_tabledescr,
l_t_components TYPE abap_component_tab,
l_s_component LIKE LINE OF l_t_components,
l_t_keys TYPE abap_keydescr_tab,
l_s_key LIKE LINE OF l_t_keys,
l_x_root TYPE REF TO cx_root,
l_line TYPE i.

FIELD-SYMBOLS:   <l_component_structure> TYPE REF TO cl_abap_structdescr,
                <l_component_table> TYPE REF TO cl_abap_tabledescr.

* First of all check whether types have already been created
READ TABLE l_t_types WITH TABLE KEY infoprov = i_infoprov INTO l_s_types.

  IF sy-subrc <> 0. " Generate All Types
  *******************************************************
  TRY.
    l_s_types-infoprov = i_infoprov.

  * 1) Create ty_comments_db
  * - Get table name of comments table
    PERFORM get_comments_tab
    USING
      i_infoprov
    CHANGING
      l_comments_table.
    IF l_comments_table IS INITIAL.
    * Content in table ZMAP_IPROV_COM is missing:
    * please enter name of table that contains comments for your info provider.
      RAISE EXCEPTION TYPE cx_fatal_exception.
    ENDIF.

    l_o_structdesc ?= cl_abap_typedescr=>describe_by_name( l_comments_table ),
    l_s_types-ty_comments_db = l_o_structdesc.

  *2) Create tt_comments_db
    l_t_components = l_o_structdesc->get_components( ).
    l_s_key-name = 'ID'.
    APPEND l_s_key TO l_t_keys.
    l_o_tabledesc = cl_abap_tabledescr=>create(
      p_line_type = l_s_types-ty_comments_db
      p_table_kind = cl_abap_tabledescr=>tablekind_sorted
    )
      p_unique = abap_false
    p_key = l_t_keys
    l_o_tabledesc.
p_key_kind = cl_abap_tabledescr=>keydefkind_use
r
).
l_s_types_tt_comments_db = l_o_tabledesc.

* 3) Create ty_comments_action
l_t_components = l_o_structdesc->get_components().
l_s_component-name = 'ACTION'.
l_s_component-type = cl_abap_elemdescr=>get_c( l ).
APPEND l_s_component TO l_t_components.
l_o_structdesc = cl_abap_structdescr=>create( l_t_components ).
l_s_types_ty_comments_action = l_o_structdesc.

* 4) Create tt_comments_action
l_o_tabledesc = cl_abap_tabledescr=>create(
   p_line_type = l_s_types-ty_comments_action
   p_table_kind = cl_abap_tabledescr=>tablekind_sort
).
l_s_types_tt_comments_action = l_o_tabledesc.

* 5) Create ty_comments_line
l_line = LINES( l_t_components ).
DELETE l_t_components INDEX l_line. " remove 'ACTION' component
l_s_component-name = 'LINE'.
l_s_component-type = cl_abap_elemdescr=>get_i( ).
APPEND l_s_component TO l_t_components.
l_o_structdesc = cl_abap_structdescr=>create( l_t_components ).
l_s_types_ty_comments_line = l_o_structdesc.

* 6) Create tt_comments_line
l_o_tabledesc = cl_abap_tabledescr=>create(
   p_line_type = l_s_types-ty_comments_line
   p_table_kind = cl_abap_tabledescr=>tablekind_sort
).
l_s_types_tt_comments_line = l_o_tabledesc.

* 7) Create ty_call_buf
   CLEAR l_t_components.
   First component: KEYFIELD type c(1)
   l_s_component-name = 'KEYFIELD'.
l_s_component-type = cl_abap_elemdescr=>get_c( l ).
   APPEND l_s_component TO l_t_components.
* Second component: T_IN type TT_COMMENTS_LINE
  l_s_component-name = 'T_IN'.
  l_s_component-type = l_s_types-tt_comments_line.
  APPEND l_s_component TO l_t_components.
  l_o_structdesc = cl_abap_structdescr=>create( l_t_components ).
  l_s_types-tt_comments_line = l_o_structdesc.

* 8) Create tt_call_buf
  CLEAR l_t_keys.
  l_s_key-name = 'KEYFIELD'.
  APPEND l_s_key TO l_t_keys.
  l_o_tabledesc = cl_abap_tabledescr=>create( p_line_type = l_o_structdesc
      p_table_kind = cl_abap_tabledescr=>tablekind_sort
      p_unique = abap_false
      p_key = l_t_keys
      p_key_kind = cl_abap_tabledescr=>keydefkind_use
    )
  l_s_types-tt_call_buf = l_o_tabledesc.

* Save data types in table
  APPEND l_s_types TO l_t_types.
  CATCH cx_sy_struct_creation cx_sy_table_creation INTO l_x_root. " e_o
 _datadesc is not bound if it has not been bound before
   ENDTry.
  ENDF.

* Return requested data type object
  CASE i_type(2).
    WHEN 'TY'.
      ASSIGN COMPONENT i_type OF STRUCTURE l_s_types TO <l_component_structure>
        IF sy-subrc = 0.
          e_o_datadesc = <l_component_structure>.
        ENDIF.
    WHEN 'TT'.
      ASSIGN COMPONENT i_type OF STRUCTURE l_s_types TO <l_component_table>
        IF sy-subrc = 0.
          e_o_datadesc = <l_component_table>.
        ENDIF.
    WHEN OTHERS.
      ENDcase.
  ENDFUNCTION.

FUNCTION Z_CREATE_HASHKEY.
**"--------------------------------------------------------------------------
**"Local Interface:
**" IMPORTING
**How To... Build a Fast and Flexible Comment Solution for BEx Web Applications**

```abap
/*
  REFERENCE(i_infoprov) TYPE RSINFOPROV
  CHANGING
  REFERENCE(x_data_struct) TYPE ANY
  REFERENCE(x_hashkey) TYPE HASH160
*/

PERFORM create_hashkey
  USING
  i_infoprov
  CHANGING
  x_data_struct
  x_hashkey.

ENDFUNCTION.

FUNCTION z_delete_comments.

/*
** Local Interface:
** IMPORTING
** REFERENCE(i_to_comments) TYPE ANY TABLE
** REFERENCE(i_handle) TYPE GUID_32
** REFERENCE(i_infoprov) TYPE RSINFOPROV
*/

DATA: l_r_datadesc TYPE REF TO cl_abap_datadescr,
     l_r_s_comments_line type REF TO data,
     l_r_to_cmt_buf TYPE REF TO data,
     l_r_t_call_buf TYPE REF TO data,
     l_r_s_cmt_buf TYPE REF TO data,
     l_r_comments_line type REF TO data,
     l_hashkey type hash160.

FIELD-SYMBOLS: <l_s_cmt_buf> TYPE ANY,
                <l_s_comments> TYPE ANY,
                <l_s_comments_line> type any,
                <l_to_comments> type any table,
                <l_t_call_buf> TYPE ANY TABLE,
                <l_to_cmt_buf> TYPE ANY TABLE,
                <l_action> TYPE ANY.

* get work area for line table
CALL FUNCTION 'Z_CREATE_DATAREF'
  EXPORTING
    i_infoprov = i_infoprov
    i_type = zco_tt_comments_line
  IMPORTING
    e_o_datadesc = l_r_datadesc.
CREATE DATA l_r_comments_line TYPE HANDLE l_r_datadesc.
 ASSIGN l_r_comments_line->* TO <l_to_comments>.
```
How To... Build a Fast and Flexible Comment Solution for BEx Web Applications

* * get work area the line structure

CALL FUNCTION 'Z_CREATE_DATAREF'
EXPORTING
  i_infoprov = i_infoprov
  i_type = zco_ty_comments_line
IMPORTING
  e_o_datadesc = l_r_datadesc.
CREATE DATA l_r_s_comments_line TYPE HANDLE l_r_datadesc.
ASSIGN l_r_s_comments_line->* TO <l_s_comments_line>.

* get the comment buffer table first

CALL FUNCTION 'Z_CREATE_DATAREF'
EXPORTING
  i_infoprov = i_infoprov
  i_type = zco_tt_comments_buf
IMPORTING
  e_o_datadesc = l_r_datadesc.
CREATE DATA l_r_to_cmt_buf TYPE HANDLE l_r_datadesc.
ASSIGN l_r_to_cmt_buf->* TO <l_to_cmt_buf>.

* get the call buffer

CALL FUNCTION 'Z_CREATE_DATAREF'
EXPORTING
  i_infoprov = i_infoprov
  i_type = zco_tt_call_buf
IMPORTING
  e_o_datadesc = l_r_datadesc.
CREATE DATA l_r_t_call_buf TYPE HANDLE l_r_datadesc.
ASSIGN l_r_t_call_buf->* TO <l_t_call_buf>.

* get the work area for the comment buf

* get the comment buffer table first

CALL FUNCTION 'Z_CREATE_DATAREF'
EXPORTING
  i_infoprov = i_infoprov
  i_type = zco_ty_comments_buf
IMPORTING
  e_o_datadesc = l_r_datadesc.
CREATE DATA l_r_s_cmt_buf TYPE HANDLE l_r_datadesc.
ASSIGN l_r_s_cmt_buf->* TO <l_s_cmt_buf>.
ASSIGN COMPONENT 'ACTION' OF STRUCTURE <l_s_cmt_buf> TO <l_action>.

* check the buffer instance

PERFORM get_buffer_instance USING i_handle i_infoprov CHANGING g_handle
  <l_to_cmt_buf> <l_t_call_buf>.

* Hashkey is added in column ID of i_to_comments and save in <l_to_comments
LOOP AT i_to_comments INTO <l_s_comments_line>.  
PERFORM create_hashkey  
USING  
i_infoprov  
CHANGING  
<l_s_comments_line>  
l_hashkey.  
INSERT <l_s_comments_line> INTO TABLE <l_to_comments>.  
ENDLOOP.

LOOP AT <l_to_comments> ASSIGNING <l_s_comments>.  
CLEAR <l_s_cmt_buf>.  
MOVE-CORRESPONDING <l_s_comments> TO <l_s_cmt_buf>.  
READ TABLE <l_to_cmt_buf> FROM <l_s_cmt_buf> INTO <l_s_cmt_buf>.  
IF sy-subrc = 0.  
* we have an entry in the buffer table  
<l_action> = c_del.  
MODIFY TABLE <l_to_cmt_buf> FROM <l_s_cmt_buf>.  
ELSE.  
* no entry yet, but maybe on the data base  
<l_action> = c_new.  
INSERT <l_s_cmt_buf> INTO TABLE <l_to_cmt_buf>.  
ENDIF.  
ENDLOOP.  

* update the shared buffer  
PERFORM set_buffer_instance USING i_handle i_infoprov  
CHANGING g_handle <l_to_cmt_buf> <l_t_call_buf>.  
ENDFUNCTION.
*  l_s_comments LIKE LINE OF l_to_comments,
*  l_s_comments_db LIKE LINE OF l_to_comments_db,
*  l_s_cmt_buf TYPE zco_comment_buf,
*  l_s_to_comment LIKE LINE OF x_to_comments.
* DATA:  l_s_call_buf TYPE zco_s_call_buf.
* DATA:  l_to_cmt_buf TYPE zco_to_cmt_buf,
*  l_t_call_buf TYPE zco_t_call_buf.
*
DATA:  l_call_db TYPE c,
    l_o_struct_desc TYPE REF TO cl_abap_structdescr,
    l_t_components TYPE abap_component_tab,
    l_r_component TYPE REF TO abap_componentdescr,
    l_lines TYPE i,
    l_tabix LIKE sy-tabix,
    l_where TYPE string,
    l_t_where TYPE STANDARD TABLE OF string,
    l_comments_tablename TYPE tabname16,
    l_hashkey TYPE hash160.

DATA:  l_r_datadesc TYPE REF TO cl_abap_datadescr,
    l_r_to_cmt_buf TYPE REF TO data,
    l_r_t_call_buf TYPE REF TO data,
    l_r_s_cmt_buf TYPE REF TO data,
    l_r_to_comments_db TYPE REF TO data,
    l_r_s_comments_db TYPE REF TO data,
    l_r_s_call_buf TYPE REF TO data,
    l_r_comments_line TYPE REF TO data,
    l_r_s_comments_line TYPE REF TO data.

FIELD-SYMBOLS:  <l_s_cmt_buf> TYPE ANY,
                 <l_s_call_buf> TYPE ANY,
                 <l_s_comments> TYPE ANY,
                 <l_s_comments_db> TYPE ANY,
                 <l_t_call_buf> TYPE ANY TABLE,
                 <l_to_cmt_buf> TYPE ANY TABLE,
                 <l_to_comments_db> TYPE ANY TABLE,
                 <l_to_comments> TYPE ANY TABLE,
                 <l_s_comments_line> TYPE ANY,
                 <l_t_in> TYPE ANY,
                 <l_cmt> TYPE ANY,
                 <l_cmt_db> TYPE ANY,
                 <l_action> TYPE ANY.
*
get the comment buffer table first
CALL FUNCTION 'Z_CREATE_DATAREF'
  EXPORTING
    i_infoprov = i_infoprov
i_type = zco_tt_comments_buf
IMPORTING
e_o_datadesc = l_r_datadesc.

CREATE DATA l_r_to_cmt_buf TYPE HANDLE l_r_datadesc.
ASSIGN l_r_to_cmt_buf->* TO <l_to_cmt_buf>.

* get the call buffer
CALL FUNCTION 'Z_CREATE_DATAREF'
  EXPORTING
    i_infoprov = i_infoprov
    i_type = zco_tt_call_buf
IMPORTING
    e_o_datadesc = l_r_datadesc.

CREATE DATA l_r_t_call_buf TYPE HANDLE l_r_datadesc.
ASSIGN l_r_t_call_buf->* TO <l_t_call_buf>.

* get the work area for the comment buf
* get the comment buffer table first
CALL FUNCTION 'Z_CREATE_DATAREF'
  EXPORTING
    i_infoprov = i_infoprov
    i_type = zco_ty_comments_buf
IMPORTING
    e_o_datadesc = l_r_datadesc.

