



Elektrobit

# EB tresos<sup>®</sup> IOC user's guide

Date: 2016-12-08, Document version 0.0.8, Status: RELEASED



## **EB tresos® IOC user's guide**

Elektrobit Automotive GmbH  
Am Wolfsmantel 46  
91058 Erlangen, Germany  
Phone: +49 9131 7701 0  
Fax: +49 9131 7701 6333  
E-mail: [info.automotive@elektrobit.com](mailto:info.automotive@elektrobit.com)

## **Technical support**

### **Europe**

Phone: +49 9131 7701 6060

### **Japan**

Phone: +81 3 5577 6110

### **USA**

Phone: +1 888 346 3813

## **Support URL**

<https://www.elektrobit.com/support>

## **Legal notice**

Confidential and proprietary information

ALL RIGHTS RESERVED. No part of this publication may be copied in any form, by photocopy, microfilm, retrieval system, or by any other means now known or hereafter invented without the prior written permission of Elektrobit Automotive GmbH.

ProOSEK®, tresos®, and street director® are registered trademarks of Elektrobit Automotive GmbH.

All brand names, trademarks and registered trademarks are property of their rightful owners and are used only for description.

© 2016 Elektrobit Automotive GmbH

# Table of Contents

1. Document history .....	5
2. Begin here .....	6
3. About this documentation .....	7
3.1. Typography and style conventions .....	7
4. Safe and correct use of EB tresos IOC .....	9
4.1. Intended usage of EB tresos IOC .....	9
4.2. Possible misuse of EB tresos IOC .....	9
4.3. Target group and required knowledge .....	9
4.4. Quality standards compliance .....	9
4.5. Suitability of EB tresos IOC for mass production .....	10
5. EB tresos product line support .....	11
6. Background information .....	12
7. Using the IOC .....	13
7.1. Configuring the IOC .....	13
7.1.1. Creating a new channel .....	13
7.1.2. Communication semantics .....	13
7.1.2.1. Last is best .....	13
7.1.2.2. Queued .....	14
7.1.3. Senders and receivers .....	14
7.1.4. Service mode .....	14
7.1.5. Lock configuration .....	15
7.1.5.1. Cross-core lock configuration .....	15
7.1.5.2. Core-local lock configuration .....	15
7.1.6. Data elements .....	16
7.1.7. Type headers .....	16
7.1.8. Channel initialization .....	17
7.2. Linking the IOC .....	17
7.2.1. Linking states of non-trapping APIs .....	17
7.2.2. Linking states of trapping APIs .....	18
7.2.3. Readable regions .....	18
8. IOC reference manual .....	20
8.1. IOC limits .....	20
8.2. IOC types .....	21
8.3. IOC error code reference .....	22
8.4. IOC API reference .....	24
8.5. IOC library interface reference .....	36
8.6. Deviations from the AUTOSAR standard .....	47
8.7. Limitations .....	48
9. Supplementary information .....	49

9.1. Operating system .....	49
9.2. CPU family specific information .....	49
9.2.1. ARM .....	49
9.2.1.1. User-mode spinlocks .....	49
9.2.1.2. Cache settings .....	49
9.2.2. ARM64 .....	50
9.2.2.1. User-mode spinlocks .....	50
9.2.2.2. Cache settings .....	50
9.2.3. Cortex-M .....	50
9.2.3.1. User-mode spinlocks .....	50
9.2.3.2. Cache settings .....	50
9.2.4. Power Architecture .....	50
9.2.4.1. User-mode spinlocks .....	50
9.2.4.2. Cache settings .....	51
9.2.5. RH850 .....	51
9.2.5.1. User-mode spinlocks .....	51
9.2.5.2. Cache settings .....	51
9.2.6. TriCore .....	51
9.2.6.1. User-mode spinlocks .....	51
9.2.6.2. Cache settings .....	51
9.2.7. Windows and Linux on IA32 or AMD64 .....	52
9.2.7.1. User-mode spinlocks .....	52
9.2.7.2. Cache settings .....	52
A. Document configuration information .....	53
Glossary .....	55
Bibliography .....	57

# 1. Document history

Version	Date	Author	State	Description
0.0.1	2016-04-14	sidu8158, mist9353	DRAFT	Write user's guide for EB tresos IOC.
0.0.2	2016-04-29	mist9353	PROPOSED	Add restrictions on Relnit and user-mode spinlocks.
0.0.3	2016-05-04	mist9353	PROPOSED	Add supplementary information about PA, RH850, CORTEXM, ARM64, and WINLIN.
0.0.4	2016-06-22	mist9353	PROPOSED	<ul style="list-style-type: none"> <li>▶ Add a remark on deadlock prevention to the lock configuration section.</li> <li>▶ Clarify relationship between locReceive and locEmptyQueue.</li> </ul>
0.0.5	2016-08-12	made3765, mist9353	PROPOSED	Incorporate TE review findings.
0.0.6	2016-11-14	mist9353	PROPOSED	Rework according to walkthrough findings <a href="#">[IOCU-GRR1]</a> .
0.0.7	2016-11-24	mist9353	PROPOSED	Rework according to walkthrough findings <a href="#">[IOCU-GRR1]</a> .
0.0.7	2016-11-30	mist9353	RELEASED	Passed walkthrough <a href="#">[IOCUGRR1]</a> .
0.0.8	2016-12-07	stpo8218	PROPOSED	Fix reference to IOC Safety Manual.
0.0.8	2016-12-08	stpo8218	RELEASED	The document has passed its verification; see <a href="#">[IOCUGRR1]</a> .

Table 1.1. Document history

## 2. Begin here

The purpose of this document is to provide the reader with the information necessary to configure and use EB tresos IOC.

It is assumed that the reader is familiar with the AUTOSAR-OS module and its interfaces. If you are unfamiliar with AUTOSAR-OS, refer to the EB tresos AutoCore OS Documentation [\[ASCOS\\_USERGUIDE\]](#) for a better description of the services provided by a standard AUTOSAR-OS.

[Chapter 2, “Begin here”](#) (this chapter) describes the layout of the document and provides a glossary and a list of referenced documents.

[Chapter 3, “About this documentation”](#) describes the style and typography of this document.

[Chapter 4, “Safe and correct use of EB tresos IOC”](#) describes the features and limitations of the IOC.

[Chapter 5, “EB tresos product line support”](#) lists contact phone numbers.

[Chapter 6, “Background information”](#) introduces the IOC.

[Chapter 7, “Using the IOC”](#) describes how to configure and link the IOC.

[Chapter 8, “IOC reference manual”](#) contains reference material including a detailed API reference.




[Chapter 9, “Supplementary information”](#) describes CPU family-dependent topics.

## 3. About this documentation

### 3.1. Typography and style conventions

Throughout the documentation you see that words and phrases are displayed in bold or italic font, or in Monospace font. To find out what these conventions mean, consult the following table. All default text is written in Arial Regular font without any markup.

Convention	Item is used	Example
Arial italics	to define new terms	The <i>basic building blocks</i> of a configuration are module configurations.
Arial italics	to emphasize	If your project's release version is mixed, all content types are available. It is thus called <i>mixed version</i> .
Arial italics	to indicate that a term is explained in the glossary	...exchanges <i>protocol data units (PDUs)</i> with its peer instance of other ECUs.
Arial boldface	for menus and submenus	Choose the <b>Options</b> menu.
Arial boldface	for buttons	Select <b>OK</b> .
Arial boldface	for keyboard keys	Press the <b>Enter</b> key
Arial boldface	for keyboard combination of keys	Press <b>Ctrl+Alt+Delete</b>
Arial boldface	for commands	Convert the XDM file to the newer version by using the <b>legacy convert</b> command.
Monospace font (Courier)	for file and folder names, also for chapter names	Put your script in the <code>function_name/abc-folder</code>
Monospace font (Courier)	for code	<pre>for (i=0; i&lt;5; i++) { /* now use i */ }</pre>
Monospace font (Courier)	for function names, methods, or routines	The <code>cos</code> function finds the cosine of each array element. Syntax line example is <code>MLGetVar ML_var_name</code>
Monospace font (Courier)	for user input/indicates variable text	Enter a <code>three-digit prefix</code> in the menu line.
Square brackets [ ]	for optional parameters; for command syntax with optional parameters	<code>insertBefore [&lt;opt&gt;]</code>

Convention	Item is used	Example
Curly brackets {}	for mandatory parameters; for command syntax with mandatory parameters (in curly brackets)	<code>insertBefore {&lt;file&gt;}</code>
Three dots ...	for further parameters	<code>insertBefore [&lt;opt&gt;...]</code>
A vertical bar	to separate parameters in a list from which one parameters must be chosen or used; for command syntax, indicates a choice of parameters	<code>allowinvalidmarkup {on off}</code>
Warning	to show information vital for the success of your configuration	<b>WARNING</b>  <b>This is a warning</b> This is what a warning looks like.
Notice	to give additional important information on the subject	<b>NOTE</b>  <b>This is a notice</b> This is what a notice looks like.
Tip	to provide helpful hints and tips	<b>TIP</b>  <b>This is a tip</b> This is what a tip looks like.



