SIEMENS

SIMATIC

STEP 7 (TIA Portal) Options Open Development Kit 1500S V2.5 SP4

Programming and Operating Manual

12/2023

A5E35253941-AH

Legal information

Warning notice system

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

DANGER

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

WARNING

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

CAUTION

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

NOTICE

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

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

Qualified Personnel

The product/system described in this documentation may be operated only by **personnel qualified** for the specific task in accordance with the relevant documentation, in particular its warning notices and safety instructions. Qualified personnel are those who, based on their training and experience, are capable of identifying risks and avoiding potential hazards when working with these products/systems.

Proper use of Siemens products

Note the following:

WARNING

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

Trademarks

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

Disclaimer of Liability

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

Siemens Aktiengesellschaft
Digital Industries
Postfach 48 48
90026 NÜRNBERG
GERMANY

A5E35253941-AH
Ⓟ 01/2024 Subject to change

Table of contents

Introduction

1

Purpose of the documentation

This documentation describes the specific characteristics of the Open Development Kit (ODK) V2.5 SP3.

Definitions and naming conventions

The following terms are used in this documentation:

- **CPU**: Designates the products named under "Scope of documentation".
- **ODK**: Open Development Kit
- **MFP**: Multifunctional platform
- **Windows**: Designates the Microsoft operating systems supported by ODK.
- **STEP 7**: For the designation of the configuring and programming software, we use "STEP 7" as a synonym for the version "STEP 7 (TIA Portal) V13 SP1 and higher".
- **DLL**: Dynamic Link Library
- **SO**: Shared Object
- **Visual Studio**: Microsoft Visual Studio
- **TCF**: Target Communication Framework

Basic knowledge required

This documentation is intended for engineers, programmers, and maintenance personnel with general knowledge of automation systems and programmable logic controllers.

To understand this documentation, you need to have general knowledge of automation engineering. You also need basic knowledge of the following topics:

- SIMATIC Industrial Automation System
- PC-based automation
- Using STEP 7
- Use of Microsoft Windows operating systems
- Programming with C/C++, C#, Visual Basic

Validity of the documentation

This documentation applies to use of ODK with the following products:

- CPU 1505SP (T/F/TF)
- CPU 1507S (F)
- CPU 1508S (F)
- CPU 1518-4 PN/DP MFP (F)

Notes

Please also observe notes labeled as follows:

Note

A note contains important information on the product described in the documentation, on the handling of the product or on the section of the documentation to which particular attention should be paid.

1.1 S7-1500/ET 200MP Documentation Guide

1.1.1 S7-1500 / ET 200MP Documentation Guide

The documentation for the SIMATIC S7-1500 automation system and the ET 200MP distributed I/O system is arranged into three areas.

This arrangement enables you to access the specific content you require. Changes and supplements to the manuals are documented in a Product Information.

You can download the documentation free of charge from the Internet (https://support.industry.siemens.com/cs/ww/en/view/109742691).

Basic information

The System Manual and Getting Started describe in detail the configuration, installation, wiring and commissioning of the SIMATIC S7-1500 and ET 200MP systems.

The STEP 7 online help supports you in the configuration and programming.

Examples:

- Getting Started S7-1500
- S7-1500/ET 200MP System Manual
- Online help TIA Portal

Device information

Equipment manuals contain a compact description of the module-specific information, such as properties, wiring diagrams, characteristics and technical specifications.

Examples:

- Equipment Manuals CPUs
- Equipment Manuals Interface Modules
- Equipment Manuals Digital Modules
- Equipment Manuals Analog Modules
- Equipment Manuals Communications Modules
- Equipment Manuals Technology Modules
- Equipment Manuals Power Supply Modules

General information

The function manuals contain detailed descriptions on general topics relating to the SIMATIC S7-1500 and ET 200MPsystems.

Examples:

- Function Manual Diagnostics
- Function Manual Communication
- Function Manual Motion Control
- Function Manual Web Server
- Function Manual Cycle and Response Times
- PROFINET Function Manual
- PROFIBUS Function Manual

Product Information

Changes and supplements to the manuals are documented in a Product Information. The Product Information takes precedence over the device and system manuals.

You can find the latest Product Information on the S7-1500 and ET 200MP systems on the Internet (https://support.industry.siemens.com/cs/de/en/view/68052815).

Manual Collection S7-1500/ET 200MP

The Manual Collection contains the complete documentation on the SIMATIC S7-1500 automation system and the ET 200MP distributed I/O system gathered together in one file.

You can find the Manual Collection on the Internet.
(https://support.industry.siemens.com/cs/ww/en/view/86140384)

SIMATIC S7-1500 comparison list for programming languages

The comparison list contains an overview of which instructions and functions you can use for which controller families.

You can find the comparison list on the Internet (https://support.industry.siemens.com/cs/ww/en/view/86630375).

1.1.2 SIMATIC Technical Documentation

Additional SIMATIC documents will complete your information. You can find these documents and their use at the following links and QR codes.

The Industry Online Support gives you the option to get information on all topics. Application examples support you in solving your automation tasks.

Overview of the SIMATIC Technical Documentation

Here you will find an overview of the SIMATIC documentation available in Siemens Industry Online Support:

Industry Online Support International (https://support.industry.siemens.com/cs/ww/en/view/109742705)

Watch this short video to find out where you can find the overview directly in Siemens Industry Online Support and how to use Siemens Industry Online Support on your mobile device:

Quick introduction to the technical documentation of automation products per video (https://support.industry.siemens.com/cs/us/en/view/109780491)

YouTube video: Siemens Automation Products - Technical Documentation at a Glance (https://youtu.be/TwLSxxRQQsA)

Retention of the documentation

Retain the documentation for later use.

For documentation provided in digital form:

1. Download the associated documentation after receiving your product and before initial installation/commissioning. Use the following download options:
   - Industry Online Support International: (https://support.industry.siemens.com)

     The article number is used to assign the documentation to the product. The article number is specified on the product and on the packaging label. Products with new, non-compatible functions are provided with a new article number and documentation.
   - ID link:

     Your product may have an ID link. The ID link is a QR code with a frame and a black frame corner at the bottom right. The ID link takes you to the digital nameplate of your product. Scan the QR code on the product or on the packaging label with a smartphone camera, barcode scanner, or reader app. Call up the ID link.
2. Retain this version of the documentation.

Updating the documentation

The documentation of the product is updated in digital form. In particular in the case of function extensions, the new performance features are provided in an updated version.

1. Download the current version as described above via the Industry Online Support or the ID link.
2. Also retain this version of the documentation.

mySupport

With "mySupport" you can get the most out of your Industry Online Support.

Registration	You must register once to use the full functionality of "mySupport". After registration, you can create filters, favorites and tabs in your personal workspace.
Support requests	Your data is already filled out in support requests, and you can get an overview of your current requests at any time.
Documentation	In the Documentation area you can build your personal library.
Favorites	You can use the "Add to mySupport favorites" to flag especially interesting or frequently needed content. Under "Favorites", you will find a list of your flagged entries.
Recently viewed articles	The most recently viewed pages in mySupport are available under "Recently viewed articles".
CAx data	The CAx data area gives you access to the latest product data for your CAx or CAe system. You configure your own download package with a few clicks: • Product images, 2D dimension drawings, 3D models, internal circuit diagrams, EPLAN macro files • Manuals, characteristics, operating manuals, certificates • Product master data

You can find "mySupport" on the Internet. (https://support.industry.siemens.com/My/ww/en)

Application examples

The application examples support you with various tools and examples for solving your automation tasks. Solutions are shown in interplay with multiple components in the system - separated from the focus on individual products.

You can find the application examples on the Internet.
(https://support.industry.siemens.com/cs/ww/en/ps/ae)

1.1.3 Tool support

The tools described below support you in all steps: from planning, over commissioning, all the way to analysis of your system.

TIA Selection Tool

The TIA Selection Tool tool supports you in the selection, configuration, and ordering of devices for Totally Integrated Automation (TIA).

As successor of the SIMATIC Selection Tools , the TIA Selection Tool assembles the already known configurators for automation technology into a single tool.

With the TIA Selection Tool , you can generate a complete order list from your product selection or product configuration.

You can find the TIA Selection Tool on the Internet.
(https://support.industry.siemens.com/cs/ww/en/view/109767888)

SINETPLAN

SINETPLAN, the Siemens Network Planner, supports you in planning automation systems and networks based on PROFINET. The tool facilitates professional and predictive dimensioning of your PROFINET installation as early as in the planning stage. In addition, SINETPLAN supports you during network optimization and helps you to exploit network resources optimally and to plan reserves. This helps to prevent problems in commissioning or failures during productive operation even in advance of a planned operation. This increases the availability of the production plant and helps improve operational safety.

The advantages at a glance

- Network optimization thanks to port-specific calculation of the network load
- Increased production availability thanks to online scan and verification of existing systems
- Transparency before commissioning through importing and simulation of existing STEP 7 projects
- Efficiency through securing existing investments in the long term and the optimal use of resources

You can find SINETPLAN on the Internet
(https://new.siemens.com/global/en/products/automation/industrial-communication/profinet/sinetplan.html).

See also

PRONETA Professional (https://support.industry.siemens.com/cs/ww/en/view/109781283)

Security information

2

Notes on Open Source Software

You can download and install various development tools of the Open Source project MinGW32 as a supplement to the product. Note that these components are optional and are provided and distributed by the MinGW32 project or other licensors under different licenses, which you can view on the project homepage or on the specific packages.

2.1 Cybersecurity information

Siemens provides products and solutions with industrial cybersecurity functions that support the secure operation of plants, systems, machines, and networks.

In order to protect plants, systems, machines, and networks against cyber threats, it is necessary to implement – and continuously maintain – a holistic, state-of-the-art industrial cybersecurity concept. Siemens' products and solutions constitute one element of such a concept.

Customers are responsible for preventing unauthorized access to their plants, systems, machines and networks. Such systems, machines and components should only be connected to an enterprise network or the internet if and to the extent such a connection is necessary and only when appropriate security measures (e.g. firewalls and/or network segmentation) are in place.

For more information on protective industrial cybersecurity measures for implementation, please visit (https://www.siemens.com/global/en/products/automation/topic-areas/industrial-cybersecurity.html).

Siemens' products and solutions undergo continuous development to make them more secure. Siemens strongly recommends that product updates are applied as soon as they are available and that the latest product versions are used. Use of product versions that are no longer supported, and failure to apply the latest updates may increase customers' exposure to cyber threats.

To stay informed about product updates at all times, subscribe to the Siemens Industrial Cybersecurity RSS Feed under (https://new.siemens.com/global/en/products/services/cert.html).

2.2 Information about third-party software updates

This product contains third-party software. Siemens accepts liability with respect to updates/patches for the third-party software only when these are distributed by Siemens in the context of a Software Update Service contract or officially approved by Siemens. Otherwise, updates/patches are installed at the user's own risk. You can find more information in our Software Update Service (http://w3.siemens.com/mcms/automation-software/en/software-update-service/Pages/Default.aspx).

2.3 Notes on protecting administrator accounts

A user with administrator rights has extensive access and manipulation possibilities.

Therefore, make sure that the administrator account is adequately protected to prevent unauthorized changes. To do this, set secure passwords and use a standard user account for regular operation. Other measures, such as the use of security policies, should be applied as required.

Product overview

3

3.1 Introduction to ODK 1500S

Overview

ODK is a development kit that allows you to program custom functions and generate files that STEP 7 can call directly.

ODK serves as a tool for:

- Windows environment
  - Execution on your Windows PC
  - Use of resources of your Windows PC
  - Use of operating system functions and system resources with access to external hardware and software components
- Realtime environment
  - Execution on your CPU
  - Synchronous function call (algorithmic, controllers)

Calling multiple applications under Windows or in the realtime environment is possible.

You must use the CPU function libraries in the STEP 7 program.

You can use C/C++ runtime applications running in SIMATIC S7-1500 MFP C/C++ Runtime independently of the STEP 7 program.

With the PLCSIM Advanced function libraries, you can run a project in a simulated environment instead of on a hardware or software CPU.

Structure and design of an CPU function library

ODK supports the interface for calling custom high-level language programs from the controller program of the CPU.

ODK supports the following templates:

- Templates in different programming languages for Microsoft Visual Studio. This allows you to generate a DLL file. The C++, C# and Visual Basic programming languages are supported.
- A template for programming in Eclipse. This allows you to generate an SO file. ODK also supplies a class library for Eclipse. The C++ programming language is supported.

You can create a CPU function library for both the Windows and the real-time environment. The programming languages mentioned are available to you for this purpose.

You can run the ODK program in the following ways:

- Synchronous, i.e. executed as part of the CPU cycle (executed in the realtime environment).
- Asynchronous, i.e. started by the CPU program and ended in the background (executed in the Windows environment).

You can run CPU function libraries both under Windows (DLL) as well as in the real-time core of the CPU (SO). You call the functions of the DLL or SO file using instructions in the user program.

The CPU can perform functions in libraries that can be loaded dynamically. There are several functions possible in a CPU function library. Specific function blocks are available for a CPU function library:

- Loading and unloading of the CPU function library
- In each case, a specific function block for calling a function.

The following illustration provides a schematic overview of how CPU function libraries work and run on a PC. The illustration applies to the S7-1500 Software Controller.

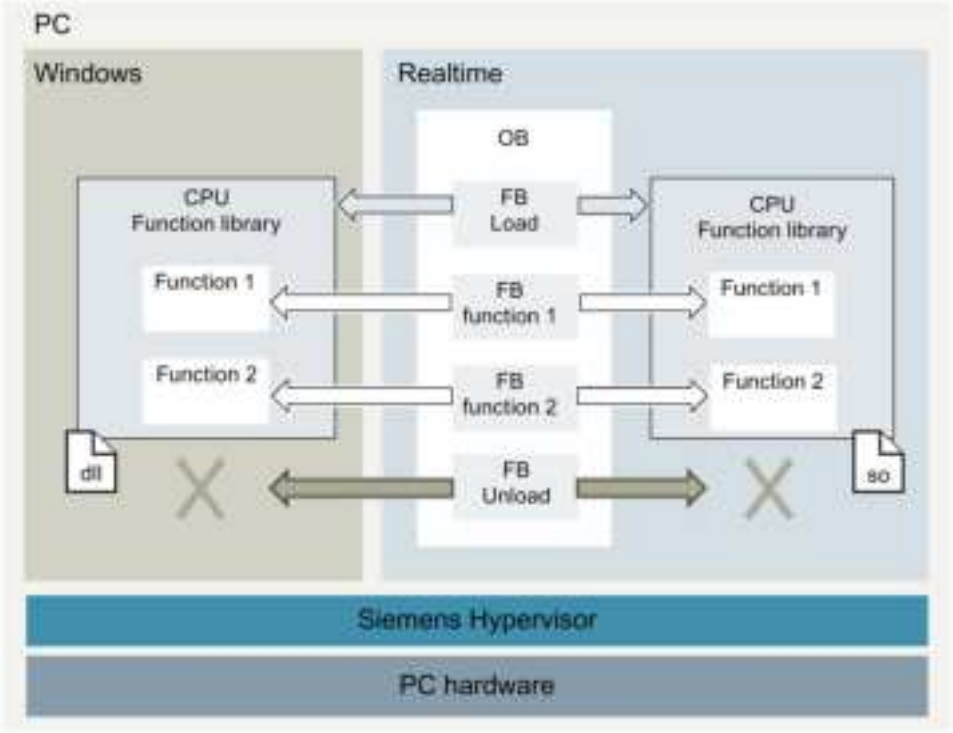


Figure 3-1 Running a CPU function library on a PC

Structure and design of a C/C++ runtime application

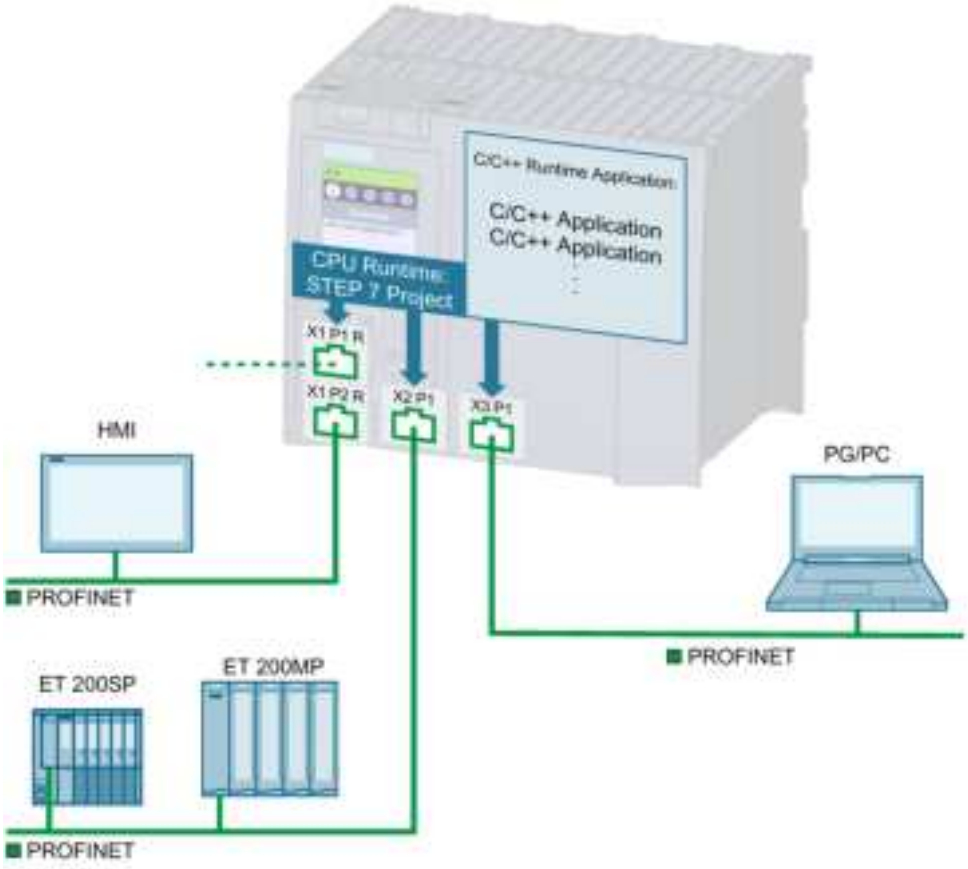


Figure 3-2 Overview of the performance segment

You can use C/C++ runtime applications to implement parallel processes to the STEP 7 user program, for example, for pre-processing or transmitting data via Industrial Ethernet. A CPU can perform several tasks at the same time. The complexity of functions is reduced and the time required for implementation is reduced.

You can reuse existing C/C++ algorithms. In order to continue using existing technological know-how, you can integrate the existing C/C++ code via the Open Development Kit as C/C++ runtime applications in the SIMATIC S7-1500 MFP C/C++ Runtime.

Once you integrate the C/C++ sources, you can execute them on the CPU.

The following options are available for communication between CPU Runtime and C/C++ Runtime:

- On Open User Communication with the "TSTEND" and "TRCV" function blocks.
- About the Communication protocol OPC UA.

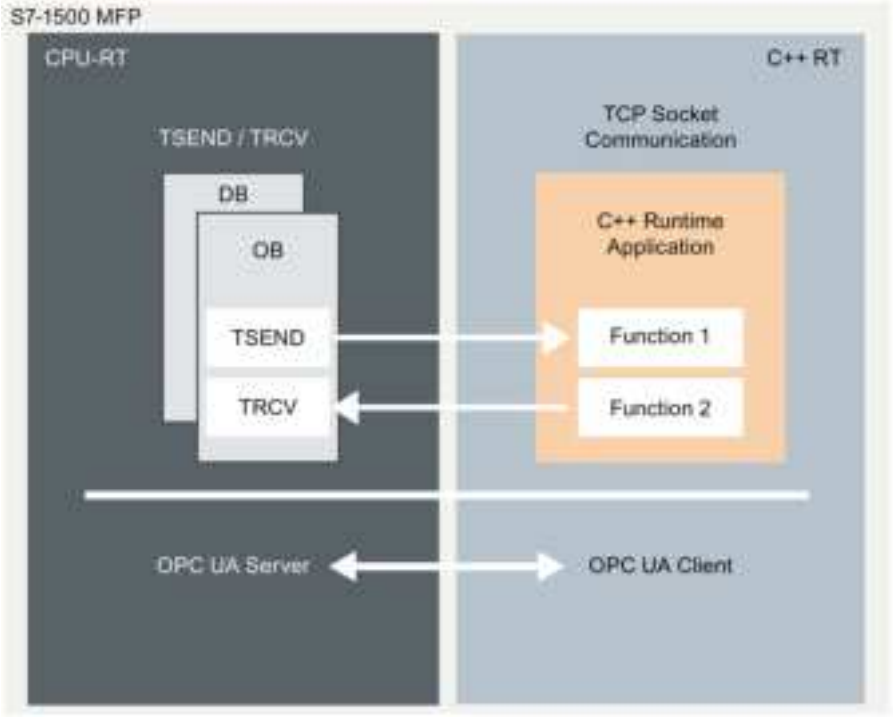


Figure 3-3 Communication between CPU Runtime and C/C++ Runtime

3.2 Development environments

The following development environments for creating an ODK project are available for selection.

- Microsoft Visual Studio for CPU function libraries for the Windows environment (DLL file) and PLCSIM Advanced function libraries.
- Eclipse CPU function libraries for the realtime environment (SO file) and C/C++ runtime applications.

Microsoft Visual Studio as a development environment

Use Microsoft Visual Studio. To help you develop a CPU function library, a template for a Microsoft Visual Studio project is included in the installation of ODK 1500S. The ODK template can be found under the entry of the corresponding programming language when a new project is created.

Eclipse as a development environment

Use Eclipse. To help you develop a C/C++ runtime application, a template for an Eclipse project is included in the installation of ODK 1500S. The template can be found in the folder "ODK 1500S Templates".

3.3 Basic procedure

The following sections describe the development tasks and procedures for the development and execution of a CPU function library/C/C++ runtime application:

- Developing a CPU function library for the Windows environment (Page 28)
- Developing a CPU function library for the realtime environment (Page 65)
- Development of a C/C++ runtime application (Page 105)
- Developing a PLCSIM Advanced function library (Page 120)

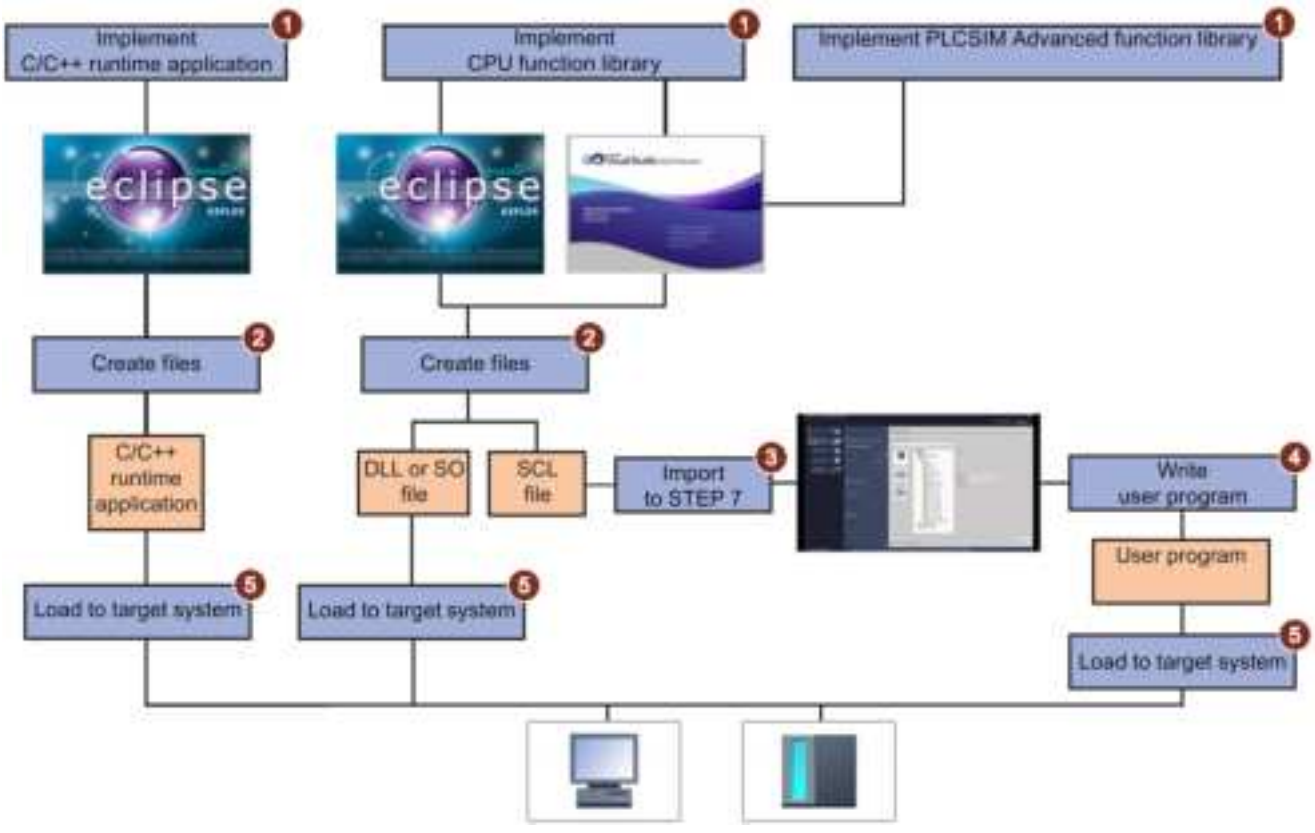


Figure 3-4 Overview of the development steps

Overview of the development steps

To develop and execute a C/C++ runtime application/CPU function library, follow these steps:

1. Implement your function.
   - Implement your function for CPU function libraries in Visual Studio (DLL file) or Eclipse (SO file).
   - Implement your function for C/C++ runtime application in Eclipse.
2. Create the C/C++ runtime application, DLL or SO file and the SCL file.
3. Import the SCL file into STEP 7.
4. Write your user application in STEP 7.
5. Load the user program in the CPU and the C/C++ runtime application or DLL or SO file into the target system.

Result

Your C/C++ runtime application/CPU function library is loaded in the target system.

The CPU function library is loaded and executed by the user program in STEP 7.

Installation

4

4.1 System Requirements

Requirements

Your PC must meet the following system requirements in order to use the ODK:

Category	Requirements
Operating system	• Microsoft Windows 8.1, 64-bit • Microsoft Windows 10, 64-bit • Microsoft Windows 11, 64-bit **Note:** Installation on Windows 8.1 Before installing ODK 1500S with the Windows 8.1 operating system, ensure that the Windows Knowledge Base article "KB2919355" is installed on the PC.
Processor and memory	PC system: • At least systems with Intel Core i5 processor • 1.2 GHz or higher • At least 4 GB of RAM
Mass storage	Depending on the already installed components, you need up to 3 GB of free space on the hard disk C:\. The exact amount of space required is displayed during the installation. **Note**: The setup files are deleted when the installation is complete.
Operator interface	Color monitor, keyboard and mouse or another pointing device (optional) supported by Microsoft Windows
SIMATIC software	• SIMATIC STEP 7 Professional (TIA Portal) V15 or higher
Additional software	Not included in the product package: • Java Runtime 32-bit as of V1.7 (for Eclipse) • Microsoft Visual Studio 2015 • Microsoft Visual Studio 2017 • Microsoft Visual Studio Community 2017 • Microsoft Visual Studio 2019 • Microsoft Visual Studio 2022 • Eclipse plugins (for MFP use) • SSH Client, for example PuTTY (for MFP use) • Microsoft Development Tool: Download Center (https://www.microsoft.com/en-us/download/developer-tools.aspx)

Note

.NET version for the Windows environment

When creating a C# project, the target framework is set as ".NET 4.8" (e.g. "C# Console Application" under Visual Studio 2019/2022).

If you need a new .NET framework, ensure that a .NET version ≥ the target framework version set on the target device is installed.

ODK 1500S V2.5 SP4 is compatible with the following devices (support for loadable function libraries depends on the device):

	CPU function library DLL (Windows)	CPU function library SO (Real-time)	C/C++ runtime application
CPU 1505SP (F/T/TF) as of V2.5	Yes	Yes	No
CPU 1507S (F) as of V2.5			
CPU 1508S (F) as of V2.6			
CPU 1518-4 PN/DP MFP (F) up to Firmware V2.6.1 CPU 1518-4 PN/DP MFP (F) Firmware V2.8 CPU 1518-4 PN/DP MFP (F) Firmware V2.9	No	Yes	Yes
PLCSIM Advanced as of V3.0	no*	Yes	No

* PLCSIM Advanced has its own function library type "PLCSIM Advanced Function Library DLL (Windows)"

4.2 Installing ODK

To install the ODK, insert the Installation DVD. Follow the instructions of the setup program.

If the setup program does not start automatically, open the "Start.exe" file on the Installation DVD manually with a double-click.

Requirements

You need administrator rights for this procedure.

It is possible to operate different ODK major versions (e.g. V2.0 and V2.5) on one PC at the same time.

It is not possible to operate the major version and service pack (e.g. V2.5 and V2.5.4) on one PC at the same time.

Note

Close applications before a repair installation/uninstall

Close all applications (especially ODK-related applications), before performing the repair installation/uninstall.

Note

Use of antivirus programs

To avoid problems during installation, disable the antivirus program during installation or close the directory "C:\Program Files\Common Files\Siemens\Automation\Siemens Installer Assistant" as well as the directory in which the Start.exe of the antivirus program is located.

Procedure

If you want to use the Microsoft Visual Studio development environment, we recommend that you install this before ODK.

