OSEK/VDX

OSEK Run Time Interface (ORTI)

Part B: OSEK Objects and Attributes

Version 2.1

17. April 2002

Preface

OSEK/VDX is a joint project of the automotive industry. It aims at an industry standard for an open-ended architecture for distributed control units in vehicles.

For detailed information about OSEK project goals and partners, please refer to the "OSEK Binding Specification".

General conventions, explanations of terms and abbreviations have been compiled in the additional inter-project "OSEK Overall Glossary".

Regarding implementation and system generation aspects please refer to the "OSEK Implementation Language" (OIL) specification.
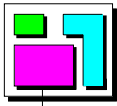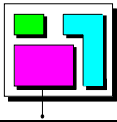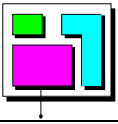
Table of Contents

1 Overview

1.1 Scope

The OSEK Run Time Interface (ORTI) is intended as a universal interface for development tools to the OSEK Operating System. The interface intends to enable the attached tool to evaluate and display information about the operating system, its state, its performance, the different task states, the different operating system objects etc.

1.1.1 Generation of Information

Information about the OSEK configuration and status availability can be made available at generation of the configuration header files by the system generator. System generators are provided with each OSEK implementation and connect configuration information to C header files. They are capable of providing all necessary information including configuration information and internal code and data addresses to the attached tool. The ORTI file is typically generated at build time by the OSEK manufacturer's System Generator, as shown in figure 1.

The configuration information will remain static during the debug session (or until a new software version is compiled) and serves as an information basis. Additionally the ORTI file contains dynamic information as a set of attributes that are represented by formulas to access corresponding dynamic values. Formulas for dynamic data access are comprised of constants, operations, and symbolic names within the target file. The debug tool to obtain internal values of the required OS objects can then evaluate the given formula.
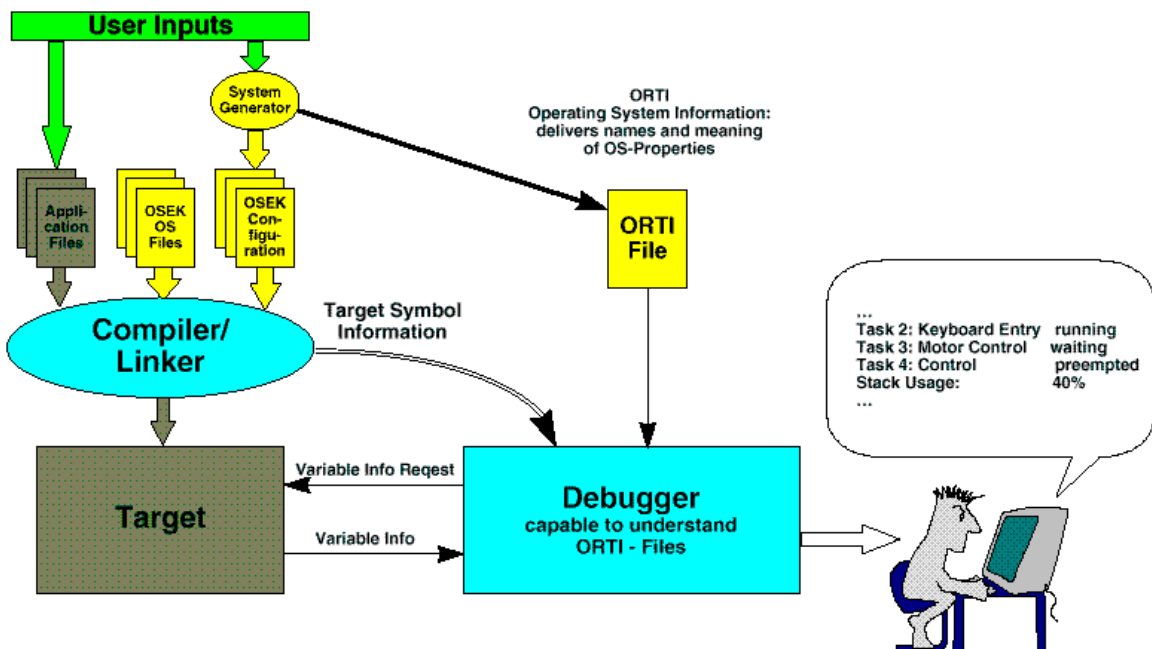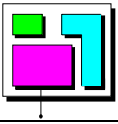

Figure 1 - ORTI Aware Debugging System

2 Standard ORTI Objects and Attributes

ORTI is intended for the description of applications in any OSEK implementation. It describes a set of attributes for system objects and a method for interpreting the data obtained. The types defined in the section are specified to allow the debugger to determine the target memory access method as well as the best way of displaying the retrieved data. In most cases the information that the user will require to see is a textual description of an attribute rather than the actual value read from the variable.

