ETAS

RTA-OS V6.1.1

Getting Started Guide
Status: Released

DRIVING | EMBEDDED EXCELLENCE

Copyright

Document: 10766-GS-6.1.1 EN-01-2021

Contents

1 Introduction

Welcome to RTA-OS.

This guide describes how to install RTA-OS, how to check that your installation works and what RTA-OS can do.

RTA-OS is a small and fast real-time operating system that conforms to the AUTOSAR R3.x, AUTOSAR R4.0, AUTOSAR R4.1, AUTOSAR R4.2, AUTOSAR R4.3, AUTOSAR R4.4, AUTOSAR R4.5 (R19-11) and OSEK/VDX OS 2.2.3 (ISO 17356) standards. The operating system is configured and built on a PC for use on a target hardware platform.

There are three main parts to RTA-OS:

1. PC Tools

   The RTA-OS tools are target independent and include:

   rtaoscfg a graphical configuration editor for configuring RTA-OS.

   rtaosgen a command-line tool for generating an RTA-OS kernel library based on your configuration. This includes the parts of the RTA-OS kernel that are shared across all targets.

2. Port Plug-ins

   RTA-OS port plug-ins (RTA-OS Targets) are used to customize RTA-OS for different target microcontroller and compiler combinations. The PC tools support multiple targets and you can switch between them at configuration time.

   Your RTA-OS tools installation includes a port plug-in for a *Virtual Target* called VRTA. VRTA allows you to build RTA-OS applications that run on your PC without the need for embedded target hardware.

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.

The separate *User Guide*, which can be found in the same directory as this *Getting Started Guide*, provides more detailed information on using RTA-OS, but you should read this *Getting Started Guide* before you install RTA-OS and before you read any other manuals.

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.

2 Installing RTA-OS

2.1 Preparing to Install

Before installing RTA-OS check that all the items have been delivered. You should have been supplied with an RTA-OS tools installation and, if you plan to run RTA-OS on embedded target hardware, one (or more) port installations.

RTA-OS is provided as a downloadable electronic installation image. You will have been provided with access to the download when the product was purchased. You may optionally have requested an installation CD which will have been shipped to you. The downloadable image and installation CD contain identical content.

The RTA-OS tools are supplied as a single installer. Each embedded port is supplied as a separate installer. A working version of RTA-OS requires the installation of the tools and at least one port.

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 8
- Windows 10

Integration Guidance 2.1:*The tools provided with RTA-OS require Microsoft's .NET Framework* v2.0 *(included as part of .NET Framework* v3.5*) and* v4.5.2 *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.x* will occur over a period of time for performance and maintenance reasons.

You will need to obtain a license key to run RTA-OS after installation. Licensing is managed by the ETAS License Manager. For further details see Section 2.4.

2.2 Installation

The RTA-OS installer will install:

1. The PC tools for RTA-OS.
2. The RTA-OS documentation (optional).

2.2.1 Tools

You must install the RTA-OS PC tools first before installing VRTA or any port plug-ins - this will create the necessary folder structures.

1. Either
   - Run the executable image; or
   - Insert the RTA-OS CD into your CD-ROM or DVD drive.
     If the installation program does not run automatically, start the installation manually by navigating to the root folder of your CD/DVD drive and running `autostart.exe`.
2. Follow the on-screen instructions to install RTA-OS.

The default location for an RTA-OS installation is `C:\ETAS\RTA-OS`. The location can be changed during the installation process.

Once installation is complete, you will be asked whether the following additional components should be installed:

1. The port plug-in for the VRTA *Virtual Target*. See Section 2.2.2.
2. The ETAS License Management software. See Section 2.4.

Integration Guidance 2.2:*You must install at least one port plug-in to be able to use the RTA-OS development tools. It is recommended that you accept the default installation of the VRTA port plug-in.*

Integration Guidance 2.3:*Installation of the additional components can be carried out later by running the installers found within sub-folders of the installation folder.*

2.2.2 VRTA

VRTA allows you to create AUTOSAR applications that run on a Windows PC and emulate the behavior of applications running on real embedded hardware - including multicore hardware. You you can develop, test and debug complete RTA-OS configurations without needing access to embedded tools or target hardware.

A default installation of RTA-OS installs a port plug-in for VRTA into `C:\ETAS\RTA-OS\Targets\VRTA_n.n.n`. VRTA also installs a set of run-time libraries and executables that provide run-time support for VRTA-based applications into `C:\ETAS\RTA-OS\bin`.

Integration Guidance 2.4:*The default installation path for the port-plugin may be different from the tools installation path! You may wish to change the installation path for the port-plugin so that it matches that of the tools.*

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

```
rtaosgen --target_include:<target_folder>
```

Integration Guidance 2.5:*The VRTA run-time libraries and executables must be on your path for VRTA applications to run. It is recommended that you add `<install dir>\bin` to your Windows PATH environment variable. If you have installed RTA-OS on a PC that also includes an installation of RTA-OSEK then you must ensure that the path to RTA-OS is placed before the path to RTA-OSEK.*

Compilers for VRTA

VRTA is configured to work with several popular PC C/C++ compilers, including:

- MinGW / gcc
- Microsoft Visual Studio 2005
- Microsoft Visual Studio 2008
- Microsoft Visual Studio 2010
- Microsoft Visual Studio 2012
- Microsoft Visual Studio 2013
- Microsoft Visual Studio 2015
- Microsoft Visual Studio 2017

These compilers are not supplied as part of the RTA-OS installation.

The MinGW (Minimalist GNU for Windows) C compiler is available from `http://www.mingw.org`. This is the default compiler used for building VRTA-based applications.

Microsoft provides an 'Express' Edition of the Visual Studio C++ compiler which is freely downloadable from `https://visualstudio.microsoft.com/`.

2.2.3 Embedded Port Plug-ins

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

1. Either
   - Run the executable image; or
   - Insert the RTA-OS 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 folder of your CD/DVD drive and run `autostart.exe` to start the setup.
2. Follow the on-screen instructions to install the port plug-in.

Integration Guidance 2.6:*The default installation path for the port-plugin may be different from the tools installation path - RTA-OS is fully compatible with port plugins from earlier releases. You may wish to change the installation path for the port-plugin so that it matches that of the tools.*

Port plug-ins can be installed into any location, but using a non-default folder requires the use of the `--target_include` argument to both **rtaosgen** and **rtaoscfg**.

2.3 What is Installed?

After installing RTA-OS, a number of new folders will be created. The location of these folders depends on where you decided to install the files. The default location is `C:\ETAS\RTA-OS`.

Folder	Contents
`Bin`	Executable programs.
`Bin\Licenses`	License signature files that tell the ETAS License Manager which license features are required for the installed programs. The licenses themselves are installed elsewhere, see Section 2.4.
`Bin\plugins`	GUI plug-ins for the **rtaoscfg** configuration tool.
`Documents`	User documentation.
`Targets`	Port plug-ins - one per sub-folder. All port-specific parts, including the *Target/Compiler Port Guide* for the port, are included in the sub-folder.
`VRTA`	This folder contains the installer for the VRTA port plug-in.
`LicenseManager`	This folder contains the installer for the ETAS License Manager.

Each port plug-in is installed in its own sub-folder under `Targets`. The folder contains the port plug-in DLL and all port-specific documentation.

All user documentation is distributed in PDF format and 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`.

The folders have the following naming convention:

<Target><CompilerVendor>_n.n.n[1]

The suffix n.n.n gives the version of the port plug-in. It is therefore possible to have multiple different versions of a port plug-in installed at the same time.

If multiple versions of the same port plug-in are installed, then RTA-OS will use the latest version (i.e. the highest numbered version) unless a specific version is given in the configuration file.

2.4 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.

2.4.1 Installing the ETAS License Manager

Integration Guidance 2.7:*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 2.4.2 and 2.4.3 for further details)

[1]The VRTA port plug-in is an exception to this naming convention, being called VRTA_n.n.n, because it can support multiple compilers.