To install ODK, follow these steps:

1. Start the "Start.exe" file from the Installation DVD manually with a double-click.
2. Select the language for performing the installation.
3. Confirm with "Next".

   If you want to upgrade an installed version V2.5, V2.5.1, V2.5.2 or V2.5.3 to V2.5.4, confirm with "Change".
4. Click "Next" to confirm the list of components that are to be installed.

   The check mark for Automation License Manager (ALM) cannot be removed.
5. Follow the instructions of the installation wizard.

6. Confirm the installation dialog with the "Install" button.
7. Choose whether you want to carry out the licensing (Page 25) during the installation or at a later time.

Result

The installation is complete. All product languages are installed by default during the installation process. The installation creates a shortcut in the Start menu of Windows.

Note

Directory and workspace path for V2.5 SP4

The name of the directory and the path for the V2.5 workspace are retained after the update to V2.5 SP4.

Example:

C:\Program Files (x86)\Siemens\Automation\ODK1500S\V2.5

C:\Program Data\Siemens\Automation\ODK1500S\V2.5

The setup program installs the following components:

- "Eclipse" development environment for the development of a CPU function library for the realtime environment or a C/C++ runtime application
- Project templates for Eclipse
  - C++ Project for CPU function library (CPU Runtime)
  - C++ Project for MFP Linux application (CPU 1518 MFP - up to FW v2.6.1)
  - C++ Project for MFP Linux application (CPU 1518 MFP FW v2.8)
  - C++ Project for MFP Linux application (CPU 1518 MFP FW v2.9 or higher)
- Project templates for Visual Studio
  - For the Windows CPU function library
  - For the PLCSIM function library
- Tool to integrate Visual Studio templates
- Installation script for MinGW32
- Code generator
- Online help
- HelpStarter tool
- Automation License Manager, if this is out of date or was not yet installed
- Certificate of license (Certificate of License)

Note

Eclipse workspace folder

If you have installed ODK and the Software Controller on the same IPC, move the Eclipse Workspace from the path "%ProgramData%\Siemens\Automation\ODK1500S\V2.5" to, for example, "C:\ODK1500S_V2.5_Workspace".

4.3 Licensing ODK 1500S

To create CPU function libraries, the software requires a product-specific license key that you install with the Automation License Manager. Each SIMATIC software product for automation that is subject to license (e.g., STEP 7) has its own license key. You must install the license key for each product.

Working with the Automation License Manager

The Automation License Manager is a product of Siemens AG and is used for managing license keys. The Automation License Manager is supplied on the installation data medium of this product by default and is transferred automatically during the installation process.

Software products that require license keys for operation register the requirement for license keys automatically in the Automation License Manager. If the Automation License Manager finds a valid license key for this software, the software can be used according to the conditions of use associated with this license key.

Certificate of license

A Certificate of License is included in the scope of delivery. It contains your unique license number. The license certificate serves as proof that you have a valid license key. Store this certificate in a safe place.

Note

Obtaining a replacement license key

You must have a valid certificate of license to get a replacement license key.

Recovering the license key in case of defective mass storage

If a error has occurred on the mass storage or USB flash drive containing your license key file, contact your Siemens representative (https://support.industry.siemens.com/cs/ww/en/). Make sure you have your certificate of license available for this.

License key

The license key for ODK 1500S is located on a USB flash drive that is included in the scope of delivery.

If the USB flash drive containing the license key is lost or damaged, you can contact Support (https://support.industry.siemens.com/cs/ww/en/) to obtain a new license key. You need the certificate of license to receive a replacement license key from Siemens.

Handling of license key for download version of ODK 1500S

The download of ODK 1500S allows you to access ordered license keys.

For access, you need:

- A personalized login that you can use to fetch all license keys assigned to "your company".
- An anonymous login that you can use to fetch an individual license key, and the corresponding license certificate. This document contains all data required for the anonymous download.

Additional information on the license key and the download is available in the Automation License Manager manual (https://support.industry.siemens.com/cs/ww/en/view/102770153).

Transferring the license key

The license key can be transferred during the installation or afterwards.

If the USB flash drive with the relevant license key is inserted in the USB port of the PC at the start of installation, the license key will be transferred automatically during the installation. If the USB flash drive is not inserted at the start of installation, you have three options for installing the license key subsequently:

- To transfer the license key **manually** from a network computer or other storage medium, select the "Manual license transfer" button.
- Insert the USB flash drive with license key, and select the "Retry license transfer" button. The Automation License Manager opens in order to transfer the license key.
- If you do not want to install a license key, select the "Skip license transfer" button.

Note

Working without license key

For legal reasons, a valid license key is required for this product.

If no valid license key is present on your PC, you cannot generate any projects. An error message will inform you at regular intervals that no valid license key is present.

Manually transferring the license key subsequently

A message is displayed if you generate a project for a CPU function library without transferred license key.

To **manually** transfer the license key for ODK subsequently, follow these steps:

1. Start the installation of ODK 1500S with administrator rights.
2. In the "License Transfer" section, select the "Manual license transfer" button.

   A dialog box for synchronization of the license opens.
3. Select the destination and the source of the license key.
4. To transfer the license key, click the "Synchronize" button.

   The license key is transferred.

4.4 Subsequently integrating project template for Windows CPU function libraries in Visual Studio

When Visual Studio is already installed, the project template for Windows CPU function libraries is automatically installed during the ODK installation. If Visual Studio is installed later, you have the following options to integrate the project template for Windows CPU function libraries:

- Perform a repair installation of ODK.
- Run the integration manually. Call your ODK installation file "ODK_VSTemplate_Integration.exe" in the "bin" folder.

Result

The project templates for Windows CPU function libraries is installed for Visual Studio. You can find this under the corresponding programming language.

4.5 Uninstalling ODK

Procedure

To remove ODK from your PC, follow these steps:

1. Close all running programs, especially ODK-related applications.
2. Select the menu "Control Panel > Programs and Features", select the entry "SIMATIC ODK 1500S" and click "Uninstall".
3. Select the "Uninstall" command in the shortcut menu.

   A dialog box for uninstalling appears.
4. Follow the steps for uninstalling.

Result

ODK is removed.

Developing a CPU function library for the Windows environment

5

5.1 Creating a CPU function library

5.1.1 Requirements

The Microsoft Visual Studio development environment is not included in the scope of delivery of ODK.

You can find the Download Center for Microsoft development tools in the Internet (https://www.microsoft.com/en-us/download/developer-tools.aspx).

5.1.2 Creating a project

To help you develop a CPU function library, a project template for CPU function libraries for a project in Visual Studio is included in the installation of ODK 1500S. The template supports 32-bit and 64-bit applications.

Procedure

To create a project in Microsoft Visual Studio using the project template, follow these steps:

1. Open Microsoft Visual Studio as a development environment.
2. In the "File > New" menu, select the command "Project..."

   The "New Project" dialog opens.

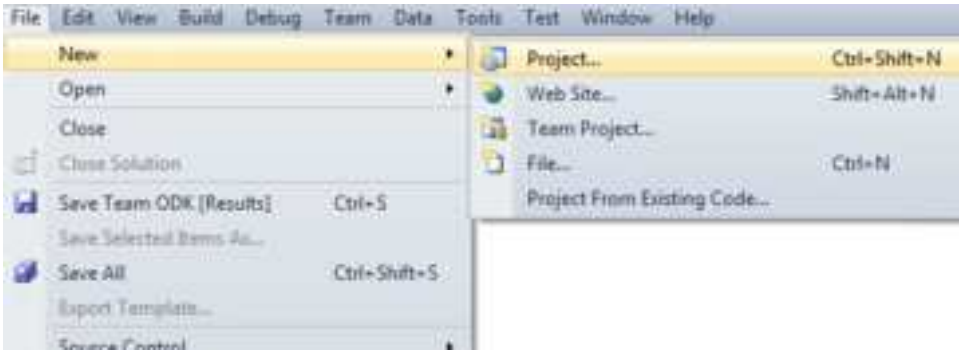


Figure 5-1 Creating a new project in Visual Studio

3. Select your preferred programming language and the corresponding project template (C++, C# or VB).
4. Enter a project name.
5. Click "OK" to confirm.

Result

The CPU function library is created using the project template and sets the following project settings:

- Project settings for generating the DLL file
- Automates the generation of the DLL and SCL file

The project template set ups various structures depending on the programming language:

- C++ project (Page 29)
- C# project (Page 32)
- VB Project (Page 33)

5.1.2.1 Solution Explorer structure: C++ project

Folder / file			Description
<project>			
	Definition File		
		<project>.odk	ODK interface description
		<project>.scl.additional	S7 blocks that are appended to the <project>.scl file. Although the file is not part of the project template, the code generator processes the file.
	Generated Files		**Files from this folder may not be edited!**
		ODK_Types.h	Definition of the ODK base types
		ODK_Functions.h	Function prototypes
		ODK_Execution.cpp	Implementation of the "Execute" method
	Header Files		Header file
	ODK Helpers		**Files from this folder may not be edited!**
		ODK_CpuReadData.h	Definition: Help functions for reading the data blocks
		ODK_CpuReadData.cpp	Implementation: Help functions for reading the data blocks
		ODK_CpuReadWriteData.h	Definition: Help functions for reading/writing the data blocks
		ODK_CpuReadWriteData.cpp	Implementation: Help functions for reading/writing the data blocks
		ODK_StringHelper.h	Definition: Help functions S7 strings / W strings
		ODK_StringHelper.cpp	Implementation: Help functions S7 strings / W strings
	Resource Files		
		<project>.rc	
	Source Files		Source Files
		<project>.cpp	Function code
		dllmain.cpp	Implementation of the "dllmain" file
	STEP7		**Files from this folder may not be edited!**
		<project>.scl	S7 blocks

The C++ Native project template supports the following applications:

Configuration and platform	Visual Studio Version older than 2015	Visual Studio 2015 and later
Debug Win32	Yes	Yes
Release Win32	Yes	Yes
Debug x64	To be created manually	Yes
Release x64	To be created manually	Yes

Note

Configuration of C/C++ Redistributables

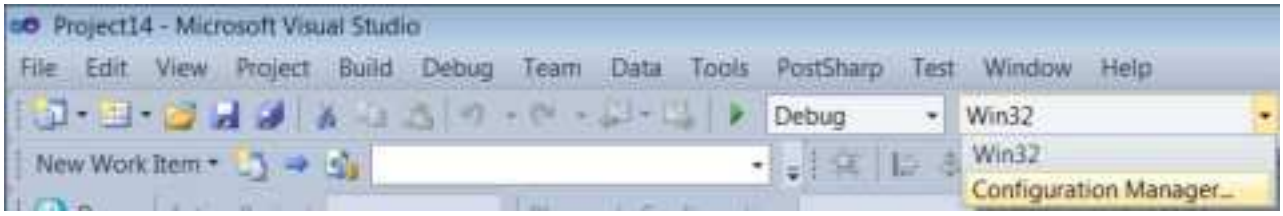
Since the software controller contains the C/C++ redistributables for the release configuration, build the CPU function library with the configuration "Release".

To use the "Debug" configuration, add the redistributables for the debug configuration on the target system.

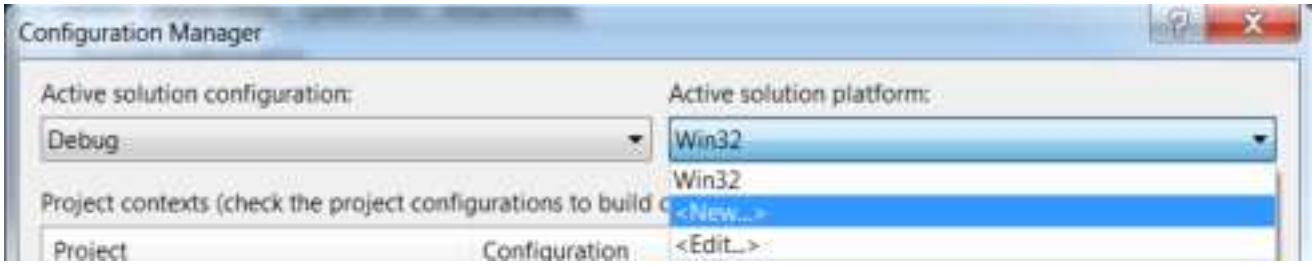
Creating a CPU function library for x64 platform with Visual Studio version older than 2015

To create a project template for an x64 platform with a Visual Studio version older than 2015, proceed as follows:

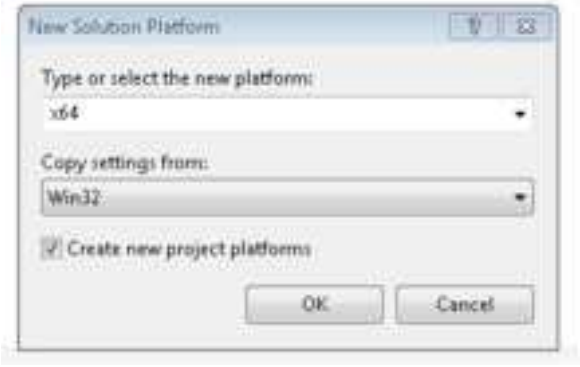
1. Open the "Configuration Manager".



2. Create an x64 platform.

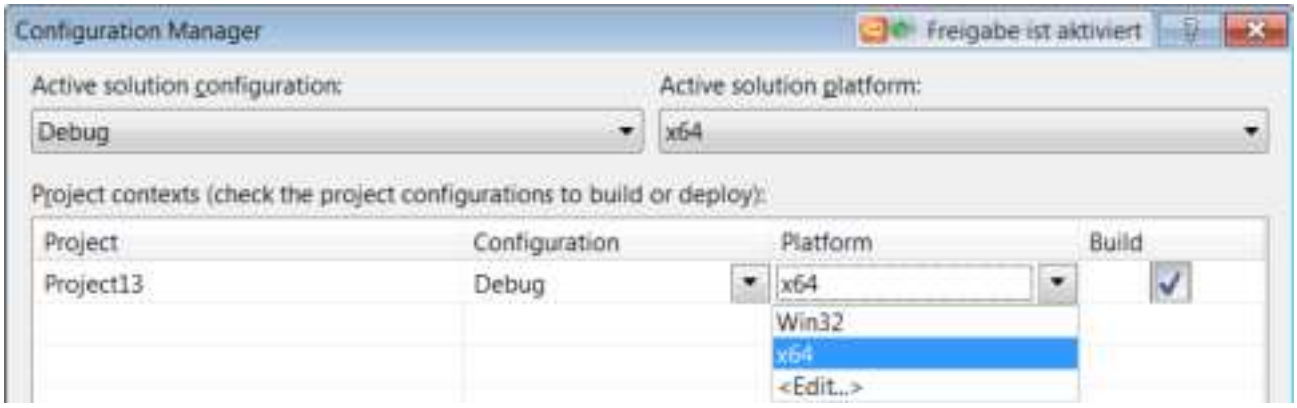


The "New Solution Platform" dialog opens.



Select "Win32" from the drop-down list box "Copy settings from:" .

3. Define a solution configuration for an x64 platform.



4. Select "Debug" or "Release" from the drop-down list box "Active solution configuration" and "x64" from the drop-down list box "Platform".

5.1.2.2 Solution Explorer structure: C# project

Directory / file			Description
<project>			
	Properties		
		AssemblyInfo.cs	
	Definition File		
		<project>.odk	ODK interface description
		<project>.scl.additional	S7 blocks that are appended to the <project>.scl file. The file is not part of the project template, but the code generator processes the file.
	Generated Files		**Files from this folder may not be edited!**
		OdkTypes.cs	Definition of the ODK base types
		OdkFunctions.cs	Function prototypes
		OdkExecution.cs	Implementation of the "Execute" method
	ODK Helpers		**Files from this folder may not be edited!**
		OdkReadVariant.cs	Help functions for reading the data blocks
		OdkReadWriteVariant.cs	Help functions for reading/writing the data blocks
	Source		Source Files
		<project>.cs	Function code
	STEP7		**Files from this folder may not be edited!**
		<project>.scl	S7 blocks

The C++ project template supports the following applications:

Configuration and platform	Visual Studio Version older than 2015	Visual Studio 2015 and later
Debug each CPU	Not supported	Yes
Release each CPU	Not supported	Yes

5.1.2.3 Solution Explorer structure: VB Project

Directory / file			Description
<project path>			
	My Project		
		AssemblyInfo.vb	
	Definition File		
		<project>.odk	ODK interface description
		<project>.scl.additional	S7 blocks that are appended to the <project>.scl file. The file is not part of the project template, but the code generator processes the file.
	Generated Files		**Files from this folder may not be edited!**
		OdkTypes.vb	Definition of the ODK base types
		OdkFunctions.vb	Function prototypes
		OdkExecution.vb	Implementation of the "Execute" method
	ODK Helpers		**Files from this folder may not be edited!**
		OdkReadVariant.vb	Help functions for reading the data blocks
		OdkReadWriteVariant.vb	Help functions for reading/writing the data blocks
	Source		Source Files
		<project>.vb	Function code
	STEP7		**Files from this folder may not be edited!**
		<project>.scl	S7 blocks
		<project>.vb	Function code

The VB project template supports the following applications:

Configuration and platform	Visual Studio Version older than 2015	Visual Studio 2015 and later
Debug each CPU	Not supported	Yes
Release each CPU	Not supported	Yes

5.1.3 Generating a CPU function library

The generation of the project data is divided into two automated steps.

- **Pre-Build**: Generation of the files created by default based on the changed <project>.odk file and generation of the SCL file.
- **Actual-Build**: Generation of the DLL file.

Procedure

To generate the project data, follow these steps:

1. Save all edited files.
2. In the "Build" menu, select the command "Build Solution".

Note

C/C++ projects

Perform the build of the CPU function library in the "Release" configuration, as the software controller has already installed the C/C++ Redistributables (Release Runtime files).

To use the "Debug" configuration, copy the Debug Runtime files to the software controller.

Note

The project data is only generated if the files have been changed.

Result

The generation of the project data is started. The automatically generated files are stored in the file system.

- DLL file: Project directory\<project>\<BuildConfiguration>\<project>.dll
- SCL file: Project directory\<project>\STEP7\<project>.scl

5.1.4 Defining the runtime properties of a CPU function library

The next step is to define the interface description of the CPU function library in the <project>.odk file. The file contains the following elements:

- Comments
- Parameters
- Definitions of functions and structures

Procedure

To define the interface description in the <project>.odk file, follow these steps:

1. Open the <project>.odk file.
2. Change the elements depending on your requirements.

Description of the elements

Comments

You can use comments for explanation purposes.

Parameters

The definition of the parameters must be within a line of code.
`<parameter name>=<value> // optional comment`

The interfaces file supports the following parameters:

Parameter	Value	Description
Context	user	Specifies that the CPU function library is loaded in the context of a Windows user (Page 35).
	system	Specifies that the CPU function library is loaded in the context of the Windows system (Page 35).
STEP7Prefix	<String>	Describes the string that precedes your functions and is shown after importing the SCL file in STEP 7. The following characters are allowed: {A...Z, a...z, 1...9, -, _} Umlauts are not permitted. The project name is entered without spaces by default.
FullClassName	<String>	The parameter is required for the C# and VB programming languages. To change the class names or namespace of the source files of the CPU function library, you need to adjust the "FullClassName" parameter.

Note

Spaces in the project name

With the STEP7 prefix, invalid characters are replaced by an underscore.

5.1.5 Environment for loading or executing the CPU function library

When the SCL file is imported into STEP 7 as an external source, the ODK instructions are created in the selected directory in STEP 7. The ODK instructions enable you to control your CPU function library regardless of the STEP 7 user program after programming and the initial loading. You can load up to 32 CPU function libraries.

Depending on whether you have created the CPU function library for a 32-bit, 64-bit system or with the "Any CPU" option, this is loaded into a 32-bit or 64-bit ODK host process.

You can choose one of two contexts for your CPU function library:

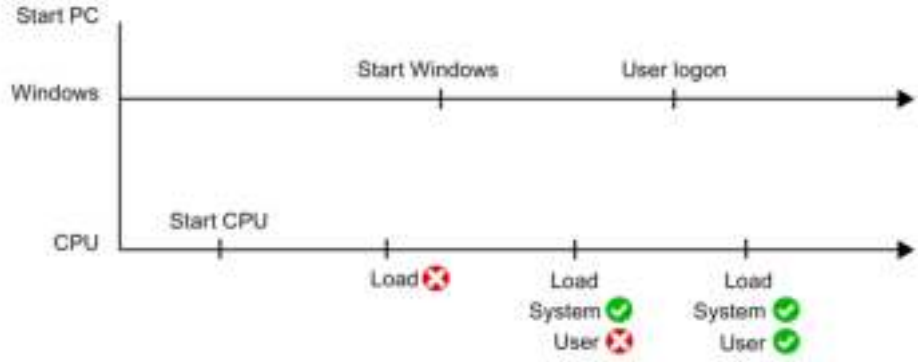
- **"System" context**

  Windows is started, a user can be logged on

- **"User" context**

  Windows is started, a user must be logged on

The following graphic shows you when a CPU function library may be loaded depending on the context.



"System" context

Change the following line of code in your <project>.odk file to use the CPU function library in the system context (Session 0):

```
Context=system
```

In the system context, the CPU function library is running without the logon of a Windows user. This means the CPU function library cannot be actively controlled with user interface elements such as message dialogs.

"User" context

Change the following line of code in the <project>.odk file to use the CPU function library in the user context:

```
Context=user
```

When you load the CPU function library in the user context, it automatically unloads as soon as the user logs off in Windows. The CPU function library can be actively controlled by Windows user interface elements such as message dialogs and provides access to additional resources of the Windows environment.

If multiple users are logged on to Windows, the CPU function library loads or unloads for the user, who has the current screen rights until he logs off in Windows.

5.1.6 Defining functions and structures of a CPU function library

Functions

Functions are defined by the following general lines of code:

```
ODK_RESULT <FunctionName>
([<InOut identifier>] <data type> <tag name>, etc.);
```

The <project>.odk file is the ODK interface description for CPU function libraries. This is available for all supported programming languages.

The <project>.odk file contains an example function description by default. You can change this description and/or add more function descriptions.

```
ODK_RESULT MyFunc1([IN] INT param1, [OUT] INT param2);
```

Syntax rules for functions

The following syntax rules apply to functions within the <project>.odk file:

- Note that the function names are case-sensitive.
- You can split function definitions into several lines.
- End a function definition with a semicolon.
- TAB and SPACE are allowed.
- Do not define a tag name in a function twice.
- Do not use any keywords for the programming language that is used (for example "EN / ENO" as parameter name)
- Use ODK_RESULT only for the return values of the function.
- The tag name must start with a letter or an underscore.
- Illegal function names are displayed during generation in the development environment.
- The following names are not allowed in combination of *<STEP 7Prefix>* and <function name>: ODK_Load, ODK_Unld, ODK_ExcA, ODK_ExcS

<FunctionName>

Function names are valid with the syntax and character restrictions of the used programming language.

<InOut-Identifier>

There are three defined InOut-Identifiers. Use these in the following order: [IN], [OUT], [INOUT]

- [IN]: Specifies an input tag. The tag is copied to the function when it is called. This is constant and cannot be changed.
- [OUT]: Specifies an output tag. The tag is copied back after the function has been completed.
- [INOUT]: Specifies an input and output tag. The tag is copied to the function when it is called. This is not constant and can be changed. The tag is copied back after the function has been completed.

<DataType>

The data type defines the type of a tag. The following table defines the possible data types and their representation in the individual programming languages or STEP 7:

Elementary data types:

ODK data type	SIMATIC data type	C++ data type	C# data type	VB data type	Description
ODK_DOUBLE	LREAL	double	double	Double	64-bit floating point, IEEE 754
ODK_FLOAT	REAL	float	float	Single	32-bit floating point, IEEE 754
ODK_INT64	LINT	long long	long	Long	64-bit signed integer
ODK_INT32	DINT	long	int	Integer	32-bit signed integer
ODK_INT16	INT	short	short	Short	16-bit signed integer
ODK_INT8	SINT	char	sbyte	SByte	8-bit signed integer
ODK_UINT64	ULINT	unsigned long long	ulong	ULong	64-bit unsigned integer
ODK_UINT32	UDINT	unsigned long	uint	UInteger	32-bit unsigned integer
ODK_UINT16	UINT	unsigned short	ushort	UShort	16-bit unsigned integer
ODK_UINT8	USINT	unsigned char	byte	Byte	8-bit unsigned integer
ODK_LWORD	LWORD	unsigned long long	ulong	ULong	64-bit bit string
ODK_DWORD	DWORD	unsigned long	uint	UInteger	32-bit bit string
ODK_WORD	WORD	unsigned short	ushort	UShort	16-bit bit string
ODK_BYTE	BYTE	unsigned char	byte	Byte	8-bit bit string
ODK_BOOL	BOOL	unsigned char	bool	Boolean	1-bit bit string, remaining bits (1..7) are empty
ODK_LTIME	LTIME	long long	long	Long	64-bit during in nanosec-onds
ODK_TIME	TIME	long	int	Integer	32-bit during in milliseconds
ODK_LDT	LDT	unsigned long long	ulong	ULong	64-bit date and time of the day in nanoseconds since 01/01/1970 00:00
ODK_LTOD	LTOD	unsigned long long	ulong	ULong	64-bit time of the day in nanoseconds since midnight
ODK_TOD	TOD	unsigned long	uint	UInteger	32-bit time of the day in milliseconds since midnight
ODK_WCHAR	WCHAR	wchar_t	char	Char	16-bit character
ODK_CHAR	CHAR	char	sbyte	SByte	8-bit character

Complex data types:

ODK data type	SIMATIC data type	C++ data type	C# data type	VB data type	Description
ODK_DTL	DTL	struct ODK_DTL	OdkInternal. Dtl (class)	OdkInternal. Dtl (class)	Structure for date and time
ODK_S7WSTRING	WSTRING	unsigned short	string	String	Character string: • For SIMATIC and C++: 16-bit character with length max. and act. (4xUSINT) • For other languages: native
ODK_S7STRING	STRING	unsigned char	string	String	Character string: • For SIMATIC and C++: 8-bit character with length max. and act. (2xUSINT) • For other languages: native
ODK_VARIANT	VARIANT	struct ODK_VARIANT	byte [ ]	byte [ ]	Classic data (each data type that can be serialized with classic data.)
[ ]	ARRAY	[ ]	[ ]	[ ]	Range of same data types. You can use all data types as array except IN_DATA / INOUT_DATA / OUT_DATA.

User-defined data types:

User-defined data types (UDT) include structured data, especially the names and data types of this component and their order.

A user-defined data type can be defined in the ODK interface description with the keyword "ODK_STRUCT".

Example

```
ODK_STRUCT <StructName>
{
 <DataType> <TagName>;
 ...
};
```

The following syntax rules apply to the structure:

- You can divide the structure into multiple lines.
- The structure definition must end with a semicolon.

- Any number of tabs and spaces between the elements is permitted.
- It is not permitted to use any keywords for the generated language used (for example "en / eno" as tag name).

You can create additional structures within a structure.

<StructName>

Structure names are valid with the syntax and character restrictions of the programming language and as defined for tag definitions in STEP 7.

In STEP 7, the structure name is extended with the STEP 7 prefix.

<TagName>

Tag names are subject to the syntax and character restrictions of the programming language.

Example

The following code example explains the definitions of functions and structures. Sort the parameters by: IN, OUT, INOUT.

```
//INTERFACE
…
ODK_STRUCT MyStruct
  {
    ODK_DWORD myDword;
    ODK_S7STRING myString;
  };
ODK_RESULT MyFct([IN] MyStruct myInStruct
                ,[OUT] MyStruct myOutStruct);
```

5.1.6.1 Using ODK_VARIANT as parameter

Restrictions of the data type ODK_VARIANT:

- When a parameter of the data type ODK_VARIANT is used, it is not permitted to use other parameters with the same InOut-Identifier, regardless of data type.
- With the data type ODK_VARIANT, an [OUT] is modeled as [INOUT] in the generated FB.

Example

```
// INTERFACE
...
// OK:
ODK_RESULT MyFunc1([IN]    ODK_VARIANT myClassicData);
ODK_RESULT MyFunc2([IN]    ODK_VARIANT myDataIn
                 , [OUT]   ODK_VARIANT myDataOut
                 , [INOUT] ODK_VARIANT myDataInout);
//
// NOT OK (Code Generator will throw an error):
// If ODK_VARIANT is used for [IN], no other [IN] parameter
// may be defined in this function
ODK_RESULT MyFunc4([IN] ODK_VARIANT myClassicData
```

```
                    , [IN] ODK_INT32 myint);
```

Application example for C++

```
#include "ODK_CpuReadData.h"
...
ODK_RESULT MyFunc1 (const ODK_VARIANT& myClassicData)
{
  CODK_CpuReadData myReader(myClassicData);
  ODK_INT32 myInt1, myInt2;
  myReader.ReadS7DINT(0, myInt1);
  myReader.ReadS7DINT(4, myInt2);
  return myInt1 + myInt2;
}
```

Helper functions (Page 139) of the following classes are available to help you access the data type ODK_VARIANT inside a user function:

- Class "CODK_CpuReadData"
- Class "CODK_CpuReadWriteData"

Note

Size of the ODK_VARIANT tags

The size of the ODK_VARIANT tags is not known at the time of compiling and is therefore not checked during the compiling process. When selecting the other parameters, consider the possible size of the ODK_VARIANT parameter in your application.

5.1.6.2 Handling strings

You can define a maximum length for strings (String or WString). Define the maximum number of characters in square brackets directly after the data type:

- ODK_S7STRING[30] or
- ODK_S7WSTRING[1000]

Without limitation, a string has a default length of 254 characters.

