

## SIMATIC

### CFC for M7 Continuous Function Chart

#### Manual

Preface, Contents	
CFC for SIMATIC M7	<b>1</b>
Handling Blocks	<b>2</b>
Configuring Tasks	<b>3</b>
Compiling and Downloading	<b>4</b>
Test and Commissioning	<b>5</b>
Creating Block Types	<b>6</b>
<b>Appendix</b>	
Technical Specifications	<b>A</b>
Abbreviations	<b>B</b>
Glossary, Index	

## Safety Guidelines

This manual contains notices intended to ensure personal safety, as well as to protect the products and connected equipment against damage. These notices are highlighted by the symbols shown below and graded according to severity by the following texts:



---

### Danger

indicates that death, severe personal injury or substantial property damage will result if proper precautions are not taken.

---



---

### Warning

indicates that death, severe personal injury or substantial property damage can result if proper precautions are not taken.

---



---

### Caution

indicates that minor personal injury can result if proper precautions are not taken.

---

---

### Caution

indicates that property damage can result if proper precautions are not taken.

---

---

### Notice

draws your attention to particularly important information on the product, handling the product, or to a particular part of the documentation.

---

## Qualified Personnel

Only **qualified personnel** should be allowed to install and work on this equipment. Qualified persons are defined as persons who are authorized to commission, to ground and to tag circuits, equipment, and systems in accordance with established safety practices and standards.

## Correct Usage

Note the following:



---

### Warning

This device and its components may only be used for the applications described in the catalog or the technical description, and only in connection with devices or components from other manufacturers which have been approved or recommended by Siemens.

This product can only function correctly and safely if it is transported, stored, set up, and installed correctly, and operated and maintained as recommended.

---

## Trademarks

SIMATIC®, SIMATIC HMI® and SIMATIC NET® are registered trademarks of SIEMENS AG.

Third parties using for their own purposes any other names in this document which refer to trademarks might infringe upon the rights of the trademark owners.

### Copyright © Siemens AG 2003 All rights reserved

The reproduction, transmission or use of this document or its contents is not permitted without express written authority. Offenders will be liable for damages. All rights, including rights created by patent grant or registration of a utility model or design, are reserved.

### Disclaimer of Liability

We have checked the contents of this manual for agreement with the hardware and software described. Since deviations cannot be precluded entirely, we cannot guarantee full agreement. However, the data in this manual are reviewed regularly and any necessary corrections included in subsequent editions. Suggestions for improvement are welcomed.

Siemens AG  
Bereich Automation and Drives  
Geschäftsgebiet Industrial Automation Systems  
Postfach 4848, D- 90327 Nuernberg

Siemens Aktiengesellschaft

©Siemens AG 2003  
Technical data subject to change.

A5E00177321-01



# Preface

## Purpose of the Manual

Along with the manual "CFC for S7", this manual "CFC for M7" provides you with all the information you require to use the CFC configuration tool in conjunction with a SIMATIC M7 CPU.

Sections in the manual "CFC for S7" that relate only to S7 or differ from M7 are indicated.

This manual "CFC for M7" contains information that relates only to M7.

## Audience

This manual is intended for personnel involved in configuring, commissioning, and service.

## Validity of the Manual

This manual is valid for CFC software version 5.1 and higher.

## Standard

The CFC software is based on the international standard DIN EN 61131-3 (IEC 1131-3) for programming languages for programmable logic controllers.

## Conventions

References to other documentation are indicated by numbers in slashes /.../. Based on the number, you can check the full title of the documentation in the References at the end of the manual.

## Further Support

If you have any technical questions, please get in touch with your Siemens representative or agent responsible.

<http://www.siemens.com/automation/partner>

## Training Centers

Siemens offers a number of training courses to familiarize you with the SIMATIC S7 automation system. Please contact your regional training center or our central training center in D 90327 Nuremberg, Germany for details:

Telephone: +49 (911) 895-3200.

Internet: <http://www.sitrain.com>

## A&D Technical Support

Worldwide, available 24 hours a day:



<b>Worldwide (Nuernberg)</b> <b>Technical Support</b>  24 hours a day, 365 days a year Phone: +49 (0) 180 5050-222 Fax: +49 (0) 180 5050-223 E-Mail: adsupport@siemens.com GMT: +1:00		
<b>Europe / Africa (Nuernberg)</b> <b>Authorization</b>  Local time: Mon.-Fri. 8:00 to 17:00 Phone: +49 (0) 180 5050-222 Fax: +49 (0) 180 5050-223 E-Mail: adsupport@siemens.com GMT: +1:00	<b>United States (Johnson City)</b> <b>Technical Support and Authorization</b>  Local time: Mon.-Fri. 8:00 to 17:00 Phone: +1 (0) 423 262 2522 Fax: +1 (0) 423 262 2289 E-Mail: simatic.hotline@sea.siemens.com GMT: -5:00	<b>Asia / Australia (Beijing)</b> <b>Technical Support and Authorization</b>  Local time: Mon.-Fri. 8:00 to 17:00 Phone: +86 10 64 75 75 75 Fax: +86 10 64 74 74 74 E-Mail: adsupport.asia@siemens.com GMT: +8:00
The languages of the SIMATIC Hotlines and the authorization hotline are generally German and English.		

## **Service & Support on the Internet**

In addition to our documentation, we offer our Know-how online on the internet at:

<http://www.siemens.com/automation/service&support>

where you will find the following:

- The newsletter, which constantly provides you with up-to-date information on your products.
- The right documents via our Search function in Service & Support.
- A forum, where users and experts from all over the world exchange their experiences.
- Your local representative for Automation & Drives via our representatives database.
- Information on field service, repairs, spare parts and more under "Services".

# Contents

<b>1</b>	<b>CFC for SIMATIC M7</b>	<b>1-1</b>
1.1	CFC in the STEP 7 Environment .....	1-2
1.1.1	Installation and Configuration.....	1-3
1.1.2	Data Flow of Configuration Data .....	1-4
1.2	Managing the PLC .....	1-5
<b>2</b>	<b>Handling Blocks</b>	<b>2-1</b>
2.1	Installing Source Files in the File System .....	2-1
2.2	Copying Block Types .....	2-3
2.3	Importing Block Types.....	2-4
2.4	Inserting Blocks.....	2-5
2.5	Deleting Block Types .....	2-6
<b>3</b>	<b>Configuring Tasks</b>	<b>3-1</b>
3.1	General.....	3-1
3.2	Configuring a Task in the "Tasks" Dialog.....	3-2
3.2.1	Startup Task, Background Task, and Error Task.....	3-3
3.2.2	Cyclic Interrupts .....	3-4
3.2.3	Software Interrupts .....	3-6
3.2.4	Hardware Interrupts .....	3-6
3.3	Task Priorities.....	3-8
3.3.1	RMOS Task Priorities.....	3-9
<b>4</b>	<b>Compiling and Downloading</b>	<b>4-1</b>
4.1	Compiling as Program.....	4-1
4.2	Customizing Compilation .....	4-2
4.2.1	Specifying the Software Version .....	4-2
4.2.2	Optimizing the Run Time.....	4-2
4.2.3	Setting RMOS Priorities .....	4-3
4.3	Compiling as Block Type.....	4-4
4.4	Downloading.....	4-5
4.4.1	Downloading the User Program .....	4-5
4.4.2	Downloading with "Manage M7 System" .....	4-5
<b>5</b>	<b>Test and Commissioning</b>	<b>5-1</b>
5.1	General.....	5-1
5.1.1	Activating or Deactivating the Test Mode .....	5-2
5.2	Monitoring and Assigning Parameters to Block I/Os .....	5-3
5.3	Working with the Oscilloscope .....	5-4
5.3.1	The "Oscilloscope" Window .....	5-5
5.3.2	Selecting, Opening, and Renaming an Oscilloscope.....	5-6
5.3.3	Creating a New Oscilloscope .....	5-6
5.3.4	Starting and Evaluating a Recording .....	5-7
5.3.5	Printing an Oscilloscope.....	5-9
5.3.6	Deleting an Oscilloscope.....	5-9

5.4	Working with Breakpoints.....	5-10
5.4.1	General Information on Working with Breakpoints.....	5-10
5.4.2	Activating Breakpoints on the PLC.....	5-12
5.4.3	Setting, Deleting, Enabling, Disabling Breakpoints.....	5-13
5.4.4	Editing Breakpoints .....	5-14
5.5	Holding and Resuming the CFC Program .....	5-16
5.5.1	Holding the Program .....	5-16
5.5.2	Resuming the Program .....	5-16
5.6	Additional Functions .....	5-18
5.6.1	CFC System Status:.....	5-18
5.6.2	CFC Interrupt Stack: .....	5-19
5.6.3	Status of CFC Tasks .....	5-19
<b>6</b>	<b>Creating Block Types</b>	<b>6-1</b>
6.1	General.....	6-1
6.2	Creating Block Types .....	6-2
6.2.1	The Source File .....	6-2
6.3	The Keywords .....	6-4
6.3.1	Defaults for the Optional Keywords.....	6-5
6.3.2	Description of the Keywords .....	6-6
6.3.3	Rules for C Code and Examples.....	6-14
6.3.4	Testing Block Types .....	6-21
<b>A</b>	<b>Technical Specifications</b>	<b>A-1</b>
A.1	Technical Specifications.....	A-1
<b>B</b>	<b>Abbreviations</b>	<b>B-1</b>
<b>Glossary</b>		
<b>Index</b>		



# 1 CFC for SIMATIC M7

## Overview

This section provides you with information about CFC, shows how it fits in to the STEP 7 software package, and describes how it works, in particular, aspects not dealt with in the manual "CFC for S7".

This includes installation and configuration of the software and configuration of the CPU.

## 1.1 CFC in the STEP 7 Environment

### Software Environment

The SIMATIC Manager is used for all PLCs as the graphic interface to coordinate the tools and objects. The SIMATIC Manager manages tools and data and is used, for example, to create and modify a project structure (CPU, CFC charts) and to start the CFC Editor.

The Windows 95 /98 /NT operating system and the STEP 7 standard package must be installed. With CFC, you require further optional software packages.

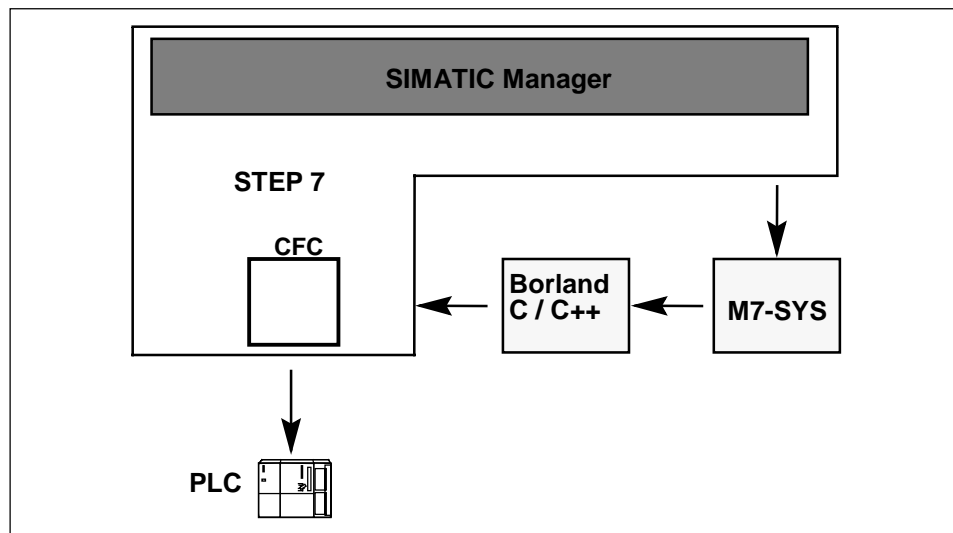


Figure 1-1 CFC in the System Environment

### 1.1.1 Installation and Configuration

With CFC for M7, you require the M7-SYS optional package and Borland C (V5.01).

Borland C++ is the programming language for creating block types for M7.

M7 blocks are shipped with CFC that you can include (import) in your CFC data.

To allow M7 block import and to ensure that the M7 compiler operates free of errors, the Borland C++ compiler must be installed completely or user-defined with the following minimum configuration:

- In the "Borland C++ Installation" window, select "user-defined"; you do not need to select the available options.
- "32-bit windows" must be selected in the "Borland C++ destination platforms".
- "Command line tools" must be selected in the "Borland C++ Tools" window.
- The boxes "Visual Tools", "Borland Database Engine", "Examples" and "Help" can be completely deselected.
- In the "Libraries" box, the "Run-time libraries" must be selected. The header files, static libraries, and dynamic libraries are also required for a minimum configuration.

