Specification of CSF version 2 format and API

C.G. Wesseling

Department of Physical Geography, University of Utrecht

14 October, 1997

Specification of CSF version 2 format and API

C.G. Wesseling

Department of Physical Geography, University of Utrecht

14 October, 1997

1. Introduction

Purpose of the Cross System Format (CSF for short) is a standard binary endian neutral format for storing geographical data used by applications in the PCRaster package. Other ASCII formats, used in the PCRaster package, such as column files, lookup and score tables are not described here. For these formats refer to the PCRaster (version 2) manual. In this document, only the two dimensional raster format having square cells is described. A binary format for temporal animations of two dimensional raster is currently under development.

This document describes the meaning of the fields in CSF files and the Application Programming Interface (API) for handling CSF files. By providing an API we want to encourage the use of CSF files through the API instead getting at the bytes directly. This allows a layer of software (the API) that can deal and if neccessary fix things between different versions of the format.

It documents version 2 of the format and API and some compatibility issues for version 1. Some terms now have *user names*, as known to them reading the PCRaster manual, and *internal names*, for programmers, which are the same as in version 1. I want to rename most functions in the future to names used in the PCRaster manual (Sorry for the inconvience).

I start by explaining the structure and various fields of the format, although we want to encourage the use of the API. This gives you an idea on what is around in a CSF file and then I explain the API functions that you can use to manipulate that information.

2. Layout of a CSF file

A CSF file contains 3 parts: headers, raster data and optional attributes. There are 2 headers: a main header that contains information necessary for all sorts of geographic data and the raster header with fields specific to raster data. Size and location of the header fields are given in the appendix. The text in this section should be sufficient to use the format through the API described in the rest of this document.

2.1. Basic types and missing values

All field and function definitions are in one of the 8 basic csf types plus a character type:

char
: series of bytes or characters.

INT1, INT2 and INT4
: signed integers of 1, 2 or 4 bytes.

UINT1, UINT2 and UINT4
: unsigned integers of 1, 2 or 4 bytes.

REAL4, REAL8
: floating point (IEEE-754) 4 or 8 bytes.

All these types are defined by a typedef in the include file of the library.

The lowest value for signed integers and the highest value for unsigned integers are omitted from the valid range. These values are called missing values (MV) and have a special meaning in the format. They specify a non specified feature or a value of no interest. The two floating point types have a bit-pattern (all bits set to 1) that is a NAN as their missing value.

2.2. Main header

signature
: Internal field to recognize a CSF file.

version
: Version of CSF. Current version is 2. Version 1 files are still readable by this API but cannot be created by this API. Minor differences between version 1 and 2 are explained in this document.

gisFileId
: Identification field in project context.

projection
: Map projection type. Version 1 had 5 different projections. In version 2 we only discern between Y increasing (PT_YINCT2B=0) and decreasing (PT_YDECT2B=1) from top to bottom. Version 1 types are automatically converted.

attrTable
: Internal field to find optional attributes.

mapType
: This field is 1 if the file describes a raster.

byteOrder
: Detects binary file compatibility (big versus little endian). This field is used by `Mopen` to determine how the file must be read. **VAX, PDP11 and other middle endians are not yet supported**

2.3. Raster header

The fields nrRows, nrCols, cellSize, projection, xUL, yUL and angle describe the exact location

and area that is covered by the map. These fields are called *location attributes.* Figure 1 is an overview of the location attributes.

