ANNALS of Faculty Engineering Hunedoara – International Journal of Engineering
ISSN: 1584 - 2673

ERROR HANDLING AND MESSAGES WITH APPLICATION SERVER ABAP

[1.] Ana Daniela CRISTEA, [2.] Adela Diana BERDIE

[1.] NWCON Technology Consulting GmbH, GERMANY
[2.]University of Timisoara , Faculty of Engineering Hunedoara, ROMANIA

ABSTRACT
Application Server ABAP is an integrated part of the application platform within SAP NetWeaver. In this paper we will present the tools and concepts we can use to develop solid ABAP based applications that deal with the error and problem situations occurring in our programs. We catch exceptions and describe it for the end user. We use client-side validation and server-side validation to inform the user about the program status and exceptions occur. ABAP offers us full support to do that through exception classes, message classes, assistance classes, a special Hook method in Web Dynpro ABAP, a special control structure for catching exceptions and more.

1. INTRODUCTION

SAP NetWeaver is an infrastructure software that supports the integration and development of heterogeneous system landscapes as they are typically found in companies today [1].

Application platform of the SAP NetWeaver integration platform has two stacks: ABAP stack and Java stack or Application Server ABAP and Application Server Java with two programming interfaces ABAP and respective Java.

Application Server ABAP (AS ABAP) offers us through ABAP language many possibilities to handle the exceptions that occurs in our applications, offers us tools to develop robust programs. A good user interface catches exceptions and describes for the user the errors that occur.

One of the key quality criteria for software is the robustness of a program. It should be able to deal with the situation and should not crash [2]. In ABAP we have classical exception handling and class based exception handling. The SAP documentation [3] recommends us to use class based exception handling where every exception class derives directly or indirectly from the CX_ROOT super-class.

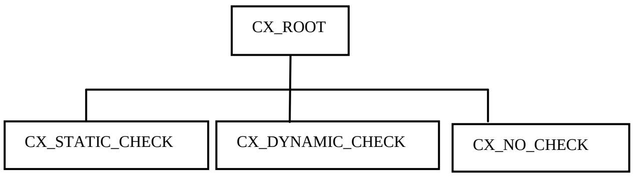
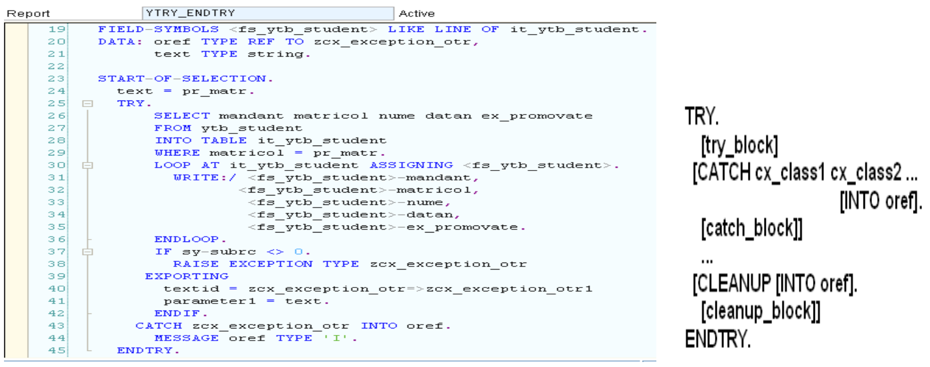


Fig. 1 The tree of exception classes in ABAP

To structure possible exceptions we have three abstract exception classes that derive from CX_ROOT super-class. Fig. 1 shows the exception classes relationship [4].

