SIEMENS

SIMATIC

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

Programmier- und Bedienhandbuch

09/2016

A5E35253935-AC

Rechtliche Hinweise

Warnhinweiskonzept

Dieses Handbuch enthält Hinweise, die Sie zu Ihrer persönlichen Sicherheit sowie zur Vermeidung von Sachschäden beachten müssen. Die Hinweise zu Ihrer persönlichen Sicherheit sind durch ein Warndreieck hervorgehoben, Hinweise zu alleinigen Sachschäden stehen ohne Warndreieck. Je nach Gefährdungsstufe werden die Warnhinweise in abnehmender Reihenfolge wie folgt dargestellt.

⚠GEFAHR
bedeutet, dass Tod oder schwere Körperverletzung eintreten **wird**, wenn die entsprechenden Vorsichtsmaßnahmen nicht getroffen werden.

⚠WARNUNG
bedeutet, dass Tod oder schwere Körperverletzung eintreten **kann**, wenn die entsprechenden Vorsichtsmaßnahmen nicht getroffen werden.

⚠VORSICHT
bedeutet, dass eine leichte Körperverletzung eintreten kann, wenn die entsprechenden Vorsichtsmaßnahmen nicht getroffen werden.

ACHTUNG
bedeutet, dass Sachschaden eintreten kann, wenn die entsprechenden Vorsichtsmaßnahmen nicht getroffen werden.

Beim Auftreten mehrerer Gefährdungsstufen wird immer der Warnhinweis zur jeweils höchsten Stufe verwendet. Wenn in einem Warnhinweis mit dem Warndreieck vor Personenschäden gewarnt wird, dann kann im selben Warnhinweis zusätzlich eine Warnung vor Sachschäden angefügt sein.

Qualifiziertes Personal

Das zu dieser Dokumentation zugehörige Produkt/System darf nur von für die jeweilige Aufgabenstellung **qualifiziertem Personal** gehandhabt werden unter Beachtung der für die jeweilige Aufgabenstellung zugehörigen Dokumentation, insbesondere der darin enthaltenen Sicherheits- und Warnhinweise. Qualifiziertes Personal ist auf Grund seiner Ausbildung und Erfahrung befähigt, im Umgang mit diesen Produkten/Systemen Risiken zu erkennen und mögliche Gefährdungen zu vermeiden.

Bestimmungsgemäßer Gebrauch von Siemens-Produkten

Beachten Sie Folgendes:

⚠WARNUNG
Siemens-Produkte dürfen nur für die im Katalog und in der zugehörigen technischen Dokumentation vorgesehenen Einsatzfälle verwendet werden. Falls Fremdprodukte und -komponenten zum Einsatz kommen, müssen diese von Siemens empfohlen bzw. zugelassen sein. Der einwandfreie und sichere Betrieb der Produkte setzt sachgemäßen Transport, sachgemäße Lagerung, Aufstellung, Montage, Installation, Inbetriebnahme, Bedienung und Instandhaltung voraus. Die zulässigen Umgebungsbedingungen müssen eingehalten werden. Hinweise in den zugehörigen Dokumentationen müssen beachtet werden.

Marken

Alle mit dem Schutzrechtsvermerk ® gekennzeichneten Bezeichnungen sind eingetragene Marken der Siemens AG. Die übrigen Bezeichnungen in dieser Schrift können Marken sein, deren Benutzung durch Dritte für deren Zwecke die Rechte der Inhaber verletzen kann.

Haftungsausschluss

Wir haben den Inhalt der Druckschrift auf Übereinstimmung mit der beschriebenen Hard- und Software geprüft. Dennoch können Abweichungen nicht ausgeschlossen werden, so dass wir für die vollständige Übereinstimmung keine Gewähr übernehmen. Die Angaben in dieser Druckschrift werden regelmäßig überprüft, notwendige Korrekturen sind in den nachfolgenden Auflagen enthalten.

Siemens AG
Division Digital Factory
Postfach 48 48
90026 NÜRNBERG
DEUTSCHLAND

A5E35253935-AC
℗ 08/2016 Änderungen vorbehalten

Vorwort

Zweck der Dokumentation

Die vorliegende Dokumentation beschreibt die besonderen Merkmale des Open Development Kit (ODK) V2.0.

Definitionen und Namenskonventionen

In dieser Dokumentation werden die folgenden Begriffe verwendet:

- **CPU**: Bezeichnet die unter "Gültigkeitsbereich der Dokumentation" genannten Produkte.
- **ODK**: Open Development Kit
- **Windows**: Bezeichnet die von ODK unterstützten Microsoft Betriebssysteme.
- **STEP 7**: Zur Bezeichnung der Projektier- und Programmiersoftware verwenden wir in der vorliegenden Dokumentation "STEP 7" als Synonym für die Version "STEP 7 ab V13 SP1 (TIA Portal)".
- **DLL**: Dynamic Link Library
- **SO**: Shared Object
- **Visual Studio**: Microsoft Visual Studio

Erforderliche Grundkenntnisse

Die vorliegende Dokumentation wendet sich an Ingenieure, Programmierer und Wartungspersonal mit allgemeinen Kenntnissen über Automatisierungssysteme und speicherprogrammierbare Steuerungen.

Um diese Dokumentation zu verstehen, benötigen Sie allgemeine Kenntnisse über das Engineering von Automatisierungssystemen. Ferner benötigen Sie Grundkenntnisse zu folgenden Themen:

- Industrieautomatisierungssystem SIMATIC
- PC-basierte Automatisierung
- Umgang mit STEP 7
- Verwendung von Microsoft Windows-Betriebssystemen
- Programmierung mit C++

Gültigkeitsbereich der Dokumentation

Diese Dokumentation gilt für die Verwendung von ODK mit folgenden Produkten:

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

Hinweise

Beachten Sie auch die folgendermaßen gekennzeichneten Hinweise:

Hinweis

Ein Hinweis enthält wichtige Informationen zum in der Dokumentation beschriebenen Produkt, zur Handhabung des Produkts oder zu dem Teil der Dokumentation, auf den besonders aufmerksam gemacht werden soll.

Security-Hinweise

Siemens bietet Produkte und Lösungen mit Industrial Security-Funktionen an, die den sicheren Betrieb von Anlagen, Systemen, Maschinen und Netzwerken unterstützen.

Um Anlagen, Systeme, Maschinen und Netzwerke gegen Cyber-Bedrohungen zu sichern, ist es erforderlich, ein ganzheitliches Industrial Security-Konzept zu implementieren (und kontinuierlich aufrechtzuerhalten), das dem aktuellen Stand der Technik entspricht. Die Produkte und Lösungen von Siemens formen nur einen Bestandteil eines solchen Konzepts.

Der Kunde ist dafür verantwortlich, unbefugten Zugriff auf seine Anlagen, Systeme, Maschinen und Netzwerke zu verhindern. Systeme, Maschinen und Komponenten sollten nur mit dem Unternehmensnetzwerk oder dem Internet verbunden werden, wenn und soweit dies notwendig ist und entsprechende Schutzmaßnahmen (z.B. Nutzung von Firewalls und Netzwerksegmentierung) ergriffen wurden.

Zusätzlich sollten die Empfehlungen von Siemens zu entsprechenden Schutzmaßnahmen beachtet werden. Weiterführende Informationen über Industrial Security finden Sie unter (http://www.siemens.com/industrialsecurity).

Die Produkte und Lösungen von Siemens werden ständig weiterentwickelt, um sie noch sicherer zu machen. Siemens empfiehlt ausdrücklich, Aktualisierungen durchzuführen, sobald die entsprechenden Updates zur Verfügung stehen und immer nur die aktuellen Produktversionen zu verwenden. Die Verwendung veralteter oder nicht mehr unterstützter Versionen kann das Risiko von Cyber-Bedrohungen erhöhen.

Um stets über Produkt-Updates informiert zu sein, abonnieren Sie den Siemens Industrial Security RSS Feed unter (http://www.siemens.com/industrialsecurity).

Siemens Industry Online Support

Aktuelle Informationen erhalten Sie schnell und einfach zu folgenden Themen:

- **Produkt-Support**

  Alle Informationen und umfangreiches Know-how rund um Ihr Produkt, Technische Daten, FAQs, Zertifikate, Downloads und Handbücher.

- **Anwendungsbeispiele**

  Tools und Beispiele zur Lösung Ihrer Automatisierungsaufgabe – außerdem Funktionsbausteine, Performance-Aussagen und Videos.

- **Services**

  Informationen zu Industry Services, Field Services, Technical Support, Ersatzteilen und Trainingsangeboten.

- **Foren**

  Für Antworten und Lösungen rund um die Automatisierungstechnik.

- **mySupport**

  Ihr persönlicher Arbeitsbereich im Siemens Industry Online Support für Benachrichtigungen, Support-Anfragen und konfigurierbare Dokumente.

Diese Informationen bietet Ihnen der Siemens Industry Online Support im Internet (http://www.siemens.com/automation/service&support).

Industry Mall

Die Industry Mall ist das Katalog- und Bestellsystem der Siemens AG für Automatisierungs- und Antriebslösungen auf Basis von Totally Integrated Automation (TIA) und Totally Integrated Power (TIP).

Kataloge zu allen Produkten der Automatisierungs- und Antriebstechnik finden Sie im Internet (https://mall.industry.siemens.com).

Hinweise zu Fremdsoftware-Updates

Dieses Produkt beinhaltet Fremdsoftware. Für Updates/Patches an der Fremdsoftware übernimmt Siemens die Gewährleistung nur, soweit diese im Rahmen eines Software Update Servicevertrags von Siemens verteilt oder von Siemens offiziell freigegeben wurden. Andernfalls erfolgen Updates/Patches auf eigene Verantwortung. Mehr Informationen rund um unser Software Update Service Angebot erhalten Sie unter (http://www.automation.siemens.com/mcms/automation-software/de/software-update-service/Seiten/Default.aspx).

Inhaltsverzeichnis

Wegweiser Dokumentation

1

Einleitung

In der vorliegenden Dokumentation zum Open Development Kit (ODK) finden Sie alle notwendigen Informationen für die Nutzung der Software.

Übersicht der Dokumentation zur CPU

Die folgende Tabelle zeigt weitere Dokumente, welche die vorliegende Beschreibung ergänzen und im Internet erhältlich sind.

Tabelle 1- 1 Dokumentation für die CPU

Thema	Dokumentation	Wichtigste Inhalte
Beschreibung der CPU 1505SP und CPU 1507S	Bedienhandbuch CPU 1505SP und CPU 1507S (http://support.automation.siemens.com/WW/view/de/90466248/133300)	Diese Dokumentation beschreibt die vollständige Funktionalität der CPU 1505SP und CPU 1507S.
Beschreibung der CPU 1518-4 PN/DP ODK	Gerätehandbuch CPU 1518-4 PN/DP ODK (https://support.industry.siemens.com/cs/products?search=CPU%201518-4%20PN%2FDP%20ODK&mfn=ps&o=DefaultRankingDesc&lc=de-WW)	Diese Dokumentation beschreibt die vollständige Funktionalität der CPU 1518-4 PN/DP ODK.
Webserver	Funktionshandbuch Webserver (http://support.automation.siemens.com/WW/view/de/59193560)	Grundlagen Funktion Bedienung Diagnose über Webserver

Produktübersicht 2

2.1 Einführung in ODK 1500S

Überblick

ODK ist ein Entwicklungspaket, das Ihnen die Programmierung von eigenen Funktionen und die Generierung von Dateien, die STEP 7 direkt aufrufen kann, ermöglicht.

ODK dient als Schnittstelle für:

- Windows-Umgebung
  - Ausführung auf Ihrem Windows-PC
  - Nutzung der Ressourcen Ihres Windows-PC
  - Nutzung der Betriebssystemfunktionen und Systemressourcen mit Zugriff auf externe Hardware- und Software-Komponenten
- Echtzeit-Umgebung
  - Ausführung auf Ihrer CPU
  - Synchroner Funktionsaufruf (Algorithmik, Regler)

Der Aufruf mehrerer Anwendungen unter Windows oder in der Echtzeitumgebung ist möglich.

Die ODK-Anwendungen müssen im STEP 7-Programm genutzt werden.

Aufbau und Ausführung einer ODK-Anwendung

ODK unterstützt die Schnittstelle zum Aufruf eigener Hochsprachen-Programme aus dem Steuerungsprogramm der CPU.

ODK unterstützt folgende Templates:

- Ein mitgeliefertes Template zur Programmierung in Microsoft Visual Studio. Damit können Sie eine DLL-Datei erzeugen.
- Ein weiteres Template zur Programmierung in Eclipse. Damit können Sie eine SO-Datei erzeugen. Für Eclipse liefert ODK zusätzlich eine Klassenbibliothek mit.

Eine ODK-Anwendung erstellen Sie mit der Programmiersprache C++. ODK-Anwendungen können sowohl für die Windows- als auch für die Echtzeitumgebung erzeugt werden.

Das ODK-Programm kann auf folgende Arten ausgeführt werden:

- Synchron, d. h. bearbeitet als Teil des CPU-Zyklus (Ausführung in der Echtzeitumgebung)
- Asynchron, d. h. vom CPU-Programm gestartet und im Hintergrund beendet (Ausführung in der Windows-Umgebung)

Das außerhalb der CPU ablaufende Programm wird mit Microsoft Visual Studio oder Eclipse erstellt und als DLL- bzw. SO-Datei generiert. ODK-Anwendungen können sowohl unter Windows (DLL) als auch im Echtzeitkern der CPU (SO) ausgeführt werden. Die Funktionen der DLL- bzw. SO-Datei rufen Sie über Anweisungen im Anwenderprogramm auf.

Die CPU kann Funktionen in dynamisch ladbaren Bibliotheken ausführen. Es sind mehrere Funktionen in einer ODK-Anwendung möglich. Es gibt spezifische Funktionsbausteine für eine ODK-Anwendung:

- Laden und Entladen der ODK-Anwendung
- Je ein Funktionsbaustein zum Aufruf einer bestimmten Funktion

Die folgende Grafik gibt Ihnen einen schematischen Überblick über die Funktionsweise wie ODK-Anwendungen auf einem PC ausgeführt werden. Diese Grafik gilt für den S7-1500 Software Controller.

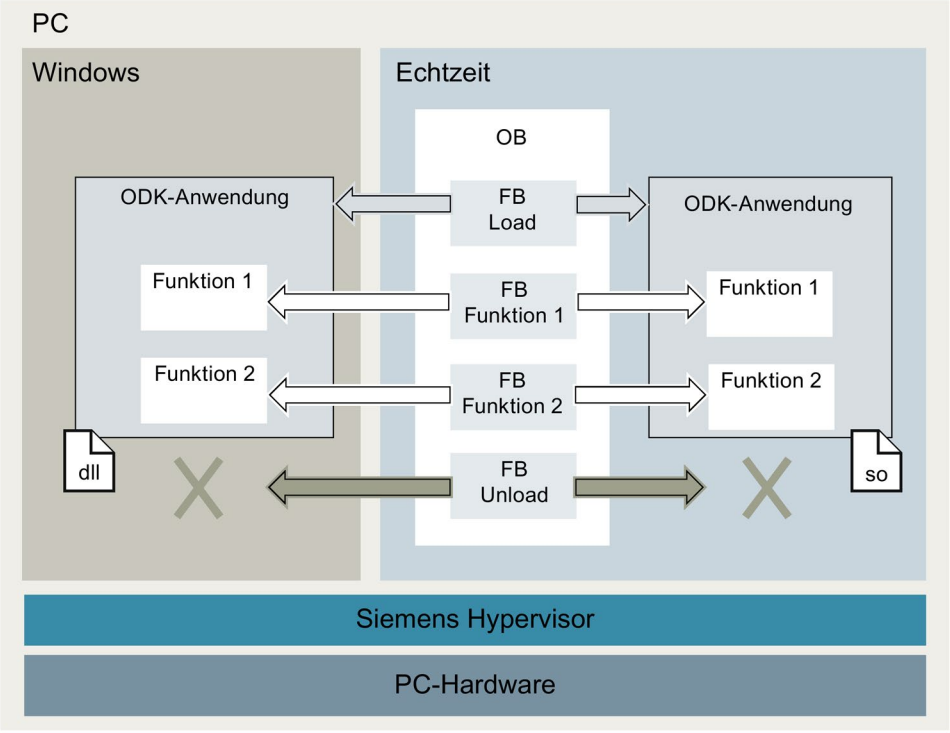


Bild 2-1 ODK-Anwendung auf einem PC ausführen

Die folgende Grafik gibt Ihnen einen schematischen Überblick über die Funktionsweise wie ODK-Anwendungen auf einer Hardware-CPU ausgeführt werden.

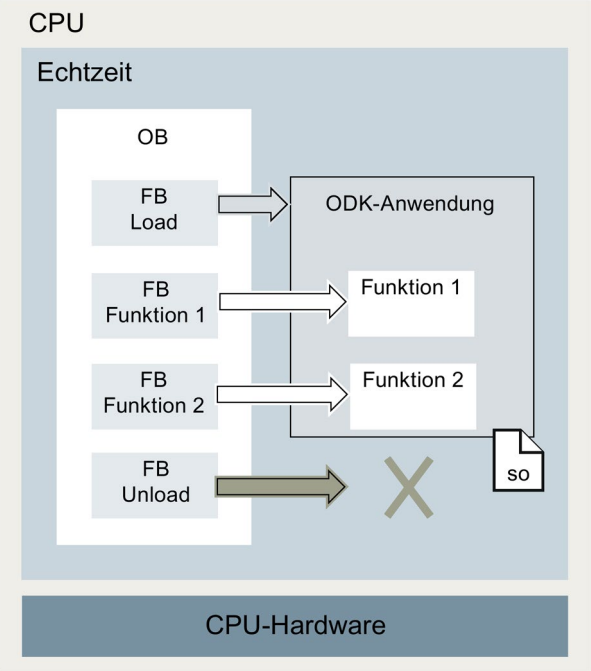


Bild 2-2 ODK-Anwendung auf einer Hardware-CPU ausführen

2.2 Entwicklungsumgebungen

Die Entwicklung einer ODK-Anwendung erfolgt mit einer Standard-Entwicklungsumgebung.

Für die Erstellung eines ODK-Projektes stehen Ihnen folgende Entwicklungsumgebungen zur Auswahl.

- Microsoft Visual Studio für Windows-Anwendungen (DLL-Datei)
- Eclipse für Echtzeit-Anwendungen (SO-Datei)

Microsoft Visual Studio als Entwicklungsumgebung

Nutzen Sie Microsoft Visual Studio. Um Ihnen die Entwicklung einer ODK-Anwendung zu erleichtern, wird bei der Installation von ODK 1500S ein Template für ein Microsoft Visual Studio-Projekt mitgeliefert. Das ODK-Template finden Sie, beim Anlegen eines neuen Projekts, unter dem Eintrag "Visual C++".

Eclipse als Entwicklungsumgebung

Nutzen Sie Eclipse. Um Ihnen die Entwicklung einer ODK-Anwendung zu erleichtern, wird bei der Installation von ODK 1500S ein Template für ein Eclipse-Projekt mitgeliefert. Das ODK-Template finden Sie im Ordner "ODK 1500S Templates".

2.3 Prinzipielles Vorgehen

Die nachfolgenden Kapitel beschreiben die Entwicklungsaufgaben und Vorgehensweise für die Entwicklung und Ausführung einer ODK-Anwendung:

- Entwicklung einer ODK-Anwendung für die Windows-Umgebung (Seite 17)
- Entwicklung einer ODK-Anwendung für die Echtzeit-Umgebung (Seite 51)

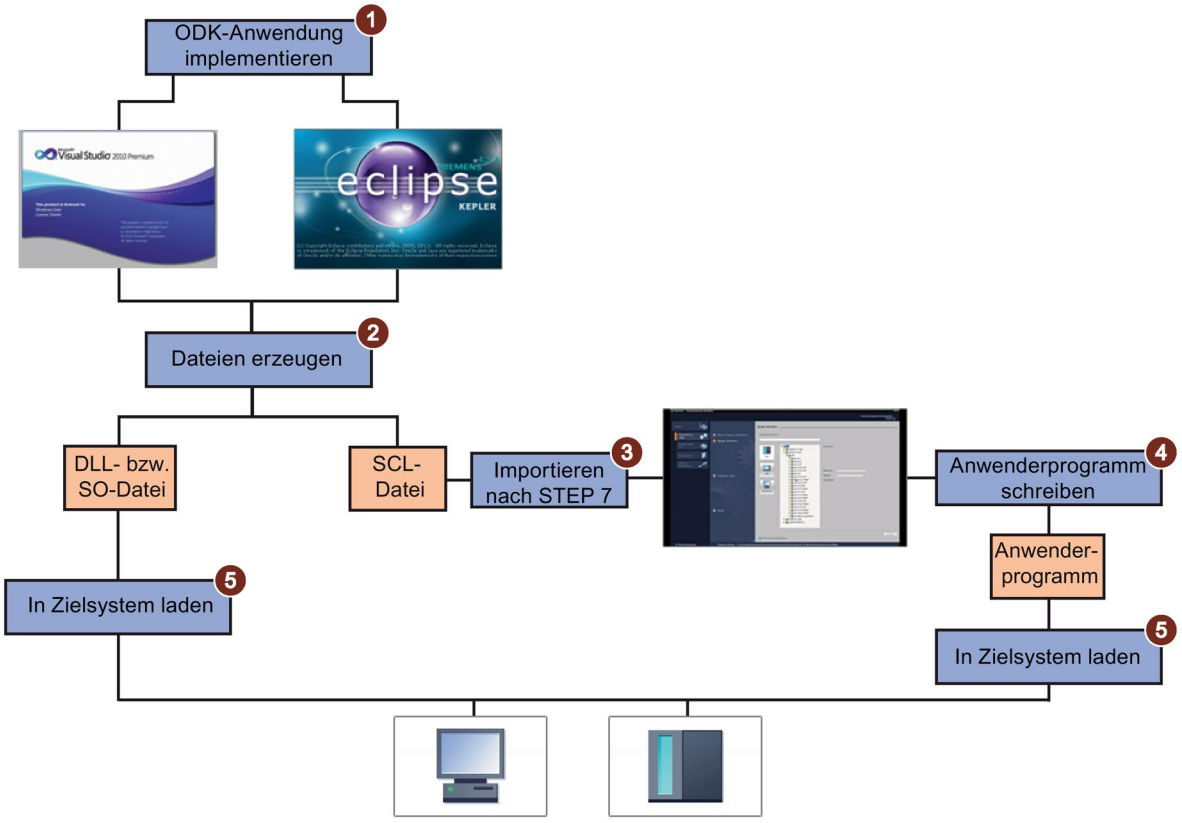


Bild 2-3 Überblick über die Entwicklungsschritte

Überblick über die Entwicklungsschritte

Um eine ODK-Anwendung zu entwickeln und auszuführen, gehen Sie folgendermaßen vor:

1. Implementieren Sie Ihre Funktion in Microsoft Visual Studio für Windows-Anwendungen (DLL-Datei) oder in Eclipse für Echtzeitanwendungen (SO-Datei).
2. Erzeugen Sie die DLL- bzw. SO-Datei und die SCL-Datei.
3. Importieren Sie die SCL-Datei nach STEP 7.
4. Schreiben Sie in STEP 7 ihr Anwenderprogramm.
5. Laden Sie das Anwenderprogramm in die CPU und die DLL- bzw. SO-Datei in das Zielsystem.

Ergebnis

Ihre ODK-Anwendung ist auf das Zielsystem geladen und wird durch das Anwenderprogramm in STEP 7 geladen und ausgeführt.

Installation

3

3.1 Systemanforderungen

Voraussetzungen

Damit Sie ODK nutzen können, muss Ihr PC die folgenden Systemanforderungen erfüllen:

Kategorie	Voraussetzung
Betriebssystem	• Microsoft Windows 7, 64-Bit • Microsoft Windows 8, 64-Bit • Microsoft Windows 10, 64-Bit
Prozessor und Speicher	PC-System: • mind. Systeme mit Intel Core i5-Prozessor • ab 1,2 GHz • Mindestens 4 GB RAM-Speicher
Massenspeicher	1,6 GB freien Speicherplatz auf der Festplatte C:\ für die Vollinstallation. **Hinweis:** Die Setup-Dateien werden nach abgeschlossener Installation wieder gelöscht.
Bedienerschnittstelle	Farbmonitor, Tastatur und Maus oder ein anderes Zeigegerät (optional), die von Windows unterstützt werden.
SIMATIC Software	• SIMATIC STEP 7 Professional (TIA Portal) ab V14
Unterstützte Steuerungen	alle ODK-unterstützenden SIMATIC CPUs (siehe nächste Tabelle)
Weitere Software	Nicht im Lieferumfang enthalten: • Java Runtime 32-Bit ab V1.6 (für Eclipse) • Microsoft Visual Studio C++ 2010 SP1 • Microsoft Visual Studio C++ 2012 • Microsoft Visual Studio C++ 2013 • Microsoft Visual Studio express C++ 2013 • Microsoft Visual Studio C++ 2015 • Microsoft Visual Studio express C++ 2015 Microsoft-Entwicklungstool: Download Center (http://www.microsoft.com/de-de/download/developer-tools.aspx)

ODK 1500S V2.0 ist mit folgenden Geräten kompatibel (die Unterstützung von ladbaren Funktionsbibliotheken ist geräteabhängig):

	DLL (Windows)	SO (Echtzeit)
CPU 1505SP (F) V2.0	ja	ja
CPU 1507S (F) V2.0	ja	ja
CPU 1518-4 PN/DP ODK (F) V2.0	nein	ja

3.2 ODK installieren

Um ODK zu installieren, legen Sie die Installations-DVD ein. Befolgen Sie die Anweisungen des Setup-Programms.

Wenn das Setup-Programm nicht automatisch gestartet wird, starten Sie die Datei "Start.exe" manuell von der Installations-DVD mit einem Doppelklick.

