



EASy.VM

Embedded Architecture System, Virtual Machine for Java

General Information Manual

v2.3, DEC/2002
(C) H+T, WWW.H-PLUS-T.COM

Contents

OVERVIEW	4
TRADEMARKS	4
INTRODUCTION	5
TARGET SYSTEMS	6
TECHNICAL OVERVIEW	7
Portability	7
Hardware/OS abstraction layer	7
Code sizes	7
Runtime Java Source Level Debugger with remote monitoring (JDWP and JDI)	7
Device driver architecture	8
File system support	8
Class package compression	8
Generic speed optimized memory management	8
Object caching / Reference caching	8
Dynamic resizable segmented stack architecture	8
3-level garbage collection	9
Preemptive multithreading with priority control on bytecode level	9
EASY.VM ARCHITECTURE	10
Restrictions/Configuration	11
Class libraries for EASy.VM	11
Class related error messages of EASy at runtime	11
HOW TO WRITE JAVA APPLICATIONS FOR EASY.VM	12
HOW TO GET EASY RUNNING ON YOUR TARGET PLATFORM	13
ADDITIONAL FEATURES OF EASY.VM	13

ENHANCEMENTS OF EASY.VM **13**

HOW TO SUPPORT CUSTOMER SPECIFIC FUNCTIONS **13**

WHAT H+T CAN SUPPLY **14**

TECHNICAL SUMMARY **15**

Overview

This document contains a technical overview, installation instructions and general information about EASy.VM, the Java virtual machine for workstations and embedded platforms.

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 the trademarks of their respective owners

Introduction

The Java™ virtual machine specification was introduced in 1992 by SUN Microsystems. It has originally been designed as a runtime system for consumer devices, which would be able to execute machine independent code with the goal to connect these devices to a network and to distribute programs very easy to them. In the meantime a couple of implementations of virtual machines for several workstation platforms have been created, since software companies got the idea to create a system which could replace classic operating systems in the future. But this moved the evaluation of Java into a direction where interpreter and the class libraries got bigger and bigger with the result to be unable to fit into small consumer devices.

EASy.VM as a portable clean-room-implementation is able to execute bytecodes according to the SUN Java-standard written by any Java Development Kit. It provides selected standard classes which are useful for interacting with microcontroller platforms. Special gimmicks are technologies like microkernel architecture, file systems, device driver model, memory management with garbage collection at runtime, remote kernel debugger, and dynamically loadable code modules, most of them more rarely seen on small controller platforms.

Since Java is a platform independent technology it is possible to execute programs which have been written under any development kit as long as runtime interpreter and development kit are compatible with the Java specification and the used library classes are available for the target system. To enhance this target system independency, EASy.VM is designed to be portable to nearly every platform. Certain hardware dependent features (such as different filesystems, drivers etc.) can be ported within days.

Target Systems

EASy.VM was created for all kinds of embedded systems and microcontroller devices, as well as workstation platforms, which have an 8bit-, 16bit-, 32bit-, and higher CISC or RISC microcontroller/processor on board. Due to its built-in system resource management (memory, I/O etc.), it is capable of running without any underlying operating system on every target platforms. On platforms with operating system, the built-in hardware abstraction layer acts as glue code between the VM and third party operating systems.

EASy.VM targets all kinds of microcontroller/processor devices such as:

- all kind of terminals, like Point of sale terminals, information kiosk systems et cetera
- (Public) telephones, cordless phones
- VCR's, TV, Hifi sets etc.
- Car information centers
- Stand alone and network connected devices
- Smart Card readers and Smart Cards
- Personal Computers

EASy.VM is ported to the following systems:

- 8086, 80x86 and higher under DOS (version 3.3 and higher)
- 80x86 and higher under Linux
- 80x86 and higher under WindowsNT/2000 and OS/2 2.x
- PowerPC G3,G4 under MacOSX
- PowerPC 60x under AIX operating system
- Hitachi H8/300H without underlying operating system
- Hitachi SH-1 without underlying operating system
- 80C167 family platforms without underlying operating system
- Philips XA family platforms without underlying operating system
- OnTime RT-Target32 realtime operating system
- VenturCom ETS10.x Embedded operating system
- VenturCom RTX5.x Embedded realtime operating system

- ENEA OSE 4.4 Embedded realtime operating system

Technical Overview

Portability

EASy.VM is written in 100% ANSI-C programming language to ensure maximum portability to platforms with a C compiler available. C-runtime libraries are not required for compilation. The architecture allows running on environments with or without underlying operating system.