In order to access the data types ODK_S7STRING or ODK_S7WSTRING within a user function, the string helper functions (Page 139) are available:

Example

```
//INTERFACE
…
ODK_RESULT MyFct(
    [IN]    ODK_S7STRING      myStrHas254Chars
  , [OUT]   ODK_S7STRING[10]  myStrHas10Chars
  , [INOUT] ODK_S7STRING[20]  myStrArrayHas20Chars5Times[5]);
```

If you use [INOUT], you can set the string with a length that differs from the [INOUT of the function block in STEP 7.

Note

Compatibility

If you use the "WSTRING" data type with more than 253 characters in a project, create a new project with ODK version ≥ V2.5 SP1.

The characters are not read/written correctly with a project created with ODK version < V2.5 SP1.

5.1.6.3 Definition of the <Project>.odk file

The function prototypes and function blocks are generated based on the selected parameters in the <project>.odk file. Define the <project>.odk file for this.

By default, the <project>.odk file contains the following:

- Description

  The possible data types that are used for the interface are described in comment lines. This simplifies the definition of the correct tag type for your task.

- Context=user

  The CPU function library is loaded in the "User" context. You can change the parameter to Context=system.

- STEP7Prefix="<project>"

  Sets a string for the SCL generation in front of the functions of the CPU function library. The string is visible in STEP 7. You can change the parameter. The string length of the prefix including the function name must not exceed a length of 125 characters (for example, ODK_App_SampleFunction)

- "SampleFunction" function definition

  You can change this default function as you wish in the <project>.odk file and add more functions. The string length may not exceed a length of 125 characters. The associated function is located in the CPP file.

  FullClassName="<OdkProject1.Source.CpuFunctionLibrary>"

  The parameter is required for the C# and VB programming languages.

  To change the class names or namespace of the source files of the CPU function library, you need to adjust the "FullClassName" parameter.

Example

```
//INTERFACE
Context=user
STEP7Prefix=ODKProject
FullClassName=ODKProject.Source.CpuFunctionLibrary

 /*
* Elementary data types:
*  ODK_DOUBLE      LREAL    64-bit floating point, IEEE 754
*  ODK_FLOAT       REAL     32-bit floating point, IEEE 754
*  ODK_INT64       LINT     64-bit signed integer
*  ODK_INT32       DINT     32-bit signed integer
*  ODK_INT16       INT      16-bit signed integer
*  ODK_INT8        SINT     8-bit signed integer
*  ODK_UINT64      ULINT    64-bit unsigned integer
*  ODK_UINT32      UDINT    32-bit unsigned integer
*  ODK_UINT16      UINT     16-bit unsigned integer
*  ODK_UINT8       USINT    8-bit unsigned integer
*  ODK_LWORD       LWORD    64-bit bit string
*  ODK_DWORD       DWORD    32-bit bit string
*  ODK_WORD        WORD     16-bit bit string
*  ODK_BYTE        BYTE     8-bit bit string
*  ODK_BOOL        BOOL     1-bit bit string
*  ODK_LTIME       LTIME    64-bit duration in nanoseconds
*  ODK_TIME        TIME     32-bit duration in milliseconds
*  ODK_LDT         LDT      64 bit date and time of day
*                           in nanoseconds
*  ODK_LTOD        LTOD     64 bit time of day in nanoseconds
*                           since midnight
*  ODK_TOD         TOD      32 bit time of day in milliseconds
*                           since midnight
*  ODK_CHAR        CHAR     8 bit character
*  ODK_WCHAR       WCHAR    16 bit character
* Complex Datatypes:
*  ODK_DTL         DTL      structure for date and time
*  ODK_S7STRING    STRING   character string with 8-bit characters
*  ODK_VARIANT     VARIANT  classic data (any datatype which can be
serialized
*                           to classic data)
*  ODK_S7WSTRING   WSTRING  character string with 16 bit characters
*  []              ARRAY    field of this datatype
* User Defined Datatype:
*  ODK_STRUCT      UDT      user defined structure
* Return Datatype:
*  ODK_RESULT      0x0000-0x6FFF function succeeded
*                               (ODK_SUCCESS = 0x0000)
*                  0xF000-0xFFFF function failed
*                               (ODK_USER_ERROR_BASE = 0xF000)
*/

// Basic function in order to show
// how to create a function in ODK 1500S.
ODK_RESULT SampleFunction([IN]   ODK_INT32   myInt   // integervalue
                                                     // as input
                        , [OUT]  ODK_BOOL    myBool  // bool value
```

```
                                                    // as output
                                  , [INOUT] ODK_DOUBLE myReal);// double value
                                                    // as input
                                                    // and output
```

5.1.6.4 Modifying the <Project>.odk file

The following examples show you how you can change the <project>.odk file to suit your needs.

```
//INTERFACE
Context=user
STEP7Prefix=ODK_SampleApp_

ODK_RESULT GetString ([OUT]    ODK_S7STRING myString);

ODK_RESULT Calculate ([IN]     ODK_INT64   In1,
                      [IN]     ODK_DOUBLE  In2,
                      [OUT]    ODK_FLOAT   Out1,
                      [OUT]    ODK_INT32   Out2,
                      [INOUT]  ODK_BYTE    InOut1[64],
                      [INOUT]  ODK_BYTE    InOut2[64]);
```

Function prototypes in the ODK file

Example for C++

```
ODK_RESULT GetString (
    /*OUT*/  ODK_S7STRING myString[256]);
#define _ODK_FUNCTION_GETSTRING ODK_RESULT GetString (/*OUT*/
ODK_S7STRING myString[256])

ODK_RESULT Calculate (
    /*IN*/     const ODK_INT64&   In1,
    /*IN*/     const ODK_DOUBLE&  In2,
    /*OUT*/    ODK_FLOAT&   Out1,
    /*OUT*/    ODK_INT32&   Out2,
    /*INOUT*/  ODK_BYTE    InOut1[64],
    /*INOUT*/  ODK_BYTE    InOut2[64]);
#define ODK_FUNCTION_CALCULATE ODK_RESULT Calculate(/*IN*/ const
ODK_INT64& In1,/*IN*/ 2480 const ODK_DOUBLE& In2,/*OUT*/ ODK_FLOAT&
Out1,/*OUT*/ ODK_INT32& Out2,/*INOUT*/ ODK_BYTE2481
InOut1[64],/*INOUT*/ ODK_BYTE InOut2[64])

#endif // ODK_FUNCTIONS_H
```

Example for C#

```
namespace OdkInternal
{
  interface IOdkFunctions
  {
    // declaration of the callback methods
    ushort OnLoad();
    ushort OnUnload();
    ushort OnRun();
    ushort OnStop();

    ushort GetString(
           /*OUT*/   out string myString);

    ushort Calculate(
           /*IN*/    ref long   In1,
           /*IN*/    ref double In2,
           /*OUT*/   out float  Out1,
           /*OUT*/   out int    Out2,
           /*INOUT*/ ref byte[] InOut1,
           /*INOUT*/ ref byte[] InOut2);
  }
}
```

Example for VB

```
Namespace Global.OdkInternal
  Public Interface IOdkFunctions
    // declaration of the callback methods
    Function OnLoad() As UShort
    Function OnUnload () As UShort
    Function OnRun () As UShort
    Function OnStop () As UShort

    Function GetString(
                      ByRef myString As String 'OUT
  ) As UShort
    Function Calculate(
                      ByRef In1 As Long,       'IN
  ByRef In2 As Double,     'IN
   ByRef Out1 As Float,     'OUT
  ByRef Out2 As Integer,   'OUT
   ByRef InOut1() As Byte,  'INOUT
                      ByRef InOut2() As Byte   'INOUT
                      ) As UShort
  End Interface
End Namespace
```

5.1.6.5 Comments

The following examples for using comments are valid for C++ and C#. Differences to Visual Basic are available under "Comments in Visual Basic (Page 47)"

Comments are started with a double slash "//" and end automatically at the end of the line.

Alternatively, you can limit comments by /* <*comment*> */, which enables new lines in a comment. Characters after the end of the comment identifier "*/" are further processed by the code generator.

Comments for functions and structures

You place comments on functions and structures directly in front of the functions/structures.

These comments are transferred to the ODK_Functions.h/.cs/.vb and <project>.scl files.

In the <project>.scl file, the comments are copied to the block properties and duplicated in the code area of the function.

Observe the following rules:

- Comments for functions and structures must be located directly in front of the functions/structures (without blank line).
- The end of the comment is located in front of the ODK_RESULT or ODK_STRUCT keyword.
- You can use both identifiers "//" and "/* */" but not in combination within a comment.

Example

```
// this comment did not appear in MyStruct, because of the empty
line.

// comment MyStruct
// ...
ODK_STRUCT MyStruct
{
  ODK_DWORD     myDword;
  ODK_S7STRING  myString;
};

/*
comment MyFct
...
*/
ODK_RESULT MyFct([IN] MyStruct myInStruct
                ,[OUT] MyStruct myOutStruct);
```

Comments for tags in functions and structures

Comments for function and structure tags are placed directly in front of or behind the tag.

These comments are transferred to the ODK_Functions.h/ and <project>.scl files.

The following rules apply to comments **in front of** tags:

- Comments must be directly in front of the tag (without blank line).
- The end of the comment is the <InOut-Identifier> of the tags.

The following rules apply to comments **after** tags:

- Comments must be after the tag name (without blank line).

The following general rules apply to comments for tags:

- You can use both identifiers "//" and "/* */" but not in combination within a comment.
- In the header file, the same comment identifier is used ("//" or "/* */").

Example

```
ODK_STRUCT MyStruct
{
  // comment myDword BEFORE definition
  ODK_DWORD    myDword;

  ODK_S7STRING myString; /* comment myString AFTER definition */
};

ODK_RESULT MyFct([IN]  MyStruct myInStruct    // comment
                                              // myInStruct ...
                                              // ... "second line"
                , [OUT] MyStruct myOutStruct); /* comment
                                                  myOutStruct ...
                                                  ...
                                               */
```

5.1.6.6 Comments in Visual Basic

Not all comments can be transferred unchanged from the Interface file to the VB source.

The following rules are valid only for comments in Visual Basic:

- Comments are marked with a apostrophe.
- To mark multiple lines as comment, you need to set an apostrophe before each line.

  Example:

<project>.odk	**ODK_Functions.vb**
/* Multi line comment 1 comment 2 comment 3*/ ODK_RESULT f1();	' This file is AUTO GENERATED … ' <automatically generated comment> ... ' Multi line comment 1 ' comment 2 ' comment 3 Function f1() As UShort

- Comments are not permitted in front of source code.

  Set the InOut identifier after the function parameter.

  Example:

<project>.odk	**ODK_Functions.vb**

`ODK_RESULT f1([IN] ODK_BYTE b);`	`Function f1(`
	` b As Byte ' [IN]`
	` ) As UShort`

- Multi-line comments are not permitted between function parameters.

  Set multiple comments in a line.

  Example:

<project>.odk	**ODK_Functions.vb**

`ODK_RESULT f1(`
` // c1`
` // c2`
` [IN] ODK_BYTE b // c3`
`  // c4`
` );`
	`Function f1(`
	`  b As Byte ' [IN] c1' c2' c3' c4`
	` ) As UShort`

5.1.7 Implementing functions

5.1.7.1 General notes

This section provides an overview of the basic topics relating to the implementation of functions in a Windows environment.

- The function call is not limited in time, because the function is called asynchronously.
- Traces are possible via OutputDebugString instructions
- All asynchronous functions are executed with equal priority - regardless of the priority of the OBs
- The complete Windows API (Application Programming Interface) and C++-Runtime library are available

5.1.7.2 Callback functions

The project template includes an execute file to define your functions.

Programming language	Name of the execute file
C++	<project>.cpp
C#	<project>.cs
VB	<project>.vb

This execute file contains functions filled by default. This file does not necessarily need to be filled with additional user code to be usable. However, neither may the functions be deleted under any circumstances.

The empty function has the following code (using the "OnLoad()" function as an example):

You can define the following functions in the execute file:

- OnLoad(): Called after loading the CPU function library
- OnUnload(): Called before unloading the CPU function library
- OnRun(): Called when the CPU changes to RUN mode after the OnLoad() function
- OnStop(): Called when the CPU changes to the STOP mode and before the function OnUnload()

The following table provides an overview of the various actions to invoke the callback functions:

Current operating state	New operating state	User action	ODK action
RUN	RUN	ODK_Load	1. OnLoad() 2. OnRun()
STOP	RUN	ODK_Load in startup OB (e.g. OB100)	1. OnLoad() 2. OnRun()
RUN	STOP	*<already loaded>*	OnStop()
STOP	RUN	*<already loaded>*	OnRun()
RUN	RUN	ODK_Unload	1. OnStop() 2. OnUnload()
RUN	SHUTDOWN / MRES	*<already loaded>*	OnStop()
any	*any*	*<already loaded>* Exit ODK host	1. OnStop() (optional, if not already executed) 2. OnUnload()

"OnLoad()" and "OnUnload()" function

The functions have a return value of type "ODK_RESULT" and typically provide information about the status of the "ODK_SUCCESS" value.

The following return values are possible:

Return value for "ODK_RESULT"	Description
ODK_SUCCESS = 0x0000	Return value following a successful execution of the "OnLoad()" or "OnUnload()" function
0x0001 – 0xEFFF	Invalid values (system-internal)
0xF000 – 0xFFFF ODK_USER_ERROR_BASE = 0xF000	You can define your own error values. The loading stops and the CPU function library unloads for the "OnLoad()" function. The CPU function library within the specified value range is still unloaded for the "OnUnload()" function.

"OnRun()" and "OnStop()" function

The functions have a return value of type "ODK_RESULT" and typically provide information about the status of the "ODK_SUCCESS" value.

The following return values are possible:

Return value for "ODK_RESULT"	Description
ODK_SUCCESS = 0x0000	Return value following a successful execution of the "OnRun()" or "OnStop()" function
0x0001 – 0xFFFF	No direct feedback to the user program is possible. The return value is sent to Windows (WindowsEventLog).

5.1.7.3 Implementing custom functions

Once you have defined the ODK interface in the <project>.odk file, you must edit the functions of the CPU function library in the Project Source file.

Procedure

To edit the function of a CPU function library, follow these steps:

1. To generate the function prototypes, execute the build.

2. Open the project source file, or create a custom source file if necessary.

3. Transfer the function prototypes from <ODK_Functions.h>/<OdkFunctions.cs/vb> to the source file.

Note

Use the function prototype macro to transfer the step 3 in the future when there is a change to the function parameters.

4. Edit the code of your CPU function library in the execute file.

CPU function library

The execute file contains a schematically represented function description by default. You can change this description with corresponding changes in the <project>.odk file and/or add more function descriptions.

Execute file based on C++ example

```
#include "ODK_Functions.h"

EXPORT_API ODK_RESULT OnLoad (void)
{
    return ODK_SUCCESS;
}
EXPORT_API ODK_RESULT OnUnload (void)
{
    return ODK_SUCCESS;
}
EXPORT_API ODK_RESULT OnRun (void)
{
    return ODK_SUCCESS;
}
EXPORT_API ODK_RESULT OnStop (void)
{
    return ODK_SUCCESS;
}
ODK_RESULT SampleFunction( const ODK_INT32& myInt,
                                 ODK_BOOL& myBool,
                                 ODK_DOUBLE& myReal)
{
    return ODK_SUCCESS;
}
```

5.2 Transferring a CPU function library to the target system

Manually transfer the DLL file to a specific Windows folder on the target system (e.g. via a network share or USB flash drive). Use the standard Windows data transfer procedure to transfer of the CPU function library. The storage location in Windows is specified by a registry key. When loading an CPU function library, the ODK service automatically searches for the file in the path specified by the registry key.

Note

CPU function library in the debug configuration

When the CPU function library has been transferred to the debug configuration, you also need to transfer the debug DLLs of the development environment to the target system.

The default value that describes the file path is:

%ProgramData%\Siemens\Automation\ODK1500S\

Note

Administrator rights

Assign write permission to this folder only for the administrator. This prevents unauthorized personnel from uploading CPU function libraries.

Please note:

The setup of the SIMATIC S7-1500 Software Controller checks whether the file path already exists and the required administrator rights are assigned.

If not, the directory is renamed to "ODK1500S_OLD1" or "ODK1500S_OLD2" and a new directory with the correct access rights is created.

The Windows file system can hide the folder based on your setting. You can view the folder using the Windows option "Show hidden files, folders, and drives".

The registry key for 32-bit systems is:
HKEY_LOCAL_MACHINE\SOFTWARE\Siemens\Automation\ODK1500S\odk_app_path

The registry key for 64-bit systems is:
HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Siemens\Automation\ODK1500S\odk_app_path

You can change the default value of the registry key and thus adapt to the expected location for the DLL file to suit your needs.

Note

Changing the path in the registry key

To protect the DLL file, select a storage location that is secured by access protection.

5.3 Importing and generating an SCL file in STEP 7

The following files are created when the project map is created:

- SCL file for importing into STEP 7
- All files depending on the configuration, e.g. DLL file

If STEP 7 is installed on another PC as the development environment, you must transfer the generated SCL file to the PC where the STEP 7 is installed.

Requirements

The project data were generated.

Procedure

To import and compile the SCL file, follow these steps:

1. Start STEP 7.
2. Open your project.
3. Select the project view.
4. Select the CPU in the project tree.
5. Select the "External Sources" subfolder.

   The "Open" dialog box opens.
6. Navigate in the file system to the SCL file that was created during the generation of the project data.
7. Confirm your selection with "Open".

   The SCL file is imported. After completion of the import process, the SCL file is displayed in the "External Sources" folder.
8. You need to compile the SCL file before you can use the blocks in your project.
9. To do this, select the SCL file in "External sources" subfolder.
10. Select the "Generate blocks from source" command in the shortcut menu.

Result

STEP 7 creates the S7 blocks based on the selected SCL file.

The created blocks are now automatically displayed in the "Program blocks" folder below the selected CPU in the project tree. You can load the function blocks during the next download to the target device.

5.4 Executing a function

5.4.1 Loading functions

Introduction

Regardless of the context in which the CPU function library is running, the loading procedure consists of the following steps:

- Call the "<*STEP7Prefix*>_Load" instruction in the STEP 7 user program.
- In the Windows context, the loading process checks if a 32-bit or 64-bit process is required and starts the appropriate host. Each CPU function library runs in a separate Windows process (ODK_Host).
- The host loads the CPU function library and calls the "OnLoad()" function and then the "OnRun()" functions.

Note

Loading the same CPU function libraries with a modified <project>.odk file

When you load an CPU function library and subsequently change the <project>.odk file, we recommend that you unload your CPU function library first before you load the newly generated CPU function library. If the "<*STEP7Prefix*>_Unload" instruction is not executed, both CPU function libraries are in the memory. This can lead to insufficient memory being available for the CPU.

"<*STEP7Prefix*>_Load" instruction

A CPU function library is loaded by calling the "<*STEP7Prefix*>_Load" instruction in the STEP 7 user program.

***<STEP7Prefix>*_Load**	
REQ	DONE
	BUSY
	ERROR
	STATUS

The following table shows the parameters of the instruction "<*STEP7Prefix*>_Load":

Section	Declaration	Data type	Description
Input	REQ	BOOL	A rising edge activates the loading of the CPU function library.
Output	DONE	BOOL	Indicates that the instruction has finished loading the CPU function library.
Output	BUSY	BOOL	Indicates that the instruction is still loading the CPU function library.
Output	ERROR	BOOL	Indicates that an error occurred during the loading of the CPU function library. STATUS gives you more information about the possible cause.
Output	STATUS	INT	Provides information about possible sources of error, if an error occurs during the loading of the CPU function library.

Input parameters

An edge transition (0 to 1) at the "REQ" input parameter starts the function.

Output parameters

The following table shows the information that is returned after loading.

DONE	BUSY	ERROR	STATUS	Meaning
0	0	0	0x7000 =28672	No active loading
0	1	0	0x7001 =28673	Loading in progress, first call
0	1	0	0x7002 =28674	Loading in progress, ongoing call
1	0	0	0x7100 =28928	CPU 1500 V2.0 and later: CPU function library is already loaded.
1	0	0	0x0000 =0	Loading was performed successfully.
0	0	1	0x80A4 =-32604	CPU function library could not be loaded. Start the ODK service manually or restart Windows.
			0x80C2 =-32574	CPU function library could not be loaded. There are currently not enough resources available from Windows. Reload the CPU function library after a few seconds.
			0x80C3 =-32573	CPU function library could not be loaded. The CPU currently does not have enough resources. Reload the CPU function library after a few seconds.
			0x8090 =-32624	CPU function library could not be loaded. An exception occurred during execution of the "OnLoad()" function.
			0x8092 =-32622	CPU function library could not be loaded because the library name is invalid.
			0x8093 =-32621	CPU function library could not be loaded because the CPU function library could not be found. Check the file name and path of the file.
			0x8094 =-32620	CPU function library could not be loaded. The CPU function library was created for the Windows user context, but no user is logged on.

DONE	BUSY	ERROR	STATUS	Meaning
			0x8095 =-32619	CPU function library could not be loaded due to the following reasons: • The DLL file is not a CPU function library • An attempt has been made to load a 64-bit application into a 32-bit system • Dependencies on other Windows DLL files could not be resolved. – Check that the release build of the CPU function library is used. – Check whether the "Visual C++ Redistributables" are installed for the Visual Studio version you are using. • The CPU does not support the utilized ODK version.
			0x8096 =-32618	The CPU function library could not be loaded because the internal identification is already being used by another loaded CPU function library.
			0x8097 =-32617	CPU 1500 V1.8 and earlier: CPU function library is already loaded.
			0x8098 =-32616	The CPU function library could not be loaded because the CPU function library is currently being unloaded.
			0x809B =-32613	CPU 1500 V2.0 and later: The CPU function library could not be loaded and returns an invalid value (the values 0x0000 and 0xF000 - 0xFFFF are allowed)
			0xF000 – 0xFFFF =-4096 – -1	CPU 1500 V2.0 and later: CPU function library could not be loaded. An error occurred during execution of the "OnLoad()" function.

Example

This example describes how the loading and execution of a Windows CPU function library can be implemented for the Windows environment in STEP 7 after communication disturbances.

When Windows is again available the CPU function library is loaded and the execution of the functions is again possible.

A communication disturbance can be caused by the following:

- Windows Restart (or Shut down)
- Windows Log off (if application in user area)
- TerminateProcess/ODK_Host crash

A flag is necessary for this (here: ODK_Loaded), which is set after successful loading and is reset following a faulty execution of the ODK function.

```
FUNCTION_BLOCK "ODK_AutoLoad"
{ S7_Optimized_Access := 'TRUE' }
VERSION: 0.1
  VAR
    ODK_Loaded : Bool;
  END_VAL
BEGIN
  // Loading of the Windows-CPU function library
  IF NOT #ODK_Loaded THEN
    // Toggle request flag if loading is not active
```

```
    IF NOT "ODKProject_Load_DB".BUSY THEN
           "ODKProject_Load_DB".REQ := NOT "ODKProject_Load_DB".REQ;
    END_IF;

    // Loading of the CPU function library
    "ODKProject_Load_DB"();

    // Set "Loaded" flag if loading is successful
    IF "ODKProject_Load_DB".DONE THEN
      #ODK_Loaded := true;
    END_IF;
  END_IF;

  // Execute the ODK function(s) (only in loaded state)
  IF #ODK_Loaded THEN
    // Toggle request flag if function call is not active
    IF NOT "ODKProjectSampleFunction_DB".BUSY THEN
           "ODKProjectSampleFunction_DB".REQ := NOT
           "ODKProjectSampleFunction_DB".REQ;
    END_IF;

    // Execute the function
    "ODKProjectSampleFunction_DB"();

    // The "Loaded" flag must be reset when
    // a) An error is present in the communication with Windows
(0x80A4)
    // b) the CPU function library was already unloaded before this
function call (0x8096)
    IF "ODKProjectSampleFunction_DB".STATUS = 16#80A4 OR
"ODKProjectSampleFunction_DB".STATUS = 16#8096
    THEN
    #ODK_Loaded := false;
    END_IF;
  END_IF;
END_FUNCTION_BLOCK
```

5.4.2 Calling functions

Introduction

Once the CPU function library is loaded, you can execute functions via your STEP 7 user program. This call is made from the corresponding "<*STEP7Prefix*>SampleFunction" instruction.

You can load up to 32 CPU function libraries at the same time.

"*<STEP7Prefix>*SampleFunction" instruction

A CPU function library is called by the "*<STEP7Prefix>*SampleFunction" instruction.

***<STEP7Prefix>*SampleFunction**	
REQ	DONE
myInt	BUSY
myReal	ERROR
	STATUS
	myBool

The following table shows the parameters of the instruction "*<STEP7Prefix>*SampleFunction":

Section	Declaration	Data type	Description
Automatically generated parameters			
Input	REQ	BOOL	A rising edge of this input value activates the execution of the CPU function library.
Output	DONE	BOOL	This output value indicates that the instruction has finished execution of the CPU function library.
Output	BUSY	BOOL	This output value indicates that the instruction is still unloading the CPU function library.
Output	ERROR	BOOL	This output value indicates that an error occurred during the execution of the CPU function library. The STATUS output value provides more information on this.
Output	STATUS	INT	This output value provides information about possible sources of error, if an error occurs during the execution of the CPU function library.
User-defined parameter			
Input	myInt		User-defined input tags
InOut	myReal		User-defined input-output tags
Output	myBool		User-defined output tags

Input parameters

An edge transition (0 to 1) at the "REQ" input parameter starts the function.

Output parameters

The following table shows the information for the output parameters returned after execution.

DONE	BUSY	ERROR	STATUS	Meaning
0	0	0	0x7000 =28672	No active process
0	1	0	0x7001 =28673	First call (asynchronous)
0	1	0	0x7002 =28674	Continuous call (asynchronous)
1	0	0	0x0000 – 0x6FFF =0 – 28671	Function has been executed and returns a value between 0x0000 and 0x6FFF. (ODK_SUCCESS = 0x0000)
0	0	1	0x80A4 =-32604	CPU function library could not be executed for the following reasons: • The "*<STEP7Prefix>*_Unload" instruction was executed during a function execution. The function execution was aborted at the CPU end. Windows terminates the execution of the function normally. No return value is sent to the CPU. Wait until the "*<STEP7Prefix>*_Unload" instruction has ended. Then load the CPU function library again. • Windows is not available • ODK service is not running Start the ODK service manually or restart Windows.
			0x80C2 =-32574	CPU function library could not be executed. There are currently not enough resources available from Windows. Execute the CPU function library again after a few seconds.
			0x80C3 =-32573	CPU function library could not be executed. The CPU currently does not have enough resources. Execute the CPU function library again after a few seconds.
			0x8090 =-32624	CPU function library could not be executed. An error occurred during execution.
			0x8091 =-32623	CPU function library could not be executed. A "STOP" occurred during the function call.
			0x8096 =-32618	CPU function library could not be executed because the CPU function library was not loaded or unloading is not yet finished.
			0x8098 =-32616	CPU function library could not be executed because the function is not supported.
			0x8099 =-32615	CPU function library could not be executed because the maximum amount of input data (1 MB) was exceeded (declarations with "In" and "InOut")
			0x809A =-32614	CPU function library could not be executed because the maximum amount of output data (1 MB) was exceeded (declarations with "Out" and "InOut")
			0x809B =-32613	The function returns an invalid value (a value between 0x0000 and 0x6FFF; 0xF000 and 0xFFFF is permitted)

DONE	BUSY	ERROR	STATUS	Meaning
			0x809C =-32612	Function uses an invalid data type: • IN_DATA • INOUT_DATA • OUT_DATA
			0xF000 – 0xFFFF =-4096 – -1	CPU 1500 V2.0 and later: The function could not be executed and returns a value between 0xF000 and 0xFFFF. (ODK_USER_ERROR_BASE = 0xF000)

Note

Call of function(s) influences the cycle time

When you call a function, the function parameters are copied. In particular in the case of large amounts of data or of structured data, this can lead to the cycle time being influenced.

5.4.3 Unloading functions

Introduction

The CPU function library is unloaded by calling the "<*STEP7Prefix*>_Unload" instruction. Call is made from the STEP 7 user program.

In addition to this call, the CPU function library is also automatically unloaded for the following reasons.

- The CPU is switched off
- Memory reset of CPU
- Windows is restarted
- Logoff off the Windows user (in the context of a Windows user)

Regardless of the context in which the CPU function library is running, the unloading procedure consists of the following steps:

- Call the "<*STEP7Prefix*>_Unload" instruction in the STEP 7 user program.
- From now on, no new executes can be carried out for this CPU function library. Still active executes are terminated at the CPU end. Windows terminates the execution of the function normally ("Unload" waits). No return value is sent to the CPU.
- The host calls the "OnStop()" and "OnUnload()" functions.
- The CPU function library is being unloaded.

"*<STEP7Prefix>*_Unload" instruction

A CPU function library is unloaded by calling the "*<STEP7Prefix>*_Unload" instruction in the STEP 7 user program.

***<STEP7Prefix>*_Unload**	
REQ	DONE
	BUSY
	ERROR
	STATUS

The following table shows the parameters of the instruction "*<STEP7Prefix>*_Unload":

Section	Declaration	Data type	Description
Input	REQ	BOOL	A rising edge activates the unloading of the CPU function library.
Output	DONE	BOOL	Indicates that the instruction has finished unloading the CPU function library.
Output	BUSY	BOOL	Indicates that the instruction is still unloading the CPU function library.
Output	ERROR	BOOL	Indicates that an error occurred during the unloading of the CPU function library. STATUS gives you more information about the possible cause.
Output	STATUS	INT	Provides information about possible sources of error, if an error occurs during the unloading of the CPU function library.

Input parameters

An edge transition (0 to 1) at the "REQ" input parameter starts the function.

Output parameter STATUS

The following table shows the information that is returned after unloading.

