

RTA-OS PPCLWHVR/GHS V2.0.2

Port Guide

Status: Released



Copyright

The data in this document may not be altered or amended without special notification from ETAS GmbH. ETAS GmbH undertakes no further obligation in relation to this document. The software described in it can only be used if the customer is in possession of a general license agreement or single license. Using and copying is only allowed in concurrence with the specifications stipulated in the contract. Under no circumstances may any part of this document be copied, reproduced, transmitted, stored in a retrieval system or translated into another language without the express written permission of ETAS GmbH.

©Copyright 2008-2019 ETAS GmbH, Stuttgart.

The names and designations used in this document are trademarks or brands belonging to the respective owners.

Document: 10575-PG-2.0.2 EN-08-2019

Safety Notice

This ETAS product fulfills standard quality management requirements. If requirements of specific safety standards (e.g. IEC 61508, ISO 26262) need to be fulfilled, these requirements must be explicitly defined and ordered by the customer. Before use of the product, customer must verify the compliance with specific safety standards.

Contents

1	Introduction	7
1.1	About You	7
1.2	Document Conventions	8
1.3	References	8
2	Installing the RTA-OS Port Plug-in	9
2.1	Preparing to Install	9
2.1.1	Hardware Requirements	9
2.1.2	Software Requirements	9
2.2	Installation	10
2.2.1	Installation Directory	10
2.3	Licensing	11
2.3.1	Installing the ETAS License Manager	11
2.3.2	Licenses	12
2.3.3	Installing a Concurrent License Server	13
2.3.4	Using the ETAS License Manager	14
2.3.5	Troubleshooting Licenses	16
3	Verifying your Installation	19
3.1	Checking the Port	19
3.2	Running the Sample Applications	19
4	Port Characteristics	21
4.1	Parameters of Implementation	21
4.2	Configuration Parameters	21
4.2.1	Stack used for C-startup	21
4.2.2	Stack used when idle	22
4.2.3	Stack overheads for ISR activation	22
4.2.4	Stack overheads for ECC tasks	22
4.2.5	Stack overheads for ISR	22
4.2.6	ORTI/Lauterbach	23
4.2.7	ORTI/winIDEA	23
4.2.8	Link Type	23
4.2.9	SDA Threshold	23
4.2.10	Optimizer Setting	23
4.3	Generated Files	24
5	Port-Specific API	25
5.1	Macros	25
5.1.1	CAT1_ISR	25
5.2	Type Definitions	25
5.2.1	Os_StackSizeType	25
5.2.2	Os_StackValueType	25

6	Toolchain	26
6.1	Compiler Versions	26
6.1.1	Green Hills Software v2015.1.6	26
6.2	Options used to generate this guide	26
6.2.1	Compiler	26
6.2.2	Assembler	28
6.2.3	Librarian	28
6.2.4	Linker	29
6.2.5	Debugger	29
6.3	Specifying the location of LWHVR header files	29
7	Hardware	30
7.1	Supported Devices	30
7.2	Register Usage	30
7.2.1	Initialization	30
7.2.2	Modification	30
7.3	Required OS resources	30
7.4	Interrupts	31
7.4.1	Interrupt Priority Levels	31
7.4.2	Allocation of ISRs to Interrupt Vectors	31
7.4.3	Vector Table	31
7.4.4	Writing Category 1 Interrupt Handlers	32
7.4.5	Writing Category 2 Interrupt Handlers	32
7.4.6	Default Interrupt	32
7.5	Memory Model	32
7.6	Processor Modes	33
7.7	Stack Handling	33
7.8	Virtual Machine Details	33
7.8.1	Memory Assigned to the VM	33
7.8.2	VM Entry Points	33
7.8.3	Os_VMStatusBlock	34
7.8.4	Untrusted Applications in SC3 and SC4	34
7.8.5	Trusted with Protection	34
7.8.6	Using ORTI	34
7.8.7	iSystem Debugger Support	35
7.8.8	Compensating for the Effects of Time-Slicing	35
8	Performance	37
8.1	Measurement Environment	37
8.2	RAM and ROM Usage for OS Objects	37
8.3	Stack Usage	38
8.4	Library Module Sizes	38
8.5	Execution Time	41
8.5.1	Context Switching Time	42
9	Finding Out More	44

10	Contacting ETAS	45
10.1	Technical Support	45
10.2	General Enquiries	45
10.2.1	ETAS Global Headquarters	45
10.2.2	ETAS Local Sales & Support Offices	45

1 Introduction

RTA-OS is a small and fast real-time operating system that conforms to both the AUTOSAR OS (R3.0.1 -> R3.0.7, R3.1.1 -> R3.1.5, R3.2.1 -> R3.2.2, R4.0.1 -> R4.4.0) and OSEK/VDX 2.2.3 standards (OSEK is now standardized in ISO 17356). The operating system is configured and built on a PC, but runs on your target hardware.

This document describes the RTA-OS PPCLWHVR/GHS port plug-in that customizes the RTA-OS development tools for the ETAS LWHVR/SPC58ECxx with the GreenHills compiler. It supplements the more general information you can find in the *User Guide* and the *Reference Guide*.

The document has two parts. Chapters 2 to 3 help you understand the PPCLWHVR/GHS port and cover:

- how to install the PPCLWHVR/GHS port plug-in;
- how to configure PPCLWHVR/GHS-specific attributes;
- how to build an example application to check that the PPCLWHVR/GHS port plug-in works.

Chapters 4 to 8 provide reference information including:

- the number of OS objects supported;
- required and recommended toolchain parameters;
- how RTA-OS interacts with the LWHVR/SPC58ECxx, including required register settings, memory models and interrupt handling;
- memory consumption for each OS object;
- memory consumption of each API call;
- execution times for each API call.

For the best experience with RTA-OS it is essential that you read and understand this document.

1.1 About You

You are a trained embedded systems developer who wants to build real-time applications using a preemptive operating system. You should have knowledge of the C programming language, including the compilation, assembling and linking of C code for embedded applications with your chosen toolchain. Elementary knowledge about your target microcontroller, such as the start address, memory layout, location of peripherals and so on, is essential.

You should also be familiar with common use of the Microsoft Windows operating system, including installing software, selecting menu items, clicking buttons, navigating files and folders.

1.2 Document Conventions

The following conventions are used in this guide:

- | | |
|------------------------------------|--|
| Choose File > Open . | Menu options appear in bold, blue characters. |
| Click OK . | Button labels appear in bold characters |
| Press <Enter>. | Key commands are enclosed in angle brackets. |
| The "Open file" dialog box appears | GUI element names, for example window titles, fields, etc. are enclosed in double quotes. |
| Activate(Task1) | Program code, header file names, C type names, C functions and API call names all appear in a monospaced typeface. |
| See Section 1.2. | Internal document hyperlinks are shown in blue letters . |



Functionality in RTA-OS that might not be portable to other implementations of AUTOSAR OS is marked with the RTA-OS icon.



Important instructions that you must follow carefully to ensure RTA-OS works as expected are marked with a caution sign.

1.3 References

OSEK is a European automotive industry standards effort to produce open systems interfaces for vehicle electronics. OSEK is now standardized in ISO 17356. For details of the OSEK standards, please refer to:

<http://www.osek-vdx.org>

AUTOSAR (AUTomotive Open System ARchitecture) is an open and standardized automotive software architecture, jointly developed by automobile manufacturers, suppliers and tool developers. For details of the AUTOSAR standards, please refer to:

<http://www.autosar.org>

2 Installing the RTA-OS Port Plug-in

2.1 Preparing to Install

RTA-OS port plug-ins are supplied as a downloadable electronic installation image which you obtain from the ETAS Web Portal. You will have been provided with access to the download when you bought the port. You may optionally have requested an installation CD which will have been shipped to you. In either case, the electronic image and the installation CD contain identical content.



Integration Guidance 2.1: *You must have installed the RTA-OS tools before installing the PPCLWHVR/GHS port plug-in. If you have not yet done this then please follow the instructions in the Getting Started Guide.*

2.1.1 Hardware Requirements

You should make sure that you are using at least the following hardware before installing and using RTA-OS on a host PC:

- 1GHz Pentium Windows-capable PC.
- 2G RAM.
- 20G hard disk space.
- CD-ROM or DVD drive (Optional)
- Ethernet card.

2.1.2 Software Requirements

RTA-OS requires that your host PC has one of the following versions of Microsoft Windows installed:

- Windows 7
- Windows 8
- Windows 10



Integration Guidance 2.2: *The tools provided with RTA-OS require Microsoft's .NET Framework v2.0 (included as part of .NET Framework v3.5) and v4.0 to be installed. You should ensure that these have been installed before installing RTA-OS. The .NET framework is not supplied with RTA-OS but is freely available from <https://www.microsoft.com/net/download>. To install .NET 3.5 on Windows 10 see <https://docs.microsoft.com/en-us/dotnet/framework/install/dotnet-35-windows-10>.*

The migration of the code from v2.0 to v4.0 will occur over a period of time for performance and maintenance reasons.

