Legal information

Warning notice system
This manual contains notices you have to observe in order to ensure your personal safety, as well as to prevent
damage to property. The notices referring to your personal safety are highlighted in the manual by a safety alert
symbol, notices referring only to property damage have no safety alert symbol. These notices shown below are
graded according to the degree of danger.

**DANGER**
indicates that death or severe personal injury will result if proper precautions are not taken.

**WARNING**
indicates that death or severe personal injury may result if proper precautions are not taken.

**CAUTION**
with a safety alert symbol, indicates that minor personal injury can result if proper precautions are not taken.

**CAUTION**
without a safety alert symbol, indicates that property damage can result if proper precautions are not taken.

**NOTICE**
indicates that an unintended result or situation can occur if the corresponding information is not taken into
account.

If more than one degree of danger is present, the warning notice representing the highest degree of danger will
be used. A notice warning of injury to persons with a safety alert symbol may also include a warning relating to
property damage.

Qualified Personnel
The device/system may only be set up and used in conjunction with this documentation. Commissioning and
operation of a device/system may only be performed by qualified personnel. Within the context of the safety notes
in this documentation qualified persons are defined as persons who are authorized to commission, ground and
label devices, systems and circuits in accordance with established safety practices and standards.

Proper use of Siemens products
Note the following:

**WARNING**
Siemens products may only be used for the applications described in the catalog and in the relevant technical
documentation. If products and components from other manufacturers are used, these must be recommended
or approved by Siemens. Proper transport, storage, installation, assembly, commissioning, operation and
maintenance are required to ensure that the products operate safely and without any problems. The permissible
ambient conditions must be adhered to. The information in the relevant documentation must be observed.

Trademarks
All names identified by ® are registered trademarks of the Siemens AG. The remaining trademarks in this
publication may be trademarks whose use by third parties for their own purposes could violate the rights of the
owner.

Disclaimer of Liability
We have reviewed the contents of this publication to ensure consistency with the hardware and software
described. Since variance cannot be precluded entirely, we cannot guarantee full consistency. However, the
information in this publication is reviewed regularly and any necessary corrections are included in subsequent
editions.
Introduction

Conventions

- **EC31**
  
  In the documentation, the term *embedded controller* or *device* is also used to designate the EC31 product.

- **S7 Modular Embedded Controller**
  
  The entire *S7 modular embedded controller* system - consisting of an embedded controller with PC upgrades, signal modules and expansion modules is abbreviated to *S7-mEC*.

Purpose of this document

This document contains information that you will need to program with the software development kit (SDK) for the *S7-mEC* system. It is intended for use by programmers who commission the device themselves.

Scope

This document applies to all supplied variants of the *S7-mEC*, and describes the EC31, product release V1.1 or higher.

Basic knowledge required

The *S7-mEC* system must only be used by qualified personnel. Knowledge of the following is considered essential:

- Set-up guidelines for SIMATIC S7-300
- PC skills
- C/C++ programming skills
- Windows XP/XP Embedded operating systems

Position in the Information Landscape

For further information on using the hardware, refer to the relevant equipment manuals.
Introduction
# Table of contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Introduction</strong></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td><strong>1 Description</strong></td>
<td>1.1 Software Development Kit</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>1.2 Sample program</td>
<td>8</td>
</tr>
<tr>
<td><strong>2 Programming</strong></td>
<td>2.1 Creating a program</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>2.2 Program structure</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>2.3 Addressing signal modules</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>2.4 Callback functions</td>
<td>15</td>
</tr>
<tr>
<td><strong>3 Functions</strong></td>
<td>3.1 Overview</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>3.2 Central I/O functions</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>3.2.1 Basic functions</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>3.2.1.1 eccio_initialize</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>3.2.1.2 eccio_deinitialize</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>3.2.1.3 eccio_output_control</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>3.2.1.4 eccio_check_bus</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>3.2.1.5 eccio_ack_alarm</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>3.2.1.6 Callback alarm_notification</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>3.2.2 Reading and writing data</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>3.2.2.1 eccio_read_data</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>3.2.2.2 eccio_write_data</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>3.2.3 Assigning parameters to signal modules</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>3.2.3.1 Basic principles - parameter assignments</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>3.2.3.2 eccio_read_dataset</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>3.2.3.3 eccio_write_dataset</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td>3.2.3.4 eccio_def_par_write_single</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>3.2.3.5 eccio_def_par_write_broadcast</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>3.2.4 LED and RUN/STOP switch functions</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>3.2.4.1 ecledrs_initialize</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>3.2.4.2 ecledrs_deinitialize</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>3.2.4.3 ecledrs_write</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>3.2.4.4 ecledrs_registerswitchchangeckb</td>
<td>38</td>
</tr>
<tr>
<td></td>
<td>3.2.4.5 ecledrs_deregisterswitchchangeckb</td>
<td>39</td>
</tr>
<tr>
<td></td>
<td>3.2.4.6 Callback switch_change_notification</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>3.3 Persistence functions</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td>3.3.1 ecpers_initialize</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td>3.3.2 ecpers_deinitialize</td>
<td>42</td>
</tr>
<tr>
<td></td>
<td>3.3.3 ecpers_readblock</td>
<td>43</td>
</tr>
<tr>
<td></td>
<td>3.3.4 ecpers_writeblock</td>
<td>44</td>
</tr>
<tr>
<td></td>
<td>3.3.5 Callback power_fail_notification</td>
<td>45</td>
</tr>
</tbody>
</table>
# Table of contents

<table>
<thead>
<tr>
<th>Appendix</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.1</td>
<td>47</td>
</tr>
<tr>
<td>A.1.1</td>
<td>47</td>
</tr>
<tr>
<td>A.1.2</td>
<td>47</td>
</tr>
<tr>
<td>A.1.3</td>
<td>49</td>
</tr>
<tr>
<td>A.1.4</td>
<td>50</td>
</tr>
<tr>
<td>A.2</td>
<td>51</td>
</tr>
<tr>
<td>A.3</td>
<td>54</td>
</tr>
</tbody>
</table>

Index: 55
1.1 Software Development Kit

Software Development Kit (SDK)

An SDK contains programs and definitions for specific software, and provides functions for creating C user programs. The SDK programming interface for the embedded controller thus allows access to the signal modules on the backplane bus (central I/O), EC31 displays and operator controls, the storage locations for retentive data (persistence), and notifications for interrupts, RUN/STOP switch changes, and power failures.

Functions

The SDK subdivides the components for the functions into the following groups:

- ECCIO ... - Components for central I/O
- ECLEDRS ... - Components for LED and RUN/STOP switches
- ECPERS ... - Components for persistence

Schematic diagram
1.2 Sample program

Sample program included as standard

The EC31 is supplied as standard with a complete program that calls up all the SDK functions for an S7-mEC structure with a digital input module, and an analog output module by way of example. This example illustrates the basic program structure, and the individual phases of a normal application. You will find the source code both preinstalled on the EC31, and on the "S7-mEC Software & Documentation" CD-ROM provided.

Opening the sample program

On the EC31, you can open the folder containing the Visual Studio sample program project from the Windows task bar using the following command:

Start > All Programs > SIMATIC > S7-mEC > EC31 > SDK Software > ecExample
2

2.1 Creating a program

Requirements

- The C/C++ programming environment, such as Visual Studio 2005, is installed on the embedded controller EC31, or on an external engineering PC.
- You have access to the SDK header files. These are contained on the EC31, and on the Software & Documentation CD-ROM under ...\SDK\inc\.
- You will find the DLL files on the EC31 under C:\Windows\System32\.

**Note**

**Remote debugging**

Remote debugging (Microsoft Visual Studio 2005 or higher) allows you to start the application directly on EC31. To do this you have to install the monitor (Msvsmon.exe), which is supplied with every Visual Studio, on EC31. You can find the required project settings and releases on the Internet at Microsoft Software Development Network (MSDN), under "Remote Debugging".