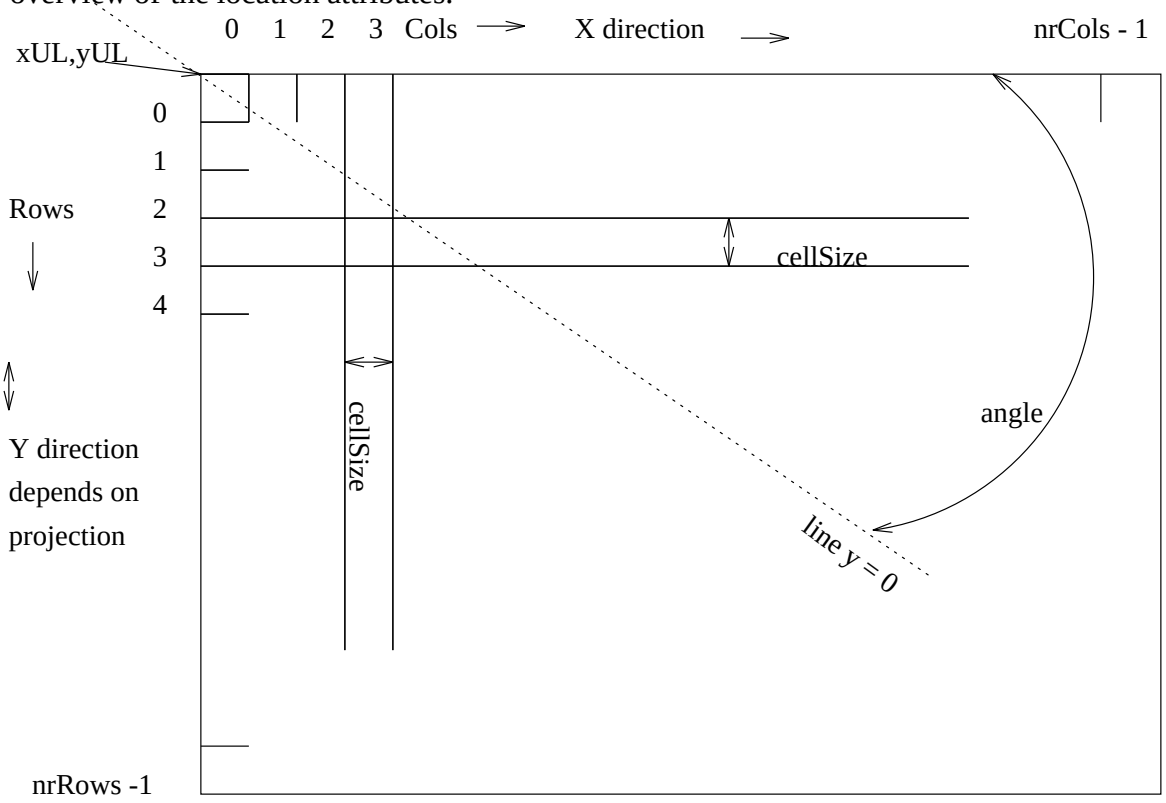

Fig.1: Scheme of a maps location attributes.

nrRows
: Number of rows.

nrCols
: Number of columns.

xUL, yUL (user name: x_{ul} y_{ul} ul=UpperLeft)
: Co-ordinate for the upper left corner of the map.

cellSize (user name: cell length)
: Size of cell in x and y direction. In version 1 different sizes could be given for the x and y direction. There are still two separate fields in the raster header, but they are always set to the same value. If the raster is an altitude map then these fields must be in the same units as the cell-values, for correct slope and orient calculations (for example, both in kilometers or meters but not in the cellSize in kilometers units and the dtm cell values in meters).

angle
: Counterclockwise rotation angle ($\in[-\frac{1}{2}\pi, \frac{1}{2}\pi]$) of grid with the line $y = 0$. This field is new in version 2. Version 1 files have an angle of 0.

cellRepr (user name: cell representation)

Specifies which of the 8 basic types is used to hold the raster data and the minVal and maxVal field. This API (version 2) only supports the four types listed below. The other basic types can still be read. There a no applications that need to create REAL8 maps at this moment. The REAL8 basic type is however important in the RuseAs function since it is the only type that can hold and convert back and forward the other three types with no loss of precision

minVal

Value smaller than or equal to the minimum value in raster data. See description of maxVal.

maxVal

Value greater than or equal to the maximum value in raster data. The minVal and maxVal fields are mainly used in programs who needs a possible range of values, e.g. for the scalar legend in display. The minVal and maxVal fields are of the same type as the raster data. In version 1 minVal and maxVal could be missing value. It is highly recommended to set minVal and maxVal **always** to value such that all values in the raster should fall between and including minVal and maxVal. If minVal and/or maxVal is missing value then an application can assume that is a *void map*: a map with only missing values.

valueScale (user name: data type)