## 4. Safe and correct use of EB tresos IOC

### 4.1. Intended usage of EB tresos IOC

EB tresos IOC is intended to be used in automotive projects based on AUTOSAR. For more information about the AUTOSAR consortium, see [www.autosar.org](http://www.autosar.org).

### 4.2. Possible misuse of EB tresos IOC

- ▶ If you use the product in mass production projects without having undergone the mass production review process, it is likely that the product functions differently than defined. Elektrobit Automotive GmbH is not liable for such misuse.

To use the product for mass production, you must undergo the mass production review process. To find out what this process consists of, please refer to the `Maintenance and Support Annex` document, chapter `Mass Production Projects`. You have received this document together with the product quote supplied by EB.

- ▶ If you use the product in applications that are not defined by the AUTOSAR consortium (see [www.autosar.org](http://www.autosar.org)), the product and its technology may not conform to the requirements of your application. Elektrobit Automotive GmbH is not liable for such misuse.

### 4.3. Target group and required knowledge

The target audience of this document are basic software engineers and application developers.

You should have programming skills and experience in programming AUTOSAR-compliant ECUs.

### 4.4. Quality standards compliance

EB tresos IOC has been developed following processes that have been assessed and awarded Automotive SPICE Level 2.

## 4.5. Suitability of EB tresos IOC for mass production

The suitability of EB tresos IOC for mass production is defined by the mass production review process. Pre-requisite for this process is the *quality statement*. This quality statement is available for each release, platform, and derivative of EB tresos AutoCore OS and EB tresos Safety OS.

The quality statement for your specific delivery, which is determined by release, platform, and derivative you ordered, is available on the *EB Command* server. The link to EB Command as well as your user login and password were sent to you via email.

EB tresos IOC is never delivered without an operating system. Its quality statement is included in the quality statement of the respective operating system. The quality statement of the EB tresos Safety OS IOC variant is included in the EB tresos Safety OS quality statement. The quality statement of the EB tresos AutoCore OS IOC variant is included in the EB tresos AutoCore OS quality statement.

## 5. EB tresos product line support

### **Europe**

Phone: +49 9131 7701 6060

### **Japan**

Phone: +81 3 5577 6110

### **USA**

Phone: +1 888 346 3813

### **Support URL**

<https://www.elektrobit.com/support>

## 6. Background information

EB tresos IOC is a module which handles the communication between AUTOSAR applications, especially between applications which are located on different cores in a multi-core system. The AUTOSAR specifications define IOC as part of the OS. Therefore, EB tresos IOC is configured in the OS configuration, but the source code of the IOC is contained in its own module.

The IOC is designed according to the procedures required for ASIL-D as laid out in [\[ISO26262\\_1ST\]](#). Thus, you can use it in safety-related systems up to that integrity level, provided that the conditions in the IOC Safety Manual [\[IOCSAFETYMAN\]](#) are observed and the quality statement, corresponding to the respective version of the EB tresos IOC, contains a safety approval.

The IOC is designed to be as lightweight as possible for reasons of performance and simplicity. In several cases it deviates from the AUTOSAR specification. The deviations are listed in [Section 8.6, “Deviations from the AUTOSAR standard”](#).

You can find further information in [\[ASCOS\\_USERGUIDE\]](#) or [\[AUTOSAROS40SPEC\]](#).

## 7. Using the IOC

### 7.1. Configuring the IOC

To use EB tresos IOC with EB tresos Studio, you need to add the `Ioc` module to your EB tresos Studio project. Instructions for adding modules to your EB tresos Studio project are available in the EB tresos Studio documentation, chapter *Editing module configurations*.

The IOC configuration in EB tresos Studio is part of the EB tresos AutoCore OS configuration. You can find a description of every configuration item in the EB tresos AutoCore OS Documentation [\[ASCOS\\_USERGUIDE\]](#), chapter *EB tresos AutoCore OS configuration language* under `OsIoc`.

#### 7.1.1. Creating a new channel

To create a new channel, add a new object to the `OsIocCommunication` list. The newly created IOC channel has the following configuration parameters:

- ▶ The communication semantic, which specifies how data is transmitted
- ▶ The [applications](#), which shall send to and receive from this channel
- ▶ The service mode of the channel
- ▶ The lock configuration, which protects the channel's state from concurrent accesses
- ▶ The data elements, which are transmitted by this channel
- ▶ Further configuration-dependant parameters

#### 7.1.2. Communication semantics

The communication semantics `OsIocCommunicationSemantics` specify, how data is transmitted. You can either choose last-is-best communication `OsIocLastIsBestCommunication` or queued communication `OsIocQueuedCommunication`.

##### 7.1.2.1. Last is best

Last-is-best channels, which are also called *unqueued* or *buffer* channels, behave like global variables. If you read them using `IocRead`, you get the last value written by `IocWrite`. In contrast to global variables, the values read from IOC channels are always consistent.

The following API functions operate on last-is-best channels: [locWrite](#), [locWriteGroup](#), [locRead](#), [locReadGroup](#), and [loc\\_ReInit](#).

### 7.1.2.2. Queued

Queued channels adhere to the FIFO principle. The message, which was sent to the channel first, is the first one to be received. The number of queue entries is specified by the `OsIocBufferLength` parameter.

The following APIs operate on queued channels: [locSend](#), [locSendGroup](#), [locReceive](#), [locReceiveGroup](#), and [locEmptyQueue](#).

## 7.1.3. Senders and receivers

The `OsIocReceiverProperties` and `OsIocSenderProperties` specify which [applications](#) receive data from and send data to this channel. Queued channels support multiple senders, but only one receiver ( $n:1$  communication). Last-is-best channels support  $n:m$  communication, so there can be multiple writers and readers. To protect the consistency of your channel, you must use an appropriate lock configuration, see [Section 7.1.5, “Lock configuration”](#).

If there are multiple senders, you must specify a sender which uses the configuration parameter `OsIocSenderId`. This `SenderId` is then appended to the identifiers of the generated functions for send operations. Those are [locWrite\[Group\]](#), [locSend\[Group\]](#) and [loc\\_ReInit](#).

## 7.1.4. Service mode

The service mode of a channel determines in which context calls to the API of an IOC channel are handled. Receive and read operations are always serviced in the caller's context. For send and write operations, this is chosen by the `OsIocSenderApiIsTrapping` parameter.

If `OsIocSenderApiIsTrapping` is set to `true`, the call to the IOC becomes a [system call](#). Hence, the send/write operation is executed in kernel context. Such sender APIs are called [trapping](#) APIs.

If `OsIocSenderApiIsTrapping` equals `false`, the call to the IOC is an ordinary function call, so the operation is executed in the context of the calling [thread](#). Such sender APIs are called [non-trapping](#) APIs.

*Trapping* APIs are also called *kernel-mode* APIs, because their requests are serviced in the [kernel's](#) context. *Non-Trapping* APIs are also called *user-mode* APIs, because they are handled in the context of the [thread](#), which uses the IOC.

If no system calls are required, user-mode APIs outperform kernel-mode APIs. But be aware: System calls could become necessary, depending on the lock configuration and other configuration parameters, see [Chapter 9](#).

[“Supplementary information”](#). In this case, using the channel in kernel-mode could become faster than user-mode.

If you use EB tresos AutoCore OS without EB tresos Safety OS and memory protection is enabled, channels that have multiple senders should run in kernel-mode. That is, because shared memory is hard to express in the EB tresos AutoCore OS memory protection model.

If you use EB tresos Safety OS, you can use kernel-mode APIs to reduce the number of memory regions necessary. If there are multiple senders in user-mode, the channel state must be mapped to a [memory region](#), which is accessible by all senders, see [Section 7.2, “Linking the IOC”](#). To achieve this, it might be necessary to add a new memory region. In some configurations memory regions might be a scarce resource. If a channel is used in kernel-mode instead, no additional regions are required, because the channel state is mapped to the [kernel partition](#).