We can handle an exception in an ABAP programme with TRY - ENDTRY control structure. Fig. 2 shows an example of catching and handling exceptions.



```
FIELD-SYMBOLS <fs_ytb_student> LIKE LINE OF it_ytb_student.
DATA: oref TYPE REF TO zcx_exception_otr,
      text TYPE string.

START-OF-SELECTION.
  text = pr_matr.
  TRY.
      SELECT mandant matricol nume datan ex_promovate
      FROM ytb_student
      INTO TABLE it_ytb_student
      WHERE matricol = pr_matr.
      LOOP AT it_ytb_student ASSIGNING <fs_ytb_student>.
        WRITE:/ <fs_ytb_student>-mandant,
               <fs_ytb_student>-matricol,
                <fs_ytb_student>-nume,
                <fs_ytb_student>-datan,
                <fs_ytb_student>-ex_promovate.
      ENDLOOP.
      IF sy-subrc <> 0.
        RAISE EXCEPTION TYPE zcx_exception_otr
     EXPORTING
       textid = zcx_exception_otr=>zcx_exception_otr1
       parameter1 = text.
      ENDIF.
    CATCH zcx_exception_otr INTO oref.
      MESSAGE oref TYPE 'I'.
  ENDTRY.
```

```
TRY.
  [try_block]
 [CATCH cx_class1 cx_class2 ...
                  [INTO oref].
  [catch_block]]
  ...
 [CLEANUP [INTO oref].
  [cleanup_block]]
ENDTRY.
```

Fig. 2 TRY-ENDTRY structure example

The messages that we can use in ABAP language are "A" termination message, "E" error message, "I" information message, "S" status message, "W" warning message ," X " exit message and define how the ABAP runtime should process the message [5].

When an exception is raised using RAISE EXCEPTION, the runtime environment searches for a handler. Once a handler is found, the control flow processes the code of the handler before continuing. If no handler can be found, the program ends with a runtime error.

We can use catch CX_ROOT to catch all errors that occurs.

2. THE STUDY

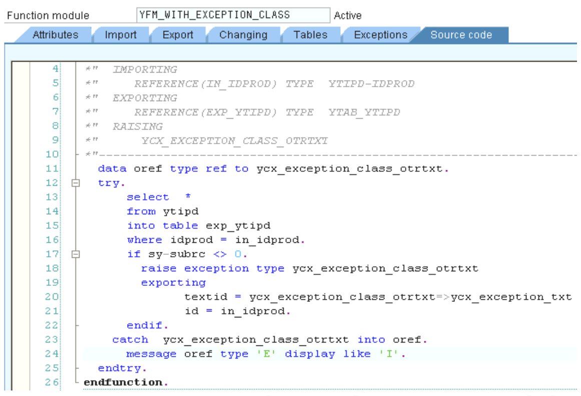
2.1 WORKING WITH EXCEPTION CLASSES AND TEXTS FOR XCEPTIONS

Each exception has assigned a text that describes the exception and can be for example Online Text Repository (OTR) text or text from a message class.

OTR is a central place for texts that offers us tools for its processing and administration. The texts that are stored here can contain maximum 255 characters [6].

We can create our own exception classes as global classes with Class Builder or as local classes in our programs. Some of the advantages of using class-based exception handling are [7]:

- Object-oriented concept of inheritance
- Exception classes have integration with ABAP message concept
- Exception class can hold many different types of exceptions



```
*"  IMPORTING
*"     REFERENCE(IN_IDPROD) TYPE  YTIPD-IDPROD
*"  EXPORTING
*"     REFERENCE(EXP_YTIPD) TYPE  YTAB_YTIPD
*"  RAISING
*"      YCX_EXCEPTION_CLASS_OTRTXT
*"----------------------------------------------------------------------
  data oref type ref to ycx_exception_class_otrtxt.
  try.
      select  *
      from ytipd
      into table exp_ytipd
      where idprod = in_idprod.
      if sy-subrc <> 0.
        raise exception type ycx_exception_class_otrtxt
        exporting
              textid = ycx_exception_class_otrtxt=>ycx_exception_txt
              id = in_idprod.
      endif.
    catch  ycx_exception_class_otrtxt into oref.
      message oref type 'E' display like 'I'.
  endtry.
endfunction.
```

Fig. 3 Exception class and Function Module

A. Exception class with OTR text

When we create an exception class in ABAP Workbench we can chose if this is with message class or without message class.

