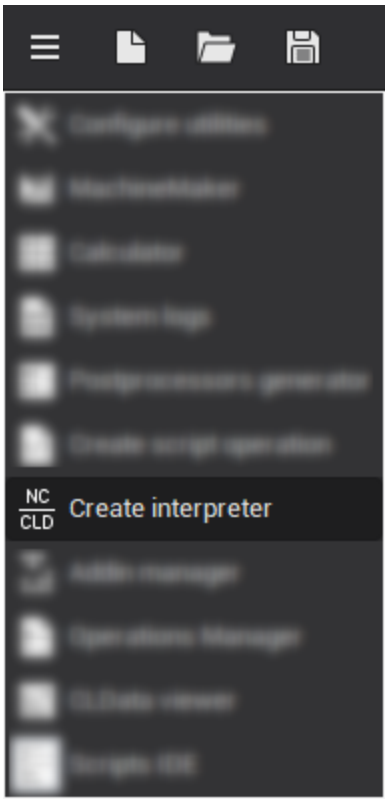
Creating your own interpreter

To create your own interpreter, complete the following steps:

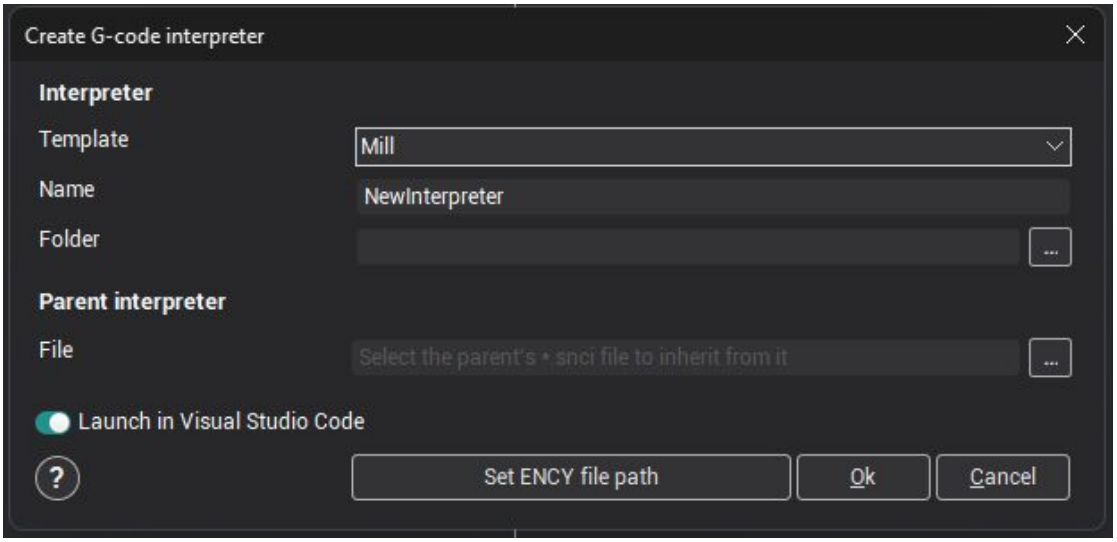
- Read the articles in this section that describe how the CAM system interacts with the interpreter when interpreting the NC program text.
- Create an interpreter settings file and develop a library for it as described on the following pages: Interpreter settings file, Interpreter library.
- If you want to develop with an AI assistant, start with the AI-assisted interpreter development section.

Developing a custom interpreter in (.NET) C#

To create your own G-code interpreters based on .NET (C#), it is recommended to use a special tool. It is the Create interpreter item in the Utilities list.



When you select the Create interpreter item, the corresponding window appears:



In this window you only need to fill in a few parameters, from which a template for your interpreter will be prepared:

- Template — the interpreter template;
- Name — the name of the interpreter being developed;

> WARNING
>
> *A directory with the same name is created from this name; it will contain all the source files of the new interpreter.*

- Folder — the directory in which the folder of your interpreter will be placed (you can enter the value manually or use the directory selection dialog by clicking the corresponding button ...).
- File — the full path to the settings file of the parent interpreter (*.snci) you want to inherit from (optional field).
- Launch in Visual Studio Code — a switch: if it is on, the interpreter folder opens in Visual Studio Code right after it is created.

After filling in all the required parameters, click the <Ok> button. As a result, a folder named after the interpreter is created in the directory you specified, and this directory immediately opens in Visual Studio Code (if the Launch in Visual Studio Code switch was on at that moment).

> NOTE
>
> Visual Studio Code must be installed on the computer beforehand.

What to fix after creation

The Create interpreter item copies the template and may not fill in everything. Before you build the interpreter and start debugging (`F5`), check the fields below and fix those that are filled in incorrectly. If all fields are already correct, skip this step.

- The version of the `EncySoftware.NCInterpreterSDK.Net` package in the `*.csproj` file: if it is a prerelease version (with a suffix such as `-dev` or `-rc`) or a version of an older major version, replace it with the released version for your product version, as described in Product versions; otherwise the project may not build.
- The `ID` field in the `*.snci` file: if it is filled with zeros, enter a new GUID.
- For a descendant interpreter, the reference to the parent: `ParentID` must contain the parent's `ID`, and `ParentFileName` the name of the parent's `*.snci` file; without it the parent is not loaded. If `ParentID` contains the parent's `LibraryID` or `ParentFileName` is empty, fix them.

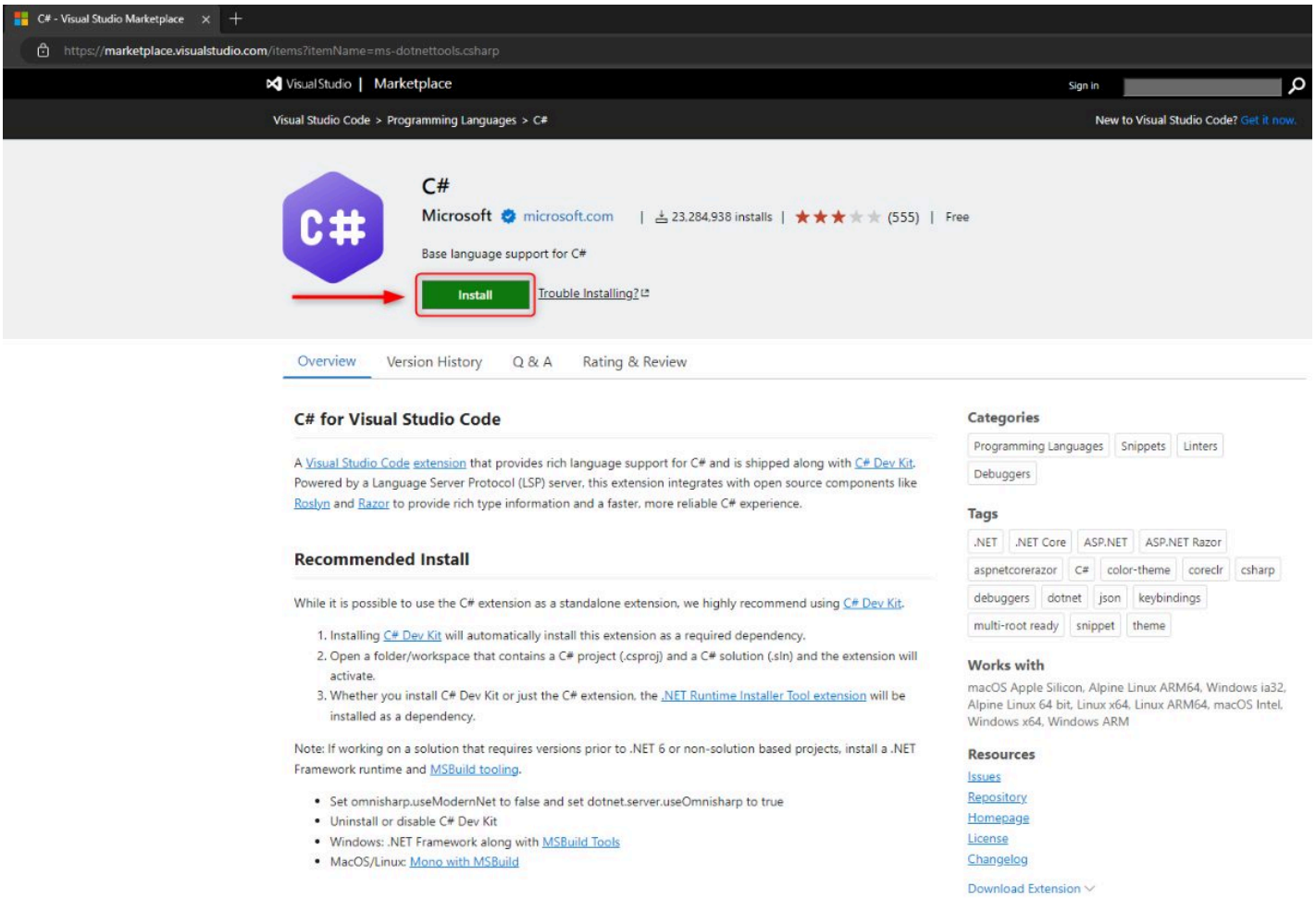
For details, see Where to start in the .NET workflow.

Interpreter debugging in Visual Studio Code

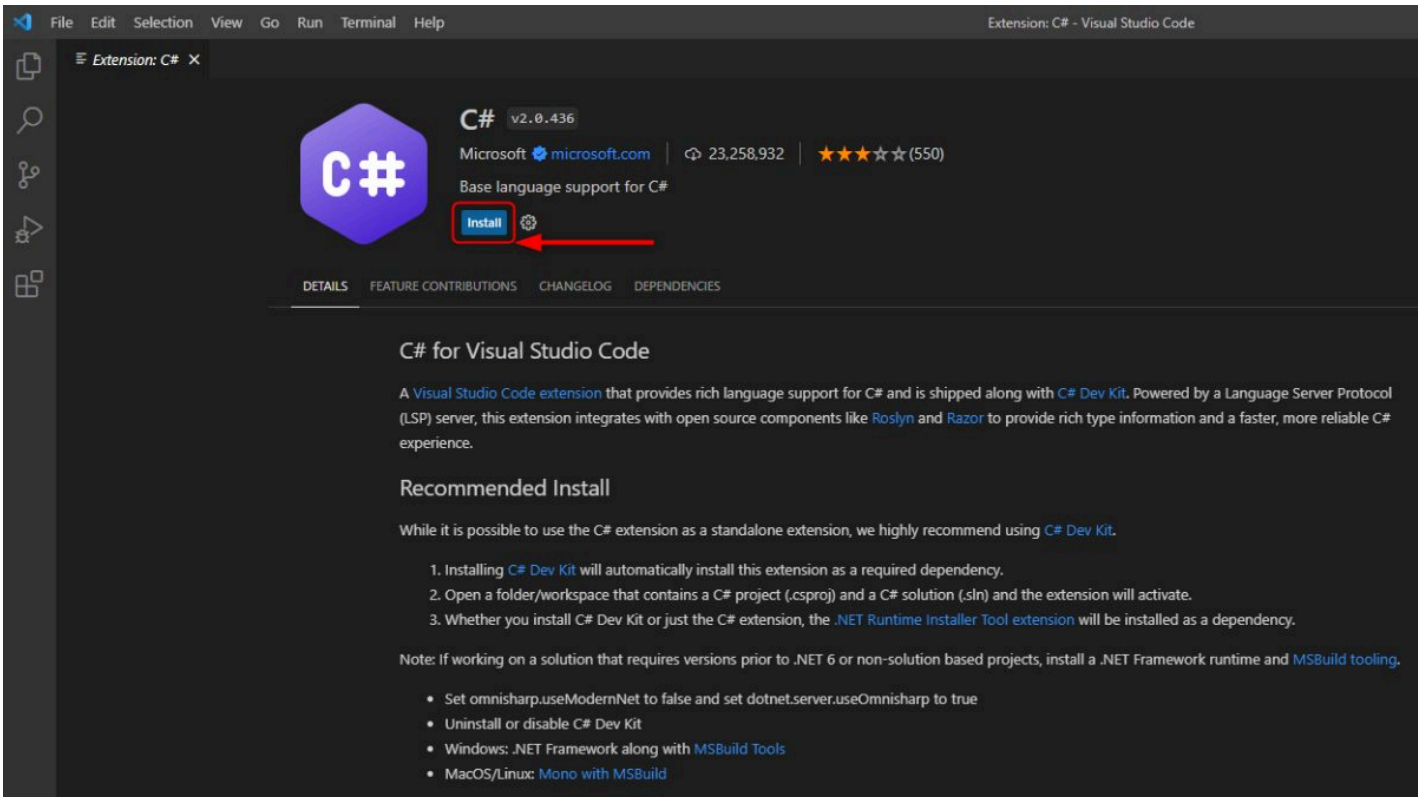
To debug your own .NET-based (C#) G-code interpreters, do the following:

1. Download and install the latest available version of Visual Studio Code (https://code.visualstudio.com/download).

2. Install the C# for Visual Studio Code extension from the Visual Studio Code extension marketplace (https://marketplace.visualstudio.com/items?itemName=ms-dotnettools.csharp).



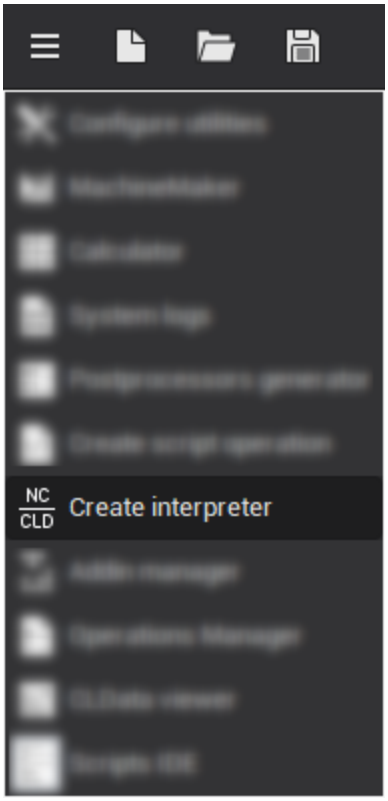
3. After you click the Install button, VS Code (short for Visual Studio Code) opens on the Extensions tab. Just click the Install button next to the extension name.



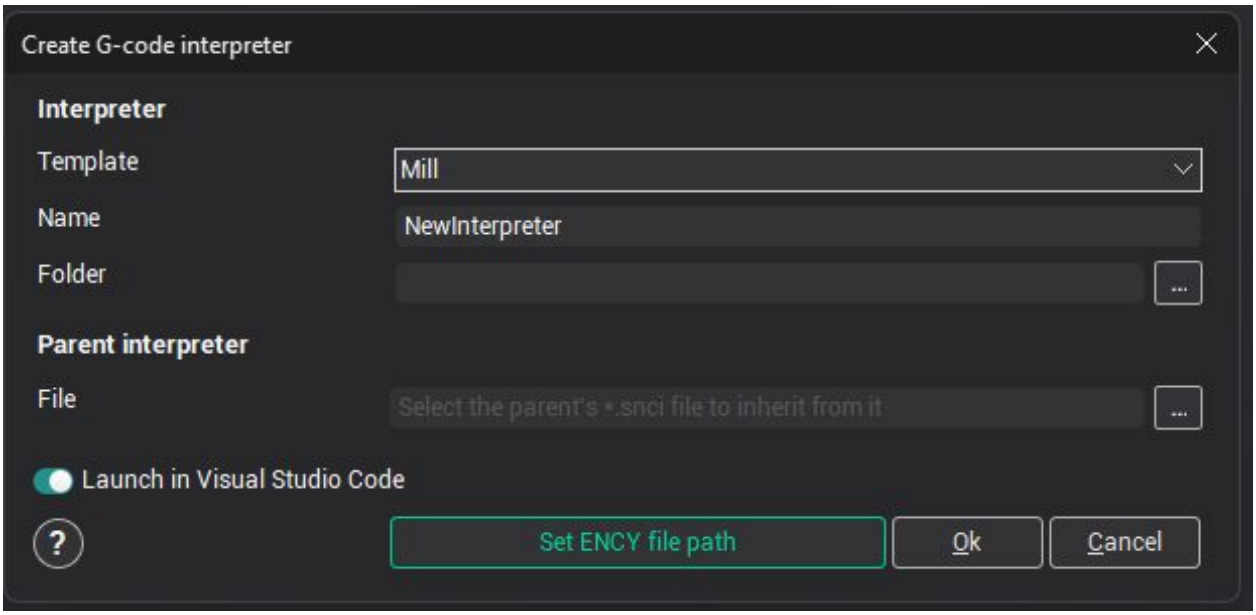
4. For more convenient editing of snci files, it is recommended (but not required) to install the XML Language Support by Red Hat extension (https://marketplace.visualstudio.com/items?itemName=redhat.vscode-xml).

5. If you have not done so yet, download and install the latest version of ENCY (version 3 is required) from the official website (https://encycam.com/download/).

6. Next, launch ENCY and select Create interpreter in the Utilities list.



7. In the window that appears, click the Set ENCY file path button and set the path to the ency.exe file located in the ENCY installation directory (by default, 'C:\Program Files\ENCY Software\ENCY 3\Bin64\ency.exe').



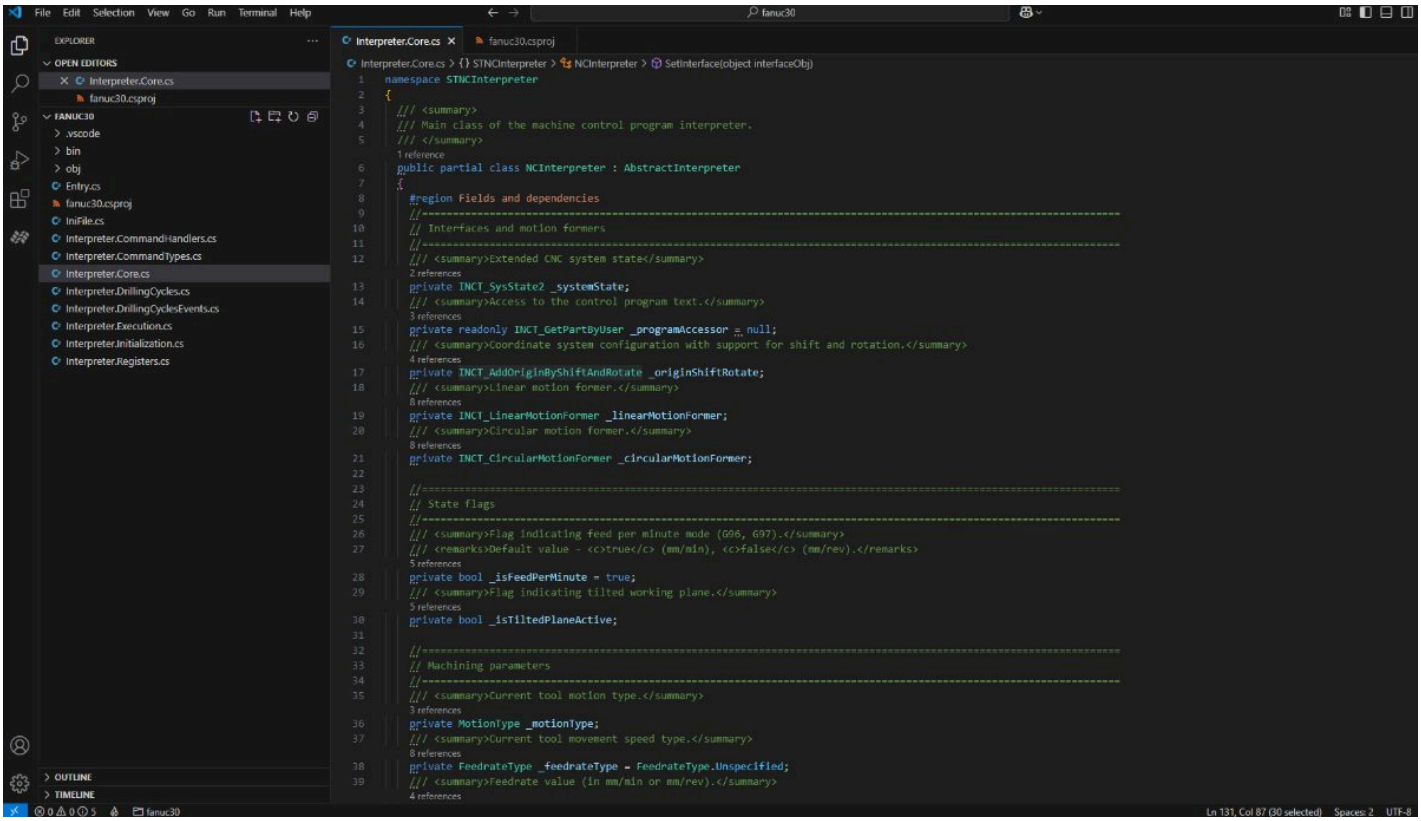
This adds the 'NCKernelDir' environment variable to the Windows environment variables. To be able to launch Visual Studio Code from ENCY, the 'Path' environment variable must include the VS Code installation folder. The Visual Studio Code installer usually adds it by default.