Voraussetzung

Für diesen Vorgang benötigen Sie Administratorrechte.

Der parallele Betrieb von unterschiedlichen ODK-Versionen auf einem PC ist möglich. Wenn auf dem PC bereits die zu installierende Version von ODK installiert ist, müssen Sie diese zuerst deinstallieren oder eine Reparaturinstallation durchführen.

Hinweis

Anwendungen schließen vor Reparaturinstallation/Deinstallation

Schließen Sie alle Anwendungen (insbesondere ODK-bezogene Anwendungen), bevor Sie die Reparaturinstallation/Deinstallation durchführen.

Vorgehen

Wenn Sie die Entwicklungsumgebung Microsoft Visual Studio nutzen möchten, wird empfohlen, diese vor ODK zu installieren.

Um ODK zu installieren, gehen Sie folgendermaßen vor:

1. Starten Sie die Datei "Start.exe" manuell von der Installations-DVD mit einem Doppelklick.
2. Folgen Sie den Anweisungen des Installations-Wizards.

Ergebnis

Die Installation wird beendet. Während des Installationsvorganges wurden standardmäßig alle Produktsprachen installiert. Durch die Installation wird ein Eintrag im Startmenü von Windows angelegt.

Das Setup-Programm installiert folgende Komponenten:

- "Eclipse" für die Entwicklung von ODK-Anwendungen für die Echtzeitumgebung
- ODK-Template für Visual Studio
- Code-Generator
- Online-Hilfen

3.3 ODK-Template nachträglich in Visual Studio integrieren

In ein bereits installiertes Visual Studio wird bei der ODK-Installation das ODK-Template automatisch installiert. Wenn Visual Studio nachträglich installiert wird, haben Sie folgende Möglichkeiten, um das ODK-Template zu integrieren:

- Führen Sie eine Reparaturinstallation von ODK durch.
- Führen Sie die Integration manuell durch. Rufen Sie dazu im Ordner "bin" Ihrer ODK-Installation die Datei "ODK_VSTemplate_Integration.exe" auf.

Ergebnis

Das ODK-Template wird für Visual Studio installiert. Sie finden dieses unter der entsprechenden Programmiersprache.

3.4 ODK deinstallieren

Vorgehen

Um ODK auf Ihrem PC zu deinstallieren, gehen Sie folgendermaßen vor:

1. Schließen alle gestarteten Programme, insbesondere ODK-bezogene Anwendungen.
2. Wählen Sie im Menü "Systemsteuerung > Programme > Programm deinstallieren" den Eintrag "SIMATIC ODK 1500S".
3. Wählen Sie den Kontextmenübefehl "Deinstallieren".

   Ein Dialog zur Deinstallation wird geöffnet.
4. Folgen Sie den weiteren Schritten zur Deinstallation.

Ergebnis

ODK wird deinstalliert.

Entwicklung einer ODK-Anwendung für die Windows-Umgebung

4

4.1 ODK-Anwendung erstellen

4.1.1 Voraussetzungen

Die Entwicklungsumgebung Microsoft Visual Studio ist nicht im Lieferumfang von ODK enthalten.

Das Download Center für Microsoft-Entwicklungstools finden Sie im Internet (http://www.microsoft.com/de-de/download/developer-tools.aspx).

4.1.2 Projekt anlegen

Um Ihnen die Entwicklung einer ODK-Anwendung zu erleichtern, wird bei der Installation von ODK 1500S ein ODK-Template für ein ODK-Projekt in Visual Studio integriert. Das Template unterstützt 32- und 64-Bit-Anwendungen.

Vorgehen

Um mit Hilfe des ODK-Templates ein ODK-Projekt in Microsoft Visual Studio zu erstellen, gehen Sie folgendermaßen vor:

1. Starten Sie Microsoft Visual Studio als Entwicklungsumgebung.
2. Wählen Sie im Menü "File > New" den Befehl "Project..."

   Der Dialog "New Project" wird geöffnet.

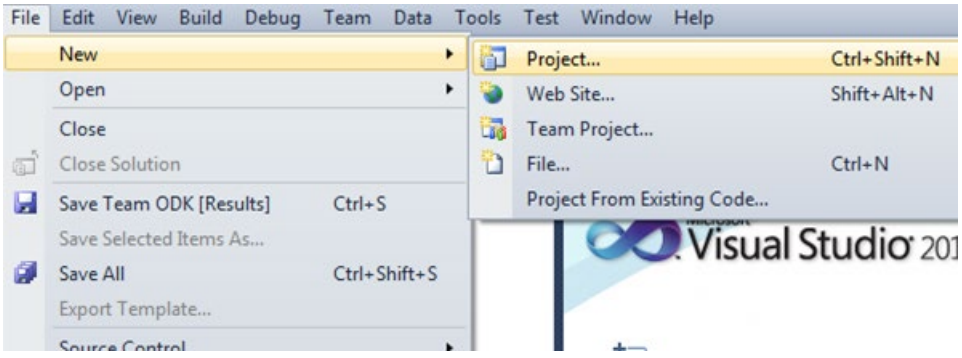


Bild 4-1 Neues Projekt anlegen in Visual Studio

3. Wählen Sie Ihre bevorzugte Programmiersprache und das zugehörige ODK-Template aus.
4. Vergeben Sie einen Projektnamen.
5. Bestätigen Sie mit "OK".

Ergebnis

Das ODK-Projekt wird mit Hilfe des ODK-Templates erstellt und legt folgende Projekteinstellungen fest:

- Projekteinstellungen für die Generierung der DLL-Datei
- Automatisiert die Erzeugung der DLL- und SCL-Datei

Das ODK-Template richtet standardmäßig die folgende Dateistruktur ein:

Ordner / Datei			Beschreibung
<Projektpfad>			
	<Projekt>.rc		
	<Projekt>.cpp		Funktions-Code: Diese Datei hat immer das Suffix CPP, unabhängig davon, ob Sie ein C- oder C++-Projekt erstellt.
	dllmain.cpp		Implementierung der Datei "dllmain"
	def		
		<Projekt>.odk	ODK-Interface-Beschreibung
		<Projekt>.scl.additional	S7-Blöcke, die an die Datei <Project>.scl angehängt werden. Die Datei ist zwar nicht Teil der Projektvorlage, aber der Code-Generator bearbeitet die Datei.
	STEP7		**Dateien aus diesem Ordner dürfen nicht editiert werden!**
		<Projekt>.scl	S7-Bausteine
	cg_src_priv		**Dateien aus diesem Ordner dürfen nicht editiert werden!**
		ODK_Types.h	Definition der ODK-BaseTypes
		ODK_Functions.h	Funktions-Prototypen
		ODK_Execution.cpp	Implementierung der Methode "Execute"
	src_odk_helpers		**Dateien aus diesem Ordner dürfen nicht editiert werden!**
		ODK_CpuReadData.h	Definition: Hilfe-Funktionen zum Lesen der Datenbausteine
		ODK_CpuReadData.cpp	Implementierung: Hilfe-Funktionen zum Lesen der Datenbausteine
		ODK_CpuReadWriteData.h	Definition: Hilfe-Funktionen zum Lesen/Schreiben der Datenbausteine
		ODK_CpuReadWriteData.cpp	Implementierung: Hilfe-Funktionen zum Lesen/Schreiben der Datenbausteine
		ODK_StringHelper.h	Definition: Hilfe-Funktionen S7-Strings/W-Strings
		ODK_StringHelper.cpp	Implementierung: Hilfe-Funktionen S7-Strings/W-Strings
	debug		
		<Projekt>.dll	ODK-Application Binary (Debug-Version)
	release		
		<Projekt>.dll	ODK-Application Binary (Release-Version)

Das ODK-Template unterstützt folgende Anwendungen:

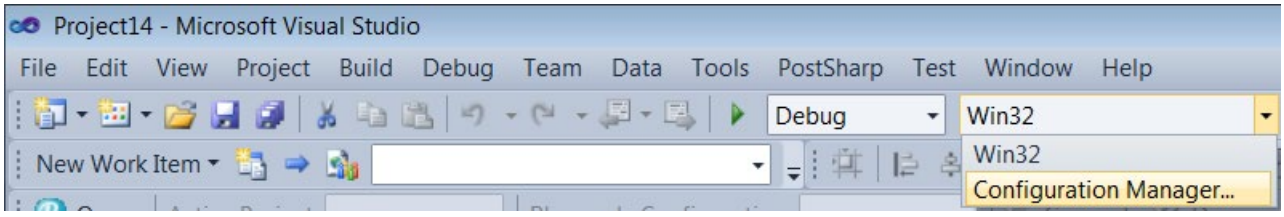
Konfiguration und Plattform	Visual Studio Version älter als 2015	Visual Studio 2015
Debug Win32	ja	ja
Release Win32	ja	ja
Debug x64	manuell anzulegen (Seite 20)	ja
Release x64	manuell anzulegen (Seite 20)	ja

4.1.2.1 ODK-Projekt erstellen mit Visual Studio Version älter als 2015

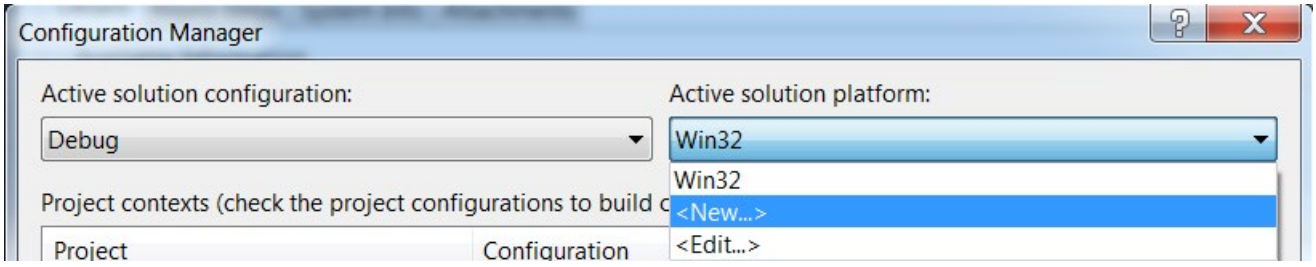
Vorgehen

Um ein ODK-Template für eine x64-Plattform mit einer Visual Studio Version älter als 2015 zu erstellen, gehen Sie folgendermaßen vor:

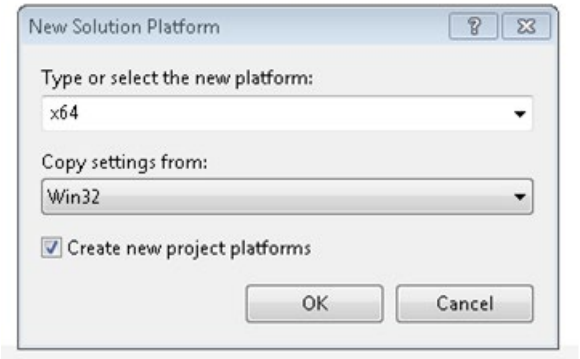
1. Öffnen Sie den "Configuration Manager".



2. Erstellen Sie eine x64-Plattform.

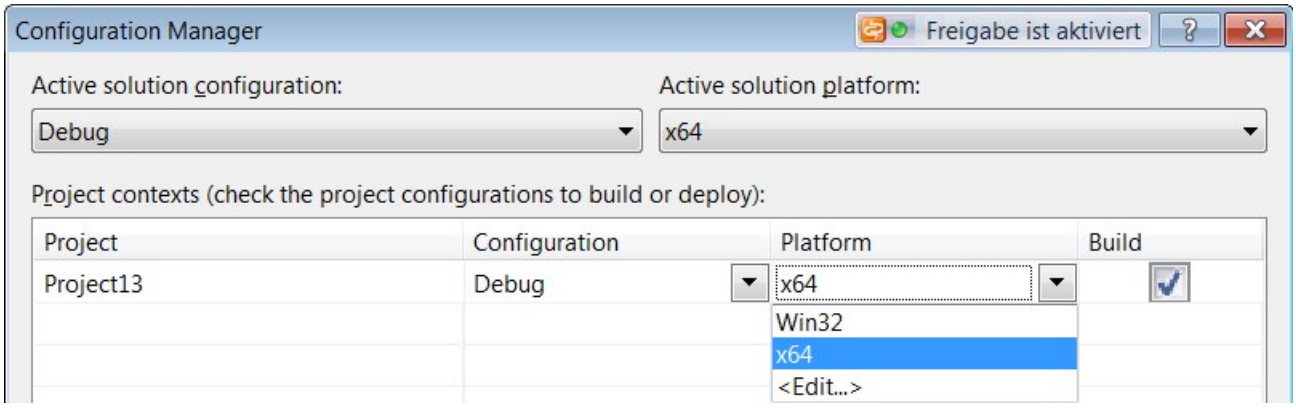


Der Dialog "New Solution Platform" öffnet sich.



Wählen Sie unter "Copy settings from:" die Auswahl "Win32" aus.

3. Definieren Sie eine Solution-Konfiguration für eine x64-Plattform.



4. Wählen Sie unter "Active solution configuration" die Auswahl "Debug" oder "Release" und unter "Platform" die Auswahl "x64" aus.

4.1.3 ODK-Anwendung generieren

Die Generierung der Projektdaten ist in zwei automatisierte Schritte unterteilt.

- **Pre-Build**: Erzeugung der standardmäßig angelegten Dateien auf Basis der geänderten Datei <Projekt>.odk
- **Build**: Generierung der DLL-Datei

Vorgehen

Um die Projektdaten zu generieren, gehen Sie folgendermaßen vor:

1. Speichern Sie alle editierten Dateien.
2. Wählen Sie im Menü "Build" den Befehl "Build Solution".

Hinweis

Die Projektdaten werden nur generiert, wenn die Dateien verändert wurden.

Ergebnis

Die Generierung der Projektdaten wird gestartet. Die automatisch erzeugten Dateien werden im Dateisystem abgelegt.

- DLL-Datei: Projekt-Verzeichnis\<Projekt>\<BuildConfiguration>\<Projekt>.dll
- SCL-Datei: Projekt-Verzeichnis\<Projekt>\STEP7\<Projekt>.scl

4.1.4 Ablaufeigenschaften einer ODK-Anwendung definieren

Als nächsten Schritt definieren Sie die Schnittstellenbeschreibung der ODK-Anwendung in der Datei <Projekt>.odk. Die Datei enthält folgende Elemente:

- Kommentare
- Parameter
- Definitionen von Funktionen und Strukturen

Vorgehen

Um die Schnittstellenbeschreibung in der Datei <Projekt>.odk zu definieren, gehen Sie folgendermaßen vor:

1. Öffnen Sie die Datei <Projekt>.odk.
2. Ändern Sie die Elemente abhängig von Ihren Anforderungen.

Beschreibung der Elemente

Kommentare

Sie können Kommentare für Erläuterungen verwenden.

Parameter

Die Definition der Parameter muss innerhalb einer Code-Zeile erfolgen.

```
<ParameterName>=<Value> // optional comment
```

Die Interface-Datei unterstützt folgende Parameter:

Parameter	Wert	Beschreibung
Context	user	Definiert, dass die ODK-Anwendung im Kontext eines Windows-Nutzers (Seite 23) geladen wird.
	system	Definiert, dass die ODK-Anwendung im Kontext des Windows-Systems (Seite 23) geladen wird.
STEP7Prefix	<String>	Beschreibt die Zeichenkette, die Ihren Funktionen vorangestellt und nach dem Import der SCL-Datei in STEP 7 dargestellt wird. Erlaubt sind folgende Zeichen: {A...Z, a…z, 1…9, -, _} Umlaute sind nicht erlaubt. Standardmäßig ist der Projektname ohne Leerzeichen eingetragen.

Hinweis

Leerzeichen im Projektnamen

Bei dem STEP7-Prefix werden die Leerzeichen durch einen Unterstrich ersetzt.

4.1.5 Umfeld zum Laden oder Ausführen der ODK-Anwendung

Wenn die SCL-Datei als externe Quelle nach STEP 7 importiert wird, werden die ODK-Anweisungen im gewählten Verzeichnis in STEP 7 erstellt. Die ODK-Anweisungen geben Ihnen nach der Programmierung und dem erstmaligen Laden die Möglichkeit, Ihre ODK-Anwendung unabhängig vom STEP 7-Anwenderprogramm zu steuern. Sie können bis zu 32 ODK-Anwendungen laden.

Je nachdem ob Sie die ODK-Anwendung für ein 32- oder 64-Bit-System erstellt haben, wird diese in einem 32- oder 64-Bit ODK_Host-Prozess geladen.

Sie können für Ihre ODK-Anwendung eine der beiden Kontexte wählen:

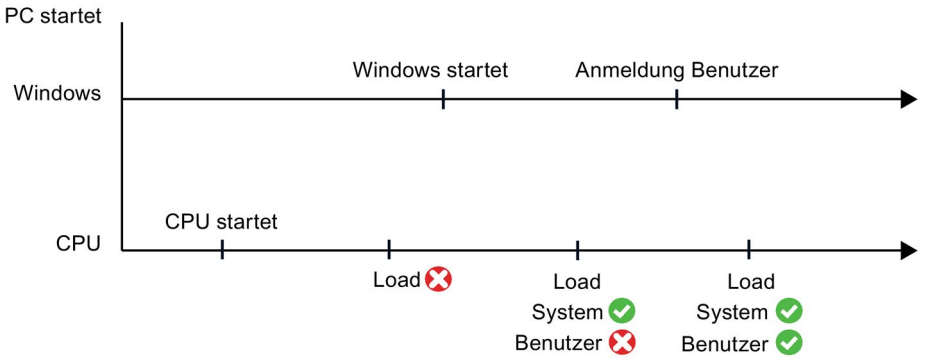
- **Kontext "System"**

  Windows ist gestartet, es kann ein Benutzer angemeldet sein

- **Kontext "Benutzer"**

  Windows ist gestartet, es muss ein Benutzer angemeldet sein

Die folgende Grafik zeigt Ihnen, wann eine ODK-Anwendung abhängig vom Kontext geladen werden kann.



Kontext "System"

Um die ODK-Anwendung im System-Kontext (Session 0) zu nutzen, ändern Sie folgende Code-Zeile in der Datei <Projekt>.odk:

```
Context=system
```

Im System-Kontext wird die ODK-Anwendung auch ohne die Anmeldung eines Windows-Benutzers ausgeführt. Die ODK-Anwendung kann somit nicht aktiv mit Oberflächen-Elementen wie z. B. Melde-Dialoge gesteuert werden.

Kontext "Benutzer"

Um die ODK-Anwendung im Benutzer-Kontext zu nutzen, ändern Sie folgende Code-Zeile in der Datei <Projekt>.odk:

```
Context=user
```

Wenn Sie die ODK-Anwendung im Benutzer-Kontext laden, wird sie automatisch entladen, sobald sich der Benutzer von Windows abmeldet. Die ODK-Anwendung kann aktiv über Windows-Oberflächen-Elemente wie z. B. Melde-Dialoge gesteuert werden und bietet Zugriff auf weitere Ressourcen auf der Windows-Umgebung.

Wenn mehrere Benutzer bei Windows angemeldet sind, lädt oder entlädt die ODK-Anwendung für den Benutzer, der die aktuellen Bildschirmrechte besitzt, bis dieser sich bei Windows abmeldet.

4.1.6 Funktionen und Strukturen einer ODK-Anwendung definieren

Funktionen

Funktionen werden durch folgende allgemeine Code-Zeilen definiert:

```
ODK_RESULT <FunctionName>
([<InOut-Identifier>] <DataType> <VariableName>, …);
```

Die Datei <Projekt>.odk enthält standardmäßig eine beispielhafte Funktionsbeschreibung. Diese Beschreibung können Sie verändern und/oder weitere Funktionsbeschreibungen hinzufügen.

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

Syntax-Regeln für Funktionen

Folgenden Syntax-Regeln gelten für Funktionen innerhalb der Datei <Projekt>.odk:

- Beachten Sie beim Funktionsnamen die Groß- und Kleinschreibung.
- Funktionsdefinitionen können Sie in mehrere Zeilen aufgeteilen.
- Beenden Sie eine Funktionsdefinition durch ein Semikolon.
- TAB und LEERZEICHEN sind erlaubt.
- Definieren Sie einen Variablenname in einer Funktion nicht doppelt.
- Verwenden Sie keine Schlüsselwörter für die genutzte Programmiersprache (z. B. "INT" als Parametername)
- Nutzen Sie ODK_RESULT nur für die Rückgabewerte der Funktion.
- Variablennamen müssen mit einem Buchstaben oder einem Unterstrich beginnen.
- Unzulässige Funktionsnamen werden beim Generieren in der Entwicklungsumgebung angezeigt.
- Folgende Namen sind in Kombination von *<STEP7Prefix>* und <Funktionsname> nicht erlaubt: ODK_Load, ODK_Unld, ODK_ExcA, ODK_ExcS

<FunctionName>

Funktionsnamen gelten mit den Syntax- und Zeichenbeschränkungen der verwendeten Programmiersprache.

<InOut-Identifier>

Es gibt drei definierte InOut-Identifiers. Nutzen Sie diese in folgender Reihenfolge: [IN], [OUT], [INOUT]

- [IN]: Definiert eine Eingangs-Variable. Die Variable wird zur Funktion kopiert, wenn diese aufgerufen wird. Diese ist konstant und kann nicht verändert werden.
- [OUT]: Definiert eine Ausgangs-Variable. Die Variable wird zurückkopiert, nachdem die Funktion beendet wurde.
- [INOUT]: Definiert eine Eingangs- und Ausgangs-Variable. Die Variable wird zur Funktion kopiert, wenn diese aufgerufen wird. Diese ist nicht konstant und kann verändert werden. Die Variable wird zurückkopiert, nachdem die Funktion beendet wurde.

<DataType>

Der Datentyp definiert den Typ einer Variablen. Die folgenden Tabellen definieren die möglichen Datentypen und ihre Darstellungsweise in C++ oder STEP 7:

- **Elementare Datentypen:**

ODK-Datentyp	SIMATIC-Datentyp	C++-Datentyp	Beschreibung
ODK_DOUBLE	LREAL	double	64-Bit Floating Point, IEEE 754
ODK_FLOAT	REAL	float	32-Bit Floating Point, IEEE 754
ODK_INT64	LINT	long long	64-Bit signed integer
ODK_INT32	DINT	long	32-Bit signed integer
ODK_INT16	INT	short	16-Bit signed integer
ODK_INT8	SINT	char	8-Bit signed integer
ODK_UINT64	ULINT	unsigned long long	64-Bit unsigned integer
ODK_UINT32	UDINT	unsigned long	32-Bit unsigned integer
ODK_UINT16	UINT	unsigned short	16-Bit unsigned integer
ODK_UINT8	USINT	unsigned char	8-Bit unsigned integer
ODK_LWORD	LWORD	unsigned long long	64-Bit Bitstring
ODK_DWORD	DWORD	unsigned long	32-Bit Bitstring
ODK_WORD	WORD	unsigned short	16-Bit Bitstring
ODK_BYTE	BYTE	unsigned char	8-Bit Bitstring
ODK_BOOL	BOOL	unsigned char	1-Bit bitstring, verbleibende Bits (1..7) sind leer
ODK_LTIME	LTIME	unsigned long long	64-Bit Dauer in Nanosekunden
ODK_TIME	TIME	unsigned long	32-Bit Dauer in Millisekunden
ODK_LDT	LDT	unsigned long long	64-Bit Datum und Uhrzeit des Tages in Nanosekunden
ODK_LTOD	LTOD	unsigned long long	64-Bit Uhrzeit des Tages in Nanosekunden seit Mitternacht
ODK_TOD	TOD	unsigned long	32-Bit Uhrzeit des Tages; in Millisekunden seit Mitternacht
ODK_WCHAR	WCHAR	wchar_t	16-Bit-Zeichen
ODK_CHAR	CHAR	char	8-Bit-Zeichen

- **Komplexe Datentypen:**

ODK-Datentyp	SIMATIC-Datentyp	C++-Datentyp	Beschreibung
ODK_DTL	DTL	struct ODK_DTL	Struktur für Datum und Uhrzeit
ODK_S7WSTRING	WSTRING	unsigned short	Zeichen-String (16-Bit-Zeichen) mit Länge max. und act. (2xUINT)
ODK_S7STRING	STRING	unsigned char	Zeichen-String (8-Bit-Zeichen) mit Länge max. und act. (2xUSINT)
ODK_CLASSIC_DB	VARIANT	struct ODK_CLASSIC_DB	Klassik-DB (global oder basierend auf UDT)
[ ]	ARRAY	[ ]	Bereich von gleichen Datentypen. Sie können alle Datentypen außer ODK_CLASSIC_DB als Array nutzen.

- **Benutzerdefinierte Datentypen:**

  Benutzerdefinierte Datentypen (UDT) beinhalten strukturierte Daten, insbesondere die Namen und die Datentypen dieser Komponente und deren Reihenfolge.

  Ein benutzerdefinierter Datentyp kann in der ODK-Schnittstellenbeschreibung mit dem Schlüsselwort "ODK_STRUCT" definiert werden.

  Beispiel

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

  Die folgenden Syntaxregeln gelten für die Struktur:

  – Sie können die Struktur in mehrere Zeilen aufteilen.
  – Die Strukturdefinition muss ein Semikolon abschließen.
  – Tabulatoren und Leerzeichen zwischen den Elementen sind in einer beliebigen Anzahl erlaubt.
  – Sie dürfen keine Schlüsselwörter für die erzeugte Sprache verwenden (z. B. "int" als Variablennamen).

  Sie können innerhalb einer Struktur weitere Strukturen anlegen.

<StructName>

Strukturnamen gelten mit den Syntax- und Zeichenbeschränkungen der Programmiersprache und wie für Variablendefinitionen in STEP7 definiert.

In STEP7 wird der Strukturname durch den STEP7-Prefix erweitert.

