RTA-OSEK

Binding Manual: PC

Contact Details

ETAS Group

www.etasgroup.com

Germany

ETAS GmbH
Borsigstraße 14
70469 Stuttgart

Tel.:+49 (711) 8 96 61-102
Fax:+49 (711) 8 96 61-106

www.etas.de

USA

ETAS Inc.
3021 Miller Road
Ann Arbor, MI 48103

Tel.: +1 (888) ETAS INC
Fax: +1 (734) 997-94 49

www.etasinc.com

Japan

ETAS K.K.
Queen's Tower C-17F,
2-3-5, Minatomirai, Nishi-ku,
Yokohama, Kanagawa
220-6217 Japan

Tel.: +81 (45) 222-0900
Fax: +81 (45) 222-0956

www.etas.co.jp

France

ETAS S.A.S.
1, place des États-Unis
SILIC 307
94588 Rungis Cedex

Tel.: +33 (1) 56 70 00 50
Fax: +33 (1) 56 70 00 51

www.etas.fr

Korea

ETAS Korea Co. Ltd.
4F, 705 Bldg. 70-5
Yangjae-dong, Seocho-gu
Seoul 137-889, Korea

Tel.: +82 (2) 57 47-016
Fax: +82 (2) 57 47-120

www.etas.co.kr

Great Britain

ETAS UK Ltd.
Studio 3, Waterside Court
Third Avenue, Centrum 100
Burton-upon-Trent
Staffordshire DE14 2WQ

Tel.: +44 (0) 1283 - 54 65 12
Fax: +44 (0) 1283 - 54 87 67

www.etas-uk.net

Copyright Notice

Version: RM00073-001

Disclaimer

The information in this document is subject to change without notice and does not represent a commitment on any part of LiveDevices. While the information contained herein is assumed to be accurate, LiveDevices assumes no responsibility for any errors or omissions.

In no event shall LiveDevices, its employees, its contractors or the authors of this document be liable for special, direct, indirect, or consequential damage, losses, costs, charges, claims, demands, claim for lost profits, fees or expenses of any nature or kind.

Trademarks

RTA-OSEK and LiveDevices are trademarks of LiveDevices Ltd.

Windows and MS-DOS are trademarks of Microsoft Corp.

OSEK/VDX is a trademark of Siemens AG.

All other product names are trademarks or registered trademarks of their respective owners.

Contents

1 About this Guide

This guide provides port specific information for the PC implementation of LiveDevices' RTA-OSEK.

A port is defined as a specific target microcontroller/target toolchain pairing. This guide tells you about integration issues with your target toolchain and issues that you need to be aware of when using RTA-OSEK on your target hardware. Port specific parameters of implementation are also provided, giving the RAM and ROM requirements for each object in the RTA-OSEK Component and execution times for each API call to the RTA-OSEK Component.

1.1 Who Should Read this Guide?

It is assumed that you are a developer. You should read this guide if you want to know low-level technical information to integrate the RTA-OSEK Component into your application.

1.2 Conventions

Important: Notes that appear like this contain important information that you need to be aware of. Make sure that you read them carefully and that you follow any instructions that you are given.

Portability: Notes that appear like this describe things that you will need to know if you want to write code that will work on any processor running the RTA-OSEK Component.

In this guide you'll see that program code, header file names, C type names, C functions and RTA-OSEK API call names all appear in the `courier` typeface. When the name of an object is made available to the programmer the name also appears in the `courier` typeface, so, for example, a task named Task1 appears as a task handle called `Task1`.

2 Toolchain Issues

In this chapter, you'll see the important details that you need to know about RTA-OSEK and your toolchain. A port of the RTA-OSEK Component is specific to both the target hardware *and* the compiler toolchain. You must make sure that you build your application with this toolchain.

If you are interested in using a different version of the same toolchain, you should contact LiveDevices to confirm whether or not this is possible.

2.1.1 Use of different toolchains for PC

Most ports of RTA-OSEK are restricted to using a single toolchain to build an application for specific target hardware.

