Dynamic Objects try/catch Cheat Sheet

jBASE Getting Started Guide

https://docs.zumasys.com/jbase/dynamic-objects/exceptions-try-&-catch-&-throw-&-$setcatch()/

Dynamic Objects supports **try**/**catch** like this:

```
    try
        /* Block of code for try */
    catch {variable_name}
        /* Block of code if an exception is thrown */
    end try
```

The **catch** statement may optionally be followed by a variable name. In this case, the variable is updated with the exception object that us created when an exception is thrown.

Simple example:

```
debian-~/tmp9: cat test9.jabba
    try
        zero = 0
        one = 1
        answer = one / zero
        crt "That should throw an exception -- why are we here?"
    catch ex
        crt "Oops, we had an exception"
        crt ex->$tojson(1)
    end try
```

And when the code is run:

```
debian-~/tmp9: test9
Divide by zero !!-- ZERO returned ,
Line     4 , Source test9.jabba
Oops, we had an exception
{
    "message":"DIVIDE_ZERO",
    "message_operands":[

    ],
    "message_long":"Divide by zero !!-
- ZERO returned ,\nLine     4 , Source test9.jabba\n",
    "catch_count":1,
    "recursive_catch_policy":-1,
    "try":{
            "source":"test9.jabba",
            "lineno":1
    },
    "throw":{
            "source":"test9.jabba",
            "lineno":4
    },
    "stack":[

    ]
}
```

The Exception Object.

As can be seen from the above example, when an exception is raised, we create an exception object. This exception object can be stored in a variable that is passed next to the **catch** statement. The properties of the exception object are:

- **message** - The message that accompanied the exception.
- **message_operands** - Any operands to the message that accompanied the exception.
- **message_long** - A concatenation of the message and message operands.
- **catch_count** - See "Recursive Catch Policy" at the above URL.
- **recursive_catch_policy** - See "Recursive Catch Policy" at the above URL.
- **try** - Shows the source name and line number where the **try** statement was executed.
- **throw** - Shows the source name and line number where the exception was raised.
- **stack** - An array of source names and line numbers showing the stack of subroutines/functions/methods below the current routine.

Raising an Exception

There are 2 basic ways of raising an exception.

First, an internal jBASE message is thrown. jBASE has a large error message collection, some of them correspond to fatal messages, some just warning or information messages. When a fatal message is displayed, and the debugger would normally be called, then instead of entering the debugger, we throw an exception. This is what is happening in the example.

Lower grade warning messages will not throw an exception.

Currently, only error messages that would enter the debugger are shown. Other errors like perhaps a READ having an error (other than no such record) will be handled as they always have and will not throw an exception.

Secondly, the application can itself throw an exception using the **throw** statement. In the examples below, we do a **throw** and no more code in the **try** block is executed, instead the code inside the **catch** block is executed.:

```
debian-~/tmp9: cat test10.jabba
    try
        crt "This is what I'm doing"
        throw
        crt "I should not be here"
    catch
        crt "I am in the catch section"
    end try
debian-~/tmp9: test10
This is what I'm doing
I am in the catch section
```

```
$option jabba
open "MD" to md else stop 201, "MD"
close md    ;* closing the file will cause the read to fail and will
run the code in the catch block

id = "list"
try
    read rec from md, id setting error number on error throw
catch
    crt "Unable to read ":id:" due to error number: " : error number
end try
```

This next example is a little more adventurous. We do a **throw** with a message and with message operands. This is picked up by the **catch** section and the message operands are displayed as part of the error message. You can see that when we do something like this:

```
    throw "ERRNAME"
```

then the exception object has the property **message** set to "ERRNAME".

```
debian-~/tmp9: cat test10.jabba
    max_age = 65
    try
        crt "What is your age ":
        input your_age
        if your_age gt max_age then
            throw "AGETOOBIG" , your_age , max_age
        end
    catch ex
        begin case
        case ex->message eq "AGETOOBIG"
            crt "Oops, age is too big"
            crt "Age entered was ":ex->message_operands[0]
            crt "Maximum age is  ":ex->message_operands[1]
        case 1
            crt "Exception thrown -- don't know what to do"
            crt ex->$tojson(1)
        end case
    end try

debian-~/tmp9: test10
What is your age ?99
Ooops, age is too big
Age entered was 99
Maximum age is  65
```

Unhandled Exceptions

If an exception is thrown by the application, and there is no active try/catch block, then an error message is generated and the debugger entered like this:

```
debian-~/tmp9: cat test11.jabba
throw
debian-~/tmp9: test11
Unhandled Exception Thrown
{
        "message":"",
        "message_operands":[

        ],
```

```
        "message_long":"",
        "throw":{
                "source":"test11.jabba",
                "lineno":1
        },
        "stack":[

        ]
}
Unhandled Exception
Trap from an error message, error message name = UNHANDLED_EXCEPTION
Source changed to ./test11.jabba
0001     throw
jBASE debugger->
```

Nested Try/Catch Blocks

You can nest try/catch blocks up to a depth of 30. If you need a depth of greater than 30, then perhaps the application needs re-thinking! For example:

```
    try
        crt "Do this"
        try
            crt "Do that"
        catch ex
            crt "Exception block 2"
        end try
    catch ex
        crt "Exception block 1"
    end try
```

Of course, the try/catch can be in different subroutines/methods/functions.