**SDK header files and libraries**

<table>
<thead>
<tr>
<th>Needed for ...</th>
<th>Header file</th>
<th>DLL file</th>
</tr>
</thead>
<tbody>
<tr>
<td>Central I/O functions</td>
<td>eccio.h</td>
<td>eccio.dll</td>
</tr>
<tr>
<td>for accessing the signal modules via the backplane bus</td>
<td></td>
<td></td>
</tr>
<tr>
<td>for assigning parameters to signal modules</td>
<td></td>
<td></td>
</tr>
<tr>
<td>for detecting interrupts using callback functions, and responding to them</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LED and RUN/STOP switch functions</td>
<td>ecledrs.h</td>
<td>ecledrs.dll</td>
</tr>
<tr>
<td>for activating LEDs on the EC31</td>
<td></td>
<td></td>
</tr>
<tr>
<td>for detecting changes in the status of the RUN/STOP switch on the EC31</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Persistence functions</td>
<td>ecpers.h</td>
<td>ecpers.dll</td>
</tr>
<tr>
<td>for saving retentive data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>for detecting a power failure (POWER OFF)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Note**

**Header files on the EC31**

On the EC31, you can open the folder containing the header files for SDK from the Windows task bar using the following command:

`Start > (All Programs) > SIMATIC > S7-mEC > EC31 > SDK Software > Header Files`
2.1 Creating a program

Creating a program

1. Incorporate the header files and the DLL files that you will need for your user program into your project.

2. Use the SDK functions to program the user program.

3. Compile your project.

**Result:** The “*.exe” user program can be transferred to the EC31.

**Note**
Under Windows XP Embedded, other programs or connected devices can have a negative impact on the time taken to access the backplane bus.
2.2  Program structure

Typical structure

When it runs, a user program is typically divided into 3 phases:

- Initialization phase
- Productive mode
- End phase

Rules

Note the following points when you program:

- The functions of the initialization phase activate the necessary components from the SDK via the user program. The functions of the end phase end these components. The functions must be used in the program.
- The functions for productive mode are optional.

Central I/O functions

<table>
<thead>
<tr>
<th>Phase</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialization</td>
<td>eccio_initialize</td>
</tr>
<tr>
<td></td>
<td>eccio_output_control</td>
</tr>
<tr>
<td></td>
<td>eccio_def_par_write_single</td>
</tr>
<tr>
<td></td>
<td>eccio_def_par_write_broadcast</td>
</tr>
<tr>
<td>Productive mode</td>
<td>eccio_check_bus</td>
</tr>
<tr>
<td></td>
<td>eccio_ack_alarm</td>
</tr>
<tr>
<td></td>
<td>eccio_write_dataset</td>
</tr>
<tr>
<td></td>
<td>eccio_read_dataset</td>
</tr>
<tr>
<td></td>
<td>eccio_read_data</td>
</tr>
<tr>
<td></td>
<td>eccio_write_data</td>
</tr>
<tr>
<td>Ending</td>
<td>eccio_deinitialize</td>
</tr>
</tbody>
</table>

LED and RUN/STOP switch functions

<table>
<thead>
<tr>
<th>Phase</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialization</td>
<td>ecledrs_initialize</td>
</tr>
<tr>
<td></td>
<td>ecledrs_registerswitchchangeck</td>
</tr>
<tr>
<td>Productive mode</td>
<td>ecledrs_write</td>
</tr>
<tr>
<td>Ending</td>
<td>ecledrs_deregisterswitchchangeck</td>
</tr>
<tr>
<td></td>
<td>ecledrs_deinitialize</td>
</tr>
</tbody>
</table>

Persistence functions

<table>
<thead>
<tr>
<th>Phase</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialization</td>
<td>ecpers_initialize</td>
</tr>
<tr>
<td>Productive mode</td>
<td>ecpers_readblock</td>
</tr>
<tr>
<td></td>
<td>ecpers_writeblock</td>
</tr>
<tr>
<td>Ending</td>
<td>ecpers_deinitialize</td>
</tr>
</tbody>
</table>
2.2 Program structure

Flow diagram for a user program

![Flow diagram for a user program]

Figure 2-1   Program flow diagram

**Note**

**Ending the user program**

Always use the eccio_deinitialize call to end the user program.
2.3 Addressing signal modules

Addressing signal modules via GeoAddr

Central I/O functions require the signal modules concerned to be addressed. The signal modules are addressed in the relevant functions with the GeoAddr data type using the Rack and Slot parameters.

The following diagram shows the maximum configuration on the central I/O with the relevant numbers for Rack and Slot. Slot 3 on each rack is reserved for the interface module (IM), so counting for the signal modules starts from 4.

![Diagram showing maximum configuration on central I/O](image)

Figure 2-2 Addressing of signal modules
Examples - addressing signal modules via the GeoAddr parameter
The signal modules marked in the picture are addressed as follows:

- Addressing for the digital input module DI
  ```
  GeoAddr digital_input;
  digital_input.rack = 0;
  digital_input.slot = 7;
  ```

- Addressing for the analog input module AI
  ```
  GeoAddr analog_input;
  analog_input.rack = 1;
  analog_input.slot = 5;
  ```

See also

GeoAddr (Page 47)
2.4 Callback functions

How it works

Callback functions are specified by the user program. A callback function may be assigned any name.

A callback event is an asynchronous event that is called by the EC-CIO interface. It interrupts the flow of the user program, and starts the callback function in a separate thread.

Callback functions

The following callback functions may be defined in the SDK for S7-mEC:

<table>
<thead>
<tr>
<th>Callback function</th>
<th>Callback event</th>
<th>Registered by ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>alarm_notification</td>
<td>Hardware interrupt / diagnostic error interrupt at a signal module</td>
<td>eccio_initialize()</td>
</tr>
<tr>
<td>switch_change_notification</td>
<td>Change in the status of the RUN/STOP switch on the EC31</td>
<td>ecledrs_registerswitchchangeckb()</td>
</tr>
<tr>
<td>power_fail_notification</td>
<td>Power failure on the EC31</td>
<td>ecpers_initialize()</td>
</tr>
</tbody>
</table>

Runtime coordination for callbacks

A callback function can interrupt the user program at any time. Callback functions for different events can also interrupt one another. A callback function must therefore be designed to run multiple times, including simultaneously (reentrant) because it can be called from different threads. In practice, this means that the writing and reading of shared tags must be protected by synchronization mechanisms.

Avoid waits in callback functions, particularly when entering Critical Sections. A further call to a callback function would be blocked by the same callback event. Instead, you should keep your stored data as separate as possible.

A separate callback function can be registered for each callback event. It is also possible to combine multiple callback events in a single callback function, however.

Sample declarations for user-defined callback functions

```c
void eccioCB_AlarmNotification (AlarmInfo* alarm_data);
void ecledrsCB_SwitchChangeNotification (unsigned char state);
void ecpersCB_PowerFailNotification (void);
```

See also

- eccio_initialize (Page 19)
- ecledrs_registerswitchchangeckb (Page 38)
- ecpers_initialize (Page 41)
Programming

2.4 Callback functions
Functions

3.1 Overview

The SDK provides components for the following functions:

