Article
Timothy Leavitt · Jul 8, 2020 7m read

Tips for debugging with %Status

Introduction

If you're solving complex problems in ObjectScript, you probably have a lot of code that works with %Status values. If you have interacted with persistent classes from an object perspective (%Save, %OpenId, etc.), you have almost certainly seen them. A %Status provides a wrapper around a localizable error message in InterSystems' platforms. An OK status ($$$OK) is just equal to 1, whereas a bad status ($$$ERROR(errorcode,arguments...)) is represented as a 0 followed by a space followed by a $ListBuild list with structured information about the error. $System.Status (see class reference) provides several handy APIs for working with %Status values; the class reference is helpful and I won't bother duplicating it here. There have been a few other useful articles/questions on the topic as well (see links at the end). My focus in this article will be on a few debugging tricks techniques rather than coding best practices (again, if you're looking for those, see links at the end).

Motivating Code Example

Note - don't ever write code like this! Always check your statuses and return them / throw them as exceptions (e.g. $$$ThrowStatus(someErrorStatus)) and it'll make debugging WAY easier.

```
Class DC.Demo.MaskedErrorStatus Extends %Persistent
{

Property Answer As %TinyInt;

ClassMethod Run() As %Status
{
    Set instance = ..%New()
    Set instance.Answer = 9000
    Do instance.%Save()

    Set instance = ..%OpenId(1,,.sc)
    Set instance.Answer = 42
    Do instance.%Save()

    Quit $$$OK
}

}
```

When run from terminal, an exception is thrown; something clearly went wrong.

```
USER>d ##class(DC.Demo.MaskedErrorStatus).Run()

 Set instance.Answer = 42
 ^
<INVALID OREF>zRun+5^DC.Demo.MaskedErrorStatus.1
```

%Status debugging trick #1: $System.OBJ.DisplayError()

You can always Do $System.OBJ.DisplayError() to print out the last error status that was created. This works because any time an error status is created (e.g., via $System.Status.Error), the variable %objlasterror is set to that status. You can also zwrite %objlasterror (equivalently). In the case from above:

```
USER 2d1>d $system.OBJ.DisplayError()
ERROR #5809: Object to Load not found, class 'DC.Demo.MaskedErrorStatus', ID '1'
```

%Status debugging trick #2: Stack traces

Inside every %Status is a stack trace for where the error was created. You can see this by zwriting the status:

```
USER 2d1>zw %objlasterror %objlasterror="0 "_$lb($lb(5809,"DC.Demo.MaskedErrorStatus"
,"1",,,,,,,$lb(,"USER",$lb("e^%LoadData+18^DC.Demo.MaskedErrorStatus.1^1","e^%Open+16
^%Library.Persistent.1^1","e^%OpenId+1^%Library.Persistent.1^1","e^zRun+4^DC.Demo.Mas
kedErrorStatus.1^1","d^^^0"))))/* ERROR #5809: Object to Load not found, class 'DC.De
mo.MaskedErrorStatus', ID '1' */
```

Want to see the stack trace in the more user-friendly error text for every status (e.g., using $System.OBJ.DisplayError() or $System.Status.GetErrorText(someStatus))? You can do that by setting ^%oddENV("callererrorinfo",$namespace)=1 or 2. You can see the effect here:

```
USER>set ^%oddENV("callererrorinfo",$namespace)=1
USER>d $system.OBJ.DisplayError()
ERROR #5809: Object to Load not found, class 'DC.Demo.MaskedErrorStatus', ID '1' [%Lo
adData+18^DC.Demo.MaskedErrorStatus.1:USER]
USER>set ^%oddENV("callererrorinfo",$namespace)=2
USER>d $system.OBJ.DisplayError()
ERROR #5809: Object to Load not found, class 'DC.Demo.MaskedErrorStatus', ID '1' [e^%
LoadData+18^DC.Demo.MaskedErrorStatus.1^1 e^%Open+16^%Library.Persistent.1^1 e^%OpenI
d+1^%Library.Persistent.1^1 e^zRun+4^DC.Demo.MaskedErrorStatus.1^1 d^^^0:USER]
USER>k ^%oddENV("callererrorinfo",$namespace)
USER>d $system.OBJ.DisplayError()
ERROR #5809: Object to Load not found, class 'DC.Demo.MaskedErrorStatus', ID '1'
```

