

# **SIEMENS**

**SIMATIC**

**WinAC Open Development Kit (ODK)**

**Version: 4.0**

# Copyright and Safety Notification

This manual contains notices that you should observe to ensure your own personal safety, as well as to protect the product and connected equipment. These notices are highlighted in the manual by a warning triangle and are marked as follows according to the level of danger:



## Danger

Indicates an imminently hazardous situation that, if not avoided, will result in death or serious injury.



## Warning

Indicates a potentially hazardous situation that, if not avoided, could result in death or severe injury.



## Caution

Used with the safety alert symbol indicates a potentially hazardous situation that, if not avoided, may result in minor or moderate injury.

## Caution

Used without the safety alert symbol indicates a potentially hazardous situation that, if not avoided, may result in property damage.

## Notice

NOTICE used without the safety alert symbol indicates a potential situation that, if not avoided, may result in an undesirable result or state.

## Qualified Personnel

The device/system may only be set up and operated in conjunction with this manual. Only qualified personnel should be allowed to install and work on this equipment. Qualified persons are defined as persons who are authorized to commission, to ground, and to tag circuits, equipment, and systems in accordance with established safety practices and standards.

## Correct Usage

Note the following:



## Caution

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

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

## Trademarks

Siemens<sup>®</sup> and SIMATIC<sup>®</sup> are registered trademarks of SIEMENS AG.

STEP 7<sup>™</sup> and S7<sup>™</sup> are trademarks of SIEMENS AG.

Microsoft<sup>®</sup>, Windows<sup>®</sup>, Windows 2000<sup>®</sup>, and Windows XP<sup>®</sup>, are registered trademarks of Microsoft Corporation.

RTX<sup>™</sup> is a registered trademark of Venturcom, Inc.

## Copyright Siemens Energy & Automation, Inc., 2003

### All rights reserved

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

Siemens Energy & Automation, PCbA  
One Internet Plaza  
Johnson City, TN 37602-4991, USA

## Disclaimer of Liability

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

© Siemens AG 2003

Technical data subject to change.

# Preface

The Windows Automation Center Open Development Kit (WinAC ODK) provides tools for creating custom C/C++ software that can be called directly from the STEP 7 control program executing in a WinAC controller.

WinAC ODK consists of the following elements:

- Application wizard for generating C/C++ project framework
- STEP 7 library containing System Function Blocks (SFBs) for accessing your custom software
- WinAC ODK User Manual (electronic)
- Sample programs

## Audience

This manual is intended for engineers, programmers, and maintenance personnel who have a general knowledge of programmable logic controllers and who are proficient in C or C++. Users also need knowledge of STEP 7 programming and of the controller that they plan to use with WinAC ODK.

## Scope of the Manual

This manual describes the operation and features of version 4.0 of the WinAC Open Development Kit (ODK).

## How to Use This Documentation

This documentation provides the following information:

- Overview and installation of WinAC ODK
- Getting started with a sample application program
- Programming an extension object (RTDLL or DLL) using the WinAC ODK Application Wizard and C/C++ Elements
- Programming the STEP 7 program using the ODK System Function Blocks
- Debugging your custom software
- Web-based descriptions of the C/C++ classes and functions that you use to produce custom software (object webs)

**Note:** The object webs are available only when viewing the documentation in online help format. The object webs are not visible or accessible from the PDF version of the documentation. For navigation ease, cross-references between topics, and access to the object webs, use the online help. For a printable format, use the PDF.

## Other Manuals

For additional information, refer to the following manual or online help system for the WinAC controller that you are using:

Title	Content
Windows Automation Center (WinAC Basis V4.0) User Manual	This manual provides information about WinAC and the WinLC controller for Windows 2000 or Windows XP
Windows Automation Center RTX (WinAC RTX 4.0) User Manual	This manual provides information about WinAC RTX and the WinLC RTX controller for Windows 2000 or Windows XP with the Venturcom Real-time extensions.

## Additional Assistance

If you have any questions not answered in this or one of the other manuals, if you need information on ordering additional documentation or equipment, or if you need information on training, please contact your Siemens distributor or sales office.

## Contacting Customer Support

You can find additional information and updates to this user manual at the Siemens Energy & Automation web site:

- [www.sea.siemens.com/software](http://www.sea.siemens.com/software)

This web site includes useful information about ODK and other PC-based control products. The Quick Downloads section provides access to manuals and software downloads. The select list for Tips, Tricks, and Examples includes sample programs for linking a Delta Tau motion control board with a WinAC controller, and other sample programs using ODK.

Customer Support			
<b>North America</b>			
Telephone	(423) 262-2522	Fax	(423) 262-2289
E-mail	simatic.hotline@sea.siemens.com		
Internet	<a href="http://www.sea.siemens.com/software">http://www.sea.siemens.com/software</a> <a href="http://www1.ad.siemens.de/meta/support/html_76/support.htm">http://www1.ad.siemens.de/meta/support/html_76/support.htm</a> <a href="http://www4.ad.siemens.de/csinfo/livelink.exe?func=cslib.csinfo2&amp;siteid=cs&amp;lang=en">http://www4.ad.siemens.de/csinfo/livelink.exe?func=cslib.csinfo2&amp;siteid=cs&amp;lang=en</a>		
<b>Europe</b>			
Telephone	++49 (0) 180 5050 222	Fax	++49 (0) 180 5050 223
E-Mail	adsupport@siemens.com		
Internet	<a href="http://www.ad.siemens.de/meta/support/html_00/support.htm">http://www.ad.siemens.de/meta/support/html_00/support.htm</a>		

# Contents

- Product Overview ..... 1**
  - WinAC ODK Overview..... 1
    - WinAC ODK System Function Blocks..... 1
    - ODK Expands the Capabilities of the Control Program ..... 2
    - Tools Provided by WinAC ODK ..... 2
    - WinAC ODK Provides a Mechanism for Defining Custom Logic ..... 3
    - Extension Object for WinLC RTX..... 3
    - Extension Object for WinLC Basis ..... 4
    - Restrictions..... 4
  - What's New? ..... 6
  - System Requirements ..... 7
    - Hardware Requirements ..... 7
    - Software Requirements..... 7
  - Installing WinAC ODK ..... 8
    - Installing ODK When a Version of ODK Is Already Installed ..... 8
    - Uninstalling (Removing) ..... 8
- Getting Started..... 9**
  - Introduction to the HistoDLL Sample Program..... 9
  - How to Use the HistoDLL Sample Program ..... 10
    - Building the HistoDLL Visual C++ Project..... 10
    - Retrieving and Configuring the HistoDLL STEP 7 Program ..... 11
    - Downloading and Running the HistoDLL STEP 7 Program..... 11
  - Overview of the HistoDLL STEP 7 Program ..... 12
    - STEP 7 Program Structure ..... 12
    - STEP 7 Program Execution Logic..... 13
    - Where to Find STEP 7 Program Listings ..... 14
  - Overview of the HistoDLL Visual C++ Program ..... 15
    - ODKCreate..... 15
    - Activate..... 15
    - DeActivate ..... 15
    - ODKRelease ..... 15
    - Execute ..... 15
    - ExecuteInit..... 15
    - ExecuteUpdate..... 16
    - ScheduleOB52 ..... 16
    - HistoDLL Object Web..... 16
  - Additional Sample Programs ..... 16

<b>Developing an ODK Application.....</b>	<b>17</b>
Creating the C/C++ Project.....	17
Creating the STEP 7 Program.....	17
Using the Application Wizard.....	18
Configuring Project Information.....	18
Entering ODK Project Subcommands.....	19
Enabling Asynchronous Processing.....	21
Enabling Asynchronous Monitoring.....	23
Generating the WinAC ODK C/C++ Project.....	25
Programming the C/C++ Software.....	27
Structure of the ODK C/C++ Project.....	27
Programming Asynchronous Events.....	30
Programming Monitor Threads.....	32
Building the Extension Object.....	33
Registering an RTDLL.....	34
Implementing the STEP 7 Project.....	34
Loading the WinAC ODK Library Into STEP 7.....	34
Programming the STEP 7 Program.....	35
Debugging and Changing the Extension Object.....	36
Debugging the Extension Object.....	36
Changing the Extension Object.....	39
<b>ODK Support Software.....</b>	<b>41</b>
Data Access Helper Classes.....	41
WinAC ODK Library for STEP 7.....	42
SFB65001 (CREA_COM).....	42
Error Codes for SFB65001.....	43
SFB65002 (EXEC_COM).....	44
Error Codes for SFB65002 (EXEC_COM).....	45
Auxiliary STEP 7 Interface Functions.....	46
ODK_ReadState.....	46
ODK_ScheduleOB.....	47
DataType2.....	48
DataType1.....	48
Error Codes from ODK_ScheduleOB.....	48
Example of Calling an Auxiliary Function.....	49

# Product Overview

## WinAC ODK Overview

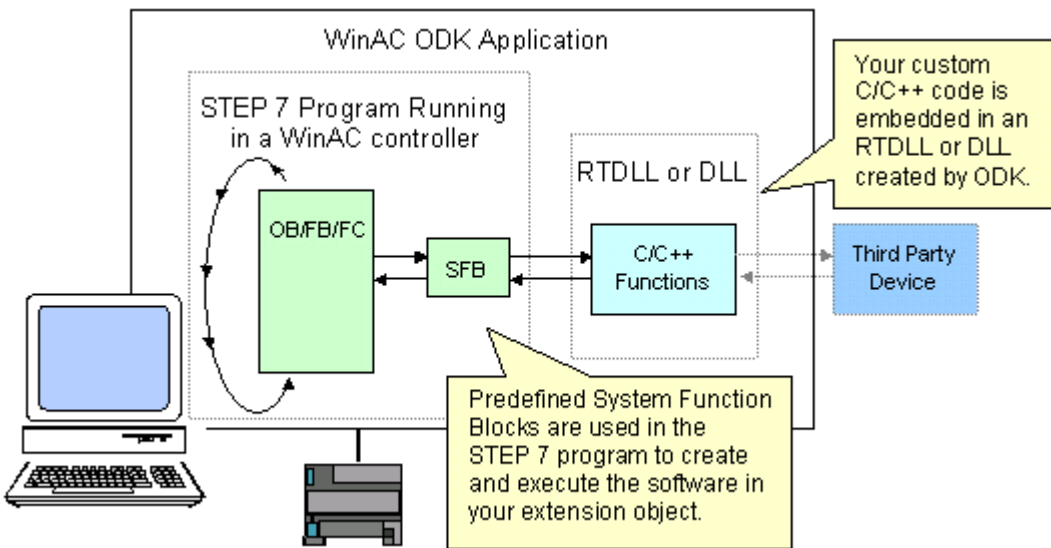
The Windows Automation Center Open Development Kit (WinAC ODK) is an open interface to WinAC controllers. It provides a set of tools that enables you to implement custom software that is executed as part of the STEP 7 control program. WinAC ODK V4.0 implements this custom interface as an extension object. To create an extension object for WinLC RTX V4.0, you create an RTDLL. To create an extension object for WinLC Basis V4.0, you create a DLL. You can also create a Proxy DLL to use with WinLC RTX V4.0 that does not run in the real-time environment.

WinAC ODK provides an application wizard for you to develop this custom software and to create the extension object for your controller. Supported programming languages are C, C++, and C#.

## WinAC ODK System Function Blocks

Two STEP 7 System Function Blocks (SFBs) provide the interface between your custom C/C++ software and the control program logic that the WinAC controller executes. These two SFBs, SFB65001 and SFB65002, enable the STEP 7 program to create the extension object that contains your custom code and to execute that code.

Your custom software is invoked by an SFB, and executed as part of the control program scan cycle. WinAC ODK publishes the interface that it expects, and you implement your custom logic according to this interface.



If your custom software requires too much time to be executed within the control program scan cycle, you can configure it to run on a separate thread of execution. Use a separate thread if the execution time of your custom software could cause a scan to exceed the configured watchdog time. WinAC ODK includes an application wizard that helps you set up one or more asynchronous threads for this purpose. Your custom software can also handle events by scheduling an asynchronous OB to be executed by the controller.



**Caution**

The STEP 7 program executes the SFB that calls the ODK custom software as a single non-interruptible instruction. During the execution of the ODK custom software, the watchdog timer, OBs, and process alarms are not available. They are handled after the SFB call completes. ODK custom software that significantly extends the cycle time can delay the processing of critical activities in the controller, possibly resulting in personal injury.

To avoid this delay, program asynchronous events to handle custom logic of a lengthy duration.

## ODK Expands the Capabilities of the Control Program

Because you can use ODK to integrate custom software into your control program, you can expand the capabilities of a STEP 7 control program. The following situations are examples where ODK can provide benefits:

- The WinAC controller can incorporate special control logic that was written in C or C++.
- The control solution uses a complex (or proprietary) calculation (such as PID or gas flow), which has higher performance requirements or is more easily written and maintained in C or C++.
- The application requires connectivity with another application or hardware, such as motion control or vision systems.
- The application needs to access features of the computer that are not accessible by standard S7 control languages.

## Tools Provided by WinAC ODK

The tools provided by WinAC ODK embed your custom software into either an RTDLL for use with the Real-Time Sub System (RTSS) or into a DLL for use with the Windows operating system. The following tools are included with the WinAC ODK installation:

- Application wizard for generating a program shell with the correct interfaces and functions for interacting with a WinAC controller
- STEP 7 library containing two SFBs that you can insert into the STEP 7 program:
  - SFB65001 (CREA\_COM) creates an instance of the extension object (RTDLL or DLL). You can have more than one extension object, and each one can contain multiple actions. SFB65001 returns the program handle for the object that it creates. This program handle can then be used by SFB65002.
  - SFB65002 (EXEC\_COM) executes a specific function in the RTDLL or DLL created by SFB65001.
- Documentation in the form of an electronic user manual and linked object webs
- Sample programs including C/C++ projects and their corresponding STEP 7 projects

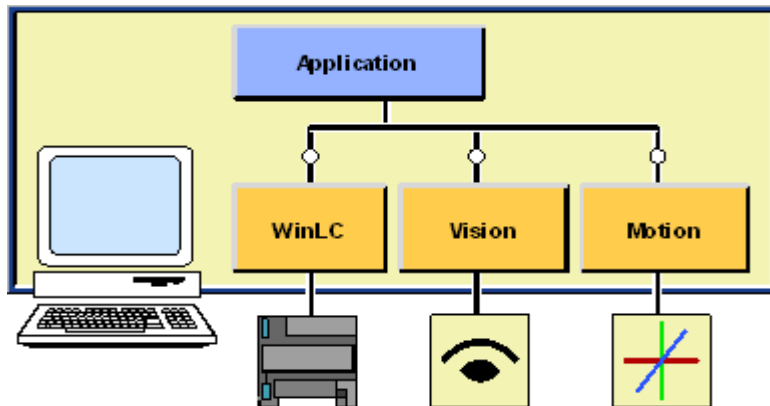
With WinAC ODK, you can define and implement one or more standard FBs or FCs that become an easy-to-use block library that you can use throughout the rest of your STEP 7 program. The blocks in this user-defined library, in turn, access the WinAC ODK SFBs. The example programs that are installed with ODK illustrate this STEP 7 program design.



