RTA-OSEK

ORTI User Guide

Contact Details

ETAS Group

www.etasgroup.com

Germany

ETAS GmbH
Borsigstraße 14
70469 Stuttgart

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

www.etas.de

USA

ETAS Inc.
3021 Miller Road
Ann Arbor, MI 48103

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

www.etasinc.com

Japan

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

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

www.etas.co.jp

France

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

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

www.etas.fr

Korea

ETAS Korea Co., Ltd.
4F, 705 Bldg. 70-5
Yangjae-dong, Seocho-gu
Seoul 137-899, Korea

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

www.etas.co.kr

Great Britain

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

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

www.etas-uk.net

People's Republic of China

2404 Bank of China Tower
200 Yincheng Road Central
Shanghai 200120

Tel.: +86 21 5037 2220
Fax: +86 21 5037 2221

www.etas.cn

LiveDevices

LiveDevices Ltd.
Atlas House
Link Business Park
Osbaldwick Link Road
Osbaldwick
York, YO10 3JB

Tel.: +44 (0) 19 04 56 25 80
Fax: +44 (0) 19 04 56 25 81

www.livedevices.com

Copyright Notice

Version: RTA-OSEK v5.0.2

Disclaimer

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

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

Trademarks

RTA-OSEK and LiveDevices are trademarks of LiveDevices Ltd.

OSEK/VDX is a trademark of Siemens AG.

Windows and MS-DOS are trademarks of Microsoft Corp.

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

Contents

1 About this Guide

This guide describes the interface between an ORTI-aware debugger and RTA-OSEK Component.

1.1 Who Should Read this Guide?

It is assumed that you are a developer who has been writing programs in C that use RTA-OSEK Component. You should be using an ORTI-aware debugger to provide information on the program's behavior. Familiarity with the other RTA manuals is also assumed.

1.2 Conventions

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

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

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

2 Introduction

ORTI is an acronym that stands for 'OSEK Run Time Interface'. ORTI has been designed to facilitate an interface between the internal operation of an OSEK operating system and a debugger. It achieves this by specifying a small language that captures two things: how to find objects and variables within the running operating system and how to interpret or display their values.

The design of the ORTI language is sufficiently general that it can support operating systems other than OSEK.

ORTI support has been integrated into RTA-OSEK. This means that, during execution of the application, you can observe values of key operating system variables for applications based on RTA.

You will learn how to configure the generation of ORTI information for your debugger. You will also see the information that RTA-OSEK provides about its applications. For details of how to view ORTI information at runtime you should consult your debugger documentation.

2.1 Compatibility

ORTI has undergone a number of revisions over time and so debuggers have been released with differing levels of support. RTA currently supports a number of different debuggers. LiveDevices will make every effort to provide support for new and updated debuggers as they appear. Please contact technical support if your ORTI-aware debugger is not currently supported.

3 Overview of ORTI and RTA-OSEK

3.1 Development Process

The following steps describe how to use ORTI with your program.

1. Use the RTA-OSEK GUI to specify that debugger support is required for your application. To do this, select the **Target** group from the navigation bar and then select the **Debugger** subgroup. Figure 3:1 shows how this is done.

