The DIDADIT Protocol (aka „R“)

(Abbrev.: Data Is Data And Data Is Transferable)

Version 0.9.1

1. Assumptions

- All binary numbers are little-endian (Intel style)
- All alphanumerical strings are ISO-8859-1 coded

2. Basic concept

Data is transported block by block. Each block consists of

- a block delimiter (EOB),
- the data and
- a CRC.

The whole block (including data) is stuffed by means of the SLIP/KISS pattern (see appendix D).

The CRC is mandatory and included in every block. Blocks with an incorrect CRC are discarded, as the reason for the CRC error cannot be determined.

The used CRC polynome is the same as used by the AutoBIN protocol.

```
<EOB><DATA><CRC><EOB>
```

3. Block types

The first two bytes of a block always specify the block type. The block type identifies the kind of following data.

```
<TYPE><DATA>
```

If a block requires an MD5 hash (noted in this spec with "**MD5: yes**"), it is included between the block type and the data.

```
<TYPE><MD5><DATA>
```

All block types except <DATA> require an answer block. No other non-DATA block may be sent until the awaited reaction has been received. If the reaction is not received for a certain timespan (implementation specific), the first block is repeated. After five retries the connection is aborted.

Certain block types contain an MD5 hash after the type tag. It is used to help a third, monitoring station identify the transferred file.

3.1 INFO

This block contains a list of TAG=VALUE pairs. Each pair is followed by a <CR>.

Currently defined tags:

Tag	Value type	Description
FILENAME	string	Name of the file being transferred.
PATH	string	Directory in which the file is written on the receiving machine.
SIZE	decimal	Size of the file in bytes.
MD5	hexadecimal ASCII	MD5 hash of the complete file.
TIME	decimal	TIME is decimal as epoch-time (seconds since 1970).
BLOCKSIZE	decimal	Size of a transfer block.
VERSION	string	DIDADIT version being used.
FEATURES	string	This field specifies DIDADIT extentions. The following strings are currently defined: CHAT

A TAG=VALUE-pair may be no longer than 512 bytes.

MD5: no
Reaction: START or ERR

3.2 START

Supercedes „#EOK#“ from the first draft.

Currently defined tags:

Tag	Value type	Description	Comment
OFFSET	decimal	as offset for a resume. If OFFSET=SIZE, the other station has to send a FIN-block immediately and then waits for a REQ-block.	

PARTMD5	hexadecimal	The file's MD5-hash from beginning to OFFSET. Used to check if we still are transferring the same file.	This command may only be sent if the receiver cannot determine whether the sent file is the same as the partially received one. In this case, the transmitting station *must* check whether its file hash up to the specified OFFSET is the same as the here transmitted MD5-hash. If they are the same, the transfer starts at the specified offset. Otherwise the transfer is started at offset 0.
BLOCKSIZE	decimal	Is set if the value announced in INIT is too high.	
VERSION	string	DIDADIT version being used.	
FEATURES	string	see INFO block	

A TAG=VALUE-pair may be no longer than 512 bytes.

MD5: no
Reaction: DATA or FIN

3.3 ERR

General negative response.

The block contains an error code followed by the reason in plain text, e.g. „100 Unexpected block type“. If the second digit in the error code is smaller than 5, it is a fatal error. If it is 5 or larger, then it is non-fatal.

The complete string may not be longer than 80 bytes.

A list of error codes is supplied in appendix B.

MD5: yes
Reaction: disconnect, end of protocol or specific reaction according to error code

3.4 DATA

A DATA block contains the following information:

- 4 bytes offset
- 2 bytes blocklength
- 0x0-0xFFFF bytes of data

```
<OFFSET><LEN><DATA>
```

The receiver should check whether the combination of offset and blocklength is valid. If not, it should send error 104.

MD5: yes
Reaction: not neccessary

3.5 REQ

A REQ block requests one or many parts of the file. This can be done at any time during the transfer, but usually it will be applied after the full filelength has been sent.

If the requested block is larger then the block size agreed upon, it has to split up into several DATA blocks.

- 4 bytes offset1
- 4 bytes length1
- 4 bytes offset2
- 4 bytes length2
- ...

A REQ block can contain more than one request. After a FIN block has reached the receiver, only one REQ block can be sent. If the sender has answered all requests, it transmits a FIN block again and waits for the answer. This procedure is repeated until the receiver has aquired the file completely.