An example of this is as follows; when a user requests the current state of a task he will expect to see something like RUNNING, WAITING, READY or SUSPENDED, instead of the actual numeric value that is used by the OS to represent this information internally. For this reason KOIL (see part A) has been introduced, which allows a kernel manufacturer to describe how an internal OS value must be mapped to a descriptive value. This approach is independent of OSEK as it allows any kernel manufacturer to specify a set of kernel objects to be displayed by any debugger.

However, KOIL treats all object types the same. So, a debugger cannot assume any specific properties of the objects displayed. For example, task objects are displayed similar to stack objects and no specific debugging features belonging to these objects can be assumed. This is "Part A" compliant debugging, which gives basic kernel awareness.

This document defines a set of recommended objects and attributes, and their semantics. Any objects and attributes that use the names in this document must also have the semantics defined.

2.1 Kernel version

The name of the semantics described in this document is "ORTI". This document describes the semantics version "2.1".

All ORTI files using this specification with the version of this document must have the following kernel version:

```
OSSEMANTICS = "ORTI", "2.1";
```

2.2 General attributes

2.2.1 Attribute VALID

Type : CTYPE

Every object declaration may contain a VALID attribute telling the debugger whether the object's attributes are currently valid. It may have an integer type of any size. Its possible values are 0 (invalid) and non zero (object is valid).

2.3 Object OS

The OS object represents an OSEK Operating System that runs on the target CPU. There can only be one OS object in an ORTI file. For example:

```
OS sampleOS {
    CURRENTAPPMODE = "osAppMode";
    RUNNINGTASK = "osRunningTask";
    SERVICETRACE = "osServiceTrace";
    RUNNINGISR2 = "osRunningISR";
};
```

The following attributes belong to the object OS:

2.3.1 Attribute RUNNINGTASK

Type: ENUM

This attribute specifies how to evaluate the task currently in state RUNNING within the OS object.

RUNNINGTASK is set to NO_TASK while no task is in running state.

Example (part of OS object):

```
ENUM [
    "NO_TASK" = 0,
    "TaskA" : TaskA = 1,
    "TaskB" : TaskB = 2
] RUNNINGTASK;
```

2.3.2 Attribute RUNNINGTASKPRIORITY

Type: ENUM or CTYPE

This attribute specifies how to evaluate the current priority of the task referred by RUNNINGTASK. The current priority can be different from the static task priority as a result of priority ceiling protocol. The priority displayed is the priority as defined in the OIL file.

2.3.3 Attribute RUNNINGISR2

Type: ENUM

This attribute specifies how to evaluate the category 2 ISR currently running within the OS object.

RUNNINGISR2 is set to NO_ISR while no category 2 ISR is running.

Example (part of OS object):

```
ENUM [
    "NO_ISR" = 0,
    "ISRA" = 1,
    "ISRB" = 2
] RUNNINGISR2;
```

2.3.4 Attribute SERVICETRACE

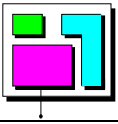Type: ENUM

This attribute indicates the entry or exit of a service routine and the ID of this service routine. The value of this attribute must be evaluated from one single memory location. This attribute must be traceable (TOTRACE).

The least significant bit of this attribute indicates the entry (= 1) or exit (= 0) of a service routine. All service IDs must be even ENUM values.

See Appendix A for a detailed description of SERVICETRACE.

2.3.5 Attribute LASTERROR

Type: ENUM

Attribute for the last error code detected not equal E_OK. At startup the error code is initialized with E_OK. It is never set back to E_OK after first error.

Example:

```
ENUM [
    "E_OK"          = 0,
    "E_OS_ACCESS"   = 1,
    "E_OS_CALLEVEL" = 2,
    "E_OS_ID"       = 3,
    "E_OS_LIMIT"    = 4,
    "E_OS_NOFUNC"   = 5,
    "E_OS_RESOURCE" = 6,
    "E_OS_STATE"    = 7,
    "E_OS_VALUE"    = 8
] LASTERROR;
```

2.3.6 Attribute CURRENTAPPMODE

Type: ENUM

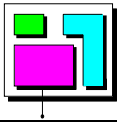Attribute that describes the current application mode.

Example:

```
ENUM [
    "OSDEFAULTAPPMODE" = 0,
    "OSDIAGNOSTIC"     = 1
] CURRENTAPPMODE;
```

2.4 Object TASK

A TASK object represents an OSEK task. For example:

```
TASK sampleTask {
    PRIORITY = "taskPriority[1]";
    STATE = "taskA->state";
    STACK = "STACK_1";
};
```

The following attributes belong to the object TASK:

2.4.1 Attribute PRIORITY

Type: ENUM or CTYPE

This attribute represents the current priority of the TASK object. The current priority can be different from the static task priority as a result of priority ceiling protocol. The priority displayed is the priority as defined in the OIL file.

