EFB Toolkit for EcoStruxure™ Control Expert

User Manual

01/2019

33003021.11

www.schneider-electric.com

The information provided in this documentation contains general descriptions and/or technical characteristics of the performance of the products contained herein. This documentation is not intended as a substitute for and is not to be used for determining suitability or reliability of these products for specific user applications. It is the duty of any such user or integrator to perform the appropriate and complete risk analysis, evaluation and testing of the products with respect to the relevant specific application or use thereof. Neither Schneider Electric nor any of its affiliates or subsidiaries shall be responsible or liable for misuse of the information contained herein. If you have any suggestions for improvements or amendments or have found errors in this publication, please notify us.

No part of this document may be reproduced in any form or by any means, electronic or mechanical, including photocopying, without express written permission of Schneider Electric.

All pertinent state, regional, and local safety regulations must be observed when installing and using this product. For reasons of safety and to help ensure compliance with documented system data, only the manufacturer should perform repairs to components.

When devices are used for applications with technical safety requirements, the relevant instructions must be followed.

Failure to use Schneider Electric software or approved software with our hardware products may result in injury, harm, or improper operating results.

Failure to observe this information can result in injury or equipment damage.

© 2019 Schneider Electric. All rights reserved.

Table of Contents

	Safety Information	7
	About the Book	9
Chapter 1	**Introduction to the EFB Toolkit Methodology**	11
	Introduction	11
Chapter 2	**EFB Toolkit Services and User Functions**	13
	Program Installation	14
	How to Register the Software	15
	Product Specifications	17
	EFB Toolkit Commands	19
	Installable Families	23
	Comparison Between the Different Block Types	29
	Multi Language Support	31
	Help on Type	34
Chapter 3	**Coding Rules**	37
	Coding Rules	38
	Addressing	41
	Constant Values	43
	Local Functions	45
	EFB Data Instances	47
	EN/ENO Pins	48
	Extensible Pins	50
Chapter 4	**Programming Examples**	53
	EF/EFB Example	54
	EF Program Code	57
	EFB Header File	60
	EFB Source Code	64
	DDT Example	66
Chapter 5	**Debugging**	71
	Debug Preparation	72
	Best Practices	79
	Recommendations	80
Chapter 6	**System Services**	81
6.1	Overview	82
	System Service Overview	82

6.2 Description 86
s_AliGetProgStat 88
s_log_to_phy 90
s_obj_to_log 92
s_obj_nbr 94
s_rd_16bits 95
s_rd_1bit 97
s_rd_bit_attrib 99
s_rd_internalwords 101
s_rd_Nbits 103
s_rd_sysbit 105
s_rd_sysword 106
s_wr_16bits 108
s_wr_1bit 110
s_wr_bits_attrib 112
s_wr_internalwords 114
s_wr_Nbits 116
s_wr_sysbit 118
s_wr_sysword 120
s_cnt_100ms 122
s_cnt_10ms 123
s_cnt_1ms 124
s_date_and_time 125
s_syscnt_10ms 128
s_current_task 129
s_set_ffb_error 131
s_set_ffb_error_addi 133
s_diag_RegisterExtError 135
s_diag_DeregisterError 137
s_of_passw_check 139
s_of_passw_test 141
s_demask_it 143
s_GetUSecs 144
s_mask_it 145
s_proc_indic 146
s_proc_type 148
Appendices 151

Appendix A PL7/Concept EF/EFB Migration 153
A.1 PL7 and Concept EF/EFB Migration 154
PL7 EF Migration 154
A.2 PL7 EF Migration Procedure 155
Retrieving Source Files from the PL7 Development Environment . . . 156
Migration Procedure 157
PL7 EF Code 160
Control Expert EF Code 161
A.3 Concept EF/EFB Migration Procedure 163
Retrieving Source Files from a Concept Development Environment . 164
Migrating the Function Block Code to Control Expert 165
Concept EF/EFB Code 168
Control Expert EF Code 169
A.4 Other Case Studies 171
User-defined Data Types (PL7 only) 172
Floating Point Constants 175
ANY... Data Types 176
REF... Data Types 177
Determining the PLC State 178
Reporting User Defined Errors 179
Extensible inputs 180
A.5 Comparison Between PL7/Concept and Control Expert 184
Common System for EF/EFBs in Control Expert. 185
Data Types Comparison: PL7/Concept and Control Expert 189
PL7 SDKC Include File: <Cstsyst.h> Versus <SystemLib.h> (PL7 Only) 191
A.6 An Empty Frame for a Control Expert EF 194
Example_EF1 Interface 195
Example_EF1 Header File. 196
Example_EF1 C-Source Template 199
Programming Hints 203
Index 205

Safety Information

Important Information

NOTICE

Read these instructions carefully, and look at the equipment to become familiar with the device before trying to install, operate, or maintain it. The following special messages may appear throughout this documentation or on the equipment to warn of potential hazards or to call attention to information that clarifies or simplifies a procedure.

The addition of this symbol to a “Danger” or “Warning” safety label indicates that an electrical hazard exists which will result in personal injury if the instructions are not followed.

This is the safety alert symbol. It is used to alert you to potential personal injury hazards. Obey all safety messages that follow this symbol to avoid possible injury or death.

DANGER
DANGER indicates a hazardous situation which, if not avoided, **will result in** death or serious injury.

WARNING
WARNING indicates a hazardous situation which, if not avoided, **could result in** death or serious injury.

CAUTION
CAUTION indicates a hazardous situation which, if not avoided, **could result** in minor or moderate injury.

NOTICE
NOTICE is used to address practices not related to physical injury.

PLEASE NOTE

Electrical equipment should be installed, operated, serviced, and maintained only by qualified personnel. No responsibility is assumed by Schneider Electric for any consequences arising out of the use of this material.

A qualified person is one who has skills and knowledge related to the construction and operation of electrical equipment and its installation, and has received safety training to recognize and avoid the hazards involved.

About the Book

At a Glance

Document Scope

This documentation describes the EFB Toolkit.

Validity Note

This documentation is valid for EFB Toolkit V14.0.

Product Related Information

Programming Language 'C'

The programming language 'C' is a powerful language that offers many features, such as pointers.

But programming in 'C' naturally contains risks, for example:
- You can define a pointer and modify the address to which the pointer points freely in your program.
- You can cast a pointer to any type you like, etc.

If you are programming function blocks in 'C', you have to be aware of outcomes from each of the function blocks. A 'C' coded function block is a piece of code compiled from a standard 'C' compiler used inside the PLC. There are no security checks inside Control Expert or inside the PLC that check the function block code for correct operation. If the code is not correct, the PLC memory may be corrupted at any location inside the PLC. As a consequence the PLC may crash or behave unexpectedly.

⚠ WARNING

UNEXPECTED PLC BEHAVIOR

- Check the function block code to make sure that it is working correctly.
- For the test and debugging phase, use the PLC Simulator of Control Expert instead of a real PLC.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

Schneider Electric does not take any warranty for 'C' coded function blocks written by users of the EFB Toolkit. This is also valid for the consequences of damages or injuries caused by incorrectly coded function blocks.

Working Manually on The Function Block Files

The files are generated and managed by the EFB Toolkit generates and absolutely necessary to generate the installable form of the family.

Any modification within this directory can lead to unexpected EFB Toolkit behavior or the generation of wrong executable code and result in unexpected PLC behavior.

⚠ WARNING

UNEXPECTED PLC BEHAVIOR

Do not modify or delete any file located in the family development directory (*FFBDev*) using tools other than the EFB Toolkit.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

Moving / Deleting Files of Installable Form

These files are generated and managed by the EFB Toolkit and are necessary to install the family into the Control Expert libset.

Any modification within this directory can lead to unexpected EFB Toolkit behavior or the generation of wrong executable code and result in unexpected PLC behavior.

⚠ WARNING

UNEXPECTED PLC BEHAVIOR

Do not move or delete any file located in the installable form directory (*FFBInst*).

Failure to follow these instructions can result in death, serious injury, or equipment damage.

Chapter 1
Introduction to the EFB Toolkit Methodology

Introduction

General

The EFB Toolkit enables you to generate customized functions and function blocks. It provides a complete development environment for programming, creating, and installing libraries. Tasks are started via menu picks in the development environment. The menus launch batch processes that carry out the required steps automatically.

Families

The elements in the user-defined functions and their associated content are stored in a unit called a *family*. A family may comprise:

- elementary function (EF)
- elementary function block (EFB)
- derived data type (DDT)

EF and EFB Representations

A user familiar with Control Expert will easily recognize how the EFB Toolkit represents its EFs and EFBs. Functions are shown as graphical units with input and output pins. Unlike in a standard Control Expert environment, with the EFB Toolkit you can:

- create your own EFs and EFBs with customized input and output pins
- manipulate the internal states of variables within the EFBs, using public and private variables

Each EFB has instance data.

User-defined EFBs

The EFB Toolkit enables experienced users to generate custom EFBs for their applications. These user-defined EFs and EFBs can be managed in libraries in a manner similar to the way standard functions and function blocks are delivered with Control Expert. (The libraries for EFB Toolkit functions are not the same ones that are delivered with Control Expert.)

NOTE: There is no difference between elementary and user-defined functions/function blocks within Control Expert.

DDTs

You can define data structures or data arrays in the DDT section of the family.

IEC Languages

Control Expert applications can be created in several languages (FBD, SFC, LD, ST, IL, LL984) using blocks provided in accordance with IEC1131.

Programming Language

EFs and EFBs are developed with a reduced-functionality version of the ’C’ programming language. The reduced functionality is described in more detail in following chapters.

Functions and Procedures

Characteristics of EF functions and procedures:

Method	Description
Function	A function has 1 output value (return pin).
Procedure	A procedure could have 1 to n output values (return pins) and 1 to m input values.

DSC Files

Do not modify any .**dsc** file using tools other than the EFB Toolkit. This can provide inconsistent data.

Chapter 2
EFB Toolkit Services and User Functions

Introduction

This chapter provides an overview of the services and user functions of the EFB Toolkit.

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
Program Installation	14
How to Register the Software	15
Product Specifications	17
EFB Toolkit Commands	19
Installable Families	23
Comparison Between the Different Block Types	29
Multi Language Support	31
Help on Type	34

Program Installation

Product Content

The EFB Toolkit consists of:

- CD-ROM 1
  - EFB Toolkit software
  - installation software
  - a registration tool *(see page 16)*
  - the Microsoft C++ compiler
  - user documentation
  - an installation note
- CD-ROM 2
  - GNU ARM C compiler
- DVD-ROM 3
  - Microsoft Visual Studio

These components are provided on 3 installable ROMs.

NOTE: No paper documentation is provided.

Operating Systems

The setup program for the EFB Toolkit must be launched on Windows 7 Professional (64 bit), Windows 10 Professional (64 bit) and Windows Server 2016.

Installation

The following table shows the steps for installation.

Step	Action
1	Install the EFB Toolkit CD-ROM. If autostart of CD-ROM is deactivated, use **\setup.exe** to start the installation.
2	Install the Gnu ARM C compiler CD-ROM. If autostart of CD-ROM is deactivated, use **\setupGNU.exe** to start the installation.
3	Install the Microsoft Visual Studio DVD-ROM. If autostart of DVD-ROM is deactivated, use **\setup.exe** to start the installation.

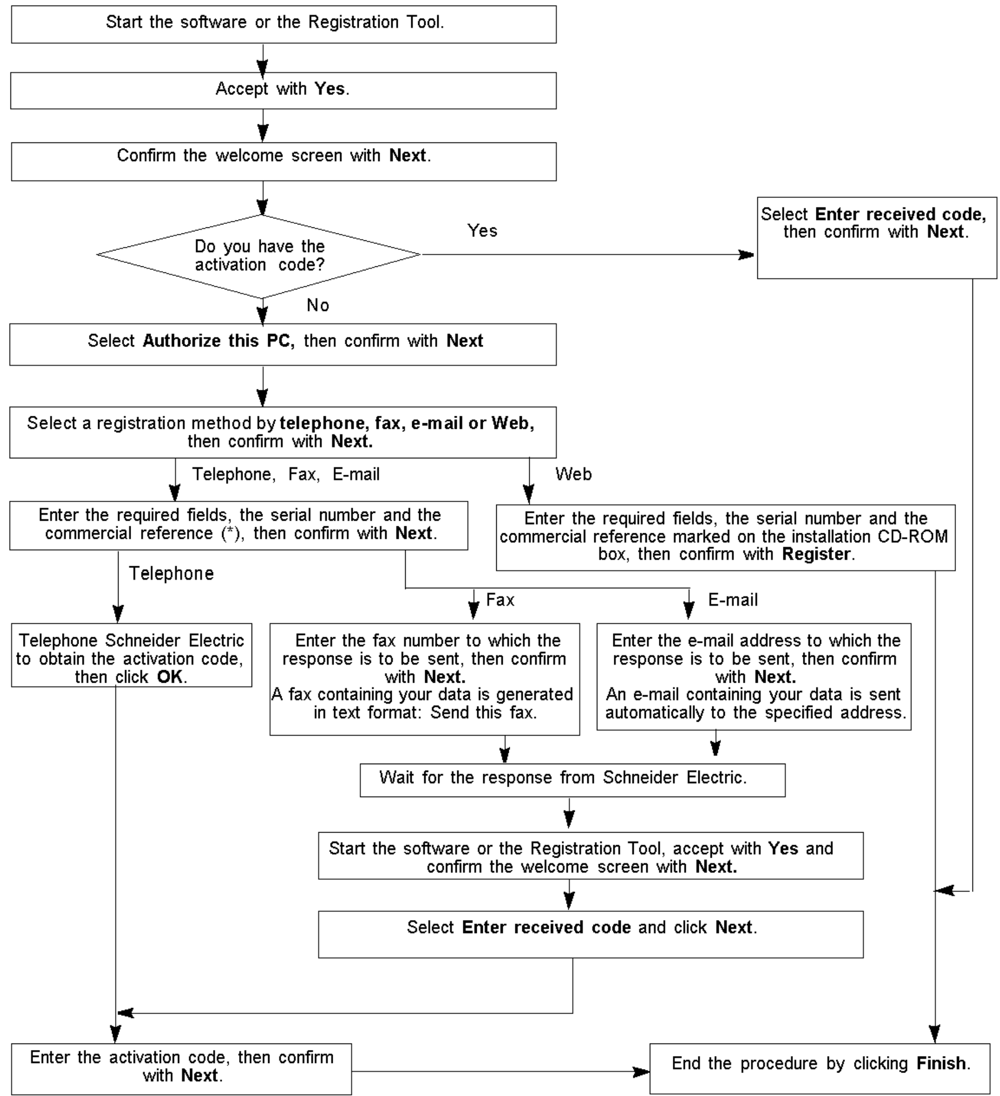
How to Register the Software

At a Glance

To obtain permanent user rights for the EFB Toolkit software, you have to register it with Schneider Electric. Once the software is installed, you have to register within 21 days.

Procedure

A wizard guides you through the registration procedure. Here is a flowchart:



(*) These numbers are marked on the label inside the box containing the software CD-ROMs.

Product Specifications

Product Overview and Description

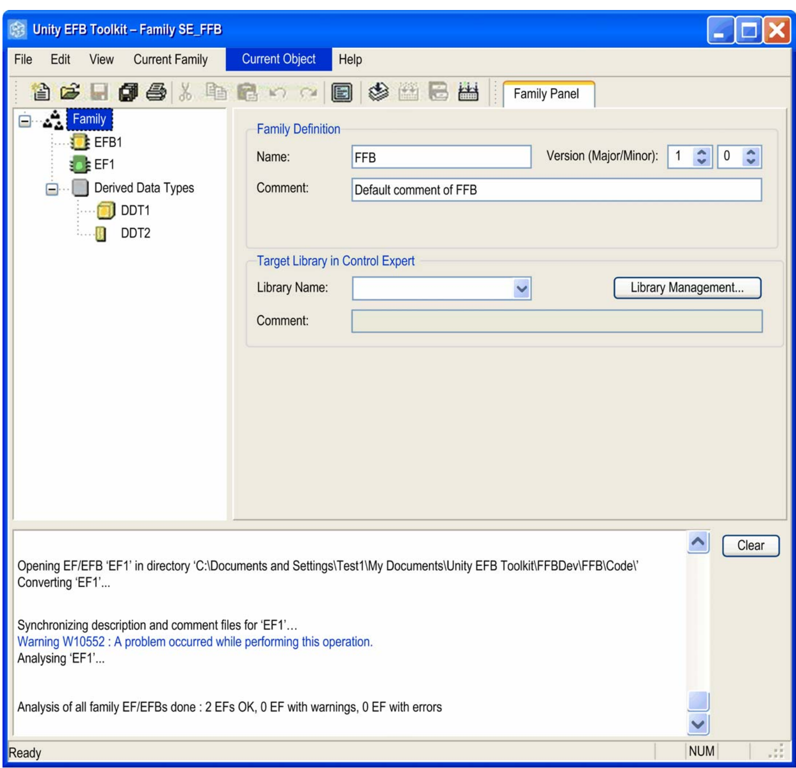
The services supplied with Unity EFB Toolkit allow you to:

- create, edit and compile families of EFs, EFBs and DDTs (see *EF and EFB Representations, page 11*)
- make an installable family for test and debug (see *Families, page 11*)
- make installable families in a library for sale
- install a family in a Control Expert libset
- print a family or an individual EF, EFB or DDT

Graphical User Interface

At start-up, the EFB Toolkit software screen displays three panes:

- the navigation pane (on the left)
- the main pane (on the right)
- the log pane (on the bottom)



Navigation Pane

The navigation pane displays a structure view of the family. The root item describes the family. Each sub-item corresponds to an EF or an EFB.

You can select a DDT defined in the family from the DDT folder.

Main Pane

The tabs on the main pane correspond to items selected in the navigation pane:

- **Family**
  - **Family description**
    Here you may select a family name, a version number and a name for the Control Expert library. You can also add comments.
    Select or generate a library in **Library Management ...**
- **EF/EFB**
  - **EFB description**
    Here you may select or generate information such as an EF/EFB name, an author´s name and a long or short description of the EF/EFB.
  - **EFB log file**
    This register displays read-only EF/EFB analysis information, i.e., detected errors and detected warnings.
  - **EFB header**
    The header file for an EF or EFB.
  - **EFB source**
    Here you may display and edit the source code for an EF or EFB.
- **DDT**
  - **DDT description**
    Here you may select or generate a DDT name, an author´s name and a long or short description of a DDT.
  - **DDT header**
    The header file for the DDT.

Log Pane

The **Log Pane** shows operation results such as detected errors and detected warnings that may occur during the generation, compilation or build of the function blocks.

EFB Toolkit Commands

Command Menu Structure

The following information describes the different commands in the EFB Toolkit GUI.

File

The **File** commands are used to create a new family or to open an existing family. You may install the current family into the Control Expert libset so that you may use the customized EFs, EFBs or DDTs.

To edit the EFB Toolkit settings, use the **File → Settings...** command. A dialog box will be displayed.

You have 3 possible settings:

- **Directories**
  Select the development and installation directory for the EFB Toolkit and the Target library.
- **Build Options**
  The **Build Options** allow you to:
  - select the **Environment** for **Microsoft Visual C 32bit**, **GNU C ARM ELF**, or **GNU C ARM5 ELF**
  - insert or edit **Compiler Options**
  - add additional link libraries
  - edit the behavior of detected warning messages
  - suppression of linker and compiler messages

  Using incorrect compiler settings can lead to the generation of incorrect executable code that result in the unexpected PLC behavior.
  You do not need to modify the compiler options but you have to define symbols of constants for your source code.

⚠ WARNING

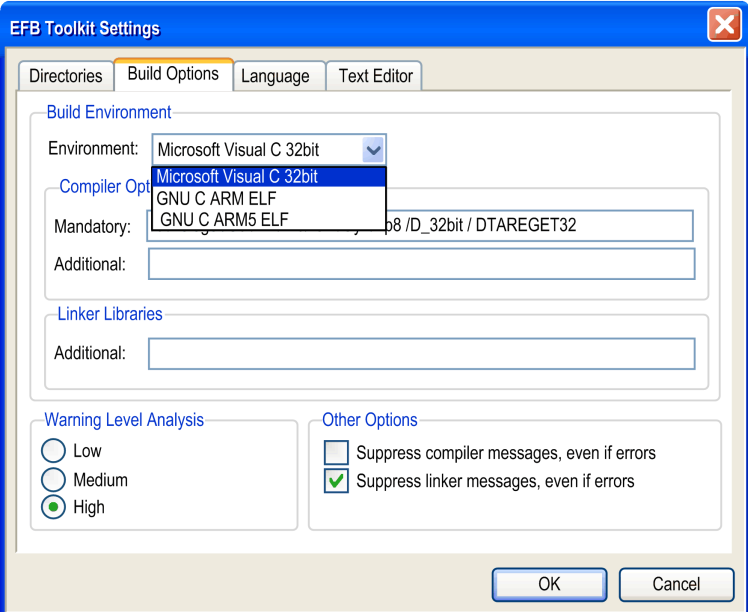
UNEXPECTED PLC BEHAVIOR

- Do not modify compiler options on the Build Options sheet of the **EFB Toolkit Settings** dialog box.
- When constants have to be defined for the source code, this must be performed by qualified personnel only and validated in a simulated environment before being put into production.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

NOTE:

The following figure shows the build option:

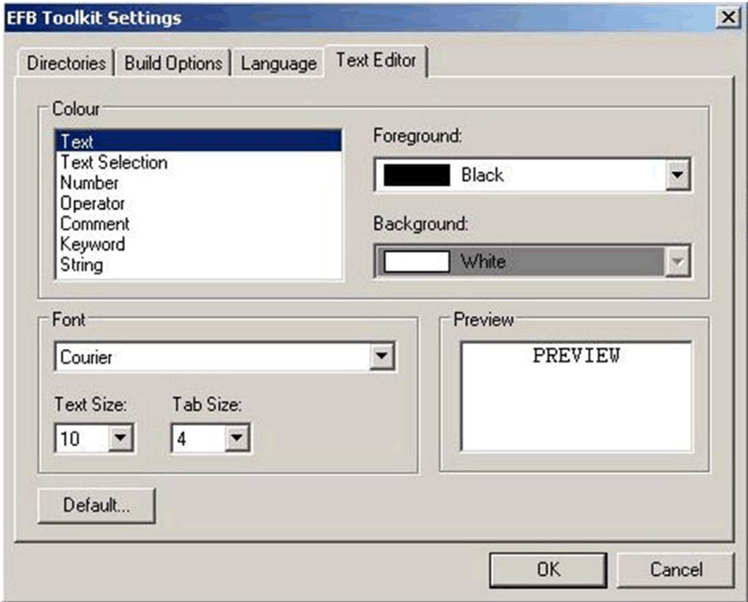


The **File** command also lets you specify some additional libraries to be linked with the family. It also lets you modify the warning levels in the analysis messaging and hide the Microsoft and Plink messages.The File command also lets you specify some additional libraries to be linked with the family.

You may launch a browser to choose an EF or an EFB to import it into the current family. When you import, the needed files are copied to the family, and the family description is updated. You have to analyze and generate the EFs or EFBs to validate the import. To avoid the two EFs or EFBs having the same name, there is an option to remove the imported EFB from its original family.

To print, select **File → Print**. Executing this command prints the whole family, including the description file, EF/EFB source files and comments. When a print operation is launched, a dialog box appears where you can select the printer settings.

- **Language**
  Select the languages for your project (see also *Multi Language Support, page 31*).
- **Text Editor**
  You can choose different color, sizes, and font for the source code that is appearing on header and source file. You can select different colors for **Keyword**, **Comment**, **Text Selection**, **Operator**, **Number** and other text appearing in the source/header file.



Edit

The **Edit** menu contains items common to most other MS Windows applications. It also lets you clear the log pane.

View

The **View** menu lets you select different views of the EFB Toolkit software. For example, you may toggle between full screen mode and the standard screen display.

Current family

The **Current family** menu appears as soon as a family has been opened or created. The options available under this menu include:

- **Create EF/EFB** or **Create DDT**, where you can create or add an EF, EFB or DDT
- **Analyze and Generate code for EF/EFB**, which analyzes the description file of each EF/EFB. If no detected errors, it generates:
  - default source code (*.c and *.h files)
  - an EF/EFB comment file
  - an EF/EFB template file
- **Compile all EF/EFB code**, which compiles each EF/EFB in 32-bit format
- **Make the installable form of the family**, which patches object files and copies the necessary files to the install directory
- **Rebuild all**, which launches successively the three operations above (useful, for example, when a comment is modified)
- **Debug all EF/EFBs**

These commands apply to the whole family. When a command is selected, it is executed successively on each EF/EFB in the family.

Current Object

The **Current Object** menu allows the code of the current EF or EFB to be analyzed, generated and compiled. Once an EF/EFB has been selected in the navigation pane, the following items are enabled:

- **Analyze and Generate code**
- **Compile code**
- **Delete Current Object**

To delete an EF, EFB or DDT, you have to select that item in this menu.

These are the same commands that appear in the **Family** menu. Here they are applied to the current EF or EFB.

Help

The **Help** command invokes the help files in the EFB Toolkit software.

Context Menu

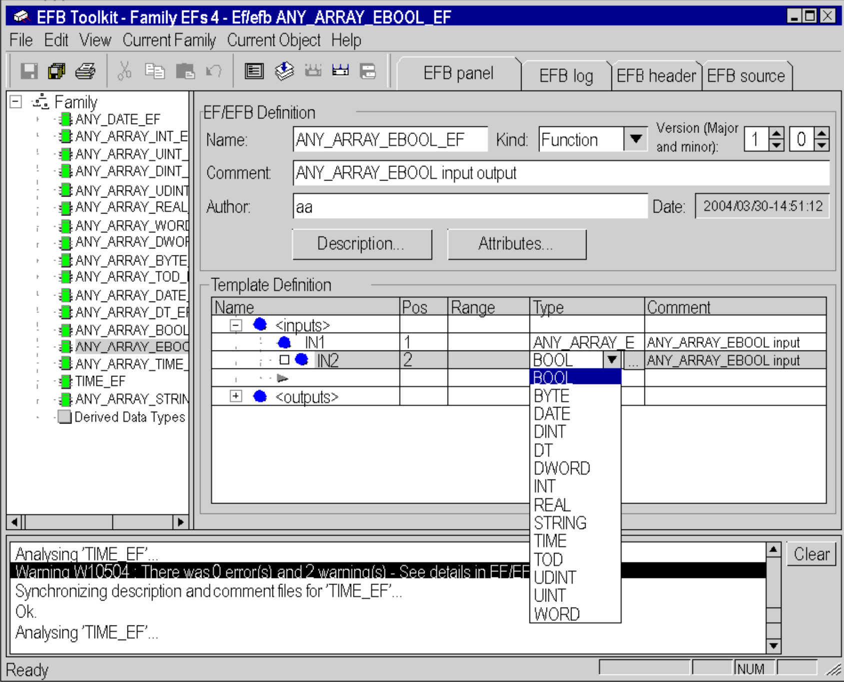
The programmer has the possibility to use the right mouse button dependent on the selected item.

Installable Families

Make an Installable Family

To build a family which can be installed in the Control Expert libset, follow the steps below.

The following figure shows the EFB Toolkit with the family descriptor.



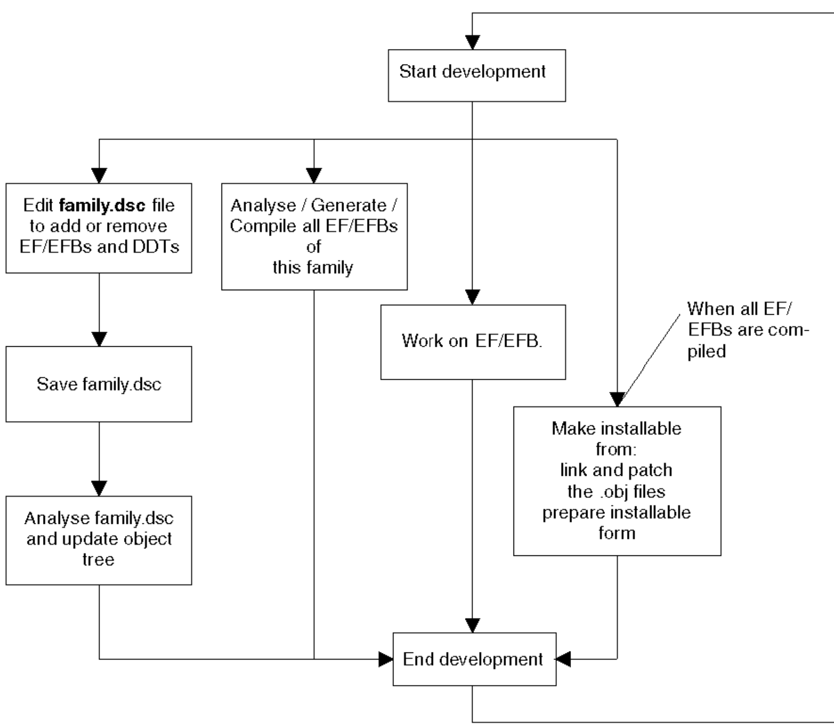
Create a New Family

The following table shows the steps to create a new family:

Step	Action	Comment
1	**File → New family**	Creates a new family project.
2	Select a family name	The user has to select a family name. This name corresponds to the name of the family displayed in Control Expert and the family directory. **Result:** After that actions the new family appears in the tree view in the navigation pane.

