The Fib multimedia format

Version V1.2.2

Betti Österholz

BioKom@gmx.de

www.BioKom.info

Potsdam, February 13, 2013

Contents

Part I
Introduction

The Fib multimedia format is used to store multimedia information in a structured, functional and hierarchical form. The structure of the Fib multimedia format supports the object oriented view of things. It is very powerful, since expressions can be combined and nested. The needed memory size of a multimedia object in Fib is depending on its complexity and not on its size (in terms of the extend in the dimensions, for example, the height and width for images), as for standard storage formats.

The second important component of the Fib system is the genetic algorithm for encoding and compressing Fib multimedia objects. The great advantage of this approach is, that the encoding and compression is tied not to a particular algorithm. The encoding and compression algorithms involved in the genetic algorithm are in fact seperated operations that can be easily added and modified. This makes it easy to introduce new encoding and compression ideas and apply a variety of these on a multimedia object.

In this sense, the Fib multimedia system is based on diversity rather than specialization.

1 Given conditions

A (natural) image is generally not a series of independent pixels (points with certain colors), but contains, for example, different objects, which are distinguished from one another and have a texture. Such objects (e.g. circles, lines, but also more complex objects) are often repeated in pictures, either as a self-similar copy or a transformed copy (e.g. tree leaves or slats in the fence). The same applies to other contents of multimedia data, such as the sound.

Furthermore, elements or objects in the multimedia data are often associated with each other or grouped, e.g. certain sounds belong often to specific objects (e.g. a C-hook), object belong to other objects (e.g. the C-hook to a duck) and several objects can be grouped into one object (e.g. a field with a group of ducks).

2 Problem

How to extract information about objects from an image, which is given in the form of a matrix of pixels? (The image is given as a pixel image, for example, because it was recorded by digital photography with a CCD.)

The extracted information can then for example be used for compression, because the information needed to encode an object, is at most the information from all points of the object with their corresponding colors (this coding is given). For example, for a green triangle displayed in the image, which consists of 200 green

points, it is sufficient in the best case, to just represent the triangle and the points by a triangular object, for this only its three corners and the color is needed.

With the information about objects in the image, you can also to recognize objects at other (sub-) images. Or the information can be used to manipulate the image, e.g. to remove objects as a whole, copy or paste them.

3 A solution

The Fib multimedia language, respectively Fib multimedia format, has be created, in which for the objects, transformations and relationships between the objects can be defined. Using these Fib multimedia format, an algorithm (e.g. the AI) then generates programs (in the multimedia language), tests and modifies them, so that they represent a multimedia object, wich are similar enough to encode the original multimedia object (what this means, will be specified). Further restrictions can be imposed on the programs, e.g. shortnes and speed of execution of a program. These criteria will then determine how "good" a program is.

Among others the genetic algorithms / programming offer good opportunities. Since with genetic algorithms the individual parameters, elements and constructs of the language are generated by mutation and adjusted, and the "best"/fittest - programs will be developed. With "crossing over", sharing code between programs or part of a program can be taken into account, this supports the case when, may be altered (e.g. scaled), objects will be reused in other places.

4 This document

This document is a short version of the original german documentation. Some parts of the german documentation are skipted in this version.

This document contains mainly the discription of the important theoretical background. The description of the source code, will be in the source code, as doxygen comments.

5 Structure of this project

The structure of this document is based on the project structure.

The project is structured hierarchically and divided into two major components. The components are the multimedia description language Fib and the genetic algorithm to produce good Fib encodings of multimedia objects.

The project, including each of the components, is build hierarchically. Whereupon the higher levels are build on the lower levels, while each higher level is more detailed. This documentation includes just the lowest levels.

6 Structure of this document

This section gives an overview of the content and structure of this document.

6.1 Design of the multimedia description language

The part II on page 4 deals with the multimedia description language Fib.

The individual elements and their possible relationships to each other, will be described in section 9 on page 5 .

The section 10 on page 53 adresses the Fib database function for Fib (objects).

In the following section 11 on page 53 definitions for the Fib multimedia language are established, to facilitate a way of dealing with it.

6.2 Design of the genetic algorithm

The part III on page 71 adresses the design of the genetic algorithm. It is mainly about the basic ideas behind the genetic algorithm for Fib.

6.3 File format

The part IV on page 80 describs the file formats for the Fib multimedia language. These are independent of the program or the computer system architecture.

Part II
The Fib multimedia description language

In this part the elements of the Fib multimedia language will be defined.

7 Requirements

Since for the classical bit representation low performance can be expected by genetic algorithms, but because of the complexity of the problem a very high performance is needed, an adapted problem representation (multimedia description language) is selected.

The multimedia description language Fib should be kept as simple as possible, with yust few alternatives, since an increases of the available alternatives will increase the number of alternatives in the application of the genetic operations and this will likely increase the amount of computation and implementation complexity.

On the other side, the multimedia description language should allow compact expressions for "normal" (occurring in the practice) multimedia objects. It should therefore provide good compression options for " normal" multimedia objects. This implies that relationships (e. g. color gradients in a surface) between parts (e. g. pixels of the surface) of a multimedia object can be represented as simply as possible.

The language should be unambiguous, reproducible and evaluable, so that an expression always evaluates to the same object. The clarity and reproducibility isn't so important for the implementation (e. g. due to different rounding errors on different architectures), if the generated multimedia object is almost always (as an example, in 0.999999 fraction of cases) very similar to the original multimedia object. But valid language objects have to be always evaluable. Otherwise restrictions on the genetic operators, by which they wher generated, have to be made or not evaluable objects could arise.

With the multimedia description language it should be possible to represent at least all raster graphics. The generation of a raster image by a Fib multimedia object in the multimedia description language should be comprehensible. That is, the multimedia description language should be designed to support the distinction of individual objects and their dependencies.

The genetic operations which change the multimedia programs should, if possible, change the program in a way, to allow a gradient descent in the hypothesis space of the multimedia programs. By running multiple operators the hypotheses, that the multimedia programs represent, should therefore be gradually improved.

8 The multimedia description language

The multimedia description language is named Fib (for "funktionale Interpretation von Bildern" or "functional interpretation of bitmaps (/bictures)"). This name orgins from the first version of the multimedia description language, when it was only good for saving images (german: Bilder).

With some additional elements the multimedia description language has been extended, so that it can store any multimedia data. The only restriction on the multimedia data is, that it can be represented as properties of points of a finite, euclidean and discrete (there are smallest units) space.

Fib is a vector representation of multimedia data, that means a representation of multimedia data using objects. As a basic framework a tree is used. The leaves are endpoints that are used for displaying and / or assignment of points or part of multimedia objects. In the branches and the alignment of these, which for example is most left, display parameters or properties of the leaves are encoded, e. g. how often it is displayed or with which color.

Each node of the tree is a Fib element. The tree is evaluated from the root to the leaves, whereupon the elements affect the evaluation.

A valid Fib object (tree) is cycle free, to ensure a finite processing time.

The elements of the multimedia description language are oriented on some of the usual imperative programming languages (e. g. C++, Java).

All units are expressed on the basis of the International System of Units (SI).

When a Fib object is evaluated, all points with their properties are determined for the multemedia object. A evaluated multimedia object thus contains only a list of specific points and their properties, and can thus be directly displayed (through displaying the properties at the points / coordinate) or evaluated.

Hereinafter "top" in the Fib object is referred to the direction, in which the root of the tree lies. The direction of "bottom" is thus the opposite direction (away from the root).

9 Elements of the Fib multimedia description language

This section describes the elements of the Fib multimedia description language.

Below Obj stands for a Fib object.

9.1 Vectors

Vectors are used for providing numerical values. There are different types of vectors (e. g. position vectors, range vectors or property vectors for RGB colors). The type of a vector (and optionally information in the root-element) gives the domain and the number of the elements that it contains. Each vector element is either a number or a variable, which is set in the Fib object above the Fib element of the vector.

If the value of an evalued element is outside of its domain (the domain for the vector element), this value is rounded to the nearest value that lies in the domain. If an element of the vector is not existing, it is evaluated as the zero value of its domain.

Vectors of one type are only used in a very specific type of Fib element.

A listing of the possible vector typs can be found in Table 1 .

vector	description	used in Fib element	number of elements	typ of elements
position	the position of a point	$point$	any (a vectors can always contain just 0 elements)	numbers
area	contains the borders of an area (inclusive the borders)	$area$	2	integers
property vector (there are more than one type)	defines a value of a property	$property$	any	numbers

Table 1: Vector typs

9.2 Points

The points are the displaying elements. At them the set properties are evaluated. Points serve as the leaves of Fib objects.

An empty point "point()" is possible. The empty point has no effect (all properties are lost). It can be used in combination with other elements (e. g. the if-element).

There are also points with an empty position vectors "point(())", these points create the background (e. g. the background color).

The properties of the point (e. g. color, sound and / or odor) are determined in the branch above the point with property elements.

Syntax: $Obj = point(PositionVector)$

Short syntax: $Obj = p(PositionVector)$

$PositionVector$: the coordinate vector of the point

Examples:

- $p((10;20))$; a point on the position $(10, 20)$
- $p((10;x))$; x is a variable that must be set in the branch to this point
- $p()$; the point has no impact and all the properties are discarded
- $p(())$; point that affects the entire background

9.3 Property element

With the property element "property" properties for Fib objects will be set.

Syntax: $Obj = property((value_1, ..., value_n)_{name}, Obj_1)$

Short syntax: $Obj = pr((value_1, ..., value_n)_{name}, Obj_1)$

$name$ The $name$ specifies the name of the property. It determines the type of the vector. All property types should belong to the vector supertype "property".

$value_i$ This is the value i of the property.

Obj_1 The contained object, for which the property will be set.

name	value	number of values	description	example
whatever	0	0	The properties of the subobject does not matter. Whichever properties are also associated with this subobject, they are correct.	pr($()_{whatever}$, Obj)
Color (all colors overwrite current set colors of the actual Fib object)				
colorRGB	1	3	color as red, green and blue fraction values	pr($(255, 16, 0)_{colorRGB}$, Obj)
colorGrayscale	2	1	luma fraction	pr($(25)_{colorGrayscale}$, Obj)
More properties				
layer	100	1	layer for the points (lower layers are covered by higher layers)	pr($(2)_{layer}$, Obj)

name	value	number of values	description	example
transparency	200	1	transparency fraction for the colors of the points	pr($(25)_{transparency}$, Obj)
persistent	210	0	This property is only useful for a time period (or the dimension of time). Points in space with this property lose their other properties only, if they are overwritten later in time by a respective property of the same type. This of course only holds as long as the particular point has the property $persistent$. The property $persistent$ is useful for example, if in a movie a objects should be visible as long as they are not overwritten by other objects. In this case, for the entire Fib object the property $persistent$ can be set. If an object is defined at a time and displayed, it will displayed in the future as long as it is not overwritten.	pr($()_{persistent}$, Obj)
Sound properties				
sound	300	4	a sound; the values are: 1. frequency in Hertz $(1/s)$, 2. sound pressure in Pascal Pa $(1Pa = 1N/m^2)$, 3. phase shift in radians, 4. duration in seconds; a sound is additive to other sounds	pr($(5000, 40, 0.5, 50)_{sound}$, Obj)

name	value	number of values	description	example
soundPolarized	301	$3 + \sharp D$	a sound; the values are: 1. frequency in Hertz ($1/s$), 2. sound pressure in Pascal Pa ($1Pa = 1N/m^2$), 3. phase shift in radians, 4. duration in seconds; r = 5 to ($3 + \sharp D$) polarization fraction (as an angle in radians) in the dimension plane, which is spanned by the respective dimensions $r-4$ and $r-3$ ($\sharp D$ is the number of dimensions), the angle origin is the $r-3$ axis a goes in positive direction; a sound is additive to other sounds	pr($(5000, 40, 2, 0.5, 5)_{soundPolarized}$, Obj)
soundAmplitude	305	3	the amplitude of a sound; the values are: 1. sound pressure in Pascal Pa ($1Pa = 1N/m^2$), 2. phase shift in radians, 3. duration in seconds; a sound is additive to other sounds; With this properties sounds can be build by their amplitude with a specific sampling rate, such as in the WAVE file format.	pr($(40, 0.5, 0.0005)_{soundAmplitude}$, Obj)
soundBarrier	310	1	speed of sound in meters per second (m/s); With this property objects can change the acoustics.	pr($(343)_{soundBarrier}$, Obj)
soundReflected	311	1	fraction of sound reflected from the object; This property applies to the surface / the edge of the object and not for all his individual points	pr($(50)_{soundReflected}$, Obj)
soundDamping	312	1	fraction of the sound swallowed by a point	pr($(2)_{soundDamping}$, Obj)
Physical properties				
kelvin	400	1	temperature in Kelvin	pr($(300)_{kelvin}$, Obj)

name	value	number of values	description	example
electroMagnetic	410	$3 + \sharp D$	an electromagnetic radiation source, the values are: 1. frequency in Hertz $(1/s)$, 2. amplitude in Candela cd, 3. phase shift in radians, 4. duration in seconds, r = 5 to $(3 + \sharp D)$ polarization fraction (as an angle in radians) in the dimension direction, which is spanned by the respective dimensions $r - 4$ and $r - 3$ ($\sharp D$ is the number of dimensions), the angular information is provided by the $r - 3$ axis in positive direction, an electromagnetic wave is additive to other electromagnetic waves	pr($(5, 3 * 10^{14}, 2, 0.5, 0.5, 50)_{electroMagnetic}$, Obj)
Properties for describing objects (They describe only the part objects, without any further impact)				
periodBegin	500	1	time in seconds (s) from the beginning of the whole multimedia object, starting at which the object is to be displayed; if possible, this property should be near the root of the multimedia object; when a multimedia object is played it can be determined with this property: the order in which subobjects should be evaluated and/or till which time to evaluate a part object	pr($(0.3)_{periodBegin}$, Obj)

name	value	number of values	description	example
periodEnd	501	1	time in seconds (s) from the beginning of the whole multimedia object, till which the object is to be displayed; if possible, this property should be near the root of the multimedia object and follow a "periodBegin" property; when a multimedia object is played it can be determined with this property: the order in which subobjects should be evaluated and/or till which time to evaluate a part object completely	pr($(0.4)_{periodEnd}$, Obj)
evaluationTime	502	1	time required for evaluating a multimedia object, in proportion to a multimedia object, which contains only one point (the value should be seen as a multiple of the evaluation time of a point); with this property in combination with the properties "periodBegin" und "periodEnd" a good evaluation order and time can be evalued for the partobjects, when playing a multimedia object; this property should stand immediately after (or below/within) "periodBegin" and "periodEnd"	pr($(15.8)_{evaluationTime}$, Obj)
Properties for the compressed storing (these have no effect on the points)				

name	value	number of values	description	example
checksum	600	3	An checksum for the object will be generated. The first parameter determines the type of the checksum. The second parameter specifies any which number of bits, a checksum is to be generated, and the third parameter defines how many bits the checksum is long. The last block of the checksum will be filled with 0 after loading the blocks, so that it too has the desired length. If there are enough bits to correct an existing error, it will be attempted to correct the error. (see section 21.3.3 on page 104)	pr($(1, 1024, 16)_{checksum}$, Obj)
boundSize	601	0	For the part object the border/-size in bits will be stored, when saving it. If an error occured while loading the part object, the (in the bitstream after the faulty part object) following part objects can still be loaded, because their beginning is known. (see section 21.3.3 on page 105)	pr($()_{boundSize}$, Obj)
Other properties				
Product Properties	240 to 255	variable	Properties which are product specific. Different producers can use this area, without getting incompatible with later defined properties.	

Table 2: properties (the prefix "property::" was omitted because of clarity)

The table 2 shows different properties, which can be set with the "property" el-

ement (the prefix "property::" for the names was omitted because of clarity). Every property has it's own vector type. Every vector type has the supertype "property". The domains of the vector typs are declared in the root-element (see section 9.14 on page 33).

In table 2 the $\sharp D$ stands for the number of dimensions in the Fib multimedia object.

In table 5 on page 43 the property typs with their default domains are listed.

If for a position a needed property dosn't exists, the zero vector from the valid domain (or maybe the default domain) will be assumed for it.

Examples:

- $pr((255, 0, 0)_{colorRGB}, p((10; 20)))$; a red point on position (10,20)
- $pr((x)_{colorGrayscale}, p((10; x)))$; x is a variable, which will be set higher in the branch, this variable influenced the position and also color / brightness of the point
- $pr((0, 0, 255)_{colorRGB}, p(()))$; the whole background is blue

9.3.1 Properties for fractions

For elements of properties relating to fractions, the upper limit (100 %) is determined by the maximum of its domain and its lower limit (0 %) is determined by the minimum of its domain. Therefore, for elements of properties relating to fractions, there should be always specified the minimum and maximum value for the domain for the lower and upper limit, even if it is not used in the Fib object.

As an example, the color red for "colorRGB" is shown here. This is assigned in the Fib object a domain of integers in the range from -10 (lower limit) to 90 (upper limit). The color red is, when viewed, a portion of the red value of a point, which goes from 0.0 for no red (lower limit) to 1.0 for maximum red (upper limit). The -10 of the property will correspond to the value 0.0 of the red value on display and the 90 corresponds to 1.0. The interim value of the property 40 will correspond to 0.5 and the intermediate value 0 will correspond to 0.1 . It should be noted that in the domain, not all integers between -10 to 90 must be present. The domain may consist of only eight numbers (eg. $D = \{-10, 0, 3, 4, 5, 21, 40, 90\}$), of which -10 is the smallest and 90 is the greatest number. The -10 and the 90 have to be in the domain, to set the limits, even if they do not appear in the Fib object. The same procedure is used for the other color values of the "colorRGB" property, which may have different domains.

Elements of properties, which are fractions:

- "colorRGB": all vector elements; 0.0 is the lower boundery when displayed and corresponds to "no color"

- ”colorGrayscale“: all vector elements; 0.0 is the lower boundery when displayed and corresponds to ”black“
- ”transparency“: all vector elements; 0.0 is the lower boundery when displayed and corresponds to ”not transparent“, 1 corresponds to ”total transparent“
- ”soundReflected“: all vector elements; 0.0 is the lower boundery when displayed and corresponds to ”no reflection“
- ”soundDamping“: all vector elements; 0.0 is the lower boundery when displayed and corresponds to ”no damping“

9.4 List element

With the list element “list” several objects can be combined into one object. For that an execution order is determined, to ensure, in case of overlap of the subobjects, that always the same subobjects cover the same other subobjects, so that this don’t change (because of the uniqueness of Fib objects).

Syntax: $Obj = list(Obj_1, \ldots, Obj_n)$

Short syntax: $Obj = l(Obj_1, \ldots, Obj_n)$

Obj_i are the branches/ subobjects of the list object, with $i \in \{1, \ldots, n\}$ and $n \geq 2$ (there have to be at least two subobjects in the list object). The subobject Obj_i will be evalued before the subobject Obj_{i+1}, so Obj_{i+1} will maybe overlap Obj_i.

9.5 Comment element

The comment element is used to name and describe subobjects. Within it, in contrast to all other elements, strings can be used.

Syntax: $Obj = comment(Key, Value, Obj_1)$

Short syntax: $Obj = c(Key, Value, Obj_1)$

Description of the elements:

Key**:** the key of the comment or description of Obj_1

$Value$**:** the value of the comment or description of Obj_1

Obj_1**:** the subobject, for which the comment or description apply

The key Key can be any string. It is advisable to choose one of the predefined keys from table 3 . In this way, all keys from the entire Fib object can be filtered or can be used for looking for a specific subobject.

The value $Value$ of a comment can be any string.

Key	description	example
unknown	unknown type of comment	c(unknown, “Waka Waka”, Obj)
autor	autor of the subobject	c(autor, “oesterholz”, Obj)
autor::email	email adress of the autor	c(autor::email, “autor@gmx.com” , Obj)
autor::adress	adress of the autor	c(autor::adress, “Some City 123456;Main Street 13”, Obj)
autor:: telephon	telephone number of the autor	c(autor::telephon, “012/345/6789”, Obj)
creation::date	creation date of the subobject	c(creation::date, “2009/10/30”, Obj)
creation::time	creation time of the subobject	c(creation::time, “15/57/32”, Obj)
creation:: coordinate	cratione position of the subobject as Geographic Coordinates	c(creation::coordinate, “Lat = 47° 25′ N, Lon = 010° 59′ E”, Obj)
creation:: location	creation position of the multimedia object as a place name	c(creation::location, “Platz der Republik 1, 10117 Berlin”, Obj)
type	type of the subobject	c(type, “tree”, Obj)
description	description of the subobject	c(description, “This is me while fishing”, Obj)
name	name of the subobject	c(name, “Statue of Liberty”, Obj)
copyright	Copyright of the subobject	c(copyright, “GPL3”, Obj)
comment	a general comment for the subobject	c(comment, “please rework again”, Obj)
link	a link for the subobject (it can be accessed when you click on the object)	c(link, “http://www.fib-development.org”, Obj)
nextElement:: description	A description for the next Fib element that the comment element contains.	c(nextElement::description, “This element is used to generate copies.”, Obj)

nextElement:: function	A description of the function of the next Fib element, which is contained in the comment element. If the key is a name of a dimension direction, the next Fib element should be an area element, that defines a variable, through which the corresponding dimension direction is generated. If the key is for example “time”, the defined variable of the contained area element can be set to generate the multimedia object for a specific time. Other possible key values: “scene” the area element iterates through the scenes of the multimedia object. With such identification, certain moments or scenes can be accessed quickly.	c(nextElement::function, “time”, Obj)

Table 3: Keys

9.6 Area element

The area element defines a variable. It sets the defined variable to integers of a specified range of integers. The area element contains, in addition to the defined variables and the subobject to which it applies, a list of areas, which the variable will go through. The variable is valid everywhere in the subobject.

Syntax: $Obj = for(Variable, (B_1, B_2, \ldots, B_n), Obj_1)$

Description of the elements:

- $Variable$: The variable, which the area element defines.
- B_i $(i = 1 \ldots n)$ are the areas. For the value n is $n \geq 1$, so there is at least one subarea in the area element.
- Obj_1: The subobject, for which the $Variable$ is defined and which will be evalued for every variable assignment of the area.

One (sub-)area B_i is a vector of degree 2, whose two integer components specify an integer field, to which the variable will be set. An area B_i consists of the two

(integers) elements of the vector B_i and all integers between them. If one element of the vector is a variable, that contains a non-integer value, it is rounded to an integer. For the rounding, the decimal digit befor the point remains the same, if the first decimal digit after the point is between 0 to 4, otherwise (from 5 to 9) the number of the decimal befor the point is increased by one for positive numbers and decreases by one for negative numbers.

The area element includes as its area the union of all its subareas B_i. It will therefore go through a range of all integers, which are contained in its subareas B_i.

Examples:

- $for(x, [(1; 3), (10; 14)], Obj)$; In this example, the variable x for the object Obj is set to the successive values: 1; 2; 3; 10; 11; 12; 13; 14
- $for(x, [(1; y = 3.4985)], Obj)$; In this example, the variable x for the object Obj is set to the successive values: 1; 2; 3
- $for(x, [(1; y = 3.5)], Obj)$; In this example, the variable x for the object Obj is set to the successive values: 1; 2; 3; 4

Note: With this definition of the area element continuous functions can not be realized, because the area element just allows integers and dosn't allow continuous transition of values. Functions can only be realized continuous up to a certain point (e. g. in the range of integers).

Example: The function $y = x^2$ for the area $x = \{0, 1, 2\}$ (chosen simple just for clarification)

If it is tried to realize it in the form: $for(x, [(0; 2)], fun(y, exp(x; 2), p((x, y))))$ gaps will result (in the transition from (1, 1) to (2, 4) a point (x, 3) is missing).

However, this can be solved by "compression of the range":
$for(x, [(0; 6)], fun(sx, div(x, 3), fun(y, exp(sx, 2), p((sx, y)))))$

Now a point (2;3) exists ($x = 5 \rightarrow sx = 5/3$ rounded up 2; $y = (5/3)^2 \simeq 2,78$ rounded up 3)

This has been choosen in favor for an easier implementation and better performance. Multimedia objects (e. g. images) which are encoded in the Fib multimedia description language, will be made of individual dots (pixels) in the representation.

However, it is possible to achieve a scalability of a Fib multimedia object in other ways.

9.7 Functions

Functions are Fib elements that assign to the variable, that they define, a value, that is calculated using a formula. For that a function contains a subfunction. A subfunction is a number, variable, or a true subfunction.

Syntax: $Obj = fun(Variable, UF, Obj_1)$ (An alternativ for “fun” is “fkt”.)

Short syntax: $Obj = f(Variable, UF, Obj_1)$

Description of the elements:

- $Variable$: The variable, which the function element defines.
- UF: This is the subfunction of the function.
- Obj_1: The subobject, for which the $Variable$ is defined and which will be evalued for the calculated variable assignment of the function.

9.7.1 Numbers and variables as subfunctions

A subfunction can be a number or a variable. This usually only makes sense, if the number or a variable subfunction is a subfunction of an other real function (like the addition). Variables must be defined in a Fib object above the function element, in which they are used (e. g. by a different function element).

Syntax number: X

Example number: 3

Syntax Variable: x

Example Variable: x

9.7.2 Real subfunctions

Every real subfunction, like each variable, represents one value. Before evaluing a subfunction, the values of its subfunctions will be evalued.

There are no volatile / non-defined values. Function values, which are not defined for the normal mathematical function, are mapped to 0. Since for Fib objects no claim is made that they are mathematically correct (they should be unambiguous, reproducible and evaluable), so the filling of gaps in the mathematically definition is appropriate.

In the following UF_1 and UF_2 are subfunctions.

Addition The addition is needed as the most basic operation.

Syntax: $UF = add(UF_1, UF_2)$

Examples:

- $fun(x, add(1, 3), Obj)$; Realizes the function: $x = 1 + 3 = 4$
- $fun(x, add(y, -3), Obj)$; Realizes the function: $x = y + (-3) = y - 3$
- $fun(x, add(add(2, y), add(z, v)), Obj)$; Realizes the function: $x = (2 + y) + (z + v) = 2 + y + z + v$

Subtraction The subtraction subtracts two values.

Syntax: $UF = sub(UF_1, UF_2)$

Examples:

- $fun(x, sub(1, 3), Obj)$; Realizes the function: $x = 1 - 3 = -2$
- $fun(x, sub(y, -3), Obj)$; Realizes the function: $x = y - (-3) = y + 3$
- $fun(x, sub(sub(2, y), add(z, v)), Obj)$; Realizes the function: $x = (2 - y) - (z + v) = 2 - y - z - v$

Multiplication The multiplication multiplies two values.

Syntax: $UF = mult(UF_1, UF_2)$

Examples:

- $fun(x, mult(2, 3), Obj)$; Realizes the function: $x = 2 * 3 = 6$
- $fun(x, mult(y, 3), Obj)$; Realizes the function: $x = y * 3 = 3y$
- $fun(x, add(y, mult(-2, z)), Obj)$; Realizes the function: $x = y+(-2z) = y - 2z$

Division The division could in fact be replaced by the multiplication and the exponential function ($div(a, b) = mult(a, exp(b, -1))$), but because this is expensive there is this separately function.

Syntax: $UF = div(UF_1, UF_2)$

Examples:

- $fun(x, div(2, 3), Obj)$; Realizes the function: $x = 2/3$
- $fun(x, div(y, 3), Obj)$; Realizes the function: $x = y/3$
- $fun(x, mult(4, div(y, z)), Obj)$; Realizes the function: $x = 4 * (y/z)$
- $fun(x, div(4, 0), Obj)$; will be evalued to 0, because $x = 4/0$ is not defined
- $fun(x, div(335, 113), Obj)$; Realizes the function: $x = 335/113 \simeq 3,1415929 \simeq \pi$

Modulo This function provides the symmetric modulo operator. The symmetric modulo operator returns the remainder of the integer division. ($mod(x, y) = x - y * int(x/y)$, where int refers to the truncation of the decimal digits)

Syntax: $UF = mod(UF_1, UF_2)$

Examples:

- $fun(x, mod(-12.3, 3), Obj)$; the evalued value is -0.3
- $fun(x, mod(6.5, 0.5), Obj)$; the evalued value is 0
- $fun(x, mod(-4.687, -3), Obj)$; the evalued value is -1.687

Exponent The exponential function is working on two subfunctions. Wherein the first value is the basis and the second the exponent.

Syntax: $UF = exp(UF_1, UF_2)$

Examples:

- $fun(x, exp(2, 3), Obj)$; Realizes the function: $x = 2^3$
- $fun(x, exp(y, 3), Obj)$; Realizes the function: $x = y^3$
- $fun(x, exp(4, div(y, z)), Obj)$; Realizes the function: $x = 4^{y/z}$

Minimum The minimum function operates on two subfunctions. It provides as the result the smallest value of the two subfunctions.

Syntax: $UF = min(UF_1, UF_2)$

Examples:

- $fun(x, min(0, 12), Obj)$; will return 0
- $fun(x, min(add(-2, 7), 4), Obj)$; because $add(-2, 7) = -2 + 7 = 5$, 4 will be returned

Maximum The maximum function operates on two subfunctions. It provides as the result the biggest value of the two subfunctions.

Syntax: $UF = max(UF_1, UF_2)$

Examples:

- $fun(x, max(0, 12), Obj)$; will return 12
- $fun(x, max(add(-2, 7), 4), Obj)$; because $add(-2, 7) = -2 + 7 = 5$, 5 will be returned

Logarithm The (natural) logarithm function works, in contrast to the previously described functions, only on one value. The natural logarithm is determine to the base e.

Syntax: $UF = ln(UF_1)$

Examples:

- $fun(x, ln(2), Obj)$; Realizes the function: $x = \ln(2) \simeq 0,6931$
- $fun(x, ln(-2), Obj)$; is evalued to 0, because $x = \ln(-2)$ is not defined

Sine The sine function works on only one value that is specified in radians.

Syntax: $UF = sin(UF_1)$

Explanatory notes: Since the sine function (or cosine) in conjunction with the Fourier transformation is common in image processing, the sine function is expected to enrich the Fib multimedia description language. The cosine function can be easily obtained from the sine function (by the addition of $\pi/2$: $cos(x) = sin(x + \pi/2)$). The tangent function can be derived using the sine function ($\tan(x) = sin(x)/sin(x + \pi/2)$).

