

**How-to Guide
SAP NetWeaver '04**



How To... Implement custom process types

Version 1.00 – November 2004

**Applicable Releases:
SAP NetWeaver '04
(Business Information Warehouse)**

© Copyright 2004 SAP AG. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP AG. The information contained herein may be changed without prior notice.

Some software products marketed by SAP AG and its distributors contain proprietary software components of other software vendors.

Microsoft, Windows, Outlook, and PowerPoint are registered trademarks of Microsoft Corporation.

IBM, DB2, DB2 Universal Database, OS/2, Parallel Sysplex, MVS/ESA, AIX, S/390, AS/400, OS/390, OS/400, iSeries, pSeries, xSeries, zSeries, z/OS, AFP, Intelligent Miner, WebSphere, Netfinity, Tivoli, and Informix are trademarks or registered trademarks of IBM Corporation in the United States and/or other countries.

Oracle is a registered trademark of Oracle Corporation.

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

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

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

Java is a registered trademark of Sun Microsystems, Inc.

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

MaxDB is a trademark of MySQL AB, Sweden.

SAP, R/3, mySAP, mySAP.com, xApps, xApp, SAP NetWeaver, and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries all over the world. All other product and service names mentioned are the trademarks of their respective companies. Data

contained in this document serves informational purposes only. National product specifications may vary.

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

These materials are provided "as is" without a warranty of any kind, either express or implied, including but not limited to, the implied warranties of merchantability, fitness for a particular purpose, or non-infringement. SAP shall not be liable for damages of any kind including without limitation direct, special, indirect, or consequential damages that may result from the use of these materials.

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

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.

1 Business Scenario

If you design a process chain, yet none of the SAP delivered process types do provide the functionality needed, you have basically two alternatives:

1. Use the process type 'ABAP Program' to call a report

This alternative is feasible when you have a stand-alone program, which does not have to interact with any other process type in the process chain.

A drawback of this process type is the fact that no status can be provided for successor process types. Therefore, the successor process types runs unconditional, which is very often not desired.

2. Implement your own process type based on an ABAP Object Oriented class

This alternative allows you to implement any desired functionality for your process type with the additional benefit of

- providing status information for successor process types
- gathering information from the predecessors of your own process type
- writing to a log, which is displayed in the process monitor
- using extended input screens for your process types, which exceed the standard variant input features

As Business Scenario for this paper we wanted to check within a process chain whether or not a succeeding process type of 'Local Process Chain' should be executed on the execution date of the process chain.

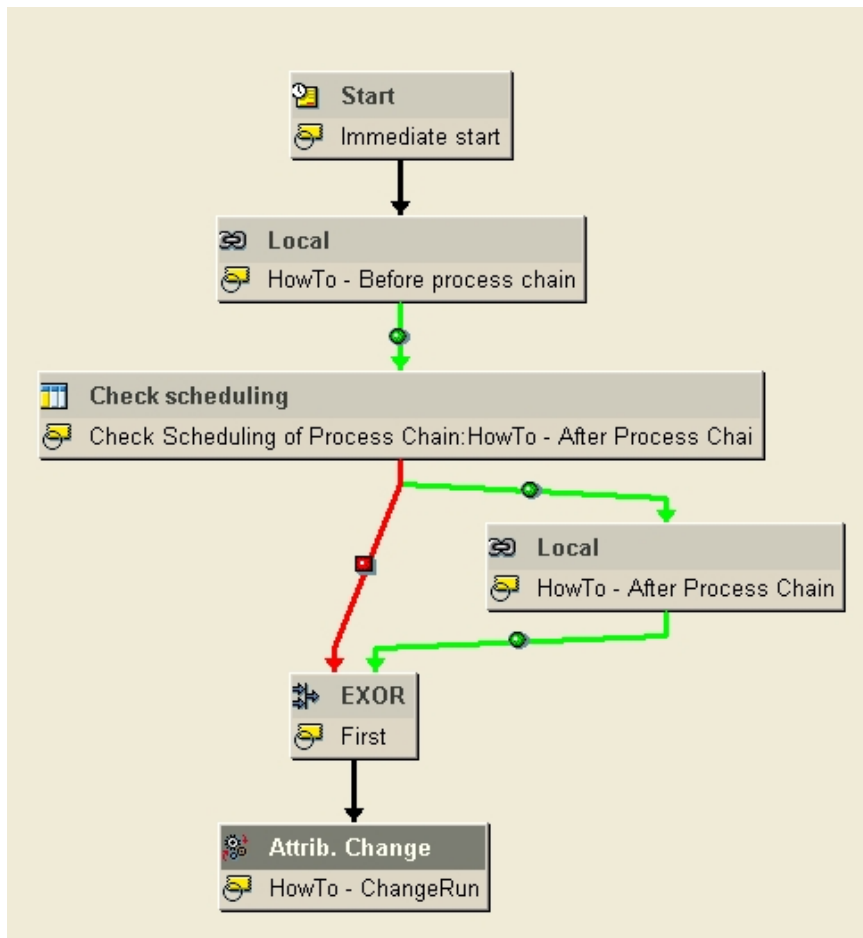
To customize the execution date of the 'Local Process Chain' we defined a custom table, which specifies at which date and time a process chain should run. In addition, you have the option of de-activating the setting by a flag provided in the table.