## 7.1.5. Lock configuration

The lock configuration is used to protect IOC channel states from concurrent accesses. There are two types of locks. They are both taken from the sender side. `OsIocUseInterCoreLock` is a cross-core lock, which coordinates senders, which run on different cores. `OsIocUseIntraCoreLock` is a core-local lock, which prevents the sender from being preempted by other senders, which run on the same core.

In the following subsections *send operation* refers to every operation, which modifies the sender state of the channel. These operations are all forms of [locWrite\[Group\]](#), [locSend\[Group\]](#), and [loc\\_ReInit](#). [locEmptyQueue](#) can also be a send operation if it is called from a sender [application](#). But in this case, it must never overlap with any other call which operates on the same channel.

Moreover, all IOC operations may be executed concurrently and without any locks as long as they do not operate on the same channel. So in the following subsections all operations are considered to run on the same channel.

### 7.1.5.1. Cross-core lock configuration

A channel needs a cross-core lock if send operations can be executed concurrently from different cores. This is never the case if you have only one sender application. Note, that if user-mode spinlocks are not supported for your hardware, this requires a [system call](#). For more information, see [Chapter 9, “Supplementary information”](#).

### 7.1.5.2. Core-local lock configuration

A channel needs a core-local lock if there are multiple sender [threads](#), which run on the same core, and at least one of them could preempt another during a send operation. In this case a core-local lock is required, which locks on the highest priority and level among the sender threads. Note, that this may require a [system call](#), see [Chapter 9, “Supplementary information”](#).

If a channel uses a cross-core lock, you should additionally configure a core-local lock. This is done to prevent deadlocks. To be precise, you need a core-local lock, if a thread could be preempted by a higher priority thread while holding the cross-core lock and this higher priority thread could then take any other cross-core lock i.e. an AUTOSAR spinlock or another IOC cross-core lock.

In addition to that, [locWrite\[Group\]](#) operations must not be preempted by [locRead\[Group\]](#) operations, which run on the same core. If this happens, [locRead\[Group\]](#) waits for the end of the [locWrite\[Group\]](#) operation forever. To prevent this, use a core-local lock, which locks on the highest priority and level among the reader threads, which can preempt a writer thread.

You must ensure that no call to [locReceive\[Group\]](#) is preempted by another call to [locReceive\[Group\]](#), regardless of the core-local lock configuration.

If you use IOC together with EB tresos Safety OS and the channel's send operations are executed in kernel-mode, you can omit the core-local lock. That is, because a microkernel system call cannot be preempted by another one. For more information on service modes, see [Section 7.1.4, "Service mode"](#).

## 7.1.6. Data elements

The `OsIocDataType` list contains all data elements, that are transmitted over this channel. At least one data element must be in the `OsIocDataType` list. If a channel has more than one element, it is called a *group* communication channel. The API of such channels has a `Group` suffix on its operation names. That is, they are named [locWriteGroup](#), [locReadGroup](#), [locSendGroup](#), [locReceiveGroup](#), instead of [locWrite](#), [locRead](#), [locSend](#), [locReceive](#).

Set the parameter `DataTypeName` to the name of the C type to be used for this data element. Make sure, that the given type name is available, by adding necessary headers to the `OsIocDataTypeIncludeHeader` list.

The `IsComplexType` and `IsArrayOrStringType` parameters exist to provide the IOC with additional information about the C type configured in `DataTypeName`. If a type is a complex type, it is passed to the [public API](#) by pointer. Composite types like C structures are complex. The opposite of a complex type is a primitive one. Primitive types are passed by value. Integer types for example are primitive types. `IsArrayOrStringType` tells the IOC, that the respective type is an array type. Arrays are always complex.

`VariableLength` enables the *variable length transmission* feature for this data element. For each such element a length parameter is added to the parameter list of the [public API](#). This parameter either specifies the number of bytes to transmit or returns the number of bytes which were transmitted.

## 7.1.7. Type headers

The IOC must know about the types, which shall be used in C-code. To achieve this, all necessary headers must be included in the `OsIocDataTypeIncludeHeader` list. These headers are then included in the IOC



header, which is generated by the [OS generator](#). In addition to the types, which are used for the [data elements](#), these headers must also define `Std_ReturnType`.

### 7.1.8. Channel initialization

Unlike queues, which are simply empty after initialization, last-is-best channels need an initialization value. For [non-group](#) channels, which do not use [variable length transmission](#), `OsIocInitValueSymbol` specifies the initialization value. Its value must be a C expression, which is assignable to a constant of the channel's data element type. For all other channels, the data elements are filled with zeros. Variable length data elements are initialized to full length.

## 7.2. Linking the IOC

Every channel has a sender state, which must be readable by the receiver. If the `OsIocSenderApiIsTrapping` configuration parameter is set to `true`, the sender state must be writeable by the [kernel](#). Otherwise it must be writeable by the sender applications themselves.

Queued channels additionally have a receiver state, which must be writeable by the receiver. The senders must be allowed to read the receiver state. There is no kernel-mode receive API. Therefore, it is never necessary, that the receiver state is writeable by the kernel.

The microkernel of EB tresos Safety OS is only allowed to write to regions, which are contained in the [kernel partition](#). For more information see the EB tresos® Safety OS User's Guide [\[SAFETYOSUSRDOC\]](#).

The kernel of EB tresos AutoCore OS is allowed to write everywhere in memory, however, you shouldn't map the sender state of a kernel-mode channel to a location that is writeable by an application.

### 7.2.1. Linking states of non-trapping APIs

The following channel states must be writeable from [applications](#):

- ▶ Sender states of channels, where `OsIocSenderApiIsTrapping` equals `false`, must be writeable from the respective sender applications.
- ▶ Receiver states of queued channels must be writeable from the respective receiver applications.

States for non-trapping APIs must be linked to regions, which are accessible by the respective applications. To support this, the [OS generator](#) emits make variables to `Ioc_gen_filelist.mak`, which contain object file lists.

The make variables adhere to the following naming scheme: `IOC_OBJS_APP(__${application_name})+`. They start with an `IOC_OBJS_APP`, which is followed by a list of application names, that is separated by double underscores `__`. The variable names appear in the order, in which they appear in the channel configurations. All objects, contained by such a variable, must be writeable by all the applications, listed in the variable name.

For example, if there is a make variable `IOC_OBJ_APP__AppA = Ioc_data_app_ch12_Send.o`, then `Ioc_data_app_ch12_Send.o` must be linked to a region that is writeable by application `AppA`. This region might be the private data region of `AppA`. If there is a variable called `IOC_OBJ_APP__AppA__AppB`, the contained objects must be linked to a region, that is writeable from applications `AppA` and `AppB`.

If an application is not allowed to send or receive from a channel, the respective states should not be writeable by it. This is necessary to establish freedom from interference. For further information, see the IOC Safety Manual [\[IOCSAFETYMAN\]](#).

## 7.2.2. Linking states of trapping APIs

The sender states of trapping channels (`OsIocSenderApiIsTrapping` equals `true`) must be linked to the kernel regions. To achieve this, either select the channels by file name or by using the `IOC_OBJS_KERN(_-${core})+ make variables`.

Kernel-mode channel states are emitted to files which start with `Ioc_data_kern_c${core}_` or `Ioc_data_kern_shared`. In case of `Ioc_data_kern_c${core}_` file names, `${core}` is the index of the core, to which the data shall be linked. Therefore, if you select `Ioc_data_kern_c${core}_*` input sections and add them to the respective kernel output section, e.g. `MK_Ram_C${core}`, it makes them writeable for the kernel. Files which start with `Ioc_data_kern_shared` contain objects, which need to be accessible from different cores. If you select `Ioc_data_kern_shared*` input sections and add them to the global kernel region, e.g. `MK_Ram`, it makes them writeable for the kernel on all cores.

Alternatively you can use the make variables which the OS generator emits to `Ioc_gen_filelist.mak`. These variables are comparable to those generated for non-trapping APIs. They adhere to the naming scheme: `IOC_OBJS_KERN(_-${core})+`. They start with an `IOC_OBJS_KERN`, which is followed by a list of core indices, that are sorted in ascending order and separated by a single underscore. All objects, contained in such a list, must be writeable by the kernel from all cores, which are referenced in the variable name.

## 7.2.3. Readable regions

If an [application](#) is listed as a sender of a trapping channel and EB tresos Safety OS is used, the application needs a valid non-empty readable region. The readable region specifies which data may be read by the [kernel](#), if the IOC is called by the respective application. The readable region should, therefore, contain all objects, which are passed by reference to the [IOC library interface](#). This includes the stack, where length and address information is stored, before it is passed to the IOC.