2.2 Installation

Target port plug-ins are installed in the same way as the tools:

1. Either

- Double click the executable image; or
- Insert the RTA-OS PPCLWHVR/GHS CD into your CD-ROM or DVD drive.

If the installation program does not run automatically then you will need to start the installation manually. Navigate to the root directory of your CD/DVD drive and double click `autostart.exe` to start the setup.

2. Follow the on-screen instructions to install the PPCLWHVR/GHS port plug-in.

By default, ports are installed into `C:\ETAS\RTA-OS\Targets`. During the installation process, you will be given the option to change the folder to which RTA-OS ports are installed. You will normally want to ensure that you install the port plug-in in the same location that you have installed the RTA-OS tools. You can install different versions of the tools/targets into different directories and they will not interfere with each other.



Integration Guidance 2.3: *Port plug-ins can be installed into any location, but using a non-default directory requires the use of the `--target_include` argument to both `rtaosgen` and `rtaoscfg`. For example:*

```
rtaosgen --target_include:<target_directory>
```

2.2.1 Installation Directory

The installation will create a sub-directory under `Targets` with the name `PPCLWHVRGHS_2.0.2`. This contains everything to do with the port plug-in.

Each version of the port installs in its own directory - the trailing `_2.0.2` is the port's version identifier. You can have multiple different versions of the same port installed at the same time and select a specific version in a project's configuration.

The port directory contains:

PPCLWHVRGHS.dll - the port plug-in that is used by `rtaosgen` and `rtaoscfg`.

RTA-OS PPCLWHVRGHS Port Guide.pdf - the documentation for the port (the document you are reading now).

RTA-OS PPCLWHVRGHS Release Note.pdf - the release note for the port. This document provides information about the port plug-in release, including a list of changes from previous releases and a list of known limitations.

There may be other port-specific documentation supplied which you can also find in the root directory of the port installation. All user documentation is distributed in PDF format which can be read using Adobe Acrobat Reader. Adobe Acrobat Reader is not supplied with RTA-OS but is freely available from <http://www.adobe.com>.

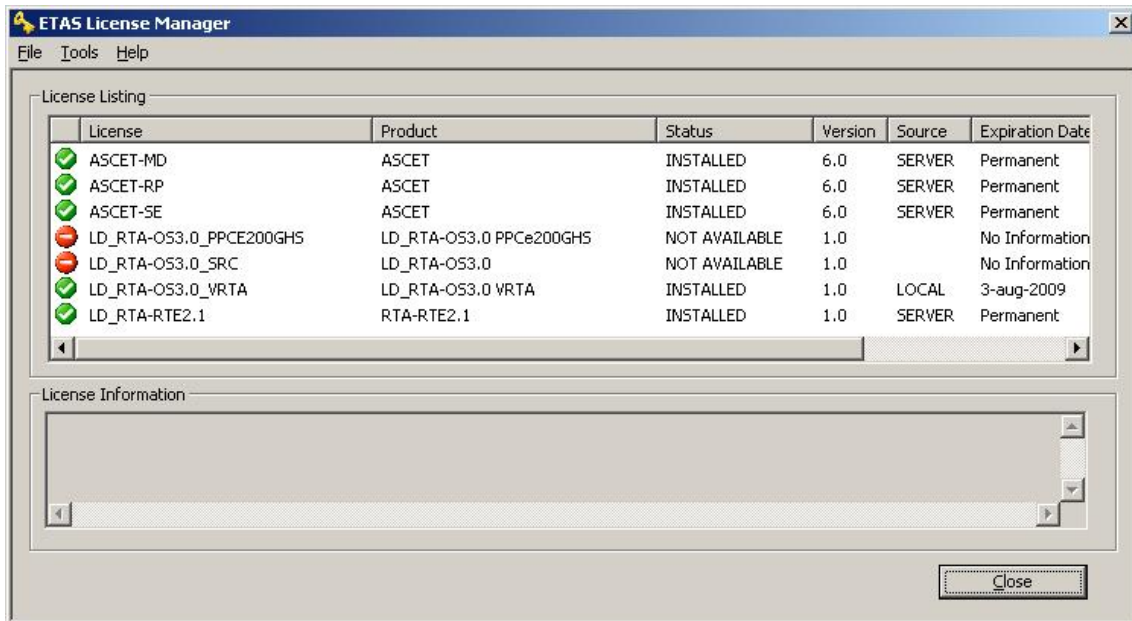


Figure 2.1: The ETAS License manager

2.3 Licensing

RTA-OS is protected by FLEXnet licensing technology. You will need a valid license key in order to use RTA-OS.

Licenses for the product are managed using the ETAS License Manager which keeps track of which licenses are installed and where to find them. The information about which features are required for RTA-OS and any port plug-ins is stored as license signature files that are stored in the folder <install_folder>\bin\Licenses.

The ETAS License Manager can also tell you key information about your licenses including:

- Which ETAS products are installed
- Which license features are required to use each product
- Which licenses are installed
- When licenses expire
- Whether you are using a local or a server-based license

Figure 2.1 shows the ETAS License Manager in operation.

2.3.1 Installing the ETAS License Manager



Integration Guidance 2.4: *The ETAS License Manager must be installed for RTA-OS to work. It is highly recommended that you install the ETAS License Manager during your installation of RTA-OS.*

The installer for the ETAS License Manager contains two components:

1. the ETAS License Manager itself;
2. a set of re-distributable FLEXnet utilities. The utilities include the software and instructions required to setup and run a FLEXnet license server manager if concurrent licenses are required (see Sections 2.3.2 and 2.3.3 for further details)

During the installation of RTA-OS you will be asked if you want to install the ETAS License Manager. If not, you can install it manually at a later time by running `<install_folder>\LicenseManager\LicensingStandaloneInstallation.exe`.

Once the installation is complete, the ETAS License Manager can be found in `C:\Program Files\Common Files\ETAS\Licensing`.

After it is installed, a link to the ETAS License Manager can be found in the Windows Start menu under **Programs → ETAS → License Management → ETAS License Manager**.

2.3.2 Licenses

When you install RTA-OS for the first time the ETAS License Manager will allow the software to be used in *grace mode* for 14 days. Once the grace mode period has expired, a license key must be installed. If a license key is not available, please contact your local ETAS sales representative. Contact details can be found in Chapter 10.

You should identify which type of license you need and then provide ETAS with the appropriate information as follows:

Machine-named licenses allows RTA-OS to be used by any user logged onto the PC on which RTA-OS and the machine-named license is installed.

A machine-named license can be issued by ETAS when you provide the host ID (Ethernet MAC address) of the host PC

User-named licenses allow the named user (or users) to use RTA-OS on any PC in the network domain.

A user-named license can be issued by ETAS when you provide the Windows user-name for your network domain.

Concurrent licenses allow any user on any PC up to a specified number of users to use RTA-OS. Concurrent licenses are sometimes called *floating* licenses because the license can *float* between users.

A concurrent license can be issued by ETAS when you provide the following information:

1. The name of the server
2. The Host ID (MAC address) of the server.
3. The TCP/IP port over which your FLEXnet license server will serve licenses. A default installation of the FLEXnet license server uses port 27000.



Figure 2.2: Obtaining License Information

You can use the ETAS License Manager to get the details that you must provide to ETAS when requesting a machine-named or user-named license and (optionally) store this information in a text file.

Open the ETAS License Manager and choose **Tools → Obtain License Info** from the menu. For machine-named licenses you can then select the network adaptor which provides the Host ID (MAC address) that you want to use as shown in Figure 2.2. For a user-based license, the ETAS License Manager automatically identifies the Windows username for the current user.

Selecting “Get License Info” tells you the Host ID and User information and lets you save this as a text file to a location of your choice.

2.3.3 Installing a Concurrent License Server

Concurrent licenses are allocated to client PCs by a FLEXnet license server manager working together with a vendor daemon. The vendor daemon for ETAS is called ETAS.exe. A copy of the vendor daemon is placed on disk when you install the ETAS License Manager and can be found in:

C:\Program Files\Common Files\ETAS\Licensing\Utility

To work with an ETAS concurrent license, a license server must be configured which is accessible from the PCs wishing to use a license. The server must be configured with the following software:

- FLEXnet license server manager;
- ETAS vendor daemon (ETAS.exe);

It is also necessary to install your concurrent license on the license server.

