ERIKA Enterprise OSEK COM Manual

Intertask communication now!

version: 1.0.1
December 11, 2012

About Evidence S.r.l.

Evidence is a spin-off company of the ReTiS Lab of the Scuola Superiore S. Anna, Pisa, Italy. We are experts in the domain of embedded and real-time systems with a deep knowledge of the design and specification of embedded SW. We keep providing significant advances in the state of the art of real-time analysis and multiprocessor scheduling. Our methodologies and tools aim at bringing innovative solutions for next-generation embedded systems architectures and designs, such as multiprocessor-on-a-chip, reconfigurable hardware, dynamic scheduling and much more!

Contact Info

Address:
Evidence Srl,
Via Carducci 56
Località Ghezzano
56010 S.Giuliano Terme
Pisa - Italy
Tel: +39 050 991 1122, +39 050 991 1224
Fax: +39 050 991 0812, +39 050 991 0855

For more information on Evidence Products, please send an e-mail to the following address: info@evidence.eu.com. Other informations about the Evidence product line can be found at the Evidence web site at: http://www.evidence.eu.com.

Contents

1 Introduction

1.1 Erika Enterprise **and** OSEK COM

Erika Enterprise is a free of charge, open-source RTOS implementation of the OSEK/VDX API, available for various microcontrollers on the web page http://erika.tuxfamily.org.

For a complete reference of the Erika Enterprise API, please refer to the Erika Enterprise Reference Manual available for download on the Erika Enterprise website.

This manual shows the API and specification of the OSEK COM layer implemented in Erika Enterprise. The current version covers the implementation of the CCCA and CCCB conformance classes as specified in the OSEK COM specification version 3.03 available for download on the web page http://www.osek-vdx.org.

The features provided by the OSEK COM implementation available in Erika Enterprise are:

- Support for Conformance Classes CCCA and CCCB[1];
- Support for internal messaging, queued and non-queued;
- Support for Notification Class 1[2];
- Support for Error Handling;
- Support for configuration using the OIL language;

1.2 Acknowledgement

We would like to thank Francesco Bertozzi for the initial implementation of the OSEK COM stack done in year 2004 as part of his Master Thesis.

[1] The project also provides a beta version of CCC0 and CCC1

[2] The project also provides a beta version for Notification Classes 2, 3 and 4.

2 API reference

2.1 Introduction

Erika Enterprise provides a communication environment according to OSEK COM specification, version 3.0.3. OSEK COM endows Erika Enterprise with a mechanism to transfer data between tasks and/or interrupt service routines (ISRs). The communication is based on message objects. A message containing application data is sent by sending message objects and received by receiving message objects. Message objects are identified by message identifiers (IDs). For internal communication, sending message objects can store messages in zero or more receiving message objects. Zero or more senders can send messages to the same sending message object. This allows a n:m communication. All message objects are statically defined using the OIL language.

The current implementation supports internal communication for queue and unqueued messages. If a receiving message object is defined as "queued", any message received by this object can only be read once and the read operation removes the oldest message from the queue. For a receiving message object defined as "unqueued", each message can be read more times. The message object returns the last received message every time it is read.

In order to used the COM support, the application should add the following includes in its source files:

```
#include "com/ee_com.h"
#include "com/com/inc/ee_cominit.h"
#include "com/com/inc/ee_api.h"
```

The COM layer and the message objects are configurated in the OIL configuration file.

2.1.1 Communication Conformance Classes

Depending on the specific application and the hardware platform, different levels of communication service can be required. OSEK COM defines these service levels through Communication Conformance Classes (CCCs). The current version of the Erika Enterprise OSEK COM implementation provides two conformance classes:

- CCCA (internal communication only). It supports only unqued messages. Message status is not supported. It provides the notification mechanism defined by Notification Class 1, except for the Flag notification mechanism.

Service	Background Task	Task	ISR2	COMErrorHook
StartCOM		√	√	
StopCOM		√	√	
GetCOMApplicationMode		√	√	
InitMessage		√	√	
SendMessage		√	√	
ReceiveMessage		√	√	
GetMessageStatus		√	√	
COMErrorGetServiceId				√
ReadFlag		√	√	
ResetFlag		√	√	

Table 2.1: This table lists the environments where primitives can be called. Please note that the Background task refers to the possibility for `StartOS` to return, which is now deprecated.