Note that this is really only appropriate in a development environment - you don't want your users to see the internals of your code. (Really, it's best to avoid showing %Status values to users directly, in favor of more user-friendly application-specific error messages, but that's a topic for another day.)

%Status debugging trick #3: Fancy zbreak

Here's where it gets tricky - in the case of this code snippet, the root cause is an unchecked %Status from %Save() earlier in the code snippet. It's easy to imagine a much more complicated example where finding what went wrong is just really hard, especially if it's an error occurring somewhere further down in platform code. My preferred method for dealing with this - short of jumping into an interactive debugger - is to use a really fancy zbreak command in terminal:

```
USER>zbreak *%objlasterror:"N":"$d(%objlasterror)#2":"set ^mtemptl($i(^mtemptl))=%obj
lasterror"
```

...what does that mean?

zbreak <any time %objlasterror changes>:<don't do anything in the debugger itself>:<as long as %objlasterror is defined and has a value (e.g., it didn't go from being defined to being undefined)>:<run code to set the next subscript of an integer-subscripted global that isn't journalled (because it starts with mtemp, in case we're in a transaction when the %Status is created and it's been rolled back by the time we're looking at the log; also, with my initials as part of the global so if someone finds it in committed code or a bloated database they know to yell at me) to the error status>

Side note on zbreak: you can see currently defined breakpoints/watchpoints by running 'zbreak' with no arguments, and you can/should turn these breakpoints off when you're done with them by running break "off" - e.g.:

```
USER>zbreak
BREAK:
 No breakpoints
%objlasterror F:E S:0 C:"$d(%objlasterror)#2" E:"set ^mtemptl($i(^mtemptl))=%objlaste
rror"
USER>break "off"
USER>zbreak
BREAK:
 No breakpoints
 No watchpoints
```