CREATE DATA l_r_s_cmt_buf TYPE HANDLE l_r_datadesc.
ASSIGN l_r_s_cmt_buf->* TO <l_s_cmt_buf>.

* get the work area for the call buf
CALL FUNCTION 'Z_CREATE_DATAREF'
  EXPORTING
    i_infoprov = i_infoprov
    i_type = zco_ty_call_buf
IMPORTING
    e_o_datadesc = l_r_datadesc.

CREATE DATA l_r_s_call_buf TYPE HANDLE l_r_datadesc.
ASSIGN l_r_s_call_buf->* TO <l_s_call_buf>.

* get work area for line table
CALL FUNCTION 'Z_CREATE_DATAREF'
  EXPORTING
    i_infoprov = i_infoprov
    i_type = zco_tt_comments_line
IMPORTING
    e_o_datadesc = l_r_datadesc.
CREATE DATA l_r_comments_line TYPE HANDLE l_r_datadesc.
ASSIGN l_r_comments_line->* TO <l_to_comments>.
* get work area the line structure
  CALL FUNCTION 'Z_CREATE_DATAREF'
  EXPORTING
  i_infoprov = i_infoprov
  i_type = zco_ty_comments_line
  IMPORTING
  e_o_datadesc = l_r_datadesc.
  CREATE DATA l_r_s_comments_line TYPE HANDLE l_r_datadesc.
  ASSIGN l_r_s_comments_line->* TO <l_s_comments_line>.

* data base table
  CALL FUNCTION 'Z_CREATE_DATAREF'
  EXPORTING
  i_infoprov = i_infoprov
  i_type = zco_tt_comments_db
  IMPORTING
  e_o_datadesc = l_r_datadesc.
  CREATE DATA l_r_to_comments_db TYPE HANDLE l_r_datadesc.
  ASSIGN l_r_to_comments_db->* TO <l_to_comments_db>.

* work area for data base table
  CALL FUNCTION 'Z_CREATE_DATAREF'
  EXPORTING
  i_infoprov = i_infoprov
  i_type = zco_ty_comments_db
  IMPORTING
  e_o_datadesc = l_r_datadesc.
  CREATE DATA l_r_s_comments_db TYPE HANDLE l_r_datadesc.
  ASSIGN l_r_s_comments_db->* TO <l_s_comments_db>.

* get components of db table
  l_o_struct_desc ?= l_r_datadesc.
  l_t_components = l_o_struct_desc->get_components( ).

* check the buffer instance
  PERFORM get_buffer_instance USING i_handle i_infoprov CHANGING g_handle < l_to_cmt_buf> <l_t_call_buf>.

* Hashkey is added in column ID of x_to_comments
  LOOP AT x_to_comments INTO <l_s_comments_line>.
    PERFORM create_hashkey
      USING
      i_infoprov
      CHANGING
      <l_s_comments_line>
      l_hashkey.
    INSERT <l_s_comments_line> INTO TABLE <l_to_comments>.
  ENDDO.
  * x_to_comments now contains hashkeys
  x_to_comments = <l_to_comments>.
CLEAR <l_to_comments>.

* check the call buffer first – unfortunately we have to do a loop...
  l_call_db = 'X'.
ASSIGN COMPONENT 'T_IN' OF STRUCTURE <l_s_call_buf> TO <l_t_in>.
LOOP AT <l_t_call_buf> INTO <l_s_call_buf>.
  IF <l_t_in> = x_to_comments.
    CLEAR l_call_db.
    EXIT.
  ENDF.
ENDLOOP.

* READ TABLE <l_t_call_buf> WITH KEY t_in = x_to_comments TRANSPORTING NO FIELDS.
* IF sy-subrc = 0.
** we had this call before
*   CLEAR l_call_db.
* ELSE.
*   l_call_db = 'X'.
* ENDIF.

IF l_call_db IS NOT INITIAL.
  CLEAR <l_s_call_buf>.
  <l_t_in> = x_to_comments.
  INSERT <l_s_call_buf> INTO TABLE <l_t_call_buf>.
  CLEAR <l_to_comments_db>.

* We select the data from the importing table into a new table
* Get name of table where comments are stored.
  PERFORM get_comments_tab USING i_infoprov CHANGING l_comments_tablename.
* IF l_comments_tablename IS INITIAL.
  * Content in table ZMAP_IPROV_COM is missing:
  * please enter name of table that contains comments for your info provider.
    RAISE EXCEPTION TYPE cx_fatal_exception.
  ENDIF.
** Generate Where-Clause for Select Statement:
** - get names of fields in table x_to_comments
  * l_lines = LINES( l_t_components ).
  * l_lines = l_lines - 1. "because fields CMT and ID are ignored
** - add one where condition per component separated by 'AND'
  * LOOP AT l_t_components REFERENCE INTO l_r_component WHERE name <> 'CMT'
    AND name <> 'ID'.
  * l_tabix = sy-tabix.
  * CLEAR l_where.
  * CONCATENATE l_r_component->name ' = x_to_comments-' l_r_component->name INTO l_where.
  * APPEND l_where TO l_t_where.
** -- do not append 'AND' for last component
  * IF l_tabix < l_lines.
How To... Build a Fast and Flexible Comment Solution for BEx Web Applications

* APPEND 'AND' TO l_t where.
* ENDF.
* ENDOFL.
* SELECT * FROM (l_comments_tablename) INTO TABLE <l_to_comments_db>
  FOR ALL ENTRIES IN x_to_comments
* WHERE (l_t where).
*ENDIF.

* Select comments from database by hashkeys

  APPEND 'id = x_to_comments-id' TO l_t where.
  SELECT * FROM (l_comments_tablename) INTO TABLE <l_to_comments_db>
  FOR ALL ENTRIES IN x_to_comments
  WHERE (l_t where).
ENDIF.

* we have to consolidate with the buffer - we loop over the table with the selections
* and then check in the buffer and the db table whether we have an entry.
  LOOP AT x_to_comments ASSIGNING <l_s_comments>.

* make sure we do not have some old (wrong) comments in the table already..
  ASSIGN COMPONENT 'CMT' OF STRUCTURE <l_s_comments> TO <l_cmt>.
  CLEAR <l_cmt>.
  CLEAR <l_s_comments_db>.
  CLEAR <l_s_cmt_buf>.

  MOVE CORRESPONDING <l_s_comments> TO <l_s_cmt_buf>.
* <l_to_cmt_buf> only contains Hash Keys for saved entries ->
  also just created comments need to get hash keys
  READ TABLE <l_to_cmt_buf> FROM <l_s_cmt_buf> INTO <l_s_cmt_buf>.

* do we have the entry in the buffer already?
  IF sy-subrc <> 0.
* no - now check the db result table if necessary. If not bd table exists just ignore the entry
    MOVE CORRESPONDING <l_s_comments> TO <l_s_comments_db>.
    READ TABLE <l_to_comments_db> INTO <l_s_comments_db> FROM <l_s_comments_db>.

* do we have a comment in the db? Only use it if it is not empty
  ASSIGN COMPONENT 'CMT' OF STRUCTURE <l_s_comments_db> TO <l_cmt_db>.
  IF sy-subrc = 0 AND <l_cmt_db> IS NOT INITIAL.
* insert it into exporting table
    <l_cmt> = <l_cmt_db>.
    INSERT <l_s_comments> INTO TABLE <l_to_comments>.

* insert it into buffer
  CLEAR <l_s_cmt_buf>.
  MOVE CORRESPONDING <l_s_comments> TO <l_s_cmt_buf>.
FUNCTION z_return_comments .
**"Local Interface: **
**IMPORTING **
**" VALUE(I_T_OBJECT_NAMES) TYPE RSBOLAP_T_OBJECT_NAME **
**" VALUE(I_T_MEMBER_NAMES) TYPE BICS_CONS_MD_EXIT_T_MEMBERNAME **
**" VALUE(I_T_VARIABLES) TYPE BICS_PROV_STATE_T_INIT_VARIABLE **
**EXPORTING **
**" VALUE(E_T_VALUES) TYPE BICS_CONS_MD_EXIT_T_VALUE **
**-------------------------------------------------------------------**

INCLUDE rsbolap_trace.
**Begin of the rfc function module**
trace_start.
rfc_function_module_start 'Z_RETURN_COMMENTS' 0.

FIELD-SYMBOLS: <comp> TYPE ANY.
DATA: l_iobjs TYPE i,
    l_index TYPE i,
    l_line TYPE i,
    l_lines TYPE i,
l_value TYPE rschavl,
l_fieldname TYPE fieldname,
l_iobjnm TYPE rsiobjnm,
l_infoprov TYPE rsinfoprov,
l_partprov TYPE rsinfoprov,
l_s_member_names LIKE LINE OF i_t_member_names,
l_s_value LIKE LINE OF e_t_values,
l_s_variables LIKE LINE OF i_t_variables,
l_object_name LIKE LINE OF i_t_object_names,
l_partprov2 TYPE rsinfoprov.

**************************************************
DATA:
l_r_to_comments_line TYPE REF TO data,
l_r_s_comments_line TYPE REF TO data,
l_r_ts_comments_line TYPE REF TO data,
l_r_datadesc TYPE REF TO cl_abap_datadescr,
l_t_variables like i_t_variables,
l_t_eltdir type standard table of RSZ_X_ELTDIR,
l_s_eltdir type RSZ_X_ELTDIR,
l_t_range type standard table of RSZ_X_RANGE,
l_s_range TYPE rsz_x_range,
l_t_var type STANDARD TABLE OF RSZ_X_VAR,
l_s_var type RSZ_X_VAR,
l_key_id type SYSUUID_25,
l_total_found TYPE c,
l_no_ex_in_conv type c,
l_length type i.

FIELD-SYMBOLS: <x_to_comments> TYPE INDEX TABLE,
<|s_comments> TYPE ANY,
<|s_comments_fix> TYPE ANY,
<|t_comments> TYPE INDEX TABLE.

CLEAR e_t_values.

***************************************************************************
****
* 1) Replace the structure elements and get the name of the InfoCube
***************************************************************************
****

l_t_variables = i_t_variables.

perform replace_structures_var changing l_t_variables l_t_eltdir l_t_range e l_t_var.

READ TABLE l_t_variables INTO l_s_variables WITH KEY name = c_field_infoprov.
IF sy-subrc = 0.
  l_infoprov = l_s_variables-low.
ENDIF.
READ TABLE l_t_variables INTO l_s_variables WITH KEY name = c_field_partprov.
IF sy-subrc = 0. " we probably are in a multiprovider but already have the correct infocube!
   l_partprov2 = l_s_variables-low.
*maybe we have a agglevel and must get the part provider
   CALL FUNCTION 'Z_RETURN_PARTPROV'
      EXPORTING
         i_infoprov = l_partprov2
      IMPORTING
         e_partprov = l_partprov
      EXCEPTIONS
         not_unique = 1
         OTHERS     = 2.
   IF sy-subrc <> 0.
      CLEAR l_partprov.
   ENDIF.
ELSE.
   CALL FUNCTION 'Z_RETURN_PARTPROV'
      EXPORTING
         i_infoprov = l_infoprov
      IMPORTING
         e_partprov = l_partprov
      EXCEPTIONS
         not_unique = 1
         OTHERS     = 2.
   IF sy-subrc <> 0.
      CLEAR l_partprov.
   ENDIF.
ENDIF.
*</span>if we could not decide which is the correct partprov we do not continue!
   IF l_partprov IS INITIAL.
      rfc_function_module_exit 0.
      EXIT.
   ENDIF.
***************************************************************************
****
* 2) Get Structure and Table Types and allocate memory
***************************************************************************
****
   CALL FUNCTION 'Z_CREATE_DATAREF'
      EXPORTING
         i_infoprov   = l_partprov
         i_type       = zco_tt_comments_line
      IMPORTING
         e_o_datadesc = l_r_datadesc.
   CREATE DATA l_r_to_comments_line TYPE HANDLE l_r_datadesc.
ASSIGN l_r_to_comments_line->* TO <x_to_comments>.

CALL FUNCTION 'Z_CREATE_DATAREF'
EXPORTING
  i_infoprov = l_partprov
  i_type = zco_ty_comments_line
IMPORTING
  e_o_datadesc = l_r_datadesc.
CREATE DATA l_r_s_comments_line TYPE HANDLE l_r_datadesc.
ASSIGN l_r_s_comments_line->* TO <l_s_comments>.

CREATE DATA l_r_s_comments_line TYPE HANDLE l_r_datadesc.
ASSIGN l_r_s_comments_line->* TO <l_s_comments_fix>.

CREATE DATA l_r_ts_comments_line LIKE STANDARD TABLE OF <l_s_comments>.
ASSIGN l_r_ts_comments_line->* TO <l_t_comments>.

***************************************************************************
****
* 3) Get characteristics
***************************************************************************
****
- how many characteristics do we have?
DESCRIBE TABLE i_t_object_names LINES l_iobjs.
CHECK l_iobjs IS NOT INITIAL.

- first get the fixed characteristics
  LOOP AT l_t_variables INTO l_s_variables.
  * we get the information of the underlying infoprov (might be multiprov, alvl etc...) in the infoprov field. If we have a multiprovider than
  * we also get the name of the part provider in the selection. we do not
  * want to transmit the infoprovider information to the read module
  * so prepare to get the proper name of the INFOCUNE!
    IF l_s_variables-name = c_field_infoprov OR l_s_variables-name = c_field_partprov .
    * fields have already been extracted
    CONTINUE.
  ENDIF.

* we can only work with single values!!
  CHECK l_s_variables-opt = 'EQ' and L_S_VARIABLES-SIGN = 'I'.

* we do not book on hierarchy nodes
  check l_s_variables-low(1) <> '+'.

  l_iobjnm = l_s_variables-name.
  CALL FUNCTION 'Z_COMNTS_FIELDS_CONVERT'
  EXPORTING
    i_iobjnm = l_iobjnm
  IMPORTING
    e_fieldname = l_fieldname.
  ASSIGN COMPONENT l_fieldname OF STRUCTURE <l_s_comments_fix> TO <comp>. 
IF sy-subrc = 0 AND <comp> IS ASSIGNED.
   l_value = l_s_variables-low.

   clear l_no_ex_in_conv.
   PERFORM convert_value using l_iobjnm l_no_ex_in_conv CHANGING l_value .
   * maybe we just have an infoobject we do not have in our data base -
     ignore it...
   <comp> = l_value.
   ENDIF.
ENDLOOP.