### 1.1.2 Data Flow of Configuration Data

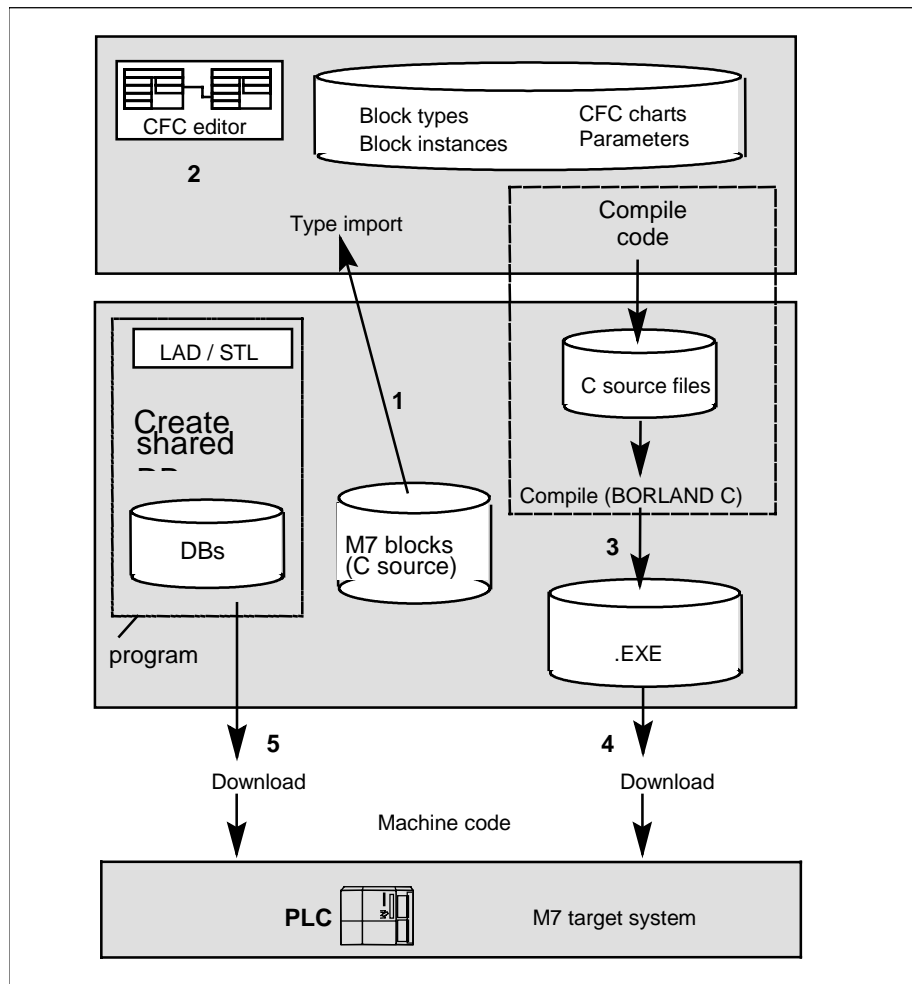


Figure 1-2 Creating Configuration Data

#### Key to Figure 1-2

1. You create the block types outside STEP 7 using the C programming language. You either install these C source files with CFC or store them in the project.
2. Using the CFC editor, you create charts by inserting blocks and then assigning parameters to them and interconnecting them.
3. You compile the CFC charts to create a CFC program.
4. You then download the CFC program to the CPU.
5. (Optional) If shared data blocks were created, these must be downloaded to the PLC using the SIMATIC Manager.

## 1.2 Managing the PLC

### Configuring

The "PLC > Manage PLC" menu command is used to configure the operating system and to copy and delete software on the current M7 CPU. It is identical to the corresponding function in the SIMATIC Manager.

---

**Note**

There may be more than one version of the M7 system software installed on your programming device/PC. CFC normally uses the latest version. This can lead to conflicts if your M7 CPU is incompatible.

If you use the "Manage PLC" function before you have compiled an M7 user program, the latest version of the M7 software is used automatically. Make sure that software compatible with your CPU is set.

For information on setting the software version: see Section 4.2.1.

---

### Downloading with "Manage PLC"

You can also download the CFC run-time system and the CFC programs with the function "PLC > Manage PLC...". See also Section 4.4.2.



## 2 Handling Blocks

### Introduction

This chapter explains how to insert, copy, and import block types into the file system and how to insert and delete them in CFC charts.

### 2.1 Installing Source Files in the File System

#### Source Files and the Block Types

M7 block types are C source files in ASCII format. A source file contains the description of the block type (name, inputs/outputs, etc.) and the program code in the C language.

Ready-made block type source files are shipped with CFC. You can purchase further files, if required. You can also define your own block types by creating the source files using an ASCII editor. You must then install these additional source files in the file system.

#### Positioning

You can place the the block type source files and code libraries at two different locations in the file system depending on their use:

- In CFC:  
Block types used by more than one project must be copied to CFC. When you install CFC, a basic pool of block types is created.
- In the project:  
Block types that are specific to a particular project, must be copied to the project. They represent the pool for all chart folders (charts) of the project.

---

#### Note

Do not use names containing umlauts (ä, ü etc.) in the project path since the Borland Compiler cannot interpret them.

---

## Copying a Source File to the Project

To copy source files to the project:

- Check the project path of your project (in the SIMATIC Manager using "Edit > Object Properties" for the project, in the "Location of the project" box) and create a folder with the name M7BLOCKS and below it the subfolders SRC, INC, and LIB in the project path using the Windows Explorer ! ?.
- Select the files you require for the project either from the source files shipped with the product or that you created yourself that you require for the project and copy them to the SRC folder with the Windows Explorer.  
Copy any include files you require to the INC folder and code libraries to the LIB folder ! ?.

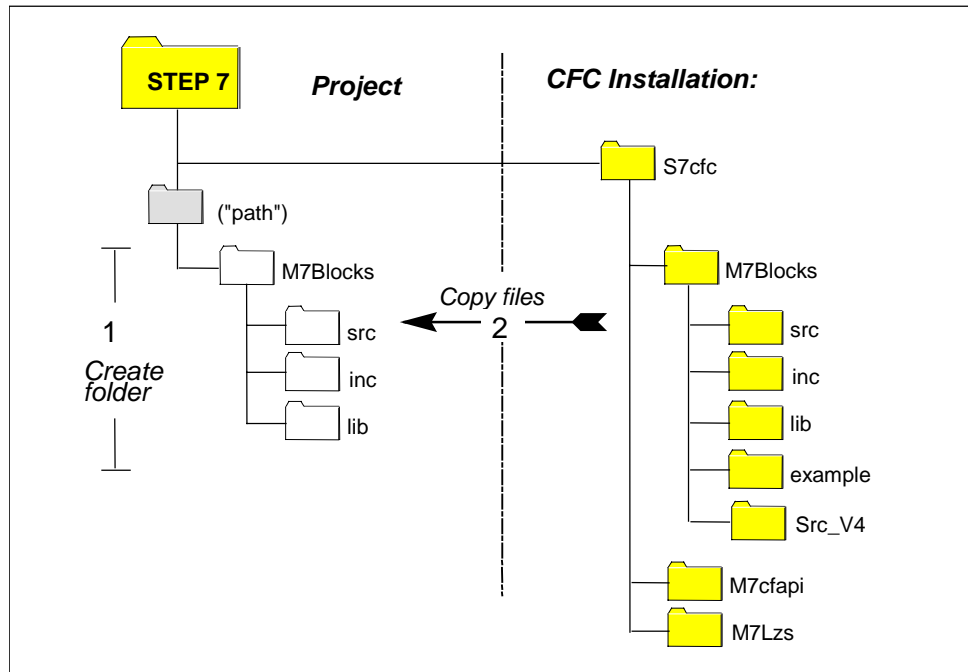


Figure 2-1: Placing the Source Files in the File Tree

## Copying a Source File to CFC

In the Windows Explorer, go to the folder containing the CFC installation (S7CFC) and open the M7BLOCKS folder. Copy the source files to SRC. Copy any include files you may require to INC and code libraries to LIB.

## Import

After creating and copying the source files, you must import the block types to the chart folder.

For more information on importing see Section 2.3.



## **2.2 Copying Block Types**

### **Copying**

You can copy block types from one project to another. Source files and code libraries that exist in CFC do not need to be copied since they are known in the destination project.

Using the Windows Explorer, you can copy the source files of the required block types from the "SRC" folder of the source project to the same folder of the destination project and the code libraries from the "LIB" folder of the source project to the same folder of the destination project.

### **Error Log**

If you forget to copy a code library that is required, the error message "Compiler reports errors" informs you of the error.

If you double-click "Compiler reports errors" or click the "Compiler Messages" button, you can display the error log.

## 2.3 Importing Block Types

In M7, the block types must exist as C source files. There are always two sources for importing block types:

- CFC installation:  
The block types are installed as well. These block types can be used for all projects.
- Project:  
Project-specific block types. This is the pool for all charts of the chart folder.

### Procedure

- Open a chart in the relevant chart folder and select the "Options > Block Types" menu command.  
A dialog box with two windows appears. The left-hand window lists the available pool of block types of the selected source file (Source files: "Project" or "CFC Installation"). In the right-hand window "Chart Folder" you will see the block types that have already been imported. Basic operations (BOP) are not displayed here since they can neither be deleted nor overwritten.
- Select the blocks you want to import in the left window. You can now drag the selected blocks to the right-hand window or click the "-->" button. The import (and with it compilation) is started and a dialog box with the log window is displayed.

If you want to import a lot of block types, the import can take a considerable time due to the test compilation. Tip: Create an M7 program with the required block types and copy this. This makes the multiple import unnecessary.

### Checks

When you import block types, the following checks are made:

- The time of the last modification in each source file and the time stamp of the corresponding block type in the M7 library and in the M7 data management (ensuring data consistency).
- The source file is checked to make sure it is free of errors (for example correct syntax and order of keywords used). If errors are found, an error list with the line numbers is displayed.
- The C code of the block type is put through a test compilation. If errors occur, they are signaled. The compiler errors are listed in their own log ("Compiler Messages").

## **Compiler Error Messages**

If you double-click "Compiler reports errors" or single-click the "Compiler Messages" button, you can display the error log. If the log file contains several messages, the text for the selected (incorrect) block is visible in the window.

## **Log**

You can also display a log later and print it out. With the "Options > Logs..." function, you display a dialog box with M7 logs. Here, select the "Block Types" tab.

## **2.4 Inserting Blocks**

### **Block Types in CFC**

Blocks you want to insert into a CFC chart must first be imported into CFC. A basic pool of block types is supplied.

The imported block types are listed in the block catalog and are sorted in families.

### **Inserting**

You can drag the blocks you require from the open block family of the block catalog and insert them in the chart.

## 2.5 Deleting Block Types

### Removing Block Types from the Chart Folder

With the "Options > Block Types..." menu command, you can delete block types that are no longer required in the chart folder. Select the relevant blocks and click the "Remove" button. The blocks are removed from the list.

**Note:** If the block type you want to delete still exists in a chart in the current chart folder, an error message will be displayed. To delete such block types, you must first delete all the instances of the block type in all the charts of the active chart folder. Afterwards, you can select the block type you want to delete in the chart folder list and delete it from the CFC data management.

## 3 Configuring Tasks

### Overview

This chapter contains information about creating tasks in your project for the assigned CPU.

### 3.1 General

A CFC program runs in various tasks on the M7 PLC that are mapped to RMOS tasks. Before you insert blocks into the CFC chart and specify run-time properties, you must decide which tasks will exist on the relevant CPU and the conditions under which they will be executed. The task "OB1" (free cycle) already exists. You create and set parameters for the tasks in the "Tasks" dialog.

Since all the run-time properties are specified using the assigned task names, the task names must be unique within a CPU.

### Numbers of Tasks and Interrupts

The following tasks and interrupts are available within a CPU:

- One startup task and one background task
- Three error tasks (a maximum of one task for each error class)
- Nine cyclic interrupts
- Eight interrupts (any mixture of software and hardware interrupts as required).

## 3.2 Configuring a Task in the "Tasks" Dialog

To configure a task:

1. Open a CFC chart assigned to the CPU for which you want to configure the tasks.
2. Select the "Options > OBs/Tasks..." menu command.  
The "Tasks" dialog box is displayed.

