SAINT2

<u>System Analysis Interface Tool 2</u>

Emulation User Guide

Version 2.5

May 27, 2010

Maintained by: SAINT2 Team
Delphi
www.saint2support.com

Table of Contents

Revision Log

Version	Revisions	Date
1.0	Initial Release of Document	3/6/09
2.0	Major Emulation System Overhaul	6/25/09
2.1	Update for RX function and status commands	8/15/09
2.2	Add file open, seek, read, getbytes, and close commands	9/10/09
2.3	Add file CRC command	2/12/10
2.4	Include note that op warning code 08 A1 01 will occur Correct max variable declaration table	4/22/10
2.5	Add array re-init statement	

1 Overview

The SAINT2's embedded emulation feature allows certain SAINT2 operations to be automated and executed with or without a connected host PC. This document describes how to create the emulation script and how to use the emulation feature.

2 Important Disclaimer

The SAINT2 emulation system is in development. Changes may be made to the syntax and function of these commands and backward compatibility may NOT be supported in future firmware releases.

3 Using the Emulator – the Basic Steps

The following steps describe the process that should be followed to use the SAINT2 emulation feature:

1. The user creates the emulation script file using a text editor such as Notepad.

2. Using the Emulator Script Compiler application, the user "compiles" the emulation script file into an emulation opcode file.

3. If the emulation should automatically execute after a SAINT2 reset, the opcode file must be named *emulator.txt*.

4. Copy the opcode file and any other files being accessed during emulation execution to the SD card root directory.

5. Reset the SAINT2.

6. If the opcode file on the SD card is named *emulator.txt* it will begin to execute.

7. If the opcode file is not named *emulator.txt*, send the defined command from the host PC to execute the opcode file.

8. Monitor the LEDs on the SAINT2 to determine if the emulation executed successfully.

9. Check the emulation response to the host PC to debug emulator errors.

To ensure that emulation works properly, the Emulator Script Compiler application must be used with compatible SAINT2 firmware. The Emulator Script Compiler's compatible firmware versions can be found in the Compiler's About Box. It is highly recommended that both the compiler application and SAINT2 firmware be kept up to date.

4 Emulation System Operation

- If the file, *emulator.txt*, resides on the root of the SD card, the SAINT2 will execute that file upon power up or reset.
- Specific emulation files can be commanded to execute from the host pc using a SAINT2 configuration message.
- Emulation specific messages can be communicated between the host PC and the emulation function using a SAINT2 configuration message (if host communications are enabled.)
- The SAINT2 provides feedback about the execution of the emulation using its LEDs
    - The PWR LED will flash at a 2Hz rate to indicate that the SAINT2 is operating.
    - LED3 (green) will be turned on solid when the emulation script has been executed without error.
    - LED4 (amber) on indicates that the SD card is currently being accessed.
    - LED5 (amber) on indicates that there has been an operational warning code set. See the Programmer's Reference for op warning code definitions. At this time – **YOU WILL HAVE OP WARNING CODE 08 A1 01**. You will not be able to operate without setting this code.
    - LED6 (red) will be turned on solid when the emulation script has encountered an execution error. If there is an execution error, an error code will be sent to the host PC. Also, the user may command LED6 to be turned on or off in order to indicate a condition in the emulation script.

Host / Emulation Interface Messages

SAINT2 header	Emulation Config ID	ID	Description
08	50	00	Start execution of emulation file
08	50	01	Cancel execution of emulation file – <mark>not yet implemented</mark>
08	50	02	Message sent from emulation to host
08	50	03	Message sent from host to emulation
08	50	04	Message indicating an error in emulation execution
08	50	05	Emulation Status Request
08	50	06	Request emulation variable value
08	51		Return CRC of file

4.1 Start Execution of Emulation File

This command starts the execution of the emulation file specified in the command. For now, use all capital letters in the command.