***************************************************************************
****
* 4) Get Comments
***************************************************************************
****
* set the fixed chars for the first entry:
   MOVE-CORRESPONDING <l_s_comments_fix> TO <l_s_comments>.

   CLEAR l_line.
   * loop over the table of characteristic/key figure values in the lines.
   LOOP AT i_t_member_names INTO l_s_member_names.
   * which characteristic are we at currently?
     l_index = sy-tabix MOD l_iobjs .
   * special case: last characteristic. Module creates a zero, we want the l_i_objs
     IF l_index = 0.
       l_index = l_iobjs.
     ENDIF.

   CLEAR l_object_name.
   READ TABLE i_t_object_names INTO l_object_name INDEX l_index.
   * first check whether we have a structure
     read table l_t_eltdir into l_s_eltdir with key eltuid = l_object_name.
     if sy-subrc = 0 and l_s_eltdir-deftp = 'STR'.
     * we have a structure
       READ TABLE l_t_eltdir into l_s_eltdir with key deftp = 'SEL'
       mapname = l_s_member_names-name.
       if sy-subrc = 0.
         l_key_id = l_s_eltdir-eltuid.
       else.
       * we already have the key!
         l_key_id = l_s_member_names-name.
       endif.
     * now get the selections
       loop at l_t_range into l_s_range where eltuid = l_key_id.
       if l_s_range-LOWFLAG = '3' or l_s_range-LOWFLAG = '6'.
     * we have a variable
read TABLE l_t_var into l_s_var with key varuniid = l_s_range-
low.
* we know the name of the variable
  clear l_s_variables.
  read table l_t_variables into l_s_variables with key name = l_s_var-
ar-vnam.
* replace the name of the variable by the name of the characteristic
  l_iobjnm = l_s_range-IOBJNM.
  l_value = l_s_variables-low.
* we can only work with single values!!
  CHECK l_s_variables-opt = 'EQ' and L_S_VARIABLES-SIGN = 'I'.
  else.
  * no variable, the value should be in l_s_range
    l_iobjnm = l_s_range-iobjnm.
    l_value = l_s_range-low.
* we can only work with single values!!
  CHECK l_s_range-opt = 'EQ' and L_S_range-SIGN = 'I'.
  endif.

CALL FUNCTION 'Z_COMNTS_FIELDS_CONVERT'
  EXPORTING
    i_iobjnm   = l_iobjnm
  IMPORTING
    e_fieldname = l_fieldname.

  ASSIGN COMPONENT l_fieldname OF STRUCTURE <l_s_comments> TO <comp>.
  * maybe we just have an infoobject we do not have in our data base -
  *null it...
  IF sy-subrc = 0 AND <comp> IS ASSIGNED.
  * if necessary convert the value
    l_no_ex_in_conv = 'X'.
    PERFORM convert_value using l_iobjnm l_no_ex_in_conv CHANGING l_v
    alue.
    <comp> = l_value.
  ENDIF.

  endloop.
  else.
  * we do not have a structure
  * convert the characteristic name into the name on the DB
    l_iobjnm = l_object_name.

  CALL FUNCTION 'Z_COMNTS_FIELDS_CONVERT'
    EXPORTING
      i_iobjnm   = l_iobjnm
    IMPORTING
      e_fieldname = l_fieldname.

  ASSIGN COMPONENT l_fieldname OF STRUCTURE <l_s_comments> TO <comp>.
  * maybe we just have an infoobject we do not have in our data base -
ignore it...

* if necessary convert the value

```plaintext
l_value = l_s_member_names-name.

if l_value = 'SUMME'.
  clear l_value.
  l_total_found = 'X'.
else.
  l_no_ex_in_conv = 'X'.
  PERFORM convert_value using l_iobjnm l_no_ex_in_conv CHANGING l_value.
endif.

<comp> = l_value.
ENDIF.
```

```plaintext
ENDIF.
```

* we have one full description of a cell

```plaintext
l_line = l_line + l.
ASSIGN COMPONENT 'LINE' OF STRUCTURE <l_s_comments> TO <comp>.
IF sy-subrc = 0 AND <comp> IS ASSIGNED.
  <comp> = l_line.
ENDIF.
```

* if we have a total we only insert the line if totals are allowed

```plaintext
if l_total_found is INITIAL or zco_no_totals is INITIAL.
  INSERT <l_s_comments> INTO TABLE <l_t_comments>.
endif.
clear l_total_found.
```

```plaintext
CLEAR <l_s_comments>.
```

* set the fixed values:
* set the fixed chars for the first entry:

```plaintext
MOVE-CORRESPONDING <l_s_comments_fix> TO <l_s_comments>.
```

```plaintext
ENDIF.
ENDLOOP.
```

* now get the comments.
```
<x_to_comments> = <l_t_comments>.
```

```plaintext
CALL FUNCTION 'Z_READ_COMMENTS'
EXPORTING
  i_handle = g_handle
  i_infoprov = l_partprov
CHANGING
  x_to_comments = <x_to_comments>.
```
* we receive the comments ordered by characteristics/key figure.
  * sort then again by line.
  
  `<l_t_comments> = <x_to_comments>.
  SORT <l_t_comments> BY ('LINE')`.

*get the number of necessary lines
  CLEAR l_index.
  CLEAR l_lines.
  * DESCRIBE TABLE <l_t_comments> LINES l_index.
  * READ TABLE <l_t_comments> INDEX l_index INTO <l_s_comments>.
  * ASSIGN COMPONENT 'LINE' OF STRUCTURE <l_s_comments> TO <comp>.
  * IF sy-subrc = 0 AND <comp> IS ASSIGNED.
  *  l_lines = <comp>.
  * ENDIF.
  DESCRIBE TABLE i_t_object_names lines l_index.
  DESCRIBE TABLE i_t_member_names lines l_lines.

  if l_index <> 0.
  l_lines = l_lines / l_index.
  endif.

  CLEAR l_index.
  CLEAR <l_s_comments>.
  DO l_lines TIMES.
  l_index = l_index + 1.
  READ TABLE <l_t_comments> INTO <l_s_comments> WITH KEY ('LINE') = l_index
  BINARY SEARCH.

  IF sy-subrc = 0.
  * we have a line with this line number
  * we have a maximum size of 250 chars.
   CLEAR l_s_value.

  DO.
  * move the first 250 chars.
   ASSIGN COMPONENT 'CMT' OF STRUCTURE <l_s_comments> TO <comp>.
   IF sy-subrc = 0 AND <comp> IS ASSIGNED.
   clear l_length.
   l_length = strlen( <comp> ).
   if l_length > 250.
   l_s_value-value = <comp>(250).
   shift <comp> by 250 places.
   else.
   l_s_value-value = <comp>.
   clear <comp>.
   endif.
   IF <comp> IS INITIAL.
  * we are at the end if the entry
   CLEAR l_s_value-concatenate_value.
   APPEND l_s_value TO e_t_values.
   EXIT. " exit do loop
BEGIN。
* we need to add a comment part
  l_s_value-concatenate_value = 'X'.
  APPEND l_s_value TO e_t_values.
  ENDF.
  ENDF.
ENDDO.

ELSE.
* no line with this number - insert an empty one.
  CLEAR l_s_value.
  APPEND l_s_value TO e_t_values.
  ENDF.
ENDDO.

* End of the rfc function module
  rfc_function_module_end 0.
ENDFUNCTION.

FUNCTION Z_RETURN_HANDLE.
*---------------------------------------------------------------------
** Local Interface:
** EXPORTING
**   REFERENCE(E_HANDLE) TYPE GUID_32
*---------------------------------------------------------------------
  e_handle = g_handle.
ENDFUNCTION.

FUNCTION Z_RETURN_PARTPROV.
*---------------------------------------------------------------------
** Local Interface:
** IMPORTING
**   REFERENCE(I_INFOPROV) TYPE RSINFOPROV
** EXPORTING
**   REFERENCE(E_PARTPROV) TYPE RSINFOPROV
** EXCEPTIONS
**   NOT_UNIQUE
*---------------------------------------------------------------------
  data: l_s_dta type RSD_S_DTA.
  DATA: lr_cl_rsd_dta TYPE REF TO cl_rsd_dta.
  clear e_partprov.
  try.
  *   cl_rsdrv_dta_buffer=>get_dta_info(
  *       EXPORTING
  *       i_infoprov = i_infoprov
IMPORTING 
   e_s_dta = l_s_dta).
*CATCH cx_root. "Ec NOCATCH
** cl_rsd_dta fills sy-msg fields...
* class cl_rsdrv_dta_buffer has been deleted in 7.40

cl_rsd_dta->factory(
   EXPORTING
       i_infoprov = i_infoprov
   RECEIVING
       r_r_dta = lr_cl_rsd_dta
   EXCEPTIONS
       not_found = 1
       OTHERS = 2).

lr_cl_rsd_dta->dta_get_info(
   IMPORTING
       e_s_dta = l_s_dta
   EXCEPTIONS
       dta_not_found = 1
       iobj_not_found = 2
       objvers_invalid = 3
       OTHERS = 4
).
*CATCH cx_root. "Ec NOCATCH

cl_rsd_dta fills sy-msg fields...

MESSAGE ID sy-msgid TYPE sy-msgty NUMBER sy-msgno
   WITH sy-msgv1 sy-msgv2 sy-msgv3 sy-msgv4
   RAISING not_unique.
ENDTRY.

case l_s_dta-cubetype.
  when rsd_c_cubetype-basic_ic.
    e_partprov = i_infoprov.
  when rsd_c_cubetype-virtual.
    if l_s_dta-cubesubtype = rsd_c_cubesubtype-simple_alvl or
       l_s_dta-cubesubtype = rsd_c_cubesubtype-alvl_part.
      e_partprov = l_s_dta-BASICCUBE.
    endif.
  when others.
endcase.
if e_partprov is initial.
   RAISE not_unique.
endif.

ENDFUNCTION.

FUNCTION z_save_comments.

** Local Interface:
** IMPORTING
** REFERENCE(I_HANDLE) TYPE GUID_32 OPTIONAL
** EXCEPTIONS
** FAILED
**

DATA: l_to_smemk TYPE zco_to_smemk,
      l_s_smemk TYPE zco_s_smemk,
      l_handle TYPE guid_32,
      l_s_buffer TYPE zcom_buf.

DATA: l_r_datadesc TYPE REF TO cl_abap_datadescr,
      l_r_cmt_buf TYPE REF TO data.

FIELD-SYMBOLS: <l_to_cmt_buf> TYPE ANY TABLE.

* what is the correct handle to use.
  IF g_handle IS NOT INITIAL.
    l_handle = g_handle.
  ELSE.
    l_handle = i_handle.
  ENDIF.

* no shared buffer accessible
  EXIT.
  ENDF.

* we save the comments for ALL underlying InfoCubes. thus go through the list of
* buffers and save each of the buffer instances to the matching db table
* read the key table
  select single * from zcom_buf into l_s_buffer WHERE id = l_handle.

* this entry contains the table with the links between the InfoCube and the buffer
* extract it
  if not l_s_buffer-map is INITIAL.
IMPORT tab = l_to_smemk FROM DATA BUFFER l_s_buffer-map.
endif.

LOOP AT l_to_smemk INTO l_s_smemk.
* create the type for the comment buffer table

CALL FUNCTION 'Z_CREATE_DATAREF'
  EXPORTING
    i_infoprov = l_s_smemk-infocube
    i_type = zco_tt_comments_buf
  IMPORTING
    e_o_datadesc = l_r_datadesc.

CREATE DATA l_r_cmt_buf TYPE HANDLE l_r_datadesc.
ASSIGN l_r_cmt_buf-* TO <l_to_cmt_buf>.

* get the corresponding buffer table
  clear l_s_buffer.
  select single * from zcom_buf into l_s_buffer WHERE id = l_s_smemk-guid.
  if not l_s_buffer-cmt_buf is initial.
    IMPORT tab = <l_to_cmt_buf> FROM DATA BUFFER l_s_buffer-cmt_buf.
  endif.

  IF <l_to_cmt_buf> IS NOT INITIAL.
    PERFORM save_comment_buffer_inst USING l_s_smemk-infocube
      CHANGING <l_to_cmt_buf>.
  ENDIF.

* we follow the same logic as in the planning delta buffer -
  delete the buffer
  * once the data has been saved and free the memory.
  delete from zcom_buf where id = l_s_smemk-guid.
ENDLOOP.

* as we want to use the shared object memory we have to release the memory.
* we follow the same logic as in the planning delta buffer -
  delete the buffer
* once the data has been saved.
  DELETE from zcom_buf where id = l_handle.

* We do not have an event to delete buffer entries when we leave the applica-
  tion.
* Thus we do delete all entries now which are too old to still be in used:
  anything older
  * than 2 days.
  data: l_date like sy-datum.

  l_date = sy-datum - 2.
delete from zcom_buf where buf_date < l_date.

* old version

* DATA: l_to_smemk TYPE zco_to_smek,
*     l_s_smek TYPE zco_s_smek,
*     l_handle TYPE guid_32.
* DATA: l_r_datadesc TYPE REF TO cl_abap_datadescr,
*     l_r_cmt_buf TYPE REF TO data.
* FIELD-SYMBOLS: <l_to_cmt_buf> TYPE ANY TABLE.

** what is the correct handle to use.
* IF g_handle IS NOT INITIAL.
*    l_handle = g_handle.
* ELSE.
*    l_handle = i_handle.
* ENDIF.

** what is the correct handle to use.
* IF l_handle IS INITIAL.
** no shared buffer accessible
*   EXIT.
* ENDIF.

** we save the comments for ALL underlying InfoCubes. thus go through the list of
** buffers and save each of the buffer instances to the matching db table

** get the buffer instance
* IMPORT tab = l_to_smek
*     FROM SHARED BUFFER indx(xy)
*     ID l_handle.
* LOOP AT l_to_smek INTO l_s_smek.
** get the corresponding buffer table

  CALL FUNCTION 'Z_CREATE_DATAREF'
  EXPORTING
  *     i_infoprov = l_s_smek-infocube
  *     i_type = zco_tt_comments_buf
  IMPORTING
  *     e_o_datadesc = l_r_datadesc.

