



# **EASy**

Embedded Architecture System

## ***Manual for the tools***

v1.1, DEC/2002  
(C) H+T, [WWW.H-PLUS-T.COM](http://WWW.H-PLUS-T.COM)

# Contents

<b>1 SCOPE</b>	<b>4</b>
<b>2 TRADEMARKS</b>	<b>4</b>
<b>3 INTRODUCTION</b>	<b>5</b>
<b>4 OVERVIEW OF THE TOOLS</b>	<b>5</b>
<b>5 DESCRIPTION OF THE TOOLS</b>	<b>6</b>
<b>5.1 JSA - Java(TM) class file disassembler</b>	<b>6</b>
5.1.1 Description	6
5.1.2 Usage and invocation	6
5.1.3 Examples	6
<b>5.2 ASJ - Java(TM) class file assembler</b>	<b>7</b>
5.2.1 Description	7
5.2.2 Usage and invocation	7
5.2.3 Example	7
<b>5.3 NZIN - NZF-file archive packaging tool</b>	<b>8</b>
5.3.1 Description	8
5.3.2 Usage and invocation	8
5.3.3 Example	9
<b>5.4 NZEX - NZF-file archive expander</b>	<b>10</b>
5.4.1 Description	10
5.4.2 Usage and invocation	10
5.4.3 Example	10
<b>5.5 SERVANT - terminal program</b>	<b>11</b>
5.5.1 Description	11
5.5.2 Usage and invocation	11
5.5.3 Example	11
5.5.4 Program usage	11
5.5.5 Additional usage info	12
<b>5.6 ARRAY - C-language data array generation</b>	<b>13</b>
5.6.1 Description	13
5.6.2 Usage and invocation	13
5.6.3 Example	13
<b>5.7 REPL - text replacement tool</b>	<b>14</b>
5.7.1 Description	14
5.7.2 Usage and invocation	14
5.7.3 Example	14
<b>5.8 ROMFILE - file image generation</b>	<b>15</b>
5.8.1 Description	15
5.8.2 Usage and invocation	15

5.8.3 Example	15
<b>5.9 V51M - 8051 simulator</b>	<b>16</b>
5.9.1 Description	16
5.9.2 Usage and invocation	16
5.9.3 Example	16
5.9.4 Program usage	16

# 1 Scope

This document contains information about tools and helper programs available for EASy.

# 2 Trademarks

Sun, Java, and JavaSoft are trademarks or registered trademarks of Sun Microsystems, Inc.

OS/2, and PC-DOS are registered trademarks of IBM Corporation

MS-DOS, Windows, WindowsNT, and Windows95 are registered trademarks of Microsoft Corporation

All other product names mentioned herein are trademarks of their respective owners

## 3 Introduction

During the development of EASy a number of tools and helper programs have been developed, which support the development process of the Virtual Machine, portation to other platforms, analysis and modification of class files, and the build process of the whole system. According to the philosophy of H+T, most of the tools have been designed and programmed platform independently using strict ANSI-C language. They are available for DOS, Linux, WindowsNT, and OS/2 where applicable.

## 4 Overview of the tools

The following tools are available:

ASJ	for DOS, OS/2, WinNT	Java(TM) class file assembler
JSA	for DOS, OS/2, WinNT	Java(TM) class file disassembler
NZIN	for DOS, OS/2, WinNT	NZF class library packaging tool
NZEX	for DOS, OS/2, WinNT	NZF class library package expander
SERVANT(*)	for DOS, OS/2, WinNT, Linux	terminal program, version 1
ARRAY	for DOS, OS/2, WinNT, Linux	C-language data array generation tool
REPL	for DOS, OS/2, WinNT, Linux	text replacement tool
ROMFILE	for DOS, WinNT, Linux	tool for ROM file image creation
V51M	only for DOS	8051 simulator

(\*) This is the 1st implementation of the Servant terminal program, supporting terminal console i/o and file i/o for the EASy.VM far filesystem (DEV\_FSN). The program shall not be used, since its predecessor "SERVANT2" is available. SERVANT2 is described in {MAN\_SERV}.

## 5 Description of the tools

### 5.1 JSA - Java(TM) class file disassembler

#### 5.1.1 Description

JSA is a tool, which converts .class files (created by any Java(TM) compiler) to mnemonic text files. The class file will be analyzed completely and all elements currently defined by the Java-standard will be found in textual statements in the output file. JSA is used to optimize the standard classfiles created by any Java(tm)-compiler during the build process of the standard classlibrary and demo programs developed for EASy.

