RTA-OS TMSx70/TI V3.0.5

Port Guide

Status: Released

DRIVING | EMBEDDED EXCELLENCE

Copyright

Document: 10566-PG-3.0.5 EN-05-2019

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

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 -> R4.3.1) 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 TMSx70/TI port plug-in that customizes the RTA-OS development tools for the Texas Instruments TMS570 with the TI ARM C/C++ 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 TMSx70/TI port and cover:

- how to install the TMSx70/TI port plug-in;
- how to configure TMSx70/TI-specific attributes;
- how to build an example application to check that the TMSx70/TI 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 TMS570, 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.
OS	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 TMSx70/TI 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 TMSx70/TI 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 TMSx70/TI 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 `TMSx70TI_3.0.5`. This contains everything to do with the port plug-in.

Each version of the port installs in its own directory - the trailing `_3.0.5` 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.

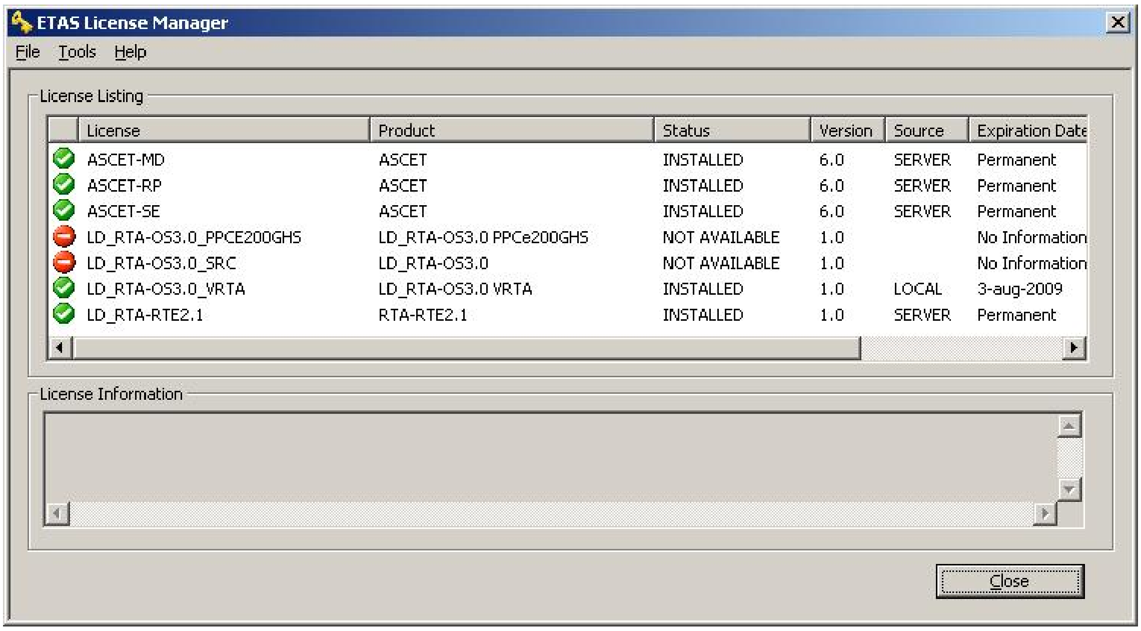
The port directory contains:

TMSx70TI.dll - the port plug-in that is used by **rtaosgen** and **rtaoscfg**.

RTA-OS TMSx70TI Port Guide.pdf - the documentation for the port (the document you are reading now).

RTA-OS TMSx70TI 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`.



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

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

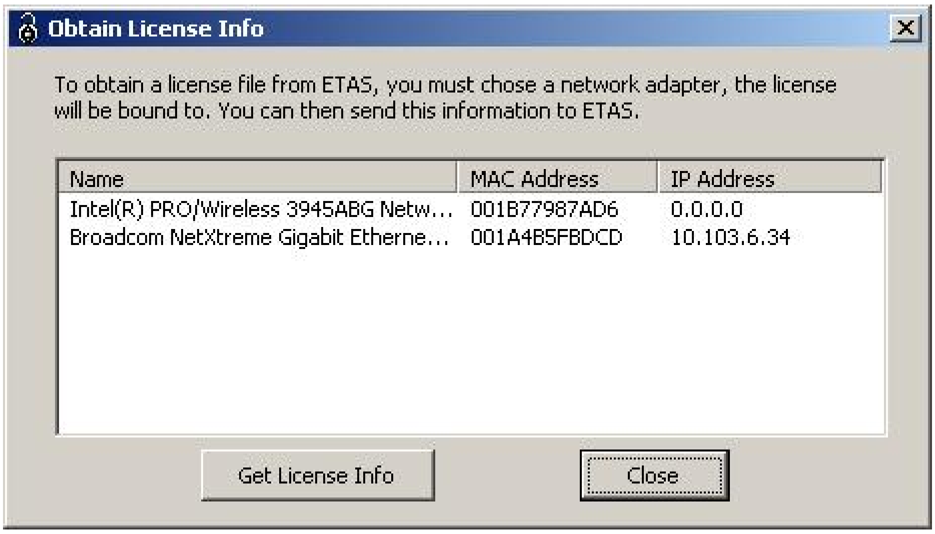
User-named licenses allow the named user (or users) to use RTA-OS on any PC in the network domain.