  CREATE DATA l_r_cmt_buf TYPE HANDLE l_r_datadesc.
  ASSIGN l_r_cmt_buf->* TO <l_to_cmt_buf>.

** get the corresponding buffer table
* IMPORT tab = <l_to_cmt_buf>
* FROM SHARED BUFFER indx(ab)
* ID l_s_smek-guid.
* 
* IF <l_to_cmt_buf> IS NOT INITIAL.
*   PERFORM save_comment_buffer_inst USING l_s_smek-infocube
*     CHANGING <l_to_cmt_buf>.
* ENDIF.
* 
** as we want to use the shared object memory we have to release the memory
** we follow the same logic as in the planning delta buffer -
** once the data has been saved and free the shared mem.
* DELETE FROM SHARED BUFFER indx(ab) ID l_s_smek-guid.
* DELETE FROM SHARED BUFFER indx(cd) ID l_s_smek-guid.
* 
** ENDLOOP.
* 
** as we want to use the shared object memory we have to release the memory
** we follow the same logic as in the planning delta buffer -
** once the data has been saved and free the shared mem.
* DELETE FROM SHARED BUFFER indx(xy) ID l_handle.

ENDFUNCTION.

FUNCTION Z_SEL_EXIT.
*--------------------------------------------------------------------
** "Local Interface:
** IMPORTING
**   REFERENCE(I_INFOPROV) TYPE RSINFOPROV
**   REFERENCE(I_SERVICE) TYPE RSPLF_SRVNM
**   REFERENCE(I_SELOBJ) TYPE RSZCOMPID
**   CHANGING
**   REFERENCE(C_T_CHARSEL) TYPE RSPLF_T_CHARSEL
** EXCEPTIONS
**   SELECTION_EMPTY
*--------------------------------------------------------------------

* Right now the exit is only used to save comments. We use a dummy
* planning function that will not be executed.
* do not forget to register this function module in the rsadmin table!!

    check i_service = 'Z_DUMMY_SAVE'.

* just call the comment save module
    CALL FUNCTION 'Z_SAVE_COMMENTS'
    EXCEPTIONS
**How To... Build a Fast and Flexible Comment Solution for BEx Web Applications**

**November 2015**

```plaintext
FAILED = 1
OTHERS = 2

IF SY-SUBRC <> 0.
MESSAGE ID SY-MSGID TYPE SY-MSGTY NUMBER SY-MSGNO
WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4.
ENDIF.

we want to make sure that the planning function is not called at all.
raise Selection_empty.

ENDFUNCTION.

FUNCTION Z_UPDATE_COMMENTS .
**----------------------------------------------------------------------
**errmsg
**----------------------------------------------------------------------

DATA:
l_read_subrc TYPE c,
l_r_datadesc TYPE REF TO cl_abap_datadescr,
l_r_to_cmt_buf TYPE REF TO data,
l_r_s_cmt_buf TYPE REF TO data,
l_r_t_call_buf TYPE REF TO data,
l_hashkey TYPE hash160.

FIELD-SYMBOLS:
<l_to_cmt_buf> TYPE INDEX TABLE,
<l_t_call_buf> TYPE INDEX TABLE,
<l_s_cmt_buf> TYPE ANY,
<l_s_cmt> type any,
<comp> TYPE ANY.

* get the comment buffer table first
CALL FUNCTION 'Z_CREATE_DATAREF'
  EXPORTING
    i_infoprov = i_infoprov
  i_type = zco_tt_comments_buf
IMPORTING
  e_o_datadesc = l_r_datadesc.

CREATE DATA l_r_to_cmt_buf TYPE HANDLE l_r_datadesc.
ASSIGN l_r_to_cmt_buf->* TO <l_to_cmt_buf>.

* get the call buffer
CALL FUNCTION 'Z_CREATE_DATAREF'
  EXPORTING
    i_infoprov = i_infoprov
    i_type = zco_tt_call_buf
  IMPORTING
```
CREATE DATA l_r_t_call_buf TYPE HANDLE l_r_datadesc.
ASSIGN l_r_t_call_buf-* TO <l_t_call_buf>.

* get the work area for the comment buf
* get the comment buffer table first
CALL FUNCTION 'Z_CREATE_DATAREF'
  EXPORTING
    i_infoprov = i_infoprov
    i_type = zco_ty_comments_buf
IMPORTING
    e_o_datadesc = l_r_datadesc.

CREATE DATA l_r_s_cmt_buf TYPE HANDLE l_r_datadesc.
ASSIGN l_r_s_cmt_buf-* TO <l_s_cmt_buf>.

* check the buffer instance
PERFORM get_buffer_instance USING i_handle i_infoprov CHANGING g_handle <
  l_to_cmt_buf> <l_t_call_buf>.

loop at i_to_comments assigning <l_s_cmt>.
* add hash key for new entry
  MOVE-CORRESPONDING <l_s_cmt> TO <l_s_cmt_buf>.
  PERFORM create_hashkey
    USING
      i_infoprov
    CHANGING
      <l_s_cmt_buf>
      l_hashkey.
* let us check whether the comment is already in the buffer
  READ TABLE <l_to_cmt_buf> FROM <l_s_cmt_buf> TRANSPORTING NO FIELDS.
  l_read_subrc = sy-subrc.
* determine the right action flag
  ASSIGN COMPONENT 'CMT' OF STRUCTURE <l_s_cmt_buf> TO <comp>.
  IF sy-subrc = 0 AND <comp> IS INITIAL.
    ASSIGN COMPONENT 'ACTION' OF STRUCTURE <l_s_cmt_buf> TO <comp>.
    <comp> = c_del.
  ELSE.
    ASSIGN COMPONENT 'ACTION' OF STRUCTURE <l_s_cmt_buf> TO <comp>.
    <comp> = c_new.
  ENDIF.
  IF l_read_subrc = 0.
* there is an entry in the buffer, update the buffer entry
    MODIFY TABLE <l_to_cmt_buf> FROM <l_s_cmt_buf>.
  ELSE.
* no buffer entry yet
    ASSIGN COMPONENT 'ACTION' OF STRUCTURE <l_s_cmt_buf> TO <comp>.
IF <comp> = c_new.
    INSERT <l_s_cmt_buf> INTO TABLE <l_to_cmt_buf>.
ENDIF.
ENDIF. "sy-subrc

endloop.

* update the shared buffer
    PERFORM set_buffer_instance USING i_handle i_infoprov CHANGING g_handle <
    l_to_cmt_buf> <l_t_call_buf>.
ENDFUNCTION.

FUNCTION Z_COB_PRO_CMP_GET.

* **Local Interface:
** IMPORTING
**   REFERENCE(I_CHANM) TYPE RSD_CHANM
**   CHANGING
**       REFERENCE(X_T_COB_PRO) TYPE RSD_T_COB_PRO
**       REFERENCE(X_SUBRC) TYPE SY-SUBRC
*

    perform cob_pro_cmp_get using i_ch
       changing x_t_cob_pro x_subrc.

ENDFUNCTION.

6.4 Include LZ_COMMENTSF01

* &---------------------------------------------------------------------*
* & Form GET_BUFFER_INSTANCE
* &---------------------------------------------------------------------*
*
* --P_G_HANDLE  text
* --P_I_HANDLE  text
* --P_G_TO_CMT_BUF  text
*---------------------------------------------------------------------*

FORM get_buffer_instance USING i_handle TYPE guid_32
i_infoprov TYPE rsinfoprov
    CHANGING x_intern_handle TYPE guid_32
    x_to_cmt_buf TYPE ANY TABLE
    x_t_call_buf TYPE ANY TABLE.

DATA: l_handle TYPE guid_32,
    l_s_smemk TYPE zco_s_smemk,
    l_to_smemk TYPE zco_to_smemk,
    l_s_buffer type zcom_buf.
CLEAR x_to_cmt_buf.
CLEAR x_t_call_buf.

* what is the correct handle to use.
  IF x_intern_handle IS NOT INITIAL.
    l_handle = x_intern_handle.
  ELSE.
    l_handle = i_handle.
  ENDIF.

* maybe the handle is still empty as we do not have a buffer instance yet
  IF l_handle IS INITIAL.
    CALL FUNCTION 'GUID_CREATE'
      IMPORTING
        ev_guid_32 = l_handle.
    x_intern_handle = l_handle.
  ENDIF.

* we do not have to create a buffer instance in the shared buffer yet -
  * this is done when setting the buffer!
    EXIT.
  ENDIF.

* read the key table
  select single * from zcom_buf into l_s_buffer WHERE id = l_handle.
  IF sy-subrc <> 0.
  * no buffer instance yet.
    EXIT.
  ENDIF.

* this entry contains the table with the links between the InfoCube and the buffer
  * extract it
    if not l_s_buffer-map is initial.
      IMPORT tab = l_to_smemk FROM DATA BUFFER l_s_buffer-map.
    endif

    READ TABLE l_to_smemk INTO l_s_smemk WITH TABLE KEY infocube = i_infoprov.
    IF sy-subrc <> 0.
    * no buffer instance yet.
      EXIT.
    ENDIF.

* read the comments buffer
  clear l_s_buffer.
  select single * from zcom_buf into l_s_buffer WHERE id = l_s_smemk-guid.
  if not l_s_buffer-cmt_buf is INITIAL.
IMPORT tab = x_to_cmt_buf FROM DATA BUFFER l_s_buffer-cmt_buf.
endif.

* get the call buffer
if not l_s_buffer-call_buf is initial.
IMPORT tab = x_t_call_buf FROM DATA BUFFER l_s_buffer-call_buf.
endif.

* old version
*
* DATA: l_handle TYPE guid_32,
* l_s_smek TYPE zco_s_smek,
* l_to_smek TYPE zco_to_smek.
*
*
CLEAR x_to_cmt_buf.
CLEAR x_t_call_buf.
*
** what is the correct handle to use.
* IF x_intern_handle IS NOT INITIAL.
*  l_handle = x_intern_handle.
* ELSE.
*  l_handle = i_handle.
* ENDIF.
*
** maybe the handle is still empty as we do not have a buffer instance yet
* IF l_handle IS INITIAL.
*  CALL FUNCTION 'GUID_CREATE'
*    IMPORTING
*      ev_guid_32 = l_handle.
*
*  x_intern_handle = l_handle.
*
** we do not have to create a buffer instance in the shared buffer yet -
** this is done when setting the buffer!
*  EXIT.
* ENDIF.
*
*
** get the memory key table
IMPORT tab = l_to_smek
* FROM SHARED BUFFER index(xy)
* ID l_handle.
*
* READ TABLE l_to_smek INTO l_s_smek WITH TABLE KEY infocube = i_infopro
v.
*
* IF sy-subrc <> 0.
** no buffer instance yet.
*    EXIT.
* ENDIF.
*
** get the buffering tables
* IMPORT tab = x_to_cmt_buf
*  FROM SHARED BUFFER indx(ab)
*  ID l_s_smemk-guid.
*
* IMPORT tab = x_t_call_buf
*  FROM SHARED BUFFER indx(cd)
*  ID l_s_smemk-guid.

ENDFORM.           " GET_BUFFER_INSTANCE

FORM set_buffer_instance USING i_handle TYPE guid_32
  i_infoprov TYPE rsinfoprov
  CHANGING x_intern_handle TYPE guid_32
               x_to_cmt_buf TYPE ANY TABLE
               x_t_call_buf TYPE ANY TABLE.

DATA: l_handle TYPE guid_32.

DATA: l_s_smemk TYPE zco_s_smemk,
       l_to_smemk TYPE zco_to_smemk,
       l_s_buffer type zcom_buf.

* make sure that the tables are not empty:
  IF x_to_cmt_buf IS INITIAL AND x_t_call_buf IS INITIAL.
    EXIT.
  ENDIF.

* what is the correct handle to use.
  IF x_intern_handle IS NOT INITIAL.
    l_handle = x_intern_handle.
  ELSE.
    l_handle = i_handle.
  ENDIF.

  IF l_handle IS INITIAL.
  * no shared buffer accessible
    EXIT.
  ENDIF.

* read the key table
  clear l_s_buffer.
  select single * from zcom_buf into l_s_buffer WHERE id = l_handle.
* this entry contains the table with the links between the InfoCube and the buffer 
* extract it 
  if not l_s_buffer-map is initial. 
    IMPORT tab = l_to_smemk FROM DATA BUFFER l_s_buffer-map. 
  endif.

  READ TABLE l_to_smemk INTO l_s_smemp WITH TABLE KEY infocube = i_infoprov .

  * do we have an entry for the infocube already? If not update the smem key table
  IF sy-subrc <> 0.
  * no entry for the infocube yet - create it
    l_s_smemp-infocube = i_infoprov.

    CALL FUNCTION 'GUID_CREATE' 
      IMPORTING 
        ev_guid_32 = l_s_smemp-guid.

    INSERT l_s_smemp INTO TABLE l_to_smemk .

  * set the buffer
    EXPORT tab = l_to_smemk TO data BUFFER l_s_buffer-map. 
    l_s_buffer-id = l_handle. 
    l_s_buffer-buf_date = sy-datum. 
    Modify zcom_buf from l_s_buffer.
  ENDIF.

  * export the two buffer tables - 
  we do not have to update them as they will be 
  * overwritten automatically 
    clear l_s_buffer.
    l_s_buffer-id = l_s_smemp-guid. 
    l_s_buffer-buf_date = sy-datum. 

    EXPORT tab = x_to_cmt_buf TO data BUFFER l_s_buffer-cmt_buf. 
    EXPORT tab = x_t_call_buf TO data BUFFER l_s_buffer-call_buf. 

    Modify zcom_buf from l_s_buffer .