Host to SAINT2 request: 08 50 00 XX XX … XX

Data Byte	Description
XX XX	Emulation filename in ASCII (capital letters, 8.3 format)

SAINT2 response: 08 50 00 XX XX … XX

Data Byte	Description
XX XX	Emulation filename in ASCII (capital letters, 8.3 format)

4.2 Cancel Execution of Emulation File <mark>*– not yet implemented*</mark>

This command stops the execution of the emulation file.

Host to SAINT2 request: 08 50 01

SAINT2 response: 08 50 01

4.3 Emulation System to Host Message

Messages that are sent from the emulation script to the host PC should have the following format so that they do not interfere with other SAINT2 functions.

Emulation to Host message: 08 50 02 XX XX … XX (total message may be up to 63 bytes long)

4.4 Host to Emulation System Message

Messages that are sent from the host to the emulation script must have the following format.

Host to Emulation message: 08 50 03 XX XX … XX (total message may be up to 63 bytes long)

4.5 Error in Emulation Execution Message

This message indicates that there has been an error in the emulation execution.

SAINT2 to Host message: 08 50 04 XX XX YY YY ZZ

Data Byte	Description		
XX XX	Script Block in which error occurred (starting with 1)		
YY YY	Opcode line in the defined block on which error occurred (starting with 1 in opcode file.)		
ZZ	Error Value	Description	
	0x01	Undefined SD card error	
	0x02	No such file or directory – SD card	
	0x05	I/O Error – SD card	
	0x09	Bad file number – SD card	
	0x0D	Permission denied – SD card	
	0x11	File Exists – SD card	
	0x13	No such device – SD card	
	0x16	Invalid Argument – SD card	
	0x18	Too many files open – SD card	
	0x1C	No space left on device – SD card	
	0x1E	Read only file system (Sharing error) – SD card	
	0x21	Emulation script file is incomplete	

	0x22	Script file syntax error – line read with no legal characters	
	0x23	Script file syntax error – line exceeds maximum length	
	0x24	The number of opcodes in a block has exceeded the limit	
	0x25	Error in operand 1	
	0x26	Error in operand 2	
	0x27	Error in operand 3	
	0x28	Error in operand 4	
	0x29	Error in Opcode	
	0x2A	The number of files referenced in a block has exceeded the limit	
	0x2B	Error writing to transmit buffer – run time	
	0x2C	Message length exceeds SAINT2 limit	
	0x2D	Script is attempting to open too many files at one time	

4.6 Emulation Status Message

This message requests the current status of the emulation execution.

Host to SAINT2 request: 08 50 05

SAINT2 to Host Response: 08 50 05 XX XX YY YY VV ZZ

Data Byte	Description		
XX XX	Script Block in which error occurred (starting with 1)		
YY YY	Opcode line in the defined block on which error occurred (starting with 1 in opcode file.)		
VV	Status Value	Description	
	0x00	Emulation is not currently running	
	0x01	Emulation is running	
ZZ	Error Value	Description	
	0x01	Undefined SD card error	
	0x02	No such file or directory – SD card	
	0x05	I/O Error – SD card	
	0x09	Bad file number – SD card	
	0x0D	Permission denied – SD card	
	0x11	File Exists – SD card	

	0x13	No such device – SD card	
	0x16	Invalid Argument – SD card	
	0x18	Too many files open – SD card	
	0x1C	No space left on device – SD card	
	0x1E	Read only file system (Sharing error) – SD card	
	0x21	Emulation script file is incomplete	
	0x22	Script file syntax error – line read with no legal characters	
	0x23	Script file syntax error – line exceeds maximum length	
	0x24	The number of opcodes in a block has exceeded the limit	
	0x25	Error in operand 1	
	0x26	Error in operand 2	
	0x27	Error in operand 3	
	0x28	Error in operand 4	
	0x29	Error in Opcode	
	0x2A	The number of files referenced in a block has exceeded the limit	
	0x2B	Error writing to transmit buffer – run time	
	0x2C	Message length exceeds SAINT2 limit	
	0x2D	Trying to open too many files at one time	

4.7 Emulation Variable Value Request

This message can be used to request the value of any of the emulation variables that have been declared in the emulation script. This request may be sent during or after emulation execution.

