**Exception handling**

# *from Обработка особых ситуаций в ABAP*

<https://abap-blog.ru/osnovy-abap/obrabotka-osobyx-situacij-v-abap/>

В RFC модулях в настоящее время используется классический способ обработки исключений. Не допускается одновременно использовать классический и основанный на классах, способы обработки исключений /в интерфейсе методов, процедур, функций/.

## **Классический способ обработки исключений**

При вызове исключения, системное поле ***sy-subrc*** будет заполнено номером, под которым исключение было обозначено при вызове ФМ, метода или процедуры

CALL FUNCTION ...

...

EXCEPTIONS

*Ошибка1 = 1*

*Ошибка2 = 2.*

Как правило, исключения вызываются с ***текстом*** сообщения - данный текст может быть описан статически – при вызове исключения оператором *MESSAGE*, либо динамически – путём получения текста из описания ФМ.

Так же исключение может быть вызвано без какого-либо текста /оператором *RAISE* Имя\_Исключения/, но данный способ лучше не использовать, т.к. вызов исключения должен как-то себя расшифровывать и говорить о том, что собственно произошло.

Напишем небольшой ФМ, рассчитывающий сумму двух чисел, оба параметра помечены как необязательные, *если первый параметр не будет задан при вызове ФМ*, система выдаст исключение – *no\_num\_1*.

FUNCTION **ZTEST\_EXC**.

  IF *i\_num\_1* IS NOT SUPPLIED.

     MESSAGE e398(00) WITH 'Число 1 не указано, расчёт невозможен' RAISING *no\_num\_1*.

  ENDIF.

  E\_SUMM = i\_num\_1 + i\_num\_2.

ENDFUNCTION.

И программа для его вызова

DATA:

  gv\_num\_2 TYPE i VALUE 10,

  gv\_summ  TYPE i.

CALL FUNCTION 'ZTEST\_EXC'

  EXPORTING  i\_num\_2  = gv\_num\_2    *" i\_num\_2  ← gv\_num\_2*

  IMPORTING   e\_summ   = gv\_summ     *" e\_summ  → gv\_summ*

  EXCEPTIONS

*no\_num\_1* = 1

    others   = 2.

IF *sy-subrc* <> 0.

MESSAGE ID *sy-msgid* TYPE sy-msgty NUMBER sy-msgno

            WITH sy-msgv1 sy-msgv2 sy-msgv3 sy-msgv4.

ENDIF.

WRITE gv\_summ.

При запуске программы произойдет вызов исключения, т.к. был использован тип сообщения *Е*, программа завершит свое выполнение после показа сообщения

[4](http://abap-blog.ru/wp-content/uploads/2013/08/4.png)

Замечу, что это вовсе не означает, что при вызове ФМ или метода и обработке исключения необходимо завершать работу программы, вы можете свободно продолжить её выполнение и далее, добавив например, сообщение об ошибке в лог программы, а не на вывод как в примере.

Ключевое слово ***OTHERS*** используется для того *чтобы поймать исключения не описанные в ФМ или явно неуказанные, при вызове ФМ*.

Пример вызова неописанного исключения

FUNCTION **ztest\_exc**.

  IF i\_num\_1 IS NOT SUPPLIED.

    MESSAGE e398(00) WITH 'Число 1 не указано, расчёт невозможен' RAISING *no\_num\_2*.

  ENDIF.

  E\_SUMM = i\_num\_1 + i\_num\_2.

ENDFUNCTION.

В данном примере вызывается неописанное в интерфейсе ФМ исключение – *no\_num\_2*, которое будет благополучно поймано с помощью ключевого слова OTHERS /системное поле sy-subrc примет значение равное 2 -   EXCEPTIONS no\_num\_1= 1 *others = 2* /.

В ФМ, могут быть добавлены новые исключения и в случае, когда при вызове ФМ они не обработаны и не указано слово OTHERS программа упадет в дамп с ошибкой времени выполнения - ***RAISE\_EXCEPTION*** ⇒ ключевое слово *OTHERS* подставляем всегда, при вызове ФМ /метода или процедуры/, когда мы точно не уверены в неизменности компонента.

*Сообщения, вызываемые в ФМ или методах, оператором MESSAGE*, без дополнения RAISING, либо сообщения вызываемые системой [например, при обработке экранов], могут быть обработаны программой с использованием дополнения ***error\_message*** = n\_error, указываемого так же после ключевого слова **EXCEPTIONS**.

При обработке сообщений

* Сообщения с типом I, W, S не обрабатываются, но записываются в журнал обработки фонового выполнения, если происходит обработка в фоне.
* Сообщения с типом E или A могут быть обработаны, при этом в поле *sy-subrc* будет записано значение *n\_error*. При вызове сообщения с типом А, происходит вызов ROLLBACK WORK.
* Сообщение с типом X не обрабатывается, программа завершается с дампом.

Пример ФМ

FUNCTION **ztest\_exc**.

  IF *i\_num\_1* IS NOT SUPPLIED.

    MESSAGE e398(00) WITH 'Число 1 не указано, расчёт невозможен'.