A user-named license can be issued by ETAS when you provide the Windows username 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.



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

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

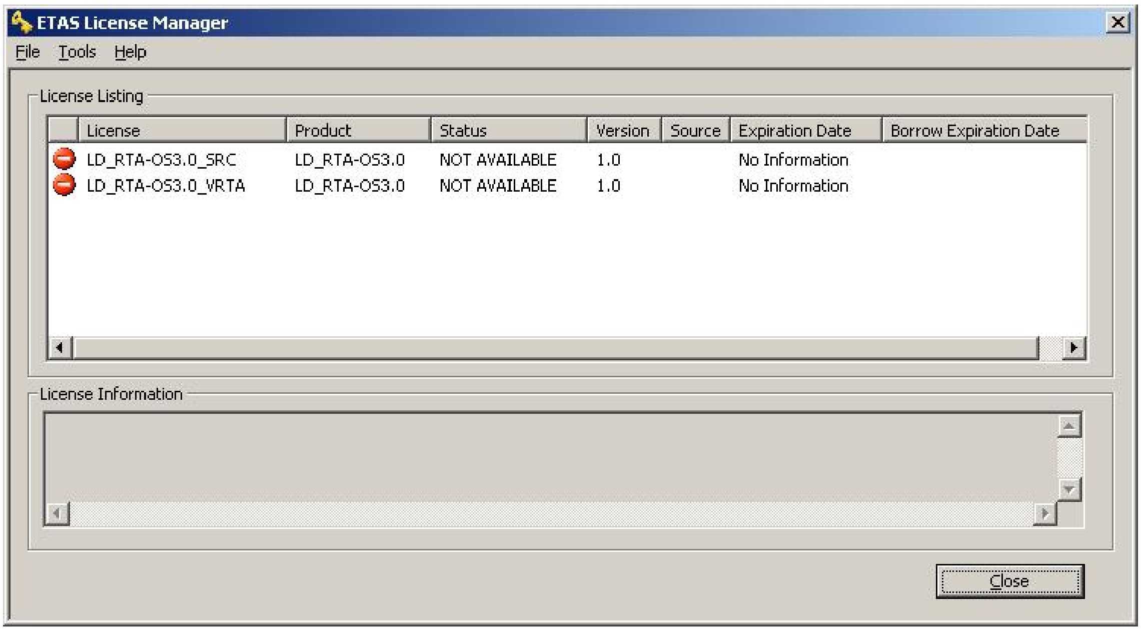
Concurrent licenses are allocated to client PCs by a FLEXnet license server manager working together with a vendor daemon. The vendor daemon for ETAS is called `ETAS.exe`. A copy of the vendor daemon is placed on disk when you install the ETAS License Manager and can be found in:

```
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.



	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

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.

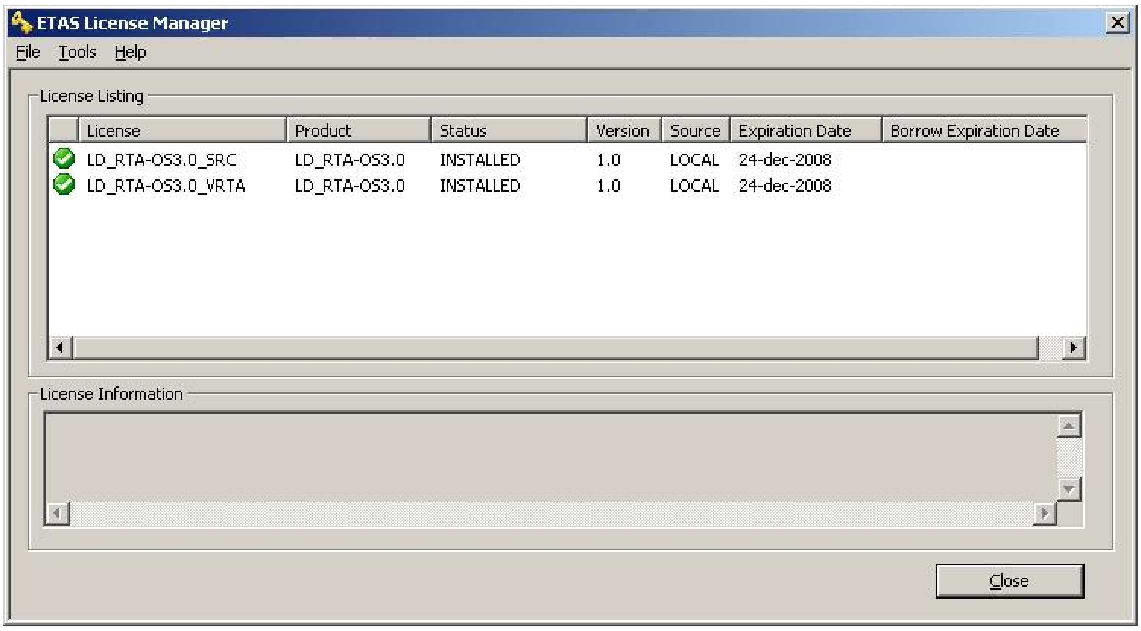
Borrowing a concurrent license

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”



	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

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

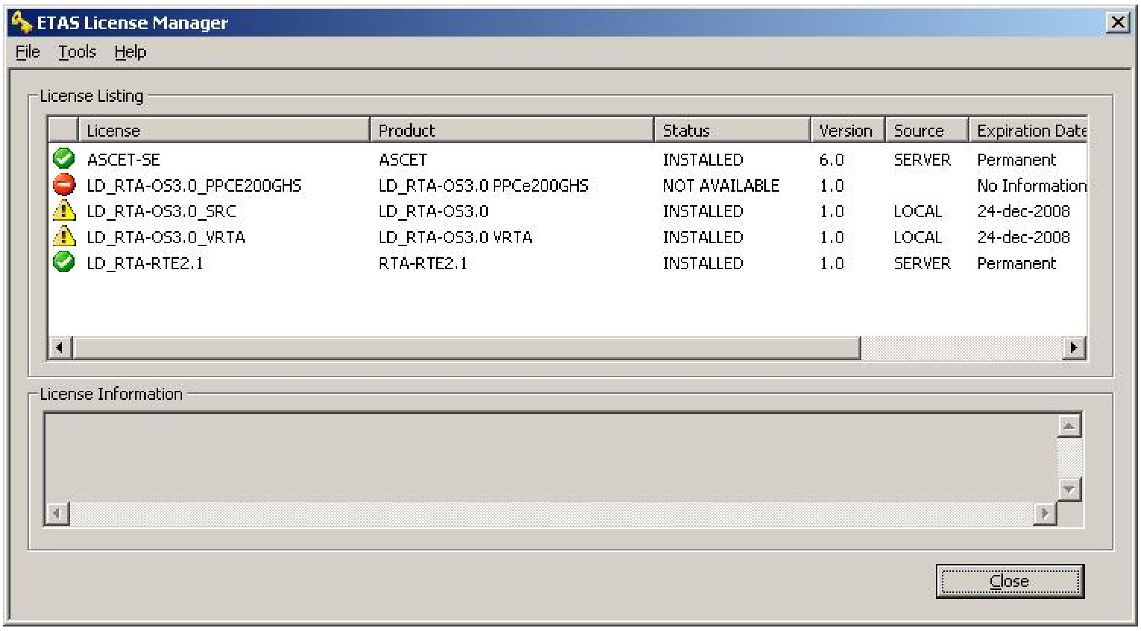
RTA-OS tools will report an error if you try to use a feature for which a correct license key cannot be found. If you think that you should have a license for a feature but the RTA-OS tools appear not to work, then the following troubleshooting steps should be followed before contacting ETAS:

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`.



	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

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 TMSx70/TI 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 TMSx70/TI 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 TMSx70/TI 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

Select ORTI generation for the Lauterbach debugger.

Settings

Value	Description
true	Generate ORTI (default)
false	No ORTI

4.2.7 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.

Settings

Value	Description
true	Generate ORTI stack information
false	No ORTI stack information (default)

4.2.8 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)

4.2.9 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 (default)
false	Do not check

4.2.10 Instruction Set

XML name OsInstructionSet

