POSITIONING AND TRACK CONTROL SYSTEM
APCI-8001 AND APCI-8008

NCC.DLL

Rev. 8/112018

www.addi-data.com

1 Introduction

The NCC compiler for compiling SAP programs for the motion control boards APCI-8001 and APCI-8008 is available as a DLL for Windows 32-bit systems. This enables users, in their own programs, to compile source text files in rw_SymPas form or G-code files according to DIN 66025 into the intermediate code (CNC binary files) needed for the above systems.
To be able to execute these files, they must be transferred to the control unit concerned [txbf() / txbf2()] using the established methods and started [startcnct()].

2 Using NCC.DLL

The NCC.DLL library provides various functions to parameterise and execute a compiling process. The system file SYSTEM.DAT is required for the compiling process. "SYSTEM.DAT" from the working directory is used by default. The source text file to be compiled is also required. The CNC output file is normally stored in the same directory, unless a different path is indicated with the function SetCncFileName().
The compiling process must always be checked to see if it was successful. If an error is returned, the following error information can be read: File name, error line and error text. The file name is necessary since the error may also have occurred in an Include file.

3 Scope of delivery

The following software elements are supplied to use the NCC.DLL functions:

NCC.DLL	
NCC.H	
NCC.LIB	for various C-compilers
NCC.DLL.pdf	this document

4 Functions in NCC.DLL

4.1 CncCompile – compile G-code file

DESCRIPTION:	This function compiles a G-code file. The resultant binary file is given the extension .CNC, and is stored in the same directory as the source text file.
C:	int CncCompile (char *SrcFileName, int NumberAxis);
PARAMETERS:	SrcFileName is the name (including drive and path) of the source text file. In NumberAxis, the number of axes for which the program should be compiled is indicated.
RETURN VALUE:	0 in the event of success, error number if an error occured during the compilation.
NOTE:	The number of axes in fact present is returned in the TOSI data structure when initialized with InitMcuSystem3() with a booted system. In certain cases (if a file is supposed to be the same for different axis configurations) it may be necessary for a file to be compiled for more than the existing axes. In this case a higher number of axes can be transferred. This corresponds to the "Full system" option with the other NCC versions. But here additional syntax errors can arise relating to axis naming. The axis names are taken from the file SYSTEM.DAT and must be entered correctly for all axes used (mcfg.exe). G code files are always compiled for TASK 3. For this, see also the function DefIpolAxis (Chapter 4.3)
EXAMPLE:	CompileError = CncCompile ("Example.SRC", 3);

4.2 CncSyntaxCheck – Syntax check at G-code file

DESCRIPTION:	Command is the same as CncCompile (), however no binary file is generated.
C:	int CncSyntaxCheck (char *SrcFileName, int NumberAxis);
PARAMETERS:	SrcFileName is the name (including drive and path) of the source text file. In NumberAxis, the number of axes for which the program should be compiled is indicated.
RETURN VALUE:	0 in the event of success, error number if an error occured during the compilation.

4.3 DefIpolAxis – Define default interpolation axis

DESCRIPTION:	Definition of the axes which are in connection with interpolation; for G-Code files; can be called up before compiling a file for the first time
C:	int DefIpolAxis (unsigned int AxisBits);
PARAMETERS:	In Axis Bits, the axes with which positioning is to be carried out in an interpolated way are bit-coded.
NOTE:	With this call, the compiling of G-code files can be parameterised [use of the command CncCompile()]. If this call is not made, all available axes in Online Mode are used. Thus, this call is particularly required if not all axes in the system are interpolation axes. In this way, it is possible that, for example, in G01 commands not all axes have to be indicated every time. At runtime, this parameter can be switched using G60.
RETURN VALUE:	Always 0

Example:
A Cartesian coordinate system is given with X = 1st axis, Y = 2nd axis and Z = 3rd axis.
Then DefIpolAxis containing the value 7 has to be called up.