Examples:

- $fun(x, sin(0), Obj)$; realizes the function: $x = \sin 0 = 0$
- $fun(x, sin(1.57), Obj)$; realizes the function: $x = \sin 1.57 = \sin 3.14/2 \simeq 1$
- $fun(x, sin(y), Obj)$; realizes the function: $x = \sin y$
- $fun(x, sin(sub(1.57, y)), Obj)$; realizes the function: $x = \sin(1.57 - y) \simeq \cos y$

Arc sine The Arkussinusfunktion is the inverse of the sine function. It returns values in radians.

Syntax: $UF = arcsin(UF_1)$

Example:

- $fun(x, arcsin(0), Obj)$; realizes the function: $x = \arcsin 0 = 0$

Absolute value The absolute value function returns a positive number. If the input of the absolute value function was negative it is multiplied by -1, otherwise it is returned directly without modification.

Syntax: $UF = abs(UF_1)$

Example:

- $fun(x, abs(-124), Obj)$; realizes the function: $x = |-124| = 124$

Round This function rounds the value of the subfunction to an integer value. For the rounding, the decimal digit befor the point remains the same, if the first decimal digit after the point is between 0 to 4, otherwise (from 5 to 9) the number of the decimal befor the point is increased by one for positive numbers and decreases by one for negative numbers.

Syntax: $UF = round(UF_1)$

Examples:

- $fun(x, round(-12.3), Obj)$; the returned value is -12
- $fun(x, round(6.5), Obj)$; the returned value is 7
- $fun(x, round(-4.687), Obj)$; the returned value is -5

Delay In the delay function retrieves the previous value of a variable X. In the evaluation of Fib objects some branches will be evalued several times (e. g. for each value of an area element). The delay function will return the value, that it has taken before UF_1 calls of the delay function.

Status: not implemented, planned for implementation

Syntax: $UF = delay(X, UF_1, UF_2)$

Description of the elements:

- X: The variable, which former value should be returned. The variable should be defined in the same branch as the Fib element of the delay function.
- UF_1: An natural number, which determines, from which former delay-call the value of X should be taken. If UF_1 is no natural number, it will be rounded to a natural number.
- UF_2: The standard value which is given back, if there is no UF_1 former value for X.

The dalay function stores for every call a of an evaluation the value W_a, of the variable X. When the dalay function is called the n'th time, it returns the value W_{n-UF_1}, which the variable X had in the $n - UF_1$ call. If $n - UF_1$ is smaler as 1, the value of the subfunction UF_2 will be returned.

An run is determined by the evaluation of the entire Fib object. For example, the evaluation of the Fib object over the top most root-element is a run. If such an evaluation over the top root-element is restarted, a new run is started and the delay function discards all previous values W_a.

Example:

- $delay(x, 1, 3)$; the variable x is set to the value, it had in the former delay-call or to the value 3, if the delay function is evaluated for the first time (in the run).

Note: With the delay function in connection with the set element, as well as area and function elements, for example, polygons can be easily generated. With the set-element, the edges of the polygon should be established. With the delay function the values for previous vertices are retrieved, which are then connected via divisional and functional elements.

if The if function works similar to the if-element (see section 9.8 on page 24). It will evalue a condition and than, depending on whether the condition result is true or false, will return the value of its first (true case) or second (false case) subfunction.

Syntax: $UF = if(Condition, UF_1, UF_2)$

Description of the elements:

- $Condition$: The condition of the if-function as described in section 9.8 on page 24. If the condition is true, the value of first subfunction UF_1 will be returned, else, if the condition is false, the value of the second subfunction UF_2 will be returned.
- UF_1: The first subfunction of the if function.
- UF_2: The second subfunction of the if function.

Example:

- $if(lo(x, 7), 1, sin(x))$; will return 1, if the variable x is lower than 7, else $sin(x)$ will be returned

9.8 Conditions with the if-element

To evalue subobjects depending on variables, the if-element can be used.

Syntax: $Obj = if(Condition, Obj_1, Obj_2)$

If the condition is true, the first subobject Obj_1 will be evalued, else, if the condition is false, the second subobject UF_2 will be evalued.

The following conditions $Condition$ are available (UF_i are subfunctions as described for function element in section 9.7 on page 17):

- $true$: the condintion is true
- $false$: the condintion is false (=not true)
- $not(Condition_1)$: the condintion is true, if the $Condition_1$ is false (respectively not true), else the condition is false
- $or(Condition_1, Condition_2)$: the condintion is true, if one or more of the conditions $Condition_1$ or $Condition_2$ are true, else the condition is false
- $and(Condition_1, Condition_2)$: the condintion is true, if both of the conditions $Condition_1$ and $Condition_2$ are true, else the condition is false
- $xor(Condition_1, Condition_2)$: the condintion is true, if exactly one of the conditions $Condition_1$ or $Condition_2$ is true, else the condition is false
- $eqInt(UF_1, UF_2)$: the condintion is true, if the rounded to an integer value of the subfunction UF_1 is equal to the rounded to an integer value of the subfunction UF_2, else the condition is false; The direct comparison of floating point numbers was rejected, because due to slightly rounding errors the condition can lead to different results on different systems.
- $lo(UF_1, UF_2)$: the condintion is true, if UF_1 is smaler than UF_2 ($UF_1 < UF_2$), else the condition is false
- $gr(UF_1, UF_2)$: the condintion is true, if UF_1 is greater than UF_2 ($UF_1 > UF_2$), else the condition is false

Notes: The if construct is one of the most powerful programming language constructs, so it was included in the Fib multimedia description language.

Examples:

- $Obj = if(lo(x, 7), Obj_1, Obj_2)$; Obj_1 will be evalued, if the variable x is smaler than 7, else Obj_2 will be evalued
- $Obj = if(and(gr(add(x, 2), 3), lo(x, 7)), Obj_1, Obj_2)$; Obj_1 will be evalued, if the variable x plus 2 is greater than 3 and x is smaler than 7 (thus if x is a number betwean 1 and 7), else Obj_2 will be evalued

9.9 Call external objects

External objects are Fib objects that are not defined in the current Fib subobject. These can be from a root-element (root) or from the Fib object database (see section 10 on page 53). In this way, parts of Fib objects can be used multiple times in the Fib object or can be reused for different Fib objects.

Syntax:

$$\begin{aligned} Obj \quad = \quad & obj(Identifier, (V_1, \ldots, V_n), (OutVar_1, \ldots \\ & , OutVar_{v_1}, Obj_1), \ldots, (OutVar_1, \ldots, OutVar_{v_m}, Obj_m)) \end{aligned}$$

Description of the elements:

- $Identifierer$: Identify (a unique integer) for the Fib object, which is to be used. Only Fib objects that come after the current Fib object should be used (to avoid recursion), in which the Fib objects from the root-elements come first, and then the Fib objects of the database. Of the root-elements, only the root-elements are examined, which contain the current Fib object, but no root-elements, which do not contain the current Fib object. When the external object comes from the database, the $Identifier$ is negative, otherwise it is positive. While searching for the external objects (with the $Identifier$) in the root-elements, first the root subelements of the next root-element $root_1$ will be searched, in which the Fib object occurs, which requires the external object. In it ($root_1$) only subroot-elements will be checked, which stands after the root subelement, which contains the external object element. So first the $Identifier$ in the next root-element $root_1$ from the external object element are investigated, and than the $Identifier$ after the root-element $root_1$ in the root-element, in which the investigated root-element $root_1$ stands. The section 9.14.4 on page 51 defines the order for the root-elements which determines, how they will searched through, when looking for a $Identifier$. You can also find their, which and in what order the root-elements will be checked while searching for an $Identifier$.

- $(V_1, \ldots, V_n)$: The vector with the input values, which are needed for the used Fib object.

- V_i: The are the input values, which are needed for the used Fib object. The input value V_i is for the i'th input variable of the corresponding root-element or the i'th input variable of the root-element will be set to the value V_i .

- Obj_o: The subobjects, that are needed for the extern object. All subobjects are resolved before the current object. The subobjects are ordinary objects, which themselves can contain external objects. All in the subobjects contained external objects are resolved in relation to the current object. Resolved here means, that they are loaded and integrated into the Fib object

(contained variables are assigned to existing variables), without evaluating them at this point. The subobjects must be provided in the root-element with the corresponding identifier $Identify$. In the corresponding root-element, the same number of output variable lists with the same number of output variables should be present. (see the root-element 9.14 on page 33)

- $OutVar_{g_o}$: The output variables, which are provided for the extern object Obj_o from the current Fib object. If an output variable is not yet provided, it is set to 0.

The number and sequence of the subobjects, input and output variables must match the definition of the external object (the root-element with the $Identifier$).

Notes: The external object (obj) element is after the root-element one of the most complicated Fib elements. But reusing subobjects or functions should be worth that effort.

Examples:

- $obj(-3, x, y)$: The root database object with the identifier -3 is used here. The variable x and y are the input parameters (e. g. they can determine the upper left corner wher the inserted object should be display).
- $obj(5, x_1, y_1, (x_2, y_2, r, pr((r, 0, 0)_{colorRGB}, p((x_2; y_2)))))$: The root object with the identifier 5 is used here. This object, can use the object $pr((r, 0, 0)_{colorRGB}, p((x_2; y_2))$, where the red fraction and the position of the point is determined by the output parameters / varaibles r, x_2 and y_2.

9.10 External subobjects

External subobjects are objects, that are already provided during the evaluation of the current Fib object. (See the subobjects Obj_o in section 9.9 on page 25)

Syntax: $Obj = sub(Number, (V_1, \ldots, V_n))$

Description of the elements:

- $Number$: The number of the external subobject (corresponding to an output variable vector in the next root-element), which is to be used here. The $Number$ can only be fixed integer value. The domain of the $Number$ is implicit in the number of external subobjects (or output variable vectors) of the next root-element (the root-element, in which the subelement appears in the main-Fib-object), see section 9.14 on page 33 . If the value of the specified number is outside the valid domain (this is equivalent to an error), it is rounded to the next value, which lies in the domain.

- $(V_1, \ldots, V_n)$: The vector with the output values, which are needed for the used Fib object.
- V_i: This are the output values, which are needed for the used Fib object. For them the output variables ($OutVar_{i_Number}$) in the im external object element (obj) are defined. The output values have the same number and order as the $Number$'th output variables in the corresponding external object element (obj). Nevertheless, if the number does not match (equivalent to a faulty Fib object), the variables that are too much are ignored, and for the missing variables, the zero value of the domain of the corresponding output variable in the root-element is used.

Notes: With external subobjects multimedia information can be provided for the Fib object, which can be used in certain places. This is useful for example, if the current Fib object, implements a calculation or function, which is more extensive and is required several times in other Fib objects.

If an external subobject dosn't exists for the evaluation of a root element, an empty point $p()$ (point without impact) will be used for it. external subobjects can even be useful for the top most root-element, if for example the multimedia object Obj is just a frame, in which an other picture can be displayed from an given start point (given by the output values).

Examples:

- $sub(1)$: At this point, the first external subobject of the next root-element is used.
- $sub(1, x, y, r)$: At this point, the first external subobject of the next root-element is used. The variables x, y and r are provided for the external subobject. These variables could for example be used for the position (x, y) and r for the amount of red of the point, which is used for the external subobject.

9.11 Retrieve domain properties

Status: not implemented, planned for implementation

This Fib element is used to retrieve parameters of the domains.

Syntax: $Obj = domainProperty(Variable, Type[.Element]*, Mode, Obj_1)$

Syntax: $Obj = dp(Variable, Type[.Element]*, Mode, Obj_1)$

Description of the elements:

- $Variable$: The variable, which the domain property element defines. If no domain parameter for the variable can be determined, the variable will be set to 0 .
- $Type$: The type of the domain whose parameter should be determined. Possible types are listed in table 5 on page 43.
- $Element$: If the domain is a vector domain (a domain for vectors), the number of the vector element, for which a property value should be returned, must be specified with $Element$. Ther can be more than one $Element$ parameters following each other, if some (more than one) vector domains are nested. Since the $Variable$ can only be assigned to scalars, no vector can be determined with the domain property element, but there have to always be a vector element selected. The counting of the vector elements of a vector starts at 1.
- $Mode$: Which property value of the domain is to be selected. Possible values are listed in table 4 .
- Obj_1: The subobject, for which the $Variable$ is defined and which will be evalued for the retrieved variable assignment of the domain property element.

For finding the correct domain first the domains of the next (higher) root-element are searched, than its value domains. If no appropriate domain is found in the first root-element the next (higher) (to the first root-element) root-element is searched and so on.

Notes: With this element properties of the environment can be obtained. This can be useful, if a multimedia object is to be scaled afterwards with a given dimension domain. The displayed objects can be automatically adjusted via the retrieval of the dimension domain properties in size and position.

Examples:

- $dp(x, dim.1, min, Obj_1)$: The variable x is set to the minimum value of the domain for the first dimension.
- $dp(x, property(colorRGB).2, max, Obj_1)$: The variable x is set to the maximum value of the domain for the color green (second vector element) of the RGB colors.
- $dp(x, property(colorGrayscale).1, size, Obj_1)$: The variable x is set to the size of the domain for SW colors. The 1 for the first vector element must be specified, because the domain for SW color is a vector, even if it contains only one element.
- $dp(x, matrix.3.2, null, Obj_1)$: The variable x is set to the zero value of the second subdomain of the third subdomain of the matrix domain.

Name	Value	Description
null	0	The zero value of the domain will be returned.
min	1	The minimum value of the domain will be returned.
max	2	The maximum value of the domain will be returned.
size	3	The size ($Maximum - Minimum$) of the domain will be returned.
scaling	4	The scaling factor of the domain will be returned. If the domain isn't scaled, 1 will be returned.
Unscaled(not scaled) values (In domains that are not scaled, these correspond to the scaled values (for example, then the return value of "unscaled min" is equal to that of "min")		
unscaled null	10	The unscaled zero value of the domain will be returned.
unscaled min	11	The unscaled minimum value of the domain will be returned.
unscaled max	12	The unscaled maximum value of the domain will be returned.
unscaled size	13	The unscaled size ($unscaled\ \ maximum - unscaled\ \ minimum$) of the domain will be returned.

Table 4: Possible retrievable properties of a domain

9.12 Set-element

The set element assigns sets of values in succession to a number of variables. The variables apply everywhere in the subobject.

Syntax:

$$\begin{aligned} Obj &= set((Variable_1, \ldots, Variable_n), [DomainNr,] \\ &\quad ((W_{1.1}, \ldots, W_{n.1}), \ldots, (W_{1.k}, \ldots, W_{n.k})), Obj_1) \end{aligned}$$

Description of the elements:

- n: The number of elements of a set. The minimum number is one element. ($n \geq 1$)
- k: The number of sets of values, with which the variables will be set. The minimum number is one set. ($n \geq 1$)
- $Variable_i$: The variables, which the set element defines.
- $DomainNr$: This is the number of the domain for the set-element. This information is optional, the default value is 0. If no set-domain with that number exist, the set-domain with the next smallest number is used. By using different, adapted for each set-element, domains, the memory requirements for storing in the compressed Fib format can be optimized.
- $W_{i.g}$ with $(i = 1 \ldots n)$ $(g = 1 \ldots k)$: This are the to set values for the variables.
- $(W_{1.g}, \ldots, W_{n.g})$: The vector with the sets of values to set.
- Obj_1: The subobject, for which the variables $Variable_i$ are defined and which will be evalued for every variable assignment.

The variables $(Variable_1, \ldots, Variable_n)$ are sequentially assigned to the individual sets of values of $(W_{1.g}, \ldots, W_{n.g})$. Where the variable $Variable_i$ will only be set to the values $W_{i.g}$ $(g = 1, \ldots k)$. Thus there are k bindings of variables, in which the variable $Variable_i$ is first set to the value of $W_{i.1}$, after this to $W_{i.2}$, etc. . If an element $W_{i.g}$ is a variable, the $Variable_i$ will be set to the value of variable of $W_{i.g}$.

Example:

- $set((x, y), ((1, 2), (3, 8), (3, -8)), Obj)$; In this example the variables x and y will be set for the subobject Obj sequentially to the values $(x = 1, y = 2)$, $(x = 3, y = 8)$ and than $(x = 3, y = -8)$.

- $set((x, y, z), 3, ((1, 2, 3), (3, g = 8, 15), (3, -8, b = -1)), Obj)$; In this example the variables x, y and z will be set for the subobject Obj sequentially to the values $(x = 1, y = 2, z = 3)$, $(x = 3, y = 8, z = 15)$ and than $(x = 3, y = -8, z = -1)$. The domain for the set-element is the third set-domain.

Note: Not all dependencies of several variables can be easily represented by functions. Therefore, the set element offer the ability to assign multiple variables sequentially to sets of values.

It is conceivable, for example, to create a database object, which codes a character set ("font"). With the input parameters/variables of this database object the letter and the display position of the letters are assignable. However, it can not be assumed, that the input variables for the letters of a text are in a simple functional dependenc. With the set element the input variables can be easily assigned to the values for each letter of the text.

9.13 Matrix element

The matrix element represents a matrix. The matrix element works similar to the set element, except that a number of counter variables is generated automatically.

Syntax:

$$\begin{aligned} Obj \;=\; & matrix((Variable_1, \ldots, Variable_d, Variable_{d+1}, \ldots, Variable_{d+i}), \\ & [DomainNr,] \\ & ((Startvalue_1, Endvalue_1), \ldots, (Startvalue_d, Endvalue_d)), \\ & ((W_{1.1}, \ldots, W_{i.1}), \ldots, (W_{1.k}, \ldots, W_{i.k})), Obj_1) \end{aligned}$$

Description of the elements:

- d: Number of dimensions of the matrix ($d \geq 1$)
- i: Number of values per set of values ($i \geq 0$)
- k: Number of sets, with which the variables are set ($k \geq 0$)
- $Variable_v$: The variables, which the matrix element defines
- $DomainNr$: This is the number of the domain for the matrix element. This information is optional, the default value is 0. If no matrix domain with that number exist, the matrix domain with the next smallest number is used. By using different, adapted for each matrix element, domains, the memory requirements for storing in the compressed Fib format can be optimized.

- $(Startvalue_h, Endvalue_h)$: vector for the area for the (/ matrix size in) dimension h
- $Startvalue_h$: Start value of the counter variable for the h'th dimension
- $Endvalue_h$: End value for the counter variable for the h'th dimension
- $W_{a.b}$ with $(a = 1 \dots i)$ and $(b = 1 \dots k)$: This are the to set values or variables, whith values to set
- $(W_{1.b}, \dots, W_{n.b})$: Vector of the to set values
- Obj_1: The subobject, for which the variables $Variable_i$ are defined and which will be evalued for every variable assignment.

The matrix element represents a matrix with d dimensions, which elements are sets of i values.

In the matrix element each dimension / counter / index variable $Variable_c$ (with $c = 1 \dots d$) goes through all integers of the corresponding area $Startvalue_h$ to $Endvalue_h$. For each integer value of the $Variable_h$ all integer values of the variable $Variable_{h-1}$ will be set. For each value allocation of the dimension variables $(Variable_1, \dots, Variable_d)$ the value variables $(Variable_{d+1}, \dots, Variable_{d+i})$ will be set to the next set of values $(W_{1.b}, \dots, W_{i.b})$. This continues until either the dimension variables $(Variable_1, \dots, Variable_d)$ have gone through all of their values or there is no next set of values $(W_{1.k+1}, \dots, W_{i.k+1})$. If an element $W_{a.b}$ is a variable, so the $Variable_{d+a}$ will be assigned according to the value of the variable $W_{a.b}$.

If there are no value variables ($i = 0$), just all the values of the dimension variables $(Variable_1, \dots, Variable_d)$ will be set and the sets of values will be ignored.

In listing 1 the operation of the matrix element is shown with C-pseudo code.

Listing 1: Pseudo algorithm of the matrix element

```
1  void evalue( matrix ){
2     a = 1;
3     for ( int Variable_d = Startvalue_d; Variable_d <= Endvalue_d;
          Variable_d += 1 ){
4        ...
5           for ( int Variable_1 = Startvalue_1; Variable_1 <=
               Endvalue_1; Variable_1 += 1 ){
6              Variable_{d+1} = W_{1.a};
7              ...
8              Variable_{d+i} = W_{i.a};
9
10             obj_1( Variable_1, ..., Variable_{d+i} );
11
12             a++;
13             if ( (k < a) && (0 < i) ){
14                return;
```

```
15                  }
16              }
17          ...
18      }
19  }
```

Example:

- $matrix((x, y, w), (1, 3), (1, 3), ((11), (12), (13), (21), (22), (23), (31), (32), (33)), Obj_1)$: In this example the variables x, y and w will be set for the subobject Obj sequentially to the values: $(x = 1, y = 1, w = 11)$, $(x = 2, y = 1, w = 12)$, $(x = 3, y = 1, w = 13)$, $(x = 1, y = 2, w = 21)$, $(x = 2, y = 2, w = 22)$, $(x = 3, y = 2, w = 23)$, $(x = 1, y = 3, w = 31)$, $(x = 2, y = 3, w = 23)$, $(x = 3, y = 3, w = 33)$

- $matrix((x, w), (1, 5), ((1), (2), (3)), Obj_1)$: In this example the variables $x$and w will be set sequentially to the values: $(x = 1, w = 1)$, $(x = 2, w = 2)$, $(x = 3, w = 3)$ end, because no other assignments for w exists

- $matrix((x, w), (1, 2), ((1), (2), (3), (4)), Obj_1)$: The variables x and w will be set sequentially to the values: $(x = 1, w = 1)$, $(x = 2, w = 2)$ end, because no other assignments for x exists to be set

- $matrix((x, y), (2, 4), (3, 4), (), Obj_1)$: The variables x and y will be set sequentially to the values: $(x = 2, y = 3)$, $(x = 3, y = 3)$, $(x = 4, y = 3)$, $(x = 2, y = 4)$, $(x = 3, y = 4)$, $(x = 4, y = 4)$

Note: Matrices and vectors (one-dimensional matrices) are often used in mathematics, computer science, image processing. With the matrix element for example subimages can directly be specified as a raster image or the values of a flow chart can be given directly.

9.14 The root-element

The root-element serves as the root-element of a Fib object. It should provide all (enviroment) informations that are needed to evaluate the Fib object. The root-element itself can just be contained in other root-elements, but not in other Fib elements.

Syntax:

$$\begin{aligned} Rootobj \;=\; & root([Multimediainformation], [Domains], \\ & [DomainsValues], [((InVar_1, S_1), \ldots, (InVar_v, S_v))], Obj, \\ & [((Identifier_1, Rootobj_1), \ldots, (Identifier_n, Rootobj_n))], \\ & [(DB\_Identifier_1, \ldots, DB\_Identifier_d)], [Optionalpart]) \end{aligned}$$

All elements, except the actual multimedia object (main-Fib-object) Obj, are optional and can therefore be omitted.

Description of the elements:

- Obj: The actual multimedia object (main-Fib-object), which is to be displayed. This in turn can represent more multimedia objects (e. g. images).
- $(Identifier_i, Rootobj_i)$: $Identifier_i$ is an identifier (a unique natural number), which can be used in an external part object (see obj element, in section 9.9 on page 25). $Rootobj_i$ is the associated external subobject (which itselves begins with a root-element). If in the Obj (main-Fib-object) an external object is not resolved, the $Identifier_i$ will be searched for the correct/same identifier as that of the external object and, if one is found, the associated $Rootobj_i$ will be used for (/resolved as) the external object. The Fib object, which is the external object, is always the basis for the search. In the search, first the next root-element, in which the basis Fib object is contain, will be searched, and then successively the root-element, which contains the last searched root-element. If in all the searched root-elements till the Fib object root (in which all the external / Fib object exists) no matching identifier is found, the database will be searched. If there also no matching identifier can be found, the empty point $p()$ will be used for the external object.
- $MultimediaInformation$: The $MultimediaInformation$ contains the necessary information for the evaluation of the multimedia data (see section 9.14.1 on page 35), this information are for example the Fib version and database version.
- $Domains$: This element contains the domains for the values and variables in the different vectors (e. g. position and property vectors) and Fib elements (see section 9.14.2 on page 35). Thus the number of assignment possibilities for each item is restricted, in order to determine in advance, if the available hardware for displaying can view the multimedia object.
- $DomainsValues$: Contains the domains of the values in the different vectors (e. g. position and property vectors) and Fib elements (see section 9.14.2 on page 35). This limits the number of possible values and therefore the number of bits to store them, it makes it possible to achieve a better compression.
- $InVar_i$: The $InVar_i$ are the input variables for the actual multimedia object (main-Fib-object) Obj. These are defined by the root-element for the main-Fib object Obj and are set by external sources (such as the external object in a Fib object). Even the top most root-element can contain input variables, these represent degrees of freedom for the multimedia data, which have to be set by external programs, e. g. to combine in a Fib object an archive of

images, whose partial images can be selected with an input variable $InVar_i$, or to choose different languages. The domain of each input variable should be specified in the $Domains$ part. The default domain for input variable is $integerB(16)$. If in the evaluation for an input variable $InVar_i$ no value is provided, it is set to its default value S_i.

- S_i: S_i is the default value of the input variable $InVar_i$. If in the evaluation for an $InVar_i$ no value is specified, the $InVar_i$ is set to the value S_i. The value S_i should be part of the domain for $InVar_i$.
- $DB\_Identifier_i$: These list can specify the identifiers of the database objects, which are used in the main-Fib-object Obj
- $Optionalpart$: In this fild optional information is stored, which is not required for evaluation of the multimedia data (see section 9.14.3 on page 48). These can include the Copyrigth, author or description texts.

9.14.1 Multimedia information

The element $MultimediaInformation$ contains information, which is absolutely necessary for decoding the multimedia object and which the output device have to be able to process.

Syntax: $MultimediaInformation = (Fib\_Version, Db\_Version)$

Description of elements:

- $Fib\_Version$: An natural number for the version of the Fib multimedia description language, with which the multimedia object is coded. This number will be incrased by one with every new version of the Fib multimedia description languag.
- $Db\_Version$: An natural number for the version of the Fib database, which is required for decoding the multimedia object. This number will be incrased by one with every new version of the Fib database.

The version numbers can be mapped to a human-readable form (such as “Fib V1.2.3”). But in the $MultimediaInformation$ element only a integer is used, otherwise a particular form would be defined, which would be very hard to changed in the future. A human-readable form of the version numbers may be specified in the optional part (see section 9.14.3 on page 48).

9.14.2 Domains

$Domains = ((Name_1, Dom_1), \ldots, (Name_n, Dom_n))$

$DomainsValues = ((Name_1, Dom_1), \ldots, (Name_k, Dom_k))$

There are two types of domain information:

- $Domains$: Domains for possible values (including values of variables) an element can contain.
- $DomainsValues$: Domains for used values in elements.

The types of the domains will be discussed below in more detail.

The specification of a domain consists of two parts $Name_i$ and Dom_i.

The first part of the domain entry is the name $Name_i$ of the element, for which the domain is. Elements/names that can be set for the domains are listed in table 5 on page 37.

Also listed there are the standard domains of the elements. Of the listed standard domains the scalar (sub-)domains can be changed, the number of scalar subdomains of a domain, however, should not be changed (if not specified otherwise). So if the standard domain of an element is for scalars values (e. g. $IntegerB$), for this element never a vector domain should be set. On the other hand, if the standard domain of an element is for vectors with n elements, for this element only domains should be set, which are for vectors with n elements. In this way, for example, properties can be always interpreted in the same way. For elements (e. g. Properties), which are not listed in table 5, the domains can of course be set freely.

The second part of the domain entry is the domain Dom_i to apply to the element $Name_i$. A list of the possible domains is given in table 6 on page 45 .

Values outside of their domain are rounded to values in the domain. The rounded value of a (scalar) number, is the value with the minimum distance to the not rounded value. If there are several of them, the smallest value of them is taken, as the rounded value.

Rounding of vectors is done with their distance. The distance between two vectors is the sum of the distances of their elements. The rounded vector of a domain is the vector in the domain with the smallest distance from the unrounded vector. If several vectors have a minimum distance from the unrounded vector, the rounded vector is the vector, whose first n elements have a minimal distance from the unrounded vector. Where n, starting from the number of elements in the unrounded vector, is reduced by one till a vector is found or 1 is reached. If even then more vectors remain, the rounded vector is the vector, whose k'th element is less than the k'th element of the other remaining vectors, where k is counted from 1 to the number of elements in the vector.

If the vector to be rounded has too many elements, the number of its elements is reduced to the correct number. Should the to round vector contain not enough elements, the missing elements are created and assigned to the zero value of the respective element domains.

Each domain has a null value. The null value of a scalar domain (a domain for numbers), is the value 0 rounded to a value in the domain. For vectors the null vector is the vector, which is generated if a vector whose elements are all 0 is rounded to a vector in the domain.

Each scalar domain has a maximum value and minimum value. The maximum value is the highest value in the domain and the minimum value is the smallest value.

The domains, which contain only integers, have a scale factor $Scale_i$, which apply for the corresponding element of the $Name_i$. This factor $Scale_i$ is a floating point number, with which the values of the unscaled domain are multiplied, to obtain the scaled domain. As standard, the scaling factor 1 is assumed. The scaling factor is useful for example, if the horizontal dimension should not be measured in meter m but in milli meter mm or when the temperature is not given in degrees Kelvin, but in tenth of a degree Kelvin.

In the following notation the scale factor is indicated by the notation, that the domain is multiplied with it "$*$". A Skalierungsfaltor of 1 is not specified here. For example, "$integerB(16) * 2^{-16}$" means that the 16-bit integer domain is scaled with the scaling factor 2^{-16}. In contrast "$integerB(16)$" means that the scaling factor is 1, that is the domain is not scaled.
Example: The basis domain is for integers from 0 to 3 (which are 4 values) with the scaling factor $F = 1/2$. Possible values are: $\{\ 0 * 1/2 = 0;\ 1 * 1/2 = 0.5;\ 2 * 1/2 = 1;\ 3 * 1/2 = 1.5\ \}$. If a number in an element for the domain is for example unscaled 1, it is interpreted as the scaled number 0.5 .

