

APPLICATION DEVELOPER'S GUIDE  
.....

Lotus

**NOTES VIP**  
RELEASE 1

Visual Programmer for Notes

---

# Appendix B

## Error Handling

Notes ViP provides two tools for handling system errors: the Error event and the ON ERROR statement. The one you use depends on the type of error handling you need to perform and on the kinds of Notes ViP objects you are working with. For example, although you can use both tools to handle most programmatic errors, only the Error event can trap errors that occur outside the scope of an object's script.

This appendix explains the following:

- How to choose an error-handling method. It explains the differences between the ON ERROR statement and the Error event and recommends how to use each.
- How Notes ViP interacts with error handlers.
- How to use an application window's Error event. This section provides a commented example of a simple Error event.
- How to use an ON ERROR handler within an object's script. This section provides a commented example of an ON ERROR handler.

---

### Choosing an error-handling method

Before using Notes ViP error handling in your application, decide what type of error you want to trap, as well as how you want your users to respond to it. The following two sections list common error scenarios and recommend which method to use for each.

#### When to use the Error event

Use the Error event when:

- You need to trap errors that occur when no script is running. Such cases include:
  - Opening an application window that has a data object with AutoQuery set to TRUE
  - Using UI gestures to execute queries, and insert, update or delete a row
  - Performing dynamic data exchange (DDE) activity

- A script is running, but you do not want to establish ON ERROR handlers in every script of every object.

The application window Error event can handle an error encountered by any object in the application window.

- You have many places in your application that might encounter errors, but you want to handle them all in the same manner.

## **When to use an ON ERROR handler**

Use an ON ERROR handler when:

- You need to trap BASIC errors.
- You want to trap a system program error within a script.
- You have many places in your application that might encounter errors, but you want to handle each case differently so that you can provide local and procedural control. See Example 1 in "Handling Errors with the ON ERROR statement" later in this appendix.

---

## **How Notes ViP interacts with error handlers**

By default, when a system error condition occurs, Notes ViP looks for an ON ERROR handler for the particular error. If there is an ON ERROR handler, Notes ViP passes control to it. (Note that the presence of any ON ERROR handler is not sufficient. The handler must either be for the particular error or for all errors.)

If there is no ON ERROR handler for the particular error, Notes ViP examines the Error event script for the application window.

If it is empty, Notes ViP displays the error in a message box and stops execution. If the script has an Error event, it passes control to the Error event handler, which follows its programmed instructions. For example, it could display a message box that presents the user with Abort, Retry, and Ignore buttons, and waits for user input.

Figure B-1 explains how Notes ViP interacts with the Error event and the ON ERROR statement.