- CCCB (internal communication only). It supports queued and unqued message objects. It provides all the notification mechanisms provided by Notification Class 1. Moreover, it supports message status information services.

2.1.2 Available primitives

Erika Enterprise provides a set of primitives to exchange messages between tasks according to OSEK COM specification. The primitives can be called in different situations. The complete list of primitives is listed in Table 2.1, together with the locations where it is legal to call these functions.

2.1.3 Notfication Classes

To determine the final status of a send or receive operation, the application can exploit the notification services provided by the COM layer. The application does not need to check directly the occurrence of an event by calling a COM API primitive, but it is notified as soon as the specific event has occurred. A notification is configured for each message object in the OIL configuration file.

Only notification class 1 is supported for conformance classes CCCA and CCCB. This means that the notification mechanism is only used to notify the receiver of a message. The configured notification mechanism is called whenever a message has been stored in the receiving message object.

The following notification mechanisms are provided:

- Callback routines provided by the application;
- Flags set by the COM layer, read and reset by the application through the specific API service;
- Tasks activated by the COM layer;
- Events set for application tasks by the COM layer.

2.2 Constants

This section provides a list of the constants that can be used by the developer for writing applications that use the COM support.

2.2.1 Error List

Description

This is the list of the error values returned by the COM primitives:

```
#define E_OK                                0
#define E_COM_ID                            1
#define E_COM_LENGTH                        2
#define E_COM_LIMIT                         3
#define E_COM_NOMSG                         4
#define E_COM_SYS_DISCONNECTED              5
```

2.2.2 Boolean constants

Description

These constants are used to represent boolean values FALSE and TRUE.

```
#define COM_FALSE                           0
#define COM_TRUE                            1
```

2.2.3 COMService IDs

Description

This is the list of Service IDs values that can be returned by COMErrorGetServiceId:

```
#define COMServiceId_NoError                0
#define COMServiceId_StartCOM               1
#define COMServiceId_StopCOM                2
#define COMServiceId_GetCOMApplicationMode  3
#define COMServiceId_InitMessage            4
#define COMServiceId_ReadFlag               7
#define COMServiceId_ResetFlag              8
#define COMServiceId_SendMessage            9
#define COMServiceId_ReceiveMessage        10
#define COMServiceId_SendZeroMessage       13
#define COMServiceId_GetMessageStatus      14
#define COMServiceId_COMErrorGetServiceId  15
```

2.2.4 COM_SHUTDOWN_IMMEDIATE

Description

This is the default and only possible value for COM Suthdown Mode that can be passed to StopCOM.

2.3 Types

This section contains a description of the data types used by the COM interface of Erika Enterprise. When the size of a type is specified to be of the size of a machine register, it is intended that the type has the same size of the CPU general purpose register.

2.3.1 ApplicationDataRef

Description

This is a pointer to application data. This pointer can be passed to `InitMessage`, `ReceiveMessage` and `SendMessage`.

2.3.2 COMApplicationModeType

Description

This type is an unsigned 8-bit integer type used to store COM Application Mode IDs.

2.3.3 COMServiceIdType

Description

This is an enumeration type used to store COM Service IDs, and it is used within the `COMErrorGetServiceId`.

2.3.4 COMShutdownModeType

Description

This type is an unsigned 8-bit integer type used to store COM Shutdown Mode IDs. The default and only possibile value is `COM_SHUTDOWN_IMMEDIATE`.

2.3.5 FlagValue

Description

This is an enumeration type used to store the current state of a message flag. Possible values are `COM_FALSE` and `COM_TRUE`, see Section 2.2.2

2.3.6 MessageIdentifier

Description

This is an unsigned char used as ID for message objects.

2.3.7 StatusType

Description

This type is an unsigned char used to store function error return values.

2.4 Object Definitions

The following macro has to be used when defining Message Callbacks. A message callback is a notification mechanism that can be configured for each message object in the OIL configuration file.

