How to read and write text files

One of the most often needed ways to import/export data is the plain text file. This tutorial shows different ways of creating, reading and writing in plain text files, and also how such files can be navigated.

saving; opening; files; text; CreateFileForWriting(); OpenFileForReading(); OpenFileForWriting(); WriteFile(); ReadFile(); OpenFileForReadingAndWriting(); ReadFileLine(); CloseFile(); GetFileSize(); StreamGetSize(); StreamSetSize(); StreamSetPos(); StreamGetPos(); NewStreamFromFileReference(); StreamWriteAsText(); StreamReadAsText(); StreamReadTextLine();

There are different possibilities to read from or write to a text file.

Variant A - Accessing text files by low-level API commands

The first, and maybe more intuitive way is to use read and write commands which are similar to the results() command but access a file instead of the results window. In order to do so, a file has first to be created or opened by one of the following commands CreateFileForWriting(), OpenFileForReading(), OpenFileForWriting(), OpenFileForReadingAndWriting(). As the name of the commands suggests, a file can be opened for either just reading, or for just writing, or to allow both at the same time. The only parameter of these commands is the path to the file, which can either be absolute or relative. When successfully executed, these commands return a number further called *fileID*. As long as file is opened, this *fileID* number refers to the file and is used to address the file in other commands. It is important to note that an *opened* file stays open even if the script is finished. This is also valid, if the script breaks during execution. *Opened* files can not be accessed by any other process, and it is thus necessary to make sure that each *opened* file is closed within the script, using the CloseFile() command. (All *opened* files are closed automatically when DigitalMicrograph is closed.)

The first example script demonstrates how a text file is created at a selected position. Some text is written using the WriteFile() command. Each new text is appended automatically at the end of the file.

```
String   filename, text
Number   fileID
If (!SaveAsDialog("Save text file as", GetApplicationDirectory(2,0) + "myText.txt", filename)) Exit(0)
Result("\n Selected file path:"+filename)
fileID = CreateFileForWriting(filename)
text = "Some typical text string with TABULATOR \t and LINEBREAKS \n as used in DigitalMicrograph"
WriteFile(fileID, "Writing some text at the beginning.")
WriteFile(fileID, "Text is continued without LINEBREAK! Add a LINEBREAK:\n")
WriteFile(fileID, "The command WriteFile() is used as the command Result() and allows a mixture of")
WriteFile(fileID, "direct strings,\n number variables:\n" + fileID + ",\n or string variables:\n" + text)
CloseFile(fileID)
```

In the next example, we open the same file and read the text and print it into the result window. Please note the NULL in the command OpenDialog() which is needed in this syntax of the command allowing a modified dialog text and a default value. First, we make use of the ReadFile() command which reads a specified number of characters into a string. Note that a LINEBREAK or a TABULATOR is also considered *one* character. Note further, that the command does not check the size of the file. If more characters should be read than are present in the file, the script will crash with an error message (and the file stays *open*!).

```
String   filename, text
Number   fileID, nr
If (!OpenDialog(NULL, "Reading from text file", GetApplicationDirectory(2,0) + "myText.txt", filename)) Exit(0)
Result("\n Selected file path:"+filename)
If (!GetNumber("How many characters should be read?", 2 , nr)) Exit(0)
fileID = OpenFileForReading(filename)
text = ReadFile(fileID, nr)
Result("\n Reading  " + nr + " characters:\n" + text)
If (!GetNumber("How many characters should now be read?", 2 , nr)) Exit(0)
text = ReadFile(fileID, nr)
Result("\n Reading  the next " + nr + " characters:\n" + text)
CloseFile(fileID)
```

The following example uses the ReadFileLine() command which reads a whole string until a LINEBREAK character or the ENDOFFILE is reached. The function returns 1 on success and 0 on failure (e.g. no more lines), which allows it to be used in a while loop and avoids errors on exceeding the file size. Note that the LINEBREAK character is part of the returned string.

```
String   filename, text
Number   fileID
If (!OpenDialog(NULL, "Reading from text file", GetApplicationDirectory(2,0) + "myText.txt", filename)) Exit(0)
```

```
fileID = OpenFileForReading(filename)
While(ReadFileLine(fileID, text)) Result("\n next line ["+text+"]")
CloseFile(fileID)
```