Try/Catch Block Scope

The scope of a try / catch block is limited to the subroutine/method/function it was executed and any child subroutines/methods/functions it executes.
The try / catch block is removed from jBASE when:
- The subroutine/function/method performs a **RETURN** statement from it. Note: This is a **RETURN** from the subroutine/function/method and any **RETURN** from a **GOSUB** is not affected.
- The **end try** statement is executed.
- The program performs a **STOP**. This means that if one program executes another with **EXECUTE** or **PERFORM**, and that second program issues an exception, it will not be caught by the parent program. All try / catch blocks are only active for a single program and do not traverse programs executed with **EXECUTE/PERFORM**.

In the following example, we go outside the try/catch block with a **GOSUB** and in the **GOSUB** we throw an exception. Even though we are executing outside of the try/catch block, the exception will still cause the 'catch' code to be executed:

```
debian-~/tmp9: cat test13.jabba
    try
        crt "Do this"
        gosub 100
        crt "Shouldn't get here" ; debug
    catch ex
        crt "We've had a catch"
    end try
    crt "Stopping now"
    stop
100       *
    crt "Subroutine 100 here"
    a = "xx"
    crt a+1
    crt "We shouldn't get here either" ; debug
    return
debian-~/tmp9: test13
Do this
Subroutine 100 here
Non-numeric value -- ZERO USED ,
Variable '(UNKNOWN)' , Line    13 , Source test13.jabba
1
We've had a catch
Stopping now
```

Recursive Catch policy and the $setcatch() method

The question to ask is this -- "What happens if you are inside a **catch** block and another exception is raised while execution is in an exception block?".
Different OOPs have different policies, and even then, it is not clear which language has what policy and seems to depend on the version of runtime or compiler or wishful thinking.
The default policy for jBASE is that once inside a **catch** block, that try/catch block is invalidated and goes out of scope. Hence any exception raised while inside that will cause an exception to be thrown by the outer try/catch block.

In the example below, we first handle a **throw**, but inside the **catch** code we do a divide by zero and that causes the outer **catch** block to be executed.

```
    try
        crt "Do this"
        try
            crt "Do that, throw an error"
            throw "OOPS"
            crt "Why am I here? "; debug
        catch ex
*
* This is the catch for the 'throw "OOPS"'
* Force an error with a divide by zero
*
            if ex->message ne "OOPS" then debug
            one = 1
            result = one / 0
            crt "Should not be here either" ; debug
        end try
    catch ex2
        crt "This should be the result of the divide by zero"
        crt "Error message for this catch is " : ex2->message
        if ex2->message ne "DIVIDE_ZERO" then debug
    end try


debian-~/tmp9: test14
Do this
Do that, throw an error
Divide by zero !!-- ZERO returned ,
Line    14 , Source test14.jabba
This should be the result of the divide by zero
Error message for this catch is DIVIDE_ZERO
```

This default behavior therefore is to invalidate and take out of context the try/catch block as soon as the **catch** is executed. This behavior can be changed with the new internal method called **$setcatch().**With **$setcatch()** you can change the recursive exception policy as follows:
- **$setcatch(-1)** This is the default policy and indicates to invalidate the try/catch block once the first exception is thrown.
- **$setcatch(0)** If an exception is thrown, go back to the 'catch' statement again. This will happen indefinitely.
- **$setcatch(nn)** If an exception is thrown, go back to the 'catch' statement again, but only do this a maximum of 'nn' times.
- **$getcatch()** Simply return the current value for the recursive exception policy.

Note that this call to **$setcatch()** only affects the current running program. It does not affect any parent programs, nor does it affect any child programs started with PERFORM/EXECUTE.
In the following example, an "unhandled exception" will be raised as an exception within the catch block has no outer block to execute when **throw "ANOTHER"** is executed.

```
debian-~/tmp9: cat test15.jabba
    try
        crt "Do this"
        throw "ERROR"
        debug
    catch ex
        throw "ANOTHER"
    end try

debian-~/tmp9: test15
Do this
Unhandled Exception Thrown
{
        "message":"ANOTHER",
        "message_operands":[

        ],
        "message_long":"ANOTHER",
        "throw":{
                "source":"test15.jabba",
                "lineno":6
        },
        "stack":[

        ]
}
Unhandled Exception
Trap from an error message, error message name = UNHANDLED_EXCEPTION
Source changed to ./test15.jabba
0006         throw "ANOTHER"
jBASE debugger->end
```