SIEMENS

SIMATIC

STEP 7 (TIA Portal) Options Open Development Kit 1500S V2.0

Programming and Operating Manual

09/2016

A5E35253941-AC

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 AG. The remaining trademarks in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owner.

Disclaimer of Liability

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

Siemens AG
Division Digital Factory
Postfach 48 48
90026 NÜRNBERG
GERMANY

A5E35253941-AC
℗ 08/2016 Subject to change

Preface

Purpose of the documentation

This document describes the special features of the Open Development Kit (ODK) V2.0.

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
- **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

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++

Validity of the documentation

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

- CPU 1505SP (F)
- CPU 1507S (F)
- CPU 1518-4 PN/DP ODK (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.

Security information

Siemens Industry Online Support

You can find current information on the following topics quickly and easily here:

- **Product support**

  All the information and extensive know-how on your product, technical specifications, FAQs, certificates, downloads, and manuals.

- **Application examples**

  Tools and examples to solve your automation tasks – as well as function blocks, performance information and videos.

- **Services**

  Information about Industry Services, Field Services, Technical Support, spare parts and training offers.

- **Forums**

  For answers and solutions concerning automation technology.

- **mySupport**

  Your personal working area in Industry Online Support for messages, support queries, and configurable documents.

This information is provided by the Siemens Industry Online Support in the Internet (http://www.siemens.com/automation/service&support).

Industry Mall

The Industry Mall is the catalog and order system of Siemens AG for automation and drive solutions on the basis of Totally Integrated Automation (TIA) and Totally Integrated Power (TIP).

Catalogs for all the products in automation and drives are available on the Internet (https://mall.industry.siemens.com).

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 obtain more information on our software update service under (http://w3.siemens.com/mcms/automation-software/en/software-update-service/Pages/Default.aspx).

Table of contents

Documentation guide 1

Introduction

You can find all information required to use the software in this documentation for the Open Development Kit (ODK).

Overview of the documentation for the CPU

The following table lists additional documents which supplement this description and are available on the Internet.

Table 1- 1 Documentation for the CPU

Topic	Documentation	Most important contents
Description of CPU 1505SP and CPU 1507S	Operating manual CPU 1505SP and CPU 1507S (http://support.automation.siemens.com/WW/view/en/90466248/133300)	This documentation describes the complete functionality of the CPU 1505SP and CPU 1507S.
Description of the CPU 1518-4 PN/DP ODK	Manual CPU 1518-4 PN/DP ODK (https://support.industry.siemens.com/cs/products?search=CPU%201518-4%20PN%2FDP%20ODK&mfn=ps&o=DefaultRankingDesc&lc=en-WW)	This documentation describes the full functionality of the CPU 1518-4 PN/DP ODK.
Web server	Function manual Web Server (http://support.automation.siemens.com/WW/view/en/59193560)	Basics Function Operation Diagnostics via web server

Product overview 2

2.1 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 provides an interface 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.

The ODK applications must be used in the STEP 7 program .

Structure and design of an ODK application

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

ODK supports the following templates:

- A supplied template for programming in Microsoft Visual Studio. This allows you to generate a DLL file.
- Another template for programming in Eclipse. This allows you to generate an SO file. ODK also supplies a class library for Eclipse.

You create an ODK application with the C++ programming language. ODK applications can be created for both the Windows and the realtime environment.

The ODK program can be executed in the following ways:

- Synchronously, i.e. operates as part of the CPU cycle (execution in the real-time environment)
- Asynchronously, i.e. started by the CPU program and finished in the background (execution in the Windows environment)

The program that runs outside of the CPU is created with Microsoft Visual Studio or Eclipse and is generated as a DLL or SO file. ODK applications can run both under Windows (DLL) and in the realtime 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 possible functions in an ODK application. There are specific function blocks for an ODK Application:

- Loading and unloading of the ODK application
- One function block each for calling a certain function

The following illustration provides a schematic overview of how ODK applications run on a PC. This graphic applies to the S7-1500 Software Controller.

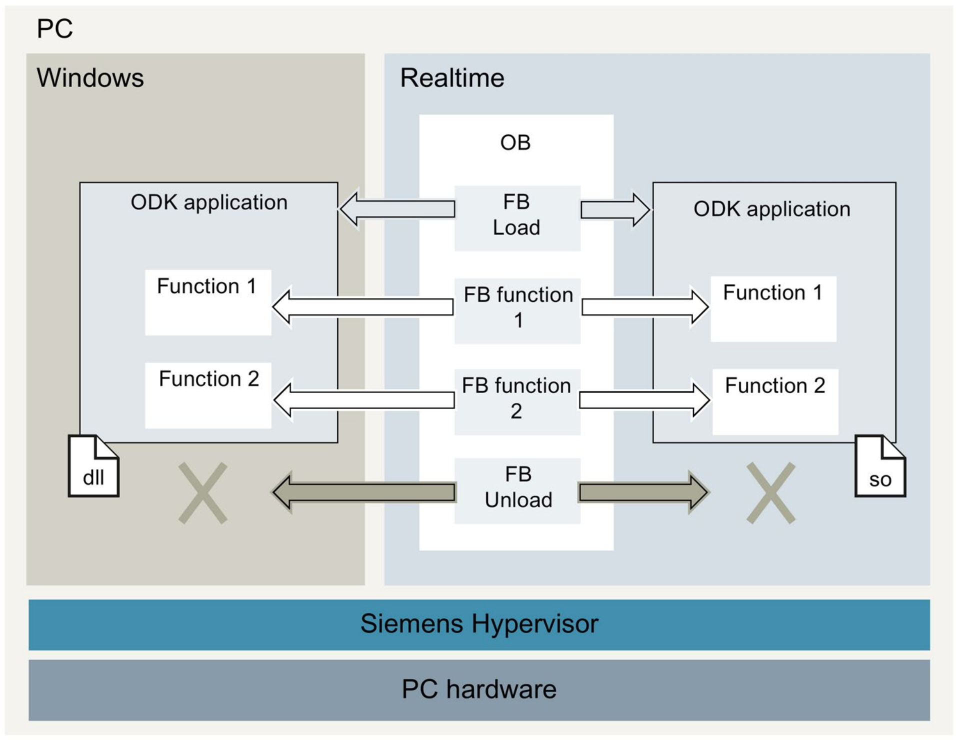


Figure 2-1 Running an ODK application on a PC

The following illustration provides a schematic overview of how ODK applications run on a hardware CPU.

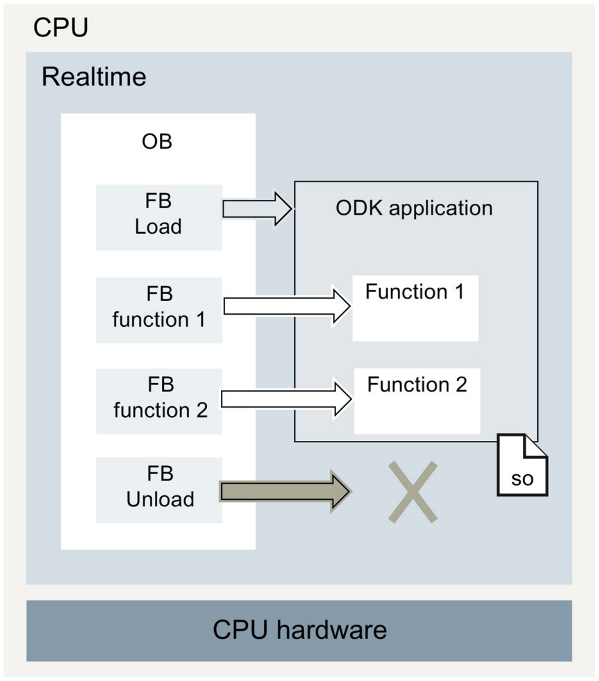


Figure 2-2 Running an ODK application on a hardware CPU

2.2 Development environments

An ODK application is written in a standard development environment.

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

- Microsoft Visual Studio for Windows applications (DLL file)
- Eclipse for realtime applications (SO file)

Microsoft Visual Studio as a development environment

Use Microsoft Visual Studio. To help you develop an ODK application, 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 "Visual C++" when a new project is created.

Eclipse as a development environment

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

2.3 Basic procedure

The following sections describe the development tasks and procedures for the development and execution of an ODK Application:

- Developing ODK application for the Windows environment (Page 17)
- Developing ODK application for the realtime environment (Page 49)

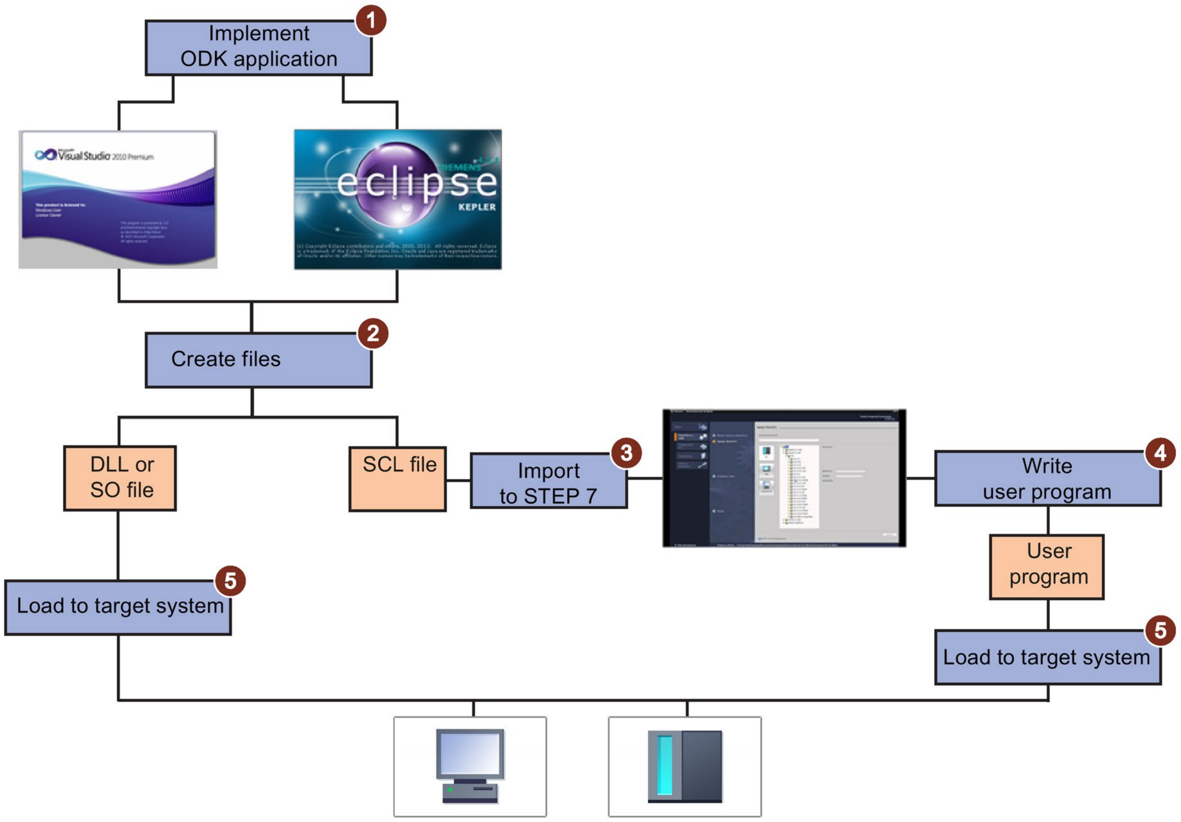


Figure 2-3 Overview of the development steps

Overview of the development steps

To develop and execute a ODK application, follow these steps:

1. Implement your function in Microsoft Visual Studio for Windows applications (DLL file) or in Eclipse for realtime applications (SO file).
2. Create the 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 DLL or SO file into the target system.