Development Cycle

The following figure shows the development cycle within the EFB Toolkit.



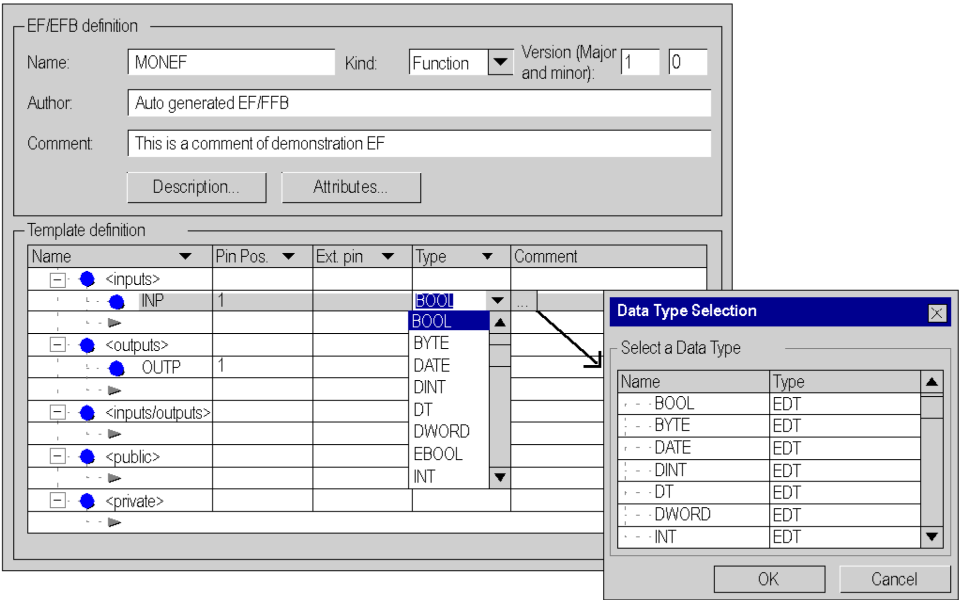
Create a New EF/EFB

The following table shows the steps to create a new EF/EFB:

Step	Action	Comment
1	**Current family → Create EF / EFB**	Creates a new EF or EFB.
2	Dialog box opens	A dialog box will be displayed to help the user enter the EF / EFB name and category. This action creates the description file of the EF/EFB. **Result:** The new EF or EFB object appears in the tree view in the navigation pane.

EF/EFB Options

The following figure shows the program window for the EF/ EFB definition after the selection of the EF/EFB name.



The possible values for **Kind** are **Function** or **Procedure** in case of EF or only **Function** in case of EFB.

The programmer is able to create different kinds of pins, basic data types or extended data. If the pin is selected by the checkbox, this one is an extensible pin. That means the user has the possibility to extend the number of pins within Control Expert. The programmer can define the number of pins from 2...32. The maximum number of input pins is 32.

The pin position can be changed by the programmer.

Descriptive Form

The descriptive form is available within Control Expert. The user has access to the information in **Control Expert → Types Lib Browser → Properties**.

Create a new DDT

The following table shows the steps to create a new DDT:

Step	Action	Comment
1	**Current family → Create DDT**	Creates a new DDT
2	Select DDT name	The user has to select the DDT name and kind in a dialog box. **Result:** The new DDT object appears in the tree view in the navigation pane.

Kinds

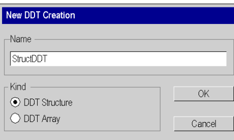
For **Kind** you can select `Structure` or `Array`.

Additionally you have to enter the following information for the new DDT:

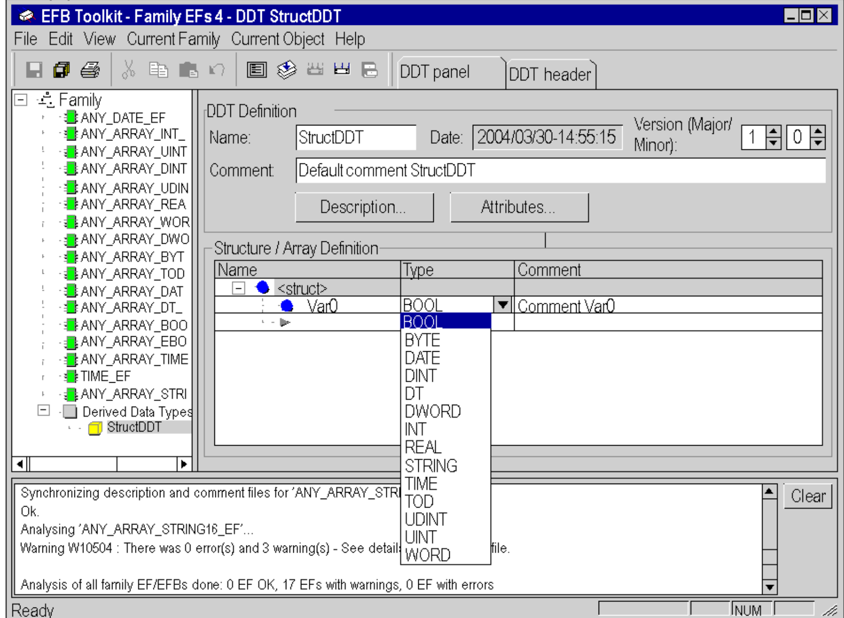
- **Name**
- **Version**
- **Comment**

This information is necessary for Control Expert after changes within used DDTs.

The following figure shows the program window for selecting a variable structure (`struct` or `array`).



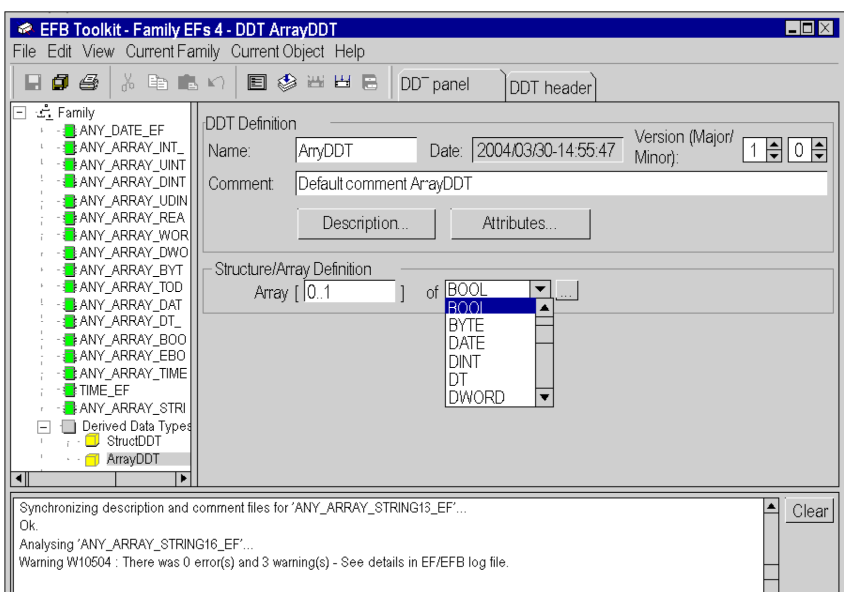
The following figure shows the program window for selecting the DDT information for a structure.



The user is able to edit the following structure definitions:

- name, version, comment, description
- structure definition
  The user creates new structure entries by double-clicking the free entry below the last data. The user has to select a name and type of the data entry. The comment is optional.

The following figure shows the program window for selecting the DDT information for an array.



The user is able to edit the following array definitions:

- name, version, comment, description
- array size
- array type

Comparison Between the Different Block Types

Comparison Table

The following table gives the comparison between an elementary function and a procedure:

Elementary Function	Procedure
An elementary function returns a function value.	A procedure does not necessarily return a function value.

In the ST language, you can use this returned value as an expression by using the EF call.
Example 1:
`INT1:= ABS_INT(INT2)`
Example 2:
`INT1:= INT2 + ABS_INT(INT3)`
	If and when a procedure does return a value, you cannot use the returned value in an expression by using the EF call unlike an elementary function.
	The procedure can be used as shown in the following examples:
	Example 1:
	`INC_INT(INT2)INT1:= INT2`
	Example 2:
	`INC_INT(INT3)INT1:=INT2+INT3`

The elementary function can have 1 or many input parameters that can be dynamically extended for specific functions.	The procedure can also have 1 or many input parameters.
The elementary function has 1 output that returns the function value and does not store values from one call to the next in internal memory.	The procedure can have more than 1 output.
The elementary function does not have an `INOUT` pin.	The procedure can have an `INOUT` pin.
The elementary function can have extensible pins. ADD_REAL nin R2 IN1 OUT R3 IN2	The procedure does not have extensible pins.

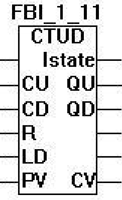
NOTE: `LOOKUP_TABLE1` is a known exception to these rules. It has several outputs and extensible pins. In this case, `LOOKUP_TABLE1` will be a procedure.

Elementary Function Block (EFB)

An elementary function block has a data structure that has a contiguous area of memory to keep internal values, inputs, and outputs from one call to the next. In the instance area, you can declare the variables in the data structure as hidden or publicly accessible.

The data structure is automatically allocated at the time of declaration. Therefore, the number of EFB inputs cannot be dynamically extended. However, the language editors may simulate extensibility for EFBs at language level only, but the EFB code sees the same data structure with maximum number of inputs.

The variable editor and manager maintain the EFB data structures. The graphical editors inform the variable manager to declare an instance implicitly when placing a function block.



Multi Language Support

Introduction

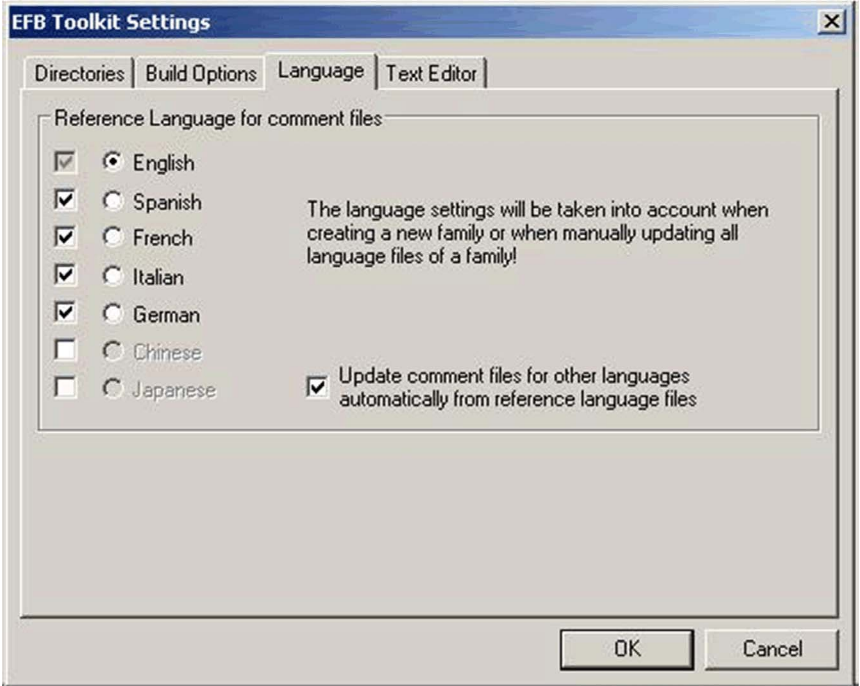
You can deliver new families of EFs, EFBs and/or DDTs with different description. These descriptions will be written in a single source language. The EFB Toolkit creates different directories for the possible languages. After the creation of the new family, the directories contain the language files for the translation. After the translation the files can be bound to the family.

NOTE: The EFB Toolkit is **not** a translation tool. The language files have to be translated by other resources.

The Language Settings Dialog

If you generate a new library, at least one new language file will be generated. The number of languages and which languages will be supported, depend on your settings in the **EFB Toolkit Settings**.

The following figure shows the **EFB Toolkit Settings** with the language support:



The check boxes enable the supported languages for the specific family. At least one check box has to be selected. For every selected language a directory with the language name will be created. The radio buttons select the source language. In this example, English is the source language.

Other directories will get a copy of the language files from the source directory, in case the **Update comment** check box is enabled. This copy will be created after every change within the family.

You can translate the copied language files. After translation, deselect the check box **Update comment files for other languages automatically from reference language files** to avoid overwriting them again.

This is useful, if you make only minor changes in the descriptions or if you make only changes at the EFs, EFBs or DDTs.

Unchecking the automatic update of comment files can lead to inconsistent variable layout of the EF/EFB or DDT.

It is recommended to uncheck this option only in the case when the comment files were translated to help prevent them from overwriting.

Using inconsistent comment files can lead to the generation of wrong executable code and result in unexpected PLC behavior.

Best practice is to translate the EF/EFB and DDT comment files not before all EF/EFB interfaces and DDT layouts are stable.

⚠ WARNING

UNEXPECTED PLC BEHAVIOR

- Validate any changes in the Language sheet settings of the **EFB Toolkit Settings** dialog box, to ensure that these changes meet the site specific requirements.
- Check the Automatic update of the comment files before changing the layout of the EF/EFB or DDT.
- Backup the translated comment files into a save location before changing the layout of the EF/EFB or DDT.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

Directory Structure for the Language Files

The EFB Toolkit creates at least one directory for the language files when you save your project. This directory is specified by the radio button in the **EFB Toolkit Settings** in the **Language** tab.

Additionally, other directories for the language files will be created. See **The Language Settings Dialog** above.

The content of these language specific directories can be updated manually by executing **Current Family → Update all language files**. To create new additional language directories it is also necessary to execute this menu command.

Supported Languages

The following languages are supported:

- English
- French
- German
- Spanish
- Italian
- Chinese
- Japanese

Help on Type

General

You can create HTML help files for EFs, EFBs and DDTs generated with the EFB Toolkit.

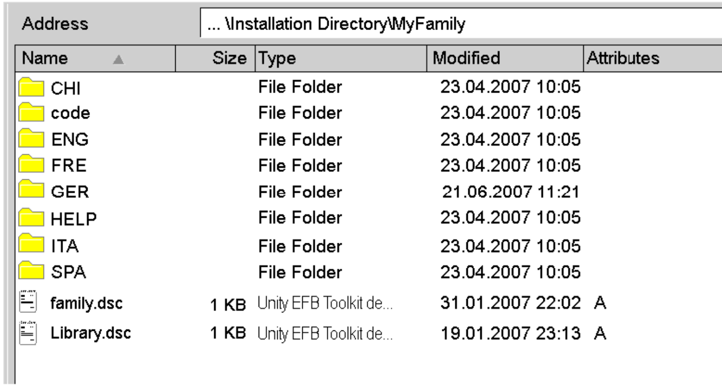
These files will be copied to the **Libset** directory when installing the family in Control Expert.

In Control Expert dialogs for selecting FFBs there is a **Help on Type** button. With this button you can launch the help for a selected FFB.

Directory Structure

For every user-defined family the EFB Toolkit creates a new directory (e.g. `MyFamily`) beneath the installation directory with the following subdirectories:

- language subdirectories (e.g. `CHI`, `ENG`, `FRE`...)
- `code` subdirectory



NOTE: The `HELP` subdirectory you have to create by yourself.

Creating Help on Type

Step	Action
1	Create an HTML help file for your FFB (e.g. with any kind of HTML editor). **Note:** The HTML file name has to be exactly the same as the name of your FFB and the extension has to be **.htm*.
2	Copy this file into all language folders (ENG, FRE ...). See structure above.
3	Copy all the files (e.g. graphics) you are referencing in your HTML file to the HELP folder. See structure above.
4	Install your user-defined family in Control Expert. **Result:** All the files will be copied to the **Libset** directory and the HTML help file will be launched when you click the **Help on Type** button.

Chapter 3
Coding Rules

Summary

This chapter provides an overview of the coding rules in the EFB Toolkit.

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
Coding Rules	38
Addressing	41
Constant Values	43
Local Functions	45
EFB Data Instances	47
EN/ENO Pins	48
Extensible Pins	50

Coding Rules

General

The following coding examples are abstracts from complete programs. They are presented here to clarify the declared rules.

Coding Rules

- EFs/EFBs are re-entrant functions with a 'C' interface that returns Boolean values (IECBool).
- EFs/EFBs have to return a valid value that describes the execution state:
  - TRUE for no detected errors; detected warnings are possible
  - FALSE for detected error

  If an EF/EFB does not detect any errors, it returns with TRUE.
- EFs/EFBs must not allocate dynamic memory; they work with assigned memory.
- The generated EF/ EFB code is entirely portable within PLC memory, and moving it does not introduce any changes.
- CONST segment and DATA segment cannot be used in EF/EFB code.
- Global data cannot be used in EF/EFB code.
- Switch cases cannot handle more than three cases.
- Physical pointers should be declared as `__far`.
- Only the LSB (bit 0) of a Boolean (IECBool) value is significant. Other bits should be masked before testing for TRUE/FALSE.
- Temporary variables may use only processor stack space.
- EFs/EFBs may access system functions via System Service calls.
- EFs/EFBs may access PLC resources only through System Service calls, never directly.
- If necessary, EFs/EFBs can use self-written static sub-functions. To improve performance, you may design functions, such as C++-inline functions, and you may write them in assembly language.
- Ensure that system service calls, self-written sub-functions and used C-library calls are reentrant.
- Each EF/EFB type gets its own source module.
- Whenever possible, write the source codes of generic EFs with generic data type and function names. The adaptation to the real data types is done by front-end compiler definitions when compiled.
- EFs/EFBs may write only permitted type and range data in memory. For example, Boolean (IECBool) values may be defined for only 0 (FALSE) and 1 (TRUE).
- The order of EF/EFB parameters is provided in the Unity EFB Toolkit.
- The passing type (by value or by logical address) of EF/EFB parameters is given by Unity EFB Toolkit.

Unaligned Memory Access

The ARM architecture, used by M340 PLCs, normally expects memory accesses to be suitably aligned. In particular, the address used for a DWORD (32bit) access should normally be DWORD aligned (address divisible by 4), and the address used for a WORD (16bit) access should normally be WORD aligned (address divisible by 2). Memory accesses which are not aligned in this way lead to an exception and the PLC enters the error state.

If you have to access unaligned data you have to use special macros:

- GET_SHORT(ptr) / SET_SHORT(ptr)
- GET_LONG(ptr) / SET_LONG(ptr)

ptr is a pointer to unaligned data.

EBOOL

The EBOOL contains the value FALSE (=0) or TRUE (=1) and also information about the management of falling or rising edges and forcing bits.

Example:

```
//////////////////////////////////////////////////////////
#if defined(__arm__) && defined(__GNUC__)
IECEBool far *pIN1;
#endif
    unsigned short    NbAcces16Bit,I;
#if defined(__arm__) && defined(__GNUC__)
pIN1 = s_log_to_phy(IN1.IECPtr_Log);
#endif
#if defined(__arm__) && defined(__GNUC__)
Tampon1 = rd_Nbits(pIN1,TAILLE_ACCES16_ARX);
#else
Tampon1 = s_rd_16bits(IN1.IECPtr_Log);
#endif
```

Test of Boolean Variables

You can use the following macro for the test of Boolean variables:

```
/////////////////////////////////////////////////////////
// Macros to test variable of type IECBool
// (from q_portdefs.h)
/////////////////////////////////////////////////////////
   #define test_BOOL(x) ((x) & TRUE)
   #define if_TRUE(x)  if (test_BOOL(x))
   #define if_FALSE(x) if (test_BOOL(x) == FALSE)
   Example:
   IECBool input;
   if_TRUE(input)
   {
      ...
   }
   if_FALSE(input)
   {
      ...
   }
```

Addressing

Addressing Rules

Before using variables, the programmer has to convert logical into physical addresses.

Logical vs. Physical Address

Usually parameters, which are managed by reference, are not passed with a physical pointer (traditional C pointer) but rather with a logical one.

These elements are typed as `IEC_RTE_OFFSET` or `IEC_PARAM_RTE_OFFSET_LG`

The following program code shows an example for physical and logical pointer addressing.

```
// Parameter given using Relocation Table Entry
(RTE)typedef unsigned short int IEC_PARAM_RTE;
// The offset size in target dependent typedef
unsigned int IEC_PARAM_OFFSET;
// Internal type definition typedef struct  tag_IEC_RTE_OFFSET
{
    IEC_PARAM_RTE   block;
    IEC_PARAM_OFFSET offset;
} IEC_RTE_OFFSET;
// Parameter given using logical address : RTE + Offset
typedef IEC_RTE_OFFSET IEC_PARAM_RTE_OFFSET;
// Parameter given using logical address+length :
// RTE + Offset + Length(number of elements)
typedef struct tag IEC_PARAM_RTE_OFFSET_LG;
{
    EC_RTE_OFFSET IECPtr_Log;
    unsigned short int Length;
} IEC_PARAM_RTE_OFFSET_LG;
```

`s_log_to_phy()` System Function

The `s_log_to_phy()` system function is used to create a physical pointer (C pointer) from an `IEC_RTE_OFFSET` typed element.

Example:

`OUT` is used in a function block as an `IEC_RTE_OFFSET` logical pointer, on a `IECDWord` variable (unsigned long int).

Follow the rules below to access a variable referenced by `OUT` logical pointer:

1. Declare a local physical pointer on an `IECDWord` variable (e.g. `Phys_OUT`)
   ```
   IECDWord *Phys_OUT;
   ```
2. Initialize this local physical pointer with the conversion of `OUT` logical pointer with `s_log_to_phy` system function
   ```
   Phys_OUT = s_log_to_phy(OUT);
   ```
3. Manage any access to access variable referenced by `OUT` logical pointer with `*Phys_OUT;`

NOTE: The local physical pointer initialization should be done each time the function block is invoked.

Constant Values

Access to Constant Values

The following information describes how to use constant values.

Constant Values

The user has to complete the following declaration to access the constants.

Constant Declaration

The programmer has to declare constant values at the global level. The following program codes show the declaration.

```
// Constant declaration
const STRING32 cllbk_Tab2[32] = { "Motor 1", "Motor 2"
};s_Declare_Logical(cllbk_Tab2);
// Constant declaration
const IECInt cllbk_calendar[12] = {31,28,31,30,31,30,31,31,30,31,30,31}
;
s_Declare_Logical(cllbk_calendar);
const IECReal cllbk_Pi = (IECReal)3.1415999;
s_Declare_Logical(cllbk_Pi);
```

NOTE: You have to follow the below mentioned conventions:
- declaration should be global
- keyword `const` is mandatory
- constant name should start with `cllbk_`
- use of `s_Declare_Logical` macro is mandatory in the declaration

Usage at Local Level

Use the constant values in the procedure by the following declarations of local variables.

NOTE: It is necessary to declare the constants at the global level before. See *Constant Declaration, page 43*.

The following program code shows an example for the declaration of constants in the procedure.

```
// Declaration of local variables
// Declare and initialize a pointer on the same data type as the consta
nt to access
IECInt __far *phy_calendar = s_Const_Instance(cllbk_calendar);
IECReal __far * x = s_Const_Instance(cllbk_Pi);
```

NOTE: You have to follow the below mentioned conventions::

- declaration should be done locally where the constant is to be used
- use of `s_Const_Instance` macro is mandatory in the pointer declaration
- pointers should be declared as `__far` for 16-bit compatibility

Access to the constant in code is managed as follows:

```
// Access to the cllbk_calendar table can be managed with
phy_calendar maxDaysinAug =  phy_calendar[8];
// get the number of days of the month August
// Access to the cllbk_Pi value can be managed with x float y; if (y <
*x) then   // 3.1415999
```

Local Functions

Declaration of Local Functions

The following information describes how to use local functions.

Usual Case

If logical addresses in local functions are not needed, you should:

- start the subroutine names with `s_`
- prototype the function call model as `__near` for 16-bit compatibility
- use no more than four bytes (including `structs` and `unions`) for return values
- provide a parameter (a pointer to a buffer where the value will be returned) in the calling program to return `structs` or `unions` larger than four bytes
- do not use floating-point return values

Special Case

If logical addresses in local functions are needed (e. g., callback functions), you should:

- start the subroutine names with `cllbk_`
- prototype the function call model as `__near` for 16-bit compatibility
- use no more than four bytes (including `structs` and `unions`) for return values
- provide a parameter (a pointer to a buffer where the value will be returned) in the calling program to return `structs` or `unions` larger than four bytes
- do not use floating-point return values

Access to Logical Addresses

To get access to the address of a logical function, use the mechanism described in *Access to Constant Values*, *page 43*.

Address Declaration at Global Level

You have to declare the logical address. The following program code shows an example of how logical addresses are declared at the global level using cllbk_functioncall. functioncall as a place holder for the function name.

```
// Function declaration void cllbk_function(void)
{
}
// Logical address declaration of a local function
s_Declare_Logical(cllbk_function);
```

NOTE:
- declarations are GLOBAL
- function names start with cllbk_
- the s_Declare_Logical macro is mandatory in the declaration

The cllbk_function function should be declared in the C code the same as any other 'C' function.

Usage at Local Level

The following code example shows logical addresses declared at the local level using cllbk_function:

```
// At local level: Declare and Initialize a local logical
// structure IEC_RTE_OFFSET function, which contains the  logical
// address of local function cllbk_function
IEC_RTE_OFFSET function = s_Logical_Instance (cllbk_function);
```

NOTE:
- Declarations are done LOCALLY where the logical address is to be used.
- The s_Logical_Instance macro is mandatory in the declaration.

EFB Data Instances

EFB Data Instance Access

An EFB data instance is defined by a C structure with content that represents the EFB parameter's value. The following code shows an example for a data instance.

```
#pragma pack(8)
typdef struct {
   // Public variables
   // Internal variables
   // Function parameters
   // Input parameters
   // Output parameters
   // In/Out parameters
} MYEFB_INSTANCE_T;
#pragma pack()
```

The declaration of the EFB entry point is:

```
IECBool fb_call_model LanguageEntryPoint  (IEC_PARAM_OFFSET MYEFB_ins
```

`MYEFB_instance` is a logical pointer value (Block:Offset) to the `MYEFB` EFB data instance.

In order to access EFB parameters in the `LanguageEntryPoint` C code, you have to:

- declare and initialize a local pointer (e.g., `myefb_instance`) on the EFB instance data type (e.g., `MYEFB_INSTANCE_T`)

```
/* Declaration of a local pointer on EFB data instance */
MYEFB_INSTANCE_T *myefb_instance;
/* MYEFB_INSTANCE_T is the Struct, which describes MYEFB instance */

/* Initialization of a local pointer on EFB data instance */
myefb_instance = s_log_to_phy(MYEFB_instance)
/* initialize physical pointer from MYEFB_instance logical pointer
passed as EFB parameter */
```

- access any EFB instance content through the C syntax `myefb_instance->struct_element_name`, where `struct_element_name` as the targeted element to access.

NOTE: Parameters defined by values in the EFB structure, can be directly accessed by parameters of type `IEC_PARAM_OFFSET` (or `IEC_PARAM_RTE_OFFSET_LG`). They should be converted to a physical pointer (`s_log_to_phy`), which allows to the referenced element, refer to *Logical vs. Physical Address, page 41*.

EN/ENO Pins

EN and ENO in Function Blocks

In the Control Expert graphical languages (LD, optional in FBD), enable input (EN) and equivalent output (ENO) pins are available. The management of the EN pin is carried out completely by the system. If the value of EN is set to FALSE in the application, the 'C' code of the EFB is not carried out. The value of the ENO pin corresponds to the return value of the 'C' function of the EFB.