4.4 GetErrFileName – get file name in case of error

DESCRIPTION:	This function can be called up in the event of an error to specify the compiling error.
C:	void DLLFUNC GetErrFileName (char **File name*); void DLLFUNC GetErrFileNameX (char **File name, int ndx*);
PARAMETERS:	Pointer to a char array
RETURN VALUE:	For GetErrFileName(): none For GetErrFileNameX(): index of error to be queried
NOTE:	The function writes the file names including drive and path to the first error arising in *File name*. *File name* must point to an area of memory that is big enough to take the file name (max. 260 characters). For GetErrFileNameX(), the error index can be indicated in ndx (see chapter 4.6).
EXAMPLE:	GetErrFileName (*File name*);

4.5 GetErrLine – get error line

DESCRIPTION:	This function can be called up in the event of an error to identify the rows in which a compilation error has occurred.
C:	int GetErrLine (void); int GetErrLineX (int ndx);
PARAMETERS:	For GetErrLine(): none For GetErrLineX(): Index of error to be queried
RETURN VALUE:	Line number in which the first (GetErrLine) or the addressed (GetErrLine X) compiling error was detected.
NOTE:	The line number relates to the file whose name was returned with GetErrFileName() or GetErrFileNameX().
EXAMPLE:	Error line = GetErrLine ();

4.6 GetErrNum – query number of compilation errors

DESCRIPTION:	This function can be called up in the event of an error to query the number of compilation errors. In some cases, more than one error may have occurred before the compilation process was completed.
C:	int GetErrNum (void);
PARAMETERS:	none
RETURN VALUE:	Number of compilation errors in the last compiler run.
NOTE:	To enable error information to be read in indexed form, functions suffixed with an “X” are provided. In most cases, the compilation process is completed after the first error has occurred.
EXAMPLE:	NumErrors = GetErrLine ();

4.7 GetErrText – get error text

DESCRIPTION:	This function can be called up in the event of an error, to specify the compiling error.
C:	void GetErrText (char * *Error text*); void GetErrTextX (char * *Error text, int ndx*);
PARAMETERS:	Pointer to a char array
RETURN VALUE:	For GetErrText(): none For GetErrTextX(): Index of error to be queried
NOTE:	The function writes a comment on the first error occurring in *Error text*. *Error text* must point to an area of memory that is big enough to take the error message (max. 256 characters).
EXAMPLE:	GetErrText (*Error text*);

4.8 GetSymAxisName – Determine symbolic axis name

DESCRIPTION:	With this function, the symbolc axis name of an axis can be read.
C:	int GetSymAxisName (int an, char *SymAxisName);
PARAMETER:	*an* is the index of the axis to be read. In *SymAxisName* the symbolic name, which has been read from the system file SYSTEM.DAT, is returned as string terminating on 0.
RETURN VALUE:	The indicated axis-index at success, -1 at failure
NOTE:	The storage space in *SymAxisName* must be large enough to store the axes name including the concluding zero sign. The maximum size is 16 bytes.
EXAMPLE:	char AxisName[16]; rvalue = GetSymAxisName (0, AxisName);

4.9 SapCompile – compile rw_SymPas file

DESCRIPTION:	This function compiles a rw_SymPas file. The resultant binary file is given the extension .CNC, and is stored in the same directory as the source text file.
C:	int CncCompile (char *SrcFileName, int NumberAxis);
PARAMETERS:	SrcFileName is the name (including drive and path) of the source text file. In NumberAxis, the number of axes for which the program should be compiled is indicated.
RETURN VALUE:	0 in the event of success, error number if an error occured during the compilation.
NOTE:	The number of axes in fact present is returned in the TOSI data structure when initialized with InitMcuSystem3() with a booted system. In certain cases (if a file is supposed to be the same for different axis configurations) it may be necessary for a file to be compiled for more than the existing axes. In this case a higher number of axes can be transferred. This corresponds to the "Full system" option with the other NCC versions. But here additional syntax errors can arise relating to axis naming. The axis names are taken from the file SYSTEM.DAT, and must be entered correctly for all axes used (mcfg.exe).
EXAMPLE:	CompileError = SapCompile ("Example.SRC", 3);

4.10 SapSyntaxCheck – Syntax check at rw_SymPas file

DESCRIPTION:	Same command as SapCompile(), however without generating a binary file.
C:	int SapSyntaxCheck (char *SrcFileName, int NumberAxis);
PARAMETERS:	SrcFileName is the name (including drive and path) of the source text file. In NumberAxis, the number of axes for which the program should be compiled is indicated.
RETURN VALUE:	0 in the event of success, error number if an error occured during the compilation.

