

A zero value of *refNum* is used in conjunction with Program-to-Program Communication (PPC), Apple events (*Macintosh*) or **ActiveX Automation** (*Windows*). Data that would normally be written to a file is appended to the PPC, Apple event or ActiveX Automation result packet.

The `fprintf` operation supports numeric (real only) and string fields from structures. All other structures field types cause a compile error.

### Output to Standard Streams

If *refNum* is -1, Igor prints to the standard output stream (stdout).

If *refNum* is -2, Igor prints to the standard error stream (stderr).

This feature was added in Igor Pro 8.00. It is useful primarily when you launch Igor from a custom application. The application may be able to capture Igor's standard stream output.

On Macintosh, output printed to either of the standard streams is typically visible only if Igor is started from the Terminal. On Windows, stdout and stderr output is not visible even if Igor is started from the command line.

### See Also

The **printf** operation for complete format and parameter descriptions. The **Open** operation and **Creating Formatted Text** on page IV-259.

## FReadLine

**FReadLine** [/N /ENCG=*textEncoding* /T] *refNum*, *stringVarName*

The FReadLine operation reads bytes from a file into the named string variable. The read starts at the current file position and continues until a terminator character is read, the end of the file is reached, or the maximum number of bytes is read.

### Parameters

*refNum* is a file reference number from the **Open** operation used to create the file.

### Flags

/N=*n*                      Specifies the maximum number of bytes to read.  
FReadLine can read lines text of length up to about 2 billion bytes.

/ENCG=*textEncoding*

/ENCG={*textEncoding*, *tecOptions* [,*mapErrorMode*]}

Specifies the text encoding of the plain text file being loaded.

This flag was added in Igor Pro 9.00.

See **Text Encoding Names and Codes** on page III-490 for a list of accepted values for *textEncoding*. The 32-bit Unicode text encodings UTF-32BE and UTF-32LE are currently not supported.

If you omit /ENCG, *textEncoding* defaults to binary (255) and does no text encoding conversion but does look for and skip a UTF-8 byte order mark.

For most purposes the default values for *tecOptions* (3) and *mapErrorMode* (1) are fine and you can use /ENCG=*textEncoding* instead of /ENCG={*textEncoding*, *tecOptions* [,*mapErrorMode*]}.  
[*mapErrorMode*].

*tecOptions* is an optional bitwise parameter that controls text encoding conversion. See **Setting Bit Parameters** on page IV-12 for details about bit settings.

*tecOptions* is defined as follows:

- Bit 0: If cleared, the presence of null bytes causes FReadLine to consider the text invalid in all byte-oriented text encodings.  
If set (default), null bytes are allowed in byte-oriented text encodings.
- Bit 1: If cleared FReadLine does not validate text if the specified text encoding is UTF-8.  
If set (default), FReadLine validates text even if the text encoding is UTF-8.
- Bit 2: If cleared (default) FReadLine validates the text in the specified text encoding except that validation for UTF-8 is skipped if bit 1 is cleared.  
If set, FReadLine assumes that the text is valid in the specified text encoding and does not validate it. Setting this bit makes FReadLine slightly faster but it is usually inconsequential.

UTF-8 because it is not valid in the input text encoding. *mapErrorMode* is one of these values:

- 1: FReadLine generates an error.
- 2: Return a substitute character for the unmappable character. The substitute character for Unicode is the Unicode replacement character, U+FFFD. For most non-Unicode text encodings it is either control-Z or a question mark.
- 3: Skip unmappable input character.
- 4: Return an escape sequence representing the unmappable code point.

If *mapErrorMode* is 2, 3 or 4, FReadLine does not return an error in the event of an unmappable character.

See the discussion of *mapErrorMode* and the examples for **ConvertTextEncoding** for further discussion.

See **FReadLine and Byte Order Marks** on page V-263 below for further discussion.

*/T=termcharStr* Specifies the terminator character.