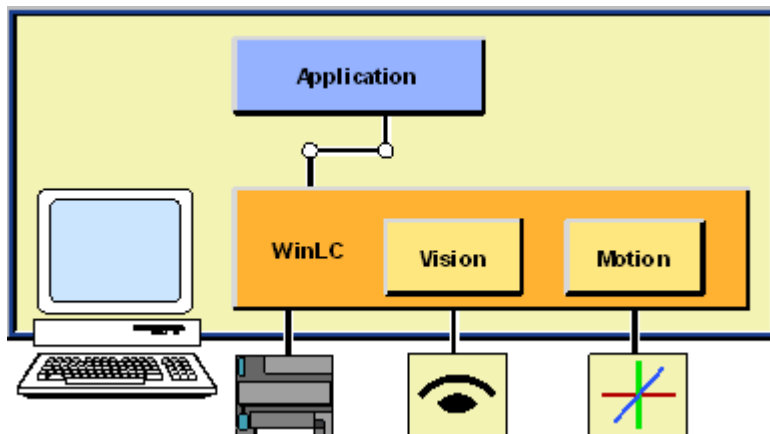
## WinAC ODK Provides a Mechanism for Defining Custom Logic

WinAC ODK provides the mechanism for you to define one or more custom system functions that can be integrated into the STEP 7 program logic. The Open Development Kit connects the SFBs in the control program to your custom software extension object.

For example, without using the Open Development Kit, a process that uses a motion control board and a vision board might have a custom application that interacts between the two boards. This application interacts with the controller by using STEP 7 asynchronous read/write functions to manipulate the I/O or by using the Data control of WinAC Computing.



By using WinAC ODK, this custom application can now interact directly with the control program. The control program can access the motion control board or vision board through ODK extension objects.



## Extension Object for WinLC RTX

For WinLC RTX, the WinAC ODK software allows you to build your extension object as either an RTDLL or a Proxy DLL. An RTDLL is in-process; it runs in the same process as WinLC RTX and is loaded in the same address space. It runs in the real-time subsystem (RTSS). A Proxy DLL is also in-process under Windows 2000/XP, but its data must be copied between the RTSS and Windows through a shared memory buffer. For this reason, an RTDLL allows for faster processing because it avoids the overhead of passing data through the shared memory buffer. In either case, there is no context switching between the RTDLL or Proxy DLL and WinLC RTX because they are in the same process. (When applications run in separate processes, the operating system must stop the current process and save its context before switching the execution focus to another process. Additionally, the function input/output data must be marshaled and moved between the processes.)

## Extension Object for WinLC Basis

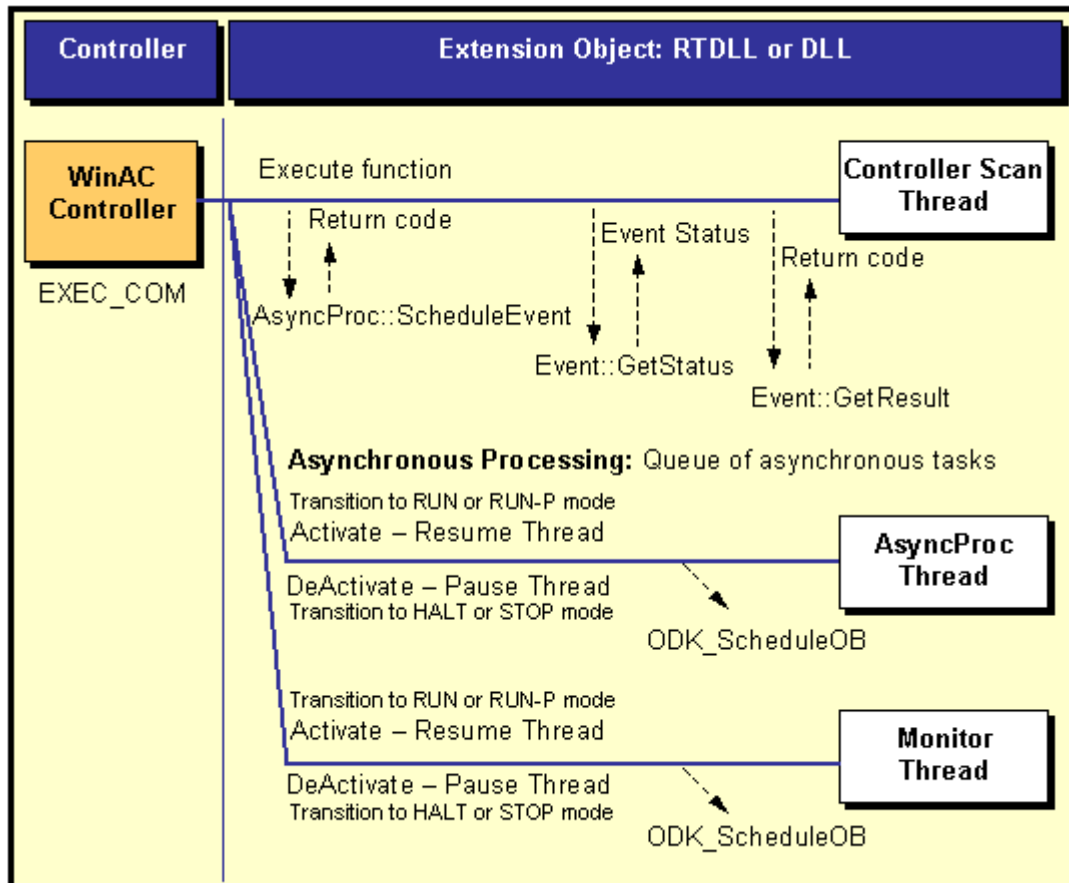
For WinLC Basis, the WinAC ODK software allows you to build your extension object as a DLL. The DLL is in-process; it runs in the same process as WinLC and is loaded in the same address space.

### Restrictions

Because a WinLC controller can be configured to start automatically when the computer restarts, you must be aware that there are some restrictions on what your custom software can do. For example, if your software activates a graphical user interface (GUI), that interface would not be visible after a reboot. The interface (GUI) needs to be a separate process started in a user context with communication to your extension object, possibly through shared memory.

WinAC ODK handles the synchronization between the calls to the interface. However, if your custom software contains other threads or handles other asynchronous events, then your application must be responsible for coordinating internally between these events and the functions in the interface.

Because the software in the extension object executes as part of the control program scan cycle, it must finish within the scan requirements. If the software in your extension object requires too much of the scan cycle, use the asynchronous processor (AsyncProc) to handle this processing outside of the scan cycle.



---

WinAC ODK provides the tools to perform the following tasks:

- Create your extension object at some user-defined time, such as startup (by calling SFB65001 in OB100)
- Release the extension object
- Call the functions of the extension object
- Notify (or activate) your extension object when it is created and before the controller makes the transition from STOP to STARTUP or from HALT to RUN or RUN-P mode
- Notify (or deactivate) your extension object when the controller goes to STOP or HALT mode
- Create an event that causes the STEP 7 program to execute one of the following OBs:
  - OB40 (Hardware interrupt)
  - OB52 - OB54 (ODK-specific interrupts)

Your ODK extension object can schedule OB52, OB53, or OB54 at any time. These OBs are not associated with an external event and are available specifically for ODK custom software. You can program any application-specific software in one or more of these OBs and control its scheduling from your custom C/C++ software.

- OB80 (Time error, such as a watchdog alarm)
- OB82 (Diagnostic alarm interrupt)
- OB83 (Insert/Remove Module interrupt)
- OB84 (CPU Hardware Fault)
- OB87 (Communication Error interrupt)

**Note**

Reserve OB84 for handling Windows operating system failures. Do not use it for other purposes.

- Read the current operating state of the controller

## What's New?

WinAC Open Development Kit 4.0 includes the following features that are new with this release:

- The ODK application wizard allows the creation of extension objects for either WinLC Basis or WinLC RTX. There are no longer separate versions of WinAC ODK for each type of WinAC controller.
- The extension objects that you build with WinAC ODK 4.0 are either standard DLLs or RTDLLs. WinAC ODK no longer builds COM objects. However, COM objects built with an earlier release of WinAC ODK can still be used by your control program.
- The ODK application wizard does not require you to specify the type of extension object when you create it. At build time you set the build target in your programming environment to specify whether to build a DLL or an RTDLL. With this feature, the same custom software can be built into different types of extension objects.
- New OBs are provided that you can schedule from an ODK extension object. These OBs, OB52 - OB54, exist specifically for ODK. You can program custom software in these OBs and then schedule them from your custom C/C++ software as required by your application.
- WinAC ODK 4.0 provides the function `ODK_SetWatchDog`. This function allows you to enable or disable the watchdog timer from debug builds of an extension object. It is particularly useful to prevent your debugging tasks from causing the watchdog timer to expire.
- WinAC ODK 4.0 provides the ability to assign a priority for an asynchronous thread relative to the main execution thread.

## System Requirements

### Hardware Requirements

To use WinAC ODK, your personal computer (PC) must meet the following criteria:

- Pentium processor running at 800 MHz or faster
- 128 Mbytes RAM
- Approximately 10 Mbytes on your hard disk
- At least 1 Mbyte free memory capacity on drive C for the Setup program (Setup files are deleted when the installation is complete.)
- A color monitor, keyboard, and mouse or other pointing device (optional) that are supported by Microsoft Windows 2000/XP.

### Software Requirements

WinAC ODK requires that the following software packages be installed on your computer:

- Microsoft Windows 2000 Professional Service Pack 3 (or higher) or Microsoft Windows XP Professional Service Pack 1
- STEP 7 version 5.2 (or higher)
- A C/C++ compiler (Microsoft Visual Developers Studio version 6 (Visual C++), service pack 5 (or higher) is recommended.)
- VenturCom Software Development Kit (SDK) version 5.1.2 or higher, required for compiling RTSS projects; not required for non-realtime projects.
- Internet Explorer 5.0 (or higher), for viewing product documentation

## Installing WinAC ODK

Before installing WinAC ODK, ensure that your computer meets the recommended system requirements.

### Notice

Do not install WinAC ODK or any other component of WinAC on a computer while any other component of WinAC is currently executing (running) on that computer.

Because SIMATIC Computing, WinAC controllers, and other elements of WinAC use common files, attempting to install any component of the WinAC software when any of the components of WinAC are executing can corrupt the software files.

Close all programs that are running before you install WinAC ODK.

The Setup program for WinAC ODK guides you step-by-step through the installation process. You can switch to the next step or to the previous step from any position. To install WinAC ODK, insert the WinAC ODK installation CD and follow the instructions displayed by the Setup program.

## Installing ODK When a Version of ODK Is Already Installed

If the Setup program finds another version of ODK on your computer, it displays a dialog that allows you to modify, repair or remove the installation. Select Remove on this dialog to uninstall the previous version. Repeat the installation procedure to install .

Your software is better organized if you uninstall any older versions before installing the new version. Overwriting an old version with a new version has the disadvantage that if you then uninstall, any remaining components of the old version are not removed.

## Uninstalling (Removing)

To use the Windows Add/Remove Programs procedure to remove the WinAC ODK software, follow these steps:

1. Select the **Start > Settings > Control Panel** menu command to display the Windows control panel.
2. Double-click the Add/Remove Programs icon to display the Add/Remove Programs Properties dialog box.
3. Select the entry for SIMATIC WinAC ODK and click the Add/Remove button.
4. Follow the dialog instructions to remove the software.

# Getting Started

## Introduction to the HistoDLL Sample Program

WinAC ODK installs several sample programs on your computer during the installation procedure. These STEP 7 programs and Microsoft Visual C++ projects contain software for collecting data about the scan time of the WinAC controller. The sample program HistoDLL collects data about the scan time of a WinAC controller. You can display the collected data as a histogram.

This sample program consists of the following elements:

- STEP 7 project HistoDLL, which includes:
  - STEP 7 program
  - Variable table (VAT) Histogram Status
- Microsoft Visual C++ Project HistoDLL

You can work with this sample program to gain an understanding about how to create your own custom software and use it with a STEP 7 program. You can generate one of three types of extension objects from the HistoDLL C++ project:

- RTDLL - for use with realtime applications for WinLC RTX
- DLL (proxy version) - for use with non-realtime applications for WinLC RTX
- DLL (standard version) - for use with WinLC Basis

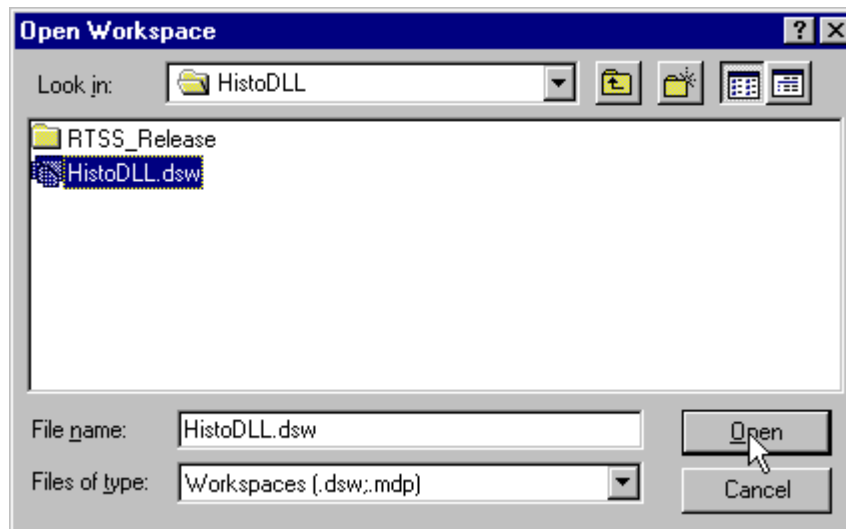
## How to Use the HistoDLL Sample Program

To work with the sample program, you must open the HistoDLL STEP 7 program and download it to your WinAC controller. You must compile the HistoDLL C/C++ Visual C++ project and generate either an RTDLL, a DLL (proxy version), or a DLL (standard version), depending on what type of WinAC controller you are using and whether you want to use the sample program as a realtime application or a non-realtime application. After you have built the appropriate extension object you can run the control program to collect scan time data. You can then observe its operation in STEP 7 through the sample Histogram Status variable table.

### Building the HistoDLL Visual C++ Project

To compile and build the sample HistoDLL project in Microsoft Visual C++, follow these steps:

1. Select **Start > Programs > Microsoft Visual Studio 6.0 > Microsoft Visual C++ 6.0** to open Visual C++. (If you are using another compiler, follow the procedures for the compiler that you are using).
2. Select **File > Open Workspace** and browse to the Siemens\WinAC\Odk\Examples\HistoDLL directory.