Result

Your ODK application is downloaded to the target system and is loaded and executed by the user program in STEP 7.

Installation 3

3.1 System Requirements

Requirements

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

Category	Requirements
Operating system	• Microsoft Windows 7, 64-bit • Microsoft Windows 8, 64-bit • Microsoft Windows 10, 64-bit
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	1.6 GB of free space on the hard disk C:\ for the full 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) V14 or higher
Supported PLCs	All SIMATIC CPUs supporting ODK (see next table)
Additional software	Not included in the product package: • Java Runtime 32-bit as V1.6 (for Eclipse) • Microsoft Visual Studio C++ 2010 SP1 • Microsoft Visual Studio C++ 2012 • Microsoft Visual Studio C++ 2013 • Microsoft Visual Studio express C++ 2013 • Microsoft Visual Studio C++ 2015 • Microsoft Visual Studio express C++ 2015 Microsoft Development Tool: Download Center (http://www.microsoft.com/en-us/download/developer-tools.aspx)

ODK 1500S V2.0 is compatible with the following devices (support of loadable function libraries is device-dependent):

	DLL (Windows)	SO (Real-time)
CPU 1505SP (F) V2.0	Yes	Yes
CPU 1507S (F) V2.0	Yes	Yes
CPU 1518-4 PN/DP ODK (F) V2.0	No	Yes

3.2 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 versions on one PC at the same time. If the ODK version to be installed is already installed on the PC, you must first uninstall it or perform a repair installation.

Note

Close applications before a repair installation/uninstall

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

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. Follow the instructions of the installation wizard.

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.

The setup program installs the following components:

- "Eclipse" for the development of ODK applications for the realtime environment
- ODK templates for Visual Studio
- Code generator
- Online help

3.3 Integrating ODK templates in Visual Studio after installation

When Visual Studio is already installed, the ODK template is automatically installed during the ODK installation. If Visual Studio is installed later, you have the following options to integrate the ODK template:

- 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 ODK template for Visual Studio is installed. You can find this under the corresponding programming language.

3.4 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 ODK application for the Windows environment 4

4.1 Creating an ODK application

4.1.1 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 (http://www.microsoft.com/en-us/download/developer-tools.aspx).

4.1.2 Creating a project

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

Procedure

To create an ODK project in Microsoft Visual Studio using the ODK 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.

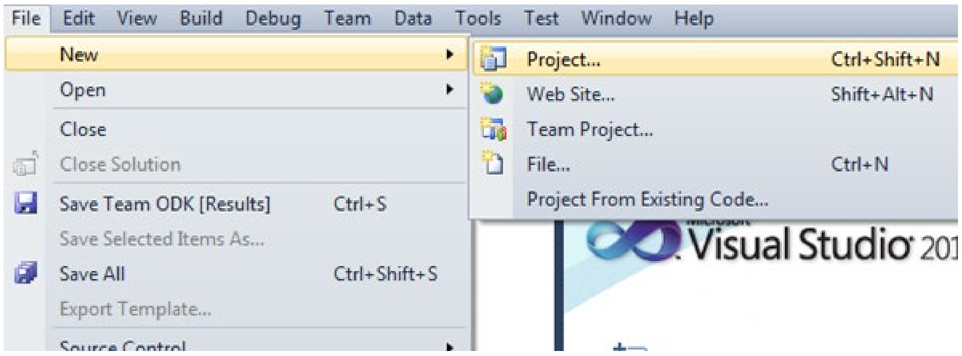


Figure 4-1 Creating a new project in Visual Studio

3. Select your preferred programming language and the corresponding ODK template.
4. Enter a project name.
5. Click "OK" to confirm.

Result

The ODK project is created using the ODK templates and sets the following project settings:

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

The ODK template sets up the following file structure as standard:

Folder / file			**Description**
<project path>			
	<project>.rc		
	<project>.cpp		Function code: This file has always the suffix CPP, regardless of whether you are creating a C or C++ project.
	dllmain.cpp		Implementation of the "dllmain" file
	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
	cg_src_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_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
	debug		
		<project>.dll	ODK application binary (debug version)
	release		
		<project>.dll	ODK application binary (release version)

The ODK template supports the following applications:

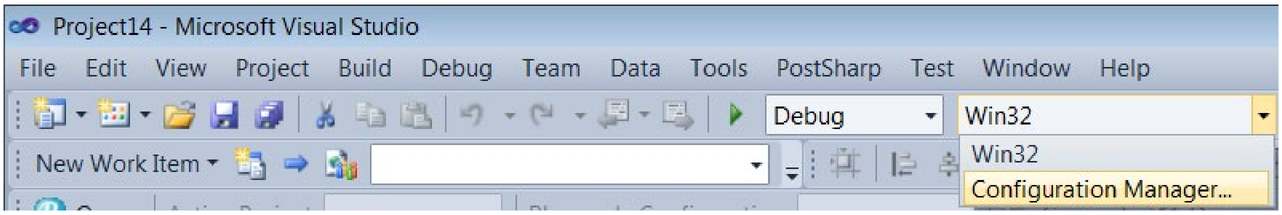
Configuration and platform	Visual Studio Version older than 2015	Visual Studio 2015
Debug Win32	Yes	Yes
Release Win32	Yes	Yes
Debug x64	to be manually created (Page 19)	Yes
Release x64	to be manually created (Page 19)	Yes

4.1.2.1 Creating an ODK project with Visual Studio version older than 2015

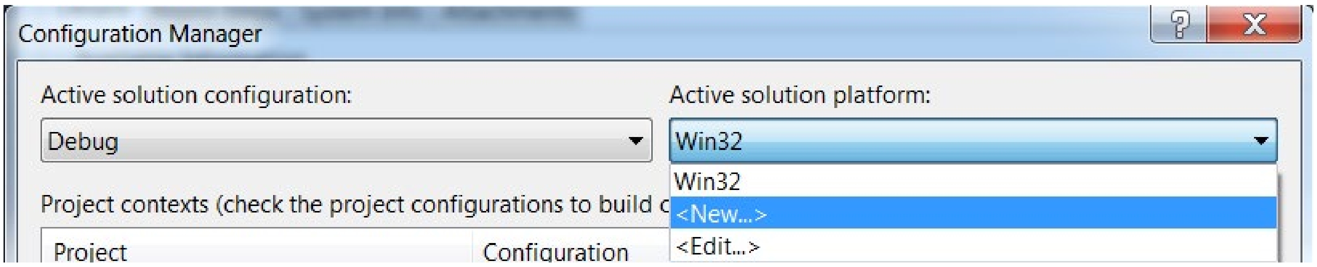
Procedure

To create an ODK template for a x64 platform with a Visual Studio version older than 2015, follow these steps:

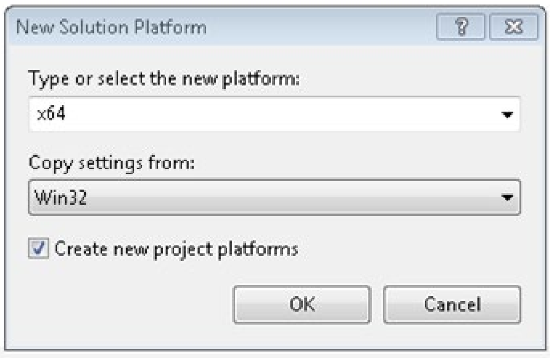
1. Open the "Configuration Manager".


Project14 - Microsoft Visual Studio
File Edit View Project Build Debug Team Data Tools PostSharp Test Window Help
Debug
Win32
New Work Item
Win32
Configuration Manager...

2. Create an x64 platform.

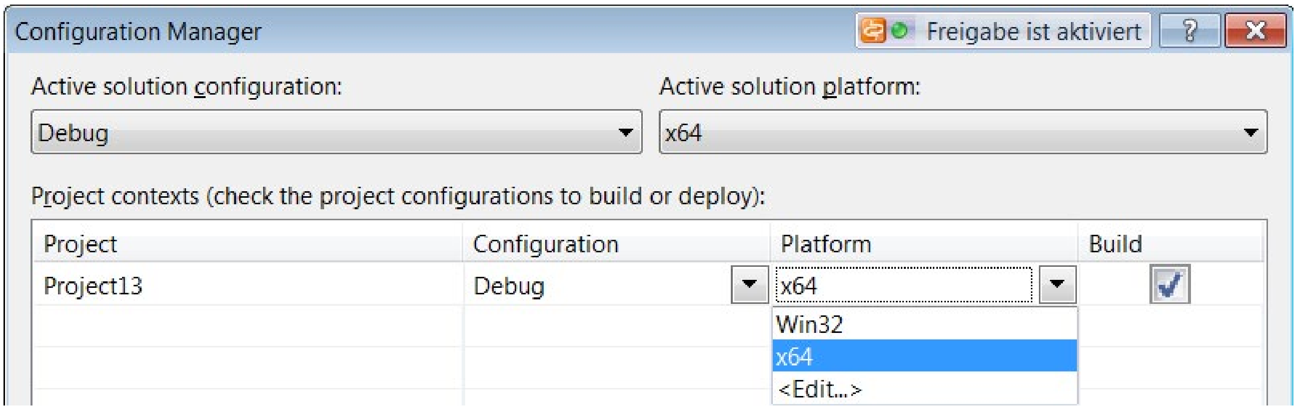

Configuration Manager
Active solution configuration:
Debug
Active solution platform:
Win32
Project contexts (check the project configurations to build c
Project
Configuration
Win32
<New...>
<Edit...>

The "New Solution Platform" dialog opens.


New Solution Platform
Type or select the new platform:
x64
Copy settings from:
Win32
Create new project platforms
OK
Cancel

Under "Copy settings from:" , select "Win32".

3. Define a solution configuration for a x64 platform.


Configuration Manager
Freigabe ist aktiviert

Active solution configuration: Debug
Active solution platform: x64

Project contexts (check the project configurations to build or deploy):

Project	Configuration	Platform	Build
Project13	Debug	x64	✓

4. Under "Active solution configuration, select "Debug" or "Release" and under "Platform", select "x64".

4.1.3 Generating an ODK application

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 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

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

4.1.4 Defining runtime properties of an ODK application

Next, define the interface description of the ODK application 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	Defines that the ODK application is loaded in a context of a Windows user (Page 22).
	system	Defines that the ODK application is loaded in a context of the Windows system (Page 22).
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

Spaces in the STEP 7 prefix are replaced by an underscore.

4.1.5 Environment for loading or running the ODK application

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 ODK application regardless of the STEP 7 user program after programming and the initial loading. You can load up to 32 ODK applications.

Depending on whether you have created the ODK application for a 32-bit or 64-bit system, this is loaded into a 32-bit or 64-bit ODK host process.

You can choose one of two contexts for your ODK application:

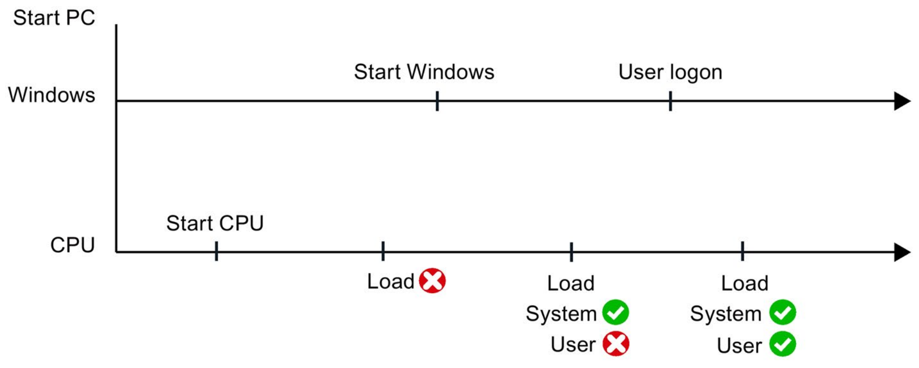
- **"System" context**

  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 an ODK application can be loaded depending on context.



