



EASy

Embedded Architecture System

User manual

for EASy.VM for workstation platforms

v1.3, DEC/2002
(C) H+T, WWW.H-PLUS-T.COM

Contents

1.	SCOPE.....	5
2.	NOMENCLATURE	6
3.	GENERAL INFORMATION.....	7
3.1	Features	7
3.2	Limitations.....	8
3.3	Current development acitivities.....	8
4.	INSTALLATION AND SETUP.....	9
4.1	EASy for workstation platforms	9
4.1.1	Installation.....	9
4.1.2	Setup.....	9
4.1.3	Starting EASy	9
4.2	EASy for RT-Target32 platform	10
4.2.1	Installation.....	10
4.2.2	Setup.....	10
4.2.3	Starting EASy	10
5.	STARTUP OF THE EASY SYSTEM.....	11
6.	ONCE YOU HAVE INSTALLED EASY.....	14
6.1	Getting started.....	14
6.2	Exploring the demonstration classes	14
6.2.1	The Merlin shell application.....	14
6.2.2	LS.....	15
6.2.3	CP	15
6.2.4	ED	16
6.2.5	RM	16
6.2.6	CLEAR.....	17
6.2.7	ALIAS.....	17
6.2.8	BENCH	17
6.2.9	MAN.....	17
6.2.10	MOUNT.....	18
6.2.11	UMOUNT.....	18
6.2.12	TTT.....	19
6.2.13	MEM.....	19
6.2.14	GC.....	19
6.2.15	SCHEDD	19
6.2.16	TELNETD.....	20
6.2.17	TELNET.....	20

7.	THE TOOLS	21
7.1	SERVANT2 - terminal program	21
7.1.1	Description	21
7.1.2	Usage and invocation.....	21
7.1.3	Example.....	22
7.1.4	Program usage.....	22
7.1.5	Additional usage info.....	22
8.	HOWTO'S	23
8.1	Howto use the far/ filesystem connected via serial line.....	23
8.2	Howto use the far/ filesystem connected to TCP/IP sockets.....	25
8.3	Howto establish a telnet session to EASy.....	26
8.4	Howto establish a remote debugging session to EASy	27
8.4.1	Debugging at startup.....	28
8.4.2	Debugging by break-in.....	29
9.	RUNNING OWN JAVA APPLICATIONS.....	30
	APPENDIX A: REFERENCES.....	31

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 the trademarks of their respective owners

1. Scope

This document focuses on the EASy.VM 3.x demonstration version for

- MS-DOS
- WindowsNT
- OnTime RT-Target32

Note: The following workstation platforms are supported by EASy.VM 3.x, too:

- MacOSX
- BSD Unix
- Windows CE for handheld PC's
- Linux
- OS/2
- AIX
- VenturCom ETS 10.x and RTX5.x
- ENEA OSE 4.4 (under development)

...also a number of embedded platforms:

- i8086, 80x86 processor based targets
- Infineon 80166 processor based targets
- Hitachi H8/300, H8S processor based targets
- Motorola PPC60x, G3, G4 processor based targets

The following chapters explain the functionality of the EASy.VM binary, including installation information. After that, information about the included Java class library and sample applications will be given, as well as instructions, how to write and start own Java applications for EASy.VM. This documentation does not contain any information about internal methods and functionality of EASy.VM. For details on that, refer to {HAR}.

2. Nomenclature

The following equations are used within this document:

<i>EASy</i>	EASy.VM runtime
<i>target</i>	any supported platform

3. General information

3.1 Features

The EASy 3.x binary provides the following features:

- interpretation of Java .class files, generated by any Java development kit
- Java threading on bytecode level supported
- serial communication interface
- timer support
- TCP/IP socket interface (uses the operating systems socket subsystem)
- EASy native method interface, refer to document {ENI}
- Java source level debugging built into the kernel (JDWP and EDI), refer to document {EDI}
- HELIOS device subsystem with the following drivers:
 - DEV_STD - driver for standard input/output/error channels
 - DEV_SCI - serial interface driver
 - DEV_FSL - platforms local filesystem
 - DEV_FSZ - .nzf/uncompressed .zip file packages filesystem
 - DEV_FSM - temporary memory block filesystem
 - DEV_FSN - filesystem over communication interfaces (e.g. serial line, sockets)
 - DEV SOCK - socket communication driver
 - DEV_VCON - virtual textual console driver

3.2 Limitations

Currently, EASy 3.x has the following limitations:

- AWT/SWING currently not supported
- limited set of class library functions

3.3 Current development activities

The following development activities are ongoing and available soon:

- J2ME (MIDP) support -> graphic, sound, and multimedia API
- EASy graphic subsystem and Java GUI
- thin TCP/IP stack for embedded systems and the workstation version for DOS
- CAN driver for EASy for embedded systems

4. Installation and Setup

This chapter explains installation, setup, and usage of the EASy.VM executables which are provided for demonstration purposes. When EASy is adapted and used in customer environments, installation, setup, and usage of the VM may be different.

