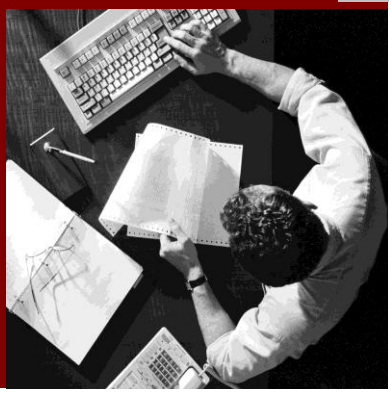

Business Process Monitoring

Application Monitoring - Customer Exit with ST-A/PI 01M

with Solution Manager Release ST 400

August 2010

THE BEST-RUN E-BUSINESSES RUN SAP

SAP AG
Neurottstraße 16
69190 Walldorf
Germany
T +49/18 05/34 34 24
F +49/18 05/34 34 20
www.sap.com

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 expressively 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.

Documentation in the SAP Service Marketplace

You can find this documentation at the following address:
http://service.sap.com//bpm

TABLE OF CONTENTS

TABLE OF FIGURES

1. Introduction

In general Business Process Monitoring with SAP Solution Manager allows monitoring of SAP Solutions for performance, errors, throughput and backlog issues. A huge number of monitors are already available.

Since sometimes there are requirements that cannot be fulfilled by the usage of “standard” monitors provided with the ST-A/PI Plug-In, SAP has introduced a customer exit for Application Monitoring. With this “customer exit” you can define your own monitoring objects; develop your own data collector(s) and display the alerts in the SAP Solution Manager together with the alerts from the standard monitoring. It is most applicable if customers want to monitor own developed applications in a detailed and customized way. It is an extension of the functionality of Business Process Monitoring with SAP Solution Manager. As this customer exit was created to provide the maximal flexibility, nearly everything can be a monitoring object, i.e. number of entries of a specific table, status of IDocs in an R/3 system, etc.

This guide explains how to implement the customer exit in a Business Process Monitoring Solution.

At first you have to define your monitoring object. This has to be done by the so called “Customizing report” /SSA/EXM. This report is part of the ST-A/PI software component and has to be used in the managed system. With this report you have to define the customer specific monitoring object, its keyfigures and the parameters which can be used for selecting the data (described in chapter 4.2 2).

The framework of the user-exit is the program /SSA/ECU which is provided with the ST-A/PI software component. Within this framework coding there are some parts of coding which is inactive (commented coding). This coding is intended to be used as a template (or example) for the development of your own coding in the customer exit. This will be handled in chapter 4.3.

Hard coded in this framework the report Z_BPM_ECU_COLLECTOR is called. This is the report where you have to develop the coding for the user exit (by following strictly some conventions which will be described in this document).

2. Prerequisites

2.1. Organizational Prerequisites

- Participation in classroom training Business Process Management (SM300/SMO610 or E2E300) and/or familiarity with the Business Process Monitoring Set-Up Guide found in the Media Library on the SAP Service Marketplace under Quick Link /bpm (http://service.sap.com/bpm).
- Participation in classroom training ABAP Workbench (BC400) or basic experience in ABAP programming

2.2. Technical Prerequisites

- Installation of ST-SER 700_2008_2, ST400 in Solution Manager system,
- Installation of the software components ST-PI and ST-A/PI (at least ST-A/PI 01L in satellite system as well as SAP Solution Manager system).

You can check the currently available version of add-ons ST, ST-SER and ST-A/PI in SAP Service Marketplace, quicklink /SWDC (http://service.sap.com/swdc). ST-PI and ST-A/PI are available in quicklink /supporttools

3. Roadmap

The development of a customer specific monitoring object with the BPMon user exit framework consists of the following steps (all steps will be explained in detail in this document):

1. **Define Monitoring Object with the customizing report /SSA/EXM in the managed system (development system)**
2. **Create the report Z_BPM_ECU_COLLECTOR in the managed system**
3. **Extend Z_BPM_ECU_COLLECTOR at least with the following form routines: CUCUSTXX and KEYFIG_SUBROUTINECUCUSTXX_YY**
4. **Start an update of the application monitoring repository in the solution manager system to make the new monitoring object available**
5. **Transport the report Z_BPM_ECU_COLLECTOR from the development system to the productive system**
6. **Define your monitoring object and its customizing with the report in the productive system (currently there is no transport mechanism for the customizing therefore this step has to be done in every system separately)**

4. Example

The following steps are necessary if you want to define your own customer exit.

First of all you should gather the requirements for the proposed solution (see chapter 4.1). Next step is the Definition of the Monitoring Object and its keyfigures, and maintenance of the parameters for your Application Monitor. The Customizing of the Monitoring Object will be done via a special Report /ssa/exm, which helps you to define the monitoring objects with their keyfigures and all necessary parameters. You will find this described in chapter **4.2 Define Application Monitor**.
The Subchapter 4.3 contains the explanation on how to **create the coding for the customized Application Monitor**. In this step you will create or enhance the report "Z_BPM_ECU_COLLECTOR" in which you can implement your special requirements for the customer exit. For this step you can use and adapt an existing template. The last step is the **Setup of the Business Process Monitoring**. You will find the description of this in the subchapter 4.4 .

4.1. Preparation

Customer Requirement: The customer wants to be alerted if there are users on particular systems which were locked automatically by the system due to incorrect logons or by the administrator.

It should be possible to use the usertype (Dialog, System and so on) as selection parameter for the collector. One additional selection parameter should be the locking reason.

It should be possible to create yellow or red alerts if the measured value exceeds the threshold value maintained during setup of BPMon. Meaningful for this type of keyfigure is a 2-step rating.

It should be possible to get a detailed list of all users who are part of the result list.

We will name the monitoring object "User account".

The name of the keyfigure will be "Number of Locked User (User exit)".

The keyfigure will be defined with the following parameters:

- User Type
- Locking Reason

The data in which we are interested are stored in the table USR02. Here is the field USTYP which identifies the user type. Field UFLAG contains the lock status. The entry in this field is '128' if the user is locked due to too many failed logon, it is 64 if it the user was locked by the administrator.

4.2. Definition of Monitoring Object (Customizing)

The definition of the monitoring object and the customizing is done with the report /SSA/EXM. This report is delivered with the software component ST-A/PI and it has to be executed in the managed system where the monitoring should be done. Run the report "**/SSA/EXM"** via transaction SE38. There are 2 parameters in the start screen of this report which are prefilled as shown in the following screenshot.

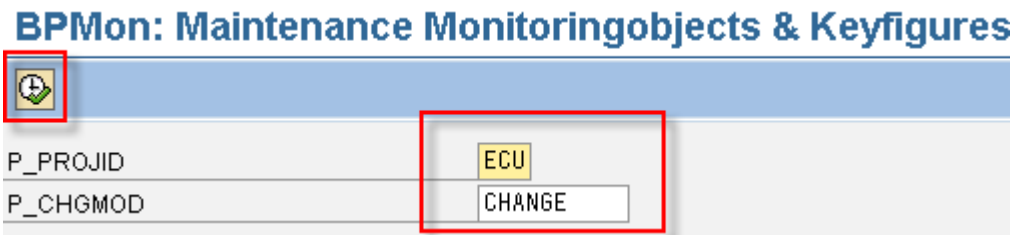

Figure 1 Entry screen Customizing Report with default parameters

After pressing the “Execute” button the main screen appears. Starting from here all settings for defining of Monitoring Objects have to be made. When you start the report the first time you will see a list of monitoring objects named CUST00 to CUST11. These list entries are delivered with the report but do not contain any functionality. They should only show as an example how the naming of the monitoring object has to be done. You see here in the first column the technical name of the monitoring object (which is later evaluated in the user-exit framework). The naming has to be done in this way: CUST plus a 2-digit number, where you are free with choosing the number. The second column is showing the description of the monitoring object. This is the name of the monitoring object as it is shown later in the setup session of BPMon in the solution manager. If you want you can delete all the example entries in this list and leave only those which will be used (otherwise you will get an error message because there is no text maintained in the field description for the other monitoring objects).

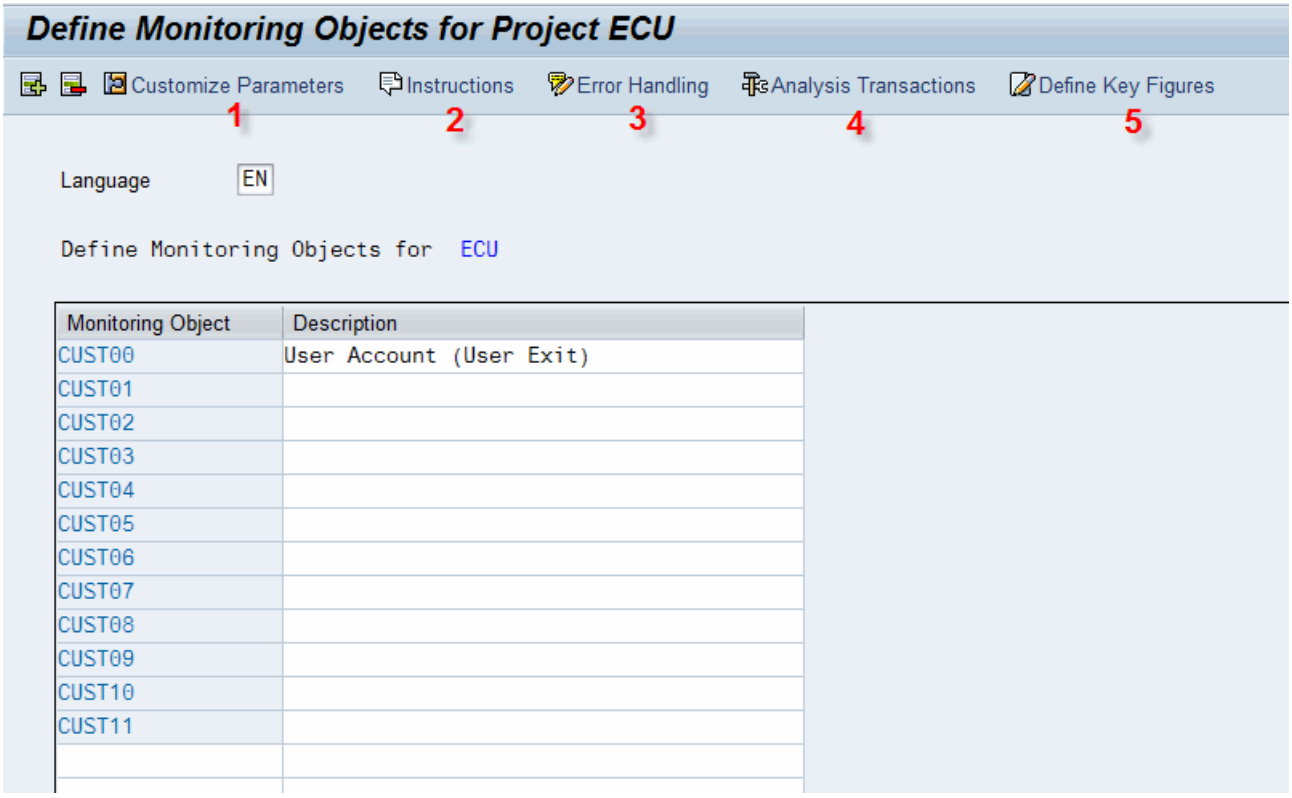

Figure 2 Define Monitoring Object “User Account (User Exit)”

As shown in Figure 2 the technical name of the monitoring object in our example is CUST00, the description is ‘User Account (User Exit)’. Adding the suffix ‘User Exit’ makes it easier during setup of BPMon in the solution manager to separate standard monitoring objects from customer developed monitoring objects.

Save the monitoring object (CTRL+S).

4.2.1. Customize Parameters of the Monitoring Object

There is only one setting which has to be done for each monitoring object. Click on the function button 'Customize Parameter' (shown in figure 2, marked with the number 1).

The following screenshot (figure 3) is showing the customizing screen you will reach now. The only thing which has to be maintained here is at least 1 active flag in the area 'Possible Assignment' (see upper red shaped box in the following screenshot figure 3). Here you have to define if it will be possible to assign this monitoring object to a business process step, an interface or a logical component.

Under the tab 'Applications' there is a table where you can enter product and application to which this monitoring object will be assigned to. This can be used in the setup session of BPMon as a filter. If you want to use this filter there you will be able to choose an application and only monitoring objects which are assigned to this application will be shown.