Depending on the event (startup, error, interrupt) that leads to the task being activated, the dialog box is divided into the following tabs:

- Startup/Background/Errors
- Cyclic Interrupt
- Software Interrupt
- Hardware Interrupt (1-4)
- Hardware Interrupt (5-8)

In each tab, you define the data for the relevant tasks. The following sections contain more detailed information on the contents of the tabs.

When you open a tab, the parameters that have already been set are displayed.

A task is defined as soon as you assign a task name. This means that as soon as you have assigned a name, you must also enter the other parameters.

3. With the "Apply" button, you save the data you have entered without closing the dialog.  
With "OK", you save the data and close the dialog.

### 3.2.1 Startup Task, Background Task, and Error Task

#### Startup and Background Task

A startup task can be assigned to every CPU of the M7 PLC. This is the first task to be executed following a reset, following return of power after a power outage or when the CPU is changed from STOP to RUN.

You can also configure a background task that is always executed when there is no other task active. The minimum and maximum cycle time can be set in HW Config in the Properties dialog of the CPU.

In the "Startup/Background/Errors" tab, you define the two tasks as follows:

- The names
- Whether or not they should be activated
- The stack sizes
- Whether or not they should react to processor errors with substitute values.

For a description of the parameters, refer to Section 3.2.2, Cyclic Interrupts.

#### Error Tasks

The run-time system is capable of detecting certain errors during the run time. To allow you to react to errors of this type, you can configure an error task and assign it to the particular error. There are three error classes, as follows:

- **I/O access errors** occur, for example when the program attempts to access a module that has been removed. If an I/O access error task is defined and activated on the CPU, the CPU does not change to STOP when an access error occurs but remains in the current mode.
- **Processor errors** occur, for example when division by 0 is attempted (software error in the user program). The error task is called only if you do not work with substitute values.
- **Timeouts** occur when the execution time of a cyclic task is longer than the cycle time in which the task is started. If an timeout error task is defined and activated on the CPU, the CPU does not change to STOP when a timeout occurs but remains in the current mode.

The error tasks "I/O Access" and "Processor" are synchronous to the task that caused them; in other words, the error task must be completed before program execution can continue. The "Timeout" error task is asynchronous.

In the "Startup/Background/Errors" tab, you can select the following options for the three error classes:

- Define an error task
- Specify the stack size
- Specify whether it should be activated

### 3.2.2 Cyclic Interrupts

Cyclic interrupts are tasks that are started at regular intervals. The cyclic interrupt task is executed at the defined time as long as no higher-priority task is being executed at this time.

In the "Cyclic Interrupt" tab, you specify the following parameters for each task.

- Name of the task
- Active yes/no
- Priority (value from 2 to 24)
- Stack (stack size)
- Substitute value yes/no
- Executed (interval) in ms
- Phase offset in ms

#### Tasks Active/Inactive

During configuration, you can set an attribute that switches each task "active" or "inactive".

All tasks configured with the "active" attribute are installed with their current priority when they are compiled. Tasks configured as "inactive" are created during compilation but are at least initially not executed. This can be useful for test purposes.

If you select the "PLC > Additional Functions" menu command and click the "Status of CFC Tasks" button, you can enable the inactive tasks during run time. See also Section 5.6.3.

#### Priority:

For more detailed information, refer to Section 3.3.

#### Stack Size

You can specify the size of the stack for each individual task. If you do not specify a size during configuration, the stack is created with a size of 1 Kbyte.

If you use a lot of local variables within the block and these are not defined with the VAR section, you may find it necessary to increase the size of the stack.



### Substitute Values:

For each task, you can decide how the system will react to the processor errors listed below:

- Integer division by 0
- Floating-point division by 0
- Floating-point overflow and underflow

If one of these events occurs, the CPU either changes to STOP or uses a substitute value with which it can continue operation; this means that the CPU remains in the currently active mode.

If no substitute value is used (default), the error task for processor errors is called if it was configured. Following this, the CPU changes to STOP.

If you decide to use substitute values, a substitute value is formed for the value that caused the processor error and the system continues to calculate with this value.

### Execution Cycle

With the execution cycle, you specify the time allowed for execution of the blocks in the task. The execution time is a multiple of the basic clock rate of 1 ms.

Note the following:

- When you define the execution cycle, remember that the time for executing the task (the time for executing the blocks contained in the task) must be less than the execution cycle otherwise a timeout will occur.
- You should allow reserves for system functions and for delays caused by high-priority interrupts that can occur during operation (for example hardware interrupts).

### Phase Offset

If there are several cyclic interrupts with the same priority and the same execution interval, it is advisable to define phase offsets. Different phase offsets mean that the execution times are offset even when other parameters are the same. This balances the load uniformly within the CPU.

The point at which a cyclic interrupt is executed is obtained from the execution interval plus the phase offset. The execution interval is always a multiple of the basic clock rate of 1 ms. The same applies to phase offset.

The following example shows when two cyclic interrupts are executed.

1. Execution = 8, phase offset = 0 (in other words no offset)
2. Execution = 8, phase offset = 3

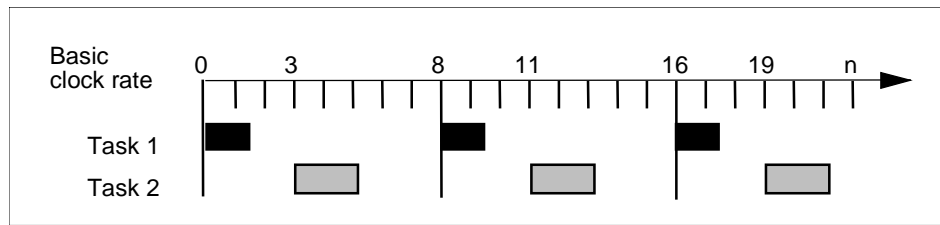


Figure 3-1: Cyclic Interrupt Tasks with and without Phase Offset

### 3.2.3 Software Interrupts

Software interrupts are tasks that can be started by the user program. Here, you use the "EVENT" block that is part of the CFC installation. The task to be started is referenced by its name.

In the "Software Interrupt" tab, you enter the following information for each task:

- Name of the task
- Active yes/no
- Priority (value from 2 to 24)
- Stack (stack size)
- Substitute value yes/no

For a description of the parameters, refer to Section 3.2.2, Cyclic Interrupts.

### 3.2.4 Hardware Interrupts

Hardware interrupts are triggered by hardware interrupt signals. You can assign these interrupt signals to a task to allow you to react to external process events. The signals must be defined in the symbol table.

In the "Hardware Interrupt (1-4)" or "Hardware Interrupt (5-8)" tabs, you specify the following parameters for each task:

- Name of the task
- Active yes/no
- Priority (value from 2 to 24)
- Stack (stack size)
- Substitute value yes/no
- Symbol (signal name)
- Channels

## Symbol

Under Symbol, you enter the hardware address of the module whose interrupt will trigger the task. This is the address at which the module was configured in hardware configuration. If, for example, you have configured an input module with interrupt capability at address 320, you must create a symbol with the address PIW 320 in the symbol table.

You select a symbol in the symbol table by positioning the cursor on the input fields and clicking the "Insert Symbol" button.

The symbol table browser is started. Here, you can select a signal that will be entered in the active input box.

## Channels

Depending on the particular design, modules that generate interrupts can have several channels that trigger individual interrupts. Modules can have up to 32 channels. You will find the required information in the relevant hardware documentation.

With the "Channels" setting, you can mask out individual channels of a module that are irrelevant. With the default setting 0x00000000, all possible channels would start the task.

There are two ways of masking out channels:

- In hexadecimal, filling in the relevant input box
- Bit-by-bit, by clicking the corresponding check boxes.

For a description of the other parameters, refer to Section 3.2.2, Cyclic Interrupts.

### 3.3 Task Priorities

Each task has a priority. For the startup task, the background task, and the timeout task, the priority is fixed by the task type. The "Process Error" and "I/O Access Error" tasks have the priority of the calling task.

For the other tasks, you can define the priority within a limited range.

The CFC priorities are converted to RMOS task priorities as shown in the following graphic:

Priorities		Tasks	Error Tasks
CFC	RMOS		
26	132		Timeout errors
25	131	Startup	Processor errors I/O access errors
24	130	Interrupts Cyclic interrupts Hardware Interrupts Software interrupts	
2	108		
1	100	Background	
0	-	Disabled tasks	

Figure 3-2: Task Priorities

Priority 1 is the lowest priority and priority 26 the highest. If a task is not activated (disabled), it is assigned priority 0 internally.

### 3.3.1 RMOS Task Priorities

If you configure the tasks so that they correspond to the defaults in CFC, the RMOS task priorities are lower than the system utilities of the M7 software. This prevents a configured application from blocking the system due to excessively long run times.

In specific situations, it may be useful to configure tasks with higher priorities if they are particularly time-critical. Otherwise there is no guarantee that your task will start reliably.

This might, be required in the following situations:

- When using time-controlled tasks with short cycle times (for example  $\leq 10$  ms).
- When you want to react quickly to hardware interrupts.

---

**Note**

If you change a CFC priority to a different RMOS task priority, all the tasks that you configure with this CFC priority or have already configured will be assigned the new RMOS task priority.

---

You can change the default RMOS task priorities in the dialog displayed by the "Options > Customize > Compile..." menu command. See Section 4.1.




## 4 Compiling and Downloading

### Overview

This chapter explains how to compile your configured CFC chart and download it to the CPU.

### 4.1 Compiling as Program

#### Compiling

You start the compiler with the "Chart > Compile > Charts as Program..." menu command or by clicking the button in the toolbar .

In the dialog box, select either "Entire program" or "Changes only" and confirm with "OK".

If errors occur during compilation (or during the consistency check), the "Logs" dialog box is opened automatically at the end of the compilation.

#### Path Name of the Borland C Compiler

If the compilation is not possible because the system cannot find the Borland C compiler, you must inform CFC of the path name used, as follows:

In the "Compile Charts as Program" dialog box, click the "Settings" button. A further dialog box appears in which you enter the path names of executable files, include files, and libraries of the Borland C compiler.

You can also make the settings for the Borland compiler separately before compiling (see Section 4.2, "Customize Compilation").

## 4.2 Customizing Compilation

You can change the settings for compilation in the dialog you call with "Options > Customize > Compilation...". The settings described below apply to all subsequent compilations.

### 4.2.1 Specifying the Software Version

In the "Customize Compilation" dialog box, you can specify the software compatible with your CPU. You display this dialog with the "Settings" button in the "Compile Charts as Program" dialog. Make your selection in the "Version of the M7 system software" drop-down list box.

---

#### Note

There may be more than one version of the M7 system software installed on your programming device/PC. CFC normally uses the latest version. This can lead to conflicts if your M7 CPU is incompatible.

---

### 4.2.2 Optimizing the Run Time

#### Optimizing the Run Time in the Dialog

In the "Customize Compilation" dialog box, you can select the option "Optimize run time for DB access". You display this dialog with the "Settings" button in the "Compile Charts as Program" dialog.

#### Optimizing the Run-Time Using Direct Addressing

If you want to optimize the run-time of interconnections to shared data blocks, instead of using the access used by the M7 code generator (see above), you can also access the shared data using direct addressing. During startup, the absolute address of the data block is queried and an error check is run that detects the following errors:

- The data block used does not exist on the M7 PLC
- The length of the data block will be exceeded

If an error is detected, the task for I/O errors is started. If the error task has not been configured, the M7 PLC changes to STOP.



## Advantages and Disadvantages

With this method, access is faster because no further error checks are made during the run time; in other words, if a DB is deleted or reloaded during the run time, the error is not detected as an I/O error. The error task configured in CFC is no longer called. The program then accesses memory areas (at random) without this being detected.

---

### Note

If you want to reload or delete data blocks during run time, the system **must** be briefly set to STOP to allow the CFC program to process the startup again (address references are updated).

---

## For users of PRO-C:

The call-back mechanism provided by the object server to inform the system of DB access can no longer be used with this method. Since the data are no longer swapped by the object server, when using direct access, only the data saved in S7 format in memory can be correctly processed (for example SKA data blocks). If the task external to CFC creates its own data blocks, make sure that the data are created in the S7 format.

If data blocks are created in other applications and logged on at the object server, make sure that this logon takes place before the CFC program starts up. This is the case with data blocks created by SKA.