We have now examples for creating a text file and reading it. The next example shows how a text file can be opened for appending some text. Whenever a text file is opened, the "pointer" or "cursor" is at the start of the file. Thus, each WriteFile() command would overwrite existing data. It is necessary to first move the pointer by reading from the file. The command GetFileSize() returns the size of a file in bytes. As in pure text files each character is stored by one byte-sized number (0-255), the file size in bytes is at the same time the number of characters in the file. Thus, reading 'file size' characters will put the file pointer to the end of the file, where we then can write the new text. Note, that we need to open the filer for both reading and writing in this case.

```
String   filename, text
Number   fileID, nr
If (!OpenDialog(NULL, "Appending to text file", GetApplicationDirectory(2,0) + "myText.txt", filename)) Exit(0)
fileID = OpenFileForReadingAndWriting(filename)
Result("\n File size (bytes): " + fileID.GetFileSize() )
ReadFile(fileID, fileID.GetFileSize())
WriteFile(fileID, "\nThis text is added!")
CloseFile(fileID)
```

Variant B - Accessing text files by streaming commands

Besides the read and write commands demonstrated so far, there is a more versatile method of dealing with text files: streaming. Streaming text files works similar to streaming binary files as described in the tutorial on binary data import and export (see page 11). The main advantage of a streamed text file is that the pointer can be easily placed at any position of the text file, and that the file is automatically closed if the stream runs 'out of scope'.
The following script opens a text file and creates a stream from it by using NewStreamFromFileReference(). The second parameter defines, if the file should be automatically closed if the stream runs out of scope. Next, the script reads its size by StreamGetSize(). The pointer is then placed at the last position by StreamSetPos(). The second parameter gives an offset (in bytes) to a reference point defined by the first parameter:

#	reference point
0	start of file
1	current cursor position
2	end of file

With the cursor being at the end position, some text is written with the command StreamWriteAsText(). Besides the fileID and the text, a number parameter defines the text encoding:

#	encoding	explanation
0	local	Local character set. Determined by settings in Windows.
1	western	DigitalMicrograph (US) character set.
2	unicode	Unicode character set.

After writing the text, the new size of the stream is given before closing the file.

```
Object file_stream
String filename, text
Number fileID
If (!OpenDialog(NULL, "Appending to text file", GetApplicationDirectory(2,0) + "myText.txt", filename)) Exit(0)
fileID = OpenFileForReadingAndWriting(filename)
file_stream = NewStreamFromFileReference(fileID, 1)
Result("\n size of file before adding:" + file_stream.StreamGetSize())
file_stream.StreamSetPos(2,0)
file_stream.StreamWriteAsText(0, "\n Adding some new text by streaming.\nNew line.")
Result("\n size of file after adding:" + file_stream.StreamGetSize())
CloseFile(fileID)
```

In the next example, navigating, reading and writing within a stream is all used together to find and replace all decimal points (.) in a text file by commas (,). This can for example be useful when working with German language programs where decimals are expressed by commas instead of points. The script loops through the file by comparing the current pointer position – gained with the command StreamGetPos() – with the size of the file. StreamReadAsText() is used to read a defined numbers of characters (one in the example) of given encoding into a string variable. If a decimal point is found, StreamSetPos() is used to set the cursor back by one character, so that the StreamWriteAsText() command overwrites the located decimal point with the comma. After replacing all points with commas, the pointer is placed at the beginning of the file and the file is read line-by-line using the command StreamReadTextLine() with local encoding. The command returns 1 on success and 0 on failure. Thus, it can be used within a while loop. Once the file ends, the command can no longer read a line and returns 0, which ends the while loop. The same is repeated with a variant of the StreamReadTextLine() command offering an additional parameter (2nd position) which determines whether or not the final LINEBREAK at the end of a read line is truncated.

```
Object file_stream
String filename, text
Number fileID, streamsize
If (!OpenDialog(NULL, "Replace . by ,", GetApplicationDirectory(2,0) + "myText.txt", filename)) Exit(0)
fileID = OpenFileForReadingAndWriting(filename)
file_stream = NewStreamFromFileReference(fileID, 1)
streamsize = file_stream.StreamGetSize()
While( file_stream.StreamGetPos() < streamsize )
   {
   text = file_stream.StreamReadAsText(0, 1)
   If (text == ".")
      {
      file_stream.StreamSetPos(1, -1)
      file_stream.StreamWriteAsText(0, ",")
      }
   }
file_stream.StreamSetPos(0,0)
Result("\n Change filed (with LINEBREAKS):\n")
While( file_stream.StreamReadTextLine(0,text) ) Result(text)
file_stream.StreamSetPos(0,0)
Result("\n Change filed (without LINEBREAKS):\n")
While( file_stream.StreamReadTextLine(0,0,text) ) Result(text)
CloseFile(fileID)
```