On the execution of our custom-defined process type we check whether a specified process chain – which should in general be the successor of our process type – is scheduled to run. If so, we provide the status 'Successful' (Green) for subsequent process types. If not the status is set to 'Errors' (Red).

A potential process chain, which depicts this scenario, is shown underneath. The process chain 'HowTo – After Process Chain' is only executed when it is scheduled to run in the custom table (see above).

Otherwise, it is skipped, and the next process type is executed.

DISCLAIMER: Our implementation is only an example for a realization of a custom-defined process type. We do not guarantee the correctness of the ABAP coding or stability of the solution in future releases.



2 Introduction

In order to realize the Business Scenario, you have to implement a custom-defined process type. The following steps have to be executed:

1. [Definition of the custom table](#)
2. [Implementation of ABAP OO class for process type](#)
3. [Specify settings for new process type](#)

We have focused in our implementation on the mandatory components of a custom-defined process. Of the optional components, like variant maintenance, log entries, providing information to the successor, transport connection and extension of the context menu, we have only implemented the variant maintenance. If you want to learn more about custom-defined process types, please read the documentation on process chains (Documentation → Administrator Workbench → Administration → Process Control → Process Chains → Process → Application Process → Implementing your own Process Type). The documentation provides you with information about which Interfaces have to be implemented for the respective functionality, when a certain functionality is called (during maintenance, run time or in the log view), and whether its implementation is mandatory or optional.

In addition, you can use the SAP delivered process types as templates or spin doctors for your custom-defined process types. To determine the implemented ABAP OO class for a specific process type, check the settings of the respective process type. In the maintenance of the process chains choose 'Settings →

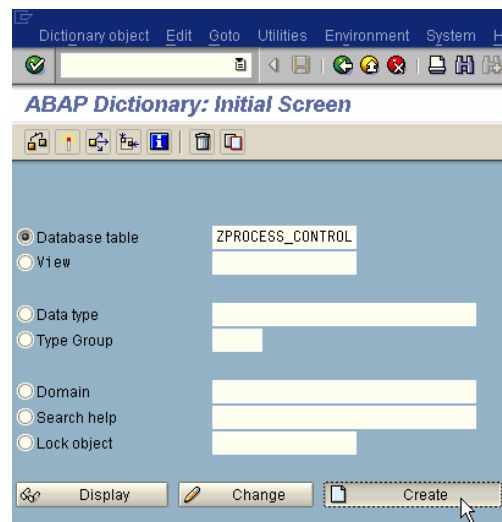
Maintain Process Types', double-click on the process type and check the object type name, which provides you with the name of the ABAP OO class for the implementation of the process type.