DONE	BUSY	ERROR	STATUS	Meaning
0	0	0	0x7000 =28672	No active unloading
0	1	0	0x7001 =28673	Unloading in progress, the first call
0	1	0	0x7002 =28674	Unloading in progress, ongoing call
1	0	0	0x0000 =0	Unloading was carried out successfully
0	0	1	0x80A4 =-32604	CPU function library could not be unloaded for the following reasons: • Windows is not available Start the ODK service manually or restart Windows.
			0x80C2 =-32574	CPU function library could not be unloaded. There are currently not enough resources available from Windows. Reload the CPU function library after a few seconds.
			0x80C3 =-32573	CPU function library could not be unloaded. The CPU currently does not have enough resources. Reload the CPU function library after a few seconds.

DONE	BUSY	ERROR	STATUS	Meaning
			0x8090 =-32624	An exception occurred during the unloading of the CPU function library. The CPU function library has been unloaded nevertheless.
			0x8096 =-32618	CPU function library could not be unloaded because the CPU function library was not loaded or unloading is not yet finished.
			0x809B =-32613	CPU 1500 V2.0 and later: The CPU function library could be unloaded and returns an invalid value (the values 0x0000 and 0xF000 - 0xFFFF are allowed)
			0xF000 – 0xFFFF =-4096 – -1	CPU 1500 V2.0 and later: CPU function library could be unloaded. An error occurred in the CPU function library during the execution of the "OnUnload()" function.

5.5 Remote debugging

If you use Microsoft Visual Studio as a development environment, you can use the debugger for debugging.

You can use the remote debugger to debug a CPU function library on a target system without Visual Studio. It should be noted that the generated CPU function libraries (DLLs) are loaded into one of the following processes:

- ODK_Host_x86.exe process (32-bit)
- ODK_Host_x64.exe process (64-bit)

The required remote debugger is dependent on the Visual Studio version used on the host system and on the system type (32-bit/64-bit) of the target system.

You can find links to download the remote debugger for the relevant Visual Studio version on the Microsoft website (https://docs.microsoft.com/en-us/visualstudio/debugger/remote-debugging?#download-and-install-the-remote-tools).

After downloading, you can install the remote debugger on the target system.

5.5.1 Performing remote debugging

Procedure

1. Start the Visual Studio remote debugger on the target system using "Start > All Programs > Visual Studio 20xx > Remote Debugger".
2. Configure the authentication.

   Select the "No authentication" option and select the "Allow any user to debug" check box.

   Observe the security information.
3. With a C++ CPU function library, copy the Visual Studio Debug DLLs from the folder "<installation path VS>\VC\redist\Debug_NonRedist\<ApplicationType>\Microsoft.<VS version>.DebugCRT" in the target folder. With a managed (C# / VB) CPU function library you can skip step 3.
   - Destination folder with 32-bit Windows and a 32-bit application:

     <windows install path>\System32
   - Destination folder with 64-bit Windows and a 64-bit application:

     <windows install path>\System32
   - Destination folder with 64-bit Windows and a 32-bit application:

     <windows install path>\SysWOW64

   Note

   You need the file "ucrtbased.dll".

   If this DLL is not present in the target system, copy it from the host in the folder:

   With 32-bit Windows under Program Files\...

   With 64-bit Windows under Program Files (x86)\...

   ...\Microsoft SDKs\Windows Kits\10\ExtensionSDKs\Microsoft.UniversalCRT.Debug\<Highest available version>\Redist\Debug\<Application type (32/64-bit)>
4. Load the CPU function library on the target system in the folder "C:\ProgramData\Siemens\Automation\ODK1500S".

   Note

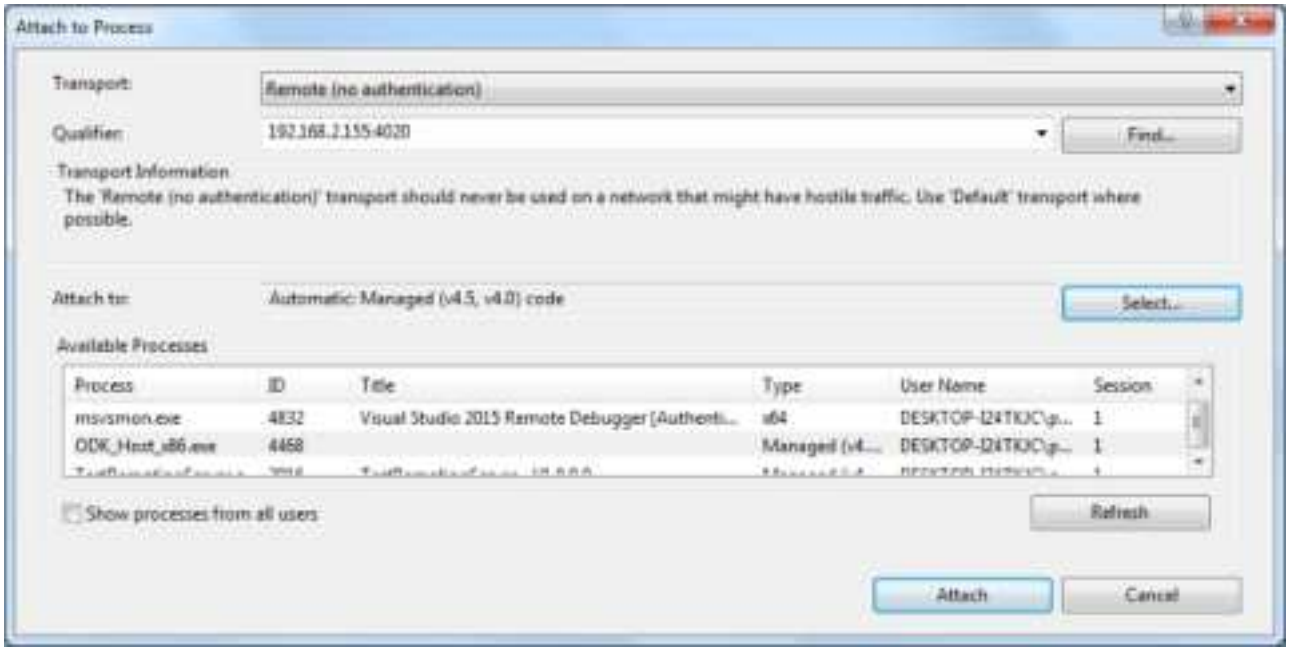
   If the CPU function library is loaded, unload (Page 60) it before copying.

5. Load (Page 54) the CPU function library on the target system.
6. Set the breakpoints in the source code and start the debugger via "Debug > Attach to Process...".

   Select the following settings in the "Attach to Process" dialog:
   - Transport: Remote
   - Qualifier: IP address of the target system and port of the remote debugger.
   - Attach to:

     Use the default value "Automatic: Managed (...) code" for managed CPU function libraries.

     Only for a C++ CPU function library: Click "Select...", and select the code type "Native" in the "Select Code Type" dialog.



Debugging OnLoad/OnRun

To attach the debugger to the OnLoad() or OnRun() function, incorporate a wait loop at the start of OnLoad().

Example of a wait loop:

```
EXPORT_API ODK_RESULT OnLoad (void)
{
#if defined _DEBUG  // available in debug configuration, only
  while (!IsDebuggerPresent()) // wait for debugger

  {
    Sleep(100);
  }
#endif
  // your code for OnLoad() ...
```

Result

The debugger stops the execution of the code after the activated breakpoint.

Developing a CPU function library for the realtime environment

6

6.1 Creating a CPU function library

6.1.1 Requirements

- ODK is installed. The Eclipse development environment is installed.
- You need administrator rights to create and edit an Eclipse project (CPU function library for the realtime environment).

Note

If you have to move the workspace to a different storage location, make sure you copy the entire workspace.

Note

SO files (CPU function libraries)

The SO files are not know-how-protected. The customer is responsible for the SO files and their know-how protection.

Note

Behavior with a large CPU function library for the real-time environment

When an exception is thrown for a CPU function library as of about 20 MB for the real-time environment, the CPU may no longer change from "STOP" to "RUN".

Make sure that you have sufficient load memory for backup of the postmortem files. Then switch the CPU off and on again.

6.1.2 Creating a project

To help you develop a CPU function library, an Eclipse Project template is included in the installation of ODK 1500S.

Procedure

To create a CPU project in Eclipse using an ODK template, follow these steps:

1. Start Eclipse as a development environment.
2. In the "File > New" menu, select the command "Project..."

   The "New Project" dialog opens.

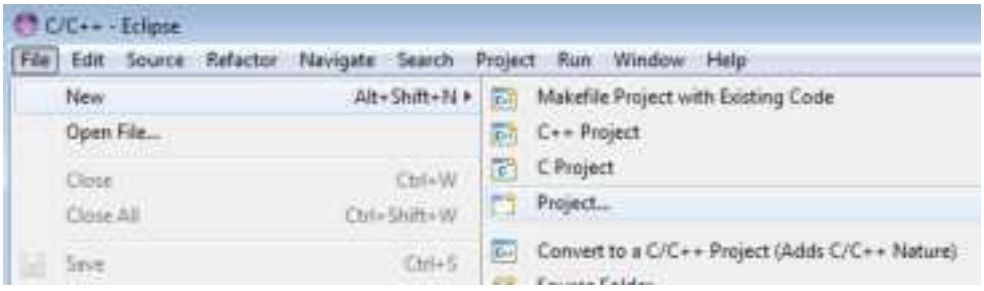


Figure 6-1 Creating a new project with Eclipse

3. Select the project template "C++ Project for CPU function library (CPU Runtime)" .

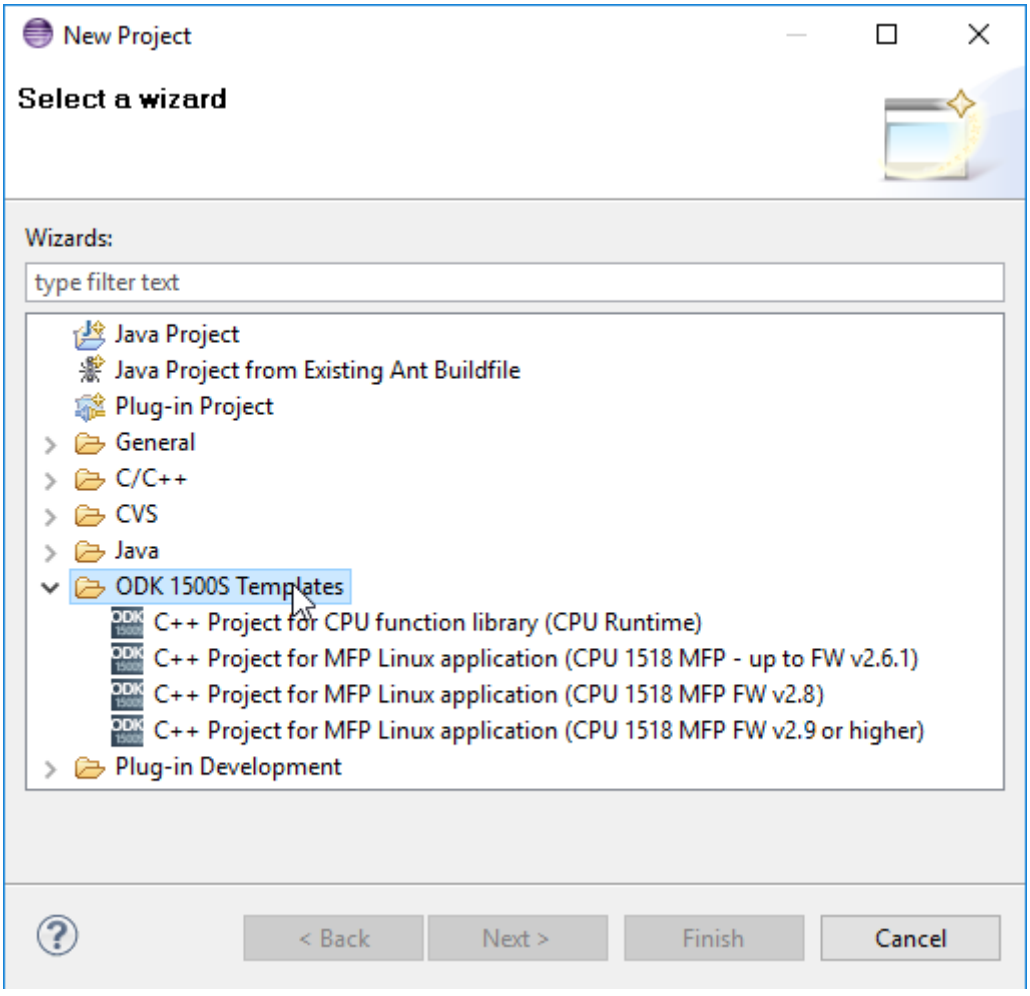


Figure 6-2 Selecting a template

4. Enter a project name.
5. Click "OK" to confirm.

Result

The CPU function library for the realtime environment is created using the project template and sets the following project settings:

- Project settings for generating the SO file
- Automates the generation of the SO and SCL file

The project template sets up the following Project Explorer by default:

Folder / file			Description
<project path>			
	def		
		<project>.odk	ODK interface description
		<project>.scl.additional	S7 blocks that are appended to the <project>.scl file. Although the file is not part of the project template, the code generator processes the file.
	STEP7		**Files from this folder may not be edited!**
		<project>.scl	S7 blocks
	scr_cg_priv		**Files from this folder may not be edited!**
		ODK_Types.h	Definition of the ODK base types
		ODK_Functions.h	Function prototypes
		ODK_Execution.cpp	Implementation of the "Execute" method
	src		
		<project>.cpp	Function code: This file has always the suffix CPP, regardless of whether you are creating a C or C++ project.
	src_odk_helpers		**Files from this folder may not be edited!**
		ODK_CpuReadData.h	Definition of the helper function for reading classic DBs.
		ODK_CpuReadData.cpp	Implementation of the helper function for reading classic DBs.
		ODK_CpuReadWriteData.h	Definition of the helper function for reading/writing classic DBs.
		ODK_CpuReadWriteData.cpp	Implementation of the helper function for reading/writing classic DBs.
		ODK_StringHelper.h	Definition of the helper function for access to S7String/S7WString.
		ODK_StringHelper.cpp	Implementation of the helper function for access to S7String/S7WString.
	release_so		
		<project>.so	ODK Application Binary (release version) that must be transferred to the target system.
		<project>.debuginfo.so	ODK Application Binary (debug version) that is required for the post mortem analysis.
		<project>.symbols	Symbol information that is required for the post mortem analysis.
	launches		
		<project>.gdb.launch	Start for the post mortem analysis.

Note

Invalid characters in the project name

All invalid characters in the project name are automatically replaced by an underscore. These characters are allowed {A...Z, a...z, 1...9, -, _}.

"My + first#project" becomes, for example, "My___first_project".

6.1.3 Generating a CPU function library

The generation of the project data is divided into two automated steps.

- **Pre-Build**: Generation of the files created by default based on the changed <Project>.odk file
- **Build**: Generation of the SO file

Procedure

To generate the project data, follow these steps:

1. Save all edited files.
2. In the "Build" menu, select the command "Build Project".

Note

The project data is only generated if the files have been changed.

Result

The generation of the project data is started. The automatically generated files are stored in the file system.

- SO file: Project directory\<Project>\<BuildConfiguration>\<Project>.so
- SCL file: Project directory\<Project>\STEP7\<Project>.scl

6.1.4 Defining the runtime properties of a CPU function library

The next step is to define the interface description of the CPU function library in the <project>.odk file. The file contains the following elements:

- Comments
- Parameters
- Definitions of functions and structures

Procedure

To define the interface description in the <project>.odk file, follow these steps:

1. Open the <project>.odk file.
2. Change the elements depending on your requirements.

Description of the elements

Comments

You can use comments for explanation purposes.

Parameters

The definition of the parameters must be within a line of code.
`<parameter name>=<value> // optional comment`

The interfaces file supports the following parameters:

Parameter	Value	Description
Context	realtime	Specifies that the CPU function library is loaded in the context of the realtime environment (Page 70).
Trace	on	Specifies the trace function in the CPU function library. In this case, the CPU function library requires 32 KB if memory as an additional trace buffer. A "GetTrace" function block is created by default for use in a STEP 7.
	off	A "GetTrace" function block is created. The trace buffer contains only one trace entry with the contents: trace is off.
HeapSize	[4...<Available CPU memory (Page 130)>]k	Specifies a memory in KB that can be used as heap for these realtime applications.
HeapMaxBlockSize	[8...<HeapSize>]	Specifies the maximum memory size in bytes that can be allocated at one time.
SyncCallParallelCount	[1...9] Default=3	If a optional parameter and defines the maximum number of parallel calls in this CPU function library. The size of the memory which is reserved for calls in this CPU function library: SyncCallParallelCount * (SyncCallStackSize + SyncCallDataSize)
SyncCallStackSize	[1...1024]k Default=32k	Is a optional parameter and defines the size of the thread stack for a call in this CPU function library. Each new call receives its own stack memory.
SyncCallDataSize	[1...1024]k	Is a optional parameter and defines the size of the data area for a call in this CPU function library. The data area contains IN, INOUT and OUT parameters. Each new call receives its own stack memory.
	Default=auto	The required data size is automatically calculated by the code generator. With an ODK_CLASSIC_DB, 65 KB is applied.
STEP7Prefix	<String>	Describes the string that precedes your functions and is shown after importing the SCL file in STEP 7. The following characters are allowed: {A...Z, a...z, 1...9, -, _} The project name is entered without spaces by default.

6.1.5 Environment for loading or running the CPU function library

When the SCL file is imported into STEP 7 as an external source, the ODK instructions are created in the selected directory in STEP 7. You can load up to 32 CPU function libraries.

You can load and run your CPU function library in the context of the realtime environment:

Realtime environment

Add the following line of code in your <project>.odk file to use the CPU function library in the context of the realtime environment:
`Context=realtime`

In this context, the CPU function library is not running in a host process at the Windows end, but instead in the realtime environment. Because the CPU function library is loaded synchronously, it should be loaded in a startup OB (e.g. OB 100).

The number of loadable CPU function libraries (Page 130) is limited in the context of the realtime environment.

If the CPU function library has to be loaded in a cyclic OB (for example, OB 1), note the following loading times:

CPU	Small SO file → Loading time	Large SO file → Loading time
CPU 1505SP	0.5 MB → 20 ms	3 MB → 70 ms
CPU 1507S (with SSD)	0.5 MB → 20 ms	5 MB → 100 ms

Determining the size of the CPU function library in the CPU memory

To determine the required size of the CPU function library in the CPU memory, follow these steps:

1. Open a command line dialog.
2. Enter the following path from the ODK installation folder (the appended option "-l" is a lower-case "L"): eclipse\ build_tools\x86_64_gcc_pc_elf_11.3.0\bin\x86_64-pc-elf-readelf.exe "*<StorageLocation\File.so>*" -l

   You can see the size of your CPU function library under the heading "Program Headers" in the "MemSiz" column.

   Additional administrative memory is required for each CPU function library in addition to the amount specified here. The administrative memory can be calculated as follows:

   Administrative memory = SyncCallParallelCount * (SyncCallStackSize + SyncCallDataSize)

6.1.6 Defining functions and structures of a CPU function library

6.1.6.1 Defining functions a CPU function library

Functions

Functions are defined by the following general lines of code:

```
ODK_RESULT <FunctionName>
([<InOut identifier>] <data type> <tag name>, etc.);
```

The <project>.odk file contains an example function description by default. You can change this description and/or add more function descriptions.

```
ODK_RESULT MyFunc1([IN] INT param1, [OUT] INT param2);
```

Syntax rules for functions

The following syntax rules apply to functions within the <project>.odk file:

- Note that the function names are case-sensitive.
- You can split function definitions into several lines.
- End a function definition with a semicolon.
- TAB and SPACE are allowed.
- Do not define a tag name in a function twice.
- Do not use any keywords for the programming language that is used (for example "EN / ENO" as parameter name)
- Use ODK_RESULT only for the return values of the function.
- The tag name must start with a letter or an underscore.
- Illegal function names are displayed during generation in the development environment.
- The following names are not allowed in combination of *<STEP 7Prefix>* and <function name>: ODK_Load, ODK_Unld, ODK_ExcA, ODK_ExcS

<FunctionName>

Function names are valid with the syntax and character restrictions of the used programming language.

<InOut-Identifier>

There are three defined InOut-Identifiers. Use these in the following order: [IN], [OUT], [INOUT]

- [IN]: Specifies an input tag. The tag is copied to the function when it is called. This is constant and cannot be changed.
- [OUT]: Specifies an output tag. The tag is copied back after the function has been completed.
- [INOUT]: Specifies an input and output tag. The tag is copied to the function when it is called. This is not constant and can be changed. The tag is copied back after the function has been completed.

<DataType>

The data type defines the type of a tag. The following tables define the possible data types and their method of representation in C++ or STEP 7:

Elementary data types:

ODK data type	SIMATIC data type	C++ data type	Description
ODK_DOUBLE	LREAL	double	64-bit floating point, IEEE 754
ODK_FLOAT	REAL	float	32-bit floating point, IEEE 754
ODK_INT64	LINT	long long	64-bit signed integer
ODK_INT32	DINT	long	32-bit signed integer
ODK_INT16	INT	short	16-bit signed integer
ODK_INT8	SINT	char	8-bit signed integer
ODK_UINT64	ULINT	unsigned long long	64-bit unsigned integer
ODK_UINT32	UDINT	unsigned long	32-bit unsigned integer
ODK_UINT16	UINT	unsigned short	16-bit unsigned integer
ODK_UINT8	USINT	unsigned char	8-bit unsigned integer
ODK_LWORD	LWORD	unsigned long long	64-bit bit string
ODK_DWORD	DWORD	unsigned long	32-bit bit string
ODK_WORD	WORD	unsigned short	16-bit bit string
ODK_BYTE	BYTE	unsigned char	8-bit bit string
ODK_BOOL	BOOL	unsigned char	1-bit bit string, remaining bits (1..7) are empty
ODK_LTIME	LTIME	long long	64-bit during in nanoseconds
ODK_TIME	TIME	long	32-bit during in milliseconds
ODK_LDT	LDT	unsigned long long	64-bit date and time of the day in nanoseconds since 01/01/1970 00:00
ODK_LTOD	LTOD	unsigned long long	64-bit time of the day in nanoseconds since midnight
ODK_TOD	TOD	unsigned long	32-bit time of the day in milliseconds since midnight
ODK_CHAR	CHAR	char	8-bit character

Complex data types:

ODK data type	SIMATIC data type	C++ data type	Description
ODK_DTL	DTL	ODK_DTL (struct)	Structure/class for date and time
ODK_S7STRING	STRING	unsigned char	Character string: • For SIMATIC and C++: 8-bit character with length max. and act. (2xUSINT) • For other languages: native
ODK_CLASSIC_DB	VARIANT	ODK_CLASSIC_DB (struct)	Classic DB (global or based on UDT)
[ ]	ARRAY	[ ]	Range of same data types. You can use all data types as an array except ODK_CLASSIC_DB.

User-defined data types:

User-defined data types (UDT) include structured data, especially the names and data types of this component and their order.

A user-defined data type can be defined in the user interface description with the keyword "ODK_STRUCT".

Example

```
ODK_STRUCT <StructName>
{
 <DataType> <TagName>;
 ...
};
```

The following syntax rules apply to the structure:

- You can divide the structure into multiple lines.
- The structure definition must end with a semicolon.
- Any number of tabs and spaces between the elements is permitted.
- It is not permitted to use any keywords for the generated language used (for example "en / eno" as tag name).

You can create additional structures within a structure.

<StructName>

Structure names are valid with the syntax and character restrictions of the programming language and as defined for tag definitions in STEP 7.

In STEP 7, the structure name is extended with the STEP 7 prefix.

<TagName>

Tag names are subject to the syntax and character restrictions of the programming language.

Example

The following code example explains the definitions of functions and structures. Sort the parameters by: IN, OUT, INOUT.

```
//INTERFACE
…
ODK_STRUCT MyStruct
  {
    ODK_DWORD myDword;
    ODK_S7STRING myString;
  };
ODK_RESULT MyFct([IN] MyStruct myInStruct
                ,[OUT] MyStruct myOutStruct);
```

6.1.6.2 Use of ODK_CLASSIC_DB as parameter

The ODK_CLASSIC_DB data type may only be used with the InOut-Identifier [IN] and [INOUT]. If a parameter of data type ODK_CLASSIC_DB with InOut-Identifier [IN] or [INOUT] is used, no other parameters, regardless of the data type, can be used with the same InOut-Identifier.

Example

```
// INTERFACE
...
// OK:
ODK_RESULT MyFunc1([IN] ODK_CLASSIC_DB myDB);
ODK_RESULT MyFunc2([IN] ODK_CLASSIC_DB myDB1, [INOUT] ODK_CLASSIC_DB
myDB2);
//
// NOT OK (Code Generator will throw an error):
// ODK_CLASSIC_DB not permitted for [OUT]
ODK_RESULT MyFunc3([OUT] ODK_CLASSIC_DB myDB);
// if ODK_CLASSIC_DB is used for [IN], no other [IN] parameter may
be
// defined in this function
ODK_RESULT MyFunc4([IN] ODK_CLASSIC_DB myDB, [IN] ODK_INT32 myint);
```

Application example for C++

```
#include "ODK_CpuReadData.h"
...
ODK_RESULT MyFunc1 (const ODK_CLASSIC_DB& myDB)
{
    CODK_CpuReadData myReader(&myDB);
    ODK_INT32 myInt1, myInt2;

    myReader.ReadS7DINT(0, myInt1);
    myReader.ReadS7DINT(4, myInt2);

    return myInt1 + myInt2;
}
```

In order to access the data type ODK_CLASSIC_DB within a user function, the helper functions (Page 139) of the following classes are available:

- Class "CODK_CpuReadData"
- Class "CODK_CpuReadWriteData"

6.1.6.3 Handling strings

You can define a maximum length for strings (String or WString). Define the maximum number of characters in square brackets directly after the data type:

- ODK_S7STRING[30] or
- ODK_S7WSTRING[1000]

Without limitation, a string has a default length of 254 characters.

In order to access the data types ODK_S7STRING or ODK_S7WSTRING within a user function, the string helper functions (Page 139) are available:

Example

```
//INTERFACE
…
ODK_RESULT MyFct(
    [IN]    ODK_S7STRING      myStrHas254Chars
  , [OUT]   ODK_S7STRING[10]  myStrHas10Chars
  , [INOUT] ODK_S7STRING[20]  myStrArrayHas20Chars5Times[5]);
```

If you use [INOUT], you can set the string with a length that differs from the [INOUT of the function block in STEP 7.

Note

Compatibility

If you use the "WSTRING" data type with more than 253 characters in a project, create a new project with ODK version ≥ V2.5 SP1.

The characters are not read/written correctly with a project created with ODK version < V2.5 SP1.

6.1.6.4 Definition of the <Project>.odk file

The function prototypes and function blocks are generated based on the selected parameters in the <project>.odk file. Define the <project>.odk file for this.

By default, the <project>.odk file contains the following:

- Description

  The possible data types that are used for the interface are described in comment lines. This simplifies the definition of the correct tag type for your task.

- Context=realtime

  The CPU function library is loaded in the context of the realtime environment.

- Trace=on

  Specifies the trace function in the CPU function library. A "GetTrace" function block is created by default for use in a STEP 7.

  When you define the "ODK_TRACE" instruction (Page 97), it is also compiled and executed. When you define the parameter Trace=on in the <project>.odk file, the instruction is automatically defined with the following code:

  ```
  #define ODK_TRACE(msg, ...);
  ```

  Example: ODK_TRACE("number=%d", 13);

  Calling the instruction creates an entry in the trace buffer.

- HeapSize

  Specifies a memory in KB that can be used as heap for these realtime applications.

- HeapMaxBlockSize

  Specifies the maximum memory size in bytes that can be allocated at one time.

- STEP7Prefix="<project>"

  Sets a string for the SCL generation in front of the functions of the CPU function library. This is visible in STEP 7. You can change the parameter. The string length of the prefix including function name must not exceed 125 characters (e.g. ODK_App_SampleFunction).

- "SampleFunction" function definition

  You can change this default function as you wish in the <project>.odk file and add more functions. The string length may not exceed a length of 125 characters. The associated function is located in the CPP file.