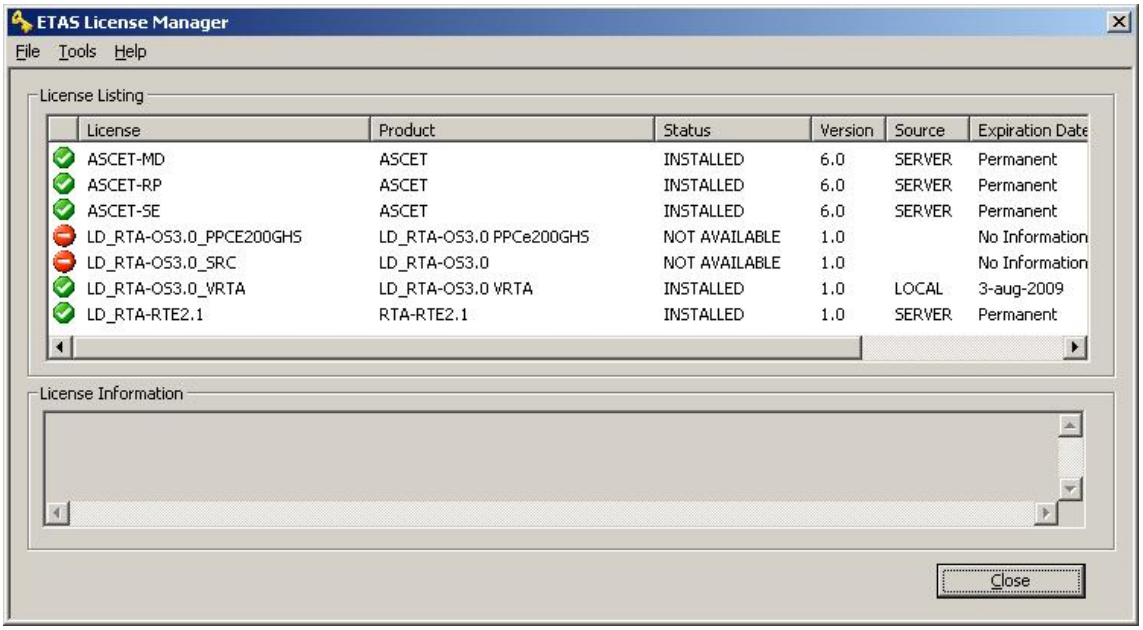

Figure 2.1: The ETAS License manager

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**.

2.4.2 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 6.

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.

Figure 2.2: Obtaining License Information

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.

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.

2.4.3 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.

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
```

2.4.4 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.

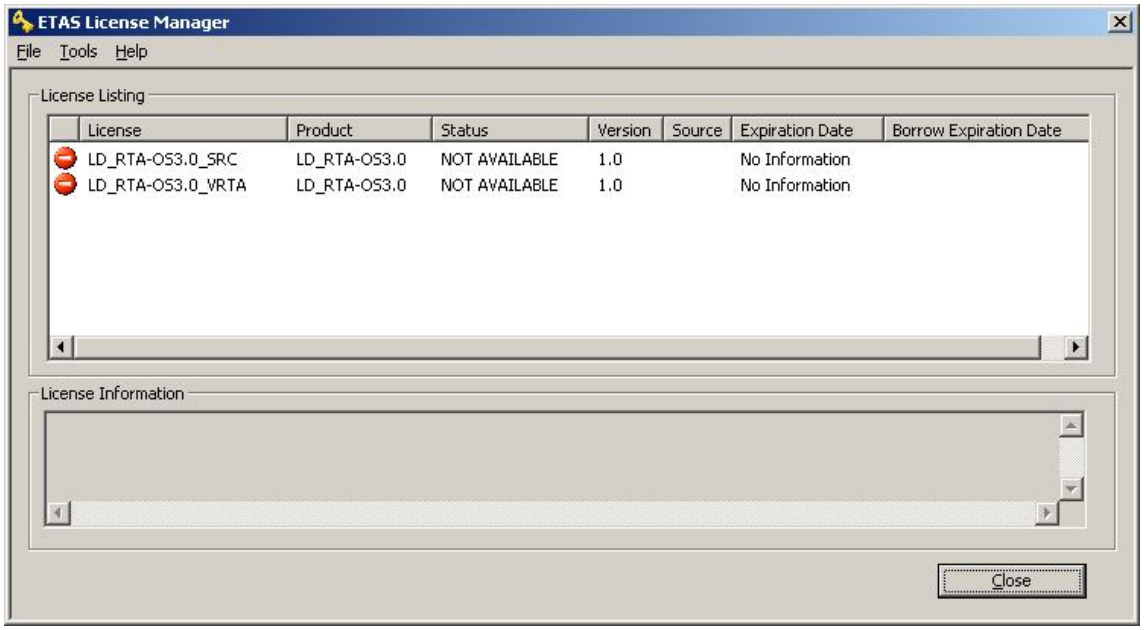

Figure 2.3: Unlicensed RTA-OS Installation

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.

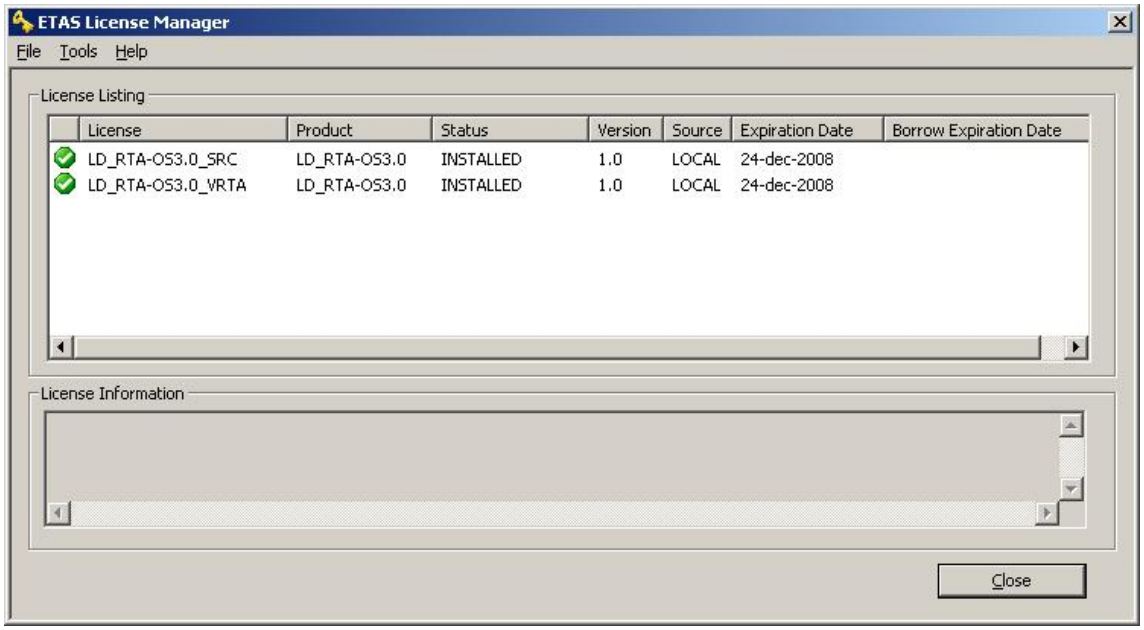

Figure 2.4: Licensed features for RTA-OS

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”.

2.4.5 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.

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

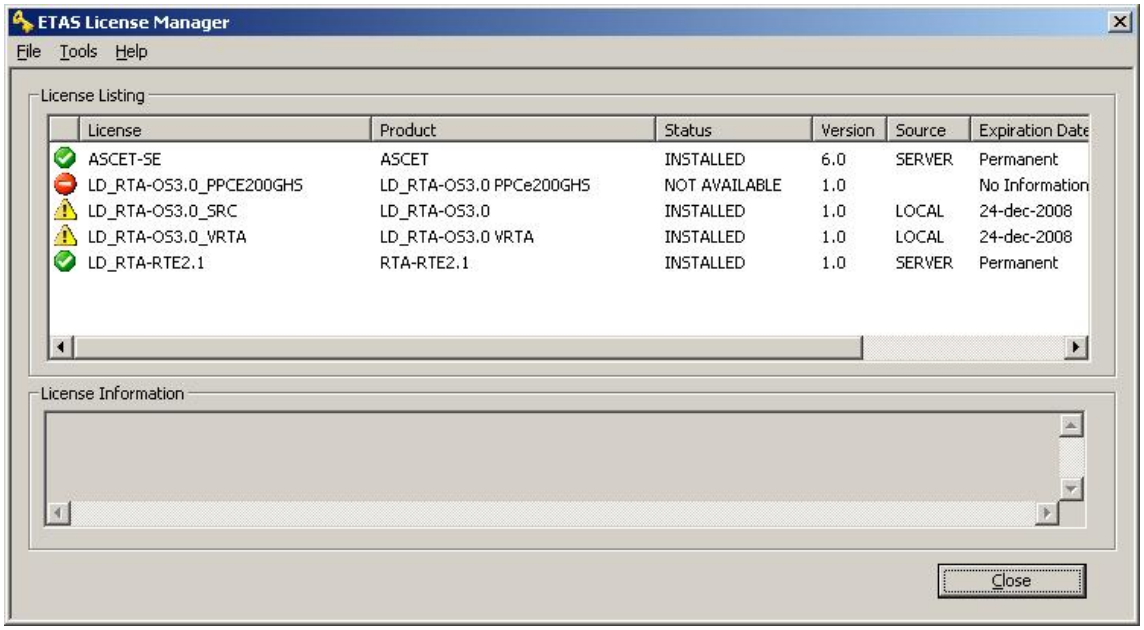

Figure 2.5: Licensed features that are due to expire

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 6.1. You must provide the contents and location of your license file and your Ethernet MAC address.

2.5 Verifying your Installation

Now that you have installed RTA-OS and have obtained and installed a valid license key you can check that things are working. You can verify that the PC tools installation has worked by running the RTA-OS code generator from the command line:

```
rtaosgen --target:?
```

If the installation has been successful, the tool will run and return a list of available targets, their versions and their associated variants. For example:

```
rtaosgen --target:?
RTA-OS Code Generator
Version x.x.x.xxxx, Copyright © ...
Available targets:
 PPCe200GHS_x.x.x [MPC5516z1,MPC5516z0,...]
 TriCoreHighTec_x.x.x [TC1732,TC1736,...]
 VRTA_x.x.x [MinGW,VS2005,VS2008,VS2010]

Errors: 0, Warnings: 0.
```

If the target port plug-ins have been installed to a non-default location, then the `--target_include` argument must be used. The above example is then rewritten as:

```
rtaosgen --target_include:<tgtDir> --target:?
```

2.6 Running RTA-OS from a Network Drive

Integration Guidance 2.8:*The RTA-OS tools are Windows .NET applications. By default a .NET assembly (application)* cannot *be run from a network drive due to the default Windows security policy. This means that the RTA-OS tools will not run from network drive without additional configuration.*

You can modify the Windows security policy to allow RTA-OS to run from a network in two ways using either:

- The Windows Control Panel; or
- The Microsoft command-line tool **CasPol.exe** which is distributed with the Microsoft .NET framework Control Panel

The Control Panel method is simple, but offers less control than that provided by the command-line method. You should consult your IT support department for recommendations on what levels of security are allowed at your workplace.

2.6.1 Using the Control Panel

Navigate to **Control Panel → Administrative Tools →.NET Framework Configuration x.x → Runtime Security Policy → Adjust Zone Security** and adjust the setting for “Local Intranet” to “Full”.

2.6.2 Using the Command-line

The **CasPol.exe** program is used to change the security policy for the share from which the application will be run. It can be found in `C:\WINDOWS\Microsoft.NET\Framework\vn.n.nnnnn\CasPol.exe`. You need to perform the following steps:

1. Disable policy change prompt using **CasPol.exe -pp off**. The `-pp off` option disables the policy change prompt.

2. Change policy for share using **CasPol.exe -m -ag 1.2 -url "file:<share_name>/*" FullTrust** where:

Argument	Description
-m	Operate on machine level
-ag 1.2	Add a code group under group 1.2. In the default policy, Group 1.2 is the LocalIntranet group, so the new code group that we're creating will only be checked if the file comes from the intranet.
-url:...	Match any file within the specified URL, where <share_name> should be replaced with the name of the share, e.g. `\networkdrive\ETAS\RTA-OS`.
FullTrust	The permission set to grant assemblies that match the code group. In this case, FullTrust.

3. Re-enable policy change prompt using **CasPol.exe -pp on**. The `-pp off` option disables the policy change prompt.

3 Developing Applications with RTA-OS

The process for using RTA-OS in your application involves three steps:

1. Configure the features of the OS you want to use
2. Generate a customized RTA-OS kernel library
3. Use the OS in your application

The following sections explain these steps in more detail. However, if you are eager to get started with RTA-OS then you can skip ahead to Chapter 4 which explains how to build and run the sample applications provided with each port plug-in.

3.1 Configuration

RTA-OS is statically configured. This means that every task and interrupt needed over the entire operation of your application must be declared at configuration time, together with any critical sections, synchronization points, counters etc.

Configuration data is stored in XML files that conform to the AUTOSAR ECU Configuration Definition standard.

RTA-OS includes a graphical configuration editor (shown in Figure 3.1) called **rtaoscfg** to assist in the creation of RTA-OS configuration files. **rtaoscfg** accepts any AUTOSAR XML file as input and allows the OS-specific parts of the configuration to be edited. If the input file contains both OS and non-OS specific configuration then only the OS configuration will be modified. Further details about how to use **rtaoscfg** are provided in the *User Guide*.

3.1.1 Using OSEK OIL Files for Configuration