The begin of the readable region of a certain application is defined by the `IOC_RSA_READABLE_${application_name}` symbol, where `${application_name}` shall be the name of the respective application. The `IOC_RLA_READABLE_${application_name}` symbol must point to the next location immediately following the last readable byte.

You must choose the readable regions according to the needs of your application concerning safety, availability, and read protection. In general, if a memory location needs to be protected from read accesses by certain applications or the kernel, it shall not appear in readable regions. To address safety, consult the IOC Safety Manual [\[IOCSAFETYMAN\]](#). For the sake of availability the kernel must be able to read from each location contained in the readable region without causing an [exception](#). Otherwise, passing invalid pointers to IOC calls might cause a shut-down or kernel panic. Note, that the readable regions chosen by our demo linker script might not be suitable for your application.

## 8. IOC reference manual

### 8.1. IOC limits

The following limits on the number of IOC objects exist:

Objects	Max. number
IOC channels	4294967295
Queue entries per queued channel	65535
Data elements per group transmission	255
Bytes per data element	65535

Table 8.1. Limits on number of IOC objects.

#### NOTE



#### Number of IOC objects

The actual number of IOC objects you can use in a particular IOC system depends on environmental conditions like available memory, toolchain limitations etc. Thus, it can be significantly lower than the maximum numbers given in [Table 8.1, “Limits on number of IOC objects.”](#)

## 8.2. IOC types

[Table 8.2, “Types used in public IOC APIs”](#) describes the types, which are used in public IOC APIs.

Type name	Header	Description
<code>ioc_return_t</code>	<code>public/Ioc_basic_types.h</code>	This is the result type of all <a href="#">IOC library interface</a> functions. It fulfills the requirements specified for <code>Std_ReturnType</code> by the Specification of RTE Software <a href="#">[AUTOSAR RTE40SPEC]</a> .
<code>ioc_varlength_t</code>	<code>public/Ioc_basic_types.h</code>	This type is used for input and output parameters of <a href="#">variable length transmission</a> operations. It fulfills the requirements specified by AUTOSAR RfC 65841 for such parameters.
<code>ioc_uint32_t</code>	<code>public/Ioc_basic_types.h</code>	An unsigned 32-bit integer type.
<code>ioc_ilenlength_t</code>	<code>public/Ioc_basic_types.h</code>	This type is used to describe the length of a data element. It is used by the <a href="#">IOC library interface</a> .
<code>ioc_extinput_t</code>	<code>public/Ioc_basic_types.h</code>	This structure is used to describe data elements, which shall be sent by <a href="#">IOC_WriteExt</a> and <a href="#">IOC_SendExt</a> . It has a <code>data</code> field, which stores the address of the data element and a <code>length</code> field, which specifies the length of the element.

Table 8.2. Types used in public IOC APIs

## 8.3. IOC error code reference

[Table 8.3, “Error code table”](#) describes all error codes, which are used by EB tresos IOC and are visible to you. It contains the identifier, the value, the location where this error code is specified by AUTOSAR, and a description.

ErrorId	ErrorValue	AUTOSAR	Description
E_OK	0	OS/IOC SWS	No error occurred.
IOC_E_NOK	1	OS/IOC SWS	An error occurred, which is not specified any further.
IOC_E_LIMIT	130	OS/IOC SWS	An <a href="#">locSend[Group]</a> failed because the respective queued channel is full. For the receiver side see <a href="#">IOC_E_LOST_DATA</a> .
IOC_E_LOST_DATA	64	OS/IOC SWS	This error code is returned by the first <a href="#">locReceive[Group]</a> , which is serviced after an <a href="#">locSend[Group]</a> on the respective channel failed with <a href="#">IOC_E_LIMIT</a> .
IOC_E_NO_DATA	131	OS/IOC SWS	An <a href="#">locReceive[Group]</a> operation was applied on an empty queue.
IOC_E_LENGTH	8 (preliminary)	AUTOSAR RfC 65841	This error code is returned if the <code>length</code> argument, which was passed for a variable length data element, is invalid. Invalid means, that it either exceeds the maximum size of the respective data element or is zero. <a href="#">IOC_E_LENGTH</a> is not yet part of the AUTOSAR SWS. Its value is not yet specified and it may change in future versions.
IOC_E_SEG_FAULT	136	Based on RTE SWS	Occurs under one of the following circumstances: <ul style="list-style-type: none"> <li>▶ A call to a trapping API was refused, because of insufficient memory access permissions.</li> <li>▶ A receive operation noticed that the channel is corrupted.</li> </ul> <p>The numeric value of this error code equals <code>RTE_E_SEG_FAULT</code> <a href="#">[AUTOSARRTE42SPEC]</a>.</p>

ErrorId	ErrorValue	AUTOSAR	Description
IOC_E_ILLEGAL_CALL	3	None	Occurs under one of the following circumstances: <ul style="list-style-type: none"><li>▶ The addressed channel does not exist.</li><li>▶ The addressed channel is not suitable for that particular operation.</li><li>▶ The application is not allowed to request that particular operation on the given channel.</li></ul>
IOC_E_INTRALOCK_FAILED	7	None	The function you entered into the core-local lock configuration of the respective channel is invalid.

Table 8.3. Error code table

## 8.4. IOC API reference

This section describes the public IOC API, which resembles the API specified by AUTOSAR. It is generated by the [OS generator](#). The API is an adapter for the IOC library interface, which is described in [Section 8.5, “IOC library interface reference”](#).

How arguments are passed depends on the configuration of the respective [data elements](#).

For the description of multiple elements there are the following conventions:

- ▶ The parameters are suffixed with indices from 1 to N.
- ▶ N is the number of data elements of the channel.
- ▶ In the description the suffix x is used, to indicate, that a statement is valid for all such parameters. So x matches 1 to N.



## Name

locWrite — writes to a last-is-best channel

## Synopsis

```
#include <loc.h>

Std_ReturnType IocWrite_<IocId>[_<SenderId>] (T data);

Std_ReturnType IocWrite_<IocId>[_<SenderId>] (T const *data);

Std_ReturnType IocWrite_<IocId>[_<SenderId>] (T
    const *data, ioc_varlength_t length);
```

## Description

IocWrite writes the `data` argument to the last-is-best channel, specified by `<IocId>`.

If the given data type is configured to be [complex](#), `data` is a pointer to the respective data type. Otherwise it is passed by value.

If the given data type is configured to have [variable length](#), there is an additional `length` parameter. This `length` parameter specifies the number of bytes, which shall be written. Note, that variable length transmission enables the partial update feature, so passing `NULL` as `data` does not cause an error in that case.

If there is a suitable lock configuration, this function is reentrant and can be used concurrently by different [threads](#) as well as concurrently to any other operation except calls to [loc\\_ReInit](#) from the same writer [application](#). For more information on lock configuration, see [Section 7.1.5, “Lock configuration”](#).

## Return value

A return value of [E\\_OK](#) indicates a successful completion of the function. In case of an error, one of the following error codes is returned, see [Section 8.3, “IOC error code reference”](#):

- ▶ [IOC\\_E\\_NOK](#)
- ▶ [IOC\\_E\\_LENGTH](#)
- ▶ [IOC\\_E\\_SEG\\_FAULT](#)
- ▶ [IOC\\_E\\_ILLEGAL\\_CALL](#)
- ▶ [IOC\\_E\\_INTRALOCK\\_FAILED](#)

## Name

locWriteGroup — writes multiple data elements to a last-is-best channel

## Synopsis

```
#include <ioc.h>
```

```
Std_ReturnType IocWriteGroup_<IocId>[_<SenderId>] (T1 [const *] da-  
tal, [ioc_varlength_t length1], T2 [const *] data2, [ioc_varlength_t  
length2], ..., TN [const *] dataN, [ioc_varlength_t lengthN]);
```

## Description

IocWriteGroup writes the given data elements to the [last-is-best](#) channel, specified by <IocId>.

If data type TX is configured to be [complex](#), dataX is a pointer to the respective data type. Otherwise it is passed by value.

If data type TX is configured to have [variable length](#), there is an additional lengthX parameter. This lengthX parameter specifies the number of bytes, which shall be written.

Last-is-best [group](#) communication supports the partial update feature for complex elements. If a data argument equals NULL, the respective data element is not updated, so its previous value is preserved. If you combine partial updates with variable length transmission, make sure the length argument is always valid, even if the corresponding data argument equals NULL.