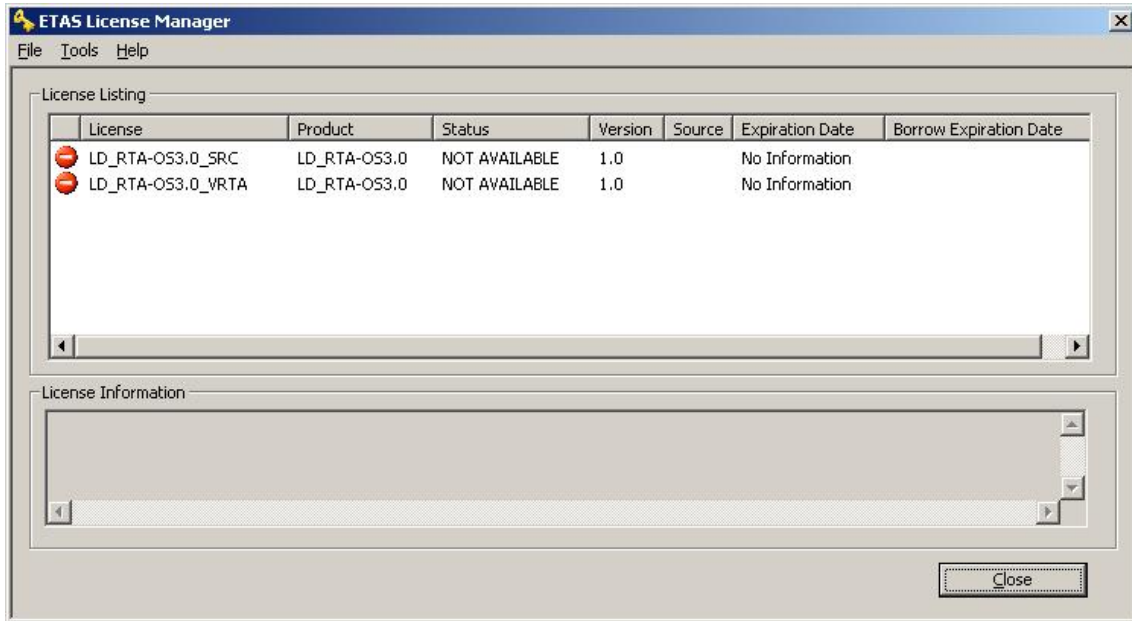


Figure 2.3: Unlicensed RTA-OS Installation

In most organizations there will be a single FLEXnet license server manager that is administered by your IT department. You will need to ask your IT department to install the ETAS vendor daemon and the associated concurrent license.

If you do not already have a FLEXnet license server then you will need to arrange for one to be installed. A copy of the FLEXnet license server, the ETAS vendor daemon and the instructions for installing and using the server (LicensingEndUserGuide.pdf) are placed on disk when you install the ETAS License manager and can be found in:

C:\Program Files\Common Files\ETAS\Licensing\Utility

2.3.4 Using the ETAS License Manager

If you try to run the RTA-OS GUI **rtaoscfg** without a valid license, you will be given the opportunity to start the ETAS License Manager and select a license. (The command-line tool **rtaosgen** will just report the license is not valid.)

When the ETAS License Manager is launched, it will display the RTA-OS license state as NOT AVAILABLE. This is shown in Figure 2.3.

Note that if the ETAS License Manager window is slow to start, **rtaoscfg** may ask a second time whether you want to launch it. You should ignore the request until the ETAS License Manager has opened and you have completed the configuration of the licenses. You should then say yes again, but you can then close the ETAS License Manager and continue working.

License Key Installation

License keys are supplied in an ASCII text file, which will be sent to you on completion of a valid license agreement.

If you have a machine-based or user-based license key then you can simply install the license by opening the ETAS License Manager and selecting **File → Add License File** menu.

If you have a concurrent license key then you will need to create a license stub file that tells the client PC to look for a license on the FLEXnet server as follows:

1. create a copy of the concurrent license file
2. open the copy of the concurrent license file and delete every line *except* the one starting with SERVER
3. add a new line containing USE_SERVER
4. add a blank line
5. save the file

The file you create should look something like this:

```
SERVER <server name> <MAC address> <TCP/IP Port>¶  
USE_SERVER¶  
¶
```

Once you have create the license stub file you can install the license by opening the ETAS License Manager and selecting **File → Add License File** menu and choosing the license stub file.

License Key Status

When a valid license has been installed, the ETAS License Manager will display the license version, status, expiration date and source as shown in Figure 2.4.

Borrowing a concurrent license

If you use a concurrent license and need to use RTA-OS on a PC that will be disconnected from the network (for example, you take a demonstration to a customer site), then the concurrent license will not be valid once you are disconnected.

To address this problem, the ETAS License Manager allows you to temporarily borrow a license from the license server.

To borrow a license:

1. Right click on the license feature you need to borrow.
2. Select "Borrow License"
3. From the calendar, choose the date that the borrowed license should expire.
4. Click "OK"

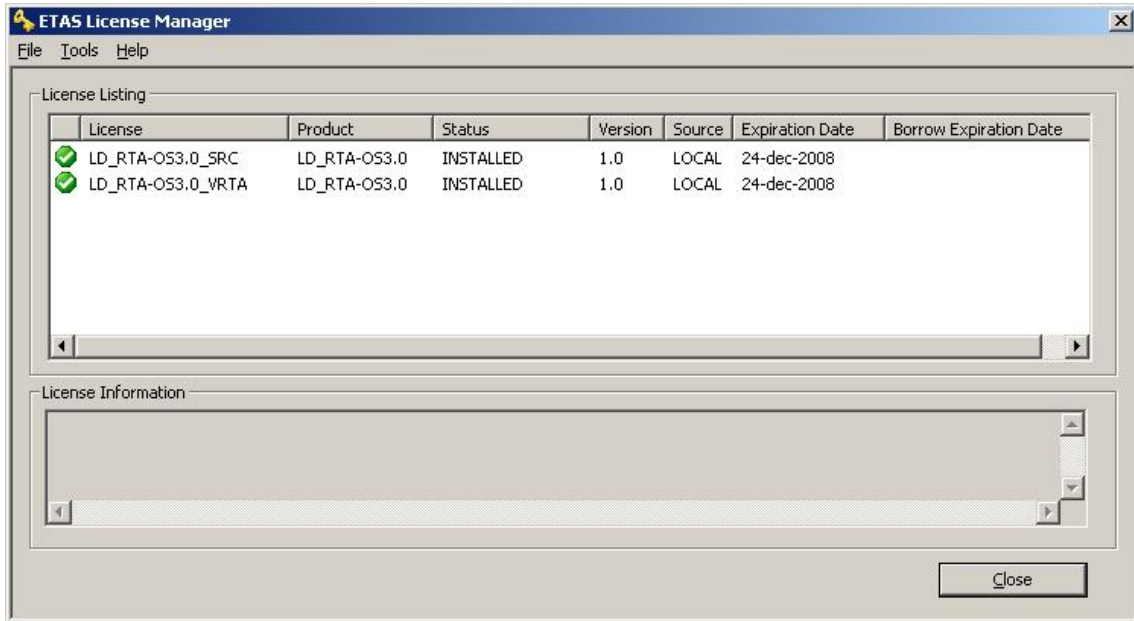


Figure 2.4: Licensed features for RTA-OS

The license will automatically expire when the borrow date elapses. A borrowed license can also be returned before this date. To return a license:

1. Reconnect to the network;
2. Right-click on the license feature you have borrowed;
3. Select "Return License".

2.3.5 Troubleshooting Licenses

RTA-OS tools will report an error if you try to use a feature for which a correct license key cannot be found. If you think that you should have a license for a feature but the RTA-OS tools appear not to work, then the following troubleshooting steps should be followed before contacting ETAS:

Can the ETAS License Manager see the license?

The ETAS License Manager must be able to see a valid license key for each product or product feature you are trying to use.

You can check what the ETAS License Manager can see by starting it from the **Help → License Manager...** menu option in **rtaoscfg** or directly from the Windows Start Menu - **Start → ETAS → License Management → ETAS License Manager**.

The ETAS License Manager lists all license features and their status. Valid licenses have status **INSTALLED**. Invalid licenses have status **NOT AVAILABLE**.

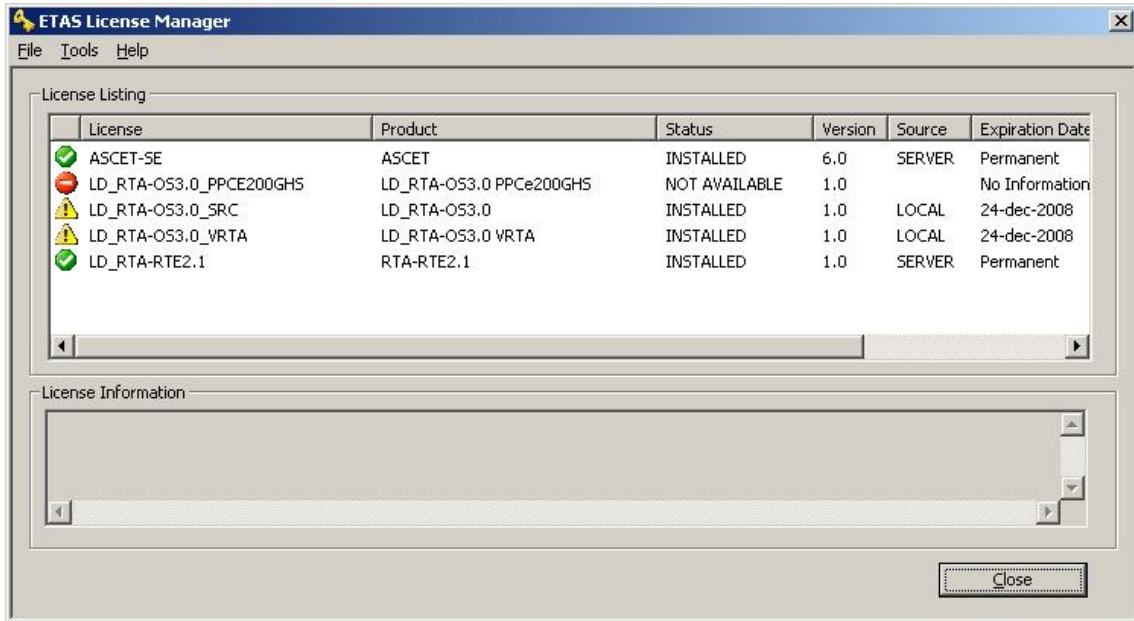


Figure 2.5: Licensed features that are due to expire

Is the license valid?

