

# **SIEMENS**

**SIMATIC**

**WinAC RTX  
Open Development Kit (ODK)**

**Version: 3.0**

## Safety Guidelines

This manual contains notices which 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:



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



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



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

### Caution

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

### Notice

NOTICE used without the safety alert symbol indicates a potential situation which, 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:



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 which have been approved or recommended by Siemens.

This product can only function correctly and safely if it is transported, stored, set up, 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 95<sup>®</sup>, Windows 98<sup>®</sup>, Windows 2000<sup>®</sup>, Windows ME<sup>®</sup> and Windows NT<sup>®</sup> are registered trademarks of Microsoft Corporation.

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

### Copyright Siemens Energy & Automation, Inc. 2000 Disclaimer of Liability

#### 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.

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

Siemens Energy & Automation, ISBU  
3000 Bill Garland Road  
Johnson City, TN 37605-1255, USA

© Siemens Energy & Automation, Inc. 2001

Technical data subject to change.

## **Preface**

The Windows Automation Center Real-time Extension Open Development Kit (WinAC RTX ODK) provides you with tools for creating custom software that interacts directly with the Windows Logic Controller (WinLC RTX) program scan cycle. With WinAC RTX ODK, you can produce software for use with the VenturCom Real-time Extensions (RTX™) for WinLC in a real-time dynamic link library (RTDLL). This in-process RTDLL is loaded within the WinLC RTX process.

Optionally, you can also produce custom software for WinLC RTX running under Windows NT. This interaction uses Microsoft's COM (Component Object Model) technology. Your custom software is a COM object that is loaded as a DLL within the WinLC RTX process.

WinAC RTX 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 RTX 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++. Knowledge of STEP 7 programming and of WinLC is also required.

## **Scope of the Manual**

This manual describes the operation and features of version 3.0 of the WinAC RTX Open Development Kit (ODK).

## How to Use This Manual

This manual provides the following information:

- Overview and installation of WinAC RTX ODK
- Getting started with a sample application program
- Programming an extension object (RTDLL or COM object) using the WinAC RTX 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 that you use to produce custom software

## Other Manuals

For additional information, refer to the following manuals:

Title	Content
Windows Logic Controller RTX (WinLC RTX) User Manual	This manual provides basic information about the WinLC controller with the real-time extensions.

## Additional Assistance

If you have any questions not answered in this or one of the other STEP 7 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 about and updates to this user manual at the Siemens Energy & Automation web site:

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

This web site includes useful information, such as application notes, in the Technical Service area. This area also includes downloadable software, including sample programs for linking a Delta Tau motion control board with WinLC.

<b>Customer Support</b>	
<b>North America</b>	
Telephone	(609) 734-6500 (609) 734-3530
E-mail	ISBU.Hotline@sea.siemens.com simatic.hotline@sea.siemens.com
Internet	<a href="http://www.sea.siemens.com/IndustrialSoftware/welcome.asp">http://www.sea.siemens.com/IndustrialSoftware/welcome.asp</a> <a href="http://www.ad.siemens.de/meta/index_76.htm">http://www.ad.siemens.de/meta/index_76.htm</a>
<b>Europe</b>	
Telephone	++49 (0) 911 895 7000
E-Mail	simatic.support@nbgm.siemens.de
Internet	<a href="http://www.ad.siemens.de/simatic-cs">http://www.ad.siemens.de/simatic-cs</a>
Fax	++49 (0) 911 895 7001

# Contents

<b>Product Overview</b> .....	<b>1</b>
WinAC RTX ODK Overview .....	1
System Requirements .....	6
Installing WinAC RTX ODK .....	7
<b>Getting Started</b> .....	<b>9</b>
Sample Program Introduction .....	9
Overview of the HistoDLL Sample STEP 7 Program.....	11
Overview of the HistoDLL RTDLL .....	14
Additional Sample Programs .....	15
<b>Basic Tasks</b> .....	<b>16</b>
Implementing an ODK Application.....	16
Using the Application Wizard.....	17
Programming the C/C++ Extension Software .....	27
Programming Asynchronous Events .....	30
Programming Monitor Threads.....	32
Building the Extension Object.....	33
Loading the WinAC RTX ODK Library Into STEP 7 .....	35
Programming the STEP 7 Program .....	36
Debugging the Extension Object .....	37
<b>ODK Support Software</b> .....	<b>40</b>
Data Access Helper Classes .....	40
WinAC RTX ODK Library for STEP 7 .....	41
Auxiliary STEP 7 Interface Functions .....	45
Object Web .....	49

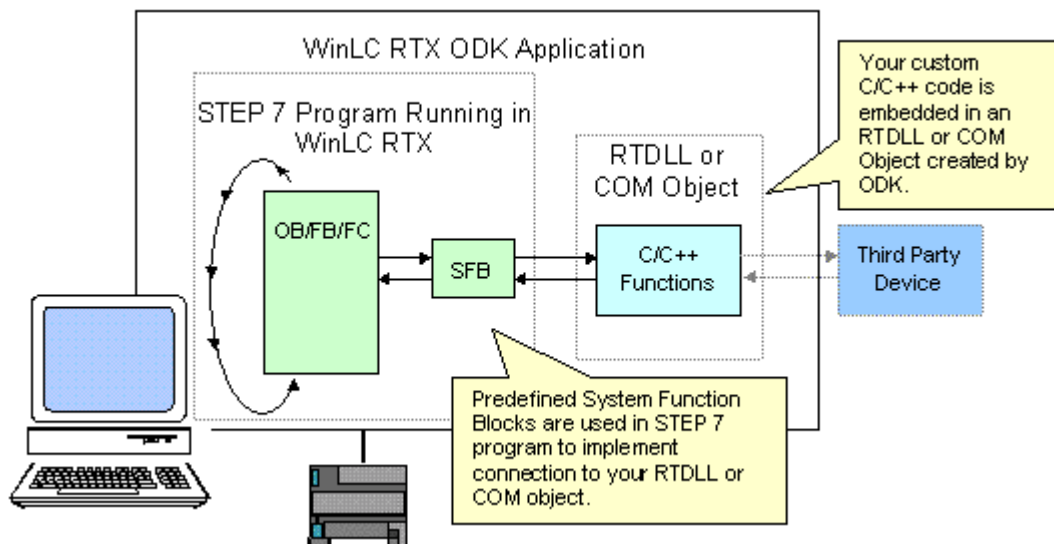
## Product Overview

### WinAC RTX ODK Overview

The Windows Automation Center Real-time Extension Open Development Kit (WinAC RTX ODK) is an open interface to the Windows Logic Controller (WinLC RTX). It provides a set of tools that enables you to implement custom software that is executed as part of the STEP 7 program execution. ODK implements this custom interface as an extension object, which is an RTDLL for use with WinLC RTX, or as a COM object installed as a DLL for use with WinLC RTX using Windows NT. WinAC RTX ODK provides an application wizard for you to develop this custom software in C/C++. It does not support Visual Basic or Java.

Two special STEP 7 System Function Blocks (SFBs) provide the interface between your custom C/C++ software and the program logic that is being executed by WinLC RTX. These two SFBs enable you to create an object containing your custom code and to execute that code.

Your custom software is invoked by an SFB, and executed as part of the WinLC RTX scan cycle. WinAC RTX 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 WinLC RTX scan cycle, it must be executed on a separate thread of execution. A separate thread is required if the execution time of your custom software could cause a scan to exceed the configured watchdog time. WinAC RTX 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 WinLC RTX.



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 WinLC RTX, 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 WinLC RTX

WinAC RTX ODK expands the capabilities of WinLC for a wide range of possibilities. ODK offers benefits, for example, in the following situations:

- The S7 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 for motion control or vision systems.
- The application needs to access features of the computer that is running the VenturCom Real-time Extensions or the Windows NT operating system that are not accessible by standard S7 control languages.

## Tools Provided by WinAC RTX ODK

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

- Application wizard for generating a program shell with the correct interfaces and functions for interacting with WinLC RTX
- 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 COM object). 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 be stored in the WinLC RTX memory to be used by SFB65002.
  - SFB65002 (EXEC\_COM) executes a specific function in the RTDLL or COM object 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

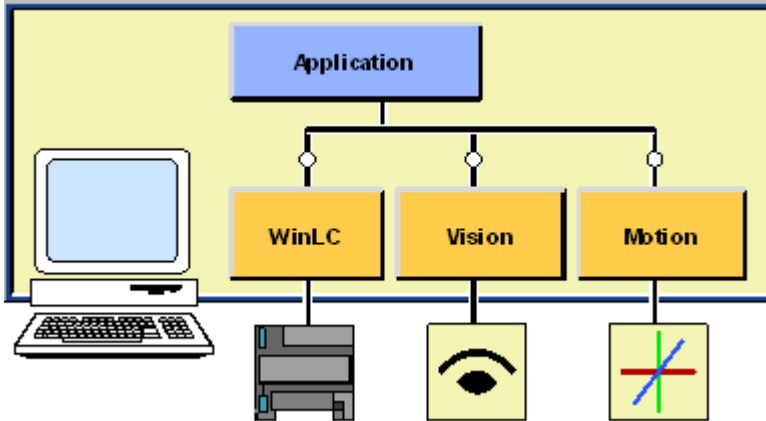
With WinAC RTX ODK, you can define and implement one or more standard FBs or FCs that become an easy-to-use block library that you can use throughout the rest of your STEP 7 program. The blocks in this user-defined library, in turn, access the WinAC RTX ODK SFBs. An example program that is installed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

## WinAC RTX ODK Provides a Mechanism for Defining Custom Logic

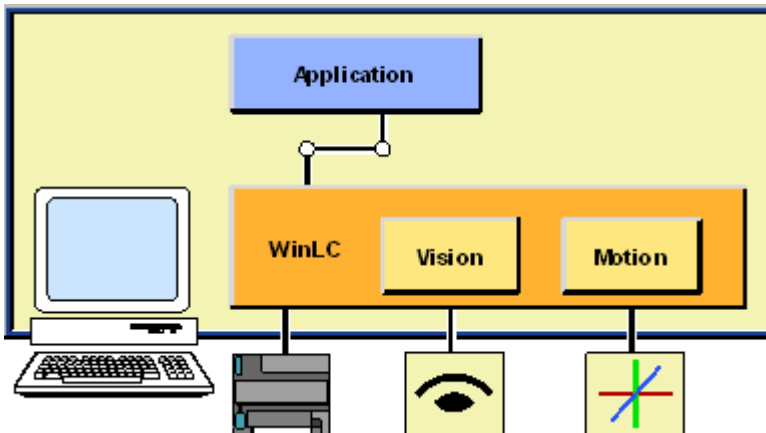
WinAC RTX ODK provides the mechanism for you to define one or more custom system functions that can be integrated into the STEP 7 program logic. The Open Development Kit connects the SFBs in the program logic to your custom logic extension.