Diagnostic Management

All function blocks return FALSE or TRUE, according to detected error/detected warning situation to manage ENO pins:

```
RETURN (TRUE);   // ENO = TRUE (no error occurs)
```

```
RETURN (FALSE);   // ENO = FALSE (an error occurs)
```

Additionally the function blocks can manage their own system detected errors by calling the system diagnostic. The EFB-Toolkit provides two possible external prototypes for system functions:

```
s_set_ffb_error (int errno)
```

and

```
s_set_ffb_error_addi (int errno, int param
```

NOTE: You can analyze the detected error code with the diagnostic viewer in Control Expert.

Here are the standard register detected error number errno messages in the diagnostic buffer:

- errno < 0
  In this case, the value returned by the function block should be FALSE, setting the ENO pin to FALSE.
  The diagnostic viewer will display:
  ERROR FFB /errno/
- errno > 0
  In this case, the value returned by the function block should be TRUE, setting the ENO pin to TRUE.
  The diagnostic viewer will display:
  WARNING FFB /errno/

A param is optional. If it is used, it is included in the standard message as an additional diagnostic code.

- `errno < 0`
  In this case, the value returned by the function block should be `FALSE`, setting the `ENO` pin to `FALSE`.
  The diagnostic viewer will display:
  ERROR FFB `/errno/ :param`
- `errno > 0`
  In this case, the value returned by the function block should be `TRUE`, setting the `ENO` pin to `TRUE`.
  The diagnostic viewer will display:
  WARNING FFB `/errno/ :param`

A set of 100 numbers (`errno`) has been reserved for detected error defined by user:

30200 - 30299 - Detected error defined by user 1 to 100

Output Values (ENO)

One of the ways the IEC specification distinguishes between EFs and EFBs is as follows:

- If the `ENO` output is evaluated to `FALSE` (0), the values of EF outputs (`_OUTPUT`, `_IN_OUT` and result) are considered undefined. You cannot use the output values of EFs when the `ENO` is set to FALSE.
- If the `ENO` output is evaluated to `FALSE` (0), the values of the EFB outputs (`_OUTPUT`) keep their states from the previous invocation. The code generation does not manipulate the states from the previous invocation. Nevertheless, confirm that the function block is called at least once.

Extensible Pins

Naming Convention for Extensible Pins and the Algorithm to Generate Names

An identifier is a sequence of letters, numbers, and underlines beginning with a letter or an underline (for example, name of a function block type, an instance, a variable, or a section). You can use the letters from national character sets (for example, ö, ü, é, õ) except in project and EFB.

Underlines are significant in identifiers (for example, A_BCD and AB_CD) are interpreted as different identifiers. Multiple leading underlines and consecutive underlines are invalid.

Identifiers cannot contain spaces and are not case-sensitive (for example, ABCD and abcd are interpreted as the same identifier).

Access to Extensible Pin Values

For different function blocks (`AND_BOOL`, for example) there is no specific number of input pins. You may expand the EF to a variable number of input pins. The EF may have a variable number of input parameters as a result of the extensible pins.

The `va_arg`, `va_end` and `va_start` macros provide access to function arguments when the number of arguments is variable. These macros are defined in `STDARG.H` for ANSI'C' compatibility.

Based on these macros, the EFB Toolkit provides specific macros for accessing extensible pin values:

- `open_VARG` - sets a pointer to the beginning of the argument list for extensible pins
- `next_VARG` - retrieves an argument from the extensible pins argument list
- `close_VARG` - resets the pointer

In the EF 'C' prototype, a `nin` parameter contains the effective number of extensible pins.

The following code shows an example for extensible pins.

```
Example: MIN with BYTE extensible pins
// Macros for accessing extensible pin values
#define open_VARG { va_list _theVariableList;
 va_start(_theVariableList, Y)
#define next_VARG va_arg(_theVariableList, IECReal)
#define close_VARG va_end(_theVariableList); }
#define NIN_MIN 2 // the minimum value for parameter nin
#define NIN_MAX 31 // the maximum value for parameter nin
IECBool fb_call_model MAX (
      const NIN_T nin,            // EXTENSIBLE PIN -
 Number of extra parameters
      IEC_PARAM_RTE_OFFSET OUT,  // RETURN VALUE - Output
      //const IECByte IN1           // EXTENSIBLE PIN - Input 2..31
      ...)
{
```

```
IECByte far  pOUT;
IECInt      incount;
IECByte     tempmax;
IECByte     tempmaxh;
pOUT = (IECByte far *)s_log_to_phy(OUT);   //create
physical pointer from logical pointer
open_VARG;   // Set pointer to beginning of
extensible pins argument list
incount = nin;   // effective number of extensible pins
incount--;
tempmax = next_VARG;   // get the 1st extensible pin  value (IECByte)
do {
    if ((tempmaxh = next_VARG) <= tempmax)   // get the
next extensible pin value (IECByte)
    continue;
    while (--incount > 0) {
        if ((tempmax = next_VARG) > tempmax)   // get the
next extensible pin value (IECByte)
           goto first_part;
    }
    tempmax = tempmaxh;
first_part:
} while (--incount > 0);
close_VARG;
*pOUT = tempmax;
return TRUE;   // ENO value
}
```

Conditional Compilation for Dynamic Alignment

M340 (mid range) CPUs are based on ARM processors. The ARM architecture normally expects the memory access to be aligned suitably. In particular, align the address used for a double word access (32 bit) to a 4-byte border (alignment 4). An unaligned memory access leads to an alignment fault detected on such a processor. Other CPUs are based on Intel processors. The alignment for the Intel architecture is variable. An alignment of 2 is chosen for instance data on Intel-based CPUs.

Until now, the PLC simulator acts like an Intel-based CPU regardless of the simulated platform. The data alignment on the simulator and the simulated platform differs in case of a M340 CPU. For some data (EDT or DDT with size of 4 bytes or more) that are mapped on direct addresses, the behavior may differ. This is a strong drawback for the user - because a program tested with the simulator may behave differently on the real platform.

To support the generation of a simulated M340 application or a real premium application with alignment 4, the libraries needed - including customer and third-party libraries - contain Intel object files generated with alignment 4 for each C coded EF or EFB. If such an object file is unavailable, Control Expert cannot link the application.

The libraries have to provide 2 Intel object files for each C coded function block. 1 object file is generated with alignment 2 and a second object file is generated with alignment 4. These 2 different object files are necessary to build Control Expert applications with alignment 2 or 4 for the simulator or real premium platform. Depending on the alignment setting, Control Expert links the application with the related object file from the libset.

For an alignment issue, use the below mentioned `pragma` inside the FFB/DDT as follows:

```
#if defined(__arm__)
      #pragma pack()
# else
      #pragma pack()
# endif
```

The memory layout of a DDT changes from alignment 2 to 4, when a DDT element with a size of at least 4 bytes (for example, `DWORD`, `REAL`) is mapped on an address that is unaligned for alignment 4.

The table shows the offset alignment 2 and 4 for the various elements:

Element	Type	Offset Alignment 2	Offset Alignment 4
Word_1	WORD	0	0
Word_2	WORD	2	2
Word_3	WORD	4	4
Dword_1	DWORD	6	8
Word_4	WORD	10	12
Real_1	REAL	12	16
Word_5	WORD	16	20

Chapter 4
Programming Examples

Summary

This chapter provides programming examples of elementary function (EFs), elementary function blocks (EFBs) and derived data types (DDTs).

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
EF/EFB Example	54
EF Program Code	57
EFB Header File	60
EFB Source Code	64
DDT Example	66

EF/EFB Example

General

This example will show how to generate a standardized header and source frame for the user defined function blocks (EF or EFB).

EF and EFB

The representation of EFs and EFBs for the Control Expert user is the same. The user of the EFB Toolkit can select to create an EF or EFB. The difference between EFs and EFBs is the possibility for the developer to use internal states of variables within EFBs.

For the software developer the EF consists of inputs and outputs. For creating EFBs the developer can use public/private variables and inputs/outputs additionally to the usual input and output pins, e. g. for storing internal states of the function block (see also *Introduction, page 11*).

Each EFB has instance data.

First Steps

This table describes the steps to create an EF.

Step	Action	Comment
1	Start the EFB Toolkit	–
2	**File → New family** or **File → Open family**	Enter a family name or select a available family. **Result:** A family will be opened or created.
3	**Current family → Create EF / EFB**	Select the block type EF/EFB and enter a name. For example `Circle`. An EF within an EF description will be created. The user is now able to edit the pins and description.
4	Select `Procedure` or `Function` as a kind of the EF.	–
5	Insert data about the EF/EFB definition and Template definition (pins).	For example: ● Input pins Name: `R` Type: `Real` ● Output pins Name: `AREA` Type: `Real` Name: `CIRCUM` Type: `Real`
6	After input of the EF information and pins, the user has to generate code by using **Current object → Analyse and Generate code**.	The EFB Toolkit generates a standardized frame for the `EFB header` and `EFB source`.

EF Header Example

The following program code shows the header frame which will be generated by the EFB Toolkit.

```
// SDKC_HEADER_BEGIN  Do not edit. Any modification would be lost
// Filename : C:\Programme\Schneider Electric\
Unity EFB Toolkit\SDKC\FFB
//                   Dev\example_efb\code\CIRCLE.h
// PROCEDURE         CIRCLE
// Version           1.0
// Author            Auto generated EF/EFB
/*
This is the comment of the demonstration EFThis is the
descriptive form of the demonstration EF
*/
#include "SystemLib.h"
// Additional header :
// None
//       +----------------------+
//       |        CIRCLE        |
//       +----------------------+
//       |                      |
//     --+- R             AREA -+---
//       |                      |
//       |              CIRCUM -+---
//       |                      |
//       +----------------------+
//
IECBool fb_call_model CIRCLE(
      const IECReal R,              // This is the input pin
      IEC_PARAM_RTE_OFFSET AREA,    // Circle area
      IEC_PARAM_RTE_OFFSET CIRCUM   // Circle circumference
);
      SDKC_HEADER_END
// TODO : Write here additionnal declarations.
```

EF Source Example

The following program code shows the source frame which will be generated by the EFB Toolkit.

```
//  SDKC_HEADER_BEGIN  Do not edit.  Any modification would be lost
// Filename: C:\Programme\Schneider Electric\ Unity EFB Toolkit\SDKC\
//               FFBDev\example_efb\code\CIRCLE.c
// PROCEDURE:     CIRCLE
// Version:       1.0
// Author:        Auto generated EF/EFB
/*
This is the comment of the demonstration EF
This is the descriptive form of the demonstration EF
*/
#include "CIRCLE.h"
//      SDKC_HEADER_END
// TODO : Write here additionnal declarations.
//      SDKC_PROTOTYPE_BEGIN  Do not edit.
Any modification would be lost
IECBool fb_call_model CIRCLE(
      const IECReal R,             // This is the input pin
      IEC_PARAM_RTE_OFFSET AREA,   // Circle area
      IEC_PARAM_RTE_OFFSET CIRCUM  // Circle circumference
)
//}} SDKC_PROTOTYPE_END
{
      // TODO : Write here variables declarations.
      // TODO : Write here the code for your function block.
      return TRUE;      //     ENO value
}
```

EF Program Code

General

The following provides information about the integration of the user defined program code for the new EF.

Circle Example

The circle example calculates the circumference and area of a circle. The function block gets the information of the radius via an input pin.

Circle Header File Example

There are no changes in the header file.

Circle Source File Example

The following program code shows the source file within the generated code of the EFB Toolkit and the user defined code for calculating the area and circumference of the circle.

```
//       SDKC_HEADER_BEGIN  Do not edit. Any modification
would be lost
// Filename:  D:\Schneider Electric\EFBToolkit\FFBDev\
Sample1\code\CIRCLE.c
// PROCEDURE: CIRCLE
// Version:   1.0
// Author:    Auto generated EF/EFB
/*
This is the comment of the demonstration EF
This is the descriptive form of the demonstration EF for the floting po
int constants declaration and their usage.
*/
#include "CIRCLE.h"
//      SDKC_HEADER_END
// TODO : Write here additionnal declarations.
//      Floating point constants declaration const float cllbk_PI = (fl
oat)3.1415999;
      s_Declare_Logical(cllbk_PI);
      const float cllbk_2_0 = (float)2.0;
      s_Declare_Logical(cllbk_2_0);
//      SDKC_PROTOTYPE_BEGIN  Do not edit. Any modification
would be lost IECBool fb_call_model CIRCLE(
      const IECReal R,              // This is the input pin
      IEC_PARAM_RTE_OFFSET AREA,    // Circle area
      IEC_PARAM_RTE_OFFSET CIRCUM   // Circle circumference
)
```

```
//      SDKC_PROTOTYPE_END
{
    // TODO : Write here variables declarations.
       IECReal *pArea, *pCircum, pi, c2;
    // TODO : Write here the code for your function block.
       pArea = s_log_to_phy( AREA );
       pCircum = s_log_to_phy( CIRCUM );
      pi  = *((IECReal *)s_Const_Instance(cllbk_PI));
      c2 =  *((IECReal *)s_Const_Instance(cllbk_2_0));
    // Area calculation
      *pArea  = (IECReal)( pi * R * R );
    // Circumference calculation
      *pCircum= c2 * pi * R;
      return TRUE ;   // ENO value
}
```

Circle Source File Differences

The following program code shows the differences between the automatically generated source file of the EFB Toolkit and the user defined code for circle example.

The following code segment is an example for the declaration area of constants.

```
// TODO : Write here additionnal declarations.
//      Floating point constants declaration
const float cllbk_PI = (float)3.1415999;
      s_Declare_Logical(cllbk_PI);
      const float cllbk_2_0 = (float)2.0;
      s_Declare_Logical(cllbk_2_0);
```

The macro `s_Declare_Logical` associates a symbol with the constant. This macro is necessary to get access to the constants.

The following code segment is an example for the prototype declaration area.

```
//       SDKC_PROTOTYPE_BEGIN  Do not edit. Any modification
would be lost IECBool fb_call_model CIRCLE(
      const IECReal R,              // This is the input pin
      IEC_PARAM_RTE_OFFSET AREA,    // Circle area
      IEC_PARAM_RTE_OFFSET CIRCUM   // Circle circumference
)
//      SDKC_PROTOTYPE_END
```

This will be generated by the EFB Toolkit. It is the instantiation of the input and output pins.

The prototype `fb_call_model` helps to ensure compatibility with the 16bit environment.

The following code segment is an example for the conversion of addresses.

```
{
    // TODO : Write here variables declarations.
      IECReal *pArea, *pCircum, pi, c2;
    // TODO : Write here the code for your function block.
      pArea = s_log_to_phy( AREA );
      pCircum = s_log_to_phy( CIRCUM );
```

The macro `s_log_to_phy` converts logical addresses to physical pointer to access the parameter (pin value).

The following code segment is an example for the user functions.

```
      pi  = *((IECReal *)s_Const_Instance(cllbk_PI));
      c2 =  *((IECReal *)s_Const_Instance(cllbk_2_0));
    // Area calculation
      *pArea  = (IECReal)( pi * R * R );
    // Circumference calculation
      *pCircum= c2 * pi * R;
      return TRUE ;   // ENO value
```

The physical pointer accesses the constant using the macro `s_Const_Instance`.

EFB Header File

General

The following text provides information about the code header file for a new EFB.

The difference between an EF and an EFB is an EFB's ability to intercept two different events. You can use the `LanguageEntryPoint` event and the `SystemEntryPoint` event.

The EF deploys just the `LanguageEntryPoint` event, and it is not explicitly listed in the EF header file.

LanguageEntryPoint

The `LanguageEntryPoint` is activated by the PLC when the program is using the EFB within a section.

SystemEntryPoint

The `SystemEntryPoint` is activated by system events in the PLC, independent of the user program. The `SystemEntryPoint` is not known by the application; it is present, even if the functions are empty.

The following table shows the possible system events. The events are defined in `Systemlib.h`

Event	Indication
pu_INIT_S0	Called at the init of an application, at the end of a download and at a cold start. System bit %S0 = 1
pu_INIT_CONF	Called at the beginning of a cold start when a valid application is detected.
pu_INIT_RECONF	Called at the beginning of a download.
pu_INIT_CLEAR	Used for EFBs having private data. It it used to tell the EFB to clear its data area without destroying the private data.
pu_WARM	Called at the beginning of a warm restart.
pu_RESTART	Called at the end of a warm start.

NOTE: Avoid using the `SystemEntryPoint` in the EFB Toolkit. In case of a serious problem, contact Schneider Electric support.

SINCOS Example

The SINCOS example demonstrates the use of sine and cosine floating point functions. It calculates the `sin(a) * sin(a) + cos(a) * cos(a)` equation and generates a detected error when the result is not 1.

SINCOS Header File Example

The header file is generated by the EFB Toolkit. There are no changes in the header file.

The following code shows the header file in the EFB Toolkit generated code:

```
//{{ SDKC_HEADER_BEGIN  Do not edit.  Any modification would be lost
// Filename : D:\Documents And Settings\gschmidt\
My Documents\Projects\
//                SDKC\Samples\Family1\code\SINCOS.h
// EFB:     SINCOS
// Version: 1.1
// Author:
/*Sine and cosine usage
This sample demonstrates the usage of sine and  cosine floating
point functions. It calculates sin(a)*sin(a) +
cos(a)*cos(a) equation and
generates an arithmetic error in case the resulting  value isn't 1.
It also reports an error in case the operation fails.
*/
#include "SystemLib.h"
// Additional header :
// None
//                 +-----------------------+
//                 |        SINCOS         |
//                 +-----------------------+
//                 |                       |
//               --+- ALPHA         OUTP -+--
//                 |                       |
//                 +-----------------------+
//
#pragma pack(8)
typedef struct {
// Public variables...
// Internal variables...
// Function parameters...
IECReal ALPHA;                // Radian value
IECReal OUTP;                 // Function result
} SINCOS_INSTANCE_T ;
#pragma pack()
#ifdef EFFBDBG
#undef LanguageEntryPoint
#undef SystemEntryPoint
#define LanguageEntryPoint LanguageEntryPoint_SINCOS
#define SystemEntryPoint SystemEntryPoint_SINCOS
#endif
```

```
IECBool fb_call_model LanguageEntryPoint(
PTR_LOG SINCOS_instance// Logical address of a  SINCOS_INSTANCE_T
) ;
extern IECByte fb_call_model SystemEntryPoint(
IECUInt type,
PTR_LOG SINCOS_instance
) ;
//}} SDKC_HEADER_END
// TODO : Write here additional declarations.
```

SINCOS Header Description

The following text is from individual parts of the program code in the header file that describe input/output parameters and variables. The parts are automatically generated by the EFB Toolkit.

The following code is a general description of the header file.

```
//{{ SDKC_HEADER_BEGIN  Do not edit.  Any modification would be lost
// Filename : D:\Documents And Settings\gschmidt\
My Documents\Projects\
//              SDKC\Samples\Family1\code\SINCOS.h
// EFB:     SINCOS
// Version: 1.1
// Author:
/*
Sine and cosine usage
This sample demonstrates the usage of sine and  cosine floating
point functions. It calculates sin(a)*sin(a) +
cos(a)*cos(a) equation and
generates an arithmetic error in case the resulting  value isn't 1.
It also reports an error in case the operation fails.
*/
#include "SystemLib.h"
// Additional header :
// None
//    +----------------------+
//    |        SINCOS        |
//    +----------------------+
//    |                      |
//  --+-- ALPHA         OUTP -+--
//    |                      |
//    +----------------------+
//
```

The following code shows the definition of the input and output parameters and the internal and public variables. Public variables can be used and modified during the runtime. They keep the information stored independent of the cycle or state of the PLC. Internal variables cannot be used by the PLC program.

```
#pragma pack(8)
typedef struct {
// Public variables...
// Internal variables...
// Function parameters...
IECReal ALPHA;                  // Radian value
IECReal OUTP;                   // Function result
} SINCOS_INSTANCE_T ;
#pragma pack()
#ifdef EFFBDBG
#undef LanguageEntryPoint
#undef SystemEntryPoint
#define LanguageEntryPoint LanguageEntryPoint_SINCOS
#define SystemEntryPoint SystemEntryPoint_SINCOS
#endif
```

The following code shows the description of the `LanguageEntryPoint`. The `LanguageEntryPoint` is automatically used by the PLC program to start the function of the EFB.

```
IECBool fb_call_model LanguageEntryPoint(
PTR_LOG SINCOS_instance// Logical address of a  SINCOS_INSTANCE_T
) ;
```

The following code shows the description of the `SystemEntryPoint`. The `SystemEntryPoint` is activated by system events. See also *SystemEntryPoint, page 60*.

```
extern IECByte fb_call_model SystemEntryPoint(
IECUInt type,
PTR_LOG SINCOS_instance
) ;
//}} SDKC_HEADER_END
// TODO : Write here additional declarations.
```

EFB Source Code

General

The following text provides information about the integration of user-defined program code into a new EFB.

SINCOS Source File Example

The following code shows the source file within the generated code of the EFB Toolkit and the user-defined code for calculating `sin(a) * sin(a) + cos(a) * cos(a).`

```
//{{ SDKC_HEADER_BEGIN  Do not edit.  Any modification would be lost
// Filename : \SDKC\Samples\Family1\code\SINCOS.c
//
// EFB:     SINCOS
// Version: 1.1
// Author:
/*
Sinus and cosinus usage
This sample demostrates the usage of sine of cosine
floating point functions. It calculates sin(a)*sin(a)
+ cos(a)*cos(a) equation and generates an arithmetic
error in case the resulting value isn't 1. It also
reports an error in case the operation fails.
*/
#include "SINCOS.h"
//}} SDKC_HEADER_END
// TODO : Write here additional declarations.
// Floating point constans declaration.
const float in_Code cllbk_1_00 = (float)1.00;
s_Declare_Logical(cllbk_1_00);
//{{ SDKC_PROTOTYPE_BEGIN  Do not edit.  Any modification would be lost
IECBool fb_call_model LanguageEntryPoint(
     PTR_LOG SINCOS_instance   // Logical address of a  SINCOS_IN-
STANCE_T
)
//}} SDKC_PROTOTYPE_END
{
     IECReal c1, alpha;
     double  result;
     SINCOS_INSTANCE_T *pInstData;
     c1 =  *((IECReal *)s_Const_Instance(cllbk_1_00));
// Get physical address to instance data
pInstData = s_log_to_phy( SINCOS_instance );
alpha = pInstData->ALPHA;
```

```
result = sin(alpha) * sin(alpha) + cos(alpha) * cos(alpha);
if( result != c1 )
{
     s_set_ffb_error_addi( E_ERR_FLOAT, (unsigned  short)_status87() );
     return FALSE;
{
// Return resulting value
     pInstData->OUTP = (IECReal)result;
return TRUE ;     // ENO value
}
//{{ SDKC_SYSTEM_BEGIN  Do not edit.  Any modification would be lost
extern IECByte fb_call_model SystemEntryPoint(
IECUInt type,
PTR_LOG SINCOS_instance
)
//}} SDKC_SYSTEM_END
{
// TODO : Write here the code for the system call.
return 0 ;
}
```

Instance Data

The instance data are defined within the header file. They consist of the input and output pins as well as the internal and external variables.

You may use the two functions `LanguageEntryPoint` and `SystemEntryPoint` to work with instance data.

The following code shows an extract of the instantiation of the data:

```
IECBool fb_call_model LanguageEntryPoint(
     PTR_LOG SINCOS_instance   // Logical address of a  SINCOS_IN-
STANCE_T
)
extern IECByte fb_call_model SystemEntryPoint(
IECUInt type,
PTR_LOG SINCOS_instance
)
```

You can react to a system event by using the parameter type `IECUInt type`. See *SystemEntryPoint, page 60*.

DDT Example

Creating a DDT

You may create data structures or arrays. After they have been created, the DDTs are available to use in EFs or EFBs.

First Steps

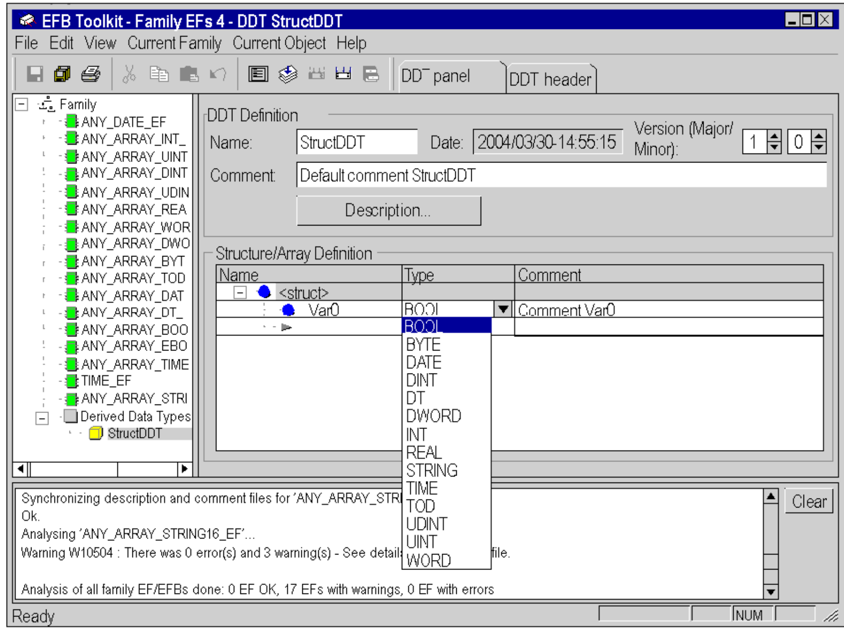
To create a DDT:

Step	Action	Comment
1	Start the EFB Toolkit	–
2	Select **File → New family** or **File → Open family**.	–
3	Enter a family name or select an available family.	A family is opened or created.
4	Select **Current family → Create DDT**. Then enter a DDT name and select the type `Array` or `Structure`. For example `testddt`.	A DDT within an DDT description is created. You may now edit the structure/array and description.
5	Edit the structure or array of the generated DDT.	–

DDT Header File

When a new DDT is created, the EFB Toolkit generates a header file. You may edit this header file.

The following figure shows the program window for selecting the DDT information for a structure:



You may edit the following structure definitions:

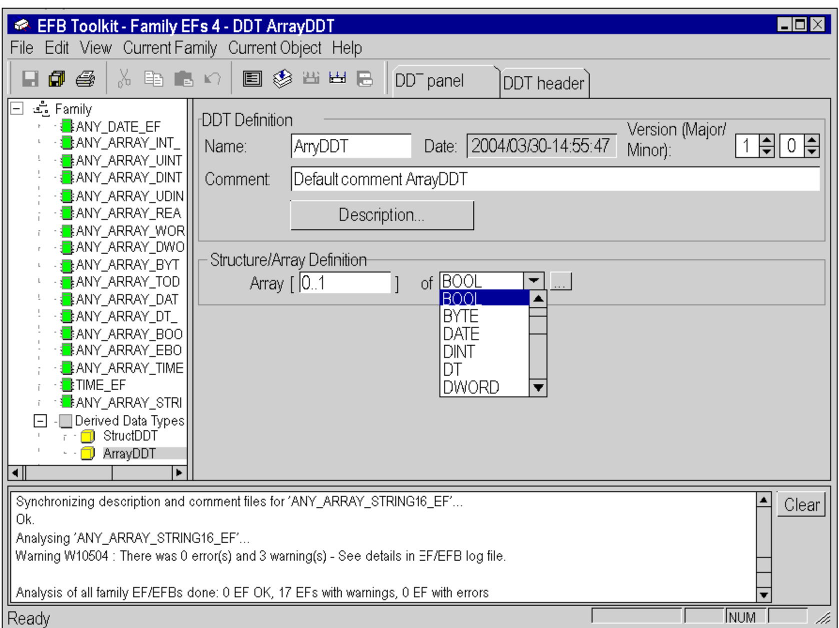
- name, version, comment, descriptive form
- structure definition
  To create a new structure entry, double-click on the empty cell **Name** (marked with an arrow). Then select a name and type of the data entry. A comment is optional.

The following examples are from the `sampleFamily1`, which is delivered with the EFB Toolkit.