This type of classes have attributes inherits from the CX_ROOT super-class. From this attributes we use very often the TEXTID attribute and PREVIEUS attribute.

Fig. 3 shows a Function Module that access data from a database table and use an user defined exception class. When the user enters an id that doesn't exist in the database table, an eception of type YCX_EXCEPTION_CLASS_OTRTXT is raised and a proper error message is shown to the end user.

Fig. 4 shows the new defined exception id and attribute for this exception class. In the Text tab we can find the exception id YCX_ECEPTION_TXT that we have used in our Function Module.

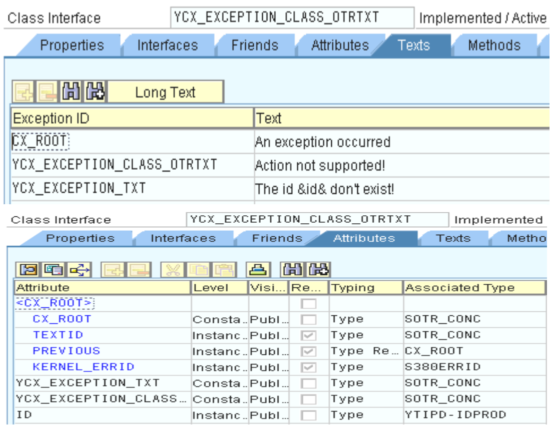
B. Exception class with text from a message class

We create an exception class YCX_EXCEPTION_CLASS_MSGTXT with text from a message class. In this case exception class implements the interface IF_T100_MESSAGE and we can use exception texts from database Table T00.

Fig. 5 shows the defined exception class.

Before we can use this class we have to create a Message Class and assign it to our exception class. In Fig. 6 we presented the Message Class YMSG_CLASS, created with Message Maintenance SE91.

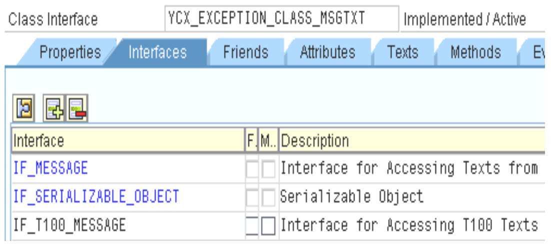
Messages are stored in the database Table T100 that has colums as: message number, short text, language key. Fig. 7 shows the structure of this table.



Exception ID	Text
CX_ROOT	An exception occurred
YCX_EXCEPTION_CLASS_OTRTXT	Action not supported!
YCX_EXCEPTION_TXT	The id &id& don't exist!

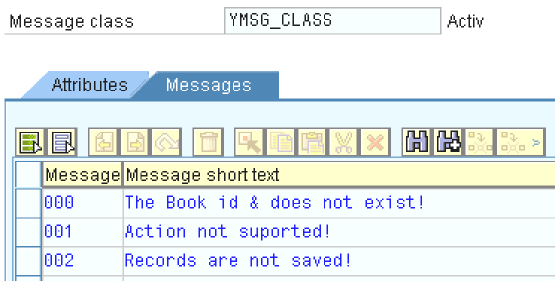
Attribute	Level	Visi...	Re...	Typing	Associated Type
<CX_ROOT>			☐		
CX_ROOT	Consta..	Publ...	☐	Type	SOTR_CONC
TEXTID	Instanc..	Publ...	✓	Type	SOTR_CONC
PREVIOUS	Instanc..	Publ...	✓	Type Re...	CX_ROOT
KERNEL_ERRID	Instanc..	Publ...	✓	Type	S380ERRID
YCX_EXCEPTION_TXT	Consta..	Publ...	☐	Type	SOTR_CONC
YCX_EXCEPTION_CLASS...	Consta..	Publ...	☐	Type	SOTR_CONC
ID	Instanc..	Publ...	☐	Type	YTIPD-IDPROD

Fig. 4 The structure of YCX_EXCEPTION_CLASS_OTRTXT



Interface	F	M..	Description
IF_MESSAGE	☐	☐	Interface for Accessing Texts from
IF_SERIALIZABLE_OBJECT	☐	☐	Serializable Object
IF_T100_MESSAGE	☐	☐	Interface for Accessing T100 Texts

