Eva Format

&

EVA Text File Format Specification v1.1

Stuttgart, July 2005

www.elxala.de

EVA Format

Introduction

EVA means Archive Variable Structure (from spanish **E**structura **V**ariable de **A**rchivo). It is a general purpose format based in a called EVA Structure which is enough flexible to handle complex information or data types. Nevertheless it is not designed to contain and handle very complex and efficient structures (i.e. CAD-CAM systems) or even document like structures like SGML and derivated formats (HTML, XML etc). EVA handles well easier but widely used structures like lists and tables. More emphasis has been done in human write/readability which is a very important aspect since it can save from developing and mantaining special editors for each new structure or at least delay its development if it is really needed until the structure is mature enough.

First we will define the *EVA Structure* from which an implementation for a specific programming language can be made, that is classes or types that can hold these structures. After this we will give a formal specification for storing/retrieving the EVA structure to/from a plain text file.

EVA Structure

The EVA structure is string based and contains named variables of such type that can hold string values, as well as list of strings as well as lists of lists of strings and therefore also tables. The whole EVA structure is defined by four containers that are represented in figure 1.

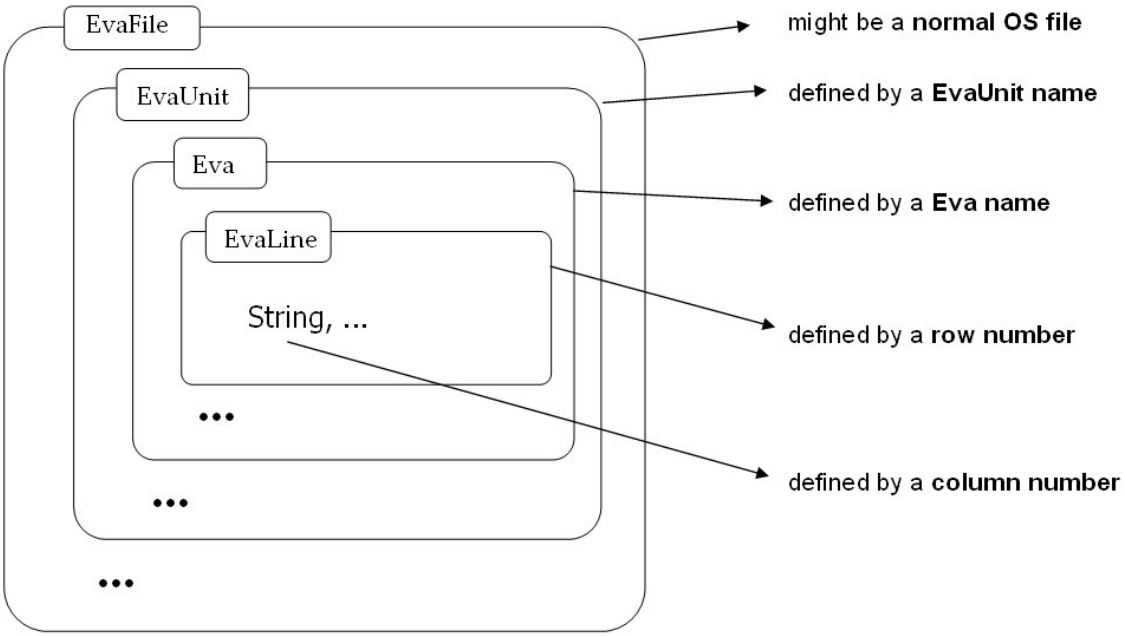

Fig. 1 EVA Structure

An **EvaFile** has a name (i.e. file name) and contains EvaUnit's. An **EvaUnit** has a name and contains Eva's. An **Eva** has a name and contains EvaLine's accesible by a **row** number and an **EvaLine** contains a list of **String**'s accesible by a **column** number. Finally, a String is any char array not containing neither return nor line feed characters.

The names that identify the objects (EvaUnit, Eva) are unique in its containers : each EvaUnit's name is unique within an EvaFile and the same for Eva's name within an EvaUnit. The order of these objects is not relevant (any can be chosen by an implementation).

This structure is storable in a plain text file. The chapter *EVA Text File Format Specification* will cover this. Note that for text files we could identify EvaFile with an Operative System file, but there are still more possibilities to store physically the EVA Structure, for instance in a database. In that case an EvaFile would be just a named container of EvaUnit's (as it is defined).

Implementation details

The idea behind this structure comes from the implementations (nowadays in C++ and java classes) where the EvaUnit is the maximum structure type and the minimum object that can be loaded from a file. Also considering that there are methods to access and handle directly the final strings from either EvaUnit or Eva classes without having to use EvaLine methods, in the praxis we will have the structure depicted in Fig 2.

