RTA-OS

RH850/CS Port Guide

Copyright

Document: 10706-PG-2.0.2 EN-02-2017

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

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 -> R4.0.3 and R4.1.1 -> R4.2.2) and OSEK/VDX 2.2.3 standards. The operating system is configured and built on a PC, but runs on your target hardware.

This document describes the RTA-OS RH850/CS port plug-in that customizes the RTA-OS development tools for the Renesas RH850 with the Renesas_CS 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 RH850/CS port and cover:

- how to install the RH850/CS port plug-in;
- how to configure RH850/CS-specific attributes;
- how to build an example application to check that the RH850/CS 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 RH850, 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. 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 RH850/CS 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 (or higher) IBM compatible PC.
- 512Mb RAM.
- 500Mb 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 2000 (Service Pack 3 or later)
- Windows XP (Service Pack 2 or later)
- Windows Vista
- Windows 7
- Windows 8

Integration Guidance 2.2:*The tools provided with RTA-OS require Microsoft's .NET Framework* v2.0 *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* `http://www.microsoft.com/net/Download.aspx`*.*
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 RH850/CS 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 RH850/CS 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 `RH850CS_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:

RH850CS.dll - the port plug-in that is used by **rtaosgen** and **rtaoscfg**.

RTA-OS RH850CS Port Guide.pdf - the documentation for the port (the document you are reading now).

RTA-OS RH850CS 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`.

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

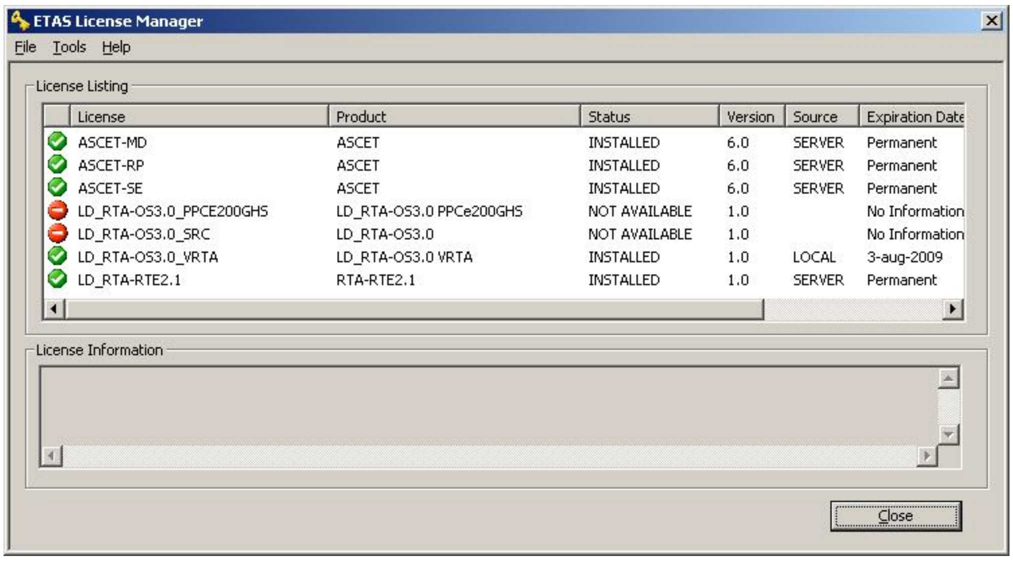
Figure 2.1 shows the ETAS License Manager in operation.

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)



	License	Product	Status	Version	Source	Expiration Date
✓	ASCET-MD	ASCET	INSTALLED	6.0	SERVER	Permanent
✓	ASCET-RP	ASCET	INSTALLED	6.0	SERVER	Permanent
✓	ASCET-SE	ASCET	INSTALLED	6.0	SERVER	Permanent
⛔	LD_RTA-OS3.0_PPCE200GHS	LD_RTA-OS3.0 PPCe200GHS	NOT AVAILABLE	1.0		No Information
⛔	LD_RTA-OS3.0_SRC	LD_RTA-OS3.0	NOT AVAILABLE	1.0		No Information
✓	LD_RTA-OS3.0_VRTA	LD_RTA-OS3.0 VRTA	INSTALLED	1.0	LOCAL	3-aug-2009
✓	LD_RTA-RTE2.1	RTA-RTE2.1	INSTALLED	1.0	SERVER	Permanent

Figure 2.1: The ETAS License manager

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 seven 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.

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.

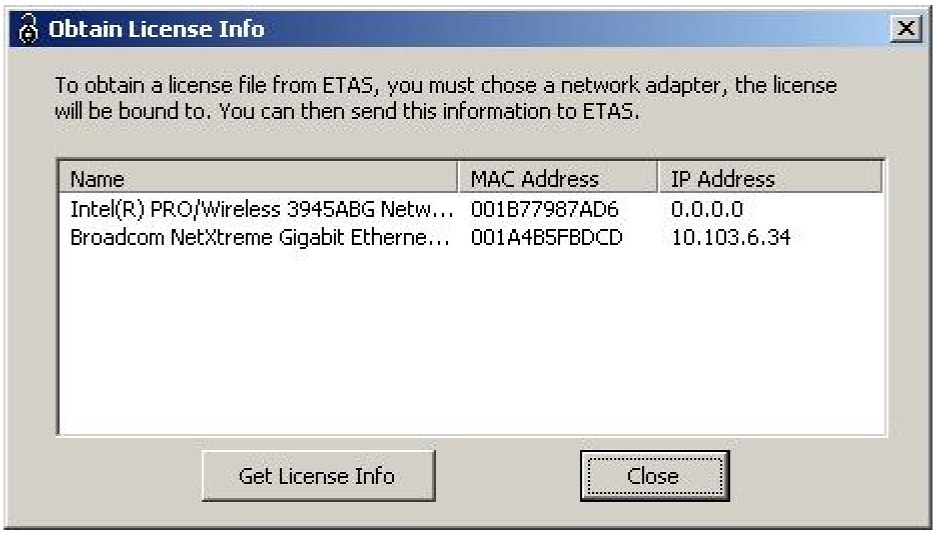
Selecting "Get License Info" tells you the Host ID and User information and lets you save this as a text file to a location of your choice.

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:


Obtain License Info

To obtain a license file from ETAS, you must chose a network adapter, the license will be bound to. You can then send this information to ETAS.

Name	MAC Address	IP Address
Intel(R) PRO/Wireless 3945ABG Netw...	001B77987AD6	0.0.0.0
Broadcom NetXtreme Gigabit Etherne...	001A4B5FBDCD	10.103.6.34

Figure 2.2: Obtaining License Information

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

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

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.