Fig. 5 Exception class with message class



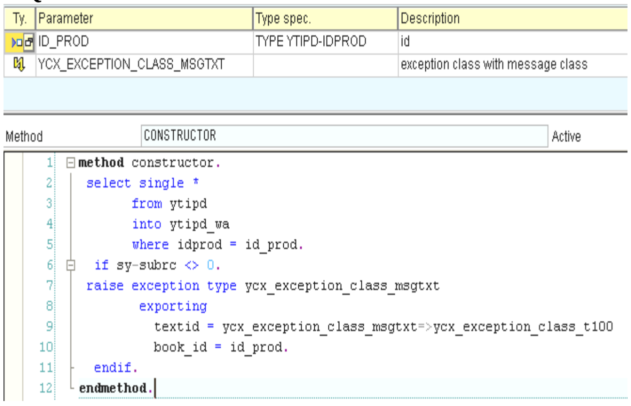
Message class YMSG_CLASS Activ

Message	Message short text
000	The Book id & does not exist!
001	Action not suported!
002	Records are not saved!

Fig. 6 Message Class YMSG_CLASS

Each exception id can be mapped to a message id from our message class. In Fig. 8 is shown a mapping example. As can be seen when we use text from a message class we have a restriction to maximum four placeholders. We can assign maximum four attributes from the exception class.

Fig. 9 illustrates the way we can use our exception class into an ABAP class that selects data from a database table through the help of SQL statements.

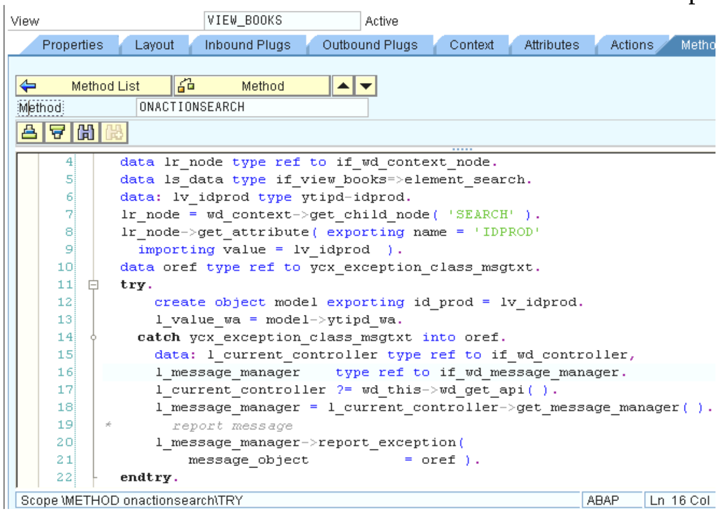


Ty.	Parameter	Type spec.	Description
	ID_PROD	TYPE YTIPD-IDPROD	id
	YCX_EXCEPTION_CLASS_MSGTXT		exception class with message class

Method CONSTRUCTOR Active

```
method constructor.
  select single *
        from ytipd
        into ytipd_wa
        where idprod = id_prod.
   if sy-subrc <> 0.
  raise exception type ycx_exception_class_msgtxt
         exporting
           textid = ycx_exception_class_msgtxt=>ycx_exception_class_t100
           book_id = id_prod.
   endif.
endmethod.
```

