THE TUFTE-LATEX DEVELOPERS

THE SIMPLE ASTRONOMICAL DATA FORMAT

SADF STANDARD VERSION 2021.1

SADF.STRW.LEIDENUNIV.NL

First printing, December 2021

Contents

List of Figures

List of Tables

1 Introduction

1.1 Raison d'être of SADF

The Simple Astronomical Data Format (SADF) was designed to improve upon the commonly used FITS standard developed by NASA in the 1970's for use in astronomy. The FITS standard is rather clunky and does not support compression or data verification methods natively. Despite these disadvantages, FITS is still widely used within astronomy for it's versatility in allowing extensions of the standard. SADF aims to keep best practices from FITS and improving upon the less-than-ideal aspects of the FITS standard.

In addition, SADF is and always will be an open standard as per the definition of the free software foundation.

1.2 Overview

A SADF file consists of a header and data blocks. A SADF standard compliant file must always contain a header. The header essentially serves as a table of contents for the SADF file. Each data block present in the file is listed in the header as a Header Index Block (HIB).

SADF files contain no padding between the header/datablocks. In additon, both the header and data blocks may be of arbitrary length to allow flexible encoding formats. For this reason, the header contains pointers (read: byte indices) and lengths for all data blocks, whereas the header itself always starts at index 0.

Data blocks can have various types. In this standard, metadata blocks, plain text, up to 16-dimensional arrays and tables(maps) are supported. Users may also extend the standard and define their own data block types.

The metadata block contains a lot of critical information about a SADF file. Each data block may link to a metadata block that contains metadata relevant to that data block. Multiple data blocks may be associated with a single metadata block[1], but no more than one metadata block can be assigned to each data block.

[1] This feature can be used, for instance, to sign multiple data blocks with the same signature.

The SADF standard natively supports encryption, digital signing of data and compression algorithms. If a data block is encrypted, signed or compressed, the type of compression, type of encryption and signature are all stored in the metadata block associated with the data block in question. The metadata(1) part of the metadata block associated with a data block shall be appended to the data block before signing/encrypting the pair, but after compressing the data block itself, since data block compression must always take place before signing.

Data blocks (including metadata blocks) may appear in any order in a SADF file (although none may appear before the header). Therefore, data blocks that refer to other data blocks must do so using the data block identification number (DB_ID). The header provides an index linking these DB_IDs to actual positions in the file using the ptr data type.

Finally, an example is shown of a possible SADF file:

header	
SADF version	2021.1
Number of data blocks	6
(index of data blocks)	(...)
data block 1	
type	metadata
id	1
metadata	none
data block 2	
type	data
id	2
metadata	1
data block 3	
type	data
id	3
metadata	1
data block 4	
type	metadata
id	4
metadata	none
data block 4	
type	data
id	5
metadata	none
data block 6	
type	data
id	6
metadata	4

Table 1.1: Example structure of a SADF file with 6 data blocks *The SADF file shown here contains a header specifying positional and length information about the data bocks, followed by 6 datablocks. Data blocks 1 and 4 are metadata, data blocks 2,3,5 and 6 are data. In this example, data blocks 2 and 3 both have their metadata stored in data block 1. Data block 5 specifies no metadata, and data block 6 has its metadata stored in data block 4. Note that while the data blocks presented here are in order, this is not required by the standard.*

1.3 *Data types*

SADF encodes the data type of a given field using data type codes. The 2021 standard includes support for real numbers, complex numbers, pointers and character arrays.

In addition to the types specified in the standard, users may define data types for their own personal use. These types MUST use data type codes in the user-reserved range, as specified in Tbl.(1.2).

Table 1.2: List of supported data types in the SADF standard, together with their sizes and abbreviations. *The raw, character array and ALL user defined types are unsized. Hence, the length of fields using these data types MUST be specified explicitly. Users may define their own types in the 0xb000-0xbfff range. These type definitions are not supported guaranteed to exist in standard-conforming implementations and hence must be implemented by the user themself.*