2.4.1 COMCallback

Synopsis

```
COMCallback(c)
```

Description

This macro is used to declare and to define a message callback.

Parameters

- `c` Name of the message callback.

Conformance

CCCA, CCCB

2.5 COM Initialization Primitives

The OSEK COM implementation of Erika Enterprise supports the specification of a set of *COM Application Modes*. Application modes are startup configurations that are used to configure the communication system support for a certain mode of operation. COM Application modes are passed to `StartCOM` in order to start the system in a specific mode. Application modes are defined by the application in the OIL configuration file. Examples of COM Application Modes are "Debug Mode", "Releas Mode", "Test Mode" and so forth.

The COM support is initialized by `StartCOM` which initializes the system resources utilized by the COM module. `StartCOM` sets the initial value of an unqueued message to 0 if an initial value is not specified in the OIL configuration file. If a message object has an initial value specified in the OIL file, `StartCOM` initializes the message object with that specific value. `StartCOM` initilaizes queued message objects with the number of receieved messages equal to 0. For internal transmit messages there is not any initialization procedure. An application can directly initialize a message by `InitMessage`. If configured in the OIL file, the application can provides a `StartCOMExtension` primitive used to add specific initialization functions, e.g. it can use `InitMessage` to initialize the application messages.

Finally, `StopCOM` shall be used to stop the communication system releasing its resources. Note that, after calling `StopCOM` the system can be re-initialized be calling `StartCOM` which resets all communication resources to their initial status.

The start-up primitives are listed in the following Sections.

2.5.1 StartCOM

Synopsis

```
StatusType StartCOM (COMApplicationModeType Mode)
```

Description

The user shall call this primitive to initialize and start the communication system in a specific application mode, defined by `Mode`. This routine shall be called from a task. When `COMSTARTCOMEXTENSION` is set to `TRUE` within the OIL configuration file, before returning, this primitive calls `StartCOMExtension`. `StartCOMExtension` is provided by the application.

Parameters

- `Mode` COM Application Mode

Return Values

- `E_OK` No errors.
- `value` A value returned from StartCOMExtension if different from `E_OK`.
- `E_COM_ID` (Extended) If `Mode` is not valid.

Conformance

CCCA, CCCB

2.5.2 StopCOM

Synopsis

```
StatusType StopCOM (COMShutdownModeType Mode)
```

Description

This primitive is called to terminate all communication activities and release their resources. The only possible value for `Mode` is `COM_SHUTDOWN_IMMEDIATE`.

Parameters

- `Mode` Shutdown Mode

Return Values

- `E_OK` No errors.
- `E_COM_ID` (Extended) If `Mode` is different from `COM_SHUTDOWN_IMMEDIATE`.

Conformance

CCCA, CCCB

2.5.3 GetCOMApplicationMode

Synopsis

```
COMApplicationModeType GetCOMApplicationMode(void)
```

Description

This primitive returns the current COM Application Mode. It shall not be called before StartCOM.

Return Values

- `value` Current COM Application Mode

Conformance

CCCA, CCCB

2.5.4 InitMessage

Synopsis

```
StatusType InitMessage (MessageIdentifier Message, ApplicationDataRef DataRef)
```

Description

This primitive can be used to initializes a message object with application data passed by the pointer `DataRef`. To change the default initialization, it is possible to call this primitive from `StartCOMExtension`.

Parameters

- `Message` Message object ID
- `DataRef` Pointer to application data for the message object being initialized.

Return Values

- `E_OK` No errors.
- `E_COM_ID` (Extended) If the message identifier `Message` is not valid.

Conformance

CCCA, CCCB

2.6 Message Communication Primitives

This section provides the COM primiteves used to transmit and receive messages.

2.6.1 SendMessage

Synopsis

```
StatusType SendMessage (MessageIdentifier Message, ApplicationDataRef DataRef)
```

Description

This primitive stores in the message object identified by `Message` the application data-pointed by `DataRef`.

Parameters

- `Message` Message object ID
- `DataRef` Pointer to the application data.

Return Values

- `E_OK` No errors.
- `E_COM_ID` (Extended) If message identifier `Message` is not valid.