Host to SAINT2 request: 08 50 06 XX YY [ZZ]

Emulation Variable Value Message Response: 08 50 06 XX YY [ZZ] VV VV VV VV …

Data Byte	Description		
XX	Variable Type	Description	
	0x00	uint8_t	
	0x01	int8_t	
	0x02	uint16_t	
	0x03	int16_t	

	0x04	uint32_t	
	0x05	int32_t	
	0x06	uint8_t array	
YY	Index of variable in order of declaration with respect to variable type (starting with 1)		
ZZ	optional: array element at which to begin reporting array data (beginning of array is at element 0)		
VV VV VV VV	variable or array bytes – all variables will be reported as a 4 byte value, up to 57 array bytes will be reported		

Example:

```
uint8_t  ubyte1 = 0x11                //send 08 50 06 00 01  to get value
uint8_t  ubyte2 = 0x22                //send 08 50 06 00 02  to get value
int8_t    byte1 = 0xF1                //send 08 50 06 01 01  to get value
uint8_t array1[7] = 0102030405        //send 08 50 06 06 01 00 to get value
uint32_t uword1 = 0xAABBCCDD          //send 08 50 06 04 01 to get value
```

send 08 50 06 00 01 → response 08 50 06 00 01 00 00 00 11
send 08 50 06 00 02 → response 08 50 06 00 02 00 00 00 22
send 08 50 06 01 01 → response 08 50 06 01 01 FF FF FF F1
send 08 50 06 06 01 00 → response 08 50 06 06 01 00 01 02 03 04 05 00 00
send 08 50 06 04 01 → response 08 50 06 04 01 AA BB CC DD

4.8 Return CRC of File

This command returns the CRC of a file. For now, use all capital letters in the command.

Host to SAINT2 request: 08 51 XX XX … XX

Data Byte	Description
XX XX	Emulation filename in ASCII (capital letters, 8.3 format)

SAINT2 response: 08 51 00 YY YY YY YY   or 08 51 EE ZZ

Data Byte	Description		
YY YY YY YY	32 Bit CRC of file		
ZZ	Error Value	Description	
	0x00	Emulation is executing or CRC calculation is executing	
	0x01	Undefined SD card error	
	0x02	No such file or directory – SD card	
	0x05	I/O Error – SD card	
	0x09	Bad file number – SD card	
	0x0D	Permission denied – SD card	
	0x11	File Exists – SD card	
	0x13	No such device – SD card	
	0x16	Invalid Argument – SD card	
	0x18	Too many files open – SD card	
	0x1C	No space left on device – SD card	
	0x1E	Read only file system (Sharing error) – SD card	
	0x20	Buffer is busy	

5   Script Requirements

- Script keywords, variables, commands, and arguments are case insensitive.
- Spaces and Tabs will be ignored
- Any referenced file must be placed on the SD root so no path is necessary.
- Any referenced file name must follow the 8.3 naming format.
- Numbers entered with a "0x" will be interpreted as hex values
- Comments are identified using a proceeding "//".
- A long script may be broken into multiple script "blocks". A single block is read into the memory of the SAINT2 firmware and executed. When the execution is complete, the next block will be read and executed. Since this is the case, looping and conditionals must be fully contained in a single block. There is no limit, other than the SD card size, that limits the number of blocks in a script.

6 Opcode File Requirements

- The opcode file name must follow the 8.3 naming format.
- The opcode file must be saved on the SD card root.