3. Select the HistoDLL.dsw file and click Open.
4. Build an Extension Object for HistoDLL as either an RTDLL or a DLL. For the sample program, the project name is HistoDLL.
5. If you built an RTDLL and your environment is not Microsoft Visual C++, then you must register the RTDLL. For the sample program, the project name is HistoDLL.





## Retrieving and Configuring the HistoDLL STEP 7 Program

To retrieve the HistoDLL STEP 7 program, and configure it for use with your controller, follow these steps:

1. Select **Start > Simatic > SIMATIC Manager** to start the SIMATIC Manager.
2. Select **File > Retrieve...** from the SIMATIC Manager.
3. Browse to the directory Siemens\WinAC\Odk\Examples\HistoDll\Step7\S7proj and double-click the file HistoDLL.zip.
4. Click OK on the "Select destination directory" dialog. The SIMATIC Manager extracts the HistoDLL STEP 7 project and puts it in the Siemens\Step7\S7proj directory. Click OK on the Retrieving dialog that informs you of the project location, if it appears.
5. Click Yes on the final Retrieve dialog to open the HistoDLL project.
6. In the HistoDLL project, change the station name corresponding to your WinAC controller to match the Station Name configured in the Station Configuration Editor. (To find the station name, click the Station Configuration Editor icon  in the Windows Taskbar, select the controller, and click the Station Name button.)

## Downloading and Running the HistoDLL STEP 7 Program

To download and run the HistoDLL STEP 7 program to your controller, follow these steps:

1. Start the WinAC controller:
  - If your WinAC controller is already operating but the control panel is not open, double-click the WinAC controller icon  in the Windows Taskbar to start the control panel.
  - If your WinAC controller is not operating, double click the control panel icon  on your desktop to start the control panel and the controller.
2. From the control panel, set the operating mode of the controller to STOP, and perform a memory reset (MRES).
3. From the SIMATIC Manager, click the Blocks folder of the HistoDLL S7 Program and select **PLC > Download** to download the HistoDLL program to your controller.
4. Change the operating mode of the controller from STOP mode to RUN or RUN-P mode. The controller begins executing the HistoDLL program.
5. From the SIMATIC Manager, use the Histogram Status variable table in the Blocks folder of the HistoDLL S7 Program to view the scan data. The table shows a running count of how many scans fall within specific time values and statistics about the scan times.

## Overview of the HistoDLL STEP 7 Program

The STEP 7 program HistoDLL stores data about the scan cycle (current execution time, last execution time, maximum execution time, minimum execution time, mode or most frequent scan time, and deviation) and calls the HistoDLL extension object that updates the scan cycle data.

The HistoDLL STEP 7 program uses the ODK System Function Blocks SFB65001 (CREA\_COM) and SFB65002 (EXEC\_COM). These SFBs encapsulate the calls to the ODK extension object functions. They provide the interface between the STEP 7 program and the HistoDLL extension object containing custom C/C++ software.

In addition, the HistoDLL STEP 7 project includes a variable table named Histogram Status. When the HistoDLL STEP 7 program is running with the ODK HistoDLL RTDLL, you can monitor this variable table to observe the scan cycle times and related statistics.

### STEP 7 Program Structure

The HistoDLL STEP 7 program uses the following blocks:

Block	Symbolic Name	Description
OB35	CYCL_INT	Cyclic Interrupt: This OB cycles at an interval of every 10 ms. It calls FC1002.
OB52	ODK_Triggered_OB	ODK Interrupt: This OB is executed when scheduled by an ODK extension object.
OB100	COMPLETE_RESTART	Complete Restart: This OB is executed on a Warm Restart (transition from STOP to RUN or RUN-P). It sets the ODK extension object status to an uninitialized state.
DB1000	HistoData	Data block containing the histogram data
DB1001	ODKLoad	Data block containing name and status of extension object
DB1002	ODKExecute	Data block containing execution parameters for the extension object
FC1002	UpdateHistogram	This FC creates and initializes the HistoDLL extension object if it is not yet created, and calls the extension object to update the histogram data. It also stores the current clock tick value and deviation count in the HistoData DB. (DB1000)
SFB65001	CREA_COM	This designated ODK SFB creates an ODK extension object.
SFB65002	EXEC_COM	This designated ODK SFB executes the custom software in an ODK extension object. The HistoDLL extension object software updates cycle time data for the histogram.
SFC78	OB_RT	Internal Timer that is used to get the current time.
SFC46	STP	Stop: This SFC changes the operating mode to STOP. The HistoDLL program only calls this SFC in the case of error.

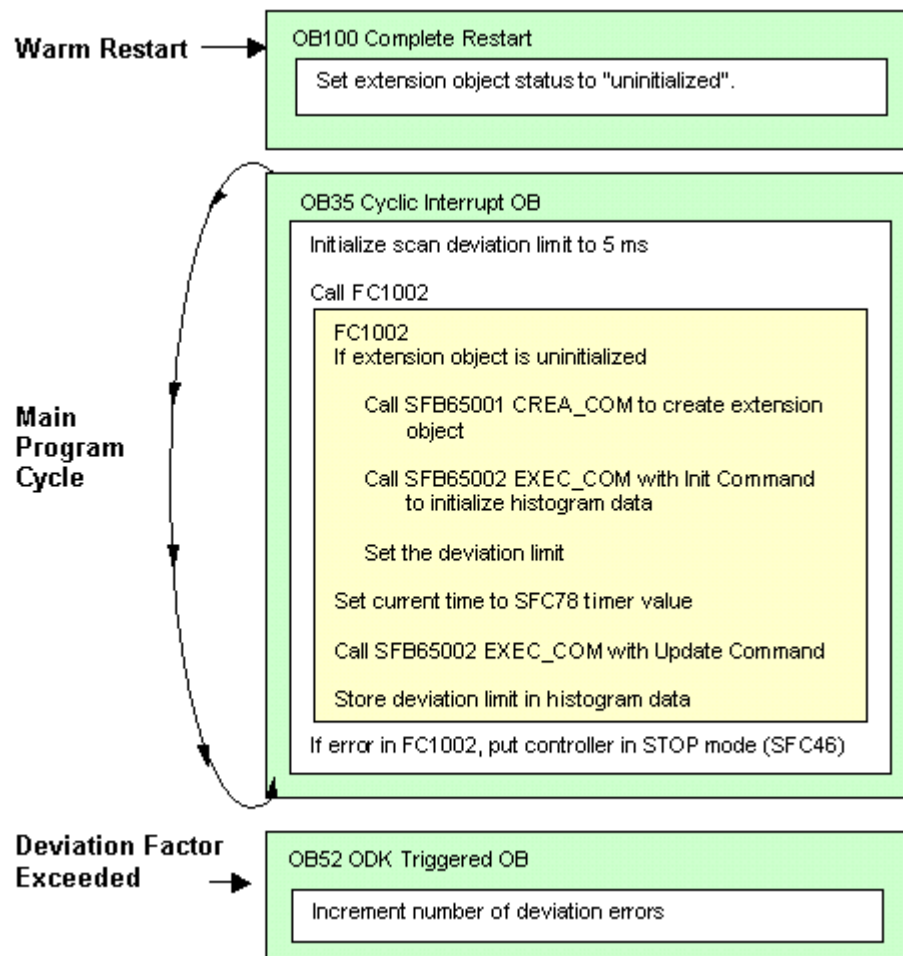
## STEP 7 Program Execution Logic

When the operating mode changes from STOP to RUN or RUN-P, the controller executes OB100 (warm restart). OB100 sets the state of the extension object to "uninitialized".

When the controller is in RUN or RUN-P mode, the controller executes OB35 every 10 milliseconds. OB35 calls FC1002 to update the histogram data. FC1002 performs these tasks:

1. Checks to see if the extension object is "uninitialized". If so, it performs these tasks:
  - a. Calls SFB65001 (CREA\_COM) to create the extension object for the histogram program if the state of the extension object is "uninitialized".
  - b. Calls SFB65002 (EXEC\_COM) with a command parameter to initialize the histogram program, and the location of the histogram data that it will be updating. The extension object sets all initial values of the histogram to 0 when called with a command to initialize.
  - c. Sets the deviation limit in the histogram data to 5. This value indicates the longest acceptable scan cycle deviation in milliseconds.
2. Calls SFC78 and stores the current clock time in the histogram data.
3. Calls SFB65002 (EXEC\_COM) with the command to update the histogram data. The extension object reads the current scan cycle data and calculates and updates the histogram data. The extension object checks to see if the deviation limit has been exceeded, and if so, schedules OB52 to increment the number of scan deviation errors.

The following picture shows a graphical representation of the HistoDLL STEP 7 program execution:



## **Where to Find STEP 7 Program Listings**

After you have retrieved the HistoDLL project or other example projects into the SIMATIC Manager, you can examine the STEP 7 program code. To examine HistoDLL program listings from the SIMATIC Manager, follow these steps:

1. Select **File > Open**.
2. Click the Browse... button, and browse to and expand the Siemens\Step7\S7proj directory.
3. Double-click the Histodll project.
4. Expand the HistoDLL S7 Program from the SIMATIC Manager browser, and open the Blocks folder.
5. Double-click any of the program blocks to examine them.

## Overview of the HistoDLL Visual C++ Program

The HistoDLL example project contains C/C++ functions that read the data about the WinAC controller scan time and write histogram data. It contains the custom functions that are called when the STEP 7 HistoDLL program calls SFB65001 or SFB65002.

The HistoDLL project includes the files HistoDLL.cpp and HistoDLLfunc.cpp as well as the data access helper class files necessary for the interface (WinLCReadData.h, WinLCReadData.cpp, WinLCReadWriteData.h, and WinLCReadWriteData.cpp). You can view these functions in the HistoDLL Object Web. The HistoDLL Object Web is accessible from the online help. You can access it from "HistoDLL Object Web" in the online help index, or from the Object Web topic in the table of contents.

The file HistoDLL.cpp contains the entry point for the extension object. The file HistoDLLfunc.cpp contains the ODK custom software to provide histogram data. The application wizard (WinAC ODK AppWizard) generated the initial framework for this project, which created an empty Execute function. This function and others were then programmed with software to collect and store histogram data. The HistoDLL custom code consists of the functions are described below:

### ODKCreate

The ODKCreate function is called when the STEP 7 program calls SFB65001 (CREA\_COM) to create the extension object. HistoDLL performs no custom software upon creation of the extension object.

### Activate

The Activate is called when the controller changes from STOP mode to RUN or RUN-P mode. It sets a handle variable for the extension object and disables the watchdog timer. HistoDLL has no other custom software to perform on a STOP to RUN transition.

### DeActivate

The DeActivate function is called when the controller changes from RUN or RUN-P mode to STOP mode. The DeActivate function sets the handle of the extension object to null and enables the watchdog timer. HistoDLL has no other custom software to perform on a RUN to STOP transition.

### ODKRelease

ODKRelease is called when the controller is shut down, or when the memory is reset. Shutting down the controller or resetting the controller memory releases the extension object. HistoDLL has no custom software to perform in its ODKRelease function.

### Execute

The Execute function is called when the STEP 7 program calls SFB65002 (EXEC\_COM). The execute function uses the data access helper classes for input and output with the STEP 7 program. The Execute function calls either Executelnit or ExecuteUpdate depending on whether the command parameter to SFB65002 is an Init command (0) or an Update command (1).

### Executelnit

The Executelnit function is called from the Execute function on an Init command (0). It sets all of the histogram data values to 0 and sets the minimum scan time to a very large value. (This allows the first scan to set the initial minimum scan time.)

## ExecuteUpdate

The ExecuteUpdate function is called from the Execute function on an Update command (1). It reads the current values of the histogram data for the last scan time, minimum scan time, maximum scan time, and mode scan time. Using the current time value, it calculates the last scan cycle time. It updates the minimum or maximum scan time value if necessary. It calculates the mode time, and stores the scan cycle time for the cycle in the histogram array of scan cycle times. It checks to see if the last scan cycle time has exceeded the deviation time limit, and if so calls ScheduleOB80.

## ScheduleOB52

The ScheduleOB52 function is called from ExecuteUpdate if the scan cycle time exceeds the deviation time limit. It uses one of the auxiliary STEP 7 interface functions, ODK\_ScheduleOB to do this. For most applications, exception cases in the Execute function can be handled with a return code, while the scheduling of an OB would more commonly be done when an asynchronous event occurs.

## HistoDLL Object Web

HistoDLL.cpp contains the custom code for the HistoDLL sample project. It contains the functions Execute, ExecuteInit, and ExecuteUpdate. ExecuteInit and ExecuteUpdate are subcommands of the Execute function. These three functions contain the custom code; the rest of the code is supplied by the application wizard.

To examine the classes and functions that comprise the HistoDLL C/C++ sample project, view the online help and access the HistoDLL Object Web. You can access it from “HistoDLL Object Web” in the index, or from the Object Web topic in the table of contents.

## Additional Sample Programs

In addition to the HistoDLL sample program, WinAC ODK installs these additional sample C/C++ projects on your computer in the Siemens\WinAC\ODK\Examples folder:

C/C++ Program	STEP 7 Program	Description
Latency	Latency	Measures latency between OB scheduling and execution
TonePulse	TonePulse	Interfaces with external hardware to generate a tone pulse
FileIO	FileIO	Monitors a counter, creates events that schedule OBs, and performs file I/O
NETHistoDLL	NETHistoDLL	HistoDLL sample program for use as a .NET application
NETFileIO	NETFileIO	FileIO sample program for use as a .NET application

You can examine each of these programs for further examples of how to use ODK. They illustrate tasks such as the following:

- Using a monitor thread (FileIO)
- Scheduling OBs (Latency, HistoDLL)
- Measuring latency in scheduled OB execution (Latency)
- Interfacing with external hardware (TonePulse)
- Reading from and writing to files (FileIO)

### Note

You must retrieve the STEP 7 example programs into the SIMATIC Manager to be able to use them.

# Developing an ODK Application

To develop an ODK application, you implement C/C++ software and a STEP 7 program that uses this software. You build an extension object (DLL or RTDLL) from the C/C++ software, and you call ODK System Function Blocks (SFBs) in your STEP 7 program to access this extension object.

## Creating the C/C++ Project

To create the C/C++ software that is the WinAC controller can execute, follow these steps:

- Use the application wizard to create the program shell for the RTDLL or DLL.
- Implement custom software in predefined functions.

## Creating the STEP 7 Program

To create a STEP 7 program that uses ODK to access your custom C/C++ functions, follow these steps:

1. Load the STEP 7 library that contains the SFBs supplied by WinAC ODK.
2. Develop your STEP 7 program with calls to the ODK SFBs.

When you have developed both the extension object and the STEP 7 program, you can run the STEP 7 program on your WinAC controller and debug your extension object.