2.4.2 Attribute STATE

Type: ENUM

This attribute contains the current state of the TASK object. The possible states of a TASK are as follows:

- SUSPENDED
- READY
- RUNNING
- WAITING

2.4.3 Attribute STACK

Type: ENUM

This attribute contains a reference to the stack object that the task is currently using.

2.4.4 Attribute CURRENTACTIVATIONS

Type: CTYPE

This attribute specifies the number of current activations for the task.

2.4.5 Attribute CONTEXT

Type: ENUM

This attribute contains a reference to the context object that the task is currently using.

2.5 Object CONTEXT

The CONTEXT object declaration describes a subset of the information saved by the operating system for a particular task.

A CONTEXT is uniquely attached to a task.

Each processor supported by ORTI will have an appendix to ORTI that names the processor resources that can be part of a context.

The _CPU_ prefix is reserved for attributes that describe processor specific information as specified in the appendixes and must not be used to describe any other information.

Example: _CPU_X will be used for the X register.
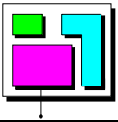
2.5.1 Attribute ADDRESS

Type: CTYPE

This attribute represents the base address of a memory area containing a subset of the context. The format and interpretation of that area is not part of this document.

The intent of this attribute is to provide access to context information, which may not be provided by other attributes.

2.5.2 Attribute SIZE

Type: CTYPE

This attribute represents the size (in bytes) of the memory area described in 2.5.1.

Example of CONTEXT object:

```
IMPLEMENTATION myOSEK {
   ...
   TASK {
      ...
      ENUM [
       "ContextA":ContextA = 0x2030, // Reference to "ContextA" CONTEXT Object
       "ContextB":ContextB = 0x2100,
       "ContextC":ContextC = 0x3000
      ] CONTEXT;
      ...
   }
   CONTEXT {
      CTYPE VALID;         /* if TRUE (!= 0), context is valid */
      CTYPE ADDRESS;
      CTYPE SIZE;
      CTYPE _CPU_PC;       /* value of program counter 'PC' */
      CTYPE _CPU_SP;       /* value of register 'SP' */
      CTYPE _CPU_R0;       /* value of register 'R0' */
      CTYPE _CPU_R1;       /* value of register 'R1' */
      ...
   }
}
...
```

Definitions of the CONTEXT objects, which may be referenced by TASK objects

```
TASK TaskA {
   ...
   CONTEXT = "&tcb[0]";
   ...
};

CONTEXT ContextA {
   VALID = "taskA->state != RUNNING";
   ADDRESS = "&tcb[0]";
   SIZE = "0x20";
   _CPU_PC = "taskA->stack->PC";
   _CPU_SP = "taskA->stack";
   _CPU_R0 = "taskA->stack->R0";
   _CPU_R1 = "taskA->stack->R1";
   ...
);
```

```
CONTEXT ContextB {
   VALID = "taskB->state != RUNNING";
   ADDRESS = "&tcb[1]";
   SIZE = "0x20";
   _CPU_PC = "taskB->stack->PC";
   _CPU_SP = "taskB->stack";
   _CPU_R0 = "taskB->stack->R0";
   _CPU_R1 = "taskB->stack->R1";
   ...
};
```

2.6 Object STACK

The STACK object defines the memory area of any stack in the system. For example, if three tasks are present each with an individual stack space, then three STACK objects will be present in the ORTI file. For example:

```
STACK sampleStack {
   SIZE = "0x200";
   BASEADDRESS = "stack[1]";
   STACKDIRECTION = "UP";
   FILLPATTERN = "0xAA55";
};
```

The STACK object can have the following attributes:

2.6.1 Attribute SIZE

Type: CTYPE

This attribute represents the size (in bytes) of a memory area allocated for stack.

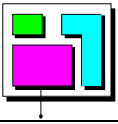### 2.6.2 Attribute BASEADDRESS

Type: CTYPE

This attribute specifies the lowest address of stack memory area, regardless of the stack direction.

2.6.3 Attribute STACKDIRECTION

Type: STRING

This attribute specifies the direction of stack growth and may have either "UP" or "DOWN" as its value. UP means growing from lower to higher addresses. DOWN means growing from higher addresses to lower addresses.

Example:

Address range	Direction	Attribute name	Attribute value
0x2ff ... 0x100	⇧	SIZE BASEADDRESS STACKDIRECTION	0x200 0x100 "UP"
0x2ff ... 0x100	⇩	SIZE BASEADDRESS STACKDIRECTION	0x200 0x100 "DOWN"

2.6.4 Attribute FILLPATTERN

Type: CTYPE

If the operating system fills the stack during initialisation, this attribute specifies with which pattern the stack area is initialised. This allows the debugger to evaluate the maximum stack usage.