Name	Description	Standard domain
dim(Dim, $Dimmap_1$, ..., $Dimmap_{Dim}$)	This is the domain for position vectors. The number Dim is the number of dimensions respectively elements of a position vector. The value $Dimmap_i$ is an integer, which determines, in which direction the dimension i will be mapped (see table 7 on page 46 for the possible values, there numerical values are specified [in the "value" column]). The domain is a vector with Dim numbers (or scalars). The default value for the number of dimensions Dim is 2, it will be used, if no domain is specified for position vectors. Then (standard case) the first dimension points in the horizontally direction and the second dimension in the vertically direction. Regardless of this, the direction value $Dimmap_i$ of the domains for elements $Domains$ (also inherited) overrides the value domains $DomainValues$. Normally, however, the direction specifications should be the same for all position vector domains.	$vector(2, integerB(16), integerB(16))$
subfunction	values which can appear in subfunctions	$integerB(16)$
area	domain for the area element (see section 9.6 on page 16)	$vector(2, naturalNumberB(8), vector(2, integerB(16), integerB(16)))$
inVar(i)	domain for the i'th input variable ($InVar_i$)	$integerB(16)$
set	zeroth domain for set-elements	$vector(3, naturalNumberB(8), naturalNumberB(32), vectorOpenEnd(integerB(32)))$
set(i)	i'th domain for set-elements	$vector(3, naturalNumberB(8), naturalNumberB(32), vectorOpenEnd(integerB(32)))$

Name	Description	Standard domain
matrix	zeroth domain for matrix elements	$vector(4, naturalNumberB(8),$ $naturalNumberB(32),$ $vector(2, integerB(8),$ $integerB(8)),$ $vectorOpenEnd($ $integerB(32)))$
matrix(i)	i'th domain for matrix elements	$vector(4, naturalNumberB(8),$ $naturalNumberB(32),$ $vector(2, integerB(8),$ $integerB(8)),$ $vectorOpenEnd($ $integerB(32)))$
externSubobject(Number)	The domain for the external subobject with the given number (see section 9.10 on page 26) of the multimedia object. The output values of the external subobject with the given number can be set to the values of this domain. In the main-Fib-object Obj for each externSubobjectOutput domain external subobjects with the given number can exist. Also for each external subobject (each number) in the main-Fib-object Obj an externSubobjectOutput domain with the number should exist. At least for the external subobjects with the highest number an externSubobject domain have to be created, so that it is known, how much external subobjects in the main-Fib-object Obj exists. The externSubobjectOutput domain isn't inherited by other root-elements.	$vector(0)$
property(Name)	domain of the properties for the given $Name$; see table 2 on page 12 for the possible names ($Name$)	dependent of the type ($Name$) of the property
property(whatever)	The properties of the points dosn't matter. Whatever properties are assigned to a point with this property, they are correct.	$vector(0)$

Name	Description	Standard domain
property(colorRGB)	domain for RGB-colors	$vector(3,$ $integerB(8), integerB(8),$ $integerB(8))$
property(col- orGrayscale)	domain for monocrome colors	$vector(1, integerB(8))$
property(layer)	number of possible layers	$vector(1, integerB(4))$
property(transparency)	transparency of the points	$vector(1, integerB(8))$
property(persistent)	This property is only useful for a period of time. Points in space with this property only lose their other properties, if they are overwritten by a respective property of the same type. This of however is only the case, as long as the particular point has the property $persistent$. This property is for example useful, if in a movie objects should be visible as long as they are not overwritten by other objects. In this case, the entire background can get the property $persisted$. If an object is defined and displayed at a time, it will be there in the future as long as it won't get overwritten. (The possible values are 1 for persistent and 0 for non persistent.)	$vector(1, integerB(1))$
property(sound)	a sound; the values are: 1. frequency in Hertz $(1/s)$, 2. sound pressure in Pascal Pa $(1Pa = 1N/m^2)$, 3. phase shift in radians, 4. duration in seconds; a sound is additive to other sounds	$vector(4,$ $integerB(16),$ $integerB(32), integerB(16) *$ $2^{-16}, integerB(32) * 2^{-8})$

Name	Description	Standard domain
property(sound-Polarized)	a sound; the values are: 1. frequency in Hertz ($1/s$), 2. sound pressure in Pascal Pa ($1Pa = 1N/m^2$), 3. phase shift in radians, 4. duration in seconds; r = 5 to ($3 + \sharp Dim$) polarization fraction (as an angle in radians) in the dimension plane, which is spanned by the respective dimensions $r-4$ and $r-3$ ($\sharp Dim$ is the number of dimensions), the angle origin is the $r-3$ axis and goes in positive direction; a sound is additive to other sounds	$vector(3 + Dim, integerB(16), integerB(32), integerB(16) * 2^{-16}, integerB(32) * 2^{-8}, [integerB(16) * 2^{-16}]^{Dim-1})$
property(soundAmplitude)	the amplitude of a sound; the values are: 1. sound pressure in Pascal Pa ($1Pa = 1N/m^2$), 2. phase shift in radians, 3. duration in seconds; a sound is additive to other sounds; With this properties sounds can be build by their amplitude with a specific sampling rate, such as in the WAVE file format.	$vector(3, integerB(32) * 2^{-16}, integerB(0) * 1/44100, integerB(8) * 2^{-16})$
property(soundBarrier)	speed of sound in meters per second (m/s); With this property objects can change the acoustics.	$vector(1, integerB(16))$
property(soundReflected)	fraction of sound reflected from the object; This property applies to the surface/the edge of the object and not for all his individual points	$vector(1, integerB(16) * 2^{-16})$
property(sound-Damping)	fraction of the sound swallowed by a point	$vector(1, integerB(16) * 2^{-16})$
property(kelvin)	temperature in Kelvin	$vector(1, integerB(16) * 2^{-4})$

Name	Description	Standard domain
property(electro-Magnetic)	an electromagnetic radiation source, the values are: 1. frequency in Hertz $(1/s)$, 2. amplitude in Candela cd, 3. phase shift in radians, 4. duration in seconds, r = 5 to $(3 + sharpDim)$, r = 5 to $(3 + \sharp Dim)$ polarization fraction (as an angle in radians) in the dimension direction, which is spanned by the respective dimensions $r-4$ and $r-3$ ($\sharp Dim$ is the number of dimensions), the angular information is provided by the $r-3$ axis in positive direction, an electromagnetic wave is additive to other electromagnetic waves	$vector(3 + Dim, integerB(64), integerB(64) * 2^{-16}, integerB(16) * 2^{-16}, integerB(32) * 2^{-8}, [integerB(16) * 2^{-16}]^{Dim-1})$
property(periodBegin)	Time in seconds (s) from the beginning of the whole multimedia object, from which the object is to be displayed; if possible, this property should be near the root of the multimedia object; when a multimedia object is played it can be determined with this property: the order in which subobjects should be evaluated and/or till which time to evaluate a part object	$vector(1, integerB(24) * 2^{-8})$
property(periodEnd)	Time in seconds (s) from the beginning of the whole multimedia object, till which the object is to be displayed; if possible, this property should be near the root of the multimedia object and follow a "periodBegin" property; when a multimedia object is played it can be determined with this property: the order in which subobjects should be evaluated and/or till which time to evaluate a part object completely	$vector(1, integerB(24) * 2^{-8})$

Name	Description	Standard domain
property(evaluationTime)	Time required for evaluating a multimedia object, in proportion to a multimedia object, which contains only one point (the value should be seen as a multiple of the evaluation time of a point); with this property in combination with the properties "periodBegin" und "periodEnd" a good evaluation order and time can be evalued for the partobjects, when playing a multimedia object; this property should stand immediately after (or below/within) the "periodBegin" and "periodEnd" properties	$vector(1, integerB(24))$
property(checksum)	A checksum for the object will be generated. The first parameter determines the type of the checksum. The second parameter specifies any which number of bits, a checksum is to be generated, and the third parameter defines how many bits the checksum is long. The last block of the checksum will be padded with 0 after loading the blocks, so that it too has the desired length. If there are enough bits to correct an existing error, it will be attempted to correct the error. (see section 21.3.3 on page 104)	$vector(3, integerB(4), integerB(8), integerB(8))$
property(boundSize)	For the part object the border/size in bits will be stored, when saving it. If an error occured while loading the part object, the (in the bitstream after the faulty part object) following part objects can still be loaded, because their beginning is known. (see section 21.3.3 on page 105)	$vector(0)$

Table 5: Names of elements for domains

Domain name	Description	Domain
naturalNumberB(X) * S	X bit integers without sign (natural numbers), which are scaled with S	$0 \ldots S * (2^X - 1)$
integerB(X) * S	X bit integers with sign, which are scaled with S	$S * -(2^{X-1}) \ldots S * (2^{X-1} - 1)$
naturalNumberB(32)	32 bit integers without sign (the scaling factor is not shown, because it is 1)	$0 \ldots 4294967295$
integerB(32)	32 bit integers with sign	$-2147483648 \ldots 2147483647$
naturalNumber(X) * S	integers in the range from 0 to X (without sign), which are scaled with S	$0 \ldots S * X$
integer(X, Y) * S	integers in the range from X to Y, which are scaled with S	$S * X \ldots S * Y$
integerValues($X_1, \ldots, X_N$) * S	all given integers X_i, which are scaled with S	$\{S * X_1, \ldots, S * X_N\}$
naturalNumberUL() * S	unlimited positiv integers, which are scaled with S	all positiv natural numbers
integerUL() * S	unlimited integers, which are scaled with S	all integers
real(D_M, D_E)	The domain for floating point numbers. A floating point number consists of two integer fields, the first is for the exponent E and the second for the mantissa M. The floating point number Z is then $Z = M * 2^E$. The domains D_M and D_E are the domains for the integers, defined like above (e. g. integerB(8)). The domain D_M is the domain for the values of the mantissa and D_E is the domain for the values of the exponent.	$\{min(D_M) * 2^{min(D_E)}, \ldots, max(D_M) * 2^{max(D_E)}\}$
realValues($X_1, \ldots, X_N$)	all given floating point numbers X_i	$\{X_1, \ldots, X_N\}$
realUL()	all floating point numbers	all floating point numbers
vector($E, D_1, \ldots, D_E$)	a vector with E elements, in which the i'th element has the domain D_i	$(D_1, \ldots, D_E)$

Domain name	Description	Domain
vectorValues($D_1, \ldots, D_T$; $V_1, \ldots, V_n$)	The basis domain are vectors. The given vectors V_i are all possible vectors in the domain. The domain D_i is the domain of the i'th vector element (if it isn't a variable). If a variable is given as a vector elements the varible identifier is left out (omitted).	$\{V_1, \ldots, V_n\}$; for $i = 1, \ldots, n$ $V_i = (E_1, \ldots E_T)$; with $E_f \in D_f, f = 1 \ldots T$
vectorOpenEnd($E, D_1, \ldots, D_E$)	This is the domain for vectors with E or more elements, in which the i'th element has the domain D_i, for i lower E, and else the domain D_E. This domain is for elements, which can contain vectors of diffent sizes. The number of elements in the vector should be determined by the containing Fib element.	$(D_1, \ldots, D_E, \ldots, D_E)$
domainReference(Name.[Element]*)	The domain is the domain of the Element of the given domain name $Name$. With the $Element$ parameters a subdomain can be choosen (e. g. $set.3.1$ for the first subdomain of the third subdomain for the set-element or $property(colorRGB).2$). If no domain for the element with the $Name$ exist, the default domain of the element is taken. If there is not even standard domain the general standard domain is used. If ther is no subdomain for $Element$, the last subdomain for $Element$ is used, that still exists.	the ($Element$ sub-)domain for the element with the domain name $Name$
defaultDomain(Domain)	The domain $Domain$ is used, if no other domain exists for the corresponding element.	$Domain$ if no other domain exists for the corresponding element

Table 6: Possible domains

Name	Value	Description
none	0	No mapping, respectively it will be mapped to nowhere.
horizontal	1	The dimension goes in the horizontal direction. Lower values are left.
vertikal	2	The dimension goes in the vertikal direction. Lower values are down.
depth	3	The dimension goes in the direction of depth. Lower values are in direction back.
time	4	The dimension stands for the time. Lower values are in the past.
anywhere	16	The dimension can be mapped in every dimension or direction.
Product Dimensions	256 to 511	Dimension which are product specific. Different producers can use these dimensions, without getting incompatible with later defined dimensions.

Table 7: Directions for dimensions

Domains for Elements With the domains for elements $Domains$, all possible values are specified, which an element may take, no matter whether this value is explicitly specified or is given by a variable. So the $Domains$ specify all values, that can be taken by elements. The minimum and maximum values are determined with it. The presentation of the multimedia device may be adapted/scaled to these values. If for example, a film should be displayed on a 100 cm (centimeters) wide screen display and the dimension of the horizontal direction is given as from -5 m (meters) to +5 m, then points at the horizontal value -5 m in the Fib object will be displayed at 0 cm on the screen and +5 m will be displayed at 100 cm (0 m at 50 cm, etc.).

For the display the domains of the top most root-element are crucial. Contained root-elements can also specify the domains, but these are ignored when the Fib object is display. Therefore, the top most root-element must specify all domains relevant for the displaying the Fib object.

The in $Domains$ contained domains are inherited by the in the current root-element contained root-elements. Contained root-elements may also overwrite all inherited domains. The input variables $InVar_i$ are root-element specific (they are not valid in any contained root-elements), their domains are therefore not inherited, because this is neither reasonable nor desirable.

When a domain for an element of the Fib object is not given or inherited, it has as its domain its default domain. If there is no default domain for a particular element, its default domain is $integerB(16)$.

Another special feature arises, when the database objects inherit domains. Even

for root-elements of database objects, it is useful to adopt domains for some elements. This is usfull for example for the dimensions of a database object, which should eventually be adopted to the displayed multimedia object. If for example a database object is a general triangle, wher its size and key points are set by input parameters, the dimensions for this database object can not be determined in a reasonable manner. For this reason, the inheritance of domains for a database object will be treated as if the database object is a sub-root-object in the root-element, in which the main-Fib-object uses the database object. Thus the database object inherits the domains, which are also valid for the external subobject element, for which it is used.

Thus in the top most root-element of a Fib object (such as a database object), domains can be omitted if they are determined by the calling/using Fib objects or mechanism. For these Fib objects the corresponding value domains $DomainsValues$ should be given for the missing domains in their top most root-element, in order to be able to store the objects.

Domains for values In the domains of the element values $DomainsValues$, all values are given, which may be take by an actual value in an element. These domains are therefore not for values, which are supplied by variables. The list of $DomainsValues$ with domains for values determines the storage space requirements for individual values and is therefore hereinafter also called **memory domains**. Since the values in the elements, may not take all values in the domains, for example, if mostly variables are used in the elements, the number of different actual values in the elements may be well below the possible values in the domain of the element $Domains$. It is for example possible that a RGB color image with $256 * 256 * 256 = 16,777,216$ possible colors only use 16 colors. To store the 16,777,216 colors of the domains 24 bits are needed, but for the 16 colors, only 4 bits are enough (similar to indexed colors in images).

The domains for values $DomainsValues$ are inherited from the domains for elements $Domains$ of the same root-element. If for an element no domain in $DomainValues$ is given, the domain of $Domains$ is taken for the values of the element. The domains, which $Domains$ inherits, are passed down to $DomainsValues$. Precedence in inheritance, however, have from the same root-element (such as the $Domains$) inherited $DomainsValues$âĂŃ. Inherited will be always the domain for an element, which appears next in the current or next higher root-elements.

Or also: For a root-element the $DomainValues$ will be inherited from the specified domains of $Domains$ of the same root-element. Still missing domains in $DomainValues$ will be inherited from the, possibly inherited, domains of the $DomainValues$ of the next higher root-element.

The in $DomainsValues$ specified domains must be compatible with the corresponding domains in $Domains$. For example, the number of elements in a vector in a domain for an element in $DomainValues$ have to be the same as in the corresponding (possibly inherited) domain in $Domains$. In particular, a domain for

values from $DomainsValues$ for an element has to be a subset of the (general) domain of $Domains$ for the element. (Elements may only take values, for which they are defined.)

9.14.3 Optional part

The $Optionalpart$ contains values, which are not necessary for the display of multimedia information. These optional data can be for example: version of the generator, Copyrigth, author, creation date, descriptions, texts, names and comments (which are outsourced here, because of the compression). To save memory space, the $Optionalpart$ can be compressed or even entirely omitted.

Syntax: $Optionalpart = ((Key_1, Value_1), \ldots, (Key_n, Value_n))$

Description of the elements:

- Key_i: This is the key of the ith information set. Some possible keys are listed in table 8. The key should not start with a "@", this character is reserved for outsourced data.
- $Value_i$: This is the value of the ith information set.

Key	Description	Example
author	author of the Fib object	(author, "Oesterholz")
author::email	E-mail address of the author	(author::email , "author@Fib-development.org")
author::adress	address of the author	(author::adress , "Example City 123456; Example Road 13")
author:: telephon	phone number of the author	(author::telephon, "123/456/789124")
type	type of object	(type, "tree")
description	description of the object	(description, "this is me while fishing")
name	the name of the object	(name, "Eifelturm")
copyright	copyright of the object	(copyright, "GPL3 ...")
version::fib	This is the human readable form of the version of the Fib multimedia language, which is required for loading the Fib multimedia object.	(version::fib, "Fib V1.2.3")
version::fibDb	This is the human readable form of the version of the Fib database, which is required for loading the Fib multimedia object.	(version::fibDb, "Fib DB V1.2.3")
version::-enviroment	This is the human readable form of the version of the genetic algorithm, which was used to code the Fib multimedia object (see Section III on page 71).	(version::enviroment, "Fib Env V1.2.3")

Key	Description	Example
inVarX::N::Art	This are the information for the Fib object, if the input variable $InVar_X$ will be set to the value N.	(inVar1::17::..., "...")
inVarX::N::-description	The value contains description for Fib object, if the input variable $InVar_X$ will be set to the value N.	(inVar2::8::description , "This picture shows the weather on the third day.")
inVarX::N::typ	The entry specifies the type of Fib object, if the input variable $InVar_X$ will be set to the value N.	(inVar2::2::typ , "picture")
inVarX::N::-identifier	The value is the identifier of the root-object, which is used, when the input variable $InVar_X$ will be set to the value N. The main-Fib-object should call the appropriate root-object as an external object directly, without using others Fib elements except if- and root-elements.	(inVar2::4::identifier , "3")
inVarX::-dimensionD::-points	The input variable $InVar_X$ is used to indicate the number of points in dimension D. The value of D is the number of the dimension of the root-element (the counting starts at 1), for which the resolution applies. The specified value indicates the number of points that are displayed by default. For example, if the Fib object represents an image and in the first dimension direction by default 1000 points are displayed in a line, the text is as shown in the example.	(inVar3::dimension1::points , "1000")
inVarX::-dimensionD::-resolution	The input variable $InVar_X$ is used to specify the resolution of dimension D in points per unit of the SI unit of the dimension. The value of D is the number of the dimension of the root-element (the counting starts at 1), for which the resolution is. The specified value is the resolution that is set by default. For example, if the Fib object represents an image and the first dimension is by default displayed with 10000 points per meter (= 10 dots per mm), the resolution is 10000 as shown in the example.	(inVar3::dimension1::-resolution , "10000")
inVarX::N::-language	The input variable $InVar_X$ is used to set the language. It should be set to the value N for the specified language.	(inVar1::1::language , "english")
inVarX::N::-subtitle	The input variable $InVar_X$ is used to set the subtitle. It should be set to the value N for the given subtitle.	(inVar2::1::subtitle , "german with comments")

Key	Description	Example
inVarX::N::-feature	The input variable $InVar_X$ provides a feature /enhancements of the original multimedia object. It has the value N for the given feature. Better and newer features should be assigned higher values of N.	(inVar2::1::feature , "The table has now its true color.")
inVarX::N::-bugfix	The input variable $InVar_X$ provides a bug fix of the original multimedia object. It has the value of N for the specified bug fix. Better or more recent bug fixes should be assigned higher values of N.	(inVar2::3::bugfix , "The car in the background is not blurred.")
inVarX::Type	The input variable is of a certain type, regardless of its value. As type, the above listed can be used (e. g. description, type, identifier). This allows that keys of the form "inVarX::N" can be used, without giving the type of the input variable each time.	(inVar1::bugfix, "The color of the penguin is corrected.")
inVarX::N	This are the information for the Fib object, if the input variable $InVar_X$ is set to the value N. The type of input variable should be set with "inVarX::Type".	(inVar1::17, "The penguin has a black back and not a dark blue.")
inVarX::random	The input variable $InVar_X$ should be set a random value from its domain.	(inVarX::random, "")
subObjN::-description	This is a description of the N-th subobject, that is a Fib object, that should be provided from outside for this root-element. The counting of the subobjects starts at 1, wher each list of output variables $OutVar_{v_k}$ represents a subobject.	(subObject1::description , "This partial image is shown in the upper right corner on a mountain.")
subObjN::-outVarX::-description	This is a description of the X-th output variable for the N-th subobject.	(subObj2::outVar3::-description , "The horizontal coordinate, at which the integrated image will be displayed.")
preview::XXX	Among the contained root-objects, there may be some who are not used to encode the multimedia object, but are **previews** for the multimedia object. Ther can be any number of preview subobjects. To recognize this quickly, the optional part can contain a "preview" entry. The value of the entry is the $Identifier$ of the root-object, which encodes the preview. The second part of the key (here XXX) is arbitrary and should indicate the type of preview. The "preview" entries in the optional part, should be at the beginning of the optional part, in order to get them faster.	(preview::minipicture , "1")

Key	Description	Example
preview::-minipicture	The root-object to the specified identifier is a preview image (no size specified).	(preview::minipicture , “17”)
preview::-minipictureX	The root-object to the specified identifier is a thumbnail with X times X pixel size.	(preview::minipicture32 , “5”)
isPointElement	The object returns the data of the points with point elements. (Counterpart is the key “isPointSubObject”)	(isPointElement , “”)
isPointSubObject	The object returns the data of the points with a subobject (subobject elements). The value is the number of the subobject for the points data. If ther is no value given, the first subobject is for returning the points data. (Counterpart is the key “isPointElement”)	(isPointSubObject , “”)
isAntialiased	The represented multimedia object is antialiased. (Counterpart is the key “isNotAntialiased”)	(isAntialiased , “”)
isNotAntialiased	The represented multimedia object is not antialiased. (Counterpart is the key “isAntialiased”)	(isNotAntialiased , “”)
DbObject::XXX	The value to the key is the identifier (a number) of the object, which is identical to this object except for the XXX property. Possible values for XXX are for example “isPointSubObject” or “isAntialiased”. If the actual object has no antialiasing, the object with the identifier of the value for the “DbObject::isAntialiased” key has, but is otherwise identical.	(DbObject::isAntialiased , “-56”)

Table 8: Keys for the optional part

9.14.4 Order of the root-Elements

All root-elements are arranged in a certain order. First in this order comes the highest root-element and then the root-elements contained in it, with their containing root-elements in the order they are defined in the respective root-elements. In diagram 1 an example of the order of the root elements is shown.

For each Fib object all root-objects are visible, that stand after the Fib objects (in the root-elements) in which it is contained, and the root-objects in the database.

If an external object in a Fib object should be resolved, first the identifiers in the next root-element, in which the Fib object is contained, are searched, then the identifiers in the next higher root-element, and so on. For this only identifiers are checked, which are after the identifier for the root-object, in which the Fib object is

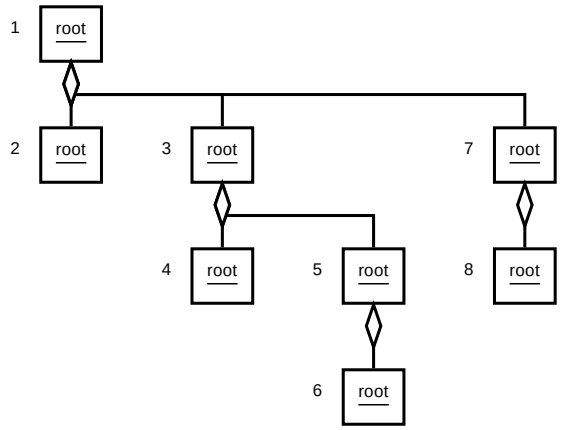

Figure 1: Example: Order of the root-Elements

contained. The main-Fib-object is befor all contained root-objects (with identifiers) of a root-element. In the end, the database objects are checked.

Example: The following root-element structure is given:

$$
\begin{aligned}
root_0 &= root(\ldots, Obj_0, ((1, root_1), (2, root_2), (3, root_3)), \ldots) \\
\\
root_1 &= root(\ldots, Obj_1, ((11, root_{11})), \ldots) \\
root_{11} &= root(\ldots, Obj_{11}, ((111, root_{111})), \ldots) \\
root_{111} &= root(\ldots, Obj_{111}, (), \ldots) \\
\\
root_2 &= root(\ldots, Obj_2, ((21, root_{21})), \ldots) \\
root_{21} &= root(\ldots, Obj_{21}, (), \ldots) \\
\\
root_3 &= root(\ldots, Obj_3, ((31, root_{31})), \ldots) \\
root_{31} &= root(\ldots, Obj_{31}, (), \ldots)
\end{aligned}
$$

The root-element $root_0$ is the top most root-element. The following list lists for some Fib objects, which identifiers and thus root-objects they can use in external objects or which root-elements can be resolved and are visible for them. (The Fib database is not considered here. Root-objects in the database would be visible for all Fib objects.) The identifier and the root-elements are specified in the order they are traversed in the search.

- Obj_0: $(1, root_1)$, $(2, root_2)$, $(3, root_3)$

- Obj_1: $(11, root_{11})$, $(2, root_2)$, $(3, root_3)$
- Obj_{11}: $(111, root_{111})$, $(2, root_2)$, $(3, root_3)$
- Obj_2: $(21, root_{21})$, $(3, root_3)$
- Obj_{21}: $(3, root_3)$
- Obj_3: $(31, root_{31})$
- Obj_{31}: none

10 The Fib database

The Fib database does not belong to any Fib multimedia object. It is supplied with the Fib libraries / the Fib system and contains frequently used database objects, which can be used in Fib objects. Fib objects can use the database objects, without including these database objects (Fib objects). Database objects can be for example lines (with the input parameters for start and end points), rectangles or circles, but also trees, cars, character sets (fonts) or fractals. If for example a circle is needed for a Fib object, a corresponding parameterized database object can be used.

The implementation of the database objects can be adapted to the application environment. If for example for the display OpenGL is used, the database objects can be implemented directly with OpenGL primitives (e. g. triangles). In this manner with database objects the application performance can be improved. During the coding of the Fib objects this can be directly taken into account, by using database objects that have a good performance for the target application /program. These Fib objects are still viewable on all systems /with all programs with a high enough Fib database version, but on some faster.

Which objects, and identifier for these objects, a database contains, is determine by the database version. Databases with a higher database versions contain at last all database objects with the same identifiers, as databases with a lower database versions. In this way, it is ensured, that Fib objects are always forward compatible with newer database versions.

All identifiers for database objects are negative.

11 Definitions for Fib

In this section some definitions for the Fib multimedia language are given. These should improve the handling and understanding of Fib.

11.1 Definition of: correct Fib object

An **Fib object** is correct, if it meets the above Fib syntax, all the variables, which it contains, are defined above it and each contained Fib element is a correct Fib element. An correct root-element must belonge to the correct Fib object. For a Fib element to be correct, it must fit its root-element (that is, among other things, correct [number of] dimensions and domains).

Correct Fib objects are also referred to shortly as Fib objects.

A **Fib element** suits the above presented Fib syntax, except that no Fib elements are contained in it.

In the following examples, comments are introduced with "//". These are not part of the displayed Fib elements.

Examples:
Correct Fib objects (It is assumed that the associated root-element is correct and fits.):

$$
\begin{aligned}
Obj &= pr((3)_{colorGrayscale}, p((1;5))) \\
Obj &= list(for(y,[(4;2)],pr((3,42,125)_{colorRGB},p((7;y))), \\
&\quad fun(x,add(mult(4,exp(3,-2)),2),pr((205,x,x)_{colorRGB},p((3;x))))
\end{aligned}
$$

Fib elements (Elm):

$$
\begin{aligned}
Elm &= p((3;2;5)) \text{//3-dimensional} \\
Elm &= for(x,[(3;8)],null) \\
&\quad \text{//null is not an object (in the implementation it is a null pointer)} \\
Elm &= p((2;5))
\end{aligned}
$$

Neither a Fib element nor a correct Fib object (Woe, $null$ is no object [in the implement it is a null pointer]) :

$$
\begin{aligned}
Woe &= list(for(x,[(3;7)],null),null) \\
Woe &= fun(x,add(4,exp(6,3)),for(y,[(6;7)],null)) \\
Woe &= pr((3)_{colorRGB},p((1;5))) \text{ : //invalid syntax} \\
&\quad \text{Property element: The colorRGB vector requires} \\
&\quad \text{3 parameters /elements.}
\end{aligned}
$$

11.2 Definition of: complete Fib object

A complete Fib object is a Fib object, which represents a displayable multimedia object. Each complete Fib object is also a correct Fib object. A complete Fib object also includes, all necessary root-elements with the entries for the correct Fib version and Fib database version.

11.3 Definition of "below" and "above" in a Fib object

You can imagine a Fib object as a tree, wher the branch elements (e. g. root and the list elements) representing the branches in the tree. Because in computer science in general the root is displayed at the top, the Fib elements, which are contained in an element Elm, are below it, and the Fib elements, which are containing the element Elm, are above it.

Below a Fib element Elm means, that the elements are meant, that the element Elm directly or indirectly contains. In contrast, above a Fib element Elm are the Fib elements, that are containing the Fib element Elm directly or indirectly.

This is illustrated in figure 2. The list element in the middle, which is marked with 1 is the element with respect to which above and below is determined.

11.4 Order of the Fib elements

On the definitions of "below" and "above" in a Fib object the order of Fib elements is established. To each Fib element in the complete Fib object an unique natural number is assigned. If in a Fib object N elements exist, the numbers 1 to N will be assigned to the Fib elements in the Fib object. Fib elements below a Fib element are assigned to higher values.

If a list element subobject Obj_U has a higher number U as another subobject Obj_K ($U > K$) of the list element, it also contains Fib elements with higher numbers as the Fib elements in Obj_K. The root-elements are treated the same way in the order as list elements, with the main-Fib-object as the first subobject after which the sub-root-objects follow in ther succession (in the root-element).

The same applies for any other branch element. The above defined syntax determines the order of the subobjects.