data type code	abbreviation	size	description
Raw byte array			
`0x0000`	raw	unsized	unformatted data of unknown length
Logic			
`0x0001`	bool	1B	8-bit boolean (0=true, else=false)
Unsigned integers			
`0x0008`	u08	1B	unsigned 8-bit integer
`0x0016`	u16	2B	unsigned 16-bit integer
`0x0032`	u32	4B	unsigned 32-bit integer
`0x0064`	u64	8B	unsigned 64-bit integer
Signed integers			
`0x0010`	i16	2B	signed 16-bit integer
`0x0020`	i32	4B	signed 32-bit integer
`0x0040`	i64	8B	signed 64-bit integer
Floating point numbers			
`0x0f20`	f32	4B	signed 32-bit floating point number
`0x0f40`	f64	8B	signed 64-bit floating point number
Character arrays			
`0xca08`	utf8	unsized	UTF-8 encoded character array
`0xca16`	utf16	unsized	UTF-16 encoded character array
Pointer			
`0xa064`	ptr	8B	64-bit pointer
Complex numbers			
`0xc016`	xu16	4B	unsigned 16x2-bit complex integer
`0xc032`	xu32	8B	unsigned 32x2-bit complex integer
`0xc064`	xu64	16B	unsigned 64x2-bit complex integer
`0xc010`	xi16	4B	signed 16x2-bit complex integer
`0xc020`	xi32	8B	signed 32x2-bit complex integer
`0xc040`	xi64	16B	signed 64x2-bit complex integer
`0xcf20`	xf32	8B	signed 32x2-bit complex floating point number
`0xcf40`	xf64	16B	signed 64x2-bit complex floating point number
User defined types			
`0xb000 - 0xbfff`	(...)	unsized	user defined types

2
The SADF Header

2.1 The header

The header starts off with a u16 integer stating the version number of the SADF standard that was used to encode the file, followed by a u16 integer listing the number of data blocks, and finally an unordered list of Header Index Blocks (HIB's) is appended. Note that each HIB is always 20 bytes long.

Table 2.1 maps version number values to their corresponding SADF versions.

hex	decimal	SADF version
`0x00d3`	211	2021.1

Table 2.1: SADF version codes *Note that all values of this field are reserved for future use, hence the user may not specify their own version number.*

A header can, theoretically, hold 65,535 entries. This means that the maximum header size is 1,310,702 bytes, or about 1.3MB. Hence, 1,310,702 bytes must always be avaliable for header decoding in programs implementing SADF reading functionality, unless the total file size is smaller than 1,310,702 bytes, in which case the entire file must be loaded in one buffer in order to guarantee the entire header is avaliable[1].

[1] This buffer need not be thrown away and may be re-used by the implementation to parse the remaining file contents.

Note: Since the length of the data blocks is specified in the header, the required buffer size to read the data blocks is always known and no memory needs to go to waste when copying the data blocks into memory, unlike in the header reading process listed above.

2.2 Header Index Blocks

A Header Index Block (HIB) points to a data block in the file. They are used exclusively in the header part of the file. An HIB is always 20 bytes long.

The HIB layout is specified in table 2.2.

bytes	data type	field
0,1	u16	data block identification number (DB_ID)
2-10	ptr	data block start (DB_ST)
10-18	u64	data block length (DB_SI)
18,19	u16	data block type identifier (DB_TY)

Table 2.2: Header Index Block (HIB) data layout.

The identification number (DB_ID) of a data block serves for internal refrences; i.e. when data blocks must refrence eachother. The Data Block Start (DB_ST) pointer points to the byte index at which the data block in question starts. The data block length (DB_SI) field specifies the length of the data blocks in bytes.

Finally, the data block type identifier (DB_TY) specifies the format that was used to store the data block. Table 2.3 defines standard values for DB_TY as well as the user-extendable range.

code	data type
`0x0000`	plain text
`0x0001`	1-dimensional array
`0x0002`	2-dimensional array (image)
`0x000n-0x000f`	n-dimensional array
`0x00f0`	table
`0xb000-0xbfff`	user-reserved range
`0xffff`	metadata

Table 2.3: Data Block Type Identifyer (DB_TY) codes

Table 2.3 refers to 'arrays' and 'tables'. In the SADF standard, arrays must consist of only one data type, as opposed to tables, which are esentially keyword-data maps in which all keywords must have the same type and all data must have the same type, but keywords and data may have different types. For more information on arrays, we refer the reader to section [YEET]. For more information on tables, we refer the reader to section [YEET2].