Hardware/OS abstraction layer

To ensure minimum effort when porting to new platforms and operating systems, EASy.VM was designed with a hardware abstraction layer (called "HELIOS") encapsulating all hardware and third party operating system dependent functions. This layer is the only platform dependent component which needs to be ported manually. See Section "EASy.VM architecture" for a description.

Code sizes

EASy.VM was designed to run on 8/16/32bit-platforms and is therefore very compact in code. The code sizes of current implementations scale down to 32Kilobytes. The class library is configurable for the functional requirements of the Java applications to be run. On customer request, the class library is enhanced, and therefore it is evolving rapidly. The maximum size (at the time this document was created) is approximately 42Kilobytes. Dependent on the application, the class library size can be scaled down to a few kilobytes.

Runtime Java Source Level Debugger with remote monitoring (JDWP and EDI)

EASy.VM can be compiled with a built-in source level debugging kernel, which allows Java debuggers running on a remote host to connect to EASy.VM and to debug the Java applications remotely. This feature allows more effective debugging than by using simulators on a PC.

Two kinds of debugging protocols are supported by EASy.VM:

1. the JDWP (Java Debug Wire Protocol), supported by most of the state of the art Java development kits (e.g. Borland Jbuilder 6.0 professional).
2. the proprietary EASy EDI (EASy Debug Interface), supported by the SERVANT Debugging Monitor (comes with the commercial EASy.VM packages). This protocol leads to a smaller code size of the target VM runtime, and runs faster than JDWP.

Device driver architecture

The architecture of EASy.VM enables developers to write portable device drivers in native code. They can easily be embedded into the system to act as interface to nearly any underlying hardware.

File system support

Native and non-native (bytecode) classes, which contain the application and common functionality, can be loaded over pluggable file system drivers by the interpreter. Currently, the following drivers are available:

- uncompressed ZIP file system (multiple ZIP class files are supported),
- NZF file system (ZIP format like file packages, which include compression techniques),
- memory block file system (for temporary data),
- distributed (network) file system,
- local file system (e.g. standard file system provided by the underlying operating system)

Furthermore, ROM- and FLASH- file systems are supported.

When connected to a JDWP or JDI debugger, no classes are required on the target platform; classes will be downloaded from the debugging monitor.

Class package compression

Classes can be compressed outside the VM. This is supported by a compression tool of the EASy.VM-package.

Generic speed optimized memory management

Due to its internal memory management, no OS-specific memory management functions are needed by EASy.VM. This feature, among others, enables EASy.VM to run WITHOUT underlying OS. In environments with an underlying operating system, Java-applications can be run completely encapsulated from non-Java-code.

Object caching / Reference caching

EASy.VM provides internal caching mechanisms for references when accessing objects like variables, constants and methods. This feature increases execution speed of Java applications when frequently calling the same functions or accessing the same objects.

Dynamic resizable segmented stack architecture

EASy.VM takes advantage of a dynamic resizable operand stack which is - different to C-code - non-contiguous but allocated in segments. This can lead to a markable decrease of space required to run applications, as the stack segments can be dynamically allocated and deallocated, dependent on the requirements of the Java application.

3-level garbage collection

EASy.VM provides a 3-level, speed optimized garbage collector which is tightly connected to the memory management. It runs either transparent to the programmer or can be invoked by a system call. To achieve the best possible performance, the GC works on three different levels:

1. Level 1 finds and deletes all Java objects and arrays that are not visible to the application any more,
2. Level 2 deletes space needed by class and object variables that contain the value 0 (Null). This can be done, as all variables have this value by default and are not allocated before they are first accessed. If a variable is deleted by GC level 2, the next access to this variable will restore it.
3. Level 3 of the GC deletes linkage information, fast access tables, and free (kept for future use) stack segments. The system can live without this information; it will just use the same linking mechanisms as when linking a new class to the runtime environment, and will allocate new stack segments when needed. But performance will decrease due to the missing fast access data.

The GC will run automatically when memory is to be allocated and not available. After each executed level, the allocation is tried again. No further level is executed after a successful allocation. If the GC is called from within an application by the call to `System.gc()`, only the first 2 levels are executed.