The following code shows a sample header file for a DDT generated by the EFB Toolkit:

```
//      BLOCK_DDT_BEGIN  Do not edit.  Any modification would be lost
//DefaultComment testddt
#pragma once
#pragma pack(2)
typedef struct testddt
{      IECBool Var0;    //comment var 0
       IECBool test;    //
}testddt;
#pragma pack()
//      BLOCK_DDT_END
```

The following figure shows the window you need to use to select the DDT information for an array:



You may edit the following array definitions:

- name, version, comment, descriptive form
- array size
- array type

The following code shows a sample header file for an array DDT generated by the EFB Toolkit:

```
//{{ BLOCK_DDT_BEGIN  Do not edit.  Any modification would be lost
//DefaultComment testt
#pragma once
#pragma pack(2)
typedef IECBool ELEM[2];
#pragma pack()
//}} BLOCK_DDT_END
```

NOTE: The code is generated automatically by the EFB Toolkit when you save the edited DDT.

Nested DDTs

If you have nested DDTs, you have to take care of the correct order of nested DDTs. The correct order is: n → n–1 → n–2 → ... → n2 → n1

Chapter 5
Debugging

Introduction

This chapter provides an overview of how Microsoft Visual C++ .NET can be used to debug an EF/EFB.

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
Debug Preparation	72
Best Practices	79
Recommendations	80

Debug Preparation

General

After generating the installable form of the family with the EFB Toolkit and after installing it in the Control Expert Type Library, you can debug the function blocks using Microsoft Visual C++ .NET.

First Steps

To prepare an EF/EFB for debugging:

Step	Action	Comment
1	Develop and build an EF/EFB	–
2	Select **Current Family → Debug all EF/EFBs**.	The files needed for an MS Visual C++ project workspace are created in the Debug sub-directory in the family development directory.

The MS Visual C++ project workspace is used to build a DLL for the Control Expert PLC Simulator. This DLL contains EFs/EFBs in the family and some additional code to support function block debugging with the Control Expert PLC Simulator.

The project workspace consists of the following files, which reside in the `\debug` folder of the family directory:

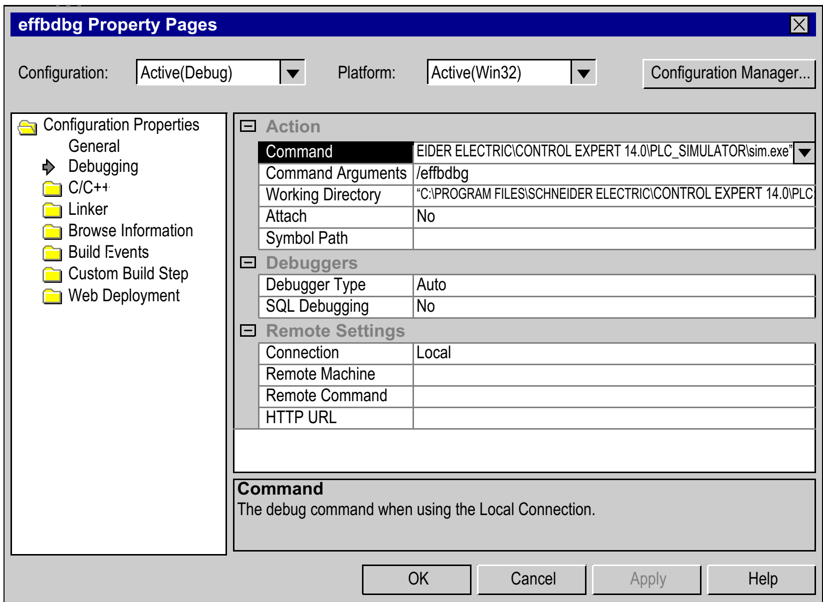
File	Description
`Effbdbg.dsw`	The workspace used to start the debug session.
`Effbdbg.dsp`	The Visual C++ project file.
`DbgMain.c`	Contains a function that is called from the standard effblib.dll of the Control Expert PLC Simulator after an application has been downloaded. This function does the routing of function block calls from the Control Expert application to your dynamic link-library `effbdbg.dll` generated by this Visual C++ project. This library contains your EF/EFB code.
`PostBuildExec.bat`	The batch file executed at the end of the project build process. It copies the generated DLL into the PLC Simulator directory.

When the program is ready to start the debug session, double-click the `effbdbg.dsw` project workspace. MS Visual C++ .NET automatically converts the `effbdbg.dsw` file to `effbdbg.vcproj` for later use in the .NET environment. Launch the project conversion inVisual C++ by clicking `Yes` in the popup window.

The project settings contain a path to the Control Expert PLC Simulator as the executable for the debug session (`sim.exe`). The project settings also contain the program argument (`/effbdbg`), which enables the EF/EFB debug feature in the PLC Simulator.

Settings

The following figure shows an example of the project settings in debugging mode:



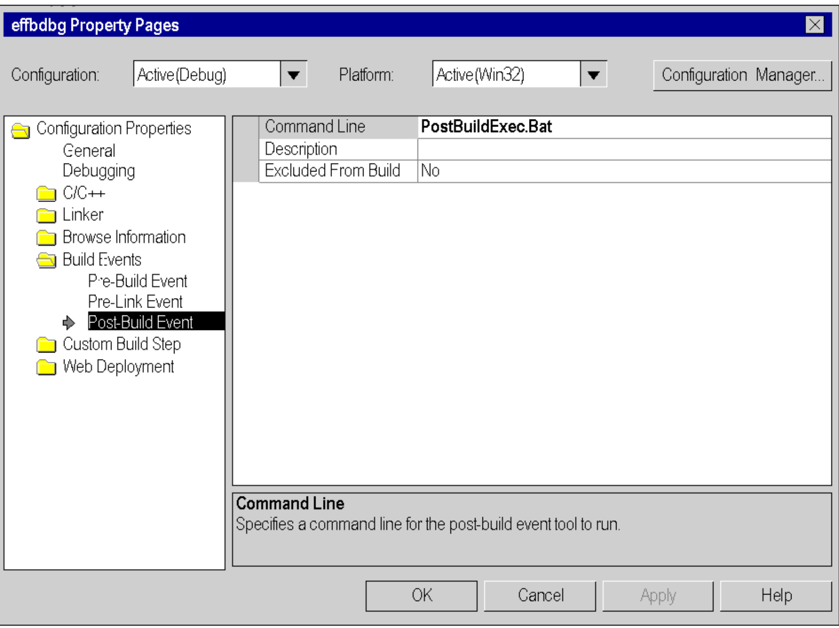
The project settings also contain a post-build command that can be used to copy the generated DLL `effbdbg.dll` to the Control Expert PLC Simulator directory. The used batch file is automatically created by the EFB Toolkit while executing the `Debug all EF/EFBs` menu command.

By default, the EFB Toolkit uses a registry entry to find the location of the PLC-Simulator and creates the commands that are inserted into the `PostBuildExec.Bat` batch file. For this to work, a version of Control Expert has to be installed on the PC.

NOTE: The batch file could also contain a copy command for the standard PLC Simulator component `effblib.dll` in the case where the DLL has to be replaced by a newer version brought in with the EFB Toolkit.

Post-Build

The following figure shows the post-build settings:

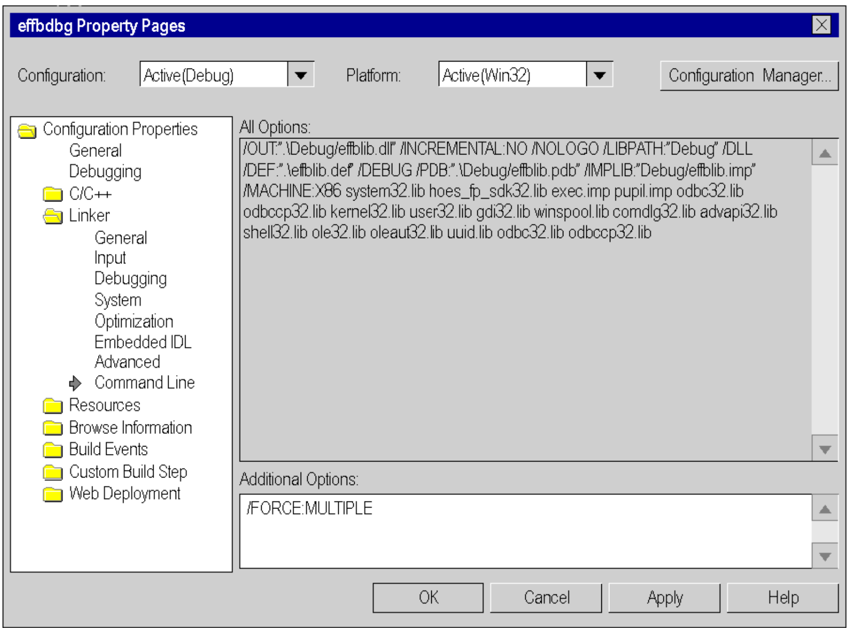


Visual C++ .NET

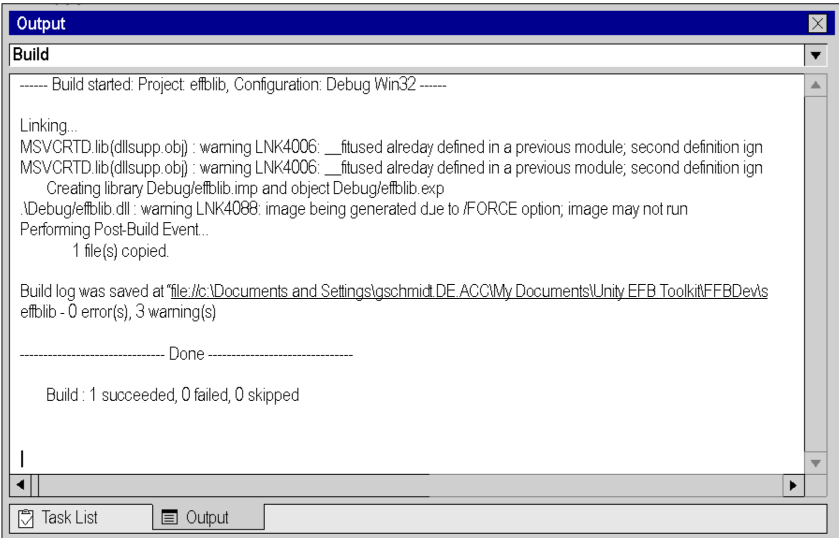
The following linker options are mandatory when you work with Visual C++ .NET:

- For a successful build, enter `/FORCE:MULTIPLE` as the linker `Command Line` option.
- You should change the `Linker General Option: Enable Incremental Linking` from "`NO (/INCR...)` to `Default`. This change suppresses the `warning LNK4075` message.

The following figure shows the necessary linker options:



After completing the above preparations, start the build process. The following figure shows an example of the output of the build process:



The output shows two detected warnings (`LNK4006`) for the `__fltused` symbol references.

NOTE: These detected warnings can be ignored.

NOTE: If the copy process is successful, the build output window shows up to two result lines for file copy operations, as shown above.
If the copy process is not successful, check the environment before continuing with the next steps.

Start Debugging

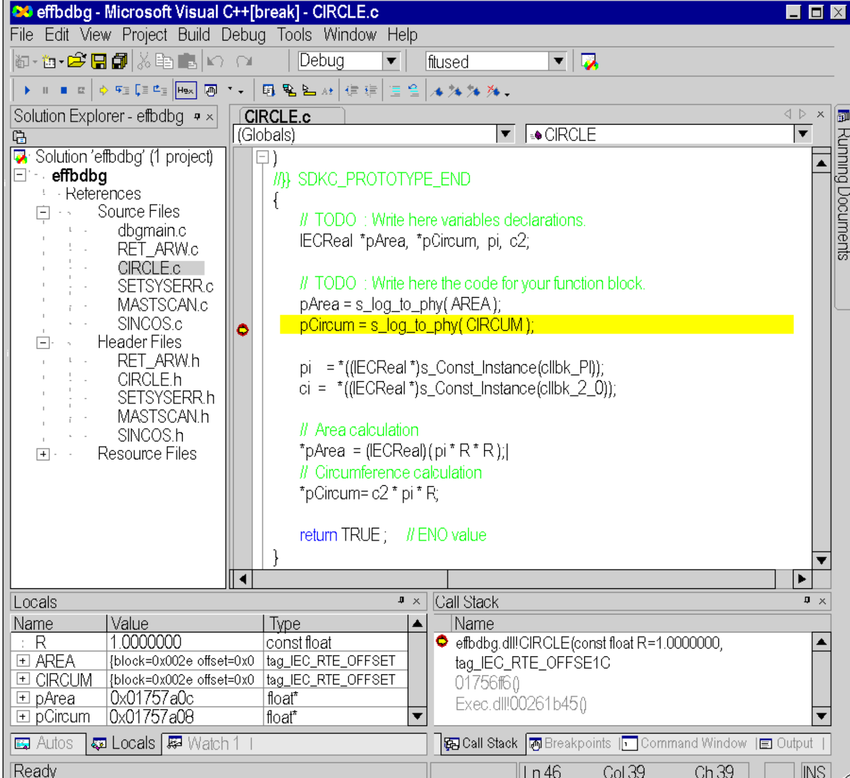
When the build process completes successfully, start the debug session by pressing the F5 key or via a menu command in MS Visual C++. (For additional information, refer to the MS Visual C++ documentation.)

This action launches the PLC Simulator with the newly generated DLL containing the function blocks to debug.

NOTE: A notification windows will appear to inform the programmer the `Simulator` does not contain any symbol. The programmer has to select the check box and press **OK**. During the debugging session the programmer will not be able to debug the simulator code, but the EF/EFB code can be debugged without limitations.

Optionally, the programmer may set a breakpoint in function block code by selecting a source code line and pressing the F9 key or by using the context menu accessed by right-clicking the mouse.

The following figure shows an example for the debugging window:



The following list contains additional debugging information:

- Create an application in Control Expert that contains the newly developed function blocks.
- Build the application and download it to the PLC Simulator. Control Expert uses the running PLC Simulator to download the application. Therefore you have to start debugging first, then connect to PLC Simulator with Control Expert.
- When the application is running, the function block code inside the DLL is executed. When a breakpoint is reached, the execution will stop. At this point, you may use MS Visual C++ debugging features, e.g., viewing variables, single stepping, etc.

Summary

Option	Part	Value
Debugging	Command	C:\Program Files (x86)\Schneider Electric\Control Expert 14.0\PLC_Simulator\sim.exe
	Command Arguments	/effbdbg
	Working Directory	C:\Program Files (x86)\Schneider Electric\Control Expert 14.0\PLC_Simulator
Linker : General	Enable Incremental Linking	Default
Linker : Command Line	Additional Options	/FORCE:MULTIPLE

Best Practices

Best Practice

During a debug session the DLL program code is executed and not the function block code inside the Control Expert application. This knowledge can be used to continue the development completely within an MS Visual C++ programming environment. The following section explains how to proceed:

- Create a framework of the family that includes DDTs and the complete function block template definition. At this time, you do not need to insert statements in the `*.c` files of the function blocks created by the EFB Toolkit.
- Build the empty function block family framework in the EFB Toolkit and install it in the Control Expert Type Library.
- Start Control Expert and create an application that uses the next function block to be debugged.
- Create the debugging environment in the EFB Toolkit with the `Debug all EF/EFBs` menu command.
- Start the workspace file `effbdbg.dsw` in the `\debug` folder of the family directory.
- Complete the project settings as described in this document (see *Debug Preparation, page 72*).
- Build the debugging dynamic library (`effbdbg.dll`) containing the empty function blocks.
- Start the debugging session in Visual C++ (F5 key).
- Connect Control Expert to the running PLC Simulator and download the Control Expert application.
- Stop the debug session and start the function block code implementation inside Visual C++ if the steps above are working properly.

NOTE: Do not change source sections created from the EFB Toolkit with a comment text such as `// DO NOT EDIT.`

- You do not need to leave Visual C++ until the requirements have changed a function block template or DDT. If this happens, you have to go back to the beginning of the list. Do not forget to install the modifications in Control Expert and to adapt it to the Control Expert application changes.
- If the function block code works well, go back to the EFB Toolkit, rebuild the family and install it again in Control Expert. Now the Control Expert Type Library contains the implemented and tested code.
- If Visual C++ and the EFB Toolkit are open at the same time, you should be able to switch from one application to the other. The two tools detect the modifications and reload the affected files.

Recommendations

Recommendations

1. When you are debugging a function block with the Visual C++ debugger, the Control Expert application debugging features are not available. Under an MS Windows operating system, only one debugger can be attached to a process at a time.
2. The connection from Control Expert to the PLC Simulator interrupts when a breakpoint is reached. You may reconnect Control Expert when the debugging session is not on a breakpoint.
3. Do not modify project and workspace files used for function block debugging, especially the project extension with additional function blocks.

Files externally modified may not be read correctly by the EFB Toolkit.

FILE MODIFICATION HAZARD

Do not modify files outside the EFB Toolkit.

Failure to follow these instructions can result in injury or equipment damage.

Chapter 6
System Services

Summary

This chapter provides an overview of the system services in the Control Expert PLC OS.

What Is in This Chapter?

This chapter contains the following sections:

Section	Topic	Page
6.1	Overview	82
6.2	Description	86

Section 6.1
Overview

System Service Overview

General

The following tables contains system services of the EFB Toolkit and a short description.

Objects Access

Function	Discription	File Interface
s_AliGetProgStat	Gets program specific status information.	SystemLib.h
s_log_to_phy	Converts a logical address into a physical address.	SystemLib.h
s_obj_to_log	Gets the logical address of an application object (%S, %SW, %M, %MW,).	SystemLib.h
s_obj_nbr	Gets the number of located objects.	SystemLib.h
s_rd_16bits	Reads 16 consecutive bits in EBOOL Memory.	SystemLib.h
s_rd_1bit	Reads one bit value in EBOOL Memory.	SystemLib.h
s_rd_bit_attrib	Reads attributes bits (F, H, D) in EBOOL Memory.	SystemLib.h
s_rd_internalwords	Reads N consecutive %MWi words.	SystemLib.h
s_rd_Nbits	Reads N consecutive bits value in EBOOL Memory.	SystemLib.h
s_rd_sysbit	Reads a System bit (%Si).	SystemLib.h
s_rd_sysword	Reads a System Word (%SWi).	SystemLib.h
s_wr_16bits	Writes N consecutive bits value in EBOOL Memory.	SystemLib.h
s_wr_1bit	Writes one bit value in EBOOL Memory.	SystemLib.h
s_wr_bits_attrib	Changes attribute bits of "N" consecutive EBOOLs.	SystemLib.h
s_wr_internalwords	Writes N consecutive %MWi words.	SystemLib.h
s_wr_Nbits	Reads N consecutive bits in EBOOL Memory.	SystemLib.h
s_wr_sysbit	Writes a System Word (%SWi).	SystemLib.h
s_wr_sysword	Writes a System bit (%Si).	SystemLib.h
s_cnt_100ms	Reads the value of an internal 100 ms counter.	SystemLib.h
s_cnt_10ms	Reads the value of an internal 10 ms counter.	SystemLib.h

Function	Discription	File Interface
s_cnt_1ms	Reads the value of an internal 1 ms counter.	SystemLib.h
s_date_and_time	Reads PLC current Date and Time.	SystemLib.h
s_syscnt_10ms	Reads the value of the system 10 ms counter.	SystemLib.h
s_current_task	Returns the current task number.	SystemLib.h

Diagnostics

Function	Description	File Interface
s_set_ffb_error	Registers an FFB detected error without an additional parameter.	SystemLib.h
s_set_ffb_error_addi	Registers an FFB detected error together with an additional parameter.	SystemLib.h
s_diag_RegisterExtError	Registers the detected error of the diagnostic system extension..	SystemLib.h
s_diag_DeregisterError	When a registered detected error disappears, the function unregisters the detected error.	SystemLib.h

Signature

Function	Description	File Interface
s_of_passw_check	Accesses PCMCIA card signature; if not OK, PLC is not operational.	SystemLib.h
s_of_passw_test	Accesses PCMCIA card signature; if not OK , returns Diagnostic Code.	SystemLib.h

RUNTIME

Function	Description	File Interface
s_demask_it	Unmasks a critical section from interrupts.	SystemLib.h
s_GetUSecs	Gets the current value of a microsecond counter.	SystemLib.h
s_mask_it	Masks a critical section from interrupts.	SystemLib.h
s_proc_indic	Returns PLC State including memory configuration.	SystemLib.h
s_proc_type	Gets PLC type.	SystemLib.h

Float

Function	Description	File Interface
_clear87	Gets and clears the floating-point status word.	SystemLib.h
_control87	Gets and sets the floating-point control word.	SystemLib.h
_status87	Gets the floating point status word.	SystemLib.h
_chgsign	Changes sign of a float (32-bit only).	SystemLib.h
acos	Calculates the arc cosine.	SystemLib.h
asin	Calculates the arc sine.	SystemLib.h
atan	Calculates the arc tangent.	SystemLib.h
atof	Converts string to double (single if 16-bit).	SystemLib.h
cos	Calculates the cosine.	SystemLib.h
exp	Calculates the exponential.	SystemLib.h
fabs	Calculates the absolute value.	SystemLib.h
fmod	Calculates the floating-point remainder.	SystemLib.h
ftoa	Converts a floating-point value to a string.	SystemLib.h
log	Calculates logarithm.	SystemLib.h
log10	Calculates base-10 logarithm.	SystemLib.h
pow	Calculates x raised to the power of y.	SystemLib.h
sin	Calculates the sine.	SystemLib.h
sqrt	Calculates the square root.	SystemLib.h
tan	Calculates the tangent.	SystemLib.h

Microsoft intrinsic

Function	Discription	File Interface
_lrotl	Rotates an unsigned long value left.	
_lrotr	Rotates an unsigned long value right.	
_rotl	Rotates an unsigned value left.	
_rotr	Rotates an unsigned value right.	
_strset	Sets the characters of a string.	
abs	Returns the absolute value of an integer.	
labs	Returns the absolute value of a long-integer.	
memcmp	Compares n bytes in 2 buffers.	
memcpy	Copies n bytes from a source buffer to a destination buffer.	
memset	Sets n bytes in a buffer.	

Function	Discription	File Interface
strcat	Appends string2 to string1.	
strcmp	Compares n bytes in 2 strings.	
strcpy	Copies string1 to string2.	
strlen	Returns the length in bytes of a string.	

Operators

Function
long +, -, *, /
long %, <<, >>
long <, >, <=, >=, ==, !=
unsigned long +, -, *, /
unsigned long %, <<, >>
unsigned long <, >, <=, >=, ==, !=
float +, -, /, *
float <, >, <=, >=, ==, !=
double +, -, /, * (32-bit only)
double <, >, <=, >=, ==, != (32-bit only)
- (float change sign operator)
- (double change sign operator) (32-bit only)
cast float to (long)
cast float to (unsigned long)
cast float to (short)
cast float to (unsigned short)
cast to (float)
cast to (double) (32-bit only)
cast double to (float) (32-bit only)
cast float to (double) (32-bit only)

Section 6.2
Description

Summary

The following section describes the system services of the EFB Toolkit. It describes the input and output parameters and presents program examples.

What Is in This Section?

This section contains the following topics:

Topic	Page
s_AliGetProgStat	88
s_log_to_phy	90
s_obj_to_log	92
s_obj_nbr	94
s_rd_16bits	95
s_rd_1bit	97
s_rd_bit_attrib	99
s_rd_internalwords	101
s_rd_Nbits	103
s_rd_sysbit	105
s_rd_sysword	106
s_wr_16bits	108
s_wr_1bit	110
s_wr_bits_attrib	112
s_wr_internalwords	114
s_wr_Nbits	116
s_wr_sysbit	118
s_wr_sysword	120
s_cnt_100ms	122
s_cnt_10ms	123
s_cnt_1ms	124
s_date_and_time	125
s_syscnt_10ms	128
s_current_task	129

Topic	Page
s_set_ffb_error	131
s_set_ffb_error_addi	133
s_diag_RegisterExtError	135
s_diag_DeregisterError	137
s_of_passw_check	139
s_of_passw_test	141
s_demask_it	143
s_GetUSecs	144
s_mask_it	145
s_proc_indic	146
s_proc_type	148

s_AliGetProgStat

Description

The `s_AliGetProgStat` system service writes program-specific state information into the parameter structure.

```
s_AliGetProgStat(ProcessState)
```

Availability

The following table shows the availability of the system service on different PLCs:

Target PLC	Available
PLC Sim	yes
M580	yes
Momentum Unity	yes
M340	yes
Premium	yes
Quantum	yes
MC80	yes

Data Types

The following program code shows the data type or structure.

```
typedef struct {     IECEBool   coldInit;  // TRUE during first cycle
                           //  after initialization
                           //  of application
                           //  (download or init).
    IECEBool   warmInit;   // TRUE during first cycle
                           //  in run cycle or after
                           //  power on.
                           //
                           // Note:
                           // During a cold start cycle
                           //  both are TRUE
                           //
    IECEBool  progError;   // TRUE when unconfirmed
                           //  errors present in system
                           //  error buffer
    IECEBool    clk1;      // 0.3125 Hz system clock signal
    IECEBool    clk2;      // 0.625 Hz system clock signal
```

```
     IECEBool    clk3;      // 1.25 Hz system clock signal
     IECEBool    clk4;      // 2.5 Hz system clock signal
     IECEBool    clk5;      // 5 Hz system clock signal
     IECTime    startTime;  // time duration since
                            //  initial Program Start
                            //  independent from real
                            //  time clock counts in
                            //  milliseconds
} PROG_STATE;
```

The behavior of the 'StartTime' timer has been changed in Control Expert, as compared to Concept. In Concept, the timer does not count when the PLC is in STOP mode. In Control Expert, the timer continues to count.

There is a probability of ‘StartTime’ timer retaining values from a previous `warmInit` (run cycle after power on) or a cold start cycle (after start of an application). The timer may continue to store values even after a previous application has stopped. This can lead to inaccurate timing calculations for a new application.

⚠ CAUTION
UNEXPECTED APPLICATION BEHAVIOR Reset the timer to its initial values before running the application. **Failure to follow these instructions can result in injury or equipment damage.**

Input Parameter

The following table describes the input parameters.

Parameter	Description
`ProcessState`	Address of the structure to get the answer.

Return Values

None

Example

```
// Example: s_AliGetProgStat
// Start dimension
PROG_STATE ProcessState
// End dimension
s_AliGetProgStat(&ProcessState);
```

s_log_to_phy

Description

The `s_log_to_phy` system service converts a logical address into a physical address.

```
ptr = s_log_to_phy(logp)
```

Availability

The following table shows the availability of the system service on different PLCs:

Target PLC	Available
PLC Sim	yes
M580	yes
Momentum Unity	yes
M340	yes
Premium	yes
Quantum	yes
MC80	yes

Data Types

The following program code shows the data type or structure.

```
// Parameter given using Relocation Table Entry (RTE)
     typedef unsigned short int IEC_PARAM_RTE;
// The offset size in target dependent
     typedef unsigned int IEC_PARAM_OFFSET;
typedef struct tag_IEC_RTE_OFFSET {
     IEC_PARAM_RTE     block;
     IEC_PARAM_OFFSET offset;
} IEC_RTE_OFFSET;
```

Input Parameters

The following table describes the input parameters.

Parameter	Description
`logp`	Block number and offset to convert.

Return Values

The following table describes the output parameters.

Parameter	Description
`ptr`	Physical pointer to a block.