To debug the interpreter in ENCY Tuner, select the Interpreter manager item in ENCY Tuner (it opens the same interpreter creation window), click the Set ENCY Tuner 3 file path button in the window and select the ltuner.exe file in the `bin64` folder of the ENCY Tuner installation folder. This sets the 'TunerDir' environment variable that the 'Launch Tuner' debug configuration uses to start ENCY Tuner.

> ⓘ NOTE
>
> For changes to the Windows environment variables to take effect, after setting the path to the ency.exe executable file (for ENCY Tuner, ltuner.exe) and finishing the VS Code installation, sign out of Windows and sign in again, or at least start VS Code again from the Start menu or File Explorer rather than from a terminal opened earlier.

8. Then launch ENCY again and select Create interpreter in the Utilities list. Now you can create a new interpreter by entering its name in the Name field and a directory in the Folder field; in this directory a folder named after the interpreter will be created with all the necessary source files.

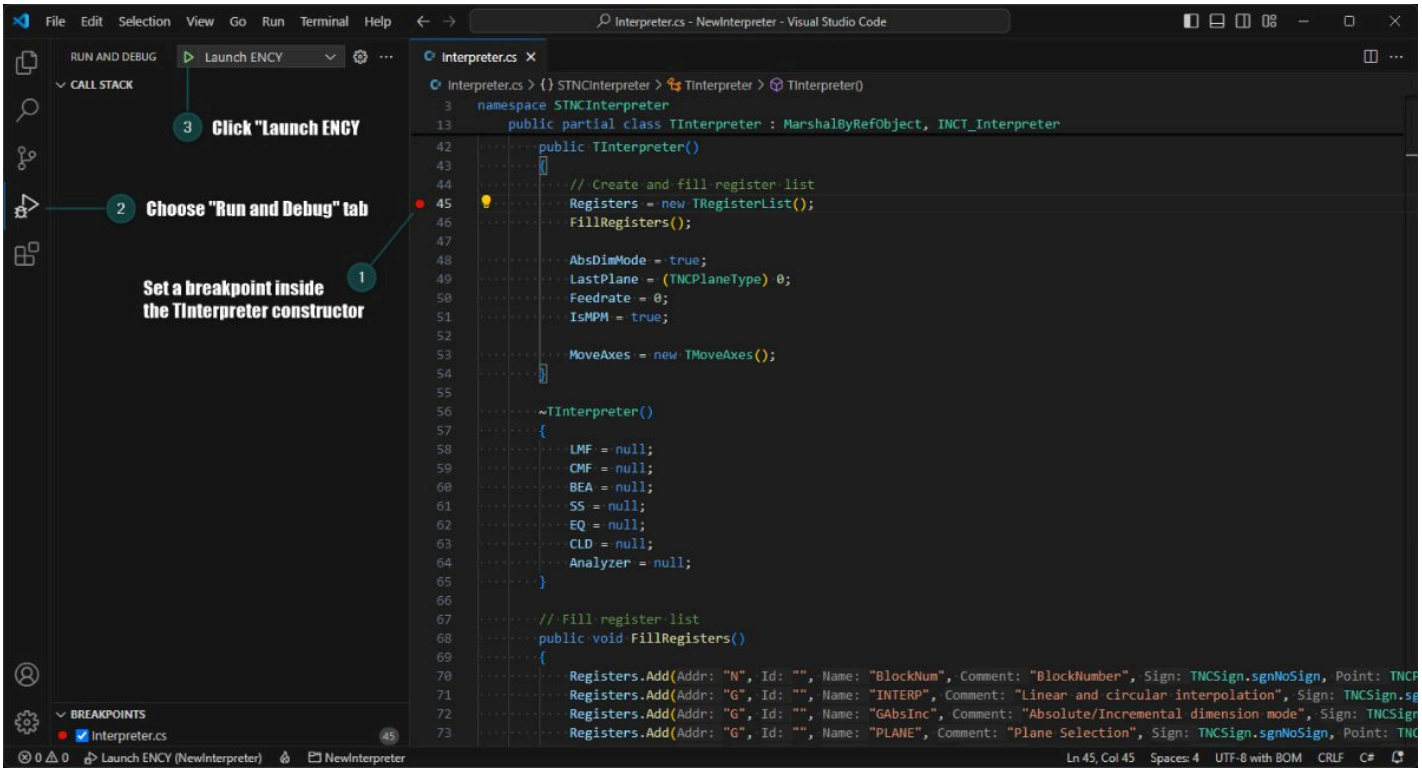
9. After you click the <Ok> button, a new Visual Studio Code window opens with the tab of the main source file of the interpreter, 'Interpreter.Core.cs'.



> **IMPORTANT**
>
> Before you start debugging, check what the Create interpreter item has filled in. If the SDK package version in the `*.csproj` file is a prerelease one (`-dev`, `-rc`) or a version of an older major version, the `ID` field in the `*.snci` file is filled with zeros or, for a descendant interpreter, `ParentID` contains the parent's `LibraryID` or `ParentFileName` is empty, fix these fields. If everything is already correct, skip this step. For what and how to check, see What to fix after creation.

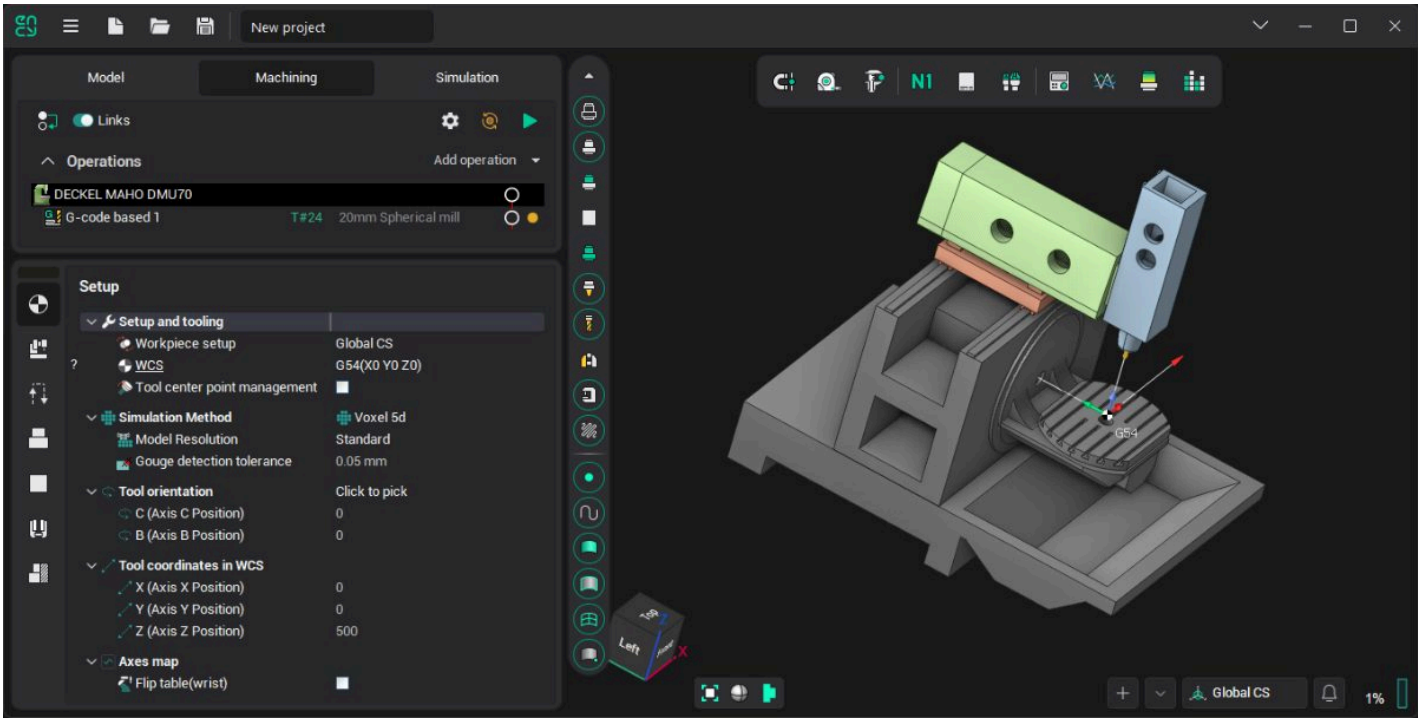
10. Then you can set a breakpoint on any method (for example, the constructor of the NCInterpreter class), switch to the 'Run and Debug' tab and click the 'Launch CAM' button (F5 key; for ENCY

Tuner, select 'Launch Tuner').



11. As a result, the interpreter is built from source, ENCY starts (with Launch Tuner, ENCY Tuner), and debugging of the source code begins.

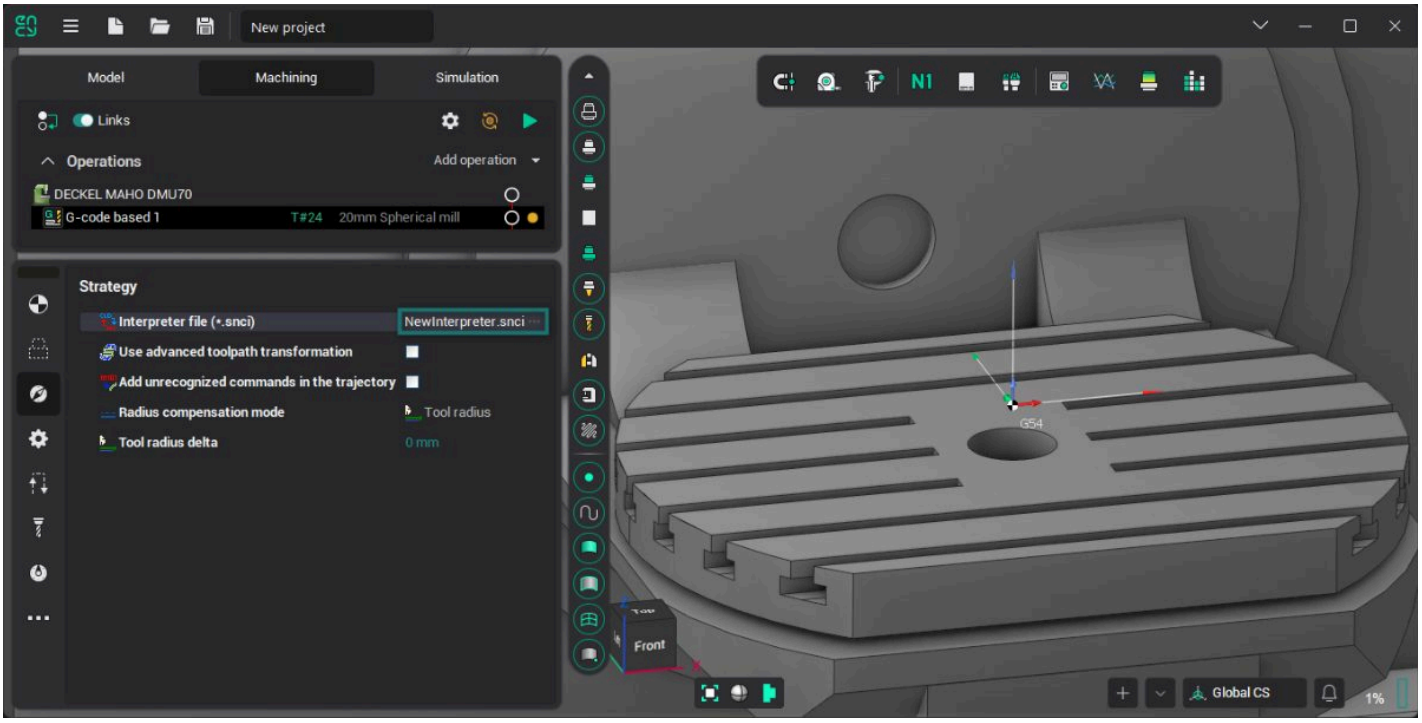
12. To check how the interpreter works, create a G-code based operation in the ENCY window that opens.



13. Enter some NC program text in the <Job assignment> parameters panel of the G-code based operation, for example:

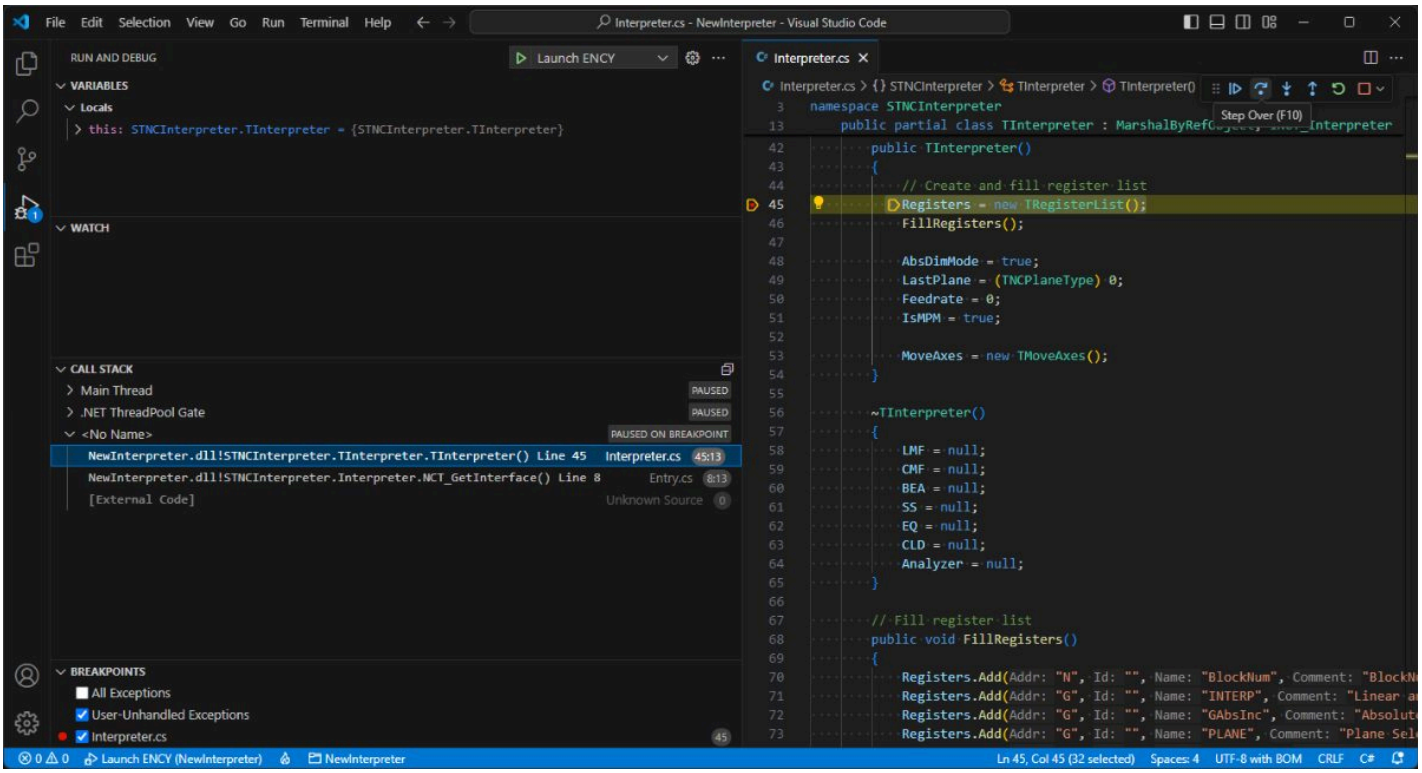
```
G0 G17 X0 Y0 Z0
X100
Y100
X0
Y0
```

14. Select the created interpreter in the <Strategy> operation parameters panel:



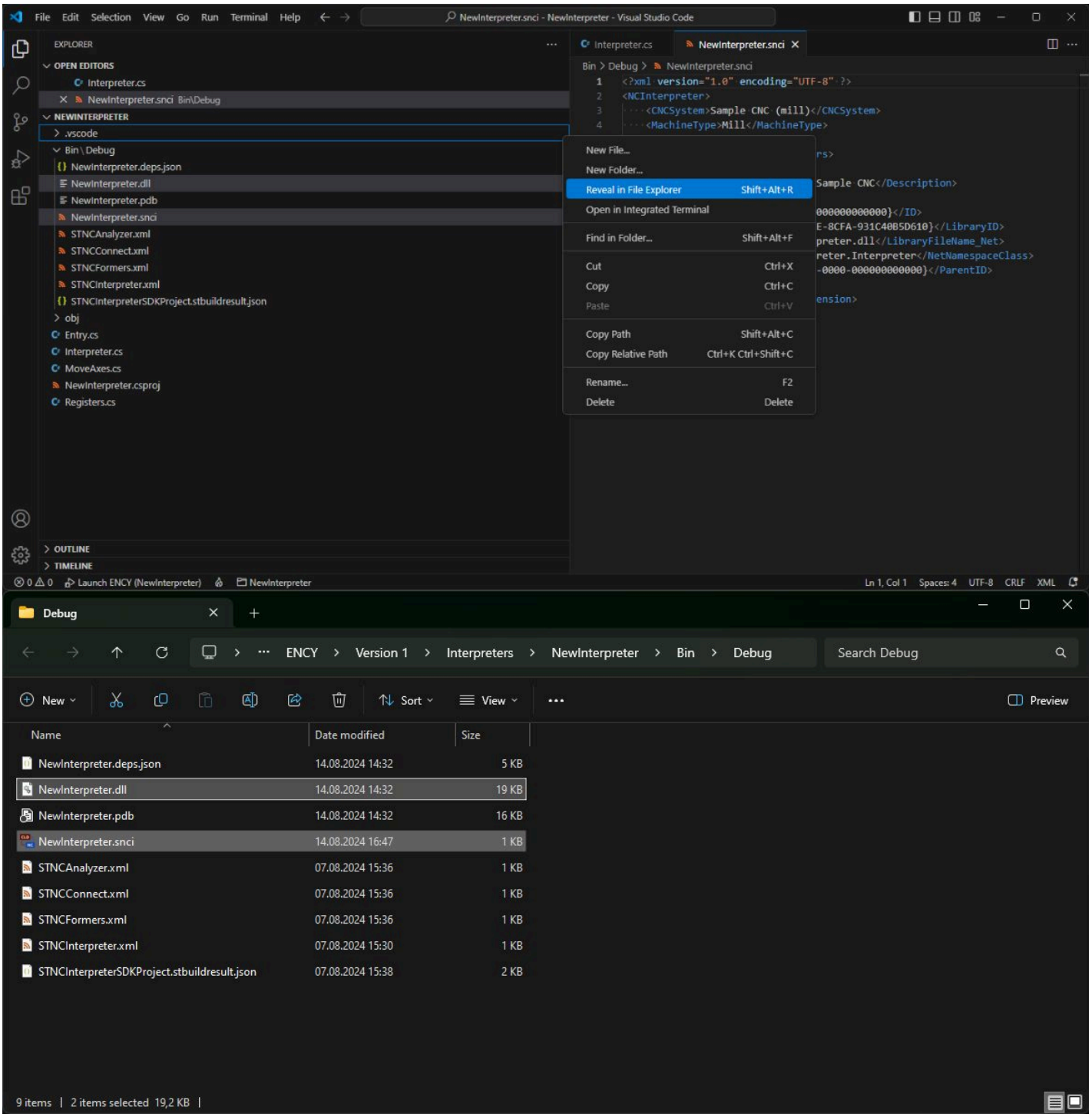
15. And click the operation calculation button Run

16. Execution should stop at the breakpoint you set. Now you can use Step Over (F10) on the toolbar to move through the lines of the interpreter code.



17. Finally, you can change the interpreter code in the Interpreter.Core.cs, Interpreter.Registers.cs and NewInterpreter.snci files as needed and debug it using all the features available in Visual Studio Code. If you are not yet familiar with the capabilities of VS Code, we recommend visiting the "Get Started" section of its official documentation (https://code.visualstudio.com/docs/getstarted/introvideos).

18. Once you have written and debugged the source code of the interpreter, you can get and use the dynamic-link library (*.dll) and interpreter settings (*.snci) files. Just select 'Reveal in File Explorer' in the context menu of the corresponding files (in this case, NewInterpreter.dll and NewInterpreter.snci) in the Visual Studio Code Explorer. The interpreter files will then be selected in File Explorer, and you can copy them and, if necessary, send them to the customer.