In the last part of this tutorial we deal with inserting and removing text from a text file. In a standard stream you can move your pointer and change the information at the indicated position. However, you can not clip or insert data directly, as this changes the position of the data behind the position. Or, in another point of view, *all* data at positions behind the current one is changed, and the file is increased/decreased in size. Thus, inserting or removing text in the middle of a text file involved rewriting *all* of the data to the end and can be quite time consuming and inefficient. Especially, if *many* such changes have to be made. In such a case it can be more efficient to work with two different text files, were the new file is built by copying parts of the old file plus additional text.

The following function will directly insert a text at a given position in a file. Note that this time the stream is created without closing the file automatically on running ‘out of scope’!

```
/* This function inserts a text at given position in a file. The file has to be open for reading/writing. */
Void TXTFile_Insert(Number fileID, Number pos, String text)
   {
   Object file_stream
   String copystring
   Number filesize
   file_stream = NewStreamFromFileReference(fileID, 0)                // Note: File not automatically closed!
   filesize = file_stream.StreamGetSize()
   If ( pos > filesize ) Return
   If ( pos < 0 ) Return
   file_stream.StreamSetPos(0 , pos)                                                     // jump to position
   copystring = file_stream.StreamReadAsText(0, filesize-pos)                           // copy rest of file
   file_stream.StreamSetPos(0 , pos)                                                     // jump to position
   file_stream.StreamWriteAsText(0 , text)                                                 // inset the text
   file_stream.StreamWriteAsText(0 , copystring)                                       // paste rest of file
   Return
   }

String   filename, text
Number   fileID, pos
If (!OpenDialog(NULL, "Inserting in a text file", GetApplicationDirectory(2,0) + "myText.txt", filename)) Exit(0)
fileID = OpenFileForReadingAndWriting(filename)
while (GetString("Enter new text", "-INSERTED-", text))
   {
   If (!GetNumber("Insert at position [0-" + fileID.GetFileSize() + "]", 0, pos)) Break
   TXTFile_Insert(fileID, pos, text)
   }
CloseFile(fileID)
```

The function below removes a text of given length from a text file. After cutting the text out, the command StreamSetSize() is used to define the new size of the file.

```
/* This function removes a text of given length and position. The file has to be open for reading/writing. */
Void TXTFile_Remove(Number fileID, Number pos, Number amount)
   {
   Object file_stream
   String copystring
   Number filesize
   file_stream = NewStreamFromFileReference(fileID, 0)                     // Note: File not automatically closed!
   filesize = file_stream.StreamGetSize()
   If ( (pos+amount) > filesize ) Return
   If ( pos < 0 ) Return
   file_stream.StreamSetPos(0 , pos+amount)                                                   // jump to position
   copystring = file_stream.StreamReadAsText(0, filesize-pos-amount)                          // copy rest of file
   file_stream.StreamSetPos(0 , pos)                                                          // jump to position
   file_stream.StreamWriteAsText(0 , copystring)                                             // paste rest of file
   file_stream.StreamSetSize(file_stream.StreamGetPos())               // truncate the file at the current position
   Return
   }

String   filename
Number   fileID, pos, amount
If (!OpenDialog(NULL, "Deleting from a text file", GetApplicationDirectory(2,0) + "myText.txt", filename)) Exit(0)
fileID = OpenFileForReadingAndWriting(filename)
If (!GetNumber("Cut at position [0-" + fileID.GetFileSize() + "]", 0, pos)) Exit(0)
If (!GetNumber("Cut how many characters [0-" + (fileID.GetFileSize()-pos) + "]", 0, amount)) Exit(0)
TXTFile_Remove(fileID, pos, amount)
CloseFile(fileID)
```