Example

```
// Example: s_log_to_phy
// Start dimension
IEC_RTE_OFFSET logp
void *ptr;
// End dimension
ptr = s_log_to_phy(logp);
```

s_obj_to_log

Description

The `s_obj_to_log` system service gets the logical address of an application object (%S,%SW,%M,%MW,...)

```
value = s_obj_to_log (object_type, PtrIecRte)
```

Availability

The following table shows the availability of the system service on different PLCs:

Target PLC	Available
PLC Sim	yes
M580	yes
Momentum Unity	yes
M340	yes
Premium	yes
Quantum	yes
MC80	yes

Data Types

The following program code shows the data type or structure.

```
// Parameter given using Relocation Table Entry (RTE)
     typedef unsigned short int IEC_PARAM_RTE;
// The offset size in target dependent
     typedef unsigned int IEC_PARAM_OFFSET;
typedef struct tag_IEC_RTE_OFFSET {
     IEC_PARAM_RTE     block;
     IEC_PARAM_OFFSET offset;
} IEC_RTE_OFFSET;
```

Input Parameters

The following table describes the input parameters:

Parameter	Description
object_type	Type of application object ● QUANTUM and PREMIUM SYSTEM_WORD_ID (0) : %SW INTERNAL_BIT_ID (1) : %M INTERNAL_WORD_ID (2) : %MW COMMON_WORD_ID (5) : %NW ● QUANTUM only INPUT_FLATBIT_ID (6) : %Iflat INPUT_FLATWORD_ID (7) : %IWflat PAGE0_ID (8) : page 0 SYSTABLE_ID (9) : system table
PtrIecRte	Pointer on logical address (OUTPUT PARAMETER)

Return Values

The following table describes the output values:

Values	Description
value	0 : STATUS_OK, read has been done -1 : ERROR, invalid parameter

Example

```
// Example: s_obj_to_log
// Start dimension
IEC_RTE_OFFSET IecRte
unsigned short object_type;
unsigned short value;
// End dimension
value = s_obj_to_log(object_type, &IecRte);
```

s_obj_nbr

Description

The s_obj_nbr system service gets the number of located objects of type (%SW,%M,%MW,%NW,...)

```
value = s_obj_nbr (object_type)
```

Availability

The following table shows the availability of the system service on different PLCs:

Target PLC	Available
M580	yes
Momentum Unity	yes
MC80	yes

Input Parameters

The following table describes the input parameters:

Parameter	Description
`object_type`	SYSTEM_WORD_ID (0) : %SW INTERNAL_BIT_ID (1) : %M INTERNAL_WORD_ID (2) : %MW COMMON_WORD_ID (5) : %NW INPUT_FLATBIT_ID (6) : %Iflat INPUT_FLATWORD_ID (7) : %IWflat

Return Values

The following table describes the output parameters:

Values	Description
`value`	Value of the word. 0 : In case of no object of this type

Example

```
// Example: s_obj_nbr
unsigned short object_type;
unsigned short value;
// End dimension
value = s_obj_nbr(object_type);
```

s_rd_16bits

Description

The `s_rd_16bits` system service concatenates the value bit of 16 IECBool type objects in a 16-bit value. The value bit of an IECBool is bit 0.

```
value = s_rd_16bits(address)
```

Availability

The following table shows the availability of the system service on different PLCs:

Target PLC	Available
PLC Sim	yes
M580	yes
Momentum Unity	yes
M340	yes
Premium	yes
Quantum	yes
MC80	yes

Data Types

The following program code shows the data type or structure.

```
// Parameter given using Relocation Table Entry (RTE)
     typedef unsigned short int IEC_PARAM_RTE;
// The offset size in target dependent
     typedef unsigned int IEC_PARAM_OFFSET;
typedef struct tag_IEC_RTE_OFFSET {
     IEC_PARAM_RTE     block;
     IEC_PARAM_OFFSET offset;
} IEC_RTE_OFFSET;
```

Input Parameters

The following table describes the input parameters:

Parameter	Description
`address`	Logical address of the first bit to be read.

Return Values

The following table describes the output values:

Value	Description
`value`	Value to the referenced bit string.

Example

```
// Example: s_rd_16bits
// Start dimension
unsigned short value;
IEC_RTE_OFFSET address;
// End dimension
value = s_rd_16bits(address);
```

s_rd_1bit

Description

The `s_rd_1bit` system service gets the value bit of a single IECBool type object. The value bit of an IECBool is bit 0.

```
value = s_rd_1bit(address)
```

Availability

The following table shows the availability of the system service on different PLCs:

Target PLC	Available
PLC Sim	yes
M580	yes
Momentum Unity	yes
M340	yes
Premium	yes
Quantum	yes
MC80	yes

Data Types

The following program code shows the data type or structure.

```
// Parameter given using Relocation Table Entry (RTE)
     typedef unsigned short int IEC_PARAM_RTE;
// The offset size in target dependent
     typedef unsigned int IEC_PARAM_OFFSET;
typedef struct tag_IEC_RTE_OFFSET {
     IEC_PARAM_RTE     block;
     IEC_PARAM_OFFSET offset;
} IEC_RTE_OFFSET;
```

Input Parameters

The following table describes the input parameters:

Parameter	Description
`address`	Logical address of the first bit to be read.

Return Values

The following table describes the output values:

Value	Description
`value`	Value to the referenced bit.

Example

```
// Example: s_rd_1bit
// Start dimension
unsigned short value;
IEC_RTE_OFFSET address;
// End dimension
value = s_rd_1bit(address);
```

s_rd_bit_attrib

Description

The `s_rd_bit_attrib` system service reads an entire IECBool type object and its attributes. For state RAM access on Quantum PLCs, only bits 0, 1 and 2 are read (value, history and force on/force off attributes, respectively).

```
value = s_rd_bit_attrib(address, resultptr)
```

Availability

The following table shows the availability of the system service on different PLCs:

Target PLC	Available
PLC Sim	yes
M580	yes
Momentum Unity	yes
M340	yes
Premium	yes
Quantum	yes
MC80	yes

Data Types

The following program code shows the data type or structure.

```
// Parameter given using Relocation Table Entry (RTE)
     typedef unsigned short int IEC_PARAM_RTE;
// The offset size in target dependent
     typedef unsigned int IEC_PARAM_OFFSET;
typedef struct tag_IEC_RTE_OFFSET {
     IEC_PARAM_RTE     block;
     IEC_PARAM_OFFSET offset;
} IEC_RTE_OFFSET;
```

Input Parameters

The following table describes the input parameters:

Parameter	Description
address	Address of the bit to read.
resultptr	Pointer that will contain the result (OUTPUT PARAMETER). The function returns the IECBool with IOIM format: bit 0: Value bit 1: History bit 2: Forcing

Return Values

The following table describes the output values:

Value	Description
value	0: No detected error. <0: In case of a detected error.

Example

```
// Example: s_rd_bit_attrib
// Start dimension
IEC_RTE_OFFSET address;
unsigned char result;
int value;
// End dimension
value = s_rd_bit_attrib (address, &result);
```

s_rd_internalwords

Description

The `s_rd_internalwords` system service reads *n* consecutive %MWi bits.

```
value = s_rd_internalwords(number, length, BuffAddr)
```

Availability

The following table shows the availability of the system service on different PLCs:

Target PLC	Available
PLC Sim	yes
M580	yes
Momentum Unity	yes
M340	yes
Premium	yes
Quantum	yes
MC80	yes

Input Parameters

The following table describes the input parameters:

Parameter	Description
`number`	ID number of the first word to be read
`length`	Number of words to be read
`BuffAddr`	Pointer to the buffer that contains the word values

Return Values

The following table describes the output values:

Value	Description
`value`	0: Work has been made. < 0: Wrong parameters number or length.

Example

```
// Example: s_rd_internalwords
// Start dimension
short value;
unsigned short number;
unsigned short length;
unsigned short Buffer;
// End dimension
value = s_rd_internalwords(number, length, &Buffer);
```

s_rd_Nbits

Description

The `s_rd_Nbits` system service concatenates the value bit of *n* IECBool type objects in a 32-bit value. The value bit of an IECBool is its 0 bit.

```
value = s_rd_Nbits(address, length, state)
```

Availability

The following table shows the availability of the system service on different PLCs:

Target PLC	Available
PLC Sim	yes
M580	yes
Momentum Unity	yes
M340	yes
Premium	yes
Quantum	yes
MC80	yes

Data Types

The following program code shows the data type or structure.

```
// Parameter given using Relocation Table Entry (RTE)
     typedef unsigned short int IEC_PARAM_RTE;
// The offset size in target dependent
     typedef unsigned int IEC_PARAM_OFFSET;
typedef struct tag_IEC_RTE_OFFSET {
     IEC_PARAM_RTE     block;
     IEC_PARAM_OFFSET offset;
} IEC_RTE_OFFSET;
```

Input Parameters

The following table describes the input parameters:

Parameter	Description
`address`	Logical address of the first object.
`length`	Number of objects to read has to be <= 32.
`state`	Contains the state of the function. Negative value in case of a detected error or 0 if its OK.

Return Values

The following table describes the output values:

Value	Description
`value`	0: In case of a detected error (invalid block number in logical address). <> 0: The concatenation of the value bits.

Example

```
// Example: s_rd_Nbits
// Start dimension
unsigned long value;
IEC_RTE_OFFSET address;
unsigned short length;
short state;
// End dimension
value = s_rd_Nbits(address, length, &state);
```

s_rd_sysbit

Description

The `s_rd_sysbit` system service reads a system bit %Si identified by its number. The consistency of the bit number is checked.

```
value = s_rd_sysbit(id, state)
```

Availability

The following table shows the availability of the system service on different PLCs:

Target PLC	Available
PLC Sim	yes
M580	yes
Momentum Unity	yes
M340	yes
Premium	yes
Quantum	yes
MC80	yes

Input Parameters

The following table describes the input parameters:

Parameter	Description
`id`	Number of the bit to be read.
`state`	Pointer to the system status word.

Return Values

When the status is `STATUS_OK`, the bit value is entered in the least significant bit.

Bit numbers 1 .. 15 are set to zero.

A bit number outside these limits produces the status value `OUT_OF_BOUNDS`.

Example

```
// Example: s_rd_sysbit
// Start dimension
unsigned short value;
unsigned short far state;
// End dimension
value = s_rd_sysbit(4, &state);
```

s_rd_sysword

Description

The `s_rd_sysword` system service reads a system word %SWi identified by its number. The consistency of the bit number is checked.

```
value = s_rd_sysword(sysword_range, state)
```

Availability

The following table shows the availability of the system service on different PLCs:

Target PLC	Available
PLC Sim	yes
M580	yes
Momentum Unity	yes
M340	yes
Premium	yes
Quantum	yes
MC80	yes

Input Parameters

The following table describes the input parameters:

Parameter	Description
`sysword_range`	Logical address of the word to be read.
`state`	Pointer to the system status word.

Return Values

The following table describes the output parameters:

Parameter	Description
`value`	Value of the word. 0: In case of a detected error or if there is no application loaded.

Example

```
// Example: s_rd_sysword
// Start dimension
unsigned short value;
unsigned short state;
// End dimension
value = s_rd_sysword(4, &state);
```

s_wr_16bits

Description

The `s_wr_16bits` system service writes the value of 16 consecutive bits of forcible memory. Because the system does not perform any checks, the EF using this utility has to ensure the validity of the 16-bit address that has become a parameter.

```
s_wr_16bits(IEC_RTE_OFFSET address, value1)
```

Availability

The following table shows the availability of the system service on different PLCs:

Target PLC	Available
PLC Sim	yes
M580	yes
Momentum Unity	yes
M340	yes
Premium	yes
Quantum	yes
MC80	yes

Data Types

The following program code shows the data type or structure.

```
// Parameter given using Relocation Table Entry (RTE)
     typedef unsigned short int IEC_PARAM_RTE;
// The offset size in target dependent
     typedef unsigned int IEC_PARAM_OFFSET;
typedef struct tag_IEC_RTE_OFFSET {
     IEC_PARAM_RTE     block;
     IEC_PARAM_OFFSET offset;
} IEC_RTE_OFFSET;
```

Input Parameters

The following table describes the input parameters:

Parameter	Description
address	Logical address of the first bit to be written.
value1	The value of the 16 bits, from bit 0 to 15.

Return Values

none

Example

```
// Example: s_wr_16bits
// Start dimension
unsigned short value1;
IEC_RTE_OFFSET AdresseBit;
// End dimension
s_wr_16bits(AdresseBit, value1);
```

s_wr_1bit

Description

The `s_wr_1bit` system service writes the value of a forcible bit. Because the system does not perform any checks, the EF using this utility has to ensure the validity of the address of the bit that has become a parameter.

```
s_wr_1bit(IEC_RTE_OFFSET address, value1)
```

Availability

The following table shows the availability of the system service on different PLCs:

Target PLC	Available
PLC Sim	yes
M580	yes
Momentum Unity	yes
M340	yes
Premium	yes
Quantum	yes
MC80	yes

Data Types

The following program code shows the data type or structure.

```
// Parameter given using Relocation Table Entry (RTE)
     typedef unsigned short int IEC_PARAM_RTE;
// The offset size in target dependent
     typedef unsigned int IEC_PARAM_OFFSET;
typedef struct tag_IEC_RTE_OFFSET {
     IEC_PARAM_RTE     block;
     IEC_PARAM_OFFSET offset;
} IEC_RTE_OFFSET;
```

Input Parameters

The following table describes the input parameters:

Parameter	Description
`address`	Address of the forcible bit to be written.
`value1`	Values to be written to a bit, either 0 or 1.

Return Values

none

Example

```
// Example: s_wr_1bit
// Start dimension
unsigned short value1;
IEC_RTE_OFFSET far AdresseBit;
// End dimension
s_wr_1bit(AdresseBit, value1);
```

s_wr_bits_attrib

Description

The `s_wr_bits_attrib` system service writes *n* consecutive IECBool type objects, where *n* <= 32. Several fields of the objects can be accessed depending on the type parameter. A type may be:

- DATA_WRITE (0)
  Writes value bits (according to the forcing attribute, history is also updated).
- UNFORCE_ALL (8)
  Resets forcing bits (bit 3).
- DEFAULT_WRITE (7)
  Writes default bit (bit 7).
- RVALUE_WRITE (4)
  Writes fallback bit (bit 4).
- RVALID_WRITE (6)
  Writes fallback validation bit (bit 6).

```
value = s_wr_bits_attrib (address, length, value1, type)
```

Availability

The following table shows the availability of the system service on different PLCs:

Target PLC	Available
PLC Sim	yes
M580	yes
Momentum Unity	yes
M340	yes
Premium	yes
Quantum	yes
MC80	yes

Data Types

The following program code shows the data type or structure.

```
// Parameter given using Relocation Table Entry (RTE)
     typedef unsigned short int IEC_PARAM_RTE;
// The offset size in target dependent
     typedef unsigned int IEC_PARAM_OFFSET;
typedef struct tag_IEC_RTE_OFFSET {
     IEC_PARAM_RTE     block;
     IEC_PARAM_OFFSET offset;
} IEC_RTE_OFFSET;
```

Input Parameters

The following table describes the input parameters:

Parameter	Description
address	Logical address of the first bit to be written.
length	Number (*n*) of bits to be written (*n* <=32).
value1	The *n* bits values to be write, from 0 to *n* - 1 bit. Not used if type is equal to UNFORCE_ALL.
type	Attribute to write.

Return Values

The following table describes the output values:

Value	Description
value	0 if OK, negative value in case of a detected error.

Example

```
// Example: s_wr_bits_attrib
// Start dimension
short value;
unsigned short length;
unsigned long value1;
unsigned short type;
IEC_RTE_OFFSET address;
// End dimension
value = s_wr_bits_attrib(address, length, value1, type);
```

s_wr_internalwords

Description

The `s_wr_internalwords` system service writes *n* consecutive %MWi words.

```
value = s_wr_internalwords(number, length, BuffAddr)
```

Availability

The following table shows the availability of the system service on different PLCs:

Target PLC	Available
PLC Sim	yes
M580	yes
Momentum Unity	yes
M340	yes
Premium	yes
Quantum	yes
MC80	yes

Input Parameters

The following table describes the input parameters:

Parameter	Description
`number`	Number of the first word to be written.
`length`	Number of words to be written.
`BuffAddr`	Far pointer to the buffer that contains the words values.

Return Values

The following table describes the output values:

Value	Description
`value`	0: Work has been made. < 0: Wrong number or length parameter.

Example

```
// Example: s_wr_internalwords
// Start dimension
short value;
unsigned short number;
unsigned short length;
unsigned short Buffer;
// End dimension
value = s_wr_internalwords(number, length, &Buffer);
```

s_wr_Nbits

Description

The `s_wr_Nbits` system service writes the value bit of *n* consecutive IECBool objects. The value bit of an IECBool is its bit 0. This function updates the history bit (bit 1).

```
s_wr_Nbits(address, length, value1)
```

Availability

The following table shows the availability of the system service on different PLCs:

Target PLC	Available
PLC Sim	yes
M580	yes
Momentum Unity	yes
M340	yes
Premium	yes
Quantum	yes
MC80	yes

Data Types

The following program code shows the data type or structure.

```
// Parameter given using Relocation Table Entry (RTE)
     typedef unsigned short int IEC_PARAM_RTE;
// The offset size in target dependent
     typedef unsigned int IEC_PARAM_OFFSET;
typedef struct tag_IEC_RTE_OFFSET {
     IEC_PARAM_RTE     block;
     IEC_PARAM_OFFSET offset;
} IEC_RTE_OFFSET;
```

Input Parameters

The following table describes the input parameters:

Parameter	Description
`address`	Logical address of the first object to be written.
`length`	Number of objects to be written.
`value1`	Value of the *n* bits, from bit 0 to *n* - 1.

Return Values

none

Example

```
// Example: s_wr_Nbits
// Start dimension
IEC_RTE_OFFSET address;
unsigned short length;
unsigned long value1;
// End dimension
s_wr_Nbits(address, length, value1);
```

s_wr_sysbit

Description

The `s_wr_sysbit` system service writes a system bit identified by its number.

```
value = s_wr_sysbit(id, value1)
```

Availability

The following table shows the availability of the system service on different PLCs:

Target PLC	Available
PLC Sim	yes
M580	yes
Momentum Unity	yes
M340	yes
Premium	yes
Quantum	yes
MC80	yes

Input Parameters

The following table describes the input parameters.

Parameter	Description
`id`	Number of the bit to be written.
`value1`	Value of the bit, only bit 0 of the word is significant.

Return Values

The following table describes the output values.

Value	Description
`value`	• `STATUS_OK`: 0 Work has been made. • < 0 Wrong number (outside valid range)

Example

```
// Example: s_wr_sysbit
// Start dimension
short value;
unsigned short id;
unsigned short value1;
// End dimension
value = s_wr_sysbit (id, value1);
```

s_wr_sysword

Description

The `s_wr_sysword` system service writes a system word %SWi identified by its number.

```
value = s_wr_sysword(number, value1)
```

Availability

The following table shows the availability of the system service on different PLCs:

Target PLC	Available
PLC Sim	yes
M580	yes
Momentum Unity	yes
M340	yes
Premium	yes
Quantum	yes
MC80	yes

Input Parameters

The following table describes the input parameters:

Parameter	Description
`number`	Number of the word to be written
`value1`	Value of the word.

Return Values

The following table describes the output parameters:

Parameter	Description
`value`	STATUS_OK: Work has been made. OUT_OF_BOUNDS: Wrong number.

Example

```
// Example: s_wr_sysword
// Start dimension
short value;
unsigned short number;
unsigned short value1;
// End dimension
value = s_wr_sysword(number, value1);
```

s_cnt_100ms

Description

The `s_cnt_100ms` system service is a system-time-base 100 ms counter. This internal counter may be modified at any moment by a WarmStandBy IOB and EF to synchronize it with another WarmStandBy PLC.

The counter is set to zero only on a cold start. The counter does not increment during power breaks.

```
value = s_cnt_100ms()
```

Availability

The following table shows the availability of the system service on different PLCs:

Target PLC	Available
PLC Sim	yes
M580	yes
Momentum Unity	yes
M340	yes
Premium	yes
Quantum	yes
MC80	yes

Input Parameters

none

Return Values

The output value of the counter has the following characteristics:

Value	Description
`value`	Unsigned value, from 0 to 4 294 967 295

Example

```
// Example: s_cnt_100ms
// Start dimension
unsigned long value;
// End dimension
value = s_cnt_100ms();
```

s_cnt_10ms

Description

The `s_cnt_10ms` system service is a system-time-base 10 ms counter. This internal counter may be modified at any moment by a WarmStandBy IOB and EF to synchronize it with another WarmStandBy PLC.

The counter is set to zero only on a cold start. It does not be increment during power breaks.

```
value = s_cnt_10ms()
```

Availability

The following table shows the availability of the system service on different PLCs:

Target PLC	Available
PLC Sim	yes
M580	yes
Momentum Unity	yes
M340	yes
Premium	yes
Quantum	yes
MC80	yes

Input Parameters

none

Return Values

The output value of the counter has the following characteristics:

Value	Description
`value`	Unsigned value, from 0 to 4 294 967 295

Example

```
// Example: s_cnt_10ms
// Start dimension
unsigned long value;
// End dimension
value = s_cnt_10ms();
```

s_cnt_1ms

Description

The `s_cnt_1ms` system service is a system-time-base 1 ms counter. This internal counter may be modified at any moment by a WarmStandBy IOB and EF to synchronize it with another WarmStandBy PLC.

The counter is set to zero only on a cold start. It does not increment during power breaks.

```
value = s_cnt_1ms()
```

Availability

The following table shows the availability of the system service on different PLCs:

Target PLC	Available
PLC Sim	yes
M580	yes
Momentum Unity	yes
M340	yes
Premium	yes
Quantum	yes
MC80	yes

Input Parameters

none

Return Values

The output value of the counter has the following characteristics:

Value	Description
`value`	Unsigned value, from 0 to 65 535

Example

```
// Example: s_cnt_1ms
// Start dimension
unsigned short value;
// End dimension
value = s_cnt_1ms();
```

s_date_and_time

Description

The `s_date_and_time` system service enables access to the real-time clock. The information (the date and current time for the PLC) is coded in BCD in nine bytes (or 18 digits), according to the format below:

Type	Range
Day of the week	01 .. 07 01: Monday 02: Tuesday 03: Wednesday 04: Thursday 05: Friday 06: Saturday 07: Sunday
Year	0000 .. 9999
Month	01 .. 12
Day	01 .. 31
Hour	00 .. 23
Minute	00 .. 59
Second	00 .. 59
Reserved	00

Leap years are managed.

```
value = s_date_and_time(date_ptr)
```

Availability

The following table shows the availability of the system service on different PLCs:

Target PLC	Available
PLC Sim	yes
M580	yes
Momentum Unity	yes
M340	yes
Premium	yes
Quantum	yes
MC80	yes

Data Types

The following code shows the data type or structure:

```
typedef struct tagIECDate
 {
     union
     {
       struct
       {
         IECByte Day;
         IECByte Month;
         IECUInt Year;
       }DMY;
       IECUDInt UDIntValue;
     };
 } IECDate;

 typedef struct tagIECTimeOfDay
 {
     union
     {
       struct
       {
         IECByte Dummy;
         IECByte Second;
         IECByte Minute;
         IECByte Hour;
       }HMS;
       IECUDInt UDIntValue;
     };
 } IECTimeOfDay;

 typedef struct tagIECDateAndTime
 {
     IECTimeOfDay Time;
     IECDate      Date;
 } IECDateAndTime;
 typedef struct
 {
     IECDAteAndTime DateTime;
     unsigned char DayWeek;
 } DATE_TIME;
```

Input Parameters

The following table describes the input parameters:

Parameter	Description
date_ptr	Pointer to the copying zone of the date and current time (9 bytes).

Return Values

The following table describes the output values:

Value	Description
value	• STATUS_OK the date and time fields are correctly indicated • CLOCK_UNAVAILABLE the fields cannot update correctly • CLOCK_NOT_SUPPORTED no a real-time clock present

Example

```
// Example: s_date_and_time
// Start dimension
unsigned short value;
DATE_TIME date;
// End dimension
value = s_date_and_time(&date);
```

s_syscnt_10ms

Description

The `s_syscnt_10ms` system service is a 10 ms system-time-base counter. It is an internal counter that you cannot modify. It is set to zero only on a cold start. It does not increment during power breaks.

```
value = s_syscnt_10ms()
```

Availability

The following table shows the availability of the system service on different PLCs:

Target PLC	Available
PLC Sim	yes
M580	yes
Momentum Unity	yes
M340	yes
Premium	yes
Quantum	yes
MC80	yes

Input Parameters

none

Return Values

The following table describes the output values:

Value	Description
`value`	Unsigned value, from 0 to 4 294 967 295

Example

```
// Example: s_syscnt_10ms
// Start dimension
unsigned long value;
// End dimension
value = s_syscnt_10ms();
```

s_current_task

Description

The `s_current_task` system service sends back the value of the current task.

```
value = s_current_task()
```

Availability

The following table shows the availability of the system service on different PLCs:

Target PLC	Available
PLC Sim	yes
M580	yes
Momentum Unity	yes
M340	yes
Premium	yes
Quantum	yes
MC80	yes

Input Parameters

none

Return Values

The following table describes the output values:

Value	Description
`value`	task number ● user tasks: event tasks pu_TSK_PRIO0 : 0 pu_TSK_PRIO1 : 1 pu_TSK_PRIO2 : 2 ● user tasks: standard tasks: Run/Stop pu_TSK_FAST : 3 pu_TSK_MAST : 4 pu_TSK_AUX0 : 5 pu_TSK_AUX1 : 6 pu_TSK_AUX2 : 7 pu_TSK_AUX3 : 8

Example

```
// Example: s_current_task
// Start dimension
unsigned short value;
// End dimension
value = s_current_task();
```

s_set_ffb_error

Description

The `s_s_set_ffb_error` system service registers an FFB detected error without an additional parameter. This function simultaneously registers and deregisters an FFB detected error—i.e., it registers the detected error in reg-dereg mode. This service is similar to `s_set_ffb_error_addi`, except that no additional detected error parameter is passed to the function.

NOTE: If the ErrNum parameter contains one of the following values:
STRINGERROR, CARRY, OVERFLOW, INDEXOVF
then it writes the corresponding %S system bit.

```
value = s_set_ffb_error(sErrNum)
```

Availability

The following table shows the availability of the system service on different PLCs:

Target PLC	Available
PLC Sim	yes
M580	yes
Momentum Unity	yes
M340	yes
Premium	yes
Quantum	yes
MC80	yes

Input Parameters

The following table describes the input parameters:

Parameter	Description
`sErrNum`	Detected error number.

Return Values

The following table describes the output values:

Value	Description
`value`	unequal 0: operation successful, else detected error err_DIAG_NO_BUFFER: no diagnostic buffer err_DIAG_BUFFER_FULL: multiple detected errors or no more free memory err_OBJUSED: diagnostic buffer being use by another task

Example

```
// Example: s_set_ffb_error
// Start dimension
unsigned short value;
short sErrNum;
// End dimension
value = s_set_ffb_error(sErrNum);
```

s_set_ffb_error_addi

Description

The `s_set_ffb_error_addi` system service registers an FFB detected error together with an additional parameter. This function simultaneously registers and deregisters an FFB detected error—i.e., it registers the detected error in reg-dereg mode. An additional descriptive parameter may optionally be passed.

NOTE: If the ErrNum parameter contains one of the following values:
STRINGERROR, CARRY, OVERFLOW, INDEXOVF
then it writes the corresponding %S system bit.

```
value = s_set_ffb_error_addi(sErrNum, sParam)
```

Availability

The following table shows the availability of the system service on different PLCs:

Target PLC	Available
PLC Sim	yes
M580	yes
Momentum Unity	yes
M340	yes
Premium	yes
Quantum	yes
MC80	yes

Input Parameters

The following table describes the input parameters:

Parameter	Description
`sErrNum`	Detected error number.
`sParam`	Additional detected error parameter.

Return Values

The following table describes the output values:

Value	Description
`value`	unequal 0: operation was successful, else detected error err_DIAG_NO_BUFFER: no diagnostic buffer err_DIAG_BUFFER_FULL: multiple detected errors or no more free memory err_OBJUSED: diagnostic buffer being use by other task

Example

```
// Example: s_set_ffb_error
// Start dimension
unsigned short value;
short sErrNum;
unsigned short sParam;
// End dimension
value = s_set_ffb_error(sErrNum, sParam);
```

s_diag_RegisterExtError

Description

The `s_diag_RegisterExtError` system service registers errors detected by the diagnostic system extension.

The function allows the caller to preset the whole detected error information, which normally is provided by the diagnostic subsystem and lets the diagnostic subsystem store the information transparently, when registering the detected error. This function returns the detected error registration id (`ErrorId`).