The PC port, on the other hand, allows use of most popular C/C++ compilers for the PC platform. In particular, it has been tested with gcc (as distributed with MinGW 3.4.2), Borland bcc32 (version 5.5.1), and Microsoft Visual Studio (version 5, and 2003).

2.2 Compiler

The RTA-OSEK Component was built using the following compiler:

Vendor	Borland
Compiler	bcc32
Version	5.5.1

The compulsory compiler options for application code are shown in the following table:

Option	Description
none	

The prohibited compiler options for application code are shown in the following table:

Option	Description
none	

The C file that RTA-OSEK generates from your OIL configuration file is called `osekdefs.c`. This file defines configuration parameters for the RTA-OSEK Component when running your application.

The compulsory compiler options for `osekdefs.c` are shown in the following table:

Option	Description
none	

The prohibited compiler options for `osekdefs.c` are shown in the following table:

Option	Description
none	

2.2.1 PC "assembler" file

The RTA-OSEK build process generates a file which is a low-level interface to the target hardware, containing, for instance, interrupt wrappers and a vector table, which must be compiled and linked with your application.

This configuration file is called `osgen.cpp`. (This file is usually an assembler file, but for the PC port, it is a C++ source code file). This file defines configuration parameters for the RTA-OSEK Component when running your application.

In the case of the PC port, it is also used to interface to the source code and library code in DLLs, which provide the virtual ECU and its devices, as running on the target PC.

This file #include's code from the folder <rta_root>\VRTA\inc which is responsible for initializing the virtual ECU, and providing an interface between RTA-OSEK and the virtual ECU.

Additionally, code to create virtual peripherals of common types is #include'd unless VRTA_EXCLUDE_SAMPLE_DEVICES is #define'd.

If VRTA_INCLUDE_DEVICES is #define'd, a file devices.cpp in the application folder is #include'd, which is a convenient way for the application to create and control virtual peripherals such as counters.

See the virtual ECU documentation for more details of how peripherals for the virtual ECU are defined and used.

2.2.2 Control of RTA-OSEK and the virtual ECU

Because the RTA-OSEK libraries are not directly linked to the application, the user must specify when certain default routines in RTA-OSEK are to be over-ridden by user supplied code.

The following symbols must be defined if the default routines are **not** required:

Routine To Override	Symbol to be #define'd
CloseCOM	OS_USER_CloseCOM
InitCOM	OS_USER_InitCOM
MessageInit	OS_USER_MessageInit

RTA-OSEK generates a second configuration file called `osgen.cpp`. (This file is usually an assembler file, but for the PC port, it is a C++ source code file). This file defines configuration parameters for the RTA-OSEK Component when running your application.

2.3 Assembler

The RTA-OSEK Component was built using the following assembler:

Vendor	Borland
Assembler	tasm32
Version	5.3

2.4 Linker/Locator

In addition to the sections used by application code, the following RTA-OSEK sections must be located:

Sections	Rom/Ram	Description
`no sections used`	RAM	entire application linked as Windows executable

2.5 Application linkage

For most ports of RTA-OSEK, an application is linked directly with the appropriate RTA-OSEK library, and possibly some start-up code and libraries provided with the target toolchain. This forms the complete code for the application, which is then loaded into a target ECU.

The PC port, on the other hand, creates an application that uses some statically linked toolchain supplied libraries, and also uses DLLs which are loaded when the Windows application starts. There is no need for static linkage to the RTA-OSEK libraries.

This application startup code in `vrtaCore.cpp` and `vrtaOSEKsupp.c` is incorporated automatically when `osgen.cpp` is compiled.

2.6 Debugger

Information about ORTI for RTA-OSEK can be found in the *RTA-OSEK ORTI Guide*

At the time of writing, we were not aware of any debuggers for the PC with Windows with support for ORTI.

If you are using an ORTI version 2.0 aware debugger on this platform you can use the "Unknown ORTI debugger" option in the RTA-OSEK GUI to generate an ORTI output file. The ORTI generated will not have been tested on the debugger and, therefore, is not guaranteed to work.