- Central I/O
- LED and RUN/STOP switch
- Persistence

Central I/O functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Basic functions</strong></td>
<td></td>
</tr>
<tr>
<td>eccio_initialize</td>
<td>The user program uses this function to register the embedded controller on the backplane bus.</td>
</tr>
<tr>
<td>eccio_deinitialize</td>
<td>The user program uses this function to deregister the embedded controller on the backplane bus.</td>
</tr>
<tr>
<td>eccio_output_control</td>
<td>This function activates / deactivates the outputs of the signal modules.</td>
</tr>
<tr>
<td>eccio_check_bus</td>
<td>This function compares the current configuration on the backplane bus with the list of stations that were identified using the eccio_initialize function.</td>
</tr>
<tr>
<td>eccio_ack_alarm</td>
<td>This function acknowledges interrupts from signal modules.</td>
</tr>
<tr>
<td><strong>Reading and writing data</strong></td>
<td></td>
</tr>
<tr>
<td>eccio_read_data</td>
<td>This function reads 1, 2 or 4 bytes from an input module.</td>
</tr>
<tr>
<td>eccio_write_data</td>
<td>This function writes 1, 2 or 4 bytes to an output module.</td>
</tr>
<tr>
<td><strong>Assigning parameters to signal modules</strong></td>
<td></td>
</tr>
<tr>
<td>eccio_read_dataset</td>
<td>This function reads data blocks up to 240 bytes long from a signal module.</td>
</tr>
<tr>
<td>eccio_write_dataset</td>
<td>This function writes data blocks up to 240 bytes long to a signal module.</td>
</tr>
<tr>
<td></td>
<td>- Assigning parameters to signal modules</td>
</tr>
<tr>
<td></td>
<td>- Setting the type of measurement and measuring ranges (voltage and current)</td>
</tr>
<tr>
<td></td>
<td>- Enabling / disabling interrupts</td>
</tr>
<tr>
<td>eccio_def_par_write_single</td>
<td>This function transfers the parameter assignment status to one signal module.</td>
</tr>
<tr>
<td>eccio_def_par_write_broadcast</td>
<td>This function transfers the parameter assignment status to all signal modules on the backplane bus.</td>
</tr>
</tbody>
</table>
### LED and RUN/STOP switch functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ecledrs_initialize</td>
<td>This function initializes the LED and RUN/STOP switch functions.</td>
</tr>
<tr>
<td>ecledrs_deinitialize</td>
<td>This function ends the LED and RUN/STOP switch functions.</td>
</tr>
<tr>
<td>ecledrs_write</td>
<td>This function is used to control LEDs on the EC31.</td>
</tr>
<tr>
<td>ecledrs_registerswitchchangeckb</td>
<td>This function activates the status monitoring for the RUN/STOP switch on the EC31.</td>
</tr>
<tr>
<td>ecledrs_deregisterswitchchangeckb</td>
<td>This function deregisters the status monitoring for the RUN/STOP switch.</td>
</tr>
</tbody>
</table>

### Persistence functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ecpers_initialize</td>
<td>This function initializes the persistence functions.</td>
</tr>
<tr>
<td>ecpers_deinitialize</td>
<td>This function ends the persistence functions.</td>
</tr>
<tr>
<td>ecpers_readblock</td>
<td>This function reads data from a retentive memory.</td>
</tr>
<tr>
<td>ecpers_writeblock</td>
<td>This function writes data to a retentive memory.</td>
</tr>
</tbody>
</table>
3.2 Central I/O functions

3.2.1 Basic functions

3.2.1.1 eccio_initialize

Description
This function initializes the backplane bus, and registers a user-defined callback function that is called in response to an interrupt at a signal module.

Once the call has been processed successfully, it sends the list of stations to all signal modules plugged into the backplane bus. The DC5V LED on the EC31 lights up to indicate that the control voltage is present at the connected signal modules.

Requirement:
- The signal modules are supplied with voltage.
- The ident and alarm_notification parameters were successfully initialized.

Note
Call the function in the user program before the other central I/O functions.

If the function is called a second time, the backplane bus is reset, and the stations are identified once again.

Syntax
```c
unsigned short eccio_initialize(
    BusEnum* ident,
    FP_EC_CIO_ALARM_NOTIFICATION alarm_notification,
    unsigned short config_flags )
```
3.2 Central I/O functions

Parameter

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ident</td>
<td>out</td>
<td>List of stations containing all the signal modules on the backplane bus.</td>
<td>BusEnum*</td>
</tr>
<tr>
<td>alarm_notification</td>
<td>in</td>
<td>Pointer to a user function that is called in the event of an interrupt.</td>
<td>-</td>
</tr>
<tr>
<td>config_flags</td>
<td>in</td>
<td>Permitted values: 0: EC_INITFLAGS_DO_DISABLE_OUTPUT_ON_STOP Safe backplane bus configuration enabled. This means that the backplane bus is dependent on the position of the RUN/STOP switch. 1: EC_INITFLAGS_DONT_DISABLE_OUTPUT_ON_STOP Safe backplane bus configuration disabled. This means that the backplane bus does not depend on the position of the RUN/STOP switch.</td>
<td>unsigned short</td>
</tr>
</tbody>
</table>

Safe backplane bus configuration

With the "Safe backplane bus configuration", the backplane bus responds according to the position of the RUN/STOP switch on the embedded controller.

**Requirement:**
The signal module outputs are enabled using the eccio_output_control function.

- **Switch in STOP position:**
  The outputs of all the signal modules are disabled.
- **Switch in RUN position once more:**
  The outputs of the signal modules are not automatically enabled.

**Note**
Always enable the outputs of the signal modules using the eccio_output_control function.

Return value

- EC_CIO_OK
- EC_CIO_E_PARAM
- EC_CIO_E_STATE
- EC_CIO_E_BUS
- EC_CIO_E_UNKnown
- EC_CIO_E_DRIVER

See also
eccio_deinitialize (Page 21)
eccio_output_control (Page 22)
Callback functions (Page 15)
Return values (Page 51)
3.2 Central I/O functions

3.2.1.2 eccio_deinitialize

Description
This function ends the use of the central I/O functions on the backplane bus.

- It stops all operations on the backplane bus.
- It disables the signal module outputs.
- The backplane bus is switched off (POWER OFF).
- The DC5V LED goes out, but this does not switch off the EC31.

Note
Call the function in the user program when all central I/O functions have ended.

Syntax
unsigned short eccio_deinitialize( )

Return value
- EC_CIO_OK
- EC_CIO_E_STATE
- EC_CIO_E_FATAL
- EC_CIO_E_BUS
- EC_CIO_E_UNKNOWN

See also
eccio_initialize (Page 19)
Return values (Page 51)
3.2 Central I/O functions

3.2.1.3 eccio_output_control

Description
This function enables or disables the outputs of the signal modules. Enable the outputs so that write functions can be executed at outputs.

Syntax
```
unsigned short eccio_output_control(
    unsigned short req_state)
```

Parameter

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>req_state</td>
<td>in</td>
<td>Permitted values</td>
<td>unsigned short</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• EC_CIO_PERI_ENABLE: enables the outputs</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• EC_CIO_PERI_DISABLE: disables the outputs</td>
<td></td>
</tr>
</tbody>
</table>

Return values
- EC_CIO_OK
- EC_CIO_E_PARAM
- EC_CIO_E_STATE
- EC_CIO_E_BUS
- EC_CIO_E_UNKNOWN
- EC_CIO_E_DRIVER

See also
- `eccio_initialize (Page 19)`
- `eccio_check_bus (Page 23)`
- Return values (Page 51)
3.2.1.4 eccio_check_bus

Description
This function compares the current configuration on the backplane bus with the list of stations that were identified using the eccio_initialize function. If the configuration differs from the saved list of stations, then the function returns the value EC_CIO_E_BUS. Use the eccio_initialize function to initialize the bus again before starting further operations.

Syntax
unsigned short eccio_check_bus (void)

Return values
- EC_CIO_OK
- EC_CIO_E_BUS
- EC_CIO_E_UNKNOWN
- EC_CIO_E_DRIVER

See also
eccio_output_control (Page 22)
Return values (Page 51)
3.2 Central I/O functions

3.2.1.5 eccio_ack_alarm

Description
This function acknowledges interrupts at a signal module. It must be called after the interrupt has been processed.

Note
The signal module cannot report a second interrupt until the first interrupt has been acknowledged.

