4CONTROL Embedded

Software Development Kit

4CONTROL Engineering

Configuration Management

Version 2.1

SDK_E_50_Configuration_Management.doc

SOFTING AG

Richard-Reitzner-Allee 6
D-85540 Haar, Germany
Tel.: +49 (89) 4 56 56 – 0
Fax: +49 (89) 4 56 56 – 399

Printed in Germany 2004

4C-21-SDK-E/e

Document History

Date	Version	Author	Changes
01.11.02	1.0	JD	Initial Version
02.10.2004	2.0	LN	Adaptation to 4CONTROL V2.1

Table of Contents

1 Introduction

The 4CONTROL Embedded SDK and targets based on that SDK use a special mechanism to generate projects. This mechanism, the structure of the SDK and the requirements to develop AddOns are described in this document.

First the system requirements will be listed in details. In addition to the installed tools some environment settings have to be done to work with the 4CONTROL generation mechanism. How to set up an environment to develop a 4CONTROL AddOn is described in the first chapter.

The next chapter describes where different things can be found. There are four different parts: (1) the directory structure after a 4CONTROL installation, (2) how is the SDK structured, (3) how should the directories of an AddOn be organized and (4) where are the tools and scripts needed for a generation located.

The 4CONTROL generation mechanism is based on some Perl scripts that can produce make files from a standard project file and an additional description file. The full mechanism, the used tools and the steps a user has to perform to work with this mechanism are described in details.

In addition to this a system generation mechanism is provided that can build a complete AddOn with setup in a batch build, every night on a dedicated system generation computer for example. A system generated version gets a build number, will be labeled in Source Safe and the final version copied to a server. This batch mechanism is described in the last chapter.

2 System Requirements

To generate a 4CONTROL AddOn a running Windows NT – SP6a, Windows 2000 – SP4 or Windows XP – SP1 system is needed. The tools listed in chapter 2.1 need to be installed and the environment variables listed in chapter 2.2 must be set correctly.

To build the target firmware a special environment is necessary. This depends on the target system. The system can be build using a cross-compiler running under Windows or a compiler running on the native system.

2.1 Tools

The following tools need to be installed to generate a target AddOn for 4CONTROL:

- ✓ Microsoft Visual C++, Version 6.0, Service Pack 5
- ✓ Microsoft Visual SourceSafe, Version 6.0, Service Pack 6
- ✓ InstallShield Windows Installer Edition Version 2.03 or a newer version of InstallShield that can generate windows installer packages.
- ✓ A Perl interpreter:

Softing uses a free available Perl interpreter:

Perl 5: PERL5 (5.6.1)

© By ActiveWare Internet Corp., http://www.ActiveWare.com

Perl for Win32

To generate the target firmware a special compiler is needed that can generate code for the target. This can be a cross-compiler running under Windows or a compiler running on the native system itself.

2.2 Settings

Some environment variables need to be set to use the 4CONTROL generation mechanism:

2.2.1 Microsoft Visual C++

Visual C++ needs some environment variables set to run correctly from command line. These variables can be set automatically by the setup of Visual Studio.

The setup also generates a batch file that can set the variables. This batch file can be found in `<VisualStudio Install Directory>\ VC98\Bin\ VCVARS32.BAT`.

The following table shows all variables for Visual Studio: Replace `C:\Programme\MicrosoftVisualStudio` with the path where Visual Studio is installed on the computer.

Environment Variable	Value
`MSDevDir`	`C:\Program Files\MicrosoftVisualStudio\Common\msdev98`
`MSVCDir`	`C:\Program Files\MicrosoftVisualStudio\VC98`

INCLUDE	%MSVCDir%\ATL\INCLUDE; %MSVCDir%\INCLUDE; %MSVCDir%\MFC\INCLUDE; %INCLUDE%
LIB	%MSVCDir%\LIB; %MSVCDir%\MFC\LIB; %LIB%
PATH	%MSDevDir%\BIN; %MSVCDir%\BIN; C:\Program Files\MicrosoftVisualStudio\Common TOOLS\%VcOsDir%; C:\Program Files\MicrosoftVisualStudio\Common\TOOLS

Table 1: Visual Studio Environment Variables

2.2.2 Perl

To be able to use Perl the path to the Perl binary directory should be added to the `PATH` environment variable. The Active Perl setup program can do this automatically. If it is not done the following table shows the necessary entry, replace `C:\Perl` with the directory where Perl is installed on the computer:

Environment Variable	Value
PATH	C:\Perl\bin

Table 2: Perl Environment Variables

2.2.3 InstallShield

To use InstallShield from the command line, two more paths have to be added to the `PATH` environment variable. The following table shows the two entries, replace `C:\Programme\InstallShield\` with the path where InstallShield is installed on the computer:

Environment Variable	Value
PATH	C:\Program Files\Common Files\InstallShield\; C:\Program Files\InstallShield\Professional - Windows Installer Edition\System

Table 3: InstallShield environment variables

2.2.4 4CONTROL

To generate a 4CONTROL project two environment variables have to be set. These variables are used to point to the source tree of 4CONTROL. It is recommended to use the directory structure described in the next chapter for the 4CONTROL installation, too.

4CONTROL source base directory will be the same as the 4CONTROL installation directory and will be referenced as `<4CINSTDIR>` in the following.

Environment Var.	Value	Example
`PROJECT_ROOT`	Installation and source base directory.	`<4CINSTDIR>; C:\4CONTROLV2`
`TARGET_BASE`	The directory where the 4CONTROL SDK is located, this can be beneath the source directory or somewhere else on a server for example.	`<4CINSTDIR>\TargetBase; C:\4CONTROLV2\TargetBase`

Table 4: 4CONTROL Environment Variables

To simplify the generation from the command line *.pl files should be associated to the Perl binary (this has probably be done by the Perl installation) and the following directory should be added to the `PATH` environment variable: `<4CINSTDIR>\gentools`. With these two changes the Perl scripts used during the generation can be started by just typing their name on the command line.

To simplify the generation of 4CONTROL projects from Visual Studio it is recommended to add a custom tool entry to the Tools main menu of Visual Studio, to do this use Tools\Customize menu and select the Tools tab, add a new tool with the following settings:

Setting	Value
Tool Name	`4C Make.pl`
Command	`perl.exe`
Arguments	`<4CINSTDIR>\gentools\make.pl $(FileDir)\$(WkspName);` `C:\4CONTROLV2\gentools\make.pl $(FileDir)\$(WkspName)`
Initial Directory	`$(WkspDir)`
Use Output Window	`yes`
Prompt for arguments	`no`

Table 5: Tools Settings in Visual Studio

In general a new Target AddOn uses the 4C license mechanism. That means you need a valid license for your new Target AddOn to run the entire target generation (otherwise the generation of the 4C library will not run). Please contact Softing to get this initial license!

3 Directory Structure

The 4CONTROL generation and installation consists of different parts. To simplify the process and to be able to see the changes of a newly generated project in the installed 4CONTROL system it is recommended to use the same base directory for the installation of 4CONTROL and the sources. This directory is called `4CINSTDIR` in the following, for example `4CINSTDIR = "C:\4CONTROLV2"`.

The environment variable `PROJECT_ROOT` should point to this directory, compare 2.2.4. This directory has to be entered in the first setup of 4CONTROL. All further setups will use the same directory. The directory is stored in the registry under `HKEY_LOCAL_MACHINE\SOFTWARE\Softing\4CONTROLV2\2.0\Path`.

3.1 Installation

Before starting to develop an AddOn for 4CONTROL the setups of the basic system provided from Softing should be installed. This is the Kernel setup and the setup of the Softing License Manager.

Softing License Manager will always install to the *CommonFiles* folder, for example `C:\Program Files\Common Files\Softing`.

Beneath this folder there are located also some common parts of 4CONTROL and other Softing products.

The Kernel Setup installs the 4CONTROL Workbench and all common parts of the 4CONTROL system.

Except the common files stored in the folder mentioned above and some shared DLLs stored in the Windows system folder, all other files are installed beneath `4CINSTDIR`.

The following directory structure is used:

Directory	Explanation
`Engineering`	The base directory for all files belonging to the workbench part of 4CONTROL.
`Engineering\bin`	All binary and needed additional files of the 4CONTROL kernel system.
`Engineering\bin\<target-dir>`	Each new target should create a subdirectory named like the target file extension here. All parts of AddOn should be installed here.
`Engineering\bin\images`	All images of the kernel system of 4CONTROL.
`Engineering\bin\images\<target-dir>`	AddOns should install their images in subdirectories of the image folder.
`Engineering\Help`	All parts of the 4CONTROL online-help.
`Engineering\Lib`	This is the standard library path used in the 4CONTROL workbench to select libraries. All standard libraries are copied here during the 4CONTROL Kernel installation. It is recommended to install the target specific libraries to this folder, also.
`Engineering\Templates`	Some templates for the wizards.
`Projects`	4CONTROL user projects can be stored here.

