
Artículo

[Alberto Fuentes](#) · 28 abr, 2021 · Lectura de 8 min

Consejos para depurar con %Status

Introducción

Si resuelves problemas complejos en ObjectScript, probablemente tienes mucho código que funciona con los valores de %Status. Si has interactuado con clases persistentes desde una perspectiva de objetos (%Save, %OpenId, etc.), casi seguro que las ha visto.

Un %Status proporciona una envoltura alrededor de un mensaje de error localizable en las plataformas de InterSystems. Un estado OK (\$\$\$OK) simplemente es igual a 1, mientras que un mal estado (\$\$\$ERROR(errorcode,arguments...)) se representa como un 0 seguido de un espacio seguido de una lista \$ListBuild con información estructurada sobre el error.

[\\$System.Status \(mira la referencia de clase\)](#) proporciona varias APIs útiles para trabajar con los valores de %Status; la referencia de clase es útil y no os molestaré duplicándola aquí. También ha habido algunos otros artículos/preguntas útiles sobre el tema (consulta los enlaces al final de esta publicación). En este artículo me centraré en algunas técnicas de depuración en vez de escribir las prácticas recomendadas (de nuevo, si las estás buscando, consulta los enlaces al final).

Ejemplo de código motivador

Nota: ¡nunca escribas un código como este! : Revisa siempre tus estados y devuélvelos/lánzalos como excepciones (por ejemplo, \$\$\$ThrowStatus(someErrorStatus)) y hará que la depuración sea MUY sencilla.

```
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
    }
}
```

Cuando se ejecuta desde el terminal, se lanza una excepción; es evidente que algo salió mal.

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

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

Truco #1 de depuración con %Status: \$System.OBJ.DisplayError()

Siempre puedes ejecutar \$System.OBJ.DisplayError() para imprimir el último estado de error que se creó. Esto funciona porque cada vez que se crea un estado de error (por ejemplo, por medio de \$System.Status.Error), la variable %objlasterror se establece en ese estado. También puedes utilizar zwrite %objlasterror (de forma equivalente). En el caso anterior:

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

Truco #2 de depuración con %Status: seguimiento de stacks

Dentro de cada %Status hay un seguimiento del stack (pila de llamadas) en el que se creó el error. Se puede ver esto al utilizar el estado zwrite:

```
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' */
```

¿Quieres ver el seguimiento de stacks en el texto de error (que es más sencillo de utilizar) para cada estado (por ejemplo, utilizando \$System.OBJ.DisplayError() o \$System.Status.GetErrorText(someStatus))? Puedes hacerlo al establecer %oddENV("callererrorinfo",\$namespace) a valores 1 o 2. Puedes ver el efecto aquí:

```
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'
```

Ten en cuenta que esto sólo es apropiado en un entorno de desarrollo - no querrás que tus usuarios vean el interior de tu código -. En realidad, es mejor evitar mostrar los valores de %Status directamente a los usuarios. Son preferibles los mensajes de error específicos de la aplicación, más fáciles de utilizar. Pero ese es un tema para otro día.

Truco #3 de depuración con %Status: el elegante zbreak

Aquí es donde se pone difícil - en el caso de este fragmento de código, la causa raíz es un %Status no verificado de %Save() que antes estaba en el fragmento de código. Es fácil imaginar un ejemplo mucho más complicado en el que encontrar lo que ha fallado es realmente difícil, especialmente si se trata de un error que se produce en algún punto del código de la plataforma. Mi método preferido para gestionar esto, sin saltar a un depurador

Consejos para depurar con %Status

Published on InterSystems Developer Community (<https://community.intersystems.com>)

interactivo, es utilizar un comando [zbreak](#) en el terminal:

```
USER>zbreak *%objlasterror:"N": "$d(%objlasterror)#2": "set ^mtemp1($i(^mtemp1))=%objlasterror"
```

...¿qué quiere decir eso?

zbreak <cada vez que %objlasterror cambie>:<no haga nada en el depurador mismo>:<mientras %objlasterror esté definido y tenga un valor (por ejemplo, no ha pasado de estar definido a estar indefinido)>:<ejecuta el código para establecer el siguiente subíndice de un global con un subíndice entero que no está en journal (porque comienza con mtemp, en caso de que estemos en una transacción cuando se crea %Status y se haya revertido para cuando observemos el registro; también, con mis iniciales como parte del global para que si alguien lo encuentra en el código comprometido o en una base de datos inflada sepa que debe llamarme) al estado de error>

Nota adicional sobre zbreak: puedes ver los breakpoints/watchpoints definidos actualmente si ejecutas "zbreak" sin argumentos, y puedes/debes desactivar estos breakpoints cuando hayas terminado con ellos ejecutando break "off", por ejemplo:

```
USER>zbreak
BREAK:
  No breakpoints
%objlasterror F:E S:0 C: "$d(%objlasterror)#2" E: "set ^mtemp1($i(^mtemp1))=%objlasterror"
USER>break "off"
USER>zbreak
BREAK:
  No breakpoints
  No watchpoints
```

Entonces, ¿qué sucede cuando el método problemático se ejecuta con el watchpoint establecido?

```
USER>zbreak *%objlasterror:"N": "$d(%objlasterror)#2": "set ^mtemp1($i(^mtemp1))=%objlasterror"
```

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

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

```
USER 2d1>zw ^mtemp1
^mtemp1=6
^mtemp1(1)="0 _$_lb($_lb(7203,9000,127,,,,,,,$lb("USER",$lb("e^zAnswerIsValid+1^DC.Demo.MaskedErrorStatus.1^1","e^%ValidateObject+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 */
^mtemp1(2)="0 _$_lb($_lb(7203,9000,127,,,,,,,$lb("USER",$lb("e^zAnswerIsValid+1^DC.Demo.MaskedErrorStatus.1^1","e^%ValidateObject+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,,,,,,,$lb("USER",$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 to '9000' */
^mtemp1(3)="0 _$_lb($_lb(7203,9000,127,,,,,,,$lb("zAnswerIsValid+1^DC.Demo.MaskedErrorStatus.1^1","e^%ValidateObject+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")))
```

```
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
' */
```

Hay un poco de ruido ahí, pero el problema clave salta a la vista:

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

¡Debería haber sabido que no debería utilizar %TinyInt! (Y, lo que es más importante, siempre debes verificar los valores de %Status devueltos por los métodos que llamas).

Lecturas relacionadas

[Mis patrones de codificación preferidos para la gestión y reporte de errores](#)

[%Status frente a otros valores de retorno en los métodos ObjectScript de Caché](#)

[Sobre %objlasterror](#)

[Cómo configurar \\$\\$SenvCallerErrorInfoGet](#)

[Fragmentos de código para gestión de errores de ObjectScript](#)

[Comando ZBREAK](#)

[#Gestión de errores](#) [#Mejores prácticas](#) [#ObjectScript](#) [#Caché](#) [#Ensemble](#) [#InterSystems IRIS](#) [#InterSystems IRIS for Health](#)

URL de fuente: <https://es.community.intersystems.com/post/consejos-para-depurar-con-status>