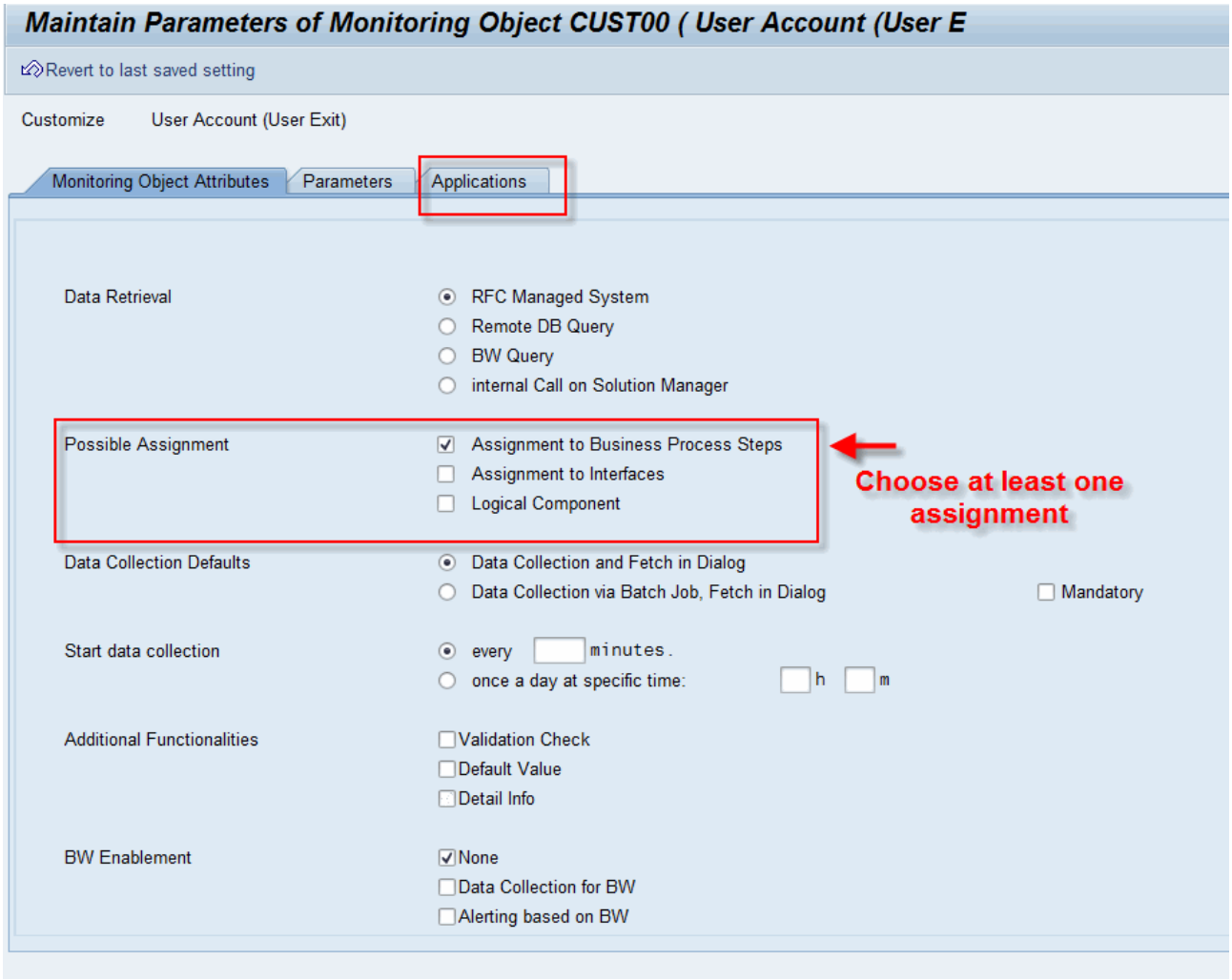

Figure 3 Monitoring Object "User Account (User Exit)"

For our example we need the possibility to assign the monitoring object to a business process step (first flag). Please activate this flag and save. We will also use the functionality of providing a detailed list for an alert. Therefore activate the flag on 'Detail Info' in the area 'Additional Functionalities

4.2.2. Define Instruction Text on Monitoring Object Level

Via button "Instructions" (Figure 3, marked with number 2) you can enter an instruction.
The instruction text should explain the purpose of the Monitoring Object. Insert some explanation on: How does the Monitoring Object work in the satellite system? Describe the available Key Figures: What do they monitor? Add important hints, if there are some, e.g. when this Monitoring Object should not be used.

4.2.3. Define Error Handling Text on Monitoring Object Level

Via button ‘Error Handling’ (Figure 3, marked with number 3) you can enter an instruction what should happen when an alert for this monitoring object is created.

4.2.4. Definition of Analysis Transaction

Via button ‘Analysis Transaction’ (Figure 3, marked with number 4) you can maintain a transaction which will be called in the satellite system from the monitoring session in the solution manager. This is usually a transaction which is related to the monitoring object and can be used to make further analysis or which allow a reprocessing of this monitoring object.

4.2.5. Define Key Figures of the Monitoring Object

One monitoring object may consist of one or more keyfigures. The measurement and alerting is done on keyfigure level (and then is propagated to the higher levels of a monitoring session). To define the keyfigure use the function button ‘Define Keyfigures’ (shown in figure 2, number 5). When you click this button you will see the following screen:

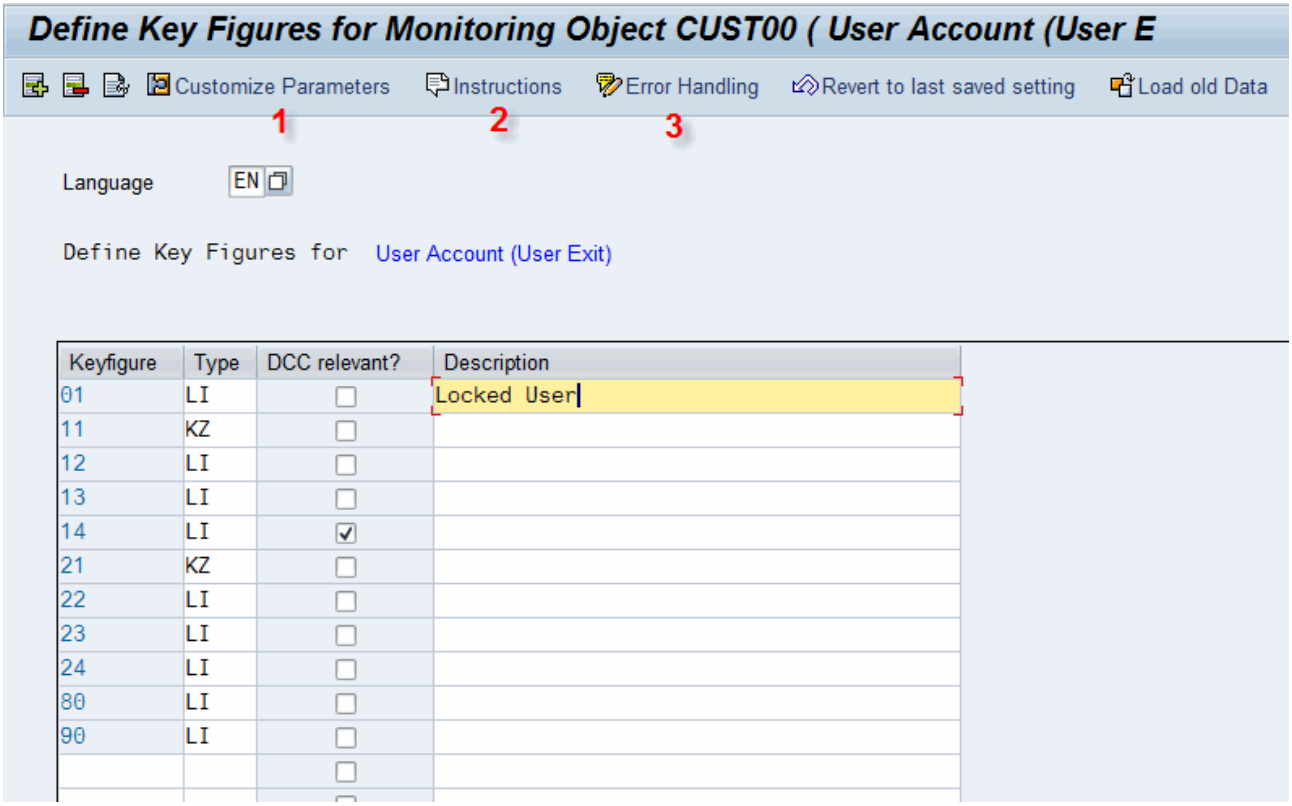

Figure 4 Define Key Figure of Monitoring Object

What you see here when you enter this screen the first time is a list of keyfigure which do not contain any business logic. This is just an example for showing that the keyfigures have to be numbered (column key figure). You can delete these entries and leave those you want to use for your purpose or create completely new ones.

In the column ‘Type’ you define the type of the keyfigure. Type LI allows the usage of more than 1 counter for this keyfigure, type ‘KZ’ is restricted to only 1 counter. There are other types available but nearly all SAP standard monitoring objects are based on type LI therefore a detailed description can be skipped in this document. For the example described in this document we need the type LI. Then enter the description of the key figure, in this example it is ‘Locked User’ and save.

The flag ‘DCC relevant?’ is related to data consistency check tools and has no impact for the functionality of a monitoring object in the area of business process monitoring.

Save the settings you have made here.

4.2.5.1. Define Attributes and Parameters of a Key Figure

To define the parameters on keyfigure level click the function button ‘Customize Parameter’ (see figure 4, number 1). Here you define some attributes of this keyfigure and which selection parameters can be used later during the setup of the BPMon session. These are the parameters you can use in the user-exit coding to select your data. You will get a screen as shown in the following screenshot (in this case the data we need for this example are already filled).

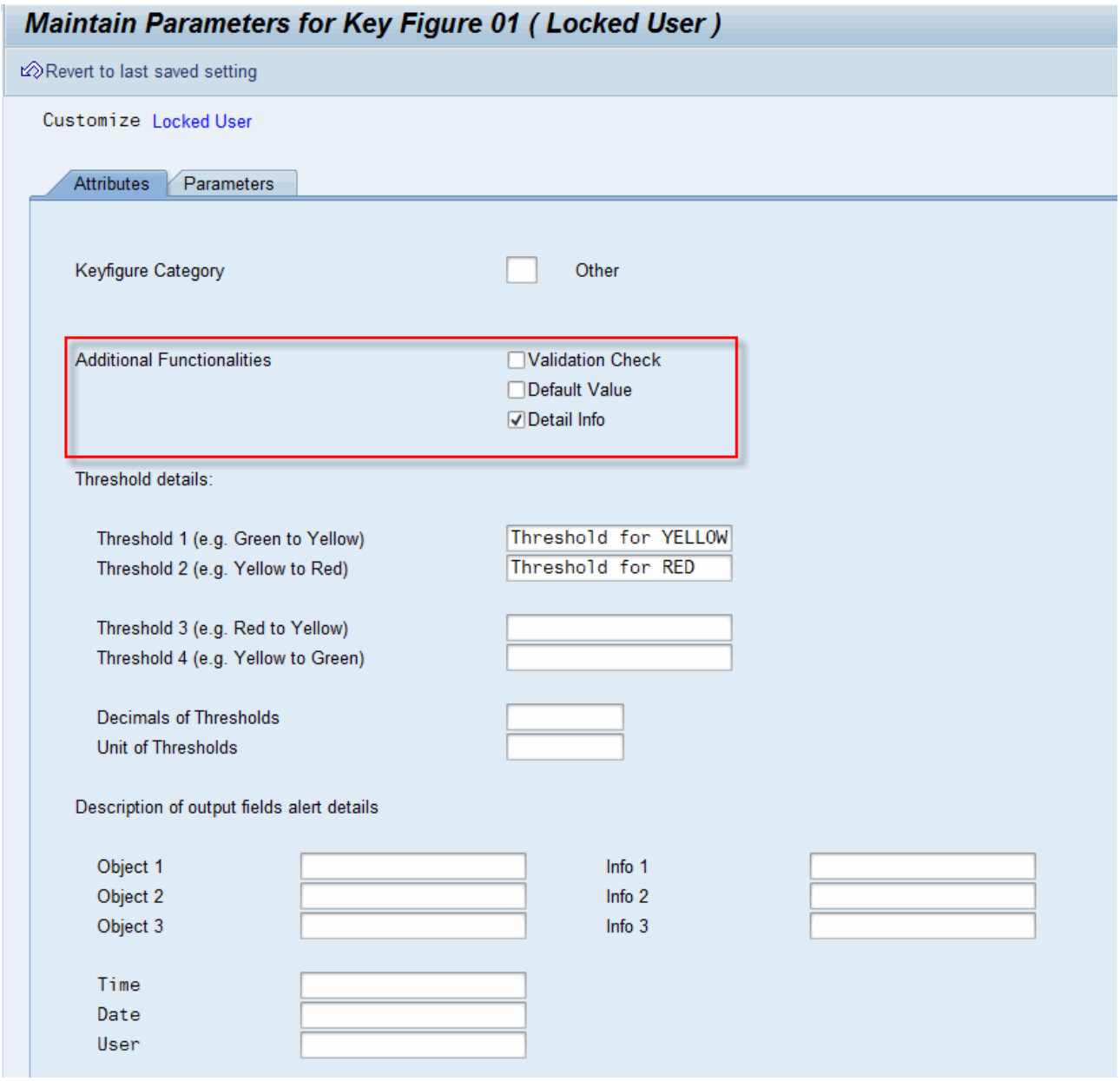

Figure 5 Define Customizing of Key Figure (Attributes)

In the area ‘Additional functionalities’ you can steer if additional function buttons in the BPMon session in the solution manager are shown or not. Despite that, the functionality behind these buttons has to be programmed in the user exit.