Example

```
//INTERFACE
Context=realtime
Trace=on
HeapSize=4k
HeapMaxBlockSize=1024
STEP7Prefix=ODK_App

 /*
* Elementary data types:
*
*  ODK_DOUBLE      LREAL    64-bit floating point, IEEE 754
*  ODK_FLOAT       REAL     32-bit floating point, IEEE 754
*  ODK_INT64       LINT     64-bit signed integer
*  ODK_INT32       DINT     32-bit signed integer
*  ODK_INT16       INT      16-bit signed integer
*  ODK_INT8        SINT     8-bit signed integer
*  ODK_UINT64      ULINT    64-bit unsigned integer
*  ODK_UINT32      UDINT    32-bit unsigned integer
*  ODK_UINT16      UINT     16-bit unsigned integer
*  ODK_UINT8       USINT    8-bit unsigned integer
*  ODK_LWORD       LWORD    64-bit bit string
*  ODK_DWORD       DWORD    32-bit bit string
*  ODK_WORD        WORD     16-bit bit string
*  ODK_BYTE        BYTE     8-bit bit string
*  ODK_BOOL        BOOL     1-bit bit string
*  ODK_LTIME       LTIME    64-bit duration in nanoseconds
*  ODK_TIME        TIME     32-bit duration in milliseconds
*  ODK_LDT         LDT      64 bit date and time of day
*                           in nanoseconds
*  ODK_LTOD        LTOD     64 bit time of day in nanoseconds
                            since midnight
*  ODK_TOD         TOD      32 bit time of day in milliseconds
                            since midnight
*  ODK_DTL         DTL      structure for date and time
*  ODK_CHAR        CHAR     8 bit character
*  ODK_S7STRING    STRING   character string with 8-bit characters
*  ODK_CLASSIC_DB  VARIANT  classic DB (global or based on UDT)
*  []              ARRAY    field of this datatype
* User Defined Datatype:
*  ODK_STRUCT      UDT      user defined structure
* Return data type:
*  ODK_RESULT      0x0000 - 0x6FFF function succeeded
*                                  (ODK_SUCCESS = 0x0000)
*                  0xF000 - 0xFFFF function failed
*                                  (ODK_USER_ERROR_BASE = 0xF000)
*/

ODK_RESULT SampleFunction([IN]    ODK_INT32    myInt
                        , [OUT]   ODK_BOOL     myBool
                        , [INOUT] ODK_DOUBLE   myReal);
```

6.1.6.5 Modifying the <Project>.odk file

The following example shows you how you can change the <Project>.odk file to suit your needs.

```
//INTERFACE
Context=realtime
Trace=on
HeapSize=4k
HeapMaxBlockSize=1024
STEP7Prefix=ODK_SampleApp_

ODK_RESULT GetString ([OUT]    ODK_S7STRING myString);

ODK_RESULT Calculate ([IN]     ODK_INT64   In1,
                      [IN]     ODK_DOUBLE  In2,
                      [OUT]    ODK_FLOAT   Out1,
                      [OUT]    ODK_INT32   Out2,
                      [INOUT]  ODK_BYTE    InOut1[64],
                      [INOUT]  ODK_BYTE    InOut2[64]);
```

6.1.6.6 Comments

Comments are started with a double slash "//" and end automatically at the end of the line.

Alternatively, you can limit comments by /* *<comment>* */, which enables new lines in a comment. Characters after the end of the comment identifier "*/" are further processed by the code generator.

Comments for Visual Basic are marked with a apostrophe.

Comments for functions and structures

You place comments on functions and structures directly in front of the functions/structures.

These comments are transferred to the ODK_Functions.h and <project>.scl files.

In the <project>.scl file, the comments are copied to the block properties and duplicated in the code area of the function.

Observe the following rules:

- Comments for functions and structures must be located directly in front of the functions/structures (without blank line).
- The end of the comment is located in front of the ODK_RESULT or ODK_STRUCT keyword.
- You can use both identifiers "//" and "/* */" but not in combination within a comment.

Example

```
// this comment did not appear in MyStruct, because of the empty
line.

// comment MyStruct
// ...
ODK_STRUCT MyStruct
{
  ODK_DWORD     myDword;
  ODK_S7STRING  myString;
};

/*
comment MyFct
...
*/
ODK_RESULT MyFct([IN] MyStruct myInStruct
                ,[OUT] MyStruct myOutStruct);
```

Comments for tags in functions and structures

Comments for function and structure tags are placed directly in front of or behind the tag.

These comments are transferred to the ODK_Functions.h/ and <project>.scl files.

The following rules apply to comments **in front of** tags:

- Comments must be located directly in front of the tag (without blank line)
- The end of the comment is the <InOut-Identifier> of the tag

The following rules apply to comments **after** tags:

- Comments must be located after the tag name (without blank line)

The following general rules apply to comments for tags:

- You can use both identifiers "//" and "/* */" but not in combination within a comment.
- In the header file, the same comment identifier is used ("//" or "/* */").

Example

```
ODK_STRUCT MyStruct
{
  // comment myDword BEFORE definition
  ODK_DWORD    myDword;

  ODK_S7STRING myString; /* comment myString AFTER definition */
};

ODK_RESULT MyFct([IN]  MyStruct myInStruct    // comment
                                              // myInStruct ...
                                              // ... "second line"
               , [OUT] MyStruct myOutStruct); /* comment
                                                 myOutStruct ...
                                                 ...
                                               */
```

6.1.7 Implementing functions

6.1.7.1 General notes

This section provides an overview of the basic topics relating to the implementation of functions in a realtime environment.

- The function call is limited in time

  Since the function is called synchronously, the function call must be adjusted to the timing of the cycle.

- Trace functionality

  ODK provides a trace function (Page 97) to check variables or the execution of functions in the realtime environment.

- The execution of synchronous functions can be interrupted by higher priority OBs (Page 93) running in the same CPU.

- Application size

  The number of loadable CPU function libraries (Page 70) is limited in the context of the realtime environment.

- C++ Runtime library

  Functions that need operating system functionality (threading) cannot be used

6.1.7.2 Callback functions

The project for the realtime CPU function library contains a CPP file (**execute file:** <project>.cpp) to define your functions. This CPP file contains functions filled by default. You do not necessarily have to fill these with additional user code to be usable. However, neither may the functions be deleted under any circumstances.

The empty function has the following code (using the "OnLoad()" function as an example):

```
ODK_RESULT OnLoad (void)
{
  // place your code here
  return ODK_SUCCESS;
}
```

You can define the following functions in the CPP file:

- OnLoad(): Called after loading the CPU function library
- OnUnload(): Called before unloading the CPU function library
- OnRun(): Called when the CPU changes to RUN mode after the OnLoad() function
- OnStop(): Called when the CPU changes to the STOP mode and before the function OnUnload()

  The OnStop() function is terminated if the execution takes longer than 50 ms when CPU changes to STOP mode.

"OnLoad()" and "OnUnload()" function

The functions have a return value of type "ODK_RESULT" and typically provide information about the status of the "ODK_SUCCESS" value.

The following return values are possible:

Return value for "ODK_RESULT"	Description
ODK_SUCCESS = 0x0000	Return value following a successful execution of the "OnLoad()" or "OnUnload()" function
0x0001 – 0xEFFF	Invalid values (system-internal)
0xF000 – 0xFFFF ODK_USER_ERROR_BASE = 0xF000	You can define your own return values. The loading stops and the CPU function library unloads for the "OnLoad()" function. The CPU function library within the specified value range is still unloaded for the "OnUnload()" function.

"OnRun()" and "OnStop()" function

The functions have a return value of type "ODK_RESULT" and typically provide information about the status of the "ODK_SUCCESS" value.

The following return values are possible:

Return value for "ODK_RESULT"	Description
ODK_SUCCESS = 0x0000	Default return value for a successful execution of the function "OnRun()" or "OnStop()"
0x0001 – 0xFFFF	Direct feedback to the user program is not possible because these functions are not called directly by the user at RUN/STOP mode transitions.

6.1.7.3 Implementing custom functions

Once you have defined the ODK interface in the <project>.odk file, you must edit the functions of the CPU function library in the Project Source file.

Procedure

To edit the function of a CPU function library, follow these steps:

1. To generate the function prototypes, execute the build.
2. Open the project source file, or create a custom source file if necessary.
3. Transfer the function prototypes from <ODK_Functions.h>/<OdkFunctions.cs/vb> to the source file.

Note

Use the function prototype macro to transfer the step 3 in the future when there is a change to the function parameters.

4. Edit the code of your CPU function library in the execute file.

CPU function library

The execute file contains a schematically represented function description by default. You can change this description with corresponding changes in the <project>.odk file and/or add more function descriptions.

Execute file based on C++ example

```
#include "ODK_Functions.h"

EXPORT_API ODK_RESULT OnLoad (void)
{
    return ODK_SUCCESS;
}
EXPORT_API ODK_RESULT OnUnload (void)
{
    return ODK_SUCCESS;
}
EXPORT_API ODK_RESULT OnRun (void)
{
    return ODK_SUCCESS;
}
```

```
EXPORT_API ODK_RESULT OnStop (void)
{
    return ODK_SUCCESS;
}
ODK_RESULT SampleFunction( const ODK_INT32& myInt,
                                 ODK_BOOL& myBool,
                                 ODK_DOUBLE& myReal)
{
    return ODK_SUCCESS;
}
```

6.1.7.4 Dynamic memory management

Introduction

ODK objects work with a dynamic memory management (heap). The following instructions and functionalities are supported by using the dynamic memory management:

- The new/delete and malloc/free instructions.
- STL (Standard Template Library)
- Software exceptions

The default setting for the heap size is 4 KB. The heap size can be from 4 KB up to the available memory of the CPU (Page 130). You change the heap size in the <project>.odk file using the following parameters:

- HeapSize
- HeapMaxBlockSize

Special features

Because the used memory area (heap) has been optimized with regard to realtime and cyclic processing, it has some special features:

- Blocks can only be allocated up to a specified size during the compiling time of the ODK object.

  Note

  You can specify the maximum block size with the HeapMaxBlockSize parameter in <project>.odk. However, this has an effect on the global memory use for CPU function libraries, because the management information of the following memories is required in addition to the actual heap:

  size_heap_admin_data = HeapMaxBlockSize * 3

  Example: Therefore, with a maximum block size of 100 KB, this project needs 300 KB of global data in addition to the heap. This data is used for heap administration.

  You can find additional information under Environment for loading or running the CPU function library (Page 70).

- Blocks can initially be requested in any size. When the blocks are released again, they are entered in free lists. There is a free list in each case for all possible block sizes (up to HeapMaxBlockSize) so that later allocations can be performed in constant time.

  There is, however, no merging of neighboring released blocks to form a larger block.

  This means continuously recurring requests can be met faster than constantly different requests.

  Example: The user allocates only blocks with 8 bytes until the heap is full. The user then releases everything again so that the heap is completely empty. An allocation of a block with 16 bytes is then no longer possible, however, because all free blocks are entered in the free list for 8 bytes and merging is not possible.

Example

```
#include <assert.h>
#include <exception>
#include <vector>
…
   // check parameter
   assert (NULL != myPointer);

   // allocate heap memory with malloc()
   char* p1 = (char*) malloc(32);
   if (NULL == p1)
   {
     ODK_TRACE("ERROR: malloc() failed");
   }
   else
   {
     ODK_TRACE("malloc() done");
     // free allocated memory
     free(p1);
```

```
  ODK_TRACE("free() done");
}

// allocate heap memory with new()
char* p2 = NULL;
try
{
  p2 = new char [64];
  ODK_TRACE("new done");
  // delete allocated memory
  delete[] p2;
  ODK_TRACE("delete done");
}
catch (std::exception& e)
{
  ODK_TRACE("exception: %s", e.what());
}
std::vector<int> vec; // empty vector of ints
```

6.1.7.5 Debug (Test)

You have the possibility to write a custom test to debug the CPU function library for the realtime environment in a Windows environment. This will ensure the quality of the code.

Requirements

You need an Internet connection for this procedure.

You need administrator rights for this procedure.

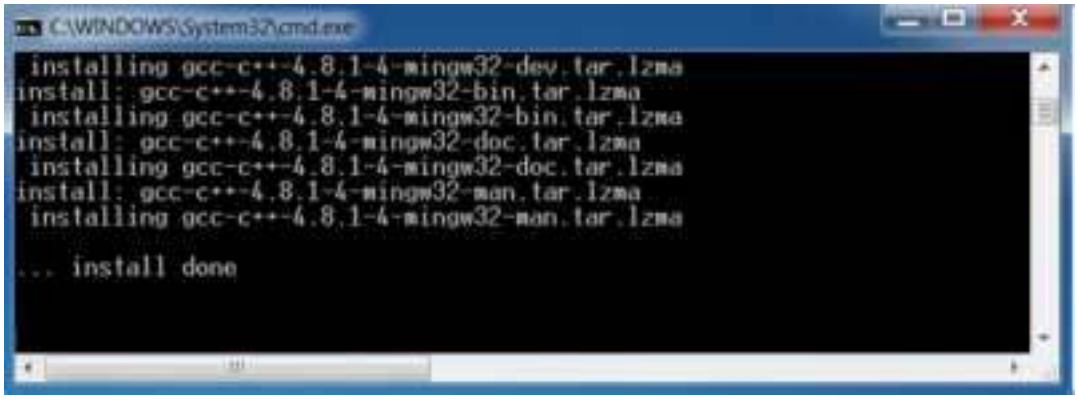
Procedure before the first debug process

To perform a test on a CPU function library for the realtime environment in a Windows environment, perform the following once:

1. Close Eclipse.
2. Open the "bin" folder of your ODK installation.

3. Run the "MinGW32_Install.cmd" file with the "Run as administrator" command from the shortcut menu.

   A text editing dialog opens. The Windows prompt installs all necessary components.



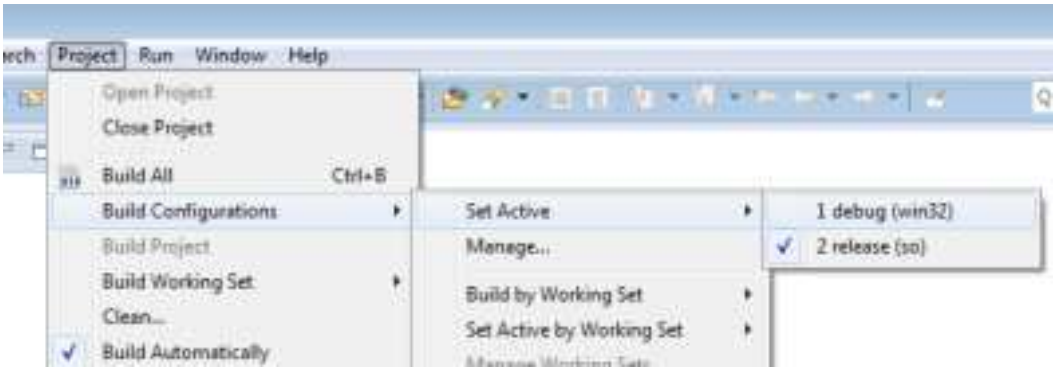
4. Click on any button.

   MinGW32 is installed.

Basic procedure

To perform the test, proceed as follows:

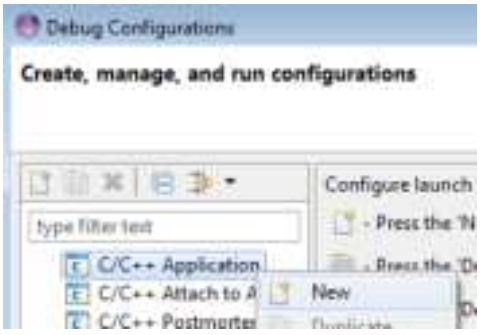
1. Open your project in Eclipse.
2. Change the debug environment to "Windows". To do this, select the "debug (win32)" option in menu "Project > Build Configurations > Set Active".



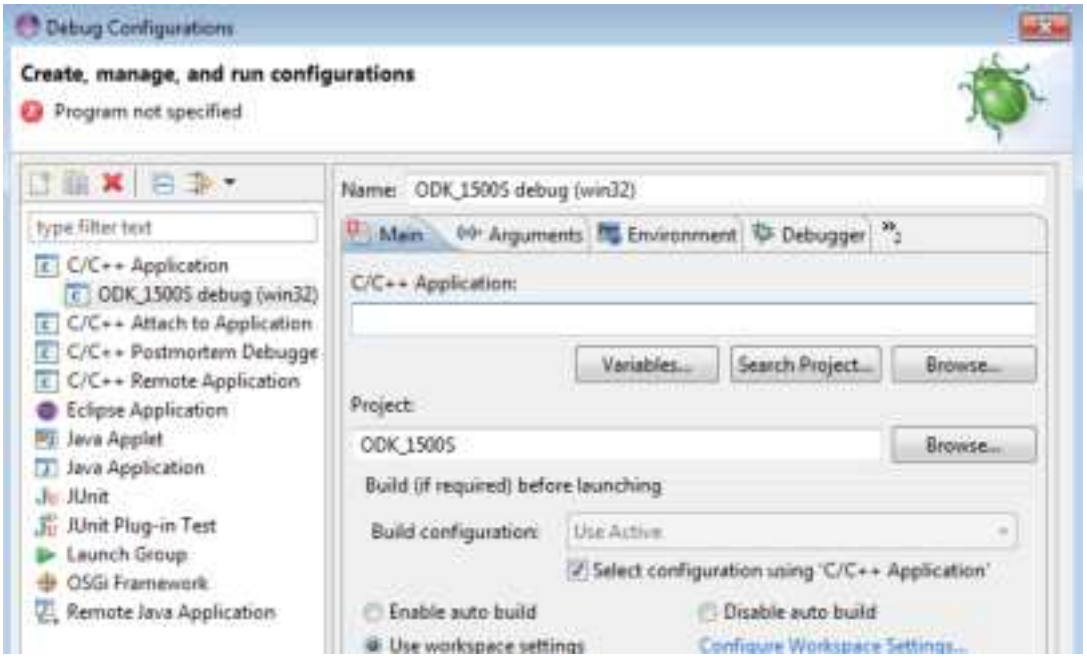
3. Create the project as debug version. To do so, select the "Build Project" command in the "Project " menu.
4. If you debug the project for the first time, you must now set the debug configuration. Otherwise, continue with step 8.
5. To do this, select the "Debug Configurations" command in the "Run" menu.

   The "Debug Configurations" dialog opens.

6. To create a new application, select the entry "C/C++ Application" and select the "New" command in the context menu.



7. Configure your test environment.
8. Click the "Search Project" button to select your application.



9. Start the debug process by clicking the "Debug" button.
10. If you want to debug your project again, select the "Local C/C++ Application" command in the menu "Run > Debug as".

Result

Eclipse suggests a change in the debug perspective.

The test code is executed. The test code for the test is compiled only in the debug environment and is implemented in the "main()" function. This function is located in the <project>.cpp file.

The "main()" function offers you the following possibilities:

- Test data are provided and results can be reviewed.
- You can monitor tags of the function.
- You can use breakpoints to check the execution.

Test code

The following sample code shows the default contents of the "main()" function.

```
/*
 * main() is defined for windows debugging, only.
 * Therefore all automatically invoked functions
 * (OnLoad, OnRun, OnStop, OnUnload) have to be called manually.
 */
#ifdef _DEBUG
int main (int argc, char* argv[])
{
    ODK_RESULT ret = ODK_SUCCESS;
    ret = OnLoad();
    // error handling
    ret = OnRun();
    // error handling

    // place your test code here

    ret = OnStop();
    // error handling
    ret = OnUnload();
    // error handling
    return ret;
}
#endif // _DEBUG
```

6.2 Transferring a CPU function library to the target system

Procedure

Manually transfer the SO file to the target system. Use the file explorer of the web server of the CPU to transfer the CPU function library.

To transfer an SO file, follow these steps:

1. Enable the Web server in your STEP 7 project.
2. Open the web server of the CPU in the browser.
3. Open the "Filebrowser" menu.
4. Open the following directory as the storage location for the CPU function libraries: \ODK1500S\

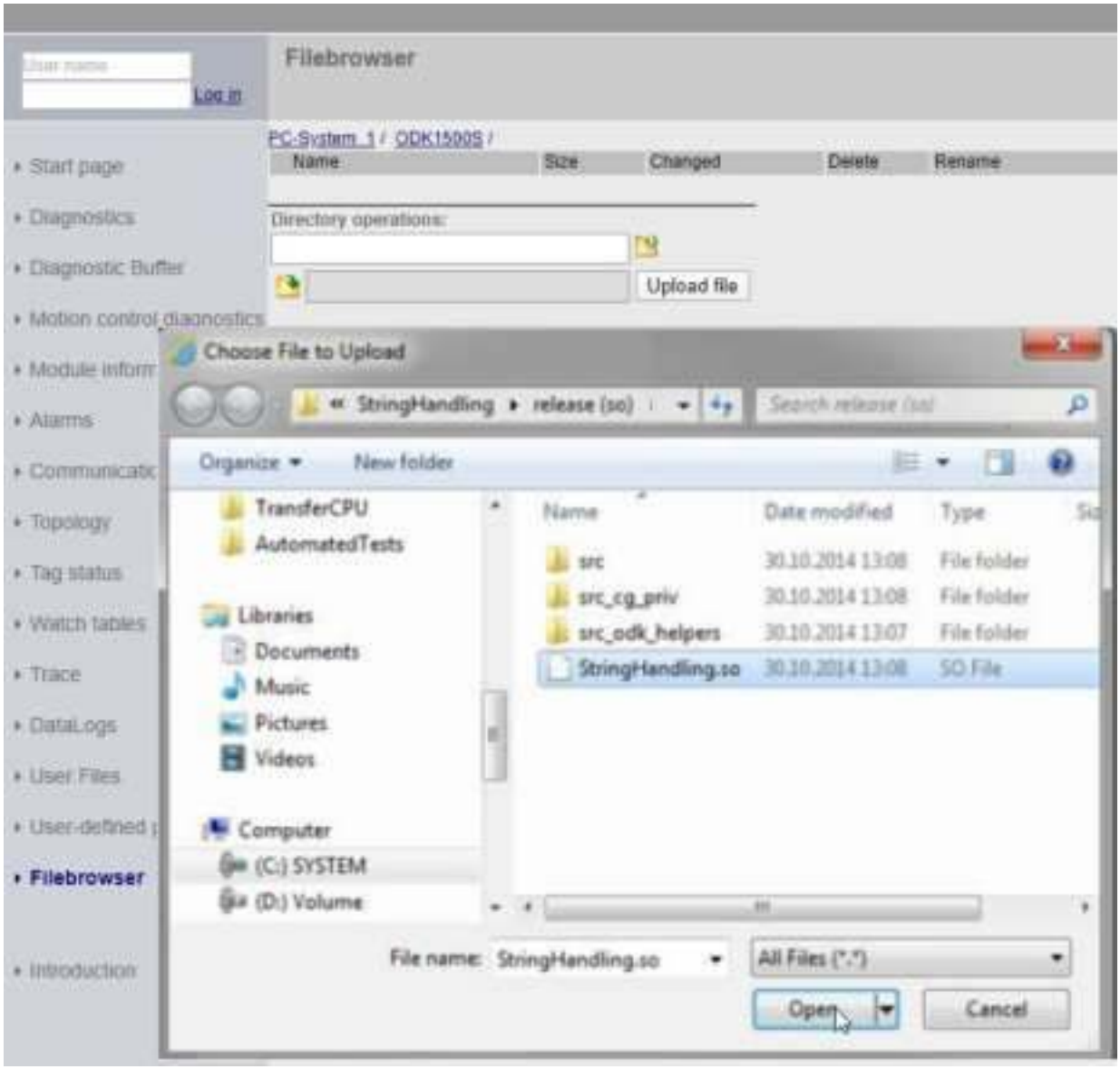


Figure 6-3 Transferring the SO file via the file explorer from the web server of the CPU

5. Click the "Browse" button.

6. Navigate in the file system to the SO file or copy the location from the properties of the SO file in Eclipse.
7. Confirm the transfer of the SO file to the web server of the CPU by pressing the "Load File" button.

Result

The SO file is transferred to the load memory of the CPU.

After a successful transfer, the SO file is loaded by calling the "<*STEP7Prefix*>_Load" instruction.

6.3 Importing and generating an SCL file in STEP 7

When generating the project data, the following files are created:

- SCL file for importing into STEP 7
- All files depending on the configuration, e.g. SO file

If STEP 7 is installed on another PC as the development environment, you must transfer the generated SCL file to the PC where the STEP 7 is installed.

Requirements

The project data were generated.

Procedure

To import and compile the SCL file, follow these steps:

1. Start STEP 7.
2. Open your project.
3. Select the project view.
4. Select the CPU in the project tree.
5. Select the "External Sources" subfolder.

   The "Open" dialog box opens.
6. Navigate in the file system to the SCL file that was created during generation of the project data or copy the storage location from the properties of the SCL file to Eclipse.
7. Confirm your selection with "Open".

   The SCL file is imported. After completion of the import process, the SCL file is displayed in the "External Sources" folder.
8. Compile the SCL file before you use the blocks in your project.
9. To do this, select the SCL file in "External sources" subfolder.
10. Select the "Generate blocks from source" command in the shortcut menu.

Result

STEP 7 creates the S7 blocks based on the selected SCL file.

The "GetTrace" function block, which makes it possible to read the trace buffer, is created by default.

The created blocks are now automatically displayed in the "Program blocks" folder below the selected CPU in the project tree. You can load the function blocks during the next download to the target device.

6.4 Executing a function

6.4.1 Loading functions

Introduction

Regardless of the context in which the CPU function library is running, the loading procedure consists of the following steps:

- Call the "<*STEP7Prefix*>_Load" instruction in the STEP 7 user program.
- The loading process takes place synchronously

  To avoid influencing the cycle time, load the CPU function library in startup OB (e.g. OB 100).

  If the CPU function library has to be loaded in a cyclic OB (for example, OB 1), note the following loading times:

CPU	Small SO file → Loading time	Large SO file → Loading time
CPU 1505SP	0.5 MB → 20 ms	3 MB → 70 ms
CPU 1507S (with SSD)	0.5 MB → 20 ms	5 MB → 100 ms

- As soon as the "<*STEP7Prefix*>_Load" instruction returns after the first call, the CPU function library is loaded.

Note

Loading the same CPU function libraries with a modified <project>.odk file

When you load a CPU function library and subsequently change the <project>.odk file, we recommend that you unload your CPU function library first before you load the newly generated CPU function library. If the "<*STEP7Prefix*>_Unload" instruction is not executed, both CPU function libraries are in the memory. This can lead to insufficient memory being available for the CPU.

"*<STEP7Prefix>*_Load" instruction

A CPU function library is loaded by calling the "*<STEP7Prefix>*_Load" instruction in the STEP 7 user program.

***<STEP7Prefix>*_Load**	
REQ	DONE
	BUSY
	ERROR
	STATUS

The following table shows the parameters of the instruction "*<STEP7Prefix>*_Load":

Section	Declaration	Data type	Description
Input	REQ	BOOL	A rising edge activates the loading of the CPU function library.
Output	DONE	BOOL	Indicates that the instruction has finished loading the CPU function library.
Output	BUSY	BOOL	Indicates that the instruction is still loading the CPU function library.
Output	ERROR	BOOL	Indicates that an error occurred during the loading of the CPU function library. STATUS gives you more information about the possible cause of the error.
Output	STATUS	INT	Provides information about possible sources of error, if an error occurs during the loading of the CPU function library.

Input parameters

An edge transition (0 to 1) at the "REQ" input parameter starts the function.

Output parameters

The following table shows the information that is returned after loading.

DONE	BUSY	ERROR	STATUS	Meaning
0	0	0	0x7000 =28672	No active loading
1	0	0	0x7100 =28928	CPU 1500 V2.0 and later: CPU function library is already loaded.
1	0	0	0x0000 =0	Loading was performed successfully.
0	0	1	0x80A4 =-32604	CPU function library could not be loaded.
			0x80C3 =-32573	CPU function library could not be loaded. The CPU currently does not have enough resources. Unload the CPU function library before you load a new CPU function library or restart the CPU.
			0x8090 =-32624	CPU function library could not be loaded. An exception occurred during execution of the "OnLoad()" function.
			0x8092 =-32622	CPU function library could not be loaded because the library name is invalid.
			0x8093 =-32621	CPU function library could not be loaded because the CPU function library could not be found. Check the file name and path of the file.

DONE	BUSY	ERROR	STATUS	Meaning
			0x8095 =-32619	CPU function library could not be loaded due to the following reasons: • The SO file is not a CPU function library. • The CPU does not support the utilized ODK version.
			0x8096 =-32618	The CPU function library could not be loaded because the internal identification is already being used by another loaded CPU function library.
			0x8097 =-32617	CPU 1500 V1.8 and earlier: CPU function library is already loaded.
			0x8098 =-32616	CPU function library could not be loaded because the CPU function library is currently being unloaded.
			0x8099 =-32615	Unable to load the CPU function library because the instruction was not called in an OB with lowest priority. Use a Startup OB (e.g. OB100) or a Program cycle OB (e.g. OB1).
			0x809B =-32613	CPU 1500 V2.0 and later: The CPU function library could not be loaded and returns an invalid value (the values 0x0000 and 0xF000 - 0xFFFF are allowed)
			0xF000 – 0xFFFF =-4096 – -1	CPU 1500 V2.0 and later: CPU function library could not be loaded. An error occurred during execution of the "OnLoad()" function.

6.4.2 Calling functions

Introduction

Once the CPU function library is loaded, you can execute C functions via your STEP 7 user program. This call is made from the corresponding "<*STEP7Prefix*>SampleFunction" instruction.

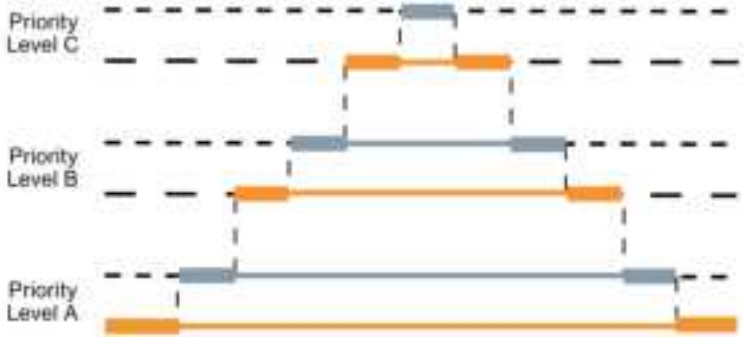


Figure 6-4 Calling functions

The execution of synchronous functions can be interrupted by higher priority OBs running in the same CPU.

- Call another ODK function
- Call the same function

  Therefore, when creating your CPU function library make sure that the function calls are implemented as re-entrant or avoid parallel execution.

If you implement more than the number of parallel calls set in "SyncCallParallelCount", the function returns the status 0x80C3.

"*<STEP7Prefix>*SampleFunction" instruction

A CPU function library is called by the "*<STEP7Prefix>*SampleFunction" instruction.

