Secure Source Backup

Script Assembler Manual

Users Manual

Version 2.0

September 3, 2014

This software is provided as-is.
There are no warranties, expressed or implied.

Copyright (C) 2014
All rights reserved

MarshallSoft Computing, Inc.
Post Office Box 4543
Huntsville AL 35815 USA

web : www.marshallsoft.com
Email: info@marshallsoft.com

MARSHALLSOFT is a registered (r) trademark of MarshallSoft Computing.

TABLE OF CONTENTS

1 Introduction

This manual discusses the **Script Assembler** (sa.exe), and is designed to be used by those who have experience with programming.

The **Script Assembler** reads a script text file and creates the equivalent binary script file which can subsequently be read and executed by **SSB**.

The advantage of using the script interpreter is that the script is programmable. That is, it can verify the existence of files & directories, verify the volume name and/or serial number of drives, check the month or day of the week, then branch accordingly.

1.1 Script Program Structure

A **SSB** script program consists of lines of text, each line consisting of an optional label, an instruction (opcode), and an operand. Lines beginning with '#' are comments. For example, the code segment

```
# set SOURCE directory
SOURCE c:\MySourceFiles
```

specifies that the backup will begin in directory c:\MySourceFiles

All text commands as discussed in the **SSB** manual are also script instructions. However, there are script instructions that can only be used in scripts, as for example the **GOTO** script instruction.

1.2 Example Script Program

The following script program tests if the directory C:\MySourceFiles exists or not. If it does, it displays the message "Source directory exists" and beeps once. Otherwise, it displays the message "Source directory does not exist" and beeps three times.

```
# does file "C:\MySourceFiles" exist?
  DIR?     C:\MySourceFiles
  IF_TRUE  good_to_go
  DISPLAY  "Source directory does not exist"
  BEEP     3
  STOP
good_to_go:
  DISPLAY  "Source directory exists"
  BEEP     1
```

1.3 Running the Script Compiler

The script compiler is run from the Windows command line. For example, to assemble the script program Verify.a into Verify.b, type:

```
sa Verify
```

Note that the extension ".a" is assumed.

2 Script Program Structure

The script instruction file is a plain ASCII text file. Each line may be no more than 256 characters. Lines beginning with the '#' character are comments.

Script files always have extension ".a" and are assembled by the script assembler **sa.exe** into a binary form (having extension ".b") that can be executed by ssb.exe.

Each line in a script program consists of an instruction (opcode) and an (optional) operand:

"**opcode operand".**

All instructions are described in Section 3 "Script Instructions".

2.1 Script Labels

Statement labels always end with a colon ':'. The operand of the instructions **IF_TRUE**, **IF_FALSE**, and **GOTO** is always a label (but without a termination colon).

2.2 Condition Code

Instructions that end with the question mark '?' always set the "condition code register". The condition code register maintains its value until changed by the execution of another instruction ending with '?'.

2.3 Branching

The instruction **IF_TRUE** branches if and only if the condition code register is set to "true". The instruction **IF_FALSE** branches if and only if the condition code register is set to "false". The instruction **GOTO** always branches.

2.4 Subroutines

The instruction **CALL** jumps to its operand label, as shown in the example in the previous section. The **RETURN** instruction jumps to the instruction immediately following the last **CALL** instruction. Subroutines may be called to a maximum stack depth of 64.

2.4 Predefined Strings

Predefined strings must start with '$', and can be defined as

```
  $STRING_NAME STRING_VALUE
```

For example, the following DISPLAY command would display "Good morning !"

```
  $HELLO "Good morning !"
  DISPLAY $HELLO
```

All strings must be defined <u>before</u> the code (program instructions).

2.6 Instruction Classes

There are 4 classes of script instructions.

(1) Class 0: No operand.
(2) Class 1: Operand is a byte constant (0 to 255).
(3) Class 2: Operand is (16-bit) code address (program label).
(4) Class 3: Operand is a (16-bit) string address.

The script instructions are described in the following section "Script Instructions".

2.7 Running the Script Compiler

The script compiler is run from the Windows command line. To assemble a script program file (which must have extension ".a"), type

```
   sa [flags] program-file
```

where flags (optional) are

```
  -u  {generate un-assembly}
  -l  {list statements}
  -ul {both of above}
```

```
Example 1: sa -ul backup
Example 2: sa restore
```

The first example above creates the file backup.b from backup.a, while the second example creates file restore.b from restore.a

Appendix A: Script Instructions

Command Instructions (* = script assembler only - cannot be used in a text command file).

