Article
Daniel Tamajon · Sep 1, 2020 5m read

Open Exchange

ObjectScript error management

Error management on InterSystems languages has been evolving along time. Next, we will show the different implementations and why you should use the TRY/THROW/CATCH mechanism.

You can read official error recommendations here.

InterSystems will not mark as obsoletes the non-recommended error management methods to allow giving support to legacy applications. We recommend using tools like objectscriptQuality to detect that legacy unrecommended usage along with many other possible issues and bugs.

$ZERROR

$ZERROR is the older error management mechanism, supporting different implementations from standard "M". Nowadays is being yet accepted but highly unrecommended.

It is really easy to do a wrong design usage of $ZERROR assuming you already have control over the variable. $ZERROR is a global public variable, accessible and alterable by any routine (from InterSystems or custom) that is being executed in the current process. So, its value is only reliable at the very same moment in which the error is produced. InterSystems does not guarantee that $ZERROR holds an old value on a call to a system library.

Let's analyze some cases.

Case 1: Error code on custom code

Line	Code	Comments	$ZERROR value
1	Set ...		""
2	Set ...		""
...	Do ...		""
...	...		""
N-m	Do CacheMethodCall()	Call to another Caché system methods	**""**
...			""
N	Set VarXX = MyMethod()	The custom method generates an ObjectScript error	**<UNDEFINED>B+3^Infinity Method *varPatient**
N+1	Set ⋯		<UNDEFINED>B+3^Infinity Method *varPatient
N+2	Do OtherCacheMethodCall()	Caché system method. $ZERROR is not updated if there is no error.	<UNDEFINED>B+3^Infinity Method *varPatient
...	If ...		<UNDEFINED>B+3^Infinity Method *varPatient
...	...		<UNDEFINED>B+3^Infinity Method *varPatient
...	While ...		<UNDEFINED>B+3^Infinity Method *varPatient
...	**If $ZERROR ' = " " Quit " Error "**		<UNDEFINED>B+3^Infinity Method *varPatient
N+m	Quit "OK"		<UNDEFINED>B+3^Infinity

Line	Code	Comments	$ZERROR value
			Method *varPatient

Caso 2: False positive given to internal Caché error

In that case, custom code has gone fine but an error has been raised from internal Caché error.

Line	Code	Comments	$ZERROR value
1	Set …		""
2	Set …		""
..	Do …		""
..	…		""
N-m	Do CacheMethodCall()	Internal error but it is managed internally and decides to continue execution	<UNDEFINED>occKl+3^ MetodoInternoCache *o0bxVar
..			<UNDEFINED>occKl+3^ MetodoInterno *o0bxVar
N	**Set VarXX = MyMethod() // OK**		<UNDEFINED>occKl+3^ MetodoInterno *o0bxVar
N+1	Set …		<UNDEFINED>occKl+3^ MetodoInterno *o0bxVar
N+2	Do OtherCacheMethodCall() // OK		<UNDEFINED>occKl+3^ MetodoInterno *o0bxVar
…	If …		<UNDEFINED>occKl+3^ MetodoInterno *o0bxVar
…	…		<UNDEFINED>occKl+3^ MetodoInterno *o0bxVar
…	While …		<UNDEFINED>occKl+3^ MetodoInterno *o0bxVar
…	**If $ZERROR ’ ="" Quit "Error"**	An error is detected while there is no error at all	**<UNDEFINED>occKl+3^ MetodoInterno *o0bxVar**
N+m	Quit "OK"		<UNDEFINED>occKl+3^ MetodoInterno *o0bxVa

Caso 3: False-negative has given to $ZERROR reset in Caché internal code

In that case, there is a direct or indirect call to an internal Caché method or routine that resets the public variable $ZERROR even while there is no error situation.

Line	Code	Comments	$ZERROR
1	Set ...		""
2	Set ...		""
...	Do ...		""
...	...		""
N-m	**Do CacheMethodCall()**		<UNDEFINED>occKl+3^ MetodoInternoCache *o0bxVar
...			<UNDEFINED>occKl+3^ MetodoInterno *o0bxVar
N	**Set VarXX = MyMethod()**		<UNDEFINED>occKl+3^ MetodoInterno *o0bxVar
N+1	Set …		<UNDEFINED>occKl+3^ MetodoInterno *o0bxVar
N+2	**Do OtherCacheMethodCall()**	Internal method that resets $ZERROR whether there is error or not	""
...	If ...		""
...	...		""

Line	Code	Comments	$ZERROR
...	While ...		""
...	**If $ZERROR ’ ="" Quit "Error"**	Error not detected	""
N+m	Quit "OK"		""

$ZTRAP

With $ZTRAP the error is managed in a context, so there is no risk to be overwritten unexpectedly outside the context. When an error arises, the control is returned to the first error control in the call stack.

When an error is raised and the error has been handled, you must clear $ZTRAP in order to avoid possible infinite loop if another error occurs.

So, $ZTRAP is more advanced than $ZERROR in error management but yet delegates the need to add operations by the developer that can generate more errors.

Check the section Handling errors with $ZTRAP in the official documentation for a deeper understanding of using this method.

%Status

That method is being used by system libraries, so this is the mechanism that must be used when doing a call to system libraries.

You can check how to use it here.

TRY/THROW/CATCH

That is the most modern error management method and the currently recommended method.

You can check how to use it here.

This method manage errors in context and does not delegate on developer the management of internal error variables.

Advantages

There is a lot of literature about the TRY/THROW/CATCH method, so let's enumerate just some advantages:

- Allows you to decide what to do in case of exception in a clean way, separating error code handling from regular code
- Simplifies error detection, as you don't need to check for errors after each operation
- Allows error propagation to upper layers
- Supports runtime error, allowing to recover and continue running after a crash

Disadvantages

The loss of slight performance is the most remarkable disadvantage, so it's important to know when you have to use it or not.

Usually, it is not necessary to use TRY/THROW/CATCH on each method and many times simple validations before some operations will avoid errors avoiding also the unnecessary use of TRY/THROW/CATCH.

Conclusions

Avoid using $ZERROR and $ZTRAP.

Use %STATUS only when doing a call to system libraries.

Manage your errors using the TRY/THROW/CATCH method but without abusing.

#Error Handling #ObjectScript #Caché #InterSystems IRIS
Check the related application on InterSystems Open Exchange