File Edit Selection View Go Run Terminal Help
NewInterpreter.snci - NewInterpreter - Visual Studio Code
EXPLORER
OPEN EDITORS
Interpreter.cs
NewInterpreter.snci Bin\Debug
NEWINTERPRETER
.vscode
Bin \ Debug
NewInterpreter.deps.json
NewInterpreter.dll
NewInterpreter.pdb
NewInterpreter.snci
STNCAnalyzer.xml
STNCConnect.xml
STNCFormers.xml
STNCInterpreter.xml
STNCInterpreterSDKProject.stbuildresult.json
obj
Entry.cs
Interpreter.cs
MoveAxes.cs
NewInterpreter.csproj
Registers.cs
OUTLINE
TIMELINE
Interpreter.cs
NewInterpreter.snci
Bin > Debug > NewInterpreter.snci
<?xml version="1.0" encoding="UTF-8" ?>
<NCInterpreter>
<CNCSystem>Sample CNC (mill)</CNCSystem>
<MachineType>Mill</MachineType>
New File...
New Folder...
Reveal in File Explorer Shift+Alt+R
Open in Integrated Terminal
Find in Folder... Shift+Alt+F
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Copy Path Shift+Alt+C
Copy Relative Path Ctrl+K Ctrl+Shift+C
Rename... F2
Delete Delete
Launch ENCY (NewInterpreter)
NewInterpreter
Ln 1, Col 1 Spaces: 4 UTF-8 CRLF XML
Debug
ENCY > Version 1 > Interpreters > NewInterpreter > Bin > Debug
Search Debug
New
Sort
View
Preview
Name
Date modified
Size
NewInterpreter.deps.json 14.08.2024 14:32 5 KB
NewInterpreter.dll 14.08.2024 14:32 19 KB
NewInterpreter.pdb 14.08.2024 14:32 16 KB
NewInterpreter.snci 14.08.2024 16:47 1 KB
STNCAnalyzer.xml 07.08.2024 15:36 1 KB
STNCConnect.xml 07.08.2024 15:36 1 KB
STNCFormers.xml 07.08.2024 15:36 1 KB
STNCInterpreter.xml 07.08.2024 15:30 1 KB
STNCInterpreterSDKProject.stbuildresult.json 07.08.2024 15:38 2 KB
9 items | 2 items selected 19,2 KB |

Interpretation of the NC program text

Interpretation is the process of converting the NC program text, its words and blocks, into CLDATA technological commands that form the tool path. During interpretation, the CAM system, using the interpreter selected for the machine:

- reads the NC program blocks one by one;
- using the register information received from the interpreter, extracts words (or semantic constructions) from the NC program block or lets the interpreter extract the words itself;
- passes the words to the interpreter in the order in which they appear in the block.

The interpreter, taking into account the features of the machine, converts the NC program words into CLDATA technological commands that form the restored tool path.

The interpreter is described in the Interpreter structure section.

The interaction of the CAM system with the interpreter to obtain the secondary (restored) tool path is described in the G-code based program interpretation section.

Interpreter structure

By an NC program interpreter we mean the settings file and the program library referenced by one of the settings file parameters.

Interpreter settings file

The settings file has the SNCI extension (System Numerical Control Interpreter) and the XML format. Its elements are described below. Required parameters are marked in the corresponding column.

The values of the interpreter's own parameters are stored not in the SNCI file but in a separate `*.sncix` file that refers to the interpreter; for how to create it and where to select it, see the section on types for an interpreter with its own parameters on the List of used types page.

Description of the SNCI file elements

Element name	Field format	Description	Required
NCInterpreter	Tag	Root element	*
CNCSystem	String	Controller/machine name	
MachineType	String	Machine type, options: • Lathe — lathe • Mill — milling machine • TurnMill — mill-turn machine • Robots — robot • WireEDM — wire EDM machine	
UseGeomCLData	Boolean	Sets the flag of using the interface for constructing the tool path in geometric axes. True — the INCT_GeomCLData interface is passed from the CAM system to the interpreter to form the tool path.	

Element name	Field format	Description	Required
		False — the INCT_CLData interface is passed from the CAM system to the interpreter to form the tool path. The default value is False.	
Date	Date	Interpreter date in the dd.mm.yyyy format	
Authors	String	Developer(s)	
Company	String	Company	
Description	String	Description	
Version	Number	Version	*
ID	GUID	Interpreter identifier (must be unique)	*
LibraryID	GUID	Program library identifier (must be unique)	*
LibraryFileName_Net	String	Link to the program library for the .NET platform	**
NetNamespaceClass	String	Namespace and name of the class implementing the interpreter (if the LibraryFileName_Net field is filled in)	***
LibraryFileName_x64	String	Link to the native x64 program library (used if	**

Element name	Field format	Description	Required
		the LibraryFileName_Net field is not filled in)	
LibraryFileName_x32	String	Link to the native x32 program library (needed only for an old 32-bit product version)	
ParentID	GUID	Parent interpreter identifier (the ID field in the parent's SNCI file)	*
ParentFileName	String	Link to the parent interpreter settings file (SNCI file)	*
FileExtension	String	Extension of the NC program files	*
Registers	Tag	List of interpreter registers	*

* — required field;

** — either the LibraryFileName_Net or the LibraryFileName_x64 field must be filled in;

*** — the field must be filled in if the LibraryFileName_Net field is filled in.

Description of each element of the register list (Registers):

Element name	Field format	Description
Address	String	Register address
Name	String	Register name
Id	String	Register identifier
Comment	String	Comment
AddrOnly	String	Possible values:

Element name	Field format	Description
		• aoValue — the NC program word consists of an address and a value only • aoAddr — the NC program word consists of an address only • aoBoth — the NC program word may consist either of an address only or of an address and a value
Sign	Integer	Value sign: • 0 — unsigned • 1 — '-' only • 2 — '+' and '-'
NumeralsBeforePoint.Count	Integer	Digits before the point
NumeralsBeforePoint.Zeros	Boolean	Leading zeros
DecimalPoint	Integer	Decimal point: • 0 — no '.' • 1 — '.' • 2 — always '.'
NumeralsAfterPoint.Count	Integer	Digits after the point
NumeralsAfterPoint.Zeros	Boolean	Trailing zeros
Scale	Number	Scale
Modal	Boolean	Register modality flag
ModalGroupList	List of values	List of values the register can take

Fill in the Version tag as in the example below; the CAM system does not use its value (reserved for future use).

The important elements of the interpreter settings file are: its own identifier (ID), the identifier (LibraryID) and the link to the .NET/x64 program library files (LibraryFileName_*), the identifier (ParentID) and the link to the parent interpreter (ParentFileName). The identifiers of the settings file and the program library must be unique (see the article on GUID: Globally Unique Identifier).

The LibraryFileName_* field must point to the program library files relative to the folder where the SNCI file is located. It is recommended to place the program library file in the folder with the SNCI file.

There are two ways to connect a program library to the CAM system:

1. The program library is developed for the .NET platform (for example, in C#).
   In this case, fill in the LibraryFileName_Net and NetNamespaceClass fields, where:
   - LibraryFileName_Net — link to the program library.
   - NetNamespaceClass — namespace and class implementing the interface of interaction with the interpreter. Format: Namespace.Class, for example: Sample.Interpreter.
2. The program library is native (written in Delphi, C++, etc.)

   In this case, the LibraryFileName_x64 field (link to the x64 program library) must be filled in. The x64 version of the program library must be named using the "64" suffix. For example: Sample_x64.dll or Sample_Win64.dll, where "Sample" is the name of the program library.

If the interpreter does not inherit from another interpreter, the parent fields are filled in with empty values:

<ParentID>{00000000-0000-0000-0000-000000000000}</ParentID>

<ParentFileName/>

Fill in the FileExtension tag as in the example below; the CAM system does not use its value (reserved for future use).

The Registers field contains the list of registers for automatic extraction of words from the NC program text. The order of NC program block translation is described in the following article. The register list in the interpreter settings file may be left empty: it will be obtained from the interpreter program library; however, the <Registers/> tag must be present.

The use of the interpreter settings file is described in the G-code based program interpretation section.

Example of the interpreter settings file for a native library (x64):

```
<?xml version="1.0" encoding="UTF-8" ?>
<NCInterpreter>
 <CNCSystem>Sample CNC (mill)</CNCSystem>
 <MachineType>Mill</MachineType>
```

```
  <Date>01.01.2019</Date>
  <Authors>Headquarters Team</Authors>
  <Company>Headquarters</Company>
  <Description>Interpreter for Sample CNC</Description>
  <Version>1</Version>
  <ID>{79B1410C-67D5-42B6-867B-A3BF30AB3D8A}</ID>
  <LibraryID>{80F754D9-EE7B-47CF-A802-A9424D9F6749}</LibraryID>
  <LibraryFileName_x64>Sample_x64.dll</LibraryFileName_x64>
  <ParentID>{00000000-0000-0000-0000-000000000000}</ParentID>
  <ParentFileName/>
  <FileExtension>*.cnc</FileExtension>
  <Registers/>
</NCInterpreter>
```

Download the example Sample.snci.

> **NOTE**
>
> This is a sample of the file format: there is no Sample_x64.dll library for it. Its ID and LibraryID fields are the same as in the sample.snci file of the `sample` interpreter from Examples of interpreters with source codes, and the `sample_inheritance` interpreter finds its parent by the sample.snci file name and its ID. Do not put the sample where the CAM system looks for interpreters, and if you make your own interpreter from it, enter new GUIDs in ID and LibraryID.

Interpreter library

The interpreter program library is a dynamic link library (DLL). Regardless of the platform the library is implemented for (.NET/native), the library must export two functions:

NCT_GetInterface — returns to the CAM system an object implementing the interface for interaction between the CAM system kernel and the program library (INCT_Interpreter);

NCT_FreeMemory — frees memory from all objects created by the library itself or received from the CAM system during operation.

Example of a class describing the interpreter library functions in C# (assembly for the .NET platform):

```
namespace sample
{
    public class Interpreter : MarshalByRefObject
    {
        public static INCT_Interpreter NCT_GetInterface()
        {
            /* implementation */
        }
        public static void NCT_FreeMemory()
        {
            /* implementation */
        }
    }
}
```

In a .NET assembly these functions can be either static or instance methods: the CAM system creates an object of the class specified by the NetNamespaceClass tag of the interpreter settings file, calls its NCT_GetInterface and, when the library is unloaded, NCT_FreeMemory. Therefore the class must have a public parameterless constructor.

Description of the native interpreter library functions in Delphi:

```
function NCT_GetInterface: INCT_Interpreter; stdcall;
procedure NCT_FreeMemory; stdcall;
```

Description of the native interpreter library functions in C++:

```
void __stdcall NCT_GetInterface(INCT_Interpreter** Result);
void __stdcall NCT_FreeMemory();
```

The CAM system declares NCT_GetInterface as a Delphi function returning an interface. Such a result is not returned in a register but through a hidden parameter: the CAM system passes the address to which the library must write the pointer to the INCT_Interpreter object.

All types required for interaction between the CAM system and the interpreter program library are given in the type libraries. They are distributed as NuGet packages published on nuget.org (see Product versions); use packages of the same major version as the SDK package the product is built with: for C++ and other languages — the `EncySoftware.NCInterpreterSDK.tlb` package (`*.tlb`, `*.h` and `*.idl` files), for Delphi — `EncySoftware.NCInterpreterSDK.bpl.x64` (`IDL.<Name>.pas` units). Both packages pull in one package per type library.

The following TLB files contain descriptions of the types used for interaction with the interpreter library:

- STNCInterpreter.tlb — main list of types;
- STNCAnalyzer.tlb — types used to manage NC program analysis;
- STNCFormers.tlb — types used to simplify tool path construction (linear/circular movements, cycles);
- STNCConnect.tlb — types used to protect the interpreter (see Protected interpreters);
- STNCCustomInterpreter.tlb — types for an interpreter with its own parameters (see List of used types).

To connect the type library to your development environment, refer to the corresponding section of the development environment documentation.

> ⓘ NOTE
>
> Registering the type library
>
> To register the type library in Microsoft Visual Studio, try the following:
>
> Find the tlbImp.exe file on your computer. Run the command "tlbimp.exe tlb-filename". The command creates the "tlb-filename.dll" file (for example, "STNCAnalyzer.dll"). Add this dll file to your project via the "Add Reference" menu item.

The Examples of interpreters with source codes page gives an example of the interpreter library source code, the 'dll' built from it, and an example of the settings file.

Examples of interpreters with source codes

The example projects were created in Microsoft Visual Studio for .NET Framework 4 and reference interop assemblies for the STNCInterpreter, STNCAnalyzer and STNCFormers type libraries: `sample_dc` references the `..\tlb\STNC*.dll` files, the others use COM references to registered type libraries. These assemblies are not included in the archives: take the `*.tlb` files from the NuGet package (see Interpreter library) and create the interop assemblies from them with the `tlbimp` tool. A new C# interpreter is easier to create from a CAM system template, see Developing a custom interpreter in (.NET) C#.

1. Interpreters for simulation and tool path import (based on machine CLData)

1.1. Interpreter `sample`.

Implements linear and circular movements and the feed change command for the Fanuc 30i controller (milling).

The source code of the interpreter library developed for Microsoft Visual Studio (C#) is available at this link.

The interpreter based on the compiled library (sample.dll) and its settings file (sample.snci) can be obtained at this link.

1.2. Interpreter with inheritance `sample_inheritance`.

Descendant of the sample interpreter. In addition to the parent's commands, it implements several basic commands: M, S, T, G04 X P (dwell).

The source code of the descendant interpreter library developed for Microsoft Visual Studio (C#) is available at this link.

The descendant interpreter based on the compiled library (sample_inheritance.dll) and its settings file (sample_inheritance.snci) can be obtained at this link.

1.3. Interpreter with inheritance `sample_Fanuc30i_Mill`.

Descendant of the Fanuc30i_Mill interpreter (included in the CAM system distribution). In addition to the parent's commands, it implements the G04 X P command (dwell).

The source code of the descendant interpreter library developed for Microsoft Visual Studio (C#) is available at this link.

The descendant interpreter based on the compiled library (sample_Fanuc30i_Mill.dll) and its settings file (sample_Fanuc30i_Mill.snci) can be obtained at this link.

1.4. Interpreter using the drilling cycle generator `sample_dc`.

Implements the CYCLE81 command for the Sinumerik 840D milling controller.

The source code of the interpreter library developed for Microsoft Visual Studio (C#) is available at this link.

The interpreter based on the compiled library (sample_dc.dll) and its settings file (sample_dc.snci) can be obtained at this link.

The following project demonstrates how the interpreter works. Download the project and the interpreter. Open the project, set the sample_dc.snci interpreter in the G-code based operation settings, calculate the operation and simulate it.

2. Interpreters for tool path import (based on geometric CLData)

2.1. Interpreter `sample_geom`.

Implements linear and circular movements for 3-axis milling and the feed change command for the Fanuc 30i controller.

The source code of the interpreter library developed for Microsoft Visual Studio (C#) is available at this link.

The interpreter based on the compiled library (sample_geom.dll) and its settings file (sample_geom.snci) can be obtained at this link.

2.2. Interpreter `sample_geom_5x`.

Implements linear and circular movements for 5-axis milling (commands: X, Y, Z, TRAORI, TRAFOOF, A3, B3, C3) and the feed change command for the Sinumerik controller.

The source code of the interpreter library developed for Microsoft Visual Studio (C#) is available at this link.

The interpreter based on the compiled library (sample_geom_5x.dll) and its settings file (sample_geom_5x.snci) can be obtained at this link.

The following project demonstrates 5-axis movements. Download the project and the interpreter. Open the project, set the sample_geom_5x.snci interpreter in the G-code based operation settings, calculate the operation and simulate it.

Protected interpreters

To ensure that the interpreter is used in accordance with the issued license, do the following:

1. The interpreter must be developed using the protection interface.
2. The interpreter must be placed in a container.
3. The file or files with secret data must be placed in the container together with the interpreter.

After the container license is issued and activated, the user can use the interpreter, see Simulation of machining by NC program.

G-code based program interpretation

Interpretation of the NC program text generated for operations consists of the following stages:

1. Before interpreting the NC program text of the first operation, the CAM system kernel loads the interpreter into memory and initializes it.
2. For each operation, the NC program text is translated and interpreted.
3. After interpreting the last operation, the CAM system unloads the interpreter from memory.

How to use already developed interpreters to interpret the NC program text of other, similar controllers is described in the Interpreter inheritance article.

The list of interfaces used and implemented in the interpreter is given in the List of used types section.

How to manage logging of the NC program text interpretation is described in the Interpreter interaction log section.

Interpreter loading and initialization

In the CAM system, loading the interpreter into memory consists of the following steps:

- Reading the interpreter settings file.
- Loading the list of registers from the interpreter settings file.
- Loading the parent interpreter into memory (loading of the parent interpreter repeats all the steps of loading the interpreter into memory in the same order).
- Loading the interpreter library into memory and getting the INCT_Interpreter interface (by calling the NCT_GetInterface function of the loaded library).
- Checking that the identifiers of the settings file and the library match (the library identifier is obtained by calling the INCT_Interpreter.GetIdentifier function), as well as the identifiers of the parent interpreter.
- Passing the values of the interpreter's own parameters if the interpreter object implements the INCT_CustomInterpreter interface (calling the INCT_CustomInterpreter.SetCustomValues method; the values are taken from the `*.sncix` file; see List of used types). This step is performed only when the interpreter is loaded for work, not for checking and not for reading its parameters in the Interpreter configurator window.
- Loading the list of registers from the library (calling the INCT_Interpreter.GetRegisters function).
- Loading the lists of registers of the parent interpreters.

As a result of loading the interpreter, the interpreter library and its parent libraries (if any) are loaded, and the resulting (base) list of registers is formed. The registers of the settings file take precedence over the registers of the library (i.e. override them), and the registers of descendant interpreters take precedence over the registers of parent interpreters if a register with the same name is present in both lists.

Initialization of the interpreter consists of the following steps:

- Initialization of the parent interpreters (initialization of the parent interpreter repeats all the steps of initializing the interpreter in the same order).
- The CAM system passes its interfaces to the library one after another (the INCT_Interpreter.SetInterface method of the interpreter library is called several times, according to the number of interfaces passed).
- Initialization of the library (calling the INCT_Interpreter.Initialize method).

As a result of initialization, the interpreter receives the CAM system objects that implement the interfaces for:

- access to the formed (base) list of registers INCT_SysState;
- managing the NC program text analysis settings INCT_Analyzer;
- access to the list of objects to interpret INCT_BlockExecArray;
- getting the equipment settings INCT_Equipment;

- forming the tool path in machine axes (if `UseGeomCLData` = False in the interpreter settings file) INCT_CLData;
- forming the tool path in geometric axes (if `UseGeomCLData` = True) INCT_GeomCLData;
- interaction with the CAM system kernel that serves the interpreter INCT_Kernel;
- the parent interpreter INCT_Interpreter (if any);
- the contour (the interface passed to INCT_Kernel.InterpreteContour), if the interpreter runs in contour mode;
- protection INCT_Connect, for protected interpreters only.

Then the initialization method of the interpreter library is executed.

G-code based program translation and interpretation

Interpretation of the NC program text.

Until the NC program blocks run out or the interpreter library sets the end-of-program flag (the INCT_Interpreter.EndOfProgram method), the CAM system reads the NC program blocks and performs the following steps for each block:

- Translation of the NC program block (analysis of the block for familiar constructions — objects to interpret);
- If the block translation has produced at least one object in the list of objects to interpret of the current block (i.e. the list of objects to interpret INCT_BlockExecArray is not empty), each object from the INCT_BlockExecArray list is interpreted in turn;
- The list of objects to interpret of the current block is cleared.

Translation of the NC program block.

Until the block characters run out, the CAM system performs the following steps:

- The unprocessed part of the block is passed to the interpreter library (i.e. at the first step the whole block is passed). In response, the CAM system expects to receive the number of characters from the beginning of the passed string that the interpreter has recognized as a separate word or another construction it understands (the INCT_Interpreter.TransLine method). The recognized characters are considered processed and are removed from the block. This step is repeated while the interpreter recognizes semantic constructions (words) at the beginning of the rest of the block, i.e. returns to the CAM system a number of recognized characters greater than zero, or until the block ends.
- As soon as the interpreter fails to recognize a semantic construction at the beginning of the rest of the block, the CAM system tries to determine the register and its value by the algorithm described in this article.
- If the register has been determined, it is added to the end of the list of objects to interpret of the current block (INCT_BlockExecArray, with the register flag). Otherwise the read and unrecognized part of the block is skipped, but it is also considered processed and is removed from the beginning of the rest of the block;

Note.