  ENDIF.

  E\_SUMM = i\_num\_1 + i\_num\_2.

ENDFUNCTION.

Программа

PROGRAM **test\_exceptions**.

DATA:

  gv\_val TYPE i VALUE 10,

  gv\_summ TYPE i.

CALL FUNCTION 'ZTEST\_EXC'

  EXPORTING i\_num\_2  = gv\_val

  IMPORTING  e\_summ   = gv\_summ

  EXCEPTIONS

    error\_message = 1

    others   = 2.

IF *sy-subrc* = 1.

  WRITE 'ФМ вызвал MESSAGE с типом E,A'.

ENDIF.

*Обработка исключения классическим способом может быть выполнена динамически*, с помощью ключевого слова *EXCEPTION-TABLE*.

Пример

DATA: line     TYPE c LENGTH 80,

      text\_tab LIKE STANDARD TABLE OF line,

      filename TYPE string,

       filetype TYPE c LENGTH 10,

      fleng    TYPE i.

DATA: func TYPE string,

      ptab TYPE abap\_func\_parmbind\_tab,

      ptab\_line TYPE abap\_func\_parmbind,

*etab* TYPE abap\_func\_excpbind\_tab,

*etab\_line* TYPE abap\_func\_excpbind.

func = 'GUI\_DOWNLOAD'.

filename = 'c:\temp\text.txt'.

filetype = 'ASC'.

ptab\_line-name = 'FILENAME'.

ptab\_line-kind = abap\_func\_exporting.

GET REFERENCE OF filename INTO ptab\_line-value.

INSERT ptab\_line INTO TABLE ptab.

ptab\_line-name = 'FILETYPE'.

ptab\_line-kind = abap\_func\_exporting.

GET REFERENCE OF filetype INTO ptab\_line-value.

INSERT ptab\_line INTO TABLE ptab.

ptab\_line-name = 'DATA\_TAB'.

ptab\_line-kind = abap\_func\_tables.

GET REFERENCE OF text\_tab INTO ptab\_line-value.

INSERT ptab\_line INTO TABLE ptab.

ptab\_line-name = 'FILELENGTH'.

ptab\_line-kind = abap\_func\_importing.

GET REFERENCE OF fleng INTO ptab\_line-value.

INSERT ptab\_line INTO TABLE ptab.

...

*etab\_line-name* = 'OTHERS'.

*etab\_line-value* = 10.

INSERT etab\_line INTO TABLE *etab*.

CALL FUNCTION func

  PARAMETER-TABLE ptab

*EXCEPTION-TABLE* etab.

CASE sy-subrc.

  WHEN 1.

    ...

    ...

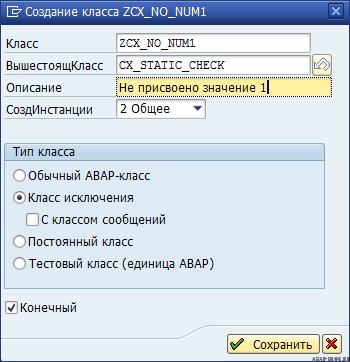
ENDCASE.

## **Обработка исключений, основанная на классах**

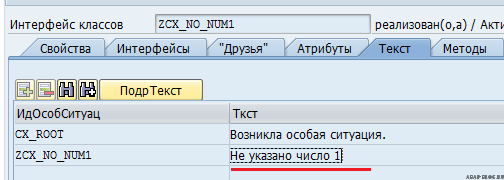
Как понятно из названия, под исключениями в данном случае понимаются объекты специальных классов исключений. Вызов такого исключения может быть выполнен либо в программе с помощью оператора *RAISE EXCEPTION*, либо системой /например, при делении на ноль будет вызвано предопределённое исключение CX\_SY\_ZERODIVIDE, [список](http://help.sap.com/abapdocu_740/en/abenabap_exception_classes.htm) таких исключений/, либо через дополнение *THROW* в условных выражениях /с версии ABAP 7.4/.

Во всех случаях инициируется создание объекта указанного класса /если указано дополнение INTO в CATCH/, в атрибутах которого содержится инф. о возникшей исключительной ситуации, доступ к ним, как правило, осуществляется через методы этого класса.

Классы особых ситуаций могут быть определены как локально, так и глобально через построитель классов – транзакция SE24, диалог создания

[](http://abap-blog.ru/wp-content/uploads/2013/08/10.png)

В данном случае галочка *С классом сообщений* означает *использование в качестве текста сообщения из класса сообщений* /транзакция **SE91**/. По умолчанию текст сообщения создается в текстах класса

[](http://abap-blog.ru/wp-content/uploads/2013/08/11.png)

### ***Категории исключений***

Все классы особых ситуаций являются производными одного из классов - CX\_NO\_CHECK, CX\_DYNAMIC\_CHECK или CX\_STATIC\_CHECK, которые сами являются производными общего суперкласса CX\_ROOT.

* *CX\_STATIC\_CHECK* – исключения которые вызываются в процедуре /ФМ или методе/, должны быть либо обработаны в ней, либо процедура должна иметь соответствующий интерфейс, чтобы вызывающий её код мог обработать эту ситуацию.

