How to write a Plugin

Preliminaries

This document assumes intermediate to advanced knowledge of the Lua programming language.

There are vast numbers of point and elevation data file types and to support them all would be a large undertaking so a Plugin mechanism has been developed to make the Regular Grid Importer capable of supporting 3rd party file types in an extensible fashion.

Setup

To write a Plugin for a given file format create a file in the Plugins directory called YOUR_EXTENSION.luax. If, for example, you were writing a Plugin for the XYZ file type you would call the file XYZ.luax

Plugin structure

The Plugin should begin with the line

local YOUR_EXTENSION = {}

and should include three functions

function YOUR_EXTENSION.Read(filename, point_callback, progress_callback, log_warning)

function YOUR_EXTENSION.SupportsExtension(ext)

function YOUR_EXTENSION.HasUnits()

and it should end with the lines

package.loaded.YOUR_EXTENSION = YOUR_EXTENSION

return YOUR_EXTENSION

Read function

The read function should be structured as follows

function YOUR_EXTENSION.Read(filename, point_callback, progress_callback, log_warning)

The filename is the name of the file you should attempt to read.

The point_callback is used to tell the main script you have found a 3d point, e.g,

while YourMorePoints() do

local x, y, z = YourReadPoint()

if not YourTransparent(z) then point_callback(x, y, z) end

end

Please note you should not use the point_callback for z values that represent transparency in your data format.

The progress_callback is used to update the host application's progress bar to indicate to the user how far the Plugin has progressed in reading its file. It takes a floating point progress value between 0 and 1. You should not call this function too often but call it once for every, say, 100 or 1000 points.

```
if point_no % 1000 == 0 then

progress_callback(point_no / point_count)

end
```

The log_warning is used to indicate that the Plugin encountered a non -fatal error while reading the file. If one or more warnings are reported a dialog will be presented to the user listing all the issues. It takes a single string argument

```
log_warning("Unable to read point on line 42")
```

The function should return true if the read was successful and false if a serious error occurred.

SupportsExtension function

The function should be structured as follows

```
function YOUR_EXTENSION.SupportsExtension(ext)
```

The ext argument is an uppercase string.

The function should return true if the passed extension is supported and false if not.

```
function YOUR_EXTENSION.SupportsExtension(ext)

return ext == "YOUR_EXTENSION"

end
```

HasUnits function

```
function YOUR_EXTENSION.HasUnits()
```

This method should return true if the data has physical units (inches or MM). Elevation data formats frequently don't have meaningful dimensions and should return false.

Guidelines

Any functions you don't intend to make visible to the entire program as whole should be prefixed with local, e.g.

```
local function YourReadPoint(line)

...

end
```