In figure 3 a sample object with the corresponding numbers (next to the elements) for the order of the Fib element is shown.

11.5 Order of particular Fib elements

Also Fib elements of a given typs are assigned to natural numbers of an order for there type. These orders are based on the order of the Fib elements. If in a correct Fib object N Fib elements of a type exist, the numbers 1 to N are assigned to the Fib elements of this type. If to a Fib element Elm_1 in the order of the Fib elements a higher value is assigned (as to another Fib element Elm_2 of the same type), so it

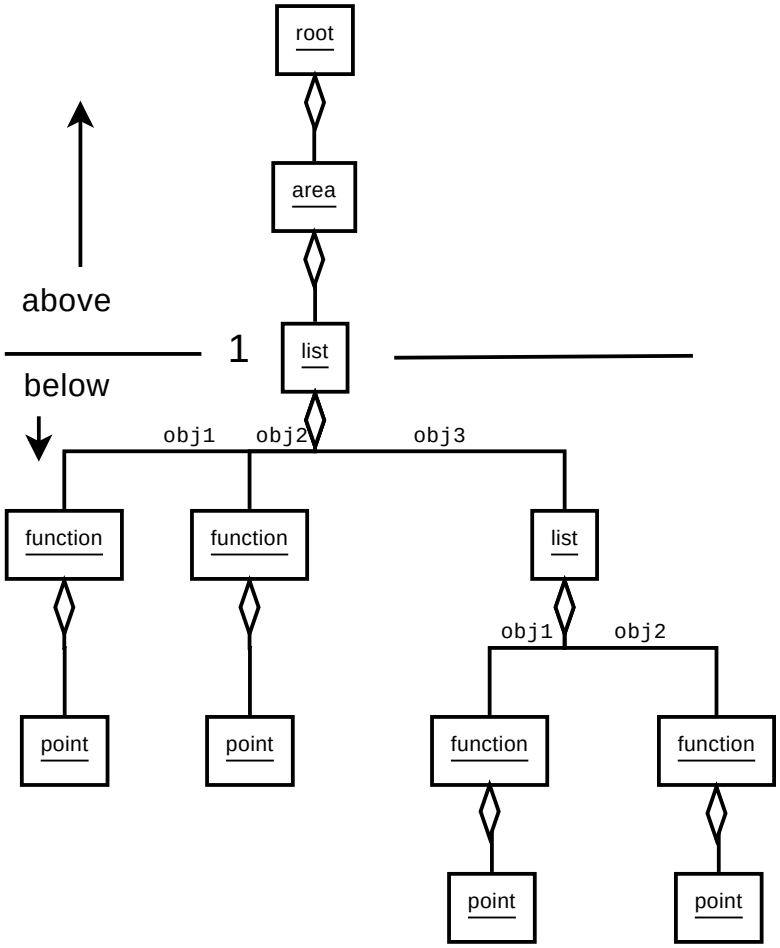

Figure 2: Example for “below” and “above” in a Fib object

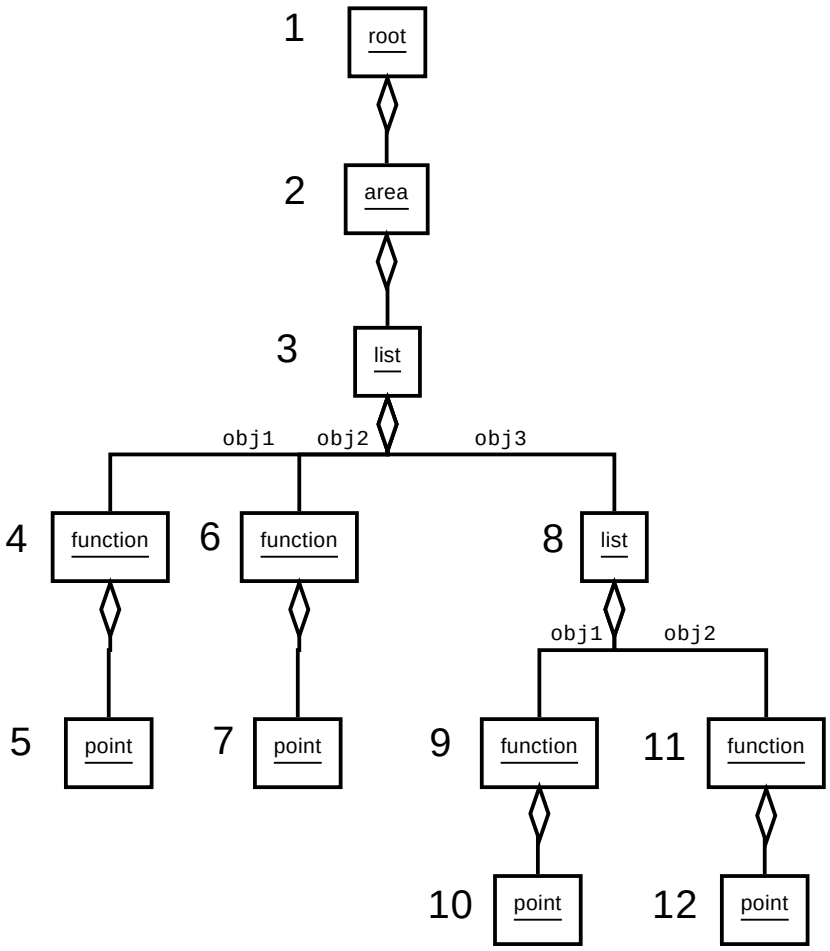

Figure 3: Example: Order of the Fib elements

(Elm_1) is assigned to a higher value in the order of the Fib elements of the same type (than the other Fib element Elm_2).

In figure 4 an example object with the corresponding numbers (next to the elements) for the order of particular Fib element is shown.

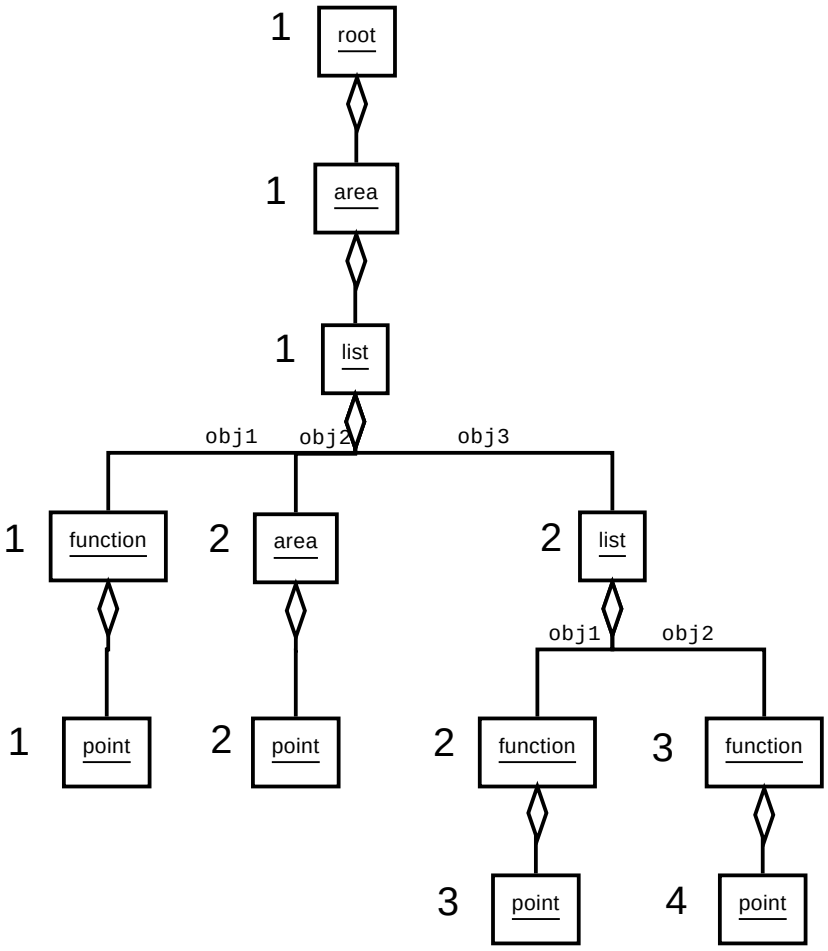

Figure 4: Example: Order of particular Fib elements

11.6 Order of move points

Another order applies to all Fib elements that can be moved. These are called move points. The orders of the move points is based on the order of the Fib elements. If in a complete Fib object N move points (movebel Fib elements) exist, the numbers 1 to N are assigned to these movebel Fib elements. If to a movebel Fib element Elm_1 in the order of the Fib elements a higher value is assigned (as to another movebel Fib element Elm_2), so it (Elm_1) is assigned to a higher value in the

order of the movebel Fib elements (than the other Fib element Elm_2).

Fib elements, which can be moved and represent move points, are all limb elements (they are containing exactly one subobject):

- property element (see section 9.3 on page 7)
- comment element (see section 9.5 on page 14)
- area element (see section 9.6 on page 16)
- functions (see section 9.7 on page 17)
- Fib elements, to retrive domain properties (see sectiont 9.11 on page 27)
- set-element (see section 9.12 on page 30)
- matrix element (see section 9.13 on page 31)

Fib elements, which can't be moved and don't represent move points, are:

- all leaf elements:
  - points (see section 9.2 on page 6)
  - Fib elements to call external subobjects (see section 9.10 on page 26)
- all branch elements:
  - root-element (see section 9.14 on page 33)
  - list element (see section 9.4 on page 14)
  - the Fib element, to call external objects (see section 9.9 on page 25)
  - conditions with the if-element (see section 9.8 on page 24)

In figure 5 an example object with the corresponding numbers (next to the elements) for the order of move points is shown.

11.7 Definition: part object

Any object that is a whole branch (e. g. a sub-list object, main-Fib-object or sub-root-object) of a branch element is part of a part object. Furthermore, to the part object belong all root-elements, in which it is contained or which it uses for external objects. Even Fib elements, which define variables that are used in the part object, belong to it. The union of two part objects is again a part object. The complete Fib object itself is also a part object. A part object can always be evaluated to a multimedia object.

A **genuine part object** is a part object, that is not the (complete) Fib object itself.

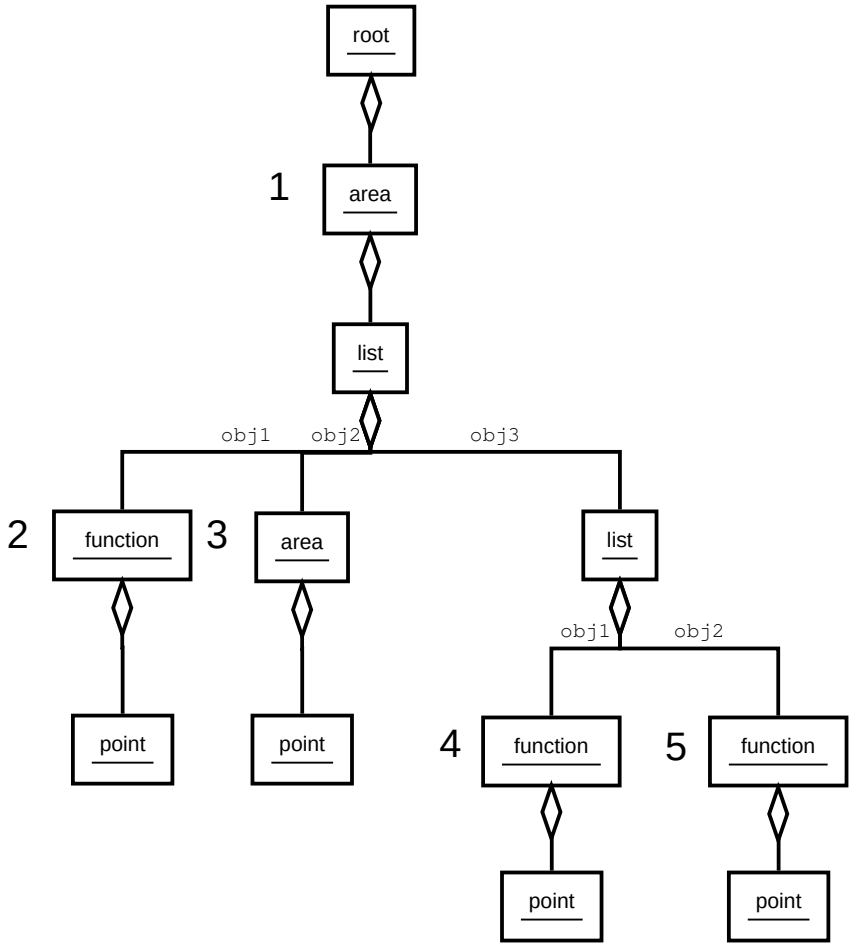

Figure 5: Example: Order of move points

A **simple part object** is a genuine part object, that contains only one leaf with one point object.

A **coherent part object** is a genuine part object that contains the whole object of one branch (subobject), of just one branch element (e. g. list element or root-element) and the required elements above it. In particular, every simple object is a coherent subobject.

To create a coherent part object, one branch element (for example list element or root-element) from the complete Fib object can be deleted and replaced by one of the subobjects contained in it. The resulting subobject must of course be correct, to be a coherent part object. (For example, if the main-Fib-object of a root-element is deleted, the result isn't a coherent part object. Also if a sub-root-element is deleted, and the next above main-Fib-object is replaced by its main-Fib-object, the domains of the replace root-element should be adapted by the next root-element.)

With figure 6 this definition is illustrated with an example of a Fib object. In the following, some examples of different types of part objects in Figure 6 are being given, for the Fib elements their numbers (from the Fib element order respectively figure 6) are given. Furthermore, it is establish that the point element with the number 10 does not use the variable that is defined by the area element with the number 2. All other variables are needed in the points, which are below the respective variable definitions.

Part objects:

- 1; 2; 4; 5
- 1; 2; 6; 7
- 1; 2; 3 (just subobjects 1 and 2); 4; 5; 6; 7
- 1; 9; 10
- 1; 2; 8; 9; 10; 11; 12
- 1; 2; 3 (just subobjects 1 and 3); 4; 5; 8; 9; 10; 11; 12
- 1; 2; 3 (just subobjects 1 and 3); 4; 5; 11; 12
- (all Fib elements) 1; 2; 3; 4; 5; 6; 7; 8; 9; 10; 11; 12

Genuine part objects:

- 1; 2; 4; 5
- 1; 2; 6; 7
- 1; 2; 3 (just subobjects 1 and 2); 4; 5; 6; 7
- 1; 9; 10

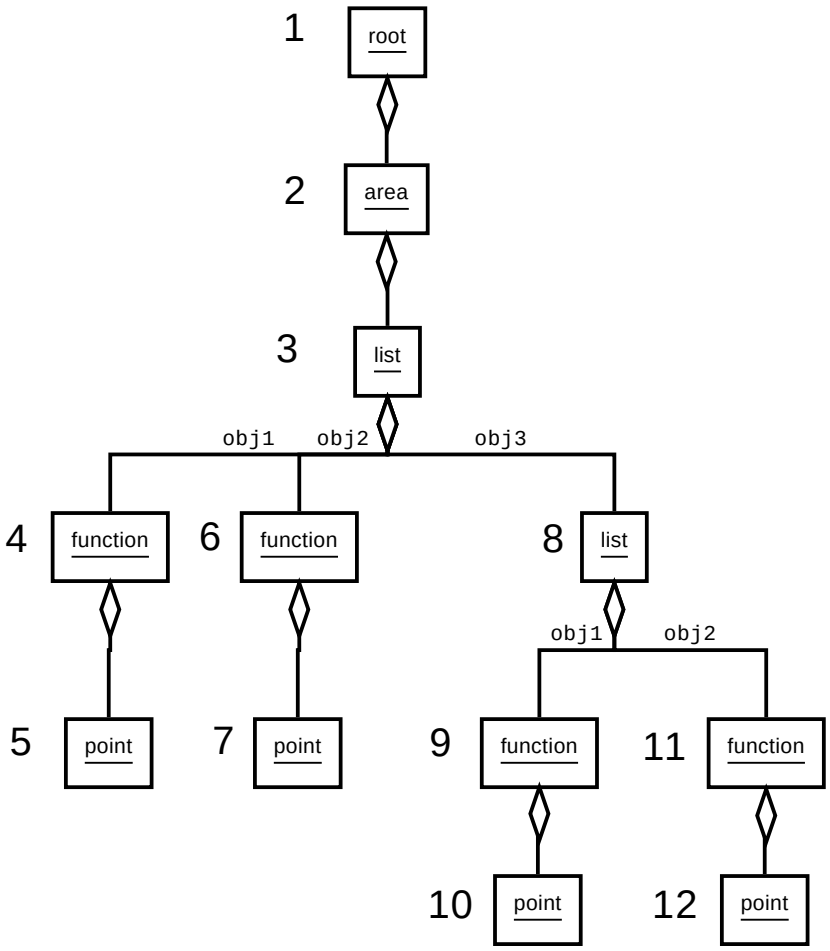

Figure 6: Example object for part objects

- 1; 2; 8; 9; 10; 11; 12
- 1; 2; 3 (just subobjects 1 and 3); 4; 5; 8; 9; 10; 11; 12
- 1; 2; 3 (just subobjects 1 and 3); 4; 5; 11; 12

Simple part objects:

- 1; 2; 4; 5
- 1; 2; 6; 7
- 1; 9; 10
- 1; 2; 11; 12

Coherent part objects:

- 1; 2; 4; 5
- 1; 2; 6; 7
- 1; 9; 10
- 1; 2; 11; 12
- 1; 2; 8; 9; 10; 11; 12

11.8 Order of the coherent part objects

There is an order on the coherent part objects as well. This will hereinafter also called order of part objects, as there are no own orders for the other types of part objects.

The orders of the (coherent) part objects is based on the order of the Fib elements. If in a complete Fib object N (coherent) part objects exist, the numbers 1 to N are assigned to these part objects. The greater the number of a coherent part object, the greater is the smallest number in order of the Fib elements, of the Fib elements in it. Or: If to the top most Fib element of the subobject, which defines the coherent part object, a higher value assigned, than the defined part object is also associated with a higher value.

In figure 7 an example object with the corresponding numbers (next to the defining subobjects of the part objects) for the order of (coherent) part objects is shown.

11.9 Definition of Fib multimedia object

If the expression Fib multimedia object is used, the Fib object with focus on the multimedia object it represents is meant.

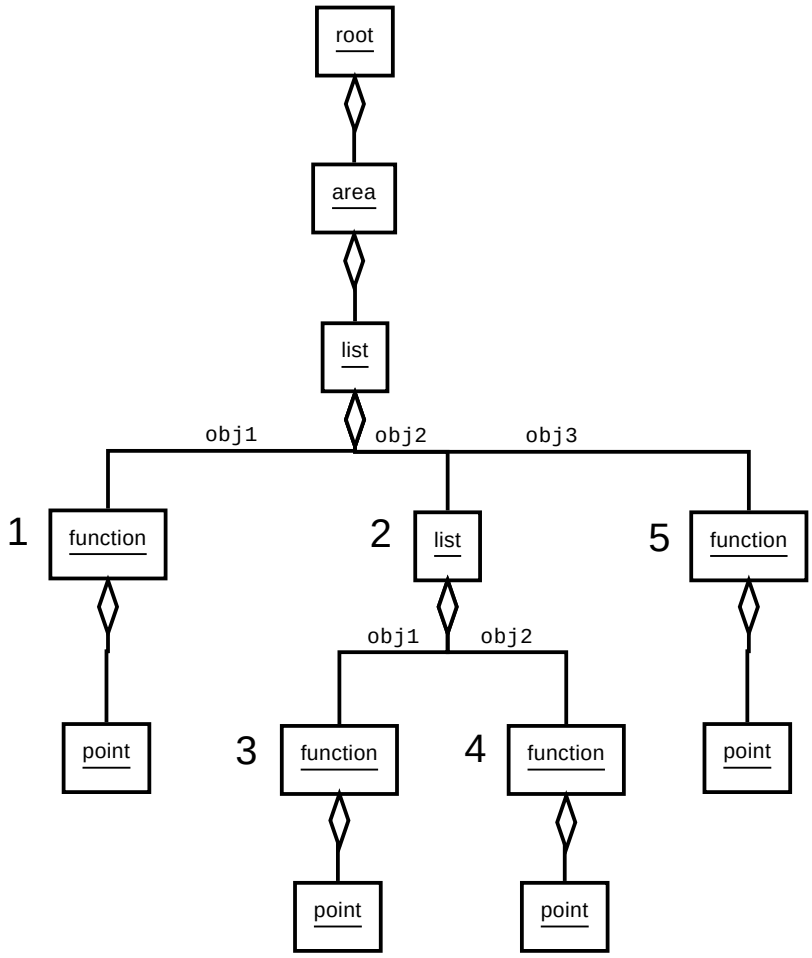

Figure 7: Example: Order of (coherent) part objects

11.10 Definition of correct Fib multimedia object

A correct Fib multimedia object is a Fib object, that fully reflects the original multimedia object, which it should represent. So, if the multimedia object that the Fib object represents, and the original multimedia object are compared, no difference between them can be found.

12 Theoretical statements for the Fib multimedia description language

Some theoretical statements for the Fib multimedia description language will follow, for wich, because of lack of time, usually not a complete proof is given. With these statements a better understanding of the Fib multimedia language should be provided.

12.1 Power of the Fib multimedia language on images

With the Fib multimedia language all as raster graphics displayable images can be represented. Images, that are displayed as raster graphics, are the most commonly used images in the digital data processing. These include Windows bitmap (BMP; file extension: .bmp), JPEG File Interchange Format (JFIF, file extension: .jpg) and Portable Network Graphics (PNG, File Extension: .png).

With using only the point element and the list element alone all possible raster graphics can (already) be represented.

Proof: A raster graphic (euclidean, two dimensional, discrete) can be represented as a matrix, the column indicats the x coordinate, the lines the y coordinate of the point and the values specify the colors of the points. The number of dots in the raster image is finite. To represent these points in the Fib multimedia language, a root-element have to be created that contains the properties of the raster graphic (size and dimension domain etc.). In this root-element a main Fib object, wich is a list element, is inserted, which contains for each point of the image a subobject. This subobject consists only of a property element, which encodes the color of the point, and one contained point element, which encodes the position of the point. So there is for every point in the raster graphic a corresponding point in the generated Fib object, so that the Fib object represents the raster image.

A mapping into the Fib multimedia language can for example be implemented by the algorithm shown in listing 2 (pseudo-C).

The indexing of the matrix begins with (0.0).
The color value of coordinates (x, y) can be determined with `matrix[ x, y ]`. With the function `getColorVector()` a Fib color vector is generated from the matrix color value.
The syntax of the Fib objects corresponds to the in section II on page 4 discussed possible syntax.

Listing 2: Algorithm to generate a correct Fib object from an image matrix

```
1  void translate( matrix ){
2
3     int xmax = number of columns of the matrix;
4     int ymax = number of lines of the matrix;
5     Fib_Object_Pointer obj = new root();
6
7     //set the properties of the raster graphic
8     obj->setPicture( xmax, ymax, colorScheme );
9
10    list mainList = new list()
11
12    obj->insertMainObject( mainList )
13
14    for( int x = 0; x == xmax; x = x + 1 ){
15
16       for( int y = 0; y == xmax; y = y + 1 ){
17          mainList->insertObject( property( getColorVector( matrix[
                x,y] ) , p( (x,y) ) ) );
18       };
19    };
20 };
```

Since the expansion with functions and other elements is optional, a correct Fib object is generated with the specified algorithm.

For each point in the matrix there is a same-colored point in the Fib object with the corresponding coordinate, but there are no points in Fib object that are not present in the matrix, because each coordinate of the matrix is traverse with the two “for” loops. Thereby a point in the Fib object is added for each coordinate and thus for every point in the raster graphic. Thus for every point in the raster graphic a corresponding point in the Fib object exists. Since only coordinates of the matrix are traverse in the “for” loops and only for them corresponding points are included in the Fib object, there are only points that appear in the matrix and hence only points in the Fib object, which also occur in the raster graphic. So there are only the points from the original raster graphic in the Fib object and no more. Thus the Fib object and the original the raster graphic represent (/shows) the same raster graphic. Therefore, all raster graphics are representable with the Fib multimedia language.

A Fib object generated by the algorithm, that represents an original raster graphic, is an upper limit to the minimum size of possible corresponding Fib objects. This means, each raster graphic can be represented by a Fib object, that is as big as the Fib object for the raster graphics that was generated with the above algorithm, that is the Fib object generated. But there are likely even shorter Fib objects for raster graphics.

With that the minimum size of a Fib object for a raster graphic is maximal:

Fib_max_{min} = (number of pixels in the image) $*$[(size of a point element) + (size of a property element for the appropriate color scheme)] + (size of a list element)+ (size of a root-element with values set)

Example: You want to shown an image of 8 x 8 = 256 pixels with 3 bytes for the color (RGB), 1 byte for the position (for each direction x and y 4 bits = 8 possible values) and 1 byte for the object name (for example “l” for the list elements and “p” for the point elements). Parentheses are not needed, because all parts have a fixed length. (The assumptions about the sizes of the Fib object elements are estimated here. In a implementation, better (/smaller) values are more likely.)

- A point element requires 2 bytes (1 byte position + 1 byte element name).
- For a property element 4 bytes are needed (3 bytes color + 1 byte element name).
- For a list element 9 bytes are needed (8 bytes for specifying the number of subobjects + 1 byte element name).
- The root-element requires 256 bytes. (Since many parts of the root-element are empty and only the domain for two dimensions and the RGB color domain must be specified, 256 bytes should be enough.)

Calculation: $256(pixel) * (2bytes + 4bytes) + 9bytes + 256bytes = 1801bytes$

So the image can definitely be displayed with a 1801 byte long Fib object. But there are other shorter Fib object representations possible. The 1801 bytes are thus the upper limit for the minimum size with which the image can be represented by a Fib object.

For comparison: The memory requirements of a raster image as a bitmap (only point information) is at least (Number of pixels in the image) * (size of a color value)

For the above example it results in: $256(pixel) * 3bytes = 768bytes$

That’s about half of the 1801 byte limit on the minimum Fib object representation.

In that the matrix is extended to three dimensions, the statement about the representability of all raster images can be easily extended to sequences of raster images (such as the images of movies).

12.2 Cardinality of Fib

Theorem: The set of possible Fib objects is countable infinite.

Sketch of proof for “countable”:
Every Fib object can be represented with a finite number of letters, and thus bits or numbers, and the amount of these is countable. This follows from the fact, that the number of Fib elements in a Fib object is always countable and every Fib element consists of countably many parts, which are themselves countable, there are for example only integer or rational numbers, and also the number of variables is countable.

Sketch of proof for “infinite”:
All natural numbers can be represented by Fib objects. In the following a possible representation of any natural number with a Fib object is described.

A point object itself is the natural number 0 . If a Fib object is inserted in a new function element the resulting Fib object is the successor of the original Fib object. In this manner the 0 and the successor function can be reproduced in the Fib multimedia language, and all natural numbers can be represented. Since the set of natural numbers is infinite, the amount of Fib objects is also infinite.

Even every multimedia object can be represented by a countable infinite set of Fib objects, because to a Fib object, that represents a multimedia object, any Fib object can be attached with the help of a list element, as long as the multimedia object representation is not changed. For example, to a Fib object a copy of itself using a list element can be as often attached as needed, without changing the multimedia object.

12.3 Any complete Fib object can be represented as a multimedia object

It is shown in this section, that it is always possible to interpret (to translate into a multimedia object) any complete Fib object (see section 11.2 on page 55) in a way, that only valid multimedia objects of a certain type (e. g. RGB images with 100 x 100 pixels) can result. In which the limitations with regard to the multimedia data, as already mentioned, are assumed, that the multimedia data can be represented as properties of points of a finite, euclidean and discrete (there are smallest units) space. This restriction is not very large, because they (almost) excludes non of the presently common multimedia data. The multimedia data may therefore represent images, sound or movies.

With the restriction that the distance / difference between two properties of the same type can always be determined as a numerical value, two multimedia objects with the same dimensions can be always compared. Two multimedia objects have the same dimensions, if for each point in the first multimedia object exactly one corresponding point in the other multimedia object exists.

The requirement of the complete Fib object is necessary to ensure that the Fib object can be always evaluated. Since the completeness of an Fib object can be check with the syntax shown in part II, if a Fib object is complete can always be determined. Incomplete Fib objects should not be generated by the algorithms or the genetic operators.

If the dimensions of a Fib object are adapted to that of a multimedia object (this should be always possible), the Fib object is always comparable with the multimedia object, because it can itself be always presented as a multimedia object, and two multimedia objects with the same dimensions can always be compared to each other and the similarity to each other can be evaluated.

Note: This is advantageous for genetic algorithms. In some other forms of representation that are generated by genetic algorithms, invalid objects (e. g. programs) arise, where a more accurate evaluation / comparison is not possible. A population in this format can contain, for example, a large class of objects that are invalid, which are all equally bad and are therefore considered same in the selection process. If the population consists only of invalid individuals, the selection of a better individual is impossible. The genetic algorithm is then on a (fitness) plane, from which it can only find away with great difficulty.

Proof that every correct Fib object can be represented as a multimedia object (of a specific type, such as a picture): The starting point is that a multimedia object (euclidean, two-dimensional, discrete) can be represented as a finite set of points with their (finitely many) properties. Since there are only finitely many points with only finitely many properties, such a finite set is always constructible.

Such a finite set of points with their (finitely many) properties is also generated by a correct Fib object. Points of the set that are too much to represent a multimedia object, will be deleted from the set. Points that are missing in the set to represent a multimedia object are inserted into the set. Properties of the points that are too much to represent a multimedia object will be deleted. Properties, which are missing at points to represent a multimedia object, are added and set with their default values (e. g. the zero values of their domains). In this way a finite set of points is created with their (finite many) properties that can be represented as a multimedia object.

Example: The Fib object should represent an RGB image with 100 x 100 pixels. For this the dimensions of the Fib object are adjusted to cover these 100 x 100 pixels in horizontal and vertical direction. That is, if the dimension (horizontal or vertical) already exists, the domain of each direction is adjusted, so that it covers at least 100 values at regular intervals. So that to each pixel a value is assigned. If a dimension (/ direction) is missing, it will be created with the appropriate domain and in all points for the dimension the standard value of 0 will be set. In this case there is no point with other values than the default value for this dimension. Dimensions that are too much, are deleted from the root elements and the points. The evaluation of the resulting Fib object produces a set of points with their properties.