To recognize semantic constructions of the block, we recommend using, wherever possible, the CAM system mechanism for determining registers and values (Algorithm for determining the register and values in the G-code block), and processing in the interpreter (in the INCT_Interpreter.TransLine method) only those constructions that the CAM system mechanism cannot recognize or recognizes inefficiently. In this case the routine work of determining typical constructions is done by the CAM system kernel, and

the interpreter handles rare cases of semantic constructions that do not fit the general rules of the algorithm.

Note.

After the interpreter has recognized a word or a semantic construction, it can store the information about it itself (using its own internal data structures) or use the list of objects to interpret of the current block (INCT_BlockExecArray), i.e. add its own item with the object or register flag to the list. We recommend the second option: in this case the order of objects in the block is preserved regardless of how the object was determined — by the CAM system kernel or by the interpreter.

Interpretation of the objects in the list of objects to interpret consists of the following steps:

- Before the list of objects is interpreted, the INCT_Interpreter.BeforeInterprete method is called; it informs the interpreter that interpretation of the current block begins;
- For each item of the list, if its IsUsed flag is not set and the end-of-translation flag (INCT_Interpreter.EndOfProgram) is not set, the INCT_Interpreter.Interprete method is executed. Then the IsUsed flag is set for the interpreted item of the list;
- After the whole list of objects has been interpreted, the INCT_Interpreter.AfterInterprete method is called; it informs the interpreter that interpretation of the current block is complete;

Note.

When executing the INCT_Interpreter.Interprete method, the interpreter must determine the type of the passed object and add the corresponding command to the list of CLDATA commands.

For example, if the object passed for the Fanuc controller is a register with the address F (feed), you need to execute the INCT_CLData.OutFeed method and pass the register value as the parameter.

Algorithm for determining the register and values in the G-code block

At the stage of recognizing a block of the NC program text, the CAM system kernel operates with tokens — sequences of valid characters of the language that make sense to the translator.

Algorithm for determining the register and value:

1. To determine a register, the kernel reads a token from the NC program line (the order of reading a token is given below).
2. If the token is not a register, the algorithm ends. The token read from the NC program text is considered processed and unrecognized.
3. If the token is a register without a value (i.e. the INCT_Register.Get_ValueAvail function of the register returns vatOnlyAddr), the algorithm ends. The token read from the NC program text is considered processed, the register is determined.
4. Otherwise the kernel reads tokens one after another, expecting to meet a number (signed or unsigned), or the '=' character (if it is allowed by the INCT_Analyzer.CanUseEqualToAssignRegister setting) followed by a number or a string in quotes. Brackets (the INCT_Analyzer.Brackets setting) are not used when determining the register value. Tokens are read until the kernel determines the register value or meets an invalid token. In both cases the algorithm ends and the tokens are considered processed. In the first case it ends successfully — the register is found, the value is determined. In the second case it ends unsuccessfully — the tokens are considered unrecognized.

Order of reading a token:

- The first character of the line is read from the NC program text;
- If the character is a digit or a dot, the token being read is considered a number, and then characters are read from the NC program text as long as they are digits. Reading of the token is completed successfully;
- If the character is one of the list: '+', '-', '=', ',' (and, when the INCT_Analyzer.CanUseEqualToAssignRegister setting is on, one of the brackets specified in the INCT_Analyzer.Brackets setting), the token is considered to consist of one character. Reading of the token is completed successfully.
- If the character is a quotation mark (i.e. it is included in the INCT_Analyzer.Quotes list), the kernel searches for the second quotation mark in the current NC program block. If it is found, the token is considered a string, and reading of it is completed successfully. If the second quotation mark is not found, all the text up to the end of the block is considered processed, and reading of the token is considered unsuccessful.
- If the character is the first character of a comment (i.e. it is included in the INCT_Analyzer.Comments list), the kernel searches for the closing character of the comment. If it is found, the token is considered a comment, and reading of it is completed successfully. If the closing character is not

found, all the text up to the end of the block is considered processed and unrecognized, and reading of the token is considered unsuccessful.

- If the character is a lowercase or uppercase letter of the Latin alphabet, the kernel tries to determine a register. The kernel reads consecutive letters. If the formed (base) list of registers contains compound addresses (for example, 'TOOL CALL', 'DL+', 'DR2'), the kernel continues reading the next parts of the block (letters, digits, '+' or '-', spaces) while they match the parts of these addresses. The INCT_Analyzer.RegistersContainSpaces, INCT_Analyzer.RegistersCanEndWithPlusOrMinus and INCT_Analyzer.RegistersCanEndWithNumber settings are obsolete and do not affect reading of the token. The resulting set of characters is passed to the interpreter to determine the register (the INCT_Interpreter.DefineRegister function). If the interpreter returns a negative answer (the register is not determined), the kernel searches for the register in the formed (base) list of registers by the "Address" field. If the register is found in the list, or the interpreter returned a positive answer at the previous step (the register is found), the token is considered determined — it is a register, and reading of the token is completed successfully. Otherwise the read characters are considered processed, and reading of the token is considered unsuccessful.

Note.

When implementing the function (INCT_Interpreter.DefineRegister), it is recommended to search only for those registers that the CAM system kernel cannot determine unambiguously. In other cases do not search and return a negative result.

Interpreter unloading

Unloading the interpreter from memory consists of the following steps:

- Releasing the INCT_Interpreter interface;
- Freeing the memory of all objects that the library created itself or received from the CAM system during operation (calling the NCT_FreeMemory function of the library);
- Unloading the interpreter library from memory;
- Unloading the parent interpreters from memory (if any).

As a result of unloading, the memory of the created objects is freed, and the interpreter library and its parent interpreters are unloaded from memory.

Interpreter inheritance

The inheritance mechanism is intended for reusing already developed interpreters to translate and interpret NC programs with a similar but different syntax. How much an NC program must differ from the original one to justify a descendant rather than a new interpreter is up to the developer.

The purpose of a descendant interpreter is to translate and interpret the differences in the NC program and to pass control to the parent interpreter for translating and interpreting the similar parts of the NC program text.

A descendant interpreter is implemented differently from an ordinary interpreter in the following:

1. The ParentID and ParentFileName fields are filled in the interpreter settings file (SNCI).
2. If necessary, existing registers are overridden or new ones are added. This can be done by adding the changes to the list of registers of the SNCI file or to the list of registers returned by the INCT_Interpreter.GetRegisters function of the descendant interpreter library.
3. The descendant interpreter library is implemented. In the library, the descendant calls or does not call the functions and methods of the parent interpreter library, depending on the algorithms it uses to translate and interpret the NC program text. The parent interpreter interface (INCT_Interpreter) is passed to the descendant during initialization.

An interpreter can inherit from only one parent. The number of descendant interpreters of one parent and the depth of inheritance are not limited.

An interpreter with its own parameters (its object implements the INCT_CustomInterpreter interface; see List of used types) cannot be a parent: the CAM system stops loading the descendant with an error.

Register inheritance.

The registers of descendant interpreters override the registers of parent interpreters. Register inheritance is implemented so that the resulting list (the list of base registers) is filled in the following order:

1. The list of registers is loaded from the settings file of the descendant interpreter.
2. The list of registers is loaded from the library of the descendant interpreter (INCT_Interpreter.GetRegisters). If a register with the same name already exists in the list of base registers, it is not loaded.
3. The list of registers is loaded from the settings file of the parent interpreter. If a register with the same name already exists in the list of base registers, it is not loaded.
4. The list of registers is loaded from the library of the parent interpreter. If a register with the same name already exists in the list of base registers, it is not loaded.
5. The list of registers is loaded from the settings file of the parent of the parent interpreter. If a register with the same name already exists in the list of base registers, it is not loaded.

6. The list of registers is loaded from the library of the parent of the parent interpreter. If a register with the same name already exists in the list of base registers, it is not loaded.
7. And so on, up to the very first parent.

Library inheritance.

When the parent interpreter library is initialized, a number of methods and functions are called by the CAM system kernel, so the descendant does not need to call them:

1. INCT_Interpreter.GetIdentifier.
2. INCT_Interpreter.GetRegisters.
3. INCT_Interpreter.SetInterface.
4. INCT_Interpreter.Initialize.

The descendant calls or does not call the other functions and methods of the parent interpreter library, depending on the algorithms it uses to translate and interpret the NC program text. To let the descendant call the parent methods, the parent interpreter interface is passed to it, among others, when the descendant interpreter library is initialized (the INCT_Interpreter.SetInterface function).

Below are recommendations for calling the functions and methods of the parent interpreter from a descendant interpreter.

1. INCT_Interpreter.TransLine - call the parent TransLine if analyzing the NC program line did not shorten the original line. Pass all the parameters returned by the parent method (ProcessedChars, NextLine, BreakLine) to the output parameters of the descendant method.
2. INCT_Interpreter.DefineRegister - call the parent DefineRegister if the descendant has not determined the register by its address. Pass the Index parameter and the result returned by the parent function to the output parameter and the result of the descendant function.
3. INCT_Interpreter.BeforeInterprete - always call the parent BeforeInterprete.
4. INCT_Interpreter.Interprete - call the parent Interprete if the descendant does not interpret the object to interpret.
5. INCT_Interpreter.AfterInterprete - always call the parent AfterInterprete.
6. INCT_Interpreter.EndOfProgram - always call the parent EndOfProgram if the descendant has not determined the end of the NC program. Return the result of the parent function as the result of the descendant function.
7. INCT_Interpreter.NextOperation - the CAM system calls NextOperation only for the descendant, at the start of interpreting the next operation. Always call the parent NextOperation.

The page Examples of interpreters with source codes gives an example of the source code of a descendant interpreter library, the 'dll' built from it, and an example of the settings file.

List of used types

All types required for the interaction of the CAM system with the interpreter library are listed in the type libraries. Where to get the packages with them and which version to choose is described on the Interpreter library page.

The following TLB files contain descriptions of the types used for interaction with the interpreter library:

STNCInterpreter.tlb — the main list of types

STNCAnalyzer.tlb — types used to manage the NC program text analysis

STNCFormers.tlb — types used to simplify tool path construction (linear/circular movements, cycles).

STNCConnect.tlb — types used to protect the interpreter.

STNCCustomInterpreter.tlb — types for an interpreter with its own parameters.

The main list of types (STNCInterpreter.tlb)

Types	Type description
TNCExecObj INCT_BlockExecObject INCT_BlockExecArray	List of objects to interpret
TNCSpindleCommand INCT_BaseReg INCT_SysState INCT_SysState2 INCT_SpindleState	Register list
TNCOperationGroup INCT_Equipment INCT_Equipment2 INCT_Equipment_Channels INCT_Channel	Equipment parameters
TNC3DPoint TNCFeedType TNCPlaneType TNCLengthCompensationMode TNCRadiusCompensationMode TNCGotoMode	Forming the tool path in machine axes

Types	Type description
TNCDrillingCycleType TNCLatheCycleType TNCSystemMeasure TNCFeedrateMeasure TNCThreadOrientation TNCThreadLeadType TNCTooltipMode TNCEulerAnglesConvention TNCFlipType TNCMillingCycleType TNCAddOriginFlipMode TNCSPositioningMode TNCOriginSolutionState INCT_CLData INCT_CLData2 INCT_CLData_CutTo5d INCT_CLData_AutoAxesBrakes INCT_CLData_Comment INCT_CLData_BreakCommand INCT_CLData_Insert INCT_CLData_Print INCT_CLData_Wait INCT_CLData_LngID INCT_AddOriginByShiftAndRotate INCT_AddOriginByPhysicAxes INCT_OriginSolver	
INCT_CLData_PocketCycleFormer	Circular pocket milling cycle
TNC3DMatrix INCT_GeomCLData	Forming the tool path in geometric axes
TNCMessageType INCT_Kernel INCT_Kernel2	CAM system kernel interface
TNCGetPartDirection TNCGetPartOrigin INCT_ProgramManager INCT_ProgramManager2	NC program management

Types	Type description
INCT_Program INCT_GetPartByUser	
TNCSign TNCPoint TNCValueAvailType INCT_RegisterValues INCT_Register INCT_RegisterList	List of interpreter registers
INCT_InterpreterState	Turning stock removal cycle
INCT_Interpreter	Interpreter interface

Types used to manage the NC program text analysis (STNCAnalyzer.tlb)

Types	Type description
INCT_Analyzer INCT_Analyzer2 INCT_Comments INCT_Brackets INCT_Quotes INCT_UnionLineChars INCT_EndOfLineChars	Managing NC program text analysis

Types used to simplify tool path construction (STNCFormers.tlb)

Types	Type description
INCT_LinearMotionFormer INCT_GeomLinearMotionFormer	Linear movements
TNCCircularMode TNCConstructArcMode TNCAxisType INCT_CircularMotionFormer	Circular movements
TNCDCAxisIndexes TNCDrillingCycleMoveType TNCReturnLevel	Drilling cycle

Types	Type description
INCT_DrillingCycleEvents INCT_DrillingCycleFormer	
TNCDrillingCycleShiftType TNCShiftAxis TNCShiftValues TNCShiftPath SetOfTNCShiftPath INCT_DrillingCycleWithShiftFormer	Drilling cycle with shift
TNCStartLevelType TNCDegressionType TNCClearanceType TNCRetractionType INCT_PeckDrillingCycleEvents INCT_PeckDrillingCycleFormer	Drilling cycle with chip breaking
TNC2DPoint TNCLCAxisIndexes TNCLatheCycleMoveType TNCGrooveDirection INCT_LatheCycleEvents INCT_LatheGroovingCycleFormer	Turning grooving cycle
TNCThreadApproach TNCThreadCuttingDirection TNCThreadSequenceType INCT_LatheThreadCuttingCycleEvents INCT_LatheThreadCuttingCycleFormer INCT_LatheThreadCuttingCycleFormer2	Turning threading cycle
TNCLSRPointType TNCCutDepthCalcMethod TNCPassReturnMethod TNCMachiningProcessing TNCLatheMachiningType TNCLatheMachiningDirection TNCLatheStepType TNCLatheStartPointMode TNCLatheMachiningStyle	Turning stock removal cycle

Types	Type description
INCT_LatheCycleArcEvents INCT_LatheStockRemovalCycleEvents INCT_LatheStockRemovalCycleEvents2 INCT_LatheStockRemovalCycleEvents3 INCT_LatheStockRemovalCycleFormer INCT_LatheStockRemovalCycleFormer2	
TNCMillingThreadType TNCMillingCycleMoveType INCT_MillingCycleEvents INCT_ThreadMillingCycleFormer	Thread milling cycle
TNCFlatSpiralStepType TNCSpiralDirection TNCTypeOfProcessing TNCPocketCycleMoveType INCT_PocketCycleEvents INCT_PocketCycleFormer	Circular pocket milling cycle

Types used to protect the interpreter (STNCConnect.tlb)

Types	Type description
INCT_Connect	Protection interface

Types for an interpreter with its own parameters (STNCCustomInterpreter.tlb)

Types	Type description
TCustomParameterType INCT_CustomParameter INCT_CustomParameters	Description of the parameters: the name (Name) and type (ParamType: `cpInt`, `cpFlt`, `cpStr`, `cpBol`, `cpArray`, `cpComplex`) of each parameter; parameters of the `cpArray` and `cpComplex` types have nested parameters (Count, Childs, ChildsByName)
INCT_CustomValue INCT_CustomValues	Parameter values: the name (Name) and value (Int, Flt, Str); nested values: Count, Childs, ChildsByName

Types	Type description
INCT_CustomInterpreter	Interface of an interpreter with its own parameters (see below)

INCT_CustomInterpreter is an interface that the interpreter object (the one returned by NCT_GetInterface) implements if the interpreter has its own parameters. The parameter values are stored in a `*.sncix` file: the file refers to the interpreter and stores the values of its parameters; it has no library of its own.

The `*.sncix` file is created in the Interpreter configurator window. The window opens with the Interpreter configurator item in the Utilities list of the CAM system or with the Interpreter configurator item in ENCY Tuner. In the window, select the interpreter file in the Interpreter field, set the parameter values and save them to a `*.sncix` file with the Save as button. For the interpreter to work with these values, select the `*.sncix` file instead of the `*.snci` file wherever the interpreter file is selected: the interpreter file selection window accepts both file types.

Method or property	Description
Parameters	Read-only property that returns INCT_CustomParameters, the list of the interpreter's parameters. The Interpreter configurator window uses it to show the parameters and save their values to the `*.sncix` file.
SetCustomValues(Values: INCT_CustomValues)	The CAM system calls the method after loading the interpreter library, before reading its registers, and passes the parameter values from the `*.sncix` file. If the interpreter is selected by its own `*.snci` file rather than a `*.sncix` file, Values is empty (nil / null).

An interpreter whose object implements INCT_CustomInterpreter cannot be a parent interpreter: the CAM system stops loading the descendant with an error.

Interfaces implemented in the interpreter

This section describes the interfaces that must be implemented in the interpreter, and the protection interface that the CAM system passes to a protected interpreter.

Interpreter interface

INCT_Interpreter - the interface that must be implemented in the interpreter library. The interpreter library returns to the CAM system an object implementing this interface as the result of the NCT_GetInterface function call. Through this interface the CAM system interacts with the interpreter at all stages of interpreting the NC program text.

Description of the interface methods:

Method	Method description	Description of use
GetIdentifier	The function must return to the CAM system a string with the identifier of the interpreter library. No input parameters.	When the interpreter is loaded into memory, the CAM system calls the GetIdentifier function and expects to receive the identifier of the interpreter library. Then the CAM system compares it with the library identifier from the interpreter settings file; if the identifiers match, it continues the loading process.
GetRegisters	The function must return to the CAM system the INCT_RegisterList interface, which implements the list of interpreter registers. No input parameters.	When the interpreter is loaded into memory, the CAM system calls the GetRegisters function and expects to receive the list of interpreter registers, then calls this function for the parents of the interpreter. As a result, the CAM system forms a common list of registers taking inheritance into account.
SetInterface	The method must receive and, if the interpreter needs it, save the interfaces passed from the CAM system for later use. The method is called several times: once for each passed interface.	When the interpreter is initialized, the CAM system passes its interfaces to the interpreter library one by one using the SetInterface method. The passed interfaces can be used when interpreting the NC

Method	Method description	Description of use
	Input parameter: Intf (IUnknown interface). No output parameters.	program text and provide access to: • the formed list of registers INCT_SysState ; • management of the NC program text analysis settings INCT_Analyzer ; • the list of objects to interpret INCT_BlockExecArray ; • the equipment settings INCT_Equipment ; • the tool path generator in machine axes (if `UseGeomCLData` = False in the interpreter settings file) INCT_CLData ; • the tool path generator in geometric axes (if `UseGeomCLData` = True) INCT_GeomCLData ; • interaction with the CAM system kernel serving the interpreter INCT_Kernel ; • the parent interpreter INCT_Interpreter (if any); • the contour (the interface passed to INCT_Kernel.InterpreteContour), if the interpreter runs in contour mode; • protection INCT_Connect, for protected interpreters only.
Initialize	The method informs the interpreter that all CAM system interfaces have been passed and the interpreter can be initialized.	At the end of the interpreter initialization, the CAM system calls the Initialize method. The call informs the interpreter that all CAM system interfaces have

Method	Method description	Description of use
	No input parameters. No output parameters.	been passed, and the interpreter can perform the actions needed for initialization (for example, get the numbers of the machine axes).
NextOperation	The method informs the interpreter that interpretation of the next operation begins. No input parameters. No output parameters.	The CAM system calls this method before interpreting each operation. The interpreter can perform the necessary actions (for example, initialize internal variables, set the order of processing the NC programs of the operation).
TransLine	The method passes to the interpreter an NC program block (or a part of the block, depending on the current iteration of the block translation) to try to read a token or a semantic construction from its beginning. If the interpreter has recognized a token or a construction, it must return the number of characters from the beginning of the string that make up the recognized token or construction. Otherwise, it returns 0. Input parameter: CurrentLine (string) - the passed block or part of the block. Output parameters: • ProcessedChars (integer) - the number of characters from the beginning of the	During the translation of an NC program block, the CAM system lets the interpreter recognize a token or a semantic construction at the beginning of the current block. To do this, the CAM system calls the TransLine method of the interpreter and passes it the current block in the CurrentLine input parameter. In response, the interpreter must return in the ProcessedChars parameter the number of characters from the beginning of the passed string that make up the token or semantic construction, if it has been recognized. Otherwise, if the token or construction has not been recognized, the interpreter returns 0. If the interpreter has recognized a token or semantic construction, it is removed from

Method	Method description	Description of use
	string that the interpreter has recognized (the CAM system does not analyze these characters); 0 if no token or semantic construction has been recognized. • NextLine (boolean) - after the translation of the current block, proceed to the translation of the next NC program block without interpreting the current one. All INCT_BlockExecArray objects of the current block are kept and processed together with the objects of the next block as a single block. • BreakLine (boolean) - interrupt the translation of the current block, keeping the position in the block, and proceed to the interpretation of the formed INCT_BlockExecArray. The next translation starts from the saved position in the current block.	the current string CurrentLine, and the string is passed to the interpreter again by calling the TransLine method to try to read the next token or semantic construction. If the interpreter has not recognized a token or semantic construction, the CAM system tries to determine the register and its value by the algorithm described in this article. After that, all the read characters are removed from the CurrentLine string, and the string is passed to the interpreter by calling the TransLine method to try to read the next token or semantic construction. The process continues until the end of the string or until the BreakLine parameter becomes True. If after the TransLine call the BreakLine output parameter is True, the translation of the current block stops, and the CAM system interprets the translated part of the block. That is, the INCT_BlockExecArray formed by this moment is interpreted. If after the TransLine call the NextLine output parameter is True, then after the translation of the current block is

Method	Method description	Description of use
		completed, the CAM system reads and translates the next NC program block without interpreting the current block. The INCT_BlockExecArray formed by the translation of the current block is not cleared and is supplemented with the objects created during the translation of the next block. When the translation of the next line is completed and the NextLine parameter has not become True again, the INCT_BlockExecArray accumulated by the translation of the two blocks is interpreted. If the NextLine parameter is True again, then after the translation of the second block is completed, the CAM system reads and translates the third block, and so on.
DefineRegister	The function must search for a register by address, return the search result and the index of the found register. Input parameter: Addr (string) - the register address. Output parameter: Index (integer) - the index of the found register. Result (boolean): True - the register is found, False - not found.	When reading a token, the CAM system tries to determine the register in the set of characters read from the NC program block. To do this, the CAM system calls the DefineRegister function of the interpreter interface and passes the read set of characters as the register address Addr. If the interpreter answers positively, the register is considered determined, and its index is contained in the Index output parameter. If the interpreter answers negatively,

Method	Method description	Description of use
		the CAM system tries to find the register by the address in the list of registers. When implementing the function (INCT_Interpreter.DefineRegister), it is recommended to search for and determine only those registers that the CAM system kernel cannot determine unambiguously. In other cases, do not search and return a negative result.
BeforeInterprete	The method informs the interpreter that the translation of the current block is finished and its interpretation begins. No input parameters. No output parameters.	Before the interpretation of the current NC program block, the CAM system calls the BeforeInterprete method of the interpreter. The method is intended to initialize the variables and objects related to the current block, if necessary.