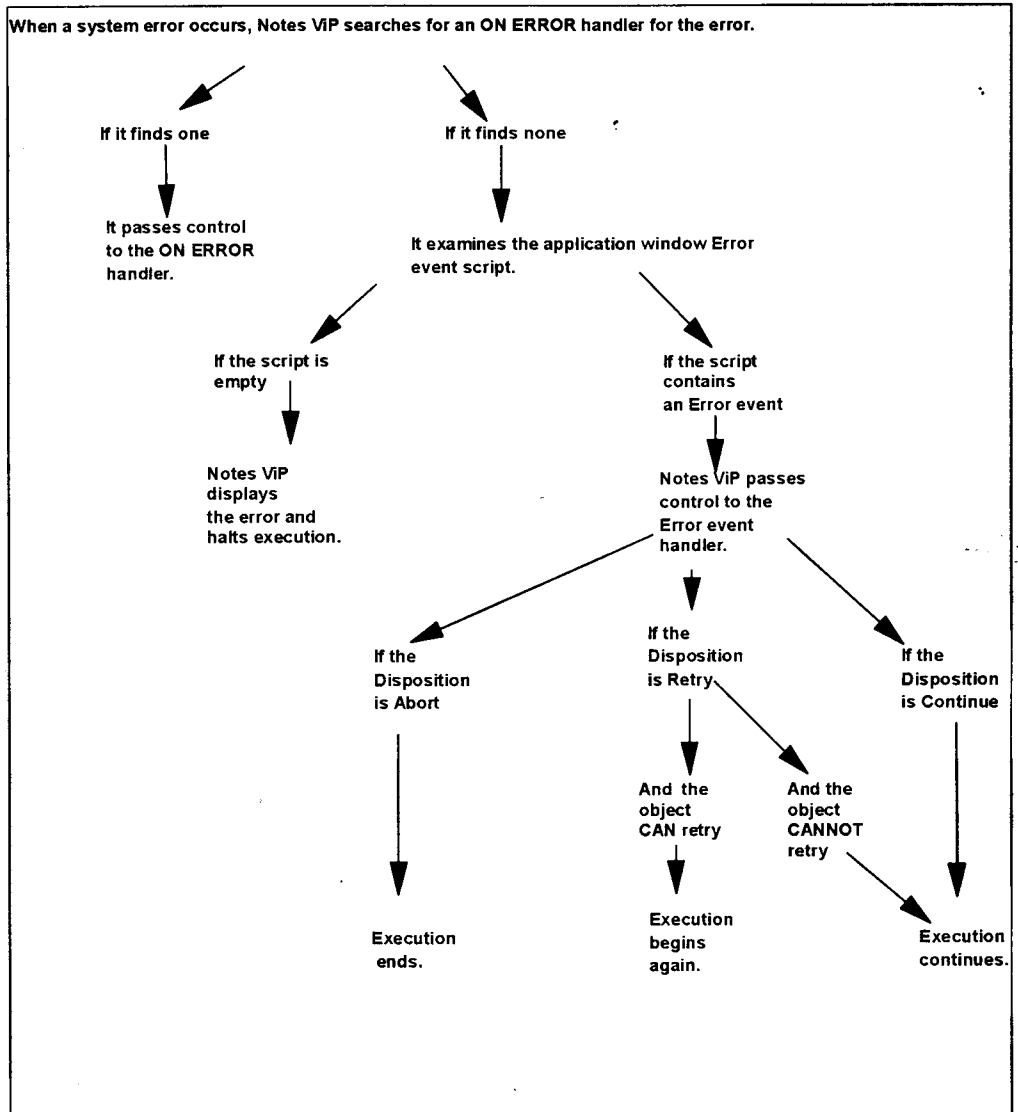


Figure B-1 How error handling works

**Note** Both error-handling tools can trap specific errors by referring to their symbolic error constants, contained in the file VIPERR.LSS. This example shows one situation in which VIPRR.LSS is used.

```
=====
'--This example shows the use of a symbolic error constant. Note that
'--Data1 is a data object.
=====
'--Include the VIPERR.LSS constants file.
#include "viperr.lss"

*****
' This ViP function may return a code indicating that the application
' window containing the data object has been closed.
*****
RetCode% = Data1.ExecuteQuery()
if (RetCode% = VIPERR_APPWIN_CLOSED) then exit function
```

---

## Handling errors with the Error event

To handle errors with the Error event, you must edit the application window's Error event script.

### Adding a script for an Error event

There are two methods you can use to add an Error event script to the application window.

#### Method 1: From the application window desktop

1. Position the cursor away from any objects within the application window.
2. Double-click the application window client area.
3. In the Script Editor window, select Error from the Event drop-down list.

The Script Editor presents an empty window and displays the syntax of the Error event above the script area. You can now enter the text of your script.

## Method 2: From within the Script Editor

1. Select the application window from the Object drop-down list.
2. In the Script Editor window, select Error from the Event drop-down list.

The Script Editor presents an empty window and displays the syntax of the Error event above the script area. You can now enter the text of your script.