- Validation check: when active, an additional function button will be shown in solution manager during setup of the keyfigure. The button will be named ‘Validation check’. It can be used to process a validation of the entered data for the parameters.
- Default Value: a function button with the name ‘Default Value’ will be shown during setup of the keyfigure. It can be used to fill some of the selection parameter with default values.
- Detail Info: A button with the name ‘Detail Info’ will be shown in the BPMon monitoring session on keyfigure level. It can be used to show a detail list of objects when marking an alert.

Very important is to enter a name for the Threshold 1 and 2 as you see it in the figure 5 (the naming here is ‘Threshold for YELLOW’ and ‘Threshold for RED’). If you miss this then there will be no possibility during BPMon setup to enter the alert threshold values.

In case of a 2-step rating you enter only the naming for the thresholds 1 and 2; in case of a 4-step rating you also enter the threshold names for threshold 3 and 4.

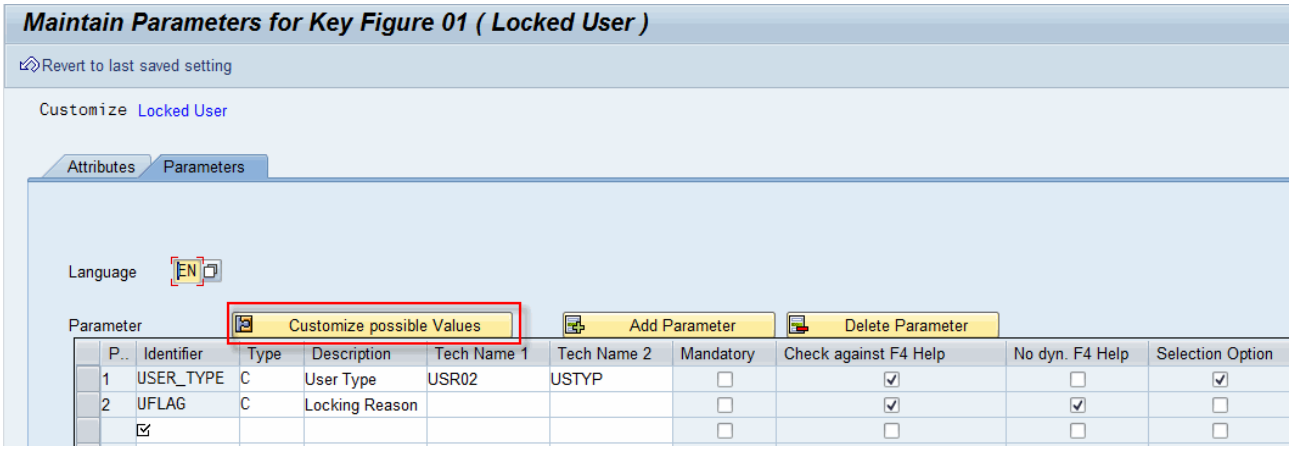

Figure 6 Define Customizing of Key Figure (Parameters)

If you want to provide a self defined F4-Help based on a few fix values for a parameter then you can use the functionality which is given with the button ‘Customize Possible values’ (see figure 6). Mark the line with the parameter you want to maintain and click the button ‘Customize Possible Values’. You will see a table where you can enter the value and a description for this value you want to provide with your F4-Help. It is important that you activate the flag in column ‘No dyn. F4-Help’ **and** the flag ‘Check against F4-Help for this parameter.

The meaning of the columns of the table shown in Figure 6 **Error! Reference source not found.**are:

- Column ‘Parameter’: this column is filled with an ongoing numbering automatically by the system after saving.
- Column ‘Identifier’: this is the technical name of the parameter which can be used later in the user-exit coding to access the content of the selection parameter. You define the name by yourself.
- Column ‘Type’: data type of the parameter (can be chosen by an F4-Help)
- Column ‘Description: free text for describing the parameter
- Columns ‘Tech Name 1’, ‘Tech Name 2’: these columns are important if you want to provide a F4-Help for a parameter which based on the existing F4-Helps in the data dictionary. Then you have to enter in ‘Tech Name 1’ the name of the table and in ‘Tech Name 2’ the name of the table field from where you want to use the F4-Help.
- Column ‘Mandatory’: can be used when you want to make a parameter mandatory during the BPMon setup.
- Column ‘Check against F4-Help’: has to be active if you want to provide a F4-Help
- Column ‘No dyn. F4-Help’. Has to be activated if you want to provide a F4-Help which is based on fixed values defined by your own (maintained under function button ‘Customize Possible Values)

- Column ‘Selection Option’: If this flag is active, then the parameter is shown during BPMon setup as a range parameter (otherwise there would only the possibility to maintain a single value)

You can scroll in this table to the right and you will see some more customizing flags but there is nothing of importance to be maintained to make this user exit working.

4.2.5.2. Define Instructions on Key Figure Level

Via button ’Instructions’ (figure 4, marked with number 2) you can enter an instruction which is keyfigure specific.

The instruction text should explain the purpose of the keyfigure. Insert some explanation on: What does the key figure measure, describe the available selection parameter and you can add important hints e.g. regarding the performance of this collector (like: recommended to run this keyfigure in background to avoid time out).

The instruction text will be shown in the business process monitoring session during the setup of the customer specific monitoring object.

4.2.5.3. Define Error Handling on Key Figure Level

Via button ‘Error Handling’ (figure 4, marked with number 3) you can enter an instruction what should happen when an alert for this keyfigure is created.

4.3. Development User-Exit

The data collectors are subroutines written in a Z-Report called Z_BPM_ECU_COLLECTOR in the satellite system. Check in the satellite system where the user exit should be available if the report Z_BPM_ECU_COLLECTOR is already there. If not, create this report as type ‘Executable program’.

This report now has to be extended with several subroutines. These are the subroutines we need at least:

- Form DC_CUCUSTXX: for every monitoring object we need a subroutine named like DC_CUCUSTXX, where XX has to be replaced by the number of the monitoring object. The number of the monitoring object was defined during the setup of the monitoring object with the customizing report /SSA/EXM. In our example the number is 00 (see figure 2). So the name of the form in this example is DC_CUCUST00.
- Form KEYFIG_SUBROUTINECUCUSTXX_YY: for every keyfigure of the monitoring object we need a form routine which is named like this, where CUSTXX has to be replaced by the name of the monitoring object to which the keyfigure belongs (in this example CUST00) and YY has to be replaced by the number of the keyfigure as it is shown in the customizing report (see figure 4), in this example 01. So the routine for the keyfigure in our example is KEYFIG_SUBROUTINECUCUST00_01.

Depending on the additional functionalities you want to realize (own value help, validation check, using of default values and detail Info lists) you have to add the following forms to the Z_BPM_ECU_COLLECTOR (these form routines are optional):

- Form GET_DETAIL_INFORMATION: in this form you can implement coding which will be executed when clicking on the ‘Detail Info’ button in the monitoring session.
- Form GET_VALUE_HELP
- Form VALIDATION_CHECK
- Form GET_DEFAULT_VALUES

For all these forms a template is provided with the customizing report /SSA/EXM. You get the template by the following way:

1. Start report /SSA/EXM
2. Execute it with the prefilled values ECU and CHANGE and you will get a screen as shown in the following screenshot

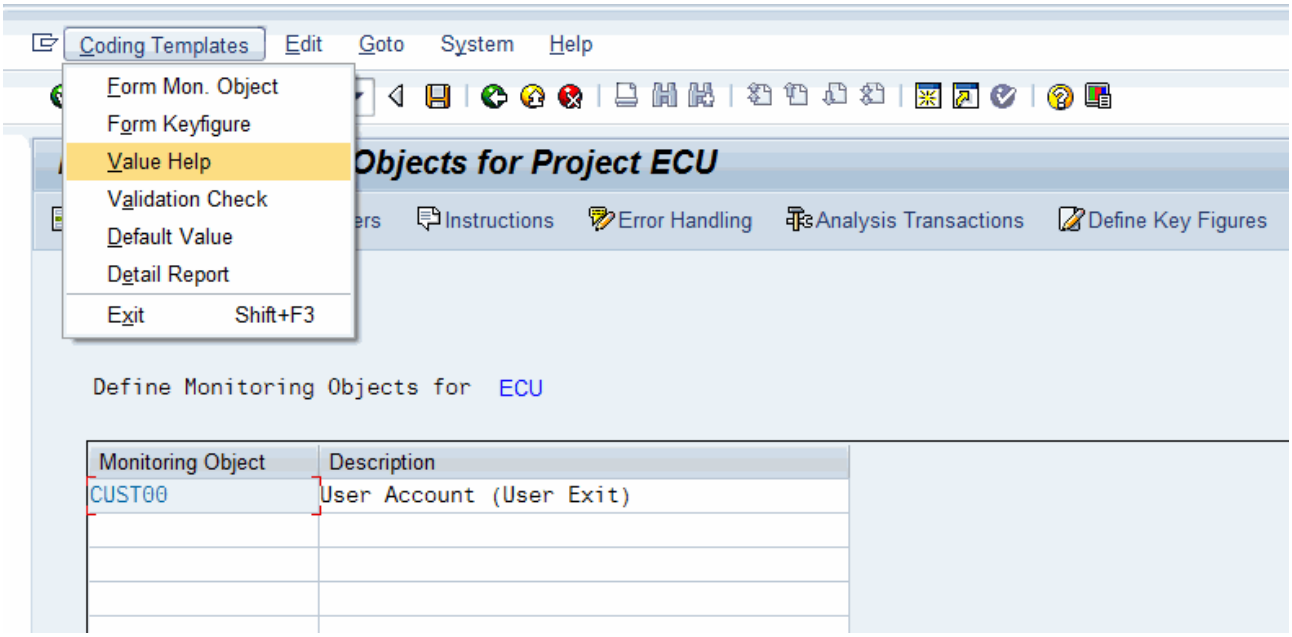

Figure 7 Coding Templates

Here you have to click in the menu bar on the button 'Coding Templates and you will get a list of form routines which are mentioned above, each form routine with a little example coding.

4.3.1. Testing during Development

During the development of this user exit it will be helpful to have the possibility of testing and debugging the new coding. This cannot be done by starting the report Z_BPM_ECU_COLLECTOR from se38 because this report is only the framework for the other form routines and these form routines are called at the end from the solution manager. But there is a tool available for this purpose which is provided with the ST-A/PI component. Call transaction ST13 and choose via the F4-Help on the field 'Toolname' the tool 'PROJBROWSER'. This is the so called 'projectbrowser'. Execute and you will get a screen as shown in the next screenshot, figure 8.

There is one prerequisite before this tool can be used: you have to setup the BPMon customizing in the solution manager for your new monitoring object. This is possible as soon as the customizing as described in chapter 3.2 is done and when the report Z_BPM_ECU_COLLECTOR is created (it is not necessary that this report contains any coding).

So you have to make the setup in a business process monitoring session for your new monitoring object, and generate and activate the monitoring session. Then the values for the selection parameters will be transferred to the satellite system and are available for our debugging purposes.