3 The Step By Step Solution

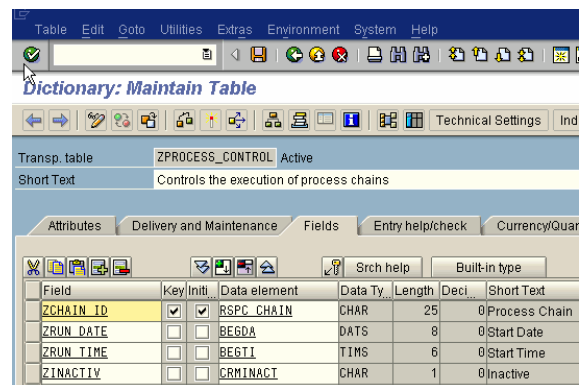
3.1 Definition of the custom table

The definition of the custom table is only an example for the implementation of our Business Scenario. If required the definition can be adapted to your needs. Please keep in mind, that the table is also referenced in the implemented methods of the ABAP OO class. Hence, they have to be adapted, too.

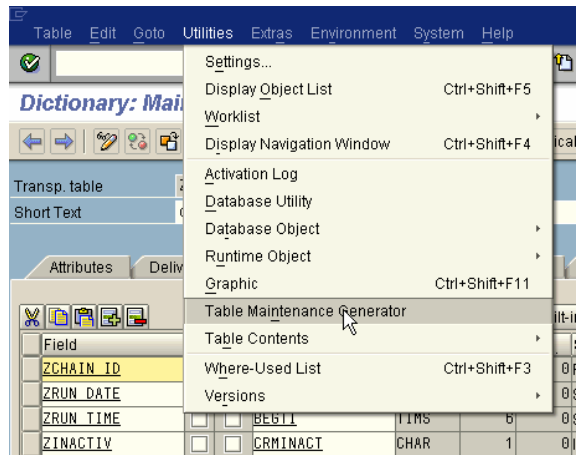
1. Enter the maintenance screen for DDIC objects (e.g. with transaction SE11). Specify the name of the table you want to create, and press the 'Create'-Button.



2. Specify the fields and the respective data elements that you want to use in your table.
In our example, we have chosen:
 - ZCHAN_ID – Process Chain name
 - ZRUN_DATE – Date, when the Process Chain should run
 - ZRUN_TIME – Time, when the Process chain should run
 - ZINAKTIV – The setting in the table is NOT active

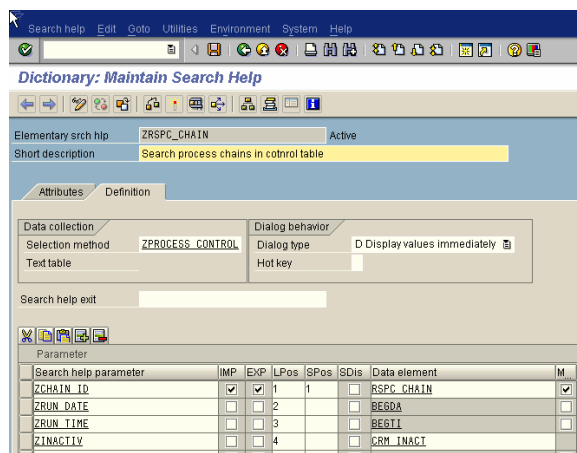


3. Activate the table (🔧), and create optionally a table maintenance and /or a search help (via transaction SE11) for supporting F4 value help in the process chain maintenance.
We have done both.



4. For the definition of the search help, we defined it as 'Elementary search help' with:
 - Selection Method: The name of our custom table (ZPROCESS_CONTROL)
 - The fields you want to display and select in the value help

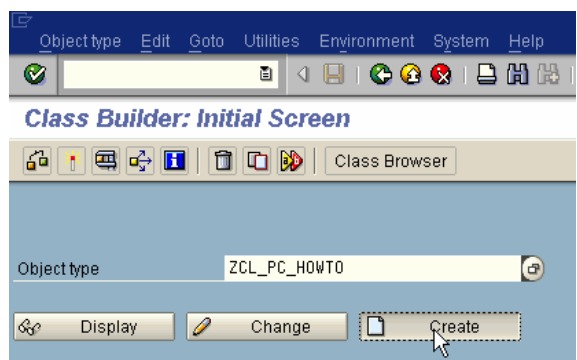
Activate the search value help (🔧).



