Description

error(rc) displays the standard Stata error message associated with return code rc and returns rc; see [P] error for a listing of return codes. error() does not abort execution; standard usage is exit(error(rc)).

_error() aborts execution and produces a traceback log.

_error(errnum) produces a traceback log with standard Mata error message errnum; see [M-2] Errors for a listing of the standard Mata error codes.

_error(errtxt) produces a traceback log with error number 3498 and custom text errtxt.

_error(errnum, errtxt) produces a traceback log with error number errnum and custom text errtxt.

If errtxt is specified, it should contain straight text; SMCL codes are not interpreted.

Syntax

real scalar error(real scalar rc)

void _error(real scalar errnum)

void _error(string scalar errtxt)

void _error(real scalar errnum, string scalar errtxt)

Remarks and examples

Remarks are presented under the following headings:

Use of _error()
Use of error()

Use of _error()

_error() aborts execution and produces a traceback log:

: myfunction(A,B)
  mysub(): 3200 conformity error
  myfunction(): - function returned error
  <istmt>: - function returned error
  r(3200);

The above output was created because function mysub() contained the line

_error(3200)
and 3200 is the error number associated with the standard message “conformability error”; see [M-2] Errors. Possibly, the code read

```plaintext
if (rows(A)!=rows(B) | cols(A)!=cols(B)) {
    _error(3200)
}
```

Another kind of mistake might produce

```plaintext
: myfunction(A,B)
    mysub(): 3498  zeros on diagonal not allowed
    myfunction(): -  function returned error
    <stmt>:  -  function returned error
r(3498);
```

and that could be produced by the code

```plaintext
if (diag0cnt(A)>0) {
    _error("zeros on diagonal not allowed")
}
```

If we wanted to produce the same text but change the error number to 3300, we could have coded

```plaintext
if (diag0cnt(A)>0) {
    _error(3300, "zeros on diagonal not allowed")
}
```

Coding _error() is not always necessary. In our conformability-error example, imagine that more of the code read

```plaintext
... if (rows(A)!=rows(B) | cols(A)!=cols(B)) {
    _error(3200)
}
C = A + B
...
```

If we simplified the code to read

```plaintext
... C = A + B ...
```

the conformability error would still be detected because + requires p-conformability:

```plaintext
: myfunction(A,B)
    +: 3200  conformability error
    mysub():  -  conformability error
    myfunction(): -  function returned error
    <stmt>:  -  function returned error
r(3200);
```

Sometimes, however, you must detect the error yourself. For instance,
if (rows(A)! = rows(B) | cols(A)! = cols(B) | rows(A)! = 2*cols(A)) {
    _error(3200)
}
C = A + B

We assume we have some good reason to require that A has twice as many rows as columns. +, however, will not require that, and perhaps no other calculation we will make will require that, either. Or perhaps it will be subsequently detected, but in a way that leads to a confusing error message for the caller.

Use of error()

error(rc) does not cause the program to terminate. Standard usage is

    exit(error(rc))

such as

    exit(error(503))

In any case, error() does not produce a traceback log:

    : myfunction(A,B)
    conformability error
    r(503);

error() is intended to be used in functions that are subroutines of ado-files:

begin example.ado

    program example
        version 16.0
        ...
        mata: myfunction(“mat1”, “mat2”)    ...
        end
    version 16.0
    mata:
        void myfunction(string scalar matname1, string scalar matname2)
        {
            ...
            A = st_matrix(matname1)
            B = st_matrix(matname2)
            if (rows(A)! = rows(B) | cols(A)! = cols(B)) {
                exit(error(503))
            }
            C = A + B
            ...
        }
    end

end example.ado

This way, when the example command is used incorrectly, the user will see

    . example ...
    conformability error
    r(503);
rather than the traceback log that would have been produced had we omitted the test and
\texttt{exit(error(503))}:

\begin{verbatim}
.example .
+: 3200 conformability error
  myfunction(): - function returned error
  <istmt>: - function returned error
  r(3200);
\end{verbatim}

\section*{Conformability}

\texttt{error(rc)}:

\begin{verbatim}
rc: 1 x 1
result: 1 x 1
\end{verbatim}

\texttt{_error(errnum)}:

\begin{verbatim}
errnum: 1 x 1
result: void
\end{verbatim}

\texttt{_error(errtxt)}:

\begin{verbatim}
errtxt: 1 x 1
result: void
\end{verbatim}

\texttt{_error(errnum, errtxt)}:

\begin{verbatim}
errnum: 1 x 1
errtxt: 1 x 1
result: void
\end{verbatim}

\section*{Diagnostics}

\texttt{error(rc)} does not abort execution; code \texttt{exit(error(rc))} if that is your desire; see [M-5] \texttt{exit()}.

The code \texttt{error(rc)} returns can differ from \texttt{rc} if \texttt{rc} is not a standard code or if there is a better code associated with it.

\texttt{error(rc)} with \texttt{rc = 0} produces no output and returns 0.

\texttt{_error(errnum), _error(errtxt), and _error(errnum, errtxt)} always abort with error. \texttt{_error()} will abort with error because you called it wrong if you specify an \texttt{errnum} less than 1 or greater than 2,147,483,647 or if you specify an \texttt{errtxt} longer than 100 characters. If you specify an \texttt{errnum} that is not a standard code, the text of the error messages will read “Stata returned error”.

\section*{Also see}

[M-2] \texttt{Errors} — Error codes

[M-5] \texttt{exit()} — Terminate execution

[M-4] \texttt{Programming} — Programming functions