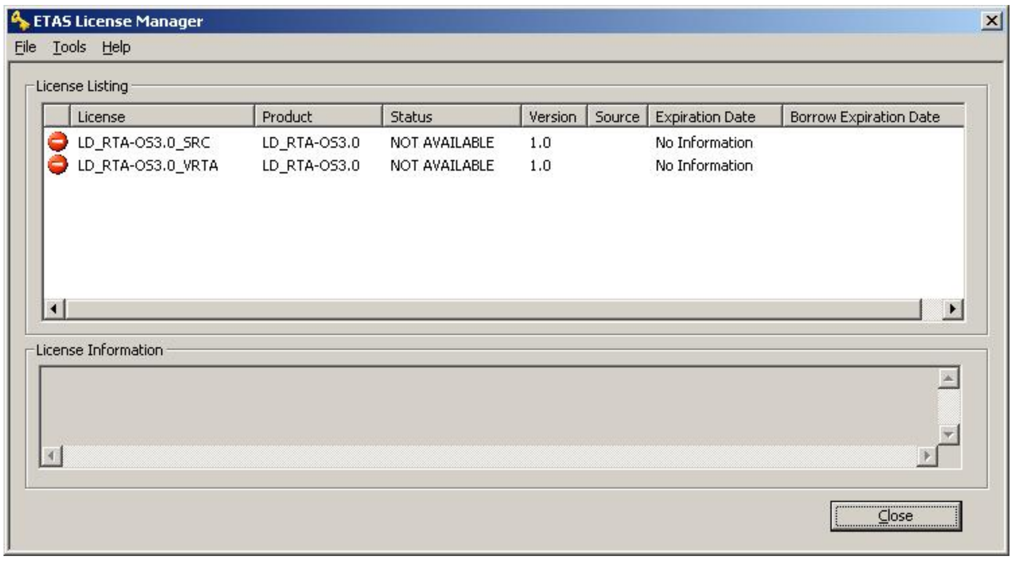
If you do not already have a FLEXnet license server then you will need to arrange for one to be installed. A copy of the FLEXnet license server, the ETAS vendor daemon and the instructions for installing and using the server (`LicensingEndUserGuide.pdf`) are placed on disk when you install the ETAS License 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 RTA-OS without a valid license, you will be given the opportunity to start the ETAS License Manager and select a license.

When the ETAS License Manager is launched, it will display the RTA-OS license state as `NOT AVAILABLE` and you will not be able to use any of the tools until a valid license is installed. This is shown in Figure 2.3.



License	Product	Status	Version	Source	Expiration Date	Borrow Expiration Date
LD_RTA-OS3.0_SRC	LD_RTA-OS3.0	NOT AVAILABLE	1.0		No Information	
LD_RTA-OS3.0_VRTA	LD_RTA-OS3.0	NOT AVAILABLE	1.0		No Information	

Figure 2.3: Unlicensed RTA-OS Installation

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.

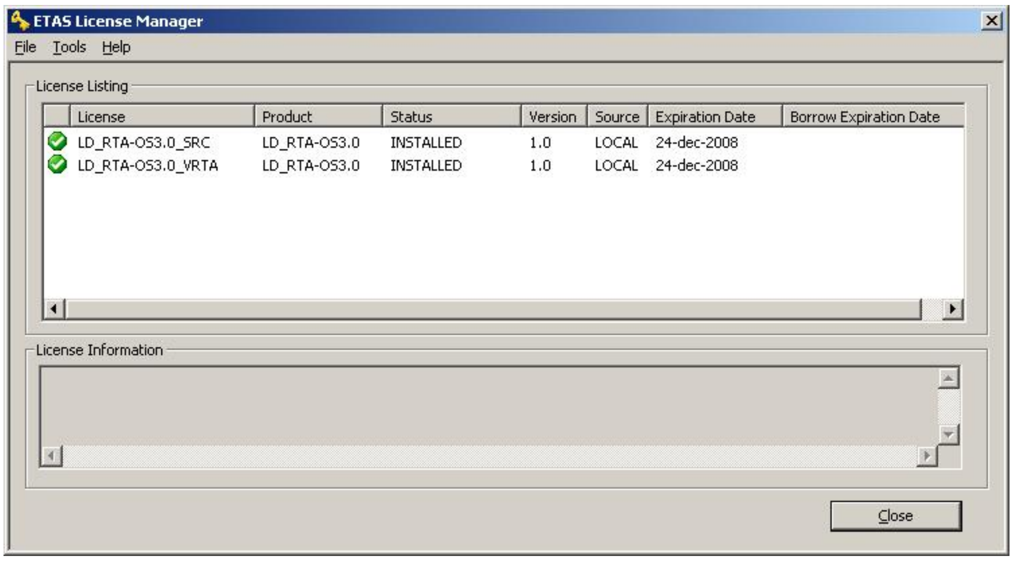
If you have a machine-based or user-based license key then you can simply install the license by opening the ETAS License Manager and selecting **File → Add License File** menu.

