

Exception Handling in Dao

September 19, 2009

Dao exceptions are organized into hierarchal structured classes. The base class is **Exception**,

```
class Exception
{
  routine Exception( content='' ){ Content = content }
  protected
  var Rout = '';
  var File = '';
  var FromLine = 0;
  var ToLine = 0;
  public
  var Name = 'Exception';
  var Content : any = 'undefined exception';
}
```

Currently supported classes are the following:

| Class | Exception Type |
|----------------------------------|--|
| Exception | Any |
| Exception.None | None |
| Exception.Any | Any or none |
| Exception.Error | Any error |
| Exception.Warning | Any warning |
| Exception.Error.Field | invalid field accessing |
| Exception.Error.Field.NotExist | field not exist |
| Exception.Error.Field.NotPermit | field not permit |
| Exception.Error.Float | invalid floating point operation |
| Exception.Error.Float.DivByZero | division by zero |
| Exception.Error.Float.Overflow | floating point overflow |
| Exception.Error.Float.UnderFlow | floating point underflow |
| Exception.Error.Index | invalid index |
| Exception.Error.Index.OutOfRange | index out of range |
| Exception.Error.Key | invalid key |
| Exception.Error.Key.NotExist | key not exist |
| Exception.Error.Param | invalid parameter list for the call |
| Exception.Error.Syntax | invalid syntax |
| Exception.Error.Type | invalid variable type for the operation |
| Exception.Error.Value | invalid variable value for the operation |

New exception classes can be derived from the above classes.

Any running time error will raise a proper exception instance. User-defined exception can be raised by using **raise** statement. Exceptions can be rescued by a block of code after a **rescue** statement. The exception handling syntax is the following:

```
stmt_tryrescue ::=
  'try' '{' stmt_block
  { '}' 'rescue' [ '(' expr ')' ] '{' stmt_block }*
```

where *expr* is a list of exception class(es). If there is no *expr* following **rescue**, any exception will be rescued. There can be multiple **rescue** statements in a try-rescue block. The raised exceptions will be checked against the exception classes presented in each of the **rescue** statement, if there is a matching, the corresponding code block is executed, and

the rescued exceptions can be accessed in a global list named **exceptions** . In the same try-rescue block different **rescue** statements can be used to handle different exceptions.

If all **rescue** statements have been tried and there is still un-rescued exception(s), the current function is aborted and the exceptions(s) is raised to its caller, if there is; otherwise, the program is aborted with the values of the un-rescued exceptions printed out. In a **rescue** statement, one may modify exception condition, and use **retry** to invoke the codes in try block.

Examples,

```
try{
  raise Exception.Error( "error test" ), Exception.Warning( "warning test" ),
  Exception( "exception" );
}rescue( Exception.Error, Exception.Warning ){
  stdio.println( exceptions );
}rescue{
  stdio.println( "rescued" );
}
```