You may have been provided with a time-limited license (for example, for evaluation purposes) and the license may have expired. You can check that the Expiration Date for your licensed features to check that it has not elapsed using the ETAS License Manager.

If a license is due to expire within the next 30 days, the ETAS License Manager will use a warning triangle to indicate that you need to get a new license. Figure 2.5 shows that the license features LD_RTA-OS3.0_VRTA and LD_RTA-OS3.0_SRC are due to expire.

If your license has elapsed then please contact your local ETAS sales representative to discuss your options.

Does the Ethernet MAC address match the one specified?

If you have a machine based license then it is locked to a specific MAC address. You can find out the MAC address of your PC by using the ETAS License Manager (**Tools → Obtain License Info**) or using the Microsoft program **ipconfig /all** at a Windows Command Prompt.

You can check that the MAC address in your license file by opening your license file in a text editor and checking that the HOSTID matches the MAC address identified by the ETAS License Manager or the *Physical Address* reported by **ipconfig /all**.

If the HOSTID in the license file (or files) does not match your MAC address then you do not have a valid license for your PC. You should contact your local ETAS sales representative to discuss your options.

Is your Ethernet Controller enabled?

If you use a laptop and RTA-OS stops working when you disconnect from the network then you should check your hardware settings to ensure that your Ethernet controller is not turned off to save power when a network connection is not present. You can do this using Windows Control Panel. Select **System → Hardware → Device Manager** then select your Network Adapter. Right click to open **Properties** and check that the Ethernet controller is not configured for power saving in **Advanced** and/or **Power Management** settings.

Is the FlexNet License Server visible?

If your license is served by a FlexNet license server, then the ETAS License Manager will report the license as NOT AVAILABLE if the license server cannot be accessed.

You should contact your IT department to check that the server is working correctly.

Still not fixed?

If you have not resolved your issues, after confirming these points above, please contact ETAS technical support. The contact address is provided in Section [10.1](#). You must provide the contents and location of your license file and your Ethernet MAC address.

3 Verifying your Installation

Now that you have installed both the RTA-OS tools and a port plug-in and have obtained and installed a valid license key you can check that things are working.

3.1 Checking the Port

The first thing to check is that the RTA-OS tools can see the new port. You can do this in two ways:

1. use the **rtaosgen** tool

You can run the command **rtaosgen --target:?** to get a list of available targets, the versions of each target and the variants supported, for example:

```
RTA-OS Code Generator
Version p.q.r.s, Copyright © ETAS nnnn
Available targets:
  TriCoreHighTec_n.n.n [TC1797...]
  VRTA_n.n.n [MinGW,VS2005,VS2008,VS2010]
```

2. use the **rtaoscfg** tool

The second way to check that the port plug-in can be seen is by starting **rtaoscfg** and selecting **Help → Information...** drop down menu. This will show information about your complete RTA-OS installation and license checks that have been performed.



Integration Guidance 3.1: *If the target port plug-ins have been installed to a non-default location, then the `--target_include` argument must be used to specify the target location.*

If the tools can see the port then you can move on to the next stage – checking that you can build an RTA-OS library and use this in a real program that will run on your target hardware.

3.2 Running the Sample Applications

Each RTA-OS port is supplied with a set of sample applications that allow you to check that things are running correctly. To generate the sample applications:

1. Create a new *working* directory in which to build the sample applications.
2. Open a Windows command prompt in the new directory.
3. Execute the command:

```
rtaosgen --target:<your target> --samples:[Applications]
```

e.g.

```
rtaosgen --target:[MPC5777Mv2]PPCe200HighTec_5.0.8
--samples:[Applications]
```

You can then use the build.bat and run.bat files that get created for each sample application to build and run the sample. For example:

```
cd Samples\Applications\HelloWorld
build.bat
run.bat
```

Remember that your target toolchain must be accessible on the Windows PATH for the build to be able to run successfully.



Integration Guidance 3.2: *It is strongly recommended that you build and run at least the Hello World example in order to verify that RTA-OS can use your compiler toolchain to generate an OS kernel and that a simple application can run with that kernel.*

For further advice on building and running the sample applications, please consult your *Getting Started Guide*.

4 Port Characteristics

This chapter tells you about the characteristics of RTA-OS for the PPCLWHVR/GHS port.

4.1 Parameters of Implementation

To be a valid OSEK (ISO 17356) or AUTOSAR OS, an implementation must support a minimum number of OS objects. The following table specifies the *minimum* numbers of each object required by the standards and the *maximum* number of each object supported by RTA-OS for the PPCLWHVR/GHS port.

Parameter	Required	RTA-OS
Tasks	16	1024
Tasks not in SUSPENDED state	16	1024
Priorities	16	1024
Tasks per priority	-	1024
Queued activations per priority	-	4294967296
Events per task	8	32
Software Counters	8	4294967296
Hardware Counters	-	4294967296
Alarms	1	4294967296
Standard Resources	8	4294967296
Linked Resources	-	4294967296
Nested calls to GetResource()	-	4294967296
Internal Resources	2	no limit
Application Modes	1	4294967296
Schedule Tables	2	4294967296
Expiry Points per Schedule Table	-	4294967296
OS Applications	-	4294967296
Trusted functions	-	4294967296
Spinlocks (multicore)	-	4294967296
Register sets	-	4294967296

4.2 Configuration Parameters

Port-specific parameters are configured in the **General** → **Target** workspace of **rtaoscfg**, under the “Target-Specific” tab.

The following sections describe the port-specific configuration parameters for the PPCLWHVR/GHS port, the name of the parameter as it will appear in the XML configuration and the range of permitted values (where appropriate).

4.2.1 Stack used for C-startup

XML name SpPreStartOS

Description

The amount of stack already in use at the point that StartOS() is called. This value is simply added to the total stack size that the OS needs to support all tasks and interrupts at run-time. Typically you use this to obtain the amount of stack that the linker must allocate. The value does not normally change if the OS configuration changes.

4.2.2 Stack used when idle

XML name SpStartOS

Description

The amount of stack used when the OS is in the idle state (typically inside Os_Cbk_Idle()). This is just the difference between the stack used at the point that Os_StartOS() is called and the stack used when no task or interrupt is running. This can be zero if Os_Cbk_Idle() is not used. It must include the stack used by any function called while in the idle state. The value does not normally change if the OS configuration changes.

4.2.3 Stack overheads for ISR activation

XML name SpIDisp

Description

The extra amount of stack needed to activate a task from within an ISR. If a task is activated within a Category 2 ISR, and that task has a higher priority than any currently running task, then for some targets the OS may need to use marginally more stack than if it activates a task that is of lower priority. This value accounts for that. On most targets this value is zero. This value is used in worst-case stack size calculations. The value may change if significant changes are made to the OS configuration. e.g. STANDARD/EXTENDED, SC1/2/3/4.

4.2.4 Stack overheads for ECC tasks

XML name SpECC

Description

The extra amount of stack needed to start an ECC task. ECC tasks need to save slightly more state on the stack when they are started than BCC tasks. This value contains the difference. The value may change if significant changes are made to the OS configuration. e.g. STANDARD/EXTENDED, SC1/2/3/4.

4.2.5 Stack overheads for ISR

XML name SpPreemption

Description

The amount of stack used to service a Category 2 ISR. When a Category 2 ISR interrupts a task, it usually places some data on the stack. If the ISR measures the stack to determine if the preempted task has exceeded its stack budget, then it will overestimate the stack usage unless this value is subtracted from the measured size. The value is also used when calculating the worst-case stack usage of the system. Be careful to set this value accurately. If its value is too high then when the subtraction occurs, 32-bit underflow can occur and cause the OS to think that a budget overrun has been detected. The value may change if significant changes are made to the OS configuration. e.g. STANDARD/EXTENDED, SC1/2/3/4.

4.2.6 ORTI/Lauterbach

XML name Orti22Lauterbach

Description

Enables ORTI generation for Lauterbach debugger.

4.2.7 ORTI/winIDEA

XML name Orti21winIDEA

Description

Enables ORTI generation for winIDEA debugger.

4.2.8 Link Type

XML name OSLinkMemModel

Description

Select the type of map used in linker samples.

Settings

Value	Description
IntRAM	Code/data in internal RAM (default)
IntFLASH	Code in internal flash, data in internal RAM

4.2.9 SDA Threshold

XML name sda_value

Description

Sets the value used for small data objects when compiling. Defaults to zero.