* Old Version
* 
* 
* DATA: l_handle TYPE guid_32.
* 
* DATA: l_s_smemp TYPE zco_s_smemp, 
*   l_to_smemk TYPE zco_to_smemk. 
* 
** make sure that the tables are not empty:
* IF x_to_cmt_buf IS INITIAL AND x_t_call_buf IS INITIAL.
  * EXIT.
  * ENDF.
  *
** what is the correct handle to use.
* IF x_intern_handle IS NOT INITIAL.
  * l_handle = x_intern_handle.
  * ELSE.
  * l_handle = i_handle.
  * ENDF.
  *
* IF l_handle IS INITIAL.
** no shared buffer accessible
  * EXIT.
  * ENDF.
  *
** get the table of the keys from the shared memory
* IMPORT tab = l_to_smemk
  FROM SHARED BUFFER indx(xy)
  ID l_handle.
  *
** do we have any entry in the key table yet? (remember -
maybe the table is empty)
* READ TABLE l_to_smemk INTO l_s_smemk WITH TABLE KEY infocube = i_infopro
  v.
  *
** do we have an entry for the infocube already? If not update the smem key
  table
* IF sy-subrc <> 0.
** no entry for the infocube yet - create it
  * l_s_smemk-infocube = i_infoprov.
  *
    CALL FUNCTION 'GUID_CREATE'
    IMPORTING
    * ev_guid_32 = l_s_smemk-guid.
    *
    INSERT l_s_smemk INTO TABLE l_to_smemk.
    *
** set the shared buffer.
* EXPORT tab = l_to_smemk
  TO SHARED BUFFER indx(xy)
  ID l_handle.
  *
  ENDF.
  *
** export the two buffer tables -
we do not have to update them as they will be
** overwritten automatically
* EXPORT tab = x_to_cmt_buf
  TO SHARED BUFFER indx(ab)
  ID l_s_smemk-guid.
EXPORT tab = x_t_call_buf
TO SHARED BUFFER indx(cd)
ID l_s_smek-guid.

ENDFORM.

" SET_BUFFER_INSTANCE

DATA: l_value type rschavl,
      l_chanm type RSD_CHANM,
      l_val_in type RSD_CHAVL_EXT,
      l_val_out type RSD_CHAVL_EXT,
      l_t_cob_pro_cmp TYPE rsd_t_cob_pro,
      l_s_cob_pro TYPE rsd_s_cob_pro,
      l_subrc like sy-subrc.

l_chanm = i_iobjnm.
l_val_in = x_value.
l_val_out = x_value.

* get the cobpro
CALL FUNCTION 'Z_COB_PRO_CMP_GET'
  EXPORTING
    I_CHANM = l_chanm
  CHANGING
    X_T_COB_PRO = l_t_cob_pro_cmp
    X_SUBRC = l_subrc.

* if we cannot get a cobpro the we do not have a proper characteristic
if l_subrc <> 0.
  exit.
endif.

** Due to an error in the java we receive some values with the name of the variable
** (see also coding for i_t_variables).
* DATA:l_string1 TYPE string,
  l_string2 TYPE string.
* SPLIT l_val_in AT ' ' INTO l_string1 l_string2.
* if l_string2 is initial.
  l_val_in = l_string1.
* else.
  *   shift l_string2 LEFT DELETING LEADING ' '.
  * _val_in = l_string2.
  * endif.

* maybe we have a compound characteristic - do standard conversion

CALL FUNCTION 'RRSV_EXT_CHA_VAL_SPLIT'

  EXPORTING
    I_CHANM = l_chanm
    I_CHAVL_EXT = _val_in

  IMPORTING
    E_CHAVL_EXT_NC = _val_out

  EXCEPTIONS
    UNKNOWN_CHANM = 1
    UNKNOWN_INFOCUBE = 2
    INVALID_CHAVL = 3
    OTHERS = 4.

IF SY-SUBRC <> 0.
  * In some cases we get a wrongly concatenated compound value from the
  * front end - no delimiter. We try to correct this situation
  _val_out = _val_in.

  PERFORM correct_compound_chars
    USING l_chanm l_t_cob_pro_cmp changing _val_out _subrc.
    if _subrc <> 0.
      exit.
    endif.
  ENDIF.

  _value = _val_out.

* Now we have to check whether we have an initial value - if so we convert
* it to '#' and also store '#' on the data base. In the old BPS solution same
* technique was used. We need this to destiguish between '#' and (sub-
* total
* values.

* we have a character type characteristic
* do we use the '#' logic?
  if _value = ' '.
    if not zco_use_hash is initial.
      _value = '#'.
    endif.
  endif.
  exit.
endif.
l_value = x_value.

if i_no_ex_in_conv is initial.
* we need to convert the external value

CALL FUNCTION 'RRSV_IOBJ_VALUE_OUTPUT'
EXPORTING
  I_IOBJNM       = I_iobjnm
  I_IOBJVL_EXT   = l_value
  I_FORMAT       = RSR_C_FORMAT-EXTERN
  I_CONCAT       = '
  I_CHECK        = RS_C_FALSE
  I_TEXT         = RS_C_FALSE
  I_SID_REQUESTED = RS_C_FALSE
IMPORTING
  E_IOBJVL_INT_NC = x_value
EXCEPTIONS
  UNKNOWN_IOBJ   = 1
  INVALID_FORMAT = 2.
if sy-subrc <> 0.
  exit.
endif.

else.
* in some cases we need to convert already internal value -
  in case of alpha conversion
  clear l_s_cob_pro.
  read TABLE l_t_cob_pro_cmp into l_s_cob_pro with key IOBJNM = l_chanm.
  if l_s_cob_pro-convexit = 'ALPHA'.

CALL FUNCTION 'RRSV_IOBJ_VALUE_OUTPUT'
EXPORTING
  I_IOBJNM       = I_iobjnm
  I_IOBJVL_EXT   = l_value
  I_FORMAT       = RSR_C_FORMAT-EXTERN
  I_CONCAT       = '
  I_CHECK        = RS_C_FALSE
  I_TEXT         = RS_C_FALSE
  I_SID_REQUESTED = RS_C_FALSE
IMPORTING
  E_IOBJVL_INT_NC = x_value
EXCEPTIONS
  UNKNOWN_IOBJ   = 1
  INVALID_FORMAT = 2.
if sy-subrc <> 0.
  exit.
endif.
endif.
we have to treat ‘#’ values as well.
if x_value = ‘#’ and zco_use_hash is initial.
clear x_value.
endif.
ENDFORM.

" CONVERT_VALUE

\*&~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\*
\*&     Form GET_COMMENTS_TAB
\*&~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\*
\*     text
\*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\*
\*  --> p1   text
\*  <-- p2   text
\*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\*
FORM get_comments_tab USING i_infoprovider TYPE rsinfoprov
   CHANGING x_tablename TYPE tablename16.

   CLEAR x_tablename.
   SELECT SINGLE comments_tab FROM zmap_iprov_com INTO x_tablename WHERE info_prov = i_infoprovider.
ENDFORM.

" GET_COMMENTS_TAB

\*&~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\*
\*&     Form SAVE_COMMENT_BUFFER_INST
\*&~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\*
\*     text
\*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\*
\*  <--P_L_TO_CMT_BUF   text
\*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\*
FORM save_comment_buffer_inst USING i_infoprov TYPE rsinfoprov
   CHANGING x_to_cmt_buf TYPE ANY TABLE.

DATA: l_r_datadesc TYPE REF TO cl_abap_datadescr,
       l_r_t_new TYPE REF TO data,
       l_r_t_del TYPE REF TO data,
       l_r_db TYPE REF TO data,
       l_hashkey TYPE hash160.

DATA: l_name_db TYPE tablename16.

FIELD-SYMBOLS: <l_s_cmt_buf> TYPE ANY,
               <l_s_db> TYPE ANY,
               <l_t_new> TYPE ANY TABLE,
               <l_t_del> TYPE ANY TABLE,
               <l_action> TYPE ANY,
               <l_cmt> TYPE ANY.

PERFORM get_comments_tab USING i_infoprov CHANGING l_name_db.
CALL FUNCTION 'Z_CREATE_DATAREF' 
EXPORTING 
  i_infoprov  = i_infoprov 
  i_type      = zco_tt_comments_db 
IMPORTING 
  e_o_datadesc = l_r_datadesc.

* we need the same type twice... 
CREATE DATA l_r_t_new TYPE HANDLE l_r_datadesc. 
ASSIGN l_r_t_new->* TO <l_t_new>.

CREATE DATA l_r_t_del TYPE HANDLE l_r_datadesc. 
ASSIGN l_r_t_del->* TO <l_t_del>.

CALL FUNCTION 'Z_CREATE_DATAREF' 
EXPORTING 
  i_infoprov  = i_infoprov 
  i_type      = zco_ty_comments_db 
IMPORTING 
  e_o_datadesc = l_r_datadesc.

CREATE DATA l_r_db TYPE HANDLE l_r_datadesc. 
ASSIGN l_r_db->* TO <l_s_db>.

LOOP AT x_to_cmt_buf ASSIGNING <l_s_cmt_buf>. 
  ASSIGN COMPONENT 'ACTION' OF STRUCTURE <l_s_cmt_buf> TO <l_action>. 
  ASSIGN COMPONENT 'CMT' OF STRUCTURE <l_s_cmt_buf> TO <l_cmt>. 

CHECK <l_action> IS NOT INITIAL. 

MOVE-CORRESPONDING <l_s_cmt_buf> TO <l_s_db>. 

* special logic - if a comment is empty we also delete it! 
  IF <l_cmt> IS INITIAL. 
    INSERT <l_s_db> INTO TABLE <l_t_del>. 
    DELETE TABLE x_to_cmt_buf FROM <l_s_cmt_buf>. 
    CONTINUE. 
  ENDIF.
  IF <l_action> = c_new. 
    * we have a new comment - insert it into DB 
    INSERT <l_s_db> INTO TABLE <l_t_new>. 
    CLEAR <l_action>. 
    * modify not necessary as we are using a field symbol 
    ELSEIF <l_action> = c_del. 
    * we want to delete the comment from the data base 
    INSERT <l_s_db> INTO TABLE <l_t_del>. 
    DELETE TABLE x_to_cmt_buf FROM <l_s_cmt_buf>. 
  ENDIF.
ENDLOOP.

IF NOT <l_t_new> IS INITIAL.
* insert new records and update existing records - both can be done with
* one statement: 'modify'.

  MODIFY (l_name_db) FROM TABLE <l_t_new>.
  CLEAR <l_t_new>.

ENDIF.

IF NOT <l_t_del> IS INITIAL.
* delete existing records -
  DELETE (l_name_db) FROM TABLE <l_t_del>.
  CLEAR <l_t_del>.
ENDIF.

ENDFORM.  " SAVE_COMMENT_BUFFER_INST

*---------------------------------------------------------------------
* Form Create_hashkey
*---------------------------------------------------------------------
* text
*---------------------------------------------------------------------
* --> p1 text
* <-- p2 text
*---------------------------------------------------------------------
FORM create_hashkey USING i_infoprov TYPE rsinfoprov
  CHANGING x_data_struct TYPE any
       x_hashkey TYPE hash160.

DATA: l_o_datadesc TYPE REF TO cl_abap_datadesc,
       l_r_comments_db TYPE REF TO data,
       l_xstring TYPE xstring,
       l_hashkey TYPE hash160,
       l_hashlength TYPE i.

FIELD-SYMBOLS: <l_s_comments_db> TYPE ANY,
                 <l_haskey_component> TYPE hash160,
                 <l_comment_component> TYPE string.

CLEAR x_hashkey.

* Create Data Type
CALL FUNCTION 'Z_CREATE_DATAREF'
  EXPORTING
     i_infoprov = i_infoprov
     i_type = zco_ty_comments_db
IMPORTING
     e_o_datadesc = l_o_datadesc.
CREATE DATA l_r_comments_db TYPE HANDLE l_o_datadesc.
ASSIGN l_r_comments_db->* TO <l_s_comments_db>.
How To... Build a Fast and Flexible Comment Solution for BEx Web Applications

Move corresponding because then also other super data structures can be used as import

MOVE-CORRESPONDING x_data_struct TO <l_s_comments_db>.

If structure contains comment (CMT) or hashkey (ID), remove it before generating hash key.

ASSIGN COMPONENT 'CMT' OF STRUCTURE <l_s_comments_db> TO <l_comment_component>.
CLEAR <l_comment_component>.
ASSIGN COMPONENT 'ID' OF STRUCTURE <l_s_comments_db> TO <l_haskey_component>.
CLEAR <l_haskey_component>.

Transform to xstring

EXPORT <l_s_comments_db> TO DATA BUFFER l_xstring.

Compute Hash Key

CALL FUNCTION 'CALCULATE_HASH_FOR_RAW'
EXPORTING
  alg = 'SHA1'
data = l_xstring
IMPORTING
  hash = l_hashkey
  hashlen = l_hashlength
EXCEPTIONS
  unknown_alg = 1
  param_error = 2
  internal_error = 3
  OTHERS = 4.
IF sy-subrc = 0.
x_hashkey = l_hashkey(l_hashlength).
ASSIGN COMPONENT 'ID' OF STRUCTURE x_data_struct TO <l_haskey_component>.
<l_haskey_component> = x_hashkey.
ENDIF.

ENDFORM.