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¶
¶
```



	License	Product	Status	Version	Source	Expiration Date	Borrow Expiration Date
✓	LD_RTA-OS3.0_SRC	LD_RTA-OS3.0	INSTALLED	1.0	LOCAL	24-dec-2008	
✓	LD_RTA-OS3.0_VRTA	LD_RTA-OS3.0	INSTALLED	1.0	LOCAL	24-dec-2008	

Figure 2.4: Licensed features for RTA-OS

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.

When a license is installed by the ETAS License Manager it is placed in: `C:\Documents and Settings\All Users\Application Data\ETAS\FlexNet`

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"

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**.

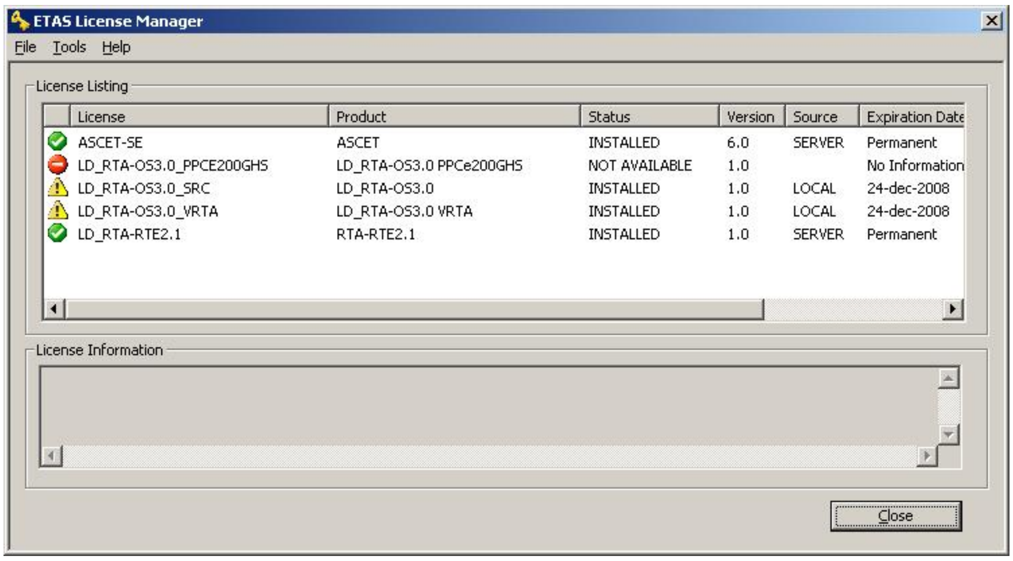
The ETAS License Manager lists all license features and their status. Valid licenses have status `INSTALLED`. Invalid licenses have status `NOT AVAILABLE`.

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.



	License	Product	Status	Version	Source	Expiration Date
✓	ASCET-SE	ASCET	INSTALLED	6.0	SERVER	Permanent
⛔	LD_RTA-OS3.0_PPCE200GHS	LD_RTA-OS3.0 PPCe200GHS	NOT AVAILABLE	1.0		No Information
⚠	LD_RTA-OS3.0_SRC	LD_RTA-OS3.0	INSTALLED	1.0	LOCAL	24-dec-2008
⚠	LD_RTA-OS3.0_VRTA	LD_RTA-OS3.0 VRTA	INSTALLED	1.0	LOCAL	24-dec-2008
✓	LD_RTA-RTE2.1	RTA-RTE2.1	INSTALLED	1.0	SERVER	Permanent

Figure 2.5: Licensed features that are due to expire

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 RH850/CS port.

4.1 Parameters of Implementation

To be a valid OSEK 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 RH850/CS 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	-	4294967295
Trusted functions	-	4294967295
Spinlocks (multicore)	-	4294967295
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 RH850/CS 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 the Lauterbach debugger('TRUE'=Generate ORTI. 'FALSE'=No ORTI (default).).

Settings

Value	Description
TRUE	Generate ORTI
FALSE	No ORTI (default)

4.2.7 ORTI/winIDEA

XML name Orti21winIDEA

Description

Enables ORTI generation for the winIDEA debugger ('TRUE'=Generate ORTI. 'FALSE'=No ORTI (default).).

Settings

Value	Description
TRUE	Generate ORTI
FALSE	No ORTI (default)

4.2.8 ORTI Stack Fill

XML name OrtiStackFill

Description

Expands ORTI information to cover stack address, size and fill pattern details to support debugger stack usage monitoring ('TRUE'=Support ORTI stack tracking. 'FALSE'=ORTI stack tracking unsupported (default).).

Settings

Value	Description
TRUE	Support ORTI stack tracking
FALSE	ORTI stack tracking unsupported (default)

4.2.9 Support winIDEA Analyzer

XML name winIDEAAnalyzer

Description

Adds support for the winIDEA profiler to track ORTI items. Context switches take a few cycles longer as additional code is inserted to support this feature. ('None'=Tracking disabled (default). 'Data'=Track by monitoring global variables. 'Soft'=Track with DBPUSH/DBTAG instructions. 'User'=Track with User-Trace (I/O Port 1).).

Settings

Value	Description
None	Tracking disabled (default)
Data	Track by monitoring global variables
Soft	Track with DBPUSH/DBTAG instructions
User	Track with User-Trace (I/O Port 1)

4.2.10 Enable Direct vector mode

XML name DirectVectors

Description

Select Direct Vector Method for handling all EI maskable interrupts with a single vector address (please refer to the Renesas documentation for more details on the vector methods).

Settings

Value	Description
TRUE	Use Direct Vector Method
FALSE	Use Table Reference Method (default)

4.2.11 Link Type

XML name OsLinkerModel

Description

Select the type of map used in linker samples.

Settings

Value	Description
RAM	Code/data in RAM (default)
FLASH	Code in FLASH, data in RAM

4.2.12 Trust Trap

XML name OsTrustTrap

Description

When there is untrusted code in an application RTA-OS needs to reserve one of the two CPU trap vectors to switch between trusted and untrusted code. This option selects which is used. This functionality must be supported if a user provided TRAP handlers are used in such applications.

Settings

Value	Description
0x40	TRAP 0x0-0xF instructions used by RTA-OS (default)
0x50	TRAP 0x10-0x1F instructions used by RTA-OS

4.2.13 Enable stack repositioning

XML name AlignUntrustedStacks

Description

Use to support realignment of the stack for untrusted code when there are MPU protection region granularity issues. Refer to the documentation for Os_Cbk_SetMemoryAccess

Settings

Value	Description
TRUE	Support repositioning
FALSE	Normal behavior (default)

4.2.14 Enable untrusted stack check

XML name DistrustStacks

Description

Extra code can be placed in interrupt handlers to detect when untrusted code has an illegal stack pointer value. Also exception handlers run on a private stack (Refer to the documentation for Os_Cbk_GetAbortStack). This has a small performance overhead, so is made optional.

Settings

Value	Description
TRUE	Perform the checks
FALSE	Do not check (default)

4.2.15 EBASE setting

XML name EBASE_value

Description

Controls EBASE register setting within Os_InitializeVectorTable(). This register is only initialized if RTA-OS generates a vector table. The register value is aligned to a 512 byte boundary. When setting the EBASE register directly ensure that the RINT bit matches the desired interrupt vector mode.

Valid values are; ’true’ initialize EBASE to the address of Os_interrupt_vectors the RTA-OS generated vector table (default), ’false’ do not initialize EBASE, the hex address used to initialize EBASE, or the address of the label used to initialize EBASE.

4.2.16 Handle FPU context

XML name handle_FPU_context

Description

Extra code can be added during context switches to additionally handle the FPSR and FPRPC registers for tasks and ISRs. This option should only be used if the majority of Tasks and ISRs in the application contain FPU instructions otherwise register sets should be used. This has a small performance overhead, so is made optional.

Settings

Value	Description
TRUE	Save FPU context
FALSE	Do not save FPU context (default)

4.2.17 CrossCore0 IPIR

XML name CrossCore0IPIR

Description

Optionally specify the IPIR used for cross-core interrupts for core 0. A free IPIR will be selected automatically if one is not specified. Used in multicore applications only.

4.2.18 CrossCore1 IPIR

XML name CrossCore1IPIR

Description

Optionally specify the IPIR used for cross-core interrupts for core 1. A free IPIR will be selected automatically if one is not specified. Used in multicore applications only.

4.2.19 Block default interrupt

XML name block_default_interrupt

Description

Where a default interrupt is specified, it will normally execute if a spurious interrupt fires. This option can change this behavior by changing the priority assigned to unused interrupt sources.

Settings

Value	Description
TRUE	Block the default interrupt
FALSE	Allow the default interrupt handler to run if a spurious interrupt fires (default)

4.2.20 GetAbortStack always

XML name always_call_GetAbortStack

Description

When the abort ISR is triggered always use the Os_Cbk_GetAbortStack() call-back to set up a safe area of memory to use as a stack executing the ProtectionHook (please refer to the documentation for Os_Cbk_GetAbortStack).

Settings

Value	Description
TRUE	Always call Os_Cbk_GetAbortStack()
FALSE	Only call Os_Cbk_GetAbortStack() when the 'Enable untrusted stack check' target option is selected (default)

4.2.21 Cache CoreID in CTPSW

XML name Cache_CoreID_in_CTPSW

Description

Optionally specify that the CTPSW is used to cache the Core ID into. This improves performance in multicore applications, especially where there is untrusted code. If selected the OS will initialize and use the CTPSW where it can to read the Core ID. It must not be modified (beware this may be unexpectedly changed in untrusted code).

Settings

Value	Description
TRUE	Cache the Core ID into CTPSW
FALSE	Read Core ID from HTCFG0.PEID (default)

4.2.22 Optimizer setting

XML name Optimizer_setting

Description

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

Settings

Value	Description
Odefault	Optimization that does not affect debug information
Osize	Improve code size over performance (default)
Ospeed	Improve code performance over size

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 RH850/CS 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 API Calls

5.1.1 Os_InitializeVectorTable

Initialize the ICn registers.

Syntax