<VariableName>

Variablennamen gelten mit den Syntax- und Zeichenbeschränkungen der Programmiersprache.

Beispiel

Das folgende Code-Beispiel erläutert Ihnen die Definitionen von Funktionen und Strukturen. Sortieren Sie die Parameter nach: IN, OUT, INOUT.

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

4.1.6.1 Verwendung von ODK_CLASSIC_DB als Parameter

Der Datentyp ODK_CLASSIC_DB darf nur mit dem InOut-Identifier [IN] und [INOUT] benutzt werden. Wird ein Parameter des Datentyps ODK_CLASSIC_DB mit dem InOut-Identifier [IN] oder [INOUT] verwendet, darf kein weiterer Parameter, egal welchen Datentyps, mit dem gleichen InOut-Identifier verwendet werden.

Beispiel

```
// INTERFACE
...
// OK:
ODK_RESULT MyFunc1([IN] ODK_CLASSIC_DB myDB);
ODK_RESULT MyFunc2([IN] ODK_CLASSIC_DB myDB1, [INOUT] ODK_CLASSIC_DB
myDB2);
//
// NOT OK (Code Generator will throw an error):
// ODK_CLASSIC_DB nicht für [OUT] zulässig
ODK_RESULT MyFunc3([OUT] ODK_CLASSIC_DB myDB);
// wenn ODK_CLASSIC_DB für [IN] verwendet, darf kein weiterer [IN]
Parameter in dieser
// Funktion definiert werden
ODK_RESULT MyFunc4([IN] ODK_CLASSIC_DB myDB, [IN] ODK_INT32 myint);
```

Anwendungsbeispiel für C++

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

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

    return myInt1 + myInt2;
}
```

Um innerhalb einer Anwenderfunktion auf den Datentyp ODK_CLASSIC_DB zugreifen zu können, stehen Ihnen die Helper-Funktionen (Seite 99) der folgenden Klassen zur Verfügung:

- Klasse "CODK_CpuReadData"
- Klasse "CODK_CpuReadWriteData"

4.1.6.2 Handhabung von Strings

Für Strings (String oder WString) können Sie eine maximale Länge definieren. Definieren Sie die maximale Zeichenanzahl in eckigen Klammern direkt nach dem Datentyp:

- ODK_S7STRING[30] oder
- ODK_S7WSTRING[1000]

Ohne eine Beschränkung hat ein String standardmäßig eine Länge von maximal 254 Zeichen.

Um innerhalb einer Anwenderfunktion auf die Datentypen ODK_S7STRING bzw. ODK_S7WSTRING zugreifen zu können, stehen Ihnen die String-Helper-Funktionen (Seite 99) zur Verfügung.

Beispiel

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

Wenn Sie [INOUT] nutzen, können Sie den String mit einer unterschiedlichen Länge als [INOUT] des Funktionsbausteins in STEP 7 festsetzen.

4.1.6.3 Definition der Datei <Projekt>.odk

Anhand der gewählten Parameter in der Datei <Projekt>.odk werden die Funktionsprototypen und Funktionsblöcke generiert. Definieren Sie dafür die Datei <Projekt>.odk.

Standardmäßig enthält die Datei <Projekt>.odk folgenden Inhalt:

- Description

  Die möglichen Datentypen, die für die Schnittstelle benutzt werden, sind in den Kommentarzeilen beschrieben. Das erleichtert Ihnen die Definition des richtigen Variablentyps für Ihre Aufgabe.

- Context=user

  Die ODK-Anwendung wird im Kontext "Benutzer" geladen. Sie können den Parameter in Context=system ändern.

- STEP7Prefix="<Projekt>"

  Stellt den Funktionen der ODK-Anwendung einen String für die SCL-Generierung voran. Der String wird in STEP 7 sichtbar. Sie können den Parameter ändern. Die String-Länge des Präfix inklusive Funktionsnamen darf eine Zeichenlänge von 125 Zeichen nicht überschreiten (z. B. ODK_App_SampleFunction)

- "SampleFunction" Funktionsdefinition

  Diese Standardfunktion können Sie in der Datei <Projekt>.odk beliebig ändern und weitere Funktionen hinzufügen. Die String-Länge darf eine Zeichenlänge von 125 Zeichen nicht überschreiten. Die zugehörige Funktion befindet sich in der CPP-Datei.

Beispiel

```
//INTERFACE
Context=user
STEP7Prefix=ODK_App

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

```
*  ODK_LTOD        LTOD     64 bit time of day in nanoseconds
*                           since midnight
*  ODK_TOD         TOD      32 bit time of day in milliseconds
*                           since midnight
*  ODK_CHAR        CHAR     8 bit character
*  ODK_WCHAR       WCHAR    16 bit character
* Complex Datatypes:
*  ODK_DTL         DTL      structure for date and time
*  ODK_S7STRING    STRING   character string with 8 bit characters
*  ODK_CLASSIC_DB  VARIANT  classic DB (global or based on UDT
*                           "optimized block access" must be
unchecked)
*  ODK_S7WSTRING   WSTRING  character string with 16 bit characters
*  []              ARRAY    field of this datatype
* User Defined Datatype:
*  ODK_STRUCT      UDT      user defined structure
* Return Datatype:
*  ODK_RESULT      0x0000-0x6FFF function succeeded
*                               (ODK_SUCCESS = 0x0000)
*                  0xF000-0xFFFF function failed
*                               (ODK_USER_ERROR_BASE = 0xF000)
*/

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

4.1.6.4 Datei <Projekt>.odk modifizieren

Das folgende Beispiel zeigt Ihnen wie Sie die Datei <Projekt>.odk Ihren Bedürfnissen anpassen.

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

ODK_RESULT GetString ([OUT]    ODK_S7STRING myString);

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

4.1.6.5 Kommentare

Kommentare werden mit einem doppelten Schrägstrich gestartet "//" und enden automatisch am Ende der Zeile.

Alternativ können Sie Kommentare durch /* *<comment>* */ abgrenzen, dadurch sind neue Zeilen in einem Kommentar möglich. Zeichen nach dem Ende der Kommentarkennung "*/" werden vom Code-Generator weiterverarbeitet.

Kommentare für Funktionen und Strukturen

Kommentare zu Funktionen und Strukturen platzieren Sie direkt vor den Funktionen/Strukturen.

Diese Kommentare werden an die Dateien ODK_Functions.h und <Project>.scl übergeben.

In der Datei <Project>.scl werden die Kommentare in die Blockeigenschaften und in den Code-Bereich der Funktion dupliziert.

Beachten Sie folgende Regeln:

- Kommentare für Funktionen und Strukturen müssen direkt vor den Funktionen/Strukturen stehen (ohne Leerzeile).
- Das Ende des Kommentars steht vor dem Schlüsselwort ODK_RESULT oder ODK_STRUCT.
- Sie können beide Kennungen "//" und "/* */" verwenden, jedoch nicht in Kombination innerhalb eines Kommentars.

Beispiel

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

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

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

Kommentare für Variablen in Funktionen und Strukturen

Kommentare für Funktions- und Strukturvariablen werden direkt vor oder hinter den Variablen platziert.

Diese Kommentare werden an die Dateien ODK_Functions.h und <Project>.scl übergeben.

Folgende Regeln gelten für Kommentare **vor** Variablen:

- Kommentare müssen direkt vor der Variablen stehen (ohne Leerzeile)
- Das Ende des Kommentars ist der <InOut-Identifier> der Variablen

Folgende Regeln gelten für Kommentare **nach** Variablen:

- Kommentare müssen nach dem Variablen-Namen stehen (ohne Leerzeile)

Folgende allgemeine Regeln gelten für Kommentare für Variablen:

- Sie können beide Kennungen "//" und "/* */" verwenden, jedoch nicht in Kombination innerhalb eines Kommentars.
- In der Header-Datei wird die gleiche Kommentar-Kennung benutzt ("//" oder "/* */).

Beispiel

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

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

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

4.1.7 Funktionen implementieren

4.1.7.1 Grundsätzliche Anmerkungen

In diesem Kapitel erhalten Sie eine Übersicht über grundsätzliche Themen im Bezug auf die Implementierung von Funktionen in einer Windows-Umgebung.

- Funktionsaufruf ist zeitlich nicht eingeschränkt, da die Funktion asynchron aufgerufen wird
- Traces sind über OutputDebugString-Anweisungen möglich
- Alle asynchronen ODK-Funktionen werden gleichberechtigt ausgeführt - unabhängig von der Priorität der OBs
- Die vollständige Windows-API (Application Programming Interface) und C++-Runtime-Bibliothek steht zur Verfügung

4.1.7.2 Callback-Funktionen

Das ODK-Projekt enthält eine CPP-Datei (**Execute-Datei: <**Projekt**>**.cpp), um Ihre Funktionen zu definieren. Diese CPP-Datei enthält standardmäßig vorausgefüllte Funktionen. Diese müssen nicht zwangsläufig mit zusätzlichem Benutzer-Code befüllt werden, um nutzbar zu sein. Die Funktionen dürfen aber unter keinen Umständen gelöscht werden.

Die leere Funktion hat folgenden Code (am Beispiel der Funktion "OnLoad()"):

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

Sie können die folgenden Funktionen in der CPP-Datei definieren:

- OnLoad(): wird nach dem Ladevorgang der ODK-Anwendung aufgerufen
- OnUnload(): wird vor dem Entladevorgang der ODK-Anwendung aufgerufen
- OnRun(): wird beim Übergang der CPU in den Betriebszustand RUN und nach der Funktion OnLoad() aufgerufen
- OnStop(): wird beim Übergang der CPU in den Betriebszustand STOP und vor der Funktion OnUnload() aufgerufen

Die folgende Tabelle gibt Ihnen einen Überblick über die unterschiedlichen Aktionen, um die Callback-Funkionen aufzurufen:

aktueller Betriebszustand	neuer Betriebszustand	Anwenderaktion	ODK-Aktion
RUN	RUN	ODK_Load	1. OnLoad() 2. OnRun()
STOP	RUN	ODK_Load in Startup-OB (z. B. OB100)	1. OnLoad() 2. OnRun()
RUN	STOP	*<schon geladen>*	OnStop()
STOP	RUN	*<schon geladen>*	OnRun()
RUN	RUN	ODK_Unload	1. OnStop() 2. OnUnload()
RUN	SHUTDOWN / MRES	*<schon geladen>*	OnStop()
any	*any*	*<schon geladen>* ODK-Host beenden	1. OnStop() (optional, falls nicht bereits ausgeführt) 2. OnUnload()

Funktion "OnLoad()" und "OnUnload()"

Die Funktionen haben einen Rückgabewert des Typs "ODK_RESULT" und geben typischerweise Auskunft über den Status des Werts "ODK_SUCCESS".

Die folgenden Rückgabewerte sind möglich:

Rückgabewert für "ODK_RESULT"	Beschreibung
ODK_SUCCESS = 0x0000	Rückgabewert bei einer erfolgreichen Ausführung der Funktion "OnLoad()" oder "OnUnload()"
0x0001 – 0xEFFF	Ungültige Werte (systemintern)
0xF000 – 0xFFFF ODK_USER_ERROR_BASE = 0xF000	Sie können Ihre eigenen Fehlerwerte definieren. Für die Funktion "OnLoad()" wird der Ladevorgang abgebrochen und die ODK-Anwendung entladen. Für die Funktion "OnUnload()" wird die ODK-Anwendung innerhalb des angegebenen Wertebereichs trotzdem entladen.

Funktion "OnRun()" und "OnStop()"

Die Funktionen haben einen Rückgabewert des Typs "ODK_RESULT" und geben typischerweise Auskunft über den Status des Werts "ODK_SUCCESS".

Die folgenden Rückgabewerte sind möglich:

Rückgabewert für "ODK_RESULT"	Beschreibung
ODK_SUCCESS = 0x0000	Rückgabewert bei einer erfolgreichen Ausführung der Funktion "OnRun()" oder "OnStop()"
0x0001 – 0xFFFF	Es ist keine direkte Rückmeldung an das Anwenderprogramm möglich. Der Rückgabewert wird an Windows gesendet (WindowsEventLog).

4.1.7.3 Eigene Funktionen implementieren

Nachdem Sie die ODK-Schnittstelle in der Datei <Projekt>.odk definiert haben, müssen Sie die ODK-Anwendungsfunktionen in der CPP-Datei editieren.

Vorgehen

Um die ODK-Anwendungsfunktionen zu editieren, gehen Sie folgendermaßen vor:

1. Führen Sie den Build aus, um die Header-Datei <ODK_Functions.h> zu aktualisieren.
2. Öffnen Sie die Datei <Projekt>.cpp, oder legen Sie bei Bedarf eine eigene Source-Datei an.
3. Übernehmen Sie die Funktionsprototypen von <ODK_Functions.h> in die Source-Datei.

Hinweis

Um bei Änderung der Funktionsparameter Schritt 3 zukünftig zu übergehen, verwenden Sie den Define des Funktionsprototyps.

4. Editieren Sie den Code Ihrer ODK-Anwendung in der CPP-Datei.

ODK-Anwendung

Die CPP-Datei enthält standardmäßig eine schematisch dargestellte Funktionsbeschreibung. Diese Beschreibung können Sie entsprechend den Änderungen in der Datei <Projekt>.odk verändern und/oder weitere Funktionsbeschreibungen hinzufügen.

```
#include "ODK_Functions.h"

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

4.2 ODK-Anwendung in das Zielsystem übertragen

Übertragen Sie die DLL-Datei manuell in einen spezifischen Windows-Ordner auf dem Zielsystem (z. B. per Netzwerkfreigabe oder USB-Stick). Nutzen Sie den üblichen Windows-Datentransfer für die Übertragung der ODK-Anwendung. Der Ablageort in Windowswird durch einen Registry Key vorgegeben. Beim Laden einer ODK-Anwendung sucht der ODK-Service automatisch in dem vom Registry Key vorgegeben Pfad nach der Datei.

Hinweis

ODK-Anwendung in der Debug-Konfiguration

Wenn die ODK-Anwendung in der Debug-Konfiguration übersetzt wurde, dann müssen Sie zusätzlich die Debug-DLLs der Entwicklungsumgebung auf das Zielsystem übertragen.

Der Standard-Wert, der den Dateipfad beschreibt, lautet:

%ProgramData%\Siemens\Automation\ODK1500S\

Hinweis

Administratorrechte

Um auf diesen Ordner zuzugreifen, benötigen Sie Administratorrechte. So wird verhindert, dass ODK-Anwendungen durch nicht berechtigte Personen eingespielt werden.

Beachten Sie

Das Setup des SIMATIC S7-1500 Software Controller prüft, ob der Dateipfad bereits existiert und die benötigen Administratorrechte hinterlegt sind.

Ist das nicht der Fall, wird das Verzeichnis nach "ODK1500S_OLD1" bzw. "ODK1500S_OLD2" umbenannt und ein neues Verzeichnis mit den korrekten Zugriffsrechten wird erstellt.

Das Dateisystem von Windows kann, abhängig von Ihrer Einstellung, den Ordner ausblenden. Sie können sich den Ordner über die Windows-Option "Ausgeblendete Dateien, Ordner und Laufwerke anzeigen" im Explorer-Menü "Organisieren > Ordner- und Suchoptionen > Ansicht" anzeigen lassen.

Der Registry Key für 32-Bit-Systeme lautet:
HKEY_LOCAL_MACHINE\SOFTWARE\Siemens\Automation\ODK1500S\odk_app_path

Der Registry Key für 64-Bit-Systeme lautet:
HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Siemens\Automation\ODK1500S\odk_app_path

Sie können den Standard-Wert des Registry Key ändern und somit den erwarteten Ablageort für die DLL-Datei an Ihre Bedürfnisse anpassen.

Hinweis

Pfad im Registry Key ändern

Um die DLL-Datei zu schützen, wählen Sie einen Ablageort, der durch einen Zugriffsschutz geschützt ist.

4.3 SCL-Datei nach STEP 7 importieren und generieren

Bei der Erstellung der Projektmappe werden folgende Dateien erzeugt:

- SCL-Datei für den Import nach STEP 7
- Alle Dateien abhängig von der Konfiguration, z. B. DLL-Datei

Wenn STEP 7 auf einem anderen PC als die Entwicklungsumgebung installiert ist, müssen Sie die erzeugte SCL-Datei auf den PC übertragen, auf dem STEP 7 installiert ist.

Voraussetzung

Die Projektdaten wurden generiert.

Vorgehen

Um die SCL-Datei zu importieren und zu kompilieren, gehen Sie folgendermaßen vor:

1. Starten Sie STEP 7.
2. Öffnen Sie Ihr Projekt.
3. Wählen Sie die Projektansicht.
4. Wählen Sie in der Projektnavigation die CPU aus.
5. Wählen Sie den Unterordner "Externe Quellen".

   Der Dialog "Öffnen" wird geöffnet.
6. Navigieren Sie im Dateisystem zu der SCL-Datei, die bei der Generierung der Projektdaten erzeugt wurde.
7. Bestätigen Sie Ihre Auswahl mit "Öffnen".

   Die SCL-Datei wird importiert. Nach Beendigung des Importvorgangs, wird die SCL-Datei im Ordner "Externe Quellen" angezeigt.
8. Kompilieren Sie die SCL-Datei, bevor Sie die Bausteine in Ihrem Projekt verwenden können.
9. Markieren Sie dazu im Unterordner "Externe Quellen" die SCL-Datei aus.
10. Wählen Sie den Kontextmenübefehl "Bausteine aus Quelle generieren".

Ergebnis

STEP 7 erzeugt die S7-Bausteine auf Basis der ausgewählten SCL-Datei.

Die erzeugten Bausteine werden nun automatisch in der Projektnavigation im Ordner "Programmbausteine" unterhalb der ausgewählten CPU angezeigt. Sie können die Funktionsbausteine mit dem nächsten Ladevorgang in das Zielgerät laden.

4.4 Funktionen ausführen

4.4.1 Funktionen laden

Einführung

Unabhängig vom Kontext, in dem die ODK-Anwendung ausgeführt wird, besteht der Ladevorgang aus den folgenden Schritten:

- Sie rufen die Anweisung "*<STEP7Prefix>*_Load" im STEP 7-Anwenderprogramm auf.
- Im Windows-Kontext prüft der Ladevorgang, ob ein 32-oder 64 Bit-Prozess benötigt wird und startet den entsprechenden Host. Jede ODK-Anwendung läuft in einem eigenen Windows-Prozess (ODK_Host).
- Der Host lädt die ODK-Anwendung und ruft die Funktionen "OnLoad()" und anschließend die Funktion "OnRun()" auf.

Hinweis

Laden gleicher ODK-Anwendungen mit geänderter Datei <Projekt>.odk

Wenn Sie eine ODK-Anwendung laden und nachträglich die Datei <Projekt>.odk ändern, wird empfohlen, dass Sie Ihre ODK-Anwendung zuerst entladen, bevor Sie die neu generierte ODK-Anwendung laden. Wenn die Anweisung "*<STEP7Prefix>*_Unload" nicht ausgeführt wird, befinden sich beide ODK-Anwendungen im Speicher. Das kann dazu führen, dass nicht genügend Speicher für die CPU verfügbar ist.

Anweisung "*<STEP7Prefix>*_Load"

Eine ODK-Anwendung wird durch den Aufruf der Anweisung "*<STEP7Prefix>*_Load" im STEP 7-Anwenderprogramm geladen.

*<STEP7Prefix>*_Load	
REQ	DONE
	BUSY
	ERROR
	STATUS

Die folgende Tabelle zeigt die Parameter der Anweisung "*<STEP7Prefix>*_Load":

Sektion	Deklaration	Datentyp	Beschreibung
Input	REQ	BOOL	Eine steigende Flanke aktiviert das Laden der ODK-Anwendung.
Output	DONE	BOOL	Zeigt an, dass die Anweisung das Laden der ODK-Anwendung beendet hat.
Output	BUSY	BOOL	Zeigt an, dass die Anweisung die ODK-Anwendung noch lädt.

Sektion	Deklaration	Datentyp	Beschreibung
Output	ERROR	BOOL	Zeigt an, dass ein Fehler während des Ladens der ODK-Anwendung aufgetreten ist. STATUS gibt Ihnen weitere Informationen über die mögliche Fehlerursache.
Output	STATUS	INT	Gibt Auskunft über mögliche Fehlerquellen, wenn ein Fehler während des Ladens der ODK-Anwendung aufgetreten ist.

Eingangsparameter

Durch einen Flankenwechsel (0 nach 1) am Eingangsparameter "REQ" wird die Funktion gestartet.

Ausgangsparameter

Die folgende Tabelle zeigt, welche Informationen nach dem Ladevorgang zurückgeliefert werden.

DONE	BUSY	ERROR	STATUS	Bedeutung
0	0	0	0x7000 =28672	Kein aktiver Ladevorgang
0	1	0	0x7001 =28673	Ladevorgang wird durchgeführt, erster Aufruf
0	1	0	0x7002 =28674	Ladevorgang wird durchgeführt, fortlaufender Aufruf
1	0	0	0x7100 =28928	Ab V2.0 einer CPU 1500: ODK-Anwendung wurde bereits geladen.
1	0	0	0x0000 =0	Ladevorgang wurde erfolgreich durchgeführt.
0	0	1	0x80A4 =-32604	ODK-Anwendung konnte nicht geladen werden. Starten Sie den ODK-Service manuell oder starten Sie Windows neu.
			0x80C2 =-32574	ODK-Anwendung konnte nicht geladen werden. Auf der Windows-Seite ist aktuell nicht genügend Speicher verfügbar. Laden Sie die ODK-Anwendung nach einigen Sekunden erneut.
			0x80C3 =-32573	ODK-Anwendung konnte nicht geladen werden. Die CPU verfügt aktuell nicht über genügend Speicher. Laden Sie die ODK-Anwendung nach einigen Sekunden erneut.
			0x8090 =-32624	ODK-Anwendung konnte nicht geladen werden. Bei der Ausführung der Funktion "OnLoad()" ist eine Exception aufgetreten.
			0x8092 =-32622	ODK-Anwendung konnte nicht geladen werden, da der Bibliotheksname ungültig ist.
			0x8093 =-32621	ODK-Anwendung konnte nicht geladen werden, da die ODK-Anwendung nicht gefunden werden konnte. Überprüfen Sie den Dateinamen und Ablageort der Datei.
			0x8094 =-32620	ODK-Anwendung konnte nicht geladen werden. Die ODK-Anwendung wurde für den Kontext des Windows-Benutzers erstellt, aber es ist kein Benutzer angemeldet.

DONE	BUSY	ERROR	STATUS	Bedeutung
			0x8095 =-32619	ODK-Anwendung konnte aus den folgenden Gründen nicht geladen werden: • die DLL-Datei ist keine ODK-Anwendung • eine 64-Bit-Anwednung sollte in ein 32-Bit-System geladen werden • Abhängigkeiten zu anderen Windows-DLL-Dateien konnten nicht gelöst werden – Überprüfen Sie, ob das Release Build der ODK-Anwendung genutzt wird. – Überprüfen Sie, ob die „Visual C++ Redistributables“ für Ihre genutzte Visual Studio Version installiert sind. • die CPU unterstützt die verwendete ODK-Version nicht.
			0x8096 =-32618	Die ODK-Anwendung konnte nicht geladen werden, weil die interne Identifizierung bereits von einer anderen geladenen ODK-Anwendung genutzt wird.
			0x8097 =-32617	Bis V1.8 einer CPU 1500: ODK-Anwendung wurde bereits geladen.
			0x8098 =-32616	Die ODK-Anwendung konnte nicht geladen werden, weil die ODK-Anwendung derzeit entladen wird.
			0x809B =-32613	Ab V2.0 einer CPU 1500: Die ODK-Anwendung konnte nicht geladen werden und gibt einen ungültigen Wert zurück (erlaubt sind die Werte 0x0000 und 0xF000 - 0xFFFF)
			0xF000 – 0xFFFF =-4096 – -1	Ab V2.0 einer CPU 1500: ODK-Anwendung konnte nicht geladen werden. Bei der Ausführung der Funktion "OnLoad()" ist ein Fehler aufgetreten.

Beispiel

In diesem Beispiel wird beschrieben, wie das Laden und Ausführen einer Windows-ODK-Anwendung projektiert sein kann, um nach Kommunikationsstörungen zu Windows wieder aufzusetzen.

Wenn Windows wieder verfügbar ist, wird die ODK-Anwendung geladen und die Ausführung der Funktionen ist wieder möglich.

Eine Kommunikationsstörung kann folgende Ursache haben:

- Windows Restart (bzw. Shutdown)
- Windows LogOff (wenn Anwendung im Userbereich)
- TerminateProcess/Absturz ODK_Host

Dazu ist ein Flag notwendig (hier: ODK_Loaded), das nach dem erfolgreichen Ladevorgang gesetzt wird, und bei fehlerhafter Ausführung der ODK-Funktion zurückgesetzt wird.

```
FUNCTION_BLOCK "ODK_AutoLoad"
{ S7_Optimized_Access := 'TRUE' }
VERSION : 0.1
  VAR
    ODK_Loaded : Bool;
  END_VAR
BEGIN
  // Laden der Windows-ODK-Anwendung
  IF NOT #ODK_Loaded THEN
    // Request Flag toggeln, wenn Ladevorgang nicht aktiv
    IF NOT "ODKProject_Load_DB".BUSY THEN
           "ODKProject_Load_DB".REQ := NOT "ODKProject_Load_DB".REQ;
    END_IF;

    // Laden der ODK-Anwendung
    "ODKProject_Load_DB"();

    // „Loaded" Flag setzen, wenn Ladevorgang erfolgreich
    IF "ODKProject_Load_DB".DONE THEN
      #ODK_Loaded := true;
    END_IF;
  END_IF;

  // Ausführen der ODK-Funktion(en) (nur in geladenem Zustand)
  IF #ODK_Loaded THEN
    // Request Flag toggeln, wenn Funktionsaufruf nicht aktiv
    IF NOT "ODKProjectSampleFunction_DB".BUSY THEN
           "ODKProjectSampleFunction_DB".REQ := NOT
           "ODKProjectSampleFunction_DB".REQ;
    END_IF;

    // Ausführen der Funktion
    "ODKProjectSampleFunction_DB"();

    // Das „Loaded" Flag muss zurückgesetzt werden, wenn
    // a) ein Kommunikationsfehler zu Windows vorliegt (0x80A4)
    // b) die ODK-Anwendung bereits vor diesem Funktionsaufruf
entladen wurde (0x8096)
    IF "ODKProjectSampleFunction_DB".STATUS = 16#80A4 OR
"ODKProjectSampleFunction_DB".STATUS = 16#8096
    THEN
    #ODK_Loaded := false;
    END_IF;
  END_IF;
END_FUNCTION_BLOCK
```

4.4.2 Funktionen aufrufen

Einführung

Nachdem die ODK-Anwendung geladen wurde, können Sie die ODK-Funktionen über Ihr STEP 7-Anwenderprogramm ausführen. Der Aufruf erfolgt über die entsprechende Anweisung "*<STEP7Prefix>*SampleFunction".