During evaluation, the smallest value of W_{min} of each dimension in the Fib object is assigned to the value 0 as the coordinate in the set, the second smallest one to 1 as the coordinate, etc.

From this set now all points are deleted that are not within the 100 x 100 pixels boundary (points for which a coordinate respectively value is less than 0 or greater than 99). For all coordinates, which are still missing (missing points, where the values respectively the coordinates is between [including] 0 and 99), points are inserted. All properties that are not RGB colors will be deleted. To all points that have no property for RGB colors, the default color $(0, 0, 0)_{colorRGB}$ is assigned. The resulting set contains for each point in the RGB image with 100 x 100 pixels a point with RGB color, but no other points or properties, and thus represents an RGB image with 100 x 100 pixels.

This can then be compared with other RGB images with 100 x 100 pixels and rated in relation to them.

Part III
The genetic algorithm

In this section, general design decisions and initial analysis for the genetic algorithm are established. The realized genetic algorithm is flexible and expandable designed.

The genetic algorithm is also an evolutionary algorithm. The term “genetic” refers to the ability of the algorithm to encode information of two or more individuals in a new individual and that it is working on information that code the (multimedia) object and do not represent those objects directly.

The algorithm is used to generate Fib objects, which represent a multimedia object as well as possible. The algorithm is given a particular multimedia object, for that it generates Fib objects /individuals, of which good will be selected. The creation of new individuals may also include analysis of the multimedia object and the use or analysis of information from other individuals. Which individuals are good, can be decided by the given parameters (by the evaluator for individuals).

The algorithm consists of five separate parts:

- the core algorithm
- the evaluator for individuals
- the mortality rating algorithm
- the evaluator for operators
- the set of operators

In Figure 8 a sketch for the flow diagram for the genetic algorithm is shown.

In the following the single Fib objects are called individuals. The set of all individuals, who are existing at a time in the algorithm, is calle population.

13 Core algorithm

The core algorithm uses the evaluator and the operators to realize the genetic algorithm. It is simple and flexible.

The evaluators for individuals or operators are (with the help of parameters) interchangeably.

The core algorithm includes the main loop of the genetic algorithm, which calls the operators and generates the individuals.

The second loop in the algorithm is the selection loop. It will delete individuals.

In addition, the core algorithm provides functions for the operators. The operators are called by the core algorithm through a method, that is the same for all

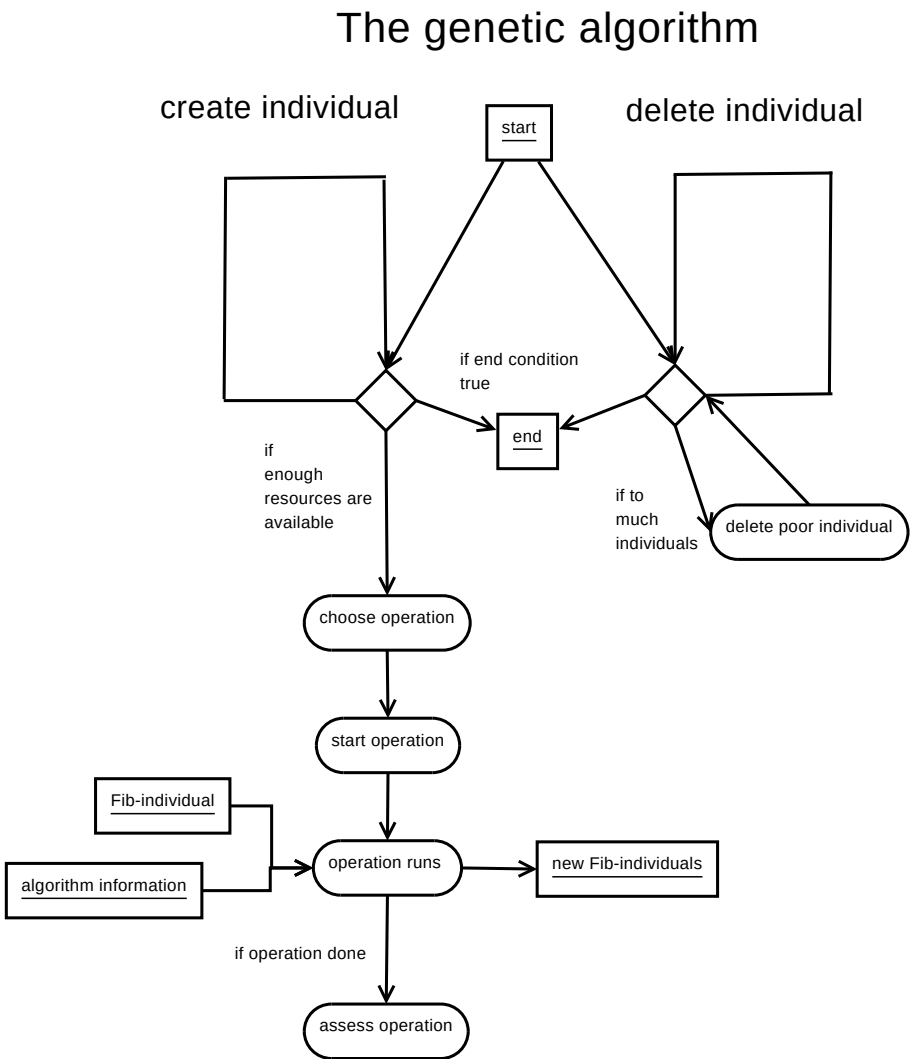

Figure 8: Flow diagram of the genetic algorithm

operators. After that, the operators can get, with the help of the functions of the core algorithm, the algorithm specific values, e. g. such as individuals, which the operations will use. In this way, the operators will always look the same in each case for the core algorithm, and the core algorithm does not need any logic specific to a particular operator. Thereby future adjustments to the operators are simplified, because not the interface of all operators have to be be adjusted, but only the functions, which are provided by the core algorithm.

The calling method for the operators should be flexible, so that operators can be started concurrently and the algorithm don't have to wait for the completion of each operation. The operators should be run separately from the core algorithm and affect it only by the specified interface. A faulty operator should not lead to errors in or termination of the entire system.

14 Evaluator for individuals

The concrete evaluator algorithm should be simple to replace, so it can be adapted to various problems. The respective evaluation algorithm is needed to calculate the fitness of an individual.

With the help of parameters for special evaluators, they can be further adjusted (for example it can be adjusted, how important a small size compared to a fast evaluation of individuals is).

14.1 The fitness of individuals

The fitness of an individuals is determined by different fitness factors.

One of the most important is, to what extent the individual is similar to the desired original multimedia data (such as the original image). The more the individual (or the phenotype of it) is similar to the original multimedia data, the higher the fitness should be and the lower is the error, which the individual has for the display of the original multimedia data.

This error (and thus the fitness of the individual) can be determine for example, with the sum of (the squared) differences (not defined points give the maximum error) of the colors of the points between the original image and the image produced by the individual, or through another self-determined distance.

If the fitness of separate part objects of the individual should be determined, this can be achieved, for example, by including in the evaluation only the area that is covered by the part object, the covered area and a border around this area or only the smallest square, which encompass this part object.

Another useful fitness factor is the size (increases with the number of elements) of the separate individuals, to give bigger individuals a lower fitness than smaller individuals, with the same error on the original multimedia data, and to prefer the smaller individuals.

A further fitness factor, which can be included, is the estimate of the time it takes to evaluate the individual for displaying the multimedia object (phenotype).

In this way maybe even the execution speed of the algorithm can be increased.

14.2 Selection by deletion of individuals

In order to spare the resources (memory, CPU time), it is necessary to limit the number of individuals (Fib objects) in the working process (individuals wich participate in the genetic algorithm). Therefore (if required) individuals have to be removed from it. In this process individuals with lower fitness are preferred.

There is therefore a **mortality rating algorithm** for the algorithm. It determines the probability, with which a individual is deleted. To delete individuals, there is a separate loop in the algorithm, in which is tested, whether the maximum number of individuals is exceeded. If this is the case, individuals will be deleted, until the number of individuals is back on track (lower the maximum number of individuals). In this case individuals with a high mortality assessment by the mortality rating algorithm have a high probability of being deleted.

The mortality assessment is based on the assessment of the fitness of individuals. However, the mortality rating algorithm can be exchanged for other mortality algorithm or be controlled by parameters. For example, it is possible to declare some individuals to be immortal, so they can not be deleted. By declaring, for example, the n ($n > 0$) best individuals to be immortal, it can be avoided, that they will be deleted and thus that one of the best individuals will be lost.

The mortality algorithm can furthermore access via the operator interface the status information of the genetic algorithm, to determine, for example, the number of previously generated individuals.

15 Evaluator for operators

To determine which operator to selected next, they are evaluated. Operators, which are rated better in a situation, have a higher probability of being selected respectively run in a similar situation.

The situation may include:

- how many operations wher executed.
- of which nature the original multimedia object is (with the help of the domains for the environment in the root-elements of it):
  - it contains colors
  - it is monochrome
  - it contains sound
  - it is a film
  - ...

- the average (relative) fitness of individuals.
- the fitness of the best/worst individual.
- the standard deviation of fitness values in the population.
- the number of individuals.
- the operations that have been executed.
- ...

The different evaluation algorithms can be selected. Thus, different evaluation algorithms can be easily interchanged and compared.

To evaluate the operators, data of their previous execution may be kept permanently (by the seperate evaluators). In this way, the algorithm can learn from previous operator calls.

The evaluation of the operators should be system independent, ie. independent of the computer, on which the algorithm is running.

The evaluation criteria can be:

- execution time of the operation
- reached deterioration or improvement
- reliability of the operator (Is always a result returned? Did it crashes sometimes?)
- ...

16 The genetic operators for Fib

The operators are to be seen separately from the genetic algorithm. It should be possible to add any number of operators for the use in the genetic algorithm, without having to adjust it.

Each operator has a unique identifier or ID, through which related values (such as its performance to date) can be associated to it.

An running operator is called operation.

The operations are intended to implement coding algorithms. Operators should therefore not be as simple as possible, but may as well involve complex algorithms.

The operators should also be numerous, and the algorithm is responsible for the selection of good operators. Therefore, it is also desirable to create own operators with good parts of other operators. In this case, both the original operators and the new operator, which contains the extracted parts, should be in the genetic algorithm.

For special applications, a genetic algorithm may be used, which set of operators was restricted to the most useful operators for this application. Also, for a application useful operators can be combined to a (not genetic) algorithm, that uses them in determinated way.

16.1 Reproduction

In the general "reproduction", a individual is choosen from the population, preferably a individual with an high fitness, than it is copied and modified by one operation. At the end it is added to the population. It can also be checked, if it is a "new" object (if there is no equal object in the population) and just added in this case, so there are no duplicate individuals in the population.

In the algorithm the reproduction is implemented by running an operator. This operation retrieves all the required data with the help of the operation interface of the algorithm. This required data may include one or more individuals or data on the recent course of the algorithm (e. g. the number of previous interactions respectively operations).

17 The social aspect of the genetic algorithm

The strict separation of the operators from the algorithm and the ability to easily add operators, has its roots in rather social considerations.

Normal coding algorithms are limited to the encoding methods of one or a few ideas of some (in the order of 1 to 100) people (the development team). But there are world wide much more people (on the order of probably 100000) who deal in a broader sense with efficiently and easy encoding of multimedia objects, thereby inevitably many ideas to code multimedia objects remain ignored. Even if some of these people have an interested in contribute their ideas, this is very difficult or impossible to realize.

Everyone can contribute new ideas to the Fib algorithm in the form of new operators (a term for this is "crowdsourcing"). To do this, it is sufficient to only possess the knowledge for the Fib multimedia description language and the interface for the operators of the algorithm. In order to introduce new ideas or operators, for efficiently coding of multimedia objects, no adjustments to the algorithm or other operators should be required. This will limit side effects and any implementation of an idea has only to worry about the idea respectively its operator. So there are no further knowledge needed of the algorithm or even other operators. In this way the system can grow, without increasing the complexity of the (base) system and with this the difficulty for maintenance and expansion of the system.

The algorithm and the image description language should be designed so that layman can train themself without much effort to implement new ideas respectively operators. In particular for students in computer science (or people with computer science as their hobby), it should be possible to train themself within a few days,

so that they can implement their own operators and integrate them. (How useful or effective this is, should be of no concern here.)

The GNU GPL license, under which the algorithm is published, clarifies the legal situation for new operators. This allows to add new operators to the algorithm, unless other rights are infringed (an injury would be, for example, that the operators use an algorithm or code which are under incompatible licenses). New operators can also be made available to the public.

The rating of the operators should motivate people to integrate their own operators. Thus, anyone who has added an operator can realisticly assess, how good his operator is in relation to other operators in a situation. Competition among operator authors should be beneficial to the creation of new and better operators.

All this should result in that not only a small development team will contribut to improve the encoding of Fib objects, but that a much larger group of people are concerned with this issue. This should give the development and propagation of Fib a further boost.

In this sense, the genetic algorithm for Fib is a trans ingenious algorithm, that is designed to extract the ideas of people to encode multimedia data from their heads and then transport/collected them into one pool. Thus, these ideas/algorithms can accomplish more than they could individually. The algorithm can thus combine more intelligence than one human (or a small group) can produce.

18 Why genetic algorithms are good for encoding multimedia objects

There is, in general, no algorithm for converting raster graphics directly into memory-friendly vector images, where the strengths of the vector description language are actually used. For the conversion from one multimedia format into another, there is often a similar problem, especially when in the first multimedia format, the properties of points of a euclidean (discrete) space are given directly (for example, to be able to store images directly) and the second multimedia format encodes the multimedia objects with complex objects (such as rectangles and / or circles).

Since a genetic algorithm has the potential to generate all possible descriptions in a multimedia language and among these, of course, also good descriptions will be, that use the strengths of multimedia language in respect of the original object, a genetic algorithm has also the potential to generate good descriptions. If existing knowledge was well implemented into genetic algorithm, it can find good descriptions probably faster than a pure random search.

Yet another advantage of genetic algorithms is the great freedom in the choice of problem description (multimedia language) and the possible operators on it. This allows complete freedom to design a multimedia language according to your own ideas, wich has certain properties, such as readability or simplicity. Into the operators arbitrarily much knowledge can be incorporated. It is, for example, possible to use known good algorithms (e. g. to translate raster images into vector

images) or parts of them in operators, so that the result from the genetic algorithm (on raster images) is at least as good as the algorithm, but even better results can be generate.

The major disadvantage of genetic algorithms that they need very much time or computing time is weakened by the fact, that this high initial cost can be "cheap" and pays off later. The genetic algorithm can run, for example, as a background process with low priority, so that it consumes superfluous processing power. Later, with the result that it has supplied, much bandwidth can be saved.

This all suggests to use genetic algorithms to encoding multimedia objects.

19 Complexity estimation

With the approach of section 12.1 on page 65, it is possible to converted an arbitrary raster image into an Fib object, with a computing time growing linearly to the number of points and ther properties, as each point and its properties can be easiely added to a list element and the required time for the creation of the other Fib elements (of the root-element) can be performed in constant time.

This approach can be extended to any multimedia data, which are represented as properties of points of a finite, euclidean and discrete space. Thus always an operator can be constructed, that converts a multimedia object in linear time with the number of points and properties into a corresponding Fib object. This Fib object grows only linearly in size with the number of points and ther properties of the original multimedia object. This operator, however, doesn't produced good Fib object representations, because it does not use the possibilities of Fib.

How much effort is required to produce better Fib objects, is very difficult to estimate, since both the operators and the multimedia objects can be arbitrary. However, it can be assumed that, for multimedia objects with simpler structures the effort is lower.

20 Analogy to the natural evolution

As an analogy to the natural evolution strains of bacteria (e. g. E. coli) are used here. They have genes, on wich basis genetic evolution occurs in them.

In these bacterial strains (e. g. E. coli) gene transfer is possible, whereby some of the genetic information (genetic material or its copies) of one bacterium is transfered to another. There are also mutation processes.

A Fib object can be seen as information, which encodes a multimedia object, as bacterial genes encoding a bacterium (structure and behavior).

In this the single Fib elements shouldn't be seen as the bases of the genes, but as the functionality of genes or set of genes. As with bacteria only with the combination of genes or of the things that they encode (e. g. enzymes) more complex functions are being put into effect (e. g. conversion of sugar into kinetic

energy), in Fib objects only by combining of the elements (partial) multimedia objects (e. g. [partial] images) are created.

Fib part objects can also be transmitted, as in gene transfer among bacteria, into other Fib objects and are subject to mutation, wher it is even with the bacteria difficult to say, whether the mutation of genes is really completely random or whether, in the course of millions of years, no mechanisms have emerged, leading to a more "intelligent" mutation, and wherein the random element consists. Individual Fib objects may be viewed not only as individual bacteria, but also as all bacteria with identical genetic information.

The original multimedia object represents the niche, to which the Fib objects (bacteria) should adapt to. They can do this in many different ways, and the adjustment need not be perfect.

However, in the genetic algorithm for Fib objects, operators are sought, which will make directed improvements. Such a mechanism is explicitly directed, which is not to be expected of bacteria evolution.

A good book on evolution in general is "The Plausibility of Life: Resolving Darwin's Dilemma" ("Die Lösung von Darwins Dilemma" [12]).

Part IV
Fib storage format

In this section, the Fib storage formats are presented, in which complete Fib objects can be stored. There are two Fib storage formats: a format for compressed storing (see section 21 on page 80) and a format to represent a Fib object as a readable XML structure (see section 22 on page 114).

21 Compressed storing of Fib objects

Since a space-saving representation is one of the main objectives of the Fib multimedia description language, the Fib elements can be stored with few bits respectively bytes. For this a compressed storage format is defined here. Ther will be no standard compression algorithm used, since it would take no account of the specific characteristics of the Fib multimedia description language.

The representation of **integers** is in two's complement system. For **natural numbers** (including the 0) the binary numeral system is used. The number of bits for vector elements or variables are determined by the corresponding domain (e.g. for numbers for the horizontal dimension or a grayscale value) or fixed specified (for example, numbers for the byte offset in the root-elements).

Real numbers are stored as floating point numbers. The number of bits for the exponent and mantissa are given by the respective domain definition.

The numbers are stored in little endian format.

All **texts** are encoded in Unicode.

Unfortunately, in the following description the **notation of the bits** isn't easy to implement. In the normal form (or spelling) numbers are written from right to left, but text is written from left to right. This means that short bit sequences, as they appear in a byte, are written rather how numbers are written from right to left. However, if the bit sequences gets longer or when the bits are considered in a file or data stream, writing them as in the spelling of text from left to right is more appropriate. Otherwise, if the number notation would be used, at the end of a long line of bits the first bit from the file would stand.

Therefore, in the following the bits of individual elements such as numbers are written from right to left, as they are usually short bit sequences. Individual elements are usually also displayed and processed on the computer in the hexadecimal system, which works, as well as numbers, from right to left. The reverse writing would make it difficult to work with compressed elements. The bits of individual bytes are for clarity mostly separated with a semicolon (';').

If, however, a series (or stream) of several elements in a data stream is displayed, the bits are arranged like in text from left to right. This unfortunately leads to the fact, that the bits of each element are presented in reverse order.

21.1 File Header

Every compressed Fib data stream begins with the three letters “fib”. The file extension for compressed Fib files should “.fib”.

21.2 Root-element

For the description of the root-element see section 9.14 on page 33 .

The Fib root-element does not need a separate introduction. The data of the top most root-element starts with the third byte (the counting starts from 0) immediately after the stream initiation of “fib”. Other root-elements follow after their identifiers.

For the root-element the following fields are written sequentially (where each element is filled in each case if needed to a full byte with 0):

1. 16-bit field to specify the optional information fields (see section 21.2.1)
2. 64-bit field to specify additional optional information fields (see section 21.2.1); only present if the bit 16 of the optional information field is set
3. a 144 ($= 16+64+64$)-bit field for the checksum (see section 21.2.2 on page 85); only present if bit 1 of the optional information fields is set
4. number of the byte of the root-object (offset), from which the domains ($Domains$ and $DomainsValues$) are defined; only present if the bit 3 of the optional information field is set
5. number of the byte of the root-object (offset), from which the input variables are defined; only present if bit 4 of the optional information field is set
6. number of the byte of the root-object (offset), at which the main-Fib-object begins
7. number of the byte of the root-object (offset), from which the sub-root-objects are defined; only present if bit 6 of the optional information field is set
8. number of the byte of the root-object (offset), from which the database identifiers are listed; only present if bit 7 of the optional information field is set
9. number of the byte of the root-object (offset), at which optional part begins; only present if bit 8 of the optional information field is set
10. number of the byte of the root-object (offset), at which root-object ends, respectively the number of byts the root-object is long
11. the multimedia information (see section 21.2.3 on page 85), only present if bit 2 of the optional information field is set

12. the domains (see section 21.2.4 on page 87); only present if bit 3 of the optional information field is set
13. input variables (see section 21.2.5 on page 100); only present if bit 4 of the optional information field is set
14. main-Fib-object (see section 21.2.6 on page 100)
15. sub-root-objects (see section 21.2.7 on page 100); only present if bit 6 of the optional information field is set
16. database identifiers of used database objects (see section 21.2.8 on page 101); only present if bit 7 of the optional information field is set
17. the optional part compressed with the Deflate-algorithm for the lossless data compression (see section 21.2.9 on page 101); only present if bit 8 of the optional information field is set

If a single bit is not set in the optional information fields for a field, this field is omitted.

For fields with the “number of the byte of the root-object, at which ...” 8 bytes or 64 bits are used each. The number in the field is in the domain of the natural numbers. Given in each case is the number of the byte, from the beginning of the root-element, at which the corresponding element beginns (i.e. for the first byte of the element). The count of bytes in the root-element starts at 0 . The optional information field thus has the offset 0 .

All texts that are not contained in the optional part, are moved into the optional part (see section 21.2.9 on page 101).

21.2.1 Optional information fields

At this point a 16-bit fild stands, whose bits indicate the presence of optional information fields in the root-element.

The individual bits (counting starts with 1) switch the following information fields:

1 checksum (see section 21.2.2 on page 85)

2 multimedia information (see section 21.2.3 on page 85)

3 domains (see section 21.2.4 on page 87)

4 input variables (see section 21.2.5 on page 100)

6 the sub-root-objects (see section 21.2.7 on page 100)

7 identifiers of used database objects (see section 21.2.8 on page 101)

8 optional part (see section 21.2.9 on page 101)

- bits 9 till 15 have not been established and are available for future use

16 additional optional fields, indicated by a following 64-bit field for additional optional information fields

The following section describes the individual bits, when they are used and their impact.

1. checksum bit

If set: An checksum field in the root-element exists.

Effect if set: A checksum field (see section 21.2.2 on page 85) exists.

If not set: Ther is no checksum field in the root-element.

Effect if not set: A checksum field dosn't exists.

2. multimedia information bit

Is set if: For the current root-element multimedia information exist, which may differ from the inherited multimedia information. The multimedia information from a root-element will be inherited by a contained root-element, when the contained root-element does not define itself (any) different multimedia information.

Effect if set: The current root-element specifies own multimedia information (see section 21.2.3 on page 85).

If not set: The multimedia information are the same as the inherited multimedia information.

In the top most root-element (respectively a root-element that exists in no other root-element), the bit must always be set, and thus the multimedia information have to be present.

Effect if not set: There are no multimedia information given in the current root-element. The valid multimedia information for the root-element are inherited from the root-element in which it is contained.

3. domains bit

Is set if: For the current root-element domains exists.

Effect if set: For the current root-element domains are given (see section 21.2.4 on page 87) and the offset, at which byte they start in the root-element.

If not set: For the current root-element no domains exists.

Effect if not set: No offset is given for the domains in the current root-element and no domains are given.

4. input variables

Is set if: For the current root-element input variables exists.

Effect if set: For the current root-element input variables are given (see section 21.2.5 on page 100) and the offset, at which byte they start in the root-element.

If not set: For the current root-element no input variables exists.

Effect if not set: No offset is given for the input variables in the current root-element and no input variables are given in the current root-element.

6. sub-root-objects

Is set if: For the current root-element sub-root-objects exists.

Effect if set: For the current root-element sub-root-objects are given (see section 21.2.7 on page 100) and the offset, at which byte they start in the root-element.

If not set: For the current root-element no sub-root-objects exists.

Effect if not set: No offset is given for the sub-root-objects in the current root-element and no sub-root-objects are given in the current root-element.

7. identifiers of used database objects

Is set if: For the current root-element identifiers of used database objects are listed.

Effect if set: For the current root-element identifiers of used database objects are listed (see section 21.2.8 on page 101) and the offset, at which byte they start in the root-element.

If not set: For the current root-element no identifiers of used database objects are listed.

Effect if not set: No offset is given for the identifiers of used database objects in the current root-element and no identifiers of used database objects are given in the current root-element.

8. optional part

Is set if: For the current root-element an optional part exists.

Effect if set: For the current root-element an optional part exists (see section 21.2.9 on page 101) and the offset, at which byte it starts in the root-element.

If not set: For the current root-element no optional part exists (maybe because it was not stored to save space).

Effect if not set: No offset is given for the optional part in the current root-element and no optional part is given in the current root-element.

16. more optional fields

Is set if: 64 more optional information bits are present.

Effect if set: After the 16-bit field, to determine the existing optional information, follows one additional 64-bit field for the determination more optional information. These bits are for future uses and are currently not used.

If not set: The 64 more optional information bits are not present.

Effect if not set: After the 16-bit field no more bits follow for additional optional information.

21.2.2 Checksum field

With this field a checksum can be provided for the root-element.

The procedure is identical to the procedure of the checksum property from section 21.3.3 on page 104.

At this position there are 3 parameters, which are one 16-bit and two 64 bit integers. The first parameter A is the type of checksum. The second parameter B is any number of bits, a checksum should be generated and the third parameter C, how many bits the checksum should have. The information applies to the area after the three parameters (even in sub-root-elements). The checksum is implemented as described in section 21.3.3 on page 104.

21.2.3 Multimedia information

In Table 9 the structure of the multimedia information section of a root-element is described. The size of the multimedia information section is $2 * 64 = 128$ bits or 16 bytes.

element	number	bits	typ	description
Fib version	1	64	natural number	The version number of the Fib multimedia description language, which is required to load the Fib object. This number increases with each new version of the Fib multimedia description language by one. It can be mapped to a human-readable form (e.g. “Fib V1.2.3”). Here, however, only a number is used, otherwise a particular form would be used, which can be changed only with difficulty afterwards. A human-readable form of the version may be specified in the optional part.
DB-version	1	64	natural number	The version number of the Fib database, which is required to load the Fib object. This number increases with each new version of the Fib database by one. It can be mapped to a human-readable form (e.g. “Fib DB V1.2.3”). Here, however, only a number is used, otherwise a particular form would be used, which can be changed only with difficulty afterwards. A human-readable form of the version may be specified in the optional part.

Table 9: Data of the multimedia information

21.2.4 Domains

The domain section consists of two lists with entries of different length. The lengths of entries are determined by ther content.

The domain section starts with a 64 bit integer that pecifies the number of entries for the domain list. After which follows the domain list.

This list in turn is followed by a 64 bit integer that specifies the number of elements in the value domains list (the domains for values), and then the list with the value domains.

In this, one or both lists can be empty. For an empty lists only the introductory 64-bit number will be stored, which is in that case 0 .

The domain list identifies the domains of the individual elements. If a value (e.g. a variable) is outside the domain, it is rounded to a value within the domain. Values outside these domains can thus not occur for the element.

The domains for values contain also domains, but these domains only apply to actual values in elements and not for values of contained variables. The domains for values will determine how many bits are needed to store an element that contains a value. The domains for values are useful when the values of an element do not cover the full possible range of the domain for the element. For example, if a subobject contains only points whose position vectors only contain variables and integer values between 0 and 10, the domain for values for position vectors can be set to “integerB” with 4 bits, even if the variables of position vectors taking values greater 100 .

If while saving a value is found in main-Fib-object that lies outside of the actual domain of the element, the domain is automatically expanded so that it includes the value. If the domain was inherited, in the root-element a corresponding new domain is created, which includes the value and the inherited domain.

The domains for values may be generated and optimized when storing the Fib object, to keep the space for the Fib object as small as possible. A domain for values is generated when saving, if it is likely that through its generation space will be saved.

Because many domains can belong to a Fib multimedia object, for them more attention is paid to their storage space. To make future upgrades easy, the emphasis is on flexibility.

The reason for the introduction of central (in the root-elements) domains is, that on the one hand as little space as possible for values should be used when saving the Fib object, without drastically limit the assignment possibilities for the values and on other hand that it can be determined in advance if and how the multimedia object can be displayed (e.g. if and how it should be scaled or whether the display of all values is impossible). If for example, the domain for the dimension takes only integer values between 0 and 50 (e.g. the horizontal in an image), then 6 bits is enough to store the values for the dimension. For larger images simply more bits for the values of the dimension can be used.

Each entry consists of two parts:

1. The name / type of the element for which the domain is (see section 21.2.4).
2. The specification of the domain for the element (see section 21.2.4).

Element names The element name / type can only consist of the specification of a fixed element or a fixed element and a parameter.

The first bit (counting begins at 1) of the fixed element determines its length:

- If it is 0, the name is 8 bits long. Possible values are listed in table 10 .
- If it is 1, the name is 64 bits long. This is currently only planned for a future use.