The version 2 value scales are listed in the paragraph and explained in the PCRaster manual. Version 1 types where classified, continuous and not determined. Only version 2 types are accepted in the creation of maps through the API described here (version 2). Read functions , such as RgetValueScale will still return the old types.

cellrepr	**user name**
UINT1	small integer
INT4	large integer
REAL4	small real
REAL8	large real

2.4. Raster data

The raster data is a stream of cells in the basic type denoted by the cellRepr field of raster header of nrCols * nrRows cells containing values or missing values. Interpretation and ranges of the cell values depends on the value scale:

value scale	**cell values are:**
classified	MV, all integers
continuous	MV, all reals
boolean	MV, 0, 1
nominal	MV, all integers
ordinal	MV, all integers
scalar	MV, all reals
direction	MV, -1 for no direction, [0, 2π >
ldd	MV, integers [1-9]

2.5. Optional attributes

The attributes or fields in the headers are all obligatory. Optional attributes such as color pallettes and legends are appended to the end of the file.

Optional attributes are identified by an unique number. Every optional attribute can only occur once in the file. Attributes defined are (See appendix for details):

Description
: The description is a (short) description of the map. In most cases this is the name of the map. For example: Elevation of the Netherlands.

Legend without a name
: Version 1 legend. See legend with a name.

Legend with a name
: Version 2 legend. A legend has a name, for example soil types or land use, and an entry for each value that maps a cell value to a description.

History
: History describes how the map is created. We recommend to use this attribute frequently. If your application creates a map, then use the history attribute to identify that the map is created with your program. HINT: copy the entire command line (including the program name) of your program to this field

Colour Palette
: List of red, green and blue intensities.

Grey Palette
: List of white intensities.

3. How to install the library

The distribution contains four directories and one file in the top directory:

src
: All c-files (.c), csfimpl.h (local include for library compilation only) and makefile.tem a template makefile with the correct OBJS definition

include
: csf.h, csfattr.h and csftypes.h: interface files

man3
: All manual pages

html
: All manual pages in Html format. Use the index, 00csf.html or csfindex.html as starting point

htm
: All manual pages in Html format with DOS filenames. Use the index, 00csf.htm or csfindex.htm as starting point

csf.ps
: this document

Create a makefile in src:

The makefile must contain all c-files from this directory. and include on of the compilation directives: CPU_BIG_ENDIAN or CPU_LITTLE_ENDIAN.

For both using and creating the library the CPU_BIG_ENDIAN or CPU_LITTLE_ENDIAN preprocessor symbols must be defined (e.g. -DCPU_BIG_ENDIAN). However, some cpu's are probed in the MACHINE AND OPERATING SYSTEM section of csftypes.h . It is a good idea to read that section of csftypes.h first. The library is strict ANSI C, add the appropriate flags for your c-compiler ANSI C compilation.

Or without make, create a project file by selecting all c-files from this directory and select options to define one of the endian preprocessor symbols and set the compiler to ANSI C.

Then put the created library in the appropriate place on your system and make the manpages (include one directory up of man3 in your MANPATH environment variable) and make the include directory visible in your programming environment (-I).

4. Using this API in programs

The CSF library is a collection of constants, macro's and functions to access CSF files. Here we discuss the contents of the library in thematic order. All functions mentioned are documented in the online manual pages. Many manual pages also include some sample code. Note that in case of conflicting information between this document and the manual pages, you should rely on the manual page for being most up to date. There are two public include files to include in your application: `csf.h` . `csf.h` contains the function prototypes and includes `csftypes.h` . `csftypes.h` contains only typedefs and constants (e.g. `UINT1 MV_UINT1`).

All public functions are prefixed by either M or R. M-functions, for example `Mopen,` are for generic map handling, the R-functions, for example `RgetRow,` are specific for raster functions. In

addition there are functions, typedefs and constants prefixed by CSF. The CSF-prefixed symbols are only for internal use in the library and do not belong to the API.

4.1. Error handling

Functions available for error handling are:

Mperror
: Write short error message associated with current Merrno value to stderr.

MperrorExit
: Write short error message associated with current Merrno value to stderr and exits.

MstrError
: Returns error message associated with current Merrno value.