Sie können bis zu 32 ODK-Anwendungen parallel laden.

Anweisung "*<STEP7Prefix>*SampleFunction"

Eine ODK-Anwendung wird durch die Anweisung "*<STEP7Prefix>*SampleFunction" aufgerufen.

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

Die folgende Tabelle zeigt die Parameter der Anweisung "*<STEP7Prefix>*SampleFunction":

Sektion	Deklaration	Datentyp	Beschreibung
Automatisch generierte Parameter			
Input	REQ	BOOL	Eine ansteigende Flanke dieses Input-Wertes aktiviert die Ausführung der ODK-Anwendung.
Output	DONE	BOOL	Dieser Output-Wert zeigt an, dass die Anweisung die Ausführung der ODK-Anwendung beendet hat.
Output	BUSY	BOOL	Dieser Output-Wert zeigt an, dass die Anweisung die ODK-Anwendung noch ausführt.
Output	ERROR	BOOL	Dieser Output-Wert zeigt an, dass ein Fehler während der Ausführung der ODK-Anwendung aufgetreten ist. Weitere Informationen darüber gibt der Output-Wert STATUS aus.
Output	STATUS	INT	Dieser Output-Wert gibt Auskunft über mögliche Fehlerquellen, wenn ein Fehler während der Ausführung der ODK-Anwendung aufgetreten ist.
Nutzer-definierte Parameter			
Input	myInt		Nutzer-definierte Eingangsvariablen
InOut	myReal		Nutzer-definierte Eingangs-Ausgangsvariablen
Output	myBool		Nutzer-definierte Ausgangsvariablen

Eingangsparameter

Durch einen Flankenwechsel (0 nach 1) am Eingangsparameter "REQ" wird die Funktion gestartet.

Ausgangsparameter

Die folgende Tabelle zeigt, welche Informationen für die Output-Parameter nach der Ausführung zurückgeliefert werden.

DONE	BUSY	ERROR	STATUS	Bedeutung
0	0	0	0x7000 =28672	Kein aktiver Vorgang
0	1	0	0x7001 =28673	Erster Aufruf (asynchron)
0	1	0	0x7002 =28674	Fortlaufender Aufruf (asynchron)
1	0	0	0x0000 – 0x6FFF =0 – 28671	Funktion wurde ausgeführt und gibt einen Wert zwischen 0x0000 und 0x6FFF zurück. (ODK_SUCCESS = 0x0000)
0	0	1	0x80A4 =-32604	ODK-Anwendung konnte aus den folgenden Gründen nicht ausgeführt werden: • Die Anweisung "*<STEP7Prefix>*_Unload" wurde während einer Funktionsausführung aufgerufen. Die Funktionsausführung wurde auf der CPU-Seite abgebrochen. Windows beendet die Funktionsausführung normal. Es wird kein Rückgabewert an die CPU übermittelt. Warten Sie bis die Anweisung "*<STEP7Prefix>*_Unload" beendet ist. Laden Sie dann die ODK-Anwendung erneut. • Windows ist nicht verfügbar • ODK-Service läuft nicht Starten Sie den ODK-Service manuell oder starten Sie Windows neu.
			0x80C2 =-32574	ODK-Anwendung konnte nicht ausgeführt werden. Auf der Windows-Seite ist aktuell nicht genügend Speicher verfügbar. Laden Sie die ODK-Anwendung nach einigen Sekunden erneut.
			0x80C3 =-32573	ODK-Anwendung konnte nicht ausgeführt werden. Die CPU verfügt aktuell nicht über genügend Speicher. Laden Sie die ODK-Anwendung nach einigen Sekunden erneut.
			0x8090 =-32624	ODK-Anwendung konnte nicht ausgeführt werden. Bei der Ausführung ist ein Fehler aufgetreten.
			0x8091 =-32623	ODK-Anwendung konnte nicht ausgeführt werden. Während des Funktionsaufrufs kam es zu einem "STOP".
			0x8096 =-32618	ODK-Anwendung konnte nicht ausgeführt werden, da die ODK-Anwendung nicht geladen wurde oder das Entladen noch nicht abgeschlossen ist.
			0x8098 =-32616	ODK-Anwendung konnte nicht ausgeführt werden, da die Funktion nicht unterstützt wird.
			0x8099 =-32615	ODK-Anwendung konnte nicht ausgeführt werden, da die maximale Menge an Eingangsdaten (32 KB) überschritten wurde (Deklarationen mit "In" und "InOut")
			0x809A =-32614	ODK-Anwendung konnte nicht ausgeführt werden, da die maximale Menge an Ausgangsdaten (32 KB) überschritten wurde (Deklarationen mit "Out" und "InOut")

DONE	BUSY	ERROR	STATUS	Bedeutung
			0x809B =-32613	Die Funktion gibt einen ungültigen Wert zurück (erlaubt ist ein Wert zwischen 0x0000 und 0x6FFF; 0xF000 und 0xFFFF)
			0x809C =-32612	Funktion verwendet einen unzulässigen Datentyp: • IN_DATA • INOUT_DATA • OUT_DATA Wenn Sie einen ODK_CLASSIC_DB verwenden, deaktivieren Sie den optimierten Bausteinzugriff.
			0xF000 – 0xFFFF =-4096 – -1	Ab V2.0 einer CPU 1500: Die Funktion konnte nicht ausgeführt werden und gibt einen Wert zwischen 0xF000 und 0xFFFF zurück. (ODK_USER_ERROR_BASE = 0xF000)

4.4.3 Funktionen entladen

Einführung

Die ODK-Anwendung wird mit dem Aufruf der Anweisung "*<STEP7Prefix>*_Unload" entladen. Der Aufruf erfolgt aus dem STEP 7-Anwenderprogramm.

Die ODK-Anwendung wird zusätzlich zu diesem Aufruf automatisch aus folgenden Gründen entladen.

- Beenden der CPU
- Durch Urlöschen der CPU
- Neustart von Windows
- Abmelden des Windows-Benutzers (im Kontext eines Windows-Benutzers)

Unabhängig vom Kontext, in dem die ODK-Anwendung ausgeführt wird, besteht der Entladevorgang aus den folgenden Schritten:

- Sie rufen die Anweisung "*<STEP7Prefix>*_Unload" im STEP 7-Anwenderprogramm auf.
- Ab jetzt können keine neuen Ausführungen (Executes) mehr für diese ODK-Anwendung ausgeführt werden. Noch aktive Ausführungen werden auf CPU-Seite abgebrochen. Windows beendet die Funktionsausführung normal ("Unload" wartet). Es wird kein Rückgabewert an die CPU übermittelt.
- Der Host ruft die Funktionen "OnStop()" und "OnUnload()" auf.
- Die ODK-Anwendung wird entladen.

Anweisung "*<STEP7Prefix>*_Unload"

Eine ODK-Anwendung wird durch den Aufruf der Anweisung "*<STEP7Prefix>*_Unload" im STEP 7-Anwenderprogramm entladen.

*<STEP7Prefix>*_Unload	
REQ	DONE
	BUSY
	ERROR
	STATUS

Die folgende Tabelle zeigt die Parameter der Anweisung "*<STEP7Prefix>*_Unload":

Sektion	Deklaration	Datentyp	Beschreibung
Input	REQ	BOOL	Eine steigende Flanke aktiviert das Entladen der ODK-Anwendung.
Output	DONE	BOOL	Zeigt an, dass die Anweisung das Entladen der ODK-Anwendung beendet hat.
Output	BUSY	BOOL	Zeigt an, dass die Anweisung die ODK-Anwendung noch entlädt.
Output	ERROR	BOOL	Zeigt an, dass ein Fehler während des Entladens der ODK-Anwendung aufgetreten ist. STATUS gibt Ihnen weitere Informationen über die mögliche Fehlerursache.
Output	STATUS	INT	Gibt Auskunft über mögliche Fehlerquellen, wenn ein Fehler während des Entladens der ODK-Anwendung aufgetreten ist.

Eingangsparameter

Durch einen Flankenwechsel (0 nach 1) am Eingangsparameter "REQ" wird die Funktion gestartet.

Ausgangsparameter STATUS

Die folgende Tabelle zeigt, welche Informationen nach dem Entladevorgang zurückgeliefert werden.

DONE	BUSY	ERROR	STATUS	Bedeutung
0	0	0	0x7000 =28672	Kein aktiver Entladevorgang
0	1	0	0x7001 =28673	Entladevorgang wird durchgeführt, erster Aufruf
0	1	0	0x7002 =28674	Entladevorgang wird durchgeführt, fortlaufender Aufruf

DONE	BUSY	ERROR	STATUS	Bedeutung
1	0	0	0x0000 =0	Entladevorgang wurde erfolgreich durchgeführt
0	0	1	0x80A4 =-32604	ODK-Anwendung konnte aus den folgenden Gründen nicht entladen werden: • Windows ist nicht verfügbar Starten Sie den ODK-Service manuell oder starten Sie Windows neu.
			0x80C2 =-32574	ODK-Anwendung konnte nicht entladen werden. Auf der Windows-Seite ist aktuell nicht genügend Speicher verfügbar. Laden Sie die ODK-Anwendung nach einigen Sekunden erneut.
			0x80C3 =-32573	ODK-Anwendung konnte nicht entladen werden. Die CPU verfügt aktuell nicht über genügend Speicher. Laden Sie die ODK-Anwendung nach einigen Sekunden erneut.
			0x8090 =-32624	ODK-Anwendung konnte nicht entladen werden. Bei der Ausführung der Funktion "OnUnload()" ist eine Exception aufgetreten.
			0x8096 =-32618	ODK-Anwendung konnte nicht entladen werden, da die ODK-Anwendung nicht geladen wurde oder das Entladen noch nicht abgeschlossen ist.
			0x809B =-32613	Ab V2.0 einer CPU 1500: Die ODK-Anwendung konnte entladen werden und gibt einen ungültigen Wert zurück (erlaubt sind die Werte 0x0000 und 0xF000 - 0xFFFF)
			0xF000 – 0xFFFF =-4096 – -1	Ab V2.0 einer CPU 1500: ODK-Anwendung konnte entladen werden. Bei der Ausführung der Funktion "OnLoad()" ist im CCX-Obekt ein Fehler aufgetreten.

4.5 Remote-Debug

Wenn Sie Microsoft Visual Studio als Entwicklungsumgebung nutzen, können Sie den Debugger zur Fehlerbeseitigung nutzen.

Mit dem Remote-Debugger können Sie auf einem Zielsystem ohne Visual Studio eine ODK-Anwendung debuggen. Dabei ist zu beachten, dass die erzeugten ODK-Anwendungen (DLLs) in einen der beiden folgenden Prozesse geladen werden:

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

Der benötigte Remote-Debugger ist abhängig von der verwendeten Visual Studio Version auf dem Host-System und vom Systemtyp (32bit/64bit) des Zielsystems.

installierte Visual Studio Version	Link zum Download Center für den Remote-Debugger
Microsoft Visual Studio 2010	Microsoft Visual Studio 2010 Remote Debugger (https://www.microsoft.com/de-de/download/details.aspx?id=475)
Microsoft Visual Studio 2012	Microsoft Visual Studio 2012 Remote Debugger (https://www.microsoft.com/de-de/download/details.aspx?id=38184)
Microsoft Visual Studio 2013	Microsoft Visual Studio 2013 Remote Debugger (https://www.microsoft.com/de-de/download/details.aspx?id=44918)
Microsoft Visual Studio 2015	Microsoft Visual Studio 2014 Remote Debugger (https://www.microsoft.com/de-de/download/details.aspx?id=48155)

Nach dem Download können Sie den Remote-Debugger auf dem Zielsystem installieren.

4.5.1 Remote-Debugging durchführen

Vorgehen

1. Starten Sie auf dem Zielsystem den Visual Studio Remote-Debugger über "Start > All Programs > Visual Studio 20xx > Remote Debugger".
2. Konfigurieren Sie die Authentifikation.

   Wählen Sie in den Optionen "Keine Authentifizierung" und setzen Sie einen Haken bei "Allen Benutzern das Debugging ermöglichen".

   Beachten Sie die Sicherheitshinweise.

3. Kopieren Sie die Visual Studio C++ Debug-DLLs aus dem Ordner "<Installationspfad VS>\VC\redist\Debug_NonRedist\<Anwendungstyp>\Microsoft.<VS version>.DebugCRT" in den Zielordner.
   - Zielordner bei einem 32-Bit-Windows und einer 32-Bit-Anwendung:

     <windows install path>\System32
   - Zielordner bei einem 64-Bit-Windows und einer 64-Bit-Anwendung:

     <windows install path>\System32
   - Zielordner bei einem 64-Bit-Windows und einer 32-Bit-Anwendung:

     <windows install path>\SysWOW64

   Hinweis

   Wenn Sie Visual Studio 2015 verwenden, benötigen Sie zusätzlich die "ucrtbased.dll".

   Wenn diese DLL im Zielsystem nicht vorhanden ist, kopieren Sie diese vom Host aus dem Ordner:

   bei 32-Bit Windows unter Program Files\...

   bei 64-Bit Windows unter Program Files (x86)\...

   ...\Microsoft SDKs\Windows Kits\10\ExtensionSDKs\Microsoft.UniversalCRT.Debug\<höchste verfügbare Version>\Redist\Debug\<Anwendungstyp (32/64-Bit)>

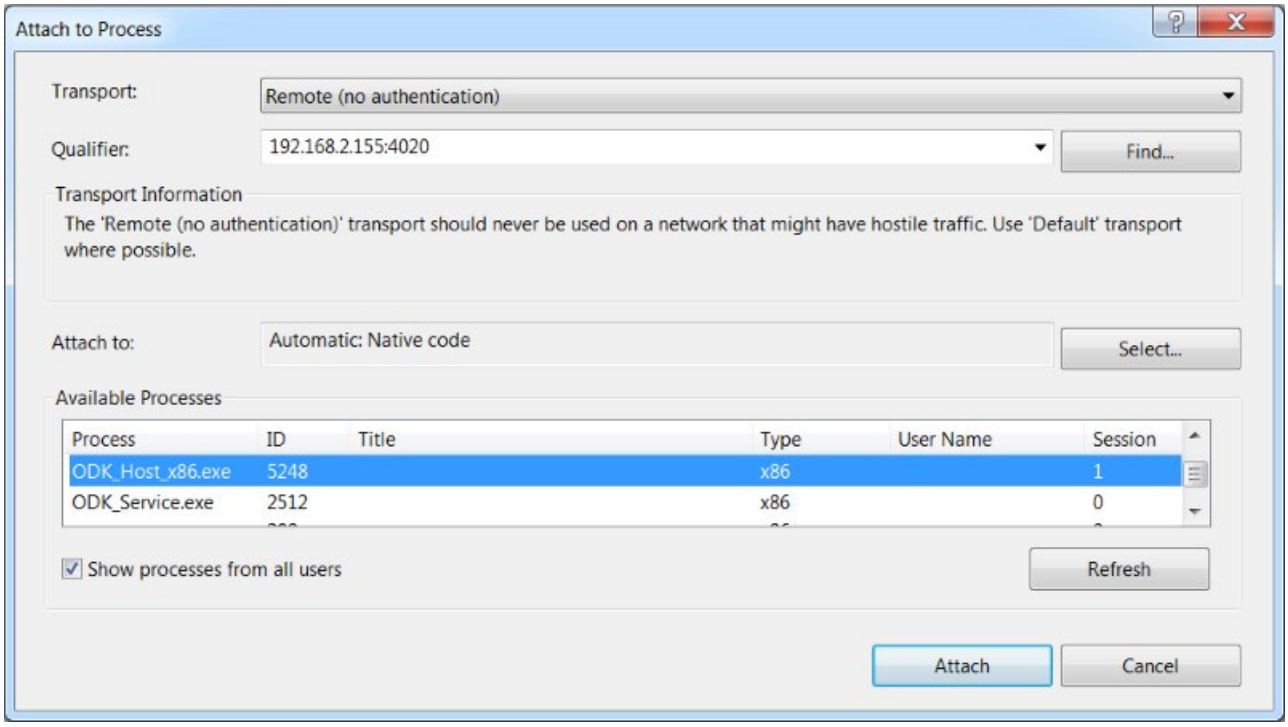
4. Kopieren Sie die ODK-Anwendung auf das Zielsystem in den Ordner "C:\ProgramData\Siemens\Automation\ODK1500S".

   Hinweis

   Ist die ODK-Anwendung geladen, entladen (Seite 45) Sie diese vor dem Kopieren.

5. Laden (Seite 39) Sie die ODK-Anwendung auf dem Zielsystem.

6. Setzen Sie Breakpoints in den Quellcode und fügen Sie den Debugger über "Debug > Attach to Process…" an.



Debugging OnLoad/OnRun

Um den Debugger in die Funktion OnLoad() oder OnRun() anzufügen, bauen Sie eine Warteschleife am Anfang von OnLoad() ein.

Beispiel einer Warteschleife:

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

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

Ergebnis

Der Debugger stoppt die Ausführung des Codes unter dem aktivierten Breakpoint.

Entwicklung einer ODK-Anwendung für die Echtzeit-Umgebung

5

5.1 ODK-Anwendung erstellen

5.1.1 Voraussetzungen

- ODK ist installiert. Die Entwicklungsumgebung Eclipse ist installiert.
- Sie benötigen Administratorrechte zum Anlegen und Bearbeiten eines ODK-Projekts.

Hinweis

Falls Sie den Workspace an einen anderen Ablageort legen müssen, achten Sie darauf, dass Sie den gesamten Workspace umkopieren.

5.1.2 Projekt anlegen

Um Ihnen die Entwicklung einer ODK-Anwendung zu erleichtern, wird bei der Installation von ODK 1500S ein ODK-Template für ein ODK-Projekt mitgeliefert.

Vorgehen

Um mit Hilfe des ODK-Templates ein ODK-Projekt in Eclipse zu erstellen, gehen Sie folgendermaßen vor:

1. Starten Sie Eclipse als Entwicklungsumgebung.
2. Wählen sie im Menü "File > New" den Befehl "Project..."

   Der Dialog "New Project" wird geöffnet.

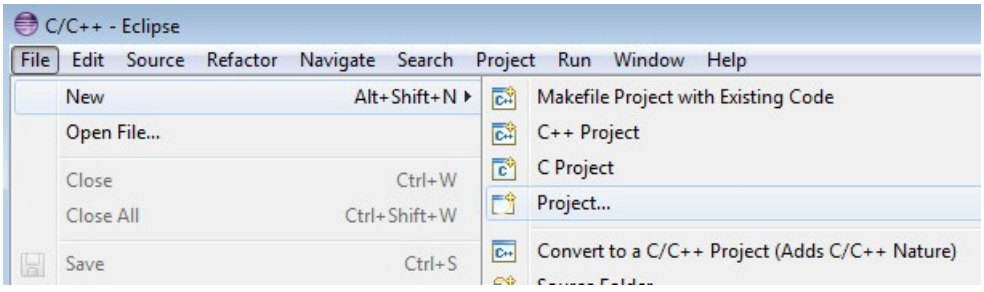


Bild 5-1 Neues Projekt anlegen mit Eclipse

3. Wählen Sie Ihre bevorzugte Programmiersprache und das zugehörige ODK-Template aus.

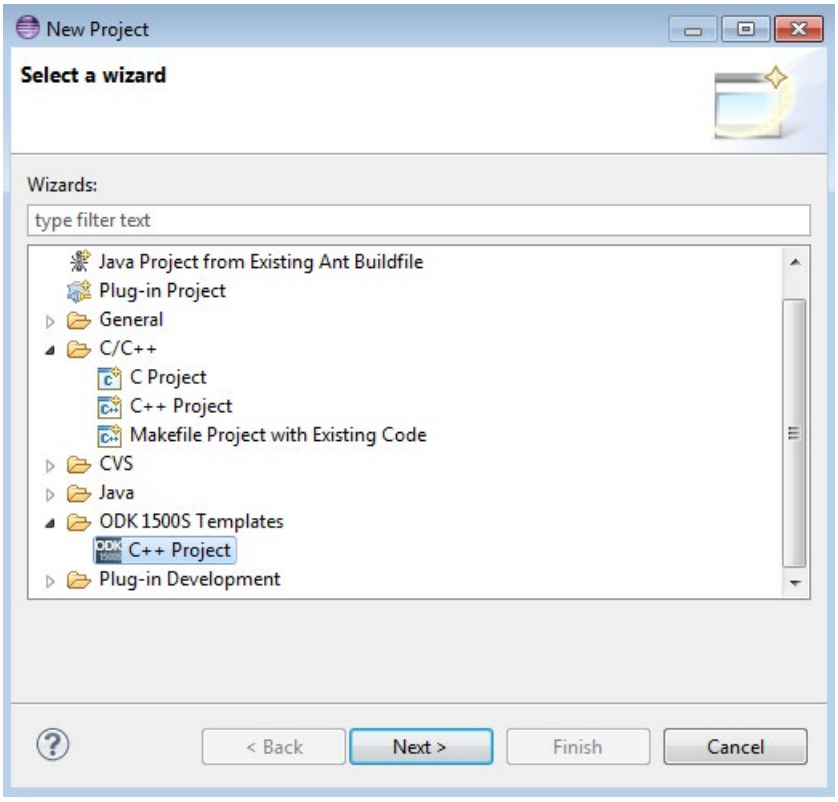


Bild 5-2 Template auswählen

4. Vergeben Sie einen Projektnamen.
5. Bestätigen Sie mit "OK".

Ergebnis

Das ODK-Projekt wird mit Hilfe des ODK-Templates erstellt und legt folgende Projekteinstellungen fest:

- Projekteinstellungen für die Generierung der SO-Datei
- Automatisiert die Erzeugung der SO- und SCL-Datei

Das ODK-Template richtet standardmäßig die folgenden Dateistruktur ein:

Ordner / Datei			Beschreibung
<Projektpfad>			
	def		
		<Projekt>.odk	ODK-Interface-Beschreibung
		<Projekt>.scl.additional	S7-Blöcke, die an die Datei <Project>.scl angehängt werden. Die Datei ist zwar nicht Teil der Projektvorlage, aber der Code-Generator bearbeitet die Datei.
	STEP7		**Dateien aus diesem Ordner dürfen nicht editiert werden!**
		<Projekt>.scl	S7-Bausteine
	cg_src_priv		**Dateien aus diesem Ordner dürfen nicht editiert werden!**
		ODK_Types.h	Definition der ODK-BaseTypes
		ODK_Functions.h	Funktions-Prototypen
		ODK_Execution.cpp	Implementierung der Methode "Execute"
	src		
		<Projekt>.cpp	Funktions-Code: Diese Datei hat immer das Suffix CPP, unabhängig davon, ob Sie ein C- oder C++-Projekt erstellt.
	release_so		
		<Projekt>.so	ODK-Application Binary (Release-Version), die auf das Zielsystem übertragen werden muss.
		<Projekt>.debuginfo.so	ODK-Application Binary (Debug-Version), die für die Post Mortem-Analyse benötigt wird.
		<Projekt>.symbols	Symbolinformationen, die für die Post Mortem-Analyse benötigt werden.
	launches		
		<Projekt>.gdb.launch	Start für die Post Mortem-Analyse.

Hinweis

Leerzeichen im Projektnamen

Alle Leerzeichen im Projektnamen werden automatisch durch einen Unterstrich ersetzt.

Aus zum Beispiel “My first project” wird “My_first_project”.

5.1.3 ODK-Anwendung generieren

Die Generierung der Projektdaten ist in zwei automatisierte Schritte unterteilt.

- **Pre-Build**: Erzeugung der standardmäßig angelegten Dateien auf Basis der geänderten Datei <Projekt>.odk
- **Build**: Generierung der SO-Datei

Vorgehen

Um die Projektdaten zu generieren, gehen Sie folgendermaßen vor:

1. Speichern Sie alle editierten Dateien.
2. Wählen Sie im Menü "Build" den Befehl "Build Project".

Hinweis

Die Projektdaten werden nur generiert, wenn die Dateien verändert wurden.

Ergebnis

Die Generierung der Projektdaten wird gestartet. Die automatisch erzeugten Dateien werden im Dateisystem abgelegt.

- SO-Datei: Projekt-Verzeichnis\<Projekt>\<BuildConfiguration>\<Projekt>.so
- SCL-Datei: Projekt-Verzeichnis\<Projekt>\STEP7\<Projekt>.scl

5.1.4 Ablaufeigenschaften einer ODK-Anwendung definieren

Als nächsten Schritt definieren Sie die Schnittstellenbeschreibung der ODK-Anwendung in der Datei <Projekt>.odk. Die Datei enthält folgende Elemente:

- Kommentare
- Parameter
- Definitionen von Funktionen und Strukturen

Vorgehen

Um die Schnittstellenbeschreibung in der Datei <Projekt>.odk zu definieren, gehen Sie folgendermaßen vor:

1. Öffnen Sie die Datei <Projekt>.odk.
2. Ändern Sie die Elemente abhängig von Ihren Anforderungen.