Description

Select the type of instructions used to compile the OS code. Used to determine the '–code_state' compiler option (see the compiler documentation for more details).

Settings

Value	Description
16	16-bit Thumb mode (default)
32	32-bit ARM mode

4.2.11 Link Type

XML name OsLinkerModel

Description

Select the type of map used in linker samples.

Settings

Value	Description
IntRAM	Code and data in internal RAM
Standalone	Code in internal flash, data in internal RAM (default)

4.2.12 Optimization level

XML name opt_value

Description

Select the level of optimization performed by the compiler. Used to determine the ’-O’ compiler option (see the compiler documentation for more details).

Settings

Value	Description
0	-O0
1	-O1
2	-O2
3	-O3 (default)

4.2.13 Symbolic debug level

XML name symdebug_value

Description

Select the type of any symbolic debug information. Used to determine the ’-symdebug:’ compiler option (see the compiler documentation for more details).

Settings

Value	Description
none	Disable all symbolic debug output (default).’dwarf’=Generate DWARF debug information
dwarf	Generate DWARF debug information
dwarf2	Generate DWARF version 2 debug information
dwarf3	Generate DWARF version 3 debug information
dwarf4	Generate DWARF version 4 debug information

4.2.14 Floating Point Support

XML name float_support

Description

Enable or disable hardware floating-point instructions Used to determine the ’-float_support=’ compiler option (see the compiler documentation for more details).

Settings

Value	Description
none	Software functions are used for all floating-point operations (default)
VFPv3D16	Hardware floating-point instructions for the VFPv3D16 compliant FPU are used for all floating-point operations

4.2.15 Lightweight Floating Point

XML name float_lightweight

Description

Select lightweight floating point support (OS does not preserve floating point registers).

Settings

Value	Description
true	Enable lightweight floating point
false	Disable lightweight floating point (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 TMSx70/TI 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 VIM and CPSR.

Syntax