Canceling of the detected error is done with the function `diag_DeregisterError`.

```
value = s_diag_RegisterExtError(OpControl, ECode, CmntAdr, EtxtAdr,
DataLen, DataAdr, pErrorId)
```

Availability

The following table shows the availability of the system service on different PLCs:

Target PLC	Available
PLC Sim	yes
M580	yes
Momentum Unity	yes
M340	yes
Premium	yes
Quantum	yes
MC80	yes

Input Parameters

The following table describes the input parameters:

Parameter	Description
`OpControl`	Type: unsigned char 1: with (ACK required) 0: without operator control (auto ACK)
ECode	Type: unsigned long detected error code number
CmntAdr	Type: IEC_RTE_OFFSET comment address
EtxtAdr	Type: IEC_RTE_OFFSET detected error text address
DataLen	Type: unsigned short length of the additional information
DataAdr	Type: IEC_RTE_OFFSET additional information address of the detected error
pErrorId	Type: unsigned short FAR * detected error registration id -returned

Return Values

The following table describes the output values:

Value	Description
`value`	unequal 0: operation successful, else detected error err_DIAG_NO_BUFFER: no diag buffer err_DIAG_BUFFER_FULL: multiple detected errors or no more free memory err_OBJUSED: diagnostic buffer in use by other task err_OUTOFBOUNDS: the length of the additional information raises the limitation

Example

```
// Example: s_diag_RegisterExtError
// Start dimension
unsigned short value, DataLen;
unsigned char  OpControl;
unsigned long  ECode;
unsigned long  ECode;
IEC_RTE_OFFSET CmntAdr, EtxtAdr, DataAdr;
unsigned short FAR * pErrorId;
// End dimension
value = s_diag_RegisterExtError(OpControl, ECode, CmntAdr,
EtxtAdr, DataLen, DataAdr, pErrorId);
```

s_diag_DeregisterError

Description

The `s_diag_DeregisterError` system service unregisters errors detected by the diagnostic system extension.

When a registered detected error disappears, the producer (FB, SFC or system) can call this function to unregister the detected error.

```
value = s_diag_DeregisterError(usErrorId)
```

Availability

The following table shows the availability of the system service on different PLCs:

Target PLC	Available
PLC Sim	yes
M580	yes
Momentum Unity	yes
M340	yes
Premium	yes
Quantum	yes
MC80	yes

Input Parameters

The following table describes the input parameters:

Parameter	Description
`usErrorId`	Type: unsigned short Detected error identifier which comes from registration.

Return Values

The following table describes the output values:

Value	Description
`value`	unequal 0: operation successful, else detected error err_DIAG_NO_BUFFER: no diag buffer err_DIAG_WRONG_ERROR_ID: wrong identifier err_OBJUSED: diagnostic buffer in use by other task

Example

```
// Example: s_diag_DeregisterError
// Start dimension
unsigned short value, usErrorId;
// End dimension
value = s_diag_DeregisterError(usErrorId);
```

s_of_passw_check

Description

The `s_of_passw_check` system service checks the signature and returns a STATUS_OK if it is correct. If the signature is not correct, the PLC goes to halt state with value 0002 in %SW125.

```
value = s_of_passw_check(sign_ptr)
```

Availability

The following table shows the availability of the system service on different PLCs:

Target PLC	Available
PLC Sim	no
M580	no
Momentum Unity	no
M340	no
Premium legacy	yes
Premium high-end	yes
Quantum legacy	no
Quantum high-end	yes
MC80	yes

Input Parameters

The following table describes the input parameter:

Parameter	Description
`sign_ptr`	Pointer to signature code array

Return Values

The following table describes the output value:

Value	Description
`value`	STATUS_OK: operation successful

Example

```
// Example: s_of_passw_check
// Start dimension
unsigned short value;
unsigned char sign(16);
// End dimension
value = s_of_passw_check(&sign);
```

s_of_passw_test

Description

The `s_of_passw_test` system service tests the password. This routine checks the signature and returns a STATUS_OK if the signature is correct. If the signature is not correct, it returns a diagnostic message.

```
value = s_of_passw_test(sign_ptr)
```

Availability

The following table shows the availability of the system service on different PLCs:

Target PLC	Available
PLC Sim	no
M580	no
Momentum Unity	no
M340	no
Premium legacy	yes
Premium high-end	yes
Quantum legacy	no
Quantum high-end	yes
MC80	yes

Input Parameters

The following table describes the input parameter:

Parameter	Description
`sign_ptr`	Pointer to signature code array

Return Values

The following table describes the output value:

Value	Description
`value`	STATUS_OK: operation successful

Example

```
// Example: s_of_passw_test
// Start dimension
unsigned short value;
unsigned char sign;
// End dimension
value = s_of_passw_test(&sign);
```

s_demask_it

Description

The `s_demask_it` system service enables access to a critical section from interrupts.

```
void s_demask_it(unsigned short value1)
```

Availability

The following table shows the availability of the system service on different PLCs:

Target PLC	Available
PLC Sim	yes
M580	no
Momentum Unity	no
M340	yes
Premium	yes
Quantum	yes
MC80	yes

Input Parameters

The following table describes the input parameter:

Parameter	Description
`value1`	Value of the CPU flag register to restore (gotten from s_mask_it).

Return Values

none

Example

```
// Example: s_demask_it
// Start dimension
unsigned short value1;
// End dimension
s_demask_it(value1);
```

s_GetUSecs

Description

The `s_GetUSecs` system service gets the current value of a microsecond counter. The overflow from the counter (in the range 0xFFFFFFFF to 0) has to be managed by the caller in case of gap time calculations.

```
unsigned int s_GetUSecs(void)
```

Availability

The following table shows the availability of the system service on different PLCs:

Target PLC	Available
PLC Sim	yes
M580	yes
Momentum Unity	yes
M340	yes
Premium legacy	no
Premium high-end	yes
Quantum legacy	yes
Quantum high-end	yes
MC80	yes

Input Parameters

none

Return Values

The following table describes the output value:

Value	Description
`value`	Current value of the microsecond counter

Example

```
// Example: s_GetUSecs
// Start dimension
unsigned int value;
// End dimension
value = s_GetUSecs();
```

s_mask_it

Description

The `s_mask_it` system service disables access to a critical section from interrupts. .

```
unsigned short s_mask_it(void)
```

Availability

The following table shows the availability of the system service on different PLCs:

Target PLC	Available
PLC Sim	yes
M580	no
Momentum Unity	no
M340	yes
Premium	yes
Quantum	yes
MC80	yes

Input Parameters

none

Return Values

The following table describes the output value:

Value	Description
`value`	Content of the CPU flag register

Example

```
// Example: s_mask_it
// Start dimension
unsigned short value;
// End dimension
value = s_mask_it();
```

s_proc_indic

Description

The `s_proc_indic` system service returns the current state of an indicator.

`s_proc_indic(StrInd)`

Availability

The following table shows the availability of the system service on different PLCs:

Target PLC	Available
PLC Sim	yes
M580	yes
Momentum Unity	yes
M340	yes
Premium	yes
Quantum	yes
MC80	yes

Data Types

The following code shows the data type or structure:

```
typedef struct {
    unsigned short bRun            : 1; // 1 if run
    unsigned short bAppliExec      : 1; // 1 if i.c. and
                                        //   configuration
                                        //    application is
                                        //    okay
    unsigned short bCartMemDetect  : 1; // 1 if a memory
                                        //   card is
                                        //   detected
    unsigned short bReservation    : 1; // 1 if reservation
                                        //   is in progress
    unsigned short bBrkptSet       : 1; // 1 if break
                                        //   point is SET
    unsigned short bAppliFailure   : 1; // 1 if application
                                        //   is HALT
    unsigned short bPartialRun     : 1; // 1 if partially
                                        //   is RUN
    unsigned short bCartFlashErase : 1; // 1 if memory
                                        //   card type EPROM
                                        //   delete in
```

```
                                            //   progress
    unsigned short bUserMemProtect : 1; // 1 if application
                                            //   in memory
                                            //   protected write
    unsigned short bPhaseInit      : 1; // Allow RUN with
                                            //   internal
                                            //   restore
    unsigned short bRestoreBck     : 1; // Internal Backup
                                            //   restore
    unsigned short bAppliInit      : 1; // Application
                                            //   initialized,
                                            //    %S %SW
                                            //   available
    unsigned short cUnused         : 4; // Alignment
} SGenInd ;
```

Input Parameters

The following table describes the input parameter:

Parameter	Description
StrInd	Address of the buffer that will be filled.

Return Values

none

Example

```
// Example: s_proc_indic
// Start dimension
unsigned short value;
SGenInd StrInd;
// End dimension
s_proc_indic(&StrInd);
```

s_proc_type

Description

The `s_proc_type` system service returns the processor's identification.

```
s_proc_type(DeviceId)
```

Availability

The following table shows the availability of the system service on different PLCs:

Target PLC	Available
PLC Sim	yes
M580	yes
Momentum Unity	yes
M340	yes
Premium	yes
Quantum	yes
MC80	yes

Data Types

The following code shows the data type or structure:

```
typedef struct {
     unsigned short ProductRange;
     unsigned short PlcIdentification;
     unsigned short PlcModel;
     unsigned short ReservedForFuture;
     unsigned short ComVersion;
     unsigned short NumPatch;
} DEVICEID;
```

Input Parameters

The following table describes the input parameter:

Parameter	Description
`DeviceId`	Address of a buffer that will contain the identification of the device.

Return Values

none

Example

```
// Example: s_proc_type
// Start dimension
DEVICEID DeviceId;
// End dimension
s_proc_type(&DeviceId);
```

Appendices

Appendix A
PL7/Concept EF/EFB Migration

Overview

This chapter provides an overview of PL7 and Concept EF/EFB migration.

What Is in This Chapter?

This chapter contains the following sections:

Section	Topic	Page
A.1	PL7 and Concept EF/EFB Migration	154
A.2	PL7 EF Migration Procedure	155
A.3	Concept EF/EFB Migration Procedure	163
A.4	Other Case Studies	171
A.5	Comparison Between PL7/Concept and Control Expert	184
A.6	An Empty Frame for a Control Expert EF	194

Section A.1
PL7 and Concept EF/EFB Migration

PL7 EF Migration

Introduction

This document answers the question: *How can I transfer my EFs to Control Expert*? It does not describe any special tools that could be used; it simply describes the process generically and gives an example.

The following discussion points out how a practical conversion of Concept EF/EFBs to Control Expert can be done. The goal of this document is to provide some guidelines for EF/EFB migration from Concept to Control Expert. Changes are considered from the Control Expert point of view. For a detailed discussion of procedures, refer to the toolkit documentation.

Since the functionality of the EF/EFB should be the same in Control Expert as in Concept, make the needed changes in the calling interface of the function block whenever possible. Changes to the function code itself should be kept to an absolute minimum.

Preparation

PL7 EF migration is primarily a copy-and-paste process. Before you begin to migrate C-coded functions, verify that a complete programming environment is installed on your system.

You will need the old as well as new programming tools:
- PL7 SDKC
- Unity EFB Toolkit

For example if you want to migrate functions created with the PL7 SDKC tool, you need at least the source files (headers and C-files) from your old function blocks.

Section A.2
PL7 EF Migration Procedure

Summary

The following section describes the PL7 EF migration procedure.

Migrated applications may not operate exactly as the original.

⚠ WARNING
UNEXPECTED EQUIPMENT OPERATION
• Perform a risk analysis of the migrated application prior to commissioning. • Apply and test the preventive and detective controls from the risk analysis before implementation. • Test each implementation of the migrated application for proper operation before placing into service.
Failure to follow these instructions can result in death, serious injury, or equipment damage.

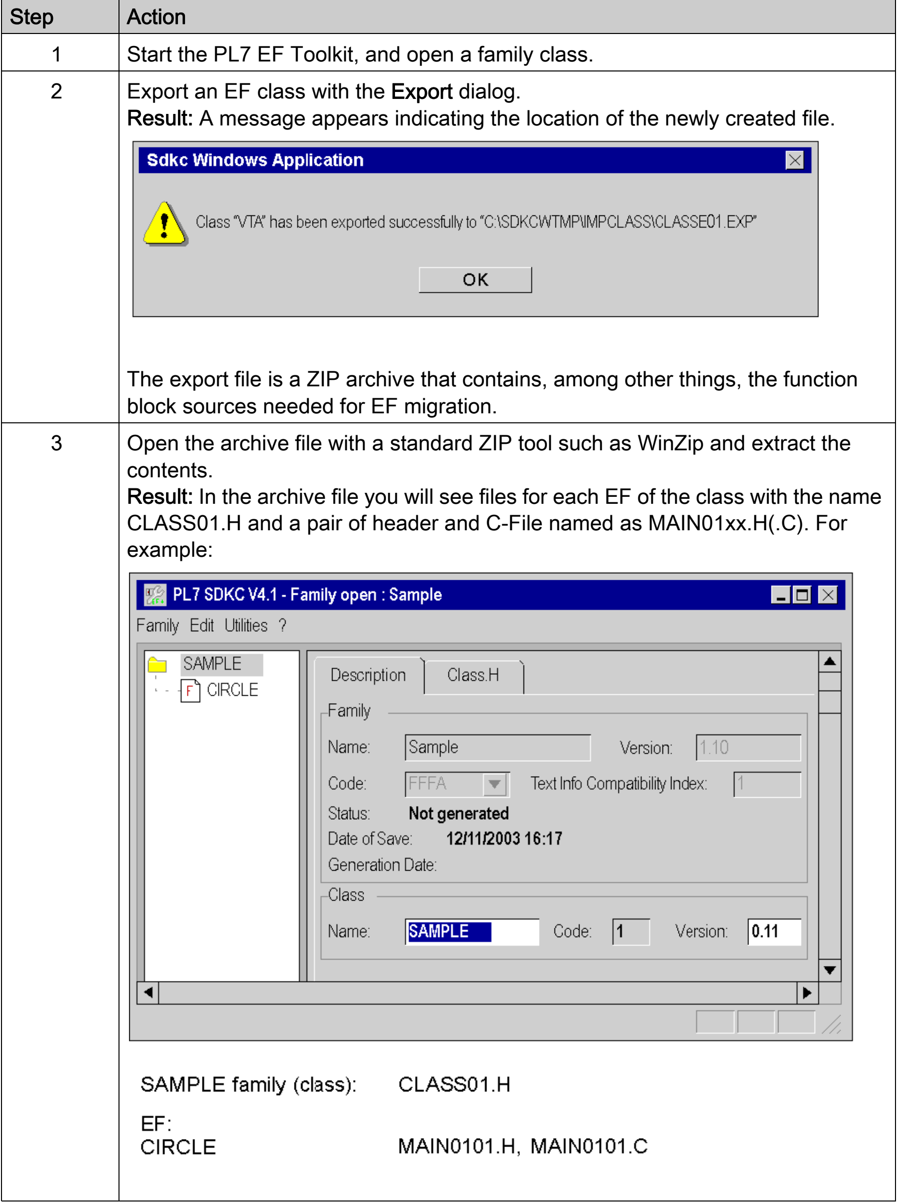
What Is in This Section?

This section contains the following topics:

Topic	Page
Retrieving Source Files from the PL7 Development Environment	156
Migration Procedure	157
PL7 EF Code	160
Control Expert EF Code	161

Retrieving Source Files from the PL7 Development Environment

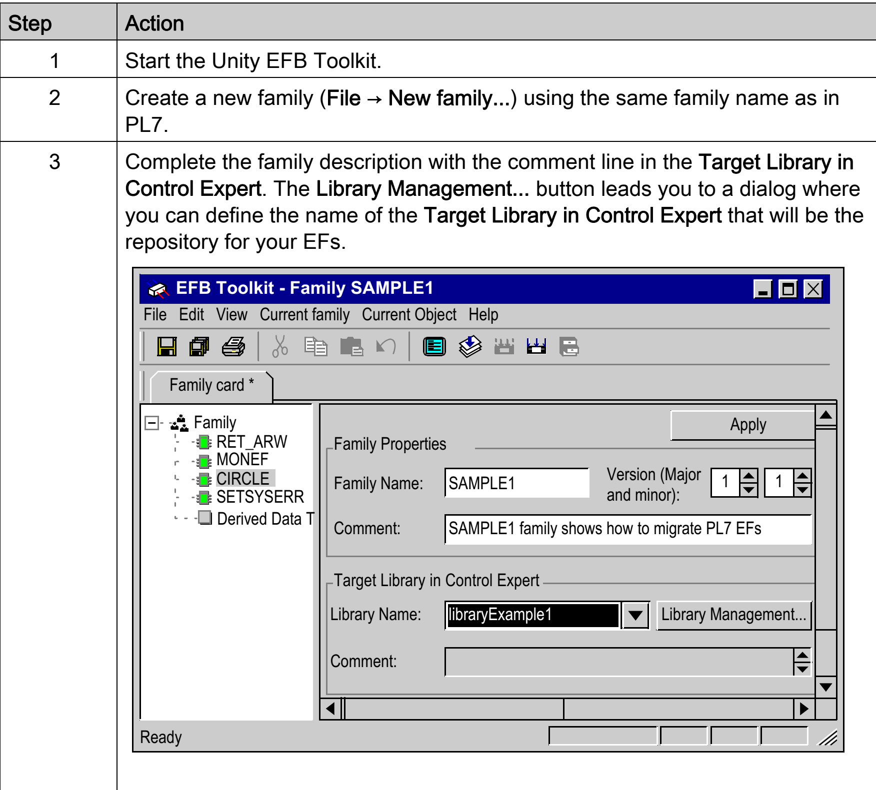
Retrieval

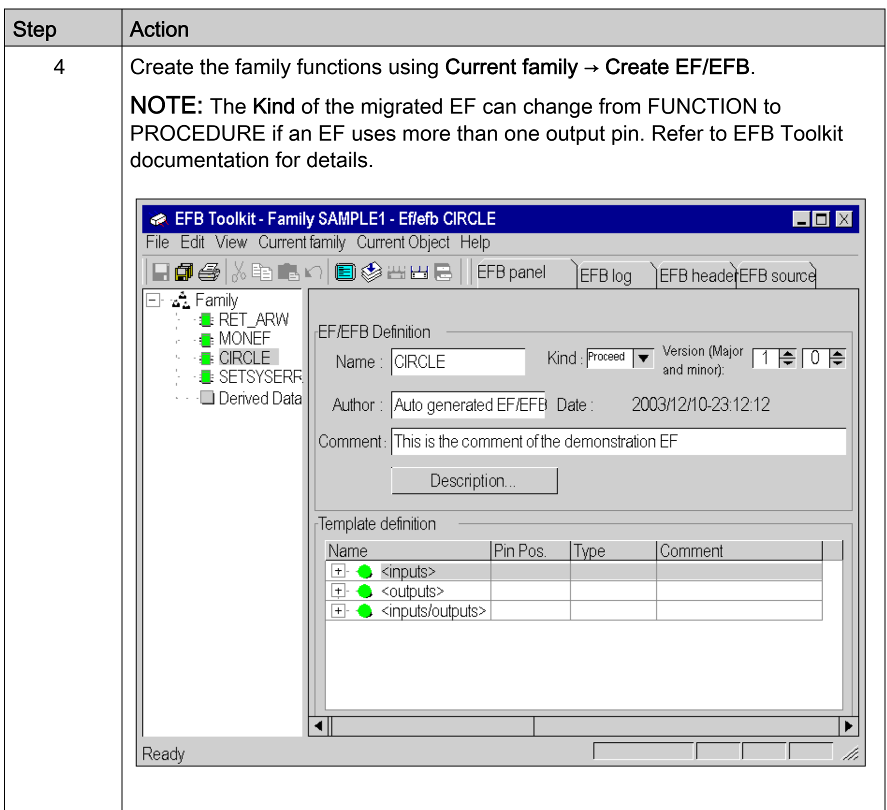


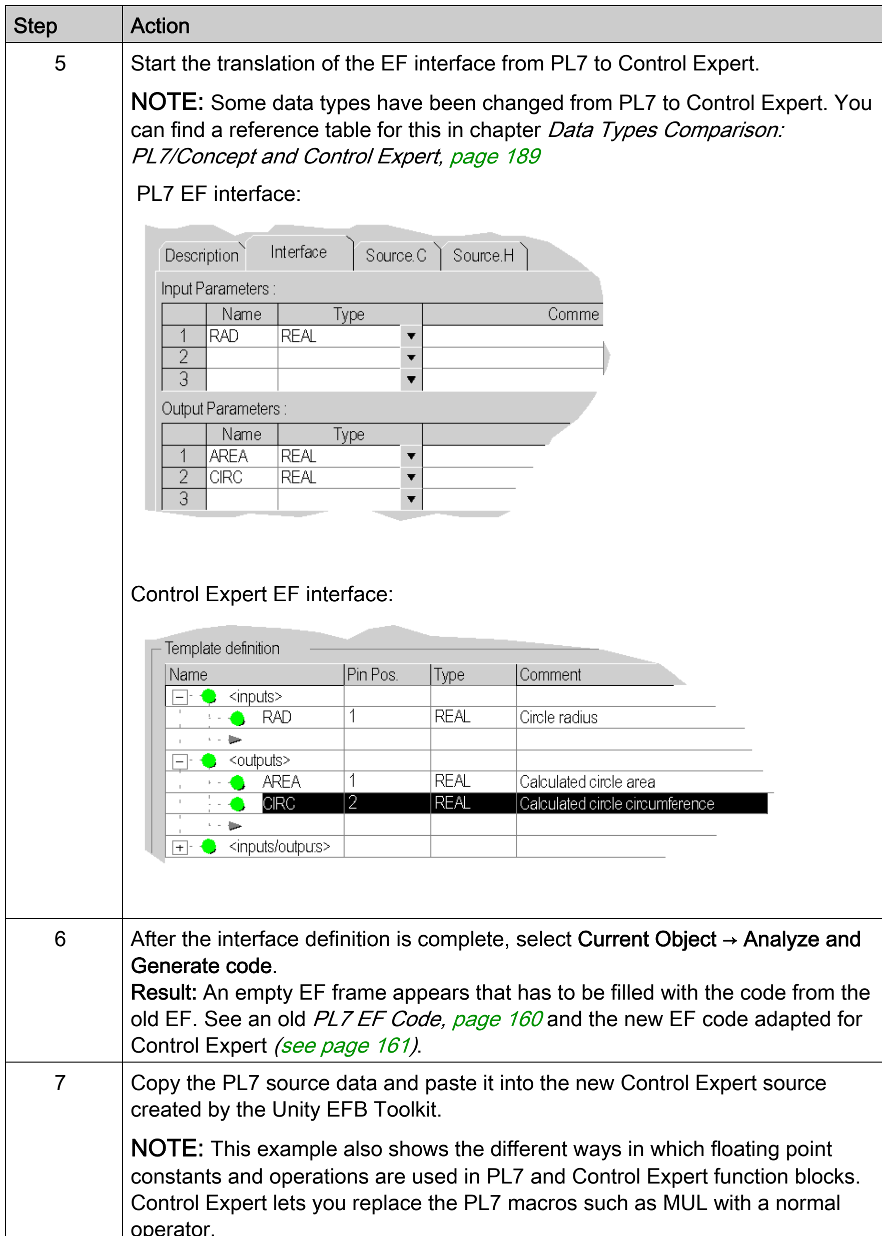
Step	Action
1	Start the PL7 EF Toolkit, and open a family class.
2	Export an EF class with the **Export** dialog. **Result:** A message appears indicating the location of the newly created file. The export file is a ZIP archive that contains, among other things, the function block sources needed for EF migration.
3	Open the archive file with a standard ZIP tool such as WinZip and extract the contents. **Result:** In the archive file you will see files for each EF of the class with the name CLASS01.H and a pair of header and C-File named as MAIN01xx.H(.C). For example: SAMPLE family (class): CLASS01.H EF: CIRCLE MAIN0101.H, MAIN0101.C

Migration Procedure

Sample Procedure

Step	Action
1	Start the Unity EFB Toolkit.
2	Create a new family (**File** → **New family...**) using the same family name as in PL7.
3	Complete the family description with the comment line in the **Target Library in Control Expert**. The **Library Management...** button leads you to a dialog where you can define the name of the **Target Library in Control Expert** that will be the repository for your EFs. 

Step	Action
4	Create the family functions using **Current family → Create EF/EFB**. **NOTE:** The **Kind** of the migrated EF can change from FUNCTION to PROCEDURE if an EF uses more than one output pin. Refer to EFB Toolkit documentation for details. 



Step	Action
5	Start the translation of the EF interface from PL7 to Control Expert. **NOTE:** Some data types have been changed from PL7 to Control Expert. You can find a reference table for this in chapter *Data Types Comparison: PL7/Concept and Control Expert, page 189* PL7 EF interface: Control Expert EF interface:
6	After the interface definition is complete, select **Current Object → Analyze and Generate code**. **Result:** An empty EF frame appears that has to be filled with the code from the old EF. See an old *PL7 EF Code, page 160* and the new EF code adapted for Control Expert *(see page 161)*.
7	Copy the PL7 source data and paste it into the new Control Expert source created by the Unity EFB Toolkit. **NOTE:** This example also shows the different ways in which floating point constants and operations are used in PL7 and Control Expert function blocks. Control Expert lets you replace the PL7 macros such as MUL with a normal operator.

PL7 EF Code

Sample Code

```
//PL7SDK_BEGIN_FUNCTION_NAME
// do not modify the generated function prototype please
// The main EF Function must be the first in this file
#include <Cstsyst.h>
#include <Fctsyst.h>
#include <C:\ASAWTEMP\SDKC\CLASSE01.H >
#include <C:\ASAWTEMP\SDKC\MAIN0101.H >
void far pascal CIRCLE(
                  unsigned long RAD, //
                  adrSCAL AREA, //
                  adrSCAL CIRC //
                  )
//PL7SDK_END_FUNCTION_NAME


{
   FLOAT radius, pi, c2;
   unsigned long far *pArea, far *pCirc ;

   radius = GET_FLOATPL7(RAD) ; // conversion to a  floating point

   /// processing
   pi = 0x40490FF9UL; // corresponds with the float  value PI=3.1415999
   c2 = 0x40000000UL; // corresponds with the float  value 2.0

   pArea = GET_ADDR( AREA.selecteur, AREA.offset );

   pCirc = GET_ADDR( CIRC.selecteur, CIRC.offset );

   if( pArea && pCirc )
   {
       *pArea = FLOAT_TOPL7(MUL(pi,radius), radius));
       *pCirc = FLOAT_TOPL7(MUL(c2,pi), radius));
   }
}
```

Control Expert EF Code

Sample Code

```
//{{ SDKC_HEADER_BEGIN  Do not edit. Any modification  would be lost
// Filename : C:\Program Files\Schneider Electric\
EFBToolkit\ FFBDev\Sample\code\CIRCLE.c
//PROCEDURE     : CIRCLE
//Version       : 1.0
//Author        : Auto generated EF/EFB
/*
PL7 migration example
This EF calculates the area and the circumference
to a given radius of a circle. */

#include "CIRCLE.h"

//}} SDKC_HEADER_END


// TODO : Write here additional declarations.

 Floating point constants declaration
const float in_Code cllbk_PI = (float)3.1415999;
s_Declare_Logical (cllbk_PI);

const float in_Code cllbk_2_0 = (float)2.0;
s_Declare_Logical (cllbk_2_0);

//{{ SDKC_PROTOTYPE_BEGIN Do not edit. Any modification
would be lost IECBool fb_call_model CIRCLE(
      const IECReal RAD,            // Circle radius
      IEC_PARAM_RTE_OFFSET AREA,    // Calculated circle area
      IEC_PARAM_RTE_OFFSET CIRC      // Calculated circle  circumference
)
//}} SDKC_PROTOTYPE_END
{
   // TODO : Write here variables declarations.
   IECReal *pArea, *pCirc, pi, c2;

   // TODO : Write here the code for your function block.
   pArea = s_log_to_phy( AREA );
   pCirc = s_log_to_phy( CIRC );

   pi = *((IECReal *)s_Const_Instance(cllbk_PI));
```

```
    c2 = *((IECReal *)s_Const_Instance(cllbk_2_0));

    // Area calculation
    *pArea = (IECReal)( pi * RAD * RAD );

    // Circumference calculation
    *pCirc = c2 * pi * RAD;

    return TRUE ; // ENO value
}
```

Section A.3
Concept EF/EFB Migration Procedure

Summary

The following section describes the Concept EF/EFB migration procedure.

Migrated applications may not operate exactly as the original.

⚠ WARNING

UNEXPECTED EQUIPMENT OPERATION

- Perform a risk analysis of the migrated application prior to commissioning.
- Apply and test the preventive and detective controls from the risk analysis before implementation.
- Test each implementation of the migrated application for proper operation before placing into service.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

What Is in This Section?