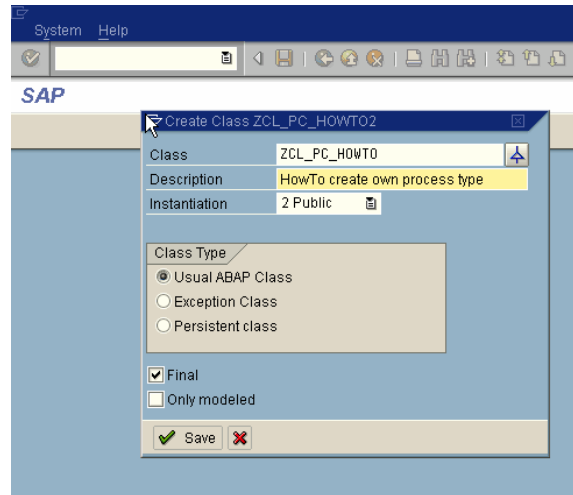
3.2 Implementation of ABAP OO class for process type

The sample coding for the implemented methods is provided in the [Appendix](#).

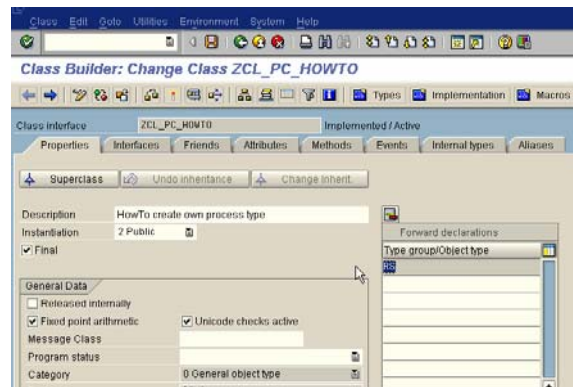
1. Enter the Class builder (e.g. with transaction SE24).
Specify the name of the class you want to define, and press the 'Create'-Button.



2. Choose the object type 'Class'.



3. Switch to the 'Properties' tab strip, and specify type group 'RS' in the 'Forward declaration' section.



4. Switch to the 'Interface' tab strip, and specify the following interfaces:

- **IF_RSPC_EXECUTE**

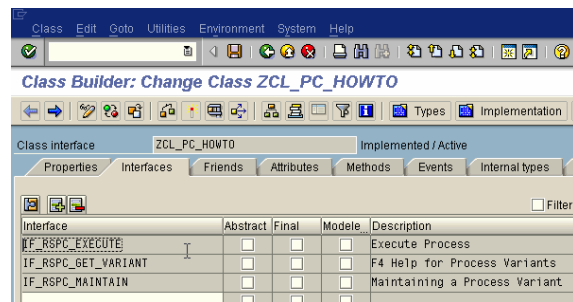
Mandatory interface, which is called on the execution of the process type

- **IF_RSPC_GET_VARIANT**

Optional interface for the selection of the variants

- **IF_RSPC_MAINTAIN**

Optional interface for the maintenance of the variants (in our case call of the Process Chain maintenance)



5. Switch to the 'Methods' tab strip. To satisfy the minimum requirements implement the following methods:

- IF_RSPC_GET_VARIANT
~GET_VARIANT

Pick Process Chain from custom table

- IF_RSPC_GET_VARIANT ~ EXISTS
Indicate variant exist (for syntax check of whole Process Chain)