***<STEP7Prefix>*SampleFunction**	
myInt	STATUS
myReal	myBool

The following table shows the parameters of the instruction "*<STEP7Prefix>*SampleFunction":

Section	Declaration	Data type	Description
Automatically generated parameters			
Output	STATUS	INT	This output value provides information about possible sources of error, if an error occurs during the execution of the CPU function library.
User-defined parameter			
Input	myInt		User-defined input tags
InOut	myReal		User-defined input-output tags
Output	myBool		User-defined output tags

Output parameters

The "*<STEP7Prefix>*SampleFunction" instruction only has the "STATUS" output parameter.

The following table shows the information for the output parameter returned after execution.

STATUS	Meaning
0x0000 – 0x6FFF =0 – 28671	Function has been executed and returns a value between 0x0000 and 0x6FFF. (ODK_SUCCESS = 0x0000)
0x80A4 =-32604	CPU function library could not be executed for the following reasons: • A stack overflow was detected after execution of the function. To avoid sequential errors, unload the CPU function library. The developer of the CPU function library must ensure that the stack is not overwritten. • The "<STEP7Prefix>_Unload" instruction was executed during a function execution. The execution of the function was interrupted and terminated immediately. No return value is sent to the CPU. Wait until the "<STEP7Prefix>_Unload" instruction has ended. Then load the CPU function library again.

STATUS	Meaning
0x80C3 =-32573	CPU function library could not be executed. The CPU currently does not have enough resources. Pay attention to the maximum number of parallel calls (SyncCallParallelCount).
0x8090 =-32624	CPU function library could not be executed. An exception occurred during execution. Each unhandled exception reduces the available heap size. An unhandled exception can damage the CPU function library and lead to this no longer being used for further calls. The CPU function library must be unloaded. The developer of the CPU function library must handle the exception and deliver an application-specific error value.
0x8091 =-32623	CPU function library could not be executed. A "STOP" occurred during the function call.
0x8096 =-32618	CPU function library could not be executed because the CPU function library was not loaded or unloading is not yet finished.
0x8098 =-32616	CPU function library could not be executed because the CPU function library is different than the ODK instructions (FBs) in STEP 7: • older • newer • different parameters
0x8099 =-32615	CPU function libraries could not be executed because the maximum amount of input data (SyncCallDataSize) was exceeded (declarations with "In" and "InOut").
0x809A =-32614	CPU function libraries could not be executed because the maximum amount of data (SyncCallDataSize) was exceeded (declarations with "In", "Out" and "InOut").
0x809B =-32613	The function returns an invalid value (a value between 0x0000 and 0x6FFF; 0xF000 and 0xFFFF is allowed).
0x809C =-32612	Function uses an invalid data type: • IN_DATA • INOUT_DATA • OUT_DATA
0xF000 – 0xFFFF =-4096 – -1	CPU 1500 V2.0 and later: The function could not be executed and returns a value between 0xF000 and 0xFFFF. (ODK_USER_ERROR_BASE = 0xF000)

Note

Call of function(s) influences the cycle time

When you call a function, the function parameters are copied. In particular in the case of large amounts of data or of structured data, this can lead to the cycle time being influenced.

6.4.3 Unloading functions

Introduction

The CPU function library is unloaded by calling the "<*STEP7Prefix*>_Unload" instruction. Call is made from the STEP 7 user program.

In addition to this call, the CPU function library is also automatically unloaded for the following reasons.

- The CPU is switched off
- The CPU is reset

Regardless of the context in which the CPU function library is running, the unloading procedure consists of the following steps:

- Call the "<*STEP7Prefix*>_Unload" instruction in the STEP 7 user program.
- From now on, no new executes can be carried out for this CPU function library. Executions still running are aborted. The execution of the function is interrupted and terminated immediately. No return value is sent to the CPU.
- The host calls the "OnStop()" and "OnUnload()" functions.

  The unloading of the cycle time can be influenced because the "OnStop()" and "OnUnload()" functions are called synchronously.
- The CPU function library is being unloaded.

"<*STEP7Prefix*>_Unload" instruction

A CPU function library is unloaded by calling the "<*STEP7Prefix*>_Unload" instruction in the STEP 7 user program.

***<STEP7Prefix>*_Unload**	
REQ	DONE
	BUSY
	ERROR
	STATUS

The following table shows the parameters of the instruction "<*STEP7Prefix*>_Unload":

Section	Declaration	Data type	Description
Input	REQ	BOOL	A rising edge activates the unloading of the CPU function library.
Output	DONE	BOOL	Indicates that the instruction has finished unloading the CPU function library.
Output	BUSY	BOOL	Indicates that the instruction is still unloading the CPU function library.
Output	ERROR	BOOL	Indicates that an error occurred during the unloading of the CPU function library. STATUS gives you more information about the possible cause.
Output	STATUS	INT	Provides information about possible sources of error, if an error occurs during the unloading of the CPU function library.

Input parameters

An edge transition (0 to 1) at the "REQ" input parameter starts the function.

Output parameter STATUS

The following table shows the information that is returned after unloading.

DONE	BUSY	ERROR	STATUS	Meaning
0	0	0	0x7000 =28672	No active unloading
0	1	0	0x7001 =28673	Unloading in progress, the first call
0	1	0	0x7002 =28674	Unloading in progress, ongoing call
1	0	0	0x0000 =0	Unloading was carried out successfully
0	0	1	0x80A4 =-32604	CPU function library could not be unloaded. A communication error between the CPU and ODK occurred during the execution of the "OnUnload()" function.
			0x80C3 =-32573	CPU function library could not be unloaded. The CPU currently does not have enough resources.
			0x8090 =-32624	An exception occurred during the unloading of the CPU function library. The CPU function library has been unloaded nevertheless.
			0x8096 =-32618	CPU function library could not be unloaded because the CPU function library was not loaded or unloading is not yet finished.
			0x809B =-32613	CPU 1500 V2.0 and later: The CPU function library could be unloaded and returns an invalid value (the values 0x0000 and 0xF000 - 0xFFFF are allowed)
			0xF000 – 0xFFFF =-4096 – -1	CPU 1500 V2.0 and later: CPU function library could be unloaded. An error occurred in the CPU function library during the execution of the "OnUnload()" function.

6.4.4 Reading the trace buffer

ODK provides a trace function to check variables or the execution of functions in the realtime environment. The trace function supports the following elements:

- An integrated trace buffer for each CPU function library.
- An "ODK_TRACE" instruction that you can add to your code
- A "GetTrace" function block, which makes it possible to read the trace buffer

"ODK_TRACE" instruction

If you define the "ODK_TRACE" instruction, it is also compiled and executed. When you define the parameter Trace=on in the <project>.odk file, the instruction is automatically defined with the following code:

```
#define ODK_TRACE(msg, ...);
```

Example: ODK_TRACE("number=%d", 13);

Calling the instruction creates an entry in the trace buffer.

When you define the parameter Trace=off in the <project>.odk file, no trace data is written.

Trace data is written automatically when an exception occurs.

Reading the trace buffer

The "GetTrace" function block enables you to read the trace buffer. The entries of the trace buffer can be read in the following ways:

- By a variable table in the web server of the CPU
- By a variable table in STEP 7 (online)
- On an HMI display

The function block is included in the standard CPP file "<project>.cpp".

GetTrace	
TraceCount	STATUS

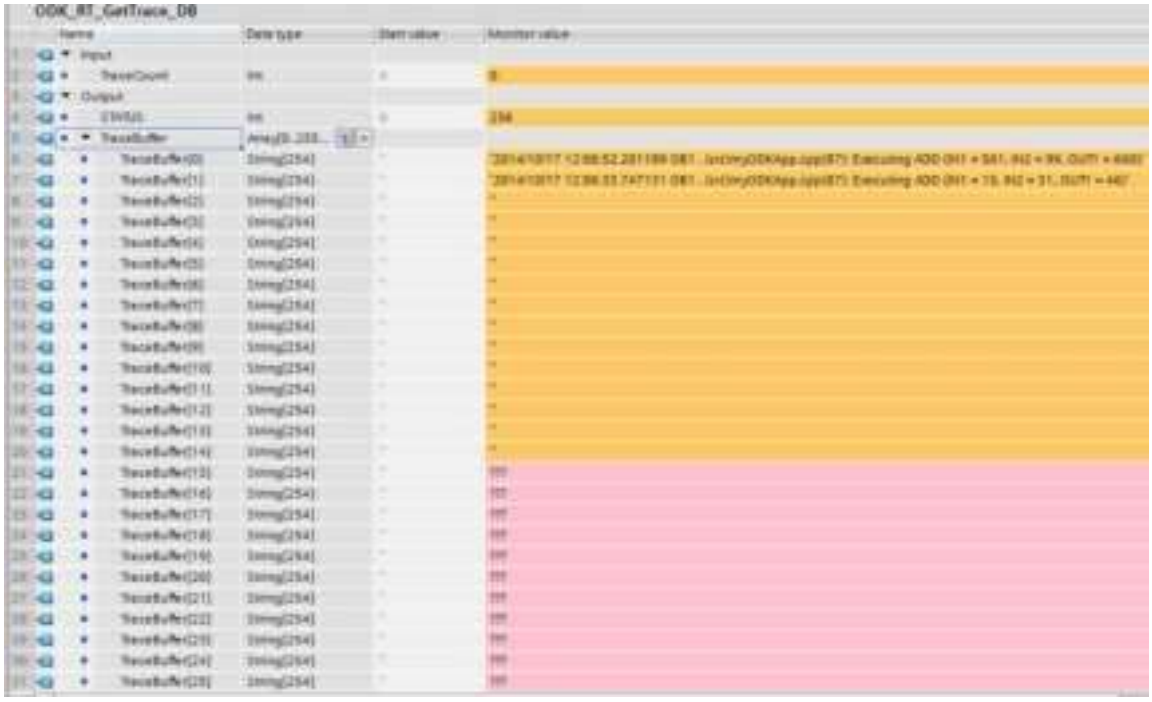
The following table shows the parameters of the "GetTrace" function block:

Section	Declaration	Data type	Description
Output	STATUS	INT	Number of trace entries actually read
Input	TraceCount	INT	Number of trace entries to be read
Output	TraceBuffer	Array [0..255] of String[125]	Trace string array for the user Each trace string consists of: • Date • Time-of-day • OB number • File name • Line number • Trace text (trace implemented by the user)

Define the function block in the SCL file as follows:

```
#ret := "ODK_App_MyFct_DB_1"(myInt:=4);
IF (#ret > 0)
{
#ret := "ODK_App_GetTraces_DB_1"(TraceCount:=20);
// ret_val = number of entries
}
```

When the "GetTrace" function block is called in STEP 7, the instance block appears as follows:



6.5 Post Mortem analysis

6.5.1 Introduction

You use the post mortem analysis to evaluate the system after an exception. The post mortem files map a snapshot at the time of the exception.

You can analyze the dump with the post mortem analysis. It includes, for example:

- Register
- Stack
- Local/global data
- Transfer parameters
- The exception number under "g_PostMortemExceptionNr" in the window "Expressions"

An exception can be triggered by one of the following cases:

- Execution of an illegal command
  - Division by zero
  - Access to protected memory
- An exception triggered by the "throw" instruction but not handled by the "try...catch" instruction

The objective of the post mortem analysis is to find the error within the CPU function library that caused the exception.

NOTICE
Exception influences the cycle time When an exception occurs in your application, the complete application memory is buffered. This may take some milliseconds and influence the cycle time.

The post mortem files for the snapshot of the first exception are not created until the CPU changes from RUN to STOP. You can use it for the following post mortem analysis. They are stored in the following directory: <load memory>/ODK1500S

The following files are created or overwritten during this process and can, for example, be downloaded via the web server:

- <project>.ed

  Binary dump of the shared object in which the exception has occurred

- <project>.es

  Stack at the time of the exception

- <project>.er

  Script for restoring the snapshot at the time of the exception

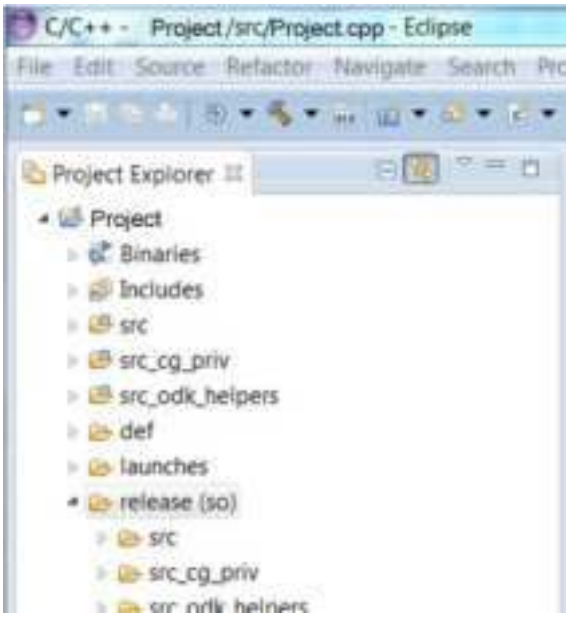
NOTICE
Insufficient load memory When there is not enough load memory, the post mortem files are not saved properly. Make sure that you have enough load memory for your applications.

6.5.2 Execute post mortem analysis

Procedure

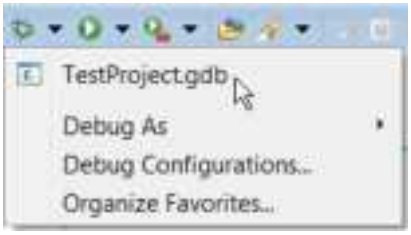
To run a post mortem analysis, follow these steps:

1. Open Eclipse.
2. Load the post mortem files to the engineering PC via the web server. Load these files to the same directory in which the SO file is stored.

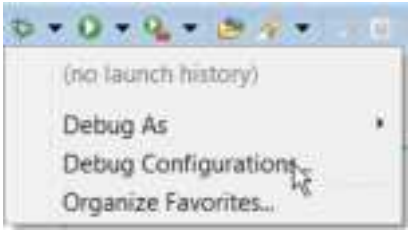


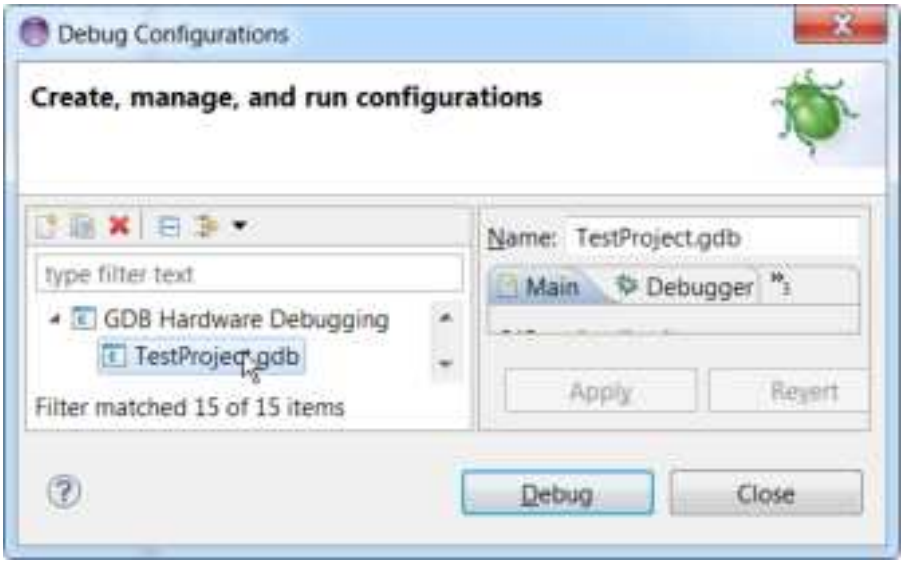
3. Select the required project.

4. Start the debugging in one of the following ways:
   - From Favorites:



   - Using "Debug Configurations"



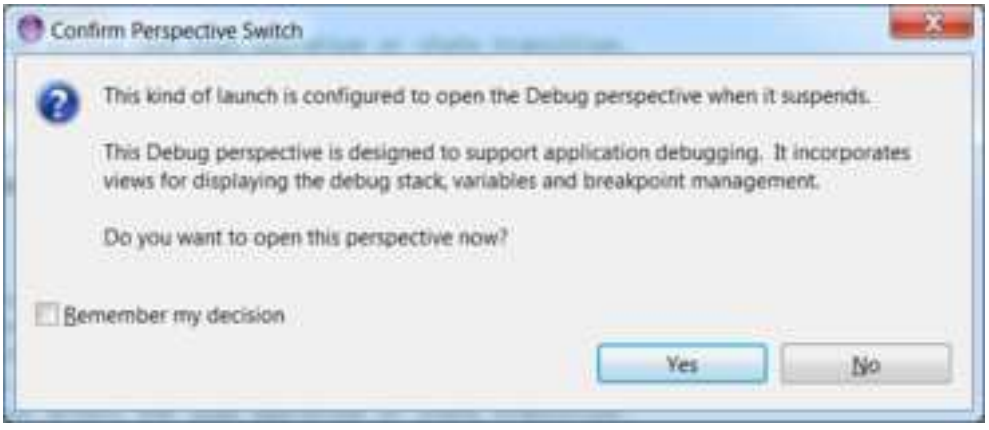


When you start a debug process for the first time, a dialog opens prompting you to select the required launch environment.

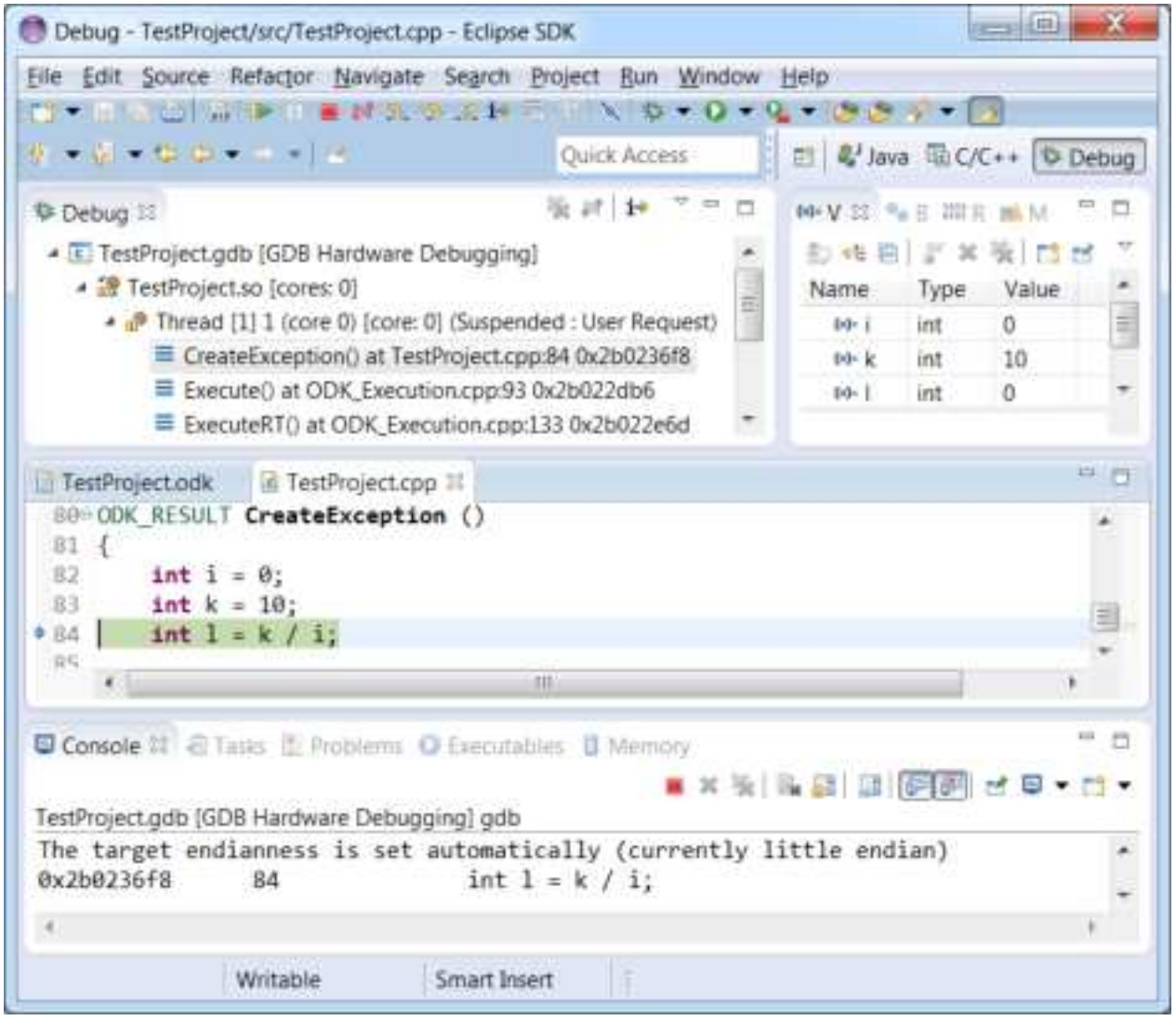
Select the item "GDB (DSF) Hardware Debugging Launcher".

A dialog opens showing you the progress of the loading process for the post mortem image. The loading process can take several minutes, depending on the size of the post mortem image.

5. Select the required debug view.



6. Run the debug process.



The exception number is displayed as "g_PostMortemExceptionNr" in the window "Expressions".

Development of a C/C++ runtime application

7

7.1 Install additional Eclipse plugins

Requirement

- ODK is installed.
- The Eclipse development environment is installed.

Procedure

1. Start Eclipse as a development environment.
2. Select the command "Install New Software..." in the menu bar under "Help".
   The "Install" dialog opens.
3. Select the "--All Available Sites--" selection under "Work with:".

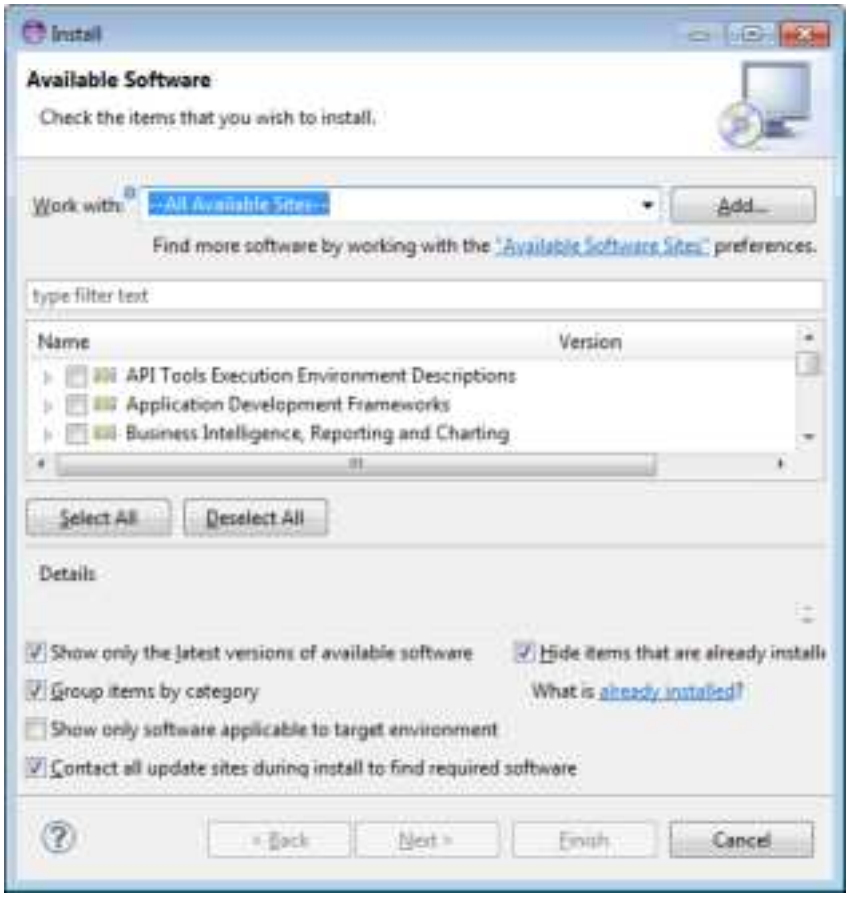


Figure 7-1 Install dialog

4. Select the following plugins:
   - C/C++ Remote Launch
   - TCF Target Explorer
   - TCF Remote System Explorer
   - TCF C/C++ Debugger

   You can filter the selection via the text box.
5. Confirm with "Next".
6. Accept the license provisions and install the plugin with "Finish".

Result

The plugins are installed and Eclipse restarted.

7.2 Create C/C++ application

7.2.1 Requirements

- ODK is installed.
- The Eclipse development environment is installed.
- Additional Eclipse plugins are installed.
- SSH client (for example, PuTTY) is installed.

Note

Root rights

The default user and the C/C++ application must not have any root rights. Create a new user to execute the C/C++ application.

Performance and jitter influence through C/C++ application

Depending on the programming type in the C/C++ application, CPU performance may be influenced by jitter.

Know-how protection

The customer is responsible for the C/C++ application and its know-how protection.

7.2.2 Creating a C/C++ Runtime Application project

A template for an Eclipse project is included in the installation of ODK 1500S to help you develop a C/C++ runtime application.

Procedure

To create a project in Eclipse using a C++-project ODK template, follow these steps:

1. Start Eclipse as a development environment.
2. In the "File > New" menu, select the command "Project..."

   The "New Project" dialog opens.

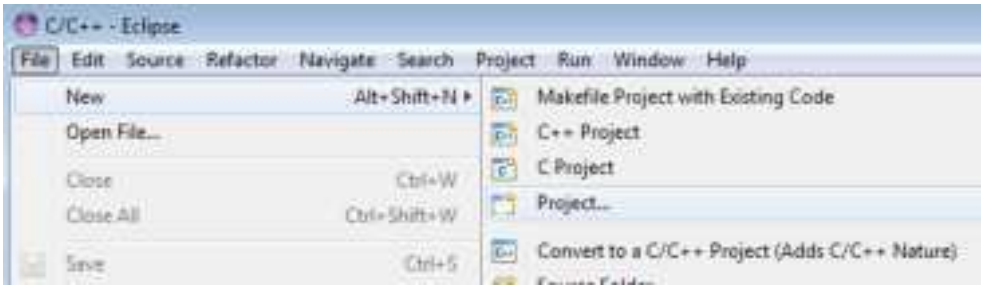


Figure 7-2 Creating a new project with Eclipse

3. Select one of the following templates depending on the CPU firmware version:
   - "C++ Project for MFP Linux application (CPU 1518 MFP - up to FW v2.6.1)"
     (Template for CC++ applications for 1518 MFP CPUs with firmware version ≤ V2.6.1)
   - "C++ Project for MFP Linux application (CU 1518 MFP FW v2.8)"
     (Template for CC++ applications for 1518 MFP CPUs with firmware version = V2.8)
   - "C++ Project for MFP Linux application (CU 1518 MFP FW v2.9 or higher)"
     (Template for CC++ applications for 1518 MFP CPUs with firmware version ≥ V2.9)

   Confirm your selection with "Next".

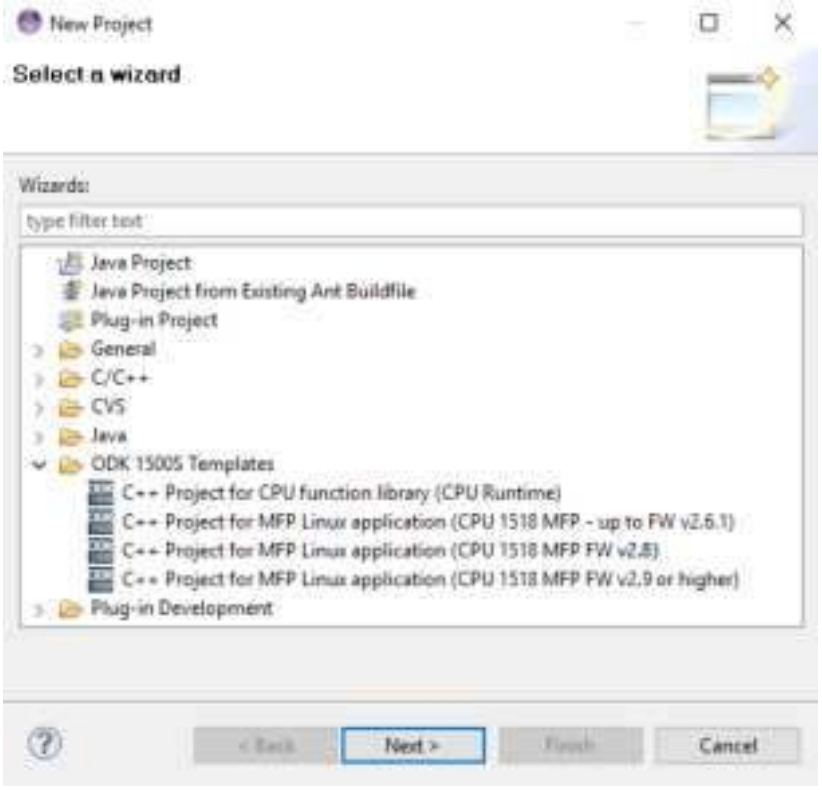


Figure 7-3 Selecting a template

4. Enter a project name.
5. Confirm with "Finish".

Result

The C/C++ project is created using the template for the C/C++ runtime application.

The template for the C/C++ runtime application configures the following data structure by default:

Project Explorer			Description
Project name:			
	src		
		<project>.cpp	Function code: This file always has the suffix CPP, regardless of whether you are creating a C or C++ project.
	launches		
		<project>.gdb.launch	Start for the post mortem analysis.
	MFP 1518_release		
		"<project>"	C/C++ Runtime Application Binary (release version) that must be transferred to the target system.

Note

Spaces in the project name

All spaces in the project name are automatically replaced by an underscore.

In the example, "My first project" becomes "My_first_project".

