Subject: Re: Generating errors
Posted by David Fanning on Thu, 16 Nov 2006 15:47:57 GMT
View Forum Message <> Reply to Message

Braedley writes:

> Is the any way to generate a generic error if a given condition fails?
> For example:
>
> catch, the_error
> if the_error ne 0 then begin
> ;fix what needs to be fixed
> catch, /cancel
> return ;so the user can try again
> endif
>
> ;some other code
>
> if n ne (*state).m then begin
> ;do something to notify user
> ;throw the error
> endif
>
> ;some more code that may generate errors
>
>
> BTW, this is in the event handler of a widget program, and it's assumed
> that the problem can be fixed before the user tries again.
> Is there any well defined mechanism to do what I want, or do I have to
> do something like
>
> a=intarr(3)
> a[4]=0
>
> in order to generate my error?

If you are going to fix the error, then you want it fixed
in THIS module. Don't RETURN out of your error handler code!
(Unless you plan to fix something that is in the state or
info structure.) If you don't RETURN, you will run through
the code again. (The CATCH, /CANCEL in your error handler
code will prevent you from getting into an infinite loop
if you don't *completely* fix the problem! :-)

In terms of throwing an error, the MESSAGE command is usually
used for this purpose. See the document ion for my ERROR_MESSAGE
program for how to Catch and Throw errors.

Cheers,

David

Cheers,

David
--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Sepore ma de ni thui. ("Perhaps thou speakest truth.")

Subject: Re: Generating errors
Posted by Braedley on Thu, 16 Nov 2006 17:13:40 GMT
View Forum Message <> Reply to Message

The error would be in the state structure, and would be due to it not being updated properly after some other error (that I still have to devise a runtime fix for). The thing is that both n and (*state).m both become useless for performing the action that the user wanted to perform after the problem was fixed. In all likelihood, the original value of (*state).m is different from what it should have been, and it then becomes impossible to retrieve the value that it should have been with 100% accuracy before the fix is implemented. Since I don't want to perform an action on (*state).m (that is not easy to undo) without being sure that it's the action that the user wants, I'm forced to return after the problem is fixed for future use.

David Fanning wrote:
> Braedley writes:
>
>> Is the any way to generate a generic error if a given condition fails?
>> For example:
>>
>> catch, the_error
>> if the_error ne 0 then begin
>> ;fix what needs to be fixed
>> catch, /cancel
>> return ;so the user can try again
>> endif
>>
>> ;some other code
>>
>> if n ne (*state).m then begin
>> ;do something to notify user
>> ;throw the error

>> endif
>>
>> ;some more code that may generate errors
>>
>>
>> BTW, this is in the event handler of a widget program, and it's assumed
>> that the problem can be fixed before the user tries again.
>> Is there any well defined mechanism to do what I want, or do I have to
>> do something like
>>
>> a=intarr(3)
>> a[4]=0
>>
>> in order to generate my error?
>
> If you are going to fix the error, then you want it fixed
> in THIS module. Don't RETURN out of your error handler code!
> (Unless you plan to fix something that is in the state or
> info structure.) If you don't RETURN, you will run through
> the code again. (The CATCH, /CANCEL in your error handler
> code will prevent you from getting into an infinite loop
> if you don't *completely* fix the problem! :-)
>
> In terms of throwing an error, the MESSAGE command is usually
> used for this purpose. See the document ion for my ERROR_MESSAGE
> program for how to Catch and Throw errors.
>
> Cheers,
>
> David
>
> Cheers,
>
> David
> --
> David Fanning, Ph.D.
> Fanning Software Consulting, Inc.
> Coyote's Guide to IDL Programming: http://www.dfanning.com/
> Sepore ma de ni thui. ("Perhaps thou speakest truth.")

Subject: Re: Generating errors
Posted by David Fanning on Thu, 16 Nov 2006 17:27:03 GMT
View Forum Message <> Reply to Message

Braedley writes:

> The error would be in the state structure, and would be due to it not

> being updated properly after some other error (that I still have to
> devise a runtime fix for). The thing is that both n and (*state).m
> both become useless for performing the action that the user wanted to
> perform after the problem was fixed. In all likelihood, the original
> value of (*state).m is different from what it should have been, and it
> then becomes impossible to retrieve the value that it should have been
> with 100% accuracy before the fix is implemented. Since I don't want
> to perform an action on (*state).m (that is not easy to undo) without
> being sure that it's the action that the user wants, I'm forced to
> return after the problem is fixed for future use.