ResetMerrno
: Reset Merrno variable to NO_ERROR.

Many functions in the library set the global variable `Merrno` (analog to `errno` from the C library) to a value unequal to zero in case of an error. This value gives further information about the most recent error. At program initialization, Merrno is set to 0 (NO_ERROR). In most of the manual pages there is a section MERRNO that lists the possible constants set by a specific function in case of an error. There is no garanty that Merrno is always set in case every possible error, but many error conditions specific to using this API are checked. The library source code error checks are extensive in those areas where improper parameters could create a map that would make no sense to other applications. The constants are:

`NOERROR`
: No error occured.

`OPENFAILED`
: File could not be opened or does not exist

`NOT_CSF`
: File is not a C.S.F.-file

`BAD_VERSION`
: Wrong C.S.F.-version

`BAD_BYTEORDER`
: Wrong byte order. Should not occur

`NOCORE`
: Not enough memory to allocate run time structures.

`BAD_CELLREPR`
: Illegal cell representation constant

`NOACCESS`
: Access denied. This can happen if the file is set read only on the file system and you want to open it for writing or if you have opened the file read only and you want to write something.

`ROWNR2BIG`
: The specified row number exceeds the number of rows in the map.

`COLNR2BIG`
: The specified column number exceeds the number of columns in the map.

`NOT_RASTER`
: Map is not a raster file but another CSF type. The function you have used is specific to a raster file.

`BAD_CONVERSION`
: Illegal conversion. Only used in version 1. Version 2 cell representation can always be converted.

`NOSPACE`
: No space on device to write the file. The functions that create a map, allocate disk space enough space for the whole map.

`WRITE_ERROR`
: A write error occurred. Note that read and write errors can always occur, especially in network environments.

`ILLHANDLE`
: Illegal handle. You are passing a map handle that is not returned from `Mopen` , `Rcreate` or `Rdup` . Not checked by all functions, will be removed entirely someday.

`READ_ERROR`
: A read error occurred. Note that read and write errors can always occur, especially in network environments.

`BADACCESMODE`
: Illegal access mode constant.

`ATTRNOTFOUND`
: Optional attribute not found.

`ATTRDUPL`
: Optional attribute already in file.

`ILL_CELLSIZE`
: The specified cell size is smaller or equal to 0.

`CONFL_CELLREPR`
: Conflict between cell representation and value scale. See CW page 5 or the manual page of `Rcreate` .

`BAD_VALUESCALE`
: Illegal value scale.

`BAD_ANGLE`
Angle must be $\in[0, 2\pi >$.

`CANT_USE_AS_BOOLEAN`
Can't read as a boolean map. See manual page of `RuseAs` .

`CANT_USE_WRITE_BOOLEAN`
Can't write as a boolean map. See manual page of `RuseAs` .

`CANT_USE_WRITE_LDD`
Can't write as a ldd map. See manual page of `RuseAs` .

`CANT_USE_AS_LDD`
Can't use as a ldd map. See manual page of `RuseAs` .

`CANT_USE_WRITE_OLDCR`
Can't write to version 1 cell representation. See manual page of `RuseAs` .

`ILLEGAL_USE_TYPE`
The usetype argument of RuseAs is not version 2 cell representation, VS_LDD or VS_BOOLEAN.

4.2. Creating, opening and closing maps

A map can be accessed by a map handle which is anolog to the file pointer (`FILE *`) file in C. This map handle is returned by the functions:

`Mopen`
Open an existing CSF file.

`Mclose`
`Mclose` closes a map. This function is automatically called,, if the program is terminated with the `exit` function or at the end of the `main` function just as in fclose, if the map is not yet closed at that point.

`Rcreate`
Create a new CSF raster file.

`Rdup`
Create a new map by cloning another one.

Constants defined and used in `Rcreate` and `Rdup` arguments are (See also `RdefaultCellRepr`):

value scale	default cellRepr	possible cellRepr
VS_BOOLEAN	CR_UINT1	CR_UINT1
VS_NOMINAL	CR_INT4	CR_UINT1 CR_INT4
VS_ORDINAL	CR_INT4	CR_UINT1 CR_INT4
VS_SCALAR	CR_REAL4	CR_REAL4 (CR_REAL8)
VS_DIRECTION	CR_REAL4	CR_REAL4 (CR_REAL8)
VS_LDD	CR_UINT1	CR_UINT1

Projection type constants: `PT_YINCT2B PT_YDECT2B`.