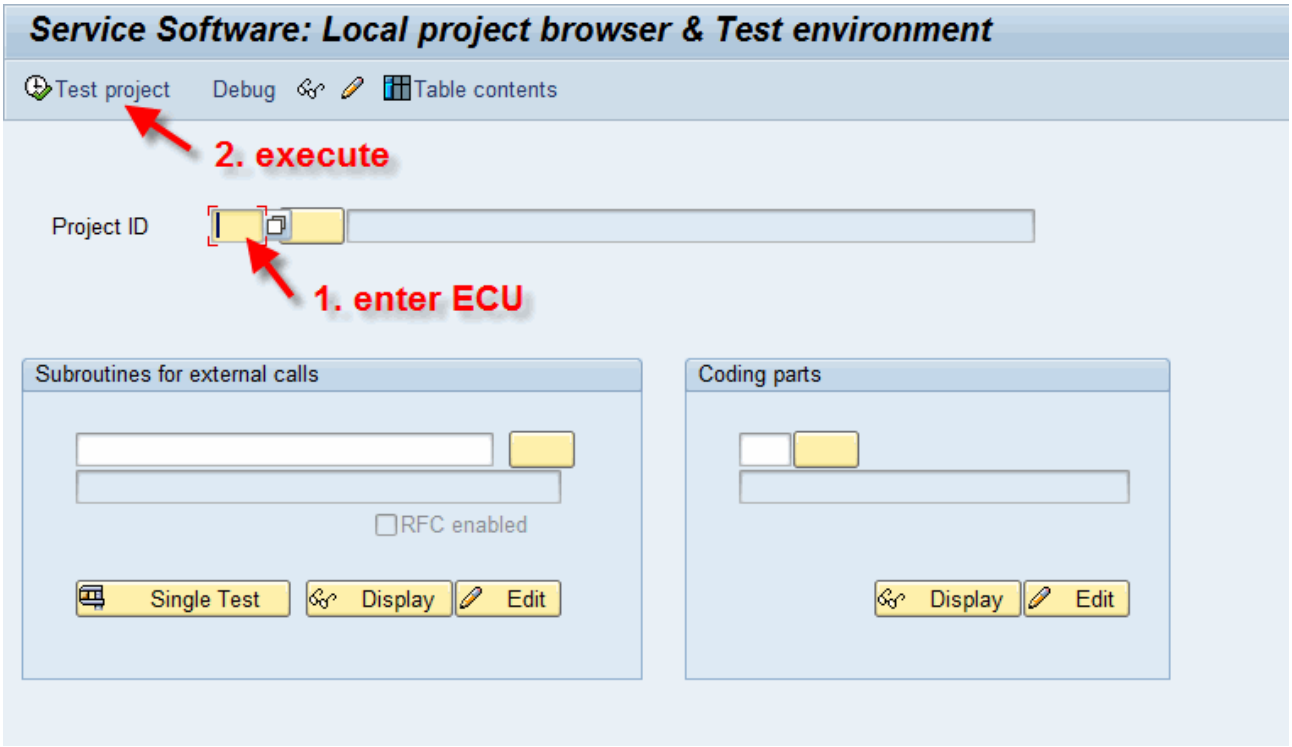

Figure 8 Entry Screen Projectbrowser

Enter in the field Project ID the value ECU and execute the function ’Test Project’ (or just press ENTER). The next screen looks like shown in figure 9.

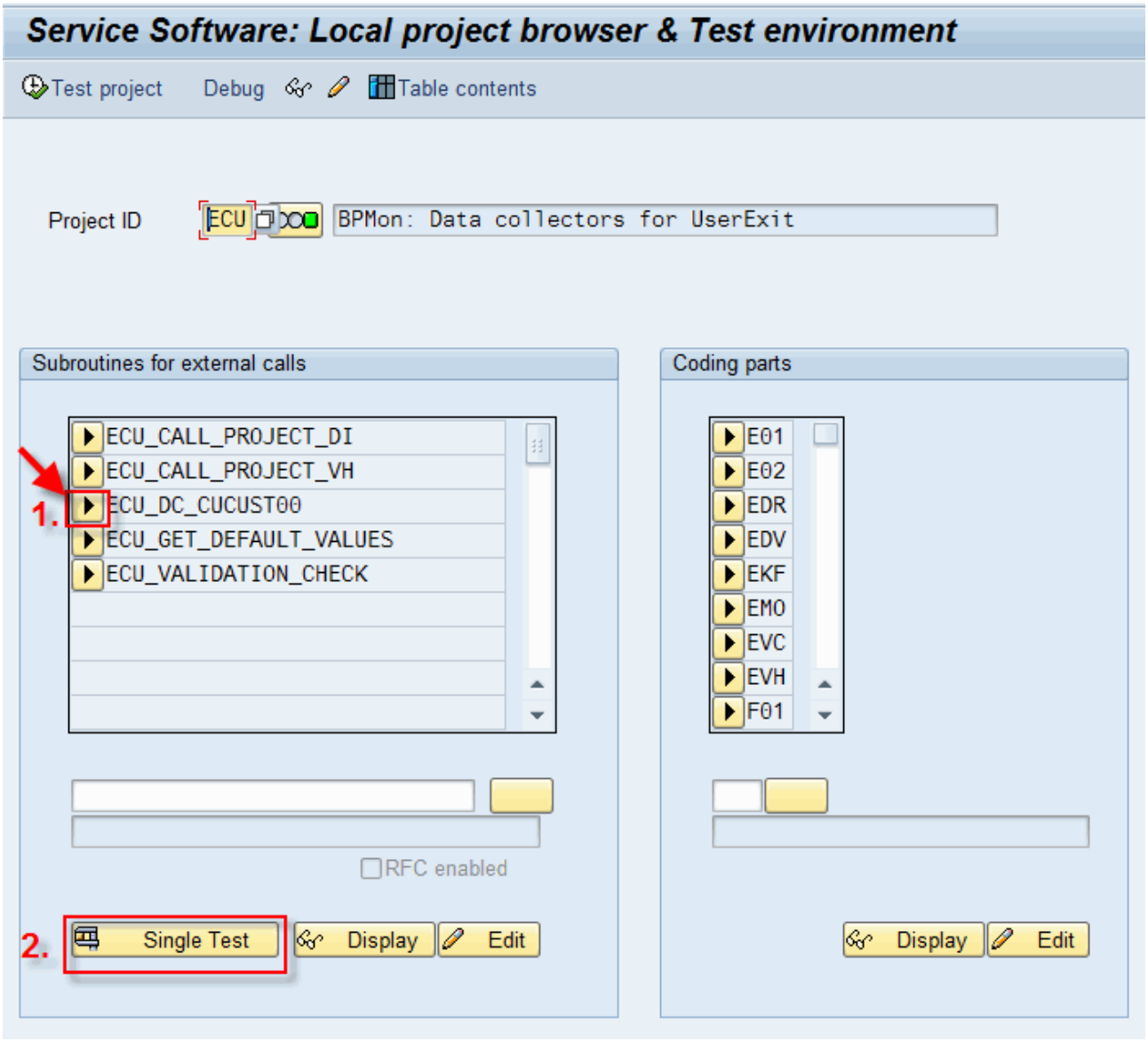

Figure 9 Project browser screen 2

Here you mark at first the arrow in front of the subroutine you want to test which is in our case ECU_DC_CUCUST00. The name of the subroutine is copied into the field under the table. Then you click on the button single test. A popup will be shown where you have to choose the execution type ‘Direct’ or ‘Debugging’ and then you can start with debugging your collector.

You will stop at a hardcoded break-point:

```
382    IF NOT ( GF_DYNP0240_DEBUGGING_FLAG IS INITIAL ).
383      *** make two single steps in the debugger ***
384      ***        to jump into subroutine        ***
385      BREAK-POINT.                          "#EC NOBREAK
386    ENDIF.
387    CALL_BPM_COLLECTOR.
388    *** press 'continue' F8 from here
389  ENDIF.
390  *  jump into viewer
```

Figure 10 Break Point in Projectbrowser

Press F5 – F5 – F7 – F5 and you will stop at the form routine ECU_DC_CUCUST00.

Debug this form routine and stop at the following coding line:

READ TABLE PT_CUSTTABA WITH KEY SOLUTION = PF_SOLUTION.

(See also the next figure). Check the content of the field PF_SOLUTION.

This field must contain the technical name of the solution where the monitoring objects are part from. If this is not the case you have to change the field content with the correct solution name in the debugger mode. (It is very usual that this field does contain the wrong solution number, when you have activated more than 1 solution for business process monitoring in your solution manager).

This is the way how to find out the correct solution number: start the business process monitoring setup session of the solution for which you want to get the solution MonID. In the Menu click on Goto -> Technical Functions -> Attribute Editor. Open the node 'SESSION' -> 'GENERAL' -> Soltion_MonID. Make a double click on this node and you will it see in the area 'Attribute values'.

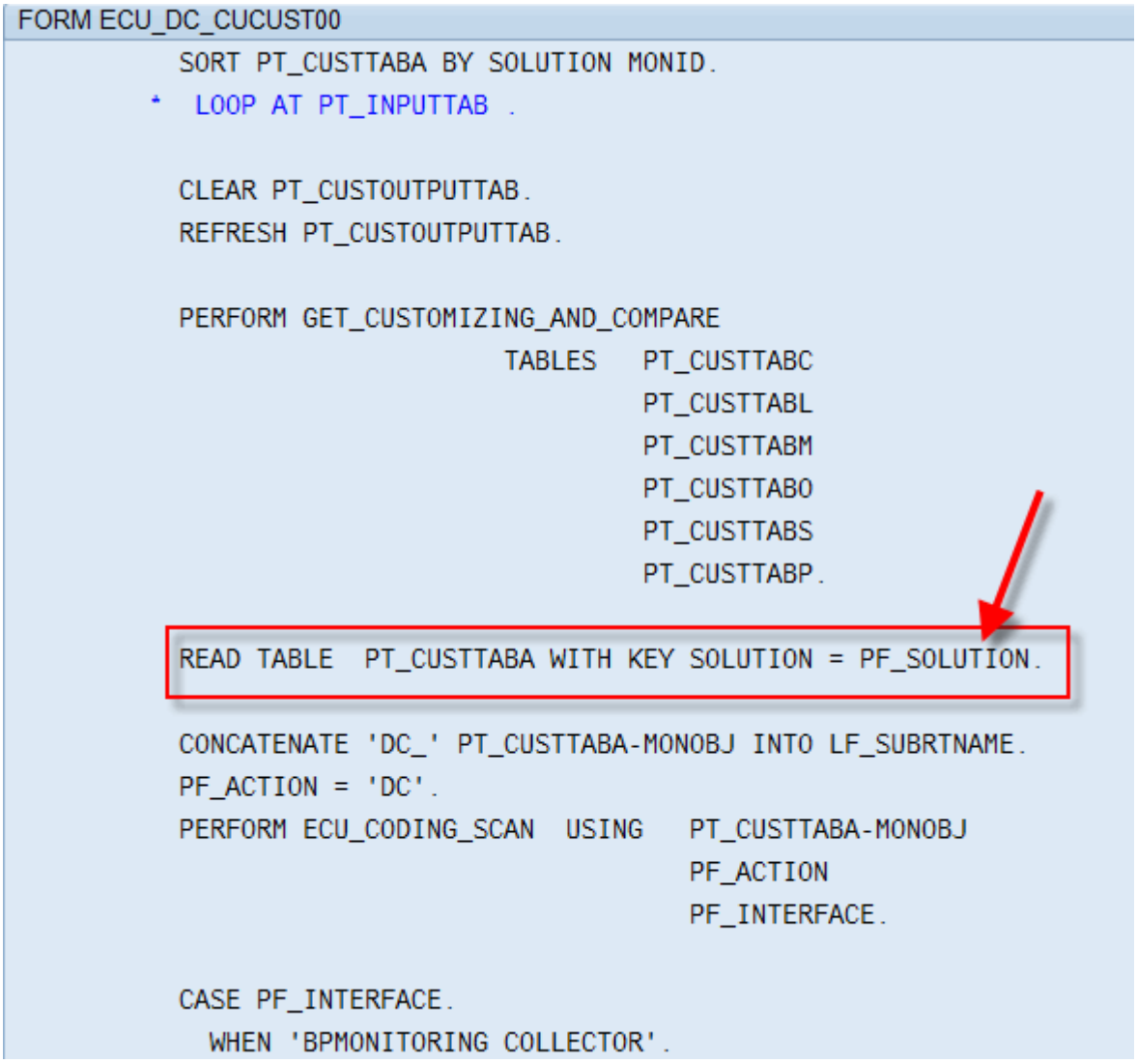

Figure 11 Change content of PF_SOLUTION

After correcting the content of PF_SOLUTION process further with debugging. A few coding lines later your user exit will be called with a coding line like this:

PERFORM (LF_SUBRTNAME) IN PROGRAM Z_BPM_ECU_COLLECTOR

4.3.2. Form DC_CUCUSTXX

As already mentioned the prerequisite is the existence of the report Z_BPM_ECU_COLLECTOR. Please create this report with exactly this name in your satellite system.