- IF_RSPC_EXECUTE ~ EXECUTE
Performs the check against custom table

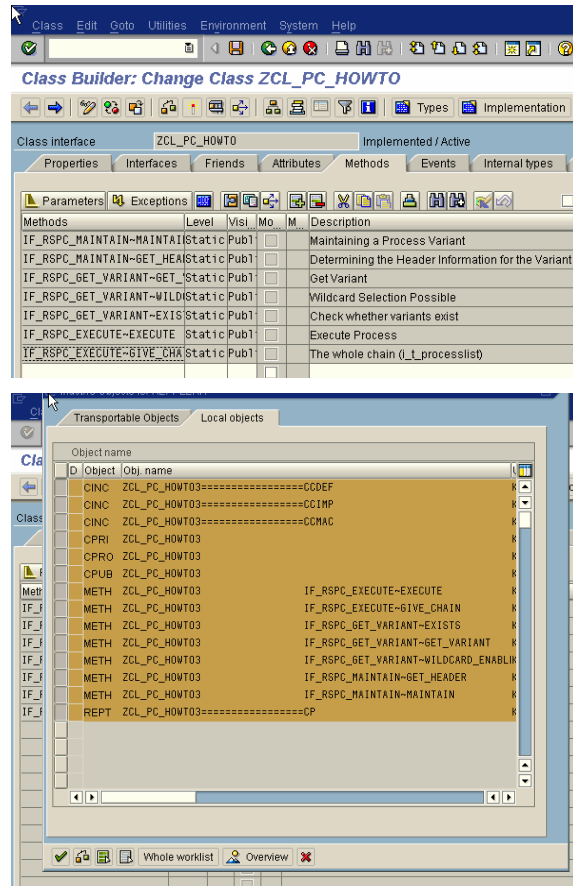
- IF_RSPC_MAINTAIN
~GET_HEADER

Define title for variant

- IF_RSPC_GET_VARIANT
~WILDCARD_ENABLED


The other methods are not needed necessarily.

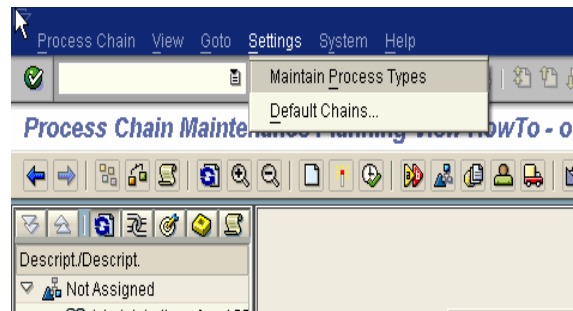
Activate the class definition ().



3.3 Specify settings for new process type

After you have implemented the class that performs the processing and maintenance of your custom-defined process type, you have to make the new process type known in the process chain maintenance.

1. Call transaction RSPC, or use the icon  in the Administrator Workbench. Open any of the existing process chains, or create a new one. Call the maintenance of the process types as shown in the screen-shot (menu 'Settings → Maintain Process Types').



2. Create a new entry in the table, and specify the following settings:

Process Type – Name of the process type
Short & Long Description

Object Type Name – Name of your implemented ABAP OO class (in our case 'ZCL_PC_HOWTO')

Object Type – CL ABAP OO Class

Possible Events – '2' Process ends "successful" or "incorrect"

Repeatable - 'X'

