Selafin files

OGR supports reading from 2D Selafin/Seraphin files. Selafin/Seraphin is the generic output and input format of geographical files in the open-source Telemac hydraulic model. The file format is suited to the description of numerical attributes for a set of point features at different time steps. Those features usually correspond to the nodes in a finite-element model. The file also holds a connectivity table which describes the elements formed by those nodes and which can also be read by the driver.

The driver supports the use of VSI virtual files as Selafin datasources.

The driver offers full read-write support on Selafin files. However, due to the particular nature of Selafin files where element (polygon) features and node (point) features are closely related, writing on Selafin layers can lead to counter-intuitive results. In a general way, writing on any layer of a Selafin data-source will cause side effects on all the other layers. Also, it is very important **not to open the same datasource more than once in update mode**. Having two processes write at the same time on a single datasource can lead to irreversible data corruption. The driver issues a warning each time a datasource is opened in update mode.

Magic bytes

There is no generic extension to Selafin files. The adequate format is tested by looking at a dozen of magic bytes at the beginning of the file:

- The first four bytes of the file should contain the values (in hexadecimal): 00 00 00 50. This actually indicates the start of a string of length 80 in the file.
- At position 84 in the file, the eight next bytes should read (in hexadecimal): 00 00 00 50 00 00 00 04.

Files which match those two criteria are considered to be Selafin files and the driver will report it has opened them successfully.

Format

Selafin format is designed to hold data structures in a portable and compact way, and to allow efficient random access to the data. To this purpose, Selafin files are binary files with a generic structure.

Elements

Selafin files are made of the juxtaposition of elements. Elements have one of the following types:

- integer,
- string,
- floating point values,
- arrays of integers,
- arrays of floating point values.

Element	Internal representation	Comments
Integer	a b c d	Integers are stored on 4 bytes in big-endian format (most significant byte first). The value of the integer is $2^{24}.a+2^{16}.b+2^{8}.c+d$.
Floating point	a b c d	Floating point values are stored on 4 bytes in IEEE 754 format and under big-endian convention (most significant byte first). Endianness is detected at run time only once when the first floating point value is read.
String		Strings are stored in three parts:

	Length 1 2 3 4 5 ... Length	• an integer holding the length (in characters) of the string, over 4 bytes; • the sequence of characters of the string, each character on one byte; • the same integer with the length of the string repeated
Array of integers	Length 1 2 3 ... Length	Arrays of integers are stored in three parts: • an integer holding the length (in bytes, thus 4 times the number of elements) of the array, over 4 bytes; • the sequence of integers in the array, each integer on 4 bytes as described earlier; • the same integer with the length of the array repeated
Array of floating point values	Length 1 2 3 ... Length	Arrays of floating point values are stored in three parts: • an integer holding the length (in bytes, thus 4 times the number of elements) of the array, over 4 bytes; • the sequence of floating point values in the array, each one on 4 bytes as described earlier; • the same integer with the length of the array repeated

Full structure

The header of a Selafin file holds the following elements in that exact order:

- a *string* of 80 characters with the title of the study; the last 8 characters shall be "SERAPHIN" or "SERAFIN" or "SERAFIND";
- an *array of integers* of exactly 2 elements, the first one being the number of variables (attributes) *nVar*, and the second is ignored;
- *nVar strings* with the names of the variables, each one with length 32;
- an *array of integers* of exactly 10 elements:
  - the third element is the x-coordinate of the origin of the model;
  - the fourth element is the y-coordinate of the origin of the model;
  - the tenth element *isDate* indicates if the date of the model has to be read (see later);
  - in addition, the second element being unused by hydraulic software at the moment, it is used by the driver to store the spatial reference system of the datasource, in the form of a single integer with the EPSG number of the projection;
- if *isDate*=1, an *array of integers* of exactly 6 elements, with the starting date of the model (year, month, day, hour, minute, second);
- an *array of integers* of exactly 4 elements:
  - the first element is the number of elements *nElements*,
  - the second element is the number of points *nPoints*,
  - the third element is the number of points per element*nPointsPerElement*,
  - the fourth element must be 1;
- an *array of integers* of exactly *nElements*nPointsPerElement* elements, with each successive set of *nPointsPerElement* being the list of the number of the points (number starting with 1) constituting an element;
- an *array of integers* of exactly *nPoints* elements ignored by the driver (the elements shall be 0 for inner points and another value for the border points where a limit condition is applied);
- an *array of floating point values* of exactly *nPoints* elements with the x-coordinates of the points;
- an *array of floating point values* of exactly *nPoints* elements with the y-coordinates of the points;