Before we start to implement the first user specific coding into the report Z_BPM_ECU_COLLECTOR we have to add the following line into this report:

include /SSA/IET.

This include is part of the ST-A/PI and contains some data definitions which will be used later.

We also need some data definitions in the beginning of this report. We need the definition of the result structure in which we want to select the data and we have to define the definition of the field catalogue if we want to provide a detail info list for this monitoring object (if a detailed result list is not necessary we can skip the definition of the field catalogue).

In our example we define a result structure which is called LT_CUST00_01 and a field catalogue for showing the detail list which is called LT_CUST00_01_FC. The coding after these first steps is shown in the following Figure 12 Example Coding (Step 1)Figure 12.

```
REPORT Z_BPM_ECU_COLLECTOR.

INCLUDE Z>IETINC.

TYPES: BEGIN OF TS_CUST02_DETAIL,
         BNAME TYPE USR02-BNAME,
         UFLAG TYPE USR02-UFLAG,
         USTYP TYPE USR02-USTYP,
       END OF TS_CUST02_DETAIL,

       TT_CUST02_DETAIL TYPE STANDARD TABLE OF TS_CUST02_DETAIL
        WITH DEFAULT KEY.

DATA: LT_CUST02_01 TYPE TT_CUST02_DETAIL.

DATA: BEGIN OF LT_CUST02_01_FC OCCURS 0,
       BNAME LIKE USR02-BNAME,
       UFLAG LIKE USR02-UFLAG,
       USTYP LIKE USR02-USTYP,
      END OF LT_CUST02_01_FC.
```

Figure 12 Example Coding (Step 1)

Now copy the coding template for DC_CUCUSTXX into the report Z_BPM_ECU_COLLECTOR (see Figure 7 in chapter 4.3) and insert it after the data definition shown in Figure 12. In our example we have to rename it to DC_CUCUST00.

Now the report Z_BPM_ECU_COLLECTOR looks like this (without the commented lines describing the interface):

```
FORM DC_CUCUST00
   TABLES PT_INPUTTAB   TYPE TT_INPUTTAB
          PT_CUSTTABC   TYPE TT_APPMONC
          PT_CUSTTABL   TYPE TT_APPMONL
          PT_CUSTTABM   TYPE TT_APPMONM
          PT_CUSTTABO   TYPE TT_APPMONO
          PT_CUSTTABA   TYPE TT_APPMONA
          PT_CUSTTABS   TYPE TT_APPMONS
          PT_CUSTTABP   TYPE TT_APPMONP
          PT_OUTPUTTAB  TYPE TT_OUTPUTTAB
    USING PF_SOLUTION   TYPE TF_SOLUTION
          PF_PARAMETER  TYPE TF_PARAMETER
 CHANGING PF_SUBRC      LIKE SY-SUBRC.

  DATA:
    LF_SUBROUTINE(30),
    LS_INPUT          TYPE TS_APPMONA,
    LS_OUTPUT         TYPE TS_APPMONI,
    LS_CUSTTABS       TYPE TS_APPMONS,
    LT_CUSTTABS       TYPE TT_APPMONS.

*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* SORT TABLE PT_CUSTTABA BY KEYFIGURE TO MAKE SURE FORM FOR MONITORING
* OBJECT IS CALLED FIRST BECAUSE MONITORING OBJECT HAS
* ALWAYS KEYFIG = 00.
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  SORT PT_CUSTTABA BY KEYFIG.

  LOOP AT PT_CUSTTABA INTO LS_INPUT   "call subroutine for each counter
   WHERE SOLUTION = PF_SOLUTION
   AND      MONOBJ = 'CUCUST00'.

* fill lt_custtabs with entries for one counter
  CLEAR LT_CUSTTABS.
  LOOP AT PT_CUSTTABS INTO LS_CUSTTABS
    WHERE SOLUTION = PF_SOLUTION
    AND       MONOBJ = 'CUCUST00'
    AND           CNTR = LS_INPUT-CNTR
    AND         MONID  = LS_INPUT-MONID.
   APPEND LS_CUSTTABS TO LT_CUSTTABS.
  ENDLOOP.

*   CREATE FORM NAME:
*   MONITORING OBJECT (CUCUST00)
*   FOR EACH KEYFIGURES( 00  LS_INPUT-KEYFIG)

  CONCATENATE 'KEYFIG_SUBROUTINECUCUST00_'
                  LS_INPUT-KEYFIG
                  INTO LF_SUBROUTINE.
```

```
    PERFORM (LF_SUBROUTINE) IN PROGRAM (SY-REPID)
      USING
        LS_INPUT     "input
        LT_CUSTTABS
      CHANGING
        LS_OUTPUT    "output
        PF_SUBRC
      IF FOUND.

  IF PF_SUBRC = 0.
*   STORE RESULT
  IF NOT LS_OUTPUT IS INITIAL.
      APPEND LS_OUTPUT TO PT_OUTPUTTAB.
      CLEAR LS_OUTPUT.
     ENDIF.
    ENDIF.
  ENDLOOP.
ENDFORM.                    "DC_CUCUST00
```

Figure 13 Example Coding (Step 2)

The following internal tables are the most important ones:

- pt_custtaba: provides the information about the alert thresholds for each counter.
  This table contains 1 entry for every counter plus 1 entry for keyfigure 00 (which is not relevant for this purpose and can be ignored).
- pt_custtabs: provides the information about the selection parameter in form of a range.
  This table contains 1 entry for every parameter and counter.
- pt_outputtab: this table has to be filled in the user exit with the measured value and the alert information

IMPORTANT: The coding template provided until ST-A/PI 01M does not evaluate pt_custtabs and therefore has to be adopted. The interface for the dynamical call of the form routine KEYFIG_SUBROUTINEXX_YY should be extended. But we do not need the complete table pt_custtabs, we need only the entries which are related to the same counter of a keyfigure. So the coding template has to be extended as follows (green marked lines):

```
 sort pt_custtaba by keyfig.
 loop at pt_custtaba into ls_input.
     CLEAR LT_CUSTTABS.
     LOOP AT PT_CUSTTABS INTO LS_CUSTTABS
       WHERE SOLUTION = PF_SOLUTION
       AND   MONOBJ   = 'CUCUST02'
       AND   CNTR     = LS_INPUT-CNTR
       AND   MONID    = LS_INPUT-MONID.
       APPEND LS_CUSTTABS TO LT_CUSTTABS.
     ENDLOOP.
```

Then you have to add the table lt_custtabs to the interface so the coding should look like this:

```
  perform (lf_subroutine)  in program (sy-repid)
                  using    ls_input    "input
                           lt_custtabs
                changing ls_output  "output
                     pf_subrc   if found.
```

Short explanation what happens in this form routine:
This form routine is called per monitoring object.
In table pt_custtaba is the information for all counters which were created for this monitoring object. We loop at pt_custtaba and call the form KEYFIG_SUBROUTINECUCUSTXX_YY for each entry in pt_custtaba.
XX has to be replaced with the number of the monitoring object (by the developer) and YY is replaced with the number of the keyfigure (dynamically during runtime).
The form KEYFIG_SUBROUTINECUCUSTXX_YY contains the coding for calculating the measured value and the kind of alert. The result is given back with the structure ls_output.

The table pt_custtaba does not only contain entries for every keyfigure and counter, it also contains 1 entry for keyfigure 00. You have 2 possibilities to handle this entry:

1. You create a form routine named KEYFIG_SUBROUTINECUCUST00_00. You can enter coding in this form routine for example to do some pre-selections which are relevant for all keyfigures. Then you do not have to do these kinds of selections for every keyfigure.
   This form routine is also part of the template for the DC_CUCUSTXX. (see also the following figure 12)
2. You can also ignore this entry in pt_custtaba. Then you don't need this form routine and you can leave it away if you don't have the opportunity for coding which is relevant for all keyfigures.

```
*&~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*
*&     FORM  KEYFIG_SUBROUTINECUCUST00_00
*&~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*
*      FORM FOR MONITORING OBJECT CUCUST00 KEYFIGURE 00
*

*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*
FORM KEYFIG_SUBROUTINECUCUST00_00
                USING PS_INPUT    TYPE TS_APPMONA
              CHANGING PS_OUTPUT    TYPE TS_APPMONI
                   PF_SUBRC  LIKE SY-SUBRC.
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* FORM FOR MONITORING OBJECT ITSELF
* Example: INITIALIZE SOME VALUES
ENDFORM.                " KEYFIG_SUBROUTINECUCUST00_00
```

Figure 14 Original Coding Template for keyfigure subroutine

Important: Because of the added pt_custtabs table in the interface you have to add it also here in the coding template. After adding this parameter (marked green) the corrected coding has to look like this:

```
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*
FORM KEYFIG_SUBROUTINECUCUST00_00
                USING PS_INPUT    TYPE TS_APPMONA
                      PT_CUSTTABS     TYPE TT_APPMONS
              CHANGING PS_OUTPUT    TYPE TS_APPMONI
                   PF_SUBRC  LIKE SY-SUBRC.
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* FORM FOR MONITORING OBJECT ITSELF
* Example: INITIALIZE SOME VALUES
ENDFORM.                " KEYFIG_SUBROUTINECUCUST00_00
```

Figure 15 Adopted Coding Template for keyfigure subroutine (keyfigure 00)

4.3.3. Form KEYFIG_SUBROUTINEXX_YY

Copy the coding template for the keyfigure (form KEYFIG_SUBROUTINECUCUSTXX_YY) (see figure 7 in chapter 3.3) into the report Z_BPM_ECU_COLLECTOR. In our example we have to rename it to KEYFIG_SUBROUTINECUCUST00_01.

The coding template of this subroutine looks as follows (the interface is already extended with the PT_CUSTTABS table):

```
*&~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*
*&      FORM  KEYFIG_SUBROUTINECUCUST00_01
*&~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*
*       FORM FOR MONITORING OBJECT CUCUST00 KEYFIGURE YY
*
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*
FORM KEYFIG_SUBROUTINECUCUST00_01
                  USING PS_INPUT         TYPE TS_APPMONA
                        PT_CUSTTABS  TYPE TT_APPMONS
                CHANGING PS_OUTPUT TYPE TS_APPMONI
                         PF_SUBRC  LIKE SY-SUBRC.
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* PREPARE OUTPUT
* TO MAKE MAPPING BETWEEN INPUT AND OUTPUT POSSIBLE
* MANDATORY FIELDS OF OUTPUT TABLE
  CONCATENATE PS_INPUT-MONOBJ PS_INPUT-KEYFIG INTO PS_OUTPUT-ALERTTYPE.
* ALERT RATING TEXT LIKE RED, YELLOW, GREEN
  PS_OUTPUT-RATING = 'YELLOW'.
* ALERT MESSAGE TEXT
  PS_OUTPUT-ATEXT = 'Example Alert message text'.
* ALERT LEVEL FOR KEYFIGURES FROM TYPE STATUS
  PS_OUTPUT-ALERTLEVEL = 'RED or YELLOW'.
* ALERT AS NUMBER 1= GREEN ; 2 = YELLOW; 3 = RED
  PS_OUTPUT-ALERT = '1 or 2 or 3'.
* MEASURED VALUE WHICH LEADS TO THE ALERT
  PS_OUTPUT-AL_MEAS_VALUE = 1.
ENDFORM.                    " KEYFIG_SUBROUTINECUCUST00_01
```

Figure 16 Coding Template for Keyfigure Subroutine

This coding only shows how the structure PS_OUTPUT has to be filled to deliver the following information to the solution manager: the measured value, the alert level, the alert text, and the monitoring object for which the measurement and alerting was done.