4.2.10 Optimizer Setting

XML name optimizer_setting

Description

Controls the optimizer strategy compiler option (see the compiler documentation for more details).

Settings

Value	Description
Osize	Improve code size over performance
Ospeed	Improve code performance over size (default)

4.3 Generated Files

The following table lists the files that are generated by **rtaosgen** for all ports:

Filename	Contents
Os.h	The main include file for the OS.
Os_Cfg.h	Declarations of the objects you have configured. This is included by Os.h.
Os_MemMap.h	AUTOSAR memory mapping configuration used by RTA-OS to merge with the system-wide MemMap.h file in AUTOSAR versions 4.0 and earlier. From AUTOSAR version 4.1, Os_MemMap.h is used by the OS instead of MemMap.h.
RTAOS.<lib>	The RTA-OS library for your application. The extension <lib> depends on your target.
RTAOS.<lib>.sig	A signature file for the library for your application. This is used by rtaosgen to work out which parts of the kernel library need to be rebuilt if the configuration has changed. The extension <lib> depends on your target.
<projectname>.log	A log file that contains a copy of the text that the tool and compiler sent to the screen during the build process.

5 Port-Specific API

The following sections list the port-specific aspects of the RTA-OS programmers reference for the PPCLWHVR/GHS port that are provided either as:

- additions to the material that is documented in the *Reference Guide*; or
- overrides for the material that is documented in the *Reference Guide*. When a definition is provided by both the *Reference Guide* and this document, the definition provided in this document takes precedence.

5.1 Macros

5.1.1 CAT1_ISR

Macro that should be used to create a Category 1 ISR entry function. This macro exists to help make your code portable between targets.

Example

```
CAT1_ISR(MyISR) {...}
```

5.2 Type Definitions

5.2.1 Os_StackSizeType

An unsigned value representing an amount of stack in bytes.

Example

```
Os_StackSizeType stack_size;  
stack_size = Os_GetStackSize(start_position, end_position);
```

5.2.2 Os_StackValueType

An unsigned value representing the position of the stack pointer.

Example

```
Os_StackValueType start_position;  
start_position = Os_GetStackValue();
```

6 Toolchain

This chapter contains important details about RTA-OS and the GreenHills toolchain. A port of RTA-OS is specific to both the target hardware and a specific version of the compiler toolchain. You must make sure that you build your application with the supported toolchain.

In addition to the version of the toolchain, RTA-OS may use specific tool options (switches). The options are divided into three classes:

kernel options are those used by **rtaosgen** to build the RTA-OS kernel.

mandatory options must be used to build application code so that it will work with the RTA-OS kernel.

forbidden options must not be used to build application code.

Any options that are not explicitly forbidden can be used by application code providing that they do not conflict with the kernel and mandatory options for RTA-OS.

Integration Guidance 6.1: *ETAS has developed and tested RTA-OS using the tool versions and options indicated in the following sections. Correct operation of RTA-OS is only covered by the warranty in the terms and conditions of your deployment license agreement when using identical versions and options. If you choose to use a different version of the toolchain or an alternative set of options then it is your responsibility to check that the system works correctly. If you require a statement that RTA-OS works correctly with your chosen tool version and options then please contact ETAS to discuss validation possibilities.*



6.1 Compiler Versions

This port of RTA-OS has been developed to work with the following compiler(s):

6.1.1 Green Hills Software v2015.1.6

Ensure that ccppc.exe is on the path.

Tested on Green Hills Software, MULTI v2015.1.6.

If you require support for a compiler version not listed above, please contact ETAS.

6.2 Options used to generate this guide

6.2.1 Compiler

Name ecomppc.exe

Version Green Hills Software, Compiler v2015.1.6

Options

Kernel Options

The following options were used to build the RTA-OS kernel for the configuration that was used to generate the performance figures in this document. If you select different target options, then the values used to build the kernel might change. You can run a Configuration Summary report to check the values used for your configuration.

- checksum** Append 4-byte checksum to initialized program sections
- cpu=ppc5746mz420** Generate code for target processor
- delete** Remove functions that are unused and unreferenced
- dwarf2** Use DWARF2 debug format
- farcallpatch** Allow the linker to insert code for far calls between modules
- floatsingle** Treat double type as float
- g** Generate source-level debugging information
- inline_prologue** Inline code sequences in prologue and epilogue
- no_commons** Zeros uninitialized global variables
- ospeed** Optimizer strategy (value set by target option)
- omax** Maximum optimization
- preprocess_assembly_files** Preprocess .s/.asm files
- sda=0** SDA threshold (value set by target option)
- vle** Enable VLE code generation
- prototype_errors** Reports an error for functions with no prototype
- quit_after_warnings** Treat all warnings as errors

Mandatory Options for Application Code

The following options were mandatory for application code used with the configuration that was used to generate the performance figures in this document. If you select different target options, then the values required by application code might change. You can run a Configuration Summary report to check the values used for your configuration.

- The same options as for kernel compilation with the exception of the following which may be omitted from application code:
 - prototype_errors
 - quit_after_warnings

Forbidden Options for Application Code

The following options were forbidden for application code used with the configuration that was used to generate the performance figures in this document. If you select different target options, then the forbidden values might change. You can run a Configuration Summary report to check the values used for your configuration.

- Any options that conflict with kernel options

6.2.2 Assembler

Name asppc.exe
Version Green Hills Software, Compiler v2015.1.6

Options

Kernel Options

The following options were used to build the RTA-OS kernel for the configuration that was used to generate the performance figures in this document. If you select different target options, then the values used to build the kernel might change. You can run a Configuration Summary report to check the values used for your configuration.

- The same options as for compilation

Mandatory Options for Application Code

The following options were mandatory for application code used with the configuration that was used to generate the performance figures in this document. If you select different target options, then the values required by application code might change. You can run a Configuration Summary report to check the values used for your configuration.

- The same options as for compilation

Forbidden Options for Application Code

The following options were forbidden for application code used with the configuration that was used to generate the performance figures in this document. If you select different target options, then the forbidden values might change. You can run a Configuration Summary report to check the values used for your configuration.

- Any options that conflict with kernel options

6.2.3 Librarian

Name ax.exe
Version Green Hills Software, Compiler v2015.1.6

6.2.4 Linker

Name elxr.exe
Version Green Hills Software, Compiler v2015.1.6

Options

Kernel Options

The following options were used to build the RTA-OS kernel for the configuration that was used to generate the performance figures in this document. If you select different target options, then the values used to build the kernel might change. You can run a Configuration Summary report to check the values used for your configuration.

- cpu=ppc5746mz420 Generate code for target processor
- vle Enable VLE code generation
- lnk="-Manx -v -Qn" Create map file, verbose, skip comment section

Mandatory Options for Application Code

The following options were mandatory for application code used with the configuration that was used to generate the performance figures in this document. If you select different target options, then the values required by application code might change. You can run a Configuration Summary report to check the values used for your configuration.

- The same options as for compilation

Forbidden Options for Application Code

The following options were forbidden for application code used with the configuration that was used to generate the performance figures in this document. If you select different target options, then the forbidden values might change. You can run a Configuration Summary report to check the values used for your configuration.

- Any options that conflict with kernel options

6.2.5 Debugger

Name Lauterbach TRACE32
Version Build 69577 or later

6.3 Specifying the location of LWHVR header files

In order to compile RTA-OS, rtaosgen.exe must be able include the header files that define the API available to an LWHVR virtual machine (VM). Therefore when rtaosgen.exe is invoked it must be given the argument -I<LWHVR-Loc> where <LWHVR-Loc> is the path of the directory containing the LWHVR header files.

7 Hardware

7.1 Supported Devices

This port of RTA-OS has been developed to work with the following target:

Name: ETAS
Device: LWHVR/SPC58ECxx

The following variants of the LWHVR/SPC58ECxx are supported:

- SPC58ECXX

If you require support for a variant of LWHVR/SPC58ECxx not listed above, please contact ETAS.

7.2 Register Usage

7.2.1 Initialization

RTA-OS requires the following registers to be initialized to the indicated values before StartOS() is called.

Register	Setting
R1	Before StartOS() is called the R1 register (stack-pointer) must be initialized to point to the start of the stack. The address put in R1 must be 8-byte aligned.

7.2.2 Modification

The following registers must not be modified by user code after the call to StartOS():

Register	Notes
All supervisor-mode registers	Since all code runs inside a LWHVR virtual-machine user-code may not access any registers that can only be accessed in supervisor-mode.

7.3 Required OS resources

RTA-OS needs the following resources for correct operation.

Resource	Description
Pseudo-interrupt handler	Executing code at the pseudo-interrupt handler address configured for the virtual-machine running RTA-OS must result in Os_PsIntHandler() being executed. For example the instruction at the configured pseudo-interrupt handler address could be e_b Os_PsIntHandler

Since all code is running inside an LWHVR virtual machine only registers accessible in user-mode may be read or written. Attempts to read or write registers only accessible in supervisor-mode will result in the virtual machine entering the error state and being forcibly stopped.

7.4 Interrupts

This section explains the implementation of RTA-OS's interrupt model on the LWHVR/SPC58ECxx.