4.3. Reading and writing cells

Most applications are designed to process raster data in one cell representation. For example, a drape program is programmed to keep the dtm-data in either REAL4 or REAL8 but not both. Or a program that reads some nominal maps to create a new nominal map is programmed to use the largest type (INT4), of the two cell representations valid for nominal maps (UINT1 or INT4), but write the output map as UINT1 if the input maps are UINT1.

After opening or creating a map and before reading or writing any cells one can use the `RuseAs` function to change the cell representation the application will use for a map. After a call to `RuseAs` we speak about the *in-file (or file) cell representation* and the *application (or app) cell representation.* The application cell representation is the one set by `RuseAs`. The file cell representation is returned by `RgetCellRepr` . If `RuseAs` is not used then the file and application cell representation are indentical. When calling a get (= read) function, the get function will read the cells and convert them to the application cell representation. When calling a put (= write) function it is the other way round, the put function will first convert the cells to the in-file cell representation and then write them to the file. `RuseAs` also affects `RputMinVal` , `RputmaxVal` , `RgetMinVal` and `RgetMaxVal` in a similar fashion. The REAL8 basic type is very useful in the RuseAs function since it is the only type that can hold and convert back and forward the other three types (UINT1, INT4, REAL4) with no loss of precision. Although, promotion from a REAL4 to REAL8 can add superficial precision. But converting back to REAL4 will get you the exact same number you started with (assuming the compiler and/or CPU conforms to a strict IEEE-754 implementation).

Conversion between any version 2 type (UINT1, INT4, REAL4 and REAL8) is possible. In read-only mode (e.g. `Mopen(..., M_READ)`) conversion from a type 1 (INT1,INT2,UINT2,UINT4) to a type 2 is possible. In addition to the cell representations constants (CR_UINT1, ..., CR_REAL8) one can pass the value scales VS_LDD and VS_BOOLEAN to `RuseAs`. The conversion routines then installed will convert a version 1 map to an ldd if possible (by taking *value modulo 10*) or any type (except direction and ldd) to a boolean (by testing *not equal to 0*). However the ldd is not repaired (See PCRaster manual).

IMPORTANT: Since the memory size of the cell representation types differ, the buffer for reading or writing cells must be big enough to hold the requested number of cells in both the in-file and the application cell representation type. `Rmalloc` allocates a buffer that is large enough to hold the both of them.

IMPORTANT: Due to endian handling (byte swapping) and a difference between in-file and application cell representation, the buffers of the Rput-functions can be invalidated on return. To be save, programmers should assume that these buffers contain crap after calling a Rput-function that do not have the const qualifier in their buffer argument specification. Or, in special cases, use the function `RputDoNotChangeValues` to check for this behaviour.

`RgetCell`
: Read one cell.

`RgetRow`
: Read one row.

`RgetSomeCells`
: Read a stream of cells.

`RputCell`
Write one cell.

`RputRow`
Write one row.

`RputSomeCells`
Write a stream of cells.

`RputAllMV`
Write missing value to all cells.

4.4. Handling missing values

For integer cell types there are the constants, MV_UINT1 and MV_INT4, that can be used as missing value. For floating point cell types there are no constants, because the missing values is not a valid floating point number. The following macros are defined instead:

`IS_MV_REAL4(x) IS_MV_REAL8(x)`
Test if x is pointer to that type of missing value.

`SET_MV_REAL4(x) SET_MV_REAL8(x)`
Set space pointed to by pointer x to that type of missing value.

`COPY_REAL4(dest,src) COPY_REAL8(dest,src)`
Copy one cell pointed to by src to cell pointed to by dest. This macro may be necessary since an assignment like `dest = src` may not work because `src` can be a NAN (MV).

All the functions, manipulating missing values work on memory locations, not on file or raster locations:

`IsMV IsMVcellRepr`
Test if a value is missing value.