```
void Os_InitializeVectorTable(void)
```

Description

Os_InitializeVectorTable() initializes the VIM and CPSR according to the requirements of the project configuration, ready for StartOS() to be called. This includes putting the CPU into system mode, enabling interrupts, and transferring the current stack pointer value to the SYS/USR stack.

Os_InitializeVectorTable() should be called before StartOS(). It should be called even if ’Suppress Vector Table Generation’ is set to TRUE.

Example

```
Os_InitializeVectorTable();
```

See Also

StartOS

5.2 Callbacks

5.2.1 Os_Cbk_ClearPendingInterruptSource

Callback routine used to clear the pending status of a specific ISR.

Syntax

```
FUNC(void, {memclass})Os_Cbk_ClearPendingInterruptSource(
    ISRType ISRID
)
```

Return Values

The call returns values of type `boolean`.

Description

The VIM interrupt controller on its own is not able to support the behavior needed to support the AUTOSAR EnableInterruptSource, DisableInterruptSource and ClearPendingInterrupt APIs because the control of the interrupt sources resides in the individual peripherals. For this reason, this target port will invoke certain callbacks that allow the application to decide how to manipulate the interrupt sources.

This callback should clear any pending interrupt for the ISR indicated.

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

Example

```
FUNC(void, OS_APPL_CODE) Os_Cbk_ClearPendingInterruptSource(ISRType ISRID)
    {
  if (ISRID == MyRTI2Int) {
    RTIINTFLAG = RTI_INT2;
  }
}
```

Required when

The callback must be present if EnableInterruptSource, DisableInterruptSource or ClearPendingInterrupt are called in your application.

See Also

ClearPendingInterrupt
DisableInterruptSource
EnableInterruptSource
Os_Cbk_ClearPendingInterruptSource
Os_Cbk_DisableInterruptSource
Os_Cbk_EnableInterruptSource

5.2.2 Os_Cbk_DisableInterruptSource

Callback routine used to disable the interrupt source for a specific ISR.

Syntax

```
FUNC(void, {memclass})Os_Cbk_DisableInterruptSource(
    ISRType ISRID
)
```

Return Values

The call returns values of type boolean.

Description

The VIM interrupt controller on its own is not able to support the behavior needed to support the AUTOSAR EnableInterruptSource, DisableInterruptSource and ClearPendingInterrupt APIs because the control of the interrupt sources resides in the individual peripherals. For this reason, this target port will invoke certain callbacks that allow the application to decide how to manipulate the interrupt sources.

This callback should disable the interrupt source for the ISR indicated.

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

Example

```
FUNC(void, OS_APPL_CODE) Os_Cbk_DisableInterruptSource(ISRType ISRID) {
  if (ISRID == MyRTI2Int) {
    RTICLEARINT = RTI_INT2;
  }
}
```

Required when

The callback must be present if EnableInterruptSource, DisableInterruptSource or ClearPendingInterrupt are called in your application.

See Also

ClearPendingInterrupt
DisableInterruptSource
EnableInterruptSource
Os_Cbk_ClearPendingInterruptSource
Os_Cbk_DisableInterruptSource
Os_Cbk_EnableInterruptSource

5.2.3 Os_Cbk_EnableInterruptSource

Callback routine used to enable the interrupt source for a specific ISR.

Syntax

```
FUNC(void, {memclass})Os_Cbk_EnableInterruptSource(
    ISRType ISRID
)
```

Return Values

The call returns values of type `boolean`.

Description

The VIM interrupt controller on its own is not able to support the behavior needed to support the AUTOSAR EnableInterruptSource, DisableInterruptSource and ClearPendingInterrupt APIs because the control of the interrupt sources resides in the individual peripherals. For this reason, this target port will invoke certain callbacks that allow the application to decide how to manipulate the interrupt sources.

This callback should enable the interrupt source for the ISR indicated.

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

Example