EvaFile could be easily implemented as a class that really contain several EvaUnit's but it would be not much efficient with big files. Thus it is enough for the EvaFile class to have static methods to load and save EvaUnits from text files. EvaLine as class is still a real container of Strings, it might be used as that but usually only intern methods of EvaFile (save and load) will need to work with it directly.

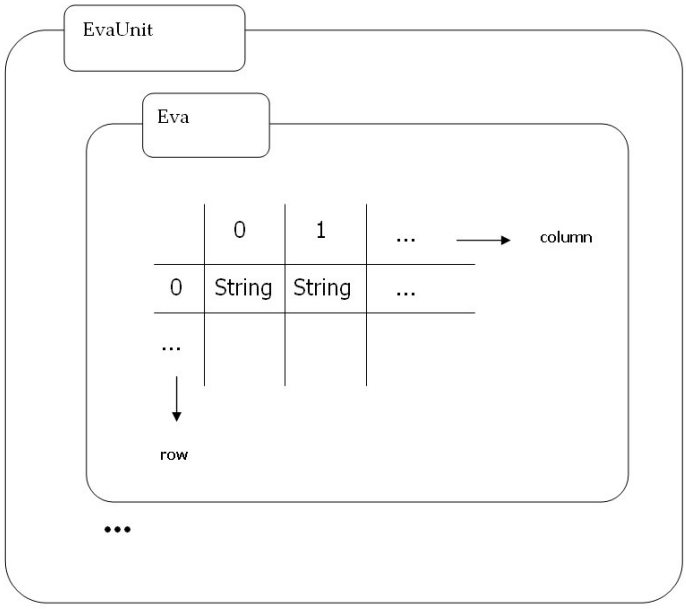

Fig. 2 EVA Structure seen from implementation

Example

To have a better idea of what are we talking about, Fig. 3 depicts a small self-contained example of an EVA Structure and its representation in a text file.

Contents of EvaFile from file example.eva

EvaUnit	Eva	col=0	col=1	row
calendar				
	week			
		Monday	Mo	0
		Tuesday	Tu	1
		Wednesday	We	2
		Thursday	Th	3
	month			
		January	Jan	0
		February	Feb	1
current day				
	day			
		28		0
	month			
		3		0
	year			
		2005		0
	format1			
		28.03.2005, Monday		0

File example.eva

```
#calendar#

      <week>

         Monday,     Mo
         Tuesday,    Tu
         Wednesday,  We
         Thursday,   Th

      <month>

         January,  Jan
         February, Feb


#current day#

      <day>     28
      <month>   3
      <year>    2005
      <format1> "28.03.2005, Monday"
```

Fig. 3 Example of EVA Structure and a way to write it in EVA text file format

EVA Text File Format Specification v1.1

Now we will give the rules for writing an EVA Structure in a plain text file or equivalently how to parse an EVA Text File. Parsing is a job of the implementation (EVA library) while writing may be done manually (human), also by the library ("save method" of EvaUnit) or any other EVA utility or extern program.

In this format an object EvaFile is just an Operative System file name, within which EvaUnit's might be found. An EvaUnit starts when its name is found and ends when the next EvaUnit starts or logical or physical end of file are reached. From an EvaUnit start until its end all the Eva's of the EvaUnit have to be found.

