

\_TY:oxfff



THE TUFTE-LATEX DEVELOPERS

# THE SIMPLE ASTRONOMICAL DATA FORMAT

SADF STANDARD VERSION 2021.1

Copyright © 2021 The Tufte-LaTeX Developers

SADF.STRW.LEIDENUNIV.NL

Licensed under the Apache License, Version 2.0 (the “License”); you may not use this file except in compliance with the License. You may obtain a copy of the License at <http://www.apache.org/licenses/LICENSE-2.0>. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

*First printing, December 2021*

# Contents

1	<i>Introduction</i>	9
1.1	<i>Raison d'être of SADF</i>	9
1.2	<i>Overview</i>	9
1.3	<i>Data types</i>	12
2	<i>The SADF Header</i>	13
2.1	<i>The header</i>	13
2.2	<i>Header Index Blocks</i>	13
3	<i>Data Blocks</i>	15
3.1	<i>General specification</i>	15
3.2	<i>Metadata [DB_TY:0xffff]</i>	15
3.3	<i>Plain text [DB_TY:0xffff]</i>	18



## *List of Figures*





# *List of Tables*

1.1	Example structure of a SADF file with 6 data blocks	11
1.2	List of supported data types in the SADF standard, together with their sizes and abbreviations.	12
2.1	SADF version codes	13
2.2	Header Index Block (HIB) data layout.	14
2.3	Data Block Type Identifier (DB_TY) codes	14
3.1	Required fields for all data blocks.	15
3.2	Required fields for all data blocks of the metadata type.	16
3.3	Optional fields for data blocks of the metadata type.	16
3.4	Metadata compression type (compression) codes	17
3.5	Metadata Entry layout	17



# 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 its 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 addition, 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<sup>1</sup>, but no more than one metadata block can be assigned to each data block.

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:

<sup>1</sup> This feature can be used, for instance, to sign multiple data blocks with the same signature.

<b>header</b>		
SADF version	2021.1	
Number of data blocks	6	
(index of data blocks)	(...)	
<b>data block 1</b>		
type	metadata	
id	1	
metadata	none	
<b>data block 2</b>		
type	data	
id	2	
metadata	1	
<b>data block 3</b>		
type	data	
id	3	
metadata	1	
<b>data block 4</b>		
type	metadata	
id	4	
metadata	none	
<b>data block 5</b>		
type	data	
id	5	
metadata	none	
<b>data block 6</b>		
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 blocks, 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.(??).

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 themselves.*

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 available 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 available<sup>1</sup>.

*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.*

<sup>1</sup> This buffer need not be thrown away and may be re-used by the implementation to parse the remaining file contents.

### 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 references; i.e. when data blocks must reference each other. 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 Identifier (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 essentially 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** <sup>1</sup>.

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.

<sup>1</sup> 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**.

### 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.

- 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<sup>2</sup> follow. We distinguish the following cases:

- 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.
- MD\_SGND = false. In this case, no additional fields are required.

<sup>2</sup> 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)<sup>3</sup>. 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

After these required fields, an optional metadata table may follow, consisting of an unordered list of Metadata Entries (ME). The metadata table cannot be compressed, but it can be signed.

ME's are appended to the fields described above in arbitrary order, without padding. ME's do not contain identification numbers, other

<sup>3</sup> 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

than they keywords themselves. Hence, ME's may not reference each other, nor may external data blocks refer to ME's. For more information on ME's, consult section 3.2.1.

### 3.2.1 Metadata Entry

A Metadata Entry is essentially a keyword-value pair (inspired by the keyword-value pairs of the FITS standard). A metadata entry contains the fields defined in table 3.5. The following restrictions

bytes	data type	field
0	u08	keyword length (x)
1-x	utf8	keyword
x,x+1	u16	value data type
x+2-...	...	value

Table 3.5: Metadata Entry layout

apply to fields specified in table 3.5:

1. The value data type must have a valid data type code, as defined

in section [yeet]

2. for data types with no fixed length (unsized data types), the first two bytes of the value field MUST indicate the length of the value field. This is not required for data fields with a fixed data size. User-defined types are always unsized.
3. The maximum length of the value field is 65,535 bytes. For values that take up more space than this maximum size, the use of separate data blocks is required.

### 3.3 Plain text [*DB\_TY:0xffff*]