Fig. 9 ABAP class and exception class



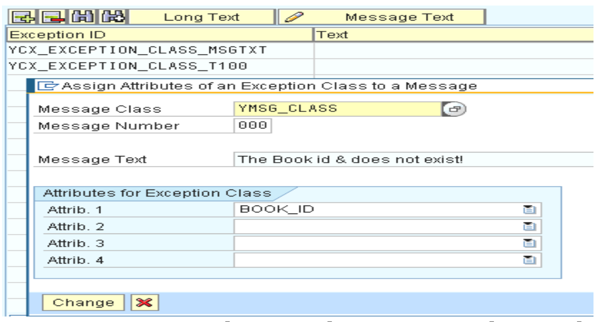
View VIEW_BOOKS Active

Method ONACTIONSEARCH

```
data lr_node type ref to if_wd_context_node.
data ls_data type if_view_books=>element_search.
data: lv_idprod type ytipd-idprod.
lr_node = wd_context->get_child_node( 'SEARCH' ).
lr_node->get_attribute( exporting name = 'IDPROD'
  importing value = lv_idprod  ).
data oref type ref to ycx_exception_class_msgtxt.
try.
    create object model exporting id_prod = lv_idprod.
    l_value_wa = model->ytipd_wa.
  catch ycx_exception_class_msgtxt into oref.
    data: l_current_controller type ref to if_wd_controller,
    l_message_manager     type ref to if_wd_message_manager.
    l_current_controller ?= wd_this->wd_get_api( ).
    l_message_manager = l_current_controller->get_message_manager( ).
*     report message
    l_message_manager->report_exception(
        message_object            = oref ).
endtry.
```

Scope \METHOD onactionsearch\TRY ABAP Ln 16 Col

Fig. 10 Catch and show of the exception message



Exception ID	Text
YCX_EXCEPTION_CLASS_MSGTXT	
YCX_EXCEPTION_CLASS_T100	

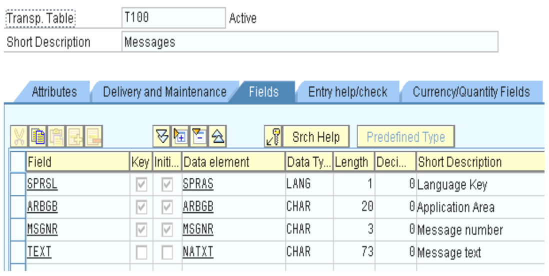
Assign Attributes of an Exception Class to a Message

Message Class	YMSG_CLASS
Message Number	000
Message Text	The Book id & does not exist!

Attributes for Exception Class

Attrib. 1	BOOK_ID
Attrib. 2	
Attrib. 3	
Attrib. 4	

Fig. 8 Exception id mapped to message class and message number



Transp. Table T100 Active
Short Description Messages

Field	Key	Initi...	Data element	Data Ty...	Length	Deci...	Short Description
SPRSL	✓	✓	SPRAS	LANG	1	0	Language Key
ARBGB	✓	✓	ARBGB	CHAR	20	0	Application Area
MSGNR	✓	✓	MSGNR	CHAR	3	0	Message number
TEXT			NATXT	CHAR	73	0	Message text

Fig. 7 The structure of T100 database table

2.2. MESSAGES, EXCEPTIONS AND WEB DYNPRO ABAP

Web Dynpro ABAP is the SAP Framework that uses Model View Controller MVC paradigm in order to build reusable multi-component web business applications.

Through a What You See Is What You Get view editor we can simply drag and drop the UI Elements that we need and we have fully support to work with messages [8].

The presentation of messages in the client is controlled by Web Dynpro Framework and Hook method wddobeforeaction( ) can us help to react to user inputs [9].

A. Exception class and Web Dynpro ABAP

Web Dynpro ABAP offers us support to work with exception classes through methods of the Message Manager.

In Fig. 10 we present the way we can use in a Web Dynpro application, the class defined hereinbefore. What is more important is the fact that we can catch the proper exception and show it in browser in a MessageArea UI Element.

All the messages that are shown with Web Dynpro ABAP are displayed as default at the begin of the screen. In our case we want to change this position and in this purpose we use a MessageArea UI Element. In fig. 11 we show the proper User Interface.