To assist in the migration from existing RTA-OSEK applications, RTA-OS includes an OIL Import assistant, accessed from the Assistants menu of the GUI, as shown in Figure 3.2

The *OIL Import Assistant* itself (shown in Figure 3.3) enables OIL files from a previous RTA-OSEK application to be converted to AUTOSAR XML.

The only mandatory input to the *OIL Import Assistant* is the input OIL file. The optional parameters are:

Output directory

Output will be generated in the directory containing the input OIL file unless an optional output directory is supplied.

Project name

By default, the generated files are derived from the input filename, this can be overridden here;

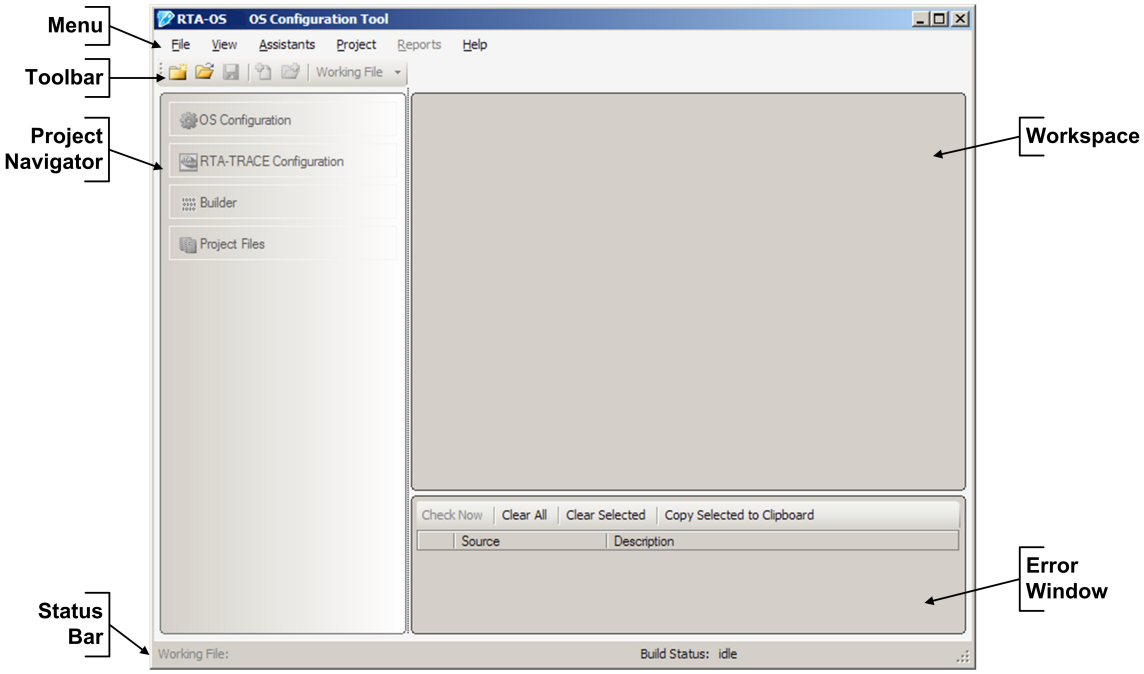

Figure 3.1: The rtaoscfg configuration tool

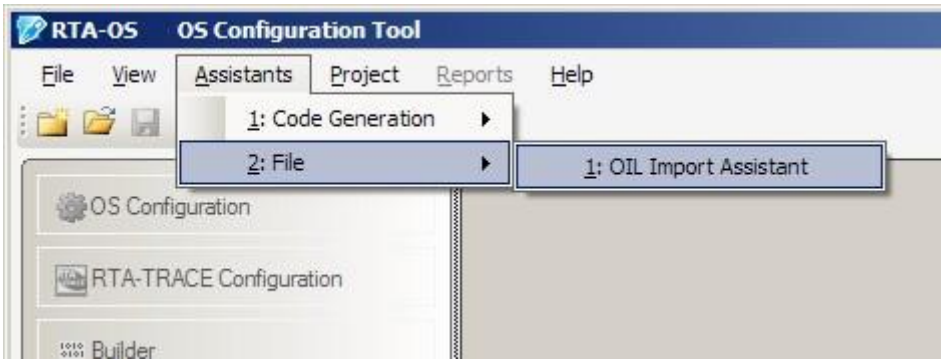

Figure 3.2: The OIL Import Assistant Menu

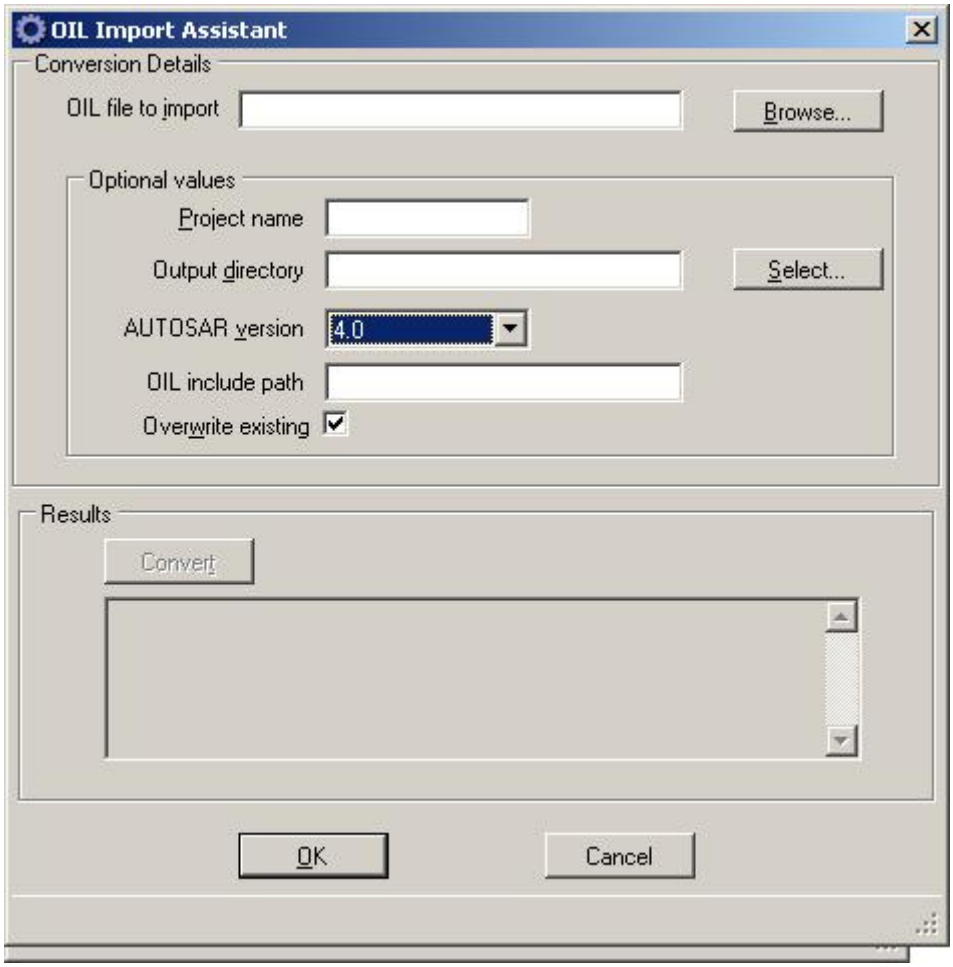

Figure 3.3: The OIL Import Assistant

AUTOSAR version

The AUTOSAR OS version can be specified if compliance to a specific version of the XML is required;

OIL include path

The input OIL file may make use of *include* directives which refer to OIL files in other directories. This argument should be used to refer to the directories including the referenced files. Multiple directories should be separated by semicolons.

To convert an OIL file, the *Convert* button should be pressed - the results of the conversion will be displayed in the *Results* area. Once an OIL file has been converted, it can be loaded into the GUI by pressing the *Ok* button - pressing *Cancel* will return to the GUI without loading the converted file.

3.2 Library Generation

Before RTA-OS can be used in an application, an RTA-OS kernel library and associated header files must be generated. The kernel library is generated using the command-line tool **rtaosgen** which carries out the following steps:

1. Analyze the XML configuration and automatically optimize the RTA-OS kernel so that it contains only those features which are used. This makes RTA-OS as small and efficient as possible.
2. Customize the optimized kernel for the chosen target using information provided by the target plug-in.
3. Build the kernel library using the same third party toolchain used for the application. This guarantees toolchain compatibility between the RTA-OS kernel library and the application code.

3.2.1 Preparing the Toolchain

To build a kernel library, **rtaosgen** needs access to the target toolchain. **rtaosgen** knows how to run the target compiler, assembler, linker and librarian as well as which options to use. You need only worry about two things:

1. Your toolchain should be accessible on your PATH, or you can use the `--env` command-line option.
2. Your toolchain must be compatible with RTA-OS.

You can find out if your compiler is on your PATH by opening a Windows Command Prompt and typing `echo %PATH%`. The compiler's executable folder should be listed. If the compiler's executable folder is not listed, it can be added by typing:

```
set PATH=%PATH%;<PathToCompilerExecutable>
```

To check whether the toolchain is compatible with RTA-OS, consult the *Target/Compiler Port Guide* for the your port.

3.2.2 Running **rtaosgen**

rtaosgen can be invoked from anywhere a Windows executable can be called from (e.g. Windows command prompt, batch file, makefile, Ant script etc.). **rtaosgen** can also be invoked from within the builder pane of **rtaoscfg** if you prefer working within a graphical environment.

3.2.3 Building the library

As an AUTOSAR Basic Software Module, RTA-OS requires access to the standard AUTOSAR header files. You must therefore include the path to the location of the AUTOSAR standard header files when invoking **rtaosgen**. For example, to build a library for the *Hello World* example application for an RTA-OS target you should type:

```
rtaosgen --include:<PathToAutosarHeaderFiles> HelloWorld.xml
```

If you do not have access to AUTOSAR header files (for example, if you are using RTA-OS for application development independently of an AUTOSAR system) then **rtaosgen** can generate them automatically for you. You have to tell RTA-OS where the generated headers can be found. This is typically in a folder named `Samples\Includes`. The command in this case would be:

```
rtaosgen --samples:[Includes] --include:Samples\Includes HelloWorld.xml
```

During the build process, **rtaosgen** will check for the presence of the files `<projectname>_prebuild.bat` and/or `<projectname>_postbuild.bat`. If these are present in the project directory (see RTA_PROJECT_BASE below), then they will be run just before (pre) or just after (post) the build. They get run from the project directory, regardless of the directory where you start **rtaosgen**. A typical use of a postbuild step is to rename one or more of the generated files, or to commit the files to source control.

Within these `.bat` files, **rtaosgen** provides the following environment variables that you can use as needed:

- `RTA_PROJECT_BASE`: This is the project directory - normally the directory where the `.rtaos` file lives.
- `RTA_PROJECT_NAME`: This is the project name - normally the name of the `.rtaos` file, minus the extension.
- `RTA_WORKING_FOLDER`: This is the directory from which the build process runs - normally the directory you are in when you run **rtaosgen**.

3.2.4 Build Messages

rtaosgen generates four classes of messages during execution:

Information

These messages tell you useful things about the configuration, for example how many tasks you have configured.

Warning

These messages warn you that your configuration might not behave as you expect.

Error

These messages tell you that there is something wrong with your configuration. **rtaosgen** will stop processing your configuration at a convenient point and no output files will be generated.

Fatal

You will get at most one fatal message. It tells you that there is something fundamentally wrong with either your configuration or **rtaosgen**. **rtaosgen** stops immediately.

3.2.5 Generated Files

When **rtaosgen** runs without generating any errors or fatal messages the following files will have been generated:

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.

There may be other files generated depending on your chosen port. Consult the relevant *Target/Compiler Port Guide* for further information.

3.3 Integration

3.3.1 Accessing the OS in your Source Code

Each C source file must include `Os.h` if it makes calls to RTA-OS. The header file is protected against multiple-inclusion.

RTA-OS makes no extra demands about source code organization (i.e. it is possible to have more than one task and interrupt per source file).

3.3.2 Implementing Tasks and ISRs

Tasks

There must be an implementation of each task declared at configuration time. Each task needs to be marked using the `TASK(x)` macro. Tasks typically have the following structure:

```
#include <Os.h>
TASK(MyTask){
  /* Do something */
  TerminateTask();
}
```

Category 2 ISRs

Each declared Category 2 ISR also needs to be implemented. These are marked by the `ISR(x)` macro:

```
#include <Os.h>
ISR(MyISR){
  /* Do something */
}
```

Integration Guidance 3.1:*A Category 2 ISR handler must not execute a return from interrupt call - RTA-OS does this automatically. Depending on the behavior of interrupt sources on your target hardware you may need to clear the interrupt pending flag as part of your ISR. Please consult the hardware documentation provided by your target processor vendor for further details.*

Category 1 ISRs

Each declared Category 1 ISR also needs to be implemented. Your compiler will use a special convention for marking a C function as an interrupt. RTA-OS provides a macro that expands to the correct directive for your compiler. Your Category 1 handler will therefore look something this:

```
#include <Os.h>
CAT1_ISR(MyCat1ISR) {
  /* Do something */
}
```

Starting the OS

RTA-OS does not take control of your hardware after it comes out of reset so any initialization which is required can be done without interference from the OS - just as if you were developing an embedded application that doesn't use an operating system. Once hardware initialization is complete, the OS is started by calling `StartOS()`. This call is usually made in the `main()` function.

RTA-OS provides the `OS_MAIN()` macro which can be optionally used to mark the `main()` function in a portable way as shown in the following example[1].

```
#include <Os.h>
OS_MAIN(){
  /* Initialize target hardware */
  ...
  /* Do any mode management, pre-OS functions etc. */
  ...
  /* Use RTA-OS to initialize interrupts */
  Os_InitializeVectorTable();
  StartOS();
  /* Call does not return so you never reach here */
}
```

3.3.3 Interacting with the RTA-OS Kernel

Interaction with RTA-OS is through kernel API calls. You can find a complete list of calls in the *Reference Guide*.

3.3.4 Compiling and Linking

When compiling an application, you must make sure that `Os.h` and `Os_Cfg.h` are on your compiler include path. Similarly, when you link your application `RTAOS.<lib>` must be on your linker library path.

You must also ensure that any tool options you use are compatible with those expected by the RTA-OS library. A full list of mandatory and prohibited toolchain options is provided in the *Target/Compiler Port Guide* for your port.

3.3.5 Common Problems

If the compiler generates errors related to RTA-OS then you should check the following:

- `Os.h` and `Os_Cfg.h` are on the compiler include path
- The kernel library has been rebuilt after making any changes to the configuration
- `RTAOS.<lib>` is on your linker's library path

3.3.6 Downloading to your Target

The output of the linker is normally a binary file in some well-known format (e.g. a.out, COFF, ELF or IEEE695). These can be read by debuggers, in-circuit emulators or in-circuit programming equipment, although in some cases it is necessary to convert the output from this to a form (such as S-Records or Intel Hex) that can be transmitted to a simple boot monitor on the target over a serial link. Tools to do this are usually supplied with your development environment. Consult the documentation on your target platform and development toolchain for details of how to program applications into non-volatile memory.

[1] For most ports, `OS_MAIN` will expand to **`void`** `main(`**`void`**`)`. However some embedded compilers do not allow the `main` program to return **`void`** so an alternative expansion may be required.

4 Sample Applications

Each port plug-in is supplied with a set of ready-to-run sample applications that demonstrate key features of RTA-OS. The sample applications show you how the OS is configured, the library is generated and how application code is linked against the library.

The following applications are provided with each port plug-in[1]:

Application	Contents
Hello World	Basic tasks, interrupts, counters, alarms.
Clive Devices	Basic tasks, interrupts, counters, alarms, schedule tables.
Pizza Pronto	Extended tasks, interrupts, alarms, counters.

Integration Guidance 4.1:*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.*

4.1 Getting Started

The sample applications and **rtaosgen** need to use your compiler toolchain and expect to find it on the Windows path. You can add the tools to your path using the Windows command prompt as follows:

```
set PATH=<PathToCompilerExecutable>;%PATH%
```

It is also recommended that the RTA-OS tools are also added to your path:

```
set PATH=<PathToRTAOS>\bin;%PATH%
```

4.1.1 Creating Sample Applications from the Command-Line

Sample applications can be generated for a specific target and target variant. Recall from Section 2.5 that the list of supported targets and variants for your installation can be obtained using the command:

```
rtaosgen --target:?
```

To access the sample applications:

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

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

[1] There may be other applications provided by your port

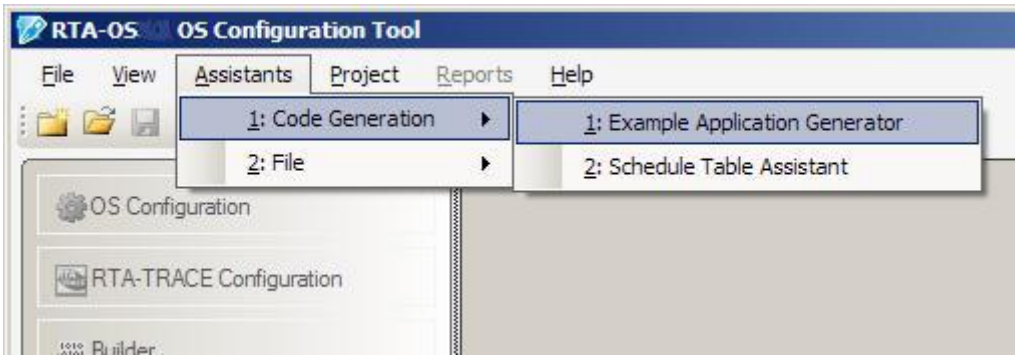

Figure 4.1: RTA-OS Assistants Menu

The <variant> is optional. If it is omitted then sample applications will be generated for the default target.

This generates a new sub-folder called Samples\Applications. Inside this folder are sub-folders for each sample application using the default settings for the chosen target and variant. If you ever need to get a clean set of sample applications, then you can repeat this process in a new working folder.

For example, to generate the sample applications for the VRTA target using the VS2008 variant, issue the command

```
rtaosgen --samples:[Applications] --target:[VS2008]VRTA
```

It is also possible to generate individual sample applications. The following commmand will generate the *PizzaPronto* application for the MinGW variant of VRTA