```
void Os_InitializeVectorTable(void)
```

Description

Os_InitializeVectorTable() initializes the interrupt controller registers and priorities according to the requirements of the project configuration.

The ICn registers for interrupts declared in the project configuration are modified so that their priority matches that expected by RTA-OS. If Table Reference method interrupt vectors is selected then the EITB bits are set for the configured interrupts. The ICn mask bits are not cleared, which can be done later using the provided macros (i.e. Os_Enable_x). The ICn registers of interrupt channels not declared in the project configuration are not modified by this function. The vector table base address register is set to match the location of Os_interrupt_vectors. If Table Reference interrupt vectors are used then the INTBP register is set to match the location of Os_EI_vectors. The PMR register is set to block all Category 2 interrupts (as these should only trigger after StartOS()) but allow Category 1 interrupts to fire. The PSW.ID bit is also set to allow EI interrupts. For G3KH core parts the FPIPR register is set so that it's priority matches that expected by RTA-OS. Os_InitializeVectorTable() should be called before StartOS(). It should be called even if 'Suppress Vector Table Generation' is set to TRUE.

In multicore applications Os_InitializeVectorTable() should be called by all cores.

Example

```
Os_InitializeVectorTable();
```

See Also

StartOS

5.2 Callbacks

5.2.1 Os_Cbk_GetAbortStack

Callback routine to provide the start address of the stack to use to handle exceptions.

Syntax

```
FUNC(void *,{memclass}) Os_Cbk_GetAbortStack(void)
```

Return Values

The call returns values of type `void *`.

Description

Untrusted code can misbehave and cause a protection exception. When this happens, AUTOSAR requires that ProtectionHook is called and the task, ISR or OS Application must be terminated.

It is possible that at the time of the fault the stack pointer is invalid. For this reason, if 'Enable untrusted stack check' is configured, RTA-OS will call Os_Cbk_GetAbortStack to get the address of a safe area of memory that it should use for the stack while it performs this processing.

Maskable interrupts will be disabled during this process so the stack only needs to be large enough to perform the ProtectionHook.

A default implementation of Os_Cbk_GetAbortStack is supplied in the RTA-OS library that will place the abort stack at the starting stack location of the untrusted code.

In systems that use the Os_Cbk_SetMemoryAccess callback, the return value is the last stack location returned in ApplicationContext from Os_Cbk_SetMemoryAccess. This is to avoid having to reserve memory. Note that this relies on Os_Cbk_SetMemoryAccess having been called at least once on that core otherwise zero will be returned. (The stack will not get adjusted if zero is returned.) Otherwise the default implementation returns the address of an area of static memory that is reserved for sole use by the abort stack.

Note: memclass is OS_APPL_CODE for AUTOSAR 3.x, OS_CALLOUT_CODE for AUTOSAR 4.0, OS_OS_CBK_GETABORTSTACK_CODE for AUTOSAR 4.1.

Example

```
FUNC(void *,{memclass}) Os_Cbk_GetAbortStack(void) {
  static uint32 abortstack[40U];
  return &abortstack[40U];
}
```

Required when

The callback must be present if 'Enable untrusted stack check' is configured and there are untrusted OS Applications. The callback is also present if the 'GetAbortStack always' target option is enabled.

5.2.2 Os_Cbk_StartCore

Callback routine used to start a non-master core on a multicore variant.

Syntax

```
FUNC(StatusType, {memclass})Os_Cbk_StartCore(
    uint16 CoreID
)
```

Return Values

The call returns values of type `StatusType`.

Value	Build	Description
`E_OK`	all	No error.
`E_OS_ID`	all	The core does not exist or can not be started.

Description

In a multicore application, the StartCore or StartNonAutosarCore OS APIs have to be called prior to StartOS for each core that is to run. For this target port, these APIs make a call to Os_Cbk_StartCore() which is responsible for starting the specified core.

RTA-OS provides a default implementation of Os_Cbk_StartCore() that will be appropriate for most normal situations. Os_Cbk_StartCore() does not get called for core 0, because core 0 must start first.

Note: memclass is OS_APPL_CODE for AUTOSAR 3.x, OS_CALLOUT_CODE for AUTOSAR 4.0, OS_OS_CBK_STARTCORE_CODE for AUTOSAR 4.1.

Example

```
FUNC(StatusType, {memclass}) Os_Cbk_StartCore(uint16 CoreID)
{
  StatusType ret = E_OS_ID;

  /* If an expected core... */
  if (CoreID == 1U) {
    /* Set the global variable Os_StartCoreVar to a known value
     * to allow a secondary core to resume code execution */
    Os_StartCoreVar = 0xA5U;

    ret = E_OK;
  }

  return ret;
}
```

Required when

Required for non-master cores that will be started.

See Also

StartCore
StartNonAutosarCore
StartOS

5.2.3 Os_Cbk_StopCore

Callback routine used to stop a non master core on a multicore variant.

Syntax

```
FUNC(StatusType, {memclass}) Os_Cbk_StopCore(void)
```

Return Values

The call returns values of type `StatusType`.

Value	Build	Description
E_OK	all	No error.
E_OS_ID	all	The core does not exist or can not be started.

Description

When the RH850 CPU comes out of reset all cores start automatically. The Os_Cbk_StopCore() callback is used to suspend the execution of the current core in preparation for execution to be resumed by the primary core calling the Os_Cbk_StartCore callback. Os_Cbk_StopCore() should be used before the primary core uses the StartCore() or StartOS() API calls.

RTA-OS provides a default implementation of Os_Cbk_StopCore() that will be appropriate for most normal situations. Here Os_Cbk_StopCore() is called from the C start-up code, it can also be called in OS_MAIN(). Os_Cbk_StopCore() should not get called for core 0 as this is the primary core.

Note: memclass is OS_APPL_CODE for AUTOSAR 3.x, OS_CALLOUT_CODE for AUTOSAR 4.0, OS_OS_CBK_STOPCORE_CODE for AUTOSAR 4.1.

Example