For a better understanding the structure of PS_OUTPUT here is documented in parts (the structure consists of nearly 50 fields and most of them are not used by this collector):

- For identifying the keyfigure the following fields have to be filled:
  SOLUTION, MONID, CNTR, SESSNO, POBJECTNO, SOBJECTNO-> all this information is given with the input structure PS_INPUT. So the following coding line should be added to the template:
  MOVE-CORRESPONDING PS_INPUT TO PS_OUTPUT.
- The measured value has to be filled into PS_OUTPUT-AL_MEAS_VALUE.
- The field PS_OUTPUT-ALERTTYPE has to be filled as shown in the template by concatenating the strings MONOBJ and KEYFIG
- The field PS_OUTPUT-ALERT has to be filled with the alert level which is
  1 in case of a green alert
  2 in case of a yellow alert and
  3 in case of a red alert.
- The field PS_OUTPUT-RATING has to be filled with the corresponding text to the alert level (GREEN, YELLOW or RED)
- The field PS_OUTPUT-ATEXT can be used to provide an alert specific text which will be shown in the monitoring session in the solution manger.

The keyfigure subroutine for evaluating the number of locked user due to incorrect log on then will look as shown in the following figure 15.

(These coding shows 2 different types of selecting data from the table USR02: the first one is doing only a select count to provide a measured value but no detail list; the second select is necessary if later a detailed list should be provided. Only one of these selects is necessary.)

```
FORM KEYFIG_SUBROUTINECUCUST00_01
                  USING PS_INPUT   TYPE TS_APPMONA
                        PT_CUSTTABS    TYPE TT_APPMONS
                CHANGING PS_OUTPUT   TYPE TS_APPMONI
                        PF_SUBRC  LIKE SY-SUBRC.
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

  DATA: LV_COUNT TYPE I.

  DATA: LS_CUSTTABS LIKE LINE OF PT_CUSTTABS,
        LV_UFLAG LIKE USR02-UFLAG,
        RT_USTYP LIKE RANGE OF USR02-USTYP,
        LS_USTYP LIKE LINE OF RT_USTYP.

  LOOP AT PT_CUSTTABS INTO LS_CUSTTABS WHERE KEYFIG = '01'.
    CASE LS_CUSTTABS-PARA_ID.
      WHEN 'USER_TYPE'.
        LS_USTYP-SIGN   = LS_CUSTTABS-SIGN.
        LS_USTYP-OPTION = LS_CUSTTABS-OPTION.
        LS_USTYP-LOW    = LS_CUSTTABS-LOW.
        LS_USTYP-HIGH   = LS_CUSTTABS-HIGH.
        APPEND LS_USTYP TO RT_USTYP.
      WHEN 'UFLAG'.
        LV_UFLAG = LS_CUSTTABS-LOW.
    ENDCASE.
  ENDLOOP.

* select count is enough when a detailed list is not required
  SELECT COUNT(*) INTO LV_COUNT FROM USR02
    WHERE UFLAG EQ LV_UFLAG
    AND USTYP IN RT_USTYP.

* for providing a detailed list we need detailed information
  SELECT BNAME UFLAG USTYP FROM USR02
    APPENDING TABLE LT_CUST00_01
    WHERE UFLAG EQ LV_UFLAG
    AND USTYP IN RT_USTYP.

  DESCRIBE TABLE LT_CUST00_01 LINES LV_COUNT.

* PREPARE OUTPUT
* TO MAKE MAPPING BETWEEN INPUT AND OUTPUT POSSIBLE
* MANDATORY FIELDS OF OUTPUT TABLE

  MOVE-CORRESPONDING PS_INPUT TO PS_OUTPUT.
  CONCATENATE PS_INPUT-MONOBJ PS_INPUT-KEYFIG INTO PS_OUTPUT-ALERTTYPE.

  PS_OUTPUT-AL_MEAS_VALUE = LV_COUNT.

* 2 step rating
IF LV_COUNT < PS_INPUT-THRES_YELLOW.
    PS_OUTPUT-RATING = 'Green'.                    "#EC NOTEXT
    PS_OUTPUT-ALERT = 1.
  ELSEIF LV_COUNT < PS_INPUT-THRES_RED.
    PS_OUTPUT-RATING = 'Yellow'.                   "#EC NOTEXT
    PS_OUTPUT-ALERT = 2.
```

```
  ELSE.
    PS_OUTPUT-RATING = 'Red'.                    "#EC NOTEXT
    PS_OUTPUT-ALERT = 3.
  ENDIF.
  PS_OUTPUT-ATEXT = 'Number of locked User:'.
ENDFORM.                                  " KEYFIG_SUBROUTINECUCUST00_01
```

Figure 17 Adopted Coding template for Keyfigure Subroutine

The following explanations are only relevant if you want to realize the possibility of providing a detailed list for the collector result.

As a prerequisite for showing a detailed list we have to save the result list during the collector run into a table. The table we can use for this is the packed table /SSF/PTAB. This table is delivered also with the software component ST-A/PI. There are also function modules delivered with the ST-A/PI which can be used to write into this table or to read from this table.

We will store the internal table LT_CUST00_01, containing the detail list for monitoring object CUST00, keyfigure 01, into the packed table /SSF/PTAB.

When the function button ‘Detail Info’ is used in the monitoring session then the coding in the form DI_CUCUSTXX is executed. In this form routine we will provide the coding to read again the packed table /SSF/PTAB and to provide and show the detailed list which was stored before during the collector run.

For writing the result table into table /SSF/PTAB we have to extend the report Z_BPM_ECU_COLLECTOR with the following form routines:

- FORM DETAIL_INFO_KEY
- FORM PTAB_SAVE_STRHEX_TABLE_BY_NAME
- FORM HANDLE_DETAIL_LIST

The following chapters do only contain the content of these form routines. Please copy this coding into your report Z_BPM_ECU_COLLECTOR if you want to provide a detailed list for the results of your user specific collector.

When this is done you have to add the following coding at the end of form routine KEYFIG_SUBROUTINECUCUST00_01:

```
* store the detailed result table into /SSF/PTAB
  APPEND PS_OUTPUT TO LT_OUTPUT.

  PERFORM HANDLE_DETAIL_LIST
    TABLES
      LT_CUST00_01
      LT_OUTPUT
    USING
      'WRITE'
      'ECU'
      PS_INPUT-KEYFIG
      SPACE .
```

4.3.3.1. Form DETAIL_INFO_KEY

```
*&---------------------------------------------------------------------*
*&      Form  DETAIL_INFO_KEY
*&---------------------------------------------------------------------*
*       text
*----------------------------------------------------------------------*
*      -->PS_APPMONI
*      -->PF_COUNTER
*      <--PF_OBJKEY
*----------------------------------------------------------------------*
FORM DETAIL_INFO_KEY
  USING
    PS_APPMONI TYPE TS_APPMONI
    PF_COUNTER TYPE I
  CHANGING
    PF_OBJKEY TYPE /SSF/PTAB-OBJKEY.

  TYPES:
    BEGIN OF TS_INPUT,
      MANDT     LIKE SY-MANDT,
      SOLUTION  TYPE TS_APPMONA-SOLUTION,
      MONID     TYPE TS_APPMONA-MONID,
      ALERTTYPE TYPE TS_APPMONI-ALERTTYPE,
      CNTR      TYPE TS_APPMONA-CNTR,
      COUNTER(10)   TYPE C,
*       COUNTER   TYPE I,
    END OF TS_INPUT,
    TT_INPUT  TYPE STANDARD TABLE OF TS_INPUT.

  CONSTANTS:
    LC_OBJKEY_PREFIX(3) TYPE C VALUE 'DI_'.              "#EC NO_TEXT

  DATA:
    LF_KEYFIG(2)      TYPE C,
    LF_COUNTER(2)     TYPE C,
    LF_OBJKEY         LIKE /SSF/PTAB-OBJKEY,
    LT_LOG            LIKE /SSF/LOG OCCURS 0,
    LT_INPUT          TYPE TT_INPUT,
    LS_INPUT          TYPE TS_INPUT,
    LF_HASHKEY        TYPE MD5_FIELDS-HASH.

*---------------------------------------------------------

  CHECK NOT PS_APPMONI IS INITIAL.
  MOVE-CORRESPONDING PS_APPMONI TO LS_INPUT.
  MOVE PF_COUNTER TO LS_INPUT-COUNTER.
```

```
  IF PS_APPMONI-MANDT IS INITIAL.
    PS_APPMONI-MANDT = SY-MANDT.
    LS_INPUT-MANDT = SY-MANDT.
  ENDIF.
  APPEND LS_INPUT TO LT_INPUT.

  CALL FUNCTION 'MD5_CALCULATE_HASH_FOR_CHAR'
*     EXPORTING
*       DATA               =
*       LENGTH               = 0
*       VERSION              = 1
    IMPORTING
      HASH               = LF_HASHKEY
    TABLES
      DATA_TAB           = LT_INPUT
    EXCEPTIONS
      NO_DATA            = 1
      INTERNAL_ERROR     = 2
      OTHERS             = 3
          .
  IF SY-SUBRC <> 0.
* MESSAGE ID SY-MSGID TYPE SY-MSGTY NUMBER SY-MSGNO
*        WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4.
  ENDIF.

  CONCATENATE
    LC_OBJKEY_PREFIX
    LF_HASHKEY
    ' '
    _
    PS_APPMONI-ALERTTYPE
    ' '
    _
    PS_APPMONI-MONID
    INTO PF_OBJKEY.

ENDFORM.                    " DETAIL_INFO_KEY
```

Figure 18 Coding for Subroutine DETAIL_INFO_KEY

4.3.3.2. Form PTAB_SAVE_STRHEX_TABLE_BY_NAME

```
&---------------------------------------------------------------------*
*&      Form  PTAB_SAVE_STRHEX_TABLE_BY_NAME
*&---------------------------------------------------------------------*
*       text
*----------------------------------------------------------------------*
*      -->PT_TABLE
*      -->PT_LOG
*      -->PF_RELID
*      -->PF_PID
*      -->PF_OBJKEY
*----------------------------------------------------------------------*
FORM PTAB_SAVE_STRHEX_TABLE_BY_NAME
 TABLES
   PT_TABLE
   PT_LOG STRUCTURE /SSF/LOG
 USING
   VALUE(PF_RELID) LIKE /SSF/PTAB-RELID
   VALUE(PF_PID) LIKE /SSF/PTAB-PID
   VALUE(PF_OBJKEY) LIKE /SSF/PTAB-OBJKEY.

 DATA:
   LT_PACKEDTAB LIKE /SSF/PTAB OCCURS 0,
   LT_LOG       LIKE /SSF/LOG OCCURS 0,
   LF_SUBRC LIKE SY-SUBRC.

 IF NOT ( PF_RELID = 'TC' OR PF_RELID = 'TT' OR PF_RELID = 'TU' ).
   APPEND 'EInvalid Relid' TO LT_LOG.
   EXIT.
 ENDIF.
 PERFORM PACK_FIELD_OR_TABLE IN PROGRAM /SSF/PACK
   TABLES
     PT_TABLE
     LT_PACKEDTAB
     LT_LOG
   USING
     PT_TABLE     "header line
     PF_RELID
     PF_PID
     PF_OBJKEY
     'TAB'
     'NAME'
     'TYPESTRING TYPEHEX'
   IF FOUND.
 LF_SUBRC = SY-SUBRC.
 IF SY-SUBRC < 8.
   DELETE FROM /SSF/PTAB
```

```
    WHERE
      RELID   = PF_RELID  AND
      PID     = PF_PID    AND
      OBJKEY  = PF_OBJKEY.
  IF NOT LT_PACKEDTAB IS INITIAL.
    INSERT /SSF/PTAB FROM TABLE LT_PACKEDTAB.
  ENDIF.
 ENDIF.
 SY-SUBRC = LF_SUBRC.

ENDFORM.                    " PTAB_SAVE_STRHEX_TABLE_BY_NAME
```