```
FUNC(void, OS_APPL_CODE) Os_Cbk_EnableInterruptSource(ISRType ISRID) {
  if (ISRID == MyRTI2Int) {
    RTISETINT = RTI_INT2;
  }
}
```

Required when

The callback must be present if EnableInterruptSource, DisableInterruptSource or ClearPendingInterrupt are called in your application.

See Also

ClearPendingInterrupt
DisableInterruptSource
EnableInterruptSource
Os_Cbk_ClearPendingInterruptSource
Os_Cbk_DisableInterruptSource
Os_Cbk_EnableInterruptSource

5.2.4 Os_Cbk_GetAbortStack

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

Syntax

```
FUNC(void *,OS_APPL_CODE) 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.

If the value zero is returned, the stack does not get adjusted.

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 location of the untrusted code.

Example

```
#pragma CODE_STATE (Os_Cbk_GetAbortStack, 32);
FUNC(void *,OS_APPL_CODE) Os_Cbk_GetAbortStack(void) {
  /* 64-bit alignment is needed for EABI. */
  static long long abortstack[50U];
  return &abortstack[50U];
}
```

Required when

The callback must be present if ’Enable untrusted stack check’ is configured and there are untrusted OS Applications.

5.2.5 Os_Cbk_IsDisabledInterruptSource

Callback routine used to discover if the source of a specific ISR has been disabled.

Syntax

```
FUNC(boolean, {memclass})Os_Cbk_IsDisabledInterruptSource(
    ISRType ISRID
)
```

Return Values

The call returns values of type boolean.

Description

The VIM interrupt controller on its own is not able to support the behavior needed to support the AUTOSAR EnableInterruptSource, DisableInterruptSource and ClearPendingInterrupt APIs because the control of the interrupt sources resides in the individual peripherals. For this reason, this target port will invoke certain callbacks that allow the application to decide how to manipulate the interrupt sources.

This callback should return TRUE only when the ISR indicated is disabled.

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

Example

```
FUNC(boolean, OS_APPL_CODE) Os_Cbk_IsDisabledInterruptSource(ISRType
    ISRID) {
  if (ISRID == MyRTI2Int) {
    return 0U == (RTISETINT & RTI_INT2);
  }
  return FALSE;
}
```

Required when

The callback must be present if EnableInterruptSource, DisableInterruptSource or ClearPendingInterrupt are called in your application.

See Also

ClearPendingInterrupt
DisableInterruptSource
EnableInterruptSource
Os_Cbk_ClearPendingInterruptSource
Os_Cbk_DisableInterruptSource
Os_Cbk_EnableInterruptSource

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.

Example

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

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 (SP_sys).

Example

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

6 Toolchain

This chapter contains important details about RTA-OS and the TI ARM C/C++ 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.*

RTA-OS supports the TI Compiler v5.2.9 and v18.12.0.LTS compilation tools.

6.1 Compiler Versions

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

6.1.1 TI ARM C/C++ Compiler v5.2.9

Ensure that armcl.exe is on the path

Tested on Tested on v5.2.9

6.1.2 TI ARM C/C++ Compiler v18.12.0.LTS

Ensure that armcl.exe is on the path

Tested on Tested on v18.12.0.LTS

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	armcl.exe
Version	TI ARM C/C++ Compiler v18.12.0.LTS

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.

--code_state=16 Use the 16-bit instruction set (value set by target option)

-mv=7R4 Select the CPU type (value derived from target variant)

-O3 Select the optimization level (value set by target option)

--abi=eabi Use the EABI

--unaligned_access=off Do not generate instructions for data on an unaligned boundary

--auto_inline=0 Prevent inlining of code within the RTA-OS library

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

6.2.2 Assembler

Name	armcl.exe
Version	TI ARM C/C++ Compiler v18.12.0.LTS

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 Linker

Name	armlnk.exe
Version	TI ARM Linker v18.12.0.LTS

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.

`--abi=eabi` Use eabi

`--be32` Link big-endian code in be-32 format

`-w` Warn if an unspecified output section is created

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.

- None

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.

- None

6.2.4 Debugger

Name	Lauterbach TRACE32
Version	Build 104140 or later

Notes on using ORTI with the Lauterbach debugger

When ORTI information for the Trace32 debugger 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. The extra instructions required are incorporated into the `CAT1_ISR` macro.

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 to show this information.

In order to ensure correct operation (as demonstrated in the sample applications) the linker file must create labels holding the start address and size for each stack (one per core). The labels automatically generated by the linker script can be used for this purpose (this approach is demonstrated in the sample applications when built with the `--target_option:ORTI Stack Fill=true` option).