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 WinLC RTX by manipulating the I/O controlled by WinLC RTX through asynchronous read/write functions provided by STEP 7 or by the Data control of WinAC Computing:



By using WinAC RTX ODK, this custom application can now interact directly with the WinLC RTX program logic:



The WinAC RTX ODK software allows you to build your extension object as either an RTDLL or a COM object. An RTDLL is in-process; it runs in the same process as WinLC RTX and is loaded in the same address space. A COM object is in-process under Windows NT, but its data must be copied between RTX and Windows NT through a shared memory buffer. For this reason, an RTDLL allows for faster processing between the application and WinLC RTX 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 COM Object 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.)

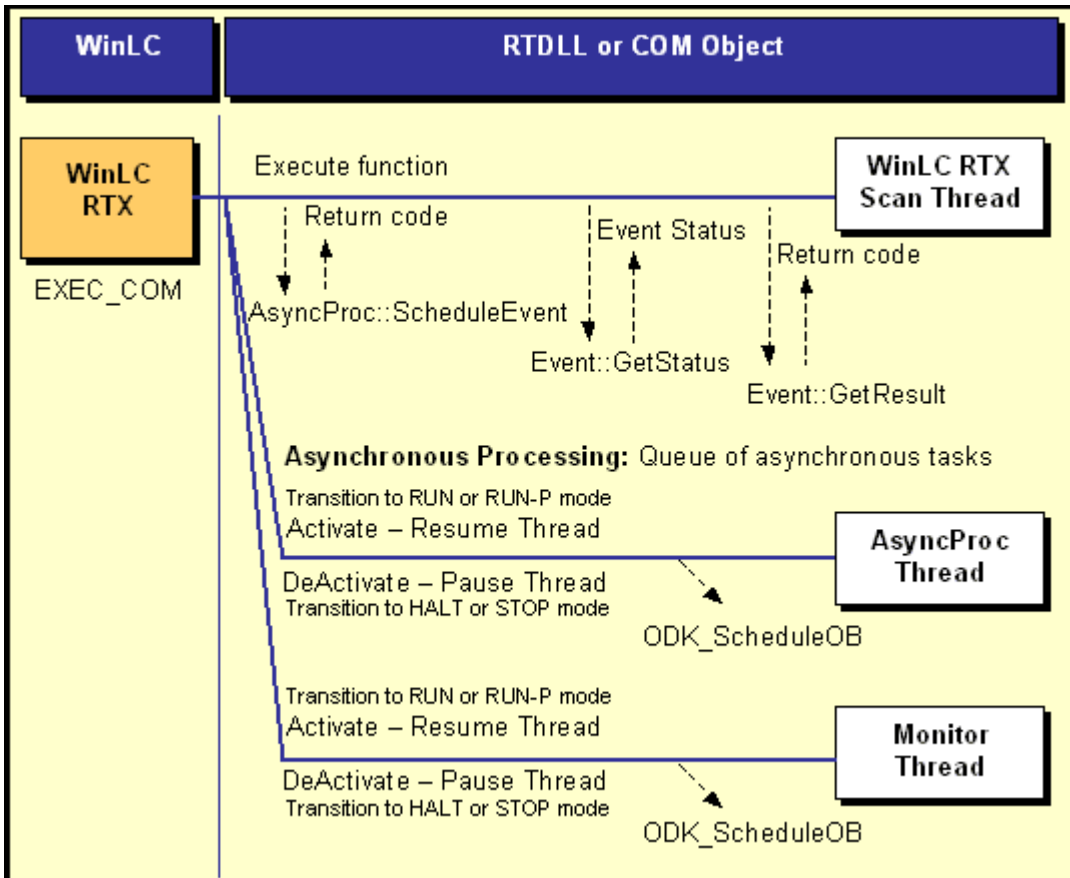
**Note**

Because WinLC RTX can be started as a service, 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. 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 RTX 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.

**Note**

Because the software in the extension object executes as part of the WinLC RTX 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 WinLC RTX scan.



WinAC RTX 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 WinLC RTX makes the transition from STOP to STARTUP or from HALT to RUN or RUN-P mode
- Notify (or deactivate) your extension object when WinLC RTX goes to STOP or HALT mode
- Notify WinLC RTX of an event. You can create an event to cause one of the following OBs to be executed:
  - OB4x (Hardware interrupts)
  - 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 NT operating system failures. Do not use it for other purposes.

- Read the current operating state of WinLC RTX

## System Requirements

### Hardware Requirements

To run WinLC RTX and WinAC RTX ODK, it is recommended that your computer meet the following criteria:

- A personal computer (PC) with the following:
  - Pentium processor running at 400 MHz or faster
  - 128 Mbytes RAM
  - 512 Kbytes level 2 cache
  - 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 VGA color monitor, keyboard, and mouse or other pointing device (optional) which are supported by Microsoft Windows NT

### Software Requirements

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

- Microsoft Windows NT version 4.0 (or higher), with service pack 6 (or higher)
- WinLC RTX version 3.0 (or higher)
- STEP 7 version 5.0, service pack 3 (or higher), with STEP 7 version 5.1 (or higher) recommended
- A C/C++ compiler (Microsoft Visual Developers Studio version 6 (Visual C++), service pack 3 (or higher) is recommended.)
- VenturCom Software Development Kit (SDK), required for compiling RTSS projects.
- Internet Explorer 4.0 (or higher), for viewing product documentation

WinAC RTX and WinAC RTX ODK will not run under Microsoft Windows 3.11, Windows 95, Windows 98, Windows 2000, Windows ME, or Windows for Workgroups.

## Installing WinAC RTX ODK

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

### Notice

Do not install any component of WinAC (such as WinAC RTX ODK) on a computer while any other component of WinAC (such as WinLC RTX, the Computing SoftContainer, programs that use the SIMATIC controls provided by Computing, or the panel for the CPU 416-2 DP ISA) is being executed (is currently running) on that computer.

Because Computing, WinLC RTX, 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 being executed by the computer can corrupt the software files. Always ensure that the following programs are not running when you install :

- WinLC RTX
- Panel for CPU 416-2 DP ISA
- Computing SoftContainer
- Computing TagFile configurator
- SIMATIC Tool Manager
- Computing Configuration
- Any program (such as a program created in Visual Basic) that uses one of the SIMATIC controls provided by Computing

The Setup program for WinAC RTX 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. Use the following procedure to start the Setup program:

1. Start the dialog box for installing software under Windows NT by double-clicking on the Add/Remove Programs icon in the Control Panel.
2. Click the Install... button.
3. Insert the CD and click the Next button. Windows NT searches automatically for the installation program SETUP.EXE.
4. 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)

Use the Windows NT Add/Remove Programs procedure to remove the WinAC RTX ODK software:

1. Select the **Start > Settings > Control Panel** menu command to display the Windows NT 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 3.0 and click the Add/Remove button.
4. Follow the instructions of the dialogs to remove the software.

# Getting Started

## Introduction

WinAC RTX ODK installs several sample programs on your computer during the installation procedure. The sample program HistoDLL contains the STEP 7 project and RTDLL (C/C++ software) for collecting data about the scan time of WinLC RTX. The data collected by this custom application can be displayed 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, which is generated as a real-time project.

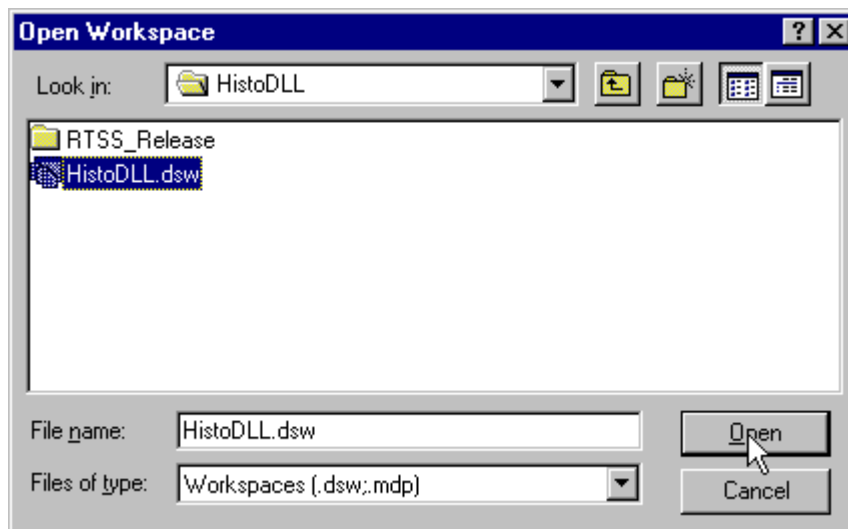
You can work with this sample program to gain an understanding about how to create your own custom RTDLL that interacts with WinLC RTX.