Syntax
unsigned short eccio_ack_alarm (unsigned char alarm_type)

Parameter

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>alarm_type</td>
<td>in</td>
<td>Permitted values:</td>
<td>unsigned char</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• EC_CIO_PROCESS_ALARM</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• EC_CIO_DIAGNOSTIC_ALARM</td>
<td></td>
</tr>
</tbody>
</table>

Return value

- EC_CIO_OK
- EC_CIO_E_PARAM
- EC_CIO_E_STATE
- EC_CIO_E_BUS
- EC_CIO_E_UNKNOWN
- EC_CIO_E_DRIVER

See also
AlarmInfo (Page 50)
Return values (Page 51)
3.2 Central I/O functions

3.2.1.6 Callback alarm_notification

Description

The user-defined callback function is called when a process interrupt, or diagnostic interrupt is triggered at a signal module. When the eccio_initialize function is called, a pointer to the callback function is passed as a parameter. You can choose any name. The return value must be of the type void. The function writes the interrupt information in a structure of the type alarminfo.

Requirement

Parameters must be assigned to the relevant signal modules to trigger interrupts via the callback functions.

Rules

- The interrupt must be acknowledged using the eccio_ack_alarm function which must be called after the interrupt has been processed.
- If synchronization mechanisms are used, any blocks must be cancelled before the callback function is ended in order to avoid blockages.

NOTICE

Processing interrupts

In SIMATIC automation systems, process and diagnostic interrupts are triggered acyclically in response to specific events, rather than cyclically. As a result, they are relatively infrequent. If signal modules trigger interrupts too frequently, they can impact negatively on the stability of the operating system. For example, several interrupts in succession could block the operating system for a long period.

We therefore recommend that you implement your applications so that interrupts are only triggered in exceptional circumstances.

Syntax

Sample declaration:

```c
void usr_alarm_cbf(AlarmInfo* alarm_data)
```

Parameter

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>alarm_data</td>
<td>in</td>
<td>Pointer to a structure with interrupt information</td>
<td>Alarminfo*</td>
</tr>
</tbody>
</table>
3.2 Central I/O functions

3.2.2 Reading and writing data

3.2.2.1 eccio_read_data

Description
This function reads 1, 2 or 4 bytes from an input module.

Requirement:
- eccio_initialize() was executed successfully.
- The DC5V LED on the EC31 lights up
- The signal module is plugged in.
- The signal module is contained in the list of stations created by eccio_initialize().
- There are no faults in the signal module.
- Signal modules that are to be assigned parameters using parameter assignment data blocks 0 and 1 must receive at least the default values before the function can be executed.

Syntax
unsigned short eccio_read_data (GeoAddr geo, void* pret_buffer, unsigned char length)

Parameter

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>geo</td>
<td>in</td>
<td>Address of the signal module</td>
<td>GeoAddr</td>
</tr>
<tr>
<td>pret_buffer</td>
<td>out</td>
<td>Pointer to a buffer that holds data.</td>
<td>void*</td>
</tr>
<tr>
<td>length</td>
<td>in</td>
<td>Permitted values: 1, 2, 4</td>
<td>unsigned char</td>
</tr>
</tbody>
</table>

Return values
- EC_CIO_OK
- EC_CIO_E_PARAM
- EC_CIO_E_STATE
- EC_CIO_E_BUS
- EC_CIO_E_UNKNOWN
- EC_CIO_E_DRIVER

Reference
For information about the signal modules, refer to the S7-300 Automation System, Module data equipment manual. You will find the documentation on the Internet at:

See also
Return values (Page 51)
3.2.2.2  

**eccio_write_data**

**Description**

This function writes 1, 2 or 4 bytes to an output module.

**Requirement:**
- `eccio_initialize()` was executed successfully.
- The DC5V LED on the EC31 lights up
- The signal module is plugged in.
- The signal module is contained in the list of stations created by `eccio_initialize()`.
- The signal module supports the writing of data of the appropriate length.
- There are no faults in the signal module.
- The signal module's outputs have been enabled.
- Signal modules that are to be assigned parameters using parameter assignment data blocks 0 and 1 must receive at least the default values before the function can be executed.

**Syntax**

```c
unsigned short eccio_write_data ( 
    GeoAddr         geo, 
    void*           pbuffer,  
    unsigned char   length )
```

**Parameter**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>geo</td>
<td>in</td>
<td>Address of the signal module</td>
<td>GeoAddr</td>
</tr>
<tr>
<td>pbuffer</td>
<td>in</td>
<td>Pointer to a buffer that holds the data.</td>
<td>void*</td>
</tr>
<tr>
<td>length</td>
<td>in</td>
<td>Permitted values: 1, 2, 4</td>
<td>unsigned char</td>
</tr>
</tbody>
</table>

**Return values**

- EC_CIO_OK
- EC_CIO_E_PARAM
- EC_CIO_E_STATE
- EC_CIO_E_BUS
- EC_CIO_E_UNKNOWN
- EC_CIO_E_DRIVER
3.2 Central I/O functions

Reference

For information about the signal modules, refer to the S7-300 Automation System, Module data equipment manual. You will find the documentation on the Internet at:

http://support.automation.siemens.com/WW/view/de/8859629

See also

Return values (Page 51)

3.2.3 Assigning parameters to signal modules

3.2.3.1 Basic principles - parameter assignments

Default settings

In their as-delivered state, all modules with parameters in the S7 automation system are set to default values that are suitable for standard applications. These default values allow the modules to be used immediately without making any additional settings. To determine whether the signal modules can be assigned parameters, and to find the default values, refer to the module descriptions in the "S7-300 Automation System, Module Data" reference manual.
Assigning module parameters

The picture below shows the program flow for assigning parameters to signal modules on the backplane bus of the EC31.

1. Initialize bus
   `eccio_initialize(&BusEnum, ...)`

2. Prepare modules for configuration
   `eccio_def_par_write_broadcast(EC_CIO_DEFAULT_PAR_SET_MODULE_DEFPAR_ENABLE)`
   `eccio_def_par_write_broadcast(EC_CIO_DEFAULT_PAR_SET_MODULE_CLR_PAR)`

3. For each module
   3a. Configure module with own parameter data record
       `eccio_write_dataset(module_geo, DS_NR, &buffer, length)`
   3b. Configure module with default parameter data record
       `eccio_def_par_write_single(module_geo, EC_CIO_DEFAULT_PAR_SET_MODULE_DEF_PAR)`

4. Activate outputs of modules
   `eccio_output_control(EC_CIO_PERI_ENABLE)`

5. Deactivate module outputs
   `eccio_output_control(EC_CIO_PERI_DISABLE)`

Note

All relevant modules must be assigned parameters

Never assign parameters to signal modules with parameters in your user program (step 3.a or 3.b).
3.2.3.2 eccio_read_dataset

Description
This function reads parameter assignment data blocks up to 240 bytes long from a
signal module.

Requirements:
- eccio_initialize() was executed successfully.
- The signal module is plugged in.
- The signal module is contained in the list of stations created by eccio_initialize().
- The signal module supports the reading of data blocks.
- The data blocks correspond to the structure described in the S7-300 Automation System,
Module data equipment manual.

Syntax

```c
unsigned long eccio_read_dataset ( 
    GeoAddr geo,
    unsigned char ds_nr,
    unsigned char* pbuffer,
    unsigned short length )
```

Parameter

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>geo</td>
<td>in</td>
<td>Address of the signal module</td>
<td>GeoAddr</td>
</tr>
<tr>
<td>ds_nr</td>
<td>in</td>
<td>Number of the data block</td>
<td>unsigned char</td>
</tr>
<tr>
<td>pbuffer</td>
<td>out</td>
<td>Pointer to a buffer that holds the data.</td>
<td>unsigned char*</td>
</tr>
<tr>
<td>length</td>
<td>in</td>
<td>Length of the data block</td>
<td>unsigned short</td>
</tr>
</tbody>
</table>

Return values

- EC_CIO_OK
- EC_CIO_E_PARAM
- EC_CIO_E_STATE
- EC_CIO_E_BUS
- EC_CIO_E_UNKNOWN
- EC_CIO_E_DRIVER
- EC_CIO_W_LENGTH

Reference

For information about the signal modules, refer to the S7-300 Automation System,
Module data equipment manual. You will find this on the Internet at:


See also

GeoAddr (Page 47)
Return values (Page 51)
3.2.3.3  **eccio_write_dataset**

**Description**
This function writes parameter assignment data blocks up to 240 bytes long to a signal module. The function can be used to carry out the following operations:
- Assigning parameters to signal modules
- Setting the types of measurement and measuring ranges (voltage and current)
- Enabling / disabling interrupts

**Note**
When they receive the parameter assignment data blocks, signal modules need several milliseconds for the internal reassignment. They cannot be accessed during this time.