"System" context

To use the ODK application in the system context (Session 0), change the following code line in the <Project>.odk file:
```
Context=system
```

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

"User" context

To use the ODK application in the user context, change the following code line in the <Project>.odk file:

```
Context=user
```

When you load the ODK application in the user context, it automatically unloads as soon as the user logs off in Windows. The ODK application 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 ODK application loads or unloads for the user, who has the current screen rights until he logs off in Windows.

4.1.6 Defining functions and structures of an ODK application

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 divide function definitions into multiple 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 keywords for the utilized programming language (e.g. "INT" 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 *<STEP7Prefix>* 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]: Defines an input tag. The tag is copied to the function when it is called. This is constant and cannot be changed.
- [OUT]: Defines an output tag. The tag is copied back after the function has been completed.
- [INOUT]: Defines 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	unsigned long long	64-bit during in nanoseconds
ODK_TIME	TIME	unsigned long	32-bit during in milliseconds
ODK_LDT	LDT	unsigned long long	64-bit date and time of the day in nanoseconds
ODK_LTOD	LTOD	unsigned long long	64-bit time of the day in nano-seconds since midnight
ODK_TOD	TOD	unsigned long	32-bit time of the day in milli-seconds since midnight
ODK_WCHAR	WCHAR	wchar_t	16-bit character
ODK_CHAR	CHAR	char	8-bit character

- **Complex data types:**

ODK data type	SIMATIC data type	C++ data type	Description
ODK_DTL	DTL	struct ODK_DTL	Structure for date and time
ODK_S7WSTRING	WSTRING	unsigned short	Character string (16-bit character) with max. und act. length (2xUINT)
ODK_S7STRING	STRING	unsigned char	Character string (8-bit character) with max. and act. length (2xUSINT)
ODK_CLASSIC_DB	VARIANT	struct ODK_CLASSIC_DB	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 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.

  – You must not use keywords for the generated language (e.g. "int" as tag name).

  You can create additional structures within a structure.

<StructName>

Structure names apply 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 by 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);
```

4.1.6.1 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 96) of the following classes are available:

- Class "CODK_CpuReadData"
- Class "CODK_CpuReadWriteData"

4.1.6.2 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 96) 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.

4.1.6.3 Definition of the <Project>.odk file

The function prototypes and function blocks are generated based on the selected parameters in the <Projekt>.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 ODK application is loaded in the "User" context. You can change the parameter to Context=system.

- STEP7Prefix="<Projekt>"

  Sets a string for the SCL generation in front of the functions of the ODK application. The string 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=user
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_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_CLASSIC_DB  VARIANT  classic DB (global or based on UDT
*                           "optimized block access" must be
unchecked)
*  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
```

4.1.6.4 Modifying the <Project>.odk file

The following example shows how you can adapt the <Project>.odk file to 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]);
```

4.1.6.5 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 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 ...
                                                 ...
                                               */
```

4.1.7 Implementing functions

4.1.7.1 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 ODK functions are executed with equal priority – independent of the priority of the OBs
- The complete Windows API (Application Programming Interface) and C++-Runtime library are available

4.1.7.2 Callback functions

The ODK project contains a CCP file (**execute file:** <Project>.cpp) to define your functions. This CCP 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):

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

You can define the following functions in the CCP file:

- OnLoad(): Called after loading the ODK application
- OnUnload(): Called before unloading the ODK application
- 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 execut-ed) 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 ODK application unloads for the "OnLoad()" function. The ODK application 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).

4.1.7.3 Implementing custom functions

Once you have defined the ODK interface in the <Project>.odk file, you must edit the ODK application functions in the CPP file.

Procedure

To edit the ODK application functions, follow these steps:

1. Execute the build in order to update the header file <ODK_Functions.h>.
2. Open the <Project>.cpp file or create your own source file if required.
3. Transfer the function prototypes from <ODK_Functions.h> to the source file.

Note

To skip step 3 in the future when function parameters are changed, use the define of the function prototype.

4. Edit the code of your ODK application in the CPP file.

ODK application

The CCP file contains an 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.

```
#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;
}
```

4.2 Transferring an ODK application 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 stick). Use the standard Windows data transfer procedure to transfer of the ODK application. The storage location in Windows is specified by a registry key. When loading an ODK application, the ODK service automatically searches for the file in the path specified by the registry key.

Note

ODK application in the debug configuration

When the ODK application 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

To access this folder, you need administrator rights. This prevents the import of ODK applications by unauthorized persons.

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" in the Explorer menu "Organize > Folder and search options > View".

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\od
k_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.

4.3 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.

4.4 Executing a function

4.4.1 Loading functions

Introduction

Regardless of the context in which the ODK application 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 ODK application runs in its own Windows process (ODK_Host).
- The host loads the ODK application and calls the "OnLoad()" function and then the "OnRun()" functions.

Note

Loading the same ODK applications with a modified <Project>.odk file

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

"*<STEP7Prefix>*_Load" instruction

An ODK application 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 ODK application.
Output	DONE	BOOL	Indicates that the instruction has finished loading the ODK application.
Output	BUSY	BOOL	Indicates that the instruction is still loading the ODK application.

Section	Declaration	Data type	Description
Output	ERROR	BOOL	Indicates that an error occurred during the loading of the ODK application. 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 ODK application.

Input parameters

A 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: ODK application has already been loaded.
1	0	0	0x0000 =0	Loading was performed successfully.
0	0	1	0x80A4 =-32604	ODK application could not be loaded. Start the ODK service manually or restart Windows.
			0x80C2 =-32574	ODK application could not be loaded. There is currently not enough memory available at the Windows end. Load the ODK application again after a few seconds.
			0x80C3 =-32573	ODK application could not be loaded. The CPU currently does not have enough memory. Load the ODK application again after a few seconds.
			0x8090 =-32624	ODK application could not be loaded. An exception occurred during execution of the "OnLoad()" function.
			0x8092 =-32622	ODK application could not be loaded because the library name is invalid.
			0x8093 =-32621	ODK application could not be loaded because the ODK application could not be found. Check the file name and path of the file.
			0x8094 =-32620	ODK application could not be loaded. The ODK application was created for the Windows user context, but no user is logged on.

DONE	BUSY	ERROR	STATUS	Meaning
			0x8095 =-32619	ODK application could not be loaded for the following reasons: • The DLL file is not an ODK application • 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 whether the release build of the ODK application is being 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 ODK application could not be loaded because the internal identification is already being used by another loaded ODK application.
			0x8097 =-32617	CPU 1500 V1.8 and earlier: ODK application has already been loaded.
			0x8098 =-32616	ODK application could not be loaded because the ODK application is currently being unloaded.
			0x809B =-32613	CPU 1500 V2.0 and later: The ODK application could not be loaded and returns an invalid value (the values 0x0000 and 0xF000 - 0xFFFF are permitted)
			0xF000 – 0xFFFF =-4096 – -1	CPU 1500 V2.0 and later: ODK application could not be loaded. An error occurred during execution of the "OnLoad()" function.

Example

This example describes how the loading and running of a Windows ODK application can be configured in order to start Windows again after communication disturbances.

When Windows is available again, the ODK application is loaded and the functions can be executed again.

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
```

```
  //Load the Windows ODK application
  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;