## How to Use the HistoDLL Sample Program

In order to work with the sample program, you must compile the HistoDLL C/C++ project into an RTDLL and download it to the real-time environment. You must also retrieve and download the HistoDLL STEP 7 program to WinLC RTX, and run it. Then you can observe its operation through the sample Histogram Status variable table.

To compile the sample RTDLL 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 for compiling the RTDLL).
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. Select **Build > Set Active Configuration...** to set the project configuration.
5. Select **HistoDLL - Win32 RTSS Release** from the list and click OK to ensure that the project is configured for real-time.
6. Select **Build > Build HistoDLL.rtdll** to build the project.

In Microsoft Visual C++, the Build command for an RTDLL also loads the RTDLL. If you are working in an environment other than Microsoft Visual C++, you must use the rtssrun command to register the RTDLL after you build it. To use rtssrun to register the RTDLL after it is built, follow these steps:

1. Select **Start > Programs > Command Prompt** to open a DOS command prompt window.
2. Type `cd Siemens\WinAC\Odk\Examples\HistoDLL\RTSS_Release` to change to the folder containing the RTDLL that you just built.
3. At the command prompt, type `rtssrun /dll HistoDLL.rtdll` to register the RTDLL.

To retrieve, download and run the HistoDLL STEP 7 program, 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. If WinLC RTX is not already running, double-click the WinLCRTX controller icon on your desktop to start it.
7. Set the operating mode of WinLC RTX to STOP, and perform a memory reset (MRES).
8. From the SIMATIC Manager, click the Blocks folder of the HistoDLL S7 Program and select **PLC > Download** to download the HistoDLL program to WinLC RTX.
9. Change the operating mode of WinLC RTX from STOP mode to RUN or RUN-P mode. WinLC RTX begins executing the HistoDLL program.
10. 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 RTDLL 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 RTDLL functions. They provide the interface between the STEP 7 program and your custom C++ library.

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 utilizes three Organization Blocks (OBs):

- OB1: Free Cycle OB - This OB cycles continually.
- OB80: Scan Deviation - This OB is executed only when a time error occurs, for example, the maximum cycle time is exceeded.
- OB100: Complete Restart - This OB is executed on a Warm Restart (transition from STOP to RUN or RUN-P).

The HistoDLL STEP 7 program contains three Functions (FCs):

- FC1000: LoadHistogram - This FC calls SFB65001 (CREA\_COM) to create the RTDLL object.
- FC1001: InitHistogram - This FC initializes the histogram slots to 0 and calls SFB65002 (EXEC\_COM) to execute the RTDLL Execute function.
- FC1002: UpdateHistogram - This FC updates the histogram on every scan cycle to load the clock time into the histogram input data.

### STEP 7 Program Execution

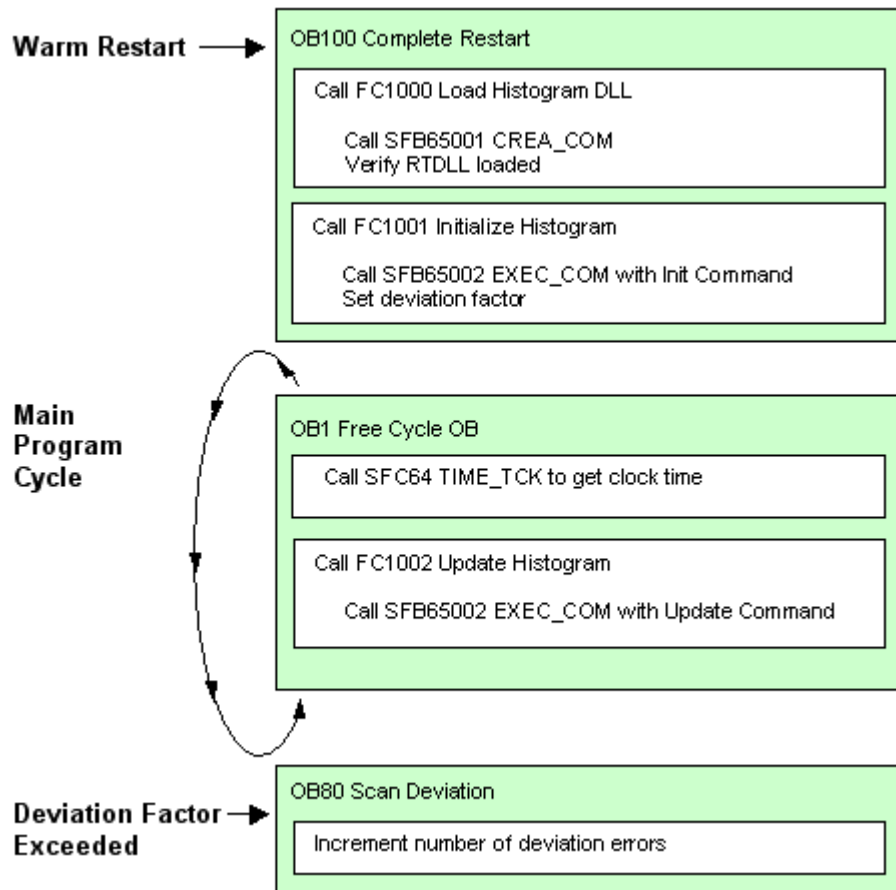
When the operating mode of WinLC RTX changes from STOP to RUN, WinLC RTX executes OB100 (warm restart). OB100 performs these tasks:

1. OB100 calls FC1000 (LoadHistogram).
  - a. FC1000 calls SFB65001 (CREA\_COM) to create the RTDLL object for the histogram program.
  - b. FC1000 verifies that the HistoDLL RTDLL was loaded.
2. OB100 calls FC1001 (InitHistogram).
  - a. FC1001 calls SFB65002 (EXEC\_COM) with a command parameter of 0 to initialize the histogram program. This command sets all initial values to 0.
  - b. FC1001 sets the deviation factor for the scan cycle in DB1000.

After OB100 finishes, WinLC RTX executes OB1 (main program cycle).

1. OB1 calls SFC64 (TIME\_TCK) and writes the current clock time to the input data.
2. OB1 calls FC1002 (UpdateHistogram).
  - a. FC1002 calls SFB65002 (EXEC\_COM) with the command UpdateCmd to update the histogram data in the RTDLL . If the C/C++ function that processes the update command detects that the scan cycle exceeded the deviation factor, it schedules OB80, the scan deviation OB.

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 RTDLL

The RTDLL HistoDLL contains C++ functions that read the data about the WinLC RTX scan time and create a histogram.

The sample C++ program includes the file Histo.cpp for the custom histogram software as well as the files necessary for the interface (WinLCReadData.h, WinLCReadData.cpp, WinLCReadWriteData.h, and WinLCReadWriteData.cpp).

The file Histo.cpp shows how to implement the histogram functionality. The application wizard (WinAC ODK AppWizard) generated the framework for this project.

Histo.cpp produces a single ODK RTDLL which can execute two functions:

- Initialize the histogram (ExecuteInit subcommand)
- Update the histogram (ExecuteUpdate subcommand)

When the STEP 7 program calls SFB65002 (EXEC\_COM), the Execute function in the RTDLL is called. If the STEP 7 program passes an Init Command (0) to the command parameter of the Execute function, the Execute function calls the ExecuteInit function. If the STEP 7 program passes an Update Command (1) to the command parameter of the Execute function, the Execute function calls the ExecuteUpdate function. The STEP 7 program calls the Init Command on a Warm Restart and it calls the Update Command during the main program cycle. Refer to the HistoDLL Sample STEP 7 Program for a description of the STEP 7 program execution.

Histo.cpp also shows how to schedule an asynchronous error OB in WinLC RTX. The function is ScheduleOB80, and is included as an example of how to schedule an OB on an exception event. 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. The scheduling of the error OB is included as an example of how to schedule an asynchronous OB in an ODK application.

### HistoDLL Object Web

Histo.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. Click the link below to examine the classes and functions that comprise the HistoDLL C/C++ sample project.

[HistoDLL Object Web](#)

## Additional Sample Programs

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

C/C++ Sample Program	STEP 7 Sample Program	Description
Histogram	Histogram	Contains the Histogram program for a Windows NT application
Demo_RTX	DemoRTX	Contains combined code from all other sample programs
Demo_RTX2	DemoRTX2	Measures latency between the scheduling of an OB and its execution
Demo_RTX3	DemoRTX3	Interfaces with external hardware to generate a tone pulse
Demo_RTX4	DemoRTX4	Performs arithmetic calculations on controller data
Demo_RTX5	DemoRTX5	Monitors a counter, creates events that schedule OBs, and performs file I/O

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

- Using a monitor thread
- Scheduling OBs
- Managing a counter
- Managing a timer
- Measuring latency in scheduled OB execution
- Performing simple mathematical operations on controller values
- Interfacing with external hardware
- Reading from and writing to files

**Note**

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

## Basic Tasks

### Implementing an ODK Application

Your ODK application combines software that you implement in C/C++ with a STEP 7 program that can interface with external hardware or software. You must develop both a C/C++ extension object (RTDLL or COM object), and a STEP 7 program. The STEP 7 program uses the ODK SFBs to call functions in your extension object during its normal execution. For a real-time (RTX) application, the extension object is an RTDLL. For a Windows NT application, the extension object is a COM object that is loaded as a DLL.

