

Re: Evaluating Exceptions, Try Except and Try Finally

Source: <http://coding.derkeiler.com/Archive/Delphi/alt.comp.lang.borland-delphi/2004-04/0363.html>

From: Skybuck Flying (*nospam_at_hotmail.com*)

Date: 04/08/04

Date: Thu, 8 Apr 2004 19:30:54 +0200

YOU CALL THIS CLEAR DOCUMENTATION ?????????????????????? :)

Try...finally Statements Sometimes you want to ensure that specific parts of an operation are completed, whether or not the operation is interrupted by an exception. For example, when a routine acquires control of a resource, it is often important that the resource be released, regardless of whether the routine terminates normally. In these situations, you can use a try...finally statement. The following example shows how code that opens and processes a file can ensure that the file is ultimately closed, even if an error occurs during execution.

Just ignore this example :P

```
Reset(F);  
try  
    ... // process file F  
finally  
    CloseFile(F);
```

end;The syntax of a try...finally statement is try statementList1 finally statementList2 end where each statementList is a sequence of statements delimited by semicolons. The try...finally statement executes the statements in statementList1 (the try clause). If statementList1 finishes without raising exceptions, statementList2 (the finally clause) is executed. If an exception is raised during execution of statementList1, control is transferred to statementList2; once statementList2 finishes executing, the exception is re-raised. If a call to the Exit, Break, or Continue procedure causes control to leave statementList1, statementList2 is automatically executed. Thus the finally clause is always executed, regardless of how the try clause terminates. If an exception is raised but not handled in the finally clause, that exception is propagated out of the try...finally statement, and any exception already raised in the try clause is lost. The finally clause should therefore handle all locally raised exceptions, so as not to disturb propagation of other exceptions.

HOW THIS THAT MORE EASY TO UNDERSTAND THEN A SIMPLE IF ELSE STATEMENT
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

OF FUCK IT LOL :)

THE IF THEN ELSE STATEMENT DOCUMENTATION IS JUST AS FUCKED :)'

"

If Statements There are two forms of if statement: if...then and the if...then...else. The syntax of an if...then statement is if expression then statement where expression returns a Boolean value. If expression is True, then statement is executed; otherwise it is not. For example, if $J <> 0$ then Result := I / J; The syntax of an if...then...else statement is if expression then statement1 else statement2 where expression returns a Boolean value. If expression is True, then statement1 is executed; otherwise statement2 is executed. For example, if $J = 0$ then

```
    Exit
else
    Result := I / J;
The then and else clauses contain one statement each, but it can be a structured statement. For example,
if  $J <> 0$  then
    begin
        Result := I / J;
        Count := Count + 1;
    end
else if Count = Last then
    Done := True
else
    Exit;
```

Notice that there is never a semicolon between the then clause and the word else. You can place a semicolon after an entire if statement to separate it from the next statement in its block, but the then and else clauses require nothing more than a space or carriage return between them. Placing a semicolon immediately before else (in an if statement) is a common programming error. A special difficulty arises in connection with nested if statements. The problem arises because some if statements have else clauses while others do not, but the syntax for the two kinds of statement is otherwise the same. In a series of nested conditionals where there are fewer else clauses than if statements, it may not seem clear which else clauses are bound to which ifs. Consider a statement of the form if expression1 then if expression2 then statement1 else statement2; There would appear to be two ways to parse this: if expression1 then [if expression2 then statement1 else statement2]; if expression1 then [if expression2 then statement1] else statement2; The compiler always parses in the first way. That is, in real code, the statement

```
if ... { expression1 } then
    if ... { expression2 } then
        ... { statement1 }
    else
        ... { statement2 }
is equivalent to
if ... { expression1 } then
    begin
```

```
if ... {expression2} then
  ... {statement1}
else
  ... {statement2}
```

end;The rule is that nested conditionals are parsed starting from the innermost conditional, with each else bound to the nearest available if on its left. To force the compiler to read our example in the second way, you would have to write it explicitly as

```
if ... {expression1} then
  begin
    if ... {expression2} then
      ... {statement1}
    end
  end
else
  ... {statement2};"Sigh.Skybuck :)
```