" Create_hashkey
*---------------------------------------------------------------------*
* Form REPLACE_STRUCTURES
*---------------------------------------------------------------------*
* text
* ---------------------------------------------------------------------*
* <-P_L_T_VARIABLES text
* <-P_L_T_MEMBER_NAMES text
* <-P_L_T_OBJECT_NAMES text
*----------------------------------------------------------------------*
FORM REPLACE_STRUCTURES_var CHANGING X_T_VARIABLES TYPE BICS_PROV_STATE_T
  X_INIT_VARIABL
    x_t_eltdir type rxx0_t_eltdir
    x_t_range type rxx0_t_range
    x_t_var type rxx0_t_var .
**How To... Build a Fast and Flexible Comment Solution for BEx Web Applications**

**November 2015**

```plaintext

data: l_s_eltdir type RSZ_X_ELTDIR,
      l_s_range type rsz_x_range,
      l_s_var type RSZ_X_VAR,
      l_s_variables like line of x_t_variables,
      l_s_variables2 like line of x_t_variables,
      l_key_id type SYSUUID_25,
      l_compid type RSZCOMPID,
      l_subrc like sy-subrc,
      l_timvl type RSCHAVL,
      l_varnt type T009B-PERIV,
      l_shift type i.

* get the name of the query
  read TABLE x_t_variables into l_s_variables with key name = c_field_query.

  l_compid = l_s_variables-low.
* Get the information about the query from the data base
  CALL FUNCTION 'RSZ_X_COMPONENT_GET'
    EXPORTING
      I_COMPID = l_compid
    IMPORTING
      E_SUBRC = l_subrc
    TABLES
      C_T_ELTDIR = x_t_eltdir
      C_T_RANGE = x_t_range
      c_t_var = x_t_var.

* find the structure with the keyfigures
  clear l_s_variables.
  LOOP at x_t_variables into l_s_variables.
  * first check whether we have a structure
    read table x_t_eltdir into l_s_eltdir with key eltuid = l_s_variables-name.
    if sy-subrc = 0 and l_s_eltdir-deftp = 'STR'.
      delete table x_t_variables from l_s_variables.
      exit.
    endif.
* maybe we have a fixed name.
  read table x_t_eltdir into l_s_eltdir with key mapname = l_s_variables-name.
  if sy-subrc = 0 and l_s_eltdir-deftp = 'STR'.
    delete table x_t_variables from l_s_variables.
    exit.
  endif.

  if l_s_variables-name cs zco_c_kyfstruc_query.
* delete x_t_variables INDEX sy-tabix.
* exit.
```
* endif.
  endloop.

if not l_s_variables is initial.
* we have the name OR the key of the keyfigure
* we find the id for the selections in table eltdir
  READ TABLE x_t_eltdir into l_s_eltdir with key deftp = 'SEL'
    mapname = l_s_variables-low.

if sy-subrc = 0.
  l_key_id = l_s_eltdir-eltuid.
else.
* we already have the key!
  l_key_id = l_s_variables-low.
endif.

* now get the selections
  loop at x_t_range into l_s_range where eltid = l_key_id.
  
    clear l_s_variables.
    if l_s_range-LOWFLAG = '3' or l_s_range-LOWFLAG = '6'.
* we have a variable
      read TABLE x_t_var into l_s_var with key varuniid = l_s_range-low.
* we know the name of the variable
      read table x_t_variables into l_s_variables with key name = l_s_var-vnam.
* replace the name of the variable by the name of the characteristic
    l_s_variables-name = l_s_range-IOBJNM.
* maybe we have a time characteristics and a shift!!
  if l_s_range-shiftlow <> '0'.
* check whether we know the fiscvarnt
    clear l_varnt.
    read table x_t_variables into l_s_variables2 with key name = '0FI
    SCVARNT'.
  
    if sy-subrc = 0.
      l_varnt = l_s_variables2-low.
    endif.

    l_shift = l_s_range-shiftlow.

CALL FUNCTION 'RST_TOBJ_SHIFT'
  EXPORTING
    I_TIMNM = l_s_range-objjnm
    I_TIMVL = l_s_variables-low
    I_SHIFT = l_shift
    I_EXTRA_PERIOD = RS_C_FALSE
    I_FISCVARNT = l_varnt
    I_INFOCUBE =

  IMPORTING
    E_TIMVL = l_timvl

  EXCEPTIONS
    NO_FISCVARNT = 1
OUT_OF_RANGE = 2
NOT_NUMERIC = 3
WRONG_CLIENT = 4
INVALID_FORMAT = 5
X_MESSAGE = 6
OTHERS = 7

IF SY-SUBRC = 0.
  * if the shift was correctly we use the result
    l_s_variables-low = l_timvl.
  ENDIF.
  endif.
else.
  l_s_variables-name = l_s_range-IOBJNM.
  l_s_variables-sign = l_s_range-sign.
  l_s_variables-opt = l_s_range-opt.
  l_s_variables-low = l_s_range-low.
  endif.

* maybe we have more than one entry for the characteristic. Delete all of them
  Delete x_t_variables where name = l_s_variables-name.
  INSERT l_s_variables INTO TABLE x_t_variables.
  endloop.
  endif.
ENDFORM.

" REPLACE_STRUCTURES
*---------------------------------------------------------------------*
* Form CORRECT_COMPOUND_CHARS                                    *
*---------------------------------------------------------------------*
* text                                                             *
* -------------------------------------------------------------------*
* -->P_L_CHANM  text                                              *
* <--P_L_VAL_OUT  text                                            *
* <--P_L_SUBRC  text                                             *
*---------------------------------------------------------------------*
FORM CORRECT_COMPOUND_CHARS USING I_CHANM type RSD_CHANM
  i_t_cob_pro_cmp TYPE rsd_t_cob_pro
  CHANGING X_VAL type RSD_CHAVL_EXT
  X_SUBRC type sy-subrc.

DATA: l_s_cob_pro  TYPE rsd_s_cob_pro,
  L_LENGTH TYPE I.

clear x_subrc.

read TABLE i_t_cob_pro_cmp into l_s_cob_pro with key IOBJNM = i_chanm.

if sy-subrc <> 0.
  x_subrc = 1.
  exit.
endif.
* get the last n digits of the value
l_length = strlen( x_val ) - l_s_cob_pro-OUTPUTLEN.

shift x_val left by l_length places.

ENDFORM.  " CORRECT_COMPOUND_CHARS
* & Form COB_PRO_CMP_GET
* &---------------------------------------------------------------*
* text
* ------------------------------------------------------------------*
* -->P_I_CHANM text
* <--P_L_T_COB_PRO_CMP text
* <--P_X_SUBRC text
* ------------------------------------------------------------------*
FORM COB_PRO_CMP_GET USING I_CHANM type RSD_CHANM
   CHANGING X_T_COB_PRO type rsd_t_cob_pro
      X_SUBRC like sy-subrc.

TYPES: BEGIN OF g_sx_cpb,
     chanm TYPE rscanm,
     subrc TYPE sy-subrc,
     cob_pro TYPE rsd_t_cob_pro,
END OF g_sx_cpb.

STATICS: s_tsx_cpb TYPE SORTED TABLE OF g_sx_cpb
         WITH UNIQUE KEY chanm.
DATA: l_sx_cpb TYPE g_sx_cpb.

READ TABLE s_tsx_cpb INTO l_sx_cpb
      WITH TABLE KEY chanm = i_chanm.

IF sy-subrc GT 0.
   CLEAR l_sx_cpb.
   l_sx_cpb-chanm = ichanm.

* get the actual cob_pro

DATA: l_s_cob_pro TYPE rsd_s_cob_pro,
      l_s_cob_pro_cmp TYPE rsd_s_cob_pro,
      l_t_iobj_cmp TYPE rsd_t_iobj_cmp.
FIELD-SYMBOLS:
   <l_s_iobj_cmp> TYPE rsd_s_iobj_cmp.

try.

l_s_cob_pro = cl_rsr->get_cob_pro( i_objnm = l_s_cpb-chanm ).
IF l_s_cob_pro-cmpsta NE rsd_c_cmpsta-ini.
   l_t_iobj_cmp = cl_rsr->get_iobj_cmp( i_objnm = l_s_cpb-chanm ).
LOOP AT l_t_iobj_cmp ASSIGNING <l_s_iobj_cmp>.
l_s_cob_pro_cmp = cl_rsr=>get_cob_pro( i_objnm = l_s_iobj_cmp=>iobjcmp ).
    APPEND l_s_cob_pro_cmp TO l_sx_cpb-cob_pro.
ENDLOOP.
ENDIF.
APPEND l_s_cob TO l_sx_cpb-cob_pro.

INSERT l_sx_cpb INTO TABLE s_tsx_cpb.
catch cx_rsr_cob_pro_not_found.
    l_sx_cpb-subrc = 1.
endtry.

ENDIF.
x_subrc = l_sx_cpb-subrc.
x_t_cob_pro = l_sx_cpb-cob_pro.
ENDFORM.  " COB_PRO_CMP_GET

6.5  Type Group ZCO

TYPE-POOL zco.

TYPES: BEGIN OF zco_s_smemk,
    infocube TYPE rsinfoprov,
    guid TYPE guid_32,
END OF zco_s_smemk.

TYPES: zco_to_smemk TYPE SORTED TABLE OF zco_s_smemk WITH UNIQUE KEY infocube.

CONSTANTS: zco_c_kyfstruc_query(20) TYPE c VALUE 'PLAN_STRUCTURE'.

CONSTANTS: zco_name_infoprov(10) TYPE c VALUE '0INFOPROV'.

CONSTANTS: zco_ty_comments_db TYPE string VALUE 'TY_COMMENTS_DB',
zco_ty_comments_line TYPE string VALUE 'TY_COMMENTS_LINE',
zco_ty_comments_buf TYPE string VALUE 'TY_COMMENTS_ACTION',
zco_tt_comments_db TYPE string VALUE 'TT_COMMENTS_DB',
zco_tt_comments_line TYPE string VALUE 'TT_COMMENTS_LINE',
zco_tt_comments_buf TYPE string VALUE 'TT_COMMENTS_ACTION',
zco_ty_call_buf TYPE string VALUE 'TY_CALL_BUF',
zco_tt_call_buf TYPE string VALUE 'TT_CALL_BUF',
zco_ty_shared_mem TYPE string VALUE 'TY_SHARED_MEM',
zco_tt_shared_mem TYPE string VALUE 'TT_SHARED_MEM'.

CONSTANTS: zco_use_hash type c VALUE ' ',
zco_no_totals type c value 'X'.

6.6  BSP Page Layout Tab

<%@page language="abap" %>
<%@extension name="htmlb" prefix="htmlb" %>
<head>
<script language="Javascript">
    var isOpen = false;
    function changeScreenSize(w,h)
    {
        window.resizeTo( w, h );
    }
    function delete_comment() {
        if (window.confirm("Are you sure you want to delete the comment?"))
        {
            window.open("z_edit_comments.htm?delete_flag=X","edit_comment");
            id = "BUTTON_GROUP_ITEM_1_btn0_acButton";
            window.parent.opener.document.getElementById(id).click();

            id = "BUTTON_GROUP_ITEM_1_btn2_acButton";
            window.parent.opener.document.getElementById(id).click();

            window.popup=window.open("","edit_comment");
            window.popup.close();
        }
    }
    function cancel() {
        if (window.confirm("You are sure you want to cancel editing the comment?"))
        {
            id = "BUTTON_GROUP_ITEM_1_btn2_acButton";
            window.parent.opener.document.getElementById(id).click();

            window.popup=window.open("","edit_comment");
            window.popup.close();
        }
    }
    function close_window() {
        if (isOpen == true)
        {
            id = "BUTTON_GROUP_ITEM_1_btn2_acButton";
            window.parent.opener.document.getElementById(id).click();

            window.popup=window.open("","edit_comment");
            window.popup.close();

            isOpen = false;
        }
    }
    function accept()
    {
        if (window.parent.opener != null)
        {
            id = "BUTTON_GROUP_ITEM_1_btn0_acButton";
            window.parent.opener.document.getElementById(id).click();
        }
    }
</script>
</head>
id = "BUTTON_GROUP_ITEM_1_btn2_acButton"
    window.parent.opener.document.getElementById(id).click();
}

window.popup=window.open("","edit_comment");
window.popup.close();
}

function set_name() {
    var liBehindFirstDot = location.hostname.indexOf( "." ) + 1;
    if (liBehindFirstDot > 0) {
        document.domain = location.hostname.substr( liBehindFirstDot );
    }
    window.name="edit_comment";
    changeScreenSize(500,350);
    isOpen = true;
    id = "BUTTON_GROUP_ITEM_1_btn1_acButton"
    window.parent.opener.document.getElementById(id).click();
}

</script>
</head>
<body onload="set_name();" onunload="close_window();">
<htmlb:content design="design2003" >
<htmlb:page title="Create/Change Comment ">
<htmlb:form>
<htmlb:textEdit id = "textedit"
    cols = "80"
    rows = "8"
    tooltip = "Create or change the selected comment"
    wrapping = "HARD"
    text = "<%= comment %>">

<table>
<tr>
<td>
<htmlb:button id = "ACCEPT"
    text = "Accept Changes"
    onClick = "ACCEPT"
    onClientClick = "javascript:accept()"
    tooltip = "Accept changes and close the window" />

<htmlb:button id = "CANCEL"
    text = "Cancel"
    onClientClick = "javascript:cancel()"
    tooltip = "Cancel without accepting the changes" />

<htmlb:button id = "DELETE"
    text = "Delete"
    onClick = "DELETE"
    onClientClick = "javascript:delete_comment()"
    tooltip = "Delete the comment" />

</td>
</tr>
</table>
</htmlb:form>
</htmlb:page>
</htmlb:content>
6.7 BSP Page OnInitialization

* event handler for data retrieval

DATA: lt_selected_fields TYPE STANDARD TABLE OF string,
  l_field_name TYPE string,
  l_field_value TYPE string,
  l_infocube type rsinfoprov.

data: l_r_datadesc TYPE REF TO CL_ABAP_DATADESCR,
  l_r_s_comment type ref to data,
  l_r_to_comments type ref to data.

FIELD-SYMBOLES: <l_to_comments> type any table,
  <l_s_comment> type any.

FIELD-SYMBOLES: <l_sel_field> TYPE string,
  <l_comment_component> TYPE ANY.

CONSTANTS: l_c_field_field_sep(1) TYPE c VALUE '6', " in URL: '%26',
  l_c_field_value_sep(1) TYPE c VALUE '='. " in URL: '%3D'.

* we need a work area of the right type.
if delete_flag is not initial.
* if we are in the delete case we are called without the url parameter for
  selection.
* use the internal buffered values instead
  SPLIT g_selection AT l_c_field_field_sep INTO TABLE lt_selected_fields.
  selection = g_selection.
else.
  SPLIT selection AT l_c_field_field_sep INTO TABLE lt_selected_fields.
endif.