Preemptive multithreading with priority control on bytecode level

EASy.VM can handle threads according to the Java™ specification, even on non-multitasking platforms/operating systems. Thread control is available for native code too.

EASy.VM architecture

The EASy.VM system has a layered architecture. A system service layer (called “HELIOS”) encapsulates hardware dependent functions. This layer is the only platform dependent component which needs to be ported manually. Additionally, dependent on the target system, startup, exit, and interrupt service functions, written in assembly or C language, may exist and have to be ported.

On top of the system layer several service modules exist, which provide memory management, Java operand stack management, device driver interface with dynamic linkable file system(s), native/non-native method management and, as the most important part, the bytecode interpreter. The microkernel design is implemented by a virtual function interface, through which native methods and system drivers are allowed to overload functions. Native and non-native (bytecode) classes, which contain the application and common functionality, are loaded over file system drivers by the interpreter. Currently, the following file system drivers are available: local file system (e.g. standard file system provided by the underlying operating system), uncompressed ZIP file system (multiple ZIP class files are supported), temporary memory block file system, distributed (network) file system, which are platform independent and easily portable.

Summarized, EASy.VM consists of few layers which are functionally separated. This has a positive impact on code size and runtime performance. See figure 1 for an overview of the modular design.

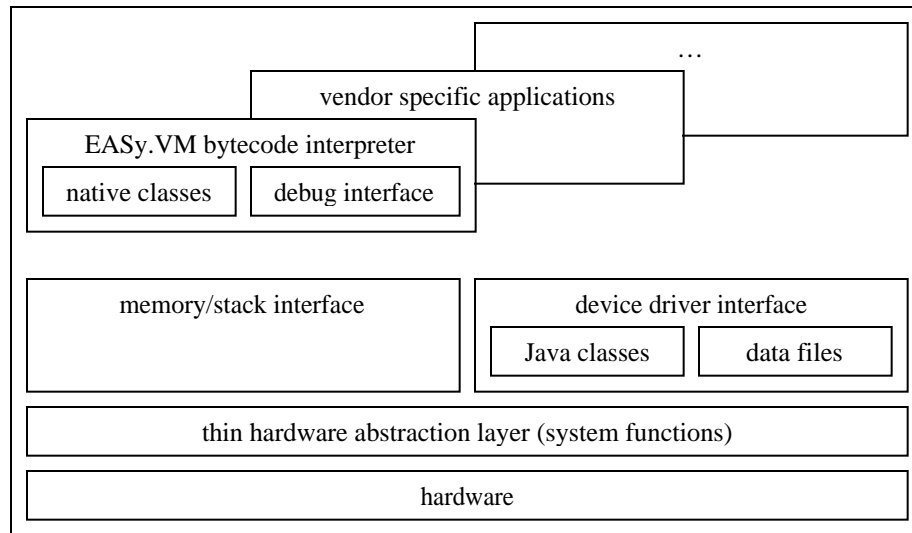


figure 1: the modular design of EASy.VM

Designing a portable system for microcontrollers means taking care of many different architectures. Therefore independency of the source code from byte order, address ranges, operand sizes, memory access logic and segmentation have been ensured during the development of EASy.VM.

Restrictions/Configuration

Java specifies at least 200 bytecodes, most of them high level language like, which require a functionality of the interpreter system, that is not easy to implement on platforms with limited resources (e.g. 8bit platforms) - unless the code size can be reduced by omitting some features (bytecodes) that are not needed for applications on these platforms. On these smaller platforms several features can be left out from the compilation of the VM.

For performance reasons and limited memory space on embedded systems it has been decided not to implement all formal bytecode verifications. They are not required on small devices which usually run only verified and tested code.

Only a subset of the standard Java classes are supported at the moment to hold the code size at lowest possible level; it depends heavily on the application which classes are needed. Of course, any class can be added to the runtime system. Please contact H+T for information on this.

Class libraries for EASy.VM

EASy.VM is ready to go with a core of classes, compatible to the Java class library specification. The classes were selected to cover common requirements like I/O-functionality on embedded systems, holding the code-size at lowest possible level. Currently more standard classes are under development. H+T is able to supply any class needed by customers for their application within a short time.

Class related error messages of EASy at runtime

If the started Java-bytecode uses a class that is not implemented, EASy.VM will stop execution with the error message:

- "cannot open file <classname>".