`SetMV SetMVcellRepr`
Set a memory location to a missing value.

`SetMemMV`
Set an array of cells to missing value.

4.5. Setting the minimum and maximum in the header

By default the header minimum and maximum (which can be different from the true minimum and maximum) are set to the smallest and largest value written by the put-functions. The resulting maximum and minimum values can be wrong if the application writes some cells more than once. To correct this, keep track of the minimum and maximum value yourself and use the functions `RputMinVal` and `RputMaxVal` after all the updates of the map.

`RdontTrackMinMax`
Disable automatic tracking of minimum and maximum value.

`RgetMaxVal`
Get maximum cell value.

`RgetMinVal`
Get minimum cell value.

`RputMaxVal`
Set new maximum cell value.

`RputMinVal`
Set new minimum cell value.

Note that these functions are also affected by `RuseAs`. But they have a local buffer to hold the in-file cell representation.

4.6. value scale

The version 1 value scales (VS_CONTINUOUS, VS_CLASSIFIED) can be compatible with the version 2 scales: VS_BOOLEAN, VS_NOMINAL, VS_ORDINAL VS_SCALAR, VS_DIRECTION and VS_LDD. VS_UNDEFINED is a special value that is guaranteed not to have the value of another defined value scale constant. `RvalueScaleIs` tests if the value scale of a map is compatible with some version 2 scale. `RputValueScale` writes a new value scale. `RgetValueScale` returns the maps value scale, which can be a version 1 scale. `RvalueScale2` test if a value scale is a version 2 scale. `RstrValueScale` returns the user name of a value scale. `RgetValueScaleVersion` returns the version number of a value scale.

4.7. location attributes

The location attributes number of rows and columns are fixed after map creation, the other location attributes can be changed.

```
MgetProjection MputProjection

RgetAngle RputAngle

RgetCellSize RputCellSize

RgetXUL RgetYUL RputXUL RputYUL

RgetNrCols RgetNrRows
```

4.8. Other header attributes

`MgetDataType`
Data type of the map

`MgetGisFileId MputGisFileId`
Gis file id

`MgetVersion`
Get CSF version.

`RgetCellRepr`
Get (in-file) cell representation.

4.9. Co-ordinate conversion

In the library, raster cells are addressed by their row and column number where (row,col) = (0,0) while the raster is a representation of some geographic area defined by the location attributes. The following functions can convert between the raster and the (true world) geographic area:

`Rcoords2RowCol RgetRowCol`
Convert true world co-ordinate to a row, column index.

`RgetCoords RrowCol2Coords`
Convert row, column index to a true world co-ordinate.

4.10. miscellaneous functions, macros and constants

`MgetFileName`
File name associated with map.

`MnativeEndian`
Test if map is in the same endian format as the current program.

`Rcompare`
Compare 2 maps for their location attributes.

`LOG_CELLSIZE(t) CELLSIZE(t) CSFSIZEOF(n,t)`
Three handy macros. Argument `t` is one of the cell representation constant values (e.g. CR_UINT1). They return, the (^{2}log) size of type `t` in bytes and the size in bytes of an n-element array of type `t`. The all return a type of `size_t`

`VS_UNDEFINED, PT_UNDEFINED, CR_UNDEFINED`
Constants that are garantueed to have a value not equal to a defined value of that attribute class. They should **NEVER** be written to a map. Just handy for some programming constructs.

`UINT1_MIN UINT1_MAX INT4_MIN ... etc`
minimum and maximum values, note that REAL?_MIN are the smallest number larger than zero (just as FLT_MIN and DBL_MIN)

`RstrCellRepr RstrValueScale`
return the user name of these attributes

4.11. Using optional attributes

There are two generic functions for optional attributes: `MattributeAvail` and `MdelAttribute`. For each specific attribute there is a group of functions:

```
MgetNrLegendEntries MgetLegend MputLegend

MgetHistorySize MgetHistory MputHistory

MgetDescriptionSize MputDescription MgetDescription

MgetNrColourPaletteEntries MputColourPalette MgetColourPalette

MgetGreyPalette MgetNrGreyPaletteEntries MputGreyPalette
```