**Syntax**

```c
unsigned long eccio_write_dataset ( 
    GeoAddr geo,
    unsigned char   ds_nr,
    unsigned char*  pbuffer,
    unsigned short  length )
```

**Parameter**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>geo</td>
<td>in</td>
<td>Address of the signal module</td>
<td>GeoAddr</td>
</tr>
<tr>
<td>ds_nr</td>
<td>in</td>
<td>Number of the data block</td>
<td>unsigned char</td>
</tr>
<tr>
<td>pbuffer</td>
<td>in</td>
<td>Pointer to a buffer with the data to be written.</td>
<td>unsigned char*</td>
</tr>
<tr>
<td>length</td>
<td>in</td>
<td>Length of the data block</td>
<td>unsigned short</td>
</tr>
</tbody>
</table>

**Return values**

- EC_CIO_OK
- EC_CIO_E_PARAM
- EC_CIO_E_STATE
- EC_CIO_E_BUS
- EC_CIO_E_UNKNOWN
- EC_CIO_E_DRIVER

**Reference**
For information about the signal modules, refer to the *S7-300 Automation System, Module data equipment manual*. You will find this on the Internet at:


**See also**

- eccio_def_par_write_single (Page 32)
- eccio_def_par_write_broadcast (Page 33)
- GeoAddr (Page 47)
- Return values (Page 51)
3.2.3.4  **eccio_def_par_write_single**

**Description**

This function transfers the default parameter assignment to one signal module.

**Note**

Do not run the default parameter assignment unless the signal module has not already been assigned parameters using the **eccio_write_dataset** function.

**Syntax**

```c
unsigned short eccio_def_par_write_single(
    GeoAddr geo,
    unsigned short par_stat )
```

**Parameter**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>geo</td>
<td>in GeoAddr</td>
<td>Address of the signal module</td>
<td>GeoAddr</td>
</tr>
<tr>
<td>par_stat</td>
<td>in</td>
<td>Permitted values:</td>
<td>unsigned short</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• EC_CIO_DEFAULT_PAR_SET_MODULE_DEF_PAR</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• EC_CIO_DEFAULT_PAR_SET_MODULE_CLR_PAR</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• EC_CIO_DEFAULT_PAR_SET_MODULE_DEFPAR_ENABLE</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• EC_CIO_DEFAULT_PAR_SET_MODULE_PAR_DS</td>
<td></td>
</tr>
</tbody>
</table>

You will find information about the meaning of the parameter values, and how they are used in the user program at **Basic principles - parameter assignments** (Page 28).

**Return value**

- EC_CIO_OK
- EC_CIO_E_PARAM
- EC_CIO_E_STATE
- EC_CIO_E_BUS
- EC_CIO_E_UNKNOWN
- EC_CIO_E_DRIVER

**See also**

- **eccio_write_dataset** (Page 31)
- Return values (Page 51)
3.2.3.5  \textit{eccio\_def\_par\_write\_broadcast}

\textbf{Description}

This function transfers the default parameter assignment to all the signal modules on the backplane bus.

\textbf{Note}

Do not run the default parameter assignment unless the signal modules have not already been assigned parameters using the \textit{eccio\_write\_dataset} function.

\textbf{Syntax}

\begin{verbatim}
unsigned short   eccio_def_par_write_broadcast(
    unsigned short   par_stat )
\end{verbatim}

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>par_stat</td>
<td>in</td>
<td>Permitted values:</td>
<td>unsigned short</td>
</tr>
</tbody>
</table>
|           |       | • EC\_CIO\_DEFAULT\_PAR\_SET\_MODULE\_DEF\_PAR
|           |       | • EC\_CIO\_DEFAULT\_PAR\_SET\_MODULE\_CLR\_PAR
|           |       | • EC\_CIO\_DEFAULT\_PAR\_SET\_MODULE\_DEFPAR\_ENABLE
|           |       | • EC\_CIO\_DEFAULT\_PAR\_SET\_MODULE\_PAR\_DS   |

You will find information about the meaning of the parameter values, and how they are used in the user program at \ldots

\textbf{Return value}

- EC\_CIO\_OK
- EC\_CIO\_E\_PARAM
- EC\_CIO\_E\_STATE
- EC\_CIO\_E\_BUS
- EC\_CIO\_E\_UNKNOWN
- EC\_CIO\_E\_DRIVER

\textbf{See also}

- \textit{eccio\_write\_dataset} (Page 31)
- Return values (Page 51)
3.3 LED and RUN/STOP switch functions

3.3.1 ecledrs_initialize

Description
This function initializes the LED and RUN/STOP switch functions. Together with the other functions on the EC31, it is used to control LEDs, and to detect changes in the status of the RUN/STOP switch.

Note
Call the function in the user program before the other LED and RUN/STOP switch functions.

Syntax
unsigned short ecledrs_initialize(void)

Return value
- EC_LEDRS_OK
- EC_LEDRS_E_MISSINGDRIVER
- EC_LEDRS_E_STATE
- EC_LEDRS_E_UNKNOWN

See also
ecledrs_deinitialize (Page 35)
Return values (Page 51)
3.3 LED and RUN/STOP switch functions

3.3.2 ecledrs_deinitialize

Description
This function ends the use of the LED and RUN/STOP switch functions on the EC31.

Note
Call the function in the user program when all LED and RUN/STOP switch functions have ended.

Syntax
unsigned short  ecledrs_deinitialize(void)

Return value
- EC_LEDRS_OK
- EC_LEDRS_E_STATE
- EC_LEDRS_E_UNKNOWN

See also
ecdledrs_initialize (Page 34)
Return values (Page 51)
3.3 LED and RUN/STOP switch functions

3.3.3 ecledrs_write

Description

This function is used to control the LEDs on the EC31. It transfers the selected status to the controlled LEDs. The defines may be ORed to control several LEDs at the same time.

Figure 3-2 LEDs on the EC31

Syntax

```
unsigned short ecledrs_write ( 
    unsigned short led, 
    unsigned char state) 
```
### 3.3 LED and RUN/STOP switch functions

#### Parameter

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>led</td>
<td>in</td>
<td>LED on the EC31 to be activated.</td>
<td>unsigned short</td>
</tr>
<tr>
<td>state</td>
<td>in</td>
<td>Status with which the activated LED should respond.</td>
<td>unsigned char</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LED</th>
<th>Color</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>EC_LEDRS_BF1</td>
<td>Red</td>
<td>Bus fault 1 LED</td>
</tr>
<tr>
<td>EC_LEDRS_BF2</td>
<td>Red</td>
<td>Bus fault 2 LED</td>
</tr>
<tr>
<td>EC_LEDRS_U1BF3</td>
<td>Red</td>
<td>User 1 / Bus fault 3 LED</td>
</tr>
<tr>
<td>EC_LEDRS_U2BF4</td>
<td>Red</td>
<td>User 2 / Bus fault 4 LED</td>
</tr>
<tr>
<td>EC_LEDRS_U3</td>
<td>Yellow</td>
<td>User 3</td>
</tr>
<tr>
<td>EC_LEDRS_U4</td>
<td>Green</td>
<td>User 4</td>
</tr>
<tr>
<td>EC_LEDRS_SF</td>
<td>Red</td>
<td>System fault LED</td>
</tr>
<tr>
<td>EC_LEDRS_DC5V</td>
<td>Green</td>
<td>5V supply for the backplane bus (cannot be programmed)</td>
</tr>
<tr>
<td>EC_LEDRS_RUN</td>
<td>Green</td>
<td>RUN LED</td>
</tr>
<tr>
<td>EC_LEDRS_STOP</td>
<td>Yellow</td>
<td>STOP LED</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>STATE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>EC_LEDRS_STATE_ON</td>
<td>Activated LED lights up</td>
</tr>
<tr>
<td>EC_LEDRS_STATE_OFF</td>
<td>Activated LED goes out</td>
</tr>
<tr>
<td>EC_LEDRS_STATE_BLINK_SLOW</td>
<td>Activated LED flashes slowly (0.5 Hz)</td>
</tr>
<tr>
<td>EC_LEDRS_STATE_BLINK_FAST</td>
<td>Activated LED flashes quickly (2 Hz)</td>
</tr>
</tbody>
</table>

#### Return value

- EC_LEDRS_OK
- EC_LEDRS_E_STATE
- EC_LEDRS_E_UNKNOWN
- EC_LEDRS_E_PARAM

#### See also

[Return values](Page 51)
3.3.4  ecledrs_registerswitchchangecbk

**Description**

This function registers a callback function that signals changes in the status of the RUN/STOP switch on the EC31. The specified callback function is called for every status change. When it is registered, the callback function is called for the first time, and the current switch position is displayed.