4.1 EASy for workstation platforms

4.1.1 Installation

EASy.VM for workstation platforms is delivered in a single directory per platform with the following files:

<platform>\	
classes.nzf	file archive containing the class library in its latest configuration
user.nzf	file archive containing some example application classes
system.cfg	startup configuration file
easy[.exe]	main binary executable

4.1.2 Setup

The runtime parameters of the VM can be changed by editing the *system.cfg* file. Without changing this file, EASy.VM is configured as follows:

- classes are taken from the target OS' local directory ("local" filesystem driver)
- classes are taken from the file archives "classes.nzf" and "user.nzf"
- temporary memory filesystem "temp" mounted
- socket interface enabled (for OS' with TCP/IP socket support)
- "far"-filesystem through serial line or socket interface not enabled
- current target OS' console is used for standard input/output

4.1.3 Starting EASy

1. change to the directory where the binary executable resides
2. type "easy" <return>

4.2 EASy for RT-Target32 platform

4.2.1 Installation

EASy.VM for RT-Target32 is delivered in a single directory with the following files:

RT32\
 easy.rtb RT32 monolith, containing the main binary executable, and the built-in files
 - classes.nzf (file archive containing the class library in its latest configuration)
 - user.nzf (file archive containing some example application classes)
 - system.cfg (startup configuration file)

4.2.2 Setup

In the delivered demonstration version, the system.cfg file cannot be edited, thus the runtime options cannot be changed. The runtime parameters of the VM, configured upon linkage of the RT32 monolith, are as follows:

- local filesystem ("local" filesystem driver) not enabled (RT-Files not linked to the monolith)
- socket support ("socket" driver) not enabled (RT-IP not linked to the monolith)
- classes are taken from the built-in file archives "classes.nzf" and "user.nzf"
- temporary memory filesystem "temp" mounted
- "far"-filesystem through serial line (COM1) enabled
- current target OS' console is used for standard input/output

4.2.3 Starting EASy

1. boot the monitor at the target PC
2. connect the target PC with the serial port COM1 of the HOST PC running WindowsNT
3. under WindowsNT, type "rtrun easy" <return>

-> the executable is now downloaded to the target and started

5. Startup of the EASy system

When starting up, EASy loads the configuration file *system.cfg*, if present. The configuration file contains the device drivers to be mounted, and files/channels to be opened at startup.

The following example printout shows what happens:

```
std mounted.
sci mounted.
local mounted.
temp mounted.
socket mounted.
configuring the kernel...
std/stdio linked.
std/stderr linked.
Welcome to
HELIOS embedded operating system [...]
v... bld ... (c) ... H+T
Have a very nice day!
nzf/classes.nzf mounted.
nzf/user.nzf mounted.

Embedded Architecture System ...
v... (c) ... H+T
calling <clinit> for class bin/merlin ...done
calling <clinit> for class java/lang/System ...done
Merlin v... - powered by EASy.VM, (c) ... H+T
merlin>
```

1. mounting the system device drivers
 - std standard input/output/error channel driver
 - sci serial communication driver
 - local local (host OS) filesystem
 - temp temporary memory block filesystem
 - socket TCP/IP socket driver
2. linking the standard handles (standard in/out / error)
 - std/stdio standard in/out
 - std/stderr standard error
3. copyright and version information of the HELIOS hardware abstraction layer
 - Welcome to...
4. mounting the protocol and filter device drivers
 - nzf/classes.nzf mount nzf filesystem (using *classes.nzf* file)

- `nzf/user.nzf` `mount nzf filesystem (using user.nzf file)`

5. starting the EASy.VM application (copyright and version information printed)

- Embedded Architecture System...

6. informational messages, e.g. when new classes are initialized

- calling <clinit> for class...

7. start the Java startup application

The first class loaded by EASy is Java class called **main**. This class must contain the method: **public static void main(String[] args)**. The mentioned class can be supplied within by any of the device drivers mounted at startup of the system. It therefore can be laid in the application package **user.nzf**, in the local directory, or provided through a communication channel connected to EASy (serial line or TCP/IP socket).