## Using the Application Wizard

The application wizard helps you perform the following tasks:

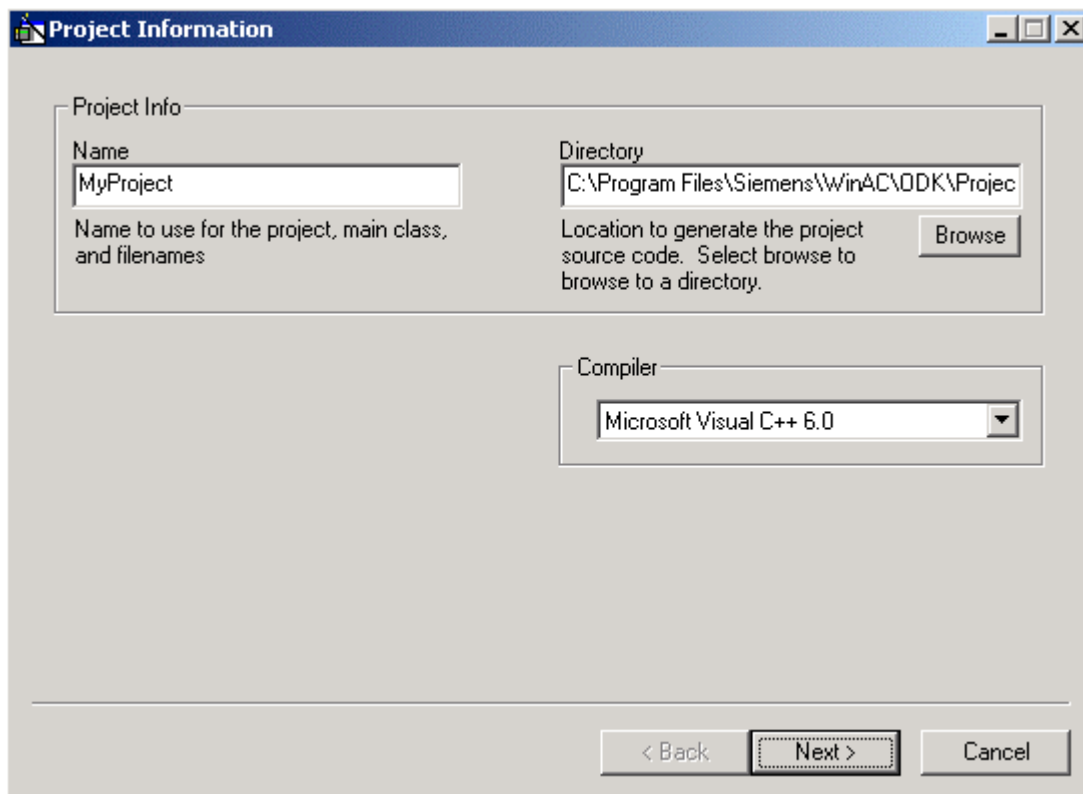
- Create the C/C++ project that implements the WinAC ODK interface.
- Configure subcommands for your extension object.
- Create a C++ class that you can use to execute functions asynchronously from the WinLC scan cycle (optional).
- Create a C++ class that you can use to asynchronously monitor one or more attributes of your system (optional).

These tasks are described in more detail in the topics that follow. After you complete these tasks, the application wizard provides a summary of the options that you selected. After you confirm these choices, the application wizard generates the C/C++ project shell.

## Configuring Project Information

To start the WinAC ODK Application Wizard, and configure project information data, follow these steps:

1. Select **Start > Simatic > PC Based Control > WinAC ODK AppWizard**.
2. Enter the name for your application in the Name field of the Project Information dialog.
3. Accept Microsoft Visual C++ 6.0 for the Compiler field, or select "other" from the drop-down list if you are using a compiler for a project other than Microsoft Visual C++ 6.0.
4. Click Next when you are finished with this dialog.





## Entering ODK Project Subcommands

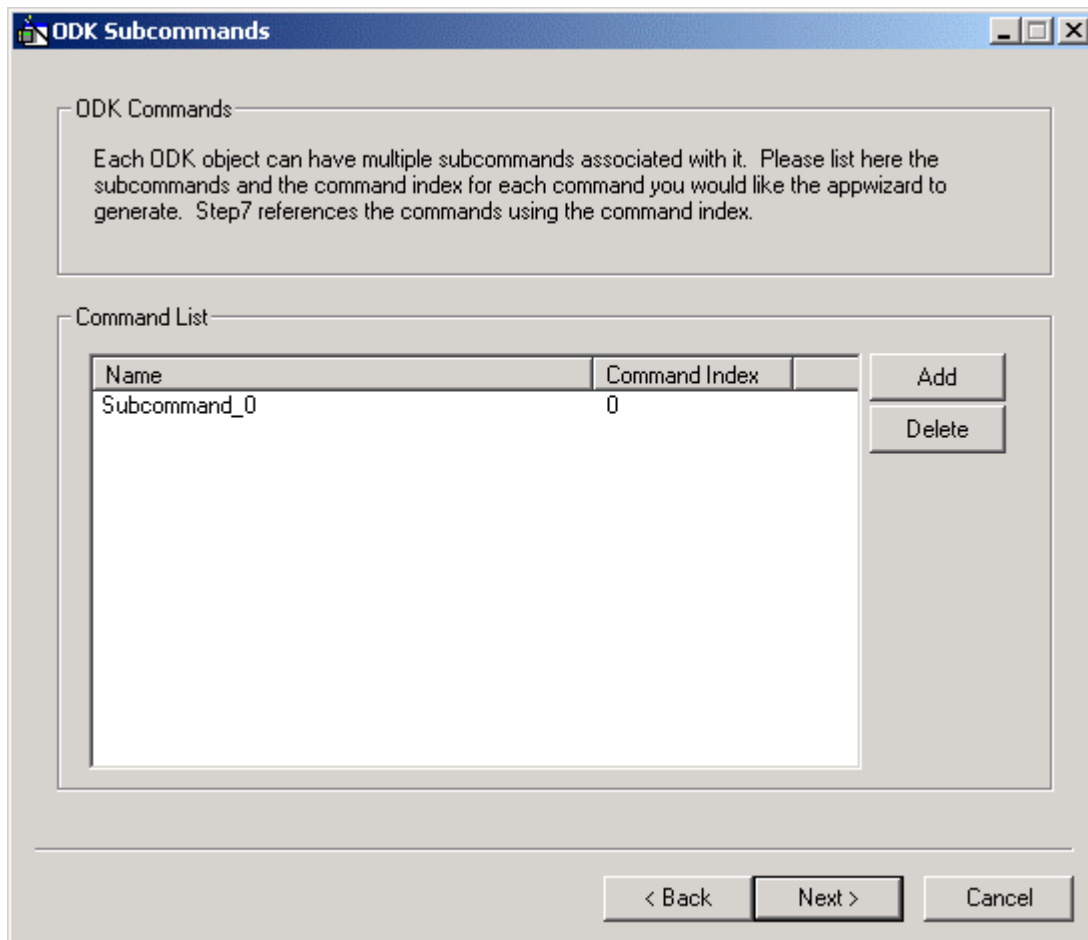
Your C/C++ program can call subcommand functions to further customize your program's functionality. A subcommand is a unique function within your C/C++ program that can be called from SFB65002 (EXEC\_COM). By using subcommands, you can create one RTDLL or DLL that performs a variety of tasks, rather than using several RTDLLs or DLLs that perform single tasks.

From the STEP 7 program, you specify which subcommand to call in the parameter Command for SFB65002 (EXEC\_COM). When your STEP 7 program calls SFB65002 (EXEC\_COM), it calls the Execute function of your DLL or RTDLL, which in turn calls the subcommand function specified by the Command parameter.

### Note

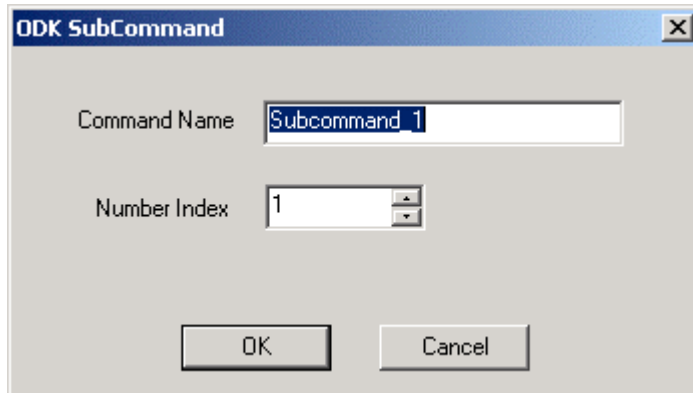
The application wizard requires that you specify at least one subcommand; however, your RTDLL or DLL does not have to make use of subcommands. If you do not have more than one type of task for your custom software to perform, you can implement the custom software directly in the Execute function and eliminate the subcommand functions produced by the application wizard.

When you click Next on the Project Information dialog, you see the ODK Subcommands dialog as shown below. You use this dialog to add or delete subcommands for your C/C++ program.



To configure the subcommands for your extension object, follow these steps as needed:

1. Click the Add button to add a new subcommand.
2. Enter the command name and number index in the ODK SubCommand dialog box, and click OK.



**Note**

Highlight a subcommand in the ODK Subcommands window, and click the Delete button if you need to delete a subcommand. If you need to rename a subcommand, first delete the subcommand, and then add it using the new name.

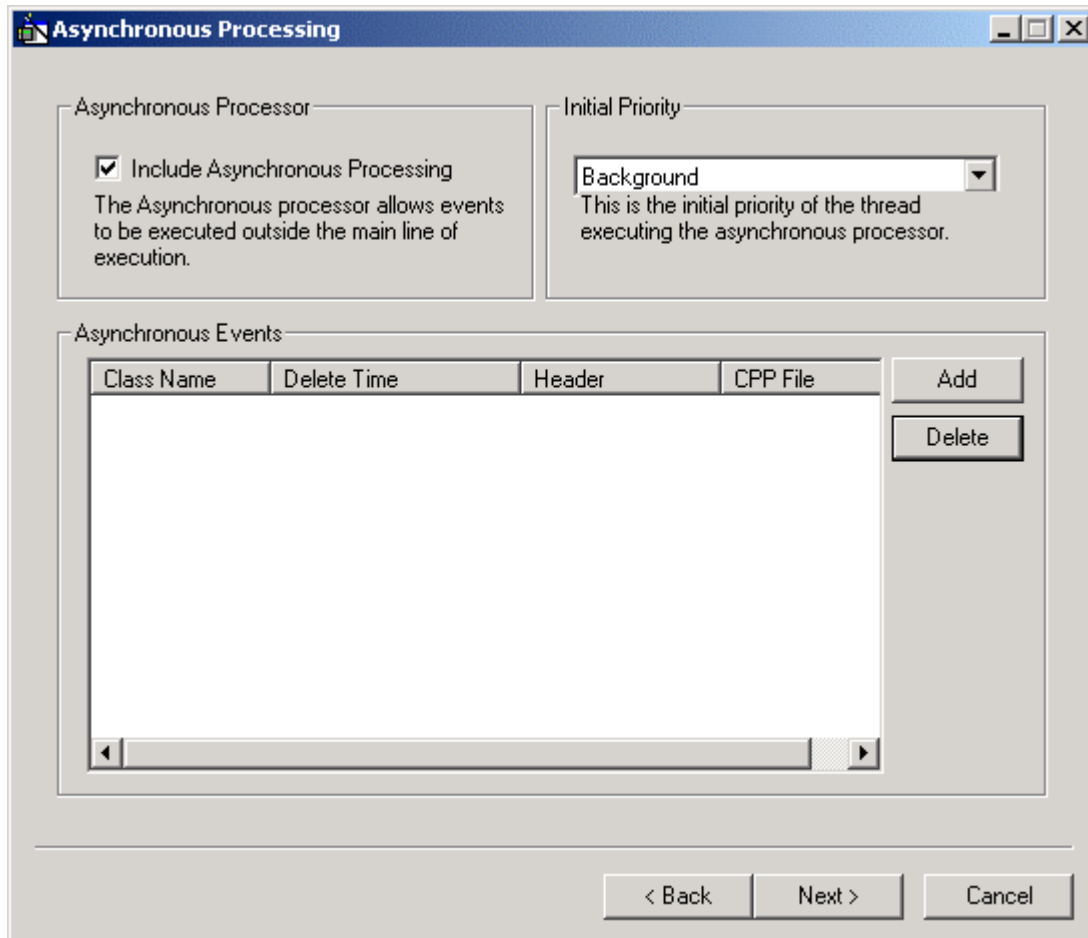
3. Click Next on the ODK Subcommands dialog when you have finished subcommand configuration.

## Enabling Asynchronous Processing

Asynchronous Processing allows commands to be executed outside of the main Execute function. These commands can then be executed without penalizing the controller scan cycle. The asynchronous processor uses objects derived from the EventMsg class to carry out event specific functions. Asynchronous processing is optional.

To use Asynchronous Processing in your WinAC ODK project, follow these steps:

1. For the Asynchronous Processing dialog of the Application wizard, select the Include Asynchronous Processing check box.

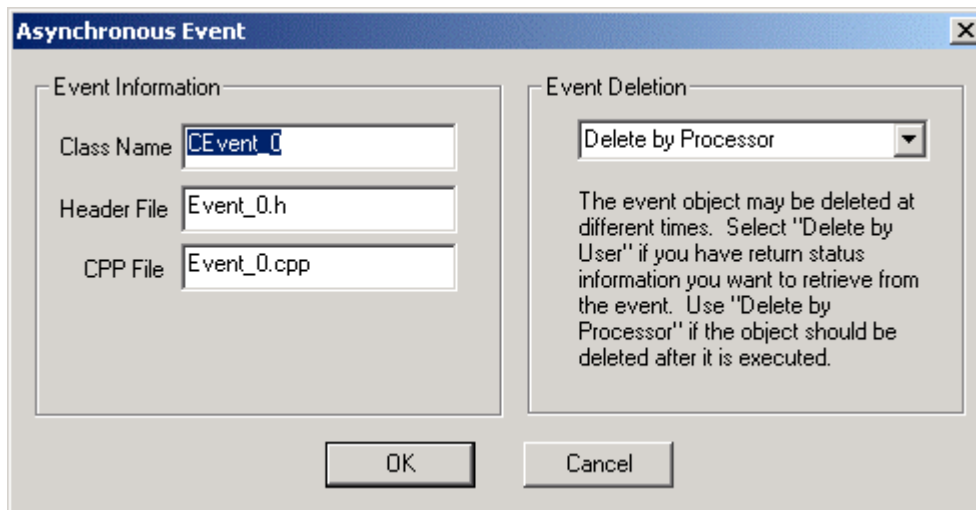


2. Choose a thread priority from the drop-down list. A background priority means that the thread executes at a lower priority than the main execution thread. A foreground priority means that the thread executes at a higher priority than the main execution thread, and is to be used only with extreme caution. Click OK to close the Monitor Thread dialog.
3. Click the Add button to add an asynchronous event.

### Note

If you need to delete an asynchronous event, highlight the event on the Asynchronous Processing dialog, and click Delete. If you need to rename an asynchronous event, first delete it, and then add it using the new name.