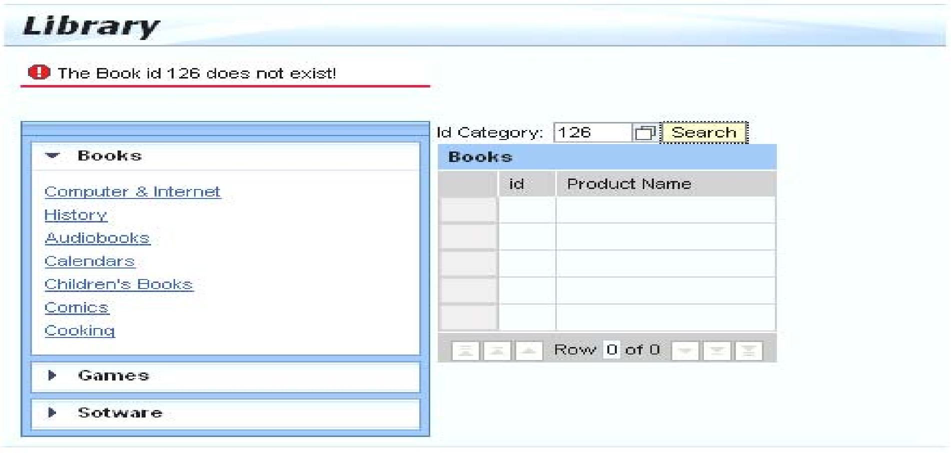


Fig. 11 The User Interface with Web Dynpro ABAP

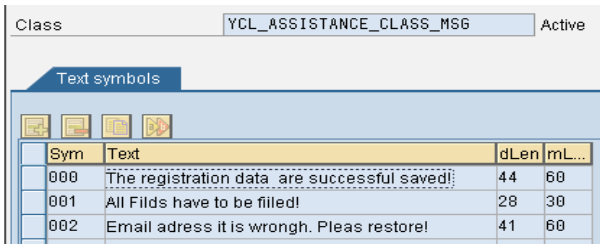
B. Assistance class and Web Dynpro ABAP

An assistance class is a regular ABAP class that inherits the CL_WD_COMPONENT_ASSISTANCE. Every Web Dynpro component has assigned just an assistance class that we can use as Model or to work with text symbols.

The class CL_WD_COMPONENT_ASSISTANCE provides central functions by which a Web Dynpro component can access text symbols of the assistance class [10]. As advantages of using assistance class we can specify [11]:

- Method calls of the assistance class have more performance as calls of methods of a Web Dynpro controller.
- Manage dynamic texts

In Fig. 12 we present the text symbols that we have defined for a created assistance class YCL_ASSISTANCE_CLASS_MSG.


Class YCL_ASSISTANCE_CLASS_MSG Active

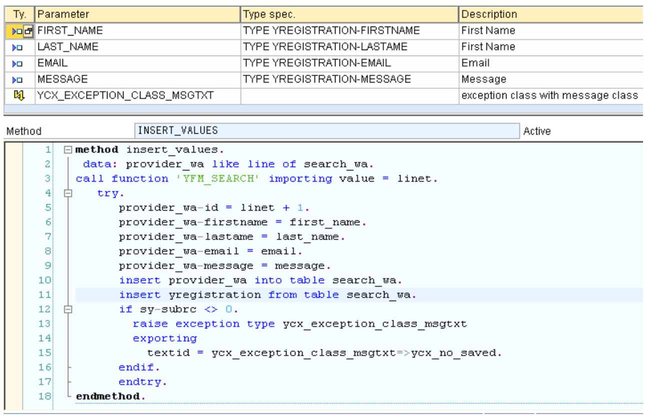
Text symbols

Sym	Text	dLen	mL...
000	The registration data are successful saved!	44	60
001	All Filds have to be filled!	28	30
002	Email adress it is wrongh. Pleas restore!	41	60

Fig. 12 Text symbols of assistance class

In our assistance class we define a new method INSERT_VALUES with which we can insert in the database table the informations that the user enter in the registration form. Fig. 13 shows the method codding.

To show a message to the user when he doesn't enter a proper value or to inform him that the data are successful saved (client-side validation) we use text symbols defined in our assistance class. When an exception occurs we use the exception class. Fig. 14 illustrates the way we can use the assistance class as model in our Web Dynpro application and the way we can catch the exceptions.