Beschreibung der Elemente

Kommentare

Sie können Kommentare für Erläuterungen verwenden.

Parameter

Die Definition der Parameter muss innerhalb einer Code-Zeile erfolgen.
`<ParameterName>=<Value> // optional comment`

Die Interface-Datei unterstützt folgende Parameter:

Parameter	Wert	Beschreibung
Context	realtime	Definiert, dass die ODK-Anwendung im Kontext der Echtzeitumgebung (Seite 56) geladen wird.
Trace	on	Definiert die Trace-Funktion in der ODK-Anwendung. In diesem Fall benötigt die ODK-Anwendung 32 KB zusätzlichen Speicher als Trace-Puffer. Für die Nutzung in STEP 7 wird standardmäßig ein Funktionsbaustein "GetTrace" erstellt.
	off	Ein Funktionsbaustein "GetTrace" wird erstellt. Der Trace-Puffer enthält nur einen Trace-Eintrag mit dem Inhalt: trace is off.
HeapSize	[4…<verfügbarer Speicher der CPU (Seite 92)>]k	Definiert eine Speichermenge in KB, die als Heap für diese Echtzeitanwendungen genutzt werden kann.
HeapMaxBlockSize	[8…<HeapSize>]	Definiert die Speichergröße in Bytes, die maximal auf ein Mal allokiert werden kann.
SyncCallParallelCount	[1...9] Default=3	Ist ein optionaler Parameter und definiert die maximale Anzahl paralleler Aufrufe in dieser ODK-Anwendung. Die Größe des Speichers der für Aufrufe in diese ODK-Anwendung reserviert wird ist: SyncCallParallelCount * (SyncCallStackSize + SyncCallDataSize)
SyncCallStackSize	[1...1024]k Default=32k	Ist ein optionaler Parameter und definiert die Größe des Thread-Stacks für einen Aufruf in dieser ODK-Anwendung. Jeder erneute Aufruf erhält einen eigenen Stack-Speicher.
SyncCallDataSize	[1...1024]k	Ist ein optionaler Parameter und definiert die Größe des Datenbereichs für einen Aufruf in dieser ODK-Anwendung. Der Datenbereich beinhaltet IN-, INOUT- und OUT-Parameter. Jeder erneute Aufruf erhält einen eigenen Stack-Speicher.
	Default=auto	Die benötigte Datengröße wird automatisch vom Code-Generator berechnet
STEP7Prefix	<String>	Beschreibt die Zeichenkette, die Ihren Funktionen vorangestellt und nach dem Import der SCL-Datei in STEP 7 dargestellt wird. Erlaubt sind folgende Zeichen: {A...Z, a…z, 1…9, -, _} Standardmäßig ist der Projektname ohne Leerzeichen eingetragen.

5.1.5 Umfeld zum Laden oder Ausführen der ODK-Anwendung

Wenn die SCL-Datei als externe Quelle nach STEP 7 importiert wird, werden die ODK-Anweisungen im gewählten Verzeichnis in STEP 7 erstellt. Die ODK-Anweisungen geben Ihnen nach der Programmierung und dem erstmaligen Laden die Möglichkeit, Ihre ODK-Anwendung unabhängig vom STEP 7-Anwenderprogramm zu steuern. Sie können bis zu 32 ODK-Anwendungen laden.

Sie können Ihre ODK-Anwendung im Kontext der Echtzeit-Umgebung laden und ausführen:

Echtzeitumgebung

Die folgende Code-Zeile ist in der Datei <Projekt>.odk notwendig, um die ODK-Anwendung im Kontext der Echtzeit-Umgebung zu nutzen:
`Context=realtime`

In diesem Kontext läuft die ODK-Anwendung nicht in einem Host-Prozess auf der Windows-Seite, sondern in der Echtzeit-Umgebung. Da die ODK-Anwendung synchron geladen wird, sollte sie in einem Startup-OB (z. B. OB100) geladen werden.

Im Kontext der Echtzeit-Umgebung ist der Anzahl der ladbaren ODK-Anwendungen (Seite 92) limitiert.

Größe der ODK-Anwendung im CPU-Speicher ermitteln

Um die benötigte Größe der ODK-Anwendung im CPU-Speicher zu ermitteln, gehen Sie folgendermaßen vor:

1. Öffnen Sie einen Kommandozeilen-Dialog.
2. Geben Sie folgenden Pfad aus dem ODK-Installationsordner ein (die angehängte Option "-l" entspricht einem kleinen "L"): eclipse\ build_tools\x86_64_gcc_pc_elf_4.8.1-1\bin\x86_64-pc-elf-readelf.exe “*<Ablageort\Datei.so>*” -l

   Die Größe Ihrer ODK-Anwendung finden Sie unter der Überschrift "Program Headers" in der Spalte "MemSiz".

   Zusätzlich zu der hier angegebenen Größe wird pro ODK-Anwendung weiterer administrativer Speicher benötigt. Der administrative Speicher lässt sich folgendermaßen berechnen:

   Administrativer Speicher = SyncCallParallelCount * (SyncCallStackSize + SyncCallDataSize)

5.1.6 Funktionen und Strukturen einer ODK-Anwendung definieren

5.1.6.1 Funktionen einer ODK-Anwendung definieren

Funktionen

Funktionen werden durch folgende allgemeine Code-Zeilen definiert:

```
ODK_RESULT <FunctionName>
([<InOut-Identifier>] <DataType> <VariableName>, …);
```

Die Datei <Projekt>.odk enthält standardmäßig eine beispielhafte Funktionsbeschreibung. Diese Beschreibung können Sie verändern und/oder weitere Funktionsbeschreibungen hinzufügen.

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

Syntax-Regeln für Funktionen

Folgenden Syntax-Regeln gelten für Funktionen innerhalb der Datei <Projekt>.odk:

- Beachten Sie beim Funktionsnamen die Groß- und Kleinschreibung.
- Funktionsdefinitionen können Sie in mehrere Zeilen aufgeteilen.
- Beenden Sie eine Funktionsdefinition durch ein Semikolon.
- TAB und LEERZEICHEN sind erlaubt.
- Definieren Sie einen Variablenname in einer Funktion nicht doppelt.
- Verwenden Sie keine Schlüsselwörter für die genutzte Programmiersprache (z. B. "INT" als Parametername)
- Nutzen Sie ODK_RESULT nur für die Rückgabewerte der Funktion.
- Variablennamen müssen mit einem Buchstaben oder einem Unterstrich beginnen.
- Unzulässige Funktionsnamen werden beim Generieren in der Entwicklungsumgebung angezeigt.
- Folgende Namen sind in Kombination von *<STEP7Prefix>* und <Funktionsname> nicht erlaubt: ODK_Load, ODK_Unld, ODK_ExcA, ODK_ExcS

<FunctionName>

Funktionsnamen gelten mit den Syntax- und Zeichenbeschränkungen der verwendeten Programmiersprache.

<InOut-Identifier>

Es gibt drei definierte InOut-Identifiers. Nutzen Sie diese in folgender Reihenfolge: [IN], [OUT], [INOUT]

- [IN]: Definiert eine Eingangs-Variable. Die Variable wird zur Funktion kopiert, wenn diese aufgerufen wird. Diese ist konstant und kann nicht verändert werden.
- [OUT]: Definiert eine Ausgangs-Variable. Die Variable wird zurückkopiert, nachdem die Funktion beendet wurde.
- [INOUT]: Definiert eine Eingangs- und Ausgangs-Variable. Die Variable wird zur Funktion kopiert, wenn diese aufgerufen wird. Diese ist nicht konstant und kann verändert werden. Die Variable wird zurückkopiert, nachdem die Funktion beendet wurde.

<DataType>

Der Datentyp definiert den Typ einer Variablen. Die folgenden Tabellen definieren die möglichen Datentypen und ihre Darstellungsweise in C++ oder STEP 7:

- **Elementare Datentypen:**

ODK-Datentyp	SIMATIC-Datentyp	C++-Datentyp	Beschreibung
ODK_DOUBLE	LREAL	double	64-Bit Floating Point, IEEE 754
ODK_FLOAT	REAL	float	32-Bit Floating Point, IEEE 754
ODK_INT64	LINT	long long	64-Bit signed integer
ODK_INT32	DINT	long	32-Bit signed integer
ODK_INT16	INT	short	16-Bit signed integer
ODK_INT8	SINT	char	8-Bit signed integer
ODK_UINT64	ULINT	unsigned long long	64-Bit unsigned integer
ODK_UINT32	UDINT	unsigned long	32-Bit unsigned integer
ODK_UINT16	UINT	unsigned short	16-Bit unsigned integer
ODK_UINT8	USINT	unsigned char	8-Bit unsigned integer
ODK_LWORD	LWORD	unsigned long long	64-Bit Bitstring
ODK_DWORD	DWORD	unsigned long	32-Bit Bitstring
ODK_WORD	WORD	unsigned short	16-Bit Bitstring
ODK_BYTE	BYTE	unsigned char	8-Bit Bitstring
ODK_BOOL	BOOL	unsigned char	1-Bit bitstring, verbleibende Bits (1..7) sind leer
ODK_LTIME	LTIME	unsigned long long	64-Bit Dauer in Nanosekunden
ODK_TIME	TIME	unsigned long	32-Bit Dauer in Millisekunden
ODK_LDT	LDT	unsigned long long	64-Bit Datum und Uhrzeit des Tages in Nanosekunden
ODK_LTOD	LTOD	unsigned long long	64-Bit Uhrzeit des Tages in Nanosekunden seit Mitternacht
ODK_TOD	TOD	unsigned long	32-Bit Uhrzeit des Tages; in Millisekunden seit Mitternacht
ODK_CHAR	CHAR	char	8-Bit-Zeichen

- **Komplexe Datentypen:**

ODK-Datentyp	SIMATIC-Datentyp	C++-Datentyp	Beschreibung
ODK_DTL	DTL	struct ODK_DTL	Struktur für Datum und Uhrzeit
ODK_S7STRING	STRING	unsigned char	Zeichen-String (8-Bit-Zeichen) mit Länge max. und act. (2xUSINT)
ODK_CLASSIC_DB	VARIANT	struct ODK_CLASSIC_DB	Klassik-DB (global oder basierend auf UDT)
[ ]	ARRAY	[ ]	Bereich von gleichen Datentypen. Sie können alle Datentypen außer ODK_CLASSIC_DB als Array nutzen.

- **Benutzerdefinierte Datentypen:**

  Benutzerdefinierte Datentypen (UDT) beinhalten strukturierte Daten, insbesondere die Namen und die Datentypen dieser Komponente und deren Reihenfolge.

  Ein benutzerdefinierter Datentyp kann in der Oberflächenbeschreibung mit dem Schlüsselwort "ODK_STRUCT" definiert werden.

  Beispiel

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

  Die folgenden Syntaxregeln gelten für die Struktur:

  – Sie können die Struktur in mehrere Zeilen aufteilen.
  – Die Strukturdefinition muss ein Semikolon abschließen.
  – Tabulatoren und Leerzeichen zwischen den Elementen sind in einer beliebigen Anzahl erlaubt.
  – Sie dürfen keine Schlüsselwörter für die erzeugte Sprache verwenden (z. B. "int" als Variablennamen).

  Sie können innerhalb einer Struktur weitere Strukturen anlegen.

<StructName>

Strukturnamen gelten mit den Syntax- und Zeichenbeschränkungen der Programmiersprache und wie für Variablendefinitionen in STEP7 definiert.

In STEP7 wird der Strukturname durch den STEP7-Prefix erweitert.

<VariableName>

Variablennamen gelten mit den Syntax- und Zeichenbeschränkungen der Programmiersprache.

Beispiel

Das folgende Code-Beispiel erläutert Ihnen die Definitionen von Funktionen und Strukturen. Sortieren Sie die Parameter nach: IN, OUT, INOUT.

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

Siehe auch

Trace-Puffer auslesen (Seite 83)

Helper-Funktionen (Seite 99)

5.1.6.2 Verwendung von ODK_CLASSIC_DB als Parameter

Der Datentyp ODK_CLASSIC_DB darf nur mit dem InOut-Identifier [IN] und [INOUT] benutzt werden. Wird ein Parameter des Datentyps ODK_CLASSIC_DB mit dem InOut-Identifier [IN] oder [INOUT] verwendet, darf kein weiterer Parameter, egal welchen Datentyps, mit dem gleichen InOut-Identifier verwendet werden.

Beispiel

```
// INTERFACE
...
// OK:
ODK_RESULT MyFunc1([IN] ODK_CLASSIC_DB myDB);
ODK_RESULT MyFunc2([IN] ODK_CLASSIC_DB myDB1, [INOUT] ODK_CLASSIC_DB
myDB2);
//
// NOT OK (Code Generator will throw an error):
// ODK_CLASSIC_DB nicht für [OUT] zulässig
ODK_RESULT MyFunc3([OUT] ODK_CLASSIC_DB myDB);
// wenn ODK_CLASSIC_DB für [IN] verwendet, darf kein weiterer [IN]
Parameter in dieser
// Funktion definiert werden
ODK_RESULT MyFunc4([IN] ODK_CLASSIC_DB myDB, [IN] ODK_INT32 myint);
```

Anwendungsbeispiel für C++

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

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

    return myInt1 + myInt2;
}
```

Um innerhalb einer Anwenderfunktion auf den Datentyp ODK_CLASSIC_DB zugreifen zu können, stehen Ihnen die Helper-Funktionen (Seite 99) der folgenden Klassen zur Verfügung:

- Klasse "CODK_CpuReadData"
- Klasse "CODK_CpuReadWriteData"

5.1.6.3 Handhabung von Strings

Für Strings (String oder WString) können Sie eine maximale Länge definieren. Definieren Sie die maximale Zeichenanzahl in eckigen Klammern direkt nach dem Datentyp:

- ODK_S7STRING[30] oder
- ODK_S7WSTRING[1000]

Ohne eine Beschränkung hat ein String standardmäßig eine Länge von maximal 254 Zeichen.

Um innerhalb einer Anwenderfunktion auf die Datentypen ODK_S7STRING bzw. ODK_S7WSTRING zugreifen zu können, stehen Ihnen die String-Helper-Funktionen (Seite 99) zur Verfügung.

Beispiel

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

Wenn Sie [INOUT] nutzen, können Sie den String mit einer unterschiedlichen Länge als [INOUT] des Funktionsbausteins in STEP 7 festsetzen.

5.1.6.4 Definition der Datei <Projekt>.odk

Anhand der gewählten Parameter in der Datei <Projekt>.odk werden die Funktionsprototypen und Funktionsblöcke generiert. Definieren Sie dafür die Datei <Projekt>.odk.

Standardmäßig enthält die Datei <Projekt>.odk folgenden Inhalt:

- Description

  Die möglichen Datentypen, die für die Schnittstelle benutzt werden, sind in den Kommentarzeilen beschrieben. Das erleichtert Ihnen die Definition des richtigen Variablentyps für Ihre Aufgabe.

- Context=realtime

  Die ODK-Anwendung wird im Kontext der Echtzeit-Umgebung geladen.

- Trace=on

  Definiert die Trace-Funktion in der ODK-Anwendung. Für die Nutzung in STEP 7 wird standardmäßig ein Funktionsbaustein "GetTrace" erstellt.

  Wenn Sie die Anweisung "ODK_TRACE" (Seite 83) definieren, wird sie auch kompiliert und ausgeführt. Wenn Sie den Parameter Trace=on in der Datei <Projekt>.odk definieren, wird die Anweisung automatisch mit folgendem Code definiert::

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

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

  Durch den Aufruf der Anweisung wird im Trace-Puffer ein Eintrag erstellt.

- HeapSize

  Definiert eine Speichermenge in KB, die als Heap für diese Echtzeitanwendungen genutzt werden kann.

- HeapMaxBlockSize

  Definiert die Speichergröße in Bytes, die maximal auf ein Mal allokiert werden kann.

- STEP7Prefix="<Projekt>"

  Stellt den Funktionen der ODK-Anwendung einen String für die SCL-Generierung voran. Dieser wird in STEP 7 sichtbar. Sie können den Parameter ändern. Die String-Länge des Präfix inklusive Funktionsnamen darf eine Zeichenlänge von 125 Zeichen nicht überschreiten (z. B. ODK_App_SampleFunction)

- "SampleFunction" Funktionsdefinition

  Diese Standardfunktion können Sie in der Datei <Projekt>.odk beliebig ändern und weitere Funktionen hinzufügen. Die String-Länge darf eine Zeichenlänge von 125 Zeichen nicht überschreiten. Die zugehörige Funktion befindet sich in der CPP-Datei.

Beispiel

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

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

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

5.1.6.5 Datei <Projekt>.odk modifizieren

Das folgende Beispiel zeigt Ihnen wie Sie die Datei <Projekt>.odk Ihren Bedürfnissen anpassen können.

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

ODK_RESULT GetString ([OUT]    ODK_S7STRING myString);

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

5.1.6.6 Kommentare

Kommentare werden mit einem doppelten Schrägstrich gestartet "//" und enden automatisch am Ende der Zeile.

Alternativ können Sie Kommentare durch /* *<comment>* */ abgrenzen, dadurch sind neue Zeilen in einem Kommentar möglich. Zeichen nach dem Ende der Kommentarkennung "*/" werden vom Code-Generator weiterverarbeitet.

Kommentare für Funktionen und Strukturen

Kommentare zu Funktionen und Strukturen platzieren Sie direkt vor den Funktionen/Strukturen.

Diese Kommentare werden an die Dateien ODK_Functions.h und <Project>.scl übergeben.

In der Datei <Project>.scl werden die Kommentare in die Blockeigenschaften und in den Code-Bereich der Funktion dupliziert.

Beachten Sie folgende Regeln:

- Kommentare für Funktionen und Strukturen müssen direkt vor den Funktionen/Strukturen stehen (ohne Leerzeile).
- Das Ende des Kommentars steht vor dem Schlüsselwort ODK_RESULT oder ODK_STRUCT.
- Sie können beide Kennungen "//" und "/* */" verwenden, jedoch nicht in Kombination innerhalb eines Kommentars.

Beispiel

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

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

Kommentare für Variablen in Funktionen und Strukturen

Kommentare für Funktions- und Strukturvariablen werden direkt vor oder hinter den Variablen platziert.

Diese Kommentare werden an die Dateien ODK_Functions.h und <Project>.scl übergeben.

Folgende Regeln gelten für Kommentare **vor** Variablen:

- Kommentare müssen direkt vor der Variablen stehen (ohne Leerzeile)
- Das Ende des Kommentars ist der <InOut-Identifier> der Variablen

Folgende Regeln gelten für Kommentare **nach** Variablen:

- Kommentare müssen nach dem Variablen-Namen stehen (ohne Leerzeile)

Folgende allgemeine Regeln gelten für Kommentare für Variablen:

- Sie können beide Kennungen "//" und "/* */" verwenden, jedoch nicht in Kombination innerhalb eines Kommentars.
- In der Header-Datei wird die gleiche Kommentar-Kennung benutzt ("//" oder "/* */).

Beispiel

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

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

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

5.1.7 Funktionen implementieren

5.1.7.1 Grundsätzliche Anmerkungen

In diesem Kapitel erhalten Sie eine Übersicht über grundsätzliche Themen im Bezug auf die Implementierung von Funktionen in einer Echtzeitumgebung.

- Funktionsaufruf ist zeitlich eingeschränkt

  Da die Funktion synchron aufgerufen wird, muss der Funktionsaufruf dem zeitlichen Ablauf des Zyklus angepasst sein.

- Trace-Funktionalität

  ODK bietet eine Trace-Funktion (Seite 83), um Variablen oder die Ausführung von Funktionen in der Echtzeitumgebung zu prüfen.

- Die Ausführung von synchronen ODK-Funktionen kann durch die Ausführung von höher priorisierten OBs (Seite 79) in der selben CPU unterbrochen werden.

- Anwendungsgröße

  Im Kontext der Echtzeitumgebung ist der Anzahl der ladbaren ODK-Anwendungen limitiert (Seite 56).

- C++Runtime-Biblithek

  Funktionen, die eine Betriebssystem-Funktionalität (threading) brauchen, können nicht genutzt werden

5.1.7.2 Callback-Funktionen

Das ODK-Projekt enthält eine CPP-Datei (**Execute-Datei:** <Projekt>.cpp), um Ihre Funktionen zu definieren. Diese CPP-Datei enthält standardmäßig vorausgefüllte Funktionen. Um nutzbar zu sein, müssen Sie diese nicht zwangsläufig mit zusätzlichem Benutzer-Code befüllen. Die Funktionen dürfen aber unter keinen Umständen gelöscht werden.

Die leere Funktion hat folgenden Code (am Beispiel der Funktion "OnLoad()"):

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

Sie können die folgenden Funktionen in der CPP-Datei definieren:

- OnLoad(): wird nach dem Ladevorgang der ODK-Anwendung aufgerufen
- OnUnload(): wird vor dem Entladevorgang der ODK-Anwendung aufgerufen
- OnRun(): wird beim Übergang der CPU in den Betriebszustand RUN und nach der Funktion OnLoad() aufgerufen
- OnStop(): wird beim Übergang der CPU in den Betriebszustand STOP und vor der Funktion OnUnload() aufgerufen

Funktion "OnLoad()" und "OnUnload()"

Die Funktionen haben einen Rückgabewert des Typs "ODK_RESULT" und geben typischerweise Auskunft über den Status des Werts "ODK_SUCCESS".

Die folgenden Rückgabewerte sind möglich:

Rückgabewert für "ODK_RESULT"	Beschreibung
ODK_SUCCESS = 0x0000	Rückgabewert bei einer erfolgreichen Ausführung der Funktion "OnLoad()" oder "OnUnload()"
0x0001 – 0xEFFF	Ungültige Werte (systemintern)
0xF000 – 0xFFFF ODK_USER_ERROR_BASE = 0xF000	Sie können Ihre eigenen Rückgabewerte definieren. Für die Funktion "OnLoad()" wird der Ladevorgang abgebrochen und die ODK-Anwendung entladen. Für die Funktion "OnUnload()" wird die ODK-Anwendung innerhalb des angegebenen Wertebereichs trotzdem entladen.

Funktion "OnRun()" und "OnStop()"

Die Funktionen haben einen Rückgabewert des Typs "ODK_RESULT" und geben typischerweise Auskunft über den Status des Werts "ODK_SUCCESS".

Die folgenden Rückgabewerte sind möglich:

Rückgabewert für "ODK_RESULT"	Beschreibung
ODK_SUCCESS = 0x0000	Standard-Rückgabewert bei einer erfolgreichen Ausführung der Funktion "OnRun()" oder "OnStop()"
0x0001 – 0xFFFF	Eine direkte Rückmeldung an das Anwenderprogramm ist nicht möglich, da diese Funktionen bei den Übergängen des Betriebszustandens RUN/STOP nicht direkt vom Anwender aufgerufen werden.

5.1.7.3 Eigene Funktionen implementieren

Nachdem Sie die ODK-Schnittstelle in der Datei <Projekt>.odk definiert haben, müssen Sie die ODK-Anwendungsfunktionen in der CPP-Datei editieren.

Vorgehen

Um die ODK-Anwendungsfunktionen zu editieren, gehen Sie folgendermaßen vor:

1. Führen Sie den Build aus, um die Header-Datei <ODK_Functions.h> zu aktualisieren.

2. Öffnen Sie die Datei <Projekt>.cpp, oder legen Sie bei Bedarf eine eigene Source-Datei an.

3. Übernehmen Sie die Funktionsprototypen von <ODK_Functions.h> in die Source-Datei.

Hinweis

Um bei Änderung der Funktionsparameter Schritt 3 zukünftig zu übergehen, verwenden Sie den Define des Funktionsprototyps.

4. Editieren Sie den Code Ihrer ODK-Anwendung in der CPP-Datei.

ODK-Anwendung

Die CPP-Datei enthält standardmäßig eine schematisch dargestellte Funktionsbeschreibung. Diese Beschreibung können Sie entsprechend den Änderungen in der Datei <Projekt>.odk verändern und/oder weitere Funktionsbeschreibungen hinzufügen.

```
#include "ODK_Functions.h"

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

5.1.7.4 Dynamische Speicherverwaltung

Einleitung

ODK-Objekte arbeiten mit einer dynamischen Speicherverwaltung (Heap). Durch die Nutzung der dynamischen Speicherverwaltung werden folgende Anweisungen und Funktionalitäten unterstützt:

- Die Anweisungen new/delete bzw. malloc/free
- STL (StandardTemplateLibrary)
- Software-Exceptions

Standardmäßig ist die Heap-Größe mit 4 KB festgelegt. Die Heap-Größe kann zwischen 4 KB bis zum verfügbaren Speicher der CPU (Seite 92) betragen. Die Heap-Größe ändern Sie in der Datei <Projekt>.odk mit Hilfe der folgenden Parameter:

- HeapSize
- HeapMaxBlockSize

Besonderheiten

Da der verwendete Speicherbereich (Heap) im Hinblick auf Echzeit und zyklische Abarbeitung optimiert ist, weist er einige Besonderheiten auf:

- Blöcke können nur bis zu einer, zur Kompilierzeit des ODK-Objektes, festgelegten Größe allokiert werden.

  Hinweis

  Die maximale Blockgröße können Sie mit dem Parameter HeapMaxBlockSize in <Projekt>.odk festlegen. Das hat allerdings Auswirkungen auf den globalen Speicherverbrauch für ODK-Anwendungen, da für Verwaltungsinformationen folgender Speicher zusätzlich zum eigentlichen Heap benötigt wird:

  size_heap_admin_data = HeapMaxBlockSize * 3

  Beispiel: Bei einer maximalen Blockgröße von 100 KB sind somit, zusätzlich zum Heap, 300 KB globale Daten für dieses Projekt notwendig, in denen die Verwaltung des Heaps hinterlegt ist.

