Generic Protocol Stack Framework

Development Rules

Memo

Author: Condat AG
Alt Moabit 91d
10559 Berlin
Germany

Date: Nov 14, 2001

ID: 8434.008.01.001

Table of Contents

0 Document Control

Condat AG
Alt Moabit 90a
10559 Berlin
Germany

Telephone:	+49.30.3949-0
Fax:	+49.30.3949-1300
Internet:	www.condat.de
E-mail:	ccgpf@condat.de

0.1 Document History

ID	Author	Date	Status	Remarks
8434.008.01.001	FR	Nov. 14, 01	Being Processed	Initial

0.2 References

[C_8415.100]	8415.100.00.103; Feb 15, 2001; gpf_memo_crules.doc C Coding Standard; Condat

0.3 Abbreviations

0.4 Terms

---

1 Introduction

This document describes rules that apply to the software development inside the CT unit of Condat AG.

2 C Coding

The C coding rules are described in [C 8415.100].

3 Use of C runtime libraries under Windows

3.1 Single/Multithread

Always use the multithread library.

REASON: Most of our programs are multithreaded.

3.2 Release/Debug

For the release version use the release library, for the debug version use the debug library.

REASON: The memory management functions differ between debug and release version A) in definition (H-file) controlled by #ifdef _DEBUG and B) in implementation (the library). Therefore mixing e.g. a program build with _DEBUG and a non-debug library is not a good idea.

Use the release version per default for all tools and for all software that is to be shipped to customers. Use the debug version per default for all internal test/simulation versions, e.g. for the protocol stack.

3.3 Static/DLL

For all our own DLLs and for all programs using these DLLs use the C runtime DLL.
Link all programs that do not use one of our own DLLs statically.

REASON: It is generally safer to use the statically linked library because in this case the program does not depend from another file, which potentially could be missing or in a wrong version. That is why: link statically if ever possible.
But some of our own libraries are implemented as DLLs for some good reasons. And DLLs itself and also the DLL using applications have to use the C runtime DLL. This is because the C runtime system contains global variables (e.g. file descriptors). If application and DLLs would be statically linked there would exist several copies of these variables within one process. This may lead to very strange behavior.

REMARK: This rule has a strange but necessary consequence for static libraries (i.e. not just import libraries): Wether such a library should be linked with the static runtime or with the DLL runtime depends on the use if the library. If it is intended to use the library inside a statically linked program then use the static C runtime. If it is intended to use the library inside a DLL or a program linked against the C runtime DLL then use the C runtime DLL in the library too. ➔ Create two versions of the library, one for each link variant.

3.4 Compiler options

For the MS compiler of the Visual Studio the following switches should be used:

/MD for release versions of tools using DLLs.
/MDd for debug versions of tools using DLLs.
/MT for release versions of libraries to be statically linked (e.g. Frame and CCD for the nucwin protocol stack).
/MTd for debug versions of libraries to be statically linked.

All those switches place the respective default C runtime library name into the .obj file. They all define also _MT so that multithread versions of the routines are selected from the header files. The two switches /MD and /MDd define _DLL so that DLL specific versions of the routines are selected from the header files. The two switches /MDd and /MTd also define _DEBUG so that debug specific versions of the routines are selected from the header files.

It is also possible to select the compiler switch /Zl instead of any of the /M* switches. In that case no default C runtime library name is placed into the .obj file. Defining _MT, _DLL (if necessary), and _DEBUG (if desired) must then be done explicitly.