## 4.2.3 Setting RMOS Priorities

You call the dialog with the "Options > Customize > Compile..." menu command. In the "RMOS Priorities" tab, you can modify the fixed RMOS task priorities for the individual CFC priorities. By clicking the "Standard" button, you can restore the standard setting at any time.

If you convert a CFC priority to a different RMOS priority, all the tasks that you configure with this CFC priority are converted to the newly selected RMOS task priority.

Before the settings take effect on the CPU, you must compile the program and then install the chart folder (in the "Program" tab) on the PLC using the "PLC > Manage PLC..." menu command.

---

### Note

If you configure the tasks so that they correspond to the defaults in CFC, the RMOS priorities are lower than the system utilities of the M7 software. This prevents an application you have configured from blocking the system due to long run times.

---

In some cases, it may be useful to configure tasks with higher priorities if they are particularly critical but do not involve long run times. This might, be required in the following situations:

- When you require time-controlled tasks with short cycle times (for example  $\leq 10$  ms).
- When you need to react quickly to hardware interrupts.

---

**Note**

If you convert the priorities, make sure that there is enough computing time left so that the system functions are affected as little as possible.

---

## 4.3 Compiling as Block Type

A CFC chart can also be compiled for further use (with block I/Os) as a block type. The procedure is identical to that described for S7.

The dialog box for M7 differs from the S7 dialog box. In the M7 dialog, a file name (.src) is specified instead of an FB number. The source file of the block type is stored in the source folder of the project path under this name.

You can then import the block type into the chart folder. In the block catalog, the block type is displayed under the family name specified for compilation.

## 4.4 Downloading


### Requirements:

Before you can download a CFC program, the "M7-SYS" system software and the CFC run-time system must be installed on the M7 CPU.

### 4.4.1 Downloading the User Program

The entire program is always downloaded.

To download the user program, select the "PLC > Download" menu command or

click the button in the toolbar  .

You can download to the CPU in the "STOP" and "RUN-P" modes. If you download in "RUN-P", the CPU is changed to STOP by CFC. Finally, you must change the CPU back to "RUN-P" with the "PLC > Operating Mode..." function.

If there is already a user program on the PLC, its identification data (name, date/time compiled) are read and compared with the data of the user program you want to download.

You will then be asked if you want to delete the existing program. If you confirm with "OK", the new user program overwrites the existing program.

The CPU is then in the STOP mode and must be restarted (automatic following a prompt).

### 4.4.2 Downloading with "Manage M7 System"

You download the **CFC run-time system** to the M7 CPU with the "PLC > Manage M7 System..." menu command (assuming M7 RMOS32 is already installed). Select the name of the chart folder under "Programming Device" in the "Programs" tab and click the "Install >" button. The chart folder does not need to contain CFC charts.

If compiled CFC charts exist, the run-time system and the CFC program are downloaded.

Following this, the M7 CPU must be restarted.

If you want to download the **CFC program** using this method, remember that the run-time system is also downloaded (unnecessarily) and that the CPU must then be restarted afterwards.



# 5 Test and Commissioning

## Overview

To support you when installing and commissioning a new project, CFC includes test functions allowing you to monitor and, if necessary, modify the values of the block I/Os on the CPU.

## 5.1 General

The CFC editor has two operating modes: The Edit mode and the Test mode. By changing to the Test mode, you have access to the following test functions:

- Monitoring and assigning parameters to block I/Os  
With these functions, you can monitor the development of values online and modify them if necessary.
- Oscilloscope  
An oscilloscope records the sequence of values adopted by one or more block I/Os during the test and displays them graphically as a curve.
- Working with breakpoints  
With breakpoints, you can influence the execution of the user program.

You can also use some of the test functions listed above in the Edit mode; in other words offline. You can, for example, select block I/Os for watching and can set or edit breakpoints.

In addition to the direct test functions, there are functions that help you to prepare for testing and commissioning, for example:

- Setting the test environment (Debug > Test Settings...),
- Displaying module information (PLC > Module Information...),
- Comparing the time stamp of the CPU program (PLC > Compare...),
- Setting the time of day (PLC > Set Time and Date...),
- Additional functions (PLC > Additional Functions...) see Section 5.6.

## Requirements:

To use the test and commissioning functions, the user program created in CFC must first be compiled free of errors and downloaded to the CPU.

### 5.1.1 Activating or Deactivating the Test Mode

#### Preparations

Before you change to the Test mode, you must set the CPU to the "RUN-P" mode with the mode selector.

With the "RUN" setting, an error message is displayed and you cannot change to the Test mode.

The Test mode relates to the CPU belonging to the currently active chart.

#### Activating the Test Mode:

Select the "Debug > Test Mode" menu command or click the button in the toolbar



The Test mode is activated. You can activate the Debug menu functions (including the buttons of the breakpoint bar), most of the functions of the Edit mode become inactive.

#### Notes

- If the user program is modified after the code has been compiled, a message will be displayed when you change to the Test mode.
- If breakpoints or assignments of block I/Os and channels of an active oscilloscope no longer exist due to modifications in the user program, these references are automatically deleted from the CFC internal management.
- Following a reset / cold restart on the CPU, the changes you make in the Test mode are lost since the program is loaded with the old parameter settings.  
Remedy: Compile the CFC chart again and download it.

#### Deactivating the Test Mode

You deactivate the Test mode by selecting the "Debug > Test Mode" menu



command again or clicking the button in the toolbar.

If the CPU is in the HOLD mode, and you deactivate the Test mode, you will be asked whether you want to resume or stop the held program. The Test mode is only completed after the mode has been changed from HOLD to RUN or to STOP. Whichever is the case, all the breakpoints are removed from the CPU.

## Changing Mode

When you change the editor mode, the relevant CPU is accessed in the background.


The following processes are started when you change to the Test mode:

- Each time you change to the Test mode, the dynamic display attributes and the breakpoints stored in the data management are available again. You can activate these by selecting the relevant menu commands.  
The CFC editor checks whether the blocks for the breakpoints were defined, whether they exist in the charts, and whether the block I/Os assigned to the channels of an activated oscilloscope actually exist.
- Following this, the connection to the PLC is opened. If a user program already exists on the PLC, the CFC editor compares its characteristic data (name, version, compilation date/time) with the data of the current user program of the CPU to which the currently active chart belongs. If the programs have the same version, the Test mode is activated.

The status bar indicates whether you are in the Edit or Test mode.

The new operating mode can only be adopted after the background operations have been completed successfully.

## 5.2 Monitoring and Assigning Parameters to Block I/Os

When you activate the Test mode, the "watch on"  function is activated for CFC charts in the "laboratory mode" automatically. This allows you to display the development of the values in the watch list in the Test mode.

You can change the options for the dynamic display and the parameters of the block I/Os in the Test mode.

Monitoring and setting parameters is the same in M7 as in S7. For more information, refer to the manual CFC for S7.

---

### Note

Please note that parameters modified in the Test mode are always included in the CFC data management.

---

## 5.3 Working with the Oscilloscope

The oscilloscope records the sequence of values adopted by one or more block I/Os during the test in the form of a graphic curve. This allows you to check how values develop **during** the running of the program and to adapt parameters if necessary.

An oscilloscope has eight channels. A block I/O can be assigned to each of these channels so that the values of eight block I/Os can be recorded and compared with one oscilloscope.

You can enter or change the parameters for recording the measured values in the dialog box.

Once you have started the recording, the development of each input value (channel) is described in a curve diagram.

### Overview

This section describes the following:

- Creating a new oscilloscope
- Assigning a block I/O to an oscilloscope and deleting the assignment
- Parameters for recording values
- Starting and evaluating a recording
- Printing an oscilloscope or list of oscilloscopes
- Deleting an oscilloscope



### 5.3.1 The "Oscilloscope" Window

You display and work with an oscilloscope in the "Oscilloscope" window.

You open an oscilloscope window with the "Debug > Oscilloscope" menu command or with the button in the breakpoint bar.

The window is either empty or the last selected oscilloscope is opened.

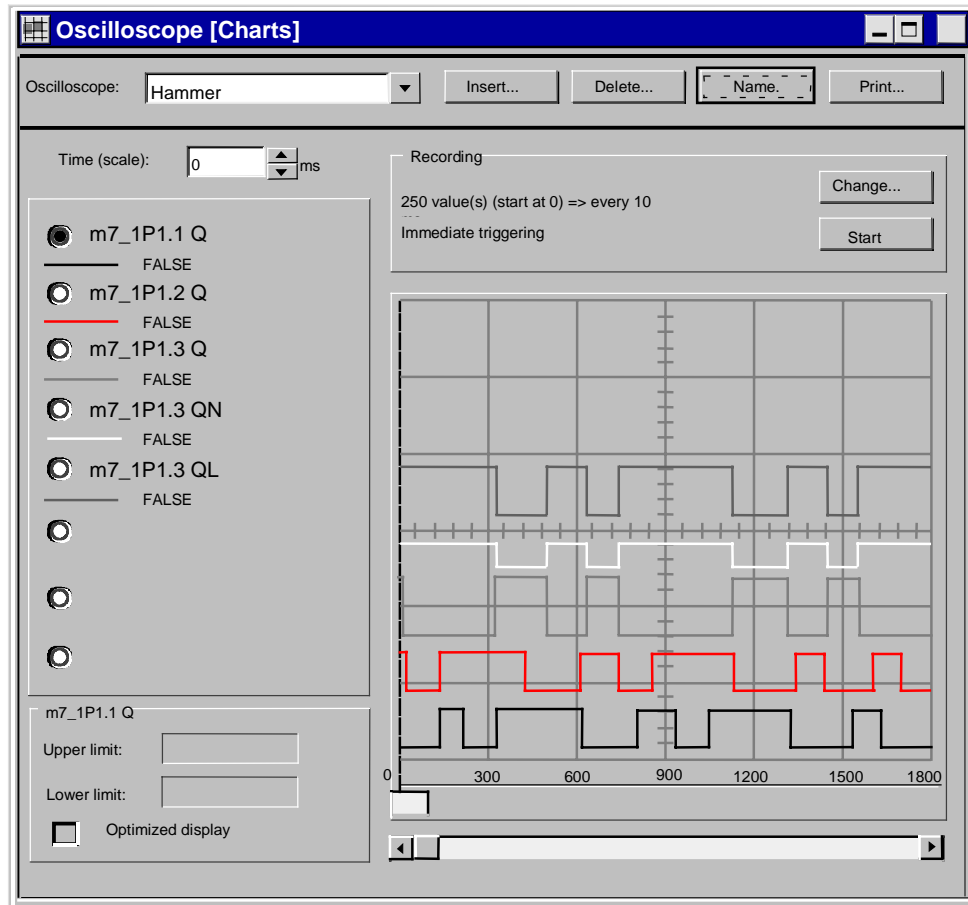


Figure 5-1 The Oscilloscope Window

For a detailed description of the oscilloscope window, refer to the online help.

### 5.3.2 Selecting, Opening, and Renaming an Oscilloscope

You can select oscilloscopes created for this user program in the open window.

#### To open an oscilloscope:

You open an existing oscilloscope by selecting its name in the alphabetical drop-down list box beside the "Oscilloscope:" box.

The oscilloscope is opened.

#### To rename an oscilloscope:

Click "Name ..." and modify the name of the oscilloscope in the dialog box.

With "OK" you return to the "Oscilloscope" window.

### 5.3.3 Creating a New Oscilloscope

To create a new Oscilloscope, follow the steps below:

- Open the oscilloscope window and assign a name:  
With the "Debug > Oscilloscope" menu command or with the function in the breakpoint bar (see Figure 5-2) you open the "Oscilloscope" window. Click "Insert ..." and enter the name of the new oscilloscope in the dialog box (maximum 32 characters).
- Assign a block I/O to a channel  
To measure the values adopted by a block I/O, assign the block I/O to a channel of the oscilloscope. Up to eight block I/Os can be assigned to one oscilloscope (eight channels).
- Specify the parameters for the recording of the measured values  
In the "Recording" box, you can display the "Settings for Recording" dialog box by clicking the "Change..." button. Here, you can enter the following:
  - The number of values to be recorded
  - The beginning and end of the recording for the required task and the time of the cyclic recording
  - The trigger conditions

For more detailed information about these steps, refer to the online help.

### 5.3.4 Starting and Evaluating a Recording

#### Changing Mode

The parameters displayed in the "Oscilloscope" window are transferred to the CPU as a recording job. When the CPU signals that the recording is completed, the CFC editor requests the entire recording. For each recording point, the time stamp of the point and the values of all the block I/Os assigned to the oscilloscope are transferred to the CFC editor.