Figure 19 Coding for Subroutine PTAB_SAVE_STRHEX_TABLE_BY_NAME

4.3.3.3. Form HANDLE_DETAIL_LIST

```
*&---------------------------------------------------------------------*
*&      Form  HANDLE_DETAIL_LIST
*&---------------------------------------------------------------------*
*       text
*----------------------------------------------------------------------*
*      -->PT_DETAIL
*      -->PT_APPMONI
*      -->PF_ACTION
*      -->PF_PID
*      -->PF_KEYFIG
*      -->PF_COUNTER
*----------------------------------------------------------------------*
FORM HANDLE_DETAIL_LIST
  TABLES
    PT_DETAIL
    PT_APPMONI TYPE TT_APPMONI
  USING
    PF_ACTION
    PF_PID
    PF_KEYFIG
    PF_COUNTER.

  DATA:
    LS_APPMONI            TYPE TS_APPMONI,
    LF_OBJKEY             LIKE /SSF/PTAB-OBJKEY,
    LT_LOG                LIKE /SSF/LOG OCCURS 0.

*---------------------------------------------------------------------

*   CHECK PT_APPMONI IS INITIAL.
  READ TABLE PT_APPMONI INTO LS_APPMONI INDEX 1.
  PERFORM DETAIL_INFO_KEY
    USING
      LS_APPMONI
      PF_COUNTER
    CHANGING
      LF_OBJKEY.

  CHECK NOT LF_OBJKEY IS INITIAL.
  CASE PF_ACTION.
    WHEN 'WRITE'.
      PERFORM PTAB_SAVE_TABLE_BY_NAME(/SSF/ULIB)
        TABLES
          PT_DETAIL
          LT_LOG
        USING
```

```
      'TC'
      PF_PID
      LF_OBJKEY.

    IF SY-SUBRC = 8.
     PERFORM PTAB_SAVE_STRHEX_TABLE_BY_NAME
      TABLES
       PT_DETAIL
       LT_LOG
      USING
       'TC'
       PF_PID
       LF_OBJKEY.
    ENDIF.
   WHEN 'READ'.
    PERFORM PTAB_READ_TABLE(/SSF/ULIB)
     TABLES
      PT_DETAIL
      LT_LOG
     USING
      'TC'
      PF_PID
      LF_OBJKEY.
   WHEN OTHERS.
  ENDCASE.
ENDFORM.                " HANDLE_DETAIL_LIST
```

Figure 20 Coding for Subroutine HANDLE_DETAIL_LIST

4.3.4. Form DI_CUCUSTXX

The form routine DI_CUCUSTXX is also monitoring object specific and has to be named DI_CUCUST00 in our example. This form routine is called when in the solution manager in the business process monitoring session the button ‘Detail Info’ is pressed. You can implement the following coding example to show a detail result list of the last collector run (see figure 19). An ALV list should appear with the 3 columns for user name, user status and user type. The first column is defined as a hotspot; when you click on it, the display transaction SU01 is called, with the user name prefilled in the corresponding field.

The calling of the display transaction SU01 is realized by calling an additional form routine which is named CALL_SU01. The content of this form routine is also shown (see figure 20).

```
FORM DI_CUCUST00
  TABLES
     PS_OUTPUTTAB TYPE TT_APPMONI
   CHANGING
     PT_ERRMSG    TYPE TF_ERRMSG
     LF_SY_SUBRC  LIKE SY-SUBRC.


   DATA:  LT_APPMONI LIKE TABLE OF PS_OUTPUTTAB.

   DATA: LF_REPID LIKE SY-REPID,
         LF_ALVTAB TYPE SLIS_TABNAME,
```

```
        LT_FIELDCAT TYPE SLIS_T_FIELDCAT_ALV,
        LS_FIELDCAT LIKE LINE OF LT_FIELDCAT,
        HOTSPOT(30) TYPE C.

DATA: PF_USERCOMMAND TYPE SLIS_FORMNAME,
        ALV_GRID       TYPE C.

  FIELD-SYMBOLS: <ALV_TABLE> TYPE STANDARD TABLE.

  LF_REPID = SY-REPID.

 LF_ALVTAB = 'LT_CUST00_01_FC'.
 APPEND PS_OUTPUTTAB TO LT_APPMONI.

  PERFORM HANDLE_DETAIL_LIST
   TABLES
     LT_CUST00_01
     LT_APPMONI
   USING
      'READ'
      'ECU'
     SPACE
     SPACE.

  CALL FUNCTION 'REUSE_ALV_FIELDCATALOG_MERGE'
    EXPORTING
      I_PROGRAM_NAME         = LF_REPID
      I_INTERNAL_TABNAME     = LF_ALVTAB
      I_INCLNAME             = LF_REPID
    CHANGING
      CT_FIELDCAT            = LT_FIELDCAT
    EXCEPTIONS
      INCONSISTENT_INTERFACE = 1
      PROGRAM_ERROR          = 2
      OTHERS                 = 3.
  IF SY-SUBRC <> 0.

* MESSAGE ID SY-MSGID TYPE SY-MSGTY NUMBER SY-MSGNO
*     WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4.
 ENDIF.

 HOTSPOT = 'BNAME'.

  IF NOT HOTSPOT IS INITIAL.
    LOOP AT LT_FIELDCAT INTO LS_FIELDCAT
      WHERE FIELDNAME = HOTSPOT.
      LS_FIELDCAT-HOTSPOT = 'X'.
      LS_FIELDCAT-KEY     = 'X'.
      MODIFY LT_FIELDCAT FROM LS_FIELDCAT.
    ENDLOOP.
  ENDIF.

  CALL FUNCTION 'REUSE_ALV_GRID_DISPLAY'
     EXPORTING
       IT_FIELDCAT                 = lt_FIELDCAT
       I_CALLBACK_PROGRAM        = LF_REPID
       I_CALLBACK_USER_COMMAND  = 'CALL_SU01'
**       I_CALLBACK_PF_STATUS_SET = L_SET_STATUS
```

```
**          I_SAVE              = 'A'
**          IS_VARIANT            = LS_VARIANT
      TABLES
         T_OUTTAB             = lt_cust00_01
      EXCEPTIONS
         PROGRAM_ERROR   = 1
         OTHERS              = 2.

ENDFORM.
```

Figure 21 Coding Example for Subroutine DI_CUCUSTXX

The following example coding is a form routine named CALL_SU01 and it is used to call the display transaction SU01 when clicking at the hotspot field of the detailed list (which is in our case the user name). You should implement an authorization check on the transaction which is called here.

```
*&---------------------------------------------------------------------*
*&     Form  call_su01
*&---------------------------------------------------------------------*
*      text
*----------------------------------------------------------------------*
*     -->PF_UCOMM     text
*     -->PS_SELFIELD  text
*----------------------------------------------------------------------*
FORM CALL_SU01
 USING
   PF_UCOMM     TYPE SY-UCOMM
   PS_SELFIELD TYPE SLIS_SELFIELD.

 DATA: BNAME LIKE USR02-BNAME.

   CALL FUNCTION 'AUTHORITY_CHECK_TCODE'
        EXPORTING
             TCODE  = 'SU01'
        EXCEPTIONS
             OK     = 0
             NOT_OK = 2
             OTHERS = 3.

   IF SY-SUBRC <> 0.
     MESSAGE E172(00) WITH 'SU01'.
   ENDIF.

 IF PF_UCOMM = '&IC1'.
   BNAME = PS_SELFIELD-VALUE.
   SET PARAMETER ID 'XUS' FIELD BNAME.
   CALL TRANSACTION 'SU01' AND SKIP FIRST SCREEN.
 ENDIF.

ENDFORM.                                    "call_su01
```

Figure 22 Coding Example for calling a display transaction

4.3.5. Form VH_CUCUSTXX

If you want to provide F4-Help for the selection parameters it is necessary to add the form routine VH_CUCUSTXX for each monitoring object to the report Z_BPM_ECU_COLLECTOR. The suffix XX stands for the number of the monitoring object, in our example the name of the form routine for the F4-Help is VH_CUCUST00.

There is also a coding template for this form routine provided with the customizing report /SSA/EXM (see Figure 7).

In our example we want to provide an F4-Help for the parameter USER_TYPE.

As a prerequisite we have activated the corresponding flag ‘Check against F4 Help’ in the customizing report (see Figure 6).

We also maintained for this parameter the entries in field ‘Tech Name 1’ and ‘Tech Name 2’ (see also Figure 6). This is a prerequisite if we want to reuse the existing F4-Help of a data dictionary field, in this case field BNAME in table USR02.

```
*&---------------------------------------------------------------------*
*&      Form VH_CUCUST00
*&---------------------------------------------------------------------*
*       text
*----------------------------------------------------------------------*

* FORM TO GET VALUE HELP FOR PARAMETER VALUES IN MONITORING SETUP
FORM VH_CUCUST00
                  TABLES   PT_PARAMETER    TYPE TT_PARAMETER
                           PT_VH_CONTENT   TYPE TT_CUSTVH
                           PT_CUSTTABS     TYPE TT_APPMONS
                  USING    PF_PATTERN      TYPE TF_PATTERN
                           PF_MONOBJ       TYPE TS_APPMONO-MONOBJ
                           PF_KEYFIG       TYPE TS_APPMONO-KEYFIG
                           PS_CUSTTABS     TYPE TS_APPMONS.

  DATA:
  LF_MONOBJ    TYPE TS_APPMONO-MONOBJ, "Monitoring Object
  LF_KEYFIG    TYPE TS_APPMONO-KEYFIG, "Key Figure
  LF_PARAID    TYPE TS_PARAMETER-PARA_ID,"Parameter ID
  LS_PARAMETER TYPE TS_PARAMETER, "Line of parameters table
* APPMONP
  LT_APPMONP   TYPE TABLE OF TS_APPMONP, "Internal table for APPMONP
  LS_APPMONP   TYPE TS_APPMONP,          "Work Area for lt_appmonp
* PTAB_READ
  LT_LOG       TYPE /SSF/LOG OCCURS 0,  "Error Log from PTAB read
* Generic F4-HELPs
  LT_RETURN    TYPE TABLE OF DDSHRETVAL,  "Picked results from F4
  LS_RETURN    TYPE DDSHRETVAL,           "Work Area for return_tab
  LF_RETFIELD  TYPE DDSHRETVAL-RETFIELD,  "Field name to identfy result
* Return Codes
  LF_SUBRC     TYPE SY-SUBRC,
* Output
  LS_CUSTVH    TYPE TS_CUSTVH,    "Line for output table
  LT_CUSTVH    TYPE TT_CUSTVH.    "Local output table
  CONSTANTS: LC_PROJID_ECU TYPE /SSF/PTAB-PID VALUE 'ECU'.
* Fill internal table for APPMONP directly from cluster table PTAB
```

```
  PERFORM PTAB_READ_TABLE(/SSF/ULIB)
      TABLES   LT_APPMONP
               LT_LOG
      USING    'TC'
               LC_PROJID_ECU
               'P'.

* Handover values from interface
  LF_MONOBJ = PF_MONOBJ.
  LF_KEYFIG = PF_KEYFIG.

* For the new typ F4-help the parameter table has one entry only
  READ TABLE PT_PARAMETER INTO LS_PARAMETER INDEX 1.
  LF_PARAID = LS_PARAMETER-PARA_ID.

** Read APPMONP for determining the F4-Help
    READ TABLE LT_APPMONP INTO LS_APPMONP WITH KEY MONOBJ = LF_MONOBJ
    KEYFIG = LF_KEYFIG PARA_ID = LF_PARAID.
    LF_SUBRC = SY-SUBRC.

*     call generic  F4-help/dropdown-popup
        CALL FUNCTION 'F4IF_FIELD_VALUE_REQUEST'
          EXPORTING
            TABNAME           = LS_APPMONP-TECHNAME1
            FIELDNAME         = LS_APPMONP-TECHNAME2
            STEPL             = 0
          TABLES
            RETURN_TAB        = LT_RETURN
          EXCEPTIONS
            FIELD_NOT_FOUND   = 1
            NO_HELP_FOR_FIELD = 2
            INCONSISTENT_HELP = 3
            NO_VALUES_FOUND   = 4
            OTHERS            = 5.

        IF SY-SUBRC <> 0.
          MESSAGE ID SY-MSGID TYPE SY-MSGTY NUMBER SY-MSGNO
                  WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4.
        ELSE.
         CONCATENATE LS_APPMONP-TECHNAME1 '-' LS_APPMONP-TECHNAME2 INTO
          LF_RETFIELD.
          READ TABLE LT_RETURN INTO LS_RETURN WITH KEY RETFIELD =
          LF_RETFIELD.
        ENDIF.

    LS_CUSTVH-VALUE    = LS_RETURN-FIELDVAL.
    LS_CUSTVH-MONOBJ   = LF_MONOBJ.
    LS_CUSTVH-KEYFIG   = LF_KEYFIG.
    LS_CUSTVH-PARA_ID  = LF_PARAID.

    APPEND LS_CUSTVH TO LT_CUSTVH.
    PT_VH_CONTENT[] = LT_CUSTVH[].

ENDFORM.
```

Figure 23 Example Coding for Value Help

4.4. Setup Business Process Monitoring

The setup procedure follows the common setup procedure explained under http://service.sap.com/bpm.

Info: In order to have the newly created monitor available in the BPMon Setup Session it is necessary to reload the monitors from the managed system where the objects were developed.

The functionality for reloading the available monitoring objects is located at different locations in the setup session of business process monitoring depending on the ST-SER release.