Если исключение определено как потомок этого класса, оно должно быть явно указано в интерфейсе метода /ФМ или формы/ в котором происходит его вызов.

Если при статической проверке, система не увидит обработки в блоке TRY..CATCH..ENDTRY подобного исключения система выдаст предупреждение.

* *CX\_DYNAMIC\_CHECK* – при проверке кода, компилятор не будет делать предупреждений об отсутствии обработки исключений их этой категории, в случае вызова исключения его обработка будет проверена динамически и если обработчик не будет найден программа упадет в дамп.

Обычно данная категория используется тогда, когда исключение может быть обработано внутри самого метода, без передачи обработки выше по стеку.

* *CX\_NO\_CHECK* – аналогичны предыдущему типу, но данную категорию нельзя объявлять в интерфейсах.

Данную категорию следует использовать для исключительных ситуаций, которые могут произойти в любое время и не могут быть обработаны непосредственно в коде метода. Кроме того, можно использовать в случаях когда одна и та же исключительная ситуация может возникнуть во множествах методов, а объявлять её в интерфейсах каждого из методов не имеет смысла, т.к. это усложнит код.

*На исключения накладываются следующие ограничения*

* Исключение не может быть объявлено в интерфейсе *статического конструктора*.
* Исключение не может быть объявлено в интерфейсе *обработчика событий*. При этом если в коде обработчика произошел вызов исключения, и он не был обработан, система вызовет исключение - CX\_SY\_NO\_HANDLER, которое может быть обработано в вызывающем его коде.
* При вызове программ через SUMBIT или CALL TRANSACTION, исключение, возникающее в вызываемой программе, не может быть передано в вызывающую программу.

Небольшой пример с локальным классом исключения

PROGRAM **test\_exceptions**.

CLASS **lcx\_no\_num** DEFINITION INHERITING FROM *cx\_static\_check*.

ENDCLASS.

CLASS **lcl\_test\_exceptions** *DEFINITION*.

  PUBLIC SECTION.

    METHODS:

***do\_summ*** IMPORTING *i\_num\_1* TYPE i OPTIONAL *i\_num\_2* TYPE i OPTIONAL

              RETURNING value(*re\_summ*) TYPE int1

              RAISING *lcx\_no\_num*.

ENDCLASS.

CLASS **lcl\_test\_exceptions** *IMPLEMENTATION*.

  METHOD ***do\_summ***.

    IF *i\_num\_1* IS NOT SUPPLIED OR *i\_num\_2* IS NOT SUPPLIED.

*" Данное исключение присутствует в интерфейсе ⇒ может быть обработано вне метода*

      RAISE EXCEPTION TYPE *lcx\_no\_num*.

    ENDIF.

    TRY.

      re\_summ = i\_num\_1 + i\_num\_2.

*" Ошибка с дин. проверкой - при её обработке обнулим результат*

    CATCH *CX\_SY\_CONVERSION\_OVERFLOW*.

      re\_summ = 0.

    ENDTRY.

  ENDMETHOD.

ENDCLASS.

DATA:

  go\_test\_exceptions TYPE REF TO lcl\_test\_exceptions,

  gv\_summ TYPE int1.

START-OF-SELECTION.

  CREATE OBJECT go\_test\_exceptions.

  TRY.

    go\_test\_exceptions->*do\_summ*( EXPORTING *i\_num\_2* = 1 RECEIVING *re\_summ* = gv\_summ ).

  CATCH *lcx\_no\_num*.

    WRITE 'Не заполнены все параметры'.

  ENDTRY.

  go\_test\_exceptions->*do\_summ*( EXPORTING *i\_num\_1* = 999 *i\_num\_2* = 1

RECEIVING *re\_summ* = gv\_summ  ).

  WRITE: / 'Результат cуммы 999 и 1:', gv\_summ.

В стандартной системе SAP имена всех классов особых ситуаций начинаются с CX\_ , пользовательские исключения рекомендуется называть, начиная с ZCX или lcx для локальных классов исключений.

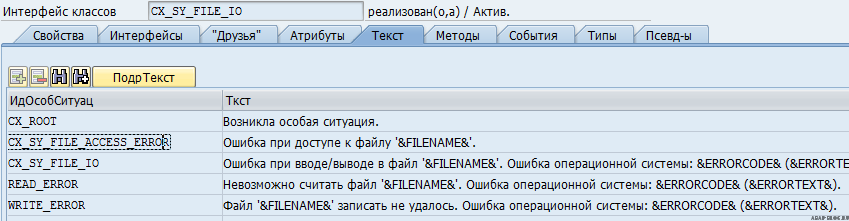
Класс ***CX\_ROOT*** предоставляет некоторые предопределенные методы, которые наследуются всеми классами особых ситуаций

* Метод *GET\_SOURCE\_POSITION* возвращает имя главной программы и /если связаны/ имена включенных программ /инклудов/ и номер строки исходного кода, в которой возникла особая ситуация.
* Метод *GET\_TEXT* возвращает текст особой ситуации в форме строки.
* Метод *GET\_LONGTEXT* возвращает подробный текст текста особой ситуации в форме строки.