For example, using the following declaration for the stack section:

```
.stack:         RUN_START(_os_stack_base)
                RUN_SIZE(_os_stack_size)
                RUN_END(_os_stack_end)
                {} > MEM
```

The symbols _os_stack_base, _os_stack_size and _os_stack_end will be defined by the linker to be the base address, size and end address (i.e. the address immediately after the last valid address) of the stack.

The fill pattern used by the debugger must be contained with in a 32 bit constant OS_STACK_FILL (i.e. for a fill pattern 0x1234ABCD).

```
const uint32 OS_STACK_FILL = 0x1234ABCD;
```

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

7 Hardware

7.1 Supported Devices

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

Name:	Texas Instruments
Device:	TMS570

The following variants of the TMS570 are supported:

- Generic570LS
- TMS570LS0714
- TMS570LS0914
- TMS570LS1114
- TMS570LS1115
- TMS570LS1224
- TMS570LS1227
- TMS570LS2125
- TMS570LS3137

If you require support for a variant of TMS570 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
CPSR	The CPSR must be set to system mode before calling `Os_InitializeVectorTable()`.
Interrupts	The processor must be configured to use the VIM for interrupt handling (the configuration needed for this is variant specific).
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
CPSR	User code may not change the operating mode.
SP	User code may not change the USR/SYS stack pointer other than as a result of normal program flow.
VIM	User code may not change the VIM channel routing or the mask. VIM RAM can be used.

7.3 Interrupts

This section explains the implementation of RTA-OS's interrupt model on the TMS570.

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 TMS570:

IPL	REQMASK	Description
0	Depends on configuration	User (task) level. No interrupts are masked.
1-62	Depends on configuration	Maskable category 1 and 2 interrupts routed through IRQ.
63	0x0000_0000_0000_0003	Phantom interrupt plus non-maskable category 1 interrupts routed through FIQ.
64	n/a	Non VIM exceptions.

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

7.3.2 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 TMS570. A ✓ indicates that the mapping is permitted and a ✗ indicates that it is not permitted:

Address	Category 1	Category 2
4 exception handler (Undefined instruction)	✓	✗
8 exception handler (SWI/SVC)	✓	✗
12 exception handler (Prefetch abort)	✓	✗
16 exception handler (Data abort)	✓	✗
32 phantom interrupt handler	✓	✗
36, 40 FIQ 0, FIQ 1	✓	✗
44+ VIM interrupt handlers	✓	✓

Note that as the interrupts at addresses 36 and 40 are both FIQs, they must (if configured) have an IPL of 63. Additionally, any VIM interrupt handler given an IPL of 63 will become an FIQ, and thus must be a category 1 ISR.

7.3.3 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
Os_ExceptionVectors	Should be located at absolute address 0x4. Its first entry is the undefined instruction handler.