    //Load the ODK application
    "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 ODK application 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
```

4.4.2 Calling functions

Introduction

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

You can load up to 32 ODK applications at one time.

"*<STEP7Prefix>*SampleFunction" instruction

An ODK application 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 ODK application.
Output	DONE	BOOL	This output value indicates that the instruction has finished execution of the ODK application.
Output	BUSY	BOOL	This output value indicates that the instruction is still unloading the ODK applica-tion.
Output	ERROR	BOOL	This output value indicates that an error occurred during the execution of the ODK application. 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 ODK application.
User-defined parameter			
Input	myInt		User-defined input tags
InOut	myReal		User-defined input-output tags
Output	myBool		User-defined output tags

Input parameters

A 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	ODK application 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 ODK application again. • Windows is not available • ODK service is not running Start the ODK service manually or restart Windows.
			0x80C2 =-32574	ODK application could not be run. There is currently not enough memory available at the Windows end. Load the ODK application again after a few seconds.
			0x80C3 =-32573	ODK application could not be run. The CPU currently does not have enough memory. Load the ODK application again after a few seconds.
			0x8090 =-32624	ODK application could not be run. An error occurred during execution.
			0x8091 =-32623	ODK application could not be run. A "STOP" occurred during the function call.
			0x8096 =-32618	ODK application could not be executed because the ODK application was not loaded or unloading is not yet finished.
			0x8098 =-32616	ODK application could not be executed because the function is not supported.
			0x8099 =-32615	ODK application could not be executed because the maximum amount of input data (32 KB) was exceeded (declarations with "In" and "InOut")
			0x809A =-32614	ODK application could not be executed because the maximum amount of output data (32 KB) 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 If you are using an ODK_CLASSIC_DB, disable the optimized block access.
			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)

4.4.3 Unloading functions

Introduction

The ODK application is unloaded be calling the "*<STEP7Prefix>*_Unload" instruction. Call is made from the STEP 7 user program.

In addition to this call, the ODK application is also automatically unloaded for the following reasons.

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

Regardless of the context in which the ODK application 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 these ODK application. 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 ODK application is unloaded.

"*<STEP7Prefix>*_Unload" instruction

An ODK application 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 ODK application.
Output	DONE	BOOL	Indicates that the instruction has finished unloading the ODK application.
Output	BUSY	BOOL	Indicates that the instruction is still unloading the ODK application.
Output	ERROR	BOOL	Indicates that an error occurred during the unloading of the ODK application. 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 ODK application.

Input parameters

A 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	ODK application could not be unloaded for the following reasons: • Windows is not available Start the ODK service manually or restart Windows.
			0x80C2 =-32574	ODK application could not be unloaded. There is currently not enough memory available at the Windows end. Load the ODK application again after a few seconds.
			0x80C3 =-32573	ODK application could not be unloaded. The CPU currently does not have enough memory. Load the ODK application again after a few seconds.
			0x8090 =-32624	ODK application could not be unloaded. An exception occurred during execution of the "OnUnload()" function.
			0x8096 =-32618	ODK application could not be unloaded because the ODK application was not loaded or unloading is not yet finished.
			0x809B =-32613	CPU 1500 V2.0 and later: The ODK application could be unloaded and returns an invalid value (the values 0x0000 and 0xF000 - 0xFFFF are permitted)
			0xF000 – 0xFFFF =-4096 – -1	CPU 1500 V2.0 and later: ODK application could be unloaded. An error occurred in the CCX object during execution of the "OnLoad()" function.

4.5 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 an ODK application on a target system without Visual Studio. It should be noted that the generated ODK applications (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.

Installed Visual Studio version	**Link to the Download Center for the remote debugger**
Microsoft Visual Studio 2010	Microsoft Visual Studio 2010 Remote Debugger (https://www.microsoft.com/en-us/download/details.aspx?id=475)
Microsoft Visual Studio 2012	Microsoft Visual Studio 2012 Remote Debugger (https://www.microsoft.com/en-us/download/details.aspx?id=38184)
Microsoft Visual Studio 2013	Microsoft Visual Studio 2013 Remote Debugger (https://www.microsoft.com/en-us/download/details.aspx?id=44918)
Microsoft Visual Studio 2015	Microsoft Visual Studio 2014 Remote Debugger (https://www.microsoft.com/en-us/download/details.aspx?id=48155)

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

4.5.1 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. Copy the Visual Studio C++ debug DLLs from the "<InstallationPath VS>\VC\redist\Debug_NonRedist\<ApplicationType>\Microsoft.<VS version>.DebugCRT" folder to the destination folder.
   – 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

   If you are using Visual Studio 2015, you also need the "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)>

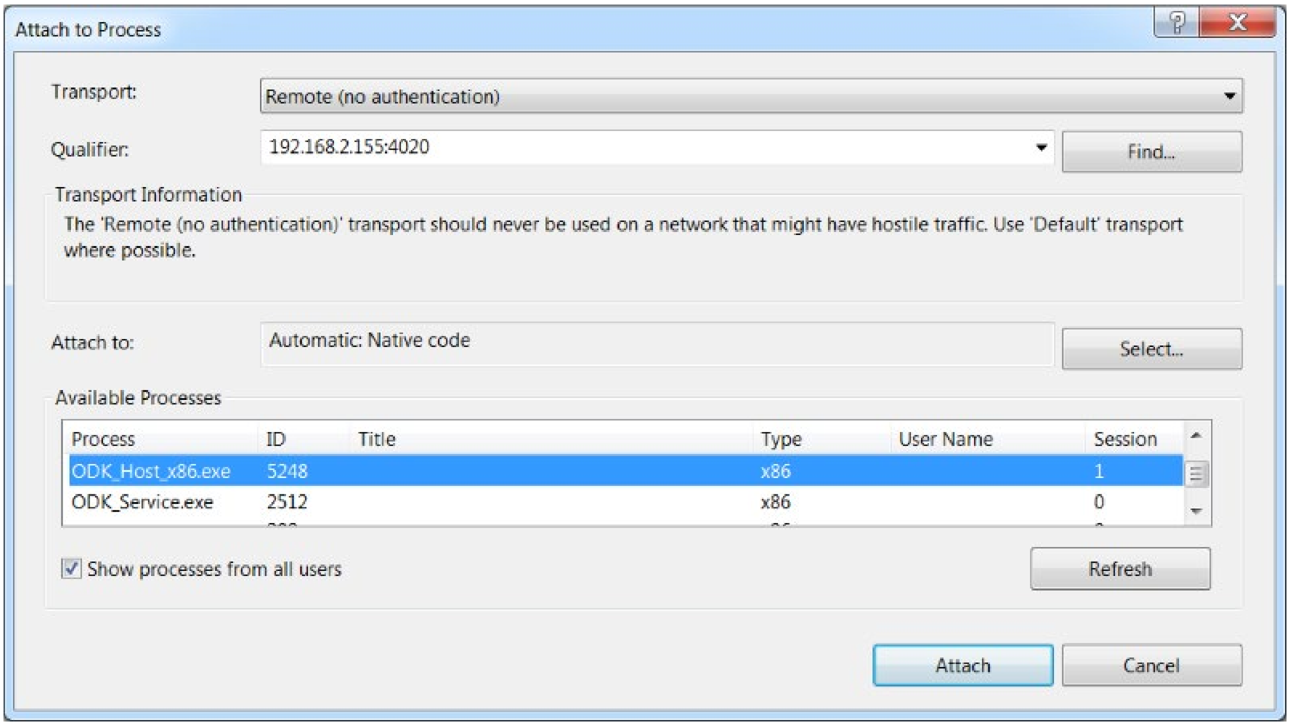
   ---
4. Copy the ODK application to the "C:\ProgramData\Siemens\Automation\ODK1500S" folder of the target system.

Note

If the ODK application is loaded, unload (Page 43) it before copying.

---

5. Load (Page 37) the ODK application on the target system.
6. Set the break points in the source code and attach the debugger using "Debug > Attach to Process…".



Attach to Process

Transport: Remote (no authentication)

Qualifier: 192.168.2.155:4020

Transport Information

The 'Remote (no authentication)' transport should never be used on a network that might have hostile traffic. Use 'Default' transport where possible.

Attach to: Automatic: Native code

Available Processes

Process	ID	Title	Type	User Name	Session
ODK_Host_x86.exe	5248		x86		1
ODK_Service.exe	2512		x86		0

[✓] Show processes from all users

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 ODK application for the realtime environment 5

5.1 Creating an ODK application

5.1.1 Requirements

- ODK is installed. The Eclipse development environment is installed.
- You need administrator rights to create and edit an ODK project.

Note

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

5.1.2 Creating a project

To help you develop an ODK application, an ODK template for an ODK project is included in the installation of ODK 1500S .

Procedure

To create an ODK 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.

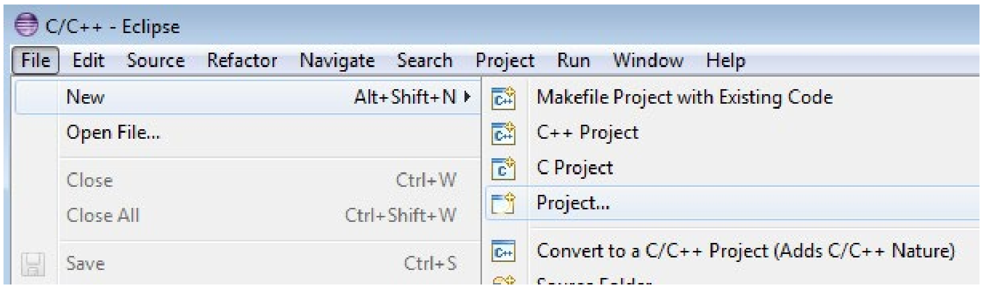


Figure 5-1 Creating a new project with Eclipse

3. Select your preferred programming language and the corresponding ODK template.

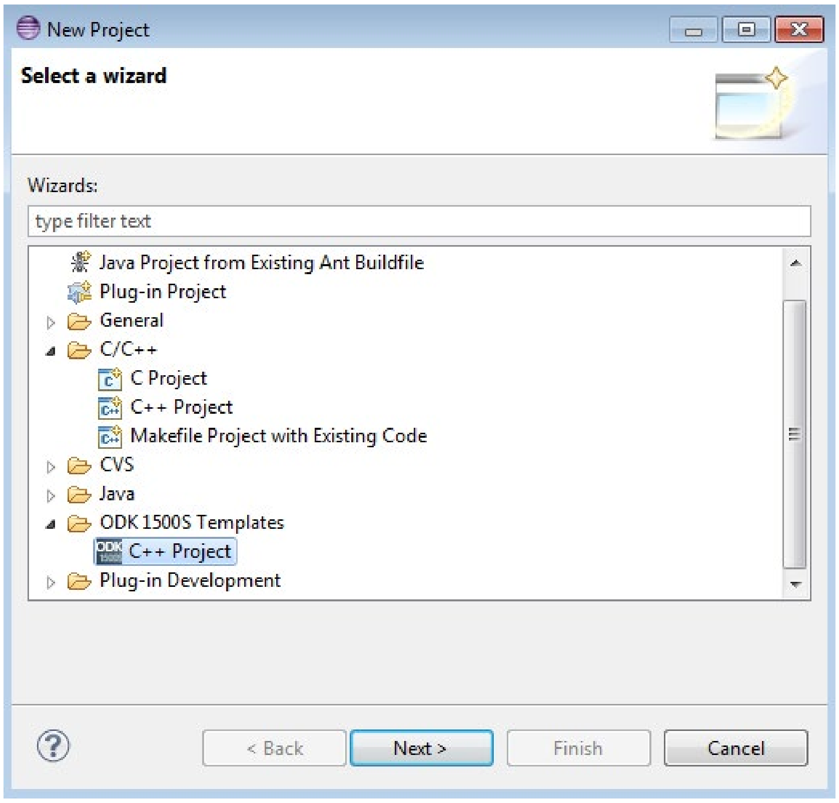


Figure 5-2 Selecting a template

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

Result

The ODK project is created using the ODK templates and sets the following project settings:

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

The ODK template configures the following data structure 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
	cg_src_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.
	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

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".

5.1.3 Generating an ODK application

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

5.1.4 Defining runtime properties of an ODK application

Next, define the interface description of the ODK application 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	Defines that the ODK application is loaded in the context of the realtime environment (Page 54).
Trace	on	Defines the trace function in the ODK application. In this case, the ODK application requires 32K if memory as an additional trace buffer. A "Get-Trace" 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 89)>]k	Defines a memory in KB that can be used as heap for these realtime applications.
HeapMaxBlockSize	[8…<HeapSize>]	Defines the maximum memory size in bytes that can be allocated at one time.
SyncCallParallelCount	[1...9] Default=3	Optional parameter that defines the maximum number of parallel calls in this ODK application. The size of the memory that is reserved for calls in this ODK application is: SyncCallParallelCount * (SyncCallStackSize + SyncCallDataSize)
SyncCallStackSize	[1...1024]k Default=32k	Optional parameter that defines the size of the thread stack for one call in this ODK application. Each new call receives its own stack memory.
SyncCallDataSize	[1...1024]k	Optional parameter that defines the size of the data area for one call in this ODK application. 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
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.

5.1.5 Environment for loading or running the ODK application

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 ODK application regardless of the STEP 7 user program after programming and the initial loading. You can load up to 32 ODK applications.

You can load and run your ODK application in the context of the realtime environment:

Realtime environment

Add the following line of code in your <Projekt>.odk file to use the ODK application in the context of the realtime environment:

```
Context=realtime
```

In this context, the ODK application is running in the realtime environment instead of a host process at the Windows end. Because the ODK application is loaded synchronously, it should be loaded in a startup OB (e.g. OB 100).

The number of loadable ODK applications (Page 89) is limited in the context of the realtime environment.

Determining the size of the ODK application in the CPU memory

To determine the required size of the ODK application 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_4.8.1-1\bin\x86_64-pc-elf-readelf.exe “*StorageLocation\File.so>*” -l

   You can see the size of your ODK application under the heading "Program Headers" in the "MemSiz" column.

   In addition to the size specified here, additional administrative memory is needed for each ODK application. The administrative memory can be calculated as follows:

   Administrative memory = SyncCallParallelCount * (SyncCallStackSize + SyncCallDataSize)

5.1.6 Defining functions and structures of an ODK application

5.1.6.1 Defining functions of an ODK application

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 divide function definitions into multiple 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 keywords for the utilized programming language (e.g. "INT" 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 *<STEP7Prefix>* 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]: Defines an input tag. The tag is copied to the function when it is called. This is constant and cannot be changed.
- [OUT]: Defines an output tag. The tag is copied back after the function has been completed.
- [INOUT]: Defines 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	unsigned long long	64-bit during in nanoseconds
ODK_TIME	TIME	unsigned long	32-bit during in milliseconds
ODK_LDT	LDT	unsigned long long	64-bit date and time of the day in nanoseconds
ODK_LTOD	LTOD	unsigned long long	64-bit time of the day in nano-seconds since midnight
ODK_TOD	TOD	unsigned long	32-bit time of the day in milli-seconds 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	struct ODK_DTL	Structure for date and time
ODK_S7STRING	STRING	unsigned char	Character string (8-bit character) with max. and act. length (2xUSINT)
ODK_CLASSIC_DB	VARIANT	struct ODK_CLASSIC_DB	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.

  – You must not use keywords for the generated language (e.g. "int" as tag name).

  You can create additional structures within a structure.

<StructName>

Structure names apply 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 by 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);
```

See also

Reading the trace buffer (Page 81)

Helper functions (Page 96)

5.1.6.2 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 96) of the following classes are available:

- Class "CODK_CpuReadData"
- Class "CODK_CpuReadWriteData"

5.1.6.3 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 96) 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.