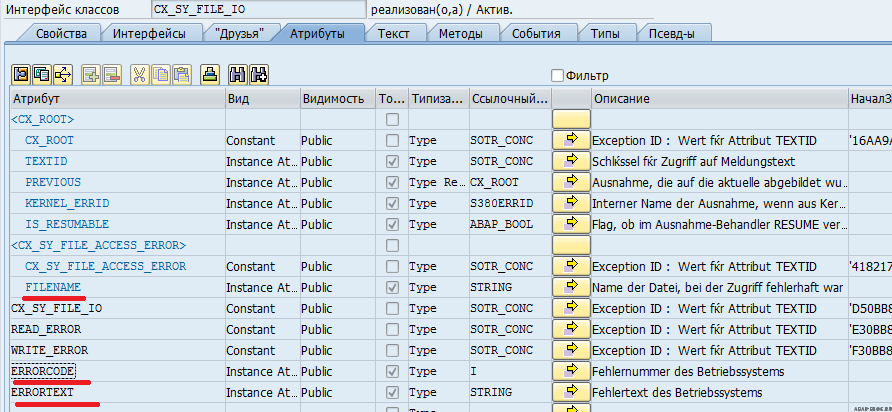
### ***Тексты исключений***

Каждому *классу* можно присвоить несколько *текстов*. Присвоенные им идентификаторы создаются построителем классов как константы в атрибутах класса. *Тексты* сохраняются в репозитарии текстов - OTR. Константы идентификаторы, представленные в шестнадцатеричном формате, уникальны на уровне системы. Доступ к хранилищу текстов можно получить через транзакцию ***SOTR\_EDIT***.

В текстах можно определить параметры, их необходимо обозначить в амперсандах. В качестве примера, можно рассмотреть текст из класса исключения *CX\_SY\_FILE\_IO*

[](http://abap-blog.ru/wp-content/uploads/2013/08/15.png)

В параметры будут переданы /при вызове метода GET\_TEXT/ соответствующие им атрибуты класса

[](http://abap-blog.ru/wp-content/uploads/2013/08/16.png)

Заполняются эти атрибуты в конструкторе при вызове исключения

DATA: lr\_ex  TYPE REF TO cx\_sy\_file\_io,

      lv\_msg TYPE string.

TRY.

   ...

   RAISE EXCPETION TYPE cx\_sy\_file\_io

      EXPORTING

*textid* = cx\_sy\_file\_io=>read\_error

*filename* = 'somefile.txt'.

CATCH cx\_sy\_file\_io INTO lr\_ex.

   lv\_msg = lr\_ex->get\_text( ).

   MESSAGE lv\_msg TYPE 'I'.

ENDTRY.

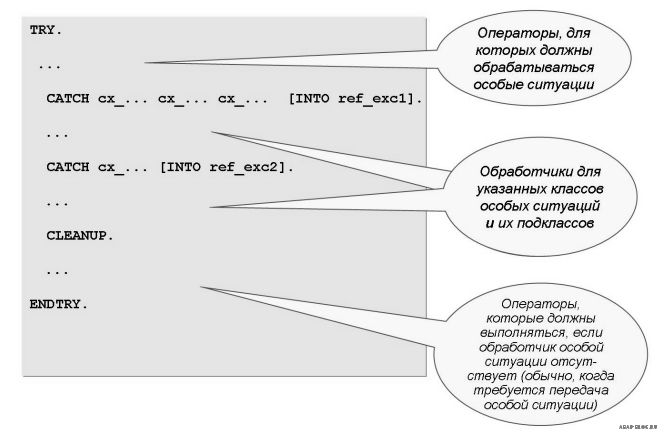
Так же в конструкторе можно указать, какой текст должен использоваться при инициировании особой ситуации, передав одну из определенных констант в параметр импорта **TEXTID**. Не рекомендуется использовать подобную методику, т.к. это может запутать код, однако как было уже показано выше SAP сам это использует (read\_error, write\_error в **CX\_SY\_FILE\_IO**). Инкапсуляция текстов в классах сообщений и их саморасшифровываемость является одним из преимуществ над классическими исключениями.

Конструктор, который генерируется автоматически в **SE24**, для нового созданного исключения (ZCX\_NO\_NUM1), выглядит так:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | CALL METHOD SUPER->CONSTRUCTOR  EXPORTING  TEXTID = TEXTID  PREVIOUS = PREVIOUS  .  IF textid IS INITIAL.     me->textid = ZCX\_NO\_NUM1 .  ENDIF. |

### Блок обработки исключений

Особая ситуация может быть обработана, только если оператор, который может инициировать ее, заключен в блок TRY-ENDTRY. Затем особая ситуация обрабатывается с помощью оператора CATCH в блоке TRY-ENDTRY.