The oscilloscope processes the values to produce curves. The way in which the values are processed depends on the settings you made when you assigned the I/O and when you made the settings for the recording.

The curve takes the form of steps as with a digital oscilloscope.

#### Starting the Recording

To start recording the measured values, click "Start" in the recording area of the "Oscilloscope" window.

While the data are being recorded, the "Start" button changes to "Stop" and the text "Recording task active" is displayed.

You can delete a recording job by clicking "Cancel".

A message is displayed to indicate the end of the recording. If you confirm with "OK", the oscilloscope with the curve display is opened or, if already opened, is brought to the foreground.

#### Horizontal Scaling of the Curves

The horizontal scale of the x axis is determined by the smallest interval in time between two points of the curve. The CFC editor assigns the smallest graphic resolution of the x axis to this time interval; all further time information is scaled to this basic unit. The physical unit for time information is adapted accordingly.

#### Vertical Scaling of the Curves

The vertical scale of the y curve depends on whether you selected the "Optimized display" option in the "Assign Input/Output" dialog or which "Upper and lower limit" you entered.

If you selected "Optimized display", the vertical scale is calculated from the lowest and highest of the values that was set at the block I/O. If you select the upper and lower limit yourself, the curve will be scaled according to your settings.

## Display of Binary Signals

The curves of binary signals are displayed so that they are assigned a fixed horizontal range in the graphic depending on your channel numbers 1 to 8. The upper limit of this section corresponds to the binary value "1", the lower limit the binary value "0". This fixed assignment of areas avoids the curves overlapping each other. For this reason, you cannot change the limits for the curve display.

## Grids and Lines of the Graphic Area

The graphic display area includes a grid with a coarse and fine division.

The recording point belonging to the y value is displayed below the vertical grid lines of the coarse division. The recording point (time) is determined by the difference compared with the trigger point (see "Entering Parameters for Recording" in Section 5.3.3).

The trigger point (assuming it is in the visible area of the recording) is represented as a red line.

## Scroll Bar

Using the scroll bar below the graphic, you can move the visible area of the recording.

## Time (scale):

The oscilloscope provides a scale to make it easier to read the curves.

Once a graphic is displayed, the scale is automatically positioned at the left grid margin of the graphic in the graphic area representing measured values. In the block I/O/channel display area, it displays the y value of each curve as it is at the current position of the scale on the x axis.

You can move the scale horizontally by dragging the "handle" at the bottom end of the scale to the right or left with the mouse. While it is being moved, the scale jumps to the next curve point.

The exact position of the scale is displayed as the recording time, relative to the trigger time in the upper part of the dialog.

You can modify this value either by entering it directly or by clicking on the up or down arrows.

### 5.3.5 Printing an Oscilloscope

You can print the currently active oscilloscope or a list with information about all the existing oscilloscopes.

#### Printing the Active Oscilloscope (Recording)

To print the oscilloscope, click "Print ..." with the oscilloscope open. In the "Log Output" dialog, select the "Current Oscilloscope" and confirm with "OK".  
The oscilloscope is printed and the dialog box is closed.

#### Printout

The printout takes two pages.

- The first page contains the identification of
  - the oscilloscope (its name, its recording parameters, the current values for the number of points on the curve, the trigger point, and the time on the scale)
  - and its channels (the complete identification of the block I/O (= chart name, block name and I/O), the data type, the display limits, and the value of the channel on the scale).
- The second page contains the actual graphic with the curves of the measured values (including the channel number) and the grid, the time scale, the trigger point, and the scale.

All the curves are printed out in black; on a color printer, the CFC editor uses the default color scheme as used for screen display.

#### Printing a List of Oscilloscopes

You can print a list of oscilloscopes by clicking the "Print ..." button in the open oscilloscope and then selecting "List of Oscilloscopes" in the "Log Output" dialog. When you click "OK", the list of oscilloscopes is printed and the dialog box is closed.

The list contains all the oscilloscopes defined for the active CPU with their names and their recording parameters.

### 5.3.6 Deleting an Oscilloscope

To delete an oscilloscope, you must first open the oscilloscope and then click the "Delete ..." button. The "Delete" dialog is opened. Here select the option beside the name of the oscilloscope and confirm with "OK".

## 5.4 Working with Breakpoints

This section deals with the following topics:

- General information about working with breakpoints (with information about restrictions and special features)
- How to define breakpoints locally ("Breakpoints Inactive") and activate them on the PLC ("Breakpoints Active")
- How to set, delete, enable, and disable breakpoints
- How to edit existing breakpoints

### 5.4.1 General Information on Working with Breakpoints

When you test the user program, you can use two types of breakpoints:

- Temporary breakpoints that are deleted when they are reached and then set implicitly in the next block by the commands:
  - Execute Next Step
  - Execute to Selection
- Fixed breakpoints that are set explicitly

When you set breakpoints, you define a point at which execution of the user program is put on hold. When such a breakpoint is reached, all the values are updated. When the program has reached a breakpoint, you can, for example, execute the program step-by-step to the next block or go to the selected block so that you can identify and monitor the values at the block I/Os in the step-by-step mode.

---

#### Note

If the CPU is in the HOLD mode, and you deactivate the Test mode, you will be asked whether you want to resume or stop the held program. The Test mode is only completed after the mode has been changed from HOLD to RUN or to STOP. In either case, all the breakpoints (including the temporary ones) are removed from the CPU.

---

When you set breakpoints, you can initially define the breakpoints locally and then activate all breakpoints on the PLC at once.

All defined breakpoints are managed in a breakpoint list. In this list, you can edit breakpoints that have already been set. You can, for example, define a "trigger condition" for each breakpoint; in other words, specify the conditions under which it becomes active.

---

#### Note

When working with breakpoints, you must only change the mode of the CPU using the menu commands. If you change the mode with the mode selector, breakpoints will be left over in the program.

---



### Danger

When you work with breakpoints, remember that the user program is halted and that the monitoring functions of the user program are also halted so that physical processes started during the testing and trials of the user program continue to run without being controlled by the program. Remember, also, that programs running parallel to the user program are also halted.

In contrast to using breakpoints, the functionality of the oscilloscope does not suspend the control mechanisms of the user program (for information on the oscilloscope, refer to Section 5.3).

### The breakpoint is not reached

After resuming the program, it is possible that the next breakpoint is not reached. This can occur, for example, when the next breakpoint is in a disabled group or is in an interrupt task when the interrupt is not active.

### Handling

In the Test mode, you can display a further toolbar known as the "Breakpoint Bar" with "View > Breakpoint Bar". This makes it more convenient to work with breakpoints and the oscilloscope.

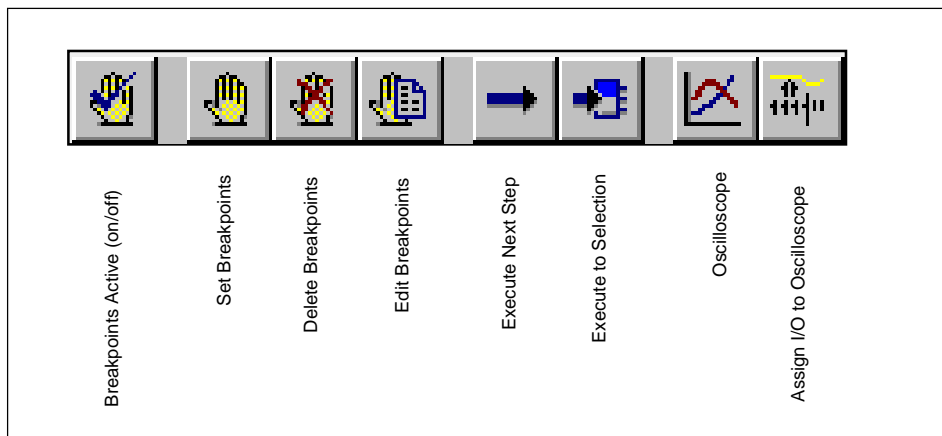


Figure 5-2 Meaning of the Buttons in the Breakpoint Bar

## 5.4.2 Activating Breakpoints on the PLC

### Procedure

When working with breakpoints, it is advisable to proceed in stages:

- First prepare for a test session by defining the breakpoints locally ("Breakpoint Inactive" = default).

In this way, the breakpoints that you set (including their status) are only included in the internal CFC data management.

- Following this, activate all the breakpoints on the PLC ("Breakpoints Active") immediately and at the same time.

Breakpoints that you do not want activating can first be disabled.

### Statuses of Breakpoints

A breakpoint can have two different statuses:

- The breakpoint is "enabled" ; in other words, it is effective as a breakpoint.
- The breakpoint is "disabled" ; in other words, the definition of the breakpoint is retained however the breakpoint itself is not effective.

(See also Section 5.4.3.)

### "Breakpoints Active" on/off

After you activate the Test mode "Breakpoints Inactive" is the default.

Each time you click the "Debug > Breakpoints Active" menu command or the button in the breakpoint bar, you toggle the active state (on or off).

**Active on:** All existing enabled breakpoints in the internal CFC data management are effective on the CPU. Each new breakpoint that you set becomes active immediately on the CPU.

**Active off:** All the breakpoints are deleted on the CPU.



### 5.4.3 Setting, Deleting, Enabling, Disabling Breakpoints

**Note on the following description:** In the "Debug" menu, you will find the functions "Set", "Delete", "Enable", and "Disable" in the "Breakpoint" submenu. Since the procedures for these four functions are very similar, the use of "Set Breakpoint" is described in detail and is used as a basis for the description of the other functions.

---

#### Note

Before you start these functions, please remember that they can also have effects on the CPU depending on whether you have set the breakpoints to "active" or "inactive" (see also Section 5.4.2).

Setting, deleting, enabling, and disabling breakpoints in the "RUN" mode causes an error message. To use the functions, the CPU must be in the "STOP" or "RUN-P" mode.

---

### Set Breakpoints

Select the block in which you want to set a breakpoint and select the "Debug > Breakpoint > Set" menu command or click the button in the breakpoint bar (see Figure 5-2).

- If the block you have selected is installed in only one task, the breakpoint is included in the CFC data management and the breakpoint list and the block is displayed in color (yellow "H" on a red background).
- If the block you have selected is installed in more than one task, a list of all the tasks containing the block is displayed. You can set a breakpoint for the block in each of these tasks by selecting the required tasks (Ctrl + left mouse button) and then clicking "OK".  
The breakpoint or breakpoints are included in the CFC data management and in the breakpoint list and the blocks are displayed in color (yellow "H" or "+" on a red background).

Breakpoints set in this way have the default "enabled" and "always"; in other words, the user program stops every time at this breakpoint (for more detailed information and information on changing this default, see Section 5.4.4 "Editing Breakpoints").

### Disabling a Breakpoint:

You disable a breakpoint by selecting a block and the "Debug > Breakpoint > Disable" menu command or the button in the breakpoint bar.

- If the selected block has only one breakpoint, the status of the breakpoint is set to "disabled" in the breakpoint list.
- If the selected block has more than one breakpoint, a list with the data of each breakpoint belonging to this block is displayed. You can select one or more entries and disable them or disable "all breakpoints".  
Following this, the breakpoint or breakpoints are set to "disabled" in the breakpoint list.

The disabled breakpoint or breakpoints can be set to the "enabled" status again.

### Enabling a Breakpoint

You enable a disabled breakpoint by selecting a block and then selecting the "Debug > Breakpoint > Enable" menu command or the button in the breakpoint bar. The remaining steps are identical to "Disable Breakpoint".

### Delete Breakpoints

You delete a breakpoint by selecting a block and then selecting the "Debug > Breakpoint > Delete" menu command or the button in the breakpoint bar. The remaining steps are identical to "Disable Breakpoint".

## 5.4.4 Editing Breakpoints

Breakpoints that you set with "Debug > Breakpoint > Set" (or in the breakpoint bar) are effective as breakpoints each time the test is run since they automatically have the default "always valid".

You can change this default in the "Breakpoints" dialog by selecting the "Debug > Edit Breakpoints" menu command or clicking the button in the breakpoint bar. This function can only be activated when a breakpoint has already been set for a block.

The "Breakpoints" dialog box contains a list of breakpoints, the buttons for general functions, and the section where you enter the trigger condition.

## Procedure

Enter the parameters as follows:

1. Enter the first address.  
A list is displayed with all the inputs and outputs of the selected block in which you can select the required I/O.
2. Then enter the relational operator.  
Relational operators are normally: equality (=), unequality (<>), greater than (>), less than (<), greater than or equal to (>=), less than or equal to (<=).  
  