* we know that the first entry in the selection is the infoprov
read table lt_selected_fields index 1 assigning <l_sel_field>. 
check <l_sel_field> is assigned.
SPLIT <l_sel_field> AT l_c_field_value_sep INTO l_field_name l_field_value.
IF l_field_name <> zco_name_infoprov.
  * something deeply wrong...
    exit.
endif.
g_infoprov = l_field_value.
CALL FUNCTION 'Z_RETURN_PARTPROV'
  EXPORTING
      I_INFOPROV = g_infoprov
  IMPORTING
      E_PARTPROV = l_infocube
  EXCEPTIONS
      NOT_UNIQUE = 1
      OTHERS = 2.
  IF SY-SUBRC <> 0.
    exit.
  ENDIF.
g_infoprov = l_infocube.

* now get the line type
CALL FUNCTION 'Z_CREATE_DATAREF'
  EXPORTING
      I_INFOPROV = g_infoprov
      I_TYPE = zco_ty_comments_line
  IMPORTING
      E_O_DATADESC = l_r_datadesc.
CREATE DATA l_r_s_comment TYPE HANDLE l_r_datadesc.
assign l_r_s_comment->* to <l_s_comment>.

CALL FUNCTION 'Z_CREATE_DATAREF'
  EXPORTING
      I_INFOPROV = g_infoprov
      I_TYPE = zco_tt_comments_line
  IMPORTING
      E_O_DATADESC = l_r_datadesc.
CREATE DATA l_r_to_comments TYPE HANDLE l_r_datadesc.
assign l_r_to_comments->* to <l_to_comments>.

* now fill the structure with the selection for the comment
LOOP AT lt_selected_fields ASSIGNING <l_sel_field> .
  SPLIT <l_sel_field> AT l_c_field_value_sep INTO l_field_name l_field_value.
  IF l_field_name = zco_name_infoprov.
    CONTINUE.
  ENDIF.

  ASSIGN COMPONENT l_field_name OF STRUCTURE <l_s_comment> TO <l_comment_component>.
  CHECK SY-SUBRC = 0 and 
      <l_comment_component> IS ASSIGNED.
  <l_comment_component> = l_field_value.
ENDLOOP.

assign component 'LINE' of STRUCTURE <l_s_comment> TO <l_comment_component> .
<l_comment_component> = 1.

* we now have a correct work area
IF delete_flag IS NOT INITIAL.
   CLEAR delete_flag.
   assign COMPONENT 'CMT' of STRUCTURE <l_s_comment> TO <l_comment_component>.
   clear <l_comment_component>
   INSERT <l_s_comment> INTO TABLE <l_to_comments>.
   CALL FUNCTION 'Z_DELETE_COMMENTS'
   EXPORTING
   i_to_comments = <l_to_comments>
   i_handle = handle
   i_infoprov = g_infoprov.

* if we are in a pure reporting scenario we want to save directly -
* please uncomment the following coding
*   CALL FUNCTION 'Z_SAVE_COMMENTS'
*   EXPORTING
*   i_HANDLE = handle
*   EXCEPTIONS
*   FAILED = 1
*   OTHERS = 2.
*   IF SY-SUBRC <> 0.
*     MESSAGE ID SY-MSGID TYPE SY-MSGTY NUMBER SY-MSGNO
*     WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4.
*   ENDIF.

   EXIT.
ENDIF.

* 1) we are not in the delete case: set the selection for later use
IF selection IS INITIAL.
   EXIT.
ENDIF.
g_selection = selection.

CLEAR comment.
CLEAR selection.

* 2) Get comment for the handle and for selected fields
INSERT <l_s_comment> INTO TABLE <l_to_comments>.
CALL FUNCTION 'Z_READ_COMMENTS'
   EXPORTING
   i_handle = handle
   i_infoprov = g_infoprov
   CHANGING
x_to_comments = <l_to_comments>.

CLEAR <l_s_comment>.
/* there is only one entry -
   but we cannot read via index as the table type is generic
loop at <l_to_comments> INTO <l_s_comment>.
endloop.
IF sy-subrc = 0.
/* transmit the comment to the html editor
   ASSIGN COMPONENT 'CMT' of STRUCTURE <l_s_comment> TO <l_comment_component>.
   comment = <l_comment_component>.
ELSE.
   CLEAR comment.
ENDIF.

6.8  BSP Page 'OnInputProcessing'
/* event handler for checking and processing user input and
/* for defining navigation

data: l_r_data TYPE REF TO cl_htmlb_textedit,
   l_r_htmlb_event TYPE REF TO cl_htmlb_event,
   lt_selected_fields TYPE STANDARD TABLE OF string,
   l_field_name TYPE string,
   l_field_value TYPE string,
   l_r_datadesc TYPE REF TO CL_ABAP_DATADESC,
   l_r_s_comment type ref to data,
   l_r_t_comments type ref to data,
   l_infocube type rsinfoprov.

FIELD-SYMBOLS: <l_s_comment> type any,
   <l_sel_field> TYPE string,
   <l_to_comments> type any table,
   <l_comment_component> TYPE ANY.

CONSTANTS: l_c_field_field_sep(1) TYPE c VALUE ' & ', " in URL: '%26',
   l_c_field_value_sep(1) TYPE c VALUE ' = '. " in URL: '%3D'.

/* get the comment first
check event_id = cl_htmlb_manager=>event_id.

/* make sure that the proper text in the text edit is set in any case
l_r_data ?= cl_htmlb_manager=>get_data( request = runtime->server->request
   name = 'textEdit'
   id = 'textedit' ).

/* now check that the correct button was pressed
l_r_htmlb_event = cl_htmlb_manager=>get_event( request ).
comment = l_r_data->text.
case l_r_htmlb_event->id.

  when 'ACCEPT'.
  * we need a work area of the right type.
    SPLIT g_selection AT l_c_field_field_sep INTO TABLE lt_selected_fields.

  * we know that the first entry in the selection is the infoproducer
    read table lt_selected_fields index 1 assigning <l_sel_field>. 
    SPLIT <l_sel_field> AT l_c_field_value_sep INTO l_field_name l_field_value.
    IF l_field_name <> zco_name_infoprov.
    * something deeply wrong...
      exit.
    endif.

    g_infoprov = l_field_value.

    CALL FUNCTION 'Z_RETURN_PARTPROV'
    EXPORTING
      I_INFOPROV = g_infoprov
    IMPORTING
      E_PARTPROV = l_infocube
    EXCEPTIONS
      NOT_UNIQUE = 1
      OTHERS = 2.
    IF SY-SUBRC <> 0.
      exit.
    ENDIF.
    g_infoprov = l_infocube.

  * now get the line type
    CALL FUNCTION 'Z_CREATE_DATAREF'
    EXPORTING
      I_INFOPROV = g_infoprov
      I_TYPE = zco_ty_comments_line
    IMPORTING
      E_O_DATADESC = l_r_datadesc.
    CREATE DATA l_r_s_comment TYPE HANDLE l_r_datadesc.
    assign l_r_s_comment->* to <l_s_comment>.

    CALL FUNCTION 'Z_CREATE_DATAREF'
    EXPORTING
      i_infoprov = g_infoprov
      i_type = zco_tt_comments_line
    IMPORTING
      e_o_datadesc = l_r_datadesc.
    CREATE DATA l_r_t_comments TYPE HANDLE l_r_datadesc.
    ASSIGN l_r_t_comments->* TO <l_to_comments>.
How To... Build a Fast and Flexible Comment Solution for BEx Web Applications

* now fill the structure with the selection for the comment
  LOOP AT lt_selected_fields ASSIGNING <l_sel_field> .
  SPLIT <l_sel_field> AT l_c_field_value_sep INTO l_field_name l_field_value.

  IF l_field_name = zco_name_infoprov.
    CONTINUE.
  ENDIF.

  ASSIGN COMPONENT l_field_name OF STRUCTURE <l_s_comment> TO <l_comment_component>.
  CHECK sy-subrc = 0 and <l_comment_component> IS ASSIGNED.
  <l_comment_component> = l_field_value.
  ENDDO.

  assign component 'LINE' of STRUCTURE <l_s_comment> TO <l_comment_component>.
  <l_comment_component> = l_r_data->text.

* we fill the table with the record
  insert <l_s_comment> into table <l_to_comments>.

  CALL FUNCTION 'Z_UPDATE_COMMENTS'
    EXPORTING
      I_TO_COMMENTS = <l_to_comments>
      i_handle = handle
      i_infoprov = g_infoprov.

* if we are in a pure reporting scenario we want to save directly -
* please uncomment the following coding
*  CALL FUNCTION 'Z_SAVE_COMMENTS'
*  EXPORTING
*    I_HANDLE = handle
*  EXCEPTIONS
*    FAILED = 1
*    OTHERS = 2.
*  IF SY-SUBRC <> 0.
*    MESSAGE ID SY-MSGID TYPE SY-MSGTY NUMBER SY-MSGNO
*      WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4.
*  ENDIF.

when 'DELETE'.
* we use a trick an call the page again from the java script
* after the confirm popup for the delete button. we delete
* the comment in the initialization event!!
How To... Build a Fast and Flexible Comment Solution for BEx Web Applications

* clear g_s_comment-cmt.
* clear comment.
* insert g_s_comment into TABLE l_to_comments.
* CALL FUNCTION 'Z_DELETE_COMMENTS'
  EXPORTING
  I_TO_COMMENTS = l_to_comments
  i_handle = handle.

when 'CANCEL'.
* do nothing - handled by java script

when others.
* also do nothing
endcase.

6.9 BSP Page 'OnDestroy'.
* this handler is called once the page reference is destroyed
  runtime->keep_context = 0.

6.10 IF_RS_BBS_BADI_HANDLER~CALL_URL

METHOD IF_RS_BBS_BADI_HANDLER~CALL_URL.

TYPE-POOLS: zco.

DATA: l_sx_mapping TYPE rsbbs_sx_mapping,
  l_s_range TYPE rrrrangesid,
  l_handle TYPE guid_32,
  l_fieldname TYPE fieldname,
  l_subrc TYPE sy-subrc,
  l_selection_value TYPE string,
  l_tabix TYPE sy-tabix,
  l_lines TYPE i,
  l_partprov TYPE rsinfoprov,
  l_partprov2 type rsinfoprov,
  l_lines2 type i.

data: l_value type rschavl,
  l_t_cob_pro_cmp TYPE rsd_t_cob_pro,
  l_s_cob_pro TYPE rsd_s_cob_pro.

CASE i_onam.
  WHEN 'COMMENT'.
  * Build the URL with handle and selection parameter
    *-------------------------------------------------------------------------
    *-------------------------------------------------------------------------
  *-------------------------------------------------------------------------
  r_url = 'http://ldciylq.wdf.sap.corp:50005/sap(bD1lbiZjPTAwMw==)/bc
  /bsp/sap/z_comments/z_edit_comments.htm'.
    *-------------------------------------------------------------------------
    *-------------------------------------------------------------------------

November 2015
* 1) get URL Prefix (server and port)

    CALL FUNCTION 'RSBB_URL_PREFIX_GET'
    EXPORTING
    i_handlerclass = 'CL_RSR_WWW_HTTP'
    * i_protocol = 'http'
    i_messageserver = '
    IMPORTING
    e_url_prefix = r_url
    e_subrc = l_subrc.
    IF l_subrc <> 0.
        RETURN.
    ENDIF.

* 2) add fixed part of URL and client

    CONCATENATE r_url '/sap/bc/bsp/sap/z_comments/z_edit_comments.htm' IN
    TO r_url.
    concatenate r_url '?sap-client=' sy-mandt into r_url.

* 3) get the handle and add it to URL

    CALL FUNCTION 'Z_RETURN_HANDLE'
    IMPORTING
    e_handle = l_handle.
    CONCATENATE r_url '&HANDLE=' l_handle INTO r_url.

* 4) add the infoprov. We use 0INFOPROV as name, see constant
    * - first convert the infoprov to the partprov
    * - we check if we receive the partprov from the selection

    READ TABLE i_thx_mapping INTO l_sx_mapping
    WITH KEY fieldnm = zco_name_infoprov.
    IF sy-subrc = 0.
        READ TABLE l_sx_mapping-range INTO l_s_range INDEX 1.
        l_partprov2 = l_s_range-low.
    ENDIF.

    * we might have an Infoprov (partprov in a multi Provider) that might be ag
    gregation levels

    CALL FUNCTION 'Z_RETURN_PARTPROV'
    EXPORTING
    i_infoprov = l_partprov2
    IMPORTING
    e_partprov = l_partprov
    EXCEPTIONS
    not_unique = 1
    OTHERS = 2.
    IF sy-subrc <> 0.
        *if we do not have a protpov we are lost...
        EXIT.
    ENDIF.
    ELSE.
* mabye we get the InfoProvider directly - in case we have a MultiProvider here
* we cannot determine the proper InfoProvider

    CALL FUNCTION 'Z_RETURN_PARTPROV'
    EXPORTING
        i_infoprov = i_infoprov
    IMPORTING
        e_partprov = l_partprov
    EXCEPTIONS
        not_unique = 1
        OTHERS = 2.
    IF sy-subrc <> 0.
*if we do not have a protprov we are lost...;-(
    EXIT.
    ENDF.
ENDIF.

    CLEAR: l_sx_mapping, l_s_range.

    CONCATENATE l_selection_value zco_name_infoprov '%3D' l_partprov '%26
    INTO l_selection_value.

* 5) add selection parameter containing field names and field values to URL
* -
get field names and values separated by '%26' and '%3D' into l_selection_v
alue

    l_lines = LINES( i_thx_mapping ).
    LOOP AT i_thx_mapping INTO l_sx_mapping.
    l_tabix = sy-tabix.

* - if we are currently working on the infoprovider we continue - it has been treated already above
    CHECK l_sx_mapping-fieldnm <> zco_name_infoprov.

* we cannot use hierachy nodes, select option, or intervals
    check l_sx_mapping-kind <> 'H' and
    l_sx_mapping-kind <> 'S' and
    l_sx_mapping-kind <> 'I' .

    clear l_lines2.
    l_lines2 = LINES( l_sx_mapping-range ).
    check l_lines2 = 1.