Script instructions are listed after the following table according to their class (0 to 3).

	AUTH	Specifies the path name of the restore authorization file.
	BEEP	Makes a beep sound.
*	CALL	Calls a block of code (subroutine).
*	CANCEL?	Prompts the user to cancel execution or not.
	COMPRESS	Enables compression.
	COPY	Specifies a file to copy.
*	DAY?	Checks the day of the month (1 to 31).
	DEBUG	Sets the SSB debug level.
	DIR?	Tests for the existence of the specified directory.
	DISGUISE	Enables both file name & directory name disguising.
	DISPLAY	Displays a user specified message.
*	DOW?	Checks the day of the week (0 to 6).
*	DRIVE?	Tests for the existence of the specified drive.
	ENCRYPT	Enables encryption.
*	FILE?	Tests for the existence of the specified file.
*	GOTO	Transfers control to the specified program address (label).
	HIDDEN	Enables the processing of hidden files.
*	IF_FALSE	Transfers control if the condition code is "true".
*	IF_TRUE	Transfers control if the condition code is "false".
	IGNORE	Ignores (does not process) specified directory.
	LOG_FILE	Specifies the name of the SSB log file.
	MAP_FILE	Specifies the name of the SSB map file.
*	MONTH?	Checks the month of the year.
	MOUNT	Request mounting a specified drive.
*	NOP	Performs no function.
	PASS	Specifies the name of the SSB password phrase file.
	READONLY	Enables processing of "read only" files.
	RESET	Resets the script program (other than USER command).
*	RETURN	Jumps to the instruction immediately following the last CALL.
	RUN	Starts a BACKUP, RESTORE, or LIST operation.
	SKIP	Specifies a file to skip (not copy).
	SLEEP_MIN	Sleeps the specified number of minutes.
	SLEEP_SEC	Sleeps specified number of seconds.
	SET_DDC	Sets the "directory disguise character".
	SOURCE	Specifies the name of the SSB source directory.
	STOP	Terminates execution.
	SUB_DIRS	Enables the processing of sub-directories.
	TARGET	Specifies the name of the SSB target directory.
	TIMESTAMP	Sets the max time between files to be considered the same.
	USER	Specifies the name of the SSB user ID file.
	VERBOSE	Enables verbose output in the log file.
*	VOL_NAME?	Tests for the specified volume name.
*	VOL_SERIAL?	Tests for the specified volume serial number.
	WRITE	Writes text to the SSB log file.

A.0 Class 0 Instructions (Opcodes)

There is no operand for class 0 instructions.

A.0.1 CANCEL?

The **CANCEL?** instruction prompts the user to cancel execution or not. The condition code is set to true if the user clicks the "Cancel" button, else the condition code is set to false.

A.0.2 NOP

The **NOP** instruction performs no function.

A.0.3 RESET

The **RESET** instruction resets the script program as if the only instruction executed thus far is the **USER** instruction. That is, all previous instruction are nullified, with the exception of the **USER** instruction.

A.0.4 RETURN

The **RETURN** instruction jumps to the instruction immediately following the last **CALL** instruction. Subroutines may call subroutine to a maximum stack depth of 64.

A.0.5 STOP

The **STOP** instruction terminates execution.

A.2 Class 1 Instructions (Opcodes)

The operand is a one byte integer (0 to 255).

A.1.1 BEEP

The **BEEP** instruction beeps the number of times equal to the operand.

Example: `BEEP 3`

A.1.2 COMPRESS

The **COMPRESS** instruction specifies if compression should be performed or not during backup. COMPRESS is ignored unless encryption is enabled.

Example: `COMPRESS Y`

A.1.3 DAY?

The **DAY?** instruction sets the condition code to "true" if the day of week matches the operand. Use 1 to 31.

The purpose of the **DOW?** instruction is to facilitate making "day of the month" backups.

Example: `DAY 7`

A.1.4 DEBUG

The **DEBUG** instruction sets the debug level, which is 0 (off), 1 (low), or 2 (high). The default debug level is 0.

Example: `DEBUG 1`

A.1.5 DISGUISE

The **DISGUISE** instruction specifies if file names should be disguised during backup. If the DDC (see SET_DDC) was also set, directory names will also be disguised.