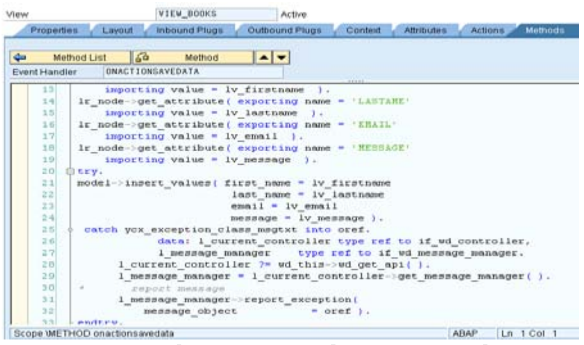


Ty.	Parameter	Type spec.	Description
	FIRST_NAME	TYPE YREGISTRATION-FIRSTNAME	First Name
	LAST_NAME	TYPE YREGISTRATION-LASTAME	First Name
	EMAIL	TYPE YREGISTRATION-EMAIL	Email
	MESSAGE	TYPE YREGISTRATION-MESSAGE	Message
	YCX_EXCEPTION_CLASS_MSGTXT		exception class with message class

Method INSERT_VALUES Active

```
method insert_values.
 data: provider_wa like line of search_wa.
call function 'YFM_SEARCH' importing value = linet.
   try.
      provider_wa-id = linet + 1.
      provider_wa-firstname = first_name.
      provider_wa-lastame = last_name.
      provider_wa-email = email.
      provider_wa-message = message.
      insert provider_wa into table search_wa.
      insert yregistration from table search_wa.
      if sy-subrc <> 0.
        raise exception type ycx_exception_class_msgtxt
        exporting
          textid = ycx_exception_class_msgtxt=>ycx_no_saved.
      endif.
      endtry.
endmethod.
```

Fig. 13 Method of our assistance class

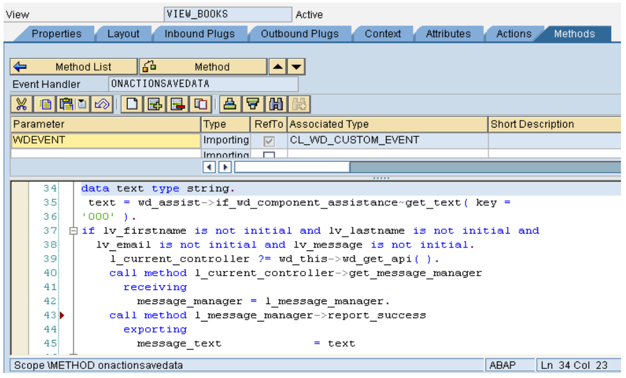

View VIEW_BOOKS Active

Event Handler ONACTIONSAVEDATA

```
       importing value = lv_firstname  ).
  lr_node->get_attribute( exporting name = 'LASTAME'
       importing value = lv_lastname  ).
  lr_node->get_attribute( exporting name = 'EMAIL'
       importing value = lv_email  ).
  lr_node->get_attribute( exporting name = 'MESSAGE'
       importing value = lv_message  ).
 try.
 model->insert_values( first_name = lv_firstname
                       last_name = lv_lastname
                       email = lv_email
                       message = lv_message ).
  catch ycx_exception_class_msgtxt into oref.
            data: l_current_controller type ref to if_wd_controller,
                  l_message_manager    type ref to if_wd_message_manager.
        l_current_controller ?= wd_this->wd_get_api( ).
        l_message_manager = l_current_controller->get_message_manager( ).
 *       report message
        l_message_manager->report_exception(
            message_object          = oref ).
 endtry.
```