## Error event example

The following commented example shows an Error event established for an application containing a data object and a command button.

```
=====
'--This Error event example handles a situation in which a user tries
'--to connect to an unavailable data source. It presents the user with
'--the option to abort, retry, or ignore.
=====

*****
'--Define the variables used with the MESSAGEBOX function.
*****

dim MBBUTTONS%, MBTITLE$, MBBODY$, MBRET%
dim UseObj%

'--Set CR to be a string.
dim CR as string
'--Set CR to a carriage return CHR$(13).
CR = CHR$(13)

'--Assemble the text of the message box title.
MBTITLE$ = "ERREVENT Error Event Script: Runtime Error " +str$(ErrCode)

'--Do not use the ErrObj parameter if it is NOTHING.
'--Note that NOTHING is a LotusScript keyword.
'--If there is no object, the handler cannot get the class and name.
'--Assemble the text of the message box body.
```

```

if (ErrObj is NOTHING) then
MBBody$ = "No object" + CR+CR
*****
'--Add the class and name to the message body.
*****
else
  MBBody$ = "ErrObj={ Class="+ErrObj.Class()+", Name="+ErrObj.Name+" }"
  MBBody$ = MBBody$ +CR+CR
end if
'--Add the error type.
MBBody$ = MBBody$+"Type="+str$(ErrType&)
'--Add the error subtype.
MBBody$ = MBBody$+", Subtype="+str$(ErrSubtype&)
'--Add the error code.
MBBody$ = MBBody$+", Code="+str$(ErrCode&)
'--Add two carriage returns.
MBBody$ = MBBody$ +CR+CR
'--Add the error message.
MBBody$ = MBBody$+ErrMsgage$
'--Add two carriage returns.
MBBody$ = MBBody$ +CR+CR
'--Add the error message to the message body.
MBBody$ = MBBody$+ErrMsgage$

'--Tell the MESSAGEBOX function to put Abort, Retry, and Ignore
'--buttons in the message box.
if (ErrObj is NOTHING) then
  MBButtons = 2
  MBRet% = MESSAGEBOX (MBBody$, MBButtons%, MBTitle$ )

```

```

'--If the user clicks on the Retry button,
'--Notes ViP retries the operation.
if MBRet%=4 then
ErrDisposition% = 1
'--If the user clicks on the Ignore button,
'-Notes ViP ignores the error.
elseif MBRet%=5 then=
ErrDisposition% = 2
*****
'--If the user clicks on the Abort button,
'--Notes ViP displays the error and halts execution.
*****
else
    ErrDisposition% = 0
end if

```

There are four types of errors that can occur in this sample application:

- A UI error when no script is running. A user clicks the Refresh icon to execute a query, but gets an error because the data object is not connected to a database. This error is represented in Figure B-2 by the screen in the top right.
- A script error for the Connection property in a data object (Data1). The user tries to set a property to an invalid value. In this case, the value is too long. This error is represented in Figure B-2 by the Property error button.
- A script error for the ExecuteQuery() method for a data object. An error occurs because Data1 is not connected to a database. This error is represented in Figure B-2 by the Method error button.
- A script error for the OpenAppWindow function. An error occurs because the function does not have an associated object; that is, it references a nonexistent application window. This error is represented in Figure B-2 by the Function error button.

The following screens appear when the user runs the application, then clicks the respective command buttons.