#### Creating the C/C++ Project

To create the C/C++ software that is compiled and loaded as part of the in the WinLC RTX process, follow these steps:

- Use the application wizard to create the program shell for the RTDLL or COM object.
- 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 RTX ODK.
2. Insert the ODK SFBs into your STEP 7 program.

When you have developed both the extension object and the STEP 7 program, you can run the STEP 7 program on WinLC RTX 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 RTX ODK interface in an RTDLL or COM object
- 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 monitor one or more attributes of your system (optional)

These tasks are described below. Following completion of these tasks, the application wizard provides a summary of the options that you selected for the type of COM object or RTDLL that you configured. After you confirm these choices, the application wizard generates the C/C++ project shell.

### Configuring Project Information

To start the WinAC RTX 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. Check Real-time for the Target Platform to create an RTDLL for use with the VenturCom Real-time Extensions, or check Windows NT to create a COM object for use with Windows NT.
4. 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 real-time project other than Microsoft Visual C++ 6.0.

#### Note

The application wizard generates a project that uses ATL to implement the COM interface. The generated project and source files are ready to use with the Microsoft Visual C++ tool chain. If you select a compiler other than Microsoft Visual C++ 6.0 that does not support ATL, then you must modify the generated source files to use a COM implementation that your tool chain supports.

5. Enter a short name for your C++ program in the Short Name field if your project is for Windows NT. All of the remaining fields for the Windows NT COM object are created automatically based on the Short Name. You can also modify these fields individually without changing any of the other fields. (These fields are not used for real-time projects.)

**Notes**

The ProgID field is required as a parameter for SFB65001 (CREA\_COM) in a Windows NT project.

You cannot change the interface name (IWinLCLogicExtension). This interface is required by SFB65002 (EXEC\_COM) and must be present in all WinAC RTX ODK COM objects.

6. Click Next after you have entered the correct information for all of the fields.

The screenshot shows the 'Project Information' dialog box. The 'Project Info' section contains a 'Name' field with 'MyProject' and a 'Directory' field with 'C:\Siemens\WinAC\ODK\Projects\MyProject'. The 'Target Platform' section has radio buttons for 'Real-time' and 'Windows NT'. The 'Compiler' section has a dropdown menu set to 'Microsoft Visual C++ 6.0'. The 'C++' section has fields for 'Short Name', 'Class', 'Header File', and 'CPP File'. The 'COM' section has fields for 'CoClass', 'Interface' (IWinLCLogicExtension), 'Type', and 'ProgID'. At the bottom are '< Back', 'Next >', and 'Cancel' buttons.



## Entering ODK Project Subcommands

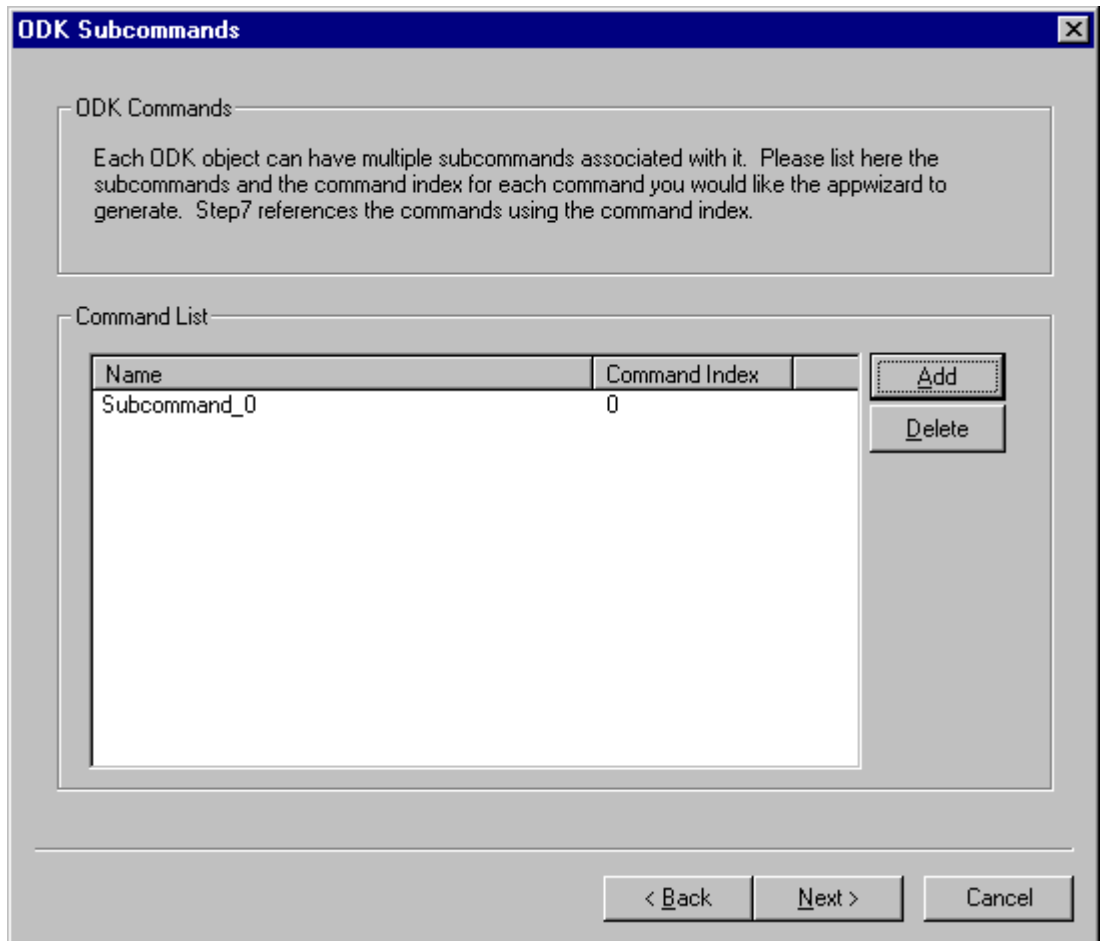
Your C/C++ program can call subcommand functions to further customize your program's functionality. A subcommand is a unique function within your C/C++ program that can be called from SFB65002 (EXEC\_COM). By using subcommands, you can create one RTDLL or COM object that performs a variety of tasks, rather than using several RTDLLs or COM objects 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 RTDLL or COM object, which in turn calls the subcommand function specified by the Command parameter.

### Note

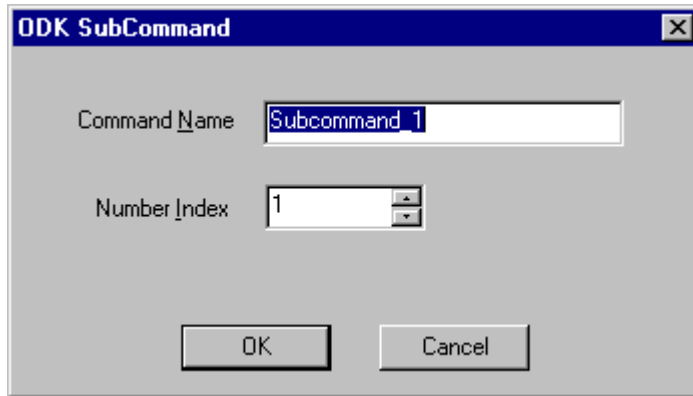
The application wizard requires that you specify at least one subcommand; however, your RTDLL or COM object does not have to make use of subcommands. If you do not have more than one type of task for your custom software to perform, you can implement the custom software directly in the Execute function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

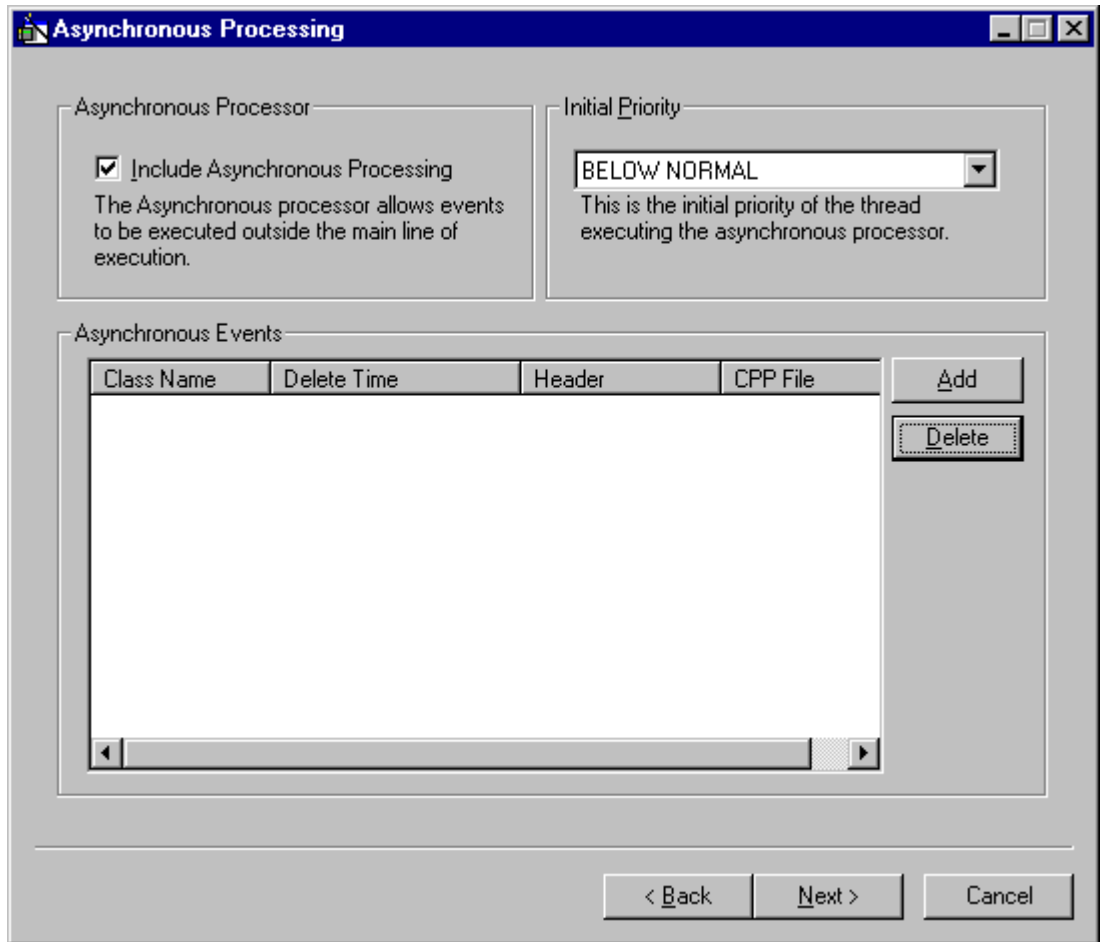
Highlight a subcommand in the ODK Subcommands window, and click the Delete button if you need to delete a subcommand. If you need to rename a subcommand, first delete the subcommand, and then add it using the new name.