For STACKDIRECTION DOWN the pattern starts at BASEADDRESS. For STACKDIRECTION UP the pattern starts at BASEADDRESS+SIZE. If no pattern is used, this attribute must be omitted.

2.7 Object ALARM

An ALARM object represents an OSEK alarm.

The following attributes belong to the object Alarm:

2.7.1 Attribute ALARMTIME

Type: CTYPE

This attribute specifies how to evaluate the time until the alarm expires next.

2.7.2 Attribute CYCLETIME

Type: CTYPE

This attribute specifies how to evaluate the cycle time for cyclic alarms. The value of CYCLETIME is 0 for non-cyclic alarms.

2.7.3 Attribute STATE

Type: ENUM

This attribute specifies if an Alarm is RUNNING or STOPPED.
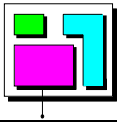
Example:

```
ENUM [
    "STOPPED" = 0,
    "RUNNING" = 1
] STATE, "Alarm State";
```

2.7.4 Attribute ACTION

Type: STRING

This attribute provides a string with a description of the action when the alarm expires, e.g. "ActivateTask TaskA".

2.7.5 Attribute COUNTER

Type: STRING

This attribute provides a string containing the name of the counter used by this alarm.

2.8 Object RESOURCE

A RESOURCE object represents an OSEK resource.

The RESOURCE object can have the following attributes:

2.8.1 Attribute STATE

Type: ENUM

This attribute represents the state of a resource (LOCKED/UNLOCKED).

Example:

```
ENUM [
    "UNLOCKED" = 0,
    "LOCKED" = 1
] STATE, "Resource State";
```

2.8.2 Attribute LOCKER

Type: ENUM

This attribute indicates the name of the locking TASK/ISR.

2.8.3 Attribute PRIORITY

Type: STRING

This attribute has two components that state: that the RESOURCE is used by TASKs only or by TASKs and ISRs, and the priority that will be used when locking the RESOURCE.

Example 1:

In the case where a RESOURCE is used by two TASKS at priorities 3 and 5 the result might be:
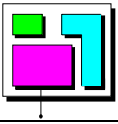
```
RESOURCE res1 {
    PRIORITY = "TASK: 5";
}
```

Example 2:

In the case where a RESOURCE is used by at least one ISR at priority 2 and one TASK at priority 5 the result might be:

```
RESOURCE res2 {
    PRIORITY = "ISR: 2";
}
```

Example 3:

In the case where a RESOURCE is used by two ISRs at priorities 2 and 4 the result might be:

```
RESOURCE res3 {
    PRIORITY = "ISR: 4";
}
```

2.9 Object MESSAGECONTAINER

Each existing combination of Messages and ACCESSORRECEIVED is described with an own MESSAGECONTAINER object.

The following attributes belong to the object MESSAGECONTAINER:

2.9.1 Attribute MSGNAME

Type: STRING

Name of the message as defined in OIL file.

2.9.2 Attribute MSGTYPE

Type: STRING

Type of the message: "QUEUED" or "UNQUEUED".

2.9.3 Attribute QUEUESIZE

Type: CTYPE

Size of the queue for queued messages. One for unqueued messages.

2.9.4 Attribute QUEUECOUNT

Type: CTYPE

Number of valid messages in the queue. One for unqueued messages.

2.9.5 Attribute FIRSTELEMENT

Type: CTYPE

Formula for evaluation of address of first valid message. This message will be received next. If no message is in the queue the value is zero.

Example:

```
MESSAGECONTAINER {
    STRING MSGNAME;          /* message name (name in OIL file) */
    STRING MSGTYPE;          /* "QUEUED" or "UNQUEUED" */
    CTYPE "unsigned char" QUEUESIZE;  /* queue size (undefined for unqueued msg.) */
    CTYPE "unsigned char" QUEUECOUNT; /* number of valid entries */
    CTYPE "cdatatype*" FIRSTELEMENT;   /* formula for evaluation of first elem. */
} , "MESSAGE";

MESSAGECONTAINER MessageA_RxTaskC
{
    MSGNAME = "MessageA";
    MSGTYPE = "QUEUED";
    QUEUESIZE = "5";
    QUEUECOUNT = "msg_a.count";
    FIRSTELEMENT = "&(msg_a.data[msg_a.rd_idx])";
}
```

3 Vendor specific objects and attributes

ORTI allows the definition of vendor specific objects and attributes. To avoid name conflicts with new versions of the ORTI specification, vendors must use the prefix vs_ for vendor specific names.