You can select a relational operator from the list displayed (only those suitable for the first address can be selected). If, for example, the block I/O entered as the first address is of the type "BOOL", only the relational operators "equality" and "unequality" are displayed.
3. Following this, you enter the second address.  
  
You can enter this address directly. You can either enter an input or output of the block suitable for the first block I/O or a constant.  
  
Alternatively, you select the second address from the list that displays the possible values depending on the first address.  
If, for example, the block I/O entered as the first address has the data type "BOOL", then only the Boolean I/Os of the block and the values "TRUE" and "FALSE" are available.
4. Below the input box of the first address, you can specify in which run the breakpoint becomes active.  
  
If you also specify a condition for the breakpoint at the same time, the breakpoint becomes active each nth time the condition is satisfied (the condition and nth run are ANDed).
5. When you click "Apply", the modified condition is included in the breakpoint list.

## 5.5 Holding and Resuming the CFC Program

### 5.5.1 Holding the Program

If you want to stop the CFC program **immediately**, select the "Debug > Hold" menu command.

The box with the run-time properties of the currently active block is highlighted on a red background. If the block is not in the current sheet view of the chart, the relevant sheet is opened.

In the status line, the information "HOLD (command)" is displayed.



#### **Danger**

Remember that the Hold function stops the entire CPU!

---

If the message HOLD without any further explanation (command, step, or breakpoint) is displayed in the status bar, the reason for the HOLD is outside the program created by the CFC editor, for example in a parallel application. You could then (if required set a breakpoint in a block) continue the program on the CPU with the "Debug > Continue" menu command.

### 5.5.2 Resuming the Program

In the "Debug" menu, there are three different ways of continuing the Test mode:

- **Execute Next Step** (also in the breakpoint bar).
- **Execute to Selection** (also in the breakpoint bar).
- **Continue**

These different options allow precise, step-by-step monitoring of the values at the relevant block I/Os.

The "**Execute Next Step**" and "**Continue**" functions can be selected when the system is in the HOLD mode due to a command, step, or breakpoint (check the status bar).

#### **Execute Next Step**

You can move on to the next block by selecting the "Debug > Execute Next Step" function (or click the button in the breakpoint bar).

The program stops at the next block and its run-time properties box is displayed with a red background. The information "HOLD (step)" is displayed in the status bar.

If this block is not in the currently active sheet of the chart, the relevant sheet view is opened.

The "next" block is determined as follows:

- Starting from the block at which the program has stopped, the next block is the block in the same task that is next according to the run sequence.
- If the "next step" leads to a run-time group, the Enable attribute and the scan rate of this run-time group are taken into account when determining the next valid block.
- Starting from the last block of a task, the program is halted at the first block of the same task after a prompt for confirmation has been displayed.

### **Execute to Selection**

If a block is selected and you start the "Execute to Selection" function, the program stops at the selected block and its run-time properties box is shown on a red background. "HOLD (step)" is displayed in the status bar.

If a block including a breakpoint is located between the starting block and the selected block, the program stops at the breakpoint.

### **Continue:**

If you select the "Debug > Continue" menu command, the user program is continued. The message "RUNNING" is displayed in the status bar. If there is a block with a set breakpoint, the program is stopped there if the condition is satisfied. In this case the message in the status bar changes to "HOLD (breakpoint)".

## 5.6 Additional Functions

With the additional functions, you can read out further detailed information and evaluate it during the test.

With the "PLC > Additional Functions..." menu command, you display a dialog box in which you can display the following information using the buttons:

- The **CFC system status** of the CPU assigned to the user program currently being edited.
- The current **CFC interrupt stack** that the CFC reads from the PLC.
- The **status of CFC tasks**.

### 5.6.1 CFC System Status:

After selecting the "PLC > Additional Functions..." menu command in the Test mode and clicking the "Status of CFC Tasks" button, the dialog with the same name is opened.

The CFC system status display provides you with information about all the tasks in the chart and the actual execution of the program on the CPU including any errors that occurred. All time information is displayed by the PLC in microseconds.

The dialog box displays the current status of the user program (for example HOLD, RUNNING) and the memory utilization of the available memory (in %).

The list shows all the tasks in the chart. The level, its current priority, the size of the level in bytes, the number of starts of this task since the last CPU start, and the current, minimum, and maximum level run time are displayed for each element in the list.

The list of diagnostic information can contain up to 5000 entries in ascending chronological order. Starting from the current event, you can follow the "history" of an error. The following information is displayed for each diagnostic element:

- The time stamp (calculated from the last CPU start)
- The event:
  - The start and end of tasks
  - Program errors (for example timeout, time delay error, request error, or request overflow)
- The name of the task
- The priority at the time the task started or ended

For more details on the additional information, refer to the CFC online help.

### 5.6.2 CFC Interrupt Stack:

After selecting the "PLC > Additional Functions..." menu command and clicking the "CFC Interrupt Stack" button, the dialog box of the same name is opened.

Note: The information in the interrupt stack can only be read when the program is halted within a CFC block.

This provides you with information about changes in the operating mode of the system to the HOLD mode (for example due to a breakpoint) or STOP (for example due to serious errors in program execution). This time information is supplied and displayed accurate to microseconds.

The list contains all the elements of the interrupt stack in descending chronological order. The entry for each stack element includes the name of the task and the identification (identifier of the run-time group, block name) of the block at which the task was interrupted.

When you select a stack element in this list, further information is displayed below the "Find", "Go To", and "Print" buttons.

For more details on the additional information, refer to the CFC online help.

### 5.6.3 Status of CFC Tasks

After selecting the "PLC > Additional Functions..." menu command in the Test mode and clicking the "Status of CFC Tasks" button, the dialog with the same name is opened.

All the defined tasks are listed. The CFC editor reads the name, status, and priority of each task from the PLC. If a task is enabled, its current priority is displayed, if it is disabled the configured priority is displayed.

You can edit the status and the current priority of the task selected in the list. You can change the priority of interrupt and error tasks. You can assign the priority 2 to 24.

With the "Apply" button, you can enter the changes on the CPU (not in the CFC data management); with the "Initial Values" button, you can discard the changes and return to the initial values from the CFC data management.

For more details on the additional information, refer to the CFC online help.





## 6 Creating Block Types

### Overview

A CFC chart can be compiled as a block type. You compile a block type in the CFC editor and the procedure is described in the manual CFC for S7.

This chapter contains information about creating your own block types for the M7 CPU. Knowledge of programming with C is assumed.

### 6.1 General

The following sections deal with the topics listed below:

- Creating block types
- Rules for C code and examples
- Testing block types

### Changing a Block Type

If block types that have already been used in CFC (block instances exist) are changed later, this leads to inconsistencies in the CFC configuration. These block types must be imported again.

## 6.2 Creating Block Types

To create a block type, you edit an ASCII file (without control characters) with the extension \*.SRC. The templates shipped with the product will help you. The templates are in the CFC installation folder under **M7BLOCKS** in **EXAMPLE**.

You can use any ASCII editor, however the NOTEPAD.EXE shipped with WINDOWS or the development environment of the Borland Compiler is recommended. You must save the file in text format.

You can select any name for the source file, however this should relate to the block type described in the file. A maximum of 8 characters are available for the name; the extension is ".src".

### 6.2.1 The Source File

All the entries must comply with valid C syntax. Preceding blanks are ignored.

The keywords must be in upper case letters. Where entries are mandatory, (for example with data types), these must also be written in upper case letters. Exception: Certain optional keywords can also be written with the S7 prefix.

You can enter comments within the source file in the following form:

`/*<comment text>*/` or `//<comment text>`

### Structure of Source Files

```
FUNCTION or FUNCTION_BLOCK
{
COMMENT:='<block comment>';
BLOCKVIEW:='<view of the block type>';
GENERIC:='<generic property of the block type>';
TASKLIST:='<default insert position>';
};
NAME:<block type name>;
FAMILY:<block type family>;
VAR_INPUT
<description of the inputs>
END_VAR
VAR_OUTPUT
<description of the outputs>
END_VAR
VAR
<static variables>
END_VAR
BEGIN
{
INCLUDES:='<include files>';
CODELIBS:='<library names>';
}
<code area>
END_FUNCTION or END_FUNCTION_BLOCK
```

## 6.3 The Keywords

The block type is determined by the entries made for the keywords described below. Unless otherwise specified, the keywords must exist in the source file in the order described below and must not be used as input or output names in the C code.

The following keywords in the C source file must always be specified:

- FUNCTION or FUNCTION\_BLOCK,
- NAME,
- BEGIN,
- END\_FUNCTION or END\_FUNCTION\_BLOCK

The following keywords are optional:

- COMMENT,
- BLOCKVIEW,
- TASKLIST
- GENERIC,
- FAMILY,
- VAR\_INPUT,
- VAR\_OUTPUT,
- VAR,
- INCLUDES,
- CODELIBS,

If you use an optional keyword, you must also make a valid entry after the keyword.

Since a data block must have at least one input or one output, you must either use the keyword VAR\_INPUT or VAR\_OUTPUT (or both).

Following the entries after the keywords VAR\_INPUT, VAR\_OUTPUT and VAR, you must always use the END\_VAR keyword.

- The conventions for the system attributes TASKLIST, UNIT, STRING\_0, STRING\_1, LINK, and DYNAMIC are as follows:
  - In upper case letters, for example *LINK:=TRUE*
  - In lower case letters in conjunction with the prefix "S7\_", for example *S7\_link:=TRUE*

### 6.3.1 Defaults for the Optional Keywords

If you do not specify an optional keyword, the following defaults are used :

Table 6-1: Defaults for Optional Keywords (Block)

Keyword	Default
COMMENT	no comment
BLOCKVIEW	BIG = big view
TASKLIST	no installation at default point
GENERIC	NONE = no generic property
FAMILY	no block family
VAR_INPUT	no inputs
VAR_OUTPUT	no outputs
VAR	no static variables
INCLUDES	no include files
CODELIBS	no libraries

Table 6-2: Defaults for Optional Keywords (Inputs/Outputs)

Keyword	Default
COMMENT	no comment
VISIBLE	TRUE = visible
UNIT	empty string
STRING_0	empty string
STRING_1	empty string
LINK	TRUE = interconnectable
DYNAMIC	FALSE = not updated in dynamic display
PARAM	TRUE = can be assigned parameters

### 6.3.2 Description of the Keywords

#### Category of Block Type

Entry: FUNCTION\_BLOCK or FUNCTION

A FUNCTION\_BLOCK is a block with memory (with an instance DB). Following each call cycle, it saves values that can be used for the next call cycle.

A FUNCTION cannot save values, but is therefore faster than a FUNCTION\_BLOCK. With a FUNCTION, the keywords VAR and END\_VAR must not be used.

#### Block Type Name

Entry: NAME: <block type name>;

Example: NAME: PID\_controller;

The name of a block type can be 1 to 24 characters long and can be assigned freely to suit your purposes, it must however be unique on the CPU in which the block type will be used.

#### Begin Identifier

Entry: BEGIN

Indicates the beginning of the code area and the block description.

#### End Identifier

Entry: END\_FUNCTION or END\_FUNCTION\_BLOCK

Indicates the end of the code area and the block description.

#### Block Type Comment

Entry: COMMENT:= '<block comment>;'

Example: COMMENT:= 'PID\_controller';

The comment can be a maximum of 80 characters long. Blanks are permitted, however a single quotation mark or brace is not permitted.

## Block View

Entry: BLOCKVIEW:=<view of the block>;

Example: BLOCKVIEW:=BIG;

With BLOCKVIEW:=BIG or BLOCKVIEW:=SMALL, CFC recognizes whether a block will be displayed in the large or small format.

## Multiple Installation

Entry: TASKLIST:=<'taskname1,taskname2,...'>;

With TASKLIST, CFC recognizes whether or not a block will be installed in more than the current task. The list of tasks can be seen in the attribute.

**Note** For TASKLIST you enter the symbolic name of the task as the value. The block attribute should then contain for example, "restart" for the restart task.

If tasks are specified in the task list of the block, and they do not exist in the chart folder when the block is imported, this is indicated in the block type but it is installed only in existing tasks. If the tasks that have been missing up to now are defined later, this has no effect on blocks that have already been imported; these are not installed afterwards in the tasks. All newly imported blocks are installed in these tasks.

## Generic Property

Entry:        `GENERIC:= <generic property of the block type>;`

Example:    `GENERIC:= SINGLE;`

This specifies whether or not the block type has a variable number of inputs. If the generic property is `SINGLE`, you can modify the number of inputs in the CFC editor; in other words every block instance can have inputs added and can be reduced to the default number. The inputs all have the same data type, for example `AND` element, `OR` element, totalizer.