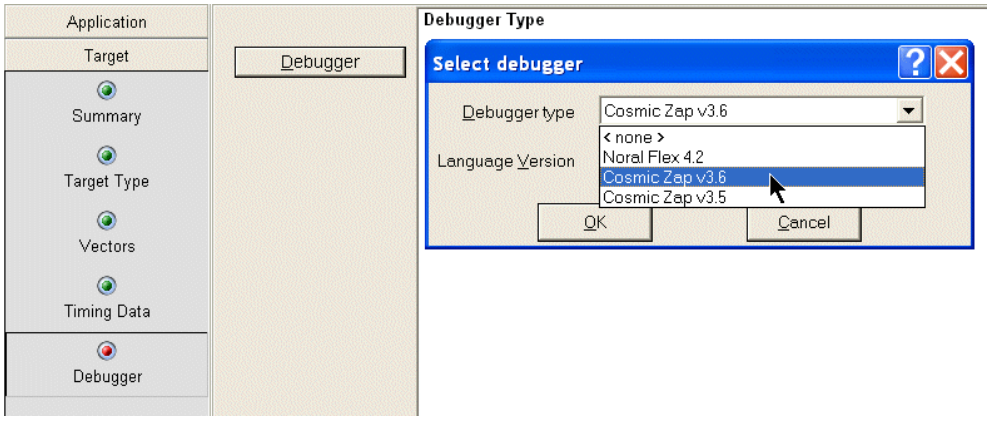

Figure 3:1 - Selecting the Debugger Type

2. Click the **Debugger** button on the workspace. The **Select Debugger** dialog opens. Set the **Debugger Type** and ORTI **Language Version** using the drop down lists and then click the **OK** button. An example is shown in Figure 3:2.

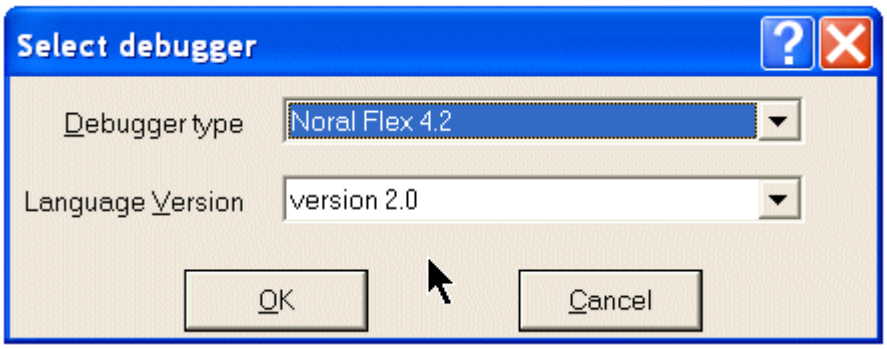

Figure 3:2 - Selecting the Debugger

3. Build the application. The ORTI code will be inserted into a file called `<projectname>.ort`. The details appear on the workspace, as shown in Figure 3:3.