**Note**

If the switch position changes very quickly (RUN-STOP-RUN), only the most recently registered switch position is signaled, rather than all the intermediate states.

Monitoring of the status of the RUN/STOP switch is only ended with the `ecledrs_deregisterswitchchangecbk` function.

**Callback function**

Sample declaration for a user-defined callback function:

```c
unsigned short  switch_cbf(
    unsigned char  newstate)
```

**Note**

The callback function should execute as few operations as possible so as not to block the system.

**Syntax**

```c
unsigned short  ecledrs_registerswitchchangecbk(
    FP_EC_LEDRS_SWITCH_CHANGE_CBK   prunstopchangecallback)
```

**Parameter**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>prunstopchangecallback</td>
<td>in</td>
<td>Pointer to user functions that are called in response to a change in status of the RUN/STOP switch on the EC31.</td>
<td>-</td>
</tr>
</tbody>
</table>

**Return value**

- EC_LEDRS_OK
- EC_LEDRS_E_PARAM
- EC_LEDRS_E_STATE
- EC_LEDRS_E_UNKNOWN

**See also**

- Callback functions (Page 15)
- Callback switch_change_notification (Page 40)
- Return values (Page 51)
3.3.5 `ecledrs_deregisterswitchchangecbk`

**Description**

This function ends the status monitoring for the RUN/STOP switch on the EC31 via the user program.

**Note**

Call the function in the user program when all RUN/STOP switch functions have ended.

**Syntax**

```c
unsigned short ecledrs_deregisterswitchchangecbk(void)
```

**Return value**

- `EC_LEDRS_OK`
- `EC_LEDRS_E_STATE`
- `EC_LEDRS_E_UNKNOWN`

**See also**

[Return values](Page 51)
3.3 LED and RUN/STOP switch functions

3.3.6 Callback switch_change_notification

Description
The user-defined callback function is called in response to changes in the status of the RUN/STOP switch on the EC31. When the `ecledrs_registerswitchchangecbk` function is called, a pointer to the callback function is passed as a parameter. You can choose any name. The return value must be of the type `void`. The response to status changes is defined by the program.

The following status changes are signaled (output parameter `state`)

<table>
<thead>
<tr>
<th>Definition</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>EC_LEDRS_SWITCH_RUN</td>
<td>Toggle to the RUN state</td>
</tr>
<tr>
<td>EC_LEDRS_SWITCH_STOP</td>
<td>Toggle to the STOP state</td>
</tr>
<tr>
<td>EC_LEDRS_SWITCH_MRES</td>
<td>Toggle to the MRES state</td>
</tr>
</tbody>
</table>

The callback function remains active, and signals any status changes that occur until it is deregistered by calling the `ecledrs_deregisterswitchchangecbk` function.

Syntax
Sample declaration:
```c
void register_switch_change_cbf (unsigned char  state)
```

Parameter

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>state</td>
<td>out</td>
<td>Status change</td>
<td>unsigned char</td>
</tr>
</tbody>
</table>

See also `ecledrs_registerswitchchangecbk` (Page 38)
3.4 Persistence functions

3.4.1 ecpers_initialize

Description
This function initializes the persistence functions on the EC31. A user-defined callback function is called if the power fails. This allows up to 256 KB data to be saved after a power failure.

**Note**
Call the function **before** the other persistence functions.

**Syntax**

```c
unsigned short ecpers_initialize(
    FP_EC_PERS_PFCALLBACK pfcallback,
    unsigned long* pmaxlength)
```

**Sample declaration for a user-defined callback function**

```c
void FP_EC_PERS_PFCALLBACK(void)
```

**Parameter**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>pfcallback</td>
<td>in</td>
<td>Pointer to the user-defined callback function to be called in response to a power failure on the embedded controller.</td>
<td>-</td>
</tr>
<tr>
<td>pmaxlength</td>
<td>out</td>
<td>Determines the size of the available retentive memory for saving data in the event of a power failure.</td>
<td>unsigned long*</td>
</tr>
</tbody>
</table>

**Return value**

- EC_PERS_OK
- EC_PERS_E_PARAM
- EC_PERS_E_OPENDRIVER
- EC_PERS_E_MAPPING_MEMORY
- EC_PERS_E_REGISTER_POWERFAIL
- EC_PERS_E_ALREADY_INITIALIZED

**See also**

- ecpers_deinitialize (Page 42)
- ecpers_readblock (Page 43)
- Return values (Page 51)
3.4 Persistence functions

3.4.2 ecpers_deinitialize

Description
This function ends the use of the persistence functions on the EC31.
The previously registered callback function will not be called again after this function.

Note
Call the function in the user program when the persistence functions have ended.

Syntax
unsigned short ecpers_deinitialize(void)

Return value
• EC_PERS_OK
• EC_PERS_E_NOT_INITIALIZED

See also
ecpers_initialize (Page 41)
Return values (Page 51)
3.4.3 ecpers_readblock

Description
This function reads data from a retentive memory.

Syntax
```
unsigned short ecpers_readblock(
    void* pbuffer,
    unsigned long offset,
    unsigned long length)
```

Parameter

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>pbuffer</td>
<td>in</td>
<td>Pointer to a buffer that holds the data.</td>
<td>void*</td>
</tr>
<tr>
<td>offset</td>
<td>in</td>
<td>Offset of the retentive memory in which the written data is to be read.</td>
<td>unsigned long</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The value must be less than *pmaxlength.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>*pmaxlength is used in the ecpers_initialize function.</td>
<td></td>
</tr>
<tr>
<td>length</td>
<td>in</td>
<td>Determines the size of the data to be read.</td>
<td>unsigned long</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The value must be greater than 0 and less than or equal to *pmaxlength - offset.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>*pmaxlength is used in the ecpers_initialize function.</td>
<td></td>
</tr>
</tbody>
</table>

Return value

- EC_PERS_OK
- EC_PERS_E_PARAM
- EC_PERS_E_NOT_INITIALIZED

See also

- ecpers_initialize (Page 41)
- Return values (Page 51)
3.4 Persistence functions

3.4.4 ecpers_writeblock

Description

This function writes data to a retentive memory.

Syntax

```c
unsigned short ecpers_writeblock(
    void* pbuffer,
    unsigned long offset,
    unsigned long length)
```

Parameter

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>pbuffer</td>
<td>in</td>
<td>Pointer to a buffer that contains the data to be written.</td>
<td>void*</td>
</tr>
<tr>
<td>offset</td>
<td>in</td>
<td>Offset of the retentive memory in which the data is to be written.</td>
<td>unsigned long</td>
</tr>
<tr>
<td>length</td>
<td>in</td>
<td>Determines the size of the data to be written.</td>
<td>unsigned long</td>
</tr>
</tbody>
</table>

- Offset of the retentive memory in which the data is to be written. The value must be less than *pmaxlength.
- *pmaxlength is used in the ecpers_initialize function.
- The value must be greater than 0 and less than or equal to *pmaxlength - offset.
- *pmaxlength is used in the ecpers_initialize function.