4. Accept the default or enter the class name for the event in the Asynchronous Event dialog box, and click OK. The application wizard names the header file (.h) and C++ source file (.cpp) based on the class name that you entered.

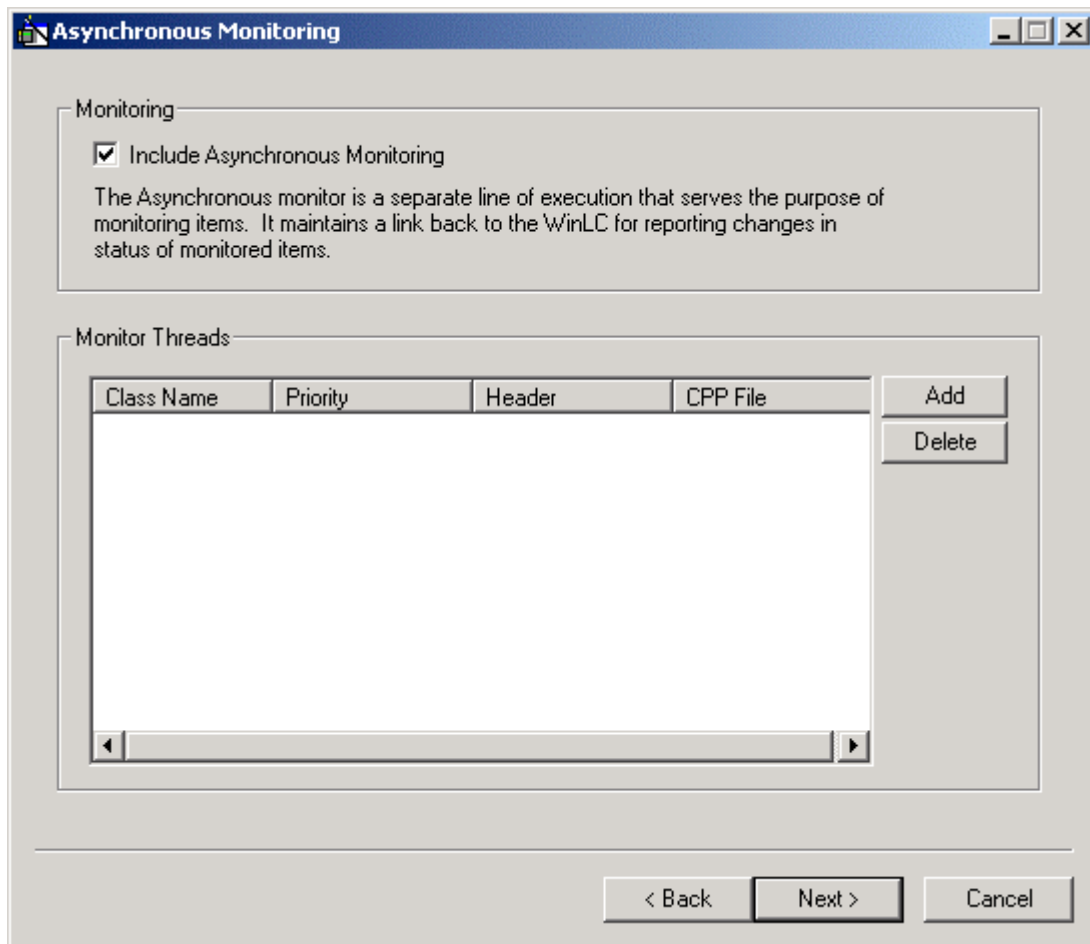


5. Select the type of Event Deletion that you prefer, based on the instructions on the dialog, and click OK.
6. Click Next when you have completed Asynchronous Processing configuration.

## Enabling Asynchronous Monitoring

After all of the event classes have been given a name, the application wizard displays the option to enable asynchronous monitoring. Like asynchronous processing, asynchronous monitoring is optional. Use asynchronous monitoring if your RTDLL or DLL needs to implement some functionality in the background (for example, to monitor data or wait for an event to occur that is asynchronous from the scan cycle). A monitor thread is a separate line of execution that allows the WinAC controller to perform this type of background activity.

1. Select the check box for Include Asynchronous Monitoring to include asynchronous monitoring in your WinAC ODK project.

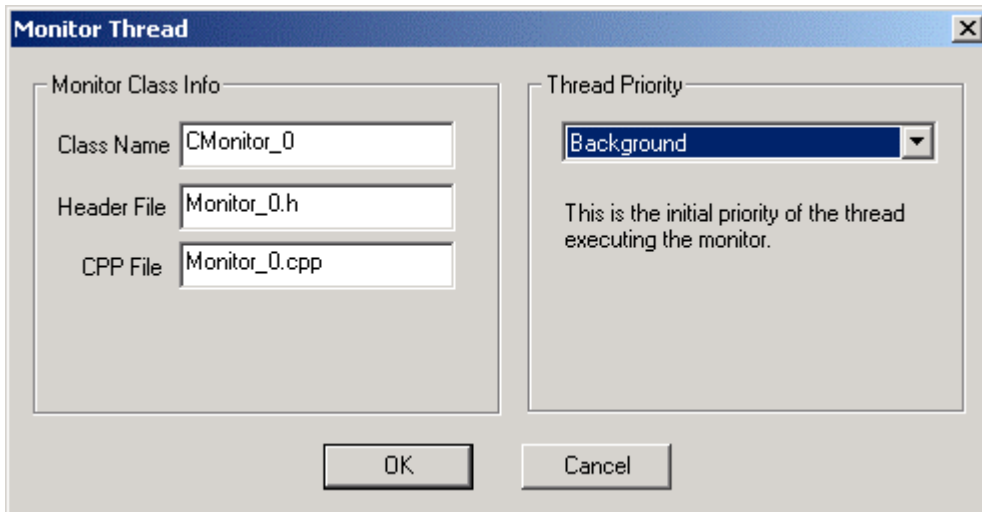


2. Click the Add button to add a monitor thread.

**Note**

Highlight a monitor thread on the Asynchronous Monitoring dialog and click Delete if you need to delete a monitor thread. If you need to rename a monitor thread, first delete it, and then add it using the new name.

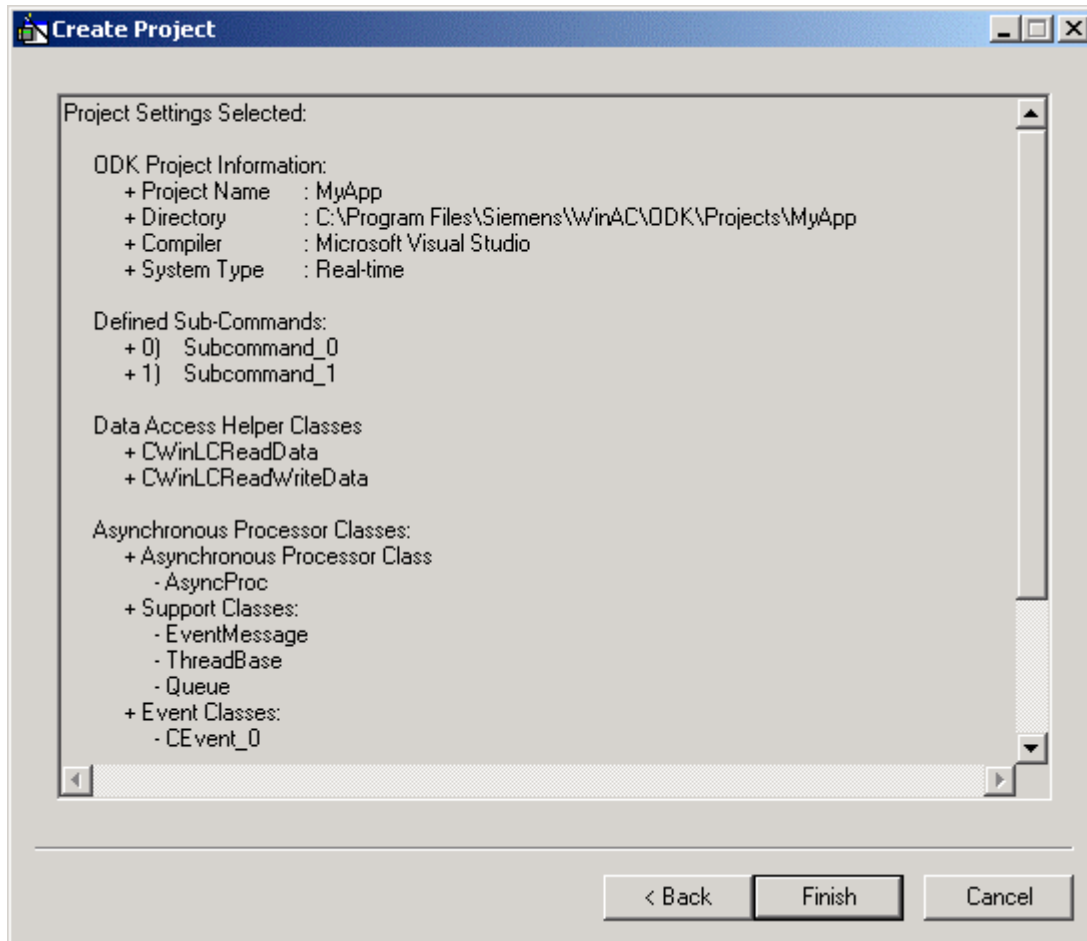
3. Enter a class name for the monitor thread in the Monitor Thread dialog. The application wizard names the header file and the cpp file based on the class name that you entered.
4. Choose a thread priority from the drop-down list. A background priority means that the thread executes at a lower priority than the main execution thread. A foreground priority means that the thread executes at a higher priority than the main execution thread, and is to be used only with extreme caution. Click OK to close the Monitor Thread dialog.



5. Click Next when you have completed asynchronous monitoring configuration.

## Generating the WinAC ODK C/C++ Project

After you have finished the configuration, the application wizard displays a project summary window, shown below:



To direct the application wizard to create the project framework, follow these steps:

1. Click Next to confirm the project options and to generate the WinAC ODK project. (If you want to make changes, click Back and correct the items that you need to change.) The application wizard creates the C/C++ project for you.
2. Click Next on the Project Creation Status dialog to display the Project Created dialog. If you are using Microsoft Visual C++ and want to immediately open the project in Visual Studio, check Open Visual Studio Project.

### Note

If you are not using Microsoft Visual C++, you must use your C++ programming interface to compile the set of .h and .cpp files that the application wizard generates. By default, these files are located under the directory configured on the Project Information dialog of the application wizard. By default, this pathname is:

Program Files\Siemens\WinAC\ODK\Projects\<>your project name>.

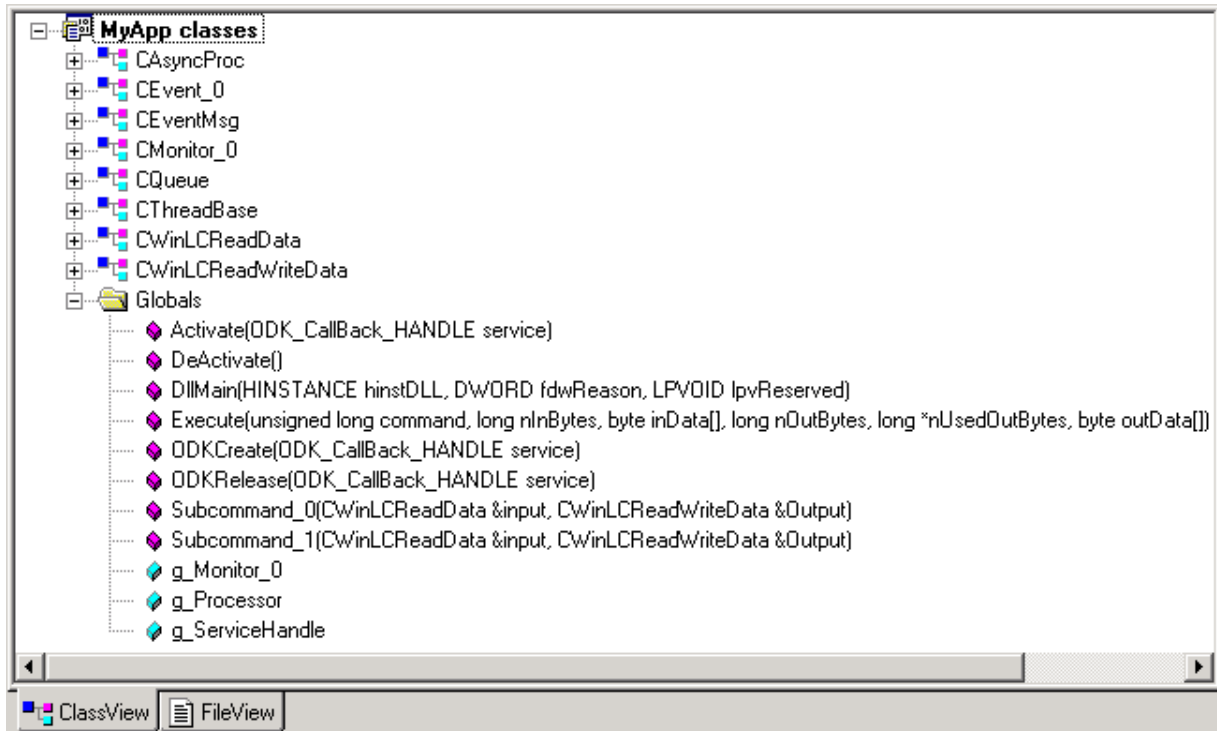
## Project Shell

The WinAC ODK application wizard creates a program shell for your project. It is a C/C++ project where the classes and functions are supplied for you. The places for you to enter your custom software are marked by comment lines that begin as follows:

```
//TODO:
```

## Class View of the C/C++ Object Shell

The following diagram shows the structure (in a Visual Studio environment) of the classes and functions in the project that the application wizard created. You supply code in the subcommand functions and optionally in the Execute function.





## Programming the C/C++ Software

### Structure of the ODK C/C++ Project

The WinAC ODK Application Wizard creates a skeleton C/C++ project that you use to generate your custom extension object.

The C/C++ project contains global functions that are the interface to the STEP 7 program. The application wizard creates a shell for each of these functions, and you fill in the custom code to accomplish your objectives. The custom functions that you implement are:

- ODKCreate (can be left empty)
- Activate (can be left empty)
- Deactivate (can be left empty)
- ODKRelease (can be left empty)
- Execute
- Execute function subcommands (if configured)

### The ODKCreate Function

The ODKCreate function is responsible for creating the extension object. If you have any processing to perform when the object is initially created, put it in ODKCreate; otherwise, you can leave this function as is.

### The Activate Function

The Activate function is called by the WinAC controller before the transition from STOP or HALT to STARTUP or RUN. Activate is always called after the extension object is created and before the first call to the Execute function. If you have any processing to perform before the first call to Execute, you can put it in the Activate function; otherwise, you can leave this function as is.

