Subject: Re: IDLVM and retall
Posted by JD Smith on Thu, 07 Sep 2006 22:57:58 GMT
View Forum Message <> Reply to Message

On Thu, 07 Sep 2006 13:14:12 -0600, David Fanning wrote:

> JD Smith writes:
>
>> Imagine you are in some deep level of the calling stack in a complex
>> widget program, and an error occurs. If you are running
>> interactively, you'd like to report the error to the user with a
>> dialog, and then after they dismiss it, continue running the
>> application. The only way to do this simply is to use RETALL, which
>> returns all the way to the active command line (a fact many have
>> discovered: when a widget app crashes, use RETALL and you're often
>> back in business).
>
> But why use RETALL!? I've always just used RETURN
> in these cases and widget programs continue to
> run pretty much forever, VM or not. Is there someplace
> you are trying to get to with RETALL that you don't
> get to with RETURN?

Here's a better example:

```
pro MyClass:f1, ev
  if something_is_wrong self->Error
  do_calc
end

pro MyClass:f2,ev
  self->f1,ev
  do_another_calc
end
```

> Can this be a difference between Windows and UNIX?

If you simply return (or return twice, if you were able to do that
from within self->Error), you'd be right back inside MyClass:f2 and
do_another_calc would run, which is not what you want. With RETALL,
the entire calling stack is returned from, not just one step in the
stack. A single RETURN only works if you are at the very tip of a
calling stack that will neatly fold up as soon as you go off the end.
It so happens that event callback functions work this way, so if your
errors always occur and are dealt with there, RETURN may be equivalent
to RETALL. The point is to be able to "continue with processing"
after issuing an error anywhere in the calling stack. This isn't

guaranteed to be a good idea (since the error may leave you in a state
such that further processing will continue the breakage), but if
you're careful, it's a nice way to abort whatever you were doing when
the breakage occurred.

Or am I missing something? Do you have a way of returning a widget
program to running anywere from anywhere in the calling stack? Keep
in mind that there are many many event callback procedure operating
simultaneously, so you don't want to put a CATCH statement in all of
them (plus it's slow for each motion event to call CATCH).

JD