If you do not want the block to have a variable number of inputs, enter the value `NONE` after the keyword.

Remember the following points about blocks with the generic property `SINGLE`:

- Only block types of the "FUNCTION" category can have the generic property.
- Block types with the generic property can only be created if their algorithm can be created by simple code doubling. For example, no `MIN` or `MAX` blocks could be created with this method.
- All the inputs have the same data type. This means that the blocks also have no `EN/ENO` parameter.  
The names must be `IN1`, `IN2` etc.
- The default for the number of inputs is set by declaring this number of inputs (minimum: 2 inputs). The first input decides the data type, comment, default value, and the visible/invisible attribute and is used as a basis for the parameters that follow. The parameters that follow the first input are irrelevant, they are simply used to determine the default number of inputs.
- The maximum value for inputs is 150.

## Block Family

Entry:        `FAMILY: <family name>;`

Example:    `FAMILY: Controllers;`

The name of a block family can be from 1 to 8 characters. It is used to group block types within CFC.

The text can contain blanks and special characters.

## Description of the Inputs/Outputs

Entry:        `VAR_INPUT or VAR_OUTPUT`

              <input or output description>

`END_VAR`

You must first define all the inputs (with `VAR_INPUT`) and then all the outputs (with `VAR_OUTPUT`). Enter a description for each input or output. Each description must start on a new line.



The order of the inputs or outputs in this description corresponds to the display in CFC. The number of visible inputs or outputs per block type must not exceed 160 (maximum that can be represented in CFC).

The input or output description has the following format:

```
<Name of the input/output> {COMMENT:= '<comment>'; VISIBLE:= <visibility
status>;}: <data type> := <default>;
```

Example:

```
VAR_INPUT
TN {COMMENT:= 'reset time'; VISIBLE:= TRUE;}: WORD := 5;
END_VAR
```

The keywords COMMENT and VISIBLE are optional. If you specify the keywords, you must always enter them in the specified order. If you do not specify an optional keyword, the following defaults are used : see Tables 6-1 and 6-2.

### **Name of the Input/Output**

Entry:      <name of the input/output>

The name of the input/output can be 1 to 24 characters long. In the CFC chart, only the first 8 characters are displayed in the block. Within the description of a block type, each input/output name must only be used once (there is no distinction made between upper and lower case characters). The name must also be assigned according to C conventions.

If the block has the generic property, the inputs must have the names IN1, IN2, IN3 etc.

### **Data Type of the Input/Output**

Entry:      <data type>

The data type of the input/output is specified using one of the following data type keywords:

BOOL, INT, DINT, REAL, BYTE, WORD, DWORD, TIME, TASK (for controlling the task), STRING, STRUCT

See Appendix A.2.

The STRUCT data type is a complex data type that can be made up of several elements.

The description of the individual structure elements is analogous to the description of the inputs and outputs except that no keywords may be used.

```
Entry:      STRUCT
            <structure elements>
            END_STRUCT
```

You can also use structures themselves as structure elements. This is possible up to a nesting depth of 8.

Example:

```
var1 : STRUCT
  value1 : WORD;
  var2 : STRUCT
    value2 : BYTE := 5;
    value3 : BOOL := TRUE;
  END_STRUCT;
END_STRUCT;
```

### Default Value of Input/Output

Entry:      <default value>

Each input/output can have a default value. The permitted entries depend on the data type of the input/output.

The STRUCT data type cannot have a default. You can, however, enter defaults for the individual elements (elementary data types).

Specifying a default value is optional. If you do not specify a default value, the following values are entered:

Default value	Data type
0	BOOL, INT, DINT, REAL, BYTE, WORD, DWORD, TIME
" "	STRING

### Static Variables

Entry:      VAR  
            <static variables>  
            END\_VAR

The entry for static variables has the following format:

```
<variablename1>:<datatype1>;
<variablename2>:<datatype2>;
```

Example:

```
VAR
ParamX : WORD;
ParamY : WORD;
END_VAR
```

Each entry must begin in a new line. For valid entries for the data type, refer to "Data Type of the Input/Output".

Static variables can only be defined when the block is a FUNCTION\_BLOCK.

To avoid conflicts with variables assigned by the system, static variables must not begin with the underscore character. The name of a static variable must not be identical to the name of an input/output.

**Note:** Static variables can have a default.

## C code

Entry:      <code area>

The functionality of the block is defined in the code area. The code specified here is interpreted by the compiler.

Only RMOS and M7 API calls can be used in the code area. DOS system calls (svc.h or dos.h) must not be used.

## References to Include Files

Entry:      INCLUDES :=<include files>

The entry for the include files (header files) has the following format:

<include file1>, <include file2>, etc.;

Example: INCLUDES := dtypen.h, param.h;

Here, you can specify include files with externally defined functions and data that are required to compile the application. The length of the include file name is a maximum of 8 characters. The file extension is ".h".

The individual entries do not have any path information. Entries are searched for in the following order:

1. In the M7BLOCKS\INC folder of the project
2. In the M7BLOCKS\INC folder of the CFC installation
3. In the M7 system software

## References to Code Libraries

Entry:      CODELIBS :=<library names>

The entry for the library names has the following format:

<library\_name1>, <library\_name2>, etc.;

Example: CODELIBS := Controller.lib, Arithm.lib;

If a block type uses functions that are already compiled, you must specify which libraries must be linked. This makes it possible to call ready-made functions from within the C code.

You can specify several library names. The length of the file name is a maximum of 8 characters. The file extension ".lib" is used.

The individual entries, do not have any path information. Entries are searched for in the following order:

1. In the M7BLOCKS\INC folder of the project
2. In the M7BLOCKS\INC folder of the CFC installation
3. In the M7 system software

### Comment for the Input/Output

Entry:      `COMMENT:= '<comment>';`

The comment can be a maximum of 80 characters long. Blanks are permitted in the comments, however a single quotation mark or brace is not permitted. If you require a quotation mark in the comment, you must type in two quotation marks one directly after the other.

### Visibility of the Input/Output

Entry:      `VISIBLE: = <TRUE / FALSE>;`

Example: `VISIBLE:=TRUE;`

With this attribute, CFC recognizes whether the block I/O should be displayed or not.

The attribute can be modified in individual instances. If a change is made to the block type, this is not made in the block instances.

### Unit for Process Values

Entry:      `UNIT := '<text>';`

Example: `UNIT:= 'Liters';`

To display the unit of process values, you can specify up to 16 characters of text that is assigned to the block I/O.

Not for the BOOL data type

The attribute can be modified in individual instances. If a change is made to the block type, this is not made in the block instances.

### Operator text 0

Entry:      `STRING_0 := '<text>';`

Example: `STRING_0 := 'ON';`

The operator text 0 that can be up to 16 characters long is assigned to the block I/O. In CFC, the attribute is used as the value identifier and, for example for displaying statuses. This is only possible with the BOOL data type.

The attribute can be modified in individual instances. If a change is made to the block type, this is not made in the block instances.

**Operator text 1**

Entry:     STRING\_1 := '<text>'

Example: STRING\_1 := 'OFF';

The operator text 1 that can be up to 16 characters long is assigned to the block I/O. In CFC, the attribute is used as the value identifier and, for example for displaying statuses. This is only possible with the BOOL data type.

The attribute can be modified in individual instances. If a change is made to the block type, this is not made in the block instances.

**Interconnectability**

Entry:     LINK := <TRUE / FALSE>

Example: LINK:=TRUE;

With this attribute, CFC recognizes whether the block I/O can be interconnected or not.

This attribute cannot be modified in individual instances. If a change is made in the block type, this is also made in the block instances. If the attribute is changed to 'FALSE', existing interconnections are deleted.

**Capable of Dynamic Display**

Entry:     DYNAMIC := <TRUE / FALSE>;

Example: DYNAMIC:=FALSE;

With this attribute, CFC recognizes whether the block I/O can be dynamically updated or not during test functions.

The attribute can be modified in individual instances. If a change is made to the block type, this is not made in the block instances.

**Parameter Assignment**

Entry:     PARAM := <TRUE / FALSE>;

Example: PARAM := TRUE;

With this attribute, CFC recognizes whether the block I/O can have parameters assigned or not.

This attribute cannot be modified in individual instances. If a change is made in the block type, this is also made in the block instances.

If the setting is changed to "FALSE", the existing parameter assignment is retained, the I/O cannot, however, have new parameters assigned.

### 6.3.3 Rules for C Code and Examples

The following rules are important when implementing your program:

- To avoid collisions with variables assigned by the system, variable names must not begin with a single or double underscore.
- Inputs/outputs and static variables of the block must be addressed in the code using \$<input/output\_name> or. \$<static\_variable\_name> (for example \$TN for the input defined in "name of the input/output").
- When using "static" variables, remember that when a block is installed more than once (in different tasks) this can have extremely unpredictable side effects.

Example:

```
....
{
    static BOOL Membit_critical           // local variable created
                                         // once per installation

    #if_INIT}
    {
        static BOOL Membit_uncritical    // validity of the
                                         // variable is unique
    }
    #else
    }
END_FUNCTION
```

For variables to be used in different tasks, you can use the static variables in the FUNCTION\_BLOCK (between VAR and END\_VAR).

- In the block code, local variables can be defined that are only known within the block.

Local variables must always be declared within parentheses. Since the code compiler can include the body of a block several times in the C code, error messages would otherwise result due to multiple definitions.

Example:

```
....
BEGIN
{
    INCLUDES:=...
}
{
    BOOL    Memory bit; // local variable
    ...
}          // <- end of parenthesis
END_FUNCTION
```

## Structure Elements

To access structure elements in the block code, specify the top level (as for other I/Os) using \$<I/O\_name> an. You can access structure elements using the element names as follows:

```
$IN1.ele = 5;           // first level of the structure
$IN1.str1.ele1 += 1;    // second level of the structure
memset (&$IN1.str1, ...); // address of the structure
```

## Multiple Installation

CFC supports so-called multiple installation, in other words a block instance can be installed in several tasks. By evaluating defines, specific code can be executed (for example hardware interrupt evaluation):

A block instance must only be installed once per task type.

```
#if __INIT
<Code>    // this section is executed during startup
#elif __TIMED
<Code>    // this section is executed in a time-controlled task
#elif __PROG_ERR
<Code>    // this section is executed if a program error occurs
#endif
```

## Defines

You can specify the following defines when creating a block type:

__INIT	Startup task
__ALARM	Hardware interrupt
__TIMED	Time-controlled task
__EVENT	Software interrupt
__FREE_CYCLE	Free cycle
__TIME_ERR	Timeout
__ACCESS_ERR	Access error
__PROG_ERR	Programming error

## Time Constant

For each run-time group assigned to a time-controlled task, and for each task, a time constant `__T0` is created that specifies the time interval at which the run-time group/task is executed. The time constant can be used in the block (for example for calculating time values in the execution cycles).

This, however, ensures that the time constant is not used in other run-time groups when the installation point is modified, and you must restrict the range of the code that uses the constant to the time-controlled tasks as in multiple installation, as follows:

```
#if __TIMED
<Code>    // this section is executed in a time-controlled task
#endif    // the constant __T0 must only be used here.
```

If you use the time constant in run-time groups or tasks that are not time-controlled, an error message is output when you compile the code.

## Notes on C Libraries

When you create C libraries for M7 blocks, remember the following points:

- The library must be a static 32-bit library for an EXE (Borland C: Target option).
- Only `*.C` files must be used for the libraries (no `*.CPP`).
- The following project option must be set:
  - Compiler > Compiler Output > Generate Underscores
  - Compiler > Floating-point: all options deactivated
- No function calls from the Borland library must be used in the library.
- When using data types, the types of the shipped `DTYP.H` file should be used. This avoids conflicts with the M7 run-time system. The file is located in the `...\S7CFC\M7LZS` folder and can be copied from this folder and included in the files of the library with `#INCLUDE`.
- If you use several libraries, make sure that a function name is not used twice: for example `void XSUM (init,int)` in library A and `short XSUM (short,short,short)` in library B. In this case, the type import functions as long as the two libraries are used by different blocks, the code generator, however links all the libraries and would signal an error.



**Blocks with GENERIC: SINGLE**

For blocks with a variable number of block inputs, parts of the code must be included several times. With the GENERIC SINGLE property, part of a line is replaced several times. The replace index (the index of the input from which the multiple replacement is started) can be selected.