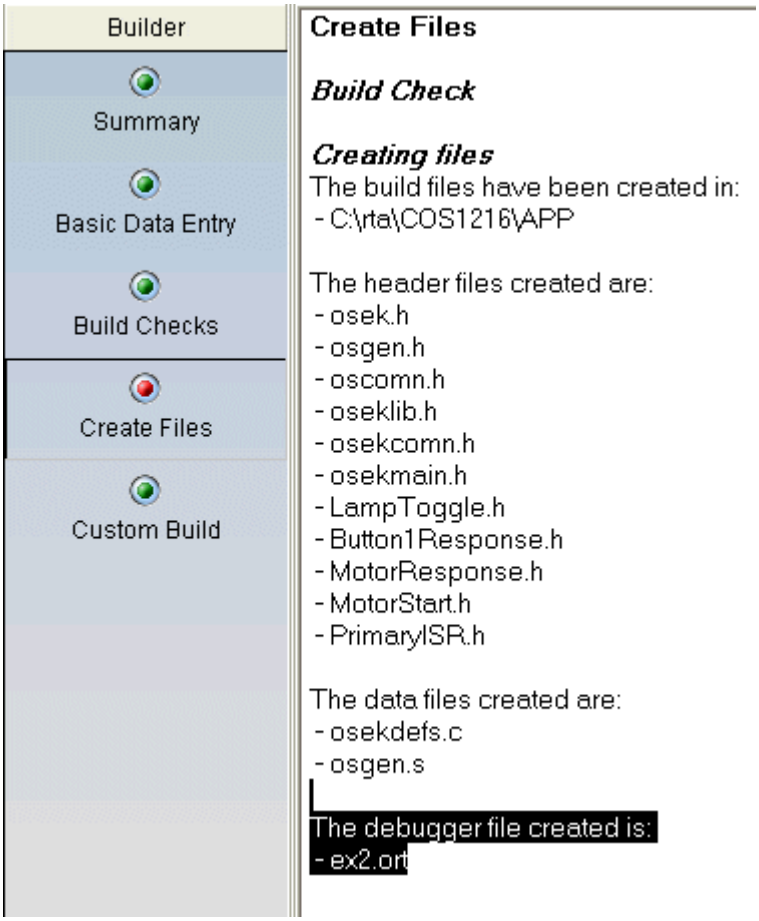

Figure 3:3 - Building an Application using the RTA-OSEK GUI

4. Start the debugger, load the application and load the ORTI file. This process is described in the debugger manual.

The debugger will then display the information shown by the ORTI file. The format of this information depends upon the debugger.

3.2 Intrusiveness

ORTI relies upon reading values from the memory of the running application. This means that the presence of ORTI can affect the operation of the application. It is useful to know the extent to which this might happen. ORTI can acquire data via four routes:

1. Constant values within the ORTI file. These are used for quantities that will not change during the execution of an application. These have no impact on the running application.

2. Values generated as part of the normal operation of the application. Data is read from variables that would be present even if ORTI were not. These have no additional impact on the application.
3. Values generated specifically for ORTI support and only present in the extended build. Such variables constitute a very small extra overhead in the application.
4. Constants generated only for ORTI support and present in all build modes. This data amounts to a small overhead in the application. These constants are only generated for debuggers that cannot obtain the information by other means. They are only present when you specify that you are using a debugger, so you may wish to disable debugger support in your final production release.

3.3 Validity

Many of the values reported by ORTI are simply those contained in the application's memory. Using ORTI to inspect the system before it has been fully initialized will lead to misleading results. RTA-OSEK is fully initialized when, as a result of calling `StartOS()`, the first task (including the idle task) or Category 2 ISR is entered.

Care should be taken where a variable may be cached in a register for a significant portion of its lifetime, especially in the case of register rich processors. ORTI can only look at the data stored in the variable's memory location. This could be out of date if the register-based copy has been updated recently.

3.4 Interactions

The ORTI output will be correct when the program is stopped at a breakpoint that is:

- In code executed by a task or Category 2 ISR that is outside of any OSEK API call.

The ORTI output may be misleading if the application is stopped at a breakpoint that is:

- Within an OSEK API call.
- In code executed by a Category 1 interrupt handler.

The output may be misleading because the OSEK data used by ORTI could be in a partially updated state. Normally it is possible to tell if the program is part way through an OSEK call by the debugger reporting the name of the function in which the processor stopped.

On a platform with more than two interrupt priority levels, however, a Category 1 interrupt can occur part way through an OSEK call. If the program is stopped at a breakpoint in a Category 1 interrupt handler, it is necessary to use the debugger's stack trace facility to determine the name of the function that was interrupted. The ORTI output can be relied upon, provided that the Category 1 interrupt did not occur within an OSEK API call.

4 ORTI Objects and Attributes

4.1 Overview

This chapter describes the ORTI objects and their attributes that are defined by RTA-OSEK. An ORTI object is some higher level encapsulation of information in RTA-OSEK such as `OS`, `TASK` or `ALARM`. An application may contain zero or more instances of each of these objects, each with a unique name. Each object has a number of attributes and each attribute has a value. For example, the `OS` object has a `RUNNINGTASK` attribute that shows the task that is currently running.

The following information is presented as a set of tables, each describing an object and all of its attributes. The tables appear in a standard format and each row contains standard information.

Object: Type	Description of Object	
Attributes	**Description**	**STATUS**
Attribute Name, "Attribute ORTI file description"	Description of attribute.	All S = Standard T = Timing E = Extended

Each object has a set of named attributes. Each row of the table names the attribute being described and gives a brief explanation of it.

The **Attributes** column contains the ORTI file description in quotes. Some debuggers display the attribute name and others display the description given in the ORTI file. The attribute prefix `vs_` is used to indicate attributes that have been added for RTA-OSEK support and are not standard ORTI attributes.