If there is a suitable lock configuration, this function is reentrant and can be used concurrently by different [threads](#) as well as concurrently to any other operation except calls to [loc\\_Relnit](#) from the same writer [application](#). For more information on lock configuration, see [Section 7.1.5, “Lock configuration”](#).

## Return value

A return value of [E\\_OK](#) indicates a successful completion of the function. In case of an error, one of the following error codes is returned, see [Section 8.3, “IOC error code reference”](#):

- ▶ [IOC\\_E\\_NOK](#)
- ▶ [IOC\\_E\\_LENGTH](#)
- ▶ [IOC\\_E\\_SEG\\_FAULT](#)
- ▶ [IOC\\_E\\_ILLEGAL\\_CALL](#)

► [IOC\\_E\\_INTRALOCK\\_FAILED](#)

## Name

`IocRead` — reads a data element from a last-is-best channel

## Synopsis

```
#include <ioc.h>

Std_ReturnType IocRead_<IocId>(T *data);

Std_ReturnType IocRead_<IocId>(T *data, ioc_varlength_t *length);
```

## Description

`IocRead` reads one data element from the [last-is-best](#) channel, specified by `<IocId>`.

If there are no errors, the last value written by [locWrite](#) or the initial value of the channel is copied to the location referenced by `data`.

If the given data type `T` is configured to have [variable length](#), there is an additional `length` parameter. If there are no errors, the number of bytes, which are written to `data`, is written to the location referenced by `length`. If there is an error, the value is undefined.

This function is reentrant and can be used concurrently by different [threads](#). It can further be used concurrently to any other operation if there is a suitable lock configuration. For more information, see [Section 7.1.5, “Lock configuration”](#).

## Return value

A return value of [E\\_OK](#) indicates a successful completion of the function. In case of an error, one of the following error codes is returned (see [Section 8.3, “IOC error code reference”](#)):

- ▶ [IOC\\_E\\_NOK](#)
- ▶ [IOC\\_E\\_ILLEGAL\\_CALL](#)
- ▶ [IOC\\_E\\_SEG\\_FAULT](#)

If a call to `IocRead` fails, the content of the variable referenced by `length` is undefined.

## Name

`IocReadGroup` — reads multiple data elements to a last-is-best channel

## Synopsis

```
#include <ioc.h>
```

```
Std_ReturnType IocReadGroup_<IocId>(T1 *data1, [ioc_varlength_t *length1], T2  
*data2, [ioc_varlength_t *length2], ..., TN *dataN, [ioc_varlength_t *lengthN]);
```

## Description

`IocReadGroup` reads multiple data elements from the [last-is-best](#) channel, specified by `<IocId>`.

The parameter `dataX` is the address, to which the data of the respective element, which is read from the channel, shall be written.

If a given data type `TX` is configured to have [variable length](#), there is an additional `lengthX` parameter. If there are no errors, the number of bytes, which are written to `dataX`, is written to the location referenced by `lengthX`. If there is an error, the value is undefined.

This function is reentrant and can be used concurrently by different [threads](#). It can further be used concurrently to any other operation, if there is a suitable lock configuration, see [Section 7.1.5, “Lock configuration”](#).

## Return value

A return value of [E\\_OK](#) indicates a successful completion of the function. In case of an error, one of the following error codes is returned, see [Section 8.3, “IOC error code reference”](#):

- ▶ [IOC\\_E\\_NOK](#)
- ▶ [IOC\\_E\\_ILLEGAL\\_CALL](#)
- ▶ [IOC\\_E\\_SEG\\_FAULT](#)

If a call to `IocReadGroup` fails, the content of the variable length output variables is undefined.

If [IOC\\_E\\_SEG\\_FAULT](#) is returned, the content of the given channel is corrupted. The output variables may contain parts of the corrupted content.

## Name

locSend — sends a data element to a queued channel

## Synopsis

```
#include <loc.h>

Std_ReturnType IocSend_<IocId>[_<SenderId>] (T data);

Std_ReturnType IocSend_<IocId>[_<SenderId>] (T const * data);

Std_ReturnType IocSend_<IocId>[_<SenderId>] (T
    const * data, ioc_varlength_t length);
```

## Description

IocSend sends the `data` argument to the [queued channel](#), specified by `<IocId>`.

If the given data type is configured to be [complex](#), `data` is a pointer to the respective data type. Otherwise it is passed by value.

If the given data type is configured to have [variable length](#), there is an additional `length` parameter. This `length` parameter specifies the number of bytes, which shall be sent.

If there is a suitable lock configuration, this function is reentrant and can be used concurrently by different [threads](#) as well as concurrently to any other operation, except sender side calls to [locEmptyQueue](#). For more information on lock configuration, see [Section 7.1.5, “Lock configuration”](#).

## Return value

A return value of [E\\_OK](#) indicates a successful completion of the function. In case of an error, one of the following error codes is returned, see [Section 8.3, “IOC error code reference”](#):

- ▶ [IOC\\_E\\_NOK](#)
- ▶ [IOC\\_E\\_LIMIT](#)
- ▶ [IOC\\_E\\_LENGTH](#)
- ▶ [IOC\\_E\\_SEG\\_FAULT](#)
- ▶ [IOC\\_E\\_ILLEGAL\\_CALL](#)
- ▶ [IOC\\_E\\_INTRALOCK\\_FAILED](#)

## Name

locSendGroup — sends multiple data elements to a queued channel

## Synopsis

```
#include <loc.h>
```

```
Std_ReturnType IocSendGroup_<IocId>[_<SenderId>] (T1 [const *] da-  
tal, [ioc_varlength_t length1], T2 [const *] data2, [ioc_varlength_t  
length2], ..., TN [const *] dataN, [ioc_varlength_t lengthN]);
```

## Description

IocSendGroup sends the given data elements to the [queued channel](#), specified by <IocId>.

If data type TX is configured to be [complex](#), dataX is a pointer to the respective data type. Otherwise it is passed by value.

If data type TX is configured to have [variable length](#), there is an additional lengthX parameter. This lengthX parameter specifies the number of bytes, which shall be written.

If there is a suitable lock configuration, this function is reentrant and can be used concurrently by different [threads](#) as well as concurrently to any other operation, except sender side calls to [locEmptyQueue](#). For more information on lock configuration, see [Section 7.1.5, “Lock configuration”](#).

## Return value

A return value of [E\\_OK](#) indicates a successful completion of the function. In case of an error, one of the following error codes is returned, see [Section 8.3, “IOC error code reference”](#):

- ▶ [IOC\\_E\\_NOK](#)
- ▶ [IOC\\_E\\_LIMIT](#)
- ▶ [IOC\\_E\\_LENGTH](#)
- ▶ [IOC\\_E\\_SEG\\_FAULT](#)
- ▶ [IOC\\_E\\_ILLEGAL\\_CALL](#)
- ▶ [IOC\\_E\\_INTRALOCK\\_FAILED](#)

## Name

`IocReceive` — receives the first data element from a queued channel

## Synopsis

```
#include <ioc.h>

Std_ReturnType IocReceive_<IocId>(T *data);

Std_ReturnType IocReceive_<IocId>(T *data, ioc_varlength_t *length);
```

## Description

`IocReceive` retrieves the first element of the [queued channel](#), specified by `<IocId>`.

The parameter `data` is the address, to which the data, which is received from the channel, shall be written.

If the given data type `T` is configured to have [variable length](#), there is an additional `length` parameter. If there are no errors, the number of bytes, which are written to `data`, is written to the location referenced by `length`. If there is an error, the value is undefined.

This function is not reentrant. It cannot be used concurrently to other calls to `IocReceive`. It can be used concurrently to any other operation except calls to [locEmptyQueue](#).

## Return value

A return value of [E\\_OK](#) indicates a successful completion of the function. [IOC\\_E\\_LOST\\_DATA](#) does not indicate an error. It indicates, that the last [locSend](#) failed with [IOC\\_E\\_LIMIT](#) on this channel. In case of an error, one of the following error codes is returned, see [Section 8.3, “IOC error code reference”](#):

- ▶ [IOC\\_E\\_NOK](#)
- ▶ [IOC\\_E\\_NO\\_DATA](#)
- ▶ [IOC\\_E\\_ILLEGAL\\_CALL](#)
- ▶ [IOC\\_E\\_SEG\\_FAULT](#)

If a call to `IocReceive` fails, the content of the variable referenced by `length` is undefined.

If variable length transmission is used, [IOC\\_E\\_SEG\\_FAULT](#) indicates that the operation failed, because the length of the next queue entry was detected to be corrupted. The corrupted entry is discarded.