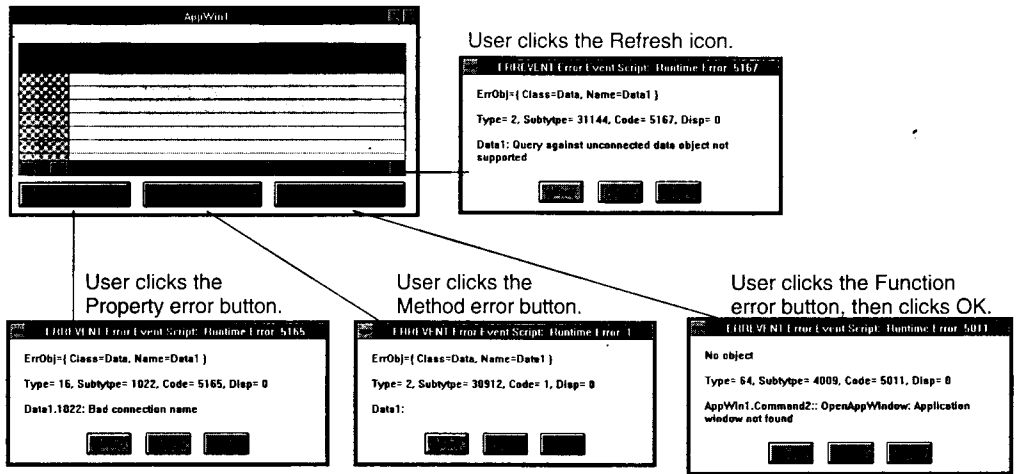


Figure B-2 Sample Error event screens

## Using the Retry disposition on data objects

In situations in which an operation may fail, you can establish an Error event using the Retry disposition parameter (`ErrDisposition% = VIPEDISP_RETRY (1)`). Since Notes ViP retries the operation if requested, you can use the Retry disposition parameter to correct failed insertions, updates, and deletions (initiated either through the UI or scripts). For example, you can use this parameter to provide these responses to an error event:

- Requery the same or another database
- Send vendor commands
- Connect to other servers
- Wait a specified interval or number of retries for a given data operation

## Limitation

The Error event only traps Notes ViP errors, not LotusScript errors (such as divide by 0).

---

## Handling errors with the ON ERROR statement

Any ON ERROR statement in an event script takes precedence over an Error event defined in the application window.

For example, if you have established an Error event to trap a Click event error, but that event already has an ON ERROR handler established, the ON ERROR handler traps the error.

### Establishing an ON ERROR handler in an event script

You can establish an ON ERROR handler for any event associated with any object in your application. However, bear in mind that an ON ERROR handler established for one event cannot apply to any other event (even for the same object).

#### To establish an ON ERROR handler

1. Position the cursor on the object for which you want to establish an ON ERROR handler.
2. Double-click the object. The Script Editor for that object appears.
3. From the Event drop-down list, select the event for which you want to establish an ON ERROR handler. The Script Editor displays the event screen. You can now enter the text of your script.

### Example 1

In this example, two ON ERROR statements handle Notes ViP errors in the context in which they occur.

```
=====
'--This ON ERROR example assumes that Data1 is a data object that has
'--columns called Name and Ext. The User wants to change the value of
'--Smiths's Ext to "1234."
=====

'--Find row for Smith.
'--Handle any error finding the old row.
ON ERROR goto ErrorHandler1
RetCode% = Data1.CellSetQuery(1,1,"Smith")
RetCode% = Data1.ExecuteQuery()
```

```

'--Handle any error updating the row.
ON ERROR goto ErrorHandler2
RetCode% = Data1.RowOpen(1)
RetCode% = Data1.CellSetValue(1, 2, "1234")
RetCode% = Data1.ExecuteUpdate()

'--Restore any previous error handler.
ON ERROR goto 0

...
exit function

ErrorHandler1:
print "Cannot find the row for Smith."
exit function

ErrorHandler2:
print "Cannot update the row for Smith."
exit function

```

## Example 2

In this example, an ON ERROR handler is established for a Click event script for the Command1 object. The application contains a check box and a command button.

```

=====
'--This example shows how an ON ERROR statement traps errors so that
'--the program can continue. Unhandled errors stop the program
'--execution.
=====

*****
'--Define variables for using the MESSAGEBOX function (built into
'--LotusScript).
*****
dim MBCode as integer
dim MBMessage as string