5.1.6.4 Definition of the <Project>.odk file

The function prototypes and function blocks are generated based on the selected parameters in the <Projekt>.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 ODK application is loaded in the context of the realtime environment.
- Trace=on

  Defines the trace function in the ODK application. A "GetTrace" function block is created by default for use in a STEP 7.

  When you define the "ODK_TRACE" instruction (Page 81), 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

  Defines a memory in KB that can be used as heap for these realtime applications.
- HeapMaxBlockSize

  Defines the maximum memory size in bytes that can be allocated at one time.
- STEP7Prefix="<Projekt>"

  Sets a string for the SCL generation in front of the functions of the ODK application. 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);
```

5.1.6.5 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]);
```

5.1.6.6 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 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 ...
                                                 ...
                                               */
```

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 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 81) to check variables or the execution of functions in the realtime environment.
- The execution of synchronous ODK functions can be interrupted by higher priority OBs (Page 77) running in the same CPU.
- Application size

  The number of loadable ODK applications (Page 54) is limited in the context of the realtime environment.
- C++ Runtime library

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

5.1.7.2 Callback functions

The ODK project contains a CCP file (**execute file:** <Project>.cpp) to define your functions. This CCP 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 CCP file:

- OnLoad(): Called after loading the ODK application
- OnUnload(): Called before unloading the ODK application
- 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()

"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 ODK application unloads for the "OnLoad()" function. The ODK application 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.

5.1.7.3 Implementing custom functions

Once you have defined the ODK interface in the <Project>.odk file, you must edit the ODK application functions in the CPP file.

Procedure

To edit the ODK application functions, follow these steps:

1. Execute the build in order to update the header file <ODK_Functions.h>.
2. Open the <Project>.cpp file or create your own source file if required.
3. Transfer the function prototypes from <ODK_Functions.h> to the source file.

Note

To skip step 3 in the future when function parameters are changed, use the define of the function prototype.

4. Edit the code of your ODK application in the CPP file.

ODK application

The CCP file contains an 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.

```
#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;
}
```

5.1.7.4 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 instructions newdelete or mallocfree
- STL (StandardTemplateLibrary)
- 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 89). 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 ODK applications, 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 ODK application (Page 54).

- 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
```

5.1.7.5 Debug (Test)

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

Requirements

You need an Internet connection for this procedure.

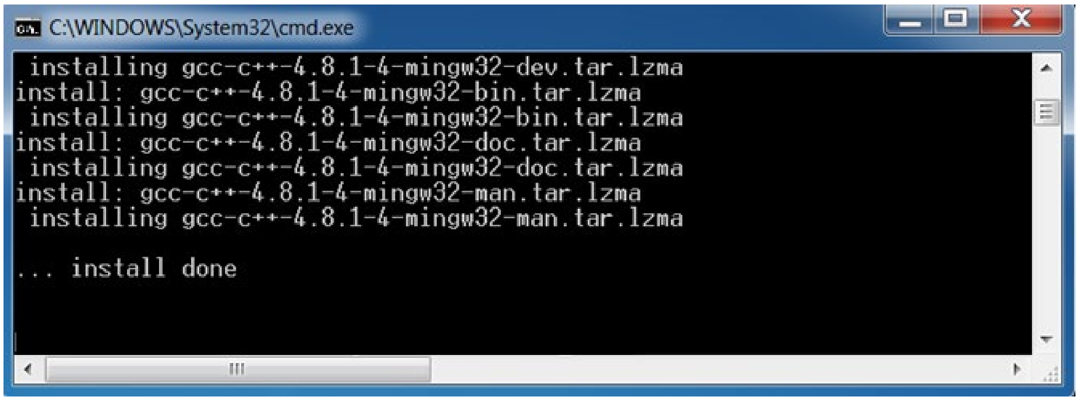
You need administrator rights for this procedure.

Procedure before the first debug process

To perform a test on a realtime application 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.



```
 installing gcc-c++-4.8.1-4-mingw32-dev.tar.lzma
install: gcc-c++-4.8.1-4-mingw32-bin.tar.lzma
 installing gcc-c++-4.8.1-4-mingw32-bin.tar.lzma
install: gcc-c++-4.8.1-4-mingw32-doc.tar.lzma
 installing gcc-c++-4.8.1-4-mingw32-doc.tar.lzma
install: gcc-c++-4.8.1-4-mingw32-man.tar.lzma
 installing gcc-c++-4.8.1-4-mingw32-man.tar.lzma

... install done
```

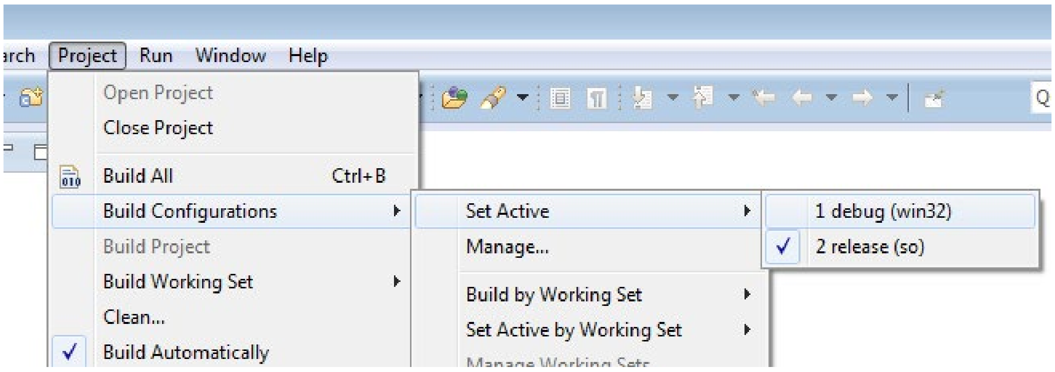
4. Click on any button.

   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".

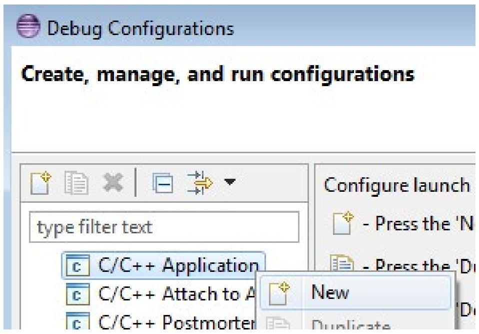


3. Create the project as debug version. To do so, select the "Build Project" command in the "Project " menu.
4. If you debug the project for the first time, you must now set the debug configuration. Otherwise, continue with step 8.

5. To do this, select the "Debug Configurations" command in the "Run" menu.

   The "Debug Configurations" dialog opens.

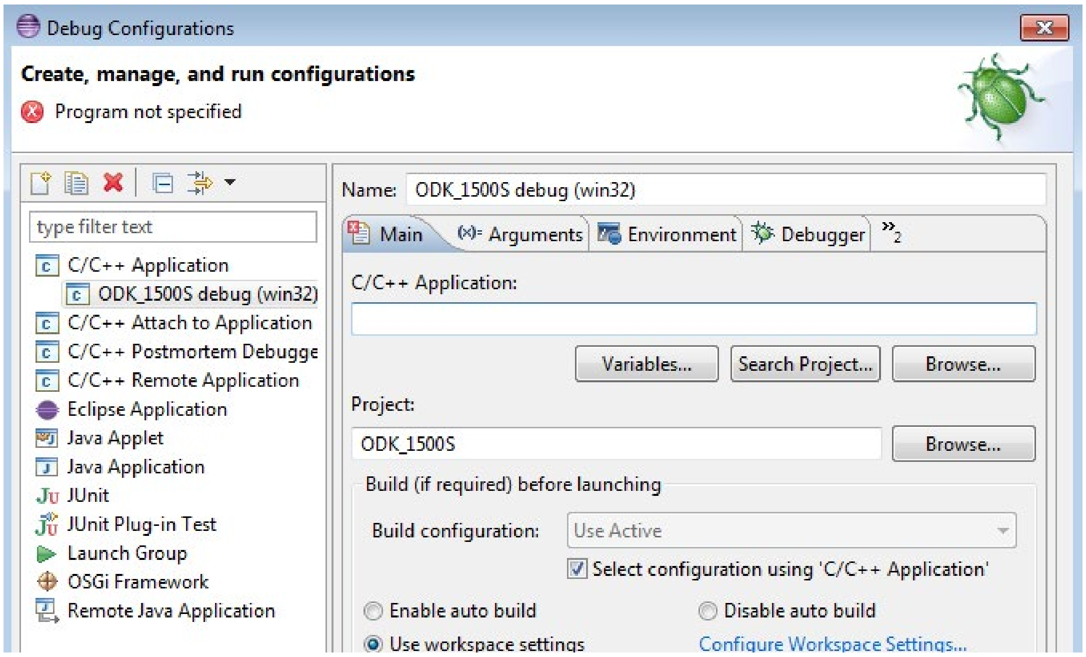
6. To create a new application, select the entry "C/C++ Application" and select the "New" command in the context menu.



Debug Configurations

Create, manage, and run configurations

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



Debug Configurations

Create, manage, and run configurations

Program not specified

Name: ODK_1500S debug (win32)

C/C++ Application:

Project: ODK_1500S

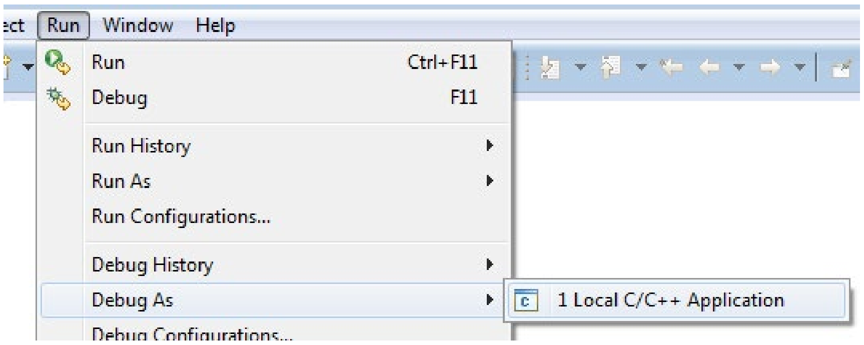
Build (if required) before launching

Build configuration: Use Active

[✓] Select configuration using 'C/C++ Application'

Enable auto build / Disable auto build / [✓ Use workspace settings]