Please contact LiveDevices if you have any questions about ORTI support in RTA-OSEK.

2.6.1 Target debuggers

An application built by RTA-OSEK for the PC port is a Windows executable. It is expected that a debugger supplied with the chosen toolchain will be used to debug the application.

There is no support for the use of ORTI debuggers with this port of RTA-OSEK.

2.7 Toolchains used to build applications

The example program is distributed with build instructions (compiler flags, library names, etc) for all of these common toolchains; a particular toolchains can be selected by using the target variant option within RTA-OSEK. This is an aid if multiple toolchains are used, but the generated headers and code are not affected. This application also illustrates how the build instructions may be tailored to keep intermediate and final files from the build process in different folders for each toolchain.

3 Target Hardware Issues

3.1 Interrupts

This section explains the implementation of RTA-OSEK's interrupt model. You can find out more about configuring interrupts for RTA-OSEK in the *RTA-OSEK User Guide*.

3.1.1 Interrupt Levels

In RTA-OSEK interrupts are allocated an Interrupt Priority Level (IPL). This is a processor independent abstraction of the interrupt priorities that are available on the target hardware. You can find out more about IPLs in the *RTA-OSEK User Guide*. The hardware interrupt controller is explained in the *Virtual ECU Reference Manual*.

The following table shows how RTA-OSEK IPLs relate to interrupt priorities on the target hardware:

IPL Value	Virtual ECU IPL	Description
0	0	User level
1 .. 32	1 .. 32	Category 1 and 2 interrupts

3.1.2 Interrupt Vectors

RTA-OSEK does not impose any restrictions on which interrupt vectors may be used.

3.1.3 Category 1 Handlers

Category 1 interrupt service routines (ISRs) must correctly handle the interrupt context themselves, without support from the operating system. The Borland C compiler can generate appropriate interrupt handling code for a C function decorated with the `void` function qualifier. You can find out more in your compiler documentation.

3.1.4 Category 2 Handlers

Category 2 ISRs are provided with a C function context by the RTA-OSEK Component, since the RTA-OSEK Component handles the interrupt context itself. The handlers are written using the OSEK OS standard `ISR()` macro, shown in Code Example 3:1.

```
#include "MyISR.h"
```

```
ISR(MyISR) {
  /* Handler routine */
}
```

Code Example 3:1 - Category 2 ISR Interrupt Handler

You must not insert a return from interrupt instruction in such a function. The return is handled automatically by the RTA-OSEK Component.

3.1.5 Vector Table Issues

When you configure your application with the RTA-OSEK GUI you can choose whether or not a vector table is generated within `osgen.cpp`.

If you choose to provide your own vector table, it must follow the format specified in the Virtual ECU documentation.

In particular, it must contain an entry for each interrupt handler, including the Category 2 interrupt handlers in RTA-OSEK.

The entries for Category 2 interrupt handlers must use `osCat2Wrapper` for the name of the handler routine, give the IPL of the ISR, and a pointer to the ISR's control block as the 'tag' entry.

The ISR control blocks are defined in `osgen.cpp`, using names of the form OS_LVVVV, where V is a hex digit. The control blocks can be identified by a member initialization of the form:

`(funcptr)osE_<isr_name>`.

3.2 Register Settings

The RTA-OSEK Component does not require the initialization of registers before calling `StartOS()`.

The RTA-OSEK Component does not reserve the use of any hardware registers.

3.3 Stack Usage

3.3.1 Number of Stacks

A single stack is used. The first argument to `StackFaultHook` is always 0.

`osStackOffsetType` is a scalar, representing the number of bytes on the stack, with C type: `unsigned int` .

3.3.2 Stack Usage within API Calls

The maximum stack usage within RTA-OSEK API calls, excluding calls to hooks and callbacks, is as follows:

Standard

API max usage (bytes): 320

Timing

API max usage (bytes): 320

Extended

API max usage (bytes): 340

To determine the correct stack usage for tasks that use other library code, you may need to contact the vendor to find out more about library call stack usage.