#### 5.1.2 Usage and invocation

JSA <inputfile> [<outputfile>] [options]

inputfile: standard Java class file

outputfile: target file holding the disassembly

options: -c outputfile is ignored, the disassembly will be placed in a file named  
<inputfile>.dis

-n creates warnings if the inputfile contains native methods

-b correct unknown constant pool tags, if possible  
(for classes which contain proprietary constants)

-l creates local variable names as stored within local variables pool of the class file

#### 5.1.3 Examples

```
JSA main.class -c
```

```
JSA main.class main.dis
```

```
JSA main.class main.dis -n -b
```

## **5.2 ASJ - Java(TM) class file assembler**

### **5.2.1 Description**

ASJ is a tool, which re-creates .class files from disassembly files created by JSA. ASJ is used during the build process of the EASy standard class library and demo programs. In several optimization stages not further needed information (like symbolic debugging informationen and other) will be removed. ASJ is used furthermore to create standard classes, which cannot be created by standard-Java-compilers (e.g. java/lang/Object - which has no super class).

Codesize and runtime performance of the class files to be created depends on controls and statements, which are found in the input file. The implemented optimization algorithms cannot be described within this short manual. For example: ASJ is able to minimize the code size by analysis and re-build of the constantpool, by removal of additional attributes and statements which are not required for method execution, and by analysis of the Java-code.

### **5.2.2 Usage and invocation**

ASJ <inputfile> [outputfile] [options]

inputfile: file containing the disassembly cretated by JSA

outputfile: Java-classfile

options: -c outputfile is ignored, the disassembly will be placed in a file named  
<inputfile>.cla

### **5.2.3 Example**

```
ASJ main.dis -c
```

```
ASJ main.dis main.class
```

## 5.3 NZIN - NZF-file archive packaging tool

### 5.3.1 Description

NZIN is a tool, which creates a file archive from one or more input files and/or directories. The format of the created .nzf file is similar to unpacked Zip-files, but not fully compatible. NZIN is able to remove redundant information from class files which decreases the size of the file archive significantly (those files are up to 25percent smaller than standard uncompressed Zip-files).

### 5.3.2 Usage and invocation

NZIN [option(s)] archivfile input1 [input2...n]

archivfile:		name of the target archive file
input1...n:		inputfile / directory to place into the archive
options:	+a	appends to an existing file archive
	-r	recurse into subdirectories
	+p	packing (removes redundant informations from class files)
	+d	creates directory index -> decreases the filesize by re-use of redundant filenames, and improves the file access at runtime
	+s	store the starting directory name
	-l	remove leading file/path separators
	-u	store file separators in Unix-style '/'
	-e	store file names without extensions
	+f	store only directory names (with appended separator)
	-f	store only directory names (without appended separator)
	-c	convert names lower case
	-C	convert names upper case
	-w path	set working directory where to start processing
	-n file	name exchange file

name exchange file: a file which contains informations, under which names input files shall be stored within the archive

format of the name exchange file content:

```
original.ext=new/long/filename1  
Pfad\orig2.ext=new/long/filename2
```

### 5.3.3 Example

```
NZIN classes.nzf java\*.* -r +s +d +p -u
```

```
NZIN classes.nzf java\*.* -n dosnames.txt -r +s -u
```

## **5.4 NZEX - NZF-file archive expander**

### **5.4.1 Description**

NZEX is a tool, which extracts the files stored within NZF- or uncompressed Zip-files.

### **5.4.2 Usage and invocation**

NZEX [options] archivfile [targetpath]

archivfile: name of the archive file to expand

targetpath: directory, where the extracted files will be stored

options: -e empty file entries won't be stored  
-> original-Zip-files often contain additional directorynames, which are stored as empty files

### **5.4.3 Example**

```
NZEX -e classes.nzf
```

```
NZIN -e classes.nzf c:\files
```

## 5.5 SERVANT - terminal program

### 5.5.1 Description

SERVANT is a textmode terminal program, which is designed especially to communicate with the EASy runtime running on embedded platforms. It supports character in- and output (servicing the STD\_IN, STD\_OUT, STD\_ERR- channels implemented by EASy), system download, download of classfiles and classlibraries, and special features like memory dump requests and other.

### 5.5.2 Usage and invocation

SERVANT <channel> [options]

channel: channel used for communication with EASy runtime

currently supported channels:

sci/sci0 or sci/sci1 - serial lines

socket/<port> - TCP/IP sockets (server mode)

socket/???.???.???.???:<port> , or

socket/<name>:<port> - TCP/IP sockets (client mode)

optionen: -i extended info mode (prints status messages)

-c disables user commands (all input is directed to the client)

-e print local echo

### 5.5.3 Example

```
SERVANT sci/sci0 -I
```

```
SERVANT socket/127.0.0.1:1024
```

```
SERVANT socket/localhost:1024
```

### 5.5.4 Program usage

SERVANT starts in terminal mode (input and output will received from / transmitted to the target platform). Pressing ESCAPE switches to the command mode and back. After every command the program continues automagically in terminal mode.

supported commands:

q	quit the program
e	sends the ESCAPE-character (0x1b) to the target platform
space	clears the console screen
f	sends a file on binary mode (byte after byte)
r	receives data to a file in binary mode

In command-less mode (option -c) the ESCAPE character is transmitted to the client, e.g. command mode is not entered. To enter the command mode, press ESCAPE three times.