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 complied 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
```

5.2 Transferring an ODK application to the target system

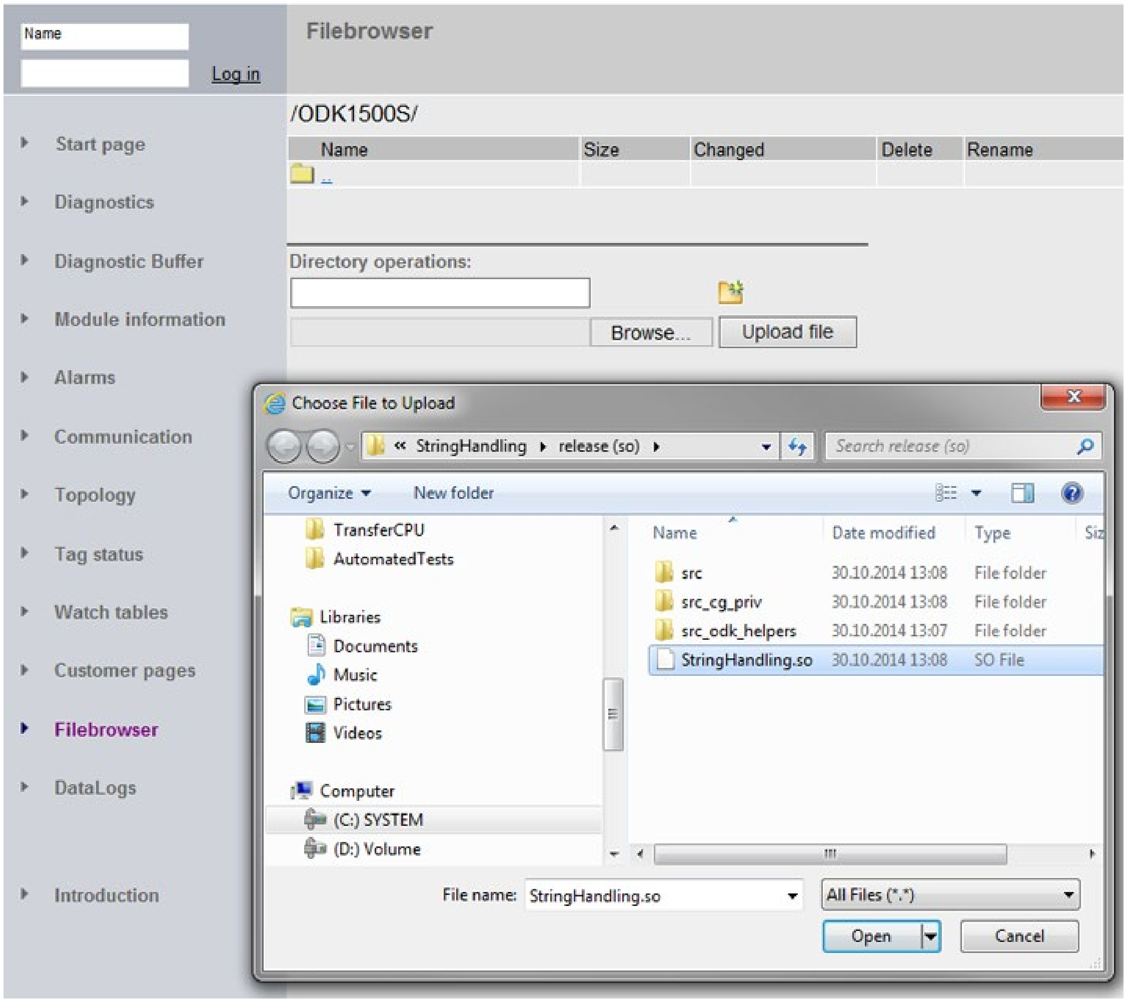
Procedure

Manually transfer the SO file to the target system. Use the file explorer of the web server of the CPU for transferring the ODK application.

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 ODK applications: \ODK1500S\



Name	Date modified	Type
src	30.10.2014 13:08	File folder
src_cg_priv	30.10.2014 13:08	File folder
src_odk_helpers	30.10.2014 13:07	File folder
StringHandling.so	30.10.2014 13:08	SO File

Figure 5-3 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.

5.3 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.

5.4 Executing a function

5.4.1 Loading functions

Introduction

Regardless of the context in which the ODK application 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
- As soon as the "*<STEP7Prefix>*_Load" instruction returns after the first call, the ODK application is loaded.

Note

Loading the same ODK applications with a modified <Project>.odk file

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

"*<STEP7Prefix>*_Load" instruction

An ODK application 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 ODK application.
Output	DONE	BOOL	Indicates that the instruction has finished loading the ODK application.
Output	BUSY	BOOL	Indicates that the instruction is still loading the ODK application.
Output	ERROR	BOOL	Indicates that an error occurred during the loading of the ODK application. 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 ODK application.

Input parameters

A 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: ODK application has already been loaded.
1	0	0	0x0000 =0	Loading was performed successfully.
0	0	1	0x80A4 =-32604	ODK application could not be loaded.
			0x80C3 =-32573	ODK application could not be loaded. The CPU currently does not have enough resources. Unload the ODK application before you load a new ODK application or restart the CPU.
			0x8090 =-32624	ODK application could not be loaded. An exception occurred during execution of the "OnLoad()" function.
			0x8092 =-32622	ODK application could not be loaded because the library name is invalid.
			0x8093 =-32621	ODK application could not be loaded because the ODK application could not be found. Check the file name and path of the file.
			0x8095 =-32619	ODK application could not be loaded for the following reasons: • The SO file is not an ODK application. • The CPU does not support the utilized ODK version.
			0x8096 =-32618	The ODK application could not be loaded because the internal identification is already being used by another loaded ODK application.
			0x8097 =-32617	CPU 1500 V1.8 and earlier: ODK application has already been loaded.
			0x8098 =-32616	ODK application could not be loaded because the ODK application is currently being unloaded.
			0x8099 =-32615	Unable to load the ODK application 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 ODK application could not be loaded and returns an invalid value (the values 0x0000 and 0xF000 - 0xFFFF are permitted)
			0xF000 – 0xFFFF =-4096 – -1	CPU 1500 V2.0 and later: ODK application could not be loaded. An error occurred during execution of the "OnLoad()" function.

5.4.2 Calling functions

Introduction

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

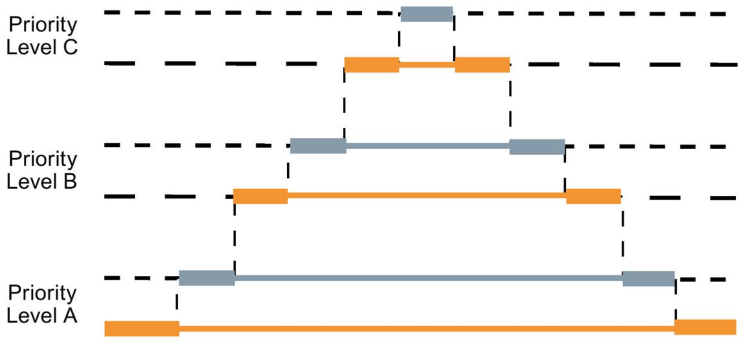


Figure 5-4 Calling functions

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

- Call another ODK function
- Call the same function ODK

  Therefore, pay attention when creating your ODK application on implementing the function calls re-entrant or avoid parallel execution.

Implement a maximum of three parallel calls. If you implement more than three parallel calls, the ODK function returns following status: 0x80C3

"*<STEP7Prefix>*SampleFunction" instruction

An ODK application 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 ODK application.
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	ODK application could not be executed for the following reasons: • A stack overflow was detected after execution of the function. To prevent follow-on errors, unload the ODK application. The developer of the ODK application is responsible for preventing the stack overflow. • 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 ODK application again.
0x80C3 =-32573	ODK application could not be run. The CPU currently does not have enough memory. Pay attention to the maximum number of parallel calls (SyncCallParallelCount).
0x8090 =-32624	ODK application could not be run. An exception occurred during execution. Each unhandled exception reduces the available heap size. An unhandled exception can damage the ODK application in such a way that it can no longer be used for additional calls. The ODK application must be unloaded. The developer of the ODK application is responsible for handling the exception and returning an application-specific error value.
0x8091 =-32623	ODK application could not be run. A "STOP" occurred during the function call.
0x8096 =-32618	ODK application could not be executed because the ODK application was not loaded or unloading is not yet finished.
0x8098 =-32616	ODK application could not be executed because the ODK application is different than the ODK instructions (FBs) in STEP 7: • older • newer • different parameters

STATUS	Meaning
0x8099 =-32615	ODK application could not be executed because the maximum amount of input data (32 KB) was exceeded (declarations with "In" and "InOut")
0x809A =-32614	ODK application could not be executed because the maximum amount of output data (32 KB) 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)