7.3.4 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 */
}
```

Cortex-Rx CPU exception handlers should be configured as Category 1 ISRs. These handlers should use the CAT1_ISR macro.

7.3.5 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 <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.

7.3.6 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 7.3.4 for further details).

The TMSx70/TI port does not enable unused interrupt channels when the default interrupt is used. `Os_InitializeVectorTable()` masks all unused VIM interrupts. If you intend to trigger a default interrupt then the modification to the VIM registers must occur after calling `Os_InitializeVectorTable()`.

7.3.7 VIM Interrupt Channels

Each Interrupt ReQuest (IRQ) received by the VIM is assigned a channel number that controls its priority, with lower numbered channels having a higher priority. This mapping is controlled by the VIM's CHANCTRL registers that are initialized by `Os_InitializeVectorTable()`.

RTA-OS allocates a single continuous block of these VIM channels. Active VIM interrupts are then allocated a VIM channel from this block based on their relative priority. This:

- Reduces the number of CHANCTRL registers than need to be initialized by `Os_InitializeVectorTable()`.
- Reduces the amount of data that RTA-OS must write to the VIM's REQENACLR and REQENASET registers when raising or lowering the interrupt priority.

For efficiency reasons RTA-OS does not support configurations that have more than 64 active VIM interrupts with 2 of these 64 active VIM interrupts being reserved for handling VIM interrupt channels 0 and 1. As these channels are always enabled by the hardware.

7.4 Memory Model

The following memory models are supported:

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

Apart from some small code sections RTA-OS uses the default compiler memory sections unless modified by the AUTOSAR memmap.h overrides. The non-default code sections all use the prefix 'Os_' (i.e. Os_setjmp)"

7.5 Processor Modes

RTA-OS can run in the following processor modes:

Mode	Notes
Trusted	All trusted code runs in system (SYS) mode.
Untrusted	All untrusted code runs in user (USR) mode.

RTA-OS uses the SVC handler to transfer between Untrusted and Trusted code in applications containing untrusted objects (i.e. ISRs, tasks and functions). This functionality must be supported if a user provided SVC handler is used in such applications.

7.6 Stack Handling

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

RTA-OS manages the USR/SYS stack (via register SP). No IRQ or FIQ stack is used. The RTA-OS function `Os_InitializeVectorTable()` transfers the current stack pointer value to the SYS/USR stack.

If there are Category 1 ISRs attached to the Cortex CPU exception handlers (i.e. Undefined instruction, SVC, Prefetch abort, Data abort and FIQ) then care must be taken that either stack has been assigned to that CPU mode or that the exception handler transfers to a mode that has a valid stack.

7.7 Processor state when calling StartOS()

At StartOS() the following conditions should be true:

- The CPU must be operating in System mode.
- The CPU must be using the System mode stack.
- The IRQ mask bit of the CPSR (i.e. CPSR.I) must be clear.
- The FIQ mask bit of the CPSR (i.e. CPSR.F) must be clear.

These conditions can be applied by calling `Os_InitializeVectorTable()`. If any of these conditions are not met then ShutdownOS() will be called.

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 TMSx70/TI 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	TMS570LS1224 on TMS570LS12x Hercules Development Kit
CPU Clock Speed	16.0MHz
Stopwatch Speed	16.0MHz

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	48
OS-Application	0	0
PeripheralArea	0	0
Resource	8	4
ScheduleTable	16	12
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_ExceptionHandlers	.Os_ExceptionVectors	.Os_IRQ	.bss	.const	.text