7.4.1 Interrupt Priority Levels

Interrupts execute at an interrupt priority level (IPL). RTA-OS standardizes IPLs across all targets. IPL 0 indicates task level. IPL 1 and higher indicate an interrupt priority. It is important that you don't confuse IPLs with task priorities. An IPL of 1 is higher than the highest task priority used in your application.

The IPL is a target-independent description of the interrupt priority on your target hardware. The following table shows how IPLs are mapped onto the hardware interrupt priorities of the LWHVR/SPC58ECxx:

IPL	psIntEnabled	Description
0	0x0000	User (task) level. No interrupts are masked.
1-32		Category 1 and 2 level. The value of <code>Os_VMStatusBlock.psIntEnabled</code> at <code>IPL=n</code> is calculated by RTA-OS to mask out any vectors assigned an IPL equal to or lower than n.

Even though a particular mapping is permitted, all Category 1 ISRs must have equal or higher IPL than all of your Category 2 ISRs.

The LWHVR interrupt vectors do not have a fixed priority. Any vector may be assigned any IPL in the range 1 to 32 (inclusive). RTA-OS implements IPLs by setting the `Os_VMStatusBlock.psIntEnabled` mask. The value of `Os_VMStatusBlock.psIntEnabled` at `IPL=n` is calculated by RTA-OS to mask out any vectors assigned an IPL equal to or lower than n.

7.4.2 Allocation of ISRs to Interrupt Vectors

The following restrictions apply for the allocation of Category 1 and Category 2 interrupt service routines (ISRs) to interrupt vectors on the LWHVR/SPC58ECxx. A ✓ indicates that the mapping is permitted and a ✗ indicates that it is not permitted:

Address	Category 1	Category 2
Vectors 0-31	✓	✓

7.4.3 Vector Table

rtaosgen normally generates an interrupt vector table for you automatically. You can configure "Suppress Vector Table Generation" as `true` to stop RTA-OS from generating the interrupt vector table.

Depending upon your target, you may be responsible for locating the generated vector table at the correct base address. The following table shows the section (or sections) that need to be located and the associated valid base address:

Section	Valid Addresses
N/A	The interrupt vector table is called <code>0s_VectorTable</code> and is within the AUTOSAR memory region <code>0S_SEC_VECTORS</code> . This table may be located anywhere in the memory assigned to the LWHVR virtual-machine.

7.4.4 Writing Category 1 Interrupt Handlers

Raw Category 1 interrupt service routines (ISRs) must correctly handle the interrupt context themselves. RTA-OS provides an optional helper macro `CAT1_ISR` that can be used to make code more portable. Depending on the target, this may cause the selection of an appropriate interrupt control directive to indicate to the compiler that a function requires additional code to save and restore the interrupt context.

A Category 1 ISR therefore has the same structure as a Category 2 ISR, as shown below.

```
CAT1_ISR(Category1Handler) {
    /* Handler routine */
}
```

7.4.5 Writing Category 2 Interrupt Handlers

Category 2 ISRs are provided with a C function context by RTA-OS, since the RTA-OS kernel handles the interrupt context itself. The handlers are written using the `ISR()` macro as shown below:

```
#include <0s.h>
ISR(MyISR) {
    /* Handler routine */
}
```

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

7.4.6 Default Interrupt

The 'default interrupt' is intended to be used to catch all unexpected interrupts. All unused interrupts have their interrupt vectors directed to the named routine that you specify. The routine you provide is not handled by RTA-OS and must correctly handle the interrupt context itself. The handler must use the `CAT1_ISR` macro in the same way as a Category 1 ISR (see Section 7.4.4 for further details).

7.5 Memory Model

The following memory models are supported:

Model	Description
Flat 32-bit address space	All code and data is addressed using a 32-bit address

7.6 Processor Modes

RTA-OS can run in the following processor modes:

Mode	Notes
User	All code runs inside a LWHVR virtual machine and therefore is in user-mode.

Both the RTA-OS code and the user code is running inside a LWHVR virtual machine. Therefore the processor is always running in user-mode.

7.7 Stack Handling

RTA-OS uses a single stack for all tasks and ISRs.

RTA-OS manages the stack (via register R1).

7.8 Virtual Machine Details

The RTA-OS library generated by this port expects to run inside a LWHVR virtual-machine (VM). This section describes special steps that must be taken as a result of RTA-OS running inside a VM.

7.8.1 Memory Assigned to the VM

Please ensure that only memory (flash and RAM) that has been assigned to the VM in the LWHVR configuration is used when linking RTA-OS and user code/data. If the VM attempts to access memory that has not been assigned to it, the LWHVR will abort execution of the VM and call the LWHVR callback `LWHVR_VMErrorCallback`.

7.8.2 VM Entry Points

When the LWHVR starts a VM it executes the code at the VM's entry-point. The code at the entry-point is responsible for setting up the stack and initializing data. The address of each VM's entry point is specified in the LWHVR configuration. Typically the entry-point will be the `_start` symbol in the compiler's C start-up code.

When the LWHVR injects a interrupt into a VM it executes the code at the VM's interrupt handler. The address of each VM's interrupt handler is specified in the LWHVR configuration. For RTA-OS the interrupt handler is the code at `0s_PsIntHandler`.

Obviously the location of `_start` and `0s_PsIntHandler` can change depending on details of the code and data linked to create the executable. The suggested way of dealing with this problem is to locate a short piece of assembly code at the start of the code memory assigned to the VM. This assembly code contains a jump to `_start` and a jump to `0s_PsIntHandler`, thus providing an alternative entry-point and interrupt handler for the VM. Since the assembly code is at a fixed location in memory the alternative entry-point and interrupt handler addresses can be easily determined.

For example, the assembly used in the sample applications is called `VMEntryPoints.s` and contains the following:

```

file      "VMEntryPoints.s"
section   ".VMEntryPoints", "vax"
vle

global   VMEntryPoints

VMEntryPoints:
    e_b _start
    e_b Os_PsIntHandler
    
```

If we locate section `.VMEntryPoints` at the start of the code memory assigned to the VM in the LWHVR configuration then we always know addresses that may be used for the VM's entry-point and interrupt handler. E.g. if the code memory assigned to a VM starts at address `0x1040000` then in the LWHVR configuration for the VM we can set the entry-point address to `0x1040000` and the interrupt handler address to `0x1040004`.

7.8.3 `Os_VMStatusBlock`

Each VM has a status block used to communicate with the LWHVR. RTA-OS creates a VM status block called `Os_VMStatusBlock` in section `.Os_VMStatusBlock`. Section `.Os_VMStatusBlock` must be located in RAM by the linker script. The address of a VM's status block must be specified in the LWHVR configuration for the VM. The easiest way to do this is to locate `.Os_VMStatusBlock` at the start of the RAM assigned to the VM.

7.8.4 Untrusted Applications in SC3 and SC4

Since a VM is always running in user-mode it is not possible for either RTA-OS or user code to modify the memory protection unit (MPU) configuration. Therefore although it is possible to configure untrusted applications in SC3 and SC4 it is not possible to apply memory access restrictions (beyond those enforced by the VM) to the untrusted applications.

7.8.5 Trusted with Protection

Since a VM is always running in user-mode it is not possible for either RTA-OS or user code to modify the memory protection unit (MPU) configuration. Therefore although it is possible to configure trusted with protection applications it is not possible to apply memory access restrictions (beyond those enforced by the VM) to the trusted with protection applications.

7.8.6 Using ORTI

ORTI relies on being able to monitor the values of some internal RTA-OS variables. When ORTI is being used to help debug an application running on RTA-OS in a LWHVR VM ORTI must monitor the variables for the copy of RTA-OS in the VM, not the variables in the copy of RTA-OS running on the master core or the copy of RTA-OS running in a different VM. Therefore if you wish to use ORTI to help debug an RTA-OS application running in a VM, ensure that only symbol information for that VM is loaded. If you are using a Lauterbach debugger this can be done with a command like:

```
Data.Load VM0.elf /NOCODE /GHS
```

This will delete any symbol information already loaded and then load the symbol information from the executable VM0.elf.

7.8.7 iSystem Debugger Support

If ORTI support is enabled for the iSystem winIDEA debugger then RTA-OS writes a value to the PPC DDAM register to indicate to the iSystem debugger when a task or category 2 ISR is dispatched and which tasks are in a ready state. Please consult iSystem for information about the debugger extensions need to make use of this support.

7.8.8 Compensating for the Effects of Time-Slicing

An LWHVR VM is time-sliced and therefore is not executing all of the time. An application running on RTA-OS that needs to keep synchronized with "wall-clock" time may need to take steps to compensate for the effects of time-slicing.

Consider an application that increments a counter every time it receives a clock-tick interrupt. If the LWHVR receives a clock-tick interrupt every 1ms then the RTA-OS application will receive a clock-tick interrupt every 1ms while its VM is running in a time-slice. The application will not see clock-tick interrupts when its VM is not running in a time-slice. Imagine that the counter were being used to activate a task every every 10 increments of the counter. If the VM only runs for half of the time then the task would only be activated every 20ms not every 10ms as we would expect/require.