*/T=(num2char (13) )* specifies carriage return (CR, ASCII code 13).

*/T=(num2char (10) )* specifies linefeed (LF, ASCII code 10).

*/T=";"* specifies the terminator as a semicolon.

*/T=""* specifies the terminator as null (ASCII code 0).

See **Details** for default behavior regarding the terminator.

### Details

If */N* is omitted, there is no maximum number of bytes to read. When reading lines of text from a normal text file, you will not need to use */N*. It may be of use in specialized cases, such as reading text embedded in a binary file.

FReadLine can read lines text of length up to about 2 billion bytes.

If */T* is omitted, FReadLine will terminate on any of the following: CR, LF, CRLF, LFCR. (Most Macintosh files use CR. Most Windows files use CRLF. Most UNIX files use LF. LFCR is an invalid terminator but some buggy programs generate files that use it.) FReadLine reads whichever of these appears in the file, terminates the read, and returns just a CR in the output string. This default behavior transparently handles files that use CR, LF, CRLF, or LFCR as the terminator and will be suitable for most cases.

If you use the */T* flag, then FReadLine will terminate on the specified character only and will return the specified character in the output string.

Once you have read all of the bytes in the file, FReadLine will return zero bytes via *stringVarName*. The example below illustrates testing for this.

## FReadLine and Text Encodings

For background information on text encodings, see **Text Encodings** on page III-459.

In Igor Pro 9.00 and later, you can specify the text encoding of the file you are reading using the /ENCG flag.

If you omit /ENCG, FReadLine treats the data as binary and does no text encoding conversion. However, it does skip a UTF-8 byte order mark at the start of the file if present. Treating the data as binary works if the data you are loading is pure ASCII or is valid UTF-8 text.

If you specify a text encoding using /ENCG, the FReadLine converts the text from the text encoding you specify into UTF-8 for internal storage. See **Text Encoding Names and Codes** on page III-490 for a list of accepted values for *textEncoding*. The 32-bit Unicode text encodings UTF-32BE and UTF-32LE are currently not supported.

In general it is not possible to accurately determine the text encoding of a plain text file programmatically so you need to know it *a priori*.

If the text is Unicode and includes a byte order mark then you can programmatically determine the text encoding by examining the byte order mark. You can do this using FBinRead to examine the first bytes of the file:

0xEF, 0xBB, 0xBF	UTF-8 byte order mark, <i>textEncoding</i> =1
0xFE, 0xFF	UTF-16BE byte order mark, <i>textEncoding</i> =100
0xFF, 0xFE	UTF-16LE byte order mark, <i>textEncoding</i> =101

If the file starts with a pattern other than these then you will need *a priori* knowledge or will need to make an assumption about the text encoding.

## FReadLine and Byte Order Marks

For background information on byte order marks, see **Byte Order Marks** on page III-471.

If you want to check for the presence of a byte order mark, use **FBinRead** instead of FReadLine.

The default text encoding, used if you omit /ENCG, is binary (255). In this case, FReadLine does no text encoding conversion. However, it does look for a UTF-8 byte order mark and skips it if present.

If you specify UTF-8 or UTF-16 as the text encoding using the /ENCG flag, and if the file data begins with a byte order mark, then FReadLine skips it.

## Example

```
Function PrintAllLinesInFile()
    Variable refNum
    Open/R refNum as ""           // Display dialog
    if (refNum == 0)
        return -1                // User canceled
    endif

    Variable lineNumber, len
    String buffer
    lineNumber = 0
    do
        FReadLine refNum, buffer
        len = strlen(buffer)
        if (len == 0)
            break                 // No more lines to be read
        endif
        Printf "Line number %d: %s", lineNumber, buffer
        if (CmpStr(buffer[len-1], "\r") != 0) // Last line has no CR ?
            Printf "\r"
        endif
        lineNumber += 1
    while (1)

    Close refNum
    return 0
End
```

## See Also

The **Open** and **FBinRead** operations.