ActivateTask						92
AdvanceCounter						4
CallTrustedFunction						22
CancelAlarm						60

Library Module	.Os_ExceptionHandlers	.Os_ExceptionVectors	.Os_IRQ	.bss	.const	.text
ChainTask						80
CheckISRMemoryAccess						36
CheckObjectAccess						94
CheckObjectOwnership						92
CheckTaskMemoryAccess						36
ClearEvent						14
ControlIdle				4		36
DisableAllInterrupts				8		36
DispatchTask						152
ElapsedTime						78
EnableAllInterrupts						32
GetActiveApplicationMode						12
GetAlarm						116
GetAlarmBase						40
GetApplicationID						36
GetCounterValue						26
GetCurrentApplicationID						36
GetElapsedCounterValue						50
GetEvent						14
GetExecutionTime						14
GetISRID						12
GetIsrMaxExecutionTime						14
GetIsrMaxStackUsage						14
GetResource						44
GetScheduleTableStatus						26
GetStackSize						4
GetStackUsage						14
GetStackValue						20
GetTaskID						16
GetTaskMaxExecutionTime						14
GetTaskMaxStackUsage						14
GetTaskState						40
GetVersionInfo						24
Idle						4
InShutdown						2
IncrementCounter						10
InterruptSource					4	132

Library Module	.Os_ExceptionHandlers	.Os_ExceptionVectors	.Os_IRQ	.bss	.const	.text
ModifyPeripheral						102
NextScheduleTable						96
Os_Cfg				536	776	200
Os_Cfg_Counters					728	3032
Os_Cfg_KL						44
Os_ExceptionVectors		64				
Os_Exception_Handler	24					
Os_FIQHandler						32
Os_GetCurrentIMask						12
Os_GetCurrentTPL						28
Os_IRQHandler						112
Os_IRQ_Handler			72			
Os_Misc						68
Os_ResetHandler						4
Os_Stack						12
Os_Trust						8
Os_Wrapper						80
Os_jump						40
Os_vec_init					30	236
ProtectionSupport						24
ReadPeripheral						78
ReleaseResource						56
ResetIsrMaxExecutionTime						14
ResetIsrMaxStackUsage						14
ResetTaskMaxExecutionTime						14
ResetTaskMaxStackUsage						14
ResumeAllInterrupts						32
ResumeOSInterrupts						32
Schedule						64
SetAbsAlarm						72
SetEvent						14
SetRelAlarm						116
SetScheduleTableAsync						34
ShutdownOS						56
StackOverrunHook						6
StartOS						100
StartScheduleTableAbs						92

Library Module	.Os_ExceptionHandlers	.Os_ExceptionVectors	.Os_IRQ	.bss	.const	.text
StartScheduleTableRel						76
StartScheduleTableSynchron						34
StopScheduleTable						48
SuspendAllInterrupts				8		36
SuspendOSInterrupts				8		44
SyncScheduleTable						38
SyncScheduleTableRel						38
TerminateTask						24
ValidateCounter						44
ValidateISR						20
ValidateResource						36
ValidateScheduleTable						36
ValidateTask						40
WaitEvent						14
WritePeripheral						72

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 0 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()`).

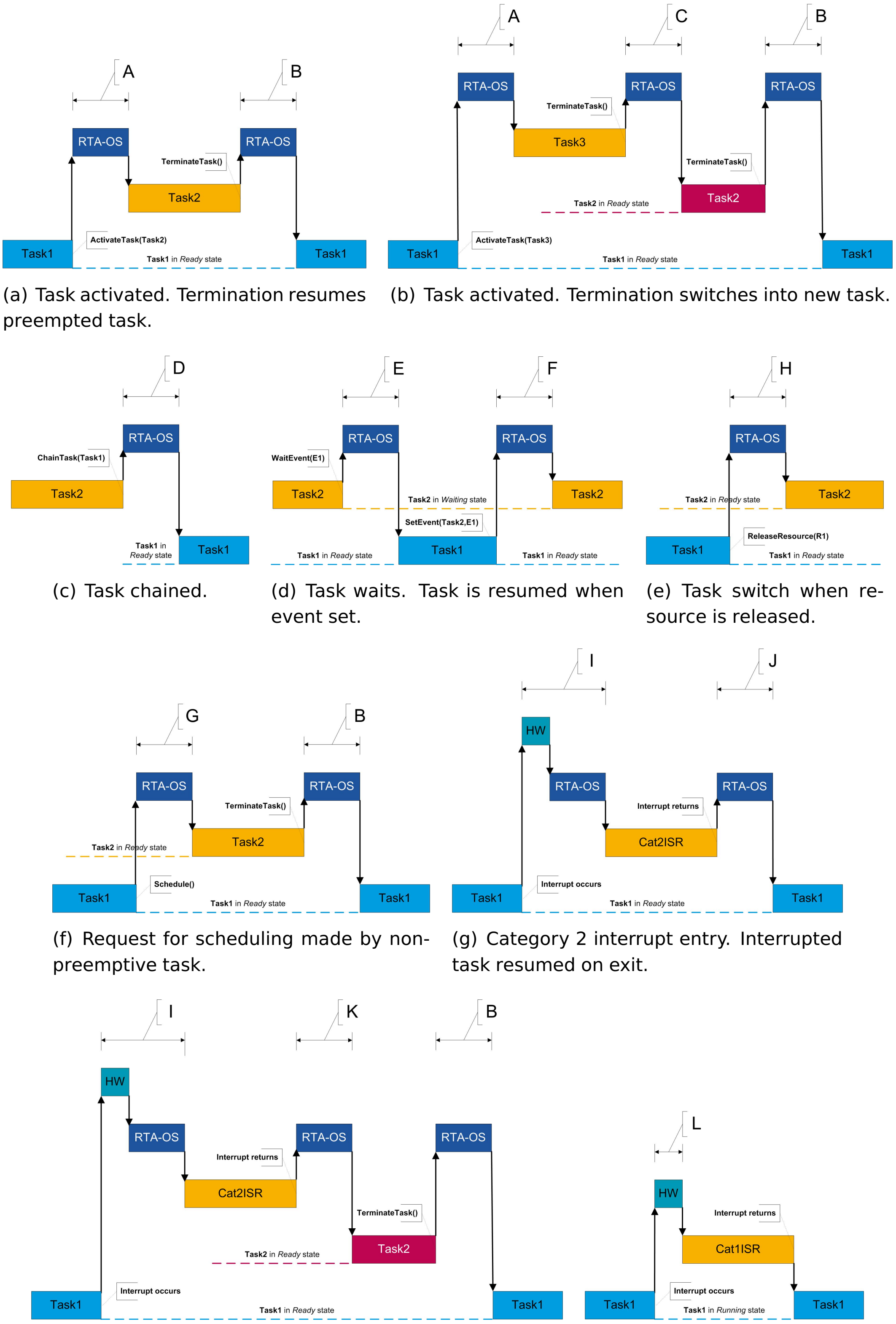
Interrupt latency is the time between an interrupt request being recognized by the target hardware and the execution of the first instruction of the user provided handler function:

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	237	14.8us
Task termination with resume	B	162	10.1us
Task termination with switch to new task	C	191	11.9us
Chaining a task	D	353	22.1us
Waiting for an event resulting in transition to the WAITING state	E	515	32.2us
Setting an event results in task switch	F	619	38.7us
Non-preemptive task offers a preemption point (co-operative scheduling)	G	229	14.3us
Releasing a resource results in a task switch	H	219	13.7us
Entering a Category 2 ISR	I	201	12.6us
Exiting a Category 2 ISR and resuming the interrupted task	J	267	16.7us
Exiting a Category 2 ISR and switching to a new task	K	252	15.8us
Entering a Category 1 ISR	L	151	9.44us



(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 TMSx70/TI-specific parts of RTA-OS can be found in the following manuals:

***TMSx70/TI Release Note*.** This document provides information about the TMSx70/TI 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`, `.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 24	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