An application can use the VM status block field `0s_VMStatusBlock.ticksSinceStart` to compensate for this effect of time-slicing. `0s_VMStatusBlock.ticksSinceStart` contains the number of clock-ticks that have elapsed since the VM started.

Consider a typical ISR used to increment an RTA-OS counter on every clock-tick interrupt:

```
void EnableInterrupt(void)
{
    0s_VMStatusBlock.psIntEnabled |= LWHVR_PS_INT_MASK_TIMER0;
    0s_VMStatusBlock.psIntGenerateOnTick |= LWHVR_PS_INT_MASK_TIMER0;
}

ISR(Millisecond)
{
    IncrementCounter(MillisecondCounter);
}
```

To compensate for the effects of time-slicing we can adapt this code as follows:

```
LWHVR_UInt32Type LastSeenTick;

void EnableInterrupt(void)
{
    LastSeenTick = 0s_VMStatusBlock.ticksSinceStart;
    0s_VMStatusBlock.psIntEnabled |= LWHVR_PS_INT_MASK_TIMER0;
```

```
    Os_VMStatusBlock.psIntGenerateOnTick |= LWHVR_PS_INT_MASK_TIMER0;
}

ISR(Millisecond)
{
    while (LastSeenTick != Os_VMStatusBlock.ticksSinceStart)
    {
        LastSeenTick++;
        IncrementCounter(MillisecondCounter);
    }
}
```

8 Performance

This chapter provides detailed information on the functionality, performance and memory demands of the RTA-OS kernel. RTA-OS is highly scalable. As a result, different figures will be obtained when your application uses different sets of features. The figures presented in this chapter are representative for the PPCLWHVR/GHS port based on the following configuration:

- There are 32 tasks in the system
- Standard build is used
- Stack monitoring is disabled
- Time monitoring is disabled
- There are no calls to any hooks
- Tasks have unique priorities
- Tasks are not queued (i.e. tasks are BCC1 or ECC1)
- All tasks terminate/wait in their entry function
- Tasks and ISRs do not save any auxiliary registers (for example, floating point registers)
- Resources are shared by tasks only
- The generation of the resource RES_SCHEDULER is disabled

8.1 Measurement Environment

The following hardware environment was used to take the measurements in this chapter:

Device	SPC58ECXX on SPC57xxMB
CPU Clock Speed	180.0MHz
Stopwatch Speed	90.0MHz

8.2 RAM and ROM Usage for OS Objects

Each OS object requires some ROM and/or RAM. The OS objects are generated by **rtaosgen** and placed in the RTA-OS library. In the main:

- 0s_Cfg_Counters includes data for counters, alarms and schedule tables.
- 0s_Cfg contains the data for most other OS objects.

The following table gives the ROM and/or RAM requirements (in bytes) for each OS object in a simple configuration. Note that object sizes will vary depending on the project configuration and compiler packing issues.

Object	ROM	RAM
Alarm	2	12
Cat 2 ISR	8	0
Counter	20	4
CounterCallback	4	0
ExpiryPoint	3.5	0
OS Overheads (max)	0	69
OS-Application	0	0
PeripheralArea	0	0
Resource	8	4
ScheduleTable	16	16
Task	20	0

8.3 Stack Usage

The amount of stack used by each Task/ISR in RTA-OS is equal to the stack used in the Task/ISR body plus the context saved by RTA-OS. The size of the run-time context saved by RTA-OS depends on the Task/ISR type and the exact system configuration. The only reliable way to get the correct value for Task/ISR stack usage is to call the `Os_GetStackUsage()` API function.

Note that because RTA-OS uses a single-stack architecture, the run-time contexts of all tasks reside on the same stack and are recovered when the task terminates. As a result, run-time contexts of mutually exclusive tasks (for example, those that share an internal resource) are effectively overlaid. This means that the worst case stack usage can be significantly less than the sum of the worst cases of each object on the system. The RTA-OS tools automatically calculate the total worst case stack usage for you and present this as part of the configuration report.

8.4 Library Module Sizes

The RTA-OS kernel is demand linked. This means that each API call is placed into a separately linkable module. The following table lists the section sizes for each API module (in bytes) for the simple configuration in standard status.

Library Module	.Os_VMStatusBlock	.bss	.rodata	.vtext
ActivateTask				154
AdvanceCounter				4
CallTrustedFunction				32
CancelAlarm				118
ChainTask				142

Library Module	.Os_VMStatusBlock	.bss	.rodata	.vtext
CheckISRMemoryAccess				66
CheckObjectAccess				110
CheckObjectOwnership				108
CheckTaskMemoryAccess				66
ClearEvent				58
ControlIdle		8		76
DisableAllInterrupts		8		48
DispatchTask				238
ElapsedTime				298
EnableAllInterrupts				62
GetActiveApplicationMode				10
GetAlarm				192
GetAlarmBase				68
GetApplicationID				48
GetCounterValue				70
GetCurrentApplicationID				48
GetElapsedCounterValue				114
GetEvent				58
GetExecutionTime				58
GetISRID				10
GetIsrMaxExecutionTime				58
GetIsrMaxStackUsage				58
GetResource				104
GetScheduleTableStatus				70
GetStackSize				4
GetStackUsage				58
GetStackValue				14
GetTaskID				16
GetTaskMaxExecutionTime				58
GetTaskMaxStackUsage				58
GetTaskState				46
GetVersionInfo				42
Idle				4
InShutdown				2
IncrementCounter				24
InterruptSource				174
ModifyPeripheral				192
NextScheduleTable				162

Library Module	.Os_VMStatusBlock	.bss	.rodata	.vtext
Os_CallLWHVR				4
Os_Cfg		625	776	424
Os_Cfg_Counters			728	17250
Os_Cfg_KL				46
Os_GetCurrentIMask				10
Os_GetCurrentTPL				28
Os_IPLToImask			132	44
Os_OuterWrapper				154
Os_Trust				74
Os_VMStatusBlock	40			
Os_Vectors		4	384	118
Os_Wrapper				132
Os_setjmp				192
ProtectionSupport				64
ReadPeripheral				198
ReleaseResource				122
ResetIsrMaxExecutionTime				58
ResetIsrMaxStackUsage				58
ResetTaskMaxExecutionTime				58
ResetTaskMaxStackUsage				58
ResumeAllInterrupts				62
ResumeOSInterrupts				62
Schedule				136
SetAbsAlarm				122
SetEvent				58
SetRelAlarm				188
SetScheduleTableAsync				76
ShutdownOS				66
StackOverrunHook				6
StartOS				176
StartScheduleTableAbs				168
StartScheduleTableRel				134
StartScheduleTableSynchron				76
StopScheduleTable				106
SuspendAllInterrupts		8		48
SuspendOSInterrupts		8		82
SyncScheduleTable				76
SyncScheduleTableRel				76

Library Module	.Os_VMStatusBlock	.bss	.rodata	.vtext
TerminateTask				24
ValidateCounter				76
ValidateISR				20
ValidateResource				48
ValidateScheduleTable				48
ValidateTask				38
WaitEvent				58
WritePeripheral				180

8.5 Execution Time

The following tables give the execution times in CPU cycles, i.e. in terms of ticks of the processor's program counter. These figures will normally be independent of the frequency at which you clock the CPU. To convert between CPU cycles and SI time units the following formula can be used:

$$\text{Time in microseconds} = \text{Time in cycles} / \text{CPU Clock rate in MHz}$$

For example, an operation that takes 50 CPU cycles would be:

- at 20MHz = $50/20 = 2.5\mu\text{s}$
- at 80MHz = $50/80 = 0.625\mu\text{s}$
- at 150MHz = $50/150 = 0.333\mu\text{s}$

While every effort is made to measure execution times using a stopwatch running at the same rate as the CPU clock, this is not always possible on the target hardware. If the stopwatch runs slower than the CPU clock, then when RTA-OS reads the stopwatch, there is a possibility that the time read is less than the actual amount of time that has elapsed due to the difference in resolution between the CPU clock and the stopwatch (the *User Guide* provides further details on the issue of uncertainty in execution time measurement).

The figures presented in Section 8.5.1 have an uncertainty of 1 CPU cycle(s).

8.5.1 Context Switching Time

Task switching time is the time between the last instruction of the previous task and the first instruction of the next task. The switching time differs depending on the switching contexts (e.g. an `ActivateTask()` versus a `ChainTask()`).

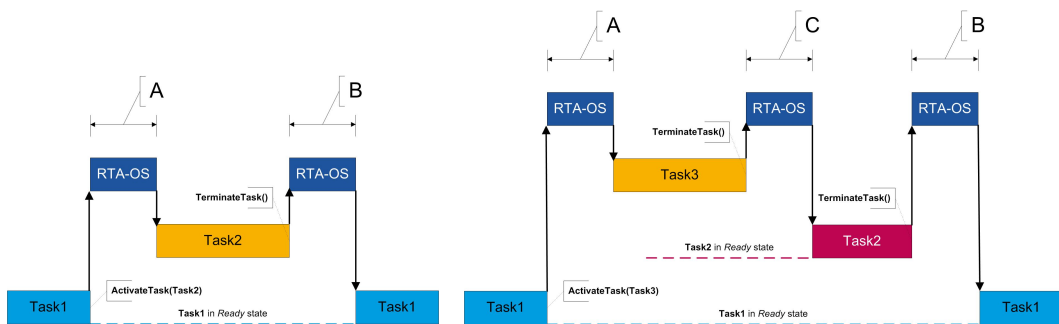
Interrupt latency is the time between an interrupt request being recognized by the target hardware and the execution of the first instruction of the user provided handler function:

For Category 1 ISRs this is the time required for the hardware to recognize the interrupt.