7 Emulator Script Compiler Commands

7.1 Comments

Comments are identified with a "//"

Example:

// create transmit message

7.2 Variable Declarations

uint8_t *varname*   to declare an unsigned 8 bit value   (0 to 255)
int8_t *varname*   to declare a signed 8 bit value   (-128 to 127)
uint16_t *varname*   to declare an unsigned 16 bit value   (0 to 65535)
int16_t *varname*   to declare a signed 16 bit value   (-32768 to 32767)
uint32_t *varname*   to declare an unsigned 32 bit value   (0 to 4294967295)
int32_t *varname*   to declare a signed 32 bit value   (-2147483648 to 2147483647)
uint8_t var*name*[*size*]   to declare an array of uint8_t

```
uint8_t varname = value               to declare an unsigned 8 bit value    ( 0 to 255)
int8_t varname= value                 to declare a signed 8 bit value       (-128 t0 127)
uint16_t varname = value              to declare an unsigned 16 bit value   (0 to 65535)
int16_t varname = value               to declare a signed 16 bit value       (-32768 to 32767)
uint32_t varname = value              to declare an unsigned 32 bit value   (0 to 4294967295)
int32_t varname = value               to declare a signed 32 bit value      (-2147483648 to 2147483647)
uint8_t varname[size] = arraystring   to declare an array of uint8_t
```

Variable Type	Maximum Declarations
uint8_t	20
int8_t	5
uint16_t	20
int16_t	5
uint32_t	20
int32_t	5
uint8_t arrays	20

Item	Requirements	Maximum Value
length of variable name (varname)	may include alpha-numerics and "_"; must begin with an alpha-numeric, case insensitive	40 characters
size	must be a constant	522 bytes
memory allocated for arrays		522 bytes
max characters of array initial value (arraystring)	string of hex byte values	1044 characters

Notes:
- Variables must be declared before they are used
- Variables should be considered global.
- Array memory allocation may be reset at the beginning of a new script block.
- Initial variable values may be assigned on the same line as the declaration.
- Arrays must be declared at the beginning of a script block.

Example:

```
uint8_t num1 = 40                      //num1 is assigned to 40
int8_t num2                            //num2 is not assigned a value
int8_t num3 = 0x10                     //num3 is assigned to 0x10 (16 decimal)
uint 8_t array1[7] = 0102030405        //array1 will contain the following values 0x01, 0x02, 0x03, 0x04, 0x05, 0x00, 0x00
```

7.3 Variable Assignments

varname = value	assign a value to a variable
arrayname[element] = value	assign a 1 byte value to an element in an array
varname = expression	assign the result of an expression to a variable
arrayname[element] = expression	assign the result of an expression to an element in an array
varname = other_varname	assign the value of a variable to another variable
varname = arrayname[element]	assign the value of an element in an array to a variable
arrayname[element] = varname	assign the value of a variable to an element in an array
arrayname[element] = other_arrayname[other_element]	assign the value of an element in an array to a another element in another array
uint8_t var*name*[original *size*] = *arraystring*	assign a new initialization string to a previously declared array – THIS CAN ONLY BE USED AT THE BEGINNING OF A BLOCK BEFORE ANY EXECUTABLE STATEMENTS

Item	Minimum Value	Maximum Value	Definition
element, other_element	0	*size* - 1 (declared array size)	may be a value, variable, or expression

Notes:
- Any variable assigned must have been previously declared at the beginning of the script block.
- Assignments greater than variable declaration size will be rolled over without throwing an error.
- One assignment is allowed per line.

- Assignments expect a variable on the left and an expression on the right of the equal sign.

Example:

uint8_t num1
int8_t num2

num1 = 255   // num1 is assigned 255   (normal)
num1 = 256   // num1 is assigned 0   (wrapped)
num2 = 128   // num2 is assigned -128   (wrapped)

7.4 Math Expressions

Math Expressions allow the value of a variable to be set based on math operations of a combination of values and variables.

Operators	Definition
+	addition
-	subtraction
&	bitwise AND
|	bitwise OR
^	bitwise XOR
~	1's complement
++	increment
--	decrement
>>	bit shift right
<<	bit shift left
*	multiplication
/	division
%	modulo

Notes:

- Multiple math operations on one line will be interpreted in the order they are typed. Standard order of operations are not currently supported.