The rest of the file actually holds the data for each successive time step. A time step contains the following elements:

- a *array of floating point values* of exactly 1 element, being the date of the time step relative to the starting date of the simulation (usually in seconds);
- *nVar array of floating point values*, each with exactly *nPoints* elements, with the values of each attribute for each point at the current time step.

Mapping between file and layers

Layers in a Selafin datasource

The Selafin driver accepts only Selafin files as data sources.

Each Selafin file can hold one or several time steps. All the time steps are read by the driver and two layers are generated for each time step:

- one layer with the nodes (points) and their attributes: its name is the base name of the data source, followed by "_p" (for Points);
- one layer with the elements (polygons) and their attributes calculated as the averages of the values of the attributes of their vertices: its name is the base name of the data source, followed by "_e" (for Elements).

Finally, either the number of the time step, or the calculated date of the time step (based on the starting date and the number of seconds elapsed), is added to the name. A data source in a file called Results may therefore be read as several layers:

- `Results_p2014_05_01_20_00_00`, meaning that the layers holds the attributes for the nodes and that the results hold for the time step at 8:00 PM, on May 1st, 2014;
- `Results_e2014_05_01_20_00_00`, meaning that the layers holds the attributes for the elements and that the results hold for the time step at 8:00 PM, on May 1st, 2014;
- `Results_p2014_05_01_20_15_00`, meaning that the layers holds the attributes for the elements and that the results hold for the time step at 8:15 PM, on May 1st, 2014;
- ...

Constraints on layers

Because of the format of the Selafin file, the layers in a single Selafin datasource are not independent from each other. Changing one layer will most certainly have side effects on all other layers. The driver updates all the layers to match the constraints:

- All the point layers have the same number of features. When a feature is added or removed in one layer, it is also added or removed in all other layers.
- Features in different point layers share the same geometry. When the position of one point is changed, it is also changed in all other layers.
- All the element layers have the same number of features. When a feature is added or removed in one layer, it is also added or removed in all other layers.
- All the polygons in element layers have the same number of vertices. The number of vertices is fixed when the first feature is added to an element layer, and can not be changed afterwards without recreating the datasource from scratch.
- Features in different element layers share the same geometry. When an element is added or removed in one layer, it is also added or removed in all other layers.
- Every vertex of every feature in an element layer has a corresponding point feature in the point layers. When an element feature is added, if its vertices do not exist yet, they are created in the point layers.
- Points and elements layers only support attributes of type "REAL". The format of real numbers (width and precision) can not be changed.

Layer filtering specification

As a single Selafin files may hold millions of layers, and the user is generally interested in only a few of them, the driver supports syntactic sugar to filter the layers before they are read.

When the datasource is specified, it may be followed immediately by a *layer filtering specification.*, as in `Results[0:10]`. The effects of the layer filtering specification is to indicate which time steps shall be loaded from all Selafin files.

The layer filtering specification is a comma-separated sequence of range specifications, delimited by square brackets and maybe preceded by the character 'e' or 'p'. The effect of characters 'e' and 'p' is to select respectively either only elements or only nodes. If no character is added, both nodes and elements are selected. Each range specification is:

- either a single number, representing one single time step (whose numbers start with 0),
- or a set of two numbers separated by a colon: in that case, all the time steps between and including those two numbers are selected; if the first number is missing, the range starts from the beginning of the file (first time step); if the second number is missing, the range goes to the end of the file (last time step);

Numbers can also be negative. In this case, they are counted from the end of the file, -1 being the last time step.