Note

If you need to store the workspace at another storage location, ensure that you copy the entire workspace.

7.2.3 Editing C/C++ code

Requirement

- You have created a project.
- Eclipse is open

Procedure

1. Select the "<project>.cpp" file in the project folder under "src".

   The editing mask opens.

   Figure 7-4 Editing a project

2. Edit the code.
3. To add the new C/C++ files to the project, right-click on the "src" folder and select "New > Source File" from the shortcut menu.

   The "New Source File" dialog opens.

   Figure 7-5 Dialog box New Source File

4. Enter a name for the CPP file in the "Source File" and confirm with "Finish".

   The new CPP file is stored in the "src" folder.

7.2.4 Generate C/C++ runtime application

The generation of the project data runs in an automated "**Build**" and generates the C/C++ runtime application.

Requirement

A project has been created for the C/C++ runtime application.

Procedure

To generate the project data, follow these steps:

1. Select the project for the C/C++ runtime application.
2. Select the "Build Project" command in the "Project" menu in the system bar.

   You can also select the "Build Project" command by right-clicking on the project for the C/C++ Runtime Application in the shortcut menu.

Note

The project data is only generated if you have changed the files.

Result

The generation of the project data starts. The automatically generated files are stored in the file system.

7.3 Load C/C++ runtime application in the target system

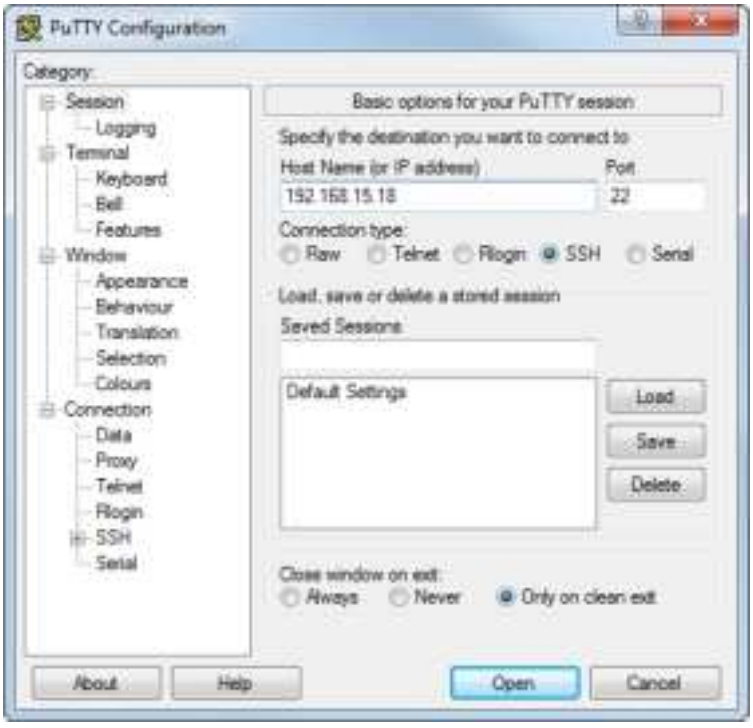
7.3.1 Configuring PuTTY

You require a configured SSH client to establish a secure connection between Eclipse and the C++ Runtime of the CPU 1518MFP (for example, PuTTY).

Procedure based on "PuTTY" example

1. Start PuTTY.
2. Enter the target address "Host Name (or IP address)" (default address: 192.168.15.18) in the text box.

   This is the IP address of the C/C++ Runtime and not the project IP address of the CPU.
3. Make sure that the following default settings are retained:
   - Port: 22
   - Connection type: SSH



4. To identify the PuTTY window and to create the association of the connection to the CPU in Eclipse, enter the title "CPU 1518MFP Linux Secure Connection" in the category "Window > Behavior" in the text box "Window title".

5. Enter the following values in the category "Connection > SSH > Tunnels".
   - Under "Source port" "1534" or "2345".
   - Under "Destination" "localhost:1534" or "localhost: 2345".

   In each case, confirm the entries with "Add".

6. Enter "CPU-1518MFP-Linux-Secure-Connection" in the category "Session" under "Saved Sessions" and confirm it with "Save".
7. To log on to the CPU 1518MFP, click "Open".

7.3.2 Commissioning C/C++ Runtime

Requirement

- You have started the CPU 1518-4 PN/DP MFP (F).

Procedure

1. Start the secure shell client (for example, PuTTY).
2. Connect the secure shell client to the CPU 1518-4 PN/DP MFP (F) using the PuTTY configuration "CPU 1518MFP Linux Secure Connection" via the target address (default address: 192.168.15.18).

3. Type in the user name and password and establish a secure shell connection.

   The default user name is "root".

   The default password is displayed under "Overview > MFP > Default Password:".
4. Change the default password after the first startup of the CPU.
5. Start the TCF Agent with the following command:

   ```
   /usr/sbin/tcf-agent -d -L- -I0 -sTCP:localhost
   ```
6. On the CPU 1518-4 PN/DP MFP (F), in the directory "/home/<user>" create a folder in which to load the application.

Reference

You can find more information on commissioning and the CPU 1518-4 PN/DP MFP (F) in the CPU manual (https://support.automation.siemens.com/WW/view/en/109749061).

7.3.3 Set up new connection to the target system in Eclipse

Requirements

- An MFP is created in Eclipse.
- An MFP is generated in Eclipse.

Procedure

Create a C/C++ remote application connection to the CPU 1518-4 PN/DP MFP (F).

1. Select the "Run Configurations..." command in the "Run" menu in the system bar.
   The "Run Configurations" dialog opens.
2. Configure your connection.

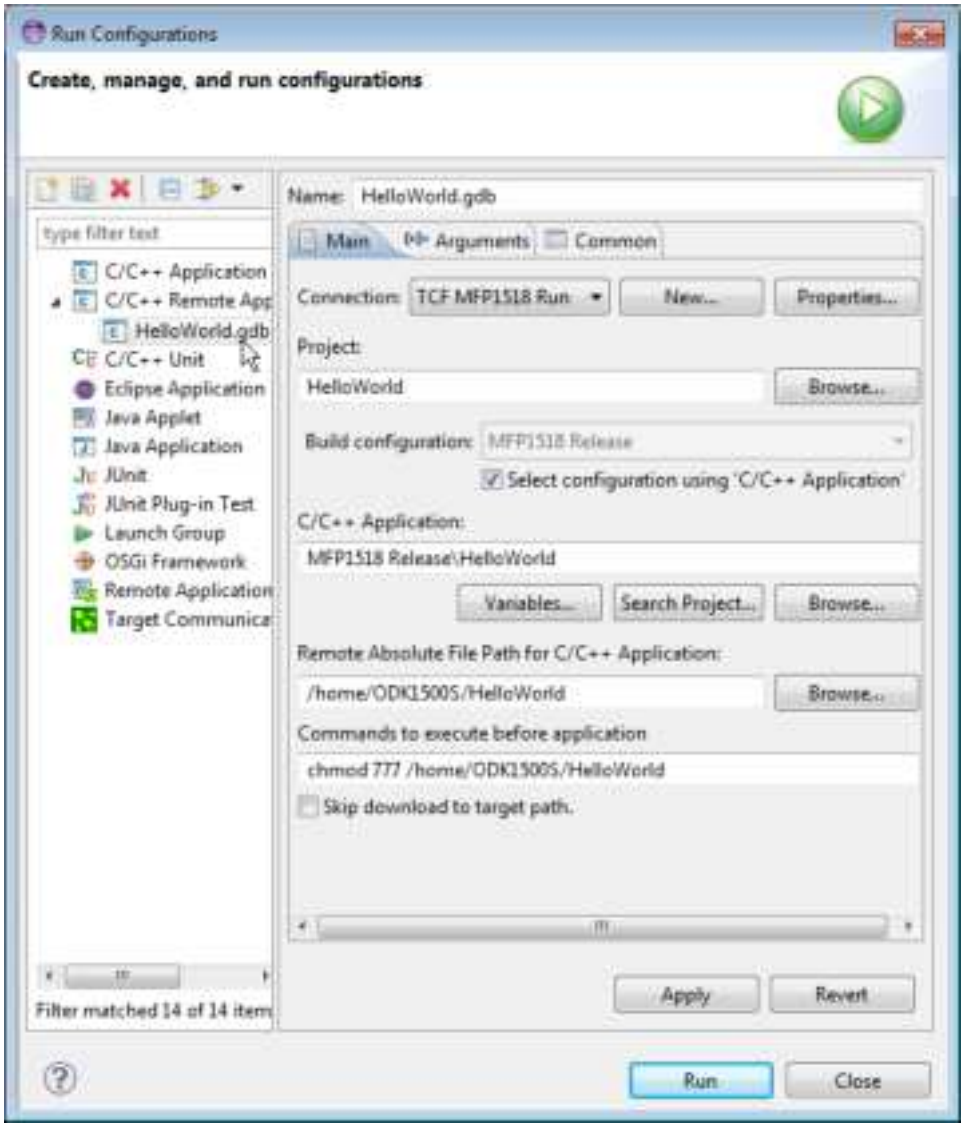


Figure 7-6 "Run Configurations" dialog with example configuration of a connection

3. To set up a new connection, click "New" in the "Main" tab under "Connection".
   The "New Connection" dialog opens.
4. Select "TCF" and confirm with "Next".

5. Fill the dialog as in the following figure and confirm with "Finish".

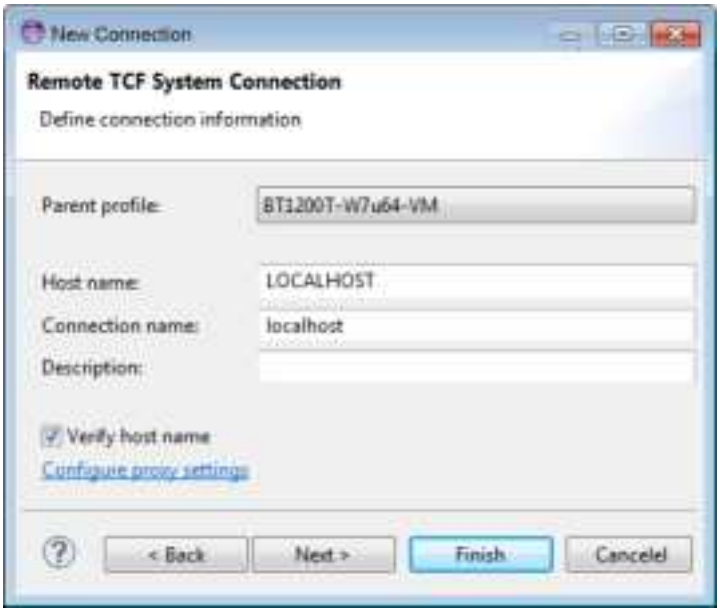


Figure 7-7 New connection dialog

6. In the "Run Configurations" dialog, select the connection "localhost" under "Connections".
7. Apply the configuration settings with "Apply"

Result

A new connection to target system has been established.

Note

Folder structure on the Linux target system

Create the folder structure manually on the Linux target system. Otherwise, remote launch is not possible and the project folder is not created automatically.

7.3.4 Load and execute C/C++ runtime application in the target system via Eclipse

Procedure

Transfer the C/C++ runtime application to the target system.

1. Select the "Run Configurations..." command in the "Run" menu in the system bar.

   The "Run Configurations" dialog opens.
2. Select the required configuration under "C/C++ Remote Application".
3. Run the loading process with "Run".

Result

Your program is executed on the CPU 1518-4 PN/DP MFP (F).

7.3.5 Load and debug C/C++ runtime application in the target system via Eclipse

To debug C/C++ applications, you have the option to write a custom test. This will ensure the quality of the code.

Procedure

To perform the test, proceed as follows:

1. Open your project in Eclipse.
2. In the "Run" menu, select the command "Debug Configurations".

   The "Debug Configurations" dialog opens.
3. If you debug the project for the first time, you must now set the debug configuration. Otherwise, continue with step 5.

4. Configure your connection in the "Main" tab as described under Set up new connection to the target system in Eclipse (Page 114).

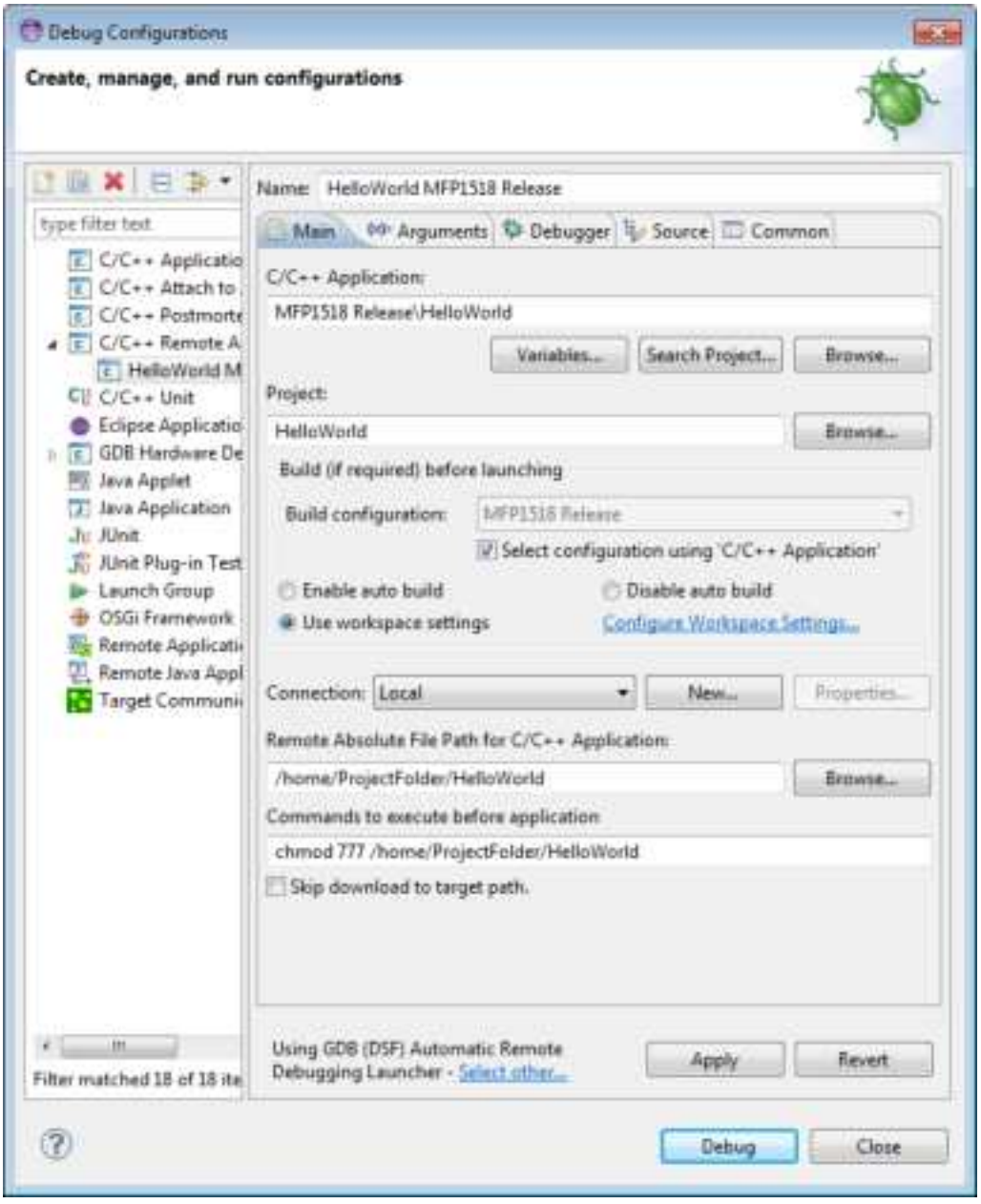


Figure 7-8 Configuring the connection

5. Select the required configuration under "C/C++ Remote Application".
6. Start the debug process by clicking the "Debug" button.

Result

Eclipse suggests a change in the debug perspective.

The test code is executed.

7.4 Execute C/C++ runtime application

7.4.1 Start application via secure shell

Requirement

The CPU is connected to a secure shell client.

Procedure

1. Open the secure shell client.
2. To decouple the application from the secure shell, enter the command "nohup" before calling the application.
3. Call the application via the secure shell client.

Result

The CPU executes the application.

Note

The CPU executes the application also after the secure shell client has terminated.

Developing a PLCSIM Advanced function library 8

8.1 Creating a PLCSIM Advanced function library

8.1.1 Requirements

The Microsoft Visual Studio development environment is not included in the ODK product package. You can find the Download Center for Microsoft development tools on the Internet (https://www.microsoft.com/en-us/download/developer-tools.aspx).

The C++ PLCSIM Advanced project template supports the applications as of Visual Studio 2015. To debug the generated DLL file, you need Visual Studio 2017 or newer.

8.1.2 Creating a PLCSIM Advanced function library with Visual Studio

To help you develop a PLCSIM Advanced function library, a project template for PLCSIM Advanced function libraries for a project in Visual Studio is included in the installation of ODK 1500S. The template supports 32-bit and 64-bit applications.

Procedure

To create a project in Microsoft Visual Studio using the project template, follow these steps:

1. Open Microsoft Visual Studio as a development environment.
2. In the "File > New" menu, select the command "Project..."

   The "New Project" dialog opens.

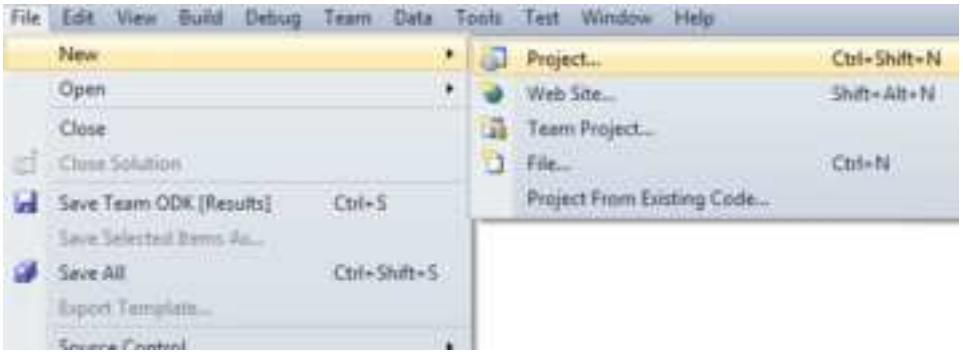


Figure 8-1 Creating a new project in Visual Studio

3. Select the project template
   "ODK 1500S V2.5.1 PLCSIM Advanced function library C++ (Windows Sync)" under "Visual C++".

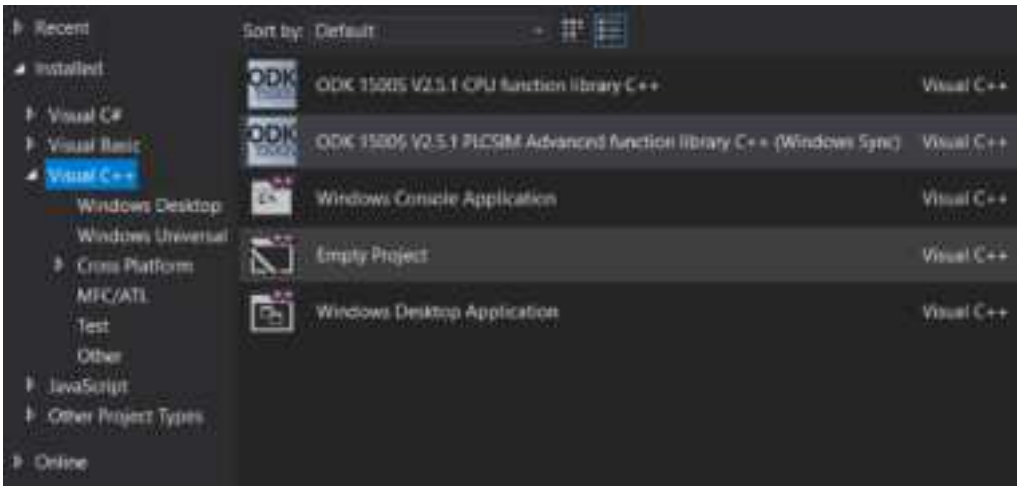


Figure 8-2 Select a template

4. Enter a project name.
5. Click "OK" to confirm.

Result

The PLCSIM Advanced function library is created using the project template and sets the following project settings:

- Project settings for generating the DLL file
- Automates the generation of the DLL and SCL file

By default, the project template sets up the following Solution Explorer structure:

Folder / file			Description
<project>			
	Definition File		
		<project>.odk	ODK interface description
		<project>.scl.additional	S7 blocks that are appended to the <project>.scl file. Although the file is not part of the project template, the code generator processes the file.
	Generated Files		**Files from this folder must not be edited!**
		ODK_Types.h	Definition of the ODK base types
		ODK_Functions.h	Function prototypes
		ODK_Execution.cpp	Implementation of the "Execute" method
	Header Files		Header file
	ODK Helpers		**Files from this folder must not be edited!**
		ODK_CpuReadData.h	Definition: Help functions for reading the data blocks

Folder / file			Description
		ODK_CpuReadData.cpp	Implementation: Help functions for reading the data blocks
		ODK_CpuReadWriteData.h	Definition: Help functions for reading/writing the data blocks
		ODK_CpuReadWriteData.cpp	Implementation: Help functions for reading/writing the data blocks
		ODK_StringHelper.h	Definition: Help functions S7 strings / W strings
		ODK_StringHelper.cpp	Implementation: Help functions S7 strings / W strings
	Resource Files		
		<project>.rc	
	Source Files		Source files
		<project>.cpp	Function code
		dllmain.cpp	Implementation of the "dllmain" file
	STEP7		**Files from this folder must not be edited!**
		<project>.scl	S7 blocks

8.2 Transferring the PLCSIM Advanced function library to PLCSIM Advanced

After creating it, transfer the PLCSIM function library to the PLCSIM Advanced program.

Transferring PLCSIM Advanced function library

Transfer the DLL file manually to PLCSIM Advanced. Use the standard Windows data transfer procedure to transfer the PLCSIM Advanced function library.

Save the DLL file on the virtual memory card with the S7-PLCSIM Advanced Control Panel.

The default value that describes the file path is:

C:\Users\<user name>\Documents\Siemens\Simatic\Simulation\Runtime\Persistence\<name of instance>\SIMATIC_MC\ODK1500S

Note

Administrator rights

Assign write permission to this folder only for the administrator. This prevents unauthorized personnel from importing PLCSIM Advanced function libraries.

8.3 Defining the runtime properties of a PLCSIM Advanced function library

The next step is to define the interface description of the PLCSIM Advanced function library in the <project>.odk file. The file contains the following elements:

- Comments
- Parameters
- Definitions of functions and structures

Procedure

To define the interface description in the <project>.odk file, follow these steps:

1. Open the <project>.odk file.
2. Change the elements depending on your requirements.

Description of the elements

Comments

You can use comments for explanation purposes.

Parameters

The definition of the parameters must be within a line of code.
`<parameter name>=<value> // optional comment`

The interfaces file supports the following parameters:

Parameter	Value	Description
Context	user	Specifies that the PLCSIM Advanced function library is loaded in the context of a user.
STEP7Prefix	<String>	Describes the string that precedes your functions and is shown after importing the SCL file in STEP 7. The following characters are allowed: {A...Z, a...z, 1...9, -, _} Umlauts are not permitted. The project name is entered without spaces by default.

Note
Spaces in the project name

With the STEP7 prefix, invalid characters are replaced by an underscore.

8.4 Definition of the <Project>.odk file

The function prototypes and function blocks are generated based on the selected parameters in the <project>.odk file. Define the <project>.odk file for this.

By default, the <project>.odk file contains the following:

- Description

  The possible data types that are used for the interface are described in comment lines. This simplifies the definition of the correct tag type for your task.

- Context=user

  The CPU function library is loaded in the "User" context. You can change the parameter to Context=system.

- STEP7Prefix="<project>"

  Sets a string for the SCL generation in front of the functions of the PLCSIM function library. The string is visible in STEP 7. You can change the parameter. The string length of the prefix including the function name must not exceed a length of 125 characters (for example, ODK_App_SampleFunction)

- "SampleFunction" function definition

  You can change this default function as you wish in the <project>.odk file and add more functions. The string length must not exceed a length of 125 characters. The associated function is located in the CPP file.

Example

```
//INTERFACE ODK 1500S V2.5.1
Context=user
STEP7Prefix=ODKProject
Trace=on

 /*
* Elementary data types:
*  ODK_DOUBLE      LREAL    64-bit floating point, IEEE 754
*  ODK_FLOAT       REAL     32-bit floating point, IEEE 754
*  ODK_INT64       LINT     64-bit signed integer
*  ODK_INT32       DINT     32-bit signed integer
*  ODK_INT16       INT      16-bit signed integer
*  ODK_INT8        SINT     8-bit signed integer
*  ODK_UINT64      ULINT    64-bit unsigned integer
*  ODK_UINT32      UDINT    32-bit unsigned integer
*  ODK_UINT16      UINT     16-bit unsigned integer
*  ODK_UINT8       USINT    8-bit unsigned integer
*  ODK_LWORD       LWORD    64-bit bit string
*  ODK_DWORD       DWORD    32-bit bit string
*  ODK_WORD        WORD     16-bit bit string
*  ODK_BYTE        BYTE     8-bit bit string
*  ODK_BOOL        BOOL     1-bit bit string
*  ODK_LTIME       LTIME    64-bit duration in nanoseconds
*  ODK_TIME        TIME     32-bit duration in milliseconds
*  ODK_LDT         LDT      64-bit date and time of day
*                           in nanoseconds
```

```
*  ODK_LTOD        LTOD     64-bit time of day in nanoseconds
*                           since midnight
*  ODK_TOD         TOD      32-bit time of day in milliseconds
*                           since midnight
*  ODK_CHAR        CHAR     8-bit character
* Complex Datatypes:
*  ODK_DTL         DTL      structure for date and time
*  ODK_S7STRING    STRING   character string with 8-bit characters
*  ODK_CLASSIC_DB  VARIANT  classic DB (global or based on UDT
*                           "optimized block access" must be
unchecked)
*  []              ARRAY    field of this datatype
* User Defined Datatype:
*  ODK_STRUCT      UDT      user defined structure
* Return Datatype:
*  ODK_RESULT      0x0000-0x6FFF function succeeded
*                               (ODK_SUCCESS = 0x0000)
*                  0xF000-0xFFFF function failed
*                               (ODK_USER_ERROR_BASE = 0xF000)
*/

// Basic function in order to show
// how to create a function in ODK 1500S.
ODK_RESULT SampleFunction([IN]   ODK_INT32   myInt   // integervalue
                                                     // as input
                        , [OUT]  ODK_BOOL    myBool  // bool value
                                                     // as output
                        , [INOUT] ODK_DOUBLE myReal);// double value
                                                     // as input
                                                     // and output
```

8.5 Modifying the <Project>.odk file

The following example shows you how you can change the <project>.odk file to suit your needs.

```
//INTERFACE ODK 1500S V2.5.1
Context=user
STEP7Prefix=SampleProject
Trace=on

// Basic function in order to show
// how to create a function in ODK 1500S.
ODK_RESULT SampleFunction([IN]  ODK_INT32 num1
                        , [IN]  ODK_INT32 num2
                        , [OUT] ODK_INT32 sum);
```

8.6 Editing PLCSIM Advanced function library

Once you have defined the ODK interface in the <project>.odk file, you must edit the functions of the PLCSIM Advanced function library in the Project Source file.

Procedure

To edit the function of a PLCSIM Advanced function library, follow these steps:

1. To generate the function prototypes, execute the build.
2. Open the project source file, or create a custom source file if necessary.
3. Transfer the function prototypes from <ODK_Functions.h> to the source file.

Note

Use the function prototype macro to transfer the step 3 in the future when there is a change to the function parameters.

4. Edit the code of your PLCSIM Advanced function library in the execute file (<project>.cpp).

PLCSIM Advanced function library

The execute file contains a schematically represented function description by default. You can change this description with corresponding changes in the <project>.odk file and/or add more function descriptions.

Execute file based on C++ example

```
#include "stdafx.h"
#include "ODK_Functions.h"
#include "tchar.h"

EXPORT_API ODK_RESULT OnLoad (void)
{
    // place your code here
    return ODK_SUCCESS;
}
EXPORT_API ODK_RESULT OnUnload (void)
{
    // place your code here
    return ODK_SUCCESS;
}
EXPORT_API ODK_RESULT OnRun (void)
{
    // place your code here
    return ODK_SUCCESS;
}
EXPORT_API ODK_RESULT OnStop (void)
{
    // place your code here
    return ODK_SUCCESS;
}
```

```
ODK_RESULT SampleFunction(
                     /*IN*/      const ODK_INT32& myInt,
                     /*OUT*/     ODK_BOOL& myBool,
                     /*INOUT*/   ODK_DOUBLE& myReal)
{
    return ODK_SUCCESS;
}
```

8.7 Generating a PLCSIM Advanced function library

The generation of the project data is divided into two automated steps.

- **Pre-Build**: Generation of the files created by default based on the changed <project>.odk file and generation of the SCL file.
- **Actual-Build**: Generation of the DLL file.

Procedure

To generate the project data, follow these steps:

1. Save all edited files.
2. In the "Build" menu, select the command "Build Solution".

Note

C/C++ projects

Perform the build of the PLCSIM Advanced function library in the "Release" configuration, because the software controller has already installed the C/C++ Redistributables (Release Runtime files).

To use the "Debug" configuration, copy the Debug Runtime files to the software controller.

Note

The project data is only generated if the files have been changed.

Result

The generation of the project data is started. The automatically generated files are stored in the file system.