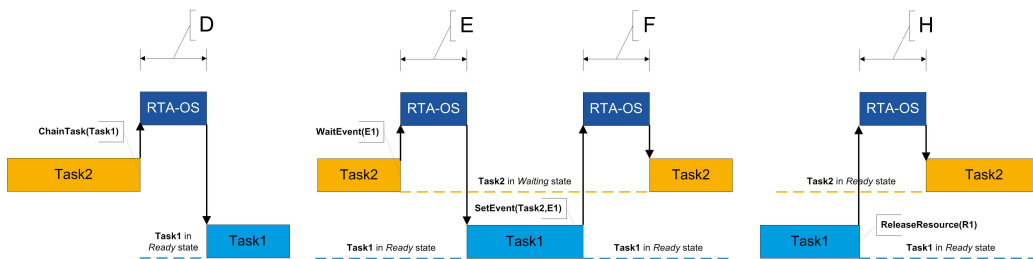
For Category 2 ISRs this is the time required for the hardware to recognize the interrupt plus the time required by RTA-OS to set-up the context in which the ISR runs.

Figure 8.1 shows the measured context switch times for RTA-OS.

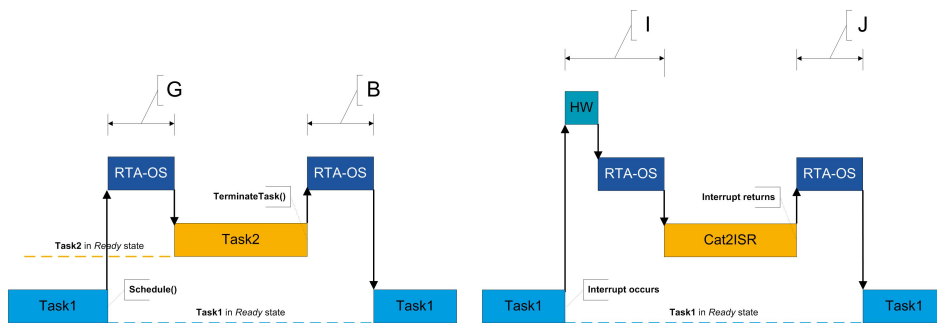
Switch	Key	CPU Cycles	Actual Time
Task activation	A	84	467ns
Task termination with resume	B	50	278ns
Task termination with switch to new task	C	56	311ns
Chaining a task	D	104	578ns
Waiting for an event resulting in transition to the WAITING state	E	232	1.29us
Setting an event results in task switch	F	312	1.73us
Non-preemptive task offers a preemption point (co-operative scheduling)	G	82	456ns
Releasing a resource results in a task switch	H	82	456ns
Entering a Category 2 ISR	I	248	1.38us
Exiting a Category 2 ISR and resuming the interrupted task	J	248	1.38us
Exiting a Category 2 ISR and switching to a new task	K	76	422ns
Entering a Category 1 ISR	L	224	1.24us



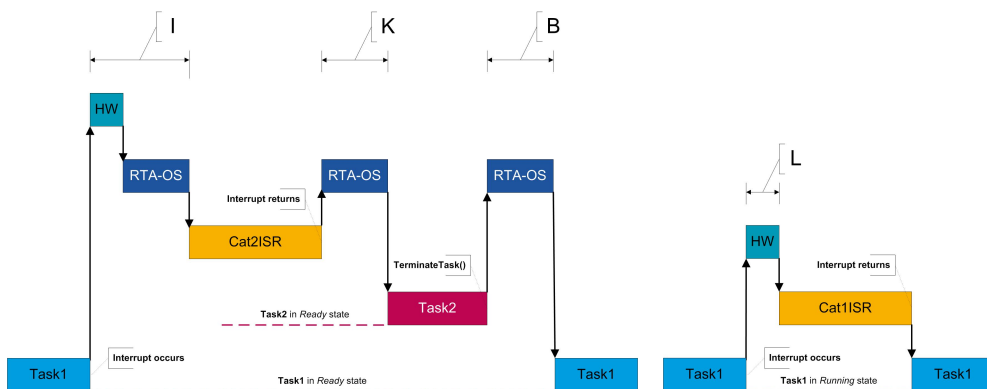
(a) Task activated. Termination resumes preempted task. (b) Task activated. Termination switches into new task.



(c) Task chained. (d) Task waits. Task is resumed when event set. (e) Task switch when resource is released.



(f) Request for scheduling made by non-preemptive task. (g) Category 2 interrupt entry. Interrupted task resumed on exit.



(h) Category 2 interrupt entry. Switch to new task on exit. (i) Category 1 interrupt entry.

Figure 8.1: Context Switching

9 Finding Out More

Additional information about PPCLWHVR/GHS-specific parts of RTA-OS can be found in the following manuals:

PPCLWHVR/GHS Release Note. This document provides information about the PP-CLWHVR/GHS port plug-in release, including a list of changes from previous releases and a list of known limitations.

Information about the port-independent parts of RTA-OS can be found in the following manuals, which can be found in the RTA-OS installation (typically in the Documents folder):

Getting Started Guide. This document explains how to install RTA-OS tools and describes the underlying principles of the operating system

Reference Guide. This guide provides a complete reference to the API, programming conventions and tool operation for RTA-OS.

User Guide. This guide shows you how to use RTA-OS to build real-time applications.

10 Contacting ETAS

10.1 Technical Support

Technical support is available to all users with a valid support contract. If you do not have a valid support contract, please contact your regional sales office (see Section 10.2.2).

The best way to get technical support is by email. Any problems or questions about the use of the product should be sent to:

rta.hotline.uk@etas.com

If you prefer to discuss your problem with the technical support team, you call the support hotline on:

+44 (0)1904 562624.

The hotline is available during normal office hours (0900-1730 GMT/BST).

In either case, it is helpful if you can provide technical support with the following information:

- Your support contract number
- Your .xml, .arxml and .rtaos files
- The command line which caused the error
- The version of the ETAS tools you are using
- The version of the compiler tool chain you are using
- The error message you received (if any)
- The file Diagnostic.dmp if it was generated

10.2 General Enquiries

10.2.1 ETAS Global Headquarters

ETAS GmbH

Borsigstrasse 24
70469 Stuttgart
Germany

Phone:	+49 711 3423-0
Fax:	+49 711 3423-2106
WWW:	www.etas.com

10.2.2 ETAS Local Sales & Support Offices

Contact details for your local sales office and local technical support team (where available) can be found on the ETAS web site:

ETAS subsidiaries	www.etas.com/en/contact.php
ETAS technical support	www.etas.com/en/hotlines.php

Index

A

Assembler, [28](#)
 AUTOSAR OS includes
 Os.h, [24](#)
 Os_Cfg.h, [24](#)
 Os_MemMap.h, [24](#)

C

CAT1_ISR, [25](#)
 Compiler, [26](#)
 Compiler (Green Hills Software v2015.1.6),
 [26](#)
 Compiler Versions, [26](#)
 Configuration
 Port-Specific Parameters, [21](#)

D

Debugger, [29](#)

E

ETAS License Manager, [11](#)
 Installation, [11](#)

F

Files, [24](#)

H

Hardware
 Requirements, [9](#)

I

Installation, [9](#)
 Default Directory, [10](#)
 Verification, [19](#)
 Interrupts, [31](#)
 Category 1, [32](#)
 Category 2, [32](#)
 Default, [32](#)
 IPL, [31](#)

L

Librarian, [28](#)
 Library
 Name of, [24](#)
 License, [11](#)
 Borrowing, [15](#)
 Concurrent, [12](#)
 Grace Mode, [12](#)

Installation, [15](#)
 Machine-named, [12](#)
 Status, [15](#)
 Troubleshooting, [16](#)
 User-named, [12](#)

Linker, [29](#)

M

Memory Model, [32](#)

O

Options, [26](#)
 Os_StackSizeType, [25](#)
 Os_StackValueType, [25](#)

P

Parameters of Implementation, [21](#)
 Performance, [37](#)
 Context Switching Times, [42](#)
 Library Module Sizes, [38](#)
 RAM and ROM, [37](#)
 Stack Usage, [38](#)
 Processor Modes, [33](#)
 User, [33](#)

R

Registers
 All supervisor-mode registers, [30](#)
 Initialization, [30](#)
 Non-modifiable, [30](#)
 R1, [30](#)
 Resource
 Pseudo-interrupt handler, [30](#)

S

Software
 Requirements, [9](#)
 Stack, [33](#)

T

Target, [30](#)
 Variants, [30](#)
 Toolchain, [26](#)

V

Variants, [30](#)
 Vector Table
 Base Address, [31](#)