Table 6: 4CONTROL Installation Directory Structure

3.2 Make Tools

The SDK contains a folder with scripts and header used for the generation mechanism. This folder is called `GenTools` and should be copied to the `4CINSTDIR` folder. The scripts are described in details in 4. This folder should be added to the `PATH` environment variable to simplify the generation of a 4CONTROL project from the command line.

3.3 SDK

The SDK contains 5 subdirectories.

The subdirectories `4Control` and `4Control_R` contain the header, libraries and source files necessary to compile a target AddOn for 4CONTROL Embedded. `4Control` contains the libraries that arose from a debug generation of the kernel and target base system and `4Control_R` contains the libraries that arose from a release generation. Depending on what version (debug/release) of the target AddOn should be generated one of the two branches has to be chosen. The structure beneath the two branches is exactly the same.

The third directory: `Setup` contains the complete setups of the Kernel (4CONTROL Workbench and basis system) and the Softing License Manager. In addition there are some merge modules that can be included into the Target AddOn setups to use some common files of 4CONTROL.

The directory structure beneath the `4Control(_R)` directories is equivalent to the necessary parts of the structure of a generation of the 4CONTROL kernel system.

The 4th directory – `Firmware` – holds the source codes of the 4CONTROL Embedded Run Time System. Depending on the actual project, these sources can be either an empty template or a real working firmware.

The 5th directory – `Documentation` – holds this documentation as PDF documents.

The following chapters give an overview over the subdirectories of the source code folders `4Control` and `4Control_R`:

3.3.1 Subdirectory: `Engineering`

This folder corresponds to the Engineering folder after an installation of the 4CONTROL Kernel setup, see 3.1.

On the computer where the generation of the target AddOn should run, this folder should arise from an installation of the setup, so that all necessary COM servers are registered correctly. If there are mismatches between debug and release versions during testing a debug generation of the target AddOn, it can be necessary to copy the debug binaries from the SDK to the `Engineering\bin` folder.

3.3.2 Subdirectory: `GenTools`

This folder contains the system generation scripts and is described in chapter 3.2.

3.3.3 Subdirectory: `Pub`

Here some shared DLLs are located. These DLLs resist in the Windows system directory after an installation. If there are mismatches between debug and release versions after a generation of a debug version of a target AddOn these DLLs have to be exchanged by the debug versions from this SDK directory.

3.3.4 Subdirectory: `Target`

This directory contains the source codes for the corresponding Target AddOn.

3.3.5 Subdirectory: `TargetBase`

Here are located all includes, libraries and sources that are needed to build a target AddOn.

Please note that all C++ source files within this directory are to be indented only for debugging purposes. It is not recommended to change these files.

- ✓ `Kernel`: All parts that are interfaces (and additional needed header and libraries files) to the kernel system of 4CONTROL, that means the workbench and system communication part, are located in the Kernel folder.
- ✓ `AddonHandler`: Here are the source files and include files for the AddOn Handler library. These are the header of the base classes that can be used to simplify the development of an AddOn.
- ✓ `ProjectWizardBase`: Here are the source files and include files for the Project Wizard library. These are the header of the base classes that can be used to simplify the development of an AddOn.
- ✓ `CCG`: Here are the include files for the 4CONTROL Embedded compilers.
- ✓ `Online`: The online communication between the workbench and the Run Time System.
- ✓ `Control`: The (empty) template files for the 4CONTROL Embedded Run Time System.
- ✓ `Libs`: The standard 4CONTROL IEC libraries.
- ✓ `inc`: This folder contains header files that are shared between the online part of the system and the run time part.
- ✓ `lib`: All static link libraries that contain the base classes and helper functions from the target base that simplify the development of an AddOn.
- ✓ `bin`: Some ready to use executables and DLLs.

3.4 Target Sources

The target sources should be located in the directory `4CINSTDIR\Target\<target name>`.