[](http://abap-blog.ru/wp-content/uploads/2013/08/17.png)

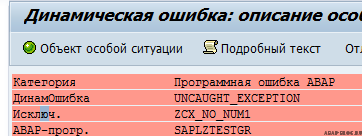
Блок TRY содержит набор операторов, обрабатывающих особые ситуации. Если в блоке TRY появляется особая ситуация, система осуществляет поиск первого оператора CATCH в том же блоке TRY-ENDTRY, а затем последовательно снаружи во всех заключающих блоках TRY-ENDTRY, обрабатывающих особую ситуацию. Если оператор находится, система осуществляет переход к его обработчику. Если обработчик найти не удается, но блок TRY-ENDTRY находится в процедуре, система осуществляет попытку передачи особой ситуации вызывающей программе.

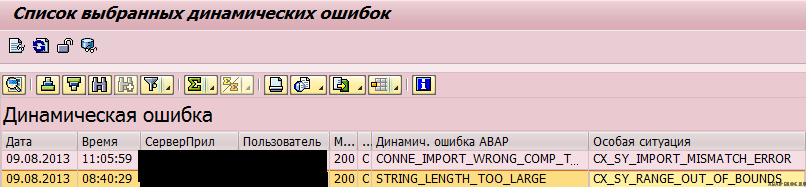
Блок CATCH содержит обработчик особых ситуаций, исполняемый при возникновении указанной особой ситуации в связанном блоке TRY. Для оператора CATCH можно указать любое количество классов особых ситуаций. Таким образом, определяется обработчик особых ситуаций для всех этих классов особых ситуаций и их подклассов.

Блоки TRY-ENDTRY могут иметь вложенность любой глубины. Поэтому блок TRY, блоки CATCH и блок CLEANUP в целом сами могут содержать полные блоки TRY-ENDTRY.

При возникновении особой ситуации система осуществляет поиск по перечисленным обработчикам особых ситуаций в указанном порядке. Затем она исполняет первый обработчик особых ситуаций, оператор CATCH которого содержит подходящий класс особой ситуации или один из ее суперклассов.

Если ошибка не будет обработана и не будет передана вызывающей программе, система выдаст дамп с ошибкой времени выполнения — **UNCAUGHT\_EXCEPTION**, в том случае когда не обрабатывается исключительная ситуация, связанная с ошибкой времени выполнения, система выдает дамп с ошибкой времени выполнения (например, CX\_SY\_CONVERSION\_CODEPAGE — CONVT\_CODEPAGE):

[](http://abap-blog.ru/wp-content/uploads/2013/08/18.png)

Просмотр ошибки в транзакции ST22:[](http://abap-blog.ru/wp-content/uploads/2013/08/Безымянный.png)

### Распространение особых ситуаций

Если возникает особая ситуация (наследуемая от CX\_DYNAMIC\_CHECK, CX\_STATIC\_CHECK), она автоматически распространяется на все уровни стека вызовов, до тех пор, пока она не будет обработана или пока не встретится такой интерфейс, в котором она (либо её предки) будет отсутствовать.

Следующий пример демонстрирует распространение особой ситуации на несколько методов:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39 | CLASS lcx\_calc\_error DEFINITION INHERITING FROM cx\_static\_check.  ENDCLASS.    CLASS lcx\_summ\_error DEFINITION INHERITING FROM lcx\_calc\_error.  ENDCLASS.    CLASS lcl\_test\_exc DEFINITION.    PUBLIC SECTION.    METHODS:      do\_calc        RAISING          lcx\_calc\_error,      do\_summ        RAISING          lcx\_summ\_error.  ENDCLASS.    CLASS lcl\_test\_exc IMPLEMENTATION.      METHOD do\_calc.      do\_summ( ).    ENDMETHOD.      METHOD do\_summ.      RAISE EXCEPTION TYPE lcx\_summ\_error.    ENDMETHOD.    ENDCLASS.    DATA:    go\_test TYPE REF TO lcl\_test\_exc.    START-OF-SELECTION.    CREATE OBJECT go\_test.    TRY.      go\_test->do\_calc( ).    CATCH lcx\_calc\_error.      WRITE 'Catched'.    ENDTRY. |

Обратите внимание на метод do\_calc, в нем описана особая ситуация от которой наследуется lcx\_summ\_error, соответственно прерывание продолжится на следующий уровень и будет обработано в блоке TRY..CATCH..ENDTRY. При правильно выстроенной архитектуре наследования исключительных ситуаций, прозрачность кода заметно повышается.

В случае, когда используется исключение, наследуемое от CX\_NO\_CHECK, описание его в интерфейсе метода может быть опущено, т.к. оно инициируется неявным способом системой:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35 | CLASS lcx\_calc\_error DEFINITION INHERITING FROM cx\_no\_check.  ENDCLASS.    CLASS lcx\_summ\_error DEFINITION INHERITING FROM lcx\_calc\_error.  ENDCLASS.    CLASS lcl\_test\_exc DEFINITION.    PUBLIC SECTION.    METHODS:      do\_calc,      do\_summ.  ENDCLASS.    CLASS lcl\_test\_exc IMPLEMENTATION.      METHOD do\_calc.      do\_summ( ).    ENDMETHOD.      METHOD do\_summ.      RAISE EXCEPTION TYPE lcx\_summ\_error.    ENDMETHOD.    ENDCLASS.    DATA:    go\_test TYPE REF TO lcl\_test\_exc.    START-OF-SELECTION.    CREATE OBJECT go\_test.    TRY.      go\_test->do\_calc( ).    CATCH lcx\_calc\_error.      WRITE 'Catched'.    ENDTRY. |

### Очистка после вызовов исключений

Блок CLEANUP исполняется, когда выполнен выход из блока TRY-ENDTRY, так как система не смогла найти обработчик для исключения в определенном блоке TRY-ENDTRY, но особая ситуация обрабатывается в окружающем блоке TRY-ENDTRY или передается вызывающей программе.