MD5: yes
Reaction: DATA or ERR

3.6 FIN

After the file has been sent completely, the transmitter sends a FIN block. This block can optionally contain data in a not yet defined format. After reception of FIN, the receiver checks for missing parts and, if neccessary, requests them with REQ.

MD5: yes
Reaction: REQ or FIN-ACK

3.7 FIN-ACK

As soon as the client has received all data, it closes the transfer by submitting a FIN-ACK.

MD5: yes
Reaction: end of protocol

3.9 ECHO-REQUEST

Request echoing of the supplied data block. The response is sent in an ECHO-REPLY block.

MD5: no
Reaction: ECHO-REPLY

3.10 ECHO-REPLY

Send data received by ECHO-REQUEST back to the originating station.

MD5: no
Reaction: not neccessary

3.11 ABORT

Can be sent by either station at any time. The DIDADIT connection is disengaged. The aborting station will not accept any further DIDADIT blocks.

MD5: no
Reaction: dependant on implementation (terminal mode, disconnect)

3.12 CHAT

Used for chat between the station's operators. The text should be displayed immediately.

This block type may only be used if given in the INFO and START block.

MD5: no
Reaction: not neccesary

4. Starting a DIDADIT transfer

A DIDADIT transfer is initiated by the string

```
<cr>#DIDADIT#<cr>
```

directly followed by an INFO block containing file information.

The receiving station must either answer with

```
<cr>#OK#<cr>
```

followed by a START-block, or

```
<cr>#ABORT#<cr>
```

if the transfer shall be aborted. This can be sent manually, if the receiver is not capable of DIDADIT.

5. Implementation details

The following has been a recommendation in previous versions of this document but is mandatory now.

- Send data in original order. Simple implementations perhaps do not use an index file and, in case of an error, would produce too many requests for block retransmission.
- For the same reason the transmitter should quickly react upon requests from the receiving station, especially REQ blocks.

Appendix A: Compression

Compression is an optional protocol extension. It has been developed by Marco, DL8NJY and is implemented in WinSTOP.

If an implementation supports compression, it will place a supplementary line into its INFO block:

```
COMPRESS=Type1,Length1[,Type2,Length2]...
```

where TypeX is the compression type and LengthX is the compressed file size if using this compression. Currently supported types are LZH as implemented in FBB-forwarding and gzip (standard GNU zip). If the receiving station supports one of the offered compression types it adds

```
COMPRESS=Type
```

to its START block. Then the sender starts transmission of the compressed data. Please notice that the filesize is now the value given in the COMPRESS line. The value given by the SIZE line regards the uncompressed file.

Appendix B: Error codes

Protocol errors:

Code	Text equivalent	Error description
100	Unexpected block type	The received block does not fit in the current situation (fatal).
101	REQ block with data out of range	A REQ block with an offset below 0 or beyond filesize has occured.
102	MD5 validation error	A block contained a wrong MD5 hash with is different from the currently transferred file.
103	PARTMD5 format error	Wrong count of characters.
104	Invalid offset/blocksize	The given data are not in the current file (e.g. offset > filesize).
150	Unknown block type	The received block does not fit in the current situation (non-fatal).

Errors within START/INFO blocks:

200	Illegal line in INFO block
201	Illegal line in START block
202	INFO block without MD5 info
203	Empty file or missing size info in INFO block
204	INFO block without BLOCKSIZE info
205	INFO block without FILENAME info
206	Block corrupt
208	Compression type not supported

Implementation-dependent errors:

300	Could not create file

Appendix C: List of all block types

Number	Type
1	INFO
2	START
3	ERR
4	DATA
5	FIN
6	REQ
7	FIN-ACK
9	ECHO-REQUEST
10	ECHO-REPLY
11	ABORT
12	CHAT

Appendix D: Stuffing

Every block is stuffed using the following system:

Transmission (TX):

- FEND := FESC+TFEND
- FESC := FESC+TFESC

Reception (RX):

- FESC+TFEND := FEND
- FESC+TFESC := FESC

Constants:

Name	Code	Name	Code
FEND	0xC0	FESC	0xDB
TFEND	0xDC	TFESC	0xDD

On TX, a FEND- or FESC-byte will change to the two-byte code; on RX, the given combinations are switched back to the FEND- or FESC-byte. After stuffing is done, the stuffed data will *nowhere* contain the FEND-byte. This is our EOB-code.