Some examples of layer filtering specifications:

`[0]`	First time step only, but return both points and elements
`[e:9]`	First 10 time steps only, and only layers with elements
`[p-4:]`	Last 4 time steps only, and only layers with nodes
`[3,10,-2:-1]`	4^{th}, 11^{th}, and last two time steps, for both nodes and elements

Datasource creation options

Datasource creation options can be specified with the "`-dsco`" flag in ogr2ogr.

TITLE
: Title of the datasource, stored in the Selafin file. The title must not hold more than 72 characters. If it is longer, it will be truncated to fit in the file.

DATE
: Starting date of the simulation. Each layer in a Selafin file is characterized by a date, counted in seconds since a reference date. This option allows providing the reference date. The format of this field must be YYYY-MM-DD_hh:mm:ss. The format does not mention the time zone.

An example of datasource creation option is: `-dsco TITLE="My simulation" -dsco DATE=2014-05-01_10:00:00`.

Layer creation options

Layer creation options can be specified with the "`-lco`" flag in ogr2ogr.

DATE
: Date of the time step relative to the starting date of the simulation (see Datasource creation options). This is a single floating-point value giving the number of seconds since the starting date.

An example of datasource creation option is: `-lco DATE=24000`.

Notes about the creation and the update of a Selafin datasource

The driver supports creating and writing to Selafin datasources, but there are some caveats when doing so.

When a new datasource is created, it does not contain any layer, feature or attribute.

When a new layer is created, it automatically inherits the same number of features and attributes as the other layers of the same type (points or elements) already in the datasource. The features inherit the same geometry as their corresponding ones in other layers. The attributes are set to 0. If there was no layer in the datasource yet, the new layer is created with no feature and attribute.In any case, when a new layer is created, two layers are actually added: one for points and one for elements.

New features and attributes can be added to the layers or removed. The behaviour depends on the type of layer (points or elements). The following table explains the behaviour of the driver in the different cases.

Operation	Points layers	Element layers
Change the geometry of a feature	The coordinates of the point are changed in the current layer and all other layers in the datasource.	The coordinates of all the vertices of the element are changed in the current layer and all other layers in the datasource. It is not possible to change the number of vertices. The order of the vertices matters.
Change the attributes of a feature	The attributes of the point are changed in the current layer only.	No effect.
Add a new feature	A new point is added at the end of the list of features, for this layer and all other layers. Its attributes are set to the values of the new feature.	The operation is only allowed if the new feature has the same number of vertices as the other features in the layer. The vertices are checked to see if they currently exist in the set of points. A vertex is considered equal to a point if its distance is less than some maximum distance, approximately equal to $1/1000^{th}$ of the average distance between two points in the points layers. When a corresponding point is found, it is used as a vertex for the element. If no point is found, a new is created in all associated layers.
Delete a feature	The point is removed from the current layer and all point layers in the datasource. All elements using this point as a vertex are also removed from all element layers in the datasource.	The element is removed from the current layer and all element layers in the datasource.

Typically, this implementation of operations is exactly what you'll expect. For example, ogr2ogr can be used to reproject the file without changing the inner link between points and elements.

It should be noted that update operations on Selafin datasources are very slow. This is because the format does no allow for quick insertions or deletion of features and the file must be recreated for each operation.

VSI Virtual File System API support

The driver supports reading and writing to files managed by VSI Virtual File System API, which include "regular" files, as well as files in the /vsizip/ (read-write) , /vsigzip/ (read-only) , /vsicurl/ (read-only) domains.

Other notes

There is no SRS specification in the Selafin standard. The implementation of SRS is an addition of the driver and stores the SRS in an unused data field in the file. Future software using the Selafin standard may use this field and break the SRS specification. In this case, Selafin files will still be readable by the driver, but their writing will overwrite a value which may have another purpose.