Данный блок обычно применяется для освобождения занятых ресурсов: очистке ссылочных переменных, закрытие локаторов или наборов данных (datasets) и т.п. Допустим, Вы записываете некоторые данные в файл на сервере приложений. Внутри блока TRY Вы открываете набор данных (dataset) и начинаете запись в него. Однако в некоторый момент, случается особая ситуация, которую вы не обработали и блок TRY прерывает свою работу, при этом, не выполнив закрытие набора данных. Для того чтобы избежать подобной ситуации воспользуемся ключевым словом CLEANUP:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | TRY.  \*  Открываем файл на запись:     OPEN DATASET lv\_file FOR OUTPUT IN TEXT MODE                          ENCODING DEFAULT.    \*  Переносим данные в файл:     LOOP AT lt\_extract INTO ls\_record.        PERFORM sub\_format\_record CHANGING ls\_record.        TRANSFER ls\_record TO lv\_file.     ENDLOOP.    \*  Закрываем файл:     CLOSE DATASET lv\_file.  CATCH cx\_sy\_file\_access\_error INTO lr\_file\_ex.  \*  Ошибки ввода, вывода (датасет в таком случае не открыт)...  CATCH lcx\_format\_error INTO lr\_format\_ex.  \*  Обрабатываем свою внутренюю ошибку при форматировании...  \*  при этом необходимо закрыть набор данных     CLOSE DATASET lv\_file.  CLEANUP.  \*  В случае если возникнет не обработанное исключение закроем набор данных:     CLOSE DATASET lv\_file.  ENDTRY. |

В случае возобновляемых оператором RESUME исключений, блок CLEANUP не выполняется. Блок CLEANUP, как и блок CATCH позволяет получить ссылочную переменную на вызванное исключение, с помощью дополнения [INTO oref].

### Передача исключений по цепочке

Необходимо понимать, что особая ситуация может передаваться через любое количество иерархий вызова перед финальной обработкой. Одна особая ситуация может инициировать вторую и т. д. Каждая инстанция должна оставаться действительной, независимо то того, был ли связанный блок CATCH уже обработан или нет. Поэтому необходимо убедиться, что предыдущая инстанция особой ситуации доступна с помощью, по крайней мере, одной ссылки. Атрибут общей инстанции PREVIOUS, наследуемый всеми классами особых состояний из CX\_ROOT, обеспечивает удобный для этого способ.

Пример:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69 | CLASS lcx\_very\_big DEFINITION INHERITING FROM cx\_static\_check.  ENDCLASS.    CLASS lcx\_calc\_error DEFINITION INHERITING FROM cx\_static\_check.  ENDCLASS.    CLASS lcl\_test\_exceptions DEFINITION.    PUBLIC SECTION.      METHODS:        do\_calc IMPORTING                  i\_num\_1 TYPE i OPTIONAL                  i\_num\_2 TYPE i OPTIONAL                RETURNING VALUE(re\_result) TYPE i                RAISING lcx\_calc\_error.    PRIVATE SECTION.      METHODS:        do\_summ IMPORTING                  i\_num\_1 TYPE i OPTIONAL                  i\_num\_2 TYPE i OPTIONAL                RETURNING value(re\_summ) TYPE i                RAISING lcx\_very\_big.  ENDCLASS.    CLASS lcl\_test\_exceptions IMPLEMENTATION.    METHOD do\_summ.      re\_summ = i\_num\_1 + i\_num\_2.      IF re\_summ > 100.        RAISE EXCEPTION TYPE lcx\_very\_big.      ENDIF.    ENDMETHOD.      METHOD do\_calc.      DATA:        lo\_very\_big TYPE REF TO lcx\_very\_big.        TRY.        me->do\_summ(          EXPORTING            i\_num\_1 = i\_num\_1            i\_num\_2 = i\_num\_2          RECEIVING            re\_summ = re\_result        ).      CATCH lcx\_very\_big INTO lo\_very\_big.        RAISE EXCEPTION TYPE lcx\_calc\_error EXPORTING previous = lo\_very\_big.      ENDTRY.    ENDMETHOD.  ENDCLASS.    DATA:    go\_test\_exceptions TYPE REF TO lcl\_test\_exceptions,    gv\_result TYPE i,    go\_calc\_error TYPE REF TO lcx\_calc\_error,    go\_big\_error  TYPE REF TO lcx\_very\_big.    START-OF-SELECTION.    CREATE OBJECT go\_test\_exceptions.      TRY.      go\_test\_exceptions->do\_calc(        EXPORTING          i\_num\_1   = 1000          i\_num\_2   = 500        RECEIVING          re\_result = gv\_result      ).    CATCH lcx\_calc\_error INTO go\_calc\_error.      go\_big\_error ?= go\_calc\_error->previous.    ENDTRY. |