## Name

`IocReceiveGroup` — receives the first group of data elements from a queued channel

## Synopsis

```
#include <ioc.h>
```

```
Std_ReturnType IocReceiveGroup_<IocId>(T1 *data1, [ioc_varlength_t *length1], T2  
*data2, [ioc_varlength_t *length2], ..., TN *dataN, [ioc_varlength_t *lengthN]);
```

## Description

`IocReceiveGroup` retrieves the first group of data elements from the [queued channel](#), specified by `<IocId>`.

The parameter `dataX` is the address, to which the data of the respective element, which is received from the channel, shall be written.

If a given data type `TX` is configured to have [variable length](#), there is an additional `lengthX` parameter. If there are no errors, the number of bytes, which are written to `dataX`, is written to the location referenced by `lengthX`. If there is an error, the value is undefined.

This function is not reentrant. It cannot be used concurrently to other calls to `IocReceiveGroup`. It can be used concurrently to any other operation except calls to [locEmptyQueue](#).

## Return value

A return value of [E\\_OK](#) indicates a successful completion of the function. [IOC\\_E\\_LOST\\_DATA](#) does not indicate an error. It indicates, that the last [locSendGroup](#) failed with [IOC\\_E\\_LIMIT](#) on this channel. In case of an error, one of the following error codes is returned, see [Section 8.3, “IOC error code reference”](#):

- ▶ [IOC\\_E\\_NOK](#)
- ▶ [IOC\\_E\\_NO\\_DATA](#)
- ▶ [IOC\\_E\\_ILLEGAL\\_CALL](#)
- ▶ [IOC\\_E\\_SEG\\_FAULT](#)

If a call to `IocReceiveGroup` fails, the content of the variable length output variables is undefined.

If [IOC\\_E\\_SEG\\_FAULT](#) is returned, the queue entry, which was read, is corrupted. The output variables may contain parts of the corrupted entry.

## Name

`IocEmptyQueue` — discards all enqueued data elements

## Synopsis

```
#include <loc.h>

Std_ReturnType IocEmptyQueue_<IocId>(void);
```

## Description

`IocEmptyQueue` discards all elements of the [queued channel](#), specified by `<IocId>`.

This function is more efficient than repeated calls to [IocReceive](#) until `IOC_E_NO_DATA` is returned.

`IocEmptyQueue` is meant to be called from the receiver side. The possibility to call it from the sender side is removed in future versions.

If `IocEmptyQueue` is called from the receiver side, it can be executed concurrently to any other operation except `IocReceive` and other calls to `IocEmptyQueue` on this channel. If you call it from the sender side, make sure, it is not executed concurrently to other operations on this channel.

## Return value

A return value of [E\\_OK](#) indicates a successful completion of the function. In case of an error, one of the following error codes is returned, see [Section 8.3, “IOC error code reference”](#):

- ▶ [IOC\\_E\\_NOK](#)
- ▶ [IOC\\_E\\_ILLEGAL\\_CALL](#)

## Name

`Ioc_ReInit` — resets a last-is-best channel to its initial state

## Synopsis

```
#include <loc.h>

Std_ReturnType Ioc_ReInit_<IocId>[_<SenderId>] ( void );
```

## Description

This function is specific to EB tresos IOC and might change in future versions.

`Ioc_ReInit` resets the [last-is-best](#) channel, specified by `<IocId>`, to its [initial state](#). In [group](#) channels all data elements are filled with zeros. Data elements, which are configured to have [variable length](#), are initialized to full length and filled with zeros. Channels, which have only a single fixed length data element, are initialized to a custom initialization value `OsIocInitValueSymbol`. If no initialization value is given, they are filled with zeros.

`Ioc_ReInit` may only be called from the writer side.

`Ioc_ReInit` must not be executed concurrently to other calls to `Ioc_ReInit` or [locWrite\[Group\]](#) of the same writer [application](#). If there is a suitable lock configuration, `Ioc_ReInit` may be executed concurrently to calls to `Ioc_ReInit` or [locWrite\[Group\]](#) of other writer applications and [locRead\[Group\]](#). For more information on lock configuration, see [Section 7.1.5, “Lock configuration”](#).

## Return value

A return value of [E\\_OK](#) indicates a successful completion of the function. In case of an error, one of the following error codes is returned, see [Section 8.3, “IOC error code reference”](#):

- ▶ [IOC\\_E\\_NOK](#)
- ▶ [IOC\\_E\\_ILLEGAL\\_CALL](#)

## 8.5. IOC library interface reference

The IOC API described in [Section 8.4, “IOC API reference”](#) is generated by the [OS generator](#). The sole purpose of these generated functions is:

- ▶ Pass the respective arguments to the library interface.
- ▶ Write variable length transmission output variables if necessary.
- ▶ Return the library interface's return value.

Since the OS generator is not reliable with respect to safety, these generated methods have to be reviewed. Hence, the library interface is described in this user documentation. For further information consult the IOC Safety Manual [\[IOCSAFETYMAN\]](#).

For the description of array parameters there are the following conventions:

- ▶  $N$  is the number of data elements of the channel.
- ▶ Array parameters have  $N$  elements.
- ▶ In the description the index  $J$  is used, to indicate, that a statement is valid for all elements of the respective array. So  $J$  matches  $0$  to  $N - 1$ .

---

**WARNING****Vendor specific API may be subject to changes**

The library interface is specific to EB tresos IOC and may change in future versions.

---

## Name

IOC\_Write — writes to a last-is-best channel

## Synopsis

```
#include <public/loc_public_api.h>

ioc_return_t IOC_Write(ioc_uint32_t channelID, void const *pointer);
```

## Description

IOC\_Write writes the data referenced by the `pointer` parameter to the [last-is-best](#) channel, specified by the `channelID` parameter. The size of the object referenced by the `pointer` parameter is deduced from the channel configuration. It implements the functionality described for [locWrite](#) without [variable length transmission](#).

## Return value

Returns the values described for [locWrite](#).

## Name

IOC\_WriteExt — writes to a last-is-best channel

## Synopsis

```
#include <public/loc_public_api.h>

ioc_return_t IOC_WriteExt(ioc_uint32_t chan-
    nelID, ioc_extinput_t const *elements);
```

## Description

IOC\_WriteExt implements the [locWriteGroup](#) functions and [locWrite](#) if [variable length transmission](#) is used. Each entry of the `elements` array describes one data element. The `elements[J].length` field specifies the number of bytes, which shall be transmitted from the respective object referenced by the `elements[J].data` field. The number of entries in `elements` must match the number of data elements in the channel configuration.

## Return value

Returns the values described for [locWrite](#) and [locWriteGroup](#).

## Name

IOC\_Read — reads from a last-is-best channel

## Synopsis

```
#include <public/loc_public_api.h>

ioc_return_t IOC_Read(ioc_uint32_t channelID, void *pointer);
```

## Description

IOC\_Read reads from the [last-is-best](#) channel, specified by the `channelID` parameter and stores it at the location referenced by the `pointer` parameter. It implements the functionality described for [locRead](#) without [variable length transmission](#).

## Return value

Returns the values described for [locRead](#).

## Name

IOC\_ReadExt — reads from a last-is-best channel

## Synopsis

```
#include <public/loc_public_api.h>

ioc_return_t IOC_ReadExt(ioc_uint32_t chan-
nelID, ioc_ilength_t *lengths, void * const *data);
```

## Description

IOC\_ReadExt implements the [locReadGroup](#) functions and [locRead](#) if [variable length transmission](#) is used. Each entries of the `lengths` and `data` arrays, which have the same index, form one data element.

The parameter `data[J]` is the address, to which the data of the `J`-th element, which is read from the channel, shall be written.

If there are no errors, the number of bytes, which are written to `data[J]`, is written to the location referenced by `lengths[J]`. IOC\_ReadExt makes sure, that the values written to `lengths` can be stored in a variable of type `ioc_varlength_t`.

## Return value

Returns the values described for [locRead](#) and [locReadGroup](#).



## Name

IOC\_Send — sends to a queued channel

## Synopsis

```
#include <public/loc_public_api.h>

ioc_return_t IOC_Send(ioc_uint32_t channelID, void const *pointer);
```

## Description

IOC\_Send enqueues the data referenced by the `pointer` parameter to the [queued channel](#), specified by the `channelID` parameter. The size of the object referenced by the `pointer` parameter is deduced from the channel configuration. It implements the functionality described for [locSend](#) without [variable length transmission](#).

## Return value

Returns the values described for [locSend](#).