Conformance

CCCA, CCCB

2.6.2 ReceiveMessage

Synopsis

```
StatusType ReceiveMessage (MessageIdentifier Message, ApplicationDataRef DataRef)
```

Description

This primitive stores the data from the message object `Message` in the application message memory pointed by `DataRef`. All flags associated with `Message` are reset. If a message queue overflow happened since the last call of this primitive, an `E_COM_LIMIT` is returned, nevertheless a message is store in `DataRef` and the overflow is cleared.

Parameters

- `Message` Message object ID
- `DataRef` Pointer to the application message data.

Return Values

- `E_OK` No errors.
- `E_COM_NOMSG` If `Message` is a queued message object and it is empty.
- `E_COM_LIMIT` If `Message` is a queued message object and a queue overflow occurred since the last call of this primitive for `Message`. `E_COM_LIMIT` indicates that at least a message was discarded.
- `E_COM_ID` (Extended) If message identifier `Message` is not valid.

Conformance

CCCA, CCCB

2.6.3 GetMessageStatus

Synopsis

```
StatusType GetMessageStatus(MessageIdentifier Message)
```

Description

This primitive returns the current status of the message object identified by `Message`.

Parameters

- `Message` Message object ID

Return Values

- `E_COM_NOMSG` If `Message` is a queued message object and it is empty.
- `E_COM_LIMIT` If `Message` is a queued message object and a queue overflow occurred since the last call of the primitive for `Message`.
- `E_OK` If there are no error indications, if `Message` is not empty and an overflow condition has not occurred since the last call to `ReceiveMessage`
- `E_COM_ID` (Extended) If the message identifier `Message` is not valid (e.g. `Message` is invalid if it is not a queued message object).

Conformance

CCCB

2.7 Notification Primitives

The following primitives can be used to read and reset the status of the flags used as notification mechanism.

2.7.1 ReadFlag

Synopsis

`FlagValue ReadFlag_<Flag>(void)`

Description

This primitive reads the status of the flag identified by the name `<Flag>`.

Return Values

- `COM_TRUE` `<Flag>` is set
- `COM_FALSE` `<Flag>` is not set.

Conformance

CCCB

2.7.2 ResetFlag

Synopsis

`void ResetFlag_<Flag>(void)`

Description

This routine reset the status of the flag identified by `<Flag>`.

Conformance

CCCB

2.8 Primitives Provided by the Application

The following routines are optional and must be provided by the application. These routines are enabled through the OIL configuration file of the specific application.

2.8.1 StartCOMExtension

Synopsis

```
StatusType StartCOMExtension(void)
```

Description

This primitive is executed at the end of the `StartCOM` routine. It can be used to add start-up configurations provided by the user. This primitive is called by `StartCOM` if `COMSTARTCOMEXTENSION` is set to `TRUE` in the OIL configuration file.

Return Values

- `E_OK` No errors
- An implementation specific status code if an error occurred.

Conformance

CCCA, CCCB

2.8.2 COMErrorHook

Synopsis

```
void COMErrorHook (StatusType err)
```

Description

This primitive is called by the COM services, such as `StartCOM`, `ReceiveMessage` etc., when a status code different from `E_OK` is returned. This primitive is enabled by setting `COMERRORHOOK` to `TRUE` in the OIL configuration file.

Parameters

- `err` ID of the error.

Conformance

CCCA, CCCB

2.9 Error Handling Primitives and COMErrorHook Macros

These macros are meaningful inside the COMErrorHook Hook function, and are used to better understand the source of the error. In particular, COMErrorHook receives as parameter the error that is raised by the primitive. Then, a call to COMErrorGetServiceId returns informations about which primitive caused the error. Finally, calls to the macros COMError_XXX_YYY returns the values of the YYY parameter of the primitive XXX.

2.9.1 COMErrorGetServiceId

Synopsis

```
COMServiceIdType COMErrorGetServiceId(void)
```

Description

This function may be used inside COMErrorHook to return the Service ID that generated the error that caused the call to COMErrorHook.

Return Values

- Service ID The service ID causing the error.