#### Note

The CPU is in HALT when it reaches a breakpoint in the STEP 7 editor.

### The DeActivate Function

The DeActivate function is called by the WinAC controller after the transition from STARTUP or RUN mode to STOP or HALT mode. If you have any processing to perform following the transition to STOP or HALT, you can put it in the DeActivate function; otherwise, you can leave this function as is.

#### Note

The WinAC controller releases your extension objects on a memory reset (MRES) or on a controller shutdown. Because both the MRES and controller shutdown first change the controller to STOP mode, DeActivate is always called before RTDLLs and/or DLLs are released.

### The ODKRelease Function

The ODKRelease function is responsible for releasing the extension object. If you have any processing to perform when the object is released, put it in ODKRelease; otherwise, you can leave this function as is.

## The Execute Function and its Subcommands

The Execute function is executed when your STEP 7 program calls SFB65002 (EXEC\_COM). The processing in your Execute function becomes part of the OB execution time. The STEP 7 program can pass parameters to the Execute function, which can, in turn, call subcommand functions based on the input parameters. You implement most of your custom software in the Execute function and its subcommand functions. Typically, your Execute function switches control to one of the subcommand functions based upon the input parameters to the Execute function.



### Caution

The custom software in the Execute function and the subcommand functions are part of the main program cycle. The STEP 7 program executes the SFB that calls the ODK custom software as a single instruction. This instruction call is non-interruptible. During the execution of the ODK custom software, the watchdog timer, OBs, and process alarms are not available. They are handled after the SFB call to the ODK custom software completes. Custom software that significantly extends the cycle time delays the processing of critical activities in WinLC RTX, which can possibly result in personal injury.

To avoid this delay, do not put any processing in the Execute or subcommand functions that would extend the scan cycle beyond an acceptable cycle time. Program asynchronous events to handle custom logic of a lengthy duration.

## ODK Support Classes

The C/C++ project produced by the application wizard also includes the following classes:

- **Data access helper classes:** The application wizard always generates the CWinLCReadData and CWinLCReadWriteData classes. These classes are used to exchange data between the WinLC RTX buffer and the C/C++ program. You do not program custom software in any of the Data access helper class functions.
- **Asynchronous processor classes:** The application wizard generates the CAsyncProc, CQueue, and CThreadBase classes only when you select the Asynchronous Processing option. The wizard generates classes for each asynchronous event that you request.
- **Monitor classes:** The application wizard generates Monitor classes only when you select the Asynchronous Monitoring option. The wizard creates one class for each monitor thread that you request.

The following table describes the support and helper classes generated by the application wizard:

Type	Class	Description
Data access helper class	CWinLCReadData	The read-only data access helper class serves as a wrapper to the input buffer passed into the Execute function. You can use functions in this class to access the data in the input buffer as STEP 7 data types.
	CWinLCReadWriteData	The read/write data access helper class serves as a wrapper to the output buffer passed into the Execute function. You can use the functions in this class to read and write STEP 7 data types to the output buffer.
Asynchronous processor class	CAsyncProc	The asynchronous processor class processes events posted to it on a thread of execution separate from the main program.
	CEventMsg	The CEventMsg class is the base class for Asynchronous Events: all asynchronous events posted to the asynchronous processor must be derived from this class. For each asynchronous event, you must override the Execute function of the class derived from CEventMsg to provide custom processing for the event.
	CQueue	This is a basic queue (first in, first out) class. The asynchronous processor uses it for scheduling events to process.
Monitor class	CThreadBase	This is the base class for asynchronous monitors: all classes for monitoring external events and processes must be derived from this class. When you use asynchronous monitoring, you must override the Execute function to provide customized monitoring actions.

## Programming Asynchronous Events

You can define events to execute asynchronously on a thread of execution separate from the WinAC controller scan cycle. This allows the extension object to schedule and execute actions that can take a long time while allowing the controller to continue processing. You specify the number of asynchronous events that you want in your extension object when you set up your project with the application wizard. The asynchronous processor executes each of these events on a thread of execution separate from the scan cycle of the controller.

Consider an example where a WinAC controller controls a stapler assembly line. One of the control program requirements is to send an E-mail notification when supplies are running low. You can program a WinAC ODK extension object to accomplish this task. However, putting all of the E-mail functionality in the Execute function could cause an unacceptable increase in the scan cycle. You could avoid this problem by using the asynchronous processor (AsyncProc) to schedule a "Send E-mail Notification" event. This allows the WinAC controller to maintain a fast scan cycle while sending E-mail at the same time.

### Note

Do not call non-deterministic functions such as RtPrintf from the main thread. Allow the asynchronous processor to handle these functions in asynchronous event threads.

Asynchronous processing uses the following classes:

- **CAsyncProc**: This class processes events posted to it on a thread of execution separate from the main program.
- **CEventMsg**: This is the base class to Asynchronous Events: all asynchronous events posted to the asynchronous processor must be derived from this class. The Execute function must be overloaded in the derived class to provide custom processing for the event.
- **CQueue**: This is a basic queue (first in, first out) class. The asynchronous processor uses it for scheduling events to process.

The application wizard adds an event class for each asynchronous event that you added. Each event class is derived from the CEventMsg base class. The Execute function of each event class is where you implement your asynchronous processing code and is indicated by the line below:

```
// TODO: Add Code to customize this Event
```

### Note

If you add asynchronous event classes outside of the application wizard, you must also derive them from the CEventMsg class and implement the asynchronous event processing code in its Execute function.

To schedule asynchronous event processing, you must implement code in the global Execute function that is called during the main control program scan cycle or in one of the Execute subcommands.

The Execute function of the main program cycle must perform the following tasks:

1. Create an object of the derived event class using the "new" operator.
2. Call the ScheduleEvent function of the CAsyncProc class to post the event to the asynchronous processor.
3. Call the GetStatus function of the event class to determine if the event has been processed.
4. Call the GetResult function of the event class to get the success/fail status returned from the event's Execute function.

## **Asynchronous Events**

All events posted to the asynchronous processor must be derived from the CEventMsg class. In the derived class, the event execution code must be placed in an override of the Execute function. The asynchronous processor calls this Execute function to perform the event-specific tasks.

You are responsible for creating the event object on the heap (for example, using the "new" operator). However, you can use the SetDelTime function to set the responsibility for deallocating the object's memory. You can specify deallocation either by the asynchronous processor or by your code, depending on whether or not your application requires post-processing status information.

## Programming Monitor Threads

You can use monitor classes to provide a separate line of execution for WinAC ODK to monitor events external to its process. With the application wizard, you can configure whether or not your application uses asynchronous monitoring and the number of monitoring threads that you need. The application wizard creates a monitor thread class for each thread, which is derived from the base class, CThreadBase. You can create monitor thread classes outside of the application wizard, but they must be derived from the CThreadBase class.

You program the thread monitoring loop in the Execute function of each monitoring thread class. Place all custom processing or monitoring immediately following the comment:

```
// TODO: Add Customized Monitoring Code here
```

### Note

The application wizard creates a function shell for the Execute function of each monitor class. You implement this function, but you do not call the Execute function.

The ODK Object Web represents a sample ODK project created with the application wizard. It contains one monitor thread whose class name is CMonitor\_0. CMonitor\_0 is derived from CThreadBase. To program this monitor thread, you would implement code in the Execute function of CMonitor\_0.

The ODK Object Web is accessible from the online help. You can access it from “Object Web” in the online help index, or from the Object Web topic in the table of contents.

## Monitor Thread Considerations

When the WinAC controller creates an extension object that has been configured or programmed to use monitor threads, it starts a separate thread of execution for each monitor thread. The software in a monitor thread runs asynchronously from the main thread of your extension object.

If an event such as a controller shutdown or memory reset causes your extension object to be released, the WinAC controller calls the DeActivate function of the main extension object execution thread. The Deactivate function calls PauseThread to pause the monitor threads, and then calls the ODKRelease function, which calls StopThread to stop the monitor threads. The StopThread function stops the threads, sets a variable named m\_ExitThread to true, and then calls ResumeThread to resumes the threads so that they can terminate appropriately. The software in the monitor thread can check the variable m\_ExitThread to see if the main thread has terminated. If your monitor thread makes calls outside of the extension object, such as a call to schedule an OB, the monitor thread code must check the m\_ExitThread variable.

### Notice

If your monitor thread makes a call outside of your extension object, and an event such as a controller shutdown or memory reset causes the extension object to be released, your code can not return properly. This is because the extension object is unloaded following the controller shutdown or memory reset event. Because it is unloaded, the call outside of the extension object from the asynchronous thread has no place to return.

You can guard against this possibility by programming the monitor thread to check the m\_ExitThread variable to see if the extension object has been released. In your monitor thread code, place any calls to functions external to the extension object within a check of this variable as follows:

```
if (!m_ExitThread)
{
    < external call >
}
```

## Examples

The programs Latency and FileIO in the folder \Program Files\Siemens\WinAC\ODK\Examples demonstrate how to use and program monitor threads.

## Building the Extension Object

After you have programmed your custom software, you must compile it into either an RTDLL or a DLL. Once it is built and loaded, your STEP 7 program can use it. To build your extension object, follow these steps:

1. Select **Build > Set Active Configuration...** to set the project configuration. The choices for project configuration are as follows:

Project Configuration	Description
<project name> - Win32 Release	Standard DLL for use with WinLC Basis
<project name> - Win32 Debug	Debug version of standard DLL for use with WinLC Basis
<project name> - Win32 Proxy Release	Proxy DLL for use with WinLC RTX non-realtime applications
<project name> - Win32 Proxy Debug	Debug version of proxy DLL for use with WinLC RTX non-realtime applications
<project name> - RTSS Release	RTDLL for use with WinLC RTX realtime applications (requires the VenturCom SDK)
<project name> - RTSS Debug	Debug version of RTDLL for use with WinLC RTX realtime applications (requires the VenturCom SDK)

2. Select the project configuration that corresponds to your needs and click OK.
3. Select **Build > Build <project name>.dll** or **Build > <project name>.rtdll** to build the extension object.

### Note

Building an RTDLL from Microsoft Visual C++ registers the RTDLL on your machine. Your STEP 7 program will use this registered RTDLL regardless of the pathname in the Data Block containing the load information.

If you built an RTDLL and your programming environment is not Microsoft Visual C++, then you must register the RTDLL. You must also register the RTDLL if it was built on a different machine. After it is registered, STEP 7 will use this RTDLL regardless of the pathname specified in the Data Block.

The pathname in the Data Block is not necessary for an RTDLL.

Further directions for using the debug versions are described in Debugging the Extension Object.

## Registering an RTDLL

Building an RTDLL extension object from Microsoft Visual C++ registers your RTDLL. Your STEP 7 program will use this registered RTDLL regardless of the pathname specified in the data block that provides the extension object location. However, in either of the following circumstances, you must register the RTDLL yourself from a DOS command prompt window:

- Your programming environment is not Microsoft Visual C++
- You are using an RTDLL on one machine that was built on another machine.

In either case, you must run a command to register your RTDLL. This step is not necessary for DLLs (either proxy or standard versions), or for RTDLLs generated from Microsoft Visual C++. In Microsoft Visual C++, the Build command for an RTDLL loads and registers the RTDLL..

To register the RTDLL from an environment other than Microsoft Visual C++, you must use the `rtssrun` command. After you build the RTDLL, follow these steps:

1. Select **Start > Programs > Command Prompt** to open a DOS command prompt window.
2. Type `cd siemens\WinAC\Odk\Projects\\RTSS_Release` to change to the folder containing the RTDLL that you just built. (If you are building one of the example projects, use the "Examples" folder instead of the "Projects" folder.)
3. At the command prompt, type `rtssrun /dll <project name>.rtdll` to register the RTDLL.

## Implementing the STEP 7 Project

### Loading the WinAC ODK Library Into STEP 7

WinAC ODK includes a custom STEP 7 library that contains the SFBs that allow the STEP 7 program to interact with your custom RTDLL or COM object. Before you can use these SFBs in your STEP 7 program, you must retrieve the library and load it into STEP 7. To retrieve and load the library, follow these steps:

1. Select **Start > SIMATIC > SIMATIC Manager** to start the SIMATIC Manager.
2. Select the **File > Retrieve...** menu command.
3. Browse to the directory `\Program Files\Siemens\WinAC\Odk\Step7\S7libs` and double-click the file `OdkLib.zip`.
4. Select `S7libs` on the "Select destination directory" dialog and click OK. The SIMATIC Manager extracts the WinAC ODK library into `OdkLib` in the `Siemens\Step7\S7libs` directory. If a dialog appears that informs you of the project location, click OK.
5. Click Yes on the final Retrieve dialog to open the WinAC ODK library. This library contains the following blocks:
  - SFB65001 (CREA\_COM)
  - SFB65002 (EXEC\_COM)



## Programming the STEP 7 Program

WinAC ODK provides two SFBs that allow you to use your custom RTDLL or DLL as part of the STEP 7 control program. These two SFBs are listed below:

- SFB65001 (CREA\_COM), which creates the instance of your extension object
- SFB65002 (EXEC\_COM), which sends an execution command to the extension object created by SFB65001

SFB65001 (CREA\_COM) calls the ODKCreate function to create your extension object, and then the Activate function of the extension object.

SFB65002 (EXEC\_COM) calls the Execute function of your ODK extension object, and does not return until the custom software in the Execute function finishes. The scan time is therefore extended by the time required to execute this function. If you need to execute a command that can take a long time relative to your scan time requirements, use the asynchronous processor (AsyncProc) to execute the command. Using the asynchronous processor avoids making the scan time longer than you require.

In order to use these SFBs in your STEP 7 program, you must have loaded the WinAC ODK library. You can then insert these SFBs into your program just like any other STEP 7 element. To open the WinAC ODK library and insert the ODK SFBs into your program, follow these steps:

1. Select **Start > SIMATIC > SIMATIC Manager** to open the SIMATIC Manager.
2. Select **File > Open...** and select your project.

### Note

If you are creating a new STEP 7 project, select **File > New...** and enter a filename for your project.

3. Select **File > Open...** and select the Libraries tab. Double-click WinAC ODK Library.
4. Select and drag the SFBs from the WinAC ODK library to the program blocks of the STEP 7 program. The ODK SFBs are now available for you to use in your STEP 7 program.
5. Edit your program to call SFB65001 (CREA\_COM).

### Note

Typically, the STEP 7 program calls SFB65001 (CREA\_COM) when the program starts in the start-up OB (such as OB100) to create the instance of the RTDLL or DLL. The program handle returned from this call is then saved to a memory location for further reference. Your STEP 7 program can also call SFB65001 from other logic blocks, such as an FB.

6. Edit your program to call SFB65002 (EXEC\_COM).

### Note

If you want your program to execute your custom software every scan cycle, then put the call to SFB65002 in OB1, or in an FB called from OB1.

Your STEP 7 program is now programmed to use the custom C/C++ software that you created. You can download it to your controller.