The provided sample application class **main** immediately starts the demo application **bin/merlin**, being supplied within the **user.nzf** package, which implements a rudimentary shell (“merlin>”).

8. now the Java shell application is ready to interact with the user

6. Once you have installed EASy...

6.1 *Getting started*

To get a first impression about EASy.VM, simply start the executable.

EASy will load the start class "main", method void main(String[]). The class "main" is located in the nzf-file "user.nzf". It is possible to get an own start class to be executed by storing it into the EASy-directory. This class has to be named "main" and must contain a public static void main(String[]) method.

6.2 *Exploring the demonstration classes*

The purpose of the provided Java applications is, to demonstrate how the provided class library can be used. In real life these applications usually won't be used in embedded EASy/Java applications. Nevertheless they are useful to understand some of the features of HELIOS and EASy.VM. The application classes described in the next chapters are stored in the user.nzf package.

6.2.1 **The Merlin shell application**

As mentioned, the main class loads bin/merlin at system startup. Merlin has been developed to provide a rudimentary shell application. Basically, Merlin reads commands from the standard input stream. It recognizes the following commands:

'?' prints a short help text to standard out stream
'exit' exits the merlin application / instance

Any other input is treated as java application to be invoked. Knowing this, one can start any provided Java demo application, as well as own classes.

6.2.2 LS

function: lists root directory, available device drivers, and files

invocation: ls [name]

examples:

ls	- lists the root directory
ls dev/	- lists all available device drivers
ls sci/	- the sci/ driver does not support the list operation
ls rom/	- list the ROM files
ls temp/	- list files in the temp/ file system
ls classes.nzf/	- list all files in the classes.nzf/ package
ls user.nzf/	- list all files in the user.nzf/ package
ls far/	- list root directory of the connected target machine (if SERVANT is used)
ls far/local/	- list files in the local/ directory, where the SERVANT application resides

Note: To use the far device, a communication channel must be established via serial line to another PC, which runs the SERVANT.EXE program. Optionally, the connection between EASy and SERVANT can be established using a TCP/IP socket. All access to the far/ file system is directed to the device drivers, which are loaded and visible within the SERVANT application.

Note furthermore: when accessing the far/ filesystem, the SERVANT.EXE program might print out informational messages, dependent on the parameters it was invoked with

6.2.3 CP

function: - copies files

invocation: cp <filesystem/source> <filesystem/target>

examples:

```
cp user.nzf/man/all.man temp/test
    - copies the file man/all.man from user.nzf/ to temp/ filesystem
    - the target file is named 'test'
```

```
cp user.nzf/man/all.man far/ test
    - copies the file man/all.man from user.nzf/ to far/ filesystem
    - the target file is named 'test'
```

Note: the file will be copied to the PC where SERVANT is running

6.2.4 ED

function: This is Mr.Ed, a simple text editor.

The following keys are supported:

F1	- mini help about supported keys (press any key to leave the mini help)
F2	- saves changes to the open file
ESC	- exits Mr.Ed
Tab	- inserts 8 spaces
up, down, right	- navigation
left, pgup, pgdn	- navigation
pos1, end	- navigation
backspace	- deletes previous character
DEL	- deletes whole line

invocation: ed <filesystem/filename>

examples:

```
ed temp/test
```

6.2.5 RM

function: - remove files

The name has to be specified with its fully qualified name ('filesystem/filename').

invocation: rm <filesystem/filename>

examples:

```
rm test
```

- attempts to remove the file test
- this will fail, since the name is not fully qualified

```
rm temp/test
```

- removes the file temp/test

```
rm far/test
```

- attempts to remove the file test at the target platform connected via far/
- this will fail, since the file name is not fully qualified for the target

```
rm far/local/test
```

- attempts to remove the file local/test at the target platform connected via far/
- this will work (given, that the file exists at the target)

6.2.6 CLEAR

function: - clears the console

invocation: clear

6.2.7 ALIAS

function: - sets a command alias using Java properties

invocation: alias <equate> <command>

examples:

alias cls clear	- sets an alias from 'cls' to 'clear'
cls	- now 'cls' has the same result as 'clear'
alias	- lists all aliases
alias cls	- show the alias for 'cls'

6.2.8 BENCH

function: - executes a benchmark program

invocation: bench

6.2.9 MAN

function: - show help pages for specific topics

- in case no parameter is specified, the file 'man/all.man' will be displayed

- one parameter is supported, which specifies the command for which the help text will be displayed