This section contains the following topics:

Topic	Page
Retrieving Source Files from a Concept Development Environment	164
Migrating the Function Block Code to Control Expert	165
Concept EF/EFB Code	168
Control Expert EF Code	169

Retrieving Source Files from a Concept Development Environment

Retrieval

The Concept EFB tool stores the sources for an EF/EFB in a folder named for the function itself; here we find the sources for migration:

```
<Library.Name>
          .BLD          Build information for  16-bit and 32-bit
          .<EF/Efb>     Function (block) definition  file, C-
 and Header files.
```

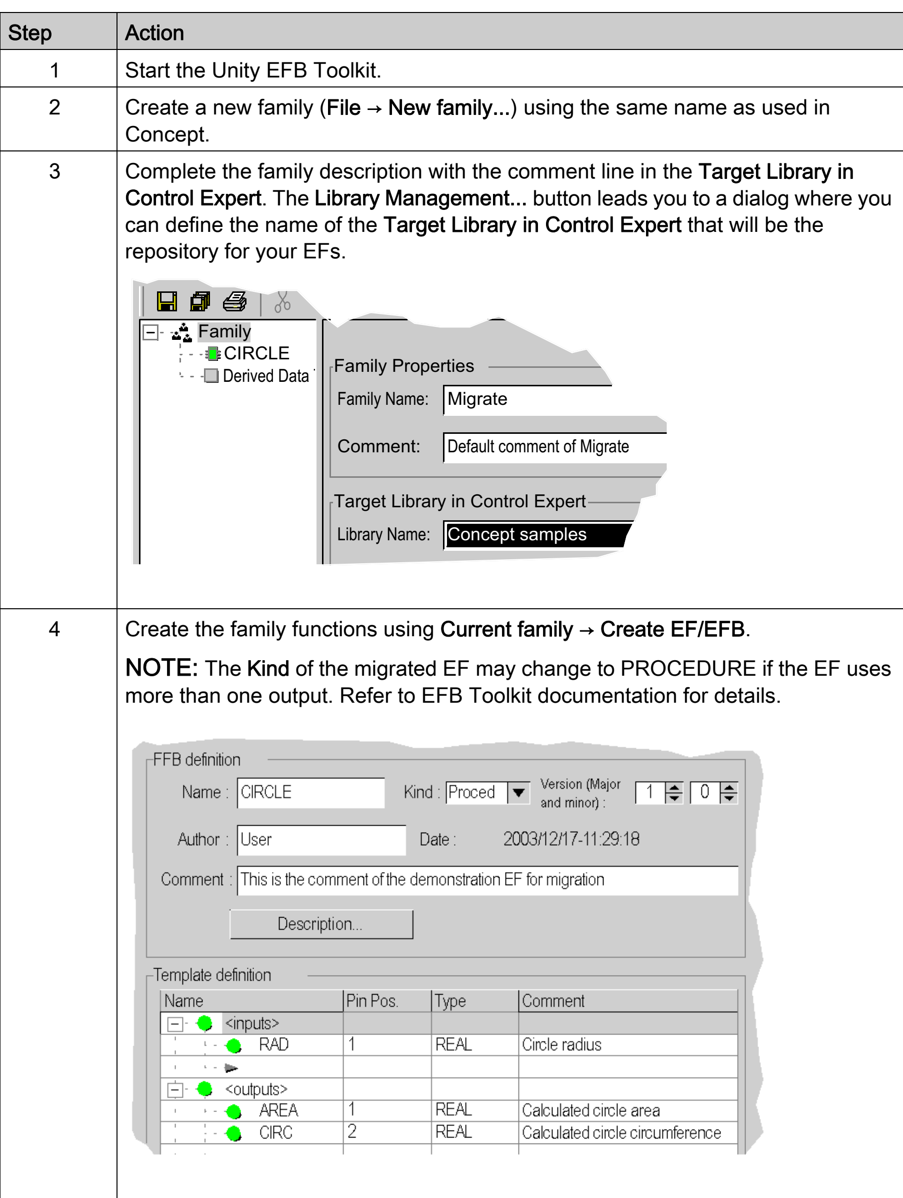
The table below compares the Concept file types to corresponding file types in Control Expert:

Concept Files		Control Expert	
Extension	Type	Extension	Type
.fb	function (block) definition file	.dsc	function (block) descriptor
.c	C source code	.c	C source code
.h	header file	.h	header file

Migrating the Function Block Code to Control Expert

Sample Procedure

Step	Action
1	Start the Unity EFB Toolkit.
2	Create a new family (**File → New family...**) using the same name as used in Concept.
3	Complete the family description with the comment line in the **Target Library in Control Expert**. The **Library Management...** button leads you to a dialog where you can define the name of the **Target Library in Control Expert** that will be the repository for your EFs.
4	Create the family functions using **Current family → Create EF/EFB**. **NOTE:** The **Kind** of the migrated EF may change to PROCEDURE if the EF uses more than one output. Refer to EFB Toolkit documentation for details.



<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>5</td><td>Start the translation of the EF interface from Concept to Control Expert, using the content from the <efb>.FB file. Here is an example:
<pre>
//:---
//:SUBSYSTEM: EFB - Elementary Function
//:
//:MODULE: ..\MIGRATE\CIRCLE.FB
//:
//:---
//:Revision: 2.1 Modtime: 22 Jan 1998 AUTHOR: Any User
//:---
Declaration of Elementary Function Block: CIRCLE
Author: Any User
Editor Group: Test Function
Major Version: 1
Minor Version: 0
Description: A full working test example.
//
// Not sure what to do ?
// Try generate-
files, make and install on this working example!
//
//Example:
// --------
Input: REAL RAD   # Circle radius
Output: REAL AREA   # Calculated circle area
Output: REAL CIRC   # Calculated circle circumference
</pre></td></tr>
<tr><td>6</td><td>From the <efb>.H file, you also can get a graphical view of the function block interface:
<pre>
// Elementary Function Block: CIRCLE
//
//
//   +--------------------+
// | CIRCLE |
//   REAL ---| RAD   AREA |--- REAL
// | CIRC |--- REAL
//   +--------------------+
//
</pre></td></tr>
</table>

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>7</td><td>Transfer the old Concept interface to Control Expert:

Template definition
<table>
<tr><th>Name</th><th>Pin Pos.</th><th>Type</th><th>Comment</th></tr>
<tr><td><inputs></td><td></td><td></td><td></td></tr>
<tr><td>RAD</td><td>1</td><td>REAL</td><td>Circle radius</td></tr>
<tr><td></td><td></td><td></td><td></td></tr>
<tr><td><outputs></td><td></td><td></td><td></td></tr>
<tr><td>AREA</td><td>1</td><td>REAL</td><td>Calculated circle area</td></tr>
<tr><td>CIRC</td><td>2</td><td>REAL</td><td>Calculated circle circumference</td></tr>
<tr><td></td><td></td><td></td><td></td></tr>
<tr><td><inputs/outputs></td><td></td><td></td><td></td></tr>
</table>
</td></tr>
<tr><td>8</td><td>Complete the interface definition. Then select the Current Object → Analyze and Generate code menu operation.
Result: An empty EF frame appears that has to be filled with the code from the old EF. See an example of old Concept EF/EFB Code, page 168 and the new EF code adapted for Control Expert (see page 169).</td></tr>
<tr><td>9</td><td>Copy the code from the Concept source and paste it into the new Control Expert source created by the Unity EFB Toolkit.</td></tr>
</table>

Note

This example shows the different ways in which floating point constants and operations are used in Concept and Control Expert function blocks. In Concept ,values are passed to EFBs by reference; this fact should be considered when migrating Concept EF/EFBs to Control Expert.

Concept EF/EFB Code

Sample Code

```
//:-------------------------------------------------------
//:SUBSYSTEM: EFB - Elementary Function Block
//:
//:MODULE: CIRCLE
//:
//:-------------------------------------------------------
//:Revision: 1.0 Modtime: 17 Dec 2003 AUTHOR: Any User
//:-------------------------------------------------------
//:DESCRIPTION: A full working test example.'
//:
//:REMARKS:
//:-------------------------------------------------------

#include "MIGRATE.I"

extern"C" BOOL FB_CALL_CONV CIRCLE(
    const PTR_REAL   RAD    ,   //Circle radius
          PTR_REAL   AREA   ,   //Calculated circle area
          PTR_REAL   CIRC   )   //Calculated circle  circumference
{
       REAL radius;

       radius = *RAD;

       if( *RAD < 0.0 )
       {
              AliPutFbdError(E_DUMMY_SOURCE_CODE);
              return FALSE;
       }
       else
       {
              *AREA = 3.14 * radius * radius;
              *CIRC = 2 * 3.14 * radius;
       }
       return TRUE;
}
```

Control Expert EF Code

Sample Code

```
//{{ SDKC_HEADER_BEGIN  Do not edit. Any modification  would be lost
// Filename : C:\Program Files\Schneider Electric\
EFBToolkit\FFBDev\Sample\code\CIRCLE.c
//PROCEDURE     : CIRCLE
//Version       : 1.0
//Author        : Auto generated EF/EFB
/*
PL7 migration example
This EF calculates the area and the circumference
to a given radius of a circle. */

#include "CIRCLE.h"

//}} SDKC_HEADER_END


// TODO : Write here additional declarations.

// Floating point constants declaration
const float in_Code cllbk_PI = (float)3.1415999;
s_Declare_Logical (cllbk_PI);

const float in_Code cllbk_2_0 = (float)2.0;
s_Declare_Logical (cllbk_2_0);

//{{ SDKC_PROTOTYPE_BEGIN Do not edit. Any modification
would be lost IECBool fb_call_model CIRCLE(
      const IECReal RAD,            // Circle radius
      IEC_PARAM_RTE_OFFSET AREA,    // Calculated circle area
      IEC_PARAM_RTE_OFFSET CIRC     // Calculated circle  circumference
)
//}} SDKC_PROTOTYPE_END
{
   // TODO : Write here variables declarations.
   IECReal *pArea, *pCirc, pi, c2;

   // TODO : Write here the code for your function block.
   pArea = s_log_to_phy( AREA );
   pCirc = s_log_to_phy( CIRC );

   pi = *((IECReal *)s_Const_Instance(cllbk_PI));
```

```
    c2 = *((IECReal *)s_Const_Instance(cllbk_2_0));

    // Area calculation
    *pArea = (IECReal)( pi * RAD * RAD );

    // Circumference calculation
    *pCirc = c2 * pi * RAD;

    return TRUE ; // ENO value
}
```

Section A.4
Other Case Studies

Summary

The following section describes the use of defined data types (PL7 only), floating point constants, `ANY` data types, extensible inputs (Concept only), PLC state determination and user defined error reporting.

What Is in This Section?

This section contains the following topics:

Topic	Page
User-defined Data Types (PL7 only)	172
Floating Point Constants	175
ANY... Data Types	176
REF... Data Types	177
Determining the PLC State	178
Reporting User Defined Errors	179
Extensible inputs	180

User-defined Data Types (PL7 only)

Introduction

The following migration example is for user-defined data types. EFs often require the use of structured data types (derived data types in Control Expert) to allow better and more formalized access to variables inside an EF. If these data types were to be used in a PL7 EF interface, the PL7 application would need to reserve sufficient memory as required by EF.

Typically in a PL7 implementation, the address to the reserved word space is passed to the EF; internally the EF maps the memory to a structured variable type.

```
short far pascal VT_AGCOM(
                  adrTABLE CONF, //configuration table
...
)
//PL7SDK_END_FUNCTION_NAME
{
  CONFIG far    *pConfig;    // Ptr. To the  configuration table
  // Get the pointer and check that enough
memory has been reserved from application
  // for configuration data.
  pConfig = GET_ADDR(CONF.selecteur, CONF.offset);
  if ((DWORD) pConfig == INVALID_ADDRESS || CONF.taille
< sizeof(CONFIG)
  / 2)
  return PARAM_ERR,
... .
}
```

With Control Expert, the user-defined DDTs can also be used in the application code. You can define the DDT inside the Unity EFB Toolkit. After that, the DDT have to be defined as the pin type in the interface definition inside the EFB Toolkit. More details will follow.

The interface in Control Expert is more in line with a standard programmer’s interface.

```
IECBool fb_call_model VT_AGCOM(
       IEC_PARAM_RTE_OFFSET CONF,  // Configuration table
...
)
{
  CONFIG *pConfig;
  pConfig = (CONFIG *)s_log_to_phy( CONF );
...
}
```

The first impression is that there is no difference between PL7 and Control Expert, but in fact the difference is on the application side. As in Control Expert, you may (and should) define a variable of type `CONFIG`, but in PL7 it is only an array of words such as %MWxyz:NN.

Control Expert helps connecting the variables of the correct type to the function block.

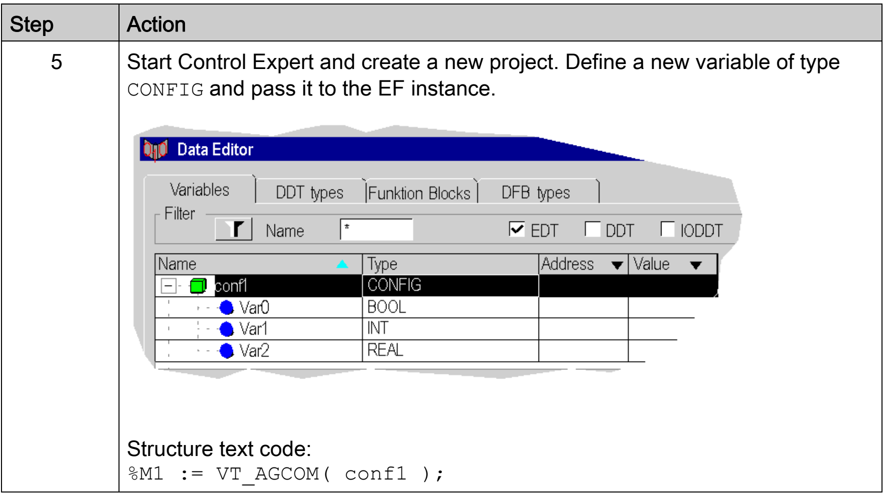
The DDTs defined in the Unity EFB Toolkit are available inside Control Expert together with the function block family.

Example

The following example shows the procedure for the EF code described above:

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>1</td><td>Create a new DDT named <code>CONFIG</code> for the structure from the PL7 code shown below. Select Current family → Create DDT to create the DDT. Define all the structure components, including the sub-type DDTs if required.

Structure definition
<table>
<tr><th>Name</th><th>Type</th><th>Comment</th></tr>
<tr><td><struct></td><td></td><td></td></tr>
<tr><td>Var0</td><td>BOOL</td><td>comment var 0</td></tr>
<tr><td>Var1</td><td>INT</td><td></td></tr>
<tr><td>Var3</td><td>REAL</td><td></td></tr>
</table></td></tr>
<tr><td>2</td><td>Change the EF interface for the use of the defined data types:<pre>
//{{ SDKC_PROTOTYPE_BEGIN Do not edit. Any modification
would be lost
IECBool fb_call_model VT_AGCOM(
        IEC_PARAM_RTE_OFFSET CONF,   // Configuration
table
        IEC_PARAM_RTE_OFFSET OUT   // RETURNED VALUE -
 This is the output pin
//}} SDKC_PROTOTYPE_END
{
   // TODO : Write here variables declarations.
   CONFIG *pConfig;
   // TODO : Write here the code for your function block.
   pConfig = (CONFIG *)s_log_to_phy(CONF);
   return TRUE ; // ENO value
}
</pre></td></tr>
<tr><td>3</td><td>Select Current family → Rebuild all to prepare the EF for installation.</td></tr>
<tr><td>4</td><td>Select File → Install family to install the family.</td></tr>
</table>

Step	Action
5	Start Control Expert and create a new project. Define a new variable of type `CONFIG` and pass it to the EF instance.  Structure text code: `%M1 := VT_AGCOM( conf1 );`

Floating Point Constants

When Not to Use Floating Point Constants

Do not use floating point constants directly in the EF/EFB code. For example, the EF cannot use the following code:

Illegal implementation

```
FunctionA(...)
{
       IECReal r1;

       ....

       If(r1<0.0) //THIS LEADS TO FLOATING  POINT CONSTANT DEFINITION
       {
            //do something
       }
}
```

A valid implementation

```
//We define on the global level the floating  point constants as first:

const float in_Code cllbk_fconst_0 = (float)0.0;
s_Declare_Logical(cllbk_fconst_0);

FunctionA(...)
{
       IECReal r1, fconst_0;

       ....

       //Initializing of the floating point constants

       fconst_0= *((IECReal *)s_Const_Instance (cllbk_fconst_0));

       if(r1<fconst_0) //NOW WE CAN USE IT WITHOUT TROUBLES
       {
            //do something
       }
}
```

ANY... Data Types

Introduction

Data types of type `ANY` are always passed to an EF by reference, whereby the member parameter `length` represents:

- the number of bytes for `ANY_<type>`
- the number of elements for `ANY_ARRAY_<type>`

Example

The following example shows the use of `ANY_ARRAY_WORD` in the EF interface

```
IECBool fb_call_model EF1(
        IEC_PARAM_RTE_OFFSET_LG ARWORD, // This is the  input pin
        IEC_PARAM_RTE_OFFSET OUT    // RETURNED VALUE -
 This is the output pin
)
//}} SDKC_PROTOTYPE_END
{
        // TODO : Write here variables declarations.
        struct
        {
                IECWord a, b, c;
        }
        structure_X;
        // TODO : Write here the code for your  function block.
        if( ARWORD.Length < sizeof( structure_X )/2 )
        {
                // Problem with the size of passed input
                return FALSE; // ENO false,  because of problems
        }
        else // processing
        {
                IECWord *pWordArr;
                pWordArr = ( IECWord *)s_log_to_phy
( ARWORD.IECPtr_Log );
                // Make a copy of the incoming data and then
process the structure_X structure_X.a = *pWordArr;
                // ...
        }
        return TRUE ; // ENO value
}
```

REF... Data Types

Introduction

Data types of type `REF` are always passed to an EF by reference, whereby the member parameter `length` represents:

- the number of bytes for `REF_<type>`
- the number of elements for `REF_<type>`

Example

The following example shows the use of `REF_ANY` in the EF interface

```
IECBool fb_call_model REF EX(
        IEC_PARAM_RTE_OFFSET_LG INP, // This is the input pin
        IEC_PARAM_RTE_OFFSET OUTP  // RETURNED VALUE -
 This is the output pin
)
//}} SDKC_PROTOTYPE_END
{
        // TODO : Write here variables declarations.
        struct
        {
                IECWord a, b, c;
        }
        structure_X;
        // TODO : Write here the code for your function block.
        if( INP.Length < sizeof( structure_X )/2 )
        {
                // Problem with the size of passed input
                return FALSE; // ENO false, because of problems
        }
        else // processing
        {
                IECWord *pWordArr;
                pWordArr = ( IECWord *)s_log_to_phy ( INP.IECPtr_Log );
        }
        return TRUE ; // ENO value
}
```

Determining the PLC State

Introduction

In some cases, an EF may need to know whether the PLC has performed a cold or warm-start. The EF may execute special code to reset or initialize data.

A System Function Example

```
// ...

PROG_STATE ps;

s_AliGetProgStat( &ps );

if( ps.coldInit || ps.warmInit )
{
       // DO SOMETHING SPECIAL

       // ...
}
```

Reporting User Defined Errors

Introduction

An EF/EFB can report its own detected error code when processing event occurs. The EF/EFB may call the system function `s_set_ffb_error` with a detected error code derived from codes predefined in the "SystemLib.h" header file. The system function creates a detected error record in the diagnostic buffer that is shown later in a diagnostic display.

A Diagnostic Code Example

```
// Map the predefined error codes to own definition
#define ERR_PROBLEM_WITH_xxx E_EFB_USER_ERROR_xy
if( error_condition )
{
         s_set_ffb_error(ERR_PROBLEM_WITH_xxx);
}
```

Extensible inputs

Introduction

The following example demonstrates migration for functions that use a variable number of inputs (extensible pins). The code comes from the Concept SAMPLE library.

Concept Definition

EXTINP.FB

```
//:-------------------------------------------------------
//:SUBSYSTEM: EFB - Elementary Function
//:
//:MODULE:          ..\SAMPLE\EXTINP.FB
//:
//:-------------------------------------------------------
Declaration of Elementary Function: SMPL_EXTINP
Author:
Editor Group: Sample
Major Version: 1
Minor Version: 0
Description: A sample to show usage of extensible inputs
  The sum of all inputs is calculated and copied to  the output OUT
  Note: No error checks are made on overflow!
Input (2..32, default=2): INT IN # extensible inputs
Output: INT hide (OUT)           # computed output
```

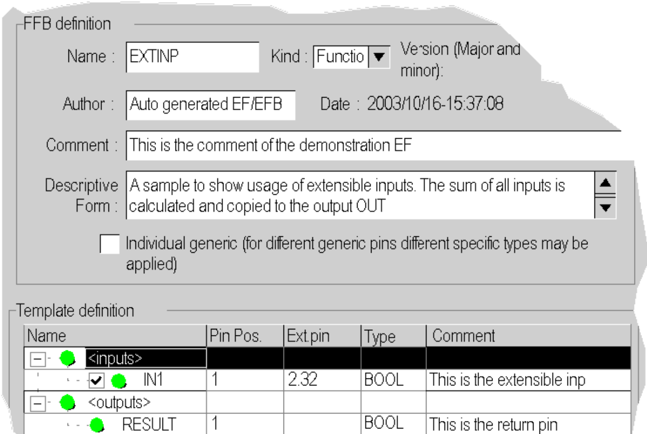
Concept implementation: EXTINP.C

```
//:-------------------------------------------------------
//:SUBSYSTEM: EFB - Elementary Function
//:
//:MODULE:          SAMPLE_EXTINP
//:
//:-------------------------------------------------------
//:Revision: 1.0 Modtime: 17 Dec 1996
//:AUTHOR:  Jochen Lichtenfels
//:-------------------------------------------------------
//:DESCRIPTION: A sample to show usage of extensible inputs
//:
//:REMARKS:
//:-------------------------------------------------------
#include "SAMPLE.I"
extern"C" BOOL FB_CALL_CONV SMPL_EXTINP(
          INT       nin   ,   // No. of inputs
          PTR_INT   OUT   ,   //computed output
// const PTR_INT    IN    ,   //extensible inputs
```

```
              ...               )
{
open_IN;
  {
  INT sum = 0;
  for (int n = 0; n < nin; n++)
    {
    sum += *next_IN;
    }
  *OUT = sum;
  }
close_IN;
return TRUE;
}
```

Control Expert Implementation

The IN1 input is marked as extensible. The column "Ext.Pin" defines the maximum number of extensible inputs.




```
//{{ SDKC_HEADER_BEGIN Do not edit. Any modifications would be lost
// Filename : C:\Program Files\Schneider Electric\EFBToolkit\FBDdev\sam
pleFamily2\code\EXTINP.c
// EF         : EXTINP
// Version    : 1.0
// Author     : Auto generated EF/EFB
/*
This is the comment of the demonstration EF
A sample to show usage of extensible inputs. The sum of all inputs is c
```

```
alculated and copied to the output OUT.
Note: An extensible function must be defined as an EF -
 Extensible EFBs are not allowed. */
#include "EXTINP.h"
// Macros for accessing extensible pin values
#define open_VARG        { va_list _theVariableList ;
va_start (_theVariableList, RESULT)
#define next_VARG        va_arg(_theVariableList,  const IECBool)
#define close_VARG       va_end(_theVariableList); }
#define NIN_MIN          2   // the minimum value  for parameter nin
#define NIN_MAX          32  // the maximum value  for parameter nin
//}} SDKC_HEADER_END
// TODO : Write here additional declarations.
//{{ SDKC_PROTPTYPE_BEGIN Do not edit. Any modifications
would be lost IECBool fb_call_model EXTINP(
        const NIN_T nin,              // EXTENSIBLE PIN -
 Number of extra parameters
        IEC_PARAM_RTE_OFFSET RESULT,  // RETURNED VALUE -
 This is the return pin
        //const IECBool IN1           // EXTENSIBLE PIN -
 This is the extensible input pin
        ...
)
//}} SDKC_PROTOTYPE_END
{
//**********************************************************
//
// Note: This is an EF. Parameters are not passed
//   using a pointer to an instance structure.
//   They are pushed directly to the stack so you
//   do not need to define a pointer to the
//   instance and set the pointer first.
//
//   An EF must have a return value (not the ENO).
//   OUT is defined as the return value in the
//   EF description. Notice there is no return
//   statement needed for OUT but the ENO must
//   still be returned.
//
//   nin is passed as a call by value parameter.
//
//   OUT is passed as a logical pointer and
//   must be converted to physical before it
//   can be used.
//   Defines for accessing the variable input
```

```
//   list are defined above.
//   Notice in the code the parameter name
//   "IN" is not used - its already in the defines.
//
//   Note also, unlike an EFB, there is no
//   system entry point.
//
//***********************************************************
  // TODO : Write here variables declarations.
  IECInt * pRet;
  IECInt   sum,n;
  // TODO : Write here the code for your function block.
  pRet = s_log_to_phy(RESULT); // get a physical pointer
to the input structure
  //Setup variables
  sum = 0;
  // Start access to extensible pins
  open_VARG;
  for (n = 0; n < nin; n++)
  {
         sum += next_VARG;
  }
  close_VARG;
  // Put the value to return
  *pRet = sum;
  return TRUE ; // ENO value
}
```

Section A.5
Comparison Between PL7/Concept and Control Expert

Summary

The following section describes the use of system functions, data types and definitions in the include file.

What Is in This Section?

This section contains the following topics:

Topic	Page
Common System for EF/EFBs in Control Expert	185
Data Types Comparison: PL7/Concept and Control Expert	189
PL7 SDKC Include File: <Cstsyst.h> Versus <SystemLib.h> (PL7 Only)	191

Common System for EF/EFBs in Control Expert

The SystemLib Header File

The "SystemLib.h" header file provides the following definitions for the use of system functions.