Example:

uint8_t num1
int8_t num2

num1 = num2++
num2 = num1 – 6
num1--

7.5 Conditionals (If Statements)

Conditionals consist of two expressions joined by a conditional operator. Also, a conditional evaluation may be performed on an rx statement. Conditionals may be nested.

if(<expression1> <op> <expression2>)
        <statements to execute if condition = TRUE>
else
        <statements to execute if condition = FALSE>
endif

if(rx(…..))
        <statements to execute if condition = TRUE>
else
        <statements to execute if condition = FALSE>
endif

Conditional Operators:

Operators	Definition
==	equal
!=	not equal
>=	greater than or equal
>	greater than
<=	less than or equal
<	less than

Example:

uint8_t var

if(Rx(056008FF1003, 500))
        var = 1
endif

if(var == 1)
        var = 0
        Delay(500)
else
         Delay(1000)
endif

7.6 Loops (For Statements)

Loops allows for repeated execution of statements. Loops may be infinite. Loops may contain breaks. For statements may be nested. A *break* statement will cause the script to immediately exit the for loop.

for(initial statement; terminating condition; repeated statement)
        <statements to execute>
exitfor

The initial and repeated statements may be any lines that could normally be in any other part of the script [barring a couple things such as the BlockEnd function]. The terminating condition may be any normal condition that could otherwise be used in an if statement. The parameters work almost exactly like C-syntax.

1) The initial expression is executed before any other part of the loop.
2) The terminating condition is evaluated. If it is false, execution jumps to the end of the loop.
3) Assuming the condition is true, the statements within the block are executed.
4) The repeated statement is executed.
5) Repeat steps 2 – 4 until the terminating condition becomes false.

All three of the parameters are optional, but both of the semicolons must remain. Leaving out the terminating condition creates an infinite loop.

break statement: *break*

Example:
for (num1 = 0; num1 < 10; num1++)
        array1[num1] = num2
        num2 = 8 + num1
exitfor

for (num1 = 0; num1 <= 6; num++)
        if (array1[num1] == num2)
                break
        end if

exitfor

```
uint8_t loop = 0
for (Delay(500); loop < 25; loop = loop + 5)
        for(uint8_t loop2 = loop; ; loop2++)
                if(loop2 == 25)
                        break
                endif
        exitfor
exitfor
```

7.7 Block Function

The block function may be used to separate a long script into blocks of commands.

BlockEnd([*reset files*]*opt*, [*reset array*]*opt*)

Function Argument Requirements

Argument	Value	Definition
reset files	0 = don't reset file list 1 = reset file list	If the file list is reset files loaded during the previous blocks will not be maintained in the new block.
reset array	0 = don't reset array memory allocation 1 = reset array memory allocation	If the array memory allocation is reset, arrays used during previous blocks will not be available for use in the new block.

Notes:
- This function may not be used within a conditional or a loop.
- The compiler will throw an error if the number of sequential opcodes exceeds the number of allowed opcodes per block.

Example:
uint8_t num1

```
uint8_t num2
uint8_t array1[7]
uint8_num3


for (num1 = 0; num1 <= 6; num++)
        if (array1[num1] == num2)
                break
        end if
exitfor

blockend( 1, 0)          //reset files but not arrays

num3 = array1[4]
```

7.8 Execute a SAINT2 File

Execute SAINT2 config.txt or group files that have been saved on the SD card.

Config(*filename*)

This command executes the SAINT2 configuration file named *filename*. A SAINT2 configuration file is used to configure the SAINT2 hardware. The format of this file is described in the SAINT2 Programmer's Reference document. Note that the SAINT2 executes any group file reference by a configuration file without timing control or guarantee of synchronous execution. This command should only be used to configure the SAINT2 hardware. It should not be used to send messages onto a serial bus, send a SWCAN high voltage wakeup, or send any other function that requires synchronous execution or timing control.