Method	Method description	Description of use
Interprete	The method passes to the interpreter the number of an element from the list of objects to interpret of the current block. The interpreter must interpret the passed object, that is, determine its type by the object number and, if necessary, add a command to form the tool path element(s) to INCT_CLData. Input parameter: ExecNum (integer) - the number of the element in the list of objects to interpret (INCT_BlockExecArray). No output parameters.	During the interpretation of the current NC program block, if the INCT_BlockExecArray list is not empty, the CAM system calls the Interprete method for each object of the list in turn. The method is intended to interpret an object from the list, that is, to form a tool path element from the previously translated object. Example for the Fanuc controller. If the passed object is a register with the address F (feed), then to change the feed of the current tool path, call the INCT_CLData.OutFeed method and pass the register value as the feed parameter.
AfterInterprete	The method informs the interpreter that the interpretation of the current block is finished. No input parameters. No output parameters.	After the interpretation of the current NC program block, the CAM system calls the AfterInterprete method of the interpreter. The method is intended to free the memory taken by the variables and objects related to the processed block, if necessary.
EndOfProgram	The function must return to the CAM system the end of program flag if a program end command was processed during the interpretation of the NC program text. The translation and interpretation of the NC program then stop.	Before each call of the Interprete method, the CAM system calls the EndOfProgram function. If the function returns True, the interpretation of the current block and the translation of the rest of the NC program text stop.

Method	Method description	Description of use
	No input parameters. Result (boolean): True - the program has ended.	

List of interpreter registers

INCT_RegisterList - the interface returned by the interpreter function INCT_Interpreter.GetRegisters. The interface provides work with the list of interpreter registers.

Description of the interface methods:

Method	Method description	Description of use
Get_Count	The function must return the number of interpreter registers to the CAM system. No input parameters.	When forming the common list of registers, the CAM system calls the Get_Count function, gets the number of registers, and then calls the Get_Item function for each register in turn.
Get_Item	The function must return the INCT_Register interface (see below) to the CAM system by the register number. Input parameter: Index (integer) - register number. Numbering starts from 0.	

INCT_Register - the interface returned by the interpreter function INCT_RegisterList.Get_Item. The interface provides work with the parameters of the selected interpreter register.

Description of the interface methods:

Method	Method description	Description of use
Get_Addr	The function must return the register address (string) to the CAM system. No input parameters.	The CAM system uses the register address to find the register in the list of registers.
Get_Id	The function must return the register identifier in the machine kinematic scheme (string) to the CAM system. No input parameters.	The register identifier in the machine kinematic scheme, if applicable. Otherwise, the string is empty.

Method	Method description	Description of use
Get_Name	The function must return the register name (string) to the CAM system. No input parameters.	The unique register name. It is used to identify the register in the list unambiguously.
Get_Comment	The function must return the register comment (string) to the CAM system. No input parameters.	The function is called, but in the current version the value it returns is not used.
Get_Sign	The function must return the register sign type (the TNCSign enumerated type) to the CAM system. sgnNoSign - no sign; sgnMinus - minus only; sgnPlusAndMinus - plus and minus. No input parameters.	The function is called, but in the current version the value it returns is not used.
Get_Point	The function must return the register point type (the TNCPoint enumerated type) to the CAM system. pntNoPoint - no point; pntPoint - the point may or may not be present; pntAlwaysPoint - the point is always present. No input parameters.	The function is called, but in the current version the value it returns is not used.
Get_BeforePoint	The function must return the number of register digits before the point (integer) to the CAM system. No input parameters.	The function is called, but in the current version the value it returns is not used.
Get_AfterPoint	The function must return the number of register digits after the point (integer) to the CAM	The function is used to check the correctness of arcs, so it is important to set the number of

Method	Method description	Description of use
	system. No input parameters.	digits after the point correctly. The tool path generator uses this function inside the kernel.
Get_LeadingZeros	The function must return the flag of leading zeros in the register (boolean) to the CAM system. No input parameters.	The function is called, but in the current version the value it returns is not used.
Get_TrailingZeroes	The function must return the flag of trailing (non-significant) zeros in the register (boolean) to the CAM system. No input parameters.	The function is called, but in the current version the value it returns is not used.
Get_ValueAvail	The function must return the register type (the TNCValueAvailType enumerated type) to the CAM system. vatValue - the register consists of an address and a value; vatOnlyAddr - the register consists of an address only; vatBoth - the register can consist either of an address only, or of an address and a value. No input parameters.	It is used when determining the register and its value.
Get_Values	The function must return the list of register values (the INCT_RegisterValues interface) to the CAM system. No input parameters.	INCT_RegisterValues.Get_Count - the number of values INCT_RegisterValues.Get_Value - the value by index (Index - integer) The list of register values is used when determining the register and its value.
Get_Modal	The function must return the register modality flag (boolean)	It is used when copying register values after the current block is

Method	Method description	Description of use
	to the CAM system. No input parameters.	interpreted. Modal registers get into INCT_BlockExecArray only if the register value has changed in the current block. Non-modal registers always get there if they are present in the current block.

Protection interface

If the container with the interpreter contains the file interpreter_library_name.dat (hereinafter, the secret data file), the INCT_Connect interface is additionally passed to such an interpreter during initialization.

INCT_Connect - the interface that gives the interpreter access to the data from the secret data file. The data file is not extracted from the container and therefore can be read only through the interface passed to the interpreter.

The interpreter must be designed so that it works only after the check of the data received through the INCT_Connect interface has been passed. The places of the checks, their number and the algorithm are chosen by the developer.

Description of the interface methods:

Method	Method description	Description of use
Get_Count	The function returns the number of bytes in the secret data file (integer).	Using these methods, check the data placed in the container by the interpreter developer against the data passed to the interpreter by the CAM system (they must match). Suppose the developer has placed in the container the file sample.dat (the interpreter library is called sample.dll) 5 bytes in size with the following bytes (hexadecimal): 01 02 03 04 05 The Get_Count function returns 5. Get_Data(0) returns 1, Get_Data(2) returns 3, Get_Data(4) returns 5. The Get_SCData method returns in the Data parameter a pointer to the memory containing the data 01 02 03 04 05.
Get_Data	The function returns the byte of the secret data file with the number given in the input parameter. Input parameter: Index (integer) - the number of the returned byte. Bytes are numbered from 0. Result: the byte of the secret data file with the number given in the Index parameter.	
Get_SCData	The method returns the pointer to the secret data from the file. Output parameter: Data (pointer) - the pointer to the secret data.	

CAM system interfaces

This section describes the interfaces implemented by the CAM system.

Basic

This section describes the basic CAM system interfaces implemented for use in the interpreter.

CAM system kernel interface

INCT_Kernel - the CAM system kernel interface. The interface provides work with the NC program text and restarting the interpreter in the contour mode.

Description of the interface methods:

Method	Method description	Description of use
Get_ProgramManager	The function returns the interface for working with the NC program of the current operation (INCT_ProgramManager).	1. If the postprocessor generates several files for the current operation, this interface allows setting the order in which the NC program texts are passed to the interpreter. 2. To get only a part of the NC program text (for example, by labels in the program).
InterpreteContour	The method starts a new interpreter instance to analyze the NC program text and get the contour. Input parameters: NCProgram (INCT_Program interface) - NC program text; Contour (IUnknown interface) - contour interface.	If the generator built into the kernel (Turning stock removal cycle) is not enough to form the tool path of a turning contour cycle (like CYCLE95 for the Sinumerik controller), a new copy of the interpreter can be started to get the geometry from the NC program of the contour path on your own. The InterpreteContour method is intended for this. To form the contour geometry: 1. Get the NC program text by the contour labels with the corresponding function (for example, GetPartByLabels). 2. Implement the Contour object, which must have an

Method	Method description	Description of use
		interface for passing and getting the tool path in the form and format the interpreter developer needs. 3. Call the InterpreteContour method and pass to it the objects obtained in the first and second steps (the NC program text of the contour and the Contour object). While InterpreteContour is executed, the CAM system gets a new INCT_Interpreter instance from the interpreter library, initializes the new interpreter, passes to it in the SetInterface method, in addition to the usual list of interfaces, the Contour interface, and translates and interprets the passed NC program text as in a usual interpretation. During the interpretation, the interpreter, knowing that it works in the contour mode (see Get_IsContourMode), must form the contour path with its own Contour object and the interface developed to store the geometric path. After InterpreteContour finishes, the Contour object contains the geometric contour path ready to be used for forming the tool path in the primary interpreter instance.

Method	Method description	Description of use
Get_IsContourMode	The function returns the flag of whether the interpreter instance works in the contour mode. No input parameters. Result: (boolean) True - in the contour mode, False - in the tool path mode.	If the interpreter starts its new copy itself (see InterpreteContour), the new interpreter instance must know in which mode it is started. If the new interpreter is started in the contour mode, it must form the contour geometry from the NC program text (see the Contour object). If the new interpreter is not started in the contour mode, it must form the tool path from the NC program text.

INCT_Kernel2 - the CAM system kernel interface.

Description of the interface methods:

Method	Method description	Description of use
DoSubprogram	The method switches the translation and interpretation to the NC program text set by the NCProgram parameter, the number of times set by the Count parameter. After that, the translation of the NC program text continues from the place where it stopped. Input parameters: NCProgram (INCT_Program interface) - NC program (subprogram) text; Count (integer) - number of repetitions.	Used to execute subprograms (CALL for Sinumerik, M98 for Fanuc). To get the subprogram text, use one of the following functions: GetPartByLabels, GetPartByLabelsFromCurPos, GetPartByRegisters.
Log	The method outputs a message to the CAM system log. Input parameters: Msg (string) - message text;	Used to debug the interpreter.

Method	Method description	Description of use
	MsgType (TNCMessageType) - message type: nmtInfo - information; nmtSuccess - success; nmtWarning - warning; nmtError - error; nmtCritical - critical; nmtException - exception; nmtDebug - debug.	
Notify	The method outputs a message to the notification panel of the CAM system. Input parameters: Msg (string) - message text; MsgType (TNCMessageType) - message type (see the description in the Log method).	Used to warn the user while the interpreter works.

NC program management

INCT_ProgramManager - an interface for working with the NC program text. The interface provides work with the NC program text and with the order in which the NC program texts are passed to the interpreter.

Description of the interface methods:

Method	Method description	Description of use
Get_Count	The function returns the number of NC program texts (files*) generated by the postprocessor for the current operation. No input parameters.	If the postprocessor generates several NC program texts (files*), it may be necessary to change the order in which the texts are passed to the interpreter. To find out whether there are several NC program texts, use the Get_Count function. In some cases the postprocessor can generate files* with unique names that contain auxiliary data (no NC program text) and must not be interpreted. Or, on the contrary, it can generate a header file from which the other NC program texts are called. The Get_Name function is intended for identifying such files by name. If the order of processing the NC program texts generated by the postprocessor must be changed for the interpretation, use the function of adding an NC program text to the interpretation list (AddToOrderList). The NC program texts are passed to the
Get_Name	The function returns the name of the file* with the NC program text (string) by the index of the NC program text (file*) passed in the input parameter. Input parameter: Index (integer) - number of the NC program text (file*) generated by the postprocessor for the current operation. Value from 0 to Get_Count - 1.	
AddToOrderList	The method adds the NC program text (file*) with the specified index to the interpretation list. The NC program texts are passed to the interpreter in the order set in this list, from the first added to the last. NC program texts not added to the list are not passed to the interpreter. If no NC program text has been added to the list, all files are passed to	

Method	Method description	Description of use
	the interpreter in the order generated by the postprocessor for the current operation. Input parameter: Index (integer) - number of the NC program text (file*) generated by the postprocessor for the current operation. Value from 0 to Get_Count - 1.	interpreter in the order set in the list, from the first added to the last.
GetPartByIndex	The function returns the NC program text (INCT_Program interface) by the index of the NC program text (file*) passed in the input parameter. Input parameter: Index (integer) - number of the NC program text (file*) generated by the postprocessor for the current operation. Value from 0 to Get_Count - 1.	To set the order in which the interpreter processes the NC program texts (files*), it may be necessary to analyze the contents of one or several NC program texts of the current operation. Calling the function with the text number returns the INCT_Program interface; read the lines of the NC program text with its functions.
GetPartByLabels	The function returns the NC program text (INCT_Program interface) by the labels passed in the input parameters. The search is made from the beginning of the NC program text. Input parameters: StartLabel (string), StopLabel (string) - start and end labels of the NC program text.	If the generator built into the kernel (Turning stock removal cycle) is not enough to form the tool path of a turning contour cycle (like CYCLE95 for the Sinumerik controller), the NC program text with the cycle path can be obtained for its own analysis. The described function is intended for this.

* - during postprocessing for interpretation, files with NC program texts are not created on disk. The NC program texts exist only in memory. File names and numbering are kept because of the existing

technology and to set the order of processing the NC program texts.

INCT_ProgramManager2 - an interface for working with the NC program text.

Description of the interface methods:

Method	Method description	Description of use
GetPartByLabelsFromCurPos	The function returns the NC program text (INCT_Program interface) by the labels passed in the input parameters. The search is made from the current position. Input parameters: StartLabel (string), StopLabel (string) - start and end labels of the NC program text; Direction (TNCGetPartDirection) - search direction: gpdForward - forward from the current position, gpdBackward - backward from the current position.	See the use of GetPartByLabels.
GetPartByRegisters	The function returns the NC program text (INCT_Program interface) by the registers passed in the input parameters. Input parameters: StartReg (INCT_Register), StartValues (INCT_RegisterValues), StopReg (INCT_Register), StopValues (INCT_RegisterValues) - registers and values of the start and end of the NC program text; Origin (TNCGetPartOrigin) - search origin: gpoFromCurPos - from the current position of the NC program, gpoEntireScope -	

Method	Method description	Description of use
	from the beginning of the NC program.	

INCT_GetPartByUser - an interface for working with the NC program text.

Description of the interface methods:

Method	Method description	Description of use
GetPartByUser	The function returns the INCT_Program interface for working with the NC program. Input parameter: Origin (TNCGetPartOrigin) - search origin: gpoFromCurPos - search from the current position of the NC program; gpoEntireScope - search from the beginning of the NC program.	The function allows determining the start and end of a subprogram by any specified labels. 1) Get the INCT_Program interface with GetPartByUser. 2) For example, the values of the subprogram start and end labels can be passed in INCT_InterpreterState, or registers can be used for this. 3) After the DoSubprogram procedure of the INCT_Kernel2 interface is started, the subprogram is searched. If INCT_InterpreterState has been used, the previously saved labels can be obtained, or the required subprogram start label can be found in the TransLine method and IsStart called. The IsEnd procedure is used in the same way.
IsStart	The procedure marks the start of the subprogram.	
IsEnd	The procedure marks the end of the subprogram.	

INCT_Program - an interface for getting the NC program text. The interface provides work with the NC program text returned by some INCT_ProgramManager functions.

Description of the interface methods:

Method	Method description	Description of use
EndOfFile	The function returns the end-of-lines flag. True - the NC program text has ended. False - the NC program text has not been read completely.	After the interface is obtained, each next call of GetNextLine returns the next line of the NC program text, from the first to the last line. After the last line is obtained, EndOfFile returns True.
GetNextLine	The function returns the next line of the NC program.	

Register list

INCT_SysState - the interface of the machine-controller system state. Currently the interface provides work with the formed list of registers (taking into account the settings file, the program library and the interpreter parents). To distinguish the formed list of registers from the original registers of the interpreter, the settings file and the parent interpreters, the formed list is called the base list of registers.

Description of the interface methods:

Method	Method description	Description of use
FindFirstRegister	The function returns the first base register found by the address (INCT_BaseReg interface). Input parameter: Addr (string) - register address.	The function can be used to find a base register by its address when analyzing the NC program text in the implementation of INCT_Interpreter.TransLine.
FindNextRegister	The function returns the next base register found by the address, starting from FromIdx (INCT_BaseReg interface). Input parameters: FromIdx (integer) - starting index; Addr (string) - register address.	The function can be used to find a base register by its address when analyzing the NC program text in the implementation of INCT_Interpreter.TransLine.
Get_BR_ByIndex	The function returns a base register by index (INCT_BaseReg interface). Input parameter: Index (integer) - register index.	The function can be used to get a base register by index when iterating over the base registers (for example, to get the initial values of the tool position axes linked to the base registers).
Get_BR_ByName	The function returns a base register by name (INCT_BaseReg interface). Input parameter: Name (string) - register name.	The function can be used to find a base register by name, to determine the register index in the implementation of INCT_Interpreter.DefineRegister.
Get_BR_Count	The function returns the number of base registers.	The function can be used to get the number of base registers for iterating over them (for example,

Method	Method description	Description of use
		to get the initial values of the tool position axes linked to the base registers).

INCT_SysState2 - the interface of the machine-controller system state.

Description of the interface methods:

Method	Method description	Description of use
Get_SpindleState	The function returns the spindle state (INCT_SpindleState interface).	The spindle state is used to store and exchange information between the interpreter and its parent(s).
Get_Reg_ByIndex	The function returns a register by index (INCT_Register interface). Input parameter: Index (integer) - register index.	The functions can be used to find a register by name (Get_Reg_ByName) or to iterate over the registers by index (Get_Reg_ByIndex, Get_Reg_Count) by the required parameter, for example, to get the register values INCT_Register.Values (not available in INCT_BaseReg).
Get_Reg_ByName	The function returns a register by name (INCT_Register interface). Input parameter: Name (string) - register name.	
Get_Reg_Count	The function returns the number of base registers.	

INCT_SpindleState - the spindle state interface. The spindle state is shared by the interpreter and its parents. It is intended for storing and exchanging the spindle state between the interpreter and its parent(s).