```
rtaosgen --samples:[Applications\PizzaPronto] --target:[MinGW]VRTA
```

4.1.2 Creating Sample Applications from the GUI

The RTA-OS GUI is capable of generating the sample applications using the *Example Application Generator Assistant*.

The Assistant is launched from the *Assistants* entry in the GUI's menu; see Figure 4.1. Choosing **Assistants → Code Generation → Example Application Generator** will, after a short pause whilst information about installed targets is looked up, show the *Example Application Generator Assistant* as shown in Figure 4.2.

After the target, variant and sample-application are selected and an output directory is entered (for the generated code), the **Generate** button is enabled and the application can be created. Once the application has been created, it can be built (and subsequently run), loaded into the GUI, or simply left on disk.

If your toolchain is not found when building an application, the path can be added to the *Additional Path* argument.

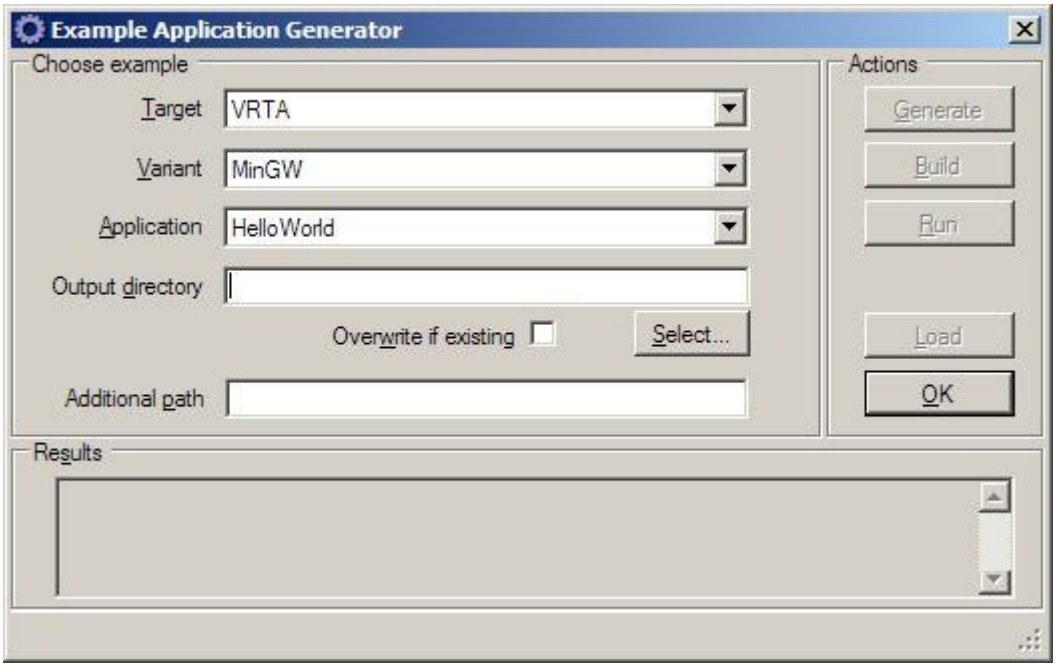

Figure 4.2: RTA-OS Example Application Generator Assistant

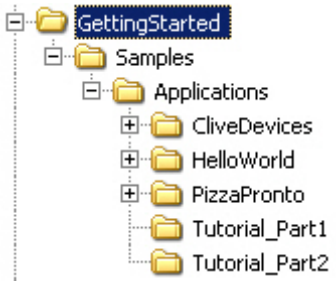

Figure 4.3: Sample Applications Folder Structure

4.2 Sample Application Structure

Figure 4.3 shows the structure of the sample applications folder for the VRTA port assuming that the working folder you created was called `C:\GettingStarted`. Note that other ports may contain different sample applications.

The sample applications use the default options for each target. However, if you need to see how to use some of the target-specific options, then the sample applications can be generated which use these. Section 4.7 provides further information.

Each sample application is self contained within its own sub-folder - all the files necessary to build and run the sample application are provided. The following sections provide a short overview of the typical files.

4.2.1 Configuration

The RTA-OS configuration is typically contained in files named as follows:

Filename	Description
<ApplicationName>.rtaos	The RTA-OS project file which references the AUTOSAR OS XML files defining the configuration.
<ApplicationName>.xml	Target-neutral configuration. This file describes all the tasks, interrupts, alarms etc. used by the sample application.
<ApplicationName>_Target.xml	Target-specific configuration. This file describes target-specific parts of the configuration such as the priority and vector for any interrupts, target-specific configuration options, stack sizes etc.

The RTA-OS configuration can be viewed by opening <ApplicationName>.rtaos using **rtaoscfg**.

4.2.2 Application Code

The *.h and *.c files contain the source code for the application. The source code is split into target-specific and target-independent parts. The target-specific parts are described in Section 4.2.3.

4.2.3 Target Support

RTA-OS does not take control of any of microcontroller peripherals like timers, counters, I/O ports etc. If you want to use these devices to drive scheduling in RTA-OS then you will need to configure the appropriate peripherals on your target.

Integration Guidance 4.2:*Configuration of the target microcontroller peripherals is your responsibility as only you know how you want to use the microcontroller.*

However, the sample applications require some minimal hardware configuration to initialize the microcontroller and this is provided by target-specific source files. For example, many of the sample applications use a 1ms timer interrupt to drive scheduling.

The following files are typically used to describe how the target is configured and provide the source code to do this:

Filename	Description
README.txt	A description of the low-level details of the application, including the target reference platform, what peripherals are used, where the application is located, what debugger(s) are supported for downloading the application, etc.
Target.c	Target support code to initialize the hardware and program the interrupt source(s).
Target.h	Target support macros, including a macro to clear a pending interrupt.

Additional files may be supplied for particular ports.

If your target hardware or compiler differs from that used by the reference platform then you may need to modify the target-specific files.

Build and Run Support

The following files are provided to help you build and run (typically by downloading the example to your target hardware) the sample application.

Filename	Description
`build.bat`	A basic script to generate an RTA-OS kernel library, compile the example and link the example code with the library to produce a downloadable executable image.
`run.bat`	A sample script showing how to run the example application.

To build the application, open a Windows command prompt and run:

- `build.bat` if working with an embedded target; or
- `build.bat` *Variant* if working with VRTA where *Variant* is the name of the target variant (i.e. compiler) you want to use to build the example application.

If `build.bat` terminates successfully then an executable program called `<ApplicationName>.<target extension>` will have been created in the folder.

The application can be downloaded and/or executed using the `run.bat` script.

The following sections explain what each of the sample applications does and what to look for in order to verify that they are working correctly.

4.3 Hello World

The “HelloWorld” sample application is used to verify that the end-to-end build process is working correctly. This is provided so you can check that RTA-OS can build a kernel library with your toolchain and then use the library in a small example application that does just enough to show that the kernel is working correctly.

The source code generated for this sample application will demonstrate multicore operation if it is created for a target variant that has more than 1 core.

4.3.1 What does the “HelloWorld” example do?

On a single-core target “HelloWorld” shows preemption between two tasks, `HighPriority` and `LowPriority`. Both tasks run for 2ms. Task `HighPriority` is the higher priority task and runs periodically every 50ms. Task `LowPriority` runs periodically every 25ms.

On a multicore target, the same tasks and intervals exist but task `HighPriority` is configured to run on core 1 and all other OS objects are configured to run on core 0.

The periodic running of the tasks is achieved using two alarms, `Alarm50` and `Alarm25`, which are attached to a counter called `MillisecondCounter`. The counter is ticked using a 1ms timer interrupt that is handled by the Interrupt Service Routine (ISR)

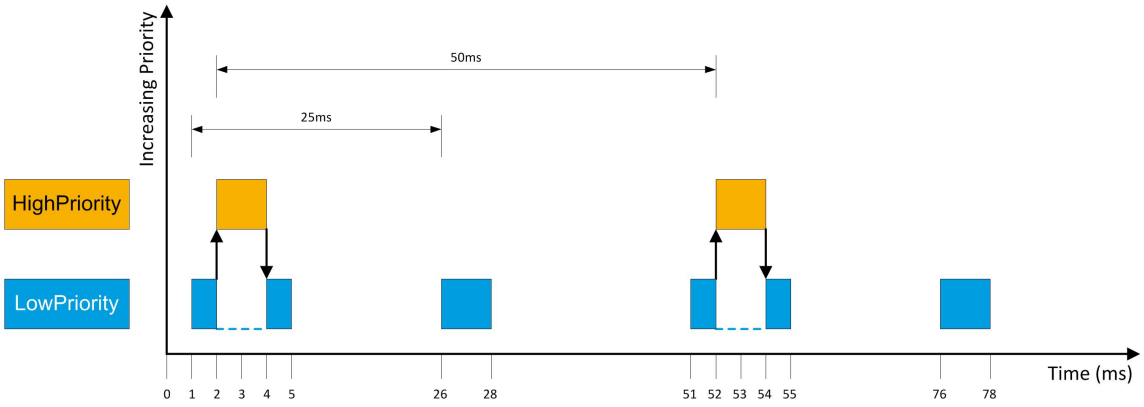

Figure 4.4: Execution of tasks in HelloWorld

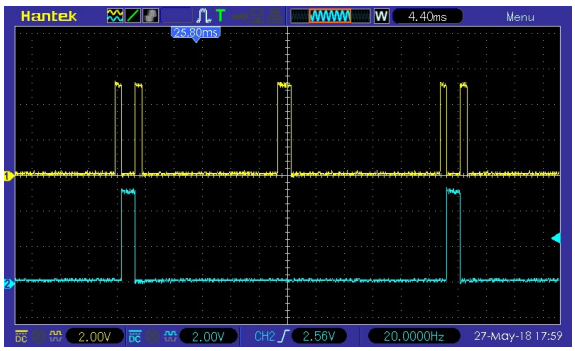

Figure 4.5: Oscilloscope Trace of HelloWorld

`MillisecondInterruptHandler`. Both alarms are auto-started. `Alarm50` is offset by 1ms relative to `Alarm25`.

When the application runs, Alarm25 will expire after 1ms and activate task `LowPriority`. Task `LowPriority` sets `IO_PIN1` high and runs for 1ms before being preempted by task `HighPriority`.

Task `HighPriority` saves the state of `IO_PIN1` and then sets it low before setting `IO_PIN2` high. It then runs for 2ms before setting `IO_PIN2` low. Finally, it restores the state of `IO_PIN1` and terminates, allowing task `LowPriority` to continue from the point at which it was preempted. Task `LowPriority` then runs for its remaining 1ms.

The pattern of execution is shown in Figure 4.4

4.3.2 Verifying Program Execution

You can monitor task activation by connecting oscilloscope probes to the IO pins defined by `IO_PIN1` and `IO_PIN2` in `Target.h`.

Figure 4.5 shows an oscilloscope trace of the state of the IO pins once the program is running. Each vertical grid line represents 10ms of time.

4.3.3 Troubleshooting

If your application doesn't appear to be running, you can use a debugger to verify that the ISR is being called. Place a breakpoint on the first instruction of the `ISR(Os_Entry_MillisecondISR)` and see if your application reaches it.