The sources are structured into the following subprojects, located in the corresponding subfolders:

- ✓ `AddOnHandler`: This folder contains the project for the AddOn Handler and ProjectWizard DLL.
- ✓ `KAD`: This folder contains the adaptation KAD's for the engineering, the compiler and the AddOn Handler. All images needed by the AddOn are stored in a subfolder.
- ✓ `Libs`: This folder contains the IEC libraries for the target.
- ✓ `inc`: Contains the header files shared between the online communication and the target Run Time System.
- ✓ `Online`: This folder contains the projects for the communication DLL and the OPC server for the target.
- ✓ `res`: In this folder some header and resource files are stored to include uniform version info to all DLLs and executables. Here the product description, the version number and the build number can be found. The file *BuildNr.h* will be modified automatically by a system generation to increase the build number for each system generation, see **Fehler! Verweisquelle konnte nicht gefunden werden.**.
- ✓ `Setup`: This folder contains the sources for the default setup of the target. The InstallShield project has to be adapted to the need of the entire setup.

4 Build Mechanism

The 4CONTROL system generation uses a special build mechanism. This mechanism and the used script files are described in the following.

4.1 Motivation

If there are circular dependencies between two or more DLLs the build mechanism of Visual Studio has some problems. Suggest the following situation: DLL A needs a function of DLL B and another function of DLL B needs a function of DLL A:

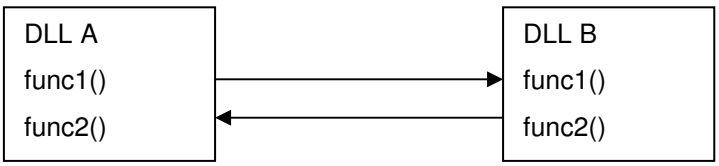

Figure 1: Cyclic dependencies

When DLL A will be built the following steps are performed:

All sources are compiled and the object files are generated.

A link library will be generated `A.lib`.

DLL A will be linked, for this step `B.lib` is needed.

This step can be performed only after B has been built. And here the same problem arises vice versa since Visual Studio performs these steps one after another for DLL B. This means that Visual Studio cannot resolve the circular dependency of DLLs in one run.

These dependencies can also be found between other generation steps.

Therefore a build mechanism is introduced that performs the single steps of the generation recursively for all projects before advancing to the next build step. But during development the single projects can be processed using the Visual Studio.

4.2 General Overview

Each project has its own project file and workspace. The workspace contains only one single project to generate a DLL or executable, for example X.DLL:

```
Workspace X
      → Project for X.DLL
            → File 1
            → File 2
            ...
            → File n
```

Figure 2: 4CONTROL Workspace

The project has no dependencies to other projects, so the `dsw` file of Visual Studio that would contain dependencies of this kind, does not need to be stored in Visual Source Safe. All information is contained in the project file (`dsp`) and in an additional `inf` file.

The files `X.dsp and X.inf` are stored in the Source Safe in addition to all source files.

`X.dsp`: This file contains all source file names, the name of the executable for debugging and a link to a custom build step pointing to an external make file. This link is not contained in the `dsp` file generated by Visual Studio. The `GenTool` script *convert.pl* can adapt a `dsp` file generated by a Visual Studio Wizard, see 4.3.1.

`X.inf`: This file contains all information needed to generate the project, name of the generated DLL or executable, paths to include and library files, links to needed libraries and other settings. The syntax is described in detail in 4.5.

As the first step of the 4CONTROL build mechanism a make file `X.mak` will be generated from these two files for each project. This file will not be checked into the Source Safe, it is a generated file. It contains path dependencies corresponding to the local computer installation.

After generating this make file, a build from Visual Studio uses this make file to generate the project, all steps of the development, checking in and out files from Source Safe, build and debug the component can be done in the development environment. Without the external make file 4CONTROL projects can't be compiled from the Visual Studio development environment. Therefore 2.2.4 describes how to include a tools menu entry in Visual Studio to generate the external make file.

4.3 Build Tools

To use the 4CONTROL build mechanism some scripts are provided. These scripts are based on Perl and are located in `4CINSTDIR\GenTools`. Two scripts are needed to generate a 4CONTROL project. These are described in the following two sections.

4.3.1 Build Tool: *convert.pl*

Convert.pl converts a project file (`dsp`) generated by a Visual Studio Wizard to a project file using the 4CONTROL build mechanism with the external make file. It replaces the settings generated by Visual Studio with a custom build step linking the external make file. The script takes the name of the project as command line parameter.