Таким образом, пройдя по цепочке, мы всегда можем определить, в каком конкретном месте было инициировано исключение и что из-за этого произошло. Иногда при построении какой-либо ООП модели, удобно собрать всю цепочку из ошибок в каком-нибудь одном виде и выдать в качестве универсального исключения, в качестве примера рекомендую посмотреть [этот](http://scn.sap.com/community/abap/application-development/objects/blog/2010/01/07/abap-objects-custom-sap-erp-hcm-class-library--example-3--exceptions) пример.

### Возобновляемые исключения и повтор блока TRY

При срабатывании исключения, выполнение программы в текущем контексте завершается. Иногда необходимо не завершать выполнение текущего контекста, для этого были созданы так называемые возобновляемые исключения. Для того чтобы вызвать такое исключение, необходимо в операторе RAISE (или в THROW) указать что вызывается именно возобновляемое исключение, при этом для того чтобы воспользоваться оператором RESUME (который возвращает код обратно в то место где было вызвано исключение), необходимо у оператора CATCH указать дополнение BEFORE UNWIND (обозначает обработку возобновляемого исключения), иначе система вызовет исключение CX\_SY\_ILLEGAL\_HANDLER. При возврате в контекст, из которого было вызвано исключение блок CLEANUP не вызывается. Если в указанном в CATCH блоке не будет вызван оператор RESUME, контекст будет удален при выходе из блока CATCH.

Пример обработки возобновляемого исключения:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61 | CLASS lcx\_no\_num DEFINITION INHERITING FROM cx\_static\_check.  ENDCLASS.    CLASS lcx\_less\_zero DEFINITION INHERITING FROM cx\_no\_check.  ENDCLASS.    CLASS lcl\_test\_exceptions DEFINITION.    PUBLIC SECTION.      METHODS:        do\_summ IMPORTING                  i\_num\_1 TYPE i OPTIONAL                  i\_num\_2 TYPE i OPTIONAL                RETURNING value(re\_summ) TYPE int1                RAISING RESUMABLE(lcx\_no\_num).  ENDCLASS.    CLASS lcl\_test\_exceptions IMPLEMENTATION.    METHOD do\_summ.      IF i\_num\_1 IS NOT SUPPLIED OR i\_num\_2 IS NOT SUPPLIED.        " Данное исключение присутствует в интерфейсе, может быть обработано вне метода        RAISE RESUMABLE EXCEPTION TYPE lcx\_no\_num.      ENDIF.        TRY.        re\_summ = i\_num\_1 + i\_num\_2.        " Динамическая ошибка, при её обработке обнулим результат      CATCH CX\_SY\_CONVERSION\_OVERFLOW.        re\_summ = 0.      ENDTRY.    ENDMETHOD.  ENDCLASS.    DATA:    go\_test\_exceptions TYPE REF TO lcl\_test\_exceptions,    gv\_summ TYPE int1.    START-OF-SELECTION.    CREATE OBJECT go\_test\_exceptions.      TRY.      go\_test\_exceptions->do\_summ(        EXPORTING          i\_num\_2 = 1        RECEIVING          re\_summ = gv\_summ      ).    CATCH BEFORE UNWIND lcx\_no\_num.      RESUME.    ENDTRY.      WRITE: / 'Cумма без указания 1-го числа', gv\_summ.      go\_test\_exceptions->do\_summ(      EXPORTING        i\_num\_1 = 999        i\_num\_2 = 1      RECEIVING        re\_summ = gv\_summ    ).      WRITE: / 'Результат cуммы 999 и 1:', gv\_summ. |

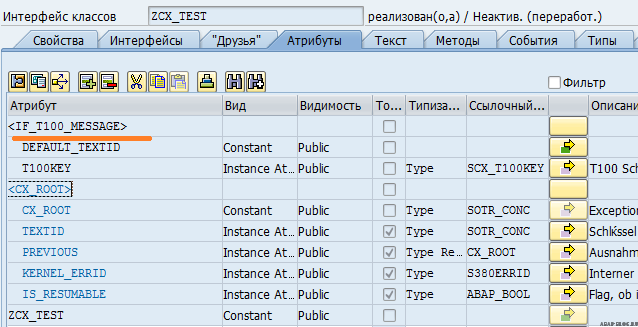
При обработке исключений так же есть возможность повтора блока TRY..CATCH, делается это с использованием оператора **RETRY**. Пример:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | PARAMETERS: number1 TYPE i,              number2 TYPE i.    DATA result  TYPE p DECIMALS 2.    TRY.      result = number1 / number2.    CATCH cx\_sy\_zerodivide.      number1 = 0.      RETRY.  ENDTRY. |