Description of the interface methods:

Method	Method description
Get_MaxSpeed	The function returns the maximum spindle speed (real number).
Set_MaxSpeed	The method sets the maximum spindle speed. Input parameter: Value (real number) - maximum spindle speed.
Get_RPMSpeed	The function returns the spindle speed (real number) when the constant cutting speed mode is off (Get_CSS = False).
Set_RPMSpeed	The method sets the spindle speed. Input parameter: Value (real number) - spindle speed.
Get_CSSSpeed	The function returns the spindle speed (real number) for the constant cutting speed mode (Get_CSS = True).
Set_CSSSpeed	The method sets the spindle speed for the constant cutting speed mode. Input parameter: Value (real number) - spindle speed.
Get_Range	The function returns the spindle speed range number (integer).
Set_Range	The method sets the spindle speed range number. Input parameter: Value (integer) - spindle speed range number.
Get_Direction	The function returns the spindle rotation direction (boolean).
Set_Direction	The method sets the spindle rotation direction. Input parameter: Value (boolean) - spindle rotation direction. True - clockwise; False - counterclockwise.

Method	Method description
Get_CSS	The function returns the constant cutting speed flag of the spindle (boolean).
Set_CSS	The method sets the constant cutting speed flag of the spindle. Input parameter: Value (boolean) - constant cutting speed flag of the spindle. True - the constant cutting speed mode is on; False - the constant cutting speed mode is off.
Get_MaxRPM	The function returns the maximum spindle speed for constant cutting speed (real number) (Get_CSS = True).
Set_MaxRPM	The method sets the maximum spindle speed for constant cutting speed. Input parameter: Value (real number) - maximum spindle speed for constant cutting speed.
Get_Angle	The function returns the oriented spindle stop angle (real number).
Set_Angle	The method sets the oriented spindle stop angle. Input parameter: Value (real number) - oriented spindle stop angle.
Get_Command	The function returns the spindle status (the last executed command) (TNCSpindleCommand).
Set_Command	The method sets the spindle status (the last executed command). Input parameter: Value (TNCSpindleCommand) - spindle status. scOn - on; scOff - off; scOrient - oriented stop.
Get_IsChanged	The function returns the flag of a change of any INCT_SpindleState parameter except the spindle number since the last spindle command was executed (boolean).

Method	Method description
	True - one of the parameters has changed; False - the parameters have not changed.
Set_IsChanged	The method sets the flag of a change of any INCT_SpindleState parameter except the spindle number since the last spindle command was executed. Used for interpreter inheritance. The command to turn the spindle on, off or stop it is executed by one of the interpreters, as a rule by the last descendant. Input parameter: Value (boolean) - parameter change flag.
Get_Num	The function returns the spindle number (integer, numbered from 1, as in INCT_CLData.SetActiveSpindle).
Set_Num	The method sets the spindle number. Input parameter: Value (integer) - spindle number.
Get_IsChangedNum	The function returns the flag of a change of the spindle number (boolean).
Set_IsChangedNum	The method sets the flag of a change of the spindle number. Input parameter: Value (boolean) - spindle number change flag.

INCT_BaseReg - the interface provides work with a base register.

Description of the interface methods:

Method	Method description	Description of use
SetValue	The method sets a new value of the base register. The base register has three fields for storing the value: string, real number and integer. If the new value is not empty, it is written to the string field and converted	The method can be used when interpreting tokens and semantic constructions that affect the register value (for example, functions).

Method	Method description	Description of use
	and written to the integer and real number fields. Input parameter: Value (string) - new register value.	
Get_Addr	The function returns the address of the base register (string). No input parameters.	The functions can be used to get the corresponding parameters of the base register when the NC program text is initialized, translated or interpreted.
Get_Name	The function returns the name of the base register (string). No input parameters.	
Get_Id	The function returns the identifier of the base register in the machine kinematic scheme (string). No input parameters.	
Get_Idx	The function returns the index of the base register in the list of registers (integer). No input parameters.	
Get_Str	The function returns the string value of the base register (string). No input parameters.	
Get_Int	The function returns the numeric value of the base register (integer).	

Method	Method description	Description of use
	No input parameters.	
Get_Flt	The function returns the numeric value of the base register (real number). No input parameters.	

List of objects to interpret

INCT_BlockExecArray - a CAM system interface that implements the list of objects of the current block for interpretation.

Description of the interface methods:

Method	Method description	Description of use
Add	The function adds an object for interpretation to the end of the list of objects and returns the index of the added object (integer). Input parameters: ExecType (TNCExecObj) - type of the object added for interpretation; ExecParam (pointer) - parameter of the interpreted object (see the description below).	1. When the NC program text is analyzed beyond the standard analysis of the CAM system kernel (implementation of the INCT_Interpreter.TransLine method), it makes sense to add the recognized tokens and semantic constructions to the list of objects for interpretation (Add method) with the object type (ExecType = exObj), and to fill the ExecParam field with an array index or with a pointer to a structure storing the object data for its later interpretation. 2. When the next command of the current INCT_BlockExecArray list is interpreted and it is known that this command must be interpreted at the end of the block, it can be added to the list with the Add function.
Insert	The function adds an object for interpretation to the list of objects at the number Index and returns the index of the added object (integer).	When the NC program text is translated, the translated objects can be added to the beginning of the interpretation list, or an object can be added after another required object

Method	Method description	Description of use
	Input parameters: Index (integer) - number of the object in the list; ExecType (TNCExecObj) - type of the object added for interpretation; ExecParam (pointer) - parameter of the interpreted object (see the description below).	with the Insert function, if the execution of the interpreted objects requires it.
Get	The function returns the object with the number Index from the interpretation list (INCT_BlockExecObject interface). Important! The result returned by the function is valid only until the next change of the INCT_BlockExecArray list of objects. Input parameters: Index (integer) - number of the object in the list.	The Get function is needed when the objects of the list are interpreted. With it, the interface of the interpreted object (INCT_BlockExecObject) can be obtained by the object number in the list. Then, depending on the object type and purpose, the actions forming the tool path or preparing for them (storing values, initializing variables, etc.) are performed.
Find	The function searches the list for an interpretation object by the ExecType, ExecParam parameters starting from the object with the number FromIdx and returns the found object (INCT_BlockExecObject interface). Important! The result returned by the function is valid only until the next change of the INCT_BlockExecArray list of objects.	The Find function can be used to find out whether a register or an object is present in the already translated part of the block. For example: 1. To add a new interpretation object after this object (Insert function). 2. To determine the register type (when implementing INCT_Interpreter.DefineRegister). Example: for the

Method	Method description	Description of use
	Input parameters: ExecType (TNCExecObj) - type of the interpreted object; ExecParam (pointer) - for a register, the base register number, for an object, the pointer filled by the interpreter; FromIdx (integer) - starting number of the object in the list. Output parameters: Index (integer) - number of the found object in the list.	Sinumerik controller, in the NC program block 'G4 F100' the register 'F' is a dwell, and in the block 'G01 X10 F100' the register 'F' is a feed. When the register is determined, knowing that the translated block already contains the register G4, it can be determined unambiguously that the register 'F' is a dwell and not a feed.
MoveTo	The function moves the interpretation object with the index FromIdx to the place in the list with the index ToIdx and returns the result (boolean). True - success, False - the object has not been moved. Input parameters: FromIdx (integer) - index of the moved object in the list; ToIdx (integer) - target index.	The MoveTo function can be used to sort the list of objects for interpretation after the block is translated. For example, the Heidenhain controller has registers that are always processed at the beginning or at the end of a block regardless of their position in the NC program text.
Get_Count	The function returns the number of interpretation objects in the list (integer). No input parameters.	The Get_Count function can be used when sorting the interpretation list to find out the number of list elements.

INCT_BlockExecObject - an interface of the CAM system (returned by the functions INCT_BlockExecArray.Get, INCT_BlockExecArray.Find) that implements an object for interpretation. The object is created by the functions INCT_BlockExecArray.Add, INCT_BlockExecArray.Insert from the object type (ExecType) and parameter (ExecParam) passed to them.

Description of the interface methods:

Method	Method description	Description of use
Get_ExecType	The function returns the type of the interpreted object (TNCExecObj): exReg - base register; exObj - interpreter object; exProc - reserved, not used by the CAM system kernel. No input parameters.	1. The Get_ExecType function is used when the objects of the list are interpreted. It determines the type of the interpretation object. Depending on the type and parameter of the interpretation object, the actions forming the tool path or preparing for them (storing values, initializing variables, etc.) are performed. 2. The function returns the value specified when the interpretation object was added to the list.
Get_ExecParam	The function returns the parameter of the interpreted object (pointer). If the interpreted object is a register (exReg), the parameter is the base register number. If the interpreted object is an interpreter object, the parameter passed by the interpreter when adding the object (INCT_BlockExecArray.Add, INCT_BlockExecArray.Insert) is returned. No input parameters.	1. The Get_ExecParam function is used when the objects of the list are interpreted. It determines the parameter of the interpretation object filled by the interpreter. Depending on the type and parameter of the interpretation object, the actions forming the tool path or preparing for them (storing values, initializing variables, etc.) are performed. 2. The function returns the value specified when the interpretation object was added to the list.
Get_IsUsed	The function returns the 'interpreted' flag (IsUsed) of the object for interpretation (boolean).	The function is used to check whether the object has been interpreted in the current block.

Method	Method description	Description of use
	True - the object has been interpreted in the current block, False - the object has not been interpreted. No input parameters.	
Set_IsUsed	The method sets the 'interpreted' flag (IsUsed) of the object for interpretation. Input parameter: Value (boolean) - True - the object has been interpreted in the current block, False - the object has not been interpreted.	The method can be used to force the interpretation flag of an object; in this case the object will not be interpreted.
Get_IsOwn	The function returns the 'own' flag (IsOwn) of an object for interpretation of the exObj type. For objects of the exReg type it always returns False. True - the object has been added by the interpreter, False - the object has been added by a descendant or the parent of the interpreter. No input parameters.	The function can be used in a descendant interpreter to decide whether to interpret the command itself or to pass it to the parent for interpretation.

Forming the tool path in machine axes

INCT_CLData - a CAM system interface that implements the commands of forming the tool path in machine axes.

Description of the interface methods:

Method	Method description	Description of use
SetSystemUnits	The method sets the project units. Input parameter: Units (TNCSystemMeasure): smMetrics - metric; smImperial - inch.	To interpret commands like G20, G21 in ISO code.
SetFeedrateUnits	The method sets the feed units. Input parameter: Units (TNCFeedrateMeasure): fmMMPerMinute - mm per minute; fmInchPerMinute - inches per minute.	To interpret commands like G700, G710 for the Sinumerik controller.
GetSystemUnits	The function returns the project units (TNCSystemMeasure).	
GetFeedrateUnits	The function returns the current feed units (TNCFeedrateMeasure).	
AddOrigin	The method adds to the tool path a command that selects a coordinate system by its number. Input parameter: WCSNumber (real number) - coordinate	To interpret commands like G54-G57 in ISO code.

Method	Method description	Description of use
	system number.	
AddOriginByShift	The method adds to the tool path a command that shifts the coordinate system by the specified value. Input parameter: Shift (TNC3DPoint) - shift along the axes (X, Y, Z).	To interpret commands like G92 in ISO code.
OutStandardFeed	The method adds to the tool path a command that changes the feed to a standard one. Input parameter: Feed (TNCFeedType) - feed type: ffWorking - working; ffRapid - rapid; ffFirst - first pass; ffEngage - engage; ffRetract - retract; ffPlunge - plunge; ffFinish - finishing pass; ffNext - transition; ffReturn - return; ffApproach - approach; ffRapid5D - rapid along the safe surface with axis limit control; ffTransitionOnSafe - transition along the safe surface; ffApproachFromSafe - approach from the safe surface; ffReturnToSafe - return to the safe surface; ffLongNext - long transition; ffThreadPitch - equal to the thread pitch.	To interpret commands like G00 (Rapid) / G01 (Working) in ISO code.

Method	Method description	Description of use
	The feed value of each type is set on the 'Feeds/Speeds' tab of the operation in the CAM system.	
OutFeed	The method adds to the tool path a feed change command with the specified type and value. Input parameters: Feed (TNCFeedType) - feed type (see OutStandardFeed); Value (real number) - feed value; MPM (boolean) - feed units flag: True - mm/min, False - mm/rev.	To interpret commands like F in ISO code.
SetCurrentPlane	The method adds to the tool path a command that changes the current plane to the specified one. Input parameter: Plane (TNCPlaneType) - plane type: ptXY - XY plane; ptYZ - YZ plane; ptZX - ZX plane.	To interpret commands like G17-G19 in ISO code.
GetActiveSpindle	The function returns the number of the active spindle, numbering starts from 1.	
SetActiveSpindle	The method makes the spindle with the specified number active. Input parameter: Number (integer) - spindle number,	To interpret commands like SETMS for the Sinumerik controller.

Method	Method description	Description of use
	numbering starts from 1.	
AddSpindleSpeedOnRPM	The method adds to the tool path a command that turns on the active spindle with the specified speed in revolutions per minute. If the spindle is already on, its speed is changed to the specified one. Input parameters: RPM (real number) - spindle speed in revolutions per minute; Range (integer) - spindle speed range number; Direction (boolean) - rotation direction (True - clockwise CW, False - counterclockwise CCW).	To interpret commands like S in ISO code.
AddSpindleSpeedOnCSS	The method adds to the tool path a command that turns on the active spindle with constant cutting speed. If the spindle is already on, its mode and speed are changed to the specified ones. Input parameters: CSS (real number) - constant cutting speed; MaxRPM (real number) - maximum spindle speed; Range (integer) - spindle speed range number; Direction (boolean) - rotation direction (True - clockwise CW, False - counterclockwise CCW).	To interpret commands like G96 for the Sinumerik controller.

Method	Method description	Description of use
AddSpindleOff	The method adds to the tool path a command that turns off the active spindle. No input or output parameters.	To interpret commands like M05 in ISO code.
AddSpindleOrient	The method adds to the tool path an oriented stop command of the active spindle. Input parameter: OrientationAngle (real number) - spindle rotation angle.	
AddStop	The method adds to the tool path a command that interrupts the NC program execution. When the stop block is reached, the NC program execution is interrupted. No input or output parameters.	To interpret commands like M00 in ISO code.
AddOpStop	The method adds to the tool path an optional stop command of the NC program. When the optional stop block is reached, the program execution is interrupted only if the corresponding mode is turned on in the CNC controller. If this mode is off, the program execution is not interrupted. No input or output parameters.	To interpret commands like M01 in ISO code.
AddCoolant	The method adds to the tool path a command that turns the	To interpret commands like M08/M09 in ISO code.

Method	Method description	Description of use
	machine coolant systems on or off. Input parameters: OnOff (boolean) - True - on, False - off; PipeNumber (integer) - number of the coolant pipe (1 - flood, 2 - mist, 3 - through the tool).	
AddDelay	The method adds to the tool path a command that pauses the program execution for the specified time. Input parameter: Value (real number) - dwell time in seconds.	To interpret commands like G04 in ISO code.
LoadTool	The method changes the current tool number to the specified one. Input parameter: ToolID (integer) - tool number.	To interpret commands like T01 in ISO code.
AddLengthCompensation	The method adds to the tool path a tool length compensation command with the specified parameters. Input parameters: Mode (TNCLengthCompensationMode) - mode: lcOff - turn off the tool length compensation; lcOn - turn on the positive tool length compensation; lcOnNegative - turn on the negative tool length compensation.	To interpret commands like G43, G44, G49, H in ISO code.

Method	Method description	Description of use
	OffsetNumber (integer) - length offset number.	
AddLengthCompensationValue	The method adds to the tool path a tool length compensation command with the specified parameters. Input parameters: Mode, OffsetNumber (see AddLengthCompensation); Value (real number) - tool length offset value. Attention! In the current version the function works like AddLengthCompensation, without the possibility to set the value.	
AddLengthCompensationWithSpecifiedAxis	The method adds to the tool path a tool length compensation command with the specified parameters. Input parameters: Mode (see AddLengthCompensation); SpecifiedAxis (integer) - number of the axis along which the length compensation is made; OffsetNumber (see AddLengthCompensation). Attention! In the current version the function works like AddLengthCompensation, without the possibility to select the axis.	To interpret commands like G43 (together with G17-G19) in ISO code.

Method	Method description	Description of use
AddRadiusCompensation	The method adds to the tool path a tool radius compensation command with the specified parameters. Input parameters: Mode (TNCRadiusCompensationMode) - mode: rcOff - turn off the tool radius compensation; rcLeft - turn on the tool radius compensation on the left; rcRight - turn on the tool radius compensation on the right. OffsetNumber (integer) - radius offset number.	To interpret commands like G41, G42, D in ISO code.
AddRadiusCompensationValue	The method adds to the tool path a tool radius compensation command with the specified parameters. Input parameters: Mode, OffsetNumber (see AddRadiusCompensation); Value, Value2 (real number) - tool radius offset values. Attention! In the current version the function works like AddRadiusCompensation, without the possibility to set the value.	
OpenMultiMotion	The method starts the input of multi-axis movements of the controlled machine coordinates with the specified type.	To interpret commands like G00/G01 X Y Z A B C in ISO code (gtmTooltip).

Method	Method description	Description of use
	Input parameter: GotoMode (TNCGotoMode) - movement type: gtmTooltip - movement relative to the current workpiece coordinate system; the base axes are interpolated with control of the path straightness at each intermediate point; gtmPhysic - movement relative to the machine zeros (physical machine coordinates); the intermediate points of the path are not controlled when the base axes are interpolated, only the reachability of the end point is controlled.	To interpret commands like G53 in ISO code, SUPA in Sinumerik, M91 in Heidenhain (gtmPhysic).
AddMotion	The method adds a movement along the specified axis to the specified coordinate. Input parameters: AxisIndex (integer) - index of the machine axis (controlled or physical); Value (real number) - coordinate value. Controlled machine axes can be linear and rotary axes, as well as axes specific to a particular machine (tool magazine, turret, chuck, etc.).	
CloseMultiMotion	The method completes the input of movements and adds to the tool path the multi-axis movement command formed	

Method	Method description	Description of use
	from the movements added by AddMotion since the start of the input by OpenMultiMotion.	
AddCircle	The method adds to the tool path a command of tool movement along a circular arc with the specified parameters. Input parameters: P (TNC3DPoint) - arc end point (X, Y, Z); Pc (TNC3DPoint) - arc center (X, Y, Z); R (real number) - arc radius (greater than zero for counterclockwise rotation, less than zero for clockwise rotation); Plane (TNCPlaneType, see SetCurrentPlane) - arc plane; CanBeFull (boolean) - the arc can be a full circle. The axes linked to the X, Y, Z parameters of the points P, Pc are determined automatically after the tool change command (LoadTool) or after setting the active spindle (SetActiveSpindle). The indexes of the axes currently linked to the X, Y, Z parameters can be obtained with the GetXYZindex function.	To interpret commands like G02/G03 X Y Z I J K R in ISO code.
OpenTooltipMotion	The method starts the input of multi-axis movements with the specified type. Input parameter: Mode (TNCTooltipMode) - movement type:	To interpret linear and arc movements. For linear movements: • OpenTooltipMotion(tmLinear) • SetTooltipPosition

Method	Method description	Description of use
	tmLinear - linear tool movement to the specified point; tmCircularMiddlePoint, tmCircularEndPoint - tool movement along an arc to the specified end point through the middle point; tmPhysic - simultaneous movement of all physical machine axes to the specified point.	• SetTooltipAttitude/SetTooltipAttitude2 • SetTooltipFlip (optional) • AddTooltipMotion (optional) • CloseTooltipMotion For arc movements: • OpenTooltipMotion(tmCircularMiddlePoint) • SetTooltipPosition • SetTooltipAttitude/SetTooltipAttitude2 • SetTooltipFlip (optional) • AddTooltipMotion (optional) • CloseTooltipMotion • OpenTooltipMotion(tmCircularEndPoint) • SetTooltipPosition • SetTooltipAttitude/SetTooltipAttitude2 • SetTooltipFlip (optional) • AddTooltipMotion (optional) • CloseTooltipMotion For movements of all axes: • OpenTooltipMotion(tmPhysic) • SetTooltipPosition • SetTooltipAttitude/SetTooltipAttitude2 • SetTooltipFlip (optional) • AddTooltipMotion (optional) • CloseTooltipMotion
SetTooltipPosition	The method sets the position of the point defined in OpenTooltipMotion in the Cartesian coordinate system relative to the part zero. Input parameter: P (TNC3DPoint) - coordinates of the point.	
SetTooltipAttitude	The method sets the angular movements around the X, Y, Z axes in the Cartesian coordinate system relative to the part zero. Input parameters: Ang1, Ang2, Ang3 (real number) - values of the angular movements; Convention (TNCEulerAnglesConvention) - order of the angular movements: eacXYZ - X, Y, Z eacXZY - X, Z, Y eacYXZ - Y, X, Z	

Method	Method description	Description of use
	eacYZX - Y, Z, X eacZXY - Z, X, Y eacZYX - Z, Y, X eacXYX - X, Y, X eacXZX - X, Z, X eacYXY - Y, X, Y eacYZY - Y, Z, Y eacZXZ - Z, X, Z eacZYZ - Z, Y, Z eacAxisAngle - rotation around an axis whose direction is set by the normalized values Ang1, Ang2, Ang3. The rotation angle around the axis equals the magnitude of the vector Ang1, Ang2, Ang3. RotationsAroundMovableAxes (boolean) - rotation around the moving axes. True - with rotation, False - without rotation. AnglesInDegrees (boolean) - the angular movements are set in degrees. True - in degrees, False - in radians.	
SetTooltipAttitude2	The method sets the tool orientation as a quaternion in the Cartesian coordinate system relative to the part zero. Input parameters: qx, qy, qz, qw (real number) - quaternion components.	
SetTooltipFlip	The method sets the position of the (robot) joints.	