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 WinLC RTX scan cycle. The asynchronous processor uses objects derived from the EventMsg class to carry out event specific functions. Asynchronous processing is optional.

1. Select the Include Asynchronous Processing check box to add asynchronous processing to the generated WinAC RTX ODK project.

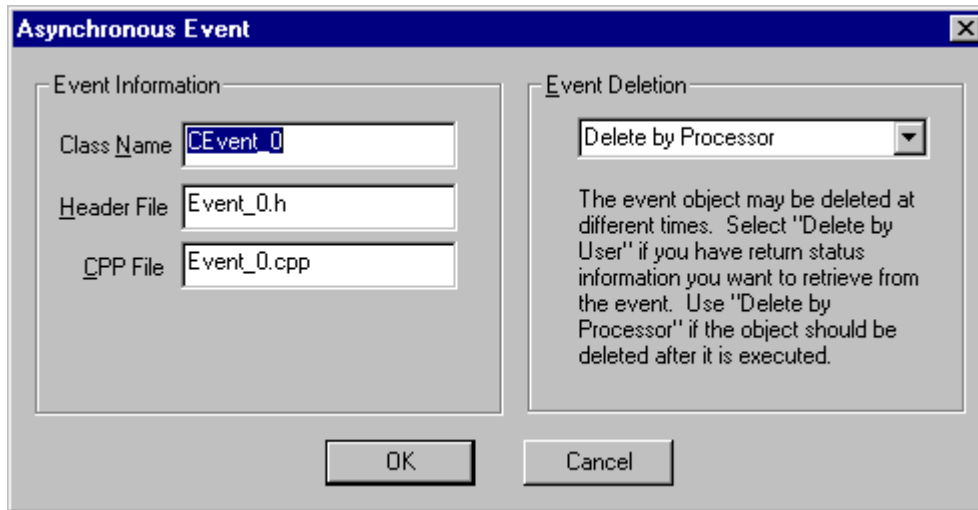


2. Click the Add button to add an asynchronous event.

**Note**

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

3. 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.

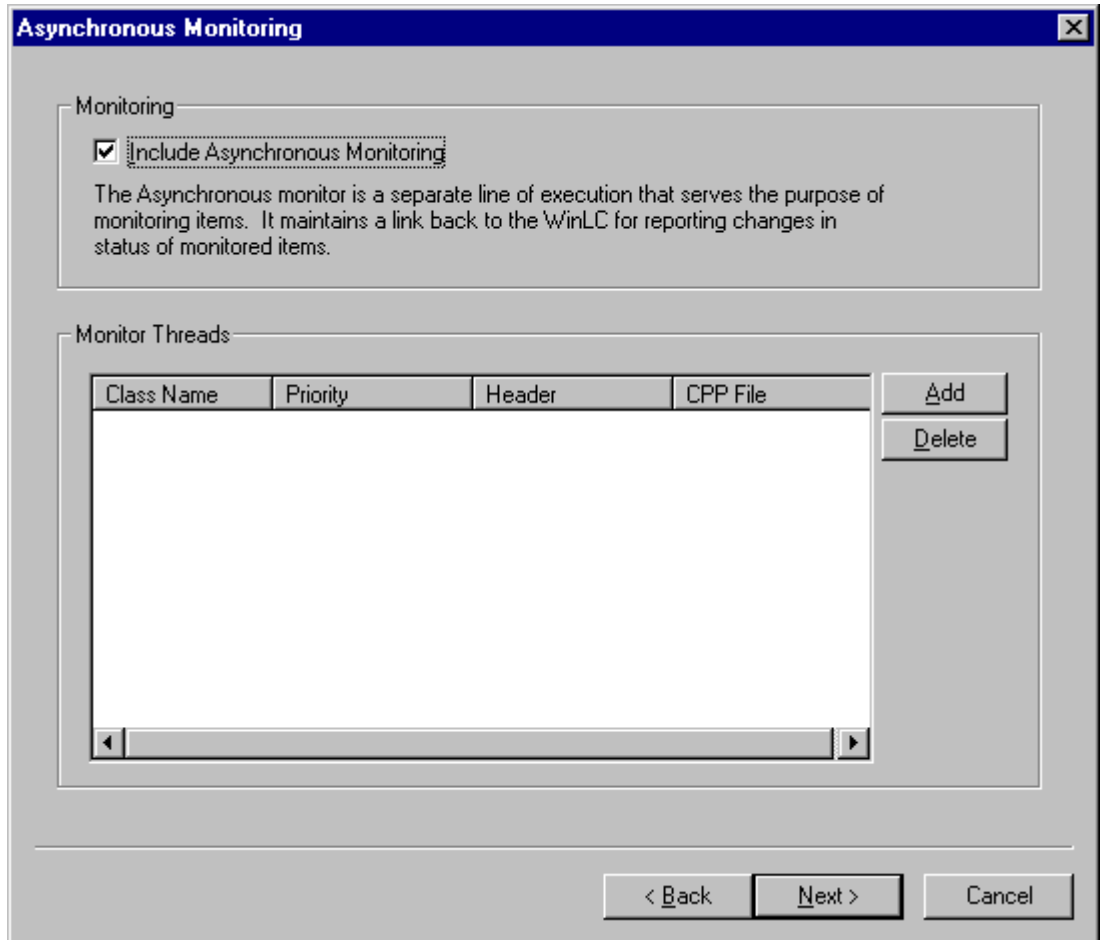


4. Select the type of Event Deletion that you prefer, based on the instructions on the dialog, and click OK.
5. Click Next when you have completed Asynchronous Processing configuration.

## 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 COM object 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 WinLC RTX to perform this type of background activity.

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

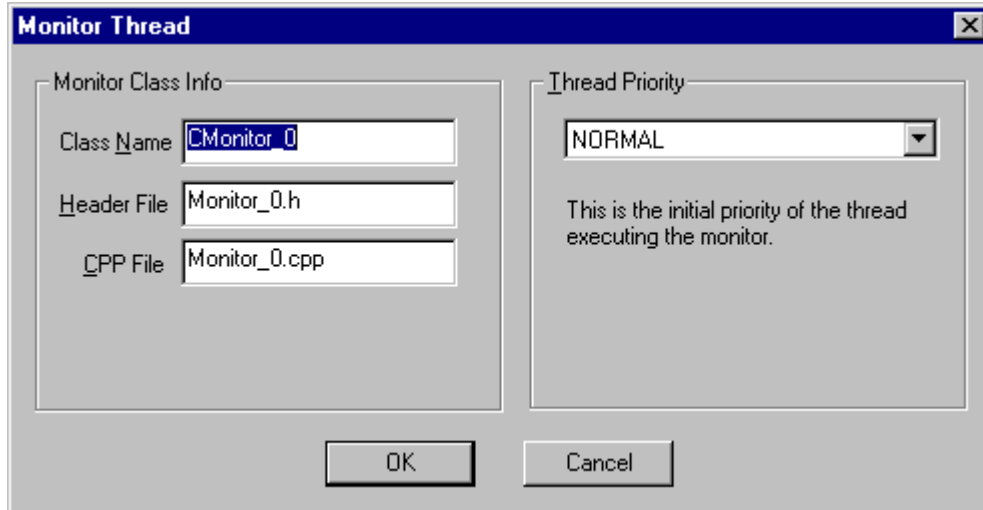


2. Click the Add button to add a monitor thread.

**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 if your project is configured as a real-time application. Then, click OK to close the Monitor Thread dialog.