Example: `DISGUISE Y`

A.1.6 DOW?

The **DOW?** instruction sets the condition code to "true" if the day of week matches the operand. Use 0 to 6 (Sunday = 0, Monday = 1, etc), or use `"Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"`.

The purpose of the **DOW?** instruction is to facilitate making "day of the week" backups. That is, a Monday backup, a Tuesday backup, etc.

Example: `DOW? Mon`

A.1.7 ENCRYPT

The **ENCRYPT** instruction specifies if encryption should be performed or not during backup. The default is no encryption.

Example: `ENCRYPT Y`

A.1.8 HIDDEN

The **HIDDEN** instruction specifies if hidden files should be processed or not. The default is that hidden files are not processed.

Example: `HIDDEN Y`

A.1.9 MONTH?

The **MONTH?** instruction sets the condition code to "true" if the month matches the operand. Use 1 to 12 (January = 1, February = 2, etc), or use `"Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec".`

The purpose of the **MONTH?** instruction is to facilitate making monthly backups.

Example: `MONTH? Feb`

A.1.10 READONLY

The **READONLY** instruction specifies if read-only files should be processed or not. The default is that read-only files are processed.

Example: `READONLY N`

A.1.11 RUN

The **RUN** command takes one of three operands: `BACKUP, RESTORE, LIST`, and `VERIFY`.

BACKUP -- Starts the process of backing up files.
RESTORE -- Starts the process of restoring encrypted files.
LIST -- Starts of process of listing disguised filenames.
VERIFY -- Verifies USER command.

After backing up or restoring is completed, the source and target directories (as specified by the **`SOURCE`** and **`TARGET`** instructions) are cleared.

Example 1: `RUN BACKUP`
Example 2: `RUN RESTORE`
Example 3: `RUN LIST`

A.1.12 SET_DDC

The **SET_DDC** command sets the "directory disguise character (DDC) that is appended to the end of each <u>disguised</u> directory component provided that the "`DISGUISE Y`" is specified. The purpose of the DDC is to allow SSB to determine if a directory name was disguised or not when restoring files. The default is that there is no DDC (no disguising of directories).

The **SET_DDC** command takes the disguise character (`'$', '#', or '@'`) or '`N`' (to disable directory disguising) as its argument.

Example: `SET_DDC $`

A.1.13 SLEEP_MIN

The **SLEEP_MIN** command causes **SSB** to sleep the number of minutes (1 to 255) specified.

Example: `SLEEP_MIN 1`

A.1.14 SLEEP_SEC

The **SLEEP_SEC** command causes **SSB** to sleep the number of seconds (1 to 255) specified.

Example: `SLEEP_SEC 5`

A.1.15 SUB_DIRS

The **SUB_DIRS** instruction specifies if sub-directories should be processed or not during backup or restore operations. The default is Y ("yes").

Example: `SUB_DIRS N`

A.1.16 TIMESTAMP

The **TIMESTAMP** instruction sets the maximum time difference (0 to 120 seconds) between 2 files to consider them unchanged. This is necessary because of the different date stamp resolutions between the various file systems used by Windows. The default value is 1 second.

To disable timestamp comparisons, use "`TIMESTAMP N`".

Example: `TIMESTAMP 0`

A.1.17 VERBOSE

The **VERBOSE** command sets the debug level to N (off) or Y (on).

Example: `VERBOSE Y`

A.2 Class 2 Instructions (Opcodes)

The operand is a 16-bit code address.

A.2.1 CALL

The **CALL** instruction calls the block of code beginning with the operand label. Control returns to the instruction following the CALL instruction when the **RETURN** instruction is executed.

Example: `CALL Hello`

A.2.2 GOTO

The **GOTO** instruction transfers control to the program address (label) specified by the operand.

Example: `GOTO Fini`

A.2.3 IF_FALSE

The **IF_FALSE** instruction transfers control to the program address (label) specified by the operand if the condition-code is false.

Example: `IF_FALSE QuitNow`

A.2.4 IF_TRUE

The **IF_TRUE** instruction transfers control to the program address (label) specified by the operand if the condition-code is true.

Example: `IF_TRUE continue`

A.3 Class 3 Instructions (Opcodes)

The operand is a 16-bit string address.