Method	Method description	Description of use
	Input parameters: FlipType (TNCFlipType) - joint type: ftFlipA1 - A1; ftFlipA3 - A3; ftFlipA5 - A5. Value (boolean) - True - flipped position, False - normal position.	
AddTooltipMotion	The method adds a movement along the specified additional axis to the specified coordinate. Input parameters: AxisIndex (integer) - index of the additional machine axis; Value (real number) - coordinate value.	
CloseTooltipMotion	The method completes the input of movements and adds to the tool path the multi-axis movement command formed from the movements added by the *Tooltip* commands since the start of the input by OpenTooltipMotion.	
GetAxisValue	The function returns the coordinate of the current tool position for the specified axis (real number). Input parameter: Index (integer) - axis index.	Used to get the current tool position, for example, to set the hole center when using CreateDrillingCycleFormer.

Method	Method description	Description of use
WasAxisMovement	The function returns the flag of whether the first movement along the specified axis has been made (boolean). True - the first movement has already been made, the current tool position can be obtained with GetAxisValue. False - no movements along the specified axis have been made yet. Input parameter: Index (integer) - axis index.	See GetAxisValue.
AddSinglePassThread	The method adds to the tool path a command that turns on the mode of continuous thread cutting with constant pitch with a turning tool by the specified parameters. Input parameters: Orient (TNCThreadOrientation) - thread orientation; LeadType (TNCThreadLeadType) - way of setting the thread pitch; Step (real number) - thread pitch; StartAngle (real number) - initial angular position of the spindle in degrees (used for multi-start threads). TNCThreadLeadType: ltDistance - by distance; ltThreadsPerUnit - by the number of threads per unit length. TNCThreadOrientation:	To interpret commands like G33 in ISO code.

Method	Method description	Description of use
	toOuter - outer; toInner - inner; toFace - face.	
CreateLinearMotionFormer	The function returns the interface of the linear movement generator (INCT_LinearMotionFormer). Using the generator in the interpreter simplifies forming a linear tool path. Working with the generator is described in the article Linear movements. No input parameters.	To interpret commands like G00/G01 X Y Z A B C in ISO code.
CreateCircularMotionFormer	The function returns the interface of the arc movement generator (INCT_CircularMotionFormer). Using the generator in the interpreter simplifies forming a tool path along an arc. Working with the generator is described in the article Circular movements. No input parameters.	To interpret commands like G02/G03 X Y Z I J K R in ISO code.
CreateDrillingCycleFormer	Depending on the specified parameter, the function returns the interface of the generator of one of the drilling cycles. Working with the generators is described in the articles Drilling cycle, Drilling cycle with shift, Drilling cycle with chip breaking.	To interpret commands like G73, G74, G76, G80-89 for the Fanuc controller.

Method	Method description	Description of use
	Input parameter: CycleType (TNCDrillingCycleType) - cycle type. dctSimple - simple drilling cycle; the returned generator supports only one interface: INCT_DrillingCycleFormer. dctWithShift - drilling cycle with shift; the returned generator supports two interfaces: INCT_DrillingCycleFormer, INCT_DrillingCycleWithShiftFormer. dctPeck - drilling cycle with chip breaking; the returned generator supports two interfaces: INCT_DrillingCycleFormer, INCT_PeckDrillingCycleFormer.	
CreateLatheCycleFormer	Depending on the specified parameter, the function returns the interface of the generator of one of the turning cycles. Working with the generators is described in the articles Turning grooving cycle, Turning threading cycle, Turning stock removal cycle. Input parameter: CycleType (TNCLatheCycleType) - cycle type. lctGrooving - turning grooving cycle; the returned generator supports one interface: INCT_LatheGroovingCycleFormer.	To interpret commands like CYCLE93, CYCLE95, CYCLE97 for the Sinumerik controller.

Method	Method description	Description of use
	IctThreadCutting - turning thread cutting cycle; the returned generator supports one interface: INCT_LatheThreadCuttingCycleFormer. IctStockRemoval - turning cycle of rough and finish contour turning; the returned generator supports one interface: INCT_LatheStockRemovalCycleFormer.	
OutPolarInterpolation	The method turns the polar interpolation on or off. Input parameter: Enable (boolean) - polar interpolation flag. True - the interpolation is on, False - the interpolation is off.	To interpret commands like G132/G133 of the Okuma turning controller.
OutCylindricalInterpolation	The method turns the cylindrical interpolation with the specified radius on or off. Input parameters: Enable (boolean) - cylindrical interpolation flag. True - the interpolation is on, False - the interpolation is off. Radius (real number) - radius.	To interpret commands like G119 of the Okuma turning controller.
AddOriginByShiftAndRotate	The method turns on the shift and rotation of the coordinate system by the specified parameters.	To interpret commands like G68.2, G53.1 of the Fanuc controller.

Method	Method description	Description of use
	Input parameters: Shift (TNC3DPoint) - shift along the axes (X, Y, Z); Ang1, Ang2, Ang3 (real number) - values of the angular movements; Convention (TNCEulerAnglesConvention) - order of the angular movements, see SetTooltipAttitude; RotationsAroundMovableAxes (boolean) - rotation around the moving axes. True - with rotation, False - without rotation; AnglesInDegrees (boolean) - the angular movements are set in degrees. True - in degrees, False - in radians.	
ResetOrigin	The method turns off the shift and rotation of the coordinate system set in AddOriginByShiftAndRotate.	
SetRTCPMode	The method adds to the tool path a command that turns the RTCP mode (five-axis interpolation with tool center point control) on or off. Input parameter: Enable (boolean) - True - the mode is on, False - the mode is off.	To interpret commands like G43.4/G49 for the Fanuc controller, TRAORI/TRAFOOF for the Sinumerik controller, M128/M129 for the Heidenhain controller.

INCT_AddOriginByShiftAndRotate - a CAM system interface that turns on the shift and rotation of the coordinate system with a choice of the rotary axes solution (the FlipMode parameter).

Description of the interface methods:

Method	Method description	Description of use
AddOriginByShiftAndRotate	The method turns on the shift and rotation of the coordinate system by the specified parameters. Input parameters: Shift (TNC3DPoint) - shift along the axes (X, Y, Z); Ang1, Ang2, Ang3 (real number) - values of the angular movements; Convention (TNCEulerAnglesConvention) - order of the angular movements, see SetTooltipAttitude; RotationsAroundMovableAxes (boolean) - rotation around the moving axes. True - with rotation, False - without rotation; AnglesInDegrees (boolean) - the angular movements are set in degrees. True - in degrees, False - in radians; PositioningMode (TNCSPositioningMode) - type of positioning the machine axes by the local coordinate system: cspmStay - the local coordinate system does not move the machine axes; cspmTurn - the local coordinate system positions only the rotary machine axes; cspmMove - the local coordinate system positions all machine	To interpret commands like G68.2, G53.1 of the Fanuc controller.

Method	Method description	Description of use
	axes so that the position of the tool tip relative to the part does not change; FlipMode (TNCAddOriginFlipMode) - solution for the tool position relative to the workpiece if the machine can reach the same position in several ways: ofmNearest - nearest; ofmDirect - direct; ofmAlternate - alternative.	

INCT_CLData2 - a CAM system interface that implements the commands of forming the tool path in machine axes.

Description of the interface methods:

Method	Method description	Description of use
CreateMillingCycleFormer	The function returns the interface of the milling cycle generator. Working with the generators is described in the article Thread milling cycle. Input parameter: CycleType (TNCMillingCycleType) - cycle type. mctThreadMilling - thread milling; the returned generator supports the interface INCT_ThreadMillingCycleFormer.	To interpret commands like CYCLE90 for the Sinumerik controller.

INCT_CLData_CutTo5d - a CAM system interface that implements the commands of tool movement by normal vectors.

Description of the interface methods:

Method	Method description	Description of use
CutTo5d	Used for movements by normal vectors. Input parameters: P (TNC3DPoint) - coordinates (X, Y, Z), real numbers; N (TNC3DPoint) - normal vector (components from -1 to 1), real numbers.	To interpret commands like TRAORI for the Sinumerik controller with the normal vectors A3 B3 C3.

INCT_CLData_AutoAxesBrakes - a CAM system interface that turns the automatic output of the axis brakes on or off.

Description of the interface methods:

Method	Method description	Description of use
Get_AutoAxesBrakes	Gets the state of the automatic output of the axis brakes. Output parameters: Value (boolean) - automatic output of the axis brake. False - the automatic output of the axis brake is off. True - the automatic output of the axis brake is on.	Used when there is no need to output to CLData the information about turning the axis brakes on or off.
Set_AutoAxesBrakes	Turns the automatic output of the axis brakes on or off. Input parameter: Value (boolean) - automatic output of the axis brake. False - turn off the automatic output of the axis brake. True - turn on the automatic output of the axis brake.	

INCT_CLData_Comment - a CAM system interface that allows adding custom comments to CLData.

Description of the interface methods:

Method	Method description	Description of use
AddComment	The method adds a comment line to CLData. Input parameter: Value (string) - comment text.	Adding explanations to individual commands or groups of commands. Adding service information for process engineers or machine operators.

The interfaces described below are requested from the INCT_CLData object. Check the result of the request: in earlier versions of the CAM system and in some of its operating modes the object may not support the interface.

INCT_CLData_BreakCommand - a CAM system interface that implements the command of turning the machine axis brakes on or off.

Description of the interface methods:

Method	Method description	Description of use
Open	The method starts the input of the axis brake command. No input or output parameters.	To interpret axis clamp and unclamp commands, for example M10/M11. Call order: Open, then SetBreak for each axis, then Close. Usually used together with turning off the automatic output of the axis brakes (Set_AutoAxesBrakes of the INCT_CLData_AutoAxesBrakes interface).
SetBreak	The method sets the brake state of an axis in the command. Input parameters: AxisIndex (integer) - machine axis index; Value (boolean) - True - the brake is on, False - the brake is off.	
Close	The method completes the input and adds to the tool path the axis brake command for the	

Method	Method description	Description of use
	axes set by SetBreak since Open. If no axis has been set, the command is not added. No input or output parameters.	

INCT_CLData_Insert - a CAM system interface that allows adding an INSERT command with arbitrary text to CLData.

Description of the interface methods:

Method	Method description	Description of use
AddInsert	The method adds to the tool path an INSERT command with the specified text. Input parameter: Value (string) - command text.	To pass to the postprocessor text that the CAM system does not simulate, for example auxiliary machine commands. The postprocessor determines how the text gets into the NC program.

INCT_CLData_Print - a CAM system interface that allows adding a PPRINT command with arbitrary text to CLData. The interface is supported by the INCT_CLData and INCT_GeomCLData objects.

Description of the interface methods:

Method	Method description	Description of use
AddPrint	The method adds to the tool path a PPRINT command with the specified text. Input parameter: Value (string) - command text.	To interpret commands like PPRINT in APT.

INCT_CLData_Wait - a CAM system interface that implements the channel synchronization of a multichannel machine.

Description of the interface methods:

Method	Method description	Description of use
AddWait	The method adds to the tool path a channel synchronization point with the specified number. The channels in which a synchronization point with the same number occurs wait for each other. Input parameter: Id (integer) - synchronization point number.	To interpret wait commands in multichannel programs, like WAITM for the Sinumerik controller.

INCT_CLData_LngID - a CAM system interface that returns the CAM system interface language.

Description of the interface methods:

Method	Method description	Description of use
GetLngID	The function returns the code of the CAM system interface language (integer) - the Windows language code, for example 1049 - Russian, 1033 - English. No input parameters.	To output interpreter messages in the CAM system interface language.

INCT_AddOriginByPhysicAxes - a CAM system interface that sets a coordinate system by the position of the machine axes.

Description of the interface methods:

Method	Method description	Description of use
AddOriginByPhysicAxes	The method adds to the tool path a command that shifts and rotates the coordinate system defined by the position of the machine axes. In the calculation, the axes with the indexes AxisIndex1-AxisIndex3 are set to the values Ang1-Ang3: the orientation of the new	To interpret commands like PLANE AXIAL for the Heidenhain controller.

Method	Method description	Description of use
	coordinate system matches the tool orientation in this machine position, and its origin is shifted by Shift relative to the current part coordinate system. The movement of the machine axes is determined by the PositioningMode parameter. Input parameters: Shift (TNC3DPoint) - shift of the coordinate system origin along the axes (X, Y, Z); AxisIndex1, AxisIndex2, AxisIndex3 (integer) - machine axis indexes, -1 - the axis is not set; Ang1, Ang2, Ang3 (real number) - values of the axes AxisIndex1-AxisIndex3 in the same units as the coordinate value in AddMotion; PositioningMode (TNCSPositioningMode) - type of positioning the machine axes by the local coordinate system, see INCT_AddOriginByShiftAndRotate. If no axis is set, the command is not added.	

INCT_OriginSolver - a CAM system interface that allows calculating in advance the machine axis positions for a coordinate system rotation without changing the machine state and without adding commands to the tool path.

Description of the interface methods:

Method	Method description	Description of use
CalculateOriginSolutions	The method calculates the machine axis positions for the	To interpret commands like PLANE SPATIAL ... SEQ+/SEQ-

Method	Method description	Description of use
	coordinate system defined by a shift and a rotation. The solutions are calculated at once for three TNCAddOriginFlipMode modes: ofmDirect - direct, ofmAlternate - alternative, ofmNearest - the nearest of the possible ones. The results are returned by the GetOriginSolutionState and GetOriginSolutionAxisValue functions. Input parameters: Shift, Ang1, Ang2, Ang3, Convention, RotationsAroundMovableAxes, AnglesInDegrees - as in the AddOriginByShiftAndRotate method.	for the Heidenhain controller: allows choosing a solution by the sign of the main rotary axis in advance, checking whether it is reachable, and then passing the selected mode to INCT_AddOriginByShiftAndRotate .AddOriginByShiftAndRotate (the FlipMode parameter).
GetOriginSolutionState	The function returns the state of the solution for the specified mode (TNCOriginSolutionState): ossNotCalculated - the solution has not been calculated; ossCalcError - there is no solution; ossOutOfLimits - the solution is outside the machine axis limits; ossReachable - the solution is reachable. Input parameter: FlipMode (TNCAddOriginFlipMode) - mode.	
GetOriginSolutionAxisValue	The function returns the value of a machine axis in the solution for the specified mode (real	

Method	Method description	Description of use
	number). If the axis does not participate in the solution, 0 is returned. Input parameters: FlipMode (TNCAddOriginFlipMode) - mode; AxisIndex (integer) - machine axis index.	
GetFirstRotaryAxisAfterCurrentTool	The function returns the index of the rotary axis nearest to the current tool in the machine kinematic chain (integer). Spindles are not taken into account. If there is no such axis, -1 is returned. No input parameters.	
GetLastRotaryAxisFromCurrentWorkpiece	The function returns the index of the rotary axis on the side of the current workpiece that is farthest from it in the machine kinematic chain (integer). Spindles are not taken into account. If there is no such axis, -1 is returned. No input parameters.	

Forming the tool path in geometric axes

INCT_GeomCLData - a CAM system interface that implements the commands of forming the tool path in geometric axes.

Description of the interface methods:

Method	Method description	Description of use
SetSystemUnits	The method sets the project units. Input parameter: Units (TNCSystemMeasure): smMetrics - metric; smImperial - inch.	To interpret commands like G20, G21 in ISO code.
SetFeedrateUnits	The method sets the feed units. Input parameter: Units (TNCFeedrateMeasure): fmMMPerMinute - mm per minute; fmInchPerMinute - inches per minute.	To interpret commands like G700, G710 for the Sinumerik controller.
GetSystemUnits	The function returns the project units (TNCSystemMeasure).	
GetFeedrateUnits	The function returns the current feed units (TNCFeedrateMeasure).	
OutStandardFeed	The method adds to the tool path a command that changes the feed to a standard one. Input parameter: Feed (TNCFeedType) - feed type:	To interpret commands like G00 (Rapid) / G01 (Working) in ISO code.

Method	Method description	Description of use
	ffWorking - working; ffRapid - rapid; ffFirst - first pass; ffEngage - engage; ffRetract - retract; ffPlunge - plunge; ffFinish - finishing pass; ffNext - transition; ffReturn - return; ffApproach - approach; ffRapid5D - rapid along the safe surface with axis limit control; ffTransitionOnSafe - transition along the safe surface; ffApproachFromSafe - approach from the safe surface; ffReturnToSafe - return to the safe surface; ffLongNext - long transition; ffThreadPitch - equal to the thread pitch. The feed value of each type is set on the 'Feeds/Speeds' tab of the operation in the CAM system.	
OutFeed	The method adds to the tool path a feed change command with the specified type and value. Input parameters: Feed (TNCFeedType) - feed type (see OutStandardFeed); Value (real number) - feed value; MPM (boolean) - feed units flag: True - mm/min, False - mm/rev.	To interpret commands like F in ISO code.

Method	Method description	Description of use
SetCurrentPlane	The method adds to the tool path a command that changes the current plane to the specified one. Input parameter: Plane (TNCPlaneType) - plane type: ptXY - XY plane; ptYZ - YZ plane; ptZX - ZX plane.	To interpret commands like G17-G19 in ISO code.
AddSpindleSpeedOnRPM	The method adds to the tool path a command that turns on the active spindle with the specified speed in revolutions per minute. If the spindle is already on, its speed is changed to the specified one. Input parameters: RPM (real number) - spindle speed in revolutions per minute; Range (integer) - spindle speed range number; Direction (boolean) - rotation direction (True - clockwise CW, False - counterclockwise CCW).	To interpret commands like S in ISO code.
AddSpindleSpeedOnCSS	The method adds to the tool path a command that turns on the active spindle with constant cutting speed. If the spindle is already on, its mode and speed are changed to the specified ones. Input parameters: CSS (real number) - constant cutting	To interpret commands like G96 for the Sinumerik controller.

Method	Method description	Description of use
	speed; MaxRPM (real number) - maximum spindle speed; Range (integer) - spindle speed range number; Direction (boolean) - rotation direction (True - clockwise CW, False - counterclockwise CCW).	
AddSpindleOff	The method adds to the tool path a command that turns off the active spindle. No input or output parameters.	To interpret commands like M05 in ISO code.
AddSpindleOrient	The method adds to the tool path an oriented stop command of the active spindle. Input parameter: OrientationAngle (real number) - spindle rotation angle.	
AddOpStop	The method adds to the tool path an optional stop command of the NC program. When the optional stop block is reached, the program execution is interrupted only if the corresponding mode is turned on in the CNC controller. If this mode is off, the program execution is not interrupted. No input or output parameters.	To interpret commands like M01 in ISO code.
AddCoolant	The method adds to the tool path a command that turns the	To interpret commands like M08/M09 in ISO code.

Method	Method description	Description of use
	machine coolant systems on or off. Input parameters: OnOff (boolean) - True - on, False - off; PipeNumber (integer) - number of the coolant pipe (1 - flood, 2 - mist, 3 - through the tool).	
AddDelay	The method adds to the tool path a command that pauses the program execution for the specified time. Input parameter: Value (real number) - dwell time in seconds.	To interpret commands like G04 in ISO code.
AddLengthCompensation	The method adds to the tool path a tool length compensation command with the specified parameters. Input parameters: Mode (TNCLengthCompensationMode) - mode: lcOff - turn off the tool length compensation; lcOn - turn on the positive tool length compensation; lcOnNegative - turn on the negative tool length compensation. OffsetNumber (integer) - length offset number.	To interpret commands like G43, G44, G49, H in ISO code.
AddLengthCompensationValue	The method adds to the tool path a tool length compensation command with the specified parameters.	