Scope \METHOD onactionsavedata ABAP Ln 1 Col 1

Fig. 14 Web Dynpro and assistance class


View VIEW_BOOKS Active

Event Handler ONACTIONSAVEDATA

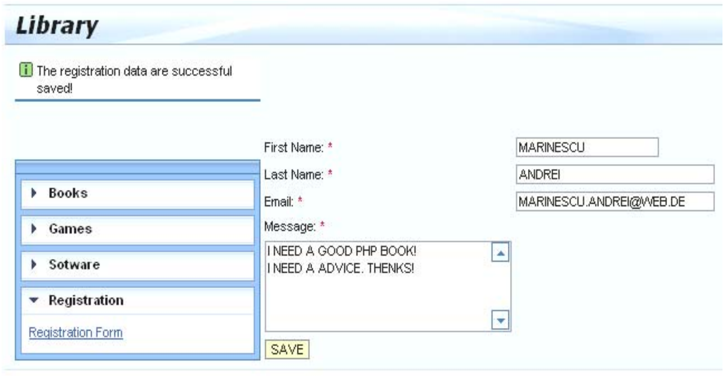
Parameter	Type	RefTo	Associated Type	Short Description
WDEVENT	Importing	✓	CL_WD_CUSTOM_EVENT	
	Importing			

```
data text type string.
 text = wd_assist->if_wd_component_assistance~get_text( key =
'000' ).
if lv_firstname is not initial and lv_lastname is not initial and
  lv_email is not initial and lv_message is not initial.
    l_current_controller ?= wd_this->wd_get_api( ).
    call method l_current_controller->get_message_manager
      receiving
        message_manager = l_message_manager.
    call method l_message_manager->report_success
      exporting
        message_text              = text
```

Scope \METHOD onactionsavedata ABAP Ln 34 Col 23

Fig. 15 Access of text symbols from assistance class

Through the attribute WD_ASSIST and the method WD_COMPONENT_ASSISTANCE ~GET_ TEXT( ) we can access text symbols of the assistance class from our component controller, Fig. 15.



The registration data are successful saved!

First Name: *	MARINESCU
Last Name: *	ANDREI
Email: *	MARINESCU.ANDREI@WEB.DE
Message: *	I NEED A GOOD PHP BOOK! I NEED A ADVICE. THENKS!

Fig. 16 User Interface with Web Dynpro ABAP

We have to specify that for client-side validation we can use even messages that are defined directly in the message class but is not recommended to involve the message texts directly in coding. In Fig. 16 we present the corresponding User Interface in Web Dynpro ABAP.

3. CONCLUSIONS

In this paper we have presented some of the concepts to develop robust software by ABAP programming language. When we develop an application we have to plan a good exception and messages handling to describe in detail the error situation, to check the data input from the user and to show the program status. We have seen the advantages of using the new class based exception concept of the ABAP language and some of the tools that Web Dynpro ABAP gives us in order to simplify the message and error handling.

REFERENCES

[1] *L. Heilig, S. Karch, O. Brottcher, C. Mutzig, J. Weber, R. Pfenning, SAP NetWeaver: The official Guide, Galileo Press 2008*
[2] *Horst Keller, Sascha Kruger, ABAP Objects, Galileo press 2007*
[3] *http://help.sap.com/saphelp_nw04s/helpdata/en/a0/ff934258a5c76ae10000000a155106/frameset.htm*
[4] *Horst Keller, The official ABAP reference, Galileo Press, 2005*
[5] *http://help.sap.com/saphelp_nw04s/helpdata/en/5f/8b2e16880111d194cb0000e8353423/frameset.htm*
[6] *http://help.sap.com/saphelp_nw04/helpdata/en/4a/fff13a62d1ad6de 100000 00a11405a/frameset.htm*
[7] *Rich Heilman, Thomas Jung, Next Generation ABAP Development, Galileo Press 2007*
[8] *The 14th international conference, The knowledge based organization, November 2008, Cristea Ana Daniela, Adela Diana Berdie, Osaci Mihaela, User Interfaces with Web Dynpro ABAP and Web Dynpro Java, "Nicolae Balcescu" land Forces Academy publishing Haus Sibiu, 2008.*
[9] *Ulli Hoffmann, Web Dynpro for ABAP, Galileo Press 2007*
[10] *http://help.sap.com/saphelp_nw04s/helpdata/en/21/ad884118aa1709e10000000a155106/frameset.htm*
[11] *Karl-Heinz Kühnhauser, Discover ABAP, Galileo Press, 2008*