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)

5.4.3 Unloading functions

Introduction

The ODK application is unloaded be calling the "*<STEP7Prefix>*_Unload" instruction. Call is made from the STEP 7 user program.

In addition to this call, the ODK application is also automatically unloaded for the following reasons.

- The CPU is switched off
- The CPU is reset

Regardless of the context in which the ODK application 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 these ODK application. 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 ODK application is unloaded.

"*<STEP7Prefix>*_Unload" instruction

An ODK application 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 ODK application.
Output	DONE	BOOL	Indicates that the instruction has finished unloading the ODK application.
Output	BUSY	BOOL	Indicates that the instruction is still unloading the ODK application.
Output	ERROR	BOOL	Indicates that an error occurred during the unloading of the ODK application. 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 ODK application.

Input parameters

A 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	ODK application could not be unloaded. A communication error between the CPU and ODK occurred during the execution of the "OnUnload()" function.
			0x80C3 =-32573	ODK application could not be unloaded. The CPU currently does not have enough memory.
			0x8090 =-32624	ODK application could not be unloaded. An exception occurred during execution of the "OnUnload()" function.
			0x8096 =-32618	ODK application could not be unloaded because the ODK application was not loaded or unloading is not yet finished.
			0x809B =-32613	CPU 1500 V2.0 and later: The ODK application could be unloaded and returns an invalid value (the values 0x0000 and 0xF000 - 0xFFFF are permitted)
			0xF000 – 0xFFFF =-4096 – -1	CPU 1500 V2.0 and later: ODK application could be unloaded. An error occurred in the CCX object during execution of the "OnLoad()" function.

5.4.4 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 ODK application
- 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=on 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 default CCP file "<Project>.cpp".

GetTrace	
TraceCount	STATUS

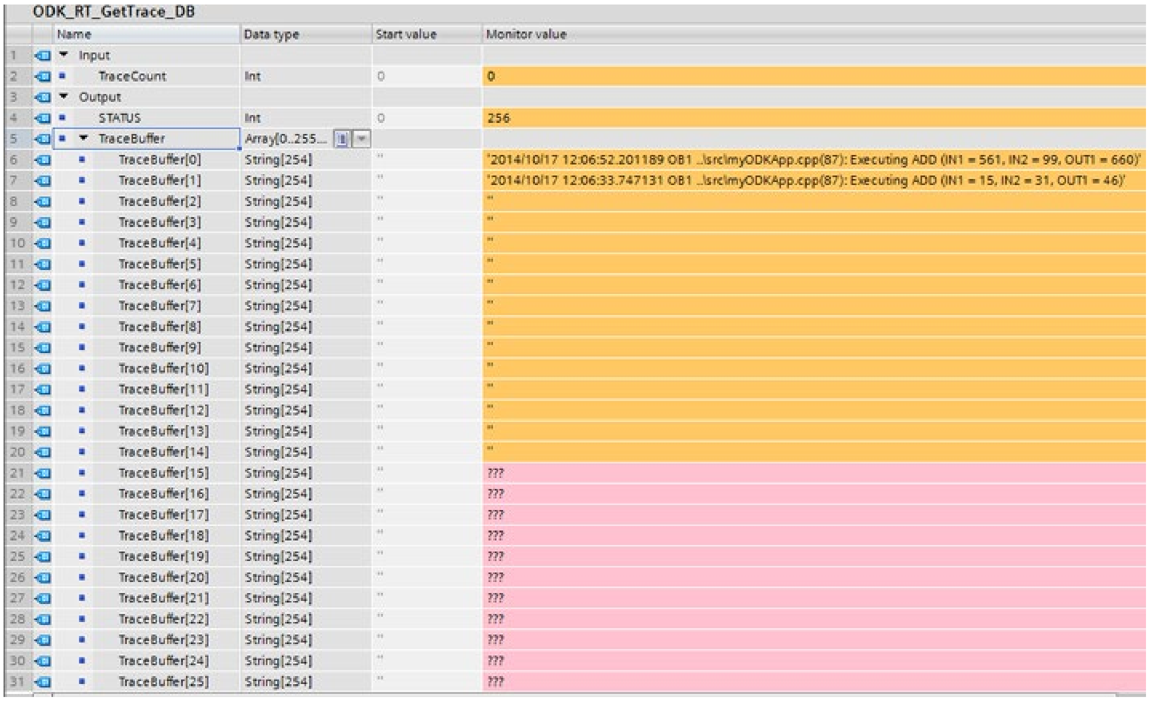
The following table shows the parameters of the "GetTrace" function block:

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:



ODK_RT_GetTrace_DB

	Name	Data type	Start value	Monitor value
1	▾ Input			
2	TraceCount	Int	0	0
3	▾ Output			
4	STATUS	Int	0	256
5	▾ TraceBuffer	Array[0..255...		
6	TraceBuffer[0]	String[254]	''	'2014/10/17 12:06:52.201189 OB1 ..\src\myODKApp.cpp(87): Executing ADD (IN1 = 561, IN2 = 99, OUT1 = 660)'
7	TraceBuffer[1]	String[254]	''	'2014/10/17 12:06:33.747131 OB1 ..\src\myODKApp.cpp(87): Executing ADD (IN1 = 15, IN2 = 31, OUT1 = 46)'
8	TraceBuffer[2]	String[254]	''	''
9	TraceBuffer[3]	String[254]	''	''
10	TraceBuffer[4]	String[254]	''	''
11	TraceBuffer[5]	String[254]	''	''
12	TraceBuffer[6]	String[254]	''	''
13	TraceBuffer[7]	String[254]	''	''
14	TraceBuffer[8]	String[254]	''	''
15	TraceBuffer[9]	String[254]	''	''
16	TraceBuffer[10]	String[254]	''	''
17	TraceBuffer[11]	String[254]	''	''
18	TraceBuffer[12]	String[254]	''	''
19	TraceBuffer[13]	String[254]	''	''
20	TraceBuffer[14]	String[254]	''	''
21	TraceBuffer[15]	String[254]	''	???
22	TraceBuffer[16]	String[254]	''	???
23	TraceBuffer[17]	String[254]	''	???
24	TraceBuffer[18]	String[254]	''	???
25	TraceBuffer[19]	String[254]	''	???
26	TraceBuffer[20]	String[254]	''	???
27	TraceBuffer[21]	String[254]	''	???
28	TraceBuffer[22]	String[254]	''	???
29	TraceBuffer[23]	String[254]	''	???
30	TraceBuffer[24]	String[254]	''	???
31	TraceBuffer[25]	String[254]	''	???

5.5 Post Mortem analysis

5.5.1 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

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 ODK application 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

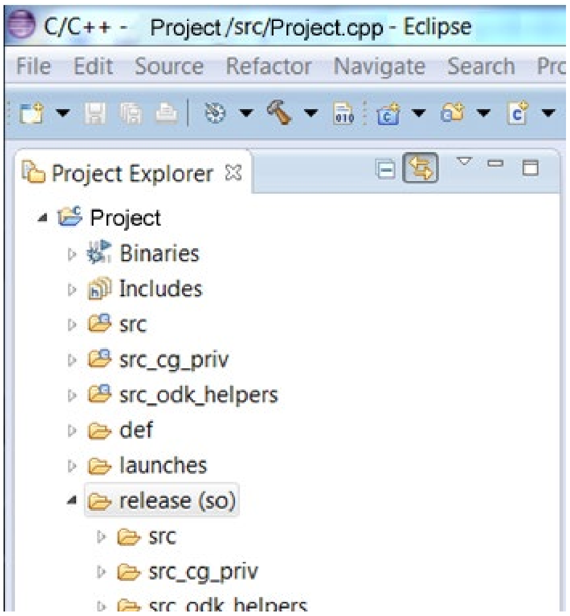
NOTICE
Insufficient load memory
When there is not enough load memory, the post mortem files are not saved properly.
Make sure that you have enough load memory for your applications.

5.5.2 Execute post mortem analysis

Procedure

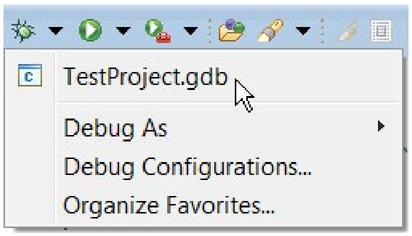
To run a post mortem analysis, follow these steps:

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.

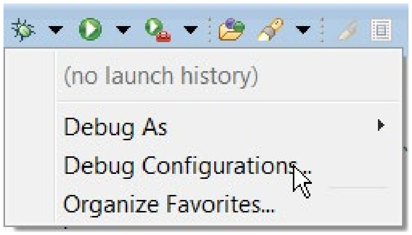


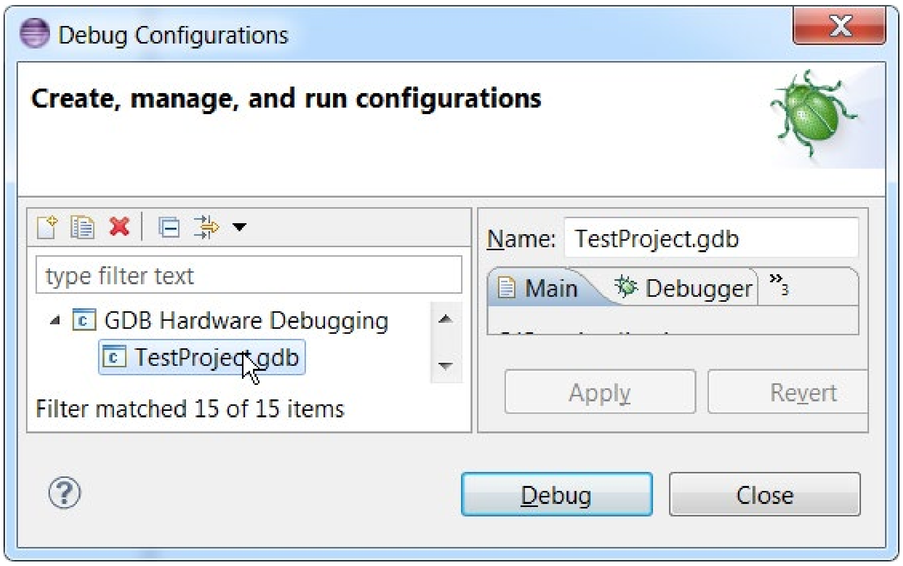
3. Select the required project.

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



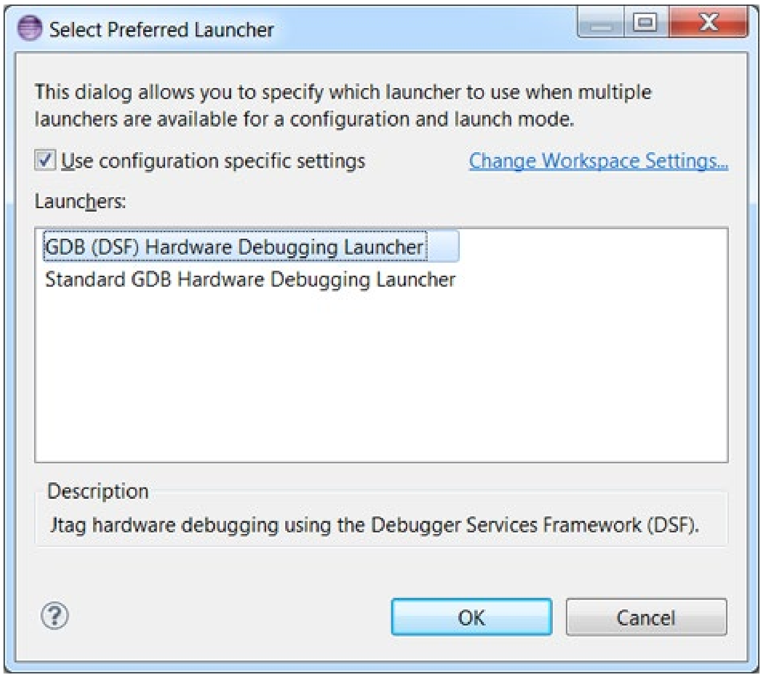
   - Using "Debug Configurations"





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".



Select Preferred Launcher

This dialog allows you to specify which launcher to use when multiple launchers are available for a configuration and launch mode.

☑ Use configuration specific settings Change Workspace Settings...

Launchers:

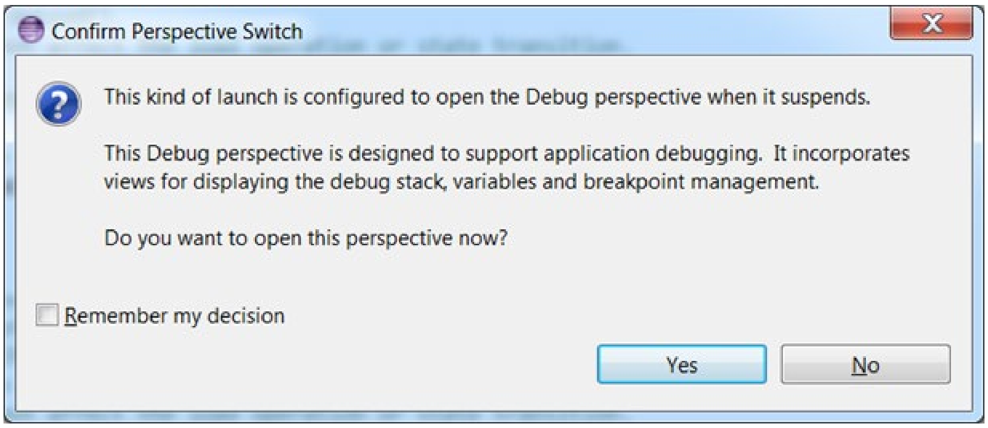
GDB (DSF) Hardware Debugging Launcher
Standard GDB Hardware Debugging Launcher

Description

Jtag hardware debugging using the Debugger Services Framework (DSF).

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.



Confirm Perspective Switch

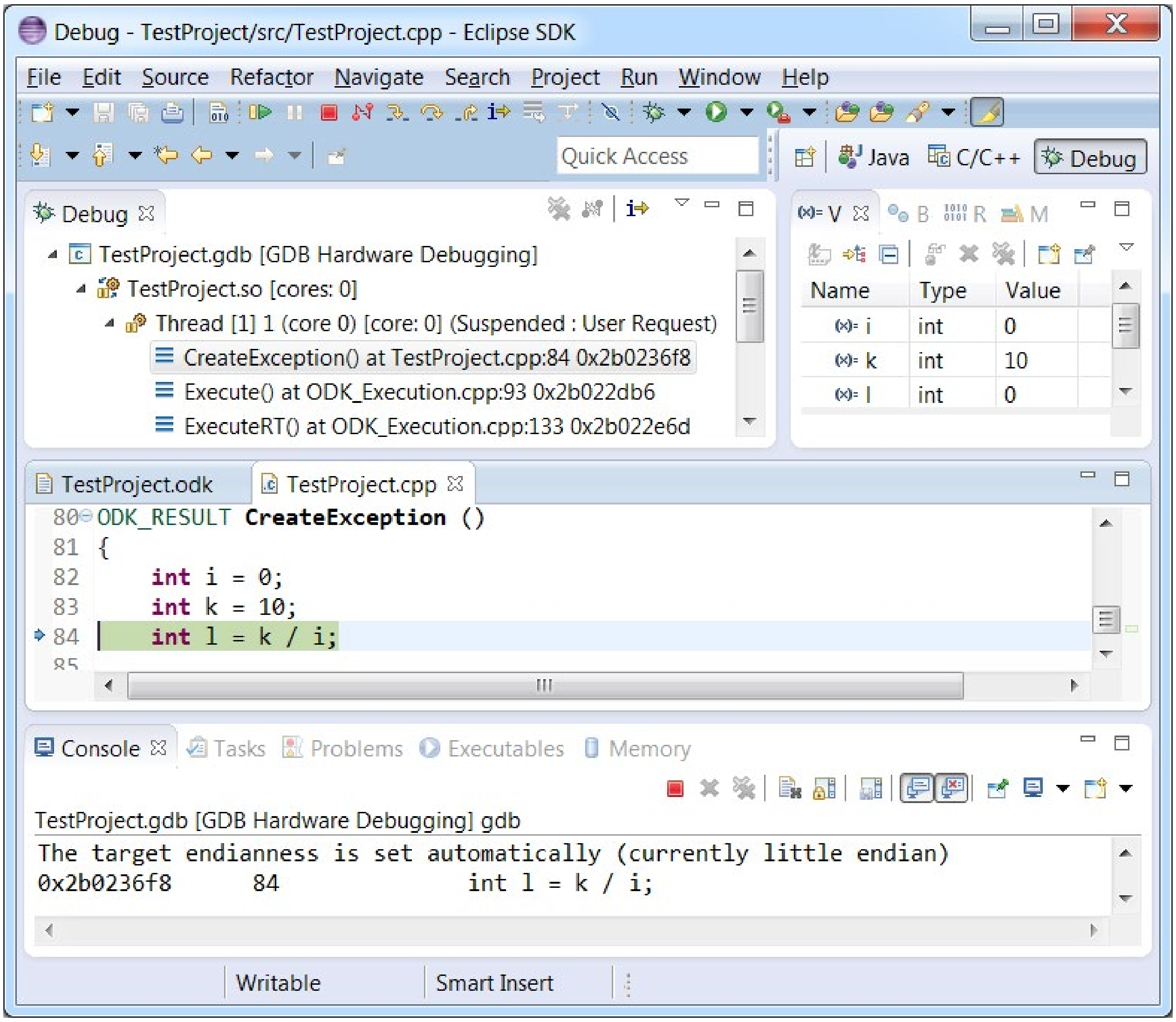
This kind of launch is configured to open the Debug perspective when it suspends.

This Debug perspective is designed to support application debugging. It incorporates views for displaying the debug stack, variables and breakpoint management.

Do you want to open this perspective now?

☐ Remember my decision

6. Run the debug process.


Debug - TestProject/src/TestProject.cpp - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java C/C++ Debug
Debug
V B R M
TestProject.odk
TestProject.cpp
Console Tasks Problems Executables Memory
Writable
Smart Insert