Function	Description	PL7/Concept equivalent
s_Declare_Logical	MACRO: associates a symbol with a logical variable	
s_Logical_Instance	MACRO: instantiates a logical address from a symbol	
s_Const_Instance	MACRO: creates a physical pointer to a constant	
s_AliGetProgStat	Gets program-specific status information	AliGetProgState(Ex)
s_log_to_phy	Converts a logical address into a physical address	phy_ptr_init
s_obj_to_log	Gets the logical address of an application object (%S, %SW, %M, %MW)	
s_rd_16bits	Reads 16 consecutive bits in EBOOL memory	rd_16bits
s_rd_1bit	Reads one bit value in EBOOL memory	rd_1bit
s_rd_bit_attrib	Read attributes bits (F, H, D) in EBOOL Memory	rd_bit_attrib
s_rd_internalwords	Read N consecutive %MWi words	
s_rd_Nbits	Reads *n* consecutive bit values in EBOOL memory	rd_bits
s_rd_sysbit	Reads a system bit (%Si)	rd_sysbit
s_rd_sysword	Reads a system word (%SWi)	rd_sysword
s_wr_16bits	Writes *n* consecutive bit values in EBOOL memory	wr_16bits
s_wr_1bit	Writes one bit value in EBOOL memory	wr_1bit
s_wr_bits_attrib	Changes attribute bits in *n* consecutive EBOOLs	

Function	Description	PL7/Concept equivalent
s_wr_internalwords	Writes *n* consecutive %MWi words	
s_wr_Nbits	Reads *n* consecutive bits in EBOOL memory	wr_bits
s_wr_sysbit	Writes a system word (%SWi)	wr_sysbit
s_wr_sysword	Writes a system bit (%Si)	wr_sysword
s_cnt_100ms	Reads the value of an internal 100 ms counter	cnt_100ms
s_cnt_10ms	Reads the value of an internal 10 ms counter	cnt_10ms
s_cnt_1ms	Reads the value of an internal 1 ms counter	
s_date_and_time	Reads current PLC date and time	date_and_time
s_syscnt_10ms	Reads the value of the system 10 ms counter	
s_current_task	Returns the current task number	
s_set_ffb_error	Registers a detected error of an EF/EFB without an additional parameter.	AliPutFbdError
s_set_ffb_error_addi	Registers a detected error of an EF/EFB with an additional parameter.	AliPutFbdError
s_of_passw_check	Accesses a PCMCIA card signature; if not OK, the PLC is inoperable.	of_passw_check
s_of_passw_test	Accesses a PCMCIA card signature; if not OK, returns a Diagnostic Code	
_clear87	Gets and clears the floating-point status word	
_control87	Gets and sets the floating-point control word	
_status87	Gets the floating point status word	GET_STATUS (macro)

Function	Description	PL7/Concept equivalent
_chgsign	Changes the sign of a float (32-bit only)	CHS
Acos	Calculates the arccosine	ACOS
Asin	Calculates the arcsine	ASIN
Atan	Calculates the arctangent	ATAN
Atof	Converts string to double (single if 16-bit)	
Cos	Calculates the cosine	COS
Exp	Calculates the exponential	EXP
Fabs	Calculates the absolute value	
Fmod	Calculates the floating-point remainder	ABS
Ftoa	Converts a floating-point value to a string	
Log	Calculates a logarithm	LN
log10	Calculates a base-10 logarithm	
Pow	Calculates *x* raised to the power of *y*	
Sin	Calculates the sine	SIN
Sqrt	Calculates the square root	SQRT
Tan	Calculates the tangent	TAN
s_demask_it	Unmasks a critical section for interrupts	
s_GetUSecs	Gets the current value of a microsecond counter	AliGetUSecs
s_mask_it	Masks a critical section from interrupts	
s_proc_indic	Returns the PLC state, including memory configuration	proc_indic
s_proc_type	Gets the PLC type	proc_type
s_event_func_delete	Suppresses an object in the FIFO event queue	
s_event_func_link	Adds an object to the FIFO event queue	

Function	Description	PL7/Concept equivalent
s_event_state	Searches an object in the FIFO event queue	
_lrotl	Rotates an unsigned long value left	
_lrotr	Rotates an unsigned long value right	
_rotl	Rotates an unsigned value left	
_rotr	Rotates an unsigned value right	
_strset	Sets all the characters of a string	
Abs	Returns the absolute value of an integer	
Labs	Returns the absolute value of a long-integer	
Memcmp	Compares *n* bytes in two buffers	
Memcpy	Copies *n* bytes from a source buffer to a destination buffer	
Memset	Sets *n* bytes in a buffer	
Strcat	Appends string2 to string1	
Strcmp	Compares *n* bytes in two strings	
Strcpy	Copies string1 to string2	
Strlen	Returns the length of a string in bytes	
long +, -, *, /		
long %, <<, >>		
long <, >, <=, >=, Identical, !=		
float +, -, /, *		ADD, SUB, MUL, DIV
float <, >, <=, >=, Identical, !=		EQU, SUP, INF
- (float change sign operator)		CHS
cast float to (long)		FTOL
cast float to (unsigned long)		FTOL
cast to (float)		LTOF

Data Types Comparison: PL7/Concept and Control Expert

Introduction

PL7/Concept	Control Expert	Input	Output	Input/Output
BOOL		Short	adrNFBIT	AdrNFBIT
	BOOL	IECBool	Address-Type1 **NOTE:** EC_PARAM_RTE_ OFFSET: [block, offset]. Structure containing memory block and offset	Address-Type1
EBOOL		AdrFBIT	AdrFBIT	AdrFBIT
	EBOOL	Not allowed	Address-Type1	Address-Type1
WORD		Short	AdrSCAL	AdrSCAL
	WORD	IECWord	Address-Type1	Address-Type1
REAL		Unsigned long	AdrSCAL	AdrSCAL
	REAL	IECReal	Address-Type1	Address-Type1
DWORD		Long	AdrSCAL	AdrSCAL
	DWORD	IECDWord	Address-Type1	Address-Type1
STRING		AdrTABLE	AdrTABLE	AdrTABLE
	STRING	Address-Type2 **NOTE:** IEC_PARAM_RTE_ OFFSET_LG: [block, offset, length]. Structure containing memory block, offset and number of type corresponding elements	Address-Type2	Address-Type2
AR_X		AdrTABLE	AdrTABLE	AdrTABLE

PL7/Concept	Control Expert	Input	Output	Input/Output
	ANY_ARRAY_BOOL	Address-Type2	Address-Type2	Address-Type2
AR_W		AdrTABLE	AdrTABLE	AdrTABLE
	ANY_ARRAY_WORD	Address-Type2	Address-Type2	Address-Type2
AR_D		AdrTABLE	AdrTABLE	AdrTABLE
	ANY_ARRAY_DWORD	Address-Type2	Address-Type2	Address-Type2
AR_R		AdrTABLE	AdrTABLE	AdrTABLE
	ANY_ARRY_REAL	Address-Type2	Address-Type2	Address-Type2

PL7 SDKC Include File: <Cstsyst.h> Versus <SystemLib.h> (PL7 Only)

Introduction

The types and constants shown below are extracted from the SystemLib.h file, which is included in the code of an EF.

Copyright Schneider Automation 1998-2002 PL7 SDKC V4.1 IE 20	UNITY EFB TOOLKIT Header file: "SystemLib.h"

Error codes
#define Status_Ok
#define STATUS_OK 0
#define OUT_OF_BOUNDS 0x19ff
#define NULL_RW_SIZE 0x19fe
#define CLOCK_UNAVAILABLE 0x3200
#define CLOCK_NOT_SUPPORTED 0x32ff
#define INVALID_LENGTH 0x55ff
 	
//declaration des constantes
#define NIL 0xFFFF
#define NILL 0xFFFFFFFF
#define INVALID_ADDRESS NILL
#define NULL_FPTR (void_far*)0
 	
//Types d'action possibles sur
//chaines de bits forcibles
(wr_bits)
 	
//Ecriture de champ data
#define DATA_WRITE 0
 	
//Ecriture du champ valeur
//de repli (ATS20 uniquement)
#define RVALUE_WRITE 4
 	
//Ecriture du champ validation
//de la valeur de repli
//(ATS20 uniquement)
#define RVALID_WRITE 6
 	
//Ecriture du défaut (ATS20
uniquement)
#define DEFAULT_WRITE 7
 	
//Déforcage global des bits
#define UNFORCE_ALL 8
	
	Not defined
	Identical
	Identical
	Identical
	Identical
	Identical
	Identical
	
	Identical
	Identical
	Identical
	NULL
	
	Identical
	
	Identical
	
	Identical
	
	Identical
	
	Identical

//Définition des types #define BOOL short #define BYTE unsigned char #define WORD unsigned short #define DWORD unsigned long	 IECBOOL, IECBool Identical Identical Identical

//Définition des structures
typedef struct {
void_near *off; //Offset
unsigned short sel; //Numéro de sélecteur
}PTR_LOG;
	
	IEC_PARAM_RTE_OFFSET

<table>
<tr><td><pre>typedef struct{
  unsigned short selecteur;
  unsigned short offset;
}adrSCAL;</pre></td><td>IEC_PARAM_RTE_OFFSET</td></tr>
<tr><td><pre>typedef struct{
  unsignded short selecteur;
  unsigned short offset;
  unsigned short taille;
}adrTABLE;</pre></td><td>IEC_PARAM_RTE_OFFSET_LG</td></tr>
<tr><td><pre>typedef struct{
  unsigned short selecteur;
  unsigned short offset;
}adrFBIT;</pre></td><td>IEC_PARAM_RTE_OFFSET</td></tr>
<tr><td><pre>typedef struct{
  unsigned short selecteur;
  unsigned short offset;
  unsigned short rang; //valeur
  discrete : 0..15
}adrNFBIT;</pre></td><td>IEC_PARAM_RTE_OFFSET</td></tr>
<tr><td><pre>typedef struct{
  unsigned char gamme;
  unsigned char version;
  unsigned char codecat;
  unsigned char unused;
}PROC_IDENT;</pre></td><td>DEVICED</td></tr>
<tr><td><pre>typedef struct{
  unsigned char reserved;
  unsigned char Second;
  unsigned char Minut;
  unsigned char Hour;
  unsigned char DayMonth;
  unsigned char Month;
  unsigned char Year;
  unsigned char Century;
  unsigned char DayWeek;
}DATE_TIME;</pre></td><td>IECDateAndTime</td></tr>
<tr><td><pre>typedef struct{
...
}PROC_INDIC;</pre></td><td>SgenInd</td></tr>
<tr><td><pre>typedef char OF_PASSW [16];</pre></td><td>unsigned char_far *</td></tr>
</table>

#ifdef SDKCV3	
//--------------------------	
typedef unsigned long FLOAT;	typedef float FLOAT;
typedef unsigned long ULONG;	Identical

typedef union
{
float fl;
unsigned long ul;
}
FLOAT_DBG;
	Undefined

typedef unsigned long ULONG;	Identical
typedef union { float fl; unsigned long ul; int i[2]; }UL_FLOAT;	Undefined
#endif	
#define flag_OK 0L	Identical
#define flag_InvalideOp 1	Identical
#define flag_Denorm 2	Identical
#define flag_ZeroDivide 4	Identical
#define flag_Overflow 8	Identical
#define flag_Underflow 16	Identical
#define EGAL_F 0	Identical
#define INF_F 1	Identical
#define SUP_F 2	Identical
#define UN_F 4	Identical
#define FLT_MAX 0x7F7FFFFF //3.402823466e+38F /*max value*/	Identical
#define FLT_MIN 0x00800000// 1.175494351e- 38F /*min positive value*	Identical
#define NEG_FLT_MAX 0xFF7FFFFF	Identical
#define INFINI_PLUS 0x7F800000	Identical
#define INFINI_MOINS 0xFF800000	Identical
#define NOT_A_NUMBER 0x7F800001 //nAN	Identical
#define F_NEGLIMIT_EXP 3266183168 //-87	Identical
#define F_POSLIMIT_EXP 1118830592 //+88	Identical

Section A.6
An Empty Frame for a Control Expert EF

Summary

The following section provides an example that shows the interface, header file, C-source template and some programming hints.

What Is in This Section?

This section contains the following topics:

Topic	Page
Example_EF1 Interface	195
Example_EF1 Header File	196
Example_EF1 C-Source Template	199
Programming Hints	203

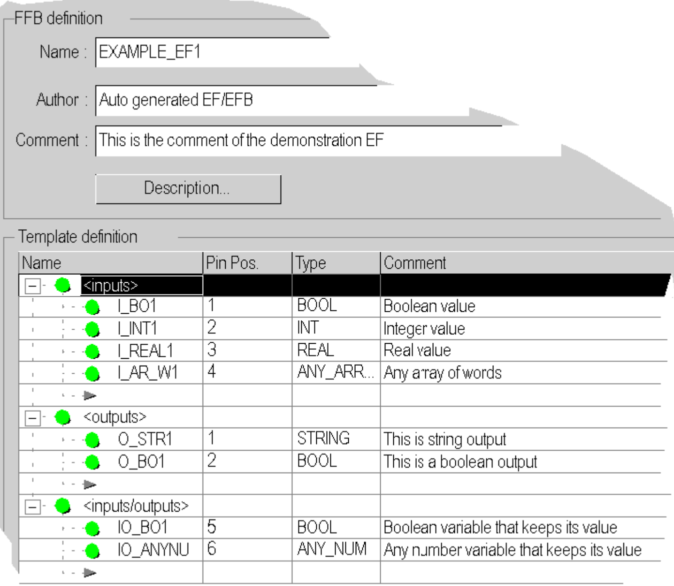
Example_EF1 Interface

Characteristics

- The **Kind** type of this function block is PROCEDURE and not FUNCTION because it uses more than one output.
- The values `IO_BO1` and `IO_ANYNU` should be used for read/write operations. Therefore they are inserted into the input/output section.
- The EF contains four code templates because of the `ANY_NUM` type of `IO_ANYNU` variable.

NOTE: When an EF has an ANY type parameter that is based on other elementary data types, it has to implement the code for each function template type created by the EFB Toolkit.

Template Definition



Example_EF1 Header File

Coding Sample

```
//{{ SDKC_HEADER_BEGIN Do not edit. Any modification  would be lost
//Filename : C:\Program Files\Schneider Electric\
EFBToolkit\FFBDev\Example1\code\EXAMPLE_EF1.h
//PROCEDURE     : EXAMPLE_EF1
//Version       : 1.0
//Author        : Auto generated EF/EFB
/*
This is the comment of the demonstration EF
This is the descriptive form of the demonstration EF */
#include "SystemLib.h"
//Additional header :
//None
//                   +--------------------+
//                   |    EXAMPLE_EF1     |
//                   +--------------------+
//                   |                    |
//                 --+- I_BO1    O_STR1 -+--
//                   |                    |
//                 --+- I-INT1            |
//                   |                    |
//                 --+- I_REAL1           |
//                   |                    |
//                 --+- I_AR_W1           |
//                   |                    |
//                 --+----- IO_BO1 ------+--
//                   |                    |
//                 --+----- IO_ANYNU-----+--
//                   |                    |
//                   +--------------------+
//
#ifdef SDKC_COMPILING_EXAMPLE_EF1_REAL
IECBool fb_call_model EXAMPLE_EF1_REAL(
       const IECBool I_BO1,            // Boolean value
       const IECInt I_INT1,            // Integer value
       const IECReal I_REAL1,          // Real value
       IEC_PARAM_RTE_OFFSET_LG I_AR_W1, // Any array  of words
       IEC_PARAM_RTE_OFFSET IO_BO1,     // Boolean variable
that keeps its value
       IEC_PARAM_RTE_OFFSET IO_ANYNU,   // Any number
variable that keeps its value
       IEC_PARAM_RTE_OFFSET_LG O_STR1,  // This is  string output
```

```
       IEC_PARAM_RTE_OFFSET O_BO1       // This is a boolean  output
) ;
#endif // SDKC_COMPILING_EXAMPLE_EF1_REAL
#ifdef SDKC_COMPILING_EXAMPLE_EF1_DINT
IECBool fb_call_model EXAMPLE_EF1_DINT(
       const IECBool I_BO1,            // Boolean value
       const IECInt I_INT1,            // Integer value
       const IECReal I_REAL1,          // Real value
       IEC_PARAM_RTE_OFFSET_LG I_AR_W1, // Any array  of words
       IEC_PARAM_RTE_OFFSET IO_BO1,     // Boolean variable
that keeps its value
       IEC_PARAM_RTE_OFFSET IO_ANYNU,   // Any number
variable that keeps its value
       IEC_PARAM_RTE_OFFSET_LG O_STR1,  // This is  string output
       IEC_PARAM_RTE_OFFSET O_BO1       // This is a boolean  output
) ;
#endif // SDKC_COMPILING_EXAMPLE_EF1_DINT
#ifdef SDKC_COMPILING_EXAMPLE_EF1_INT
IECBool fb_call_model EXAMPLE_EF1_INT(
       const IECBool I_BO1,            // Boolean value
       const IECInt I_INT1,            // Integer value
       const IECReal I_REAL1,          // Real value
       IEC_PARAM_RTE_OFFSET_LG I_AR_W1, // Any array  of words
       IEC_PARAM_RTE_OFFSET IO_BO1,     // Boolean variable
that keeps its value
       IEC_PARAM_RTE_OFFSET IO_ANYNU,   // Any number
variable that keeps its value
       IEC_PARAM_RTE_OFFSET_LG O_STR1,  // This is  string output
       IEC_PARAM_RTE_OFFSET O_BO1       // This is a boolean  output
) ;
#endif // SDKC_COMPILING_EXAMPLE_EF1_INT
#ifdef SDKC_COMPILING_EXAMPLE_EF1_UDINT
IECBool fb_call_model EXAMPLE_EF1_UDINT(
       const IECBool I_BO1,            // Boolean value
       const IECInt I_INT1,            // Integer value
       const IECReal I_REAL1,          // Real value
       IEC_PARAM_RTE_OFFSET_LG I_AR_W1, // Any array  of words
       IEC_PARAM_RTE_OFFSET IO_BO1,     // Boolean variable
that keeps its value
       IEC_PARAM_RTE_OFFSET IO_ANYNU,   // Any number
variable that keeps its value
       IEC_PARAM_RTE_OFFSET_LG O_STR1,  // This is  string output
       IEC_PARAM_RTE_OFFSET O_BO1       // This is a boolean  output
) ;
#endif // SDKC_COMPILING_EXAMPLE_EF1_UDINT
```

```
#ifdef SDKC_COMPILING_EXAMPLE_EF1_UINT
IECBool fb_call_model EXAMPLE_EF1_UINT(
       const IECBool I_BO1,            // Boolean value
       const IECInt I_INT1,            // Integer value
       const IECReal I_REAL1,          // Real value
       IEC_PARAM_RTE_OFFSET_LG I_AR_W1, // Any array  of words
       IEC_PARAM_RTE_OFFSET IO_BO1,     // Boolean variable
that keeps its value
       IEC_PARAM_RTE_OFFSET IO_ANYNU,   // Any number
variable that keeps its value
       IEC_PARAM_RTE_OFFSET_LG O_STR1,  // This is  string output
       IEC_PARAM_RTE_OFFSET O_BO1       // This is a boolean  output
) ;
#endif // SDKC_COMPILING_EXAMPLE_EF1_UINT
// }} SDKC_HEADER_END
TODO : Write here additional declaration.
```

Example_EF1 C-Source Template

Coding Sample

```
//{{ SDKC_HEADER_BEGIN Do not edit. Any modification  would be lost
//Filename : C:\Program Files\Schneider Electric\
EFBToolkit\FFBDev\Example1\code\EXAMPLE_EF1.c
//PROCEDURE   : EXAMPLE_EF1
//Version     : 1.0
//Author      : Auto generated EF/EFB

/*
This is the comment of the demonstration EF
This is the descriptive form of the demonstration EF */

#include "EXAMPLE_EF1.h"

//}} SDKC_HEADER_END

// TODO : Write here additional declarations.

//{{ SDKC_PROTOTYPE_EXAMPLE_EF1_REAL_BEGIN Do not edit.
 Any modifications would be lost
#ifdef  SDKC_COMPILING_EXAMPLE_EF1_REAL
IECBool fb_call_model EXAMPLE_EF1_REAL(
       const IECBool I_BO1,            // Boolean value
       const IECInt I_INT1,            // Integer value
       const IECReal I_REAL1,          // Real value
       IEC_PARAM_RTE_OFFSET_LG I_AR_W1, // Any array  of words
       IEC_PARAM_RTE_OFFSET IO_BO1,     // Boolean variable
that keeps its value
       IEC_PARAM_RTE_OFFSET IO_ANYNU,   // Any number
variable that keeps its value
       IEC_PARAM_RTE_OFFSET_LG O_STR1,  // This is  string output
       IEC_PARAM_RTE_OFFSET O_BO1       // This is a  boolean output
)
//}} SDKC_PROTOTYPE_EXAMPLE_EF1_REAL_END
{
       // TODO : Write here variables declarations.

       // TODO : Write here the code for your  function block.

       return TRUE : // ENO value
}
#endif
```

```
//{{ SDKC_PROTOTYPE_EXAMPLE_EF1_DINT_BEGIN
Do not edit. Any modifications would be lost
#ifdef SDKC_COMPILING_EXAMPLE_EF1_DINT
IECBool fb_call_model EXAMPLE_EF1_DINT(
       const IECBool I_BO1,            // Boolean value
       const IECInt I_INT1,            // Integer value
       const IECReal I_REAL1,          // Real value
       IEC_PARAM_RTE_OFFSET_LG I_AR_W1, // Any array  of words
       IEC_PARAM_RTE_OFFSET IO_BO1,     // Boolean variable
that keeps its value
       IEC_PARAM_RTE_OFFSET IO_ANYNU,   // Any number
variable that keeps its value
       IEC_PARAM_RTE_OFFSET_LG O_STR1,  // This is  string output
       IEC_PARAM_RTE_OFFSET O_BO1       // This is a  boolean output
)
//}} SDKC_PROTOTYPE_EXAMPLE_EF1_DINT_END
{
       // TODO : Write here variables declarations.

       // TODO : Write here the code for your  function block.

       return TRUE : // ENO value
}
#endif

//{{ SDKC_PROTOTYPE_EXAMPLE_EF1_INT_BEGIN
Do not edit. Any modifications would be lost
#ifdef SDKC_COMPILING_EXAMPLE_EF1_INT
IECBool fb_call_model EXAMPLE_EF1_INT(
       const IECBool I_BO1,            // Boolean value
       const IECInt I_INT1,            // Integer value
       const IECReal I_REAL1,          // Real value
       IEC_PARAM_RTE_OFFSET_LG I_AR_W1, // Any array  of words
       IEC_PARAM_RTE_OFFSET IO_BO1,     // Boolean variable
that keeps its value
       IEC_PARAM_RTE_OFFSET IO_ANYNU,   // Any number
variable that keeps its value
       IEC_PARAM_RTE_OFFSET_LG O_STR1,  // This is  string output
       IEC_PARAM_RTE_OFFSET O_BO1       // This is a  boolean output
)
//}} SDKC_PROTOTYPE_EXAMPLE_EF1_INT_END
{
       // TODO : Write here variables declarations.
```

```
        // TODO : Write here the code for your  function block.

        return TRUE : // ENO value
}
#endif

//{{ SDKC_PROTOTYPE_EXAMPLE_EF1_UDINT_BEGIN
Do not edit. Any modifications would be lost
#ifdef SDKC_COMPILING_EXAMPLE_EF1_UDINT
IECBool fb_call_model EXAMPLE_EF1_UDINT(
        const IECBool I_BO1,            // Boolean value
        const IECInt I_INT1,            // Integer value
        const IECReal I_REAL1,          // Real value
        IEC_PARAM_RTE_OFFSET_LG I_AR_W1, // Any array  of words
        IEC_PARAM_RTE_OFFSET IO_BO1,     // Boolean variable
that keeps its value
        IEC_PARAM_RTE_OFFSET IO_ANYNU,   // Any number
variable that keeps its value
        IEC_PARAM_RTE_OFFSET_LG O_STR1,  // This is  string output
        IEC_PARAM_RTE_OFFSET O_BO1       // This is a  boolean output
)
//}} SDKC_PROTOTYPE_EXAMPLE_EF1_UDINT_END
{
        // TODO : Write here variables declarations.

        // TODO : Write here the code for your  function block.

        return TRUE : // ENO value
}
#endif

//{{ SDKC_PROTOTYPE_EXAMPLE_EF1_UINT_BEGIN
Do not edit. Any modifications would be lost
#ifdef SDKC_COMPILING_EXAMPLE_EF1_UINT
IECBool fb_call_model EXAMPLE_EF1_UINT(
        const IECBool I_BO1,            // Boolean value
        const IECInt I_INT1,            // Integer value
        const IECReal I_REAL1,          // Real value
        IEC_PARAM_RTE_OFFSET_LG I_AR_W1, // Any array  of words
        IEC_PARAM_RTE_OFFSET IO_BO1,     // Boolean variable
that keeps its value
        IEC_PARAM_RTE_OFFSET IO_ANYNU,   // Any number
variable that keeps its value
        IEC_PARAM_RTE_OFFSET_LG O_STR1,  // This is  string output
        IEC_PARAM_RTE_OFFSET O_BO1       // This is a  boolean output
```

```
)
//}} SDKC_PROTOTYPE_EXAMPLE_EF1_UINT_END
{
	// TODO : Write here variables declarations.

	// TODO : Write here the code for your  function block.

	return TRUE : // ENO value
}
#endif
```

Programming Hints

Introduction

The following coding provides an interface to the example described in this section. There are no restrictions for variable inputs passed by a value. Variables passed as their logical addresses may be accessed only by their physical addresses. You have to resolve the addressing differences before accessing variables.

Coding Sample

```
// Floating point constants declaration
const float in_Code cllbk_float0 = (float)0.0;
s_Declare_Logical (cllbk_float0);
//{{ SDKC_PROTOTYPE_EXAMPLE_EF1_REAL_BEGIN
Do not edit. Any modifications would be lost
#ifdef SDKC_COMPILING_EXAMPLE_EF1_REAL
IECBool fb_call_model EXAMPLE_EF1_REAL(
       const IECBool I_BO1,            // Boolean value
       const IECInt I_INT1,            // Integer value
       const IECReal I_REAL1,          // Real value
       IEC_PARAM_RTE_OFFSET_LG I_AR_W1, // Any array  of words
       IEC_PARAM_RTE_OFFSET IO_BO1,     // Boolean variable
that keeps its value
       IEC_PARAM_RTE_OFFSET IO_ANYNU,   // Any number
variable that keeps its value
       IEC_PARAM_RTE_OFFSET_LG O_STR1,  // This is  string output
       IEC_PARAM_RTE_OFFSET O_BO1       // This is a  boolean output
)
//}} SDKC_PROTOTYPE_EXAMPLE_EF1_REAL_END
{
       // TODO : Write here variables declarations.
       IECReal *pReal;
       // TODO : Write here the code for your  function block.
       if( I_BO1 == TRUE ) // Then we reset all the
outputs for example
{
          // Reset an boolean output
          *(IECBool *) s_log_to_phy( O_BO1 ) = FALSE;
          // Reset an boolean input/output
          *(IECBool *) s_log_to_phy( IO_BO1 ) = FALSE;
          // Reset an string
          // Caution! DON'T OVERWRITE the 'Length'
configuration parameter of a STRING type.
*(IECByte *)s_log_to_phy(O_STR1.IECPtr_Log ) = 0; // write '\0'
          // For floating point values we have to
```

```
use floating point constants to be defined outside
of the EF code block.
          pReal = (IECReal *)s_log_to_phy(IO_ANYNU);
          // Reset a floating point value
          *pReal = *((IECReal *)s_Const_Instance (cllbk_float0));
       }
       return TRUE ; // ENO value
}
#endif
```

Index

Symbols

*.dsc, *11*
  edit dsc files, *12*
*.h, *62*

A

Address
  Logical, *41*
  Physical, *41*
Addressing, *41*
addressing
  at the global level, *46*
ARM architecture, *39*
Array, *25*, *26*

B

best practices, *79*

C

coding rules, *38*
compiler options
  GNU C ARM ELF, *19*
  Microsoft Visual C 32bit, *19*
Concept EF/EFB migration procedure, *163*
Constants, *43*
  Global Level, *43*
  In procedures, *44*

D

data
  access to unaligned data, *39*
  unaligned data, *39*
data instance
  EFB, *47*
DDT, *53*
debug
  best practices, *79*
  test of boolean variables, *40*
detected error, *49*
detected warning, *49*
detected warnings, *76*
diagnostic management, *48*
directory, *32*
dsc
  edit dsc files, *12*

E

edit
  dsc files, *12*
EF/EFB, *53*
EF/EFB differences, *60*
EFB source code, *64*
EFB Toolkit installation, *14*
EFB toolkit services and user functions
  comparison between the different block types, *29*
EFBs, *11*
EFs, *11*
elementary function blocks, *11*
elementary functions, *11*
EN/ENO, *48*
event
  LanguageEntryPoint, *60*
  SystemEntryPoint, *60*
example, *53*

F

family
  description file, *11*
Family descriptor., *23*
fb_call_model, *58*
for the EFB Toolkit software, *15*
function
  characteristics, *12*

G

GNU C ARM ELF, *19*
graphical user interface
for the EFB Toolkit, *17*
GUI, *17*

H

Header file, *62*
help files for EFBs, *34*
help on type, *34*

I

instance, *11*
Instance, *54*
Instance Data, *65*
internal variables, *63*

K

Kind, *26*

L

language, *31*
multi language support, *31*
supported languages, *33*
LanguageEntryPoint, *60*, *63*

M

Microsoft Visual C 32bit, *19*
multi language support
directory structure, *32*

N

nested DDTs, *69*

O

operating system
EFB Toolkit requirements, *14*

P

PL7 EF migration, *154*
PL7 EF migration procedure, *155*
procedure
characteristics, *12*
public variables, *63*

R

recommendations, *80*
registration, *15*
requirements, *14*
return code, *48*

S

s_AliGetProgStat, *88*
s_cnt_100ms, *122*
s_cnt_10ms, *123*
s_cnt_1ms, *124*
s_Const_Instance, *59*
s_current_task, *129*
s_date_and_time, *125*
s_Declare_Logical, *58*
s_demask_it, *143*
s_diag_DeregisterError, *137*
s_diag_RegisterExtError, *135*
s_GetUSecs, *144*
s_Local_Instance, *44*
s_log_to_phy, *59*, *90*
s_log_to_phy(), *42*
s_mask_it, *145*
s_obj_nbr, *94*
s_obj_to_log, *92*
s_of_passw_check, *139*
s_of_passw_test, *141*
s_proc_indic, *146*
s_proc_type, *148*
s_rd_16bits, *95*
s_rd_1bit, *97*
s_rd_bit_attrib, *99*
s_rd_internalwords, *101*
s_rd_Nbits, *103*
s_rd_sysbit, *105*
s_rd_sysword, *106*

s_set_ffb_error, *131*
s_set_ffb_error_addi, *133*
s_syscnt_10ms, *128*
s_wr_16bits, *108*
s_wr_1bit, *110*
s_wr_bits_attrib, *112*
s_wr_internalwords, *114*
s_wr_Nbits, *116*
s_wr_sysbit, *118*
s_wr_sysword, *120*
Structure, *25*, *26*
SystemEntryPoint, *60*

T

test
    boolean variables, *40*

U

unaligned data, *39*
user functions, *13*

V

variables, *63*