4.12. Obsolete and changed functions from version 1 to 2

This is a list functions that are superseded by other functions. The new functions are also given. In addition to this list, a number of function names are changed from R-functions to M-functions. For example MputLegend was RputLegend in version 1.

old function	**what happened to it?**
`Rcreate`	Arguments are changed
`Rdup`	Arguments are changed
`RreadAs`	replaced by `RuseAs`
`RputCellSizeX`	replaced by `RputCellSize`
`RputCellSizeY`	replaced by `RputCellSize`
`RgetCellSizeX`	replaced by `RgetCellSize`
`RgetCellSizeY`	replaced by `RgetCellSize`
`_Rput...`	replaced by Rput...
`_Rget...`	replaced by Rget...

Appendix A. Layout of the file

This appendix contains a precise definition of the file-format.

The position in bytes, from the start of the file, of the format parts are:

part name	position
main header	0
raster header	64
raster data	256
optional attributes	256+(nrRows*nrCols*sizeof(cell representation))

The gaps between the main and the raster header and the raster header and raster data were undefined in version 1, they contained garbage. From version 2 on they contain zero valued bytes.

Contents main header:

field name	type	position
signature	32 char	0
version	UINT2	32
gisFileId	UINT4	34
projection	UINT2	38
attrTable	UINT4	40
dataType	UINT2	44
byteOrder	UINT4	46

At creation there is a 1 written to the byteOrder field. If an *other endian* proccessor reads this field then the value will read 0x01000000 and knows that it has to swap the file contents.

Contents raster header:

field name	type	position
valueScale	UINT2	64
cellRepr	UINT2	66
minVal	8 char	68
maxVal	8 char	76
xUL	REAL8	84
yUL	REAL8	92
nrRows	UINT4	100
nrCols	UINT4	104
cellSizeX	REAL8	108
cellSizeY	REAL8	116
angle	REAL8	124

Note that minVal and maxVal can hold any type of value depending on the cell representation type. Angle is not defined in version 1. The API returns 0 as the angle for version 1 maps.

Appendix B. Optional attributes

The list of optional attributes is implemented by a chain of attribute control blocks. Each block contains 10 attribute descriptors and a file pointer to the next attribute control block. The file pointer to the first block is stored in the field attrTable of the main header. A file pointer of 0

means that there are no more attribute control blocks. An attribute descriptor has 3 fields:

UINT2 attrId
: Attribute identifier.

UINT4 attrOffset
: File pointer to attribute.

UINT4 attrSize
: Size of attribute in bytes.

The field attrId can have two special values. A value of 0 means that the space pointed to by attrOffset of size attrSize is no longer used. A value of 1023 (0xFFFF) means that the previous attribute descriptor was the last one (1023 is a design flaw, it should have been 0xFFFFFFFF, but we stick with 1023).

Note that two empty holes in the attribute area are never merged in the current implementation.

The attributes defined are defined as:

Description, id = 5
: C-string of variable length

Legend without a name, id = 1
: Array of an INT4 class value followed by a C-string in a 60-byte array

Legend with a name, id = 6
: Same as legend without a name, but the first entry contains the name in the string (at least an empty string) (See `RgetNrLegendEntries RgetLegend RputLegend`).

History, id = 2
: C-string of variable length

Colour Palette, id = 3
: Array of 3 UINT2 items defining the intensity of Red, Green and Blue

Grey Palette, id = 4
: Array of UINT2 defining the intensity of White (0 is black)

For the implementation of the Motif based display we once made the following implementation decisions concerning colour picking.

Classified maps: The first entry specifies colour of class value 0, the second entry for class value 1 and so on. Class values that cannot be used as an index in the palette (such as -1 or a value greater or equal the palette size should be displayed in another colour than the colours than are used from the palette. So if the palette contains 12 colours and the map contains the values 0,1,2,3 and 15 one can pick any palette entry larger than 4 for class 15 or choose another colour as long it is very distinct from palette entries 0,1,2 and 3.

Continuous maps: The entry for value X is computed as floor((X-minVal)/(maxVal-minVal)*P) where P are the number of entries in the palette. The palette must be sampled at regular intervals

if the palette size exceeds the number of colours that can be displayed simultaneously on the output device.