  Weitere Informationen finden Sie unter Umfeld zum Laden oder Ausführen der ODK-Anwendung (Seite 56).

- Blöcke können initial in beliebigen Größen angefordert werden. Werden die Blöcke wieder freigegeben, so werden sie in Freilisten eingetragen. Für alle möglichen Blockgrößen (bis HeapMaxBlockSize) gibt es jeweils eine eigene Freiliste, damit spätere Allokierungen in konstanter Zeit erfüllt werden können.

  Es findet allerdings keine Verschmelzung von benachbarten freigegebenen Blöcken zu einem größeren Block statt!

  Ständig wiederkehrende Anforderungen können somit schneller erfüllt werden als ständig unterschiedliche Anforderungen.

  Beispiel: Der Anwender allokiert nur Blöcke mit 8 Byte, bis der Heap voll ist. Dann gibt er alle wieder frei, sodass der Heap komplett leer ist. Eine Allokation eines Blockes mit der Größe 16 Byte ist dann allerdings nicht mehr möglich, da alle freien Blöcke in der Freiliste für 8 Byte eingetragen sind und eine Verschmelzung nicht möglich ist.

Beispiel

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

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

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

5.1.7.5 Debug (Test)

Sie haben die Möglichkeit selbst einen Test zu schreiben, um Echtzeit-Algorithmen in einer Windows-Umgebung zu debuggen. Somit wird die Qualität des Codes sichergestellt.

Voraussetzung

Für diesen Vorgang benötigen Sie eine Internetverbindung.

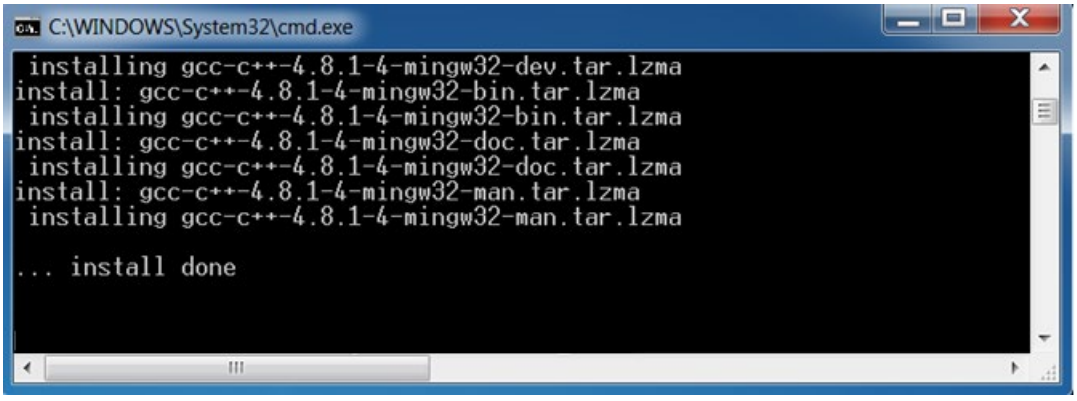
Für diesen Vorgang benötigen Sie Administratorrechte.

Vorgehen vor dem ersten Debug-Vorgang

Um einen Test zu einer Echtzeit-Anwendung in einer Windows-Umgebung durchzuführen, gehen Sie einmalig folgendermaßen vor:

1. Schließen Sie Eclipse.
2. Öffnen Sie den Ordner "bin" Ihrer ODK-Installation.
3. Führen Sie die Datei "MinGW32_Install.cmd" mit dem Kontextmenübefehl "Als Administrator ausführen" aus.

   Ein Textausgabedialog wird geöffnet. Die Windows-Eingabeaufforderung installiert alle notwendigen Komponenten.



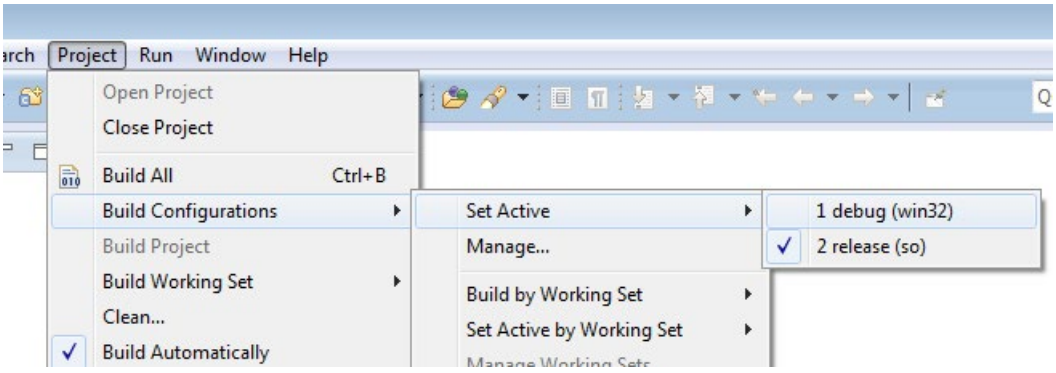
4. Klicken Sie auf eine beliebige Taste.

   MinGW32 ist installiert.

Prinzipielles Vorgehen

Um den Test durchzuführen, gehen Sie folgendermaßen vor:

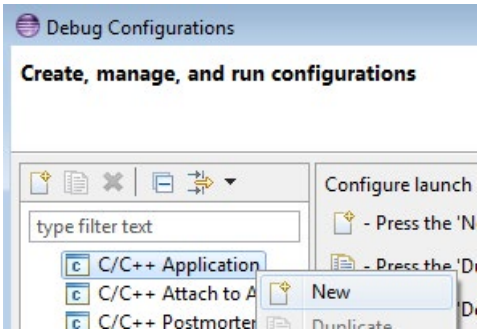
1. Öffnen Sie Ihr Projekt in Eclipse.
2. Ändern Sie die Debug-Umgebung auf "Windows". Wählen Sie dazu im Menü "Project > Build Configurations > Set Active" die Option "debug (win32)".



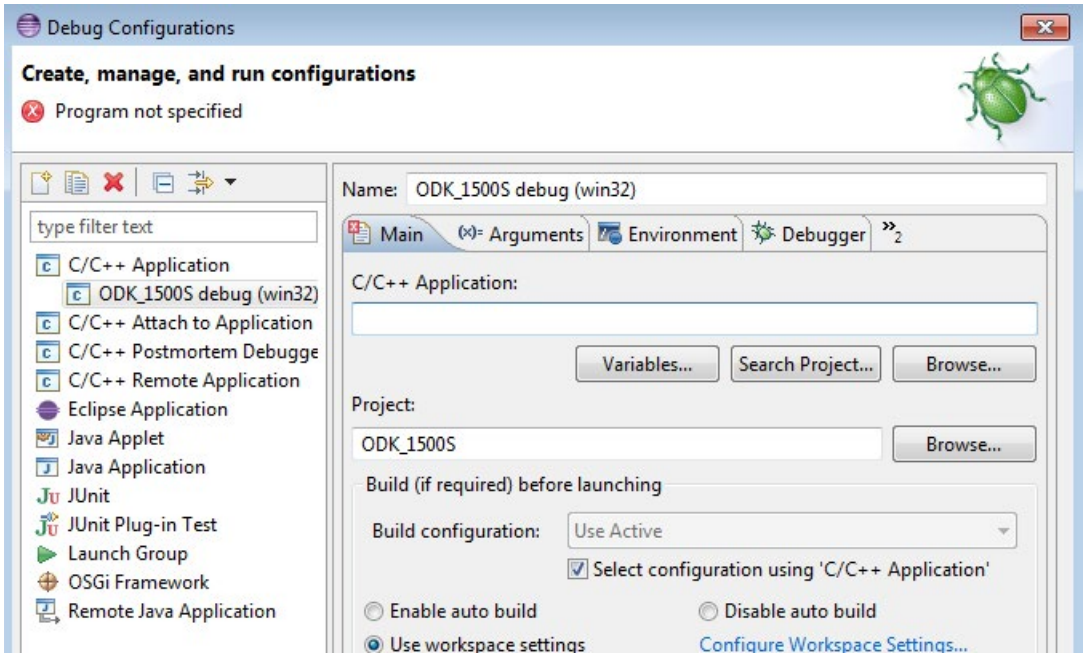
3. Erstellen Sie das Projekt als Debug-Variante. Wählen Sie dazu im Menü "Project " den Befehl "Build Project".

4. Wenn Sie das Projekt zum ersten Mal debuggen, müssen Sie nun die Debug-Konfigurationen einstellen. Fahren Sie ansonsten fort mit Schritt 8.
5. Wählen Sie dazu im Menü "Run" den Befehl "Debug Configurations".

   Der Dialog "Debug Configurations" wird geöffnet.
6. Um eine neue Anwendung anzulegen, markieren Sie den Eintrag "C/C++ Application" und wählen Sie den Kontextmenübefehl "New".

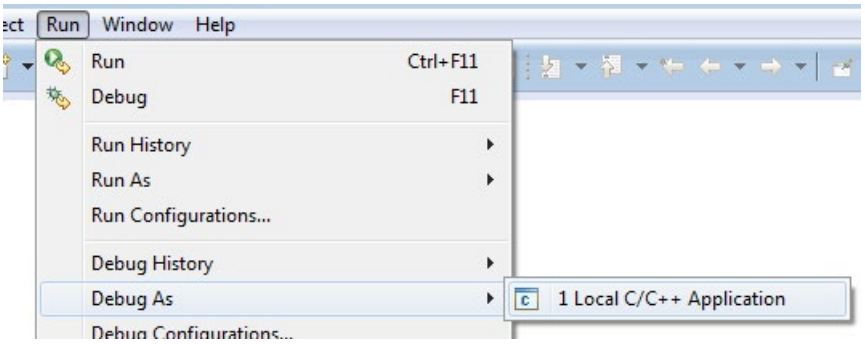


7. Konfigurieren Sie Ihre Testumgebung.
8. Um ihre Anwendung auszuwählen, klicken Sie auf die Schaltfläche "Search Project".



9. Starten Sie den Debug-Vorgang mit der Schaltfläche "Debug".

10. Wenn Sie ihr Projekt erneut debuggen möchten, wählen Sie im Menü "Run > Debug as" den Befehl "Local C/C++ Application".



Ergebnis

Eclipse schlägt einen Wechsel in die Debug-Perspektive vor.

Der Test-Code wird ausgeführt. Der Test-Code für den Test wird nur in der Debug-Umgebung übersetzt und wird in der "main()"-Funktion implementiert. Diese Funktion befindet sich in Datei <Projekt>.cpp

Durch die Funktion "main()"-Funktion haben Sie folgende Möglichkeiten:

- Testdaten werden bereitgestellt und Ergebnisse können überprüft werden.
- Sie können Variablen der Funktion überwachen.
- Sie können Breakpoints nutzen, um die Ausführung zu überprüfen.

Test-Code

Der folgende Beispiel-Code zeigt den standardmäßigen Inhalt der "main()"-Funktion.