If a method in a class is used that is not yet implemented, EASy will post the error message:

- "cannot load method <method-name>".

How to write Java applications for EASy.VM

EASy.VM is designed to execute Java classes, which have been compiled by any standard Java compiler. Everyone can write code for EASy.VM using his favorite Java compiler and/or Java development kit. The syntactic, semantic and runtime tests can be performed under any development kit. After that the classes can be enclosed into a standard uncompressed ZIP-file (using the Java development kit or any packer which supports uncompressed ZIP-files, like PKZIP). Additionally the Java classes can be executed under the DOS, WindowsNT, Linux, MacOSX, OS/2, or AIX version of EASy to test them on a workstation environment before downloading them to the embedded system. There are special versions of EASy.VM available having Java source level debugging features built-in, allowing comfortable remote debugging of the Java code using a third party debugger.

Step by step example for getting Java code rolling under any of the workstation versions of EASy:

1. it is strongly recommended to use tools which run under WindowsNT, or other platforms being able to deal with long file names to make sure long names can be used for the classes to be created
2. design the classes and methods
3. start your favorite Java development kit and write down the code
 - make sure to use only class libraries currently supported by EASy.VM
4. test the code under the development kit
5. copy the EASy.VM runtime to the directory where the Java classes are stored
6. invoke EASy.VM with the class name holding the main() method, e.g. **"easy MyProg"** or **"easy java/myproject/MyProg"**

Note: 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.

How to get EASy running on your target platform

In general EASy is portable to any platform within a short time, provided that an ANSI-C compiler is available for that platform. Please contact H+T to discuss the details.

Additional features of EASy.VM

The following additional features are part of the system:

- minimization of system and user class library packages by class packaging tools
- runtime Java source level debugging kernel
- runtime configuration and reconfiguration of the kernel

Enhancements of EASy.VM

The following enhancements to the EASy.VM environment are planned / currently under development:

- extending the system class library with additional standard classes that are useful on embedded systems
- J2ME (MIDP) extension to EASy.VM, thus support of the MIDP graphics, sound, and multimedia API
- graphical subsystem and java.awt / java.swing package

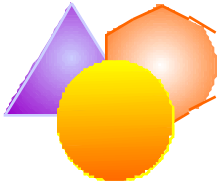
How to support customer specific functions

Due to the architecture of the native method interface it is very easy for customers to add classes and methods to support specific hardware. Please note: EASy.VM's native method interface is proprietary, not compatible to JNI.

What H+T can supply

- porting of EASy.VM to any 16/32/64bit-platform within days
- portation to additional 8bit platforms is possible, but further qualifications are needed
- customization of the VM runtime system for special applications and platforms
- extensions and enhancements to standard class library routines
- design and implementation of user application classes
- support for embedding customer specific hardware and functions
- services and consulting in EASy.VM, JAVA and embedded systems related topics

Please contact us for further information.



H+T

e-mail: info@h-plus-t.com

<http://www.h-plus-t.com>

Technical summary

- interprets instructions according to Java™ bytecode standard
- written in 100% ANSI-C programming language to ensure maximum portability to platforms with a C compiler available; standard C libraries are not required
- the architecture allows running on environments without underlying operating system
- current implementations run under
 - 8086, 80x86 and higher under DOS (version 3.3 and higher)
 - 80x86 and higher under Linux, WindowsNT, Windows2000, and OS/2 2.x
 - PowerPC G3,G4 under MacOSX
 - PowerPC under AIX operating system
 - Hitachi H8/300H and SH-1 without underlying operating system
 - 80C167, PhilipsXA family systems
 - OnTime RT-Target32 embedded realtime operating system
 - VenturCom ETS10.x and RTX5.x embedded realtime operating systems
 - ENEA OSE 4.4 embedded realtime operating systems
 - ... (other under development)
- generic, portable, speed optimized memory management
- dynamic resizeable, easily portable, generic stack architecture
- preemptive multithreading with priority control on bytecode level
- 3-level garbage collection optimized for performance
- device driver architecture
- supports file systems
- runtime Java debugging interfaces (JDWP and EDI supported)
- class package reduction and configuration by user

code size of the runtime version without class libraries: 32...120kB

code size of the class library: 35kB -> configurable down to a few kilobytes