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

### Guidelines for Selecting a Thread Priority

For each asynchronous monitor thread in a real-time application, you must assign it an initial priority level. RTX priorities range from 0 to 127. The application wizard offers five choices for the initial priority that are relative to the default main program thread priority:

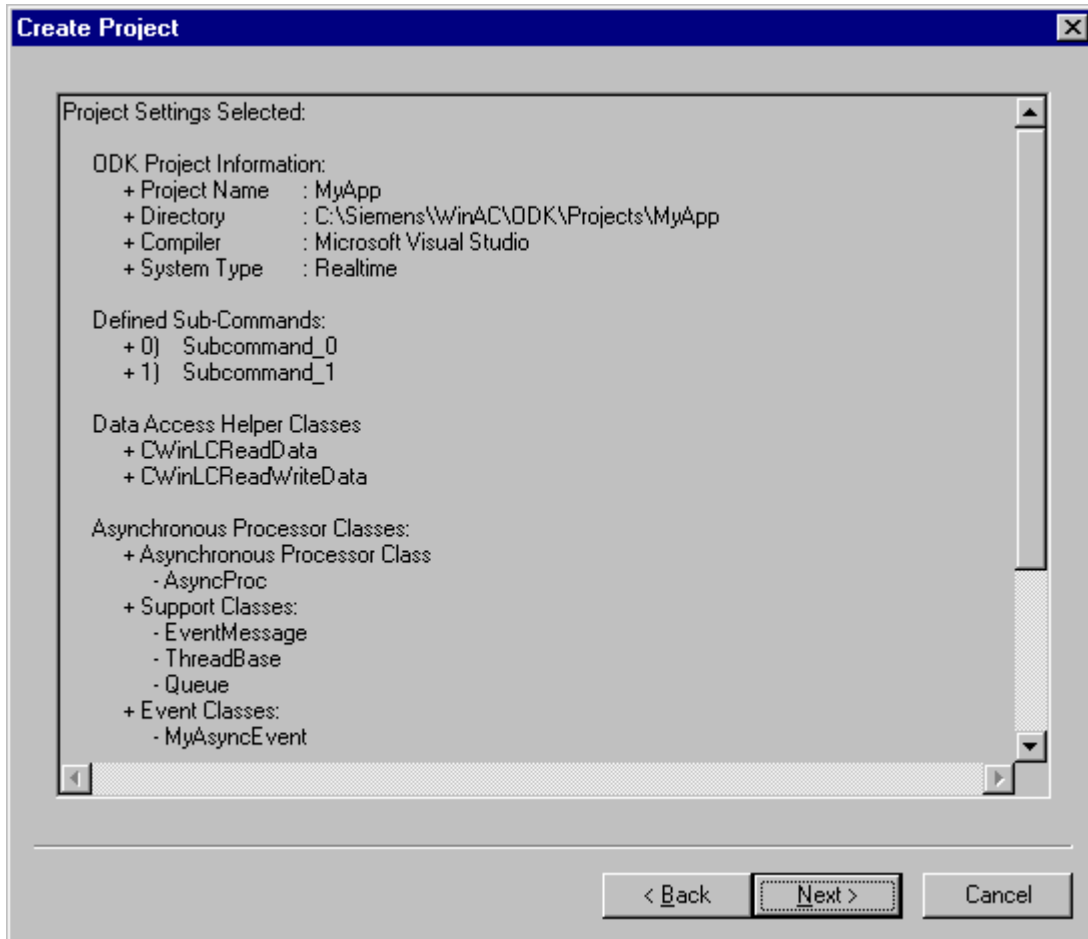
Priority Description	RTX Priority Value
LOWEST	25
BELOW_NORMAL	50
NORMAL	75
ABOVE_NORMAL	100
HIGHEST	125

The default monitor thread priority is NORMAL. If an event is of a background nature, then set the thread priority to be either BELOW\_NORMAL, or LOWEST. If the event corresponds to an important system event or interrupt, assign it a priority of ABOVE\_NORMAL or HIGHEST. Reserve the priority of HIGHEST for critical events in your application.

The thread priority can be changed during run time by your application. You can query the priority of the main program thread by calling the function `GetThreadPriority` from a function in this thread, for example, `Activate`. Then you can use the function `ChangePriority` to assign a monitor thread priority to be less than or greater than the main program thread priority. Use of the functions `GetThreadPriority` and `ChangePriority` allow you to assign a thread priority of an event relative to the priority of the main program, or relative to any other event. The example program `DemoRTX` illustrates the manipulation of thread priority values.

## Generating the WinAC RTX ODK Project

After you have finished the configuration described above, 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 RTX 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. This pathname is Siemens\WinAC\ODK\Projects\

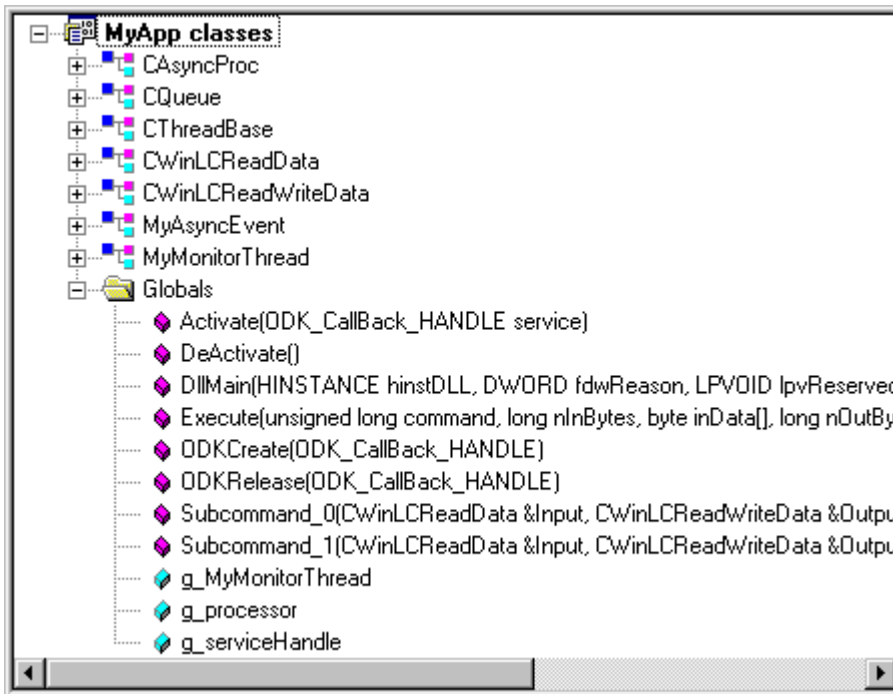
## Project Shell

The WinAC RTX 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++ 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++ Extension Software

The WinAC RTX ODK Application Wizard creates a skeleton C/C++ project that, when compiled, produces either a real-time dynamic link library (RTDLL) or a COM object. The project is configured for a RTDLL if you selected Real-time for the target platform on the Project Information dialog of the application wizard. It is configured for a COM object if you selected Windows NT for the target platform.

The C/C++ project contains functions that are the interface to the STEP 7 program. In a real-time project, these functions are global functions. In a Windows NT COM project, these functions belong to a COM object named IWinLCLogicExtension. 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 (real-time only - can be left empty)
- Activate (can be left empty)
- Deactivate (can be left empty)
- ODKRelease (real-time only - can be left empty)
- Execute
- Execute function subcommands (if configured)

### The ODKCreate Function

The ODKCreate function is responsible for creating the extension object for a real-time application. 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.

#### Note

The function ODKCreate exists only in real-time applications. The class constructor for the COM object provides this functionality for a Windows NT application.

### The Activate Function

The Activate function is called by WinLC RTX 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 WinLC RTX 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

WinLC releases your extension objects on a memory reset (MRES) or on CPU shutdown. Because both the MRES and CPU shutdown first change WinLC RTX to STOP mode, DeActivate is always called before the RTDLL and/or COM objects are released.

## The ODKRelease Function

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

### Note

The function ODKRelease exists only in real-time applications. The class destructor for the COM object provides this functionality for a Windows NT application.

## The Execute Function and its Subcommands

The Execute function is executed when SFB65002 (EXEC\_COM) is called in your STEP 7 program. It is the function that is called as part of your STEP 7 program and 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.



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. Functions are provided to access the data in the 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. Functions are provided in this class to read and write STEP 7 data types to the 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 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.
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 monitoring is enabled, the Execute function must be overloaded to provide customized monitoring actions.

## Programming Asynchronous Events

You can define events to execute asynchronously on a thread of execution separate from the WinLC RTX scan cycle. This allows the extension object to schedule and execute actions that can take a long time while allowing WinLC RTX 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 WinLC RTX scan cycle.

Consider an example where WinLC RTX controls a stapler assembly line. One of the WinLC RTX tasks is to send an E-mail notification when supplies are running low. This can be accomplished by using WinAC RTX ODK to create a COM object for this purpose. 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 WinLC RTX 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.

The following classes comprise the asynchronous processor class:

- **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 `Execute` function that is called during the main WinLC RTX scan cycle or in one of the `Execute` subcommands. This `Execute` function is a global function in a real-time project and a member function of the `IWinLCLogicExtension` class accessed through a pointer in a Windows-NT target platform project. This `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 RTX 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.

## Monitor Thread Considerations

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 DLL to be released, WinLC RTX calls the DeActivate function, which calls PauseThread to pause the monitor threads, and then calls the ODKRelease function, which calls StopThread to stop the monitor threads. (For a Windows NT DLL, WinLC RTX calls the class destructor for your COM object, which stops the monitor threads.) The StopThread function stops the threads, sets a variable named m\_ExitThread to true, and then 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 DLL, 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 DLL, and an event such as a controller shutdown or memory reset causes the DLL to be released, your code can not return properly. This is because the DLL is unloaded following the controller shutdown or memory reset event. Therefore, the call outside of the DLL from the asynchronous thread has no place to return.

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

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

You can examine the program DemoRTX in the folder Siemens\WinAC\ODK\Examples to see 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 COM object. Once it is built and loaded, your STEP 7 program can use it.

### Building an RTDLL

To compile your project as an RTDLL in Microsoft Visual C++, follow these steps:

1. Select **Build > Set Active Configuration...** to set the project configuration.
2. Select **<project name> - Win32 RTSS Release** from the list and click OK to ensure that the project is configured for real-time.
3. Select **Build > Build <project name>.rtdll** to build the project.