```

```

'--Controls which buttons appear in message box.
dim MBButtons as integer
dim MBTitle as string

'--Define variables used to generate a run-time error.
dim x, y as double

'--Establish an ON ERROR handler only if the user clicks on the check
'--box.
if Check1.State then ON ERROR goto Error_Handler

*****
'--This section of code forces an error.
*****
x = 0
'--This line produces a divide by zero error.
y = 1/x

'--When this script runs, and the user clicks OK, the following
'--print statement appears. However, if the user clicks Cancel,
'--the print statement does NOT appear.
print "y="+str$(y)
exit function

*****
'--This is the error handler called by the ON ERROR statement.
*****
Error_handler:
'--Display a message box showing the error number (ERR) and the error
'--text (ERROR$). Ask the user if they wish to continue.
'--NOTE: chr$(10) is a carriage return.
'--Add an error number.
MBMessage$ = "Error number = "+str$(ERR)

```

```

'--Add two carriage returns.
MBMessage$ = MBMessage$+CHR$(10)+CHR$(10)
'--Add the error text.
MBMessage$ = MBMessage$+ERROR$
'--Add two more carriage returns.
MBMessage$ = MBMessage$+CHR$(10)+CHR$(10)
'--Add a message to press OK to continue.
MBMessage$ = MBMessage$+"Press OK to continue"

'--1 is a MESSAGEBOX argument meaning MB_OKCANCEL
'--In other words, the message box contains OK and Cancel buttons.
MBButtons = 1
MBTitle = "Sample Error Handler"
MBCode = MsgBox(MBMessage, MBButtons, MBTitle)
'--If the user clicks OK, continue execution.
if (MBCode = 1) then resume next
'--If the user clicks Cancel, end this script.
if (MBCode = 2) then exit function

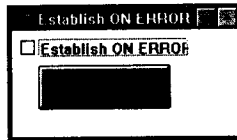
```

If the user selects the check box labeled Establish ON ERROR, then clicks the Generate Error command button, a message box appears, displaying information about the error. If the user clicks OK, another message box appears with additional information. This sequence is displayed on the right side of Figure B-3.

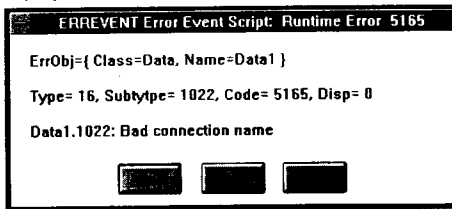
If the user leaves the box unchecked and clicks Generate Error, the ON ERROR handler is not invoked, a system error appears, and the application terminates. This sequence is displayed on the left side of Figure B-3.

Figure B-3 shows a graphical representation of the ON ERROR screens that Notes ViP displays for this sample application.

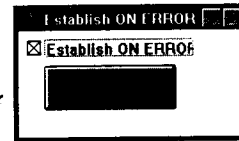
If the user does not check the check box and clicks the Generate error button ...



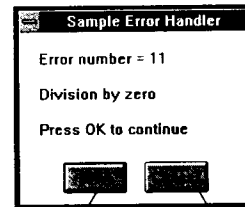
... the ON ERROR handler is not invoked. Notes ViP displays a run-time error and halts execution.



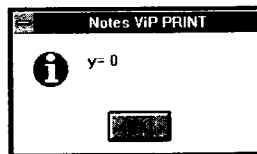
If the user checks the check box and clicks the Generate error button ...



... the ON ERROR handler is invoked.



If the user clicks OK ...



... the program continues, displaying more information.

If the user clicks Cancel, the application terminates.

Figure B-3 Sample ON ERROR example

## Limitations

- ON ERROR handlers only work in the script in which they are established. They do not affect any other script.
- You cannot use ON ERROR handlers in the application's Public script or any application window (Declarations) script.
- While you can use an ON ERROR handler in an application window's Init event, the handler only traps errors that occur during the execution of that Init event.