## Name

IOC\_SendExt — sends to a queued channel

## Synopsis

```
#include <public/loc_public_api.h>

ioc_return_t IOC_SendExt(ioc_uint32_t chan-
    nelID, ioc_extinput_t const *elements);
```

## Description

IOC\_SendExt implements the [locSendGroup](#) functions and [locSend](#) if [variable length transmission](#) is used. Each entry of the `elements` array describes one data element. The `elements[J].length` field specifies the number of bytes, which shall be transmitted from the respective object referenced by the `elements[J].data` field. The number of entries in `elements` must match the number of data elements in the channel configuration.

## Return value

Returns the values described for [locSend](#) and [locSendGroup](#).

## Name

IOC\_Receive — receives from a queue

## Synopsis

```
#include <public/loc_public_api.h>

ioc_return_t IOC_Receive(ioc_uint32_t channelID, void *pointer);
```

## Description

IOC\_Receive dequeues a data elements from the [queued channel](#), specified by the `channelID` parameter and stores it at the location referenced by the `pointer` parameter. It implements the functionality described for [locReceive](#) without [variable length transmission](#).

## Return value

Returns the values described for [locReceive](#).

## Name

IOC\_ReceiveExt — receives from a queue

## Synopsis

```
#include <public/loc_public_api.h>

ioc_return_t IOC_ReceiveExt(ioc_uint32_t chan-
nelID, ioc_ilenlength_t *lengths, void * const *data);
```

## Description

IOC\_ReceiveExt implements the [locReceiveGroup](#) functions and [locReceive](#) if [variable length transmission](#) is used. Each entries of the `lengths` and `data` arrays, which have the same index, form one data element.

The parameter `data[J]` is the address, to which the data of the  $J$ -th element, which is received from the channel, shall be written.

If there are no errors, the number of bytes, which are written to `data[J]`, is written to the location referenced by `lengths[J]`. IOC\_ReceiveExt makes sure, that the values written to `lengths` can be stored in a variable of type `ioc_varlength_t`.

## Return value

Returns the values described for [locReceive](#) and [locReceiveGroup](#).

## Name

IOC\_EmptyQueue — empties a queue

## Synopsis

```
#include <public/loc_public_api.h>

ioc_return_t IOC_EmptyQueue(ioc_uint32_t channelID);
```

## Description

IOC\_EmptyQueue() empties the [queued channel](#) specified by the `channelID` parameter. It implements the [locEmptyQueue](#) functions.

## Return value

A return value of `E_OK` indicates a successful completion of the function. Any other return value indicates an error code as described in [locEmptyQueue](#).

## Name

IOC\_ReInit — resets a last-is-best channel

## Synopsis

```
#include <public/loc_public_api.h>

ioc_return_t IOC_ReInit(ioc_uint32_t channelID);
```

## Description

IOC\_ReInit resets the [last-is-best](#) channel, specified by the `channelID` parameter, to its initial value. It implements the [loc\\_ReInit](#) functions.

## Return value

Returns the values described for [loc\\_ReInit](#).

## 8.6. Deviations from the AUTOSAR standard

EB tresos IOC mainly deviates from AUTOSAR with respect to error codes. For further information on error codes, see [Section 8.3, “IOC error code reference”](#).

<p><b>Deviation.Autosar.ErrorCode.IntraLockFailed</b></p> <p>EB tresos IOC has configurable locks. If EB tresos IOC fails to take the intra-core lock of a channel, it returns IOC_E_INTRALOCK_FAILED. That is to provide more information to you. It means that you entered an invalid function into the core-local lock configuration of the respective channel.</p> <p>Deviates: <i>Autosar.SWS_Os_00718, Autosar.SWS_Os_00728</i></p>
<p><b>Deviation.Autosar.ErrorCode.IllegalCall</b></p> <p>To provide more information to you, EB tresos IOC returns IOC_E_ILLEGAL_CALL instead of IOC_E_NOK if a request is invalid.</p> <p>Deviates: <i>Autosar.SWS_Os_00718, Autosar.SWS_Os_00728, Autosar.SWS_Os_00738, Autosar.SWS_Os_00746, Autosar.SWS_Os_00754</i></p>
<p><b>Deviation.Autosar.ErrorCode.SegFault</b></p> <p>To provide more information to you, EB tresos IOC returns IOC_E_SEG_FAULT instead of IOC_E_NOK if a caller has insufficient memory access permissions or if a channel has been corrupted. This is the result of checks, which can be necessary to fulfill safety, availability, and read-protection requirements.</p> <p>Deviates: <i>Autosar.SWS_Os_00718, Autosar.SWS_Os_00728, Autosar.SWS_Os_00738, Autosar.SWS_Os_00746</i></p>
<p><b>Deviation.Autosar.SenderEmptyQueue</b></p> <p>The AUTOSAR standard is unclear about the possible callers of locEmptyQueue. EB tresos IOC expects it to be called from the receiver side, since it does not have a SenderId and it is comparable to calling locReceive until the queued channel is empty. If AUTOSAR intended to allow to call locEmptyQueue from sender side, EB tresos IOC deviates from AUTOSAR by the following: Sender side calls to locEmptyQueue must not run concurrently to any other operation which modifies the channel's state.</p> <p>Deviates: <i>Autosar.SWS_Os_00754</i></p>
<p><b>Deviation.Autosar.Notification</b></p> <p>EB tresos IOC doesn't implement a receiver notification mechanism.</p> <p>Deviates: <i>Autosar.SWS_Os_00757, Autosar.SWS_Os_00758, Autosar.SWS_Os_00759, Autosar.SWS_Os_00760, Autosar.SWS_Os_00761</i></p>

## 8.7. Limitations

EB tresos IOC doesn't implement the notification mechanism described in section "7.10.3.2 Notification" of [\[AUTOSAR42SPEC\]](#). However, you can call operating system services like `ActivateTask` or `SetEvent`, to notify the receiver, after a send operation returns successfully.

For environmental limits, see [Section 8.1, "IOC limits"](#). For more information about deviations, see [Section 8.6, "Deviations from the AUTOSAR standard"](#).



## 9. Supplementary information

This chapter provides supplementary information about target hardware and operating system, including the need of system calls for certain features.

---

**NOTE****Use of features**

One argument to use [non-trapping](#) APIs is performance. Using features, which require [system calls](#), affects the performance of such an API.

---

### 9.1. Operating system

When you use EB tresos Safety OS and the [lock configuration](#) parameter `OsIocIntraCoreLockType` is set to `INTERRUPT_LOCK`, sender-side API calls will always cause a [system call](#). EB tresos AutoCore OS offers the `OsFastInterruptLocking` feature. If this feature is enabled, no system calls are required, but the applications which use the respective channels must be `TRUSTED`. Otherwise system calls are required.

### 9.2. CPU family specific information

The following sections describe hardware specific properties and requirements concerning spinlocks and caches.

#### 9.2.1. ARM

##### 9.2.1.1. User-mode spinlocks

User-mode spinlocks are not supported for ARM. This means that the usage of [cross-core locks](#) causes system calls.

##### 9.2.1.2. Cache settings

The cache on ARM has to be configured in a way that guarantees cache coherency across cores.

## 9.2.2. ARM64

### 9.2.2.1. User-mode spinlocks

User-mode spinlocks are not supported for ARM64. This means, that the usage of [cross-core locks](#) causes system calls.

### 9.2.2.2. Cache settings

The cache on ARM64 must be configured in a way that guarantees cache coherency across cores.

## 9.2.3. Cortex-M

### 9.2.3.1. User-mode spinlocks

User-mode spinlocks are not supported for Cortex-M. This means that the usage of [cross-core locks](#) causes system calls.

### 9.2.3.2. Cache settings

Cross-core communication is currently not supported for any of the supported Cortex-M derivatives. Apart from that no special cache settings are required.

## 9.2.4. Power Architecture

### 9.2.4.1. User-mode spinlocks

User-mode spinlocks are not supported for Power Architecture. This means, that the usage of [cross-core locks](#) causes system calls.

### 9.2.4.2. Cache settings

None of the supported Power Architecture derivatives can guarantee cache coherency across cores. Therefore, you must disable the caches for RAM locations, where IOC channel states are placed, which are used for cross-core communication.

## 9.2.5. RH850

### 9.2.5.1. User-mode spinlocks

User-mode spinlocks are not supported for RH850. This means, that the usage of [cross-core locks](#) causes system calls.

### 9.2.5.2. Cache settings

None of the supported RH850 derivatives has data caches, which operate on RAM locations. Therefore, there are no further requirements concerning caches and EB tresos IOC.