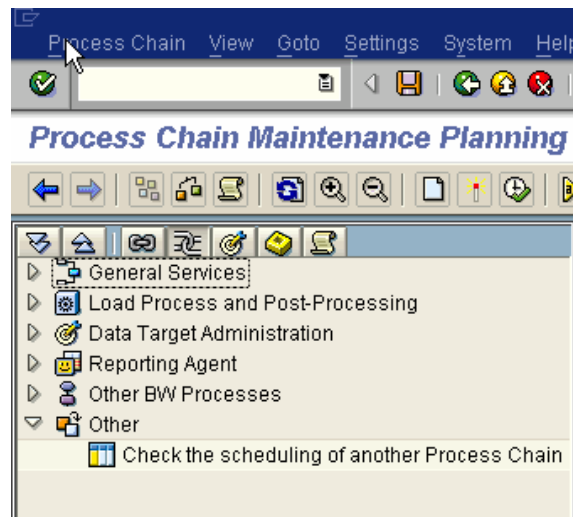
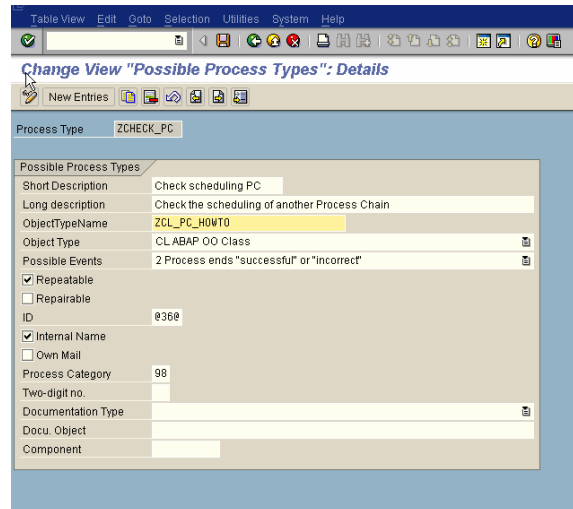
ID - Choose an icon via the F4-value help

Internal name – 'X'

Process Category – Choose a category where you want to find your process type later in the process chain maintenance

Save your entries.

3. Now you can find /pick your new process type underneath the process category specified. We have chosen the 'Others' category.



4 Appendix

The Appendix provides sample implementations for the methods of the implemented class. If you want to debug your implementation, set a break-point in the respective source coding, and use the OK-Code '=synchron' or '=debug' to run the process chain in synchron mode. This will cause the system to stop at the break-points in the source coding.

- **IF_RSPC_EXECUTE~EXECUTE**

Performs the check against custom table

```
method IF_RSPC_EXECUTE~EXECUTE .
```

```
data: ls_current_process type RSPC_S_PROCESSLIST,  
      ls_succ_process type RSPC_S_PROCESSLIST,  
      l_unique_id TYPE sysuuid_25,  
      lt_predecessor_info type RS_T_RSCEDST,  
      wa_zprocess_control type zprocess_control.
```

```
* ==== Get Instance ====
```

```
CALL FUNCTION 'RSSM_UNIQUE_ID'  
IMPORTING  
  e_uni_idc25 = l_unique_id.  
e_instance = l_unique_id.  
e_state = 'A'.
```

```
* ... Check wether process chain should run ...
```

```
SELECT SINGLE *  
  into wa_zprocess_control  
  FROM ZPROCESS_CONTROL  
  WHERE ZCHAIN_ID = i_variant.  
if sy-subrc <> 0. "=> not even in table  
  e_state = 'R'.  
else.  
  if wa_zprocess_control-zrun_date >= sy-datum and "=> not future AND  
    wa_zprocess_control-zinactiv is initial. " not switched off  
    e_state = 'G'.  
  else.  
    e_state = 'R'.  
  endif.  
endif.
```

```
endmethod.
```

- **IF_RSPC_GET_VARIANT ~GET_VARIANT**

Pick Process Chain from custom table

```
method IF_RSPC_GET_VARIANT~GET_VARIANT .
```

```
DATA:l_t_chainid TYPE rspc_t_chainid,  
      l_s_chainid TYPE rspc_s_chainid,
```

```

l_t_chaint TYPE rspc_t_chaint,
l_s_chaint TYPE rspcchaint,
l_s_chain  TYPE rspc_s_chain,
l_t_table  type table of sval,
l_s_table  like line of l_t_table.

```

* ... Read Process chains from control table and choose one

```

READ TABLE i_t_chain INTO l_s_chain INDEX 1.

```

```

l_s_table-tabname = 'ZPROCESS_CONTROL'.
l_s_table-fieldname = 'ZCHAIN_ID'.
l_s_table-value   = '*'.
append l_s_table to l_t_table.

```

```

CALL FUNCTION 'POPUP_GET_VALUES'
  EXPORTING
    POPUP_TITLE      = 'Process Chain to Check'
  TABLES
    FIELDS           = l_t_table.