A.3.1 AUTH

The **AUTH** instruction specifies the path name of the restore (file recovery) authorization file. The authorization file is unique for each customer and is used only when restoring previously backed up files. The purpose of the authorization file is to prevent someone else, who has a copy of the pass.bin file, from decrypting files.

Example: `AUTH Auth.bin`

A.3.2 COPY

The **COPY** instruction contains the file specification (using ? and * wildcards) of the files to be processed during backup or restore. Up to 200 **COPY** commands may be specified.

Example: `COPY *.c`

A.3.3 DIR?

The **DIR?** instruction sets the condition code to "true" if the directory as specified by the operand exists.

Example: `DIR?  D:\SourceFiles\`

A.3.4 DISPLAY

The display instruction pops up a Windows message box with a "OK" button on it. The user must click the "OK" button before execution can continue.

Example: **DISPLAY** `"Hello, world"`

A.3.5 DRIVE?

The **DRIVE?** instruction sets the condition code to "true" if the drive as specified by the operand exists.

Example: `DRIVE?  F:`

A.3.6 FILE?

The **FILE?** instruction sets the condition code to "true" if the file as specified by the operand exists.

Example: `FILE? D:\SourceBackup\Code`

A.3.7 IGNORE

The **IGNORE** instruction contains the directory (folder) name to be ignored (not processed) during backup. Up to 200 **IGNORE** commands may be specified. For example, "`IGNORE OBJ`" will ignore all directories named OBJ (and their sub-directories). Wildcards are not supported.

Example: `IGNORE OBJ`

A.3.8 LOG_FILE

The **LOG_FILE** instruction specifies the name of the log file, which contains the details of the backup or restore run. The default log file name is SSB.LOG.

Example: `LOG_FILE Backup.log`

A.3.9 MAP_FILE

The **MAP_FILE** instruction specifies the name of the "disguise map", the mapping from the original filename (or folder names) to the disguised filename (or folder name) created during backup.

Example: `MAP_FILE Backup.map`

A.3.10 MOUNT

The **MOUNT** instruction verifies that the specified drive is ready to use. If it is not, a prompt message is displayed, to which the user must reply.

Example: `MOUNT E:`

A.3.11 PASS

The **PASS** instruction specifies the path name of the pass phrase file previously created by MakePass.exe program. The **PASS** instruction is required before encryption or decryption can be performed.

Example: `PASS Pass.bin`

A.3.12 SKIP

The **SKIP** instruction contains the file specification (using ? and * wildcards) of the files to be skipped during backup or restore. Up to 200 SKIP commands may be specified.

Example 1: `SKIP *.bak`
Example 2: `SKIP *.*.bak`

A.3.13 SOURCE

The **SOURCE** instruction specifies the full path location of the root directory containing the files to be processed during backup.

Example 1: `SOURCE C:\SourceFiles\Code`
Example 2: `SOURCE \\Remote\SourceFiles`

A.3.14 TARGET

The **TARGET** instruction specifies the full path location of the root directory of the backed-up files.

Example 1: `TARGET D:\SourceBackup\Code`
Example 2: `TARGET \\Remote\SourceBackup`

A.3.15 USER

The **USER** instruction specifies the path name of the user ID file "user.bin". The User ID file contains the customer's customer ID (100 = evaluation version) and the customer's registration string. The User ID file is read by both the backup software ssb.exe and the pass phrase file creation software MakePass.exe

Example: `USER c:\ssb\User.bin`

A.3.16 VOL_NAME?

The **VOL_NAME?** instruction sets the condition code to "true" if the volume name as specified by the operand exists. The volume name can be found by using the Windows VOL command (e.g.: `VOL C:`).

Example: `VOL_NAME?  F:MIKE2`

A.3.17 VOL_SERIAL?

The **VOL_SERIAL?** instruction sets the condition code to "true" if the volume serial number as specified by the operand exists. The operand must be a number. Decimal numbers must end with '.' otherwise the number is assumed to be hexadecimal (as displayed by the Windows **VOL** command).

The volume serial number can be found by using the Windows VOL command (e.g.: `VOL C:`).

Example 1: `VOL_SERIAL? C:8035CD42`
Example 2: `VOL_SERIAL? C:2151009602.`

A.3.18 WRITE

The **WRITE** instruction writes text to the log file.

Example: `WRITE "March Backup"`