So, what happens when the problematic method is run with the watchpoint set?

```
USER>zbreak *%objlasterror:"N":"$d(%objlasterror)#2":"set ^mtemptl($i(^mtemptl))=%obj
lasterror"

USER>d ##class(DC.Demo.MaskedErrorStatus).Run()

 Set instance.Answer = 42
 ^
<INVALID OREF>zRun+5^DC.Demo.MaskedErrorStatus.1

USER 2d1>zw ^mtemptl
^mtemptl=6
^mtemptl(1)="0 "_$lb($lb(7203,9000,127,,,,,,,$lb(,"USER",$lb("e^zAnswerIsValid+1^DC.D
emo.MaskedErrorStatus.1^1","e^%ValidateObject+3^DC.Demo.MaskedErrorStatus.1^4","e^%Se
rializeObject+3^%Library.Persistent.1^1","e^%Save+4^%Library.Persistent.1^2","d^zRun+
3^DC.Demo.MaskedErrorStatus.1^1","d^^^0"))))/* ERROR #7203: Datatype value '9000' gre
ater than MAXVAL allowed of 127 */
^mtemptl(2)="0 "_$lb($lb(7203,9000,127,,,,,,,$lb(,"USER",$lb("e^zAnswerIsValid+1^DC.D
emo.MaskedErrorStatus.1^1","e^%ValidateObject+3^DC.Demo.MaskedErrorStatus.1^4","e^%Se
rializeObject+3^%Library.Persistent.1^1","e^%Save+4^%Library.Persistent.1^2","d^zRun+
3^DC.Demo.MaskedErrorStatus.1^1","d^^^0")),"0 "_$lb($lb(5802,"DC.Demo.MaskedErrorStat
us:Answer",9000,,,,,,,$lb(,"USER",$lb("e^EmbedErr+1^%occSystem^1"))))))/* ERROR #7203
: Datatype value '9000' greater than MAXVAL allowed of 127-   > ERROR #5802: Datatype
 validation failed on property 'DC.Demo.MaskedErrorStatus:Answer', with value equal t
o "9000" */
^mtemptl(3)="0 "_$lb($lb(7203,9000,127,,,,,,,$lb("zAnswerIsValid+1^DC.Demo.MaskedErro
rStatus.1","USER",$lb("e^zAnswerIsValid+1^DC.Demo.MaskedErrorStatus.1^1","e^%Validate
Object+3^DC.Demo.MaskedErrorStatus.1^4","e^%SerializeObject+3^%Library.Persistent.1^1
","e^%Save+4^%Library.Persistent.1^2","d^zRun+3^DC.Demo.MaskedErrorStatus.1^1","d^^^0
"))))/* ERROR #7203: Datatype value '9000' greater than MAXVAL allowed of 127 */
^mtemptl(4)="0 "_$lb($lb(5802,"DC.Demo.MaskedErrorStatus:Answer",9000,,,,,,,$lb("Embe
dErr+1^%occSystem","USER",$lb("e^EmbedErr+1^%occSystem^1"))))/* ERROR #5802: Datatype
 validation failed on property 'DC.Demo.MaskedErrorStatus:Answer', with value equal t
```

```
o "9000" */
^mtemptl(5)="0 "_$lb($lb(7203,9000,127,,,,,,$lb("zAnswerIsValid+1^DC.Demo.MaskedErro
rStatus.1","USER",$lb("e^zAnswerIsValid+1^DC.Demo.MaskedErrorStatus.1^1","e^%Validate
Object+3^DC.Demo.MaskedErrorStatus.1^4","e^%SerializeObject+3^%Library.Persistent.1^1
","e^%Save+4^%Library.Persistent.1^2","d^zRun+3^DC.Demo.MaskedErrorStatus.1^1","d^^^0
")),"0 "_$lb($lb(5802,"DC.Demo.MaskedErrorStatus:Answer",9000,,,,,,,$lb("EmbedErr+1^%
occSystem","USER",$lb("e^EmbedErr+1^%occSystem^1"))))))/* ERROR #7203: Datatype value
 '9000' greater than MAXVAL allowed of 127-   > ERROR #5802: Datatype validation fail
ed on property 'DC.Demo.MaskedErrorStatus:Answer', with value equal to "9000" */
^mtemptl(6)="0 "_$lb($lb(5809,"DC.Demo.MaskedErrorStatus","1",,,,,,,$lb(,"USER",$lb("
e^%LoadData+18^DC.Demo.MaskedErrorStatus.1^1","e^%Open+16^%Library.Persistent.1^1","e
^%OpenId+1^%Library.Persistent.1^1","e^zRun+4^DC.Demo.MaskedErrorStatus.1^1","d^^^0")
)))/* ERROR #5809: Object to Load not found, class 'DC.Demo.MaskedErrorStatus', ID '1
' */
```

There's a bit of noise in there, but the key problem pops right out:

/* ERROR #7203: Datatype value '9000' greater than MAXVAL allowed of 127 */

Should've known better than to use %TinyInt! (And, more importantly, you should always check %Status values returned by methods you call.)

Related Reading

My preferred coding patterns for error handling and reporting
%Status vs Other Return Values in Caché ObjectScript Methods
About %objlasterror
How to set $$$envCallerErrorInfoGet
ObjectScript error handling snippets
ZBREAK command

#Best Practices #Error Handling #ObjectScript #Caché #Ensemble #InterSystems IRIS #InterSystems IRIS for Health

Source URL:https://community.intersystems.com/post/tips-debugging-status