In Microsoft Visual C++, the Build command for an RTDLL also loads the RTDLL. If you are working in an environment other than Microsoft Visual C++, you must build the RTDLL according to your development environment, and then use the `rtssrun` command to register the RTDLL after you build it. To use `rtssrun` to register the RTDLL after it is built, follow these steps:

1. Select **Start > Programs > Command Prompt** to open a DOS command prompt window.
2. Type `cd siemens\WinAC\Odk\Projects\<Project Name>\RTSS_Release` to change to the folder containing the RTDLL that you just built.
3. Type `rtssrun /dll <project name>.rtdll` at the DOS command prompt to register the RTDLL.

### Building a COM Object for Windows NT

To compile your project as a COM object DLL in Microsoft Visual C++, follow these steps:

1. Select **Build > Set Active Configuration...** to set the project configuration.
2. Select **<project name> - Win32 Release** from the list and click OK to ensure that the project is configured for Windows NT.
3. Select **Build > <project name>.dll** to build the project.

You do not have to load the DLL for your COM object.

## **Building a Windows NT Debug Version of an RTDLL**

To compile your real-time project so that you can debug it from Windows NT, follow these steps:

1. Select **Build > Set Active Configuration...** from the Visual C++ menu to set the project configuration.
2. Select **<project name> - Win32 RTX Debug** from the list and click OK to ensure that the project is configured as a Windows NT debug version of a real-time project.
3. Select **Build > Rebuild All** to recompile the project. This step produces a DLL that can run in Windows NT under the NTRTX\_Debug directory of your project.

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



## Loading the WinAC RTX ODK Library Into STEP 7

WinAC RTX 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 Siemens\WinAC\Odk\Step7\S7libs and double-click the file OdkRtLib.zip.
4. Select S7libs on the "Select destination directory" dialog and click OK. The SIMATIC Manager extracts the WinAC RTX ODK library into OdkRtLib in the Siemens\Step7\S7libs directory. If it appears, click OK on the Retrieving dialog that informs you of the project location.
5. Click Yes on the final Retrieve dialog to open the WinAC RTX ODK library.

STEP 7 opens the library. This library contains the following blocks:

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

## Programming the STEP 7 Program

The two SFBs that WinAC RTX ODK provides allow you to use your custom RTDLL or COM object as part of the STEP 7 program that WinLC RTX executes. These two SFBs are listed below:

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

WinLC RTX executes these SFBs like any other SFB. SFB65002 (EXEC\_COM) calls the Execute function of your ODK RTDLL or COM object, and the scan time is 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. Because SFB65002 (EXEC\_COM) does not finish its execution until the COM object finishes, the time required to execute your custom application is added to the scan cycle.

In order to use these SFBs in your STEP 7 program, you must have loaded the WinAC RTX ODK library. You can then insert these SFBs into your program just like any other STEP 7 element. To open the WinAC RTX 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 from the list of User Projects.

**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 RTX ODK Library.
4. Select and drag the SFBs from the WinAC RTX 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 COM object. 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 RTDLL software every scan cycle, then put this call in OB1, or an FB called from OB1.

Your STEP 7 program is now programmed to use the custom C/C++ software that you created.

## Debugging the Extension Object

You use the debugger of Visual C++ to test your extension object with the STEP 7 program running on WinLC RTX.

### Note

The use of breakpoints in your extension object can cause WinLC RTX to exceed the maximum scan cycle.

For real-time applications, you must first build your project as a Windows NT DLL for debugging. To debug either a Windows NT DLL or a COM object, you must perform these tasks:

- Create the STEP 7 program (using the SFBs).
- Provide the path name for the WinLC RTX executable.
- Download the STEP 7 program to WinLC RTX.
- Debug your application (RTDLL or COM object).

## Building a Windows NT Debug Version of an RTDLL

If you are debugging a COM object, it is already built as a Windows NT DLL. If you are debugging an RTDLL, you must build an NTRTX\_DEBUG version of the RTDLL. To do this, follow these steps:


1. Select **Build > Set Active Configuration...** from the Visual C++ menu.
2. Select **Win32 RTX Debug** for your project and click OK.
3. Select **Build > Rebuild All** to recompile the project. This step produces a DLL that can run in Windows NT under the NTRTX\_Debug directory of your project.
4. Find and open the block that calls SFB65001 (CREA\_COM) in your STEP 7 program. Change the initial value of the variable InstanceID in the block's variable declaration table to specify the full pathname to the debug DLL, using the syntax in this example:

```
' *dll:C:\Siemens\WinAC\ODK\Examples\HistoDLL\NTRTX_Debug\HistoDLL
.dll'
```

5. Update any blocks that call this block and recompile and download all affected blocks to WinLC RTX.

## 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 WinLC RTX (typically, Siemens\WinAC\WinLCRTX) and select the following file:  
  
S7WLCRTX.EXE
3. Wait for Visual C++ to start WinLC RTX, and then start the SIMATIC Manager and open the STEP 7 program that uses the SFBs of .
4. Download your STEP 7 program to WinLC RTX.
5. Place WinLC RTX in RUN mode or RUN-P mode.
6. Test your RTDLL or COM object by triggering events and setting breakpoints.

### Note

If you use breakpoints to test your application, you could cause WinLC RTX to exceed the maximum scan cycle time.

You can use the tuning panel of WinLC RTX to monitor the effects of the extension object on the scan cycle.

## Recovering from a WinLC RTX Crash

If your application causes WinLC RTX to crash (to abort unexpectedly), use this procedure:

- If you are using the Debugger of Visual C++ to test your application, select the **Debug > Stop Debugging** to recover from the crash.
- If you are not testing your application with the Visual C++ Debugger, terminate the following process:

S7WLCRTX.EXE

To confirm that the real-time components have terminated, follow these steps:

1. Open an MS-DOS window.
2. Type **rtsskill** in the DOS window. This command displays all currently loaded and running real-time processes, with line numbers on each process. If the s7wlcvmx.rtss process is still running, type **rtsskill <line number>** where the line number is the line number corresponding to the s7wlcvmx.rtss process.

Do not terminate the S7WLSAPX.EXE process. This is a daemon process that always runs in the background.

## Recovering from an Infinite Loop

If your custom software executes an infinite loop, your computer will lock up. You must reboot the computer if this happens. An infinite loop exhibits the following symptoms:

- When an RTDLL executes an infinite loop, the system either displays a blue screen or it displays a dialog such as the following:



Regardless of any attempted corrective action, a Windows NT system failure ("blue screen" event) occurs. You can not recover with RTSSkill or the debugger.

- When a COM object executes an infinite loop, the system locks up with only mouse movement possible.

In either case, you must reboot the system.

## ODK Support Software

### Data Access Helper Classes

The data access helper classes provide access to data in WinLC RTX. The two data access helper classes are:

- CWinLCReadData
- CWinLCReadWriteData

The read and write functionality has been divided up as a means of providing very basic security on the input and output parameters of the Execute function.

Use the functions of the data access helper classes to access data in WinLC RTX. These functions help 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 operating systems, including Windows NT.

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



Your in-process function (thread) can corrupt WinLC RTX 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 WinLC RTX or any other application.

## WinAC RTX ODK Library for STEP 7

WinAC RTX ODK provides a STEP 7 library ("WinAC RTX 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 WinLC RTX scan cycle. In order to use these SFBs in your STEP 7 program, you must perform the following tasks:

- Load the WinAC RTX 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 (Includes a routing prefix that indicates whether the extension object is a COM object, an RTDLL, or an NT debug version of an RTDLL)
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>'
NT debug version of an RTSS DLL	'*DLL:<DLL name>'

SFB65001 evaluates input conditions and performs the following actions:

1. If the extension object has not already been created, SFB65001 creates it, using the InstanceID parameter to create a ClassID. (There is only one instance of this object created.) The specific actions depend upon whether the application is targeted for Windows NT or RTX.
  - In a Windows NT application, SFB65001 calls the CoInitializeEx function (or ensures that it was previously called) with the COINIT\_MULTITHREADED option. SFB65001 then converts the input parameter ExtensionID to a ClassID and then calls the CoCreateInstance function to create the object.
  - In a real-time application, SFB65001 calls the ODKCreate function.
  - In both cases, SFB65001 adds this object instance to the internal list of created WinAC RTX ODK objects.
2. If the extension object has already been created, SFB65001 maintains the WinAC RTX ODK handle for the previously created object (an index to locate the object pointer).
3. If this is the first call to this SFB after leaving STOP or if the extension object was just created, SFB65001 invokes the Activate function
4. SFB65001 sets the Status parameter to the WinAC RTX ODK handle (or error code) and sets the BR bit. To see ways to check for the return value, refer to the sample program "HistoDLL" installed by WinAC RTX ODK.

### 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	An NT response timeout occurred.



## 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 RTX 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)**

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

<b>Error Code</b>	<b>Message</b>	<b>Description</b>
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.

## Auxiliary STEP 7 Interface Functions

WinAC RTX 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 WinLC RTX controller. The value of state will be one of the following modes:

- STOP
- HALT
- STARTUP\_100
- STARTUP\_101
- STARTUP\_102
- RUN

**Note**

ODK\_ReadState is the function name in a real-time application. It is named ReadState in a Windows NT application.

### ODK\_ScheduleOB

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

**Note**

ODK\_ScheduleOB is the function name in a real-time application. It is named ScheduleOB in a Windows NT application.