Hint: to see, which .man pages are supported, list the user.nzf/

invocation: man [command]

examples:

man	- displays 'man/all.man'
man ls	- displays 'man/l.s.man'
man ed	- displays 'man/e.d.man'

6.2.10 MOUNT

- function:
- mount new instances of device drivers to the system
 - at least one parameter must be given, which specifies the device to mount
 - some devices (filter devices, like 'nzf/' or 'far/') needs an additional parameter, to specify which file or connection is used by the driver instance

Hint: to see, which devices are available, list the dev/ directory ('ls dev/')

invocation: mount <device> [argument]

examples:

```
mount nzf user.nzf      - mount (another) instance of 'nzf/' as 'user.nzf' filesystem
mount far sci/sci0     - mount (another) instance of 'far/', using 'sci/sci0'
```

6.2.11 UMOUNT

- function:
- unmounts (removes) instances of device drivers from the system
 - one parameter must be given, which specifies the instance to be removed

Hint: to see, which instances are available, list the root directory ('ls')

Note: file systems cannot be unmounted, if there are opened files associated to them

invocation: umount <instance>

examples:

```
umount user.nzf      - unmount instance of 'nzf/'
umount far           - unmount instance of 'far/'
```

6.2.12 TTT

function: - TicTacToe game

invocation: ttt

6.2.13 MEM

function: - prints available runtime memory

invocation: mem

6.2.14 GC

function: - forces the garbage collector to run

invocation: gc

6.2.15 SCHEDD

function: - scheduler demon

This program simply starts a high prioritized thread, which interrupts the processing of other Java threads every 100 milliseconds and causes other threads to become active. Using this, the EASy runtime acts as time sliced multithreading system with a duration of 100ms.

The schedd program supports the parameters 'on' and 'off'. Invoking without any parameter gives the information whether schedd is already running or not.

invocation: schedd [on|off]

6.2.16 TELNETD

- function:
- simple telnet demon
 - once started, telnetd redirects the standard input / output channels to a client, connected via TCP/IP socket.
 - the program supports one parameter: port to listen on
 - per default, port 23 will be used.

Caution: telnetd shall not be executed under platforms without TCP/IP socket support.

invocation: telnetd [port]

6.2.17 TELNET

- function:
- simple telnet client
 - once started, telnet redirects the standard input / output channels to a telnet demon, connected via TCP/IP socket.
 - the program supports one or two parameters:

target address to connect to

port to connect to (optional parameter, default is: 23)

Caution: telnet shall not be executed under platforms without TCP/IP socket support.

Note: Currently it is not possible to connect telnet to the telnetd application running on the local host.

invocation: telnet <address> [port]

7. The tools

This chapter highlights one of the tools which is referred to within this documents, and which is used for development and operation of the VM.

7.1 *SERVANT2 - terminal program*

7.1.1 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 things. Additionally, SERVANT implements a monitor for the EASy source level debugging interface.

7.1.2 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: -l extend CR to CRLF

7.1.3 Example

SERVANT sci/sci0

SERVANT socket/127.0.0.1:1024

SERVANT socket/localhost:1024

7.1.4 Program usage

SERVANT uses the virtual console driver (“DEV_VCON”) to provide different virtual text consoles for specific operations. F1, F2, F3, F4 switches between the virtual consoles.

Virtual console 1 is used for the following SERVANT commands:

q	quit the program
f	sends a file in binary mode (byte after byte)

Virtual console 2 is used to display error messages generated by the SERVANT program or its subsystems.

Virtual console 3 is used for textual console communication to the target.

Virtual console 4 is used by the Java source level debugging monitor.

7.1.5 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 reenters 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.

8. Howto's

8.1 Howto use the far/ filesystem connected via serial line

The far/ filesystem can be used to exchange files through a communication channel (serial channel, or socket). To exchange files through a serial line, please perform the following steps:

1. copy the files

SERVANT.EXE, CLASSES.NZF, USER.NZF from the SERVANT2/<platform> directory to another (remote) PC

2. connect the remote PC through a null-modem cable to the local PC

3. start the SERVANT executable at the remote machine by invoking

'servant sci/sci0' (for use of COM1) or

'servant sci/sci1' (for use of COM2)

4. start the EASy executable at the local machine

5. to mount the far/ device driver into the EASy system:

5.1 **either** add one of the following lines to the devices section of the *system.cfg* file

dev=far,sci/sci0 (for use of COM1)

or

dev=far,sci/sci1 (for use of COM2)

or

Ink=0x04,sci/sciX X={0,1} (links COMx to channel 0x04)

dev=far,channel/0x04 (for use of channel 0x04, linked to COM0 or COM1 above)

5.2 **or** enter at the merlin command shell:

'mount far sci/sci0' (for use of COM1)

or

'mount far sci/sci1' (for use of COM2)

6. Now you can work with the new file system driver

By listing the filesystems ('ls') you should see the far/ filesystem driver.