```
FUNC(StatusType, {memclass}) Os_Cbk_StopCore(void)
{
  /* If an expected core, shutdown until the Os_StartCoreVar
   * global variable holds a value set by Os_Cbk_StartCore(). */
  /* Put the Os_StartCore variable in a known state until
   * the master core sets it */
  while (Os_StartCoreVar != 0xA5U) {
    /* Use the RH850 SNOOZE instruction to put the core in
     * standby. */
    OS_SNOOZE();
  }

  return E_OK;
}
```

Required when

Required for non master cores that will be started.

See Also

StartCore
StartNonAutosarCore
StartOS

5.3 Macros

5.3.1 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. It should be used with both EI and FE interrupts.

Example

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

5.3.2 Os_Clear_x

Use of the Os_Clear_x() will clear the interrupt request bit of the EI level interrupt control register for the named interrupt channel. The macro can be called using either the EI channel number or the RTA-OS configured vector name. In the example, this is Os_Clear_EI_Channel_20() and Os_Clear_Millisecond() respectively. To use the Os_Clear_x macro the file Os_ConfigInterrupts.h must be included through the use of #include. The macro is provided so the interrupt channel can be cleared without corrupting the interrupt priority value configured by calling Os_InitializeVectorTable(). It may not be used by untrusted code.

Example

```
Os_Clear_EI_Channel_20()
Os_Clear_Millisecond()
```

5.3.3 Os_DisableAllConfiguredInterrupts

The Os_DisableAllConfiguredInterrupts macro will disable all configured EI interrupt channels. To use the Os_DisableAllConfiguredInterrupts macro the file Os_ConfigInterrupts.h must be included through the use of #include. The macro is provided so the interrupt channels can be disabled without corrupting the interrupt priority values configured by calling Os_InitializeVectorTable(). It may not be used by untrusted code.

Example

```
Os_DisableAllConfiguredInterrupts()
...
Os_EnableAllConfiguredInterrupts()
```

5.3.4 Os_Disable_x

Use of the Os_Disable_x macro will disable the named interrupt channel. The macro can be called using either the EI channel number or the RTA-OS configured vector name. In the example, this is Os_Disable_EI_Channel_20() and Os_Disable_Millisecond() respectively. To use the Os_Disable_x macro the file Os_ConfigInterrupts.h must be included through the use of #include. The macro is provided so the interrupt channel can be masked without corrupting the interrupt priority value configured by calling Os_InitializeVectorTable(). It may not be used by untrusted code.

Example

```
Os_Disable_EI_Channel_20()
Os_Disable_Millisecond()
```

5.3.5 Os_EnableAllConfiguredInterrupts

The Os_EnableAllConfiguredInterrupts macro will enable all configured EI interrupt channels. To use the Os_EnableAllConfiguredInterrupts macro the file Os_ConfigInterrupts.h must be included through the use of #include. The macro is provided so the interrupt channels can be enabled without corrupting the interrupt priority values configured by calling Os_InitializeVectorTable(). It may not be used by untrusted code.

Example

```
Os_DisableAllConfiguredInterrupts()
...
Os_EnableAllConfiguredInterrupts()
```

5.3.6 Os_Enable_x

Use of the Os_Enable_x macro will enable the named interrupt channel. The macro can be called using either the EI channel number or the RTA-OS configured vector name. In the example, this is Os_Enable_EI_Channel_20() and Os_Enable_Millisecond() respectively. To use the Os_Enable_x macro the file Os_ConfigInterrupts.h must be included through the use of #include. The macro is provided so the interrupt channel can be enabled without corrupting the interrupt priority value configured by calling Os_InitializeVectorTable(). It may not be used by untrusted code.

Example

```
Os_Enable_EI_Channel_20()
Os_Enable_Millisecond()
```

5.3.7 Os_IntChannel_x

The Os_IntChannel_x macro can be used to get the vector number associated with the named INTC interrupt (0, 1, 2...). The macro can be called using either the INTC vector name or the RTA-OS configured vector name. In the example, this is Os_IntChannel_INTTAUD0I11 and Os_IntChannel_Millisecond respectively. To use the Os_IntChannel_x macro the file Os_ConfigInterrupts.h must be included through the use of #include.

Example

```
trigger_interrupt(Os_IntChannel_INTTAUD0I11);
trigger_interrupt(Os_IntChannel_Millisecond);
```

5.4 Type Definitions

5.4.1 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);
```

5.4.2 Os_StackValueType

An unsigned value representing the position of the stack pointer (ESP).

Example

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

6 Toolchain

This chapter contains important details about RTA-OS and the Renesas_CS 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 Renesas (CS+) RH850 Compiler CC-RH v1.0.2

Ensure that ccrh.exe is on the path.

Tested on Release tests were performed on this version.

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	ccrh.exe
Version	RH850 Family Compiler V1.02.00.04 [21 Jan 2015]

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.

`-Xcommon=v850e3v5` Specify the target processor instruction set

`-Xcpu=g3k` Specify the processor core type (i.e. g3k or g3m)

`-Xreserve_r2` Reserves the r2 register

`-Osize` Optimizer strategy (other values are supported using the target option ’Optimizer setting’ value)

`-Otail_call=off` Do not use tail call optimization

`-Ounroll=0` Do not unroll loops when optimizing

`-Xcall_jump=32` Use jarl32/jr32 instructions for branching to functions (32 bit displacement)

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:

  -Otail_call=off

  -Ounroll=0

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.

`-Xreg_mode=common` Restrict the general purpose registers used by the compiler

`-Xreg_mode=22` Restrict the general purpose registers used by the compiler

`-Xenum_type=auto` Enumerated types are handled as the smallest integer type

`-Xpack` Pack structures with the specified byte alignment

`-Xbit_order` Set the order of bit field members

`-Xep` Set ep register use

6.2.2 Assembler

Name	asrh.exe
Version	RH850 Family Assembler V1.02.00.05 [28 Jan 2015]

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	rlink.exe
Version	Renesas Optimizing Linker V2.02.00 [25 Dec 2014]

6.2.4 Linker

Name	rlink.exe
Version	Renesas Optimizing Linker V2.02.00 [25 Dec 2014]

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.

`ENTRY _Os_sample_reset` Set the application entry function

`DEBUG` Include debug information in the output file

`NOCOMPRESS` Do not compress debug information

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 the kernel

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 10654 or later

Notes on using ORTI with the debugger

ORTI with the Lauterbach and iSYSTEM debuggers

When ORTI information is enabled entry and exit times for Category 1 interrupts are increased by a few cycles to support tracking of Category 1 interrupts by the debugger.

ORTI Stack Fill with the Lauterbach debugger

The ’ORTI Stack Fill’ target option is provided to extend the ORTI support to allow evaluation of unused stack space. The Task.Stack.View command can then be used in the Trace32 debugger.