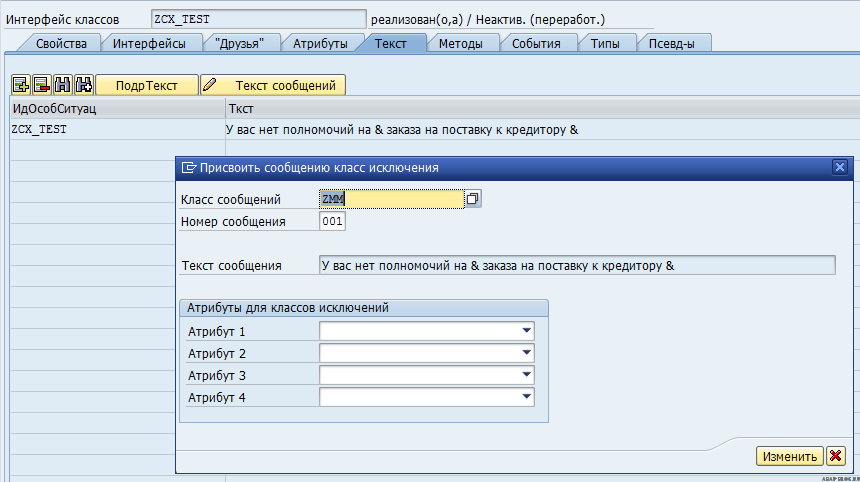
В данном случае если номер 2 будет равен нулю, система вызовет исключение, с помощью RETRY мы заново запустим блок TRY..CACTH, при этом уже исключение не возникнет, т.к. при делении нуля на ноль результатом будет ноль.

### Отображение сообщений из классов сообщений в тексты исключений

Начиная с версии 6.40, появилась возможность связывать тексты исключительных сообщений с классами сообщений (транзакция **SE91**). Как уже упоминалось выше для этого необходимо в конструкторе класса, указать галочку класс сообщений. При этом вместо интерфейса IF\_MESSAGE будет внедрен интерфейс IF\_T100\_MESSAGE (таблица T100 хранит в себе эти сообщения):

[](http://abap-blog.ru/wp-content/uploads/2013/08/19.png)

При редактировании текста, необходимо будет привязать его к классу и номеру сообщения, при этом заполнить параметры, если это необходимо:

[](http://abap-blog.ru/wp-content/uploads/2013/08/20.png)

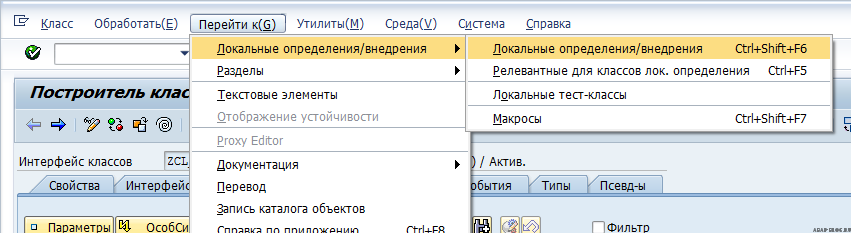
Начиная с NW 2004, оператор MESSAGE позволяет напрямую обработку исключений, внедряющих интерфейс IF\_T100\_MESSAGE:

|  |  |
| --- | --- |
| 1  2  3  4  5 | TRY.     ...  CATCH cx\_some\_exception INTO lr\_ex.     MESSAGE lr\_ex TYPE 'E'.  ENDTRY. |

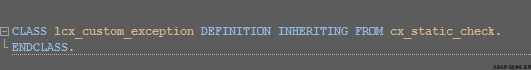
### Локальный класс исключения в приватном методе глобального класса

Бывают ситуации, когда для какого-либо приватного метода необходимо реализовать исключение, которое может быть вызвано исключительно данным методом (классом).  Реализовать подобное можно, если создать локальный класс исключений:

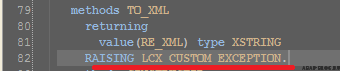
* Перейти в локальные определения/внедрения:

[](http://abap-blog.ru/wp-content/uploads/2013/08/21.png)

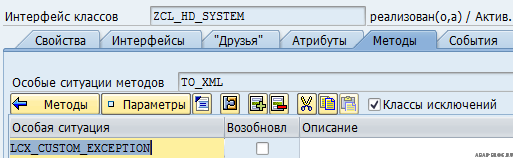
* Создать класс исключения:

[](http://abap-blog.ru/wp-content/uploads/2013/08/22.png)

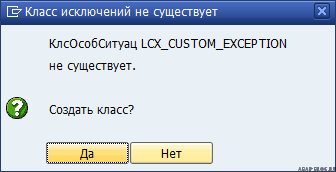
* Указать в методе имя локального класса исключения (обязательно в режиме редактирования исходного кода):

[](http://abap-blog.ru/wp-content/uploads/2013/08/23.png)

Результат:

[](http://abap-blog.ru/wp-content/uploads/2013/08/24.png)

Если попробовать сделать тоже самое в режиме редактирования на основе формуляров, выскочит предупреждение о том, что такого класса не существует:

[](http://abap-blog.ru/wp-content/uploads/2013/08/25.png)