```
TestProject.gdb [GDB Hardware Debugging]
  TestProject.so [cores: 0]
    Thread [1] 1 (core 0) [core: 0] (Suspended : User Request)
      CreateException() at TestProject.cpp:84 0x2b0236f8
      Execute() at ODK_Execution.cpp:93 0x2b022db6
      ExecuteRT() at ODK_Execution.cpp:133 0x2b022e6d
```

Name	Type	Value
i	int	0
k	int	10
l	int	0

```
ODK_RESULT CreateException ()
{
    int i = 0;
    int k = 10;
    int l = k / i;
```

```
TestProject.gdb [GDB Hardware Debugging] gdb
The target endianness is set automatically (currently little endian)
0x2b0236f8      84              int l = k / i;
```

Using example projects 6

To facilitate you introduction to the topic of ODK, 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 are available for download on the Internet (https://support.industry.siemens.com/cs/document/106192387/simatic-odk-1500s-examples?dti=0&lc=en-WW).

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 DLL or SO file to the target system.

Appendix A

A.1 General conditions of ODK applications

A.1.1 Number of loadable ODK applications

You can load a total of up to 32 ODK applications for the Windows and real-time environments.

Configuration limits for ODK applications:

- ODK applications for the Windows environment:
  - Up to 32 parallel function calls (total)
  - Up to 1 MB input or output data (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.

- ODK applications for the real-time environment:
  - Parallel function calls in an ODK application are defined by the "SyncCallParallelCount" parameter.
  - Up to 26 parallel function calls (total)
  - Up to 1 MB input data and output data per function call

The available memory for loading of ODK applications is limited in the context of the realtime environment. The table below provides an overview of the available memory of the different CPUs for loading ODK applications:

CPU	Available memory for loading ODK applications	Maximum size of the SO file
CPU 1505SP (F)	10 MB	3.8 MB
CPU 1507S (F)	20 MB	5.8 MB
CPU 1518-4 PN/DP ODK (F)	20 MB	5.8 MB

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

- SO file name may not exceed 56 characters.

A.1.2 Compatibility

If you are using ODK version V2.0, note the following:

- An ODK project created with ODK version < V2.0 is not compatible. You must re-create your ODK project in version V2.0.
- An ODK application created with ODK version < V2.0 is compatible with newer CPUs.

A.2 Syntax interface file <Project>.odk

A.2.1 Data types

The data type defines the type of a tag. The following table defines the possible data types and their 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	unsigned long long	64-bit during in nanoseconds
ODK_TIME	TIME	unsigned long	32-bit during in milliseconds
ODK_LDT	LDT	unsigned long long	64-bit date and time of the day in nanoseconds
ODK_LTOD	LTOD	unsigned long long	64-bit time of the day in nano-seconds since midnight
ODK_TOD	TOD	unsigned long	32-bit time of the day in milli-seconds since midnight
ODK_WCHAR	WCHAR	wchar_t	**Only for Windows:** 16-bit char-acter
ODK_CHAR	CHAR	char	8-bit character

- **Complex data types:**

ODK data type	**SIMATIC data type**	**C++ data type**	**Description**
ODK_DTL	DTL	struct ODK_DTL	Structure for date and time
ODK_S7STRING	STRING	unsigned char	Character string (8-bit character) with max. and act. length (2xUSINT)
ODK_S7WSTRING	WSTRING	unsigned short	**Only for Windows:** Character string (16-bit character) with max. und act. length (2xUINT)
ODK_CLASSIC_DB	VARIANT	struct ODK_CLASSIC_DB	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 an array except ODK_CLASSIC_DB.

- **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.

  – You must not use keywords for the generated language (e.g. "int" as tag name).

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

A.2.2 Parameters

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

- Developing ODK application for the Windows environment
- Developing ODK application 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 <Projekt>.odk file supports the following parameters:

Parameter	Value	Description
Context	user	Defines that the ODK application is loaded in a context of a Windows user.
	system	Defines that the ODK application is loaded in a 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, -, _}

Parameters for the realtime environment

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

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

The <Projekt>.odk file supports the following parameters:

Parameter	Value	Description
Context	realtime	Defines that the ODK application is loaded in the context of the realtime environment.
Trace	on	Defines the trace function in the ODK application. In this case, the ODK application requires 32K if memory as an additional trace buffer. A "Get-Trace" 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 89)]k	Defines a memory in KB that is used as heap for realtime applications.
HeapMaxBlockSize	[8…<HeapSize>]	Defines the memory size in bytes that can be allocated at one time.
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, -, _}

A.3 Error messages of the code generator

The code generator generates the following error messages:

File errors:

Error number	Error message	Possible solution
100	‘<Project>.odk’ is missing	Rename the file to <Project>x.odk.
101	Context is missing in resorce file	Error in the resource file (.rc).
102	resource file ‘...’ is missing	The resource file (.rc) is missing.
103	‘...’ write protected	The indicated file is write protected.

Parameter errors:

Error number	Error message	Possible solution
200	parameter ‘...’ is not allowed for current con-text	The indicated parameter is not allowed here.
201	missing ‘...’ definition	The indicated parameter (Page 52) is not defined.
202	more than one defition for ‘...’	There is more than one definition for the indicated parameter (Page 52).
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 "relatime" 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 of an ODK application (Page 55)
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.
511	'...' no valid name	Observe the syntax rules in section Defining functions of an ODK application (Page 55).
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 type	As soon as the ODK_CLASSIC_DB is used, no other tag with the same InOut identifier may be defined.
522	no array allowed for ODK_CLASSIC_DB type	No Array may be defined for the ODK_CLASSIC_DB data type.
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.

A.4 Helper functions

String-helper functions for ODB application for Windows and real-time 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 ODB application 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 class "CODK_CpuReadData" allows read access to Classic DBs:

Value	Description
CODK_CpuReadData	Class constructor: Initializes the input data area and the data size.
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.
ReadS7S5TIME	Reads a "16-bit" (2 bytes) time value from the data area.
ReadS7DATE	Reads a date value (2 bytes) from the data area.
ReadS7TIME_OF_DAY	Reads the time of day (4 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.
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.

Value	Description
ReadS7REAL	Reads a "real number" (4 bytes) from the data area.
ReadS7LREAL	Reads a "long real number" (8 bytes) from the data area.
ReadS7LINT	Reads a "long integer" (8 bytes) from the data area.
ReadS7ULINT	Reads a "unsigned long integer" (8 bytes) from the data area.
ReadS7TIME	Reads a time value (4 bytes) from the data area.
ReadS7CHAR	Reads a "char" (1 byte) from the data area.
ReadS7BOOL	Reads a "bool" (1 byte) from the data area.
ReadS7STRING_LEN	Reads the information of string length from an S7 string in the data area.
ReadS7STRING	Reads an S7 string from the data area and returns it as C++ character string.
ReadS7DATE_AND_TIME	Reads the general data and time area.

Class "CODK_CpuReadWriteData" (Windows and real-time environment)

The class "CODK_CpuReadWriteData" also allows all access of "CODK_CpuReadData" to Classic-DBs, as well as the following write access:

Value	Description
CODK_CpuReadWriteData	Class constructor: Initializes the output data area and the data size.
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.
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.
WriteS7S5TIME	Writes a 16-bit (2 bytes) time value to the data area.
WriteS7TIME	Writes a time value (4 bytes) to the data area.
WriteS7DATE	Writes a date value (2 bytes) to the data area.
WriteS7TIME_OF_DAY	Writes a time of day (4 bytes) to the data area.
WriteS7CHAR	Writes a "char" (1 byte) 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.
WriteS7LINT	Writes a "long integer" (8 bytes) to the data area.
WriteS7ULINT	Writes a "unsigned long integer" (2 bytes) to the data area.
WriteS7BOOL	Writes a "bool" (1 byte) to the data area.
WriteS7STRING	Writes a S7 string to the data area.
WriteS7DATE_AND_TIME	Write data and time data to the date and time area.

A.5 "Load" instruction

The "*<STEP7Prefix>*_Load" instruction has different parameters that depending on the development environment:

- Developing ODK application for the Windows environment (Page 37)
- Developing ODK application for the realtime environment (Page 75)

A.6 "Unload" instruction

The "*<STEP7Prefix>*_Unload" instruction has different parameters that depending on the development environment:

- Developing ODK application for the Windows environment (Page 43)
- Developing ODK application for the realtime environment (Page 79)

A.7 "GetTrace" instruction

The function block (Page 81) "GetTrace" is included in the standard CPP file "<Projckt>.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