### **5.5.5 Additional usage info**

When using a server socket for communication, SERVANT waits for connection attempts from TCP/IP clients at the specified port. Once connected, no further connection will be accepted, as long as the initial connection remains established. After connection abortion by the client, SERVANT re-enters the state of accepting connections.

When using a client socket for communication, SERVANT tries to connect to an open server socket. After connection abortion by the server, SERVANT loses the client connection and shall be terminated.

Note: SERVANT can act as very simple telnet demon or telnet client, when connecting with/to port 23.

## **5.6 ARRAY - C-language data array generation**

### **5.6.1 Description**

ARRAY is a tool, which reads the content of a file and converts it to a byte-per-byte plaintext hexadecimal representation. This tool is used e.g. during the build process of EASy when importing class library files to C-sources. The data output is always printed to the system STDOUT channel.

### **5.6.2 Usage and invocation**

ARRAY inputfile [options]

inputfile:        any file

options:        -q        add quotation marks

                 -c        C-style byte format (default)

                 -a        ASM-style byte format

                 -h        hex string format

                 -lp "<string>"    line prefix string

                 -ls "<string>"    line suffix string

### **5.6.3 Example**

```
ARRAY user.nzf -c >user.new
```

## **5.7 REPL - text replacement tool**

### **5.7.1 Description**

REPL is a tool, which analyses the content of a file and replaces text/data strings. This tool is used e.g. during the build process of EASy when replacing the identifiers “eASY/” to “java/”.

### **5.7.2 Usage and invocation**

REPL [-h] <searchpattern> [-h] <new pattern> <inputfile> <outputfile>

inputfile:        any file

options:        -h        the searchpattern / new pattern is given as hexadecimals

### **5.7.3 Example**

```
REPL OLDTEXT NEWTEXT file.in file.out
```

```
REPL -h 0x20,0x20,0x20 NEWTEXT file.in file.out
```

```
REPL -h 0x0d,0x0a -h 0x0a file.in file.out
```

## 5.8 ROMFILE - file image generation

### 5.8.1 Description

ROMFILE is a tool, which creates a ROMable file image as hexadecimal file. This file can be downloaded directly to FLASH / EEPROM memory, or programmed to ROM memory of a target platform. In case no input files are given, the program generates an empty ROM file image. The file image has the following structure:

```
[
  U2 filenamelen = 0x????          - motorola byteorder
  U1 filename[filenamelen]
  U4 datalen = 0x????????        - motorola byteorder
  U1 data[datalen]
  | [
  ...
  ]
]
U2 end_of_filechain = 0x0000
```

### 5.8.2 Usage and invocation

```
ROMFILE <address> <type> [inputfile1|[inputfile2|...]]
```

address: start address, where the ROM file chain will be programmed to

type: output file type

supported types: '02' intel hex format

inputfile?: - optional

### 5.8.3 Example

```
ROMFILE 0x00010000 02 classes.nzf user.nzf > romfiles.hex
```

```
ROMFILE 0x00010000 02 > no_files.hex
```

## **5.9 V51M - 8051 simulator**

### **5.9.1 Description**

V51M has been developed during the portation of EASy to the 8051 family controllers. Currently not all features of the 8051 prozessor are supported, but V51M is useful to analyze and debug EASy runtime for 8051. V51M currently runs only under DOS, since a part of the simulation ALU-engine is code in assembly language. V51M is not optimized. On Pentium 133MHz-Systems it runs 8051 code 20times slower than a 12Mhz clocked real 8051 hardware, therefore it cannot replace a real emulator. V51M supports sourcecode debugging (only for .ABS-files created by Keil8051 compilers). V51M emulates the proprietary ServiceIo-Interface over which the EASy runtime downloads class files, therefore a complete execution of Java-classes will be simulated.

### **5.9.2 Usage and invocation**

V51M inputfile

inputfile: Intel-Hex or Keil L51-ABS file

### **5.9.3 Example**

V51M easy.abs

V51M easy.hex

### **5.9.4 Program usage**

The program ist self describing. Available commands and keys are displayed at the bottom of the simulation window.



## Appendix A: References

{MAN\_SERV} H+T: SERVANT. User manual, v1.0, August 2001.

This page has intentionally left blank.