4 Sample ORTI File

```
/*********************/
/* Declaration Section */
/*********************/

VERSION {
   KOIL = "2.1";
   OSSEMANTICS = "ORTI", "2.1";
};


IMPLEMENTATION VendorX_ORTI {

   OS {
      ENUM [
         "basicTaskFirst" = "&(gTaskControlBlocks[0])",
         "extendedTaskFirst" = "&(gTaskControlBlocks[1])",
         "extendedTaskSecond" = "&(gTaskControlBlocks[2])"
      ] RUNNINGTASK, "Running Task Identification";

      ENUM "unsigned char" [
         "51" = 0,
         "50" = 1,
         "30" = 2,
         "10" = 3
      ] RUNNINGTASKPRIORITY, "Priority of Running Task";

      ENUM "unsigned char" [
         "NO_ISR"    = 0,
         "Timer2Int" = 1,
         "CanRxInt"  = 2,
         "CanTxInt"  = 3
      ] RUNNINGISR2, "Running ISR of category 2";

      TOTRACE ENUM "unsigned char" [
         "ActivateTask"   =  2,
         "TerminateTask"  =  4,
         "ChainTask"      =  6,
         "Schedule"       =  8,
         /* ... */
         "StartOS"        = 52,
         "ShutdownOS"     = 54
      ] SERVICETRACE, "OS Services Watch";

      ENUM "unsigned char" [
         "E_OK"         = 0,
         "E_OS_ACCESS"   = 1,
         "E_OS_CALLEVEL" = 2,
         "E_OS_ID"       = 3,
         "E_OS_LIMIT"    = 4,
         "E_OS_NOFUNC"   = 5,
         "E_OS_RESOURCE" = 6,
         "E_OS_STATE"    = 7,
         "E_OS_VALUE"    = 8
      ] LASTERROR, "Last OSEK error";

      ENUM "unsigned char" [
         "DEFAULT_APPMODE"    = 0,
         "DIAGNIOSTIC_MODE"   = 1
      ] CURRENTAPPMODE, "Current application mode";
   }, "OS";

   TASK {
      ENUM "unsigned char" [
         "51" = 0,
         "50" = 1,
         "30" = 2,
         "10" = 3
      ] PRIORITY , "Actual Prio";
```
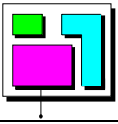
```
    ENUM "unsigned char" [
        "READY"=0,
        "RUNNING"=1,
        "WAITING"=2,
        "READY"=3,
        "SUSPENDED"=4
    ] STATE, "Current State";

    ENUM "unsigned short *" [
        "Stack0" : taskStack0 = "&(taskStack0[0])",
        "Stack1" : taskStack1 = "&(taskStack1[0])",
        "Stack2" : taskStack2 = "&(taskStack2[0])"
    ] STACK, "Task Stack";

    CTYPE "unsigned char" REMAININGACTIVATIONS, "Remaining task activations";

    ENUM "unsigned short *" [
        "Context 0" : Context_0 = 0,
        "Context 1" : Context_1 = 1,
        "Context 2" : Context_2 = 2
    ] CONTEXT, "Task Context";

    STRING vs_Home_Priority, "Home Priority";
    STRING vs_Task_Type, "Task Type";  /* BASIC / EXTENDED */
    STRING vs_Schedule, "Schedule";     /* non preemptive, full preemptive */
    CTYPE "unsigned short" vs_Event_Mask, "Wait Mask";
    CTYPE "unsigned short" vs_Event_Flags, "Event Flags";
    STRING vs_max_Activations, "max. Activations";
}, "Tasks";


STACK {
    CTYPE SIZE, "Stack Size (Byte)";
    CTYPE "unsigned short *" BASEADDRESS, "Base Address";
    STRING STACKDIRECTION, "Stack Direction";
    CTYPE "unsigned short" FILLPATTERN, "Stack Fill Pattern";
}, "Stacks";

ALARM {
    CTYPE "unsigned long"  ALARMTIME, "Alarm Time";
    CTYPE "unsigned long"  CYCLETIME, "Cycle Time";
    ENUM [
        "STOPPED" = 0,
        "RUNNING" = 1
    ] STATE, "Alarm state";
    STRING ACTION, "Action";
    STRING COUNTER, "Counter";
}, "Alarms";

CONTEXT {
    CTYPE "unsigned long" _CPU_R0, "CPU register R0";
    CTYPE "unsigned long" _CPU_R1, "CPU register R1";
    CTYPE "unsigned long" _CPU_R2, "CPU register R2";
    CTYPE "unsigned long" _CPU_R3, "CPU register R3";
    CTYPE "unsigned long" _CPU_PC, "CPU register PC";
    CTYPE "unsigned long" _CPU_SR, "CPU register SR";
    CTYPE "unsigned char" VALID;
}, "Task Context";

RESOURCE {
    ENUM "unsigned char" [
        "UNLOCKED" = 0,
        "LOCKED" = 1
    ]     STATE, "Resource State";
    ENUM "unsigned char" [
        "Task basicTaskFirst" = 0,
        "Task extendedTaskSecond" = 1,
        "ISR Timer2Int" = 2
    ] LOCKER, "Resource Locker";
    STRING PRIORITY, "Ceiling priority";
}, "Resources";
```
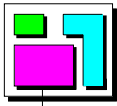
```
    MESSAGECONTAINER {
        STRING MSGNAME, "Message Name";
        STRING MSGTYPE, "Message Type";
        CTYPE QUEUESIZE, "Queue Size";
        CTYPE QUEUECOUNT, "Entry Count";
        CTYPE FIRSTELEMENT, "First Message";
    }, "Message Container";

    vs_ISR {
        STRING vs_Priority, "Priority";
    }, "ISRs";
}; /* END OF IMPLEMENTATION */


/**********************/
/* Information Section */
/**********************/

OS VendorX_arch {
    RUNNINGTASK = "gActiveTaskTcb";
    RUNNINGTASKPRIORITY = "osRunTaskPrio";
    RUNNINGISR2 = osORTIIsr2Id";
    SERVICETRACE = "osORTIServiceId";
    LASTERROR = "Last_OSEK_err";
    CURRENTAPPMODE = "osAppMode";
};