3
Data Blocks

3.1 *General specification*

Data Blocks hold the data of the SADF file. 'Data' refers to both metadata and normal data. All data block must start with the fields defined in table 3.1

bytes	data type	field
0,1	u16	data block type identifier (DB_TY)
2,3	u16	data block identification number (DB_ID)
4,5	u16	DB_ID of associated metadata block (MD_ID)
...	raw	data or further formatting

Table 3.1: Required fields for all data blocks.

If a data block is linked to certain metadata (for instance, an image may be associated with a metadata block containing the telescope, location, time, instrument etc...), then the data block identification number (DB_ID) of the metadata block will be specified in the MD_ID field.

If a data block has no metadata associated with it, the MD_ID field shall be set to 0x0000 [1].

[1] The MD_ID value of 0x0000 is reserved for this use only. A SADF file may never contain an actual metadata block with DB_ID 0x0000.

Data block encoding, encrypting and signing is handled by the metadata block associated with the data block. (see section 1.2). Further information can be found in the metadata data block.

3.2 *Metadata [`DB_TY:0xffff`]*

A metadata Data Block starts with the standard (mandatory) fields, followed by (mandatory) metadata extension, as specified in table 3.2. The following restrictions apply to these mandatory fields:

1. The DB_ID field of any data block, including metadata blocks, may never be set to `0x0000`. See Section 3.1 for details.

2. The MD_ID field of a metadata block may be set to 0x0000 or

to its own DB_ID to indicate that no metadata other metadata blocks are connected to this metadata block. The MD_ID field may never be set to the DB_ID of one of the data blocks pointing to the metadata block. Setting the MD_ID field to any other value indicates is currently not standard-compliant, but may be used in the future to implement metadata inheritance.

bytes	data type	field
0,1	u16	data block type identifier (DB_TY)
2,3	u16	data block identification number (DB_ID)
4,5	u16	DB_ID of associated metadata block (MD_ID)
6,7	u16	compression type
8,9	u16	encryption type
10	bool	signature present (MD_SGND)
...	...	...

Table 3.2: Required fields for all data blocks of the metadata type.

Depending on the value of the MD_SGND field, more required fields may[2] follow. We distinghuish the following cases:

1. MD_SGND = true. In this case, more fields must follow the fields defined in table 3.2. These fields are specified in table 3.3.

2. MD_SGND = false. In this case, no additonal fields are required.

[2] To clarify: if the signed field is set to true, the signature fields MUST be specified. If the signed field is set to false, then the signature fields MUST be omitted.

bytes	data type	field
11-15	u32	signature type (MD_SG_TY)
15,16	u16	signature length (MD_SG_SI)
...	raw	signature (MD_SG)

Table 3.3: Optional fields for data blocks of the metadata type.

Encryption and digital signatures are discussed in section [REE]. This section covers the use and meaning of the MD_SGND, MD_SG, MD_SG_TY and MD_SG_SI fields.

The value of the compression type field indicates the type of compression used for the data block(s) [3]. Users may use the user-reserved range of the compression type field to define their own compression algorithms. The values specified in table 3.4

[3] If multiple data blocks point to a single metadata block, all MUST be compressed in the same way, namely according to the compression type specified in the shared metadata block.

code	compression algorithm
General purpose algorithms	
`0x0000`	no compression
`0x0001`	run-length encoding
`0x0002`	huffman encoding
`0x0003`	arithmetic encoding
`0x0004`	asymmetric number system (LZFSE)
`0x0005`	Lempel-Ziv compression (LZ77)
`0x0006`	Lempel-Ziv compression (LZ78)
`0x0007`	Lempel-Ziv-Szymanski (LZSS)
`0x0008`	Lempel-Ziv-Welch (LZW)
`0x0009`	Lempel-Ziv-Markov chain algorithm (LZMA)
`0x000a`	deflate
Bitmap/image compression algorithms	
`0x0100`	AOMedia video 1 image file format (AVIF)
`0x0101`	Free lossless image format (FLIF)
`0x0102`	High efficiency image file format (HEIF0)
`0x0103`	JBIG2
`0x0104`	JPEG 2000
`0x0105`	Discrete cosine transformation (LDCT)
User reserved	
`0xb000-0xbfff`	user-reserved range

Table 3.4: Metadata compression type (compression) codes

Index