```

* ... Check selected process chain

```

Read table l_t_table into l_s_table index 1.

```

```

CALL FUNCTION 'RSPC_GET_CHAIN'
  EXPORTING
    l_chain      = l_s_table-value
    i_objvers    = 'A'
    i_with_dialog = rs_c_true
    i_one_no_dialog = rs_c_true
  IMPORTING
    e_t_chain_ids = l_t_chainid
    e_t_chaint    = l_t_chaint
  EXCEPTIONS
    aborted_by_user = 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
    RAISING nothing_selected.
ENDIF.
READ TABLE l_t_chainid INTO l_s_chainid INDEX 1.
IF l_s_chainid-chain_id = l_s_chain-chain_id.
  MESSAGE i056(rspc).
  EXIT.
ENDIF.
e_variant = l_s_chainid-chain_id.
READ TABLE l_t_chaint INTO l_s_chaint
  WITH KEY chain_id = l_s_chainid-chain_id
    objvers = 'M'.
e_variant_text = l_s_chaint-txtlg.

```

```

endmethod.

```

- **IF_RSPC_GET_VARIANT ~ EXISTS**

Indicate variant exist (for syntax check of whole Process Chain)

```
method IF_RSPC_GET_VARIANT ~ EXISTS .
    r_exists = rs_c_true.
endmethod.
```

- **IF_RSPC_MAINTAIN ~ GET_HEADER**

Define title for variant

```
method IF_RSPC_MAINTAIN ~ GET_HEADER .
```

```
DATA:l_chain    TYPE rspc_chain,
      l_t_chainattr TYPE rspc_t_chainattr,
      l_t_chaint   TYPE rspc_t_chaint,
      l_s_chainattr TYPE rspcchainattr,
      l_s_chaint   TYPE rspcchaint.
```

```
l_chain = i_variant.
```

```
CALL FUNCTION 'RSPC_GET_CHAIN'
EXPORTING
    i_chain    = l_chain
    i_objvers  = 'M'
IMPORTING
    e_t_chainattr = l_t_chainattr
    e_t_chaint   = l_t_chaint
EXCEPTIONS
    aborted_by_user = 1
    OTHERS          = 2.
IF sy-subrc <> 0.
    MESSAGE ID sy-msgid TYPE 'I' NUMBER sy-msgno
        WITH sy-msgv1 sy-msgv2 sy-msgv3 sy-msgv4.
    EXIT.
ENDIF.
READ TABLE l_t_chainattr INTO l_s_chainattr INDEX 1.
MOVE-CORRESPONDING l_s_chainattr TO e_s_changed.
e_control = l_s_chainattr-control.
e_conttimestamp = l_s_chainattr-conttimestamp.
READ TABLE l_t_chaint INTO l_s_chaint INDEX 1.
concatenate 'Check Scheduling of Process Chain:' l_s_chaint-txtlg
    into e_variant_text .
```

```
endmethod.
```

- **IF_RSPC_MAINTAIN ~ MAINTAIN**

Maintenance of the custom table for scheduling.

```
method IF_RSPC_MAINTAIN ~ MAINTAIN .
```

```
DATA:l_chain    TYPE rspc_chain,
```

```
l_t_chaint TYPE rspc_t_chaint,  
l_s_chaint TYPE rspcchaint.
```

```
l_chain = i_variant.
```

* ... Call Maintenance of custom table ...

```
CALL FUNCTION 'VIEW_MAINTENANCE_CALL'  
EXPORTING  
  ACTION           = 'U'           "Change mode  
  VIEW_NAME        = 'ZPROCESS_CONTROL'.
```

```
endmethod.
```

- **IF_RSPC_GET_VARIANT ~ WILDCARD_ENABLED**
Whether the variant input popup should allow user entry.

```
method IF_RSPC_GET_VARIANT ~ WILDCARD_ENABLED .  
  result = ' '.  
endmethod.
```

www.sdn.sap.com/irj/sdn/howtoguides