The following Figure 24 shows the location of the reload button in service sessions with a ST-SER release before 701_2010_1.

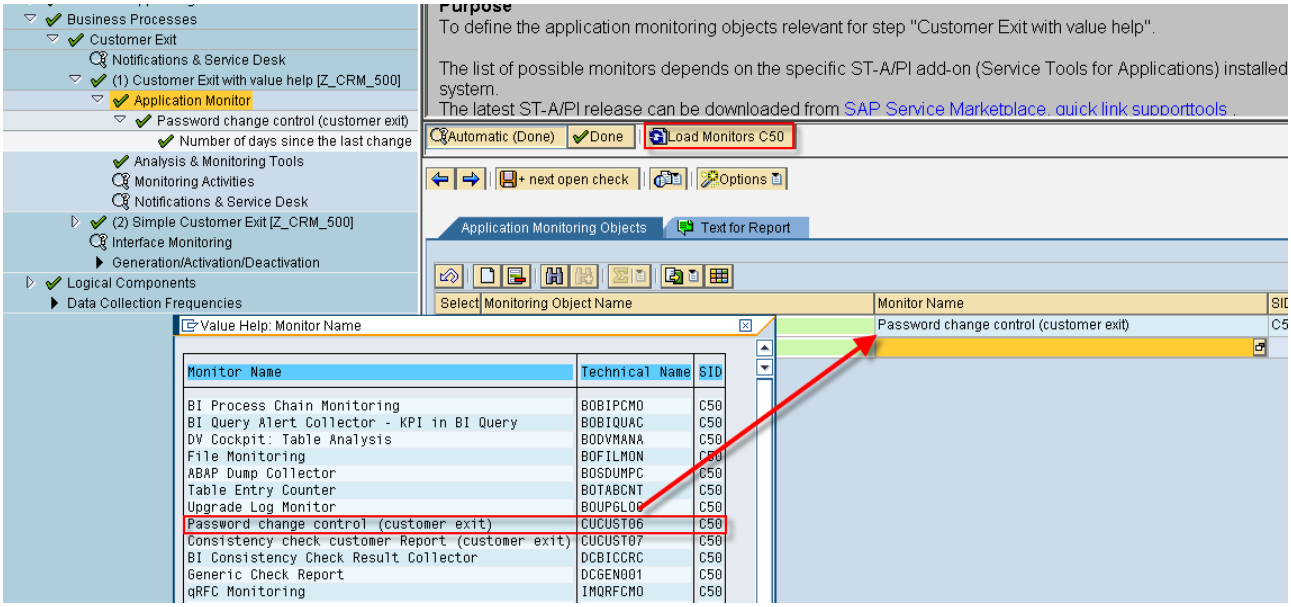

Figure 24 Reload Monitoring Objects from the managed system (old ST-SER)

The next Figure 25 shows the location of the reload functionality since the ST-SER release 701_2010_1.

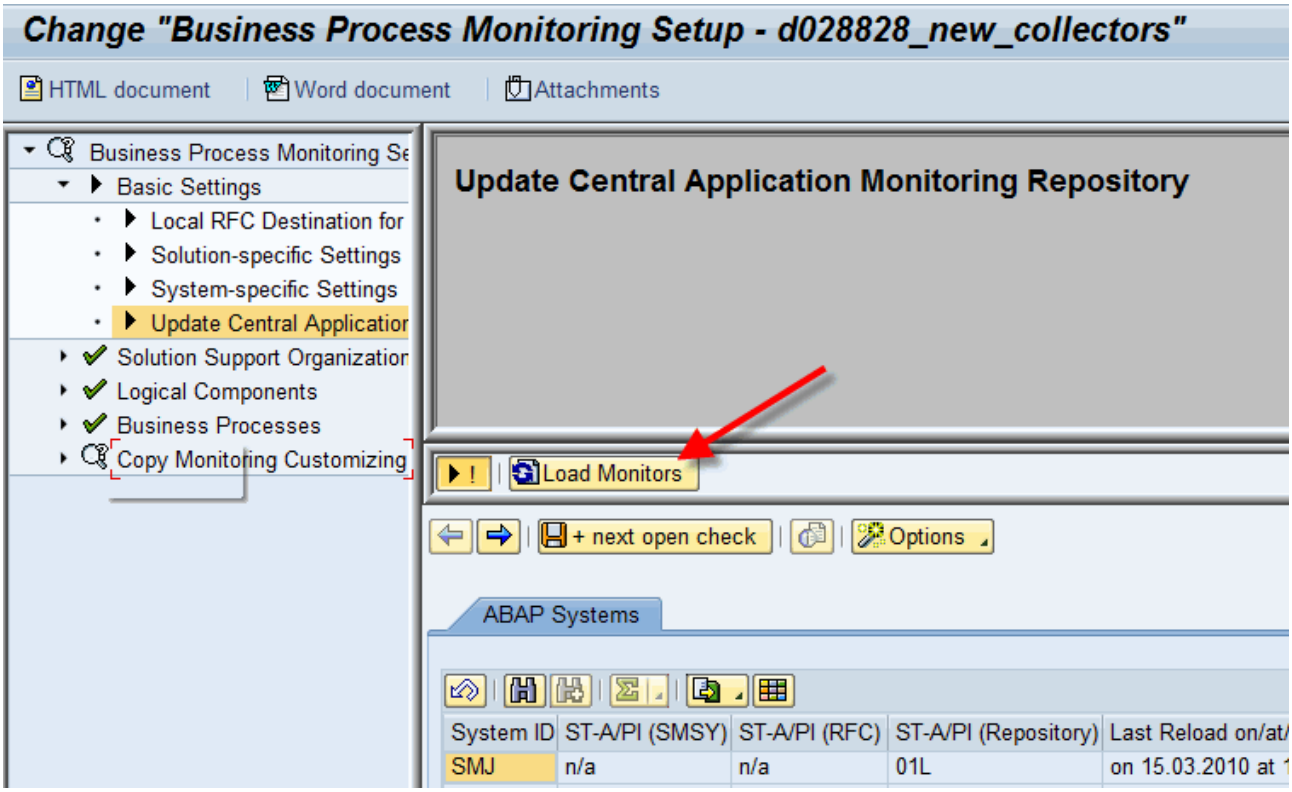

Figure 25 Reload Monitoring Objects from the managed system (new ST-SER)

5. Interfaces

5.1. ST-SER Version >= 2008_2

The call of the subroutine with the new interface will look like this:

```
PERFORM dc_cucustXX IN PROGRAM z_bpm_ecu_collector
                      TABLES   PT_INPUTTAB
                               PT_CUSTTABC
                               PT_CUSTTABL
                               PT_CUSTTABM
                               PT_CUSTTABO
                               PT_CUSTTABA
                               PT_CUSTTABS
                               PT_CUSTTABP
                               PT_OUTPUTTAB
                      USING    PF_SOLUTION
                               PF_PARAMETER
                      CHANGING PF_SUBRC
```

Tables **PT_CUSTTABS and PT_CUSTTABP** are new in the latest delivered interface.

PT_CUSTTABS = Table containing the customizing of the SAP Solution Manager. The content of this table are the parameters, counters and select options that the customer has maintained previously for every Application Monitoring Object defined. Structure: appmons = ts_appmons, (Active Alert Monitoring Object / Key Figure / Counter and Select Options)

Field	Description
SOLUTION	Solution ID
POBJECTNO	Process object number (in Solution Repository)
SOBJECTNO	Process step object number (in Solution Repository)
SID	System ID
MONTY	Monitoring Type
MONID	Monitoring ID
MONOBJ	Alert Monitoring Object
KEYFIG	Key Figure
PARAM	Parameter
PARA_ID	Parameter ID
CNTR	Row Counter (sel.opt. line counter)
SIGN	'I'ncluding or 'E'xcluding
OPTION	Comparison operator
LOW	Low value

HIGH	High value
MANDT	Client

PT_CUSTTABP = Table containing the definition of the customizing which is visible in the SAP Business Process Setup session.

5.2. ST-SER Version <= 2008_1

The call of the subroutine will look like this:

```
PERFORM dc_cucustXX IN PROGRAM z_bpm_ecu_collector
                    TABLES  pt_inputtab
                            pt_custtabc
                            pt_custtabl
                            pt_custtabm
                            pt_custtabo
                            pt_custtaba
                            pt_custoutputtab
                    USING   pf_solution
                            pf_parameter     "lf_pid
                    CHANGING pf_subrc .
```

- **PT_CUSTTABA** = Table containing the customizing (thresholds) of the SAP Solution Manager. The content of this table are the parameters that the customer has maintained previously for every Application Monitoring Object defined. Structure: appmona = ts_appmona, (Active Alert Monitoring Object / Key Figure)
- **A P P M O N A**

Field	Description
solution	Solution ID
sessno	Session number of monitoring session
pobjectno	Process object number (in Solution Repository)
sobjectno	Process step object number (in Solution Repository)
monid	Monitoring ID
monobj	Alert Monitoring Object
spras	Language
keyfig	Key Figure
keyfigty	Type of Key Figure (KZ,ST,LI)
para1	parameter 1
para2	parameter 2
para3	parameter 3

para4	parameter 4
para5	parameter 5
thres_red	Threshold red
thres_yellow	Threshold yellow
numdecimals	Number of decimals digits for threshold
unit_string	Unit of measure for thresholds as string
alertvalue	Alert value for status keyfigure 2=yel, 3=red
MANDT	Client
THRES_R2Y	Threshold red to yellow
THRES_Y2G	Threshold yellow to green
CNTR	Select option line counter
CNTR_TEXT	Shorttext for counter
timestamp	

- **PT_CUSTOUTPUTTAB** = Table containing alerts (from the data collector) to be transferred to the SAP Solution Manager. This is the table that customers have to fill in the data collector subroutine to deliver alerts to SAP Solution Manager. Structure: (= ts_appmoni, Output structure for alert information).

Field	Description
Solution	Solution ID
sessno	Session number of monitoring session
Pobjectno	Process object number (in Solution Repository)
Sobjectno	Process step object number (in Solution Repository)
Monid	Monitoring ID
Rating	Alert rating text
Adate	Date of alert creation
Atime	Time of alert creation
object1	Object information 1
object2	Object information 2
object3	Object information 3
info1	Information field 1
Info2	Information field 2
Info3	Information field 3

User	User name
Time	Time
Date	Date
Program	Program name
Tcode	Transaction code
Opmode	Operation mode (Batch, batch input)
Atext	Alert message text
Alertyel	Alert threshold yellow
Alertred	Alert threshold red
Alertlevel	Alert level for status
Alerttype	Alert type (start delay, duration , ...)
Alert	Alert as number
Monty	Monitoringtype
Msgty	Message type
Msgid	Message ID
Msgno	Message Number
threshr2y	Threshold red to yellow
threshy2g	Threshold yellow to green
Valunit	Value Unit
AL_MEAS_VALUE	Measured value

The next picture shows a connection between the customizing (“Which columns should be displayed?”) and the infrastructure (“Where is the content of those columns stored?”). As you have seen, the table **pt_custtaba** contains the customizing from the Set Up session, i.e. the input for the collector. In our example, it is the information which runtime in seconds has a job to exceed and what is the number of jobs that have to exceed it in order to raise an alert. The table **pt_custoutputtab** would be the output of the collector, and every line of this table will create an alert line in the Business Process Monitoring session in the SAP Solution Manager.

6. Further Information

How to create the coding in a 3-System Landscape (Development System, Test-System, Production System)?

1. Create the Report (Z_BPM_ECU_COLLECTOR) in your Development System
2. Create a new Solution only for testing of your new Customer Exit
    a. The leading system has to be your Development System or your Testing System respectively
3. After testing of your developed Customer Exit transport the ABAP report Z_BPM_ECU_COLLECTOR into your Productive System.
   You have to maintain the “customizing” for the Monitoring Objects manually in the Productive System. Currently there is no transport mechanism for the customizing.

What isn’t possible with the ST-SER Version <= 2008_1?

If you use ST-SER Version <= 2008_1 you cannot implement the Validation Check and the possibility to provide default values because the respective Push-Buttons are not available in the Setup Session.

There are 2 more restrictions coming from the fact that the table pt_custtabs is not available in the interface of the program DC_CUCUSTXX:

- the number of parameters which can be used per keyfigure is restricted to 5
- the parameters cannot be used as range parameter.

Related documents

Business Process Monitoring Set-Up Guide found in the Media Library on the SAP Service Marketplace under Quick Link /bpm (http://service.sap.com/bpm).