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

    // place your test code here
```

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

5.2 ODK-Anwendung in das Zielsystem übertragen

Vorgehen

Übertragen Sie die SO-Datei manuell in das Zielsystem. Nutzen Sie den Datei-Explorer des Webservers der CPU für die Übertragung der ODK-Anwendung.

Um die SO-Datei zu übertragen, gehen Sie folgendermaßen vor:

1. Schalten Sie den Webserver in Ihrem STEP 7-Projekt frei.
2. Öffnen Sie im Browser den Webserver der CPU.
3. Öffnen Sie das Menü "Filebrowser".

4. Öffnen Sie das folgende Verzeichnis als Ablageort, der ODK-Anwendungen: \ODK1500S\

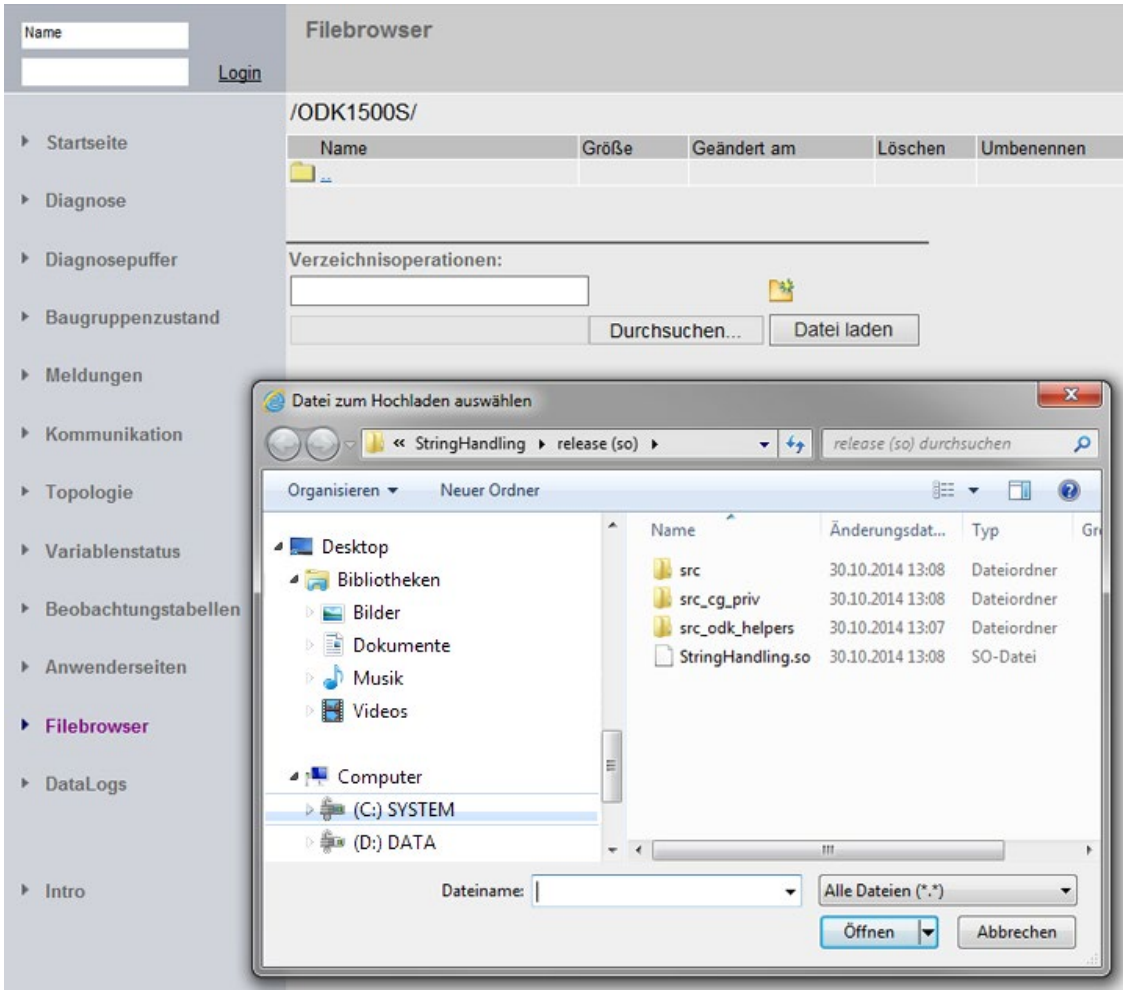


Bild 5-3 Übertragung der SO-Datei über den Datei-Explorer vom Webserver der CPU

5. Klicken Sie auf die Schaltfläche "Durchsuchen".
6. Navigieren Sie im Dateisystem zu der SO-Datei oder kopieren Sie den Ablageort aus den Eigenschaften der SO-Datei in Eclipse.
7. Bestätigen Sie die Übertragung der SO-Datei in den Webserver der CPU mit der Schaltfläche "Datei laden".

Ergebnis

Die SO-Datei wird in den Ladespeicher der CPU übertragen.

Nach einer erfolgreich abgeschlossenen Übertragung wird die SO-Datei durch den Aufruf der Anweisung "<*STEP7Prefix*>_Load" geladen.

5.3 SCL-Datei nach STEP 7 importieren und generieren

Bei der Generierung der Projektdaten werden folgende Dateien erzeugt:

- SCL-Datei für den Import nach STEP 7
- Alle Dateien abhängig von der Konfiguration, z. B. SO-Datei

Wenn STEP 7 auf einem anderen PC als die Entwicklungsumgebung installiert ist, müssen Sie die erzeugte SCL-Datei auf den PC übertragen, auf dem STEP 7 installiert ist.

Voraussetzung

Die Projektdaten wurden generiert.

Vorgehen

Um die SCL-Datei zu importieren und zu kompilieren, gehen Sie folgendermaßen vor:

1. Starten Sie STEP 7.
2. Öffnen Sie Ihr Projekt.
3. Wählen Sie die Projektansicht.
4. Wählen Sie in der Projektnavigation die CPU aus.
5. Wählen Sie den Unterordner "Externe Quellen".

   Der Dialog "Öffnen" wird geöffnet.
6. Navigieren Sie im Dateisystem zu der SCL-Datei, die bei der Generierung der Projektdaten erzeugt wurde, oder kopieren Sie den Ablageort aus den Eigenschaften der SCL-Datei in Eclipse.
7. Bestätigen Sie Ihre Auswahl mit "Öffnen".

   Die SCL-Datei wird importiert. Nach Beendigung des Importvorgangs, wird die SCL-Datei im Ordner "Externe Quellen" angezeigt.
8. Kompilieren Sie die SCL-Datei, bevor Sie die Bausteine in Ihrem Projekt verwenden.
9. Markieren Sie dazu im Unterordner "Externe Quellen" die SCL-Datei.
10. Wählen Sie den Kontextmenübefehl "Bausteine aus Quelle generieren".

Ergebnis

STEP 7 erzeugt die S7-Bausteine auf Basis der ausgewählten SCL-Datei.

Der Funktionsbaustein "GetTrace", der es ermöglicht den Trace-Puffer auszulesen, wird standardmäßig angelegt.

Die erzeugten Bausteine werden nun automatisch in der Projektnavigation im Ordner "Programmbausteine" unterhalb der ausgewählten CPU angezeigt. Sie können die Funktionsbausteine mit dem nächsten Ladevorgang in das Zielgerät laden.

5.4 Funktionen ausführen

5.4.1 Funktionen laden

Einführung

Unabhängig vom Kontext, in dem die ODK-Anwendung ausgeführt wird, besteht der Ladevorgang aus den folgenden Schritten:

- Sie rufen die Anweisung "*<STEP7Prefix>*_Load" im STEP 7-Anwenderprogramm auf.
- Der Ladevorgang erfolgt synchron
- Sobald die Anweisung "*<STEP7Prefix>*_Load" nach dem ersten Aufruf zurückkehrt, ist die ODK-Anwendung geladen.

Hinweis

Laden gleicher ODK-Anwendungen mit geänderter Datei <Projekt>.odk

Wenn Sie eine ODK-Anwendung laden und nachträglich die Datei <Projekt>.odk ändern, wird empfohlen, dass Sie Ihre ODK-Anwendung zuerst entladen, bevor Sie die neu generierte ODK-Anwendung laden. Wenn die Anweisung "*<STEP7Prefix>*_Unload" nicht ausgeführt wird, befinden sich beide ODK-Anwendungen im Speicher. Das kann dazu führen, dass nicht genügend Speicher für die CPU verfügbar ist.

Anweisung "*<STEP7Prefix>*_Load"

Eine ODK-Anwendung wird durch den Aufruf der Anweisung "*<STEP7Prefix>*_Load" im STEP 7-Anwenderprogramm geladen.

*<STEP7Prefix>*_Load	
REQ	DONE
	BUSY
	ERROR
	STATUS

Die folgende Tabelle zeigt die Parameter der Anweisung "*<STEP7Prefix>*_Load":

Sektion	Deklaration	Datentyp	Beschreibung
Input	REQ	BOOL	Eine steigende Flanke aktiviert das Laden der ODK-Anwendung.
Output	DONE	BOOL	Zeigt an, dass die Anweisung das Laden der ODK-Anwendung beendet hat.
Output	BUSY	BOOL	Zeigt an, dass die Anweisung die ODK-Anwendung noch lädt.

Sektion	Deklaration	Datentyp	Beschreibung
Output	ERROR	BOOL	Zeigt an, dass ein Fehler während des Ladens der ODK-Anwendung aufgetreten ist. STATUS gibt Ihnen weitere Informationenüber die mögliche Fehlerursache.
Output	STATUS	INT	Gibt Auskunft über mögliche Fehlerquellen, wenn ein Fehler während des Ladens der ODK-Anwendung aufgetreten ist.

Eingangsparameter

Durch einen Flankenwechsel (0 nach 1) am Eingangsparameter "REQ" wird die Funktion gestartet.

Ausgangsparameter

Die folgende Tabelle zeigt, welche Informationen nach dem Ladevorgang zurückgeliefert werden.

DONE	BUSY	ERROR	STATUS	Bedeutung
0	0	0	0x7000 =28672	Kein aktiver Ladevorgang
1	0	0	0x7100 =28928	Ab V2.0 einer CPU 1500: ODK-Anwendung wurde bereits geladen.
1	0	0	0x0000 =0	Ladevorgang wurde erfolgreich durchgeführt.
0	0	1	0x80A4 =-32604	ODK-Anwendung konnte nicht geladen werden.
			0x80C3 =-32573	ODK-Anwendung konnte nicht geladen werden. Die CPU verfügt aktuell nicht über genügend Ressourcen. Entladen Sie die ODK-Anwendung, bevor Sie eine neue ODK-Anwendung laden oder starten die CPU neu.
			0x8090 =-32624	ODK-Anwendung konnte nicht geladen werden. Bei der Ausführung der Funktion "OnLoad()" ist eine Exception aufgetreten.
			0x8092 =-32622	ODK-Anwendung konnte nicht geladen werden, da der Bibliotheksname ungültig ist.
			0x8093 =-32621	ODK-Anwendung konnte nicht geladen werden, da die ODK-Anwendung nicht gefunden werden konnte. Überprüfen Sie den Dateinamen und Ablageort der Datei.
			0x8095 =-32619	ODK-Anwendung konnte aus den folgenden Gründen nicht geladen werden: • die SO-Datei ist keine ODK-Anwendung. • die CPU unterstützt die verwendete ODK-Version nicht.
			0x8096 =-32618	Die ODK-Anwendung konnte nicht geladen werden, weil die interne Identifizierung bereits von einer anderen geladenen ODK-Anwendung genutzt wird.
			0x8097 =-32617	Bis V1.8 einer CPU 1500: ODK-Anwendung wurde bereits geladen.

DONE	BUSY	ERROR	STATUS	Bedeutung
			0x8098 =-32616	Die ODK-Anwendung konnte nicht geladen werden, weil die ODK-Anwendung derzeit entladen wird.
			0x8099 =-32615	Die ODK-Anwendung konnte nicht geladen werden, weil der Aufruf der Anweisung nicht in einem OB mit niedrigster Priorität erfolgt ist. Verwenden Sie einen Startup OB (z. B. OB100) oder einen Program cycle OB (z. B. OB1).
			0x809B =-32613	Ab V2.0 einer CPU 1500: Die ODK-Anwendung konnte nicht geladen werden und gibt einen ungültigen Wert zurück (erlaubt sind die Werte 0x0000 und 0xF000 - 0xFFFF)
			0xF000 – 0xFFFF =-4096 – -1	Ab V2.0 einer CPU 1500: ODK-Anwendung konnte nicht geladen werden. Bei der Ausführung der Funktion "OnLoad()" ist ein Fehler aufgetreten.

5.4.2 Funktionen aufrufen

Einführung

Nachdem die ODK-Anwendung geladen wurde, können Sie die ODK-Funktionen über Ihr STEP 7-Anwenderprogramm ausführen. Der Aufruf erfolgt über die entsprechende Anweisung "*<STEP7Prefix>*SampleFunction".

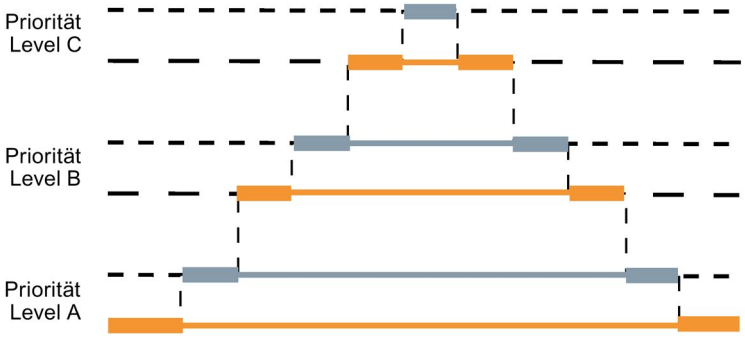


Bild 5-4 Funktionen aufrufen

Die Ausführung von synchronen ODK-Funktionen kann durch die Ausführung von höher priorisierten OBs in der selben CPU unterbrochen werden:

- Aufruf einer anderen ODK-Funktion
- Aufruf der selben ODK-Funktion

  Achten Sie daher bei der Erstellung Ihrer ODK-Anwendung darauf, die Funktionsaufrufe wiedereintrittsfähig (re-entrant) zu implementieren oder die parallele Ausführung zu vermeiden.

Implementieren Sie maximal drei parallele Aufrufe. Implementieren Sie mehr als drei parallele Aufrufe, gibt die ODK-Funtkion folgenden Status zurück: 0x80C3

Anweisung "*<STEP7Prefix>*SampleFunction"

Eine ODK-Anwendung wird durch die Anweisung "*<STEP7Prefix>*SampleFunction" aufgerufen.

*<STEP7Prefix>*SampleFunction	
myInt	STATUS
myReal	myBool

Die folgende Tabelle zeigt die Parameter der Anweisung "*<STEP7Prefix>*SampleFunction":

Sektion	Deklaration	Datentyp	Beschreibung
Automatisch generierte Parameter			
Output	STATUS	INT	Dieser Output-Wert gibt Auskunft über mögliche Fehlerquellen, wenn ein Fehler während der Ausführung der ODK-Anwendung aufgetreten ist.
Nutzer-definierte Parameter			
Input	myInt		Nutzer-definierte Eingangsvariablen
InOut	myReal		Nutzer-definierte Eingangs-Ausgangsvariablen
Output	myBool		Nutzer-definierte Ausgangsvariablen

Ausgangsparameter

Die Anweisung "*<STEP7Prefix>*SampleFunction" hat nur den Output-Parameter "STATUS".

Die folgende Tabelle zeigt, welche Informationen für den Output-Parameter nach der Ausführung zurückgeliefert werden.

STATUS	Bedeutung
0x0000 – 0x6FFF =0 – 28671	Funktion wurde ausgeführt und gibt einen Wert zwischen 0x0000 und 0x6FFF zurück. (ODK_SUCCESS = 0x0000)
0x80A4 =-32604	ODK-Anwendung konnte aus den folgenden Gründen nicht ausgeführt werden: • Nach dem Ausführung der Funktion wurde ein StackOverflow festgestellt. Um Folgefehler zu vermeiden, entladen Sie die ODK-Anwendung. Der Entwickler der ODK-Anwendung muss vermeiden, dass der Stack überschritten wird. • Die Anweisung "<STEP7Prefix>_Unload" wurde während einer Funktionsausführung aufgerufen. Die Funktionsausführung wurde abgebrochen und sofort beendet. Es wird kein Rückgabewert an die CPU übermittelt. Warten Sie bis die Anweisung "<STEP7Prefix>_Unload" beendet ist. Laden Sie dann die ODK-Anwendung erneut.
0x80C3 =-32573	ODK-Anwendung konnte nicht ausgeführt werden. Die CPU verfügt aktuell nicht über genügend Speicher. Achten Sie auf die Anzahl der maximalen parallelen Aufrufe (SyncCallParallelCount).
0x8090 =-32624	ODK-Anwendung konnte nicht ausgeführt werden. Bei der Ausführung ist eine Exception aufgetreten. Jede unbehandelte Exception verringert die verfügbare HeapSize. Eine unbehandelte Exception kann die ODK-Anwendung beschädigen, wodurch diese nicht mehr für weitere Aufrufe genutzt werden kann. Die ODK-Anwendung muss entladen werden. Der Entwickler der ODK-Anwendung muss die Exception behandeln und einen anwendungspezifischen Fehlerwert liefern.

STATUS	Bedeutung
0x8091 =-32623	ODK-Anwendung konnte nicht ausgeführt werden. Während des Funktionsaufrufs kam es zu einem "STOP".
0x8096 =-32618	ODK-Anwendung konnte nicht ausgeführt werden, da die ODK-Anwendung nicht geladen wurde oder das Entladen noch nicht abgeschlossen ist.
0x8098 =-32616	ODK-Anwendung konnte nicht ausgeführt werden, da die ODK-Anwendung einen Unterschied zu den ODK-Anweisungen (FBs) in STEP 7 aufweist: • älter • neuer • unterschiedliche Parameter
0x8099 =-32615	ODK-Anwendung konnte nicht ausgeführt werden, da die maximale Menge an Eingangsdaten (32 KB) überschritten wurde (Deklarationen mit "In" und "InOut")
0x809A =-32614	ODK-Anwendung konnte nicht ausgeführt werden, da die maximale Menge an Ausgangsdaten (32 KB) überschritten wurde (Deklarationen mit "Out" und "InOut")
0x809B =-32613	Die Funktion gibt einen ungültigen Wert zurück (erlaubt ist ein Wert zwischen 0x0000 und 0x6FFF; 0xF000 und 0xFFFF)
0xF000 – 0xFFFF =-4096 – -1	Ab V2.0 einer CPU 1500: Die Funktion konnte nicht ausgeführt werden und gibt einen Wert zwischen 0xF000 und 0xFFFF zurück. (ODK_USER_ERROR_BASE = 0xF000)

5.4.3 Funktionen entladen

Einführung

Die ODK-Anwendung wird mit dem Aufruf der Anweisung "*<STEP7Prefix>*_Unload" entladen. Der Aufruf erfolgt aus dem STEP 7-Anwenderprogramm.

Die ODK-Anwendung wird zusätzlich zu diesem Aufruf automatisch aus folgenden Gründen entladen.

- Beenden der CPU
- Durch Urlöschen der CPU

Unabhängig vom Kontext, in dem die ODK-Anwendung ausgeführt wird, besteht der Entladevorgang aus den folgenden Schritten:

- Sie rufen die Anweisung "*<STEP7Prefix>*_Unload" im STEP 7-Anwenderprogramm auf.
- Ab jetzt können keine neuen Ausführungen (Executes) mehr für diese ODK-Anwendung ausgeführt werden. Noch aktive Ausführungen werden abgebrochen. Die Funktionsausführung wird abgebrochen und sofort beendet. Es wird kein Rückgabewert an die CPU übermittelt.
- Der Host ruft die Funktionen "OnStop()" und "OnUnload()" auf.
- Die ODK-Anwendung wird entladen.

Anweisung "*<STEP7Prefix>*_Unload"

Eine ODK-Anwendung wird durch den Aufruf der Anweisung "*<STEP7Prefix>*_Unload" im STEP 7-Anwenderprogramm entladen.

*<STEP7Prefix>*_Unload	
REQ	DONE
	BUSY
	ERROR
	STATUS

Die folgende Tabelle zeigt die Parameter der Anweisung "*<STEP7Prefix>*_Unload":

Sektion	Deklaration	Datentyp	Beschreibung
Input	REQ	BOOL	Eine steigende Flanke aktiviert das Entladen der ODK-Anwendung.
Output	DONE	BOOL	Zeigt an, dass die Anweisung das Entladen der ODK-Anwendung beendet hat.
Output	BUSY	BOOL	Zeigt an, dass die Anweisung die ODK-Anwendung noch entlädt.
Output	ERROR	BOOL	Zeigt an, dass ein Fehler während des Entladens der ODK-Anwendung aufgetreten ist. STATUS gibt Ihnen weitere Informationen über die mögliche Fehlerursache.
Output	STATUS	INT	Gibt Auskunft über mögliche Fehlerquellen, wenn ein Fehler während des Entladens der ODK-Anwendung aufgetreten ist.

Eingangsparameter

Durch einen Flankenwechsel (0 nach 1) am Eingangsparameter "REQ" wird die Funktion gestartet.

Ausgangsparameter STATUS

Die folgende Tabelle zeigt, welche Informationen nach dem Entladevorgang zurückgeliefert werden.

DONE	BUSY	ERROR	STATUS	Bedeutung
0	0	0	0x7000 =28672	Kein aktiver Entladevorgang
0	1	0	0x7001 =28673	Entladevorgang wird durchgeführt, erster Aufruf
0	1	0	0x7002 =28674	Entladevorgang wird durchgeführt, fortlaufender Aufruf
1	0	0	0x0000 =0	Entladevorgang wurde erfolgreich durchgeführt
0	0	1	0x80A4 =-32604	ODK-Anwendung konnte nicht entladen werden. Bei der Ausführung der Funktion "OnUnload()" ist ein Kommunikationsfehler zwischen der CPU und ODK aufgetreten.
			0x80C3 =-32573	ODK-Anwendung konnte nicht entladen werden. Die CPU verfügt aktuell nicht über genügend Speicher.

DONE	BUSY	ERROR	STATUS	Bedeutung
			0x8090 =-32624	ODK-Anwendung konnte nicht entladen werden. Bei der Ausführung der Funktion "OnUnload()" ist eine Exception aufgetreten.
			0x8096 =-32618	ODK-Anwendung konnte nicht entladen werden, da die ODK-Anwendung nicht geladen wurde oder das Entladen noch nicht abgeschlossen ist.
			0x809B =-32613	Ab V2.0 einer CPU 1500: Die ODK-Anwendung konnte entladen werden und gibt einen ungültigen Wert zurück (erlaubt sind die Werte 0x0000 und 0xF000 - 0xFFFF)
			0xF000 – 0xFFFF =-4096 – -1	Ab V2.0 einer CPU 1500: ODK-Anwendung konnte entladen werden. Bei der Ausführung der Funktion "OnLoad()" ist im CCX-Obekt ein Fehler aufgetreten.

5.4.4 Trace-Puffer auslesen

ODK bietet eine Trace-Funktion, um Variablen oder die Ausführung von Funktionen in der Echtzeitumgebung zu prüfen. Die Trace-Funktion unterstützt die folgenden Elemente:

- einen integrierten Trace-Puffer für jede ODK-Anwendung
- eine Anweisung "ODK_TRACE", die Sie zu Ihrem Code hinzufügen können
- einen Funktionsbaustein "GetTrace", der es ermöglicht den Trace-Puffer auszulesen

Anweisung "ODK_TRACE"

Wenn Sie die Anweisung "ODK_TRACE" definieren, wird sie auch kompiliert und ausgeführt. Wenn Sie den Parameter Trace=on in der Datei <Projekt>.odk definieren, wird die Anweisung automatisch mit folgendem Code definiert:

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

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

Durch den Aufruf der Anweisung wird im Trace-Puffer ein Eintrag erstellt.

Wenn Sie den Parameter Trace=off in der Datei <Projekt>.odk definieren, werden keine Trace-Daten geschrieben.

Beim Auftritt einer Exception werden automatisch Trace-Daten geschrieben.

Trace-Puffer auslesen

Mit dem Funktionsbaustein "GetTrace" habe Sie die Möglichkeit den Trace-Puffer auszulesen. Die Einträge des Trace-Puffers können auf folgende Arten ausgelesen werden:

- durch eine Variablen-Tabelle im Webserver der CPU
- durch eine Variablen-Tabelle in STEP 7 (online)
- auf einem HMI-Display

Der Funktionsbaustein ist in Standard-CPP-Datei "<Projekt>.cpp" enthalten.

GetTrace	
TraceCount	STATUS

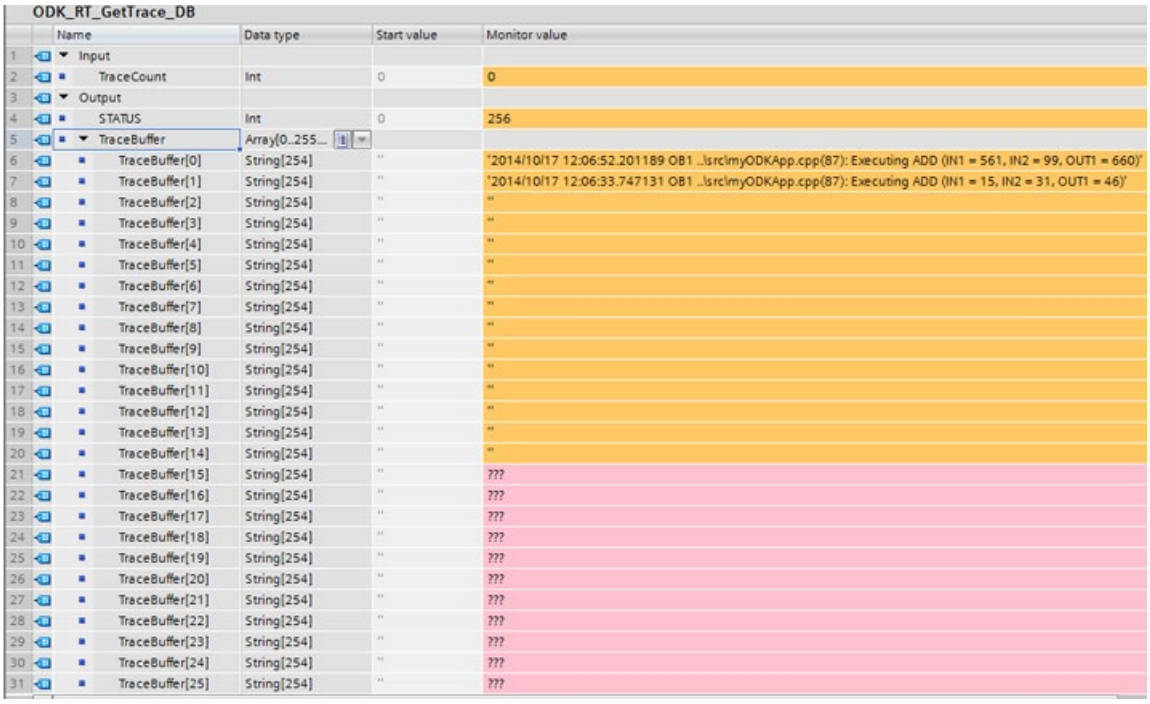
Die folgende Tabelle zeigt die Parameter des Funktionsbausteins "GetTrace":

Sektion	Deklaration	Datentyp	Beschreibung
Output	STATUS	INT	Anzahl der tatsächlich gelesenen Trace-Einträge
Input	TraceCount	INT	Anzahl der Trace-Einträge, die gelesen werden sollen
Output	TraceBuffer	Array [0..255] of String[125]	Trace String Array für den Nutzer jeder Trace String besteht aus: • Datum • Uhrzeit • OB-Nummer • Dateiname • Zeilennummer • Trace-Text (der vom Anwender implementierte Trace)

Definieren Sie den Funktionsbaustein in der SCL-Datei wie folgt:

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

Wenn Sie den Funktionsbaustein "GetTrace" in STEP 7 aufrufen, sieht der Instanzbaustein folgendermaßen aus:



ODK_RT_GetTrace_DB

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

5.5 Post Mortem-Analyse

5.5.1 Einleitung

Mit Hilfe der Post Mortem-Analyse können Sie das System nach einer Exception auswerten. Die Post-Mortem-Dateien bilden eine Momentaufnahme zum Zeitpunkt der Exception ab.

Mit der Post Mortem-Analyse kann der Speicherabzug analysiert werden. Dieser enthält z. B.:

- Register
- Stack
- lokale/globale Daten
- Übergabeparameter

Eine Exception kann beispielsweise durch folgende Fälle ausgelöst werden:

- unerlaubtes Kommando ausführen
  - Division durch Null
  - Zugriff auf geschützten Speicher
- eine durch die Anweisung "throw" ausgelöste, aber nicht durch die Anweisung "try...catch" behandelte, Exception

Das Ziel einer Post Mortem-Analyse ist es, den Fehler innerhalb der ODK-Anwendung, der die Exception verursacht hat, zu finden.

ACHTUNG
Exception beeinflusst die Zykluszeit
Wenn in Ihrer Anwendung eine Exception auftritt, wird der komplette Anwendungsspeicher zwischengepuffert. Dies kann einige Millisekunden dauern und beeinflusst die Zykluszeit.

Erst bei einem Wechsel des Betriebszustandes der CPU von RUN nach STOP werden die Post Mortem-Dateien zur Momentaufnahme der ersten Exception erstellt. Diese können Sie zur nachfolgenden Post Mortem-Analyse verwenden. Die Ablage erfolgt in folgendem Verzeichnis: <Ladespeicher>/ODK1500S

Folgende Dateien werden bei diesem Vorgang erstellt bzw. überschrieben und können z. B. über den Webserver heruntergeladen werden:

- <Projekt>.ed

  Binärabzug des Shared-Objects, in dem die Exception aufgetreten ist

- <Projekt>.es

  Stack zum Zeitpunkt der Exception

- <Projekt>.er

  Skript zum Wiederherstellen der Momentaufnahme zum Zeitpunkt der Exception

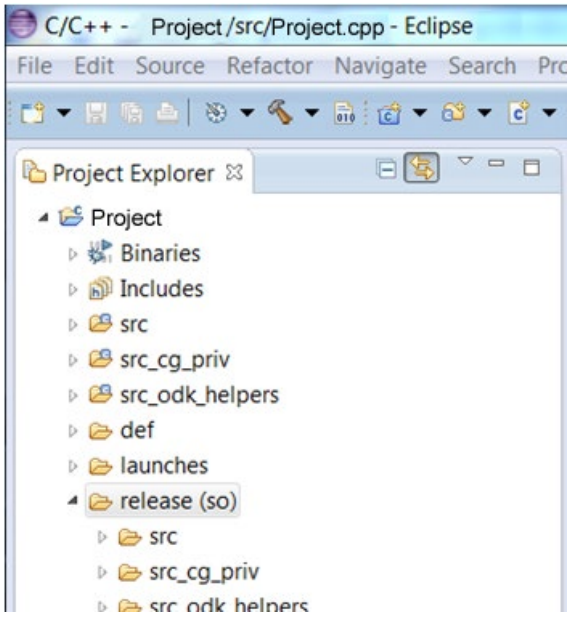
ACHTUNG
Nicht genügend Ladespeicher
Wenn der Ladespeicher nicht ausreicht, werden die Post Mortem-Dateien nicht korrekt gespeichert.
Stellen Sie sicher, dass für Ihre Anwendungen genügend Ladespeicher zur Verfügung steht.

5.5.2 Post Mortem-Analyse durchführen

Vorgehen

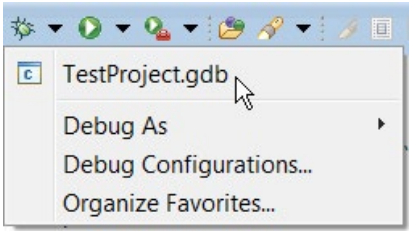
Um eine Post Mortem-Analyse durchzuführen, gehen Sie folgendermaßen vor:

1. Öffnen Sie Eclipse.
2. Laden Sie die Post Mortem-Dateien mit dem Webserver auf den Engineering-PC. Laden Sie diese Dateien in das gleiche Verzeichnis, in dem auch die SO-Datei liegt.

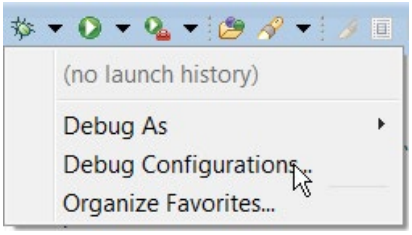


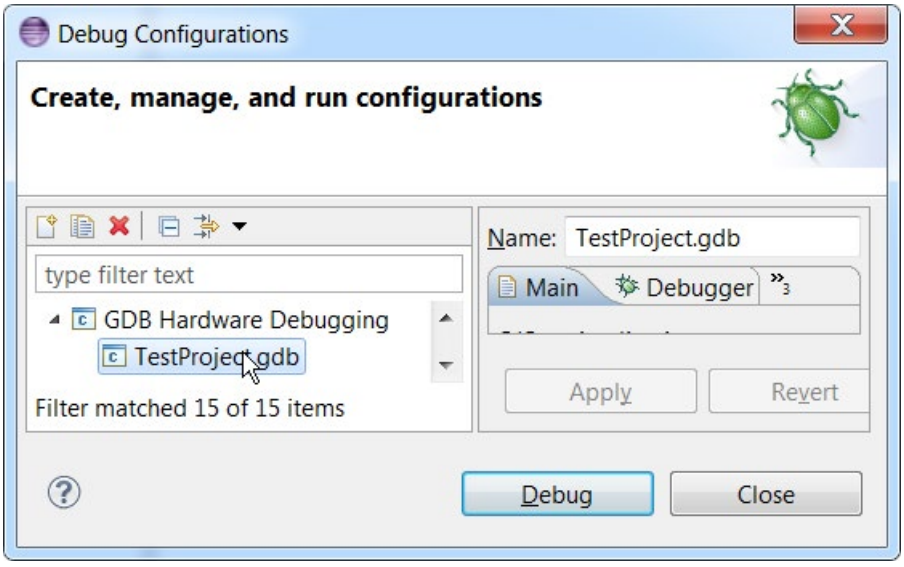
3. Wählen sie das gewünschte Projekt aus.

4. Starten Sie den Debug-Vorgang über eine der folgenden Möglichkeiten:
   - über die Favoriten



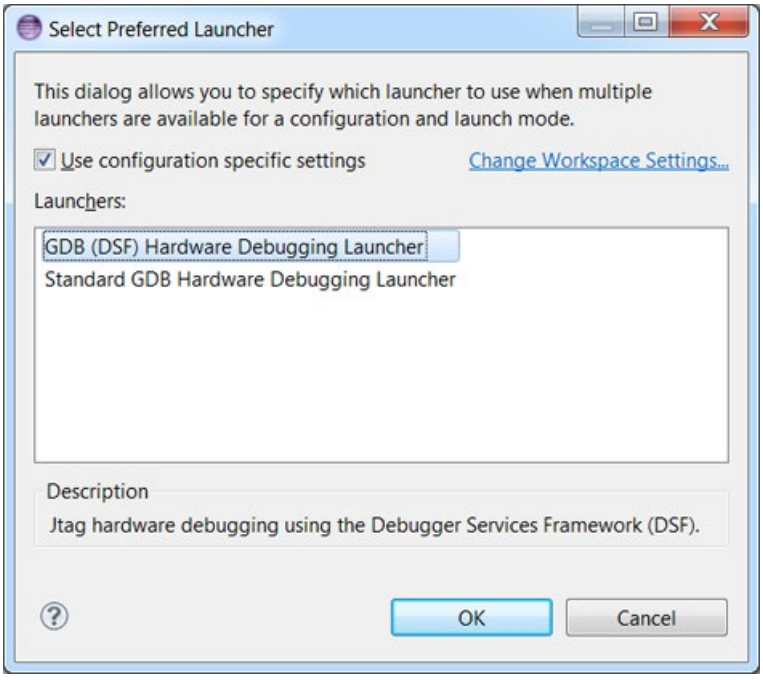
   - über "Debug Configurations"





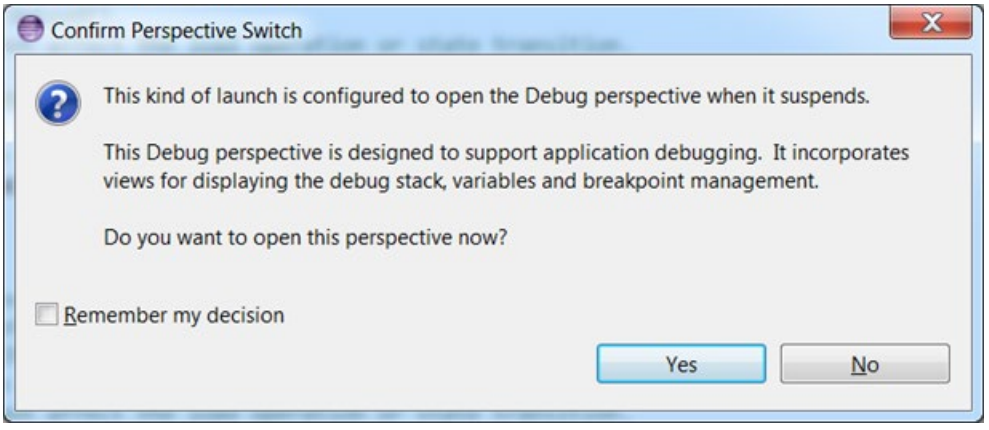
Wenn Sie zum ersten Mal einen Debug-Vorgang starten, wird ein Dialog geöffnet, der Sie dazu auffordert, die gewünschte Launch-Umgebung auszuwählen.

Wählen Sie den Eintrag "GDB (DSF) Hardware Debugging Launcher".

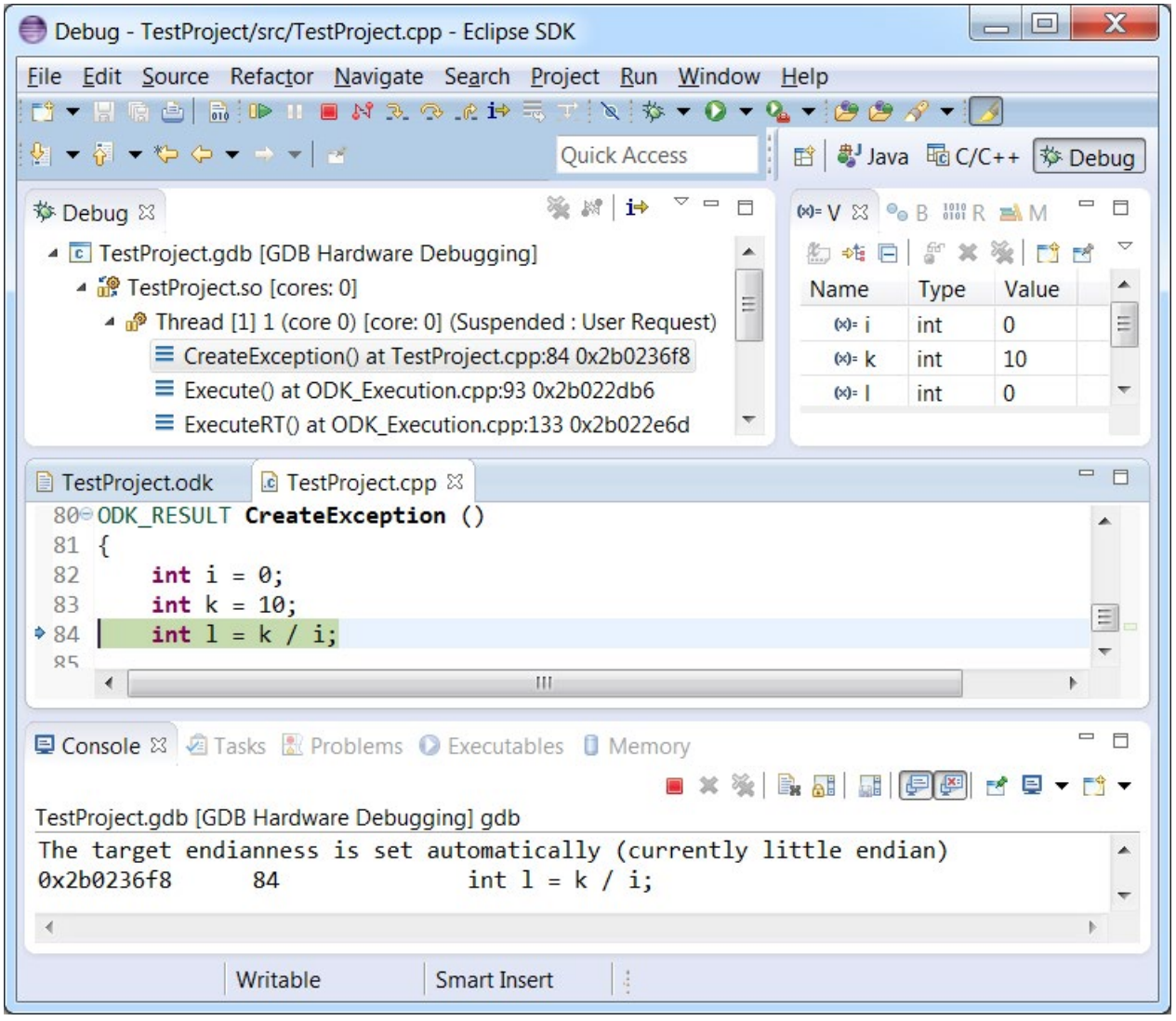


Ein Dialog wird geöffnet, der Ihnen den Fortschritt des Ladevorgangs des Post Mortem-Images anzeigt. Der Ladevorgang kann abhängig von der Größe des Post Mortem-Images einige Minuten dauern.

5. Wählen Sie die gewünschte Debug-Ansicht.



6. Führen sie den Debug-Vorgang durch.



Beispielprojekte nutzen

6

Um den Einstieg in die ODK-Thematik zu erleichtern, bietet Ihnen ODK 1500S Beispielprojekte für beide Entwicklungsumgebungen. Die Beispielprojekte bestehen aus den folgenden Elementen:

- einem Projekt für Microsoft Visual Studio oder Eclipse
- eine kompilierte Binär- und SCL-Quelle, damit Sie die Beispielprojekte sofort testen können
- ein STEP 7-Beispielprojekt

Ablageort der Beispielprojekte

Die Beispielprojekte liegen Im Internet (https://support.industry.siemens.com/cs/document/106192387/simatic-odk-1500s-beispiele?dti=0&lc=de-WW) zum Download bereit.

Beispielprojekte nutzen

Um die Beispielprojekte zu nutzen, gehen Sie folgendermaßen vor:

1. Übertragen Sie die Beispielprojekte auf die Festplatte Ihres PCs.
2. Übertragen Sie die DLL- bzw. SO-Datei in das Zielsystem.

Anhang

A

A.1 Rahmenbedingungen der ODK-Anwedungen

A.1.1 Anzahl der ladbaren ODK-Anwendungen

Sie können für Windows- und Echtzeit-Umgebung insgesamt bis zu 32 ODK-Anwendungen laden.

Mengengerüste für ODK-Anwedungen:

- ODK-Anwendungen für die Windows-Umgebung:
  - bis zu 32 parallele Funktionsaufrufe (gesamt)
  - bis zu 1 MB Eingangs- bzw. Ausgangsdaten (gesamt)
  - bis zu 1 MB Eingangsdaten pro Funktionsaufruf
  - bis zu 1 MB Ausgangsdaten pro Funktionsaufruf

  Hinweis

  Die Zuordnung von Speicher für Ein-/Ausgangsparameter erfolgt dynamisch je nach der benötigten Menge. Dabei erfolgt die Zuordnung in Blöcken von je 8 KB.

- ODK-Anwendungen für die Echtzeit-Umgebung:
  - parallele Funktionsaufrufe in einer ODK-Anwendung definert durch den Parameter "SyncCallParallelCount"
  - bis zu 26 parallele Funktionsaufrufe (gesamt)
  - bis zu 1 MB Eingangs- und Ausgangsdaten pro Funktionsaufruf

Im Kontext der Echtzeit-Umgebung ist der verfügbare Speicher zum Laden von ODK-Anwendungen limitiert. Die folgende Tabelle gibt einen Überblick über den verfügbaren Speicher der unterschiedlichen CPUs zum Laden von ODK-Anwendungen:

CPU	verfügbarer Speicher zum Laden von ODK-Anwendungen	maximale Größe der SO-Datei
CPU 1505SP (F)	10 MB	3,8 MB
CPU 1507S (F)	20 MB	5,8 MB
CPU 1518-4 PN/DP ODK (F)	20 MB	5,8 MB

Im Kontext der Echtzeit-Umgebung gibt es des Weiteren folgende Einschränkungen:

- SO-Dateiname darf maximal 56 Zeichen lang sein

A.1.2 Kompatibilität

Wenn Sie eine ODK-Version V2.0 verwenden ist Folgendes zu beachten:

- Ein mit ODK-Version < V2.0 erstelltes ODK-Projekt ist nicht kompatibel. Sie müssen Ihr ODK-Projekt in der Version V2.0 neu anlegen.
- Eine mit ODK-Version < V2.0 erstellte ODK-Anwendung ist kompatibel zu neueren CPUs.

A.2 Syntax Interface-Datei <Projekt>.odk

A.2.1 Datentypen

Der Datentyp definiert den Typ einer Variablen. Die folgende Tabelle definiert die möglichen Datentypen und ihre Darstellungsweise in C++ oder STEP 7:

- **Elementare Datentypen:**

ODK-Datentyp	SIMATIC-Datentyp	C++-Datentyp	Beschreibung
ODK_DOUBLE	LREAL	double	64-Bit Floating Point, IEEE 754
ODK_FLOAT	REAL	float	32-Bit Floating Point, IEEE 754
ODK_INT64	LINT	long long	64-Bit signed integer
ODK_INT32	DINT	long	32-Bit signed integer
ODK_INT16	INT	short	16-Bit signed integer
ODK_INT8	SINT	char	8-Bit signed integer
ODK_UINT64	ULINT	unsigned long long	64-Bit unsigned integer
ODK_UINT32	UDINT	unsigned long	32-Bit unsigned integer
ODK_UINT16	UINT	unsigned short	16-Bit unsigned integer
ODK_UINT8	USINT	unsigned char	8-Bit unsigned integer
ODK_LWORD	LWORD	unsigned long long	64-Bit Bitstring
ODK_DWORD	DWORD	unsigned long	32-Bit Bitstring
ODK_WORD	WORD	unsigned short	16-Bit Bitstring
ODK_BYTE	BYTE	unsigned char	8-Bit Bitstring

ODK-Datentyp	SIMATIC-Datentyp	C++-Datentyp	Beschreibung
ODK_BOOL	BOOL	unsigned char	1-Bit bitstring, verbleibende Bits (1..7) sind leer
ODK_LTIME	LTIME	unsigned long long	64-Bit Dauer in Nanosekunden
ODK_TIME	TIME	unsigned long	32-Bit Dauer in Millisekunden
ODK_LDT	LDT	unsigned long long	64-Bit Datum und Uhrzeit des Tages in Nanosekunden
ODK_LTOD	LTOD	unsigned long long	64-Bit Uhrzeit des Tages in Nanosekunden seit Mitternacht
ODK_TOD	TOD	unsigned long	32-Bit Uhrzeit des Tages; in Millisekunden seit Mitternacht
ODK_WCHAR	WCHAR	wchar_t	**Nur für Windows:** 16-Bit-Zeichen
ODK_CHAR	CHAR	char	8-Bit-Zeichen

- **Komplexe Datentypen:**

ODK-Datentyp	SIMATIC-Datentyp	C++-Datentyp	Beschreibung
ODK_DTL	DTL	struct ODK_DTL	Struktur für Datum und Uhrzeit
ODK_S7STRING	STRING	unsigned char	Zeichen-String (8-Bit-Zeichen) mit Länge max. und act. (2xUSINT)
ODK_S7WSTRING	WSTRING	unsigned short	**Nur für Windows:** Zeichen-String (16-Bit-Zeichen) mit Länge max. und act. (2xUINT)
ODK_CLASSIC_DB	VARIANT	struct ODK_CLASSIC_DB	Klassik-DB (global oder basierend auf UDT)
[ ]	ARRAY	[ ]	Bereich von gleichen Datentypen. Die maximale Anzahl von Array-Elementen beträgt 2^{20} (=1.048.576). Sie können alle Datentypen außer ODK_CLASSIC_DB als Array nutzen.

- **Benutzerdefinierte Datentypen:**

  Benutzerdefinierte Datentypen (UDT) strukturierte Daten, insbesondere die Namen und die Datentypen dieser Komponente und deren Reihenfolge.

  Ein benutzerdefinierter Datentyp kann in der Oberflächenbeschreibung mit dem Schlüsselwort "ODK_STRUCT" definiert werden.

  Beispiel

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

  Die folgenden Syntaxregeln gelten für die Struktur:

  – Sie können die Struktur in mehrere Zeilen aufteilen.
  – Die Strukturdefinition muss ein Semikolon abschließen.
  – Tabulatoren und Leerzeichen zwischen den Elementen sind in einer beliebigen Anzahl erlaubt.
  – Sie dürfen keine Schlüsselwörter für die erzeugte Sprache verwenden (z. B. "int" als Variablennamen).

Der Datentyp ODK_CLASSIC_DB darf nur mit dem InOut-Identifier [IN] und [INOUT] benutzt werden.Wird ein Parameter des der Datentyps ODK_CLASSIC_DB mit dem InOut-Identifier [IN] oder [INOUT] verwendet, darf kein weiterer Parameter, egal welchen Datentyps, mit dem gleichen InOut-Identifier verwendet werden.

A.2.2 Parameter

Die Parameter der Datei <Projekt>.odk sind unterschiedlich:

- Entwicklung einer ODK-Anwendung für die Windows-Umgebung
- Entwicklung einer ODK-Anwendung für die Echtzeit-Umgebung

Parameter für die Windows-Umgebung

Die Definition der Parameter muss innerhalb einer Code-Zeile erfolgen.