Return value

- EC_PERS_OK
- EC_PERS_E_PARAM
- EC_PERS_E_NOT_INITIALIZED

See also

ecpers_initialize (Page 41)
3.4.5 Callback power_fail_notification

Description
This user-defined callback function is called in response to a power failure on the EC31 if there is still data to be saved, for example. When the `ecpers_initialize` function is called, a pointer to the callback function is passed as a parameter. You can choose any name. The return value must be of the type `void`.

The callback function remains active until it is deregistered by calling the `ecpers_deinitialize` function.

Only one callback function of this type may be registered.

Syntax
Sample declaration:

```c
void   FP_EC_PERS_PFCALLBACK(void)
```

Parameter
None
Functions

3.4 Persistence functions
Appendix

A.1 Data types

A.1.1 GeoAddr

Description

The GeoAddr structure contains the address of a signal module.

Syntax

typedef struct {
   unsigned char rack;
   unsigned char slot;
   unsigned char reserved;
   unsigned char subaddress;
} GeoAddr;

Parameter

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
<th>Range of values</th>
</tr>
</thead>
<tbody>
<tr>
<td>rack</td>
<td>Number of the rack that contains the signal module.</td>
<td>unsigned char</td>
<td>0...3</td>
</tr>
<tr>
<td>slot</td>
<td>Slot that contains the signal module.</td>
<td>unsigned char</td>
<td>4...11</td>
</tr>
<tr>
<td>subaddress</td>
<td>Offset of the logical address</td>
<td>unsigned char</td>
<td>0...255</td>
</tr>
</tbody>
</table>

See also

Addressing signal modules (Page 13)
A.1 Data types

A.1.2 BusEnum

Description

The BusEnum structure contains a list of the signal modules on the backplane bus.

Syntax

typedef struct
{
    Module_Info Peril[MAX_MODULE_COUNT];
    unsigned short Module_Count;
    unsigned short Rack_Count;
    unsigned short Rack_Slot_Count[MAX_RACK_COUNT];
    unsigned short reserved;
    unsigned char reserved;
    unsigned char reserved;
} BusEnum;

Parameter

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
<th>Range of values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module_Info Peril</td>
<td>List of signal modules</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Module_Count</td>
<td>Number of all signal modules</td>
<td>unsigned short</td>
<td>0...32</td>
</tr>
<tr>
<td>Rack_Count</td>
<td>Number of all racks</td>
<td>unsigned short</td>
<td>1...4</td>
</tr>
<tr>
<td>Rack_Slot_Count</td>
<td>Number of signal modules</td>
<td>unsigned short</td>
<td>0...8</td>
</tr>
<tr>
<td>Rack_Im_Type</td>
<td>reserved</td>
<td>unsigned short</td>
<td>-</td>
</tr>
<tr>
<td>Rack_Is_Im_Available</td>
<td>reserved</td>
<td>unsigned char</td>
<td>-</td>
</tr>
<tr>
<td>Rack_Is_Im_Plugged</td>
<td>reserved</td>
<td>unsigned char</td>
<td>-</td>
</tr>
</tbody>
</table>
A.1.3 Module_Info

Description

The `Module_Info` structure contains the address of a plugged-in signal module.

Syntax

```c
typedef struct {
    unsigned short reserved;
    unsigned char Plugged;
    unsigned char Rack;
    unsigned char Slot;
} Module_Info;
```

Parameter

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
<th>Range of values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>reserved</td>
<td>unsigned short</td>
<td>-</td>
</tr>
<tr>
<td>Plugged</td>
<td>Is the signal module plugged in?</td>
<td>unsigned char</td>
<td>1: true, 0: false</td>
</tr>
<tr>
<td>Rack</td>
<td>Number of the rack that contains the signal module.</td>
<td>unsigned char</td>
<td>0...3</td>
</tr>
<tr>
<td>Slot</td>
<td>Slot into which the signal module is plugged.</td>
<td>unsigned char</td>
<td>4...11</td>
</tr>
</tbody>
</table>
A.1.4 AlarmInfo

Description

The AlarmInfo structure contains the address of a signal module that signals an interrupt, and information about the interrupts.

Syntax

```c
typedef struct {
    GeoAddr geo_address;
    unsigned short Status_Wd1;
    unsigned short Status_Wd2;
    unsigned short Alarm_Coming;
    unsigned char Alarm_Type;
} AlarmInfo;
```

Parameter

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
<th>Range of values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geo_address</td>
<td>Address of the signal module that triggered the interrupt</td>
<td>GeoAddr</td>
<td>-</td>
</tr>
<tr>
<td>Status_Wd1</td>
<td>First status word contained in the interrupt message.</td>
<td>unsigned short</td>
<td>Depends on the module</td>
</tr>
<tr>
<td>Status_Wd2</td>
<td>Second status word contained in the interrupt message.</td>
<td>unsigned short</td>
<td>Depends on the module</td>
</tr>
</tbody>
</table>
| Alarm_Coming | • A process interrupt is always "coming".  
               • A diagnostic interrupt may be "going", or "coming". | unsigned short | 0: EC_ALARM_COMING 
                                                     1: EC_ALARM_GOING |
| Alarm_Type   | Interrupt type:  
               • Process interrupt  
               • Diagnostic interrupt | unsigned char | 0: EC_CIO_PROCESS_ALARM 
                                                     1: EC_CIO_DIAGNOSTIC_ALARM |

Reference

For information about the signal modules, refer to the S7-300 Automation System, Module data equipment manual. You will find it on the Internet at:  

See also

eccio_ack_alarm (Page 24)
## A.2 Return values

### Return values

The following tables contain the return values for the functions, and options for eliminating errors.

### Return values and remedies

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Remedy</th>
</tr>
</thead>
<tbody>
<tr>
<td>EC_CIO_OK</td>
<td>The call was successfully processed.</td>
<td></td>
</tr>
<tr>
<td>EC_CIO_E_PARAM</td>
<td>A parameter is incorrect or does not correspond to the range of values.</td>
<td>Check the parameters, and call the function again.</td>
</tr>
<tr>
<td>EC_CIO_E_STATE</td>
<td>The call is not possible in this state.</td>
<td>Check the function and order of the calls:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- A call was sent after the backplane bus was deinitialized with eccio_deinitialize.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- A call was sent before the backplane bus was initialized with eccio_initialize.</td>
</tr>
<tr>
<td>EC_CIO_E_FATAL</td>
<td>The driver was unable to process the call.</td>
<td>Check whether the signal modules are plugged in correctly.</td>
</tr>
</tbody>
</table>
### Appendix

#### A.2 Return values

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Remedy</th>
</tr>
</thead>
<tbody>
<tr>
<td>EC_CIO_E_BUS</td>
<td>Error on the backplane bus</td>
<td>- A module is not detected. Check if the module is correctly inserted. Call <code>eccio_initialize</code> to update the list of stations.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The address of a module is incorrect: Check the parameters in the <code>GeoAddr</code> structure.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The signal module has not been assigned parameters, or the wrong parameters were assigned: Call <code>eccio_write_dataset</code> to assign parameters to the signal module. Follow the documentation for the signal modules.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Check whether the module supports the function. If a Read / Write function failed, check whether the function supports the necessary byte length. Follow the documentation for the signal modules.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The current configuration differs from the saved list of stations: Call <code>eccio_check_bus</code> to compare the current configuration on the backplane bus with the list of stations. If there are any differences, call <code>eccio_initialize</code> to update the list of stations.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The identification of the stations on the backplane bus is incomplete. Cause: Modules are in a temporarily unavailable state: Call <code>eccio_initialize</code> again after a short waiting period.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- A module was unplugged or plugged in while the backplane bus was being initialized, and the stations need to be identified.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Check whether the device is defective.</td>
</tr>
<tr>
<td>EC_CIO_E_UNKNOWN</td>
<td>Internal error</td>
<td>- Check the access rights</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Contact your local SIEMENS partner.</td>
</tr>
<tr>
<td>EC_CIO_E_DRIVER</td>
<td>Driver missing or is defective.</td>
<td>Install the driver, and restart the call.</td>
</tr>
<tr>
<td>EC_CIO_W_LENGTH</td>
<td>Data record too long.</td>
<td>Only for <code>eccio_read_dataset()</code>: Some signal modules support only the reading of 4 bytes or 16 bytes for data record 0 and data record 1. If you try to read a greater length of one of these two data records for such signal modules, the return value <code>EC_CIO_W_LENGTH</code> indicates that only 4 bytes (data record 0) or 16 bytes (data record 1) were actually read.</td>
</tr>
</tbody>
</table>
### A.2 Return values