Huh? Not sure I'm following everything here... Too many
fingers pointing in too many directions, I guess.

In most of the widget programs I've ever written, the
user can get himself out of it, if the widget program
just keeps running. That is to say, there is *usually*
some way the user can fix the problem for themselves.
So, I usually just tell the user what the problem was,
and keep the program alive for them to figure out the
solution. An error handler like this usually works
well for me:

```
  Catch, theError
  IF theError NE 0 THEN BEGIN
    Catch, /Cancel
    void = Error_Message()
    ; Whatever clean-up is needed in THIS module. For
    ; example, put the info structure back in TLB, etc.
    RETURN
  ENDIF
```

If you have errors that you *can't* recover from (this is
what your explanation seems to imply to me), then I think
you are basically hosed. Destroy your TLB and start over. :-)

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Sepore ma de ni thui. ("Perhaps thou speakest truth.")

Subject: Re: Generating errors
Posted by Braedley on Thu, 16 Nov 2006 17:47:41 GMT
View Forum Message <> Reply to Message

Sorry for the confusion. The program will make changes to parts of it's state structure. In doing so, the original event sent by the user can no longer be performed until the user triggers the event again, after the problem has been fixed, so the error handler returns out of the event handler and waits for another event. If the user does totally **** things up, there is another option for him, but he still does have to reload all the data he was working on. And I was already using the same type of error handler that you suggested.

David Fanning wrote:
> Braedley writes:
>
>> The error would be in the state structure, and would be due to it not
>> being updated properly after some other error (that I still have to
>> devise a runtime fix for). The thing is that both n and (*state).m
>> both become useless for performing the action that the user wanted to
>> perform after the problem was fixed. In all likelihood, the original
>> value of (*state).m is different from what it should have been, and it
>> then becomes impossible to retrieve the value that it should have been
>> with 100% accuracy before the fix is implemented. Since I don't want
>> to perform an action on (*state).m (that is not easy to undo) without
>> being sure that it's the action that the user wants, I'm forced to
>> return after the problem is fixed for future use.
>
> Huh? Not sure I'm following everything here... Too many
> fingers pointing in too many directions, I guess.
>
> In most of the widget programs I've ever written, the
> user can get himself out of it, if the widget program
> just keeps running. That is to say, there is *usually*
> some way the user can fix the problem for themselves.
> So, I usually just tell the user what the problem was,
> and keep the program alive for them to figure out the
> solution. An error handler like this usually works
> well for me:
>
> Catch, theError
> IF theError NE 0 THEN BEGIN
> Catch, /Cancel
> void = Error_Message()
> ; Whatever clean-up is needed in THIS module. For
> ; example, put the info structure back in TLB, etc.
> RETURN
> ENDIF
>

> If you have errors that you *can't* recover from (this is
> what your explanation seems to imply to me), then I think
> you are basically hosed. Destroy your TLB and start over. :-)
>
> Cheers,
>
> David
>
>
> --
> David Fanning, Ph.D.
> Fanning Software Consulting, Inc.
> Coyote's Guide to IDL Programming: http://www.dfanning.com/
> Sepore ma de ni thui. ("Perhaps thou speakest truth.")

Subject: Re: Generating errors
Posted by David Fanning on Thu, 16 Nov 2006 17:50:29 GMT
View Forum Message <> Reply to Message

Braedley writes:

> Sorry for the confusion. The program will make changes to parts of
> it's state structure. In doing so, the original event sent by the user
> can no longer be performed until the user triggers the event again,
> after the problem has been fixed, so the error handler returns out of
> the event handler and waits for another event. If the user does
> totally **** things up, there is another option for him, but he still
> does have to reload all the data he was working on. And I was already
> using the same type of error handler that you suggested.

Sounds like we are on the same page then. :-)

Cheers,

David
--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Sepore ma de ni thui. ("Perhaps thou speakest truth.")