The second and third bits determine the length of the parameter:

00 ther is no parameter

01 parameter with a total length of 8 bits follows

10 parameter with a total length of 64 bits follows

11 After the element name field follows an 16 bit natural number, wich specifies the parameter length in bytes. The 16 bits for the length of the parameters are not included in the evaluation for the parameter length.

name	value bit 4 till 8	description
dim	0000 1	The domain is for position vectors (see section 9.2 on page 6 and section 21.3.2 on page 103) respectively dimensions. The length of the parameter list is variable. The second and third bit is thus 11 and after bit 8 follows a 16 bit natural number L, which specifies the length in bytes of the parameter list. The first parameter following is a 16 bit natural number and specifies the number of dimension Dim. Then Dim more parameters $Dimmap_1$ till $Dimmap_{Dim}$ follow up, they are a natural numbers with each of them of the length $L_{Dimmap} = \lfloor((L-2)*8)/Dim\rfloor$ (per parameter the still remaining bits of the parameter L divided by the number of dimensions and the result is rounded to an integer). The values that can be taken by the $Dimmap_i$ parameters are described in table 7 on page 46. The length of the parameter list L is to be determined in such a way, that it is just enough room for all parameters.
subfunction	0001 0	domain for the elements of subfunctions (see section 9.7 on page 17 and section 21.3.7 on page 107)
property	0010 0	This is the domain of a property element with the given name (see section 9.3 on page 7 and section 21.3.3 on page 104). The value for the name respectively the property type is passed in the parameter. Possible values are listed in table 2 on page 12. The value that is specified as a parameter, is a natural number. The parameter is only so long (e.g. 8 bits), as for the representation of the value as a natural number from table 2 is required.
inVar	0010 1	This is the domain for the i'th input variable (see section 9.14 on page 33 and section 21.2.5 on page 100). The following parameter is an integer and specifies the number i of the input variable. (The count of the input variables of a root-element starts with 1.)
Names for elements of domains, which are created (if needed) when storing a Fib object		

name	value bit 4 till 8	description
area	0001 1	This type is for domains for the area element (see section 9.6 on page 16 and section 21.3.6 on page 107). The corresponding domain is a vector domain with 2 elements / subdomains. The first element or the first subdomain is used for the number (n) of subareas / vectors, it is part of the domain of the natural numbers. The second element is the domain for the subareas (B_1), it is a vector domain with two elements or subdomains, each of which come from the domain of integers.
variable	1000 1	Values that are needed to encode variables. The domain should include the natural numbers from 0 to maximum number of variables defined in the Fib leafs in the main-Fib object. The Fib tree-leaf in the main-Fib-object above which the most variables are defined respectively the branch with the most defined variables, thus determins the domain. This entry is created when saving.
comments	1001 0	Values that are needed to encode comments (see section 9.5 on page 14, section 21.2.9 on page 101 and section 21.3.5 on page 106). The domain should include the natural numbers from 0 to the number of comments in the main-Fib-object. This entry is created when saving.
externObject	1100 0	This is the domain type for external objects (see section 9.9 on page 25 and section 21.3.9 on page 111) in the main-Fib-object. The domain is a vector with 4 elements. The vector elements are in ther the order for the identifier, the number of input values, the number of subobjects and the number of output variables. All vector element domains, except for the identifier, are part of the natural numbers. The vector element domain for the identifier is part of the integers. This domain is usually created when saving.
externObject-Input	1110 0	This domain is for the input values for external objects (see section 9.9 on page 25 and section 21.3.9 on page 111) . The domain is an vector domain and is usually created when saving. The following parameter is an integer and determines the identifier of the external object, for which elements the domain is.

name	value bit 4 till 8	description
externSubobject	1100 1	This domain type is for the input values for external subobjects (see section 9.10 on page 26 and section 21.3.10 on page 112). The domain is an vector domain and is usually created when saving. The following parameter is a natural number and determines the number of the external subobject, for which elements the domain is.
setElement	1101 0	This type is for the domain for the set-element (see section 9.12 on page 30 and section 21.3.12 on page 113). The corresponding domain is a vector domain with 3 elements / subdomains. The first element or the first subdomain is used for the number (n) of variables and number of values to be set per set, it is part of the domain of the natural numbers. The second element or the second subdomain is used for the number (k) of the sets of values to be set. It is also part of the domain of the natural numbers. The third and final element is the domain for the vectors for the values to be set ($W_{i.g}$) and is a domain for vectors, which element- or subdomains are domains for numbers (scalar). Further as parameter a natural number can be specified for the domain number $DomainNr$. If the parameter is missing, the domain number $DomainNr$ is 0 .

name	value bit 4 till 8	description
matrixElement	1101 1	This type is for the domain for the matrix element (see section 9.13 on page 31 and section 21.3.13 on page 113). The corresponding domain is a vector domain with 4 elements / subdomains. The first element or the first subdomain is used for the number (d) of dimension variables, the number i of value variables and number of values i to be set per set, it is part of the domain of the natural numbers. The second element or the second subdomain is used for the number (k) of the sets of values to be set. It is also part of the domain of the natural numbers. The third element is the domain for the areas respectively for the start and end values for the different dimension variables, it is a vector domain with two elements or subdomains, each of which come from the domain of integers. The fourth and final element is the domain for the vectors for the values to be set ($W_{a.b}$) and is a domain for vectors, which element- or subdomains are domains for numbers (scalar). Further as parameter a natural number can be specified for the domain number $DomainNr$. If the parameter is missing, the domain number $DomainNr$ is 0 .

Table 10: Names of fixed 8-bit elements for domains

Domains The domain consists of a basic domain and maybe parameters, for further specification of the domain.

that specify the basic domain further.

The first bit (counting starts with 1) of the basic domain determines the length of the field:

- If it is 0, the basic domain field is 8 bits long. Possible 8-bit basic domains are shown in table 11.
- If it is 1, the basic domain field is 64 bits long. This is currently only planned for a future use.

The second bit indicates whether the domain is scaled:

- If it is 0, the domain is not scaled.

- If it is 1, the domain is scaled.

If the domain is scaled, a scaling factor S is specified. This factor S is a float, the unscaled values of the domain will be multiplied with it to obtain (the values of) the scaled domain.

The scaling part is specified after the domain information (including parameters). It consists of three fields: the length field, the mantissa field and the exponent field. The first field (the length field) is 8 bits long, it is a natural number and specifies the number of bytes for both the mantissa and the exponent field. The mantissa field (second field) S_M and the exponent field (third field) S_E are each integers. The scaling factor S is then $S = S_M * 2^{S_E}$.
Example: The domain is 2 bit natural numbers with the scaling factor of $F = 1 * 2^{-1} = 1/2$ (the mantissa is 1 and the exponent is -1, the length field is 1, for one byte scaling factor field). The bits of the scaling factor are thus: 0000 0001; 0000 0001; 1111 1111. Possible values for the elements for the domain are: $\{ 0 * 1/2 = 0; 1 * 1/2 = 0.5; 2 * 1/2 = 1; 3 * 1/2 = 1.5 \}$. If in the element to the domain a number is set that would mean 1 unscaled, it means with the scaling 0.5. The bits of the entire domain are: 0; 1; 000000; 00000010; 00000001; 00000001; 11111111 (Fields in this order: 8-bit basic domain; scaled; natNumberB, with 2 bits per element; scaling factor: 1 byte per field, $S_M = 1$, $S_E = -1$) (in the file, first bit at the front: 01000000 01000000 10000000 10000000 11111111)

The individual domains (including scaling factor) are padded with 0 to full Byts. So if a domain is 13 bits long, the remaining 3 bits are filled with 0, so that the domain field is 16 bit (=2 byte) long.

name	value bit 3 till 8	description	parameter
naturalNumberB	0000 00	The basic domain are the natural numbers.	The following 8-bit parameter is a natural number X, which represents the number of bits for values of the domain. The corresponding basic domain is then $0 \dots (2^X - 1)$.
integerB	0100 00	The basic domain are the integer numbers.	The following 8-bit parameter is a natural number X, which represents the number of bits for values of the domain. The corresponding basic domain is then $-(2^{X-1}) \dots (2^{X-1} - 1)$.

name	value bit 3 till 8	description	parameter
naturalNumber	0000 01	The basic domain are the natural numbers.	The following 64-bit parameter is a natural number X, which is the largest natural number in the domain. The corresponding basic domain is then $0 \dots X$. The bits that are needed for each value of the domain are $\lceil \log_2(X+1) \rceil$.
integer	0100 01	The basic domain are the integer numbers.	The following two 64-bit parameters are two integers X and Y. The first parameter X is the smallest number in the domain. The second parameter Y is the biggest number in the domain. The corresponding basic domain is then $Y \dots X$. The bits that are needed for each value of the domain are $\lceil \log_2(Y-X+1) \rceil$. When interpreting the bits, all values W (W is the value of the bits as a natural number), which are greater than X, will be implemented as negative values with a value of $W-(X-Y+1)$. If the smallest number X is greater than 0 or largest number Y is less than 0, smallest number X is subtracted from all numbers to save them ($W+X$).
integerValues	0100 10	The basic domain are the integer numbers	The first 64-bit parameter is a natural number N, which indicates the number of possible values. Following after it is a second 8-bit parameter B, wich determines the number of bits per following value. After the first two parameters N integers follow, each with B bits in two's complement. This N integers are all values which are contained in the basic domain. In the implementation the value W in an element for the domain is mapped to / interpreted as the W'th number in the list (the numbering starts at 0). The bits that are needed for each value W of the domain are $\lceil \log_2(N) \rceil$.

name	value bit 3 till 8	description	parameter
real	1000 00	The basic domain are floating point numbers. A floating point number consists of two integer fields, one the first, for the exponent E and one, the second, for the mantissa M. The floating point number Z is then $Z = M * 2^E$.	It follow two parameters. The first parameter specifies the domain of the mantissa and the second for the exponent. The specification of the domains is as described in this table (without the domains individually padded with 0). Both domains have to be domains for integers (ie. integer... or naturalNumber...).

name	value bit 3 till 8	description	parameter
realValues	1000 01	The basic domain are floating point numbers. A floating point number consists of two integer fields, one the first, for the exponent E and one, the second, for the mantissa M. The floating point number Z is then $Z = M * 2^E$.	The first following 64 bit parameter is a natural number N, which indicates the number of possible values. Then follow two parameters. The first parameter specifies the domain of the mantissa and the second for the exponent. The specification of the domains is as described in this table (without the domains individually padded with 0). Both domains must come from the domains of integers (ie. integer... or naturalNumber...). After the first three parameters N floating point numbers follow, each with $B = B_M + B_E$ bits in the floating point number representation for the given mantissa and exponent domains. In which B_M are the bits per mantissa and B_E are the bits per exponent. This N floating point numbers are all values the elements for the domain can be set to. For this the value of W, in an element to the domain is mapped / interpreted to the W'th number in the list (counting begins at 0). The bits that are needed for each value of the domain are $\lceil \log_2(N) \rceil$.
vector	1100 00 and 1100 01	The basic domain are vectors.	The following parameters E is the number of elements in the vectors. The ("number of") parmeter after the introduction "11 0000" is 8 bits long and after the introduction "11 0001" 64 bits. After this parameter follows a list of E domains, as defined in this section. All the values of the vector domain will have the form $(D_1, \ldots, D_E)$, where D_i is a value of the i'th domain in the domains list.

name	value bit 3 till 8	description	parameter
vectorValues	1100 10 and 1100 11	The basic domain are vectors.	The following parameters E is the number of elements in the vectors. The second parameter N gives the number of possible vectors. Each ("number of") parmeter after the introduction "1100 10" is 8 bits long and after the introduction "1100 11" 64 bits. After the second parameter a list of E domains follows, as defined in this section. Then follows a list of N vectors, as described in section 21.3.1 on page 102. The domains of the vectors are the given domains of the preceding list. Variables can also occur in the stored vectors. The domain for variables in the list has 0 bits. For a variable in the vector only the initial 1 is written. A value of W for the domain in a Fib element is interpreted as W'ter vector of the second list. (The counting starts at 0.) If the specified vector contains variables, directly after the value of W follow the variable identifiers of variables, as they wher defined above the current Fib element. The number of bits for the variable is given by the corresponding domain for variables "variable" (see section 21.2.4 on page 87).

name	value bit 3 till 8	description	parameter
vectorOpenEnd	1110 00 and 1110 01	The basic domain are vectors.	The following parameters E is the minimum number of elements of the vector. The ("number of") parmeter after the introduction "1110 00 " is 8 bits long and after the introduction "1110 01 " 64 bits. After this parameter follows a list of E domains, as defined in this section. All the values of the vector domain will have the form $(D_1, \ldots, D_E, \ldots, D_E)$, where D_i is a value of the i'th domain in the list. This domain is for elements that may contain vectors of different size. The number of elements for a vector of this domain is determined by the element containing the vector.

name	value bit 3 till 8	description	parameter
domainReference	1111 00	This is a reference to the (sub-)domain of an other element. The domain is the domain of the element with the given domain name $Name$.	The first parameter is the in the compressed format coded $Name$ of the element, to which the domain refer to (see section 21.2.4 on page 88, without padding to a full byte). After it follows the $Element$ parameter, for the choosen subdomain. First follows (each) an $Element$-startbit, which indicates if an $Element$ parameter follows. If it is 0 no $Element$ parameter follows, if it is 1 an $Element$ parameter follows. If it is 1 this first $Element$-startbit is followed by a 1 byte (8 bits) long natural number $Bits$, which indicates how many bits per $Element$ parameter are used. After this follows the first $Element$ parameter. After each $Element$ parameters (stored with the domain $naturalNumberB(Bits)$) follows again an $Element$-startbit and after it maybe the next $Element$ parameter and so forth. Example $matrix$.3.1: 2 Bits are needed to store the $Element$ parametes; the bits are (first bit on the front): 0 0 001111 0 00 11011 1 00000010 11 1 10 0 (in ther order the fields are for: domain name is 8 bit long; not scaled; domainReference; 8 bit element name; no parameter; matrixElement; $Element$ parameter follows; with each 2 bit; 3'th subdomain; $Element$ parameter follows; first subdomain; no $Element$ parameter follows)

name	value bit 3 till 8	description	parameter
defaultDomain	1111 01	The specified domain will be used only if for the corresponding element so far no other domain was given.	As a parameter follows a domain as described in this table. The specified domain is only used, if for the corresponding element so far no other domain was given.

Table 11: 8 Bit parameters for domains

21.2.5 Input variables

In compressed Fib objects variables are numbered and represented as natural numbers. In this, the first n variables values in the Fib object are reserved for the n input variables. Therefore, at this point for the variables only the number of input variables V_E is interesting. For this a 64 bit natural number is stored at this location, which contains the number of input variables. After the field for the number of input variables (V_E) follow V_E values, each, in ther order, for the default value S_i of the input variable $inVar_i$. The number of bits and the encoding of the default value of S_i is determined by the domain of the input variable $inVar_i$.

The input variable field is padded with 0 bits to a full byte.

21.2.6 Main-Fib-object

At this point the data from the main-Fib-object in form of its elements and their parameters follows, as it is described below in the section 21.3 on page 102 .

The field for the main-Fib-object is padded with 0 bits to a full byte.

21.2.7 Sub-root-objects

Here a list of sub-root-objects and their identifiers stand. The list is initiated with a 64-bit natural number N, which specifies the number of elements in the list. This is followed by a 8-bit integer B, which indicates the bytes per identifier.

Then follows a list of N pairs of identifiers and root-elements. The identifier of each pair is an integer (in the root-element always greater than 0, in the database less than 0). It is the first element in the list element pairs and is B bytes long. It is followed by the root-element, as described in this section 21.2 (beginning at page 81). Each pair is padded with 0 bits to full a byte.

21.2.8 Identifiers of used database objects

At this point there is a list of all identifiers of Fib database objects, which are used in the main-Fib-object or in sub-root-objects. This specification of the identifiers is optional. If identifiers are present, it can be tested from the outset, that all external Fib objects from the database that are needed exists, or whether it is likely that display errors occur, since Fib database objects are missing. Whether the identifiers of Fib database objects that are used in a main-Fib object of a root-element are given in this root-element, or in a higher root-element, depends on several factors. For the changing of Fib objects it is advantageous, that identifier of Fib database objects are listed in a root-element, which is as near as possible to the place of ther use. To save space it may be useful to subsume the identifiers of Fib database objects in as few as possible root-elements.

The list is initiated with a 64-bit natural number N, which specifies the number of elements of the list. This is followed by a 8-bit natural number B, which specifies the bits per identifier. Then follows the list of N integer identifiers, which is each B bits long.

The field for the identifiers of used database objects is padded with 0 bits to a full byte.

21.2.9 Optional part

The optional part is the last part of a root-element. It shouldn't contain important information for the presentation of multimedia object, so that it can be omitted entirely when storing. In this way, storage space can be saved.

The length of the optional part is calculated from the difference between its starting byte and the end of the root-element (for the offset see section 21.2 on page 81).

The optional part is initiated by a 16-bit natural number C, which indicates the compression type of the optional part. Possible values for the compression type C are listed in table 12 . The remaining optional part will be completely compressed using the specified method.

value C	description
0	no compression
1	The data is compressed in the zlib format. This is a warper (documented in RFC 1950) for a deflate stream (lossless data compression, documented in RFC 1951).

Table 12: Compression types

The decompressed optional part consists of a list of key, value pairs. It is initiated by a 64 bit integer N, which specifies the number of elements in the list. The second parameter is a 16 bit integer, which determines the coding. Possible

values for encoding are listed in table 13 . In this UTF-8 should be regarded as the standard encoding and other encodings should be selected only if UTF-8 is no longer sufficient. Then follows the list with the pairs / elements.

Each list element (pair) contains two null-terminated strings in the specified encoding. The first string is the key of the element and the second is the value.

name	value	description
UTF-8	0	8-Bit Unicode Transformation Format
UTF-16	1	Universal Multiple-Octet Coded Character Set (UCS) Transformation Format for 16 Planes of Group 00
UTF-32	2	Unicode coded one character with 32 Bit

Table 13: Encoding types for strings

From the main-Fib-object all text from the comments (see section 21.3.5 on page 106) are moved into the optional part. The comments also contain key (Key), value ($Value$) pairs. To each of these key, value pairs from the comments will be assigned a natural number K (starting with 0). In the optional part a separate list entry is generated for the moved pair. In this the key of the list entry is "@" followed by the number of the comment A, then followed by a "@" and then the original key (Key) of the comment. The value of the generated list entry is equal to the value of the comment. So if the key value pair in the A'th comment is $(Key, Value)$, the entry in the optional part for it is $(@A@Key, Value)$.

Example: The 2. comment is: "c("autor","ich", Obj)", the generated list entry for it in the optional part is: "("@2@autor","ich")".

The outsourced texts should be at the end of the optional part.

21.3 Fib elements

All Fib elements except the root-element will be introduced by a (at least four bit) bit field. Vectors are directly included in the Fib elements, without ther own introduction.

The introduction 0000 is reserved, so that with 0 padded areas can not be interpreted as a Fib element. The introduction 1111 announces that the introduction is at least 16 bits long and not 4 bits.

All Fib elements and vector elements follow directly behind each other. If positions are padded with the 0 bit, this is explicitly stated hereafter.

21.3.1 Vectors

Vectors consist of several elements, an element is either a value or a variable. The number of elements of the vector is either Fib element specific or is determined by an associated domain (see section 21.2.4 on page 87).

For each element there are two fields:

- one 1 bit field, which determines, if the element is a value (the bit is 0) or a variable (the bit is 1)
- the field with the value for the element (the genuine value or number / identifier for the variable)

If the element is a value, then follows after the 1 bit field with 0 directly a value. The number of bits of the value is determined by the corresponding domain. The value comes from the unscaled domain and is scaled after restoring.

If the element is a variable, then follows after the 1 bit field with 1 directly the variable identifier (which is the same natural number as in the variable definition) of the variable, which was defined above the current Fib element. The number of bits of the variable is determined by the appropriate domain for variables "variable" (see section 21.2.4 on page 87).

The in a particular Fib object-branch highest defined variable has the variable value respectively variable identifier 1 . All other variables have the value $i + 1$, where i is the value of the variable, which is defined next above variable. So the variables are numbered in the Fib object-branches from the top (starting from the root) to bottom (the leaves), with the starting value of 1.

21.3.2 Point

Introduction of a normal point: 0001
Five bits introduction for a point with an empty position vector ($point(())$): 00010
Five bits introduction for a point with no position vector ($point()$): 10010

For the description of the point element see section 9.2 on page 6 .

Normal points contain a position vector (see 21.3.1). The number of vector elements is determined by the number of dimensions respectively elements in the domain for dimensions ("dim"). The bits per position vector element is determined by the particular element of the domain for the dimensions ("dim") or by the domain variables ("variable") (see section 21.2.4 on page 87).

The individual vector elements follow directly after the introduction of the point.

Points with an empty or without a position vector does not contain any bits for the position vector. For these points there are only the 5 bits of the respectively introduction.

Example: A point is encoded with the position vector (2, 10). The multimedia object has two dimensions. For both dimensions, 5 bits are needed, there are natural numbers.

- introduction point: 0001

- first position vector element (leading 0 for: “vector element is a value”) 2 : 0 00010
- second position vector element (leading 0 for: “vector element is a value”) 10 : 0 01010
- coded point object (first bit in front): 1000010000010100

21.3.3 Properties

Introduction: 0011

For the description of the property element see section 9.3 on page 7 .

The property vector type is determined by a natural number, which immediately follows after the introduction. In the list of the valid domains the possible properties are listed (see section 9.14.2 on page 35, table 2 on page 12 and table 5 on page 43). The number for the property (vector) type is the number of the property in the valid domains list. Counted are only property domains and the count starts with 0 .

In the valid domain list first domains for values (see section 9.14.2 on page 47) and than the domains for elements (see section 9.14.2 on page 45 are listed. Inherited property domains, which are not overwritten, will be counted as if they are behind the domains of the current (respectively next) root-element. (The closer the inherited root-element is to the inheriting root-element, the smaller are the numbers of inherited property domains [respectively for values and element domains].)

For property types, that exists in the Fib object, but for which no domain exists, a domain (e.g. the standard domain) must inserted in the domains for values. Only then the property vector types can be numbered.

The number of bits, that are needed for the number of property vector type, results from the number of defined (including inherited) property domains DE as a whole, it is $\lceil \log_2(DE) \rceil$. If for example property domains `colorRGB`, `layer` and `sound` are defined. Then $\lceil \log_2(3) \rceil = 2$ bits are needed to store the property vector type and the property `layer` has the value 01 .

The property vector type is followed by the elements of the property vector (see section 21.3.1 on page 102). In which the number of elements and bits are determined by the respectively domain (see section 21.2.4 on page 87).

At the end the subobject of the property follows.

For the properties there are two special properties, that have influence on the decoding of the Fib object. These properties are “checksum” and “boundSize”.

Checksum The checksum property is used to find erroneous Fib objects and to maybe correct these errors.

Its name is “checksum”. It has 3 parameters. The first parameter A is for the type of the checksum. The second parameter B determines for how many number

of bits the checksum should be generated and the third parameter C determines how many bits of the checksum should have.

Checksum of the type 1 If the first parameter A is equal to 1 the checksum is generated in the manner described in this section.

The checksum stands before the block it belongs to.

The checksum property applies to the entire Fib object respectively Fib branch that is contained in the property element (its entire subobject), except those which have a separate / own checksum property. When loading the Fib object, blocks of the size of $C + B$ bits are loaded, where the block, wich lie at the end of the checksum field, is padded with 0. From each Fib object it will be only read as much as belongs to the checksum field and the rest of the block is padded with 0 .

If the checksum field Cb_1 is interrupted by another checksum field (object) CB_2, after the checksum field (object) CB_2 a new block for the checksum field Cb_1 will be start and read.

When storing the subobject of the property element of the checksum, the entire subobject is first brought into the compressed form, as described in this section 21 . From the created bit field, all areas which not belong to the checksum property are cut out, since they belong to different checksum properties, which are treated separately. As a result of this from the compressed subobject of the checksum property several bit fields Bf_i for the checksum property are created. These bit fields Bf_i are divided into blocks of B bits, where each of the last blocks of a bit fields Bf_i are padded with the 0 bit to B bits. Then for each of the blocks the checksum of C bits computed. After this step the blocks for each Bf_i are assembled in their original order into bit fields, in which for each block first the checksum field of the block and then the associated data block comes. These bit fields are then concatenate into their original order with the other properties checksum fields, which were treated separately, into a bit field for the entire subobject.

The checksum is not only used for error detection, but if possible (if enough checksum bits are present) also for error correction.

Size of Fib objects With the property “boundSize” it will be specified, that the size in bits for an Fib object should be stated. Although no vector elements exists for this property in the created Fib object, one vector element is generated when storing the object, also for the vector a domain is created in the root-element when storing (see section 21.2.4 on page 87).

The vector element in a compressed format contains the number of bits, that the contained Fib object is long (including any checksums). If the Fib object is irreparably defective, in this way the beginning of the next Fib object can still be found and this Fib object can be restored. Without specifying the length of the Fib object Obj_1 the beginning of following Fib object Obj_2 in the data stream can probably not be determined, since it is unknown how long the elements of Obj_1 are. If Obj_1 is irreparably flawed, but the starting point respectively the starting bit

of the next Fib object Obj_2 is known, then indeed the Obj_1 can not be displayed, but this will have no effect on multimedia information outside of Obj_1, so that the error is limited to Obj_1 .

The property of "boundSize" should best be used as the first / top element of a subobject in branch elements.

21.3.4 List element

Possible introductions:

- 0100: introduction for two subobjects
- 0101: less than 256 subobjects
- 0110: less than 2^{64} subobjects

For the description of the list element see section 9.4 on page 14 .

The list element indicates that hereafter come several Fib objects in a row. Directly after the introduction of the list element follows a natural number, which indicats the number of the following subobjects. The length of the field, for the number of subobjects, depends on the introduction of the list element.

The following lengths for the number field exists:

- Introductions of the list element 0100: The field for the number of subobjects is 0 bits long, respectively it is omitted.
- Introductions of the list element 0101: The field for the number of subobjects is 8 bits long.
- Introductions of the list element 0110: The field for the number of subobjects is 64 bits long.

After the field for the number of subobjects follow the subobjects. Where the first following subobject, is the first subobject of the list element, the following is the second subobject of the list element, etc..

More information for the list element is not needed, because the length of subobjects is apparent from the subobjects themselves.

21.3.5 Comment element

Introduction: 0111

For the description of the comment element see section 9.5 on page 14 .

The comment element contains after its introduction a natural number N, which indicates the number of the comment. The domain "comments" of this number is determined by the root-element (see section 21.2.4 on page 87).

This is followed by the subobject of the comment element.

In the optional part of the associated root-element for each comment an entry is created. More details can be found in the section 21.2.9 on page 102 .

21.3.6 Area element

Introduction: 1000

For the description of the area element see section 9.6 on page 16 .

The introduction is follows by a natural number, for the number of subareas. The bits for this number resulting from the first element / subdomain of the vector domain for the “area” type.

After this follow the subareas. The subareas are vectors that consist of two elements (for vectors see section 21.3.1 on page 102). The corresponding domains for the elements of the subarea vectors are determined by the second subdomain of the domain “area” for the values and the “variable” domain for the variables (see section 21.2.4 on page 87).

At the end of the area object follows its subobject.

21.3.7 Function

Introduction: 1011

For the description of the function element see section 9.7 on page 17 .

After the introduction follows the subfunction of the function.

At the end of the function object follows its subobject.

Subfunction
Subfunctions are initiated by two bits for the type of subfunction:

- 00: value
- 01: variable
- 10: subfunction with arity of one
- 11: subfunction with arity of two

If the subfunction is a value (Introduction: 00), the value is directly following after the introduction 00 . The number of bits of the value is determined by the appropriate domain for subfunctions “subfunction” (see section 21.2.4 on page 87).

If the subfunction is a variable (Introduction: 01), the variable identifier /name, of a variable defined above the function element, is directly following after the introduction 01 . The number of bits of the variable identifier is determined by the appropriate domain for variables “variable” (see section 21.2.4 on page 87).

If the subfunction has an arity of one (introduction: 10), a (at least 2 bits) bit field follows directly the introduction 10, which indicates the type of subfunction. Immediately after the bit field follows another subfunction (initiated by the 2 bits for its type), which is the subfunction of the subfunction with arity of one.

Values for the type of a subfunction with an arity of one:

- 00: absolut value
- 01: sine function
- 10: The introduction of rarely used one arity functions. The next two bits indicate the type of the one arity subfunction:
  - 00 10: logarithm
  - 01 10: arc sine
  - 10 10: round
  - 11 10: free for future assignments
- 11: The introduction of rarely used one arity functions. The next 6 bits indicate the type of one arity subfunction:
  - **** ** 11: free for future assignments

If the subfunction has an arity of two (introduction: 11), a (at least 3 bits) bit field follows directly the introduction 11, which indicates the type of subfunction. Immediately after the bit field follow two other subfunctions (initiated each by the 2 bits for their type), which are the subfunctions of the subfunction with arity of two.

Values for the type of a subfunction with an arity of two:

- 000: adding
- 001: subtraction
- 010: multiplication
- 011: division
- 100: exponentiation
- 101: minimum
- 110: maximum
- 111: The introduction of rarely used two arity functions. The next 5 bits indicate the type of two arity subfunction:

- 0000 0 111: if-function (before the two other subfunctions, first the condition the if-subfunction is stored)
- 0000 1 111: delay
- 0001 0 111: modulo
- **** * 111: free for future assignments

Example:

- example object: fun(x_3 , mult(sin(5), x_1), ...)
- bits for "fun" (introduction): 1011
- bits for "mult" (introduction for "two arity subfunction" and "mult"): 11 010
- bits for "sin" (introduction for "one arity subfunction" and "sin"): 10 01
- bits for the value 5 (introduction for "value" and 5 as a 4 bit natural number with the value domain "subfunction" as "naturalNumberB(4)", which means a 4 bit natural number): 00 0101
- bits for die Variable x_1 (the first variable in the part object branch, introduction for "variable" and the 4 bit variable identifier): 01 0001
- bits for the entire function element: 1011 11 010 10 01 00 0101 01 0001 ... (=Bits for the subobject)
- bits for the entire function element (first bit at the front): 1101 1101 0011 0001 0101 0100 0 ... (=Bits for the subobject)

21.3.8 If-element

Introduction: 1100

For the description of the if-element see section 9.8 on page 24 .

After the introduction follows the condition of the if-element.

At the end of the if-object follow the two subobjects. If the condition is true, only the first subobject is evaluated, if the condition is false, only the second subobject is evaluated.

Conditions Conditions are introduced by a 4-bit value for the type of the condition. After the introduction of the condition follows as a parameter either 0 till 2 subconditions or two subfunctions (see section 21.3.7 on page 107)

The possible conditions are listed in table 14.