The **STATUS** column describes which STATUS attribute is available in (Standard, Timing, Extended or all three).

4.2 Objects and Attributes

4.2.1 OS

Object: OS	There is only one OS object. It is always present and is called "RTKOS".	
Attributes	**Description**	**STATUS**
RUNNINGTASK, "Running task"	The name of the TASK that is currently running. If an ISR interrupts a task this attribute will continue to display the name of the task that was interrupted while the ISR is executing.	All
RUNNINGTASKPRIORITY, "Running task priority"	The current priority of the running task, using the same terms as in the OIL file. RUNNINGTASKPRIORITY does not show the effect of locking a resource shared by tasks and ISRs.	All
RUNNINGISR2, "Running cat 2 ISR"	The value of this attribute is the name of the Category 2 ISR that is currently running (if there is one). NO_ISR is displayed if no Category 2 ISR is currently running.	T,E
vs_INTERRUPTPRIORITY, "Interrupt priority"	The current interrupt priority level (0 when running at 'user' level). This attribute is not supported on all targets.	Target - specific
SERVICETRACE, "OS Services Watch"	Indicates the entry or exit of a service routine (an RTA-OSEK Component API call) and the name of this routine. Some debuggers recognize this attribute as a special totrace attribute and can provide additional diagnostic support. On other debuggers, you will be shown which API call was most recently entered or completed.	E
LASTERROR, "Last OSEK error"	Gives the name of the last error that has occurred. Initially set to E_OK.	E

Object: OS	There is only one OS object. It is always present and is called "RTKOS".	
Attributes	**Description**	**STATUS**
CURRENTAPPMODE, "Current AppMode"	Current application mode using the names stated in the OIL file. The value "unknown AppMode" is reported if the application mode does not conform to a value in the OIL file.	All
vs_SPEED, "Processor speed"	Gives the speed of the processor as defined in the OIL file.	All
vs_STOPWATCHSPEED, "Stopwatch speed"	Gives the stopwatch speed as defined in the OIL file.	All

4.2.2 Task

Object: TASK	Generated in response to task declarations in the configuration file.	
Attributes	**Description**	**STATUS**
STATE, "State"	The task state. One of SUSPENDED, RUNNING, READY and WAITING.	All
vs_BASEPRIORITY, "Base priority"	Gives the base priority of the task. The base priority is the priority of the task as defined in the OIL file.	All
vs_DISPATCHPRIORITY, "Dispatch priority"	Gives the dispatch priority of the task. The dispatch priority is the priority that the task starts running at. This can be higher than the base priority if internal resources are used or if the task is non-preemptable.	All
vs_ACTIVATIONS, "Maximum activations"	Gives the maximum number of activations allowed.	All
vs_TYPE, "Conformance type"	Shows the conformance class of the task. It can take the values: BCC1, BCC2, ECC1 or ECC2.	All
vs_RESOURCES, "Resources"	Lists the resources that the task uses.	All

Object: TASK	Generated in response to task declarations in the configuration file.	
Attributes	**Description**	**STATUS**
vs_EVENTS, "Events"	Lists the events that the task can wait for.	All
vs_TERMINATION, "Termination"	This is the termination type of the task. HEAVY or LIGHT.	All
vs_SCHEDULE, "Preemptability"	Indicates whether the task is preemptable or non-preemptable.	All
vs_USESFP, "Floating point"	Indicates whether this task uses floating point. It is TRUE if the task uses floating point arithmetic and FALSE otherwise.	All

4.2.3 Category 2 ISR

Object: ISR2	Generated in response to Category 2 ISR declarations in the configuration file.	
Attributes	**Description**	**STATUS**
vs_PRIORITY,"Priority"	Gives the priority of the ISR.	All
vs_VECTOR,"Vector"	Gives the vector to which the ISR is bound.	All
vs_USESFP, "Floating point"	Indicates whether this ISR uses floating point. It is TRUE if the task uses floating point arithmetic.	All
vs_RESOURCES, "Resources"	Lists the resources that the ISR uses.	All
vs_BUFFERING, "Buffering"	Shows the ISR buffering type. This matches the buffering shown in the RTA-OSEK GUI.	All