3.3.3 Stack allocation

Because of the interaction of Windows and the virtual ECU code, and the variety of toolchains that can be used to build an application, it is not possible to give precise figures for stack use by the application and RTA-OSEK library code.

It is recommended that a generous allowance is made in the declared stack use of tasks and ISRs, above the value used by the task code.

During initialization of the virtual ECU, the user stack is pre-extended to a size of 1 megabyte (the default stack size using Microsoft Devstudio's linker).

3.4 Floating point

All Pentium-class PCs have hardware floating-point. RTA-OSEK supports the use of floating point, using library code for default floating-point context save and restore operations.

The user may over-ride these routines if required; the interface and the code of the supplied default routines is provided is <rta-base>\VRTA\inc\osfptgt.c and osfptgt.h.

If the user wishes to over-ride any of these routines, the following functions must be called during initialization – each has a single parameter which is the address of the over-riding function:

Routine To Override	Initialisation Function To Be Called
os_fp_save	vrtaAssign_fp_save
os_fp_restore	vrtaAssign_fp_restore
os_fp_save_to	vrtaAssign_fp_save_to
os_fp_restore_from	vrtaAssign_fp_restore_from

Here is a short example of a possible initialization to for user-supplied routines:

```
OS_API(void) local_save(void);
OS_API(void) local_restore(void);
OS_API(void) local_save_to(PTR(OS_CONTEXT) a);
OS_API(void) local_restore_from(PTR(OS_CONTEXT) b);

void     setup_special_routines(void)
{
    vrtaAssign_save(local_save);
    vrtaAssign_restore(local_restore);
    vrtaAssign_save_to(local_save_to);
    vrtaAssign_restore_from(local_restore_from);
}
```

4 Parameters of Implementation

This chapter provides information on the functional limits of the RTA-OSEK Component for PC.

Detailed information on the performance and memory demands of the RTA-OSEK Component is not provided. Run-time performance obviously depends on the specification of the PC running the Virtual ECU code. Measurements on a particular PC would not be very useful; in general modern PCs have execution speeds much faster than most target ECUs actually used in production. Memory requirements for an application would depend of the chosen toolchain as well as the supplied libraries. The design of RTA-OSEK for PC, and of the Virtual ECU, has assumed that RAM memory will not be a limiting resource.

4.1 Functionality

The OSEK Operating System Specification specifies four conformance classes. These attributes apply to *systems* built with OSEK OS objects. The following table specifies the number of OSEK OS and COM objects supported per conformance class.

Configuration	**Application Uses**					
Events	**No**			**Yes**		
Shared Task Priorities	**No**		**Yes**	**No**		**Yes**
Multiple Task Activations	**No**	**Yes**		**No**	**Yes**	
Maximum number of tasks	1024	1024	1024	1024	1024	1024
Maximum number of not suspended tasks	1024	1024	1024	1024	1024	1024
Maximum number of priorities	1024	1024	1024	1024	1024	1024
Number of tasks per priority (for BCC2 and ECC2)	n/a	1024	1024	n/a	1024	1024
Upper limit for number of basic task activations per task priority	1	255	255	1	255	255
Maximum number of events per task	0	0	0	32	32	32
Limits for the number of alarm objects (per system / per task)	not limited by RTA-OSEK					
Limits for the number of standard resources (per system)	255	255	255	255	255	255
Limits for the number of internal resources (per system)	not limited by RTA-OSEK					
Limits for the number of nested resources (per system / per task)	255	255	255	255	255	255
Limits for the number of application modes	4294967295					

4.2 Hardware Resources

4.2.1 Reserved Hardware Resources

Timer units, interrupts, traps and other hardware resources are not reserved by RTA-OSEK.

Support

For product support, please contact your local ETAS representative.

Office locations and contact details can be found on the ETAS Group website www.etasgroup.com.