If the ISR runs this means that the counter is being ticked. You can then set breakpoints on the two tasks (`Os_Entry_HighPriority` and `Os_Entry_LowPriority`) to see whether or not they run. If the tasks run, but you do not see an output, check your hardware initialization.

If your oscilloscope does not show a trace, check the settings for `IO_PIN1` and `IO_PIN2` in `Target.h`. You should also check the `IO_PIN` control macros to ensure that they all reference the same I/O port on your target hardware. If the trace shows different timing behavior, check that your timer hardware is configured correctly and that instruction rate on the target hardware matches that specified in your configuration.

4.4 Clive Devices

"Clive Devices" shows you a more advanced RTA-OS configuration. Like "HelloWorld", when a multicore target is selected the sample application divides the work between two cores with the `Eating` and `OnThePhone` tasks being on core 1.

The application models the life of "Clive Devices". Clive's day is 24 hours long and in this time he does the following tasks in descending order of priority:

Task	Duration	When
`OnThePhone`	15 mins	Every hour between 7am and 7pm
`Eating`	1 hour	Breakfast at 7am, Lunch at 12pm, Evening meal at 8pm
`Working`	8 hours	Once a day, starting at 8am.
`Sleeping`	8 hours	Once a day, starting at 10pm.

Clive's free time is modeled using the idle mechanism - `Os_Cbk_Idle`.

Tasks `Eating`, `Sleeping` and `Working` are activated from a schedule table called `DailyRoutine`. The schedule table wraps every 24 ticks, representing every 24 hours.

Task `OnThePhone` is driven by the alarm `PhoneRings` which runs every hour, but only between 7:00 and 19:00 (this stops Clive from being woken up during the night).

The schedule table and the alarm are both driven by a software counter that is ticked using a 1ms timer interrupt that is handled by the ISR `MillisecondInterruptHandler`. This means that one hour of Clive's life is mapped onto 1 millisecond of real-time.

Clive cannot be `Eating` and `OnThePhone` at the same time because both activities require his mouth. A standard resource called `Mouth` is used to protect this critical section and is shared between tasks `Eating` and `OnThePhone`. However, the `Eating` tasks only uses `Mouth` for the middle 30 minutes - Clive spends 15 minutes preparing his meal before eating it and 15 minutes letting it digest afterwards.

Clive's life starts when RTA-OS is started. However, he does not live forever. There is a second alarm called `GrimReaper` which is set to expire 365 days after Clive is born. This alarm executes a callback called `TimeToDie` which sets a flag indicating that Clive will die the next time he has free time, i.e. the next time that `Os_Cbk_Idle` is resumed.

Both the alarms and the schedule table are auto-started 1ms from the time at which RTA-OS starts.

4.5 Pizza Pronto

"Pizza Pronto" shows how extended tasks can be configured and used. In particular, this sample application includes example stack usage values required to run extended tasks on your system (you can find out more about stack configuration for extended tasks in the *User Guide*).

The code generated for "Pizza Pronto" assigns the `EatPizza` task to core 1 on multicore targets.

The application models a software developer who is working on projects. Each project takes 100 minutes to develop and new projects arrive every 200 minutes.

The developer is always hungry, and every 20 minutes into every project their hunger needs to be satisfied by ordering a pizza for delivery. This task takes 5 minutes.

The time it takes the pizza to be delivered is random - it depends on how many other orders are already being processed by the pizza delivery company and how long it takes to make the delivery. In the best case, it will take 1 minute and in the worst case 61 minutes.

Once the developer has ordered the pizza, work can resume on the project. When the pizza delivery arrives, the developers doorbell is rung, the pizza is received and eaten. This takes 20 minutes.

After the pizza has been consumed, the project can be finished.

The following tasks, in decreasing priority order, are defined:

Task	Duration	Type	Period
`OrderPizza`	5 mins	Basic	Triggered by `WriteCode`
`EatPizza`	20 mins	Extended	
`WriteCode`	100 mins	Basic	200 mins.

As with 'CliveDevices" time is quantized onto milliseconds with 1 minute being represented as 1 millisecond of real time. As before, a 1ms timer interrupt that is handled by the Interrupt Service Routine (ISR) `MillisecondInterruptHandler` is used.

Task `WriteCode` is activated every 200ms using an auto-started alarm called `Project`. Each time `WriteCode` runs, it executes for 20ms then activates task `OrderPizza`. `OrderPizza` will always preempt as it has higher priority than `WriteCode`. `WriteCode` will resume after the pizza has been ordered and will run for a further 80ms.

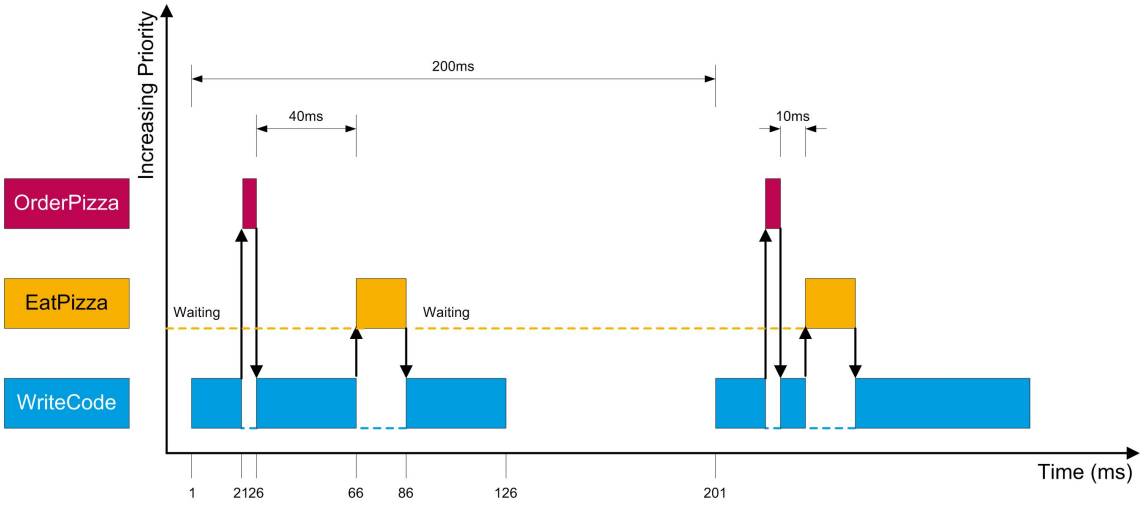

Figure 4.6: Example execution of tasks in PizzaPronto

Task `OrderPizza` runs for 5ms and then sets up a single-shot alarm called `Doorbell`. The alarm expiry is programmed as a random number between 1ms and 61ms. Each time the alarm expires, it sets an event called `Delivery` to indicate the delivery of the pizza.

Task `EatPizza` is an extended task that waits for the `Delivery` of the pizza, runs for 20ms and then waits for the next `Delivery`. `EatPizza` is implemented in the common way for extended tasks: an infinite loop containing a `WaitEvent()` call. The task is configured to auto-start when RTA-OS starts.

Figure 4.6 shows an example of how the tasks would execute if the first pizza delivery took 40ms and the second took 10ms.

As this sample application uses and extended task (`EatPizza`) it is necessary to specify the stack usage for each task and ISR as well as the base stack size. RTA-OS uses this information to manage the stack at run-time. You can see what values have been used for your port by looking at the **General → Target → Target Specific** configuration in **rtaoscfg**.

4.6 Integration with RTA-TRACE

The *Hello World*, *CliveDevices* and *PizzaPronto* sample applications are each supplied ready to work with RTA-TRACE if it is enabled in the configuration. RTA-TRACE is not enabled by default, but a sample configuration is provided in the `RTA-TRACE-Configuration.xml` file.

Embedded targets use a debugger or serial (RS232) data-link to transfer trace data from the target to the host PC. For embedded targets, **rtaosgen** can generate a sample serial link to use with your hardware.

The VRTA target uses a high speed TCP/IP data-link that transfers data directly between the Virtual ECU and RTA-TRACE.

If you want to integrate RTA-TRACE with one of these sample applications then you need to:

1. Open the `.rtaos` project file in **rtaoscfg**;
2. Change to the RTA-TRACE Configuration workspace;
3. Set **Enable Tracing** to TRUE;
4. Save the project;
5. Re-build the sample application by running `build.bat`;
6. Run the sample application using `run.bat`.

To look at the trace data, you should do the following:

1. Start RTA-TRACE and from the main menu select **File → New Connection**.
2. In the dialog asking for the location of the target accept the "localhost" default.
3. In the dialog asking you to select the OS interface choose either:
   - "RTAOSEK-Debugger" or "RTAOSEK-RS232" if you are working with an embedded target; or
   - "RTAOSEK-VRTA" if you are working with VRTA.
4. In the file selection dialog, choose the file `<ApplicationName>.rta` from the sample application's folder

If you are using VRTA, the application should start after a short pause, and you will see data appearing in the RTA-TRACE window. If an embedded target is being used, the application will need to be started manually.

4.7 Using Target Options

Targets may also provide target-specific options, for example support for different memory models, running from different memory areas, debugging features etc.

Sample applications can be generated which make use of these features by specifying the relevant target option on the command line.

A list of available options for your target can be obtained using the command:

```
rtaosgen --target:<target> --target_option:?
```

The required options can be added to the command for generating the sample applications as follows:

```
rtaosgen --samples:[Applications] --target:[<variant>]<target>
    --target_option:option1=value1  --target_option:option2=value2 ...
```

Integration Guidance 4.3:*Where target options contain spaces (i.e. `Stack used for C-startup`) the option must be enclosed within double quotes, as follows:*

```
rtaosgen --target:[MinGW]VRTA --target_option:"Stack used for
    C-startup"=64 HelloWorld.rtaos --include:Samples\Includes
```

5 Finding out more

Your RTA-OS distribution includes the following manuals:

`<install dir>\Documents`

***Getting Started Guide*.** The document that you are reading now. This guide explains how to install the product and describes the underlying principles of the operating system.

***Release Note*.** This document provides information about the release, including a list of changes from previous releases and a list of known issues.

***User Guide*.** This guide explains the concepts behind AUTOSAR OS and shows you how to use RTA-OS to configure the OS and integrate it into your application.

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

`<install dir>\Targets\VRTA_n.n.n`

***VRTA Port Guide*.** This guide explains implementation-specific details for the VRTA port plug-in.

***VRTA Release Note*.** This document provides information about the VRTA port plug-in release, including a list of changes from previous releases and a list of known issues.

***Virtual ECU User Guide*.** This guide explains how to use the Virtual ECU environment included with the VRTA port plug-in.

`<install dir>\Targets\<TargetCompiler>_n.n.n`

***Target/Compiler Port Guide*.** Each port of RTA-OS is supplied with a Port Guide. The Port Guide tells you specific information about the interaction between RTA-OS, your toolchain and your target hardware. For example, valid compiler options, register settings, interrupt handling etc. The Port Guide also gives performance and resource usage information for the OS.

***Target/Compiler Release Note*.** This document provides information about the port plug-in release, including a list of changes from previous releases and a list of known issues.

5.1 Related Reading

5.1.1 OSEK

OSEK/VDX Operating System (Version 2.2.3 17th February 2005) is the OSEK OS specification and describes the behavior of the OS in detail.

OSEK/VDX Binding Specification (Version 1.4.2 15th July 2004) describes how OSEK OS inter-operates with other OSEK standards.

OSEK Run-Time Interface (ORTI) (Part A: Language Specification Version 2.2 14th November 2005 and Part B: OSEK Objects and Attributes Version 2.2 25th November 2005) defines the core ORTI debugging language and the standard attributes supported by OSEK OS.

ISO 17356 OSEK is now standardized as ISO 17356.

5.1.2 AUTOSAR

RTA-OS is compatible with a range of AUTOSAR releases, from R3.0 up to R4.5.0 (R19-11). Each of the documents below is available at different versions.

AUTOSAR SWS OS is the AUTOSAR OS specification and describes AUTOSAR's extensions to OSEK OS in detail.

AUTOSAR SWS GPT Driver explains the AUTOSAR "General Purpose Timer" (Gpt) driver that the OS uses[1]

AUTOSAR SWS RTE describes the AUTOSAR RTE. This is required reading if you want to know how an AUTOSAR RTE interacts with an AUTOSAR OS.

AUTOSAR SRS General describes the general properties that apply to all AUTOSAR basic software modules. Note that the AUTOSAR OS document defines specific deviations from this general SRS that apply to the OS

AUTOSAR SWS StandardTypes defines the standard AUTOSAR types and the include structure for basic software modules like the OS.

AUTOSAR SWS PlatformTypes describes how target platform characteristics are abstracted in AUTOSAR.

AUTOSAR SWS MemoryMapping describes how AUTOSAR handles memory mapping of software. RTA-OS uses this concept to place parts of the OS in user nominated sections.

AUTOSAR SWS CompilerAbstraction describes how compiler properties such as type definitions, pointers, access to near and far memory are abstracted in AUTOSAR. RTA-OS uses this concept to ensure that all internal OS code conforms to the AUTOSAR standard.

[1] Note - while the AUTOSAR OS specification requires that the OS uses the Gpt, in practice the Gpt is not sufficiently powerful enough to be of any practical use. RTA-OS therefore does not make use of the Gpt driver directly. Please consult the *Release Note* for further details.

6 Contacting ETAS

6.1 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 below).

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@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
- The version of the ETAS tools you are using
- The version of the compiler tool chain you are using
- The command line (or reproduction of steps) that result in an error message
- The error messages or return codes you received (if any)
- Your `.xml`, `.arxml` and `.rtaos` files
- The file `Diagnostic.dmp` if it was generated

6.2 General Enquiries

6.2.1 ETAS Global Headquarters

ETAS GmbH
Borsigstrasse 24
70469 Stuttgart
Germany

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

6.2.2 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