## Debugging and Changing the Extension Object

### Debugging the Extension Object

While the WinAC controller is executing the STEP 7 control program that calls your extension object, you can test and debug your custom software using the Visual C++ debugger.

**Note**

The controller can exceed the maximum scan cycle time when you use breakpoints in your extension object. You can use the ODK\_DISABLE\_WATCHDOG macro in your C/C++ code to prevent overruns in the scan cycle time.

To prepare to debug an ODK extension object you must perform these tasks:

- Build a debug version DLL of your custom software.
- Configure and download the STEP 7 program.
- In a Windows-only environment without RTX, prevent the control panel from automatically starting the controller.

**Note:** This step is not necessary for WinLC RTX extension objects; however, to debug an RTDLL for use with WinLC RTX, you must build it as a proxy debug DLL. WinAC ODK does not support debugging of RTDLLs directly.

- Add any debugging code to your application that you need such as MessageBox calls or ODK watchdog macro calls.
- Test your application.

### Building a Debug Version of an Extension Object

To debug an extension object, you must build it as either a debug DLL or a proxy debug DLL.

1. Select **Build > Set Active Configuration...** from the Visual C++ menu.
2. From the list of project configurations, select the one that corresponds to your project.

Project Configuration	Description
<project name> - Win32 Debug	Debug version of standard DLL for use with WinLC Basis
<project name> - Win32 Proxy Debug	Debug version of proxy DLL for use with WinLC RTX

3. Select **Build > Rebuild All** to recompile the project. This step produces a debug version for use with the Visual C++ debugger.

## Configuring and Downloading the STEP 7 Program


To configure the STEP 7 program to use your extension object, you must provide the name of your extension object and download your program to the controller. To do this, follow these steps:

1. From the SIMATIC Manager, edit the Data Block of your ODK STEP 7 program that contains the name of your DLL or RTDLL to be loaded. For example, for the HistoDLL program, this value is in address 0.0 of DB1001. Specify the name of the debug DLL or the proxy debug DLL in this location. Note that you cannot debug an RTDLL.

An example of this string is shown below:

```
'*dll:C:\Program Files\SIEMENS\WINAC\ODK\Projects\TestODK\Debug\MyApp.dll'
```



2. Click the desktop icon for your controller:  to start the control panel and the controller.
3. From the SIMATIC Manager, download the STEP 7 program to the controller.
4. Shut down the controller and control panel.

## Preventing the Control Panel from Automatically Starting the Controller

When you start the control panel in a Windows environment that does not have the VenturCom RTX extensions, the panel automatically starts the controller. When debugging an extension object in a Windows-only environment, you must modify the system registry so that the control panel does not start the controller. Your application running in debug mode will start the controller instead of the control panel.

**Note:** Modification of the registry is not necessary when debugging a proxy debug DLL for use with WinLC RTX.

To modify this registry setting, follow these steps:



1. Open the registry editor. (Select the **Start > Run** menu command and enter "regedit" for the program to open.)
2. Navigate to **HKEY\_LOCAL\_MACHINE > SOFTWARE > SIEMENS > WINLC > <\_BASIS**.
3. Double-click the "VirtualCPU Name" entry, and modify the "Value data" field so that it no longer specifies s7wlcvmx.exe.

### Note

Any change is valid, but you must change it back after your debug session. If you precede the name with an "x", for example, then you can just delete the "x" from this key name when you have completed your debug session.

## Testing Your Custom Application

To test your custom application with WinLC RTX, follow these steps:

1. Select **Project > Settings...** from the Visual C++ menu and then select the Debug tab.
2. Click  next to the "Executable for debug session:" field, and then click Browse. Browse to the directory where you installed your WinAC controller, for example \Program Files\Siemens\WinAC\WinLCRTX and double-click the executable file for your WinAC controller. For WinLC Basis, this filename is s7wlcvmx.exe. For WinLC RTX, this filename is s7wlcrtx.exe.
3. Start the control panel by clicking the desktop icon for your controller: . Because of the registry settings you changed, the control panel starts but does not connect to the controller.
4. Set any breakpoints that you need in your program and start the debugger. The debugger starts the controller and the control panel connects to it.
5. Set the controller to RUN mode or RUN-P mode.
6. Test your software by triggering events, watching variables, stopping at breakpoints and checking the behavior of your program.

### Note


Remember to change the registry setting for the VirtualCPUName back to its original value after your debug session.

## Using the ODK Watchdog Macros

Using breakpoints while debugging can cause your control program to exceed the scan cycle. You can use the WinAC controller tuning panel to monitor the effects of the extension object on the scan cycle. To avoid scan cycle overruns, you can call the macro `ODK_DISABLE_WATCHDOG` to disable the watchdog timer. When you disable the watchdog timer, your debugging activities will not cause the control program to exceed the scan cycle. To enable the watchdog timer, call `ODK_ENABLE_WATCHDOG` from your C/C++ program. The ODK watchdog macros only affect the watchdog timer in a debug build. If you build a release version of your extension object, the ODK watchdog macro calls have no effect.

## Ending a Debug Session for a WinLC RTX Proxy DLL

When you have finished debugging a Proxy DLL for WinLC RTX, follow these steps to shut the controller down:

1. If the control panel is not running, double-click the controller icon in the Windows taskbar: 
2. Select the **CPU > Shut Down Controller** menu command.

The panel shuts WinLC RTX down, which automatically terminates the debug session in Visual Studio.

### Notice

Do not stop a debug session for WinLC RTX from Visual Studio. If you do, WinLC RTX is left running and can not be shut down normally from the panel.

If you do stop the debug session from Visual Studio, you must follow these steps to shut WinLC RTX down:

1. Open a DOS command prompt window.
2. Enter "rtsskill" followed by a carriage return.
3. Note the index of s7wlcvmx.rtss.
4. Enter "rtsskill <index>" followed by a carriage return, where <index> is the index of the s7wlcvmx.rtss process.

WinLC RTX is now shut down. (These steps are only necessary for WinLC RTX; they are not required when debugging an extension object for WinLC Basis.)

## Changing the Extension Object

If you make changes to your custom software, you must replace the existing extension object with the DLL or RTDLL containing your changes, and then restart the controller. To make and use the new extension object, follow these steps:

1. Make changes to your C/C++ software as needed.
2. Build your extension object.
3. From the control panel, shut the WinAC controller down.
4. From the control panel, start the WinAC controller.

### Note

If your extension object has a new name, or is at a different location, you must edit the Data Block of your ODK STEP 7 program that contains the name of your DLL or RTDLL, and then download that block to the controller. For RTDLLs, a complete pathname is not necessary; the build command registers the RTDLL.



# ODK Support Software

## Data Access Helper Classes

The data access helper classes provide access to data in the WinAC controller. The two data access helper classes are:

- CWinLCReadData
- CWinLCReadWriteData

The functions in these classes allow you to read and write data of various data types. The read functions and write functions are separated into the two classes to provide very basic security on the input and output parameters of the Execute function.

The functions in the data access helper classes can help you avoid programming errors, such as out-of-range values or writing to invalid pointers. They also perform the necessary byte-swapping to convert data from the "big endian" format used in the S7 CPU architecture to the "little endian" format required for Microsoft Windows operating systems, including Windows 2000 and Windows XP.

In the ODK Execute function, the data input and output buffers are declared to be of classes CWinLCReadData and CWinLCReadWriteData respectively. They are not accessible outside of the Execute function. The data access helper classes are described in the ODK Object Web.

The ODK Object Web is accessible from the online help. You can access it from "Object Web" in the online help index, or from the Object Web topic in the table of contents.



### Caution

Your in-process function (thread) can corrupt controller memory if invalid addresses are used when writing data. Memory corruption can result in injury or property damage.

When developing your application, always follow proper programming guidelines and industrial standards. Always ensure that your application has been carefully tested before running the application with a WinAC controller or any other application.

## WinAC ODK Library for STEP 7

WinAC ODK provides a STEP 7 library ("WinAC ODK") that includes two SFBs:

- SFB65001 (CREA\_COM)
- SFB65002 (EXEC\_COM)

You insert these SFBs into your STEP 7 program to execute your application program as part of the controller scan cycle. To use these SFBs in your STEP 7 program, you must perform the following tasks:

- Load the WinAC ODK library into STEP 7.
- Insert both SFBs into your STEP 7 program.

### SFB65001 (CREA\_COM)

SFB65001 creates an instance of the extension object specified by the InstanceID parameter. Your STEP 7 program must supply the InstanceID parameter. The following table shows the interface for SFB65001 (CREA\_COM):

Address	Declaration	Name	Data Type	Comment
0.0	in	InstanceID	STRING[254]	ID of the extension object to be created  This ID includes a routing prefix that indicates whether the extension object is a DLL or an RTDLL (or a COM object created from a previous version of WinAC ODK), as well as the filename of the extension object.
256	out	Status	WORD	SFB return code: Error code or object instance handle

The InstanceID string contains a routing prefix that specifies the execution context for the extension object. The forms of the InstanceID are as follows:

Extension Object Type	Syntax of InstanceID
COM object	'<InstanceID>'
RTSS DLL	'*RTSS:<DLL name>'
Non-RTSS DLL	'*DLL:<DLL name>'

**Note**

WinAC ODK 4.0 does not generate COM objects, but if you have COM objects that you created with a previous version of WinAC ODK, you can direct SFB65001 to create it.



SFB65001 evaluates input conditions and performs the following actions:

1. If the extension object has not already been created, SFB65001 calls ODKCreate to create the extension object, using the InstanceID parameter to create a ClassID. (SFB65001 creates only one instance of this object.) SFB65001 adds this object instance to the internal list of created WinAC ODK objects.
2. If the extension object has already been created, SFB65001 maintains the WinAC ODK handle for the previously created object (an index to locate the object pointer).
3. If this is the first call to SFB65001 after leaving STOP mode or if the extension object was just created, SFB65001 invokes the Activate function.
4. SFB65001 sets the Status parameter to the WinAC ODK handle (or error code) and sets the BR bit. To see software for handling the return value, refer to the WinAC ODK sample program "HistoDLL".

### Error Codes for SFB65001

The following table lists the error codes for SFB65001:

Error Code	Message	Description
0	NO_ERRORS	Success
0x807F	ERROR_INTERNAL	An internal error occurred.
0x8001	E_EXCEPTION	An exception occurred.
0x8102	E_CLSID_FAILED	The call to CLSIDFromProgID failed.
0x8103	E_COINITIALIZE_FAILED	The call to CoInitializeEx failed.
0x8104	E_CREATE_INSTANCE_FAILED	The call to CoCreateInstance failed.
0x8105	E_LOAD_LIBRARY_FAILED	The library failed to load.
0x8106	E_NT_RESPONSE_TIMEOUT	A Windows response timeout occurred.
0x8107	E_INVALID_OB_STATE	Controller in an invalid state for scheduling OB
0x8108	E_INVALID_OB_SCHEDULE	Schedule information for OB is invalid
0x8109	E_INVALID_INSTANCEID	Instance ID for SFB65001 call is invalid.

Error Code	Message	Description
0x810A	E_START_ODKPROXY_FAILED	Controller could not load proxy DLL.
0x810B	E_CREATE_SHAREMEM_FAILED	WinLC RTX could not create or initialize shared memory area.
0x810C	E_OPTION_NOT_AVAILABLE	Attempt to access unavailable option

### SFB65002 (EXEC\_COM)

SFB65002 calls the Execute function of the extension object specified by the OBJHandle parameter. The following table shows the interface for SFB65002 (EXEC\_COM):

Address	Declaration	Name	Data Type	Description
0.0	in	OBJHandle	WORD	Handle returned from SFB65001 (CREA_COM)
2.0	in	Command	DWORD	Index of function or command to execute
6.0	in	InputData	ANY	Pointer to input function area
16.0	in	OutputData	ANY	Pointer to function output area
26.0	out	Status	WORD	SFB error code or return code from Execute.

SFB65002 performs the following actions:

1. SFB65002 verifies that SFB65001 (CREA\_COM) was called and that the object handle is valid.
2. SFB65002 processes the ANY pointers and returns error codes for invalid ANY pointer parameters.
3. SFB65002 invokes the customer WinAC ODK Execute function.
4. SFB65002 assigns the input and output ANY pointer areas to the WinLC Data Access Helper Classes.
5. SFB65002 sets the Status parameter with the Execute return code (unless there was a previous error) and returns to the STEP 7 program.

## Error Codes for SFB65002 (EXEC\_COM)

The following table lists the error codes for SFB65002:

Error Code	Message	Description
0	NO_ERRORS	Success
0x807F	ERROR_INTERNAL	An internal error occurred.
0x8001	E_EXCEPTION	An exception occurred.
0x8002	E_NO_VALID_INPUT	Input: the ANY pointer is invalid.
0x8003	E_INPUT_RANGE_INVALID	Input: the ANY pointer range is invalid.
0x8004	E_NO_VALID_OUTPUT	Output: the ANY pointer is invalid.
0x8005	E_OUTPUT_RANGE_INVALID	Output: the ANY pointer range is invalid.
0x8006	E_OUTPUT_OVERFLOW	More bytes were written into the output buffer by the extension object than were allocated.
0x8007	E_NOT_INITIALIZED	ODK system has not been initialized: no previous call to SFB65001 (CREA_COM).
0x8008	E_HANDLE_OUT_OF_RANGE	The supplied handle value does not correspond to a valid extension object.
0x8009	E_INPUT_OVERFLOW	More bytes were written into the input buffer by the extension object than were allocated.

All other error codes are user-defined in the individual applications.

## Auxiliary STEP 7 Interface Functions

WinAC ODK provides a set of auxiliary functions that are available to your extension object. These three functions are discussed below.

### ODK\_ReadState

ODK\_ReadState retrieves the current state (operating mode) of the controller, which is one of the following values:

CPU State	Description	Value (hexadecimal)
STOP_Update	Not used in WinAC controllers	01
STOP_Reset	STOP state during a memory reset	02
STOP_Init	STOP state for transition to STOP_Internal	03
STOP_Internal	STOP state: control program does not execute	04
START_OB102	STARTUP state for cold restart (OB102 executes on startup.)	05
START_OB100	STARTUP state for warm restart (OB100 executes on startup.)	06
START_OB101	Not used in WinAC controllers	07
RUN	RUN state: control program is executing	08
HALT	HALT or HOLD state: control program execution is suspended for testing, see STEP 7 online help for specific details	0A
DEFECTIVE	A fault has occurred	0D
POWER_Off	WinAC controller is shut down or in process of shutting down	0F

The ODK\_ReadState function page in the ODK Object Web contains the function header and parameter descriptions. The ODK Object Web is accessible from the online help. You can access it from “Object Web” in the online help index, or from the Object Web topic in the table of contents.

## ODK\_ScheduleOB

ODK\_ScheduleOB schedules an OB to be run by the controller. The OB is scheduled to run at a priority relative to other OBs as configured by the Hardware Configuration utility of STEP 7. (For example, if an OB80 is scheduled, it interrupts OB1, OB35, or any OB at a lower priority.)