    READ TABLE l_sx_mapping-range INTO l_s_range INDEX 1.

    check l_s_range-sign = 'I'.
    check l_s_range-opt = 'EQ'.

* - translate the iobj_name into fieldname
    CALL FUNCTION 'Z_COMNTS_FIELDS_CONVERT'
    EXPORTING
        i_iobjnm = l_sx_mapping-fieldnm
IMPORTING
e_fieldname = l_fieldname.

l_value = l_s_range-low.

if l_fieldname <> 'KYFNM'.
  * do we use the hash logic?
    if l_value = ' ' or l_value = '#'.
      if zco_use_hash is initial.
        l_value = ' '.
      else.
        l_value = '#'.
      endif.
    endif.
  endif.
endif.
endif.

CONCATENATE l_selection_value l_fieldname '%3D' l_value INTO l_selection_value.

* CONCATENATE l_selection_value l_fieldname '%3D' l_s_range-
low INTO l_selection_value.
  IF l_tabix < l_lines. " do not add separator %26 for last field
  CONCATENATE l_selection_value '%26' INTO l_selection_value.
  ENDIF.
ENDLOOP.

* - add selection value to URL
  CONCATENATE r_url '& ' 'SELECTION' '=' l_selection_value INTO r_url.

WHEN OTHERS.
ENDCASE.
ENDMETHOD.

6.11 IF_RS_BBS_BADI_HANDLER~GET_TARGETS

method IF_RS_BBS_BADI_HANDLER~GET_TARGETS.

type-pools icon.
data:   ls_f4_list   type rsbbs_s_f4_list.

* These entries appear as value help
  clear ls_f4_list.
  ls_f4_list-icon  = icon_complete.

append ls_f4_list to e_t_f4_list.

ls_f4_list-objnm = 'COMMENT'.
ls_f4_list-txtlg = 'Modify Comment'.
append ls_f4_list to e_t_f4_list.
endmethod.
6.12 XHTML for Web Application

```xml
  <head>
    <title>BEx Web Application</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  </head>
  <body>
    <bi:QUERY_VIEW_DATA_PROVIDER name="DP_1">
      <bi:INITIAL_STATE type="CHOICE" value="QUERY"/>
      <bi:QUERY value="GS_COMMENT_Q1" text="gs_comment_q1"/>
      <bi:INITIAL_STATE/>
    </bi:QUERY_VIEW_DATA_PROVIDER>
    <bi:TEMPLATE_PARAMETERS name="TEMPLATE_PARAMETERS"/>
    <p>
      <bi:SCRIPT_ITEM name="SCRIPT_ITEM_1" designwidth="300" designheight="70">
        <bi:SCRIPT value="biLargeData:3MB816716DECES4KHE0CJCGDY"/>
      </bi:SCRIPT_ITEM>
    </p>
    <p>
      <span style="DISPLAY: none">
        <bi:BUTTON_GROUP_ITEM name="BUTTON_GROUP_ITEM_1" designheight="23" designwidth="300">
          <bi:BUTTON_LIST type="ORDEREDLIST">
            <bi:BUTTON type="COMPOSITE" index="1" >
              <bi:CAPTION value="Refresh Comments"/>
              <bi:ENABLED value="X"/>
              <bi:ACTION type="CHOICE" value="SCRIPT_FUNCTION">
                <bi:SCRIPT_FUNCTION value="executeJS_DELAY_REFRESH"/>
              </bi:ACTION>
            </bi:BUTTON>
            <bi:BUTTON type="COMPOSITE" index="2" >
              <bi:CAPTION value="DISABLE_PAGE"/>
              <bi:ENABLED value="X"/>
              <bi:ACTION type="CHOICE" value="SCRIPT_FUNCTION">
                <bi:SCRIPT_FUNCTION value="show_popup"/>
              </bi:ACTION>
            </bi:BUTTON>
            <bi:BUTTON type="COMPOSITE" index="3" >
              <bi:CAPTION value="ENABLE_PAGE"/>
              <bi:ACTION type="CHOICE" value="SCRIPT_FUNCTION">
                <bi:SCRIPT_FUNCTION value="hide_popup"/>
              </bi:ACTION>
            </bi:BUTTON>
          </bi:BUTTON_LIST>
        </bi:BUTTON_GROUP_ITEM>
      </span>
    </p>
    <p>
      <bi:BUTTON_GROUP_ITEM name="BUTTON_GROUP_ITEM_2" designheight="23" designwidth="300">
        <bi:BUTTON_LIST type="ORDEREDLIST">
          <bi:BUTTON type="COMPOSITE" index="2" >
            <bi:CAPTION value="SAVE ALL"/>
            <bi:ACTION type="CHOICE" value="INSTRUCTION">
              <bi:INSTRUCTION>
                <bi:EXEC_PLANNING_FUNCTION_SIMPLE>
                  <bi:SELECTOR_DATA_PROVIDER_REF value="DP_1"/>
                  <bi:PLANNING_FUNCTION value="Z_DUMMY_SAVE" text="z_dummy_save (do not touch!)">
                    <bi:PROCESS_CHANGED_DATA type="CHOICE" value="X"/>
                  </bi:PLANNING_FUNCTION>
                </bi:EXEC_PLANNING_FUNCTION_SIMPLE>
              </bi:INSTRUCTION>
              <bi:SAVE_DATA/>
            </bi:ACTION>
          </bi:BUTTON>
        </bi:BUTTON_LIST>
      </bi:BUTTON_GROUP_ITEM>
    </p>
  </body>
</html>
```
How To... Build a Fast and Flexible Comment Solution for BEx Web Applications

November 2015
<bi:MOD_PARAMETER type="COMPOSITE" index="6">
  <bi:MOD_PARAM_NAME value="TOOLTIP" />
  <bi:MOD_PARAM_VALUE type="CHOICE" value="BOOLEAN" />
</bi:MOD_PARAMETER>

<bi:MOD_PARAMETER type="COMPOSITE" index="7">
  <bi:MOD_PARAM_NAME value="WRAPPING" />
  <bi:MOD_PARAM_VALUE type="CHOICE" value="BOOLEAN" />
</bi:MOD_PARAMETER>

<bi:MOD_PARAMETER type="COMPOSITE" index="8">
  <bi:MOD_PARAM_NAME value="MAX_ROW_NUMBER" />
  <bi:MOD_PARAM_VALUE type="CHOICE" value="INTEGER" />
    <bi:INTEGER value="4" />
  </bi:MOD_PARAMETER>
</bi:MOD_PARAMETER>

<bi:ACTIVE value="X" />

<bi:MOD_SELECT type="CHOICE" value="MOD_GENERIC_MODULE">
  <bi:MOD_GENERIC_MODULE type="COMPOSITE">
    <bi:MOD_REFERENCE value="com.sap.ip.bi.rig.Command" />
    <bi:MOD_PARAMETER_LIST type="ORDEREDLIST">
      <bi:MOD_PARAMETER type="COMPOSITE" index="1">
        <bi:MOD_PARAM_NAME value="ACTION_ON_CLICK" />
        <bi:MOD_PARAM_VALUE type="CHOICE" value="ACTION">
          <bi:ACTION type="CHOICE" value="INSTRUCTION">
            <bi:INSTRUCTION>
              <bi:RRI>
                <bi:TARGET_DATA_PROVIDER_REF value="DP_1" />
                <bi:RRI_RECEIVER value="QURY0002" />
              </bi:RRI>
            </bi:INSTRUCTION>
          </bi:ACTION>
        </bi:MOD_PARAMETER_VALUE>
      </bi:MOD_PARAMETER>
      <bi:MOD_PARAMETER type="COMPOSITE" index="2">
        <bi:MOD_PARAM_NAME value="ACTION_IN_CONTENT_VISIBLE" />
        <bi:MOD_PARAM_VALUE type="CHOICE" value="BOOLEAN" />
      </bi:MOD_PARAMETER>
      <bi:MOD_PARAMETER type="COMPOSITE" index="3">
        <bi:MOD_PARAM_NAME value="COLUMN_5" />
        <bi:MOD_PARAM_VALUE type="CHOICE" value="BOOLEAN" />
      </bi:MOD_PARAMETER>
      <bi:MOD_PARAMETER type="COMPOSITE" index="4">
        <bi:MOD_PARAM_NAME value="ONLY_DATA_ENTRY_ENABLED" />
        <bi:MOD_PARAM_VALUE type="CHOICE" value="BOOLEAN" />
      </bi:MOD_PARAMETER>
    </bi:MOD_PARAMETER_LIST>
  </bi:MOD_GENERIC_MODULE>
</bi:MOD_SELECT>
</bi:MOD_SINGLE_MODULE>
6.13 Java Script for Web Application

```javascript
sapbi_page.registerOnBodyOnLoadEvent( 'ONLOAD', 'afterBodyOnLoad (b_isInitialLoad)');

function afterBodyOnLoad (b_isInitialLoad)
{
    hide_popup();
}

function show_popup()
{
    debugger;
    var popup = document.getElementById("popup");
    if (popup == null) {
        popup = document.createElement("div");
        popup.id = "popup";
        popup.style.zindex = "2";
        popup.style.border = "0px";
        popup.style.position = "absolute";
        popup.style.background = "#ffffff";
        popup.style.top = "0px";
        popup.style.left = "0px";
        popup.style.filter = "alpha(opacity = 75)";
        popup.style.opacity = ".75";
        document.getElementsByTagName("body")[0].appendChild(popup);
    }
    popup.style.display = "block";
    popup.style.height = document.body.clientHeight;
    popup.style.width = document.body.clientWidth;
    ```
function hide_popup()
{
    var popup = document.getElementById("popup");
    if (popup != null) {
        popup.style.display = "none";
    }
}

function executeJS_DELAY_REFRESH( currentState, defaultCommandSequence ){
    setTimeout("executeJS_TRANSFER_STATE_R()", 500);
}

function executeJS_SET_ITEM_PARAMETERS_EAA( currentState, defaultCommandSequence ){
    var commandSequence = new sapbi_CommandSequence();
    var commandSET_ITEM_PARAMETERS_1 = new sapbi_Command( "SET_ITEM_PARAMETERS" );
    var paramITEM_TYPE = new sapbi_Parameter( "ITEM_TYPE", "ANALYSIS_ITEM" );
    commandSET_ITEM_PARAMETERS_1.addParameter( paramITEM_TYPE );
    var paramINIT_PARAMETERS = new sapbi_Parameter( "INIT_PARAMETERS" );
    var paramListINIT_PARAMETERS = new sapbi_ParameterList();
    commandSET_ITEM_PARAMETERS_1.addParameter( paramINIT_PARAMETERS );
    paramListINIT_PARAMETERS.addParameter( paramVISIBILITY );
    paramINIT_PARAMETERS.setChildList( paramListINIT_PARAMETERS );
    var paramTARGET_ITEM_REF = new sapbi_Parameter( "TARGET_ITEM_REF", "ANALYSIS_ITEM_1" );
    commandSET_ITEM_PARAMETERS_1.addParameter( paramTARGET_ITEM_REF );
    commandSequence.addCommand( commandSET_ITEM_PARAMETERS_1 );

    var commandSET_ITEM_PARAMETERS_2 = new sapbi_Command( "SET_ITEM_PARAMETERS" );
    var paramITEM_TYPE = new sapbi_Parameter( "ITEM_TYPE", "ANALYSIS_ITEM" );
    commandSET_ITEM_PARAMETERS_2.addParameter( paramITEM_TYPE );
}
var paramINIT_PARAMETERS = new sapbi_Parameter( "INIT_PARAMETERS" );
var paramListINIT_PARAMETERS = new sapbi_ParameterList();
commandSET_ITEM_PARAMETERS_2.addParameter( paramINIT_PARAMETERS );

var paramVISIBILITY = new sapbi_Parameter( "VISIBILITY", "VISIBLE" );
paramListINIT_PARAMETERS.addParameter( paramVISIBILITY );
paramINIT_PARAMETERS.setChildList( paramListINIT_PARAMETERS );
var paramTARGET_ITEM_REF = new sapbi_Parameter( "TARGET_ITEM_REF", "ANALYSIS_ITEM_1" );
commandSET_ITEM_PARAMETERS_2.addParameter( paramTARGET_ITEM_REF );

commandSequence.addCommand( commandSET_ITEM_PARAMETERS_2 );
return sapbi_page.sendCommand( commandSequence );

/**
 * Javascript functions that are to be integrated into a web template and
 * that
 * are to be used later in the web application ALWAYS have to have the same
 * function
 * signature, i.e. the parameters that will be passed.
 * *
 * @param currentState - a list of parameters that describe the state the
 * web item
 * @param defaultCommandSequence - the initially used sequence of commands
 * that
 * would have been used instead of the custom
 * script.
 * **/
function executeJS_TRANSFER_STATE_R( currentState, defaultCommandSequence ){
    //Note: information can be extracted using the parameter 'currentState'
    // and 'defaultCommandSequence'. In either case create your own object
    // of type 'sapbi_CommandSequence' that will be sent to the server.
    // To extract specific values of parameters refer to the following
    // snippet:
    // var key = currentState.getParameter( PARAM_KEY ).getValue();
    // alert( "Selected key: " + key );
    //
    // ('PARAM_KEY' refers to any parameter's name)
    //Create a new object of type sapbi_CommandSequence
var commandSequence = new sapbi_CommandSequence();

/*
 * Create a new object of type sapbi_Command with the command named
"TRANSFER_STATE"
*/
var commandTRANSFER_STATE_1 = new sapbi_Command( "TRANSFER_STATE" );
/* Create parameter ALL_DATA_PROVIDERS */
var paramALL_DATA_PROVIDERS = new sapbi_Parameter( "ALL_DATA_PROVIDERS", "x" );

commandTRANSFER_STATE_1.addParameter( paramALL_DATA_PROVIDERS );

/* End parameter ALL_DATA_PROVIDERS */

// Add the command to the command sequence
commandSequence.addCommand( commandTRANSFER_STATE_1 );
/*
 * End command commandTRANSFER_STATE_1
 */

//Send the command sequence to the server
return sapbi_page.sendCommand( commandSequence );