The application must contain 32 bit constant values referencing the base and size of the stack section (e.g for a stack section defined in the linker command file as ’stack.bss’). This is demonstrated in the sample applications:

In assembler code:

```
.public _OS_STACK0_BASE
.public _OS_STACK0_SIZE
.section ".Os_rodata.const", const
.align 2
_OS_STACK0_BASE:
.dw #__s.stack.bss
_OS_STACK0_SIZE:
.dw #__e.stack.bss - #__s.stack.bss
```

Then in the C code (needed to get the debugger information in the application):

```
extern const uint32 OS_STACK0_BASE;
extern const uint32 OS_STACK0_SIZE;
```

The fill pattern used by the debugger must be contained within a 32 bit constant `OS_STACK_FILL` (e.g. for a fill pattern `0xCAFEF00D`).

```
const uint32 OS_STACK_FILL = 0xCAFEF00D;
```

The stack must be initialized with this fill pattern either in the application start-up routines or during debugger initialization.

ORTI Stack Fill with the iSYSTEM debugger

This is similar to the support in the Lauterbach debugger. Again 32 bit constant values containing the base, size and fill pattern of the stack section are required (demonstrated in the sample applications). These values do not reference linker generated labels in the assembler code and are hard coded in the C code. For a stack located in the memory range 0xFEDFF800 to 0xFEDFFFFF with a fill pattern of 0xCAFEF00D the C code would be as follows:

```
const uint32 OS_STACK_FILL = 0xCAFEF00D;
const uint32 OS_STACK0_BASE = 0xFEDFF800;
const uint32 OS_STACK0_SIZE = 0x00000800;
```

6.3 Generating OS Size Information Reports

When generating OS Size Information reports from an RTA-OS library file `readelf.exe` is used to gather the details on code and data sizes. This utility program is used as the Renesas compilation tools does not include an alternative program that can read the content of the compiler object or library files. RTA-OS requires that `readelf.exe` from the v2.22 MinGW binutils is located on the path. The MinGW tools can be found at `http://www.mingw.org/`.

7 Hardware

7.1 Supported Devices

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

Name:	Renesas
Device:	RH850

The following variants of the RH850 are supported:

- C1H
- C1M
- CCC
- D1x
- E1L
- E1MS
- E1xFCC
- E1xFCC1
- F1H
- F1K
- F1L
- F1M
- GenericRH850_16IPL
- GenericRH850_8IPL
- P1HC
- P1LC
- P1M
- P1MC
- R1L

If you require support for a variant of RH850 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
EBASE	The EBASE register must be set to match the interrupt vector mode declared in the configuration. If using Direct Vector interrupt vectors the RINT bit should be set to configure a single vector entry. This can be done by calling `Os_InitializeVectorTable()`.
EIBDn	The INTC interrupt core binding (in multicore parts) must be set to match the values declared in the configuration. This can be done by calling `Os_InitializeVectorTable()`.
EICn	The INTC priorities must be set to match the values declared in the configuration. This can be done by calling `Os_InitializeVectorTable()`.
FPIPR	The FPIPR register (not supported on G3K and G3M cores) must be set to match the interrupt priority declared in the configuration
INTBP	The interrupt vector base address must be set to match the application vector table address when using Table Reference interrupt vectors. This can be done by calling `Os_InitializeVectorTable()`.
INTCFG	The interrupt function setting register must be set to disable updating of the ISPR register. This is done by calling `Os_InitializeVectorTable()`.
P1M	When winIDEA User tracing is required the I/O Port 1 Mode register (PM1) must configure all of the I/O pins to general purpose output before calling `StartOS()`.
PSW.UM	The CPU must be in Supervisor mode (i.e. PSW.UM=0).
SP	The stack must be allocated and SP initialized before calling `Os_InitializeVectorTable()`.

7.2.2 Modification

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

Register	Notes
CTPC/CTPSW	RTA-OS does not preserve the (CTPC) and (CTPSW) during task and interrupt pre-emption. The table reference call (callt) should not be used. Used by RTA-OS in multi-core applications when the target option Cache CoreID is selected
INTCFG	User code may not directly change the Interrupt function setting Register.
ISPR	User code may not directly change the In-Service Priority Register.
P1	I/O Port 1 is reserved to support winIDEA User tracing when configured in the application.
PMR	User code may not directly change the Priority Mask Register.
PSW	User code may not directly change the Program Status Word other than as a result of normal program flow.
R1	Reserved for use by the Renesas assembler.
R2	Reserved with the `-Xreserve_r2` compiler option and preserved by the OS. The r2 register is reserved for future OS use
SP	User code may not change the stack pointer other than as a result of normal program flow.

7.3 Interrupts

This section explains the implementation of RTA-OS's interrupt model on the RH850.

7.3.1 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 RH850:

IPL	PMR	Description
0	0x0000	User (task) level. No Interrupts are masked
1	0x8000	G3M core EI Maskable Category 1 and 2 interrupts
1	0x0080	G3K core EI Maskable Category 1 and 2 interrupts
2	0xC000	G3M core EI Maskable Category 1 and 2 interrupts
2	0x00C0	G3K core EI Maskable Category 1 and 2 interrupts
...	...	EI Maskable Category 1 and 2 interrupts
8	0xFF00	G3M core EI Maskable Category 1 and 2 interrupts
8	0x00FF	G3K core EI Maskable Category 1 and 2 interrupts
...	...	EI Maskable Category 1 and 2 interrupts
16	0xFFFF	G3M EI Maskable Category 1 and 2 interrupts
17	N/A	EI Category 1 interrupts maskable through the PSW.ID bit, EITRAPx and SYSCALL
18	N/A	FE Category 1 interrupts maskable through the PSW.NP bit and FETRAP
19	N/A	FE Category 1 traps and FE level non-maskable interrupts (FENMI)

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 range of IPLs that EI interrupts support is not consistent over all RH850 variants. EI interrupts can have up to 16 IPLs for G3M core devices. On G3K cores this is restricted to 8 levels.

The RH850 interrupt controller hardware expects IPL values to work counter to the standardized RTA-OS scheme. In the RH850 zero is always the highest IPL and depending upon the core variant either 7 or 15 is the lowest IPL. RTA-OS converts the IPLs from its standardized scheme to that expected by the RH850 hardware. The function Os_InitializeVectorTable() can be used to initialize the EI interrupt priorities to that detailed in the RTA-OS application configuration. If this function is not used then care must be taken that interrupts are initialized to the IPLs expected by RTA-OS otherwise the OS will not operate correctly.

7.3.2 Using FETRAP TRAP and SYSCALL Instructions

The SYSCALL, TRAP and FETRAP instructions are not masked by the state of the PSW or the PMR registers. Care must then be taken if these instructions are used in application code. These interrupts have been nominally been allocated RTA-OS IPLs of 17 and 18 even though they can never be masked.