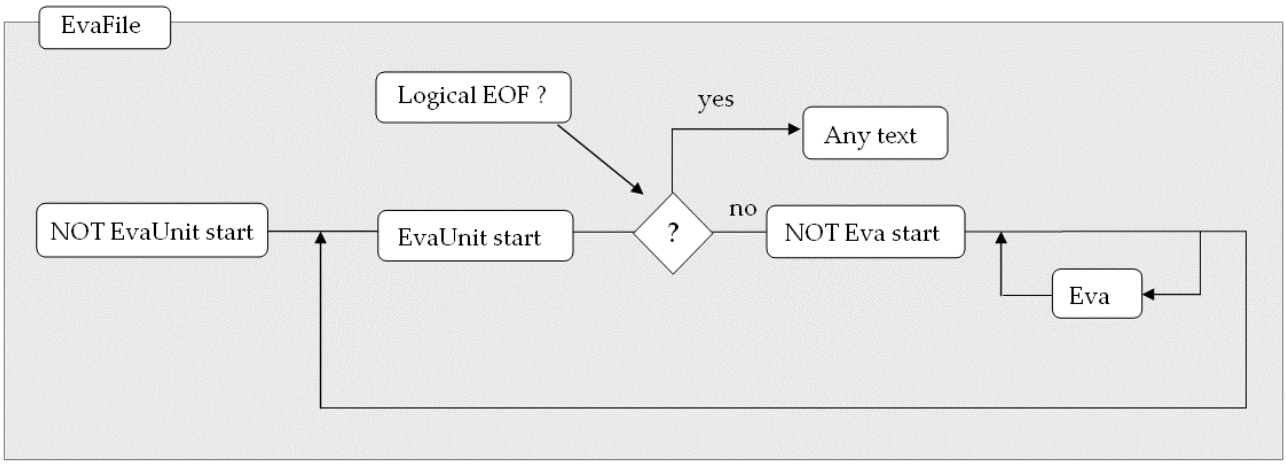

Fig. 4 Parsing an EvaFile

In Fig.4 it is represented the flow diagram for parsing an EvaFile. The definition of the items contained in it are:

[Logical EOF]
The name of the last started EvaUnit starts with ****** (e.g. "****END EVAFILE**")

[NOT EvaUnit start]
Any text (or empty text) that could not be interpreted as **[EvaUnit start]**

[NOT Eva start]
Any text (or empty text) that could not be interpreted as **[Eva start]**

[Any text]
Any text until end of file.

EvaUnit start simply means that an *EvaUnit name* is going to be assigned. An *EvaUnit name* can be given only in one line of the text that, being left trimmed (removing blanks from left), starts with a # character and contains at least a second # character. Then the *EvaUnit name* is the text between these two characters. The rest of the line has no meaning for Eva and can be removed (not parsed). *EvaUnit start* is depicted in Fig.5

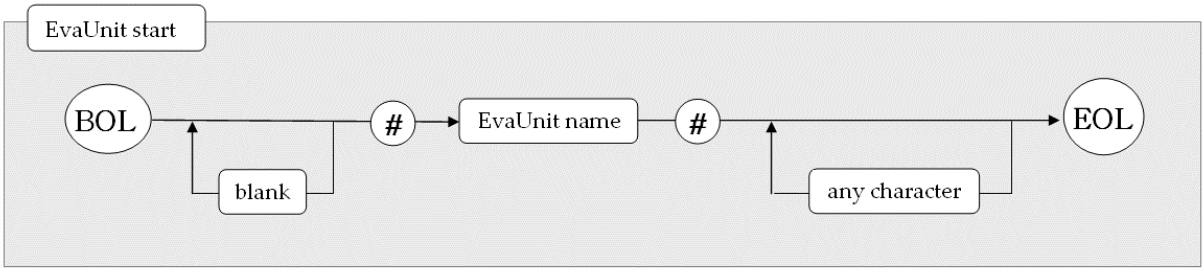

Fig. 5 Getting an EvaUnit name

Where

[BOL]
Begin of line

[EOL]
End of line (either return [13] or line feed [10] or return-line feed [13][10])

[blank]
White space or tabulator

[any character]
any except EOL

[EvaUnit name]
any string not containing **[EOL]** or **#** character at any position (Note : blanks are valid at any position! thus **[EvaUnit name]** must not be trimmed). An empty string is also a valid *EvaUnit name*. For convenience *EvaUnit names* starting with the characters ***** (star), **'** (apostroph) and ! (exclamation) are reserved, that is, not valid EvaUnit names.

If an EvaFile contains two or more EvaUnit's with the same name, only the first one[1] is valid and may be loaded.

An Eva is a set of EvaLine's (each identified by a row number), it starts when an *Eva name* is given (Eva start) and ends when the next Eva starts or the next EvaUnit starts or the end of file or **[Logical EOF]** are reached. Diagram of Fig. 6 shows this

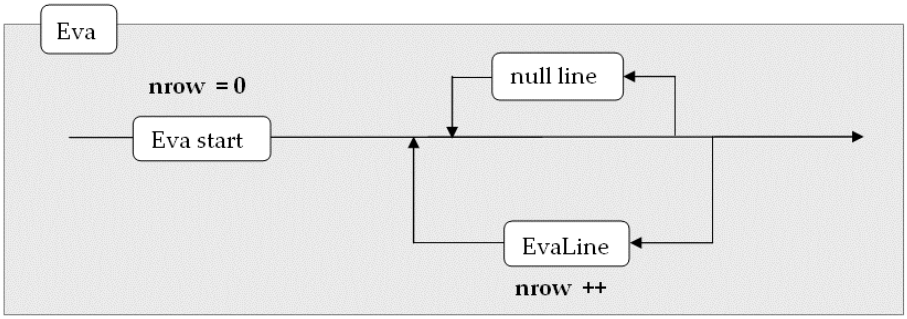

Fig. 6 Parsing an Eva

[null line]
A line having been left trimmed: either results in an empty string, or starts with #!, or starts with <!, or starts with **#** character and has only one **#** character, or starts with **<** character and has no **>** character. These null lines may be used for comments.