Example command line:

```
perl.exe  $(PROJECT_ROOT)\gentools\convert.pl  X.dsp
```

The project should not be open in any development environment during the conversion. It has to be closed before converting and reopened after the conversion.

4.3.2 Build Tool: *make.pl*

`Make.pl` generates the make file from a project file and a corresponding `inf` file. The name of the `dsp` and the `inf` file must be the same. *Make.pl* detects all dependencies and enters these into the make file. The used files are taken from the `dsp` file. Files with the following extension are used:

```
.obj .def .cpp .c .y .l .rc .mc .odl .idl .java .lib
```

The `inf` file has to be created by the user, see 4.5 for the detailed syntax.

Make.pl can be called from the tools menu of Visual Studio if the steps described in 2.2.4 have been performed or from command line.

Command line example:

```
perl.exe  $(PROJECT_ROOT)\gentools\make.pl   X
```

The generated make file will be used to build the component. This can be done by using build from the developing environment of Visual Studio or the command line.

Command line example:

```
nmake -f X.mak <make-target> <options>
```

Different standard make-targets can be given, so only these build steps will be executed. These are described in detail in the next chapter.

The following options can be set:

Option	Explanation
SUB=0	Don't process subprojects. As a standard the make mechanism processes all subprojects defined in the current project, too.
DEBUG=0	Generate a release version of the component.
DEBUG=1	Generate a debug version of the component. This is the standard setting, if the debug option is omitted.

Table 7: Make File Options

Each generated make file includes some standard definitions by including the file *global.inc* which can be found in `$(PROJECT_ROOT)\gentools`. This includes defines for standard directories, compiler and linker flags and settings and defines for all build tools. All these defines can be used in the `inf` files, the most import ones are described in 4.5.

4.4 The Build Steps

The build of a project can be split into several steps. Each step is represented by a make target in the MAK file. These make targets are presented in this chapter in details. Each step can be executed as a single step; the step will be executed for all subprojects linked in the project's INF file, too, if subproject processing is not switched of. The make target "all" will build the whole target and will process several targets one after another. Each step is done for the project and all subprojects before advancing to the next step. The make target all is the standard target and will be used if no target is given. Some targets are not part of the all sequence and have to be called manually.

The make targets in detail:

Make Target	Explanation
`all`	The following targets will be executed in the given order: `makefirst oleclasses javaclasses precompile compile prelink link makelast`
`makefirst`	First target, this can be used for user steps given in the `inf` file.
`oleclasses`	Call `midl` for `idl` files and make type libraries.
`javaclasses`	Call Java compiler for all java files.
`precompile`	Some copy actions can be inserted here. This step is also used for calling the tools: `lex` and `yacc` and `msjavah` for the corresponding files.
`compile`	Compile all source files.
`prelink`	Some steps that have to be done before linking and after compiling. Creating `libs` for example.
`link`	Link the executables.
`makelast`	The last build step. This step can be used to insert user commands in `inf` file, to copy some files for example.
`makemake`	This build step should be executed before any other to generate the make files in all subprojects. This step will call make.pl recursively for all subprojects.

make_lib	This is a step that can be executed manually to create IEC libraries. This step needs a completed generation or installation of 4CONTROL.
clean	Clean up all generated files of a project. This step has to be used to rebuild all.

Table 8: Make Targets

4.5 Inf-File Syntax

The project file (dsp) describes which source files belong to the project. All lines SOURCE=x.y from the project file will be used to create the make file. The script *make.pl* additionally uses the inf file to create the make file. Here is described what should be generated and where. Possible dependencies from link libraries and additional make steps can be given here.

The syntax of the inf file is line oriented. Each line follows the syntax:

```
OPTION=VALUE
```

There is no space allowed before and after the =. Spaces after = with option USER will be copied to the make file as given in the inf file. In addition to the option lines there can be empty lines and comment lines in the inf file. Comment lines start with#.

```
# This is a comment
```

The following options are allowed, standard options are marked with *:

Option	Values	Explanation
TARGETTYPE	EXE*|DLL| LIB	What kind of target should be generated, an executable, a dynamic link library or a static link library.
TARGETTYPE	GUI*|CON	Mark if the executable is a command line executable or an executable with a graphical user interface.
TARGET	x.exe; x.dll; x.lib	The name of the target with extension and fill path information.
UNICODE	1|ON|0|OFF*	Switch on or off Unicode defines and link with Unicode libraries.
IMPLIB	x.lib	Generate an import library for a DLL. Should be given with path information.
LIBRARY	x.lib	A library that should be used given with path information. The corresponding line will be included in the dependencies in the make file. The following libraries are preset: *kernel32.lib*, *user32.lib*, *gdi32.lib*, *advapi32.lib*
DEFLIB	x.lib	A library that should be used without path information. LIBPATH will be used to search for this library.
OBJECT	x.obj	An object file, given without path.
INCLUDE		Add a directory to the include path where header files are searched for.
LIBPATH		Add a directory to the search path for libraries.
DEFINE		Set a define.
COPY		Copy files. Files given here will be copied before the compiler is running (make target: precompile).
LATECOPY		Copy files. Files given here will be copied after the linker has been running(make target: makelast)
MASTER_RC	x.rc	If there are more than one rc files, the rc file that should be translated can be specified here. This option is optional.
PCHFILE	stdafx	Specify the precompiled header file without path and extension.

SUBPROJECT	x.mak	Specify a subproject. The make file of the subproject has to be specified with path information relative to the project.
STATIC_RUNTIME	1|ON|0|OFF*	Generate with option MT runtime instead of MD.
SUBPROJECT	x.mak	Specify a subproject. The make file of the subproject has to be specified with path information relative to the project.
USER		Lines after the option user are copied to the make file as given in the inf files. The user can add make commands like that.

Table 9: INF File Options

In the most options make variables can be used, the most important predefined ones are:

Variable	Explanation
PROJECT_ROOT	This is the environment variable PROJECT_ROOT, see 2.2.4.
TARGET_BASE	This is the environment variable TARGET_BASE, see 2.2.4.
EXE_DIR	This variable points to PROJECT_ROOT, for global executables of 4CONTROL for the setup for example.
ENGBIN_DIR	This variable points to the Engineering\bin directory.
4CL_DIR	This variable points to the standard library directory for 4CONTROL IEC-libraries: Engineering\lib
4CG	This variable can be used to call the 4CONTROL compiler.

Table 10: Predefine Make Variables

4.6 How To

In this part the typical steps of working with the 4CONTROL generation mechanism will be explained.

4.6.1 Setting up a project

A new project should be started as usual with the Microsoft Visual Studio wizards. After finishing with the wizards an inf file with the same name as the project should be created and added to the project. After filling this inf file with the necessary information, the project has to be closed in Visual Studio. Then the project file is converted on command line and one test generation can be done.

Example: TestProject

Files:

- ✓ *TestProject.dsp*
- ✓ *TestProject.inf*
- ✓ Source files

Dos Box commands to convert and test the build mechanism for the project:

```
perl.exe $(PROJECT_ROOT)\gentools\convert.pl TestProject
perl.exe $(PROJECT_ROOT)\gentools\make.pl TestProject
nmake -f TestProject.mak
```

If the PATH environment variable has been set as described in 2.2 and files with extension pl are associated with Perl, the command line will be simpler:

```
convert.pl TestProject
make.pl TestProject
nmake -f TestProject.mak
```

4.6.2 Working in Visual Studio

After converting the project file and inserting the link to the external make file, the project can be handled in the Visual Studio development environment. After changing the `inf` file or adding or removing files from the project make.pl has to be called again to generate a new make file. This will be done automatically if the `inf` file has been added to the project. The generation of a new make file can be forced with the Tools menu entry of Visual Studio created in 2.2.1.

Known problems: Some of the ATL-object wizards have problems with converted projects. These wizards should be used before converting the project.

4.6.3 Command line generation

With the following command lines a project `TargetX` can be build with all subprojects referenced in the `inf` file (debug version):

```
make.pl TargetX
nmake -f TargetX.mak makemake
nmake -f TargetX.mak
nmake -f TargetX.mak make_lib
```

The following command lines will produce a release version:

```
make.pl TargetX
nmake -f TargetX.mak makemake
nmake -f TargetX.mak DEBUG=0
nmake -f TargetX.mak make_lib
```

The build target `makemake` will recursively call make.pl in all subprojects. This must always be the first step if the make files do not already exist.

The target `make_lib` will make all IEC libraries contained in any project or subproject. This is not part of the standard target `all` because this step needs an already installed or generated 4CONTROL kernel and target system with all compiler components.

In the example targets and the targets generated from the templates the build of all target components prepares everything so that the 4CONTROL compiler can be used to create the IEC libraries after all other build steps.