TASK basicTaskFirst {
    PRIORITY = "(gTaskControlBlocks[0].actualPrio)";
    STATE = "(gTaskControlBlocks[0].taskState)";
    STACK = "&(taskStack0[0])";
    REMAININGACTIVATIONS = "(gTaskControlBlocks[0].actCount)";
    CONTEXT = "0";
    vs_Home_Priority = "10";
    vs_Task_Type = "BASIC";
    vs_Schedule = "FULL Preemptive";
    vs_Event_Mask = "0";
    vs_Event_Flags = "0";
    vs_max_Activations = "1";
};

TASK extendedTaskFirst {
    PRIORITY = "(gTaskControlBlocks[1].actualPrio)";
    STATE = "(gTaskControlBlocks[1].taskState)";
    STACK = "&(taskStack1[0])";
    REMAININGACTIVATIONS = "(gTaskControlBlocks[1].actCount)";
    CONTEXT = "1";
    vs_Home_Priority = "30";
    vs_Task_Type = "EXTENDED";
    vs_Schedule = "FULL Preemptive";
    vs_Event_Mask = "(gTaskControlBlocks[1].eventMask)";
    vs_Event_Flags = "(gTaskControlBlocks[1].eventFlag)";
    vs_max_Activations = "1";
};

TASK extendedTaskSecond {
    PRIORITY = "(gTaskControlBlocks[2].actualPrio)";
    STATE = "(gTaskControlBlocks[2].taskState)";
    STACK = "&(taskStack2[0])";
    REMAININGACTIVATIONS = "(gTaskControlBlocks[2].actCount)";
    CONTEXT = "2";
    vs_Home_Priority = "50";
    vs_Task_Type = "EXTENDED";
    vs_Schedule = "FULL Preemptive";
    vs_Event_Mask = "(gTaskControlBlocks[2].eventMask)";
    vs_Event_Flags = "(gTaskControlBlocks[2].eventFlag)";
    vs_max_Activations = "1";
};
```
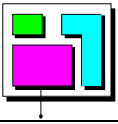
```
STACK taskStack0 {
    SIZE = "128";
    STACKDIRECTION = "UP";
    BASEADDRESS = "&(taskStack0[0])";
    FILLPATTERN = "0xAA55";
};

STACK taskStack1 {
    SIZE = "128";
    STACKDIRECTION = "UP";
    BASEADDRESS = "&(taskStack1[0])";
    FILLPATTERN = "0xAA55";
};

STACK taskStack2 {
    SIZE = "256";
    STACKDIRECTION = "UP";
    BASEADDRESS = "&(taskStack2[0])";
    FILLPATTERN = "0xAA55";
};


ALARM myFirstAlarm {
    ALARMTIME = "gAlarm[0].alarmTime";
    CYCLETIME = "gAlarm[0].cycleTime";
    STATE = "(gAlarm[0].alarmTime == 0) ? 0 : 1";
    ACTION = "ActivateTask basicTaskFirst";
    COUNTER = "SystemTimer";
};

RESOURCE resBasic {
    STATE = "(gResource[0].resourceCounter != 0) ? 1 : 0";
    LOCKER  = "gResource[0].owner";
    PRIORITY = "TASK: 51";
};

RESOURCE resTimerData {
    STATE = "(gResource[1].resourceCounter != 0) ? 1 : 0";
    LOCKER  = "gResource[1].owner";
    PRIORITY = "ISR: 3";
};


CONTEXT Context_0 {
    _CPU_R0 = "((unsigned long *)(gTaskControlBlocks[0].sp))[5]";
    _CPU_R1 = "((unsigned long *)(gTaskControlBlocks[0].sp))[4]";
    _CPU_R2 = "((unsigned long *)(gTaskControlBlocks[0].sp))[3]";
    _CPU_R3 = "((unsigned long *)(gTaskControlBlocks[0].sp))[2]";
    _CPU_PC = "((unsigned long *)(gTaskControlBlocks[0].sp))[1]";
    _CPU_SR = "((unsigned long *)(gTaskControlBlocks[0].sp))[0]";
    VALID = "(gTaskControlBlocks[0].taskState != 1) &&
(gTaskControlBlocks[0].taskState != 4)";
};

CONTEXT Context_1 {
    _CPU_R0 = "((unsigned long *)(gTaskControlBlocks[1].sp))[5]";
    _CPU_R1 = "((unsigned long *)(gTaskControlBlocks[1].sp))[4]";
    _CPU_R2 = "((unsigned long *)(gTaskControlBlocks[1].sp))[3]";
    _CPU_R3 = "((unsigned long *)(gTaskControlBlocks[1].sp))[2]";
    _CPU_PC = "((unsigned long *)(gTaskControlBlocks[1].sp))[1]";
    _CPU_SR = "((unsigned long *)(gTaskControlBlocks[1].sp))[0]";
    VALID = "(gTaskControlBlocks[1].taskState != 1) &&
(gTaskControlBlocks[1].taskState != 4)";
};
```
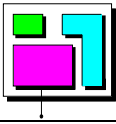
```
CONTEXT Context_2 {
    _CPU_R0 = "((unsigned long *)(gTaskControlBlocks[2].sp))[5]";
    _CPU_R1 = "((unsigned long *)(gTaskControlBlocks[2].sp))[4]";
    _CPU_R2 = "((unsigned long *)(gTaskControlBlocks[2].sp))[3]";
    _CPU_R3 = "((unsigned long *)(gTaskControlBlocks[2].sp))[2]";
    _CPU_PC = "((unsigned long *)(gTaskControlBlocks[2].sp))[1]";
    _CPU_SR = "((unsigned long *)(gTaskControlBlocks[2].sp))[0]";
    VALID = "(gTaskControlBlocks[2].taskState != 1) &&
(gTaskControlBlocks[2].taskState != 4)";
};

MESSAGECONTAINER DrvTx_Msg_0_3 {
    MSGNAME = "DrvTx";
    MSGTYPE = "QUEUED";
    QUEUESIZE = "5";
    QUEUECOUNT = "osQMsg[0].msgCount";
    FIRSTELEMENT = "osQMsg[0].rdPtr";
};

MESSAGECONTAINER DataA_Msg_1_3 {
    MSGNAME = "DataA";
    MSGTYPE = "UNQUEUED";
    QUEUESIZE = "1";
    QUEUECOUNT = "1";
    FIRSTELEMENT = "osUnqMsg[0].msgPtr";
};

MESSAGECONTAINER DataA_Msg_1_4 {
    MSGNAME = "DataA";
    MSGTYPE = "UNQUEUED";
    QUEUESIZE = "1";
    QUEUECOUNT = "1";
    FIRSTELEMENT = "osUnqMsg[0].msgPtr";
};

vs_ISR Timer2Int {
    vs_Priority = "3";
};
vs_ISR CanRxInt {
    vs_Priority = "1";
};
vs_ISR CanTxInt {
    vs_Priority = "7";
};
```
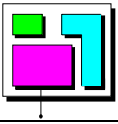
Appendix A Interpretation of SERVICETRACE