There are two keywords that define the replacement range. The replacement index must be specified with the \$\$B keyword and must not be higher than the number of inputs of the block type (without duplication).

Area begin within one line    \$\$B(<Index>)

Area end within one line    \$\$E

There is also the following additional keyword:

Input to be replaced several times    \$\$IN\$\$IN

There are blocks with a variable number of inputs in which the replacement section is longer than one line.

The replacement index for code doubling can be higher than the default for the number of block inputs. This means that code doubling only takes place when the number of inputs is higher than the replacement index.

### Example of GENERIC SINGLE

```
VAR_INPUT
  IN1:WORD;
  IN2:WORD;
END_VAR
VAR_OUTPUT
  OUT:WORD;
END_VAR
```

When the code line  $\$OUT = \$IN1 \ \$\$B(2) + \$\$IN \ \$\$E;$  is compiled without multiplication of the inputs, it becomes  $\$OUT1 = \$IN1 + \$IN2;$

When the code line  $\$OUT1 = \$\$B(2) \ \$\$IN + \$\$E \ 9;$  is compiled without multiplication of the inputs, it becomes  $\$OUT1 = \$IN2 + 9;$

If the number of inputs is increased, for example to 5, the following line results from the code generation:  $\$OUT1 = \$IN2 + \$IN3 + \$IN4 + \$IN5 + 9;$

If  $\$B(1)$  is specified instead of  $\$B(2)$  in the code line above, the following line results:

$\$OUT1 = \$IN1 + \$IN2 + \$IN3 + \$IN4 + \$IN5 + 9;$

If  $\$B(4)$  is specified instead of  $\$B(2)$  in the code line above and the block is only three inputs long, the following line results:  $\$OUT = 9;$

If  $\$B(4)$  is specified instead of  $\$B(2)$  in the code line above and the block is five inputs long, the following line results:

$\$OUT = \$IN4 + \$IN5 + 9;$

With IN1 to IN3, individual operations can be executed outside the replacement range.

It is important that the relevant math operations (as for example here +) are included in the replacement range, otherwise the multiple replacement of the calculation formula is incorrect.

**Example 1 (mandatory keywords only)**

In this example, there are no optional keywords.

```

/*****
  ADDER
*****/
//Created by: Mr. T. Smith
//Date: 20.06.95
//Version 1.0

FUNCTION;
NAME : Adder;

VAR_INPUT
I1 : WORD;
I2 : WORD;
END_VAR

VAR_OUTPUT
Q1 : WORD;
END_VAR

BEGIN
  $Q1 = $I1+$I2;
END_FUNCTION

```

**Example 2 (with optional keywords)**

In this example, optional keywords have been used. You will find more examples in the EXAMPLE folder of the CFC installation.

```

/*****
      DIFFERENTIATOR
X      = input variable
TV     = deriv. action time in ms
EN     = enable
Y1     = output variable
*****/

```

//Created by: Mr. T. Smith

//Date: 07.06.95

//Version 1.0

FUNCTION\_BLOCK

```
{
COMMENT      :=      'Differentiator';
BLOCKVIEW    :=      BIG;    // large view
GENERIC :=      NONE; // not generic
};
NAME      :      DIVRE;
FAMILY    :      Controller;
```

VAR\_INPUT

```
X {COMMENT:= 'input variable'; VISIBLE:= TRUE;} : REAL := 0;
TV {COMMENT:= 'deriv. action time in ms'; VISIBLE:= TRUE;} : WORD := 1;
EN {COMMENT:= 'enable'; VISIBLE:= TRUE;} :      BOOL := 0;
END_VAR
```

VAR\_OUTPUT

```
Y1 {COMMENT:= 'output variable'; VISIBLE:= TRUE;} : REAL := 0;
END_VAR
```

VAR

```
XA : REAL;
END_VAR
```

BEGIN

```
{
INCLUDES:= math.h, arithm.h;
CODELIBS:= controller.lib;
}
```

```
if ($EN == 0)
{
/* no enable */
$Y1 = 0;
}
else
{
/* enabled */
$Y1 = Diff_Calc($TV, $X, $XA); /* calculate actuating signal */
/* The Diff_Calc is from the controller.lib*/
}
$XA = $X; /* X(n-1) save */
END_FUNCTION_BLOCK
```

### **6.3.4 Testing Block Types**

#### **When Importing**

When importing, note any error messages (syntax check or block type source file and test compilation) from the compiler (see 2.3).

#### **In the Test Mode**

After inserting the block and then compiling and downloading it to the CPU (see Section 4.1 and 4.4), you can test it in the Test mode (see Section 5.2). Since debugging the block with fine granularity (line by line in the source code) is not possible, it is advisable in debugging to write interim results to temporarily added block outputs and to view them in the test mode of CFC.

#### **Complex Blocks**

In complex blocks, it is advisable to create and test the C code first with the Borland development system and then to transfer it to the block type source file.



# **A Technical Specifications**

## **A.1 Technical Specifications**

### **Hardware Requirements**

- SIMATIC PG or PC with:
- Pentium Processor
- RAM minimum 64 Mbytes  
(recommended 128 Mbytes or more  
and 256 KB secondary cache)
- Free hard disk space minimum 200 Mbytes less RAM
- Graphics card VGA 640 x 480  
(recommended: SVGA 1024 x 768 or higher)
- MPI connection for online operation
- SIMATIC M7-300, M7-400

### **Software Requirements**

- Microsoft Windows 95  
or
- Microsoft Windows 98  
or
- Microsoft Windows NT  
(with Service Pack 3)
- STEP 7
- M7-SYS
- Borland C (V5.01 or higher)

## A.2 Data Types

Table A-1: Data Types of SIMATIC M7

Abb.	Keyword	Meaning	Bits
BO	BOOL	Logical number	1
BY	BYTE	Sequence of 8 bits	8
DI	DINT	Double integer	32
DW	DWORD	Sequence of 32 bits	32
I	INT	Integer	16
R	REAL	Floating-point number	32
ST	STRUCT	Structure	
TI	TIME	Duration	32
TK	Task	Pointer to a task	16
W	WORD	Sequence of 16 bits	16



## **B Abbreviations**

<b>BOP</b>	Basic operation (e.g. AND, OR, ...)
<b>BOP</b>	Basic operation (e.g. AND, OR, ...)
<b>C / C++</b>	High-level language for programming computers
<b>CFC</b>	Continuous Function Chart
<b>CPU</b>	Central Processing Unit
<b>DB</b>	Data Block
<b>ES</b>	Engineering System
<b>FB</b>	Function Block
<b>FC</b>	Function Code (function block without memory)
<b>OB</b>	Organization Block
<b>PC</b>	Personal Computer
<b>PCS 7</b>	Process Control System (SIMATIC)
<b>PG</b>	Programming Device

<b>PLC</b>	Programmable (Logic) Controller
<b>SFB</b>	System Function Block
<b>SFC</b>	System Function Call
<b>SFC</b>	Sequential Function Chart
<b>STEP 7</b>	Software development environment for SIMATIC S7 / M7

# Glossary

## B

### Background task

A background task is always executed when there is no other task active.

### Block type

The block types are C source files for an M7 CPU.

### Breakpoint

In the Test mode, breakpoints can be set in the CFC chart. When a breakpoint is reached, the program stops.

## C

### C program

Part of an M7 program, consisting of source files created in C and the executable machine code created from them for programmable M7 modules.

### Cyclic interrupt task

Cyclic interrupts are tasks that are started at regular intervals; in other words cyclically.

## E

### Error task

Error tasks allow defined reactions to certain errors detected by the run-time system during run time.  
There are three classes of error: I/O access errors, processor errors, timeouts.

## H

### **Hardware interrupt task**

Hardware interrupts are triggered by hardware interrupt signals. These interrupt signals can be assigned to a task to allow you to react to external process events.

## M

### **M7 program**

A folder for charts and C programs for programmable M7 modules.

## O

### **Oscilloscope**

An oscilloscope records the sequence of values adopted by one or more block I/Os during the test and displays them graphically as a curve.

## P

### **PLC (as target system)**

The PLC is the device (CPU) on which the user program runs.  
These include SIMATIC S7 and SIMATIC M7.

### **Programmable controller**

A programmable controller in SIMATIC M7 is also known as an automation computer.

## S

### **Software interrupt task**

Software interrupts are tasks that can be started by the user program.

**Source file**

Part of a program that is created with a textual editor for M7 that is compiled to create the machine code for the M7 CPU.

**Startup task**

A startup task can be assigned to every CPU of the M7 PLC. This is the first task to be executed following a reset, following return of power after a power outage.

**T****Task**

Tasks form the interface between the operating system of the M7 CPU and the user program. The order of execution of the user program is specified in tasks. A task corresponds to an organization block (OB) in S7.



# Index

## A

Additional functions ..... 5-18

## B

Background task ..... 3-3  
 Block I/O  
   monitoring ..... 5-3  
 Block type  
   compiling ..... 4-4  
   copying ..... 2-3  
   deleting ..... 2-6  
   importing ..... 2-4  
 Block type source file ..... 2-1  
 Borland C compiler  
   installation ..... 1-3  
   path name ..... 4-1  
 Breakpoint ..... 5-10  
   activating ..... 5-12  
   deleting ..... 5-14  
   disabling ..... 5-14  
   editing ..... 5-14  
   enabling ..... 5-14  
   setting ..... 5-13

## C

C libraries ..... 6-16  
 C source file ..... 2-1  
   structure ..... 6-3  
 CFC interrupt stack  
   displaying ..... 5-19  
 CFC system status  
   displaying ..... 5-18  
 CFC tasks  
   displaying ..... 5-19  
 Code library  
   location ..... 2-1  
   reference to ..... 6-11  
 Compiler errors ..... 2-3, 2-5  
 Cyclic interrupt (task) ..... 3-4

## D

Data flow ..... 1-4  
 Debug  
   resume ..... 5-17  
   step mode ..... 5-16  
 Defines ..... 6-15

Downloading  
   run-time system ..... 4-5  
   user program ..... 4-5  
 Downloading the run-time system ..... 4-5  
 Downloading the user program ..... 4-5

## E

Edit mode ..... 5-1  
 Error task ..... 3-3

## G

Generic property ..... 6-8

## H

Hardware interrupt (task) ..... 3-6

## I

Include file  
   copying ..... 2-2  
   reference to ..... 6-11  
 Installing  
   Borland C compiler ..... 1-3  
 Interrupt task ..... 3-6

## K

Keyword  
   BLOCKVIEW ..... 6-7  
   CODELIBS ..... 6-11  
   COMMENT ..... 6-6  
   DYNAMIC ..... 6-13  
   END\_FUNCTION ..... 6-6  
   END\_FUNCTION\_BLOCK ..... 6-6  
   END\_VAR ..... 6-8  
   FAMILY ..... 6-8  
   FUNCTION ..... 6-6  
   FUNCTION\_BLOCK ..... 6-6  
   GENERIC ..... 6-8  
   INCLUDES ..... 6-11  
   LINK ..... 6-13  
   NAME ..... 6-6  
   STRING\_0 ..... 6-12  
   STRING\_1 ..... 6-13  
   STRUCT ..... 6-9  
   TASKLIST ..... 6-7  
   UNIT ..... 6-12  
   VAR ..... 6-10

VAR_INPUT .....	6-8
VAR_OUTPUT .....	6-8
VISIBLE .....	6-12
Keywords in the C source file .....	6-4

## L

Logs .....	2-5
------------	-----

## M

M7 blocks .....	2-2
M7-SYS .....	1-3
Multiple installation of blocks .....	6-15

## O

Optimizing the run time .....	4-2
Oscilloscope .....	5-4
assigning block I/Os .....	5-6
creating .....	5-6
deleting .....	5-9
opening .....	5-6
printing .....	5-9
renaming .....	5-6
starting recording job .....	5-7

## P

Phase offset (cyclic interrupt) .....	3-4
Priority of a task .....	3-8

Program	
hold .....	5-16
resuming .....	5-16

## R

RMOS task priorities .....	3-8, 4-3
----------------------------	----------

## S

Scale .....	5-8
SIMATIC Manager .....	1-2
Software interrupt (task) .....	3-6
Source file	
block types .....	2-1
code libraries .....	2-1
Startup task .....	3-3
Structure elements .....	6-15

## T

Task	
configuring .....	3-1
priority .....	3-8
Test mode .....	5-1, 5-2
Time constant (task) .....	6-16

## V

Version of the M7 system software .....	4-2
---	-----