[1] Note that for other specification (i.e. Eva Structure as Stream) a different rule would be more interesting

An Eva name can only be given at the begining of a text line and it is obtained from it in the following way: being the line left trimmed (like in EvaUnit names), starts with a < character but not with **<!** and contains at least one **>** character. Then the Eva name is the text between these two characters. Unlike in *EvaUnit name* where the whole line is "consumed" , an Eva name ends with the > character and thus the next character to it until the EOL is considered the first text line to find EvaLine's for this Eva (but not necessarily the first EvaLine!). This is depicted in Fig. 7

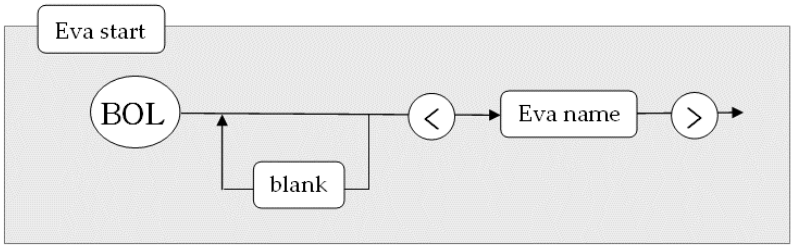

Fig. 7 Getting an Eva name

[Eva name]
any string not containing at any position **[EOL]** or > character (Note : blanks are valid at any position! thus **[Eva name]** must not be trimmed). For convenience Eva names starting with the character ***** (star) and ! (exclamation) are reserved, that means not valid Eva names.

If an EvaUnit contains two or more Eva's with the same name, only the first one is valid and may be loaded.

Finally, an EvaLine is an array of Strings (each identified by a column number) at least of size 1. It can start after an Eva name declaration or in a new text line. The separator of this string array or list is the comma character (**,**). This is shown in Fig 8.

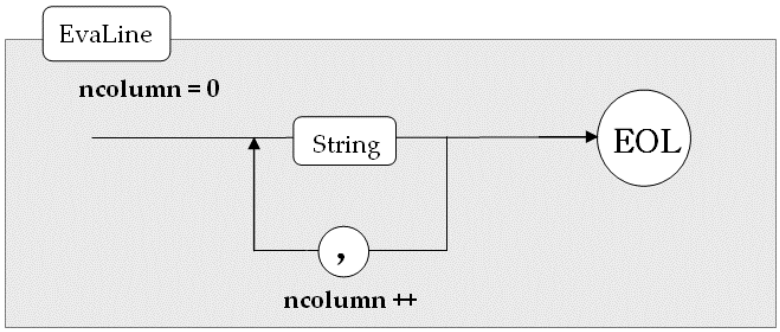

Fig. 8 Getting an Eva name

[String]
By String we mean a character array of any length (length 0 is also possible) but not containing **[EOL]** . Some rules about strings are needed.

Last string rule: if while expecting the begin of a string and the character ’ (apostroph) or two / characters are found then the string starts at the next character and ends at the end of the line, no matter what characters are found[2].

Double quoted rule: This rule does not aply for the string obtained with the "Last string rule" but for the rest of the strings.A string at every column position can be always double quoted (enclosed between " characters), for instance the string *HOLA EVA* will produce the same final string as if it was written *"HOLA EVA"* in he text file.

In the following cases the String **has to** be double quoted:

- a String contains a comma character **(,)**
- a String contains a double quoting character **(")**, also the character **"** inside the String has to be expresed with two double quotes **("")**.
- a String contains blanks at the beginning or at the end (per default all Strings are left and right trimmed)
- a String in the first column (column 0) starts with either the character **#** or **<**
- a String at any column starts with ’ or //

Some examples of these rules are shown Fig. 9

	In Eva Structure		
In text file	column 0	column 1	column 2
one, two, "three"	[one]	[two]	[three]
"one, two", three	[one, two]	[three]	
"""one"", two", three	["one", two]	[three]	
"", one, two	[]	[one]	[two]
" one", "two ", " three "	[one]	[two]	[three]
"<one", <two, <three	[<one]	[<two]	[<three]
"#one", #two, #three	[#one]	[#two]	[#three]
one, ’//rest, of, "line"	[one]	[//rest, of, "line"]	
//"#one", "//two", "’three"	["#one", "//two", "’three"]		

Fig. 9 Examples of writing Strings

[2] Note that this rule is exactly the same for comments in C (//) or other languages (’) and therefore many editors facilitates adding or removing these special marks by blocks. A “column mode“ functionality is also helpful for this task.