Example:

Config(conf.txt)   // configures the SAINT2 with "conf.txt"

Group(*filename*)

This command executes the group file named *filename*. A SAINT2 group file contains a list of messages and an associated delay from the previous message. The delay must be a 4-digit decimal value in milliseconds from the previous message. The message must follow the SAINT2 Programmer's Reference format (i.e., the message must include the SAINT2 header byte). Group files may be used to send any message (within the allowable length) that may be sent from the host PC to the SAINT2.

Example*:*

Group(grp.txt) // executes group file "grp.txt"

Group File Example:
```
0000 54 01 8D AF     // set baud rate to 33k
0010 54 04 02        // set transceiver to SWCAN after 10ms
0010 54 02 02        // set transceiver to high voltage mode after 10ms
0010 50 01 00        // send high voltage wakeup after 10ms
0020 54 02 03        // set transceiver back to normal mode after 20ms
```

Function Argument Requirements

filename	up to 8 character name, up to 3 character extension

Notes:
- The compiler should throw an error if the file name doesn't meet the requirements.
- The compiler should throw an error if the maximum number of files loaded is exceeded.

7.9 Delay

Pauses the SAINT2 execution of the emulation script for time in ms.

Delay(*time*)

Function Argument Requirements

argument	allowable values
time	0 - to 4294967295ms

Example:

delay (2000)   //delay 2 seconds

7.10 Host Communication Setting

This command turns the SAINT2 communication with a host PC off or on.

OpMode(*mode*)

Function Argument Requirements

Argument	Value	Result
mode	0x00	all host communication off
mode	0x03	turn RS232 communication on / USB communication off
mode	0x0C	turn USB communication on / RS232 communication off
mode	0x0F	turn both RS232 and USB communication on

Example:

Opmode(0x00)   //turn all host communication off

7.11 Trigger Out Function

This command sets the output state on the SAINT2's trigger out pin.

Trigger(*active*)

Function Argument Requirements

Argument	Value	Result
active	0	TRIGOUT = 0V
	1	TRIGOUT = 5V

Example:

trigger(1)   //set trig out to 5V

7.12 LED Function

This command sets the status of LED 6 on the SAINT2.

SetLed(*active*)

Function Argument Requirements

Argument	Value	Result
active	0	LED = off
active	1	LED = on solid

Example:

setled(1)   //turn led 6 on

7.13 Transmit a Message on a Serial Bus

This command may be used to transmit a message onto a serial bus or to the host PC.

tx(*msgstring*)
tx(*arrayname*)

Function Argument Requirements

Argument	Requirement	Description
msgstring	2 to 128 characters	a string of bytes representing the serial message to transmit
arrayname	must be previously declared	array that contains the serial message to transmit

Notes:
- The first byte of a message array must indicate the length of the message including the SAINT2 header byte.
- When the message is entered as a *msgstring* the length byte will be calculated by the script compiler.
- The second byte of the message array will define what serial bus is being used to transmit the message by using the SAINT2 header byte.

Example:

uint8_t txmsg[12] = 11 50 01 22 11 22 33 44 55 66 77 88   //define a SAINT2 format CAN message

tx(txmsg)   //transmit the CAN message
tx(60112233445566778899AABBCC)   //transmit a class2 message

7.14 Receive a Message from a Serial Bus

This command may be used to command the emulation to wait a given time to receive a message from a serial bus.

rx(*cmpstring*, *wait*, [*rxarray*]*opt*)
rx(*cmparray*, *wait*, [*rxarray*], [*maskstring*]*opt*)
rx(*cmparray*, *wait*, [*rxarray*], [*maskarray*]*opt*)

Function Argument Requirements

Argument	Requirement	Description
cmpstring	2 to 128 characters	a string of bytes representing the serial message to compare to the received message
wait	0 - to 4294967295ms	time to wait for received message
rxarray	must be previously declared	array used to store the received message that matches the compare array or string
maskstring	2 to 128 characters	a string of bytes to use for masking the serial message to compare to the received message
maskarray	must be previously declared	an array used for masking the serial message to compare to the received message