These notes are a supplement to the ORTI specification version 2.1 and describe the intended use by debuggers of the SERVICETRACE attribute. The ORTI specification states how SERVICETRACE should be maintained by the embedded application as it runs. However, the ORTI language does not allow complete interpretation of SERVICETRACE - the debugger needs to remember additional state and adopt addition display mechanisms in order to use SERVICETRACE properly. It is intended that this supplement provide enough information to allow debugger implementers to provide as much information as is possible for this attribute.

A.1   Operation of SERVICTRACE

This short section describes how the application deals with SERVICERACE. SERVICETRACE consists of two components: a service number and an entry/exit flag. All service numbers are even, and the flag occupies the lowest bit. Upon an OS service being entered the service number of and the flag are both set. Upon exit the service number is set and flag is cleared.

If the OS is implemented in such a way that all OS services are atomic (i.e. once entered they run to completion without pre-emption by interrupts etc.) then SERVICETRACE gives an indication of the last OS service called and whether or not the OS is still in it. However, such implementations are not likely to be usual. A typical OS service will consist of the following steps.

> set SERVICETRACE and entry flag
>
> parameter validation
>
> disable interrupts
>
> perform required service
>
> enable interrupts
>
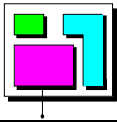> set SERVICETRACE and exit flag

In such an implementation an interrupt can occur whilst in the service and after SERVICETRACE has been set. The ISR may lead to another OS service being called, which in turn updates, SERVICETRACE. When the ISR returns to the interrupted OS service the value in SERVICETRACE indicates that the inner nested service was just left rather than still being in the initial service. This is because SERVICETRACE stores state changes rather than state.