4.11 SetCaseInsensitive - Differentiation capitalization/lower case printing on/off

DESCRIPTION:	For the compiling of G-code files, the differentiation between capitalisation/lower case printing can be disabled with this function.
C:	void DLLFUNC SetCaseInsensitive (int CaseInsensitive);
PARAMETER:	0 or 1 (default is 0)
RETURN VALUE:	None

4.12 SetCncFileName

DESCRIPTION:	From version V2.5.3.58, a file name for the CNC output file including path and drive information can be transferred with this function.
C:	void SetCncFileName (char *str);
PARAMETERS:	File name with optional drive and path information
RETURN VALUE:	0 for success, 1 for failure
NOTE:	If this function is called up with a void string, a previously assigned file name can be deactivated again. In this case, the file name and the output path are determined by the name of the source text file.
EXAMPLE:	SetCncFileName (“C:\Cnc\Userfile.cnc”);

4.13 SetPostIncFileName

DESCRIPTION:	From version V2.5.3.57, a file name can be transferred with this function. The corresponding file is then included as an Include file at the end of the source text file without the instruction $I. This Include file may contain declarations from subroutines that can be used in the program, but which are not visible to the machine user. This functionality is only possible with programs according to DIN 66025 (G-code programs). If there is no path specified, the path of the source text file to be compiled is used.
C:	void SetPostIncFileName (char *str);

PARAMETERS:	File name with optional drive and path information
RETURN VALUE:	0 for success, 1 for failure
NOTE:	If this function is called up with a void string, a previously assigned file can be deactivated again.
EXAMPLE:	SetPostIncFileName (“C:\Temp\Subroutines.inc”);

4.14 SetPreIncFileName

DESCRIPTION:	From version V2.5.3.57, a file name can be transferred with this function. The corresponding file is then included as an Include file at the beginning of the source text file without the instruction $I. This Include file may contain variable and procedure declarations that can be used in the program, but which are not visible to the machine user. This functionality is only possible with programs according to DIN 66025 (G-code programs). If there is no path specified, the path of the source text file to be compiled is used.
C:	void SetPreIncFileName (char *str);
PARAMETERS:	File name with optional drive and path information
RETURN VALUE:	0 for success, 1 for failure
NOTE:	If this function is called up with a void string, a previously assigned file can be deactivated again.
EXAMPLE:	SetPreIncFileName (“C:\Temp\Declaration.inc”);

4.15 SetSystemDatName – set system file

DESCRIPTION:	This function can specify the system file (drive, path, file name) with which the compiling process is carried out.
C:	void SetSystemDatName (char *str);
PARAMETERS:	File name with optional drive and path information
RETURN VALUE:	None
NOTE:	If this function is not called, the file "SYSTEM.DAT" is selected in the working directory.
EXAMPLE:	SetSystemDatName (“C:\Temp\System.dat”);

4.16 SetTaskNum – set task number

DESCRIPTION:	This function can specify the task number for which the source text file is compiled. This is only advisable when compiling rw_SymPas files, since G code files are always compiled for task 3.
C:	void SetTaskNum (int TaskNr);
PARAMETERS:	Task number (0..3) for which the NC file is to be produced. After loading, a file generated in this way can only be used in the corresponding task.
RETURN VALUE:	none
NOTE:	If the task is selected in the source text file with {$TASK ?} this instruction is void. The default value is 0.
EXAMPLE:	

4.17 SetTrac – set default track acceleration

DESCRIPTION:	This function sets the default track acceleration.
C:	void SetTRAC (double track);
PARAMETERS:	Acceleration value in the interpolation units currently set
RETURN VALUE:	None
NOTE:	See also SetTrvl and SetUnits

4.18 SetTrvl – set default track speed

DESCRIPTION:	This function sets the default track speed.
C:	void SetTRVL (double trvl);
PARAMETERS:	Speed value in the interpolation units currently set
RETURN VALUE:	None
NOTE:	See also SetTrac and SetUnits

4.19 SetUnits – set default travel and time unit

DESCRIPTION:	This function defines the default interpolation units. Without this call, the position unit is set to mm and the time unit to seconds.
C:	void SetUnits (int pu, int tu);
PARAMETERS:	Position unit pu and time unit tu
RETURN VALUE:	None
NOTE:	See also PHB manual, chapter "ctru, change trajectory units“
EXAMPLE:	SetUnits (0, 1); // select units mm and min

4.20 SetUserSpecCode – set user-specific code

DESCRIPTION:	With this function, a user-specific code can be programmed for compilation. With this code, user-specific functions can be activated.
C:	void DLLFUNC SetUserSpecCode (unsigned UserSpecCode);
PARAMETERS:	project-based integer variable
RETURN VALUE:	none

4.21 UseLineNumbers – line numbering on / off

DESCRIPTION:	This function can deactivate the need for line numbers (Nxxx) for compiling G code files.
C:	void DLLFUNC UseLineNumbers (int UseLineNum);
PARAMETERS:	0 or 1
RETURN VALUE:	None