Notes:
- The use of a preceding 0x to indicate hex is not allowed in the message strings. Characters are assumed to represent hex bytes.
- In *cmpstring* "X" is used for a don't care nibble.
- The first byte of a message array must indicate the length of the message including the SAINT2 header byte.
- The second byte of the message array will define what serial bus is being used to receive the message

Examples:

Example 1 :

uint8_t rxmsg[13]
if(rx(50 01 22 XX X1 0X 56, 2000, rxmsg))   //Xs may be used for don't care nibbles
        …
endif

Example 2 :
uint8_t cmpmsg[7] = 06500122002056   //don't cares are not allowed in messages expressed in an array
uint8_t rxmsg[13]

rx(cmpmsg, 2000, rxmsg, FFFFFF0029FF)   //use the mask string for don't cares

Example 3:
uint8_t cmpmsg[7] = 06500122002056   //define the compare array
uint8_t rxmsg[13]
uint8_t cmpmask[7] = 06FFFFFF0029FF   //define a mask array for don't cares

if (rx(cmpmsg, 2000, rxmsg, cmpmask))
          …
endif

7.15 Open a Binary File

Opens a Binary File on the SD card to read, write or append.

fileopen(*filename, type)*

Function Argument Requirements

filename	up to 8 character name, up to 3 character extension
max number of files referenced in the block	EMU_MAXFILES
max number of open files at one time	SDNUSERFILES - 2

Argument	Value	Description
type	0	open a binary file to read
type	1	open a binary file to write

type	2	open a binary file to create or append

7.16 Seek through a Binary File

Sets the file pointer at an offset from the specified file location.

fileseek(*filename, offset, whence)*

Function Argument Requirements

filename	up to 8 character name, up to 3 character extension
max number of files referenced in the block	EMU_MAXFILES

Argument	Value	Description
offset	0 to 4294967295 bytes	number of bytes of offset from the file location specified by *whence*
whence	0	offset from beginning of the file
whence	1	offset from current location of the file pointer
whence	2	offset back from the end of the file

7.17 Read Data from a BinaryFile

Read a specified number of data bytes from a binary data file starting at the current file pointer location.

fileread(*filename, arrayname, array_element, varname,*)

Function Argument Requirements

filename	up to 8 character name, up to 3 character extension
max number of files referenced in the block	EMU_MAXFILES

Argument	Acceptable Values	Description

arrayname	any previously declared array	the name of the array that will be used to store data read from the file
array_element	0 to *size* of arrayname	the element in the array in which to save the first data byte read from the file
varname	a variable containing a value from 1 to (*size* – *array_element*)	number of bytes to read from the file into the specified array

7.18 Get Number of Bytes Read or Written to a Binary File

Store the number of bytes read or written to a binary file in the previous read or write to a variable.

getbytecount(varn*ame)*

Function Argument Requirements

varname	previously declared variable

7.19 Close Binary File

Closes a file that has been opened on the SD card.

fileclose(*filename)*

Function Argument Requirements

filename	up to 8 character name, up to 3 character extension
max number of files referenced in the block	EMU_MAXFILES

8 Limits

Miscellaneous Limits

Description	Value
Number of uint8_t declarations allowed	20

Number of int8_ t declarations allowed	5
Number of uint16_ t declarations allowed	20
Number of int16_ t declarations allowed	5
Number of uint32_ t declarations allowed	20
Number of int32_ t declarations allowed	5
Number of uint8_t array declarations allowed	20
Number of bytes available for array storage	522
Maximum number of bytes in a single array	522
Maximum number of executable opcodes in a block	100
Maximum number of files that may be referenced in a block	7
Maximum number of files that may be opened at one time	4
Number of bytes in a group file	1000
Variable name maximum length	40
Maximum TX or RX message	63 bytes