7.3.3 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 RH850. A ✓ indicates that the mapping is permitted and a ✗ indicates that it is not permitted:

Address	Category 1	Category 2
0x10, SYSERR	✓	✗
0x20, HVTRAP	✓	✗
0x30, FETRAP	✓	✗
0x40, EITRAP0	✓	✗
0x50, EITRAP1	✓	✗
0x60, RIE	✓	✗
0x70, FPP/FPI Category 1 trap maskable through the PSW.ID bit (G3K and G3M cores), maskable through the PMR register (G3KH cores)	✓	✗
0x80, UCPOP	✓	✗
0x90, MIP/MDP/ITLBE/DTLBE	✓	✗
0xA0, PIE	✓	✗
0xB0, DEBUG	✓	✗
0xC0, MAE	✓	✗
0xD0, Reserved Category 1 trap	✓	✗
0xE0, FENMI	✓	✗
0xF0, FEINT Category 1 trap maskable through the PSW.NP bit	✓	✗
0x100+, Maskable EI interrupt handlers	✓	✓

7.3.4 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
Os_intvect	Should be located at absolute address 0x10 or located in accordance with the EBASE register. The first entry is the HVTRAP interrupt handler offset 0x10.
Os_EI_vect	Should be located in accordance with the INTBP register when using Table Reference interrupt vectors.

The function `Os_InitializeVectorTable` should be called before `StartOS()` to set the EBASE register to the address of the label `Os_interrupt_vectors` aligned to a 0x200 byte boundary.

The RTA-OS generated vector table does not include the reset vector. You should provide this and locate it before the HVTRAP interrupt handler.

When the default interrupt is configured the RTA-OS generated vector table contains entries for all supported interrupts for the selected chip variant. If the default interrupt is not configured then entries are created up the highest configured interrupt.

RTA-OS reserves one EI trap vector (i.e. configurable to be either 0x40 or 0x50) for applications that use untrusted code (see section 4.2.11 Trust Trap for more details).

RTA-OS currently supports two vector table formats: Table reference method and Direct vector.

When you supply the vector table (i.e. 'Suppress Vector Table Generation' is set to TRUE) the label `Os_interrupt_vectors` should be placed at the start of the vector table so that the function `Os_InitializeVectorTable()` can set the EH_BASE register to the correct address. Note that the minimum alignment of the EBASE register is 0x200 bytes so the vector table must be aligned accordingly.

7.3.5 Using Raw Exception Handlers

RTA-OS supports direct branches in the interrupt vector table for interrupt vectors with address offsets less than 0x100. Normally RTA-OS produces wrapper code around the interrupt handler functions for these exceptions to enforce the correct interrupt controller IPL settings. If interrupt handlers in this address offset range are given names starting with "b_" then the interrupt vector table entry is an unconditional jump relative "jr" instruction to the handler function. When using these raw exception handlers it is the user's responsibility that:

- The correct register context is saved and restored.
- The correct return instruction is used.
- Interrupts are not re-enabled in these handlers.
- The RTA-OS API is not used in these handlers.

7.3.6 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 */
}
```

7.3.7 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 <Os.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.

7.3.8 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 7.3.6 for further details).

When the RH850 comes out of reset all maskable EI interrupts (i.e. those with address offsets 0x100 and above) are masked and set to trigger at the lowest priority. When the function Os_InitializeVectorTable() is used to initialize the EI interrupt priorities only the configured interrupts in the application are set up. All unused EI interrupt channels are left in their reset state even if the RTA-OS Default Interrupt is configured. For a Default Interrupt to trigger on an unattached EI interrupt channel then the these unattached channels must have been initialized by application code.

7.4 Memory Model

The following memory models are supported:

Model	Description
Standard	The standard 32-bit EABI memory model is used.

7.4.1 Small Data Area (SDA)/Global Pointer relative addressing

The Renesas compiler supports the use the small data area with the **#pragma** section gp_disp16 and **#pragma** section gp_disp23 directives (see the compiler documentation for more details). To make use of the SDA these pragmas can be applied to the RTA-OS library code with `Os_MemMap.h`. The following code demonstrates how to apply SDA usage to a data section:

```
/*OS Initialized RAM: unspecified size */
#elif defined (OS_START_SEC_VAR_UNSPECIFIED)
#undef OS_START_SEC_VAR_UNSPECIFIED
#pragma section gp_disp16 ".Os_sda"
#elif defined (OS_STOP_SEC_VAR_UNSPECIFIED)
#undef OS_STOP_SEC_VAR_UNSPECIFIED
#pragma section default
```

7.5 Processor Modes

RTA-OS can run in the following processor modes:

Mode	Notes
Trusted	All trusted code runs in Supervisor mode (i.e. `PSW.UM` clear).
Untrusted	All untrusted code runs in User mode (i.e. `PSW.UM` set).

Trusted-with-protection (TWP) requires that the RH850 MPU can restrict supervisor mode memory accesses. This is not supported by all RH850 hardware variants (i.e. F1L and R1L do not allow this). When using TWP please check the Renesas documentation to ensure that the variant MPU hardware can supports this feature (i.e. check the MPM.SVP and MPATn.SX|SW|SR). If TWP has been configured in an application for a variant that is known not to support TWP RTA-OS will generate an error when building the library.

7.6 Stack Handling

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

RTA-OS manages the stack (via register R3).

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 RH850/CS 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	F1L on Renesas RH850/F1x 176 Pin F1L (R7F7010352)
CPU Clock Speed	80.0MHz
Stopwatch Speed	40.0MHz

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:

- `Os_Cfg_Counters` includes data for counters, alarms and schedule tables.
- `Os_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
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_EI_vect.const	.Os_intvect.text	.Os_primitives.text	.bss	.const	.text
ActivateTask						116
AdvanceCounter						4
CallTrustedFunction						30
CancelAlarm						88
ChainTask						110
CheckISRMemoryAccess						48
CheckObjectAccess						100
CheckObjectOwnership						96
CheckTaskMemoryAccess						48
ClearEvent						30
ControlIdle				4		54
DisableAllInterrupts				8		52
DispatchTask						186
ElapsedTime						158
EnableAllInterrupts						40
GetActiveApplicationMode						10
GetAlarm						168
GetAlarmBase						60
GetApplicationID						44
GetCounterValue						42
GetCurrentApplicationID						44
GetElapsedCounterValue						66
GetEvent						30
GetExecutionTime						30
GetISRID						10
GetIsrMaxExecutionTime						30
GetIsrMaxStackUsage						30
GetResource						70
GetScheduleTableStatus						42
GetStackSize						6
GetStackUsage						30
GetStackValue						14
GetTaskID						16
GetTaskMaxExecutionTime						30
GetTaskMaxStackUsage						30
GetTaskState						50

Library Module	.Os_EI_vect.const	.Os_intvect.text	.Os_primitives.text	.bss	.const	.text
GetVersionInfo						34
Idle						4
InShutdown						2
IncrementCounter						18
NextScheduleTable						142
Os_Cfg				625	776	276
Os_Cfg_Counters					728	4790
Os_Cfg_KL						56
Os_FE_wrapper			134			
Os_GetAbortStack				160		12
Os_GetCurrentIMask						6
Os_GetCurrentTPL						34
Os_Trust						12
Os_Vectors	320	240	54			
Os_Wrapper						94
Os_abort			52			
Os_mid_wrapper			130			
Os_setjmp			120			
Os_vec_init						114
ProtectionSupport						40
ReleaseResource						78
ResetIsrMaxExecutionTime						30
ResetIsrMaxStackUsage						30
ResetTaskMaxExecutionTime						30
ResetTaskMaxStackUsage						30
ResumeAllInterrupts						40
ResumeOSInterrupts						40
Schedule						102
SetAbsAlarm						108
SetEvent						30
SetRelAlarm						162
SetScheduleTableAsync						58
ShutdownOS						80
StackOverrunHook						16
StartOS						148
StartScheduleTableAbs						136

Library Module	.Os_EI_vect.const	.Os_intvect.text	.Os_primitives.text	.bss	.const	.text
StartScheduleTableRel						124
StartScheduleTableSynchron						58
StopScheduleTable						80
SuspendAllInterrupts				8		52
SuspendOSInterrupts				8		64
SyncScheduleTable						58
SyncScheduleTableRel						58
TerminateTask						24
ValidateCounter						40
ValidateISR						14
ValidateResource						34
ValidateScheduleTable						34
ValidateTask						34
WaitEvent						30

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:

Time in microseconds = Time in cycles / CPU Clock rate in MHz

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

- at 20MHz = 50/20 = 2.5μs
- at 80MHz = 50/80 = 0.625μs
- at 150MHz = 50/150 = 0.333μs

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()`).

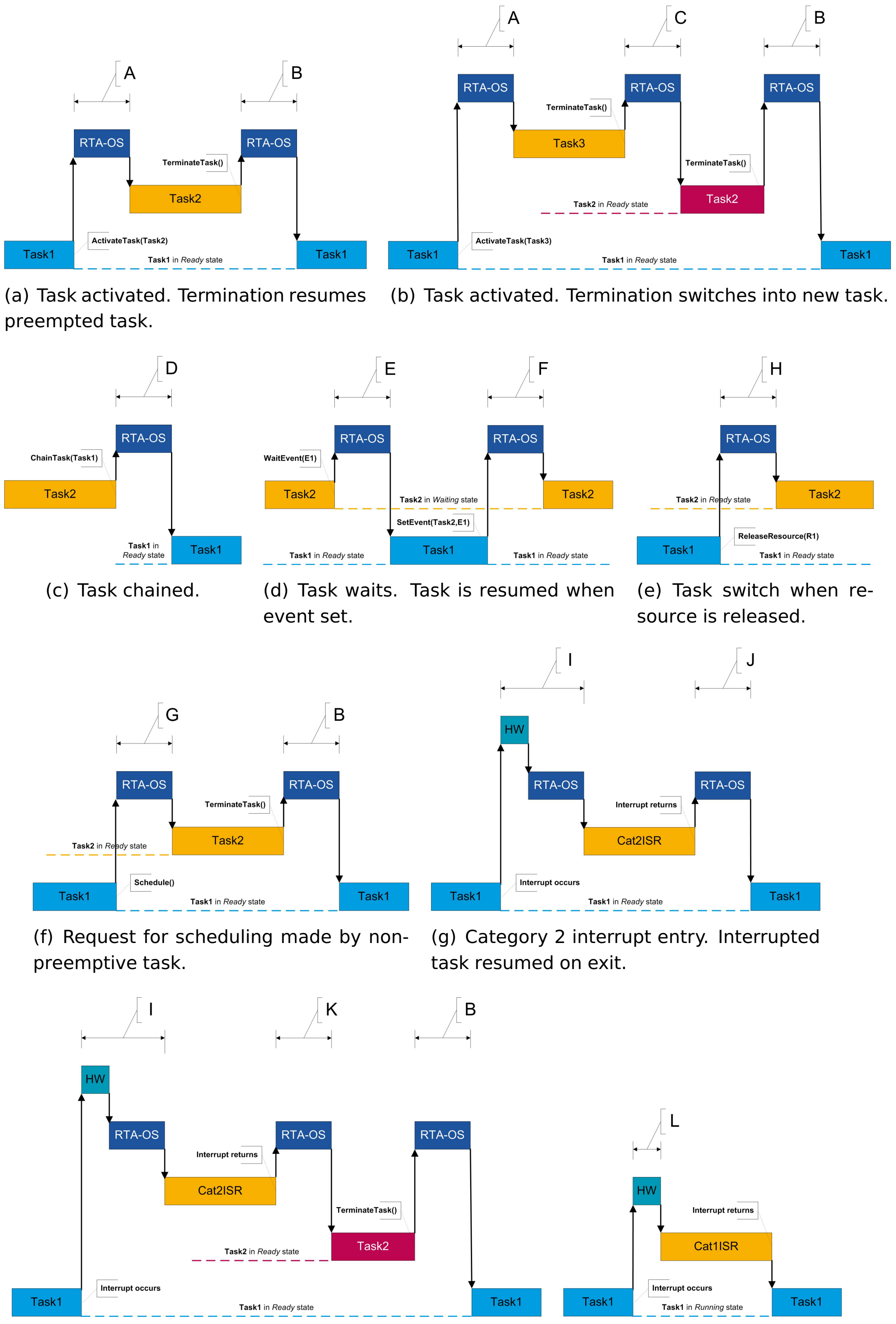
Interrupt latency is the time between an interrupt request being recognized by the target hardware and the execution of the first instruction of the user provided handler function:

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	110	1.38us
Task termination with resume	B	66	825ns
Task termination with switch to new task	C	74	925ns
Chaining a task	D	130	1.62us
Waiting for an event resulting in transition to the WAITING state	E	444	5.55us
Setting an event results in task switch	F	528	6.6us
Non-preemptive task offers a preemption point (co-operative scheduling)	G	104	1.3us
Releasing a resource results in a task switch	H	98	1.22us
Entering a Category 2 ISR	I	84	1.05us
Exiting a Category 2 ISR and resuming the interrupted task	J	102	1.28us
Exiting a Category 2 ISR and switching to a new task	K	110	1.38us
Entering a Category 1 ISR	L	50	625ns



(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 RH850/CS-specific parts of RTA-OS can be found in the following manuals:

***RH850/CS Release Note*.** This document provides information about the RH850/CS 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:

***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`, `.rtaos` and/or `.stc` 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 14	Phone:	+49 711 3423-0
70469 Stuttgart	Fax:	+49 711 3423-2106
Germany	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