## 9.2.6. TriCore

### 9.2.6.1. User-mode spinlocks

User-mode spinlocks are supported for TriCore. That means that no [system calls](#) are necessary to use [cross-core locks](#).

User-mode spinlocks are not noticed by the operating system's bookkeeping mechanisms. This means, if a [thread](#) is terminated asynchronously while it performs a send/write operation, the sender side of the used channel might stay locked. An asynchronous termination might happen if one [application](#) terminates another one using `TerminateApplication`. Furthermore, `ErrorHook` or `ProtectionHook` might decide to terminate a thread, e.g. due to execution budget monitoring. You can unlock the sender side of such a channel by restarting the operating system.

### 9.2.6.2. Cache settings

If a variable is shared between different cores, you must disable the caches for this memory location. This is a limitation imposed by the AURIX Safety Manual - AP32224 [\[AURIX\\_SM11\]](#), see `[SM_AURIX_16]`. Therefore,

the caches must be disabled for RAM locations, where IOC channel states are placed, which are used for cross-core communication.

## **9.2.7. Windows and Linux on IA32 or AMD64**

### **9.2.7.1. User-mode spinlocks**

User-mode spinlocks are not supported on Windows and Linux. However, there's no kernel mode for these targets anyway.

### **9.2.7.2. Cache settings**

No special cache settings are required, just operate your Windows/Linux system as normal.

# Appendix A. Document configuration information

This document has been created by the DocBook engine using the source files and revisions listed below. All paths are relative to the directory [http://subversion.ebgroup.elektrobit.com/svn/autosar/asc\\_loc/trunk/doc/public/userguides](http://subversion.ebgroup.elektrobit.com/svn/autosar/asc_loc/trunk/doc/public/userguides).

Filename	Revision
../../../../project/fragments/diagrams/mempart.svg	2
../../../../project/fragments/glossary/Glossary.xml	1360
../fragments/About_This_Documentation/About_This_Documentation.xml	1074
../fragments/entities/IOC.ent.m4	792
../fragments/Quality_Statement.xml	1074
../fragments/Safe_And_Correct_Use.xml	1243
ApiRef.xml	1258
BackgroundInformation.xml	1235
BeginHere.xml	791
ConfigRef.xml	711
Deviations.xml	1316
ErrorRef.xml	1258
History.xml	1382
HowToUse.xml	1320
IOC_users_guide.xml	792
Limitations.xml	1316
Limits.xml	1254
Ref_IOC_EmptyQueue.xml	1255
Ref_IOC_Read.xml	795
Ref_IOC_ReadExt.xml	1314
Ref_IOC_Receive.xml	1255
Ref_IOC_ReceiveExt.xml	1314
Ref_Ioc_ReInit_public.xml	1258
Ref_IOC_ReInit.xml	1258
Ref_IOC_Send.xml	1255

Filename	Revision
Ref_IOC_SendExt.xml	1314
Ref_IOC_Write.xml	795
Ref_IOC_WriteExt.xml	1314
Ref_IocEmptyQueue.xml	1255
Ref_IocRead.xml	1243
Ref_IocReadGroup.xml	1313
Ref_IocReceive.xml	1313
Ref_IocReceiveGroup.xml	1313
Ref_IocSend.xml	1313
Ref_IocSendGroup.xml	1313
Ref_IocWrite.xml	1313
Ref_IocWriteGroup.xml	1313
ReferenceManual.xml	711
RefTypes.xml	1251
SafeApiRef.xml	1314
Supplement_ARM.xml	1235
Supplement_ARM64.xml	795
Supplement_CORTEXM.xml	1235
Supplement_OS.xml	1235
Supplement_PA.xml	1251
Supplement_RH850.xml	795
Supplement_TRICORE.xml	1251
Supplement_WINLIN.xml	1235
Supplement.xml	1235
UGLinks.ent	1255

# Glossary

exception	Processor-internal event (e.g. division by zero). The current program flow is interrupted and continued at the address which is associated with the specific exception. Usually accompanied by a switch to <a href="#">privileged mode</a> . Exceptions are commonly used to indicate a hardware detected error, e.g. a detected memory protection violation.
kernel	The kernel is the privileged part of the operating system. It is responsible for dispatching and scheduling of <a href="#">threads</a> . It receives system calls (like Activate-Task) and either handles them directly or passes them to a server. If code is executed in the kernel's context, it has the same privileges and access rights as the dispatcher, scheduler or system call handlers.
kernel partition	From the IOC's point of view, the kernel partition is the set of <i>memory regions</i> which may be written by the <a href="#">kernel</a> , but not by applications.
memory region	A memory region (or memory protection region) is a contiguous range of memory addresses which has certain access rights attached to it. These access rights are enforced by hardware. The number of memory regions that can simultaneously be managed by the hardware is usually limited.
non-privileged mode	CPU mode with restricted access rights. Opposite of <a href="#">privileged mode</a> .
OS-Application	An OS-Application is a group of OS-objects, i.e. tasks, ISRs, counters, alarms, etc. All objects of an OS-Application belong to one entity and can share data among each other and have memory areas with common write access. For more information consult the documentation of the respective operating system
OS generator	The OS generator creates C-source code from the OS configuration in EB tresos Studio.
privileged mode	CPU mode with elevated rights. In this mode, it is allowed to alter the memory and register protection. Opposite of <a href="#">non-privileged mode</a> .
system call	A system call is how an OS-object requests a service from an operating system's kernel (with hardware support). A system call separates OS-Applications and the operating system via a context switch. The hardware ensures that the control flow continues in the kernel and switches to the <a href="#">privileged mode</a> . The IOC configuration refers to system calls as <a href="#">traps</a> .
thread	From the IOC's point of view the term "thread" is a generalization for tasks and category 2 ISRs. Only tasks and category 2 ISRs are allowed to request IOC

services. For further information about threads (or tasks and ISRs) consult the user documentation of the respective operating system.

#### trap

The IOC uses "trap" as a synonym for [system call](#). I.e. IOC APIs which are "trapping" perform a system call, while "non-trapping" APIs don't. Note, that in general "trap" can also be used as a synonym for [exception](#).



# Bibliography

- [ASCOS\_USER-GUIDE]** *EB tresos AutoCore OS Documentation*
- [AURIX\_SM11]** Infineon: *AURIX Safety Manual - AP32224*, V1.1, September 2014
- [AU-TOSAROS40SPEC]** AUTOSAR: *Specification of Operating System*, Version 4.0.0, revision 3  
[http://www.autosar.org/fileadmin/files/releases/4-0/software-architecture/system-services/standard/AUTOSAR\\_SWS\\_OS.pdf](http://www.autosar.org/fileadmin/files/releases/4-0/software-architecture/system-services/standard/AUTOSAR_SWS_OS.pdf)
- [AU-TOSAROS42SPEC]** AUTOSAR: *Specification of Operating System*, Version 4.2.2  
[http://www.autosar.org/fileadmin/files/releases/4-2/software-architecture/system-services/standard/AUTOSAR\\_SWS\\_OS.pdf](http://www.autosar.org/fileadmin/files/releases/4-2/software-architecture/system-services/standard/AUTOSAR_SWS_OS.pdf)
- [AUTOSAR-RTE40SPEC]** AUTOSAR: *Specification of RTE Software*, Version 4.0.0, revision 2  
[http://www.autosar.org/download/R4.0/AUTOSAR\\_SWS\\_RTE.pdf](http://www.autosar.org/download/R4.0/AUTOSAR_SWS_RTE.pdf)
- [AUTOSAR-RTE42SPEC]** AUTOSAR: *Specification of RTE Software*, Version 4.2.2  
[http://www.autosar.org/fileadmin/files/releases/4-2/software-architecture/rte/standard/AUTOSAR\\_SWS\\_RTE.pdf](http://www.autosar.org/fileadmin/files/releases/4-2/software-architecture/rte/standard/AUTOSAR_SWS_RTE.pdf)
- [IOCSAFETYMAN]** *IOC Safety Manual*  
[8.6\\_IOC\\_safety\\_manual.pdf](#)
- [IOCUGRR1]** *Review Report 1 of the IOC User's Guide*
- [ISO26262\_1ST]** *INTERNATIONAL STANDARD ISO 26262: Road vehicles - Functional safety*, 2011
- [SAFETY-OSUSRDOC]** *EB tresos® Safety OS User's Guide*  
[8.1\\_Safety\\_OS\\_documentation\\_<TARGET>.pdf](#)