When you schedule an OB, select one whose standard S7 behavior is closest to the way you plan to use the OB. You should also choose an OB that 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:

- OB80 (Time error, such as a watchdog alarm)
- OB4x (Hardware interrupts)
- OB82 (Diagnostic Alarm interrupt)
- OB83 (Insert/Remove Module interrupt)
- OB84 (CPU Hardware Fault)
- OB87 (Communication Error interrupt)

**Note**

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

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. WinLC RTX 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:

**Data Type 2**

Value (hexadecimal)	Description
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

**Data Type 1**

Value (hexadecimal)	Description
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:

Error	Message	Description
0x8107	E_INVALID_OB_STATE	WinLC RTX 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.

## ODK\_ReadSysData

ODK\_ReadSysData is not implemented in the current version of WinLC RTX ODK.

### Note

ODK\_ReadSysData is the function name in a real-time application. It is named ReadSysData in a Windows NT application.

## Examples of Calling an Auxiliary Function

In a real-time application, 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:

```
// ScheduleOB80
void ScheduleOB80(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
            80, // execute OB80
            0xC4, // dataType2 (for long word) - two 16 bit words
            0x58, // dataType1 (for short word) - time in milliseconds
            deviation, // data1
            mode);
    }
}
```

In a Windows NT application, you must access these functions through a pointer to IWinLCServices. When your STEP 7 program calls SFB65001 (CREA\_COM), WinLC RTX creates your extension object and calls the Activate function, which returns a pointer to IWinLCServices. You can call any of the IWinLCServices member functions from the Execute, Activate, or DeActivate functions. The following example (taken from the Histogram sample project) shows a function that calls ScheduleOB in a Windows NT application:

```
// ScheduleOB80
void
CHistogram::ScheduleOB80(unsigned short mode, unsigned short
deviation) const
{
    m_WinLCSvc->ScheduleOB(
        0x35, // execute asynchronous error OB
        0x01, // use cycle time error event number
        0xFE, // fill in configured sequence layer
        80, // execute OB80
        0xC4, // dataType2 (for long word) - two 16 bit words
        0x58, // dataType1 (for short word) - time in milliseconds
        deviation, // data1
        mode); // data2 (half is 0)
}
```

The auxiliary STEP 7 functions are included in the Object Web.

## **Object Web**

The class, function, and parameter descriptions that are used in ODK are described in an object web. This is a web-based display of the classes used in your C/C++ project. Directions for navigating within the site are included on the site home page.

There are differences in the project structure between a real-time application and a Windows NT application. Most of the functions that you use in programming your extension object have the same interface, but not all. Click the appropriate link below to view the object web for the platform of your choice:

[Real-time ODK Object Web](#)

[Windows NT ODK Object Web](#)

# Index

## A

- Activate function, 27, 42
- Add/Remove Programs, 7
- Additional Assistance, iii
- Additional sample programs, 15
- Application wizard, 16, 17
- Applications (Customer Support), v
- Asynchronous Events, 17, 30
- Asynchronous monitoring, 17, 32
  - CThreadBase, 32
- Asynchronous processing, 17, 27, 30
- Asynchronous processor, 17, 27, 30
  - CAsyncProc class, 29, 30
  - CEventMsg, 29, 30
  - CQueue, 29, 30
- Authorization, 7
- Auxiliary STEP 7 Interface functions, 45

## B

- Basic tasks, 16
- Breakpoints, 37
- Building, 33, 37
  - COM object, 33
  - extension object, 33
  - RTDLL, 33
  - Windows NT debug version, 33, 37

## C

- C/C++, 27, 49
- C/C++ application, 1, 27
  - C/C++ custom software, 36
  - Non-deterministic functions, 30
  - Programming, 27
  - SFBs, 36
- CAsyncProc, 29, 30
- CEventMsg, 29, 30
- Classes, 27, 30, 49
- CoCreateInstance, 42
- COINIT\_MULTITHREADED, 41
- CoInitializeEx, 42
- COM Object, 1, 33, 37
  - Debugging, 37
- Computer requirements, 6

- Configuring, 17

- asynchronous events, 17
  - asynchronous monitoring, 17
  - asynchronous processing, 17
  - monitor threads, 17
  - project information, 17
  - subcommands, 17

- CQueue, 29, 30

- CREA\_COM, 35, 36, 41

- Creating, 16, 30

- asynchronous events, 17, 30

- C/C++ project, 16

- monitor threads, 17, 32

- STEP 7 program, 16

- CThreadBase, 29, 32

- Custom COM Interface, 16

- Implementing, 16

- Customer Support, v

- CWinLCReadData, 27, 40

- CWinLCReadWriteData, 27, 40

- Cycle, iii, 1, 11, 17, 27, 30, 36, 37, 41

## D

- Data access helper classes, 40

- DataType1, 45

- DataType2, 45

- DeActivate function, 27

- Debugger, 37

- Visual C++, 37

- Debugging, 37

- COM Object, 37

- effect on scan cycle, 37

- RTDLL, 37

- Delta Tau sample program, v

- DemoRTX, 15

- DemoRTX2, 15

- DemoRTX3, 15

- DemoRTX4, 15

- DemoRTX5, 15

## E

- E-mail (Customer Support), v

- Enabling, 17

- Asynchronous monitoring, 17



- Asynchronous processing, 17
- Error Codes, 42
  - ODK\_ScheduleOB, 47
  - SFB65001, 42
  - SFB65002, 44
- Events, 30, 32
- Example Program, 11
- Example project, 9
- EXEC\_COM, 35, 36, 41
- Execute function, 27, 30, 32, 40
- Execution threads, 1
- Extension object, 1
  - building, 33
- Extension Software, 27

## F

- Functions, 49

## G

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

## H

- Hardware requirements, 6
- HistoDLL, 9, 11, 14
- HistoDLL STEP 7 program, 11
  - execution, 11
  - listings, 11
  - Object Web, 14
  - structure, 11
- Histogram, 9
- Hotline, v

## I

- Implementing, 16
  - COM object, 27
  - custom software, 16
  - ODK Application, 16
  - RTDLL, 27
- In-process applications, 1
- Inserting, 36
  - SFBs, 36
- Installation, 6, 7
  - System requirements, 6
- InstanceID, 37, 41
- Internet (Customer Support), v
- Introduction, 1, 9

- Invalid addresses, 40
- IWinLCLogicExtension, 17, 27, 30
- IWinLCServices, 45

## L

- Loading, 35
- Loading STEP 7 library, 16
- Loading the WinAC RTX ODK Library, 9
  - WinAC RTX ODK Library, 35

## M

- Microsoft Visual C++, 17
- Monitor classes, 17, 32
- Monitor threads, 17, 32

## N

- Non-deterministic functions, 30
- Number, 17
  - Events, 17
  - Monitors, 17

## O

- OB, 45
- OB1, 36
- OB100, 36
  - scheduling, 45
- Object Shell, 17
- Object Web, 14, 49
- OBs, 1, 45
- ODK Application, 16
  - Implementing, 16
- ODK support classes, 27
- ODK\_ReadState, 45
- ODK\_ReadSysData, 47
- ODK\_ScheduleOB, 45
- ODKCreate function, 27
- ODKRelease function, 27
- OdkRtLib, 35
- Open Development Kit, 1
- Open Development Kit Installation, 7
- Overview, 1

## P

- Priority, 17
  - Thread, 17
- Product Overview, 1
- Program cycle, iii, 1, 11, 17, 27, 30, 36, 37, 41

Programming, 27  
  C/C++, 27  
  Extension software, 27  
Project Information, 17  
Project Shell, 17

## **Q**

Queue Class, 27, 30

## **R**

RAM, 6  
ReadState, 45  
ReadSysData, 47  
Recovering, 37  
  from a blue screen event, 37  
  from a WinLC RTX crash, 37  
  from an infinite loop, 37  
Removing Installation, 7  
Requirements, 6  
RTDLL, 1, 14, 33, 37  
  debugging, 37  
  sample program, 14  
rtsskill, 37  
rtssrun, 33  
Running in Debug, 37

## S

- S7WLCRTX.EXE, 37
- S7WLSAPX.EXE, 37
- Sample program, 9, 11, 14
- Scan cycle, iii, 1, 11, 17, 27, 30, 36, 37, 41
- ScheduleEvent function, 30
- ScheduleOB, 45
- ScheduleOB80, 14
- Scheduling, 45
  - OB, 45
- Set Active Configuration, 33
- SetDelTime, 30
- SFB65001, 36, 41
  - Error Codes, 42
- SFB65002, 36, 41
  - Error Codes, 44
- SFBs, 1, 35, 36, 41
- Short Name, 17
- SIMATIC Manager, 9, 11, 35, 36, 37
- Software requirements, 6
- STEP 7 interface functions, 45
- STEP 7 library, 41
- STEP 7 program, 9, 11, 36
- STEP 7 program cycle, iii, 1, 11, 17, 27, 30, 36, 37, 41
- Stop Debugging, 37
- Subcommands, 17, 27
- Support, v
- Support classes, 27
- System Function Blocks, 41

System Requirements, 6

## T

- Technical support, v
- Telephone (Customer Support), v
- Testing, 37
  - your custom application, 38
- Thread priorities, 17
- Threads, 1, 17, 27

## U

- Uninstalling, 7
- Updates (Customer Support), v
- Using subcommands, 17
- Using the Application Wizard, 17

## V

- Visual C++, 37
  - Debugger, 37
- Visual Studio (application wizard), 17

## W

- WinAC RTX ODK, 1
- WinAC RTX ODK handle, 41
- WinAC RTX ODK Library, 35, 36, 41
- Windows NT Debug Version, 33
  - Building, 33
- WinLC RTX data, 40
- WinLC RTX scan cycle, iii, 1, 11, 17, 27, 30, 36, 37, 41



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 M/S 519  
3000 BILL GARLAND ROAD  
PO BOX 1255  
JOHNSON CITY TN USA 37605-1255

Include this information:

**From**

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