4.2.4 Category 1 ISR

Object: ISR1	Only generated in response to Category 1 ISR declarations in the configuration file.	
Attributes	**Description**	**STATUS**
`vs_PRIORITY,"Priority"`	Gives the priority of the ISR.	All
`vs_VECTOR, "Vector"`	Gives the vector to which the ISR is bound.	All
`vs_BUFFERING, "Buffering"`	Shows the ISR buffering type. This matches the buffering shown in the RTA-OSEK GUI.	All

4.2.5 Counter

Object: COUNTER	Only generated in response to counter declarations in the configuration file.	
Attributes	**Description**	**STATUS**
`vs_COUNT, "Count"`	Indicates the current count value.	All
`vs_TICKRATE, "Tick rate"`	Indicates the expected tick rate.	All
`vs_DRIVER, "Driver"`	Indicates the ISR or task that is expected to drive the counter.	All
`vs_MAXALLOWEDVALUE, "MaxAllowedValue"`	Indicates the counter maximum value.	All
`vs_MINCYCLE, "MinCycle"`	Indicates the counter minimum cycle value.	All
`vs_TICKSPERBASE, "TicksPerBase"`	Indicates the counter `tickserbase` value.	All

4.2.6 Alarm

Object: ALARM	Only generated in response to alarm declarations in the configuration file.	
Attributes	**Description**	**STATUS**
`ALARMTIME, "Alarm Time"`	Shows when the alarm expires next. Refer to the `Count` value in the `COUNTER` object to establish the current count value.	All
`CYCLETIME, "Cycle Time"`	Gives the period of the cycle for a cyclic alarm. `CYCLETIME` will be zero for a single-shot alarm.	All
`ACTION, "Action"`	The action to perform when the alarm expires. This can include the following: • Activate a task. • Set an event. • Execute a callback function. In RTA-OSEK Component all of the actions can be performed and ORTI presents them as a list.	All
`STATE, "Alarm state"`	Indicates whether the alarm is running. Takes the value `RUNNING` or `STOPPED`.	All
`"COUNTER", "Counter"`	Gives the name of the counter to which this alarm is attached.	All

4.2.7 MessageContainer

Object: MESSAGECONTAINER	Only generated in response to message receive accessor declarations in the configuration file.	
Attributes	**Description**	**STATUS**
`MSGNAME,"Message Name"`	The name of the message	All
`vs_CDATATYPE,"C type"`	Indicates the C data type for the message.	All
`MSGTYPE,"Message Type"`	Gives the message type. It can take the value `QUEUED` or `UNQUEUED`.	All

Object: MESSAGECONTAINER	Only generated in response to message receive accessor declarations in the configuration file.	
Attributes	**Description**	**STATUS**
"QUEUESIZE", "Queue size"	Gives the size of the queue for queued messages. It is zero for unqueued messages.	All
ACTION, "Action performed when message is received"	Gives the action performed when message is received. This can include up to 1 each of the following: • Activate a task. • Set an event. • Call a function. • Set a flag.	All
SENDER, "Sender"	Gives the name of the task that sends the message.	All
vs_SENDCOPY, "Send copy"	Reflects whether the send accessor uses a copy of the message data. Set to TRUE if WithCopy and FALSE if WithoutCopy.	All
RECEIVER, "Receiver"	Gives the name of the task(s) that receive the message.	All
vs_RECEIVECOPY, "Receive copy"	Reflects whether the receive accessor uses a copy of the message data. Set to TRUE if WithCopy and FALSE if WithoutCopy.	All

4.2.8 COM

Object: COM	There is only one COM object. It is always present and has the name "RTACOM".	
Attributes	**Description**	**STATUS**
vs_RUNNING, "Running"	Indicates whether COM is running or not. It is TRUE if COM is running, otherwise it is FALSE.	E

Support

For product support, please contact your local ETAS representative.

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