The diagram below illustrates this situation. Boxes that contain underlined text represent changes to SERVICETRACE.

A.2   Debuggers' Interpretations

A.2.1 Overview

This section describes two possible interpretations of SERVICETRACE.

A.2.2 Simple Interpretation

The simple interpretation just involves displaying the last value of SERVICETRACE without further interpretation. In case of task switches and ISRs the value doesn't give any useful information.

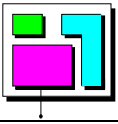## A.2.3 Intelligent Interpretation

Tracing the value of SERVICETRACE by a debugger allows a more intelligent interpretation.

SERVICETRACE has its TOTRACE attribute set. This means that a debugger potentially can trace each modification to it and therefore maintain a stack of values written. By pushing a new value onto the stack when SERVICETRACE is written with the entry flag set, and popping the top of the stack when a write takes place with the entry flag clear, the top of the stack will always indicate the service. However, again in case of task switches and ISRs the interpretation may be misleading. The following figure gives an example:
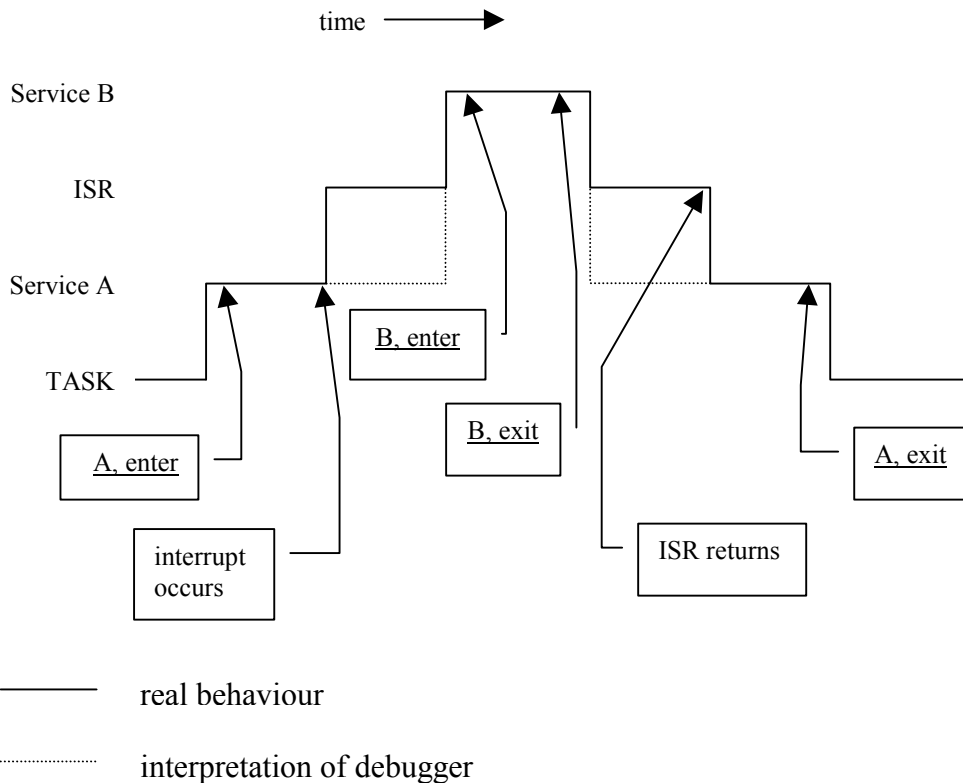
The solid line gives the real behaviour of the system and the dotted line shows the interpretation the debugger is able to display.

Debuggers with finite trace buffers can use them to track SERVICETRACE. In such circumstances the finite buffer size may mean that earlier modifications to SERVICETRACE are lost and therefore only a limited history may be provided.

History

Version	Date	Remarks
1.0 – 2.0		Not published by OSEK/VDX
2.1	17. April 2002	Authors:
		Mr. Barthelmann (3SOFT)
		Mr. Büchner (Hitex)
		Mr. Dienstbeck (Lauterbach)
		Mr. Elies (Hitex)
		Mr. Fathi (Cosmic)
		Mr. Hoogenboom (Green Hills)
		Mr. Janz (Vector)
		Mr. Kriesten IIIT, (University of Karlsruhe)
		Mr. Morgan (LiveDevices)
		Mrs. Nieser (Lauterbach)
		Mr. Nishikawa (Toyota, Europe)
		Mr. Schimpf (ETAS)
		Mr. Stehle (Vector)
		Mr. Ulcakar (iSystem)
		Mr. Vetterli (Metrowerks)
		Mr. Wertenauer (Cosmic)
		Mr. Winters (Motorola)