To perform file operations on far/ like described above, try the following commands:

```
'ls far/'
```

```
'ls far/user.nzf/'
```

```
'cp far/user.nzf/man/all.man local/all.man'
```

```
'cp local/all.man far/temp/man/all.man'
```

```
'ls far/temp/'
```

```
'rm far/temp/man/all.man'
```

```
'ls far/temp/'
```

7. unmount the user.nzf/ filesystem from the EASy-kernel

```
'umount user.nzf'
```

- this disables the local user.nzf/ filesystem and causes EASy to load the classes over the serial line

- try to start one of the demonstration classes (at the merlin prompt, type 'ttt', and see whats

happening in the SERVANT-console at the remote PC)

8.2 Howto use the far/ filesystem connected to TCP/IP sockets

The far/ filesystem can be used to exchange files through a socket interface, too. To exchange files through a socket, please perform the following steps:

1. open a command line, change to the directory EASYSDK/, and type 'servant socket/2000'

- this causes servant to open a server socket at port 2000

2. start the EASy executable

3. to mount the far/ device driver into the EASy system, enter at the merlin command shell:

- 'mount far socket/127.0.0.1:2000'

4. Now you can work with the new file system driver as described in the previous chapter

8.3 Howto establish a telnet session to EASy

1. start the EASy runtime
2. in the merlin shell, enter 'telnetd' or 'telnetd <port>'
3. do one of the following:

- start the servant program from the local machine

 - 'servant socket/127.0.0.1:23' or

 - 'servant socket/127.0.0.1:<port>'

- start the servant program from a remote machine

 - 'servant socket/<IP-address>:23' or

 - 'servant socket/<IP-address>:<port>'

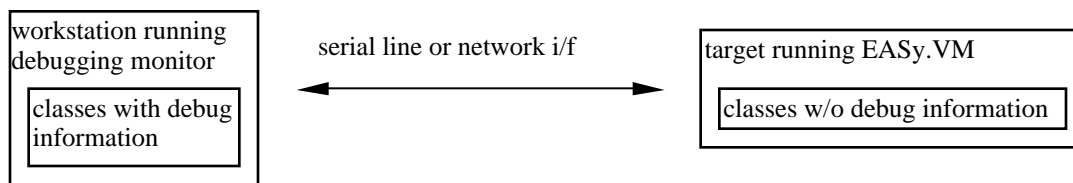
- start any other telnet client program

Note: using telnet clients (except SERVANT) can lead to misbehaviour when using special keys (like F1...F12, Up, Down, PgUp, PgDown,...) because currently EASy does not recognize ANSI escape sequences for input (while generated by some telnet clients). Additionally, some telnet clients have to be configured (no local echo, CRLF for linefeed).

8.4 Howto establish a remote debugging session to EASy

Prerequisite for remote debugging is that the Java source level debugging kernel was compiled into the EASy runtime (this is NOT the case in the official demo releases of EASy.VM available at the EASy website).

Remote debugging of Java code running under EASy is supported for embedded, and workstation platforms. The classes of the class library, as well as the application classes will be debugged, even if the classes do not contain any symbolic debugging information. Neither line number, nor variable name information are needed at the debugging target. The symbolic debugging information are retrieved by the debugging monitor from an own set of Java classes. See the following scheme:



The classes used by the monitor and the classes executed by EASy.VM must be compiled from exactly the same source code, otherwise the monitor won't be able to determine program counter position and variable content correctly.

The debugging connection can be performed in two ways:

1. start of the debugging session at startup of the EASy target ("debugging at startup")
2. start of the debugging session after the EASy target is already running ("debugging by break-in")

For detailed information how to operate the debug monitor SERVANT2, refer to {MAN_SERV}.

8.4.1 Debugging at startup

This method allows of tracing/debugging Java classes from the very beginning of bytecode execution. The disadvantage is that the EASy runtime has a short delay before startup, even if a debugging monitor is not started. This delay is because the EASy kernel debugger has transmitted a SYNCH-event message and awaits an answer from the monitor. If no monitor is answering, then the kernel debugger assumes that no monitor is present, thus continueing with execution of the 1st bytecode.

