## How to import and export data in binary format

*There are several script commands for "simple" file handling. However, importing/exporting data is often an essential step when dealing with different programs, and sometimes these 'simple' methods are not enough, and data has to be transferred as 'raw data' which is often also called 'binary data' and stores pixel values (numbers) one after the other. This is often the least common denominator regarding data exchange of different programs.*

saving; opening; files; image; streaming; OpenFileForReading(); CreateFileForWriting(); CloseFile(); StreamSetPos(); StreamGetSize(); ImageReadImageDataFromStream(); ImageWriteImageDataToStream(); NewStreamFromFileReference();

Saving data in binary "raw" format can be easily achieved by the SaveAsRawData() command. Importing, however, is more difficult. The main point is, that the file itself does not contain information on the image size or data type. This information thus has to be provided 'from outside' as user input, and it has to match with the real data in the file. Usually, raw data is imported in two steps: First, an image of proper dimension and type (=number format) is created. Second, data is streamed from a file into that image. The following script demonstrates these two steps. It assumes that the imported image contains *floating point* numbers (*4-byte real images* in DM). After asking for the filename, the user has to enter the image dimensions (x/y). Then the file is opened for reading, and a file_ID number is returned for further addressing that file. From the file, a file-stream object is created. The second parameter determines that the file will be automatically closed, if the stream runs out of scope (i.e. the stream tries to access a part of the file lying outside the size of the file). Next, an image of proper size and type is created, and the data is read from the stream into the image. (The size and type of the image determine the memory-size which is read from the stream. e.g. a 10 x 10 pixel real images has 10 x 10 x 4 = 400 bytes). The third parameter of the ImageReadDataFromStream() command defines the *endianness*[1] used for the data, and is most often 0.  Before showing the image, the file itself is closed by CloseFile().

```
Object file_stream
Image img
String filename
Number size_x, size_y, file_ID
If (!OpenDialog(filename)) Exit(0)
If (!GetNumber("Image size x (pixels)", 0, size_x)) Exit(0)
If (!GetNumber("Image size y (pixels)", 0, size_y)) Exit(0)
file_ID = OpenFileForReading(filename)
file_stream = NewStreamFromFileReference(file_ID, 1)
img := RealImage("Imported", 4, size_x, size_y)
ImageReadImageDataFromStream(img, file_stream, 0)
CloseFile(file_ID)
img.ShowImage()
```

For the sake of completeness, the following script stores an image as a file stream:

```
Object file_stream
Image img
String filename
Number file_ID
img.GetFrontImage()
If (!img.ImageIsValid()) Exit(0)
If (!SaveAsDialog("Save image as binary data", GetApplicationDirectory(2,0), filename)) Exit(0)
file_ID = CreateFileForWriting(filename)
file_stream = NewStreamFromFileReference(file_ID, 1)
ImageWriteImageDataToStream(img, file_stream, 0)
CloseFile(file_ID)
```

However, as stated before, the same can be achieved more easily with the SaveAsRawData() command:

```
Image    img
String   filename
img.GetFrontImage()
If (!img.ImageIsValid()) Exit(0)
If (!SaveAsDialog("Save image as binary data", GetApplicationDirectory(2,0), filename)) Exit(0)
img.SaveAsRawData(filename)
```

[1] The *endianess* defines the bit-order in which a number is stored. 123456789 = 07|5B|CD|15   or   15|CD|5B|07

Binary data stored by other software often has some ‘header’ in the file before the actual data starts. In this case it is necessary to skip these first few bytes prior to importing the data. The following script shows how this is done using the StreamSetPos() command. The size of the header (in bytes) has to be known and entered by the user. The first parameter of the StreamSetPos() command defines the origin of the stream: 0 = start, 1 = current position, 2 = end.

```
Object file_stream
Image img
String filename
Number size_x, size_y, file_ID, header_size
If (!OpenDialog(filename)) Exit(0)
If (!GetNumber("Image size x (pixels)", 0, size_x)) Exit(0)
If (!GetNumber("Image size y (pixels)", 0, size_y)) Exit(0)
If (!GetNumber("File header size (byte)", 0, header_size)) Exit(0)
file_ID = OpenFileForReading(filename)
file_stream = NewStreamFromFileReference(file_ID, 1)
img := RealImage("Imported", 4, size_x, size_y)
file_stream.StreamSetPos(0, header_size)
ImageReadImageDataFromStream(img, file_stream, 0)
CloseFile(file_ID)
img.ShowImage()
```

If the header size is unknown, but the type and size of data is known, then the header can also be calculated from the size of the stream, as shown below.

```
Object file_stream
Image img
String filename
Number size_x, size_y, file_ID, header_size, datatype_size
If (!OpenDialog(filename)) Exit(0)
If (!GetNumber("Image size x (pixels)", 0, size_x)) Exit(0)
If (!GetNumber("Image size y (pixels)", 0, size_y)) Exit(0)
If (!GetNumber("Enter size of data type (byte)", 0, datatype_size)) Exit(0)
file_ID = OpenFileForReading(filename)
file_stream = NewStreamFromFileReference(file_ID, 1)
img := RealImage("Imported", 4, size_x, size_y)
header_size = file_stream.StreamGetSize() - size_x * size_y * datatype_size
Result("\n Header (bytes): " + header_size)
file_stream.StreamSetPos(0, header_size)
ImageReadImageDataFromStream(img, file_stream, 0)
CloseFile(file_ID)
img.ShowImage()
```