When you schedule an OB, select one of the ODK-specific OBs, or one whose standard S7 behavior is closest to the way you plan to use the OB and is normally triggered by some asynchronous event, such as an error or a diagnostic event. When selecting an OB to schedule, consider the following OBs:

- OB52 - OB54 (ODK-specific interrupts)

**Note:** OB52 - OB54 do not correspond to any specific external event in the controller. The events for these OBs (16# 1x71, 16# 1x72, and 16# 1x73) are only generated when your ODK extension object schedules these OBs, and they are executed. The ODK-specific OBs provide an additional set of OBs that are available for application-specific needs.

- OB40 (Hardware interrupt)
- OB80 (Time error, such as a watchdog alarm)
- OB82 (Diagnostic Alarm interrupt)
- OB83 (Insert/Remove Module interrupt)
- OB84 (CPU Hardware Fault)

**Note**

Reserve OB84 for handling Windows operating system failures. Do not use it for other purposes.

- OB87 (Communication Error interrupt)

The ODK\_ScheduleOB function page in the ODK Object Web contains the function header and parameter descriptions. The ODK Object Web is accessible from the online help. You can access it from "Object Web" in the online help index, or from the Object Web topic in the table of contents.

To understand how the parameters of ODK\_ScheduleOB function relate to the local data of the specific OB to be scheduled, refer to the STEP 7 manual *System and Standard Functions for S7-300 and S7-400*. The arguments in the ScheduleOB function follow the same order as that in the documentation.

**Note**

The last 8 bytes of the local data contain the time stamp when the event was created. This data is entered when you call the ODK\_ScheduleOB function.

The documentation also describes data words for each OB. Depending on the type of OB, the documentation divides the data words (the last two parameters) in different ways. However, you can use these data words according to your own requirements. The WinAC controller does not interpret the data type or data parameters when scheduling the OB. The data words are copied to the local data (L memory) for the OB when the OB is scheduled to be executed. You can then access this information in your implementation of the OB that you scheduled.

**Note**

If you require that the entry in the Module Information/Diagnostic Buffer of STEP 7 be displayed with descriptive text, you must use the correct data types. These data types are typically listed as Reserved entries and are not documented in the S7 or STEP 7 documentation.

The tables below show valid data values and descriptions for the dataType2 and dataType1 parameters of the ODK\_ScheduleOB function:

**DataType2**

<b>Value (hexadecimal)</b>	<b>Description</b>
C1	32-bit double word
C4	Two 16-bit binary values
C8	32-bit signed value
C9	Two 16-bit signed values
CA	32-bit floating point value
CD	32 Relative time in milliseconds

**DataType1**

<b>Value (hexadecimal)</b>	<b>Description</b>
51	16-bit field: unspecified numeric value
58	16-bit field: time in milliseconds
59	16-bit integer value
5B	Two 8-bit binary value

**Error Codes from ODK\_ScheduleOB**

The table below lists the error codes that the ODK\_ScheduleOB function can return:

<b>Error Code</b>	<b>Message</b>	<b>Description</b>
0x8107	E_INVALID_OB_STATE	The controller is not in a valid state (RUN or RUN-P) to execute a scheduled OB.
0x8108	E_INVALID_OB_SCHEDULE	A scheduled OB is invalid.

## Example of Calling an Auxiliary Function

The auxiliary functions are available in the project framework and you can call them directly from your code. The following example (taken from the HistoDLL sample project) shows a function that calls ODK\_ScheduleOB in a real-time application:

```
// ScheduleOB52
void ScheduleOB52(unsigned short mode, unsigned short deviation)
{
    if (g_serviceHandle)
    {
        ODK_ScheduleOB(g_serviceHandle,
            0x35, // execute asynchronous error OB
            0x01, // use cycle time error event number
            0xFE, // fill in configured sequence layer
            52, // execute OB52
            0xC4, // dataType2 (for long word) - two 16 bit words
            0x58, // dataType1 (for short word) - time in milliseconds
            deviation, // data1
            mode);
    }
}
```

The auxiliary STEP 7 functions are included in the ODK Object Web, and the sample code shown above is also included in the HistoDLL Object Web. The ODK Object Web and the HistoDLL Object Web are accessible from the online help. You can access them from “Object Web” in the online help index, or from the Object Web topic in the table of contents.





# Index

## A

Activate function, 27, 35, 42  
Add/Remove Programs, 8  
Adding asynchronous events, 21  
Additional Assistance, iii  
Additional sample programs, 16  
Application wizard, 1, 17, 18, 19, 27, 32  
Asynchronous event processing, 1, 15, 18, 30  
Asynchronous monitor classes, 27  
Asynchronous monitoring, 18, 23, 32  
Asynchronous processor classes, 27  
Authorization, 8  
Auxiliary STEP 7 Interface functions, 46

## B

Background priority, 23  
Breakpoints, 36  
Building  
    debug version, 33, 36  
    extension object (DLL, RTDLL), 33, 39  
    HistoDLL extension object, 10  
Building, 33, 36

## C

C/C++ software, 1, 17, 27  
Calling  
    non-deterministic functions, 30  
    WinAC ODK SFBs, 35  
Calls outside of extension object, 32  
CAsyncProc, 27, 30  
CEventMsg, 27, 30  
Changing extension objects, 39  
Classes, 27, 30  
CoCreateInstance, 42  
COINIT\_MULTITHREADED, 42  
CoInitializeEx, 42  
Commands, 19

Communication error interrupts, 46  
Compiler selection, 18  
Compiling extension object, 33  
Compiling the HistoDLL extension object, 10  
Complete restart OB, 12  
Computer requirements, 7  
Configuring  
    asynchronous monitoring, 23  
    asynchronous processing, 21  
    HistoDLL STEP 7 program, 10  
    monitor threads, 23  
    project information, 18  
Configuring, 10  
Contents of WinAC ODK, iii  
CQueue, 27, 30  
CREA\_COM, 34, 35, 42  
Creating  
    asynchronous events, 18, 21, 30  
    C/C++ project, 18, 27  
    monitor threads, 18, 23, 32  
    ODK application, 17  
    STEP 7 program, 35  
    subcommands, 19  
Creating, 30  
CThreadBase, 27, 32  
Customer feedback, 57  
Customer support, iii  
CWinLCReadData, 27, 41  
CWinLCReadWriteData, 27, 41  
Cycle, 1, 12, 27, 30, 35, 36, 42

## D

Data access helper classes, 27, 41  
Data block changes, 39  
DataType1, 46  
DataType2, 46

DeActivate function, 27, 32  
Deallocating asynchronous events, 30  
Debugging extension objects, 36, 39  
Deleting asynchronous events, 21  
Delta Tau sample program, iii  
Developing  
    ODK application, 17  
    STEP 7 program, 35  
Disabling watchdog timer, 36  
Disk space, 7  
DLL, 1, 17, 33, 36  
DLL name in STEP 7 program, 39  
Downloading HistoDLL STEP 7 program, 10

## E

E-mail (Customer Support), iii  
Enabling  
    asynchronous monitoring, 18, 23  
    asynchronous processing, 18, 21  
    watchdog timer, 36  
Entering subcommands, 19  
Environments other than Microsoft Visual C++,  
    34  
Error Codes  
    ODK\_ScheduleOB, 46  
    SFB65001, 42  
    SFB65002, 42  
Error Codes, 42  
Events, 30, 32  
Example programs, 9, 10, 12, 15, 16  
Exception event processing, 15  
EXEC\_COM, 19, 34, 35, 42  
Executable for debug session, 36  
Execute function, 15, 19, 27, 30, 32, 35  
ExecuteInit function, 15  
ExecuteUpdate function, 15  
Executing HistoDLL STEP 7 program, 10  
Execution threads, 1, 30, 32

Extension object  
    building, 33  
    debugging, 36  
    developing, 17  
    name in STEP 7, 36, 39  
    structure, 27  
Extension object, 1, 17

## F

File reading and writing, 16  
FileIO, 16  
Foreground priority, 23  
Free cycle OB, 12  
Functions  
    Activate, 27  
    DeActivate, 27, 32  
    GetResult, 30  
    GetStatus, 30  
    ODK\_ReadState, 46  
    ODK\_ScheduleOB, 15, 46  
    ODKCreate, 27  
    ODKRelease, 27, 32  
    PauseThread, 32  
    ResumeThread, 32  
    StopThread, 32

## G

GetResult function, 30  
GetStatus function, 30  
Getting Started, 9

## H

Hardware  
    using external, 16  
Hardware interrupts  
    faults, 46  
    interrupts, 46  
Hardware requirements, 7  
Histo.cpp, 15  
HistoDLL C++ program, 9, 12, 15

HistoDLL STEP 7 program, 10, 12

Histogram, 9, 10, 12, 15

Hotline, iii

## I

Implementing

DLLs and RTDLLs, 27

ODK Application, 17

STEP 7 program, 35

InitHistogram function, 12

Initializing histogram, 15

In-process applications, 1

Insert/Remove module interrupts, 46

Inserting

SFBs, 35

Inserting, 35

Installation

System requirements, 7

Installing WinAC ODK, 8

InstanceID, 36, 42

Internet (Customer Support), iii

Interrupts, 46

Introduction, 1

Invalid addresses, 41

IWinLCLogicExtension, 27, 30

IWinLCServices, 46

## L

Latency, 16

LoadHistogram function, 12

Loading the WinAC ODK STEP 7 Library, 9, 17, 34

## M

m\_ExitThread variable, 32

Measuring latency, 16

Memory corruption, 41

Mode transition, 27

Module interrupts, 46

Monitor classes, 27, 32

Monitor threads, 16, 23, 32

## N

Name changes (DLL, RTDLL), 39

Name of extension object, 36

NETFileIO, 16

NETHistoDLL, 16

Non-deterministic functions, 30

Non-Microsoft Visual C++ environments, 34

## O

OB, 46

OB1, 12, 35

OB100, 12, 35

OB4x, 46

OB80, 12, 46

OB82, 46

OB83, 46

OB84, 46

OB87, 46

Object Web, 15

OBs, scheduling, 16, 46

ODK Application, 17

ODK contents, iii

ODK Installation, 8

ODK SFBs, 35

ODK support classes, 27

ODK System Function Blocks, 1, 12, 17

ODK\_ReadState, 46

ODK\_ScheduleOB, 15, 46

ODKCreate function, 27, 35

OdkLib, 34

ODKRelease function, 27, 32

Opening HistoDLL C/C++ project, 10

Operating mode transition, 27

Operating system requirements, 7

Overview, 1

## P

PauseThread function, 32  
Priority of Threads, 23  
Product Overview, 1  
Program cycle, 1, 12, 27, 30, 35, 36, 42  
Program listings  
    STEP 7 HistoDLL, where to find, 12  
Programming  
    Asynchronous events, 30  
    C/C++ extension software, 27  
    Monitor threads, 32  
    STEP 7 program, 35  
    subcommands, 19  
Programming, 27  
Programming environments, 34  
Project Information, 18, 25  
Project option summary, 18  
Proxy DLL, 10, 33

## Q

Queue Class, 27, 30

## R

RAM, 7  
Reading controller data, 41  
Reading from files, 16  
ReadState, 46  
Recovering  
    from a blue screen event, 36  
    from a WinLC RTX crash, 36  
    from an infinite loop, 36  
Recovering, 36  
Registering RTDLL requirements, 10, 34  
Registry settings, 36  
Reinstalling WinAC ODK, 8  
Removing Installation, 8  
Renaming an asynchronous event, 21  
Replacing extension objects, 39  
Requirements, 7

Response form, 57  
Restrictions, 1  
Retrieving HistoDLL program, 10  
Retrieving WinAC ODK library, 34  
RTDLL  
    debugging, 36  
    sample program, 15  
RTDLL, 1, 15, 17, 34  
RTDLL name in STEP 7 program, 39  
rtsskill, 36  
rtssrun command, 34  
Running HistoDLL STEP 7 program, 10  
Running in Debug, 36

## S

s7wlcvmx.exe, 36  
s7wlcvmx.rtss, 36  
Safety guidelines, 2  
Sample programs, 9, 10, 12, 15, 16  
Scan cycle, 1, 12, 27, 30, 35, 36, 42  
Scan deviation OB, 12  
ScheduleEvent function, 30  
ScheduleOB, 46  
ScheduleOB52, 15  
Scheduling OBs, 15, 16, 46  
Set Active Configuration, 33, 36  
SetDelTime function, 30  
SFB65001  
    Error Codes, 42  
SFB65001, 1, 12, 34, 35, 42  
SFB65002  
    Error Codes, 42  
SFB65002, 1, 12, 15, 19, 27, 34, 35, 42  
SFBs, 1, 42  
SIMATIC Manager, 9, 12, 34, 35  
Software requirements, 7

**STEP 7**

- interface functions, 46
- library, 1, 34, 42
- name of extension object, 36
- program, 1, 9, 12, 17, 35
- program cycle, 27, 30, 36, 42

StopThread function, 32

Subcommands, 19, 27

Summary of project options, 18

Support classes, 27

Support, customer, iii

System Function Blocks, 42

System Requirements, 7

**T**

Task summary, 17

Technical support, iii

Telephone (Customer Support), iii

Testing your custom application, 36

Thread priorities, 23

Threads, 1, 16, 23, 27, 30, 32

Time errors, 46

TonePulse, 16

Transition, operating mode, 27

**U**

Uninstalling WinAC ODK, 8

UpdateHistogram function, 12

Updates (Customer Support), iii

Updating histogram, 15

**Using**

- application wizard, 18
- external hardware, 16
- files, 16
- HistoDLL sample program, 10
- monitor threads, 16
- ODK manual, iii

**V**

Variable table, 10

Version requirements, 7

VirtualCPU Name, 36

Visual C++ Debugger, 36

**W**

Warm restart, 12

watchdog alarm, 46

Watchdog macros, 36

WinAC controller scan cycle, iii, 1, 12, 27, 30, 35, 42

WinAC ODK

- contents, iii

- installing, 8

- overview, 1

- removing, 8

- SFBs, 35

- STEP 7 library, 34, 35, 42

Windows operating system failures, 46

Writing controller data, 41

Writing to files, 16





Please check any industry that applies to you:

- Automotive
- Chemical
- Electrical Machinery
- Food
- Instrument and Control
- Non-electrical Machinery
- Petrochemical
- Pharmaceutical
- Plastic
- Pulp and Paper
- Textiles
- Transportation
- Other \_\_\_\_\_

**Mail your response to:**

Siemens Energy & Automation, Inc.  
ATTN: Technical Communications  
One Internet Plaza  
Johnson City TN USA 37604

Include this information:

**From**

Name: \_\_\_\_\_  
Job Title: \_\_\_\_\_  
Company Name \_\_\_\_\_  
Street: \_\_\_\_\_  
City and State: \_\_\_\_\_  
Country: \_\_\_\_\_  
Telephone: \_\_\_\_\_