To configure EASy for debugging at startup,

1. edit *system.cfg* file and add one of the following statements to the linkage section:

<code>lnk=0x04,sci/sci0</code>	(for use of COM1 as communication channel)
<code>lnk=0x04,sci/sci1</code>	(for use of COM2 as communication channel)
<code>lnk=0x04,socket/<address>:<port></code>	(for use of TCP/IP as communication channel)

As one can see, the socket connection must be established as client, which connects to the debug monitor running as server.

2. start the debug monitor at a remote machine, using the communication channel as specified for EASy

```
servant sci/sci0
servant sci/sci1
servant socket/<port>
```

Switch to SERVANTs virtual console 4.

3. now start EASy.VM runtime

Note (for RTTarget32):

If the RT debugging kernel is used, and EASy runtime was installed using the RTRUN program, then the communication channel used by the EASy Java source level debugging kernel must be different from the channle used for downloading the EASy runtime to the RT-target. This is because the RT debugging kernel is listening at the mentioned channel for the RTD32 debugger.

8.4.2 Debugging by break-in

This method allows of tracing/debugging Java classes actually running. The disadvantage is that the startup class “main” often can’t be analyzed from the beginning of its execution, because the debug monitor usually will connect at a later time.

To configure EASy for debugging by break-in:

1. either

1.1.1 edit *system.cfg* file and add one of the following statements to the linkage section:

<code>lnk=0x04,sci/sci0</code>	(for use of COM1 as communication channel)
<code>lnk=0x04,sci/sci1</code>	(for use of COM2 as communication channel)
<code>lnk=0x04,socket/<port></code>	(for use of TCP/IP as communication channel)

1.1.2 start EASy.VM runtime

or

1.2.1 start EASy.VM runtime

1.2.2 at the merlin> prompt, enter one of the following:

<code>link sci/sci0 0x04</code>	(for use of COM1 as communication channel)
<code>link sci/sci1 0x04</code>	(for use of COM2 as communication channel)
<code>link socket/<port>0x04</code>	(for use of TCP/IP as communication channel)

As one can see, the socket connection must be established as server, which listens for incoming connection attempts from the debug monitor.

2. start the debug monitor at a remote machine, using the communication channel as specified for EASy

```
servant sci/sci0
servant sci/sci1
servant socket/<address>:<port>
```

Switch to SERVANT virtual console 4.

Note (for RTTarget32):

If the RT debugging kernel is used, and EASy runtime was installed using the RTRUN program, then the communication channel used by the EASy Java source level debugging kernel must be different from the channel used for downloading the EASy runtime to the RT-target. This is because the RT debugging kernel is listening at the mentioned channel for the RTD32 debugger.

9. Running own Java applications

EASy provides the possibility to run own Java applications. Any state of the art Java compiler / JDK can be used to develop applications for EASy. Since EASy supports only a subset of the JDK1.2 API, the EASy class library documentation shall be used to check, whether the class / methods to be used are supported or not. For an actual list, please refer to www.h-plus-t.com.

The best way to start:

1. take one of the Java demo applications provided with the EASy package
2. store it under a new name, e.g. *myclass.java*
3. modify the functionality
4. compile the class using any Java compiler

Then,

either

- copy the file *myclass.class* to the directory, where EASY.EXE program resides and is invoked
- within the Merlin shell, enter the name of the new class '*myclass*'

or

- copy the file *myclass.class* to the directory of the connected PC, where the SERVANT.EXE program resides and is invoked
- start SERVANT.EXE
- start EASy
- within the Merlin shell, enter the name of the new class '*myclass*'
- (the file should be loaded via serial line)

Note (1): if the user application class is named *main*, the user.nzf package is not required at runtime, since EASy will start the *main* class first and none of the other applications

Note (2): for better compatibility with platforms having restrictions in file names (e.g. the 8.3-naming restriction under MS-DOS), the '.class'-extension can be removed from class file names. The classes will be found by the VM, too.

Appendix A: References

- {EDI} H+T: EASy Source Level Java Debugging Interface, v1.2, Jan 2001.
- {ENI} H+T: EASy Native Interface, v1.0, August 2001.
- {HAR} H+T: HELIOS. Internal Design Manual for API 2.x, v2.2, August 2001.
- {MAN_SERV} H+T: SERVANT. User Manual, v1.0, August 2001.