Example:

name	introduction	description	parameter
false	0000	The condition is false.	none
true	1111	The condition is true.	none
not	0001	The condition is true if and only if the subcondition is false, otherwise it is false.	one subcondition
or	0010	The condition is true if and only if at least one of its two subconditions is true, otherwise it is false.	two subconditions
and	0011	The condition is true if and only if the two subconditions are true, otherwise it is false.	two subconditions
xor	0100	The condition is true if and only if exactly one of the two subconditions is true, otherwise it is false.	two subconditions
eqInt	1000	Comparison of two to integers rounded numbers on equality.	two subfunctions, as described in section 21.3.7 on page 107
lo	1001	Comparison, if the first number is less than the second.	two subfunctions, as described in section 21.3.7 on page 107
gr	1010	Comparison, if the first number is greater than the second.	two subfunctions, as described in section 21.3.7 on page 107
	0101 till 0111 and 1011 till 1110	Free for future conditions.	

Table 14: Introduction for conditions

- example object: if(and(true, gr(x_1, 3)), ...)
- bits for “if” (introduction): 1100
- bits for “and” (introduction for “and”): 0011
- bits for “true” (introduction for “true”): 1111
- bits for “gr” (introduction for “gr”): 1010
- bits for the variable x_1 (the first variable in the part object branch, introduction for “variable” and the variable identifier; the variable domain “variable” is “naturalNumberB(4)”, which means a 4 bit natural number): 01 0001
- bits for the value 3 (introduction for “value” and 3 as a 4 bit natural number with the value domain “subfunction” as “naturalNumberB(4)”, which means a 4 bit natural number): 00 0011
- bits for the entire if-element (first bit at the front):
  00111100 11110101 10100000 1100 ... (=Bits for the two subobjects)

21.3.9 External object

Introduction: 1101

For the description of the external objects see section 9.9 on page 25 .

Directly after the introduction follows the number of the identifier of the external object. Its domain, and therefore the number of bits for it, is determined by the first subdomain of the domain “externObject” (see section 21.2.4 on page 87) .

The identifier is followed by the number of input values E, which has the domain of second subdomain of the domain ‘externObject”. If this value is 0 the number of input values is the number of vector elements of the vector for input values, this means the number of elements of the domain ”externObjectInput” with the corresponding identifier, if non such exists it is assumed that no / 0 input values ($E = 0$) exist.

Then follows the vector (see section 21.3.1 on page 102) with the input values. The domain for the vector is the domain for the “externObjectInput” with the corresponding identifier or $vectorOpenEnd(integerB(8))$, if non such exist.

After the list of input values follow the subobjects of the external object. These are preceded by a number U for the number of subobjects. The domain of this number U is the third subdomain of the domain “externObject”. Then follow one after the other the U records of the U subobjects.

Each record for a subobject is introduced by a number representing the number of output variables A_i ($i = 1 \ldots U$). The domain of this number A_i is the fourth subdomain of the domain “externObject”.

At the end follows the subobject of the record, which is a normal Fib object.

21.3.10 External subobject

Introduction: 1110

For the description of the external subobjects see section 9.10 on page 26 .

After the introduction follows a natural number for the number of the external subobject. The domain of this number is implicit determined by the number N of the subobjects (respectively the highest number for a "externSubobject" domain) in the current / next in the next root-element, see section 21.2.4 on page 92 . The domain is: $integer(N)$, see table 11 on page 100 .

This is followed by the vector (see section 21.3.1 on page 102) of the input values, respectively the input variables for the subobject. The domain for the output values vector is the domain for the "externSubobject" with the corresponding number or the standard domain for "externSubobject", if non such domain exist.

21.3.11 Retrieve domain properties

Introduction: 0000 0000 0000 1111

For the description of the Fib element to retrieve domain properties see section 9.11 on page 27 .

After the introduction follows the domain, from wich a value should be retrieved. The specification of the domain is in the form described in section 21.2.4 on page 87 (without padding to a full byte).

Then follows the $Element$ parameter, for the choosen subdomain. First follows (each) an $Element$-startbit, which indicates if an $Element$ parameter follows. If it is 0 no $Element$ parameter follows, if it is 1 an $Element$ parameter follows. If it is 1 this first $Element$-startbit is followed by a 1 byte (8 bits) long natural number $Bits$, which indicates how many bits per $Element$ parameter are used. After this follows the first $Element$ parameter. After each $Element$ parameters (stored with the domain $naturalNumberB(Bits)$) follows again an $Element$-startbit and after it maybe the next $Element$ parameter and so forth. Example $matrix$.3.1: 2 Bits are needed to store the $Element$ parametes; the bits are (first bit on the front): 1111 0000 0000 0000 0 00 11011 1 00000010 11 1 10 0 (in ther order the fields are for: Fib element to retrieve domain; 8 bit element name; no parameter; matrixElement; $Element$ parameter follows; with each 2 bit; 3'th subdomain; $Element$ parameter follows; first subdomain; no $Element$ parameter follows)

After the bits for the element follows the $Mode$, this means what property value of the domain is selected. This is an 8-bit integer, as described in table 4 on page 29 in the column "value".

At the end of the Fib element to retrieve domain properties follows its subobject.

21.3.12 Set-element

Introduction: 0000 0000 0001 1111

For the description of the set-element see section 9.12 on page 30 .

The domain for the set-element is “setElement” (see section 21.2.4 on page 87).

After the introduction first follows one bit, that indicates whether a particular domain is used. If this bit is 0 no domain number $DomainNr$ is given and the $DomainNr$ is set to 0 . Otherwise, if it is 1, a natural number follows for the used domain $DomainNr$. The number of bits for this number results from the largest number $max(i)$ for a domain for the set-element and is $\lceil \log_2(max(i)) \rceil$. The $DomainNr$ determines the domain of the element. It is the first “setElement” domain with a domain number equal or lower of the set (loaded) domain number $DomainNr$.

Then follow two natural numbers.

The first number indicates the number (n) of the variables and to set values per set. The bits for this number resulting from the first element / subdomain of the vector domain for the element.

The second number indicates the number (k) of the sets of values to be set. The bits for this number results from the second element / subdomain of the vector domain for the element.

Following the two count values are k sets respectively vectors (see section 21.3.1 on page 102) with each n values (respectively it follow $k * n$ values / vector elements in a row). The domain for the vectors respectively sets is the third element / subdomain of the vector domain for the element.

At the end of the set-element follows its subobject.

21.3.13 Matrix element

Introduction: 0000 0000 0010 1111

For the description of the matrix element see section 9.13 on page 31 .

The domain for the matrix element is “matrixElement” (see section 21.2.4 on page 87).

After the introduction first follows one bit, that indicates whether a particular domain is used. If this bit is 0 no domain number $DomainNr$ is given and the $DomainNr$ is set to 0 . Otherwise, if it is 1, a natural number follows for the used domain $DomainNr$. The number of bits for this number results from the largest number $max(i)$ for a domain for the matrix element and is $\lceil \log_2(max(i)) \rceil$. The $DomainNr$ determines the domain of the element. It is the first “matrixElement” domain with a domain number equal or lower of the set (loaded) domain number $DomainNr$.

Then follow three natural numbers.

The first number indicates the number (d) of the dimensions / dimension variables.

The second number indicates the number (i) of the to set values per set. The bits for the first two numbers resulting from the first element / subdomain of the vector domain for the element.

The third number indicates the number (k) of the sets of values to be set. The bits for this number resulting from the second element / subdomain of the vector domain for the element.

Following the three count values are d vectors (see section 21.3.1 on page 102) for the areas respectively start and end values for the respective dimensions. In this, the first vector stands for the first dimension variable, the second vector for the second dimension variable, etc. . The domain for the vectors / sets is the third element / subdomain of the vector domain for the element.

After the dimension vectors follow k vectors / sets (see section 21.3.1 on page 102) with each i values (respectively it follow $k * i$ values / vector elements in a row). The domain for the vectors / sets is the forth element / subdomain of the vector domain for the element.

At the end of the matrix element follows its subobject.

22 XML format

Each Fib element and each vector has its own XML element in the XML format. Simple values of Fib elements and vectors are stored as attributes.

In the following listing a simple example of a Fib object in XML format is specified.

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<fib_object
   xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://www.fib-development.org/"
   xsd:schemaLocation="http://www.fib-development.org/fib.xsd">

   <root>
      <multimedia_info fib_version="1" db_version="0"/>
      <optionalpart>
         <pair key="autor" value="Oesterholz"/>
      </optionalpart>
      <domains>
         <dim count="2">
            <dimension number="1" direction="horizontal"/>
            <dimension number="2" direction="vertical"/>
            <vector elements="2">
```

```
               <naturalNumberB bit="8">
               <integer min="10" max="123">
            </vector>
         </dim>
      </domains>
      <main_fib_object>
         <area defined_variable="1">
            <vector type="subarea">
               <value>3</value>
               <value>15</value>
            </vector>

            <property>
               <vector type="property.colorGrayscale">
                  <value>200</value>
               </vector>
               <point>
                  <vector type="position">
                     <value>4</value>
                     <variable>1</variable>
                  </vector>
               </point>
            </property>
         </area>
      </main_fib_object>
   </root>
</fib_object>
```

22.1 XML header

The header of the Fib XML format is structured the same in each case. It specifies the XML format. The top element of the Fib XML format is always an element named `fib_object`. This contains the attributes to specify the XML format and one root-object.

Layout:

```
<?xml version="1.0" encoding="UTF-8"?>
<fib_object
   xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://www.fib-development.org/"
   xsd:schemaLocation="http://www.fib-development.org/fib.xsd">

      ... <!-- root-element -->
```

```
</fib_object>
```

22.2 Root-element

For the description of the root-element see section 9.14 on page 33 .

The element for the root-element has the name `root`.

The root-element can contain the following elements:

1. multimedia information (see section 22.2.1 on page 116)
2. the optional part (see section 22.2.2 on page 117)
3. domains (see section 22.2.3 on page 117)
4. domains for values (see section 22.2.3 on page 117)
5. input variables (see section 22.2.4 on page 125)
6. main-Fib-object (see section 22.2.5 on page 126); this element is always present
7. sub-root-objects (see section 22.2.6 on page 126)
8. identifiers of used database objects (see section 22.2.7 on page 127)
9. maybe a checksum field (see section 22.2.8 on page 127), this is irrelevant for the Fib XML format and it only serves for the data storage

22.2.1 Multimedia information

The element for the multimedia information has the name `multimedia_info`.

The element for the multimedia information has two attributes one for the version of Fib and one for the version of the Fib database.

The attribute `fib_version` indicates the Fib version of the Fib object and the attribute `db_version` the database version. Both version numbers are natural numbers. They can be mapped to a human-readable form (e.g. “Fib V1.2.3”). In the multimedia information element only numbers are used, otherwise a certain form would have to be established, which can be changed afterwards only with difficulty. There may be a human readable form of the versions specified in the optional part.

An example for the multimedia information:

```
<multimedia_info fib_version="1" db_version="0"/>
```

22.2.2 Optional part

The element for the optional part has the name `optionalpart`.

It contains a list of elements for the entries in the optional part. These elements have the name `pair`. The `pair` elements have two attributes `key` and `value`. The attribute `key` contains the key and the attribute `value` contains the value of the entry.

An example for the optional part:

```
<optionalpart>
   <pair key="copyright" value="GNU GPL 3"/>
   <pair key="type" value="the Berlin wall"/>
</optionalpart>
```

22.2.3 Domains

The element for the domains element has the name `domains` and consists of a domain list.

The domain for values element has the name `valueDomains` and has the same structure as the domain element with the name `domains`. This element (`valueDomains`) has no effect when saving in XML format, but is for keeping the information.

The domain list identifies the domains of each element, if a value (e.g. a variable) is outside its domain, it is rounded to a value within the domain. Values outside these domains can thus not occur for the element.

The domains for values include also domains, but these domains only apply to actual values in elements and not for values of contained variables. The domains for values determine how many bits are needed for compressed storing (see section 21 on page 80) of an element that contains a value. The domains for values are useful when the values of an element do not cover the full possible range of the domain for the element. For example, if a subobject contains only points whose position vectors only contain variables and integer values between 0 and 10, the domain for values for position vectors can be set to “integerB” with 4 bits, even if the variables of the position vectors taking values greater 100 .

The reason for the introduction of central (in the root-elements) domains is, that on the one hand as little space for values should be used when compressed storing the Fib object, without drastically limit the assignment possibilities for the values and on other hand that it can be determined in advance if and how the multimedia object can be displayed (e.g. if and how it should be scaled or whether the display of all values is impossible). If for example, the domain for the dimension takes only integer values between 0 and 50 (e.g. the horizontal in an image), then 6 bits is enough to store the values for the dimension. For larger images simply more bits for the values of the dimension can be used.

Each of the two domain lists contains a number of domain entries. A domain entry is an XML element for the type of Fib element for which it applies. This XML element in turn contains the actual domain.

XML element names for Fib elements The table 15 lists the names of XML elements and the attributes of the XML elements for the various types respectively Fib elements.

name	description
dim	The domain is for position vectors respectively dimensions. The attribute is `count`, which indicates the number of dimensions Dim. This element contains one element for each dimension with the name `dimension`. A `dimension` element has two attributs `number` and `direction`. The attribute `number` is a natural number greater than 0, which indicates the number of the dimension. In the position vectors the `number`'th element is for the dimension. The attribute `direction` indicates the direction in which the dimension goes. This is a natural number, which indicates, in which direction the `number`'th dimension is mapped. The possible values are described in table 7 on page 46.
subfunction	domain for the elements of subfunctions
property	This is the domain of property elements with the given name. The attribute `name` contains the name of the property. Possible names are listed in table 2 on page 12.
inVar	This is the domain for the `number`'th input variable. The attribute is `number`, which indicates the number of the input variable. (The count of the input variables of a root-element starts at 1 .)
Names of XML elements for domains that are relevant for compressed storing	
area	This type is for domains for the area element (see section 9.6 on page 16). The corresponding domain is a vector domain with 2 elements / subdomains. The first element or the first subdomain is used for the number (n) of subareas / vectors, it is part of the domain of the natural numbers. The second element is the domain for the subareas (B_1), it is a vector domain with two elements or subdomains, each of which come from the domain of integers.
variable	Values that are needed to encode variables. The domain should include the natural numbers from 0 to the maximum number of variables defined in the Fib leafs in the main-Fib-object. The Fib tree-leaf in the main-Fib-object above which the most variables are defined respectively the branch with the most defined variables, thus determins the domain.

name	description
comments	Values that are needed to encode comments. The domain should include the natural numbers from 0 to the number of comments in the main-Fib-object.
externObject	This is the domain for external objects (see section 9.9 on page 25) in the main-Fib-object. The domain is a vector with 4 elements. The vector elements are in ther the order for the identifier, the number of input values, the number of subobjects and the number of output variables. All vector element domains, except for the identifier, are part of the natural numbers. The vector element domain for the identifier is part of the integers.
externObjectInput	This domain is for the input values for external objects (see section 9.9 on page 25) . The domain is an vector domain. The attribute `identifier` is an integer and determines the identifier of the external object, for which elements the domain is.
externSubobject	This domain type is for the input values for external subobjects (see section 9.10 on page 26). The domain is an vector domain. The attribute `number` is a natural number and determines the number of the external subobject, for which elements the domain is.
externSubobject	This domain type is for the number of input variables for external subobjects (see section 9.10 on page 26). The domain is always a subset of the natural numbers.
setElement	This type is for the domain for the set-element (see section 9.12 on page 30). The corresponding domain is a vector domain with 3 elements / subdomains. The first element or the first subdomain is used for the number (n) of variables and number of values to be set per set, it is part of the domain of the natural numbers. The second element or the second subdomain is used for the number (k) of the sets of values to be set. It is also part of the domain of the natural numbers. The third and final element is the domain for the vectors for the values to be set ($W_{i.g}$) and is a domain for vectors, which element- or subdomains are domains for numbers (scalar). Further as an attribut `domainNr` a natural number can be specified for the domain number $DomainNr$. If the attribute `domainNr` is missing, the domain number $DomainNr$ is 0 .

name	description
matrixElement	This type is for the domain for the matrix-element (see section 9.13 on page 31). The corresponding domain is a vector domain with 4 elements / subdomains. The first element or the first subdomain is used for the number (d) of dimension variables, the number i of value variables and number of values i to be set per set, it is part of the domain of the natural numbers. The second element or the second subdomain is used for the number (k) of the sets of values to be set. It is also part of the domain of the natural numbers. The third element is the domain for the areas respectively for the start and end values for the different dimension variables, it is a vector domain with two elements or subdomains, each of which come from the domain of integers.The fourth and final element is the domain for the vectors for the values to be set ($W_{a.b}$) and is a domain for vectors, which element- or subdomains are domains for numbers (scalar). Further as an attribut `domainNr` a natural number can be specified for the domain number $DomainNr$. If the attribute `domainNr` is missing, the domain number $DomainNr$ is 0 .

Table 15: Names of XML elements for domains

Possible domains Each domain is assigned its own XML element. Each XML element for a domain can have an attribute `scalingfactor` for the scaling factor. This may be omitted, if the scaling factor is 1, respectively the scaling factor is 1 if it is omitted.

The scaling factor `scalingfactor` is a floating point number, with which the values of the unscaled domain or the basic domain are multiplied to obtain the scaled domain.

Example: The domain is 2 bit natural numbers with the scaling factor of $F = 0.5$. Possible values for the element to the domain are: $\{\ 0 * 0.5 = 0;\ 1 * 0.5 = 0.5;\ 2 * 0.5 = 1;\ 3 * 0.5 = 1.5\ \}$. If in the element to the domain a number is set that would mean 3 unscaled, it means with the scaling 1.5. The XML Element for the domain: `<naturalNumberB scalingfactor="0.5" bit="2"/>`

The table 16 lists the names of XML elements and attributes of XML elements for the various domains.

name	description	parameter
naturalNumberB	The basic domain are the natural numbers.	The attribute `bit` is a natural number, which indicates the number of bits for values of the domain. The corresponding basic domain is $0 \dots (2^bit - 1)$.
integerB	The basic domain are the integer numbers.	The attribute `bit` is a natural number, which indicates the number of bits for values of the domain. The corresponding basic domain is $-(2^{bit-1}) \dots (2^{bit-1} - 1)$.
naturalNumber	The basic domain are the natural numbers.	The attribute `max` is a natural number, which is the largest natural number in the domain. The corresponding basic domain is $0 \dots max$.
integer	The basic domain are the integer numbers.	The two attributes `min` and `max` are two integers. The attribute `min` is the smallest number in the domain. The attribute `max` is the biggest number in the domain. The corresponding basic domain is $min \dots max$.
integerValues	The basic domain are the integer numbers.	For each value of the basic domain the `integerValues` element contains one `value` element, in which the value is given (for example `<value>17</value>`).
real	The basic domain are floating point numbers. A floating point number consists of two integer fields, one the first, for the exponent E and one, the second, for the mantissa M. The floating point number Z is then $Z = M * 2^E$.	This element contains two elements. The first element specifies the domain of the mantissa and the second for the exponent. The specification of the domains is as described in this table. Both domains must come from the domains of integers (ie. integer... or naturalNumber...). Example: `<real>` `<integer min="-100" max="100"/>` `<naturalNumberB scalingfactor="0.1" bit="8"/></real>` The mantissa is a natural number from -100 to 100 and the exponent is an eight-bit natural number scaled by 0.1 .

name	description	parameter
realValues	The basic domain are floating point numbers. A floating point number consists of two integer fields, one the first, for the exponent E and one, the second, for the mantissa M. The floating point number Z is then $Z = M * 2^E$.	This element contains several elements. The first element specifies the domain of the mantissa and the second for the exponent. The specification of the domains is as described in this table. Both domains must come from the domains of integers (ie. integer... or naturalNumber...). After the first two parameters the `value`-elements for the possible floating point numbers of the domain follow (e.g. `<value>17.14</value>`). Example: `<real> <integer min="-100" max="100"/> <naturalNumberB scalingfactor="0.1" bit="8"/> <value>7*2^0.5</value> <value>31*2^10</value> </real>` The mantissa is a natural number from -100 to 100 and the exponent is an eight-bit integer scaled by 0.1 . The values in the domain are $7 * 2^{0.5} \approx 9.9$ and $31 * 2^{10} \approx 31744$
vector	The basic domain are vectors.	The element has one attribute `elements`, which indicates the number of elements E in the vectors. The XML element `vector` also contains for each of the vector elements (this means `elements` piece) a XML element with the domain for the element of the vector. Where the first XML element contains the domain of the first vector element, the second of the second and so on. All the values of the vector domain will have the form $(D_1, \ldots, D_E)$, where D_i is a value of the i'th domain in the domains list.

name	description	parameter
vectorValues	The basic domain are vectors.	The element has one attribute `elements`, which indicates the number of elements E in the vectors. The XML element `vectorValues` also contains for each of the vector elements (this means `elements` piece) an XML element with the domain for the element of the vector. Where the first XML element contains the domain of the first vector element, the second of the second and so on. Then follows a list with the vectors, as described in section 22.3.1 on page 128 without the `type` attributes. The domains of the vectors are the given domains of the preceding list. Variables can also occur in the stored vectors. If a variable is an element in a the vector the identifier for the variable is omitted. Example: `<vectorValues elements="2"> <integer min="-100" max="100"/> <naturalNumberB scalingfactor="0.1" bit="8"/> <vector> <value>99</value> <value>2</value> </vector> <vector> <value>-17</value> <variable></variable> </vector> </vectorValues>` ; This domain dosn't change anything for the notation of a vector in a Fib element in the XML format. Vectors will be stored like with the `vector` domain.

name	description	parameter
vectorOpenEnd	The basic domain are vectors.	The element has one attribute `elements`, which indicates the minimum number of elements E in the vectors. The XML element `vectorOpenEnd` also contains for each of the vector elements (this means `elements` piece) an XML element with the domain for the element of the vector. Where the first XML element contains the domain of the first vector element, the second of the second and so on and the E'te domain is for E'th and the following elements. All the values of the vector domain will have the form $(D_1, \ldots, D_E, \ldots, D_E)$, where D_i is a value of the i'th domain in the domains list. This domain is for elements that may contain vectors of different size. The number of elements for a vector of this domain is determined by the element containing the vector.
domainReference	This is a reference to the (sub-)domain of an other element. The domain is the domain of the element with the given type element.	The element can contain the optional attribute `subdomain`. The attribute `subdomain` specifies the subdomain which should be used. In it the numbers (counting starts with 1) of the subdomains to use is stored. Subdomains of more levels can be given separated by a point betwean them. In which the number of a subdomain stands befor the number of the subdomains contained in it. The `domainReference` contains a XML element for the type element to which it refers to. This XML element is generated as described in table 15 on page 120 . Example `<domainReference subdomain="3.1"> <matrixElement/> </domainReference>`: The used domain is the first subdomain of the third subdoamin of the matrix element.
defaultDomain	The specified domain will be used only if for the corresponding element so far no other domain was given.	The element contains a Xml element of a domain as described in this table. The specified domain is only used, if for the corresponding element so far no other domain was given.

name	description	parameter

Table 16: Elements for domains

An example for the domains:

```
<domains>
   <dim count="2">
      <dimension number="1" direction="horizontal"/>
      <dimension number="2" direction="vertical"/>
      <vector elements="2">
         <naturalNumberB scalingfactor="0.1" bit="8"/>
         <integer scalingfactor="0.1" min="10" max="110"/>
      </vector>
   </dim>
   <property name="colorRGB">
      <vector elements="3">
         <naturalNumberB bit="8"/>
         <integer min="-10" max="246"/>
         <naturalNumberB bit="9"/>
      </vector>
   </property>
</domains>
<valueDomains>
   <property name="colorRGB">
      <vector elements="3">
         <naturalNumberB bit="4"/>
         <integer min="-10" max="10"/>
         <integerValues>
            <value>17</value>
            <value>-3</value>
            <value>12</value>
            <value>124</value>
         </integerValues>
      </vector>
   </property>
</valueDomains>
```

22.2.4 Input variables

The XML element for the input variables is named `input_variables`. It contains for each input variable an XML element named `variable`. The `variable`

element has two attributes `number` and `default`. The attribute `number` is the number of input variable. With the attribute `default` the default value of the input variable is set.

An example for the input variable element:

```
<input_variables>
   <variable number="1" default="3"/>
   <variable number="2" default="17.3"/>
</input_variables>
```

22.2.5 Main-Fib-object

At this point the data from the main-Fib-object stand in the form of its elements and their parameters, as described in the section 22.3 on page 127. It is packaged in a XML element named `main_fib_object`.

An example for the main-Fib-object:

```
<main_fib_object>
   ...<!-- main-Fib-object -->
</main_fib_object>
```

22.2.6 Sub-root-objects

The XML element for the sub-root-objects is named `sub_roots`. It contains for each sub-root-object an XML element named `sub_root`. The element `sub_root` has the attribute `identifier`, which specifies the identifier of the sub-root-object (it is an integer). The element `sub_root` also contains the corresponding root-object, as described in this section 22.2.

An example for the sub-root-objects:

```
<sub_roots>
   <sub_root identifier="1">
      <root>
          ...
      <\root>
   </sub_root>
   <sub_root identifier="3">
      <root>
          ...
      <\root>
   </sub_root>
</sub_roots>
```

22.2.7 Identifiers of used database objects

Here follows a list of all identifiers of Fib database objects, which are used in the main-Fib-object or also in sub-root-objects. The storing of this identifiers is optional. If identifiers are present, it can be tested from the outset, that all external Fib objects from the database, that are needed, exists or whether it is likely that display errors occur, since database objects are missing. Whether the identifier of a Fib database object is given in the root-element in which main-Fib-object the database object is needed or in a higher root-element, depends on several factors. For the changing of the Fib objects, it is advantageous, that the identifier of a Fib database object is given in a root-element, which is near the place of its use. For storage space reasons in the compressed storage form it may be useful to store the identifier of used database objects in as few as possible root-elements.

The name of the corresponding XML element for the database identifier is `database_identifier`. It contains, for each specified database identifier, an XML element named `identifier`. This XML element `identifier` in turn, contains an (negative) integer for the used database identifier.

An example for the the identifiers of used database objects:

```
<database_identifier>
   <identifier>-21</identifier>
   <identifier>-632</identifier>
</database_identifier>
```

22.2.8 Checksum field

With this element a checksum for the root-element in the compressed storage format can be provided. For the Fib XML format, these information is irrelevant.

The checksum element is a property vector for a checksums, as described in section 22.3.3 on page 129.

An example for the checksum element / vector:

```
<vector type="property.checksum">
   <value>1</value>
   <value>256</value>
   <value>64</value>
</vector>
```

22.3 Fib elements

Each Fib element has its own XML element. Their layout will be described below.

22.3.1 Vectors

Vectors consist of several elements, each element being either a value or a variable. The number of elements of the vector is either determined by the Fib element or an associated domain (see section 9.14.2 on page 35).

All vector XML elements have the name `vector`. The `vector` element has the attribute `type`, which indicates the type of vector (Example: `position` or `property.colorRGB`). The `vector` element contains further for each element of the vector an XML element. The XML element for vector elements, which are values, has the name `value`. However, the XML element for vector elements, which are variables, has the name `variable`. Both XML elements for vector elements have the optional attribute `number`, which is the number (the counting starts at 1) of the element in the vector. Furthermore, both contain the value of the vector element. Where for the variables the variable identifier / number is given, as it was defined in the definition of the variable.

An example for a vector:

```
<vector type="position">
   <value>17</value>
   <variable>3</variable>
</vector>
```

An example for a vector with the optional attribute `number`:

```
<vector type="position">
   <variable number="1">5</variable>
   <value number="2">3.9</value>
</vector>
```

22.3.2 Point

For the description of the point element see section 9.2 on page 6 .

A normal point element has the name `point`. A point without a position vector ($point()$) has also the name `point` but contains no position vector. If the point is for the entire background ($point(())$: point in which the position vector has no elements), the name of the XML element is `background`.

Normal points included a position vector (see section 22.3.1). The number of vector elements is determined by the number of dimensions (see section 22.2.3 on page 117) the number of possible values for each position vector element, is determined by the domain of the dimension (see section 22.2.3 on page 120).

An example for a normal point:

```
<point>
   <vector type="position">
```

```
      <variable>2</variable>
      <value>21</value>
   </vector>
</point>
```

An example for a point with no impact respectively a point with no position vector:

```
<point/>
```

An example for a point with an empty position vector for the entire background:

```
<background/>
```

22.3.3 Properties

For the description of the property element see section 9.3 on page 7 .

The name of the property element is `property`. It contains a property vector and a Fib object (its subobject).

The property vector has an attribute named `type` for the type of property. The type is always `"property."` followed by the name of the property, as listed in table 2 on page 12 .

An example for a property element:

```
<property>
   <vector type="property.colorGrayscale">
      <value>200</value>
   <vector>
   ...<!-- Fib subobject -->
</property>
```

For the properties there are two special properties, that have influence on the decoding of the Fib object in the compressed format. These properties are “checksum” (see section 21.3.3 on page 104) and “boundSize” (see section 21.3.3 on page 105) . For the XML format they have no significance, but serve only to hold the information that they exists. The vector for the property “boundSize” has in this case 0 vector elements.

22.3.4 List element

For the description of the list element see section 9.4 on page 14 .

The name of the list element is `list`. It contains a number of subobjects that are listed in their order. (The first element in the XML list element is for the first subobject of the list element, etc. .)

An example for a list element:

```
<list>
   <!-- 1. list subobject -->
   <property>
      <vector type="property.boundSize">
      </vector>
      ...<!-- Fib subobject -->
   </property>

   <!-- 2. list subobject -->
   <point>
      <vector type="position">
         <variable>1</variable>
         <variable>2</variable>
      </vector>
   </point>

   <!-- 3. list subobject -->
   <point>
      <vector type="position">
         <variable>52</variable>
         <value>36</value>
      </vector>
   </point>

</list>
```

22.3.5 Comment element

For the description of the comment element see section 9.5 on page 14 .

The comment element has the name `comment`. It has two attributes `key` and `value`. The attribute `key` is the key of the comment element and the attribute `value` the value. Instead of using the attribute `value` the value of the comment element can be stored in a XML element named `value` as the text. Furthermore, the comment element contains a Fib object (its subobject).

An example for a comment element:

```
<comment key="name" value="Mr X">
   ...<!-- Fib subobject -->
</comment>
```

An example for a comment element with a `value` XML element:

```
<comment key="description">
   <value>
```

```
      This is a long text
      line 2
   </value>
   ...<!-- Fib subobject -->
</comment>
```

22.3.6 Area element

For the description of the area element see section 9.6 on page 16 .

The name of the area XML element is `area`. The area XML element has an attribute `defined_variable`. The attribute `defined_variable` indicates, which variable is defined by the area element. It is a natural number for a variable, that is not used above the area element for another variable.

The area element also contains for each of its subareas a vector of the type `subarea`. Wherein the first subarea is the first XML element, then comes the second etc. .

After the subareas follows the XML element for the contained subobject.

An example for a area element:

```
<area defined_variable="1">
   <vector type="subarea">
      <value>4</value>
      <value>10</value>
   </vector>
   <vector type="subarea">
      <value>15</value>
      <value>19</value>
   </vector>
   ...<!-- Fib subobject -->
</area>
```

22.3.7 Function

For the description of the function element see section 9.7 on page 17 .

The name of the function XML element is `function`. The function XML element has an attribute `defined_variable`. The attribute `defined_variable` indicates, which variable is defined by the function element. It is a natural number for a variable, that is not used above the function element for another variable.

Furthermore, the function element contains a subfunction element (this is described in subsection 22.3.7), which is contained in a separate XML element named `subfunction`.

After the subfunction element follows the XML element for the contained subobject.

An example for a function element:

```
<function defined_variable="2">
   <subfunction>
      <add>
         <value>4</value>
         <div>
            <variable>1</variable>
            <value>3.14</value>
         </div>
      </add>
   </subfunction>
   ...<!-- Fib subobject -->
</function>
```

Subfunction
The XML elements for subfunctions for variables and values (see section 9.7.1 on page 18) have the names:

- `value`: value
- `variable`: variable

They contain a number each. Where the variable subfunction (`variable`) contains the value of the number / identifier of the variable, as it is defined above.

Furthermore, there are real subfunctions (see section 9.7.2 on page 18). These contain for each of ther subfunction a XML element.

Names of subfunctions with an arity of one are:

- `abs`: absolute value
- `sin`: sine
- `arcsin`: arc sine
- `log`: logarithm
- `round`: round
- `delay`: delay

Names of subfunctions with an arity of two are:

- `add`: addition
- `sub`: subtraction

- `mult`: multiplication
- `div`: division
- `mod`: modulo
- `exp`: exponent
- `min`: minimum
- `max`: maximum

In the `sub`, `div` and `exp` subfunctions the the sequence of the contained XML elements is important. In them the first contained XML element is the first contained subfunction of the subfunction.

Other subfunctions:

- `if`: if-function (before its two subfunctions, first the condition of the if-subfunction is stored, the order of the XML subelements is: condition, true case (subfunction), false case (subfunction); see section 22.3.8 on page 133)

22.3.8 if-element

For the description of the if-element see section 9.8 on page 24 .

The name of the XML if-element is `if`.

The element contains a condition element `condition` (this is described in subsection 22.3.8).

After the condition element follow two XML elements for the contained subobjects. The first XML element is evaluated when the condition is true, and the second otherwise (if the condition is false).

An example for an if-element:

```
<if>
   <condition>
      <and>
         <true/>
         <lo>
            <variable>1</variable>
            <value>3.14</value>
         </lo>
      </and>
   </condition>
   ...<!-- Fib subobject,
     if the condition is true -->
   ...<!-- Fib subobject,
      if the condition is false -->
</if>
```

Conditions Conditions contain either XML elements for 0 to 2 subconditions (respectively conditions) or XML elements for two subfunctions (see section 22.3.7 on page 132).

The possible conditions for XML elements are listed in table 17.

element name	description	contained XML elements
`false`	The condition is false.	non
`true`	The condition is true.	non
`not`	The condition is true if and only if the subcondition is false, otherwise it is false.	one subcondition
`or`	The condition is true if and only if at least one of its subconditions is true, otherwise it is false.	two subconditions
`and`	The condition is true if and only if both of its subconditions are true, otherwise it is false.	two subconditions
`xor`	The condition is true if and only if exactly one of its two subconditions is true, otherwise it is false.	two subconditions
`eqInt`	Comparison of two to integers rounded numbers on equality. (for rounding see section 9.6 on page 16)	two subfunctions, as described in section 21.3.7 on page 107
`lo`	Comparison, if the first number is less than the second.	two subfunctions, as described in section 21.3.7 on page 107
`gr`	Comparison, if the first number is greater than the second.	two subfunctions, as described in section 21.3.7 on page 107

Table 17: XML element names for conditions

22.3.9 External object element

For the description of the external object element see section 9.9 on page 25 .

The XML element for external object has the name `obj`. It has one attribute named `identifier`, which indicates the identifier of the external object to be used. Moreover an `obj` XML element contains an XML element for the input variables and an XML element for each Fib subobject that can be used by the external object.

The corresponding XML element for the input values is a vector (see section 22.3.1 on page 128) with the type `externObjectInput` (`<vector type=`

"externObjectInput">), it contains the input values.

A XML element for a subobject has the name subobject. The subobject XML element has the optional attribute number, which specifies the number (counting begins at 1) of the subobject. Furthermore the subobject XML element contains an XML element for the output variables of the external object and an XML element for the Fib subobject, which is a normal Fib object, as described in this section 22.3 .

The corresponding XML element for the output variable has the element named output_variables. It keeps for each output variable one XML element named variable. The variable XML element has an optional attribute number, which indicates the number (counting begins at 1) of the output variable. Furthermore the variable XML element contains a value for the number / identifier of the defined variable.

An example for a external object element:

```
<obj identifier="123">
   <vector type="externObjectInput">
      <value>5</value>
      <variable>1</variable>
      <variable>3</variable>
   </vector>
   <subobject>
      <output_variables>
         <variable>6</variable>
         <variable>7</variable>
         <variable>8</variable>
         <variable>9</variable>
      </output_variables>
      ...<!-- Fib subobject number 1 -->
   </subobject>
   <subobject>
      <output_variables/>
      ...<!-- Fib subobject number 2 -->
   </subobject>
   <subobject>
      <output_variables>
         <variable>6</variable>
         <variable>7</variable>
      </output_variables>
      ...<!-- Fib subobject number 3 -->
   </subobject>
</obj>
```

An example for a external object element with the optional attribute number:

```
<obj identifier="34">
   <vector type="externObjectInput">
      <variable number="1">2</variable>
      <value number="2">3.7</value>
      <variable number="3">5</variable>
   </vector>
   <subobject number="1">
      <output_variables>
         <variable number="1">6</variable>
         <variable number="2">7</variable>
         <variable number="3">8</variable>
      </output_variables>
      ...<!-- Fib subobject number 1 -->
   </subobject>
   <subobject number="2">
      <output_variables>
         <variable number="1">6</variable>
         <variable number="2">7</variable>
      </output_variables>
      ...<!-- Fib subobject number 2 -->
   </subobject>
</obj>
```

22.3.10 External subobject

For the description of the external subobject element see section 9.10 on page 26 .

The XML element for an external subobject has the name `subobject`. It has an attribute named `number`, which indicates the number of external subobject, which is to be used for it. Furthermore the `subobject` XML element contains an XML element for its output variables.

The corresponding XML element for the output values is a vector (see section 22.3.1 on page 128) with the type `externSubobject` (`<vector type= "externSubobject">`), it contains the output values.

An example for a external subobject element:

```
<subobject number="6">
   <vector type="externSubobject">
      <value>5</value>
      <variable>7</variable>
   </vector>
</subobject>
```

An example for a external subobject element with the optional attribute `number`:

```
<subobject number="2">
   <vector type="externSubobject">
      <variable number="1">2</variable>
      <value number="2">3.7</value>
      <variable number="3">5</variable>
   </vector>
</subobject>
```

22.3.11 Retrieve domain properties

For the description of the Fib element to retrieve domain properties see section 9.11 on page 27 .

The XML element for the Fib element to retrieve domain properties has the name `domainProperty`. The XML element has two till tree attributes. The first attribute has the name `defined_variable` and specifies which variable is defined by the Fib element to retrieve domain properties. It is a natural number for a variable, that is not used above the Fib element to retrieve domain properties for another variable.

If the domain is a vector domain and thus contains multiple subdomains, there is an attribute `subdomain`. The attribute `subdomain` specifies the subdomain which should be used. In it the numbers (counting starts with 1) of the subdomains to use is stored. Subdomains of more levels can be given separated by a point betwean them. In which the number of a subdomain stands befor the number of the subdomains contained in it.

The attribute `mode` specifies the type of the property, which should be returned. The value of this attribute comes from the “name” column of the table 4 on page 29 .

The Fib element to retrieve domain properties contains two XML elements.

The first is for the used type of the Fib element, for which the domain is to which the Fib element to retrieve domain properties refers to. It has the name `type` and contains the XML element for the type. This XML element is generated as described in table 15 on page 120 .

The Fib element to retrieve domain properties contains also the XML element for the contained subobject.

An example for a Fib element to retrieve domain properties:

```
<domainProperty defined_variable="3" subdomain="2"
      mode="unscaled max">
   <type>
      <dim count="2">
         <dimension number="1" direction="horizontal"/>
         <dimension number="2" direction="vertical"/>
      </dim>
```

```
   </type>
   ...<!-- Fib subobject -->
</subobject>
```

Example wher the used domain is the first subdomain of the third subdoamin of the matrix element:

```
<domainProperty defined_variable="1" subdomain="3.1"
      mode="unscaled min">
   <type>
      <matrixElement/>
   </type>
   ...<!-- Fib subobject -->
</subobject>
```

Example without subdomains:

```
<domainProperty defined_variable="14" mode="scaling">
   <type>
      <inVar number="4"/>
   </type>
   ...<!-- Fib subobject -->
</subobject>
```

22.3.12 Set-element

For the description of the set-element see section 9.12 on page 30 .

The name of the set-XML element is `set`. It has an optional attribute named `domainNr`, which specifies the number of the domain for the set-element. If the attribute `domainNr` is missing, the domain number is 0 .

The element contains first a XML element named `defined_variables`, wich contains for each variable, which the set-element defines, in their order, a `variable` element. These have the optinal attribute `number`, which is the number (counting begins at 1) of the element respectively variable defined in the set-element. Furthermore the `variable` element contains a natural number for the variable, that is not used above the set-element for another variable.

After the XML element for variable definitions `defined_variables` follows the XML element named `values`, wich contains the vectors with the values, to which the variables should be set. They are listed in their order in the set-element. The vectors all have the type `set` (`type="set"`).

At the end, after the XML element for values vectors `values`, the set-element contains the XML element for the contained the subobject.

An example for a set-element:

```
<set>
   <defined_variables>
      <variable>7</variable>
      <variable>8</variable>
      <variable>9</variable>
   </defined_variables>
   <values>
      <vector type="set">
         <variable>1</variable>
         <value>3</value>
         <value>26.14</value>
      </vector>
      <vector type="set">
         <value>33.4</value>
         <value>-47</value>
         <variable>4</variable>
      </vector>
   </values>

   ...<!-- Fib subobject -->
</set>
```

An example for a set-element with the optional attribute `domainNr` and `number`:

```
<set domainNr="5">
   <defined_variables>
      <variable number="1">2</variable>
      <variable number="2">3</variable>
      <variable number="3">4</variable>
      <variable number="4">5</variable>
   </defined_variables>
   <values>
      <vector type="set">
         <variable>1</variable>
         <variable>1</variable>
         <variable>1</variable>
         <variable>1</variable>
      </vector>
      <vector type="set">
         <value>8</value>
         <value>4</value>
         <variable>1</variable>
         <value>3</value>
      </vector>
   </values>
```

```
   ...<!-- Fib subobject -->
</set>
```

22.3.13 Matrix element

For the description of the matrix element see section 9.13 on page 31 .

The name of the XML element is `matrix`. It has the attribute `dimensions`, which indicates how many dimensions d the matrix element has. Furthermore it has an optional attribute named `domainNr`, which specifies the number of the domain for the matrix element. If the attribute `domainNr` is missing, the domain number is 0 .

The element contains first a XML element named `defined_variables`, wich contains for each variable, which the matrix element defines, in their order a `variable` element. These have the optional attribute `number`, which is the number (counting begins at 1) of the element respectively variable defined in the matrix element. Furthermore the `variable` element contains a natural number / identifier for the variable, that is not used above the matrix element for another variable.

After the XML element for variable definitions `defined_variables` follows the XML element named `areas`, wich contains the vectors with the areas for the dimension ($(Startvalue_k, \ldots, Endvalue_k)$ with $k = 1 \ldots d$), to which the dimension variables ($Variable_1, \ldots, Variable_d$) should be set to. They are listed in their order in the matrix element. The vectors all have the type `area` (`type="area"`).

After the XML element for the areas `areas` follows the XML element named `values`, wich contains the vectors with the values ($Value_{d+1}, \ldots, Value_{d+i}$), to which the variables should be set. They are listed in their order in the matrix element. The vectors all have the type `matrix` (`type="matrix"`).

At the end, after the XML element for the values vectors `values`, the matrix element contains the XML element for the contained subobject.

An example for a matrix element:

```
<matrix dimensions="2">
   <defined_variables>
      <variable>7</variable>
      <variable>8</variable>
      <variable>9</variable>
      <variable>10</variable>
      <variable>11</variable>
   </defined_variables>
   <areas>
      <vector type="area">
```

```
         <value>3</value>
         <value>4</value>
      </vector>
      <vector type="area">
         <variable>2</variable>
         <value>7</value>
      </vector>
   </areas>
   <values>
      <vector type="matrix">
         <variable>1</variable>
         <value>3</value>
         <value>26.14</value>
      </vector>
      <vector type="matrix">
         <value>33.4</value>
         <value>-47</value>
         <variable>4</variable>
      </vector>
   </values>
   ...
   ...<!-- Fib subobject -->
</matrix>
```

An example for a matrix element with the optional attribute `domainNr` and `number`:

```
<matrix dimensions="2" domainNr="7">
   <defined_variables>
      <variable number="1">8</variable>
      <variable number="2">9</variable>
      <variable number="3">10</variable>
      <variable number="4">11</variable>
   </defined_variables>
   <areas>
      <vector type="area">
         <variable>3</variable>
         <variable>2</variable>
      </vector>
      <vector type="area">
         <value>-2</value>
         <value>0</value>
      </vector>
   </areas>
   <values>
      <vector type="matrix">
```

```
         <value>3</value>
         <variable>1</variable>
      </vector>
      <vector type="matrix">
         <value>0.44</value>
         <value>-7</value>
      </vector>
      <vector type="matrix">
         <value>6</value>
         <value>5</value>
      </vector>
      <vector type="matrix">
         <value>-1</value>
         <value>-1</value>
      </vector>
   </values>
   ...
   ...<!-- Fib subobject -->
</matrix>
```

Part V
Project structur of the implementation

The project is organized into different modules. These modules are intended to be (as far as possible) separate entities, their relationship to each other is clearly defined.

Each module has its own namespace.
These modules are (the names of the namespaces are given in front, before the colon):

- “fib”: the Fib multimedia language (see section II on page 4 for the description)
- “enviroment”: the general genetic algorithm (see section III on page 71 for the description)
- “operator”: operators for the general genetic algorithm
- “enviroment.fib”: the genetic algorithm for Fib
- “fib.operator”: operators for the genetic algorithm for Fib
- “fib.algorithm”: algorithms for processing Fib objects (currently no designs available)
- “fib.converter”: converters to convert objects to and from the Fib multimedia language
- “fib.player”: A player for Fib multimedia objects (currently no designs available)

23 Dependencies of the modules

In figure 9 the dependencies of the modules is shown. The inheritance arrows between the “enviroment” and “enviroment.fib” as well as the “operation” and “fib.operator” modules are used, to represent that the corresponding Fib modules are specializations of the “enviroment” or the “operator” module.

The figure 10 shows the dependencies of the main modules again in a different graphically form.

The “enviroment.fib” module requires for its individuals the Fib objects, but only as a name (for Fib objects). The “enviroment.fib” module thus only needs one name for Fib objects and no knowledge of the functionality (methods) of the Fib objects. Therefore the “enviroment.fib” module is not dependent on the “fib” module.

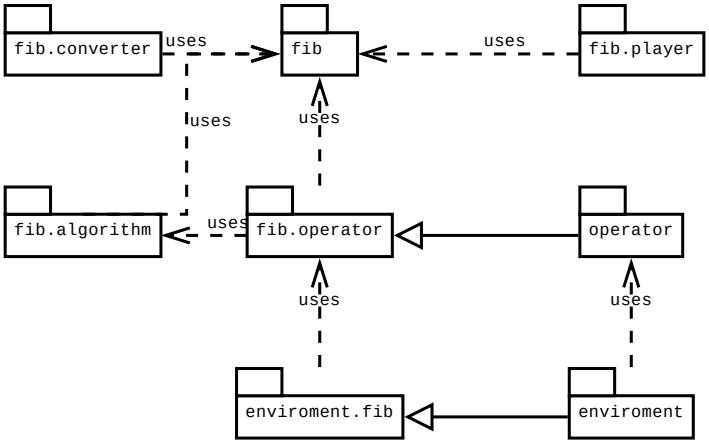

Figure 9: Dependencies of the modules

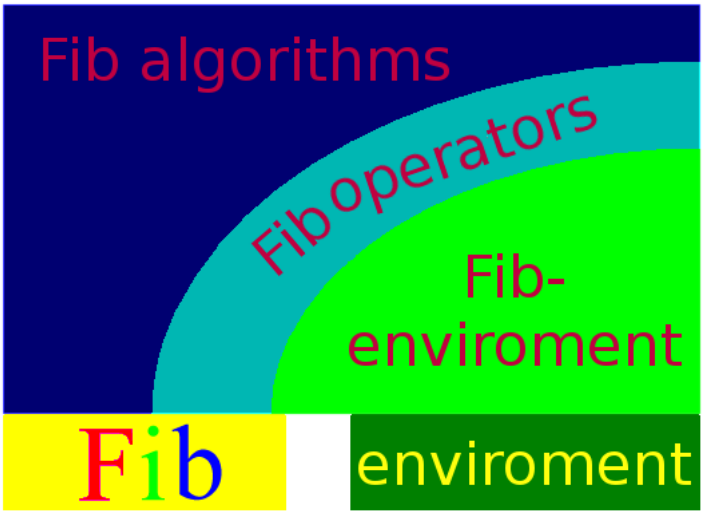

Figure 10: Dependencies of the main modules

Part VI
Appendix

24 GNU GENERAL PUBLIC LICENSE

In the following the english original text of the GPL (GNU GENERAL PUBLIC LICENSE) is listed.

This text was taken from the GNU website `www.gnu.org` and adapted for this document layout.

24.1 "GNU GENERAL PUBLIC LICENSE"

Date: Version 3, 29 June 2007

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

   The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

   A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

   The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

   The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

   The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

   All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

   You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

   Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

   No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

   When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

   You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

   You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

   (a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

   (b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

   (c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

   (d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

   A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

   You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

   (a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

(b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

(c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

(d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

(e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless

such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on

material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

(a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

(b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

(c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

(d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

(e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

(f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify

it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

   You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

    Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

    An “entity transaction” is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party’s predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

    You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a

license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work au-

thorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

    If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

    Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

    The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will

be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

    THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

    IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

    If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it does.>

Copyright (C) <textyear>  <name of author>

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.
```

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
<program>  Copyright (C) <year>  <name of author>

This program comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type 'show c' for details.
```

The hypothetical commands `show w` and `show c` should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see `http://www.gnu.org/licenses/`.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read `http://www.gnu.org/philosophy/why-`

25 GNU LESSER GENERAL PUBLIC LICENSE

In the following the english original text of the LGPL (GNU LESSER GENERAL PUBLIC LICENSE) is listed.

This text was taken from the GNU website `www.gnu.org` and adapted for this document layout.

25.1 "GNU LESSER GENERAL PUBLIC LICENSE"

Date: Version 3, 29 June 2007

This version of the GNU Lesser General Public License incorporates the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below.

0. Additional Definitions.

   As used herein, "this License" refers to version 3 of the GNU Lesser General Public License, and the "GNU GPL" refers to version 3 of the GNU General Public License.

   "The Library" refers to a covered work governed by this License, other than an Application or a Combined Work as defined below.

   An "Application" is any work that makes use of an interface provided by the Library, but which is not otherwise based on the Library. Defining a

subclass of a class defined by the Library is deemed a mode of using an interface provided by the Library.

A "Combined Work" is a work produced by combining or linking an Application with the Library. The particular version of the Library with which the Combined Work was made is also called the "Linked Version".

The "Minimal Corresponding Source" for a Combined Work means the Corresponding Source for the Combined Work, excluding any source code for portions of the Combined Work that, considered in isolation, are based on the Application, and not on the Linked Version.

The "Corresponding Application Code" for a Combined Work means the object code and/or source code for the Application, including any data and utility programs needed for reproducing the Combined Work from the Application, but excluding the System Libraries of the Combined Work.

1. Exception to Section 3 of the GNU GPL.

   You may convey a covered work under sections 3 and 4 of this License without being bound by section 3 of the GNU GPL.

2. Conveying Modified Versions.

   If you modify a copy of the Library, and, in your modifications, a facility refers to a function or data to be supplied by an Application that uses the facility (other than as an argument passed when the facility is invoked), then you may convey a copy of the modified version:

   (a) under this License, provided that you make a good faith effort to ensure that, in the event an Application does not supply the function or data, the facility still operates, and performs whatever part of its purpose remains meaningful, or

   (b) under the GNU GPL, with none of the additional permissions of this License applicable to that copy.

3. Object Code Incorporating Material from Library Header Files.

   The object code form of an Application may incorporate material from a header file that is part of the Library. You may convey such object code under terms of your choice, provided that, if the incorporated material is not limited to numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length), you do both of the following:

   (a) Give prominent notice with each copy of the object code that the Library is used in it and that the Library and its use are covered by this License.

(b) Accompany the object code with a copy of the GNU GPL and this license document.

4. Combined Works.

You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications, if you also do each of the following:

(a) Give prominent notice with each copy of the Combined Work that the Library is used in it and that the Library and its use are covered by this License.

(b) Accompany the Combined Work with a copy of the GNU GPL and this license document.

(c) For a Combined Work that displays copyright notices during execution, include the copyright notice for the Library among these notices, as well as a reference directing the user to the copies of the GNU GPL and this license document.

(d) Do one of the following:

. Convey the Minimal Corresponding Source under the terms of this License, and the Corresponding Application Code in a form suitable for, and under terms that permit, the user to recombine or relink the Application with a modified version of the Linked Version to produce a modified Combined Work, in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.

i. Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (a) uses at run time a copy of the Library already present on the user's computer system, and (b) will operate properly with a modified version of the Library that is interface-compatible with the Linked Version.

(e) Provide Installation Information, but only if you would otherwise be required to provide such information under section 6 of the GNU GPL, and only to the extent that such information is necessary to install and execute a modified version of the Combined Work produced by recombining or relinking the Application with a modified version of the Linked Version. (If you use option 4d0, the Installation Information must accompany the Minimal Corresponding Source and Corresponding Application Code. If you use option 4d1, you must provide the Installation Information in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.)

5. Combined Libraries.

You may place library facilities that are a work based on the Library side by side in a single library together with other library facilities that are not Applications and are not covered by this License, and convey such a combined library under terms of your choice, if you do both of the following:

(a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities, conveyed under the terms of this License.

(b) Give prominent notice with the combined library that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

6. Revised Versions of the GNU Lesser General Public License.

   The Free Software Foundation may publish revised and/or new versions of the GNU Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Library as you received it specifies that a certain numbered version of the GNU Lesser General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that published version or of any later version published by the Free Software Foundation. If the Library as you received it does not specify a version number of the GNU Lesser General Public License, you may choose any version of the GNU Lesser General Public License ever published by the Free Software Foundation.

   If the Library as you received it specifies that a proxy can decide whether future versions of the GNU Lesser General Public License shall apply, that proxy's public statement of acceptance of any version is permanent authorization for you to choose that version for the Library.

26 GNU Free Documentation License

This section lists the english original text of the "'GNU Free Documentation License"'.

This text was taken from the GNU website `www.gnu.org` and adapted for this document layout.

26.1 "GNU Free Documentation License"

Version 1.2, November 2002

51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "‘free"’ in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "‘copyleft"’, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "‘Document"’, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "‘you"’. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "‘Invariant Sections"’ are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Docu-

ment is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "‘Cover Texts"’ are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "‘Transparent"’ copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "‘Transparent"’ is called "‘Opaque"’.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "‘Title Page"’ means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "‘Title Page"’ means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

A section "‘Entitled XYZ"’ means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "‘Acknowledgements"’, "‘Dedications"’, "‘Endorsements"’, or "‘History"’.) To "‘Preserve the Title"’ of such a section when you modify the Document means that it remains a section "‘Entitled XYZ"’ according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition.

Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "‘History"’, Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "‘History"’ in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These

may be placed in the "‘History"’ section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "‘Acknowledgements"’ or "‘Dedications"’, Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "‘Endorsements"’. Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "‘Endorsements"’ or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "‘Endorsements"’, provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that

you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "‘History"’ in the various original documents, forming one section Entitled "‘History"’; likewise combine any sections Entitled "‘Acknowledgements"’, and any sections Entitled "‘Dedications"’. You must delete all sections Entitled "‘Endorsements"’.

COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "‘aggregate"’ if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "‘Acknowledgements"’, "‘Dedications"’, or "‘History"’, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "’or any later version"’ applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (c)  YEAR  YOUR NAME.
```

```
Permission is granted to copy, distribute
and/or modify this document under the
terms of the GNU Free Documentation License,
Version 1.2 or any later version published
by the Free Software Foundation; with no
Invariant Sections, no Front-Cover Texts,
and no Back-Cover Texts. A copy of the
license is included in the section entitled
"`GNU Free Documentation License"'.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "‘with...Texts."’ line with this:

```
with the Invariant Sections being LIST
THEIR TITLES, with the Front-Cover
Texts being LIST, and with the
Back-Cover Texts being LIST.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

References

[1] *Genetic programming for feature discovery and image discrimination walter alden tackett*, `http://citeseer.ist.psu.edu/tackett93genetic.html`.

[2] *Genetische und evolutionÃd're algorithmen*, `http://www.bwl.uni-mannheim.de/Heinzl/de/downloads/ki-kapitel-4-6.pdf`, WS 2002/03.

[3] Heinz Bauer, *Wahrscheinlichkeitstheorie*, 4 ed., New York, 1991.

[4] Erhard Behrends, *ÃIJberall zufall: Eine einfÃijhrung in die wahrscheinlichkeitsrechnung*, Mannheim, 1994.

[5] W.D. Cannon, *The wisdom of the body*, W.W. Norton, New York, 1932.

[6] Charles Darwin, *Die entstehung der arten*, Nikol, 1963.

[7] David B. Fogel, *Evolutionary computation - the fossil record*, New York, 1998.

[8] ———, *Evolutionary computation: toward a new philosophy of machine intelligence - 2nd ed.*, Institute of Electrical and Electronics Engineers, 2000.

[9] GÃijnther GÃűrz, Claus-Rainer Rollinger, and Josef Schneeberger, *Handbuch der kÃijnstlichen intelligenz*, 4 ed., Oldenbourg Verlag MÃijnchen Wien, Oldenbourg Wissenschaftsverlag GmbH; Rosenheimer Str. 145; D-81671 MÃijnchen, 2003.

[10] JÃűrg HeitkÃűtter and David Beasley, *The hitch-hiker's guide to evolutionary computation*, `http://surf.de.uu.net/hhg2ec/`.

[11] John H. Holland, *Genetic programming*, 4 ed.

[12] Marc W. Kirscher and John C. Gerhart, *Die lÃűsung von darwins dilemma*, Rowohlt Taschenbuch Verlag, 2007.

[13] Steve McConnell, *Code complete*, 2 ed., Microsoft Press Deutschland, 2004.

[14] Zbigniew Michalewicz, *Genocop – optimization via genetic algorithms*, `http://www.cs.sunysb.edu/~algorith/implement/genocop/implement.shtml`.

[15] ———, *Genetic algorithms + data structures = evolution programs*, third revision ed., Springer, 1996.

[16] Kurt Nawrotzki, *Lehrbuch der stochastik: eine einfÃijhrung in die wahrscheinlichkeitstheorie und die mathematische statistik*, Harri Deutsche, 1994.

[17] A. Nischwitz, M. Fischer, and P. HaberÃd'cker, *Computergrafik und bildverarbeitung*, 2 ed., Friedr. Vieweg und Sohn Verlag; GWV Fachverlag GmbH, Wiesbaden 2007, 2007.

[18] Paula Pfiester, Nina Tretter, Tina KoppenhÃűfer, Matthias Ecker, and Projektbetreuerin Annett Steiner, *Projekt bildformate*, `http://www.scheib.info/downloads/Projekt_Bildformate.pdf`, 10.2001-2.2002.

[19] David Poole, Alan Mackworth, and Randy Goebel, *Computational intelligence: a logical approach*, Oxford University Press, 1998.

[20] Roger S. Pressman, *Software engineering - a practitioner's approach*, 6 ed., McGraw-Hill, 2005.

[21] Stuart Russell and Peter Norvig, *Artificial intelligence: A modern approach*, 2nd edition ed., Prentice-Hall, Englewood Cliffs, NJ, 2003.

[22] Herbert Schildt, *C++ die professionelle referenz*, 1 ed., Osborne/McGraw-Hill, 2004.

[23] Friedrich Schmid and Mark Trede, *Skript zur vorlesung: EinfÃijhrung in die stochastik der finanzmÃd'rkte: Stochastische prozesse, simulation und anwendungen*, `http://www.wiwi.uni-muenster.de/~05/hauptstudium/vorlesungen/stofin/stofin.pdf`, SS 2000.

[24] Andrew S. Tanenbaum and Maarten van Steen, *Verteilte systeme*, 1 ed., Pearson Education Deutschland GmbH, 2003.

[25] Astro Teller and Manuela Veloso, *Algorithm evolution for face recognition: What makes a picture difficult*, `http://www.cs.cmu.edu/~mmv/papers/tellerICEC95.pdf`.

[26] Astro Teller and Manuela M. Veloso, *A controlled experiment: Evolution for learning difficult image classification*, Proceedings of the Seventh Portuguese Conference on Artificial Intelligence, Springer Verlag, October 1995, pp. 165–176.

[27] A.M. Turing, *Computing machinery and intelligence*, Mind **59** (1950).

[28] Thomas Wieland, *C++ entwicklung mit linux*, dpunkt.verlag, 2001.

Index