Conformance

CCCA, CCCB

2.9.2 COMError_GetMessageStatus_Message

Synopsis

```
MessageIdentifier COMError_GetMessageStatus_Message(void)
```

Description

The function returns the Message parameter passed to GetMessageStatus. The function must be used inside COMErrorHook after having discovered using COMErrorGetServiceId that the error was caused by that function.

Return Values

- Message The value of the Message ID passed to GetMessageStatus.

Conformance

CCCA, CCCB

2.9.3 COMError_InitMessage_Message

Synopsis

```
MessageIdentifier COMError_InitMessage_Message(void)
```

Description

The function returns the `Message` parameter passed to `InitMessage`. The function must be used inside `COMErrorHook` after having discovered using `COMErrorGetServiceId` that the error was caused by that function.

Return Values

- `Message` The value of the `Message` ID passed to `InitMessage`.

Conformance

CCCA, CCCB

2.9.4 COMError_InitMessage_DataRef

Synopsis

```
ApplicationDataRef COMError_InitMessage_DataRef(void)
```

Description

The function returns the `DataRef` parameter passed to `InitMessage`. The function must be used inside `COMErrorHook` after having discovered using `COMErrorGetServiceId` that the error was caused by that function.

Return Values

- `DataRef` The value of the pointer `DataRef` passed to `InitMessage`.

Conformance

CCCA, CCCB

2.9.5 COMError_ReceiveMessage_Message

Synopsis

```
MessageIdentifier COMError_ReceiveMessage_Message(void)
```

Description

The function returns the `Message` parameter passed to `ReceiveMessage`. The function must be used inside `COMErrorHook` after having discovered using `COMErrorGetServiceId` that the error was caused by that function.

Return Values

- `Message` The value of the `Message` ID passed to `ReceiveMessage`.

Conformance

CCCA, CCCB

2.9.6 COMError_ReceiveMessage_DataRef

Synopsis

```
ApplicationDataRef COMError_SendMessage_DataRef(void)
```

Description

The function returns the `DataRef` parameter passed to `ReceiveMessage`. The function must be used inside `COMErrorHook` after having discovered using `COMErrorGetServiceId` that the error was caused by that function.

Return Values

- `DataRef` The value of the pointer `DataRef` passed to `ReceiveMessage`.

Conformance

CCCA, CCCB

2.9.7 COMError_SendMessage_Message

Synopsis

```
MessageIdentifier COMError_SendMessage_Message(void)
```

Description

The function returns the `Message` parameter passed to `SendMessage`. The function must be used inside `COMErrorHook` after having discovered using `COMErrorGetServiceId` that the error was caused by that function.

Return Values

- `Message` The value of the `Message` ID passed to `SendMessage`.

Conformance

CCCA, CCCB

2.9.8 COMError_SendMessage_DataRef

Synopsis

```
ApplicationDataRef COMError_SendMessage_DataRef(void)
```

Description

The function returns the `DataRef` parameter passed to `SendMessage`. The function must be used inside `COMErrorHook` after having discovered using `COMErrorGetServiceId` that the error was caused by that function.

Return Values

- `DataRef` The value of the pointer `DataRef` passed to `SendMessage`.

Conformance

CCCA, CCCB

2.9.9 COMError_StartCOM_Mode

Synopsis

```
COMApplicationModeType COMError_StartCOM_Mode(void)
```

Description

The function returns the `Mode` parameter passed to `StartCOM`. The function must be used inside `COMErrorHook` after having discovered using `COMErrorGetServiceId` that the error was caused by that function.

Return Values

- `Mode` The value of the `Mode` parameter passed to `StartCOM`.

Conformance

CCCA, CCCB

2.9.10 COMError_StopCOM_Mode

Synopsis

```
COMShutdownModeType COMError_StopCOM_Mode(void)
```

Description

The function returns the `Mode` parameter passed to `StopCOM`. The function must be used inside `COMErrorHook` after having discovered using `COMErrorGetServiceId` that the error was caused by that function.

Return Values

- `Mode` The value of the `Mode` parameter passed to `StopCOM`.

Conformance

CCCA, CCCB

3 History

Version	Comment
1.0.0	First version of the OSEK COM manual.
1.0.1	Fixed OSEK/VDX Copyright notice.

Index