#### Table A-2  Return values for LED functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Remedy</th>
</tr>
</thead>
<tbody>
<tr>
<td>EC_LEDRS_OK</td>
<td>The call was successfully processed.</td>
<td>-</td>
</tr>
<tr>
<td>EC_LEDRS_E_MISSINGDRIVER</td>
<td>The driver is not installed, or is defective.</td>
<td>Install the driver, and restart the call.</td>
</tr>
<tr>
<td>EC_LEDRS_E_STATE</td>
<td>The call is not possible in this state.</td>
<td>Check the order of the calls:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• A call was sent after the LED component was deinitialized with ecledrs_deinitialize.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• A call was sent before the LED component was initialized with ecledrs_initialize.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ecledrs_initialize was called more than once.</td>
</tr>
<tr>
<td>EC_LEDRS_E_UNKNOWN</td>
<td>Internal error</td>
<td>Check the access rights</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Contact your local SIEMENS partner.</td>
</tr>
<tr>
<td>EC_LEDRS_E_PARAM</td>
<td>At least one parameter is incorrect, or does not correspond to the range of values.</td>
<td>Check the parameters, and restart the call.</td>
</tr>
</tbody>
</table>

#### Table A-3  Return values for persistence functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Remedy</th>
</tr>
</thead>
<tbody>
<tr>
<td>EC_PERS_OK</td>
<td>The call was successfully processed.</td>
<td>-</td>
</tr>
<tr>
<td>EC_PERS_E_PARAM</td>
<td>A parameter is 0, or is not permitted for the Read / Write function.</td>
<td>Check the parameters, and restart the call.</td>
</tr>
<tr>
<td>EC_PERS_E_OPENDRIVER</td>
<td>The persistence driver could not be opened.</td>
<td>Check whether the driver is installed.</td>
</tr>
<tr>
<td>EC_PERS_E_MAPPING_MEMORY</td>
<td>Driver error while assigning the retentive memory.</td>
<td>• Check whether the driver is installed, and is working.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Check the hardware configuration</td>
</tr>
<tr>
<td>EC_PERS_E_REGISTER_POWERFAIL</td>
<td>Another application has already registered a callback function.</td>
<td>Check whether other applications are active.</td>
</tr>
<tr>
<td>EC_PERS_E_ALREADY_INITIALIZED</td>
<td>The persistence components have already been initialized.</td>
<td>Check the order of the calls.</td>
</tr>
<tr>
<td>EC_PERS_E_NOT_INITIALIZED</td>
<td>The persistence components were not initialized.</td>
<td>Check the order of the calls.</td>
</tr>
</tbody>
</table>
A.3 Values for data block 0

Some modules can generate interrupts when there is diagnostic information available. These "diagnostic interrupts" must be enabled by the entry in parameter assignment data block 0 (DS0). The settings are shown in the table below. Please note that some modules also require settings in parameter assignment data block 1.

DS0 values for analog modules

<table>
<thead>
<tr>
<th>Module</th>
<th>Order number</th>
<th>DS0 value if &quot;Diagnostic interrupt disabled&quot;</th>
<th>DS0 value if &quot;Diagnostic interrupt enabled&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>SM 331; AI 8 x 13 bit</td>
<td>6ES7331-1KF01-0AB0</td>
<td>(no interrupts, parameter assignment information only)</td>
<td></td>
</tr>
<tr>
<td>SM 331; AI 8 x 12 bit diagnostic interrupt / process interrupt</td>
<td>6ES7331-7KF02-0AB0</td>
<td>00 00</td>
<td>FF 01</td>
</tr>
<tr>
<td>SM 331; AI 2 x 12 bit diagnostic interrupt / process interrupt</td>
<td>6ES7331-7KB02-0AB0</td>
<td>00 00</td>
<td>03 01</td>
</tr>
<tr>
<td>SM 332, AO 4 x 12 bit diagnostic interrupt</td>
<td>6ES7332-5HD01-0AB0</td>
<td>00 00</td>
<td>0F 00</td>
</tr>
<tr>
<td>SM 332; AO 2 x 12 bit</td>
<td>6ES7332-5HB01-0AB0</td>
<td>00 00</td>
<td>03 00</td>
</tr>
<tr>
<td>SM 332; AO 8 x 12 bit</td>
<td>6ES7332-5HF00-0AB0</td>
<td>00 00</td>
<td>FF 00</td>
</tr>
</tbody>
</table>

Further references

For further information about the parameter assignment data block 1 for the signal modules, refer to the S7-300 Automation System, Module Data equipment manual, 02/2007.

You will find a detailed description of how to analyze the diagnostic data from signal modules in the user program in the STEP 7 documentation.
Index

A
Addresses
  Signal modules, 13
  Examples, 14
  AlarmInfo, 50

B
Backplane bus
  Configuration, 23
  Initializing, 19
  Basic knowledge, 3
  BusEnum, 48

C
Central I/O functions, 17
  Conventions, 3

D
Data
  Read, 26
  Writing, 27
  Data blocks
    Reading, 30
    Writing, 31
  Data type
    AlarmInfo, 50
    BusEnum, 48
    GeoAddr, 47
    Module_Info, 49
  Declaration
    Interrupt callback function, 15
    Default parameter assignment, 32, 33
    DLL files, 9

E
EC31, 3
  ECCIO, 7
  eccio_ack_alarm, 24
  eccio_check_bus, 23
  eccio_def_par_write_broadcast, 33
  eccio_def_par_write_single, 32
  eccio_deinit, 21
  eccio_init, 19
  eccio_output_control, 22
  eccio_read_data, 26
  eccio_read_dataset, 30
  eccio_write_data, 27
  eccio_write_dataset, 31
  ECLEDRS, 7
  ecledrs_deinit, 35
  ecledrs_deregisterswitchchangeckb, 39
  ecledrs_init, 34
  ecledrs_registerswitchchangeckb, 38
  ecledrs_write, 36
  ECPERS, 7
  ecpers_deinit, 42
  ecpers_init, 41
  ecpers_readblock, 43
  ecpers_writeblock, 44
  Embedded Controller EC31, 3
  End phase, 11
  Examples
    Addressing, 14

G
GeoAddr, 47

H
Header files, 9
  Sample program, 9

I
Initialization phase, 11
  Interrupts
    acknowledge, 24
    Activate, 31
    Disable, 31
Index

L
LED functions, 18
Libraries, 9
List of stations, 19

M
Measuring ranges, 31
Module_Info, 49

O
Outputs
   Disabling, 22
   Enabling, 22

P
Parameter
   subaddress, 47
Parameter
   rack, 47
   slot, 47
Persistence, 7
Persistence functions, 18
Productive mode, 11
Program
   create, 10
   Program flow, 12
   Programming environment, 9

R
Read
   Data, 26
   Parameter assignment data blocks, 30
Rules
   User program, 11
   RUN/STOP switch functions, 18

S
S7 Modular Embedded Controller, 3
S7-mEC, 3
Sample declaration
   Interrupt callback function, 15
Sample program, 8
Scope, 3
Signal modules
   Assign parameters, 31
   Software Development Kit, 7

T
Types of measurement, 31

U
User program
   create, 10
   Rules, 11
   Sequence of operations, 12
   Structure, 11

W
Writing
   Data, 27
   Parameter assignment data blocks, 31