- DLL file: Project directory\<project>\<BuildConfiguration>\<project>.dll
- SCL file: Project directory\<project>\STEP7\<project>.scl

8.8 Executing a function

Executing functions is described in the section "Executing a function (Page 54)" using the example of a CPU function library for the Windows environment.

Special features about PLCSIM Advanced, as well as advanced error codes are described in the manual "SIMATIC S7-1500 S7-PLCSIM Advanced (https://support.automation.siemens.com/WW/view/en/109760835)".

8.9 Debugging C/C++ Code

The debugging of Visual Studio C/C++ code is described in the section "Remote debugging (Page 62)". To debug the generated DLL file, you need Visual Studio 2017 or newer.

While Visual Studio is connected to the client, PLCSIM Advanced is also in debug mode and therefore remains in the "RUN" state. The cycle time is not exceeded.

Using example projects

9

To facilitate your introduction , ODK 1500S offers example projects for both development environments. The example projects consist of the following elements:

- A project for Microsoft Visual Studio or Eclipse
- A compiled binary and SCL source that enables you to immediately test the example projects
- A STEP 7 example project

Storage location of example projects

- The example projects for the CPU function libraries are available on the Internet (https://support.industry.siemens.com/cs/document/106192387/simatic-odk-1500s-examples?dti=0&lc=en-WW) for download.
- Example projects for C/C++ Runtime applications:
  - Setting up communication between CPU and C/C++ runtime for a multifunctional platform using OPC UA (https://support.industry.siemens.com/cs/ww/en/view/109749176)
  - Establishment of Open User Communication between CPU runtime and C/C++ runtime of a multifunctional platform (https://support.industry.siemens.com/cs/ww/en/view/109756757)

Using example projects

To open the example projects, follow these steps:

1. Transfer the example projects onto the hard disk of your PC.
2. Transfer the C/C++ runtime application, DLL or SO file to the target system.

General conditions

A

A.1 Number of loadable CPU function libraries

You can load up to 32 CPU function libraries for Windows and realtime environment.

Configuration limits for CPU function libraries:

- CPU function libraries for the Windows environment:
  - Up to 32 parallel function calls (total)
  - Up to 1 MB input and output data (in total)
  - Up to 1 MB input data per function call
  - Up to 1 MB output data per function call

  Note

  The memory for input and output parameters is allocated dynamically, depending on the quantity needed. The memory is allocated here in blocks of 8 KB each.

- Development of a CPU function library for the real time environment
  - Parallel function calls in a CPU function library defined by the "SyncCallParallelCount" parameter
  - Up to 32 parallel function calls (in total)
  - Up to 1 MB input data and output data per function call

Memory for loading CPU function libraries

The available memory for loading of CPU function libraries is limited in the context of the real time environment. The table below provides an overview of the available memory of the different CPUs for loading CPU function libraries:

CPU	Memory available for loading	Maximum size of the SO file
CPU 1505SP (T)(F)	20 MB	5.8 MB
CPU 1507S (F)	50 MB	9.8 MB
CPU 1518-4 PN/DP MFP (F)	50 MB	9.8 MB

The following restrictions are also in effect in the context of the realtime environment:

- SO file name may not exceed 56 characters.

A.2 Compatibility

If you use an ODK version V2.5, note the following:

- Engineering:

  A CPU function library project that was created with an ODK version < V2.5 is not compatible. You need to recreate a CPU function library in the version V2.5.

- Runtime:

  A CPU function library that was created with an ODK version < V2.5 is not compatible with newer CPU versions.

Syntax Interface file <project>.odk for CPU function libraries

B

B.1 Data types

The data type defines the type of a tag. The following table defines the possible data types and their representation in the individual program languages or in C++ or STEP 7:

Elementary data types:

ODK data type	SIMATIC data type	C++ data type	C# data type	VB data type	Description
ODK_DOUBLE	LREAL	double	double	Double	64-bit floating point, IEEE 754
ODK_FLOAT	REAL	float	float	Single	32-bit floating point, IEEE 754
ODK_INT64	LINT	long long	long	Long	64-bit signed integer
ODK_INT32	DINT	long	int	Integer	32-bit signed integer
ODK_INT16	INT	short	short	Short	16-bit signed integer
ODK_INT8	SINT	char	sbyte	SByte	8-bit signed integer
ODK_UINT64	ULINT	unsigned long long	ulong	ULong	64-bit unsigned integer
ODK_UINT32	UDINT	unsigned long	uint	UInteger	32-bit unsigned integer
ODK_UINT16	UINT	unsigned short	ushort	UShort	16-bit unsigned integer
ODK_UINT8	USINT	unsigned char	byte	Byte	8-bit unsigned integer
ODK_LWORD	LWORD	unsigned long long	ulong	ULong	64-bit bit string
ODK_DWORD	DWORD	unsigned long	uint	UInteger	32-bit bit string
ODK_WORD	WORD	unsigned short	ushort	UShort	16-bit bit string
ODK_BYTE	BYTE	unsigned char	byte	Byte	8-bit bit string
ODK_BOOL	BOOL	unsigned char	bool	Boolean	1-bit bit string, remaining bits (1..7) are empty
ODK_LTIME	LTIME	long long	long	Long	64-bit during in nanoseconds
ODK_TIME	TIME	long	int	Integer	32-bit during in milliseconds
ODK_LDT	LDT	unsigned long long	ulong	ULong	64-bit date and time of the day in nanoseconds
ODK_LTOD	LTOD	unsigned long long	ulong	ULong	64-bit time of the day in nanoseconds since midnight
ODK_TOD	TOD	unsigned long	uint	UInteger	32-bit time of the day in milliseconds since midnight
ODK_WCHAR	WCHAR	wchar_t	char	Char	**Only for Windows:** 16-bit character
ODK_CHAR	CHAR	char	sbyte	SByte	8-bit character

Complex data types:

ODK data type	SIMATIC data type	C++ data type	C# data type	VB data type	Description
ODK_DTL	DTL	struct ODK_DTL	OdkInternal. Dtl (class)	OdkInternal. Dtl (class)	Structure for date and time
ODK_S7STRING	STRING	unsigned char	string	String	Character string (8-bit character) with max. and act. length (2xUSINT)
ODK_S7WSTRING	WSTRING	unsigned short	string	String	**Only for Windows:** Character string (16-bit character) with max. and act. length (2xUINT)
ODK_VARIANT	VARIANT	struct ODK_VARIANT	byte [ ]	byte [ ]	**For Windows only:** Classic data (each data type that can be serialized with classic data.)
ODK_CLASSIC_DB	VARIANT	struct ODK_CLASSIC_DB	-	-	**Only for realtime environment:** Classic DB (global or based on UDT)
[ ]	ARRAY	[ ]	[ ]	[ ]	Range of same data types. The maximum number of array elements is 2^{20} (=1,048,576). You can use all data types as array except IN_DATA / INOUT_DATA / OUT_DATA.

User-defined data types:

User-defined data types (UDT) include structured data, especially the names and the data types of this component and their order.

A user-defined data type can be defined in the user interface description with the keyword "ODK_STRUCT".

Example

```
ODK_STRUCT <StructName>
{
 <DataType> <TagName>;
 ...
};
```

The following syntax rules apply to the structure:

- You can divide the structure into multiple lines.
- The structure definition must end with a semicolon.
- Any number of tabs and spaces between the elements is permitted.
- It is not permitted to use any keywords for the generated language used (for example "en / eno" as tag name).

Restrictions of the data type ODK_VARIANT:

- When a parameter of the data type ODK_VARIANT is used, it is not permitted to use other parameters with the same InOut-Identifier, regardless of data type.
- With the data type ODK_VARIANT, an [OUT] is modeled as [INOUT] in the generated FB.

Restrictions of the data type ODK_CLASSIC_DB:

- The data type ODK_CLASSIC_DB can only be used with the InOut-Identifier [IN] and [INOUT].
- When a parameter of the data type ODK_CLASSIC_DB is used with the InOut-Identifier [IN] or [INOUT], it is not permitted to use other parameters with the same InOut-Identifier, regardless of data type.

B.2 Parameters

The parameters of the <project>.odk file are different:

- Developing a CPU function library for the Windows environment
- Developing a CPU function library for the realtime environment

Parameters for the Windows environment

The definition of the parameters must be within a line of code.

```
<parameter name>=<value> // optional comment
```

The <project>.odk file supports the following parameters:

Parameter	Value	Description
Context	user	Specifies that the CPU function library is loaded in the context of a Windows user.
	system	Specifies that the CPU function library is loaded in the context of the Windows system.
STEP7Prefix	<String>	Describes the string that precedes your functions and is shown after importing the SCL file in STEP 7. The following characters are allowed: {A...Z, a...z, 1...9, -, _}
FullClassName	<String>	The parameter is required for the C# and VB programming languages. To change the class names or namespace of the source files of the CPU function library, you need to adjust the "FullClassName" parameter.

Parameters for the realtime environment

The definition of the parameters must be within a line of code.

```
<parameter name>=<value> // optional comment
```

The <project>.odk file supports the following parameters:

Parameter	Value	Description
Context	realtime	Specifies that the CPU function library is loaded in the context of the real time environment.
Trace	on	Specifies the trace function in the CPU function library. In this case, the CPU function library requires 32 KB if memory as an additional trace buffer. A "GetTrace" function block is created by default for use in a STEP 7.
	off	A "GetTrace" function block is created. The trace buffer contains only one trace entry with the contents: trace is off.
HeapSize	[4...<Available CPU memory> (Page 130)]k	Specifies a memory in KB that is used as heap for realtime applications.
HeapMaxBlockSize	[8...<HeapSize>]	Specifies the memory size in bytes that can be allocated at one time.
SyncCallParallelCount	[1...9] Default=3	If a optional parameter and defines the maximum number of parallel calls in this CPU function library. The size of the memory which is reserved for calls in this CPU function library: SyncCallParallelCount * (SyncCallStackSize + SyncCallDataSize)
SyncCallStackSize	[1...1024]k Default=32k	Is a optional parameter and defines the size of the thread stack for a call in this CPU function library. Each new call contains a separate stack memory.
SyncCallDataSize	[1...1024]k	Is a optional parameter and defines the size of the data area for a call in this CPU function library. The data area contains IN, INOUT and OUT parameters. Each new call contains a separate stack memory.
	Default=auto	The required data size is automatically calculated by the code generator.
STEP7Prefix	<String>	Describes the string that precedes your functions and is shown after importing the SCL file in STEP 7. The following characters are allowed: {A...Z, a...z, 1...9, -, _} By default the name is entered without blanks.

Changes in the interface file

If changes to the interface file (.odk) are not automatically recognized with the next build, run a manual rebuild.

Code generator messages for CPU function libraries

C

C.1 Error messages of the code generator

The code generator stops the build process and generates the following error messages:

File errors:

Error number	Error message	Possible solution
100	'<Project>.odk' is missing	Rename the file to <project>.odk.
101	Context is missing in resource file	Valid for Visual Studio only. One of the following resource files is faulty: • C++: <project>.rc • C#: AssemblyInfo.cs • VB: AssemblyInfo.vb
102	resource file '...' is missing	Valid for Visual Studio only. One of the following resource files is missing: • C++: <project>.rc • C#: AssemblyInfo.cs • VB: AssemblyInfo.vb
103	'...' write protected	One of the following files is write-protected: • C++ – <project>.rc (only for Visual Studio) – ODK_Types.h – ODK Functions.h – ODK_Execution.cpp • C# (only for Visual Studio) – AssemblyInfo.cs – OdkTypes.cs – OdkFunctions.cs – OdkExecution.cs • VB (for Visual Studio only) – AssemblyInfo.vb – OdkTypes.vb – OdkFunctions.vb – OdkExecution.vb • General – cg.tmp Temporary file for the code generator to detect changes in the interface file. – <project>.scl

Error number	Error message	Possible solution
110	license key missing	Transfer a current license key.
111	retrieve license key not possible	Install the ALM with the version ≥ 6.0.

Parameter errors:

Error number	Error message	Possible solution
200	parameter '...' is not allowed for current context	The indicated parameter is not allowed here.
201	missing '...' definition	The indicated parameter (Page 68) is not defined.
202	more than one definition for '...'	There is more than one definition for the indicated parameter (Page 68).
203	Context has to be one of 'user' or 'system' for Microsoft Visual Studio	Choose the context "system" or "user" for Visual Studio.
204	Context has to be 'realtime' for Eclipse	Choose the context "realtime" for Eclipse.
205	Trace has to be on or off	The "Trace" parameter must have the value "on" or "off" (only for realtime environment).
206	STEP7Prefix must not be longer than 120 characters	The STEP 7 prefix must not exceed 120 characters.
207	HeapSize has to be interval of [4...100000]k	Ensure that the HeapSize parameter is within the value range [4...100000]k.
208	HeapMaxBlockSize has to be interval of [8...<HeapSize>]	Ensure that the HeapMaxBlockSize parameter is within the value range [8...<HeapSize>].
209	SyncCallDataSize must be interval of [1...1024]k	Ensure that the SyncCallDataSize parameter is within the value range [1...1024]k.
210	SyncCallStackSize must be interval of [1...1024]k	Ensure that the SyncCallStackSize parameter is within the value range [1...1024]k.
211	SyncCallParallelCount must be interval of [1...9]	Ensure that the SyncCallParallelCount parameter is within the value range [1...9].

Syntax errors:

Error number	Error message	Possible solution
500	unexpected end-of-file found	Always end the file with a semicolon.
501	'...' should be alpha numeric	The following characters are allowed: a - z, A - Z, 0 - 9, _ Umlauts are not permitted.
502	'...' should be numeric	The following characters are allowed: 0 - 9
503	'...' undefined keyword	Use only the keywords [IN], [OUT] and [INOUT] and the defined data types.
504	... missing before ...	Add the character displayed by the error message.
	missing space	Add a space.
506	'...' undefined type	Use only the defined data types.
507	'...' type not allowed	Observe the syntax rules in section Defining functions a CPU function library (Page 71)
508	'...' type redefinition	The function or parameter name is already assigned. Choose a different name.
509	'...' variable redefinition	The tag name is already assigned. Choose a different name.
510	Structure '...' must not be empty	Fill the structure with a data type.

Error number	Error message	Possible solution
511	'...' no valid name	Observe the syntax rules in section Defining functions a CPU function library (Page 71).
512	unexpected variable order (must be [IN], [OUT], [INOUT] order)	There are three defined InOut identifiers. Use these in the following order: [IN], [OUT], [INOUT]
513	size of ODK_S7STRING could not be bigger than 254	A string can have a maximum length of 254 characters.
514	size of ODK_S7WSTRING could not be bigger than 16382	A Wstring can have a maximum length of 16382 characters.
515	Prefix + Function name '....' exceeds 125 characters	Prefix and function name together are longer than 125 characters.
516	variable name '...' exceeds 128 characters	The tag name is longer than 128 characters.
517	'...' IN_BUFFER + INOUT_BUFFER could not be greater than 1 MB	Altogether, the InOut identifiers [IN] and [INOUT] in a function must not exceed 1 MB.
518	'...' INOUT_BUFFER + OUT_BUFFER could not be greater than 1 MB	Altogether, the InOut identifiers [OUT] and [INOUT] in a function must not exceed 1 MB.
519	'...' needs '...k', but data size (SyncCallDataSize) is limited to '...k'	The amount of data is too high.
520	'...' has an array size of '...', but max. array size is limited to '...'	The maximum Array size is exceeded.
521	no other variable in the same direction for ODK_CLASSIC_DB / ODK_VARIANT type	As soon as the data type ODK_CLASSIC_DB or ODK_VARIANT is used, no other parameter may defined with the same InOut identifier.
522	no array allowed for ODK_CLASSIC_DB / ODK_VARIANT type	No Array may be defined for the data type ODK_CLASSIC_DB or ODK_VARIANT.
523	no [OUT] direction allowed for ODK_CLASSIC_DB type	The InOut identifier [OUT] may not be defined for the ODK_CLASSIC_DB data type.
524	function declarations lead to identical hashes (change name of one parameter): '...', '...'	Change a parameter name.

C.2 Warnings of the code generator

The code generator continues to execute the build process and generates the following warnings:

Warning number	Warning message	Description
4100	built project with ODK 1500S trial mode - '...' day(s) left	Use the test version. The warning shows when the test version runs.

Helper functions for CPU function libraries

D

D.1 C++ helper functions

String helper functions for CPU function library for the Windows and realtime environment

The following helper functions provide access to S7 strings:

Helper functions	Description
Convert_S7STRING_to_SZSTR	Convert PLC string types to C/C++ string types ("char" array, null-terminated)
Convert_SZSTR_to_S7STRING	Convert C/C++ string types ("char" array, null-terminated) to PLC string types.
Get_S7STRING_Length	Returns the current length of a PLC string type.
Get_S7STRING_MaxLength	Returns the maximum length of a PLC string type.

String helper functions for CPU function library for the Windows environment

The following helper functions provide access to S7WStrings:

Helper functions	Description
Convert_S7WSTRING_to_SZWSTR	Convert PLC WString types to C/C++ WString types ("wchar_t" array, null-terminated)
Convert_SZWSTR_to_S7WSTRING	Convert C/C++ WString types ("wchar_t" array, null-terminated) to PLC WString types.
Get_S7WSTRING_Length	Returns the current length of a PLC Wstring type.
Get_S7WSTRING_MaxLength	Returns the maximum length of a PLC WString type.

Class "CODK_CpuReadData" (Windows and real-time environment)

The "CODK_CpuReadData" class allows read access to classic DBs / classic data:

Value	Description
CODK_CpuReadData	Class constructor, initializes the input data area and the data size.
SetBuffer	Initializes the input data area and the data size.
ReadS7BOOL	Reads "bool" (1 byte) from the data area.
ReadS7BYTE	Reads a "byte" (1 byte) from the data area.
ReadS7WORD	Reads a "word" (2 bytes) from the data area.
ReadS7DWORD	Reads a "double word" (4 bytes) from the data area.
ReadS7LWORD	Reads a "long word" (8 bytes) from the data area.
ReadS7SINT	Reads a "short integer" (1 byte) from the data area.
ReadS7INT	Reads a "integer" (2 bytes) from the data area.
ReadS7DINT	Reads a "double integer" (4 bytes) from the data area.
ReadS7LINT	Reads "long integer" (8 bytes) from the data area.

Value	Description
ReadS7USINT	Reads a "unsigned short integer" (1 byte) from the data area.
ReadS7UINT	Reads a "unsigned integer" (2 bytes) from the data area.
ReadS7UDINT	Reads a "unsigned double integer" (4 bytes) from the data area.
ReadS7ULINT	Reads "unsigned long integer" (8 bytes) from the data area.
ReadS7REAL	Reads a "real number" (4 bytes) from the data area.
ReadS7LREAL	Reads a "long real number" (8 bytes) from the data area.
ReadS7S5TIME	Reads a 16 bit (2 bytes) from the data area.
ReadS7DATE	Reads a date value (2 bytes) from the data area.
ReadS7TIME	Reads a time value (4 bytes) from the data area.
ReadS7LTIME	Reads a time value (8 bytes) from the data area as nanoseconds.
ReadS7TIME_OF_DAY	Reads the time of day (4 bytes) from the data area.
ReadS7LTIME_OF_DAY	Reads the time of day (8 bytes) from the data area as nanoseconds since midnight.
ReadS7DATE_AND_TIME	Reads a general date and time area.
ReadS7DATE_AND_LTIME	Reads a date and time value (8 bytes) from the data area as nanoseconds since 01/01/1970 00:00.
ReadS7DTL	Reads a date and time information (12 bytes) as a predefined structure from the data area.
ReadS7CHAR	Reads a "char" (1 byte) from the data area.
ReadS7STRING_LEN	Reads the information of the string length for a S7 string in the data area.
ReadS7STRING	Reads an S7 string from the data area and returns it as language dependent string. The string is shortened when there is insufficient space in the target string.
ReadS7WCHAR	Only available for CPU function libraries for the Windows environment. Reads "wide char" (2 bytes) from the data area.
ReadS7WSTRING_LEN	Only available for CPU function libraries for the Windows environment. Reads the information of the string length for a S7W string in the data area.
ReadS7WSTRING	Only available for CPU function libraries for the Windows environment. Reads an S7W string from the data area and returns it as language dependent string. The string is shortened when there is insufficient space in the target string.

Class "CODK_CpuReadWriteData" (Windows and real-time environment)

The "CODK_CpuReadWriteData" class allows the following write accesses in addition to the all read accesses from "CODK_CpuReadData" to classic DBs / classic data:

Value	Description
CODK_CpuReadWriteData	Class constructor, initializes the output data area and the data size.
SetBuffer	Initializes the output data area and the data size.
LastByteChanged	Retrieves the index of the last byte changed in the data area.
FirstByteChanged	Retrieves the index of the first byte changed in the data area.
WriteS7BOOL	Writes a "bool" (1 byte) to the data area.
WriteS7BYTE	Writes a "byte" (1 byte) to the data area.
WriteS7WORD	Writes a "word" (2 bytes) to the data area.
WriteS7DWORD	Writes a "double word" (4 bytes) to the data area.
WriteS7LWORD	Writes a "long word" (8 bytes) to the data area.
WriteS7SINT	Writes a "short integer" (1 byte) to the data area.
WriteS7INT	Writes a "integer" (2 bytes) to the data area.
WriteS7DINT	Writes a "double integer" (4 bytes) to the data area.
WriteS7LINT	Writes a "long integer" (8 bytes) to the data area.
WriteS7USINT	Writes a "unsigned short integer" (1 byte) to the data area.
WriteS7UINT	Writes a "unsigned integer" (2 bytes) to the data area.
WriteS7UDINT	Writes a "unsigned double integer" (4 bytes) to the data area.
WriteS7ULINT	Writes a "unsigned long integer" (2 bytes) to the data area.
WriteS7REAL	Writes a "real number" (4 bytes) to the data area.
WriteS7LREAL	Writes a "long real number" (8 bytes) to the data area.
WriteS7S5TIME	Writes a 16-bit (2 bytes) time value to the data area.
WriteS7DATE	Writes a date value (2 bytes) to the data area.
WriteS7TIME	Writes a time value (4 bytes) to the data area.
WriteS7LTIME	Writes a time value (8 bytes) to the data area as nanoseconds.
WriteS7TIME_OF_DAY	Writes a time of day (4 bytes) to the data area.
WriteS7LTIME_OF_DAY	Writes the time of day (8 bytes) to the data area as nanoseconds since midnight.
WriteS7DATE_AND_TIME	Writes a "System.DateTime" to the data area.
WriteS7DATE_AND_LTIME	Writes a date and time value (8 bytes) to the data area as nanoseconds since 01/01/1970 00:00.
WriteS7DTL	Writes a date and time information (12 bytes) as a predefined structure to the data area.
WriteS7CHAR	Writes a "char" (1 byte) to the data area.
WriteS7STRING	Writes a S7 string to the data area. The string is shortened when there is insufficient space in the target string. If no maximum string length is set in the case of a "[OUT] Variant", the current string length is set as maximum string length.
WriteS7STRING_MAX_LEN	Only available for CPU function libraries for the Windows environment. Writes the maximum string length to an S7 string. Is only required for "[OUT] Variant".

Value	Description
WriteS7WCHAR	Only available for CPU function libraries for the Windows environment. Writes a "char" (2 bytes) to the data area.
WriteS7WSTRING	Only available for CPU function libraries for the Windows environment. Writes an S7W string to the data area. The string is shortened when there is insufficient space in the target string. If no maximum string length is set in the case of a "[OUT] Variant", the current string length is set as maximum string length.
WriteS7WSTRING_MAX_LEN	Only available for CPU function libraries for the Windows environment. Writes the maximum string length to an S7W string. Is only required for "[OUT] Variant".

D.2 C#/VB helper functions

Access to classic data

For the C# and VB programming languages, the following classes are available for reading and writing in a classic data stream:

- OdkReadVariant

  Supports all "ReadS7..." methods.

- OdkReadWriteVariant

  Supports all "ReadS7..." and "WriteS7..." methods.

ReadS7 methods	WriteS7 methods	Description
ReadS7Bool	WriteS7Bool	Writes/writes/reads a "bool" (1 byte) to/to/from the data area.
ReadS7Byte	WriteS7Byte	Writes/writes/reads a "byte" (1 byte) to/from the data area.
ReadS7Word	WriteS7Word	Writes/writes/reads a "word" (2 bytes) to/from the data area.
ReadS7DWord	WriteS7DWord	Writes/writes/reads a "double word" (4 bytes) to/from the data area.
ReadS7LWord	WriteS7LWord	Writes/writes/reads a "long word" (8 bytes) to/from the data area.
ReadS7Sint	WriteS7Sint	Writes/writes/reads a "short integer" (1 byte) to/from the data area.
ReadS7Int	WriteS7Int	Writes/writes/reads a "integer" (2 bytes) to/from the data area.
ReadS7Dint	WriteS7Dint	Writes/writes/reads a "double integer" (4 bytes) to/from the data area.
ReadS7Lint	WriteS7Lint	Writes/writes/reads a "long integer" (8 bytes) to/from the data area.
ReadS7USint	WriteS7USint	Writes/writes/reads a "unsigned short integer" (1 byte) to/from the data area.
ReadS7Uint	WriteS7Uint	Writes/writes/reads a "unsigned integer" (2 bytes) to/from the data area.

ReadS7 methods	WriteS7 methods	Description
ReadS7UDint	WriteS7UDint	Writes/writes/reads a "unsigned double integer" (4 bytes) to/from the data area.
ReadS7ULint	WriteS7ULint	Writes/writes/reads a "unsigned long integer" (8 bytes) to/from the data area.
ReadS7Real	WriteS7Real	Writes/writes/reads a "real number" (4 bytes) to/from the data area.
ReadS7LReal	WriteS7LReal	Writes/writes/reads a "long real number" (8 bytes) to/from the data area.
ReadS7S5Time	WriteS7S5Time	Writes/writes/reads a 16-bit (2 bytes) time value to/from the data area.
ReadS7Time	WriteS7Time	Writes/reads a time value (4 bytes) to/from the data area.
ReadS7LTime	WriteS7LTime	Writes/reads a time value (8 bytes) to/from the data area.
ReadS7Date	WriteS7Date	Writes/reads a date and time value (2 bytes) to/from the data area.
ReadS7TimeOfDay	WriteS7TimeOfDay	Writes/reads the time of day (4 bytes) to/from the data area.
ReadS7LTimeOfDay	WriteS7LTimeOfDay	Writes/reads the time of day (8 bytes) to/from the data area.
ReadS7DateAndTime	WriteS7DateAndTime	Writes/reads a "System.DateTime" to/from the data area.
ReadS7DateAndLTime	WriteS7DateAndLTime	Writes/reads a date and time value (8 bytes) to/from the data area as nanoseconds since 01/01/1970 00:00.
ReadS7Dtl	WriteS7Dtl	Writes/reads a date and time value (12 bytes) as a predefined structure to/from the data area.
ReadS7Char	WriteS7Char	Writes/reads a "char" (1 byte) to/from the data area.
ReadS7String	WriteS7String	Writes/reads an SIMATIC S7 string to/from the data area and returns it as language-dependent string. The string is shortened when there is insufficient space in the target string. If no maximum string length is set in the case of a "[OUT] Variant", the current string length is set as maximum string length.
ReadS7StringCurLen	-	Reads the current string length of a S7 string exception if the current string length is larger than the maximum string length.
ReadS7StringMaxLen	WriteS7StringMaxLen	Writes/reads the maximum string length to/from a S7 string. Is only required for "[OUT] Variant".
ReadS7WChar	WriteS7WChar	Writes/reads a "wide char" (2 bytes) to/from the data area.
ReadS7WString	WriteS7WString	Writes/reads an S7W string to/from the data area and returns it as language dependent string. The string is shortened when there is insufficient space in the target string. If no maximum string length is set in the case of a "[OUT] Variant", the current string length is set as maximum string length.
ReadS7WStringCurLen	-	Reads the current string length of a S7W string exception if the current string length is larger than the maximum string length.
ReadS7WStringMaxLen	WriteS7WStringMaxLen	Writes/reads the maximum string length to/from a S7W string. Only required for "[OUT] variant".

Access to classic DBs

Use in C#

```
using OdkInternal;

public ushort SampleFunction (byte[] myDB)
{
 OdkReadVariant rv = new OdkReadVariant(myDB);
 int i = rv.ReadS7DINT(0);
 // do something with i
 return ODK_SUCSESS;
}
```

Use in VB

```
Imports OdkInternal;

Public Function SampleFunction (ByRef myDB As Byte[]) As UShort
{
 Dim wv As OdkReadWriteVariant = new OdkReadWriteVariant(myDB)
 Dim value As Short = 5    ' calculate the value somehow
 wv.WriteS7INT(8, value)
 return ODK_SUCSESS;
}
```

Instructions for CPU function libraries

E

E.1 "Load" instruction

The "<*STEP7Prefix*>_Load" instruction has different parameters that depending on the development environment:

- Development of a CPU function library for the Windows environment (Page 54)
- Development of a CPU function library for the realtime environment (Page 91)

E.2 "Unload" instruction

The "<*STEP7Prefix*>_Unload" instruction has different parameters that depending on the development environment:

- Development of a CPU function library for the Windows environment (Page 60)
- Development of a CPU function library for the realtime environment (Page 96)

E.3 "GetTrace" instruction

The function block (Page 97) "GetTrace" is included in the default CPPfile "<project>.cpp".

GetTrace	
TraceCount	STATUS

The following table shows the parameters of the "GetTrace" function block:

Section	Declaration	Data type	Description
Output	STATUS	INT	Number of trace entries actually read
Input	TraceCount	INT	Number of trace entries to be read
Output	TraceBuffer	Array [0..255] of String[125]	Trace string array for the user Each trace string consists of: • Date • Time-of-day • OB number • File name • Line number • Trace text (trace implemented by the user)

Index

T

U

W