```
<ParameterName>=<Value> // optional comment
```

Die Datei <Projekt>.odk unterstützt folgende Parameter:

Parameter	Wert	Beschreibung
Context	user	Definiert, dass die ODK-Anwendung im Kontext eines Windows-Nutzers geladen wird.
	system	Definiert, dass die ODK-Anwendung im Kontext des Windows-Systems geladen wird.
STEP7Prefix	<String>	Beschreibt die Zeichenkette, die Ihren Funktionen vorangestellt und nach dem Import der SCL-Datei in STEP 7 dargestellt wird. Erlaubt sind folgende Zeichen: {A...Z, a…z, 1…9, -, _}

Parameter für die Echtzeit-Umgebung

Die Definition der Parameter muss innerhalb einer Code-Zeile erfolgen.

```
<ParameterName>=<Value> // optional comment
```

Die Datei <Projekt>.odk unterstützt folgende Parameter:

Parameter	Wert	Beschreibung
Context	realtime	Definiert, dass die ODK-Anwendung im Kontext der Echtzeitumgebung geladen wird.
Trace	on	Definiert die Trace-Funktion in der ODK-Anwendung. In diesem Fall benötigt die ODK-Anwendung 32 KB zusätzlichen Speicher als Trace-Puffer. Für die Nutzung in STEP 7 wird standardmäßig ein Funktionsbaustein "GetTrace" erstellt.
	off	Ein Funktionsbaustein "GetTrace" wird erstellt. Der Trace-Puffer enthält nur einen Trace-Eintrag mit dem Inhalt: trace is off.
HeapSize	[4…<verfügbarer Speicher der CPU> (Seite 92)]k	Definiert eine Speichermenge in KB, die als Heap für Echtzeitanwendungen genutzt wird.
HeapMaxBlockSize	[8…<HeapSize>]	Definiert die Speichergröße in Bytes, die auf ein mal allokiert werden kann.
STEP7Prefix	<String>	Beschreibt die Zeichenkette, die Ihren Funktionen vorangestellt und nach dem Import der SCL-Datei in STEP 7 dargestellt wird. Erlaubt sind folgende Zeichen: {A...Z, a…z, 1…9, -, _}

A.3 Fehlermeldungen des Code-Generators

Der Code-Generator erzeugt folgende Fehlermeldungen:

Datei-Fehler:

Fehler-nummer	Fehlermeldung	Mögliche Abhilfe
100	'<Project>.odk' is missing	Benennen Sie die Datei in <Projekt>x.odk um.
101	Context is missing in resorce file	Die Resource-Datei (.rc) ist fehlerhaft.
102	resource file '...' is missing	Die Resource-Datei (.rc) fehlt.
103	'...' write protected	Die angezeigte Datei ist schreibgeschützt.

Parameter-Fehler:

Fehler-nummer	Fehlermeldung	Mögliche Abhilfe
200	parameter '...' is not allowed for current context	Der angezeigte Parameter ist hier nicht erlaubt.
201	missing '...' definition	Der angezeigte Parameter (Seite 54) ist nicht definiert.
202	more than one defition for '...'	Es gibt mehr als eine Definition für den angezeigten Parameter (Seite 54).
203	Context has to be one of 'user' or 'system' for Microsoft Visual Studio	Wählen Sie für Visual Studio den Kontext "system" oder "user".
204	Context has to be 'realtime' for Eclipse	Wählen Sie für Eclipse den Kontext "relatime".
205	Trace has to be on or off	Der Parameter "Trace" muss den Wert "on" oder "off" haben (nur für Echtzeit-Umgebung).
206	STEP7Prefix must not be longer than 120 characters	Der STEP7-Prefix darf nicht mehr als 120 Zeichen beinhalten.
207	HeapSize has to be interval of [4…100000]k	Stellen Sie sicher, dass der Parameter HeapSize im Wertebereich [4…100000]k liegt.
208	HeapMaxBlockSize has to be interval of [8…<HeapSize>]	Stellen Sie sicher, dass der Parameter HeapMaxBlockSize im Wertebereich [8…<HeapSize>] liegt.
209	SyncCallDataSize must be interval of [1...1024]k	Stellen Sie sicher, dass der Parameter SyncCallDataSize im Wertebereich [1…1024]k liegt.
210	SyncCallStackSize must be interval of [1...1024]k	Stellen Sie sicher, dass der Parameter SyncCallStackSize im Wertebereich [1…1024]k liegt.
211	SyncCallParallelCount must be interval of [1...9]	Stellen Sie sicher, dass der Parameter SyncCallParallelCount im Wertebereich [1…9] liegt.

Syntax-Fehler:

Fehler-nummer	Fehlermeldung	Mögliche Abhilfe
500	unexpected end-of-file found	Schließen Sie die Datei immer mit einem Semikolon ab.
501	'...' should be alpha numeric	Erlaubt sind folgende Zeichen: a - z, A - Z, 0 - 9, _ Umlaute sind nicht erlaubt
502	'...' should be numeric	Erlaubt sind folgende Zeichen: 0 - 9

Fehler-nummer	Fehlermeldung	Mögliche Abhilfe
503	'...' undefined keyword	Verwenden Sie nur die Schlüsselwörter [IN], [OUT] und [INOUT] sowie die definierten Datentypen.
504	... missing before ...	Fügen Sie das von der Fehlermeldung angezeigte Zeichen hinzu.
	missing space	Fügen Sie ein Leerzeichen hinzu.
506	'...' undefined type	Verwenden Sie nur die definierten Datentypen.
507	'...' type not allowed	Beachten Sie die Syntax-Regeln aus Kapitel Funktionen einer ODK-Anwendung definieren (Seite 57)
508	'...' type redefinition	Der Funktions- oder Parametername ist bereits vergeben. Wählen Sie einen anderen Namen.
509	'...' variable redefinition	Der Variablenname ist bereits vergeben. Wählen Sie einen anderen Namen.
510	Structure '...' must not be empty	Befüllen Sie die Struktur mit einem Datentyp.
511	'...' no valid name	Beachten Sie die Syntax-Regeln aus Kapitel Funktionen einer ODK-Anwendung definieren (Seite 57).
512	unexpected variable order (must be [IN], [OUT], [INOUT] order)	Es gibt drei definierte InOut-Identifiers. Nutzen Sie diese in folgender Reihenfolge: [IN], [OUT], [INOUT]
513	size of ODK_S7STRING could not be bigger than 254	Ein String darf nur eine Länge von maximal 254 Zeichen haben.
514	size of ODK_S7WSTRING could not be bigger than 16382	Ein WString darf nur eine Länge von maximal 16382 Zeichen haben.
515	Prefix + Function name '....' exceeds 125 characters	Präfix und Funktionsname sind zusammen länger als 125 Zeichen.
516	variable name '…' exceeds 128 characters	Der Variablenname ist länger als 128 Zeichen.
517	'...' IN_BUFFER + INOUT_BUFFER could not be greater than 1 MB	Die InOut-Identifier [IN] und [INOUT] dürfen in einer Funktion zusammen nicht größer als 1 MB sein.
518	'...' INOUT_BUFFER + OUT_BUFFER could not be greater than 1 MB	Die InOut-Identifier [OUT] und [INOUT] dürfen in einer Funktion zusammen nicht größer als 1 MB sein.
519	'...' needs '...k', but data size (SyncCallDataSize) is limited to '...k'	Die Datenmenge ist zu groß.
520	'...' has an array size of '...', but max. array size is limited to '...'	Die maximale Array-Größe ist überschritten.
521	no other variable in the same direction for ODK_CLASSIC_DB type	Sobald der Datentyp ODK_CLASSIC_DB verwendet wird, darf keine weitere Variable mit dem selben InOut-Identifier definiert werden.
522	no array allowed for ODK_CLASSIC_DB type	Für den Datentyp ODK_CLASSIC_DB darf kein Array definiert werden.
523	no [OUT] direction allowed for ODK_CLASSIC_DB type	Für den Datentyp ODK_CLASSIC_DB darf der InOut-Identifier [OUT] nicht definiert werden.
524	function declarations lead to identical hashes (change name of one parameter): '...', '...'	Verändern sie einen Parameternamen.

A.4 Helper-Funktionen

String-Helper-Funktionen für ODK-Anwendung für die Windows- und Echtzeit-Umgebung

Folgende Helper-Funktionen geben Zugriff auf S7Strings:

Helper-Funktion	Beschreibung
Convert_S7STRING_to_SZSTR	Konvertiert PLC-String-Typen in C/C++ String-Typen ("char" Array, Null-terminiert)
Convert_SZSTR_to_S7STRING	Konvertiert C/C++ String-Typen ("char" Array, Null-terminiert) in PLC-String-Typen.
Get_S7STRING_Length	Gibt die aktuelle Länge eines PLC-String-Typen zurück.
Get_S7STRING_MaxLength	Gibt die maximale Länge eines PLC-String-Typen zurück.

String-Helper-Funktionen für ODK-Anwendung für die Windows-Umgebung

Folgende Helper-Funktionen geben Zugriff auf S7WStrings:

Helper-Funktion	Beschreibung
Convert_S7WSTRING_to_SZWSTR	Konvertiert PLC-WString-Typen in C/C++ WString-Typen ("wchar_t" Array, Null-terminiert)
Convert_SZWSTR_to_S7WSTRING	Konvertiert C/C++ WString-Typen ("wchar_t" Array, Null-terminiert) in PLC-WString-Typen.
Get_S7WSTRING_Length	Gibt die aktuelle Länge eines PLC-WString-Typen zurück.
Get_S7WSTRING_MaxLength	Gibt die maximale Länge eines PLC-WString-Typen zurück.

Klasse "CODK_CpuReadData" (Windows- und Echtzeit-Umgebung)

Die Klasse "CODK_CpuReadData" erlaubt lesenden Zugriff auf Classic-DBs:

Wert	Beschreibung
CODK_CpuReadData	Klassen-Konstruktor: Initialisiert den Input-Daten-Bereich und die Datengröße.
ReadS7BYTE	Liest ein "byte" (1 Byte) aus dem Datenbereich.
ReadS7WORD	Liest ein "word" (2 Bytes) aus dem Datenbereich.
ReadS7DWORD	Liest ein "double word" (4 Bytes) aus dem Datenbereich.
ReadS7LWORD	Liest ein "long word" (8 Bytes) aus dem Datenbereich.
ReadS7S5TIME	Liest einen 16-Bit (2 Bytes) Zeitwert aus dem Datenbereich.
ReadS7DATE	Liest einen Datumswert (2 Bytes) aus dem Datenbereich.
ReadS7TIME_OF_DAY	Liest die Tageszeit (4 Bytes) aus dem Datenbereich.
ReadS7SINT	Liest ein "short integer" (1 Byte) aus dem Datenbereich.
ReadS7INT	Liest ein "integer" (2 Bytes) aus dem Datenbereich.
ReadS7DINT	Liest ein "double integer" (4 Bytes) aus dem Datenbereich.
ReadS7USINT	Liest ein "unsigned short integer" (1 Byte) aus dem Datenbereich.
ReadS7UINT	Liest ein "unsigned integer" (2 Bytes) aus dem Datenbereich.

Wert	Beschreibung
ReadS7UDINT	Liest ein "unsigned double integer" (4 Bytes) aus dem Datenbereich.
ReadS7REAL	Liest ein "real number" (4 Bytes) aus dem Datenbereich.
ReadS7LREAL	Liest ein "long real number" (8 Bytes) aus dem Datenbereich.
ReadS7LINT	Liest ein "long integer" (8 Bytes) aus dem Datenbereich.
ReadS7ULINT	Liest ein "unsigned long integer" (8 Bytes) aus dem Datenbereich.
ReadS7TIME	Liest einen Zeitwert (4 Bytes) aus dem Datenbereich.
ReadS7CHAR	Liest ein "char" (1 Byte) aus dem Datenbereich.
ReadS7BOOL	Liest ein "bool" (1 Bit) aus dem Datenbereich.
ReadS7STRING_LEN	Liest die Information der Sting-Länge für einen S7-String im Datenbereich.
ReadS7STRING	List einen S7-String aus dem Datenbereich und gibt ihn als C++-Zeichen-String zurück.
ReadS7DATE_AND_TIME	Liest einen allgemeinen Datums- und Zeitbereich.

Klasse "CODK_CpuReadWriteData" (Windows- und Echtzeit-Umgebung)

Die Klasse "CODK_CpuReadWriteData" erlaubt zusätzlich zu allen lesenden Zugriffen von "CODK_CpuReadData" auf Classic-DBs auch die folgenden schreibenden Zugriffe:

Wert	Beschreibung
CODK_CpuReadWriteData	Klassen-Konstruktor: Initialisiert den Output-Daten-Bereich und die Datengröße.
WriteS7BYTE	Schreibt ein "byte" (1 Byte) in den Datenbereich.
WriteS7WORD	Schreibt ein "word" (2 Bytes) in den Datenbereich.
WriteS7DWORD	Schreibt ein "double word" (4 Bytes) in den Datenbereich.
WriteS7LWORD	Schreibt ein "long word" (8 Bytes) in den Datenbereich.
WriteS7SINT	Schreibt ein "short integer" (1 Byte) in den Datenbereich.
WriteS7INT	Schreibt ein "integer" (2 Bytes) in den Datenbereich.
WriteS7DINT	Schreibt ein "double integer" (4 Bytes) in den Datenbereich.
WriteS7USINT	Schreibt ein "unsigned short integer" (1 Byte) in den Datenbereich.
WriteS7UINT	Schreibt ein "unsigned integer" (2 Bytes) in den Datenbereich.
WriteS7UDINT	Schreibt ein "unsigned double integer" (4 Bytes) in den Datenbereich.
WriteS7S5TIME	Schreibt einen 16-Bit (2 Bytes) Zeitwert in den Datenbereich.
WriteS7TIME	Schreibt einen Zeitwert (4 Bytes) in den Datenbereich.
WriteS7DATE	Schreibt einen Datumswert (2 Bytes) in den Datenbereich.
WriteS7TIME_OF_DAY	Schreibt die Tageszeit (4 Bytes) in den Datenbereich.
WriteS7CHAR	Schreibt ein "char" (1 Byte) in denDatenbereich.
WriteS7REAL	Schreibt ein "real number" (4 Bytes) in den Datenbereich.

Wert	Beschreibung
WriteS7LREAL	Schreibt ein "long real number" (8 Bytes) in den Datenbereich.
WriteS7LINT	Schreibt ein "long integer" (8 Bytes) in den Datenbereich.
WriteS7ULINT	Schreibt ein "unsigned long integer" (2 Bytes) in den Datenbereich.
WriteS7BOOL	Schreibt ein "bool" (1 Bit) in den Datenbereich.
WriteS7STRING	Schreibt einen S7-String in den Datenbereich.
WriteS7DATE_AND_TIME	Schreibt Datums- und Zeitdaten in den Datums- und Zeitbereich.

A.5 Anweisung "Load"

Die Anweisung "*<STEP7Prefix>*_Load" weist je nach Entwicklungsumgebung unterschiedliche Parameter auf:

- Entwicklung einer ODK-Anwendung für die Windows-Umgebung (Seite 39)
- Entwicklung einer ODK-Anwendung für die Echtzeit-Umgebung (Seite 77)

A.6 Anweisung "Unload"

Die Anweisung "*<STEP7Prefix>*_Unload" weist je nach Entwicklungsumgebung unterschiedliche Parameter auf:

- Entwicklung einer ODK-Anwendung für die Windows-Umgebung (Seite 45)
- Entwicklung einer ODK-Anwendung für die Echtzeit-Umgebung (Seite 81)

A.7 Anweisung "GetTrace"

Der Funktionsbaustein (Seite 83) "GetTrace" ist in Standard-CPP-Datei "<Projekt>.cpp" enthalten.

GetTrace	
TraceCount	STATUS

Die folgende Tabelle zeigt die Parameter des Funktionsbausteins "GetTrace":

Sektion	Deklaration	Datentyp	Beschreibung
Output	STATUS	INT	Anzahl der tatsächlich gelesenen Trace-Einträge
Input	TraceCount	INT	Anzahl der Trace-Einträge, die gelesen werden sollen
Output	TraceBuffer	Array [0..255] of String[125]	Trace String Array für den Nutzer jeder Trace String besteht aus: • Datum • Uhrzeit • OB-Nummer • Dateiname • Zeilennummer • Trace-Text (der vom Anwender implementierte Trace)

Index

P

S

T

U

W

Z