Method	Method description	Description of use
	Input parameters: Mode, OffsetNumber (see AddLengthCompensation); Value (real number) - tool length offset value. Attention! In the current version the function works like AddLengthCompensation, without the possibility to set the value.	
AddLengthCompensationWithSpecifiedAxis	The method adds to the tool path a tool length compensation command with the specified parameters. Input parameters: Mode (see AddLengthCompensation); SpecifiedAxis (integer) - number of the axis along which the length compensation is made; OffsetNumber (see AddLengthCompensation). Attention! In the current version the function works like AddLengthCompensation, without the possibility to select the axis.	To interpret commands like G43 (together with G17-G19) in ISO code.
AddRadiusCompensation	The method adds to the tool path a tool radius compensation command with the specified parameters. Input parameters: Mode (TNCRadiusCompensationMode) - mode:	To interpret commands like G41, G42, D in ISO code.

Method	Method description	Description of use
	rcOff - turn off the tool radius compensation; rcLeft - turn on the tool radius compensation on the left; rcRight - turn on the tool radius compensation on the right. OffsetNumber (integer) - radius offset number.	
AddRadiusCompensationValue	The method adds to the tool path a tool radius compensation command with the specified parameters. Input parameters: Mode, OffsetNumber (see AddRadiusCompensation); Value, Value2 (real number) - tool radius offset values. Attention! In the current version the function works like AddRadiusCompensation, without the possibility to set the value.	
CutTo3d	The method adds to the tool path a command of linear tool movement to the specified point. Input parameter: P (TNC3DPoint) - coordinates of the point.	To interpret commands like G00/G01 X Y Z (CutTo3d) or G00/G01 X Y Z A B C (CutTo5d, CutTo6d) in ISO code.
CutTo5d	The method adds to the tool path a command of linear tool movement to the specified point with the specified tool direction.	

Method	Method description	Description of use
	Input parameters: P (TNC3DPoint) - coordinates of the point; N (TNC3DPoint) - tool normal vector in the part coordinate system.	
CutTo6d	The method adds to the tool path a command of linear tool movement to the specified position with the specified tool orientation. Input parameter: P (TNC3DMatrix) - tool position and orientation in the part coordinate system. TNC3DMatrix - homogeneous 4×4 matrix with the fields vX, A, vY, B, vZ, C, vT, D: vX, vY, vZ (TNC3DPoint) - directions of the tool coordinate system axes (vZ - tool axis); vT (TNC3DPoint) - tool position (X, Y, Z); A, B, C (real number) - 0; D (real number) - 1.	
ArcTo2d	The method adds to the tool path a command of tool movement along a circular arc with the specified parameters. Input parameters: Pe (TNC3DPoint) - arc end point (X, Y, Z); Pc (TNC3DPoint) - arc center (X, Y, Z); Plane (TNCPlaneType, see SetCurrentPlane) - arc plane; R (real number) - arc radius	To interpret commands like G02/G03 X Y Z I J K R in ISO code.

Method	Method description	Description of use
	(greater than zero for counterclockwise rotation, less than zero for clockwise rotation); CanBeFull (boolean) - the arc can be a full circle.	
ArcTo3d	The method adds to the tool path a command of tool movement along a circular arc with the specified parameters. Input parameters: Pe (TNC3DPoint) - arc end point (X, Y, Z); Pc (TNC3DPoint) - arc center (X, Y, Z); Nc (TNC3DPoint) - normal vector of the arc plane; R (real number) - arc radius (greater than zero for counterclockwise rotation, less than zero for clockwise rotation); CanBeFull (boolean) - the arc can be a full circle.	
ArcTo5d	The method adds to the tool path a command of tool movement along a circular arc with the specified parameters. Input parameters: Pe (TNC3DPoint) - arc end point (X, Y, Z); Ne (TNC3DPoint) - tool normal vector at the end point in the part coordinate system; Pc (TNC3DPoint) - arc center (X, Y, Z); Nc (TNC3DPoint) - normal vector of the arc plane; R (real number) - arc radius (greater than zero for counterclockwise rotation, less than zero for	

Method	Method description	Description of use
	clockwise rotation); CanBeFull (boolean) - the arc can be a full circle.	
ArcTo6d	The method adds to the tool path a command of tool movement along a circular arc with the specified parameters. Input parameters: Pe (TNC3DMatrix) - tool position and orientation at the arc end point in the part coordinate system (for the type, see CutTo6d); Pc (TNC3DPoint) - arc center (X, Y, Z); Nc (TNC3DPoint) - normal vector of the arc plane; R (real number) - arc radius (greater than zero for counterclockwise rotation, less than zero for clockwise rotation); CanBeFull (boolean) - the arc can be a full circle.	
GetLastPosition	The function returns the coordinates of the current tool position and the tool normal vector relative to the part coordinate system. No input parameters. Output parameters: P (TNC3DPoint) - current tool position; N (TNC3DPoint) - tool normal vector. Note: the tool position is not defined before the first	Used to get the current tool position.

Method	Method description	Description of use
	movement, so check WasMovement before the call.	
WasMovement	The function returns the flag of whether the first movement has been made (boolean). True - the first movement has already been made, the current tool position can be obtained with GetLastPosition. False - no movements have been made yet. During simulation by the NC program (including ENCY Tuner) the function always returns True.	See the note to GetLastPosition.
CreateLinearMotionFormer	The function returns the interface of the linear movement generator (INCT_GeomLinearMotionFormer). Using the generator in the interpreter simplifies forming a linear tool path. Working with the generator is described in the article Linear movements. No input parameters.	To interpret commands like G00/G01 X Y Z in ISO code.
CreateCircularMotionFormer	The function returns the interface of the arc movement generator (INCT_CircularMotionFormer). Using the generator in the interpreter simplifies forming a tool path along an arc. Working with the generator is described	To interpret commands like G02/G03 X Y Z I J K R in ISO code.

Method	Method description	Description of use
	in the article Circular movements. No input parameters.	
CreateDrillingCycleFormer	Depending on the specified parameter, the function returns the interface of the generator of one of the drilling cycles. Working with the generators is described in the articles Drilling cycle, Drilling cycle with shift, Drilling cycle with chip breaking. Input parameter: CycleType (TNCDrillingCycleType) - cycle type. dctSimple - simple drilling cycle; the returned generator supports only one interface: INCT_DrillingCycleFormer. dctWithShift - drilling cycle with shift; the returned generator supports two interfaces: INCT_DrillingCycleFormer, INCT_DrillingCycleWithShiftFormer. dctPeck - drilling cycle with chip breaking; the returned generator supports two interfaces: INCT_DrillingCycleFormer, INCT_PeckDrillingCycleFormer.	To interpret commands like G73, G74, G76, G80-89 for the Fanuc controller.
CreateLatheCycleFormer	Depending on the specified parameter, the function returns the interface of the generator of one of the turning cycles. Working with the generators is	To interpret commands like CYCLE93, CYCLE95, CYCLE97 for the Sinumerik controller.

Method	Method description	Description of use
	described in the articles Turning grooving cycle, Turning threading cycle, Turning stock removal cycle. Input parameter: CycleType (TNCLatheCycleType) - cycle type. lctGrooving - turning grooving cycle; the returned generator supports one interface: INCT_LatheGroovingCycleFormer . lctThreadCutting - turning thread cutting cycle; the returned generator supports one interface: INCT_LatheThreadCuttingCycleFormer. lctStockRemoval - turning cycle of rough and finish contour turning; the returned generator supports one interface: INCT_LatheStockRemovalCycleFormer.	

Equipment parameters

INCT_Equipment - a CAM system interface that provides access to some equipment parameters.

Description of the interface methods:

Method	Method description	Description of use
GetAxisIndex	The function returns the index of a machine axis by its address (integer). Input parameter: Addr (string) - axis address ('X', 'Y', 'Z', etc., as specified in the machine kinematic scheme).	To control the tool movement (with INCT_CLData.AddMotion or CreateLinearMotionFormer), the axis indexes returned by GetAxisIndex must be specified.
GetAxisIndexByNum	The function returns the index of a main or additional robot axis by its number (integer). Input parameters: AxisNum (integer) - robot axis number (1-6); Ext (boolean): False - robot axis, True - additional axis.	If a robot axis is set by a number (not by an address), GetAxisIndexByNum can be used to get the axis index. As a rule, the axes of Motoman robots are obtained this way.
GetXYZindex	The function returns the index of the axis linked to the movement along the X, Y or Z coordinate by the specified parameter (see INCT_CLData.AddCircle). Input parameter: Ort (integer): 1 - X; 2 - Y; 3 - Z.	To get the indexes of the axes linked to X, Y, Z of the current tool.
CurrentLatheToolWidth	The function returns the width of the current turning tool (real number).	Used to set the groove width step in INCT_LatheGroovingCycleFormer .

Method	Method description	Description of use
GetSpindlesCount	The function returns the number of spindles of the current machine (integer).	To execute commands that control a machine spindle by its number. For example, SPOS[n] = Value for the Sinumerik controller: spindle numbers in the NC program start from 1 and spindle indexes start from 0, so the axis of spindle n is GetAxisIndexBySpindleIndex(n - 1).
GetAxisIndexBySpindleIndex	The function returns the spindle axis number by the spindle index (integer). Input parameter: SpindleIndex (integer) - spindle index, possible values from 0 to GetSpindlesCount - 1.	
GetOperationGroup	The function returns the type of the operation in the CAM system (TNCOperationGroup). The TNCOperationGroup type: ogUnknown - unknown; ogMill - milling; ogLathe - turning; ogAuxiliary - auxiliary; ogWireEDM - wire EDM; ogWelding - various kinds of welding; ogAdditive - various additive operations such as cladding; ogWaterJet - water jet cutting; ogSawing - sawing.	Use it in interpreters only as a last resort. The function was added to compensate for the position of a turning tool in a turning operation of the CAM system where the offset value cannot be set.

INCT_Equipment2 - a CAM system interface.

Description of the interface methods:

Method	Method description	Description of use
GetCurrentToolRadius	The function returns the radius of the current tool (real number).	Can be used to calculate the overshoots for CYCLE90 in Sinumerik.

INCT_Equipment_Channels - a CAM system interface that provides access to the machine channels. The interface is requested from the INCT_Equipment object; check the result of the request: in earlier versions of the CAM system the object may not support it.

Description of the interface methods:

Method	Method description	Description of use
Get_Count	The function returns the number of machine channels (integer).	
Get_CurrentIndex	The function returns the index of the channel the current operation belongs to (integer). Numbering starts from 0.	To interpret multichannel programs: allows determining which channel's program the interpreter is interpreting.
Get_Channel	The function returns the channel interface (INCT_Channel) by index. Input parameter: Index (integer) - channel index, possible values from 0 to Get_Count - 1.	

INCT_Channel - a CAM system interface that provides access to the machine channel parameters.

Description of the interface methods:

Method	Method description	Description of use
Get_Name	The function returns the channel name specified in the machine kinematic scheme (string).	

Managing NC program text analysis

INCT_Analyzer - a CAM system interface that provides access to the settings of the NC program text analysis. The analyzer settings affect the algorithm of determining registers and values in the NC program text.

Description of the interface methods:

Method	Method description	Description of use
BeginUpdate	Signals the start of an INCT_Analyzer update. No input or output parameters.	Call BeginUpdate to start changing the INCT_Analyzer settings. When the changes are finished, call EndUpdate to apply all of them. Each BeginUpdate call must be followed by a corresponding EndUpdate call. If only one INCT_Analyzer setting is changed at a time, BeginUpdate and EndUpdate are optional.
EndUpdate	Signals the end of the update. All changes made between the start and the end of the INCT_Analyzer update are applied. No input or output parameters.	
Get_CanUseEqualToAssignRegister	The function returns the value of the setting 'The = character can be used to set a register value' (boolean). True: the NC program text analyzer uses the '=' character to set a register value. False: when a register value is determined, the '=' character is considered an invalid token. No input parameters.	If the setting is on (True), the following combination of registers and values in the NC program text is considered correct: X=10.0 Y10.0 Z-1 A=-1 B+3.5 C=-3.5 If the setting is off (False), the following way of setting register values is considered correct: Y10.0 Z-1 B+3.5
Set_CanUseEqualToAssignRegister	The method sets the value of the setting 'The = character can be used to set a register value'.	

Method	Method description	Description of use
	Input parameter: Value (boolean), see the parameter description in Get_CanUseEqualToAssignRegister. No output parameters.	
Get_RegistersContainSpaces	Deprecated The function returns the value of the setting 'A register can contain spaces' (boolean). True: when a register is determined, the NC program text analyzer takes into account that a register address can contain spaces. False: when a register is determined, a space is considered only a token separator. No input parameters.	Deprecated If the setting is on (True), the following combination of registers and values in the NC program text is considered correct: TOOL CALL 1 CYCL CALL provided that the registers with the addresses 'TOOL CALL' and 'CYCL CALL' are defined in the list. The register with the address 'TOOL CALL' gets the value 1. The register with the address 'CYCL CALL' must be defined as a register without a value (vatOnlyAddr). If the setting is off, the analyzer considers 'TOOL', 'CALL', 'CYCL', 'CALL' separate tokens.
Set_RegistersContainSpaces	Deprecated The method sets the value of the setting 'A register can contain spaces'. Input parameter: Value (boolean), see the parameter description in Get_RegistersContainSpaces. No output parameters.	
Get_RegistersCanEndWithPlusOrMinus	Deprecated	Deprecated

Method	Method description	Description of use
	The function returns the value of the setting 'A register can end with a plus or minus' (boolean). True: when a register is determined, the NC program text analyzer takes into account that a register address can end with a plus or minus character. False: when a register is determined, the plus and minus characters are considered separate tokens that are not part of the register address. No input parameters.	If the setting is on (True), the following combination of registers and values in the NC program text: DL+0 DR+0 is recognized as the register 'DL+' with the value 0 and the register 'DR+' with the value 0, provided that the registers with the addresses 'DL+', 'DR+' are defined in the list of addresses. If the setting is off, the analyzer considers 'DL', '+0', 'DR', '+0' separate tokens.
Set_RegistersCanEndWithPlusOrMinus	Deprecated The method sets the value of the setting 'A register can end with a plus or minus'. Input parameter: Value (boolean), see the parameter description in Get_RegistersCanEndWithPlusOrMinus. No output parameters.	
Get_RegistersCanEndWithNumber	Deprecated The function returns the value of the setting 'A register can end with a number' (boolean). True: when a register is determined, the NC program text analyzer takes into account that a register address can end with an integer. False: when a register is	Deprecated If the setting is on (True), the following combination of registers and values in the NC program text: DR2+0 R0

Method	Method description	Description of use
	determined, the number characters are considered separate tokens that are not part of the register address. No input parameters.	is recognized as the register 'DR2' with the value 0 and the register 'R0' without a value, provided that the registers with the addresses 'DR2', 'R0' are defined in the list of addresses. The register with the address 'R0' must be defined as a register without a value (vatOnlyAddr). If the setting is off, the analyzer considers 'DR', '2', '+0', 'R', '0' separate tokens.
Set_RegistersCanEndWithNumber	Deprecated The method sets the value of the setting 'A register can end with a number'. Input parameter: Value (boolean), see the parameter description in Get_RegistersCanEndWithNumber. No output parameters.	
Get_LineBeginWithNumber	The function returns the value of the setting 'An NC program line can begin with a number' (boolean). True: when an NC program text line is analyzed, the first token, if it is a number, is ignored. False: when an NC program text line is analyzed, the first number is considered a token. No input parameters.	If the setting is on (True), the first token of an NC program line, if it is a number, is skipped.
Set_LineBeginWithNumber	The method sets the value of the setting 'An NC program line can begin with a number'.	

Method	Method description	Description of use
	Input parameter: Value (boolean), see the parameter description in Get_LineBeginWithNumber. No output parameters.	
Get_Comments	The function returns the interface for managing the settings of determining comments in the NC program text (INCT_Comments interface). No input parameters.	If the following combinations of opening and closing characters are added to the comment settings: '!', '' '(*', '*)' then the NC program text lines ! X10 Y20 Z30.5 skip A+1 (* test G01 X100 F100 *) are considered comments (skipped).
Get_Brackets	The function returns the interface for managing the settings of determining brackets in the NC program text (INCT_Brackets interface). No input parameters.	Added for determining brackets in expressions. Brackets are recognized as separate tokens when the CanUseEqualToAssignRegister setting is on, but they are not used when determining the register value.

Method	Method description	Description of use
Get_Quotes	The function returns the interface for managing the settings of determining strings in the NC program text (INCT_Quotes interface). No input parameters.	If the character '"' is added to the string settings and the setting 'The = character can be used to set a register value' is on, then the NC program text line T="1" is recognized by the analyzer as the register T with the value 1. The string value "1" is converted to the number 1.
Get_UnionLineChars	The function returns the interface for managing the settings of joining the NC program text lines (INCT_UnionLineChars interface). No input parameters.	If the character "~" is added to the line joining settings, then the lines 1 CYCL DEF 200 ~ Q200=+1~ Q201=-30.645~ Q206=+200~ are translated and interpreted as one line: 1 CYCL DEF 200 Q200=+1 Q201=-30.645 Q206=+200
Get_EndOfLineChars	The function returns the interface for managing the settings of splitting the NC program text into lines (INCT_EndOfLineChars interface). No input parameters.	If the character ";" is added to the line splitting settings, then the NC program lines MoveL [10,10,10,10];MoveL [20,20,20,20]; MoveC [10,10,10,10], [10,10,10,10];

Method	Method description	Description of use
		are passed to the interpreter for translation according to the specified separator: MoveL [10,10,10,10]; MoveL [20,20,20,20]; MoveC [10,10,10,10], [10,10,10,10];
Get_ExcludeEOLChars	The function returns the value of the setting 'Exclude end-of-line characters from the NC program text' (boolean). True: when lines are split according to the settings set in INCT_EndOfLineChars, the end-of-line character is excluded from the text and is not passed to the interpreter. False: the end-of-line character is left in the text and passed to the interpreter. No input parameters.	If the setting is on (True) and the specified separator is ';' (INCT_EndOfLineChars), then the following NC program text: MoveL [10,10,10,10];MoveL [20,20,20,20]; MoveC [10,10,10,10], [10,10,10,10]; is passed to the interpreter as follows: MoveL [10,10,10,10] MoveL [20,20,20,20] MoveC [10,10,10,10], [10,10,10,10] that is, the line separator character is excluded. If the setting is off, the text is passed as follows: MoveL [10,10,10,10]; MoveL [20,20,20,20];

Method	Method description	Description of use
Set_ExcludeEOLChars	The method sets the value of the setting 'Exclude end-of-line characters from the NC program text'. Input parameter: Value (boolean), see the parameter description in Get_ExcludeEOLChars. No output parameters.	MoveC [10,10,10,10], [10,10,10,10];

INCT_Analyzer2 - a CAM system interface that provides access to the settings of the NC program text analysis.

Description of the interface methods:

Method	Method description	Description of use
Get_RegistersCaseSensitivity	The function returns the value of the setting "Case sensitivity". True: when the NC program text is analyzed, the register addresses are determined case-sensitively (registers with the addresses "A" and "a" are different). False: when the NC program text is analyzed, the register addresses are determined case-insensitively (registers with the addresses "A" and "a" are the same). No input parameters.	If the "Case sensitivity" setting is on (True) and all registers of the interpreter are set in capital letters, the registers written in lowercase letters are skipped. For the following NC program text: G0 X0 Y0 Z0 x10 y10 z10 all registers of the first line are recognized and added to the list of objects for interpretation INCT_BlockExecArray. All registers of the second line are skipped (not recognized).

Method	Method description	Description of use
Set_RegistersCaseSensitivity	The method sets the value of the setting "Case sensitivity". Input parameter: Value (boolean), see the parameter description in Get_RegistersCaseSensitivity. No output parameters.	

INCT_Comments - a CAM system interface that implements the list of settings for managing comments. The settings consist of pairs of strings: an opening character set and a closing character set. When an NC program block is analyzed, the characters between any pair of opening and closing character sets are considered a comment and skipped. The closing character set can be an empty string; in this case, the rest of the line after the opening character set is considered a comment.

Description of the interface methods:

Method	Method description	Description of use
Add	The function adds a pair of settings to the list and returns the index of the added pair in the list (integer). Input parameters: Open (string) - opening comment character set; Close (string) - closing comment character set.	To add comment settings when setting up the analyzer.
Get	The method returns the comment settings by index. Input parameter: Idx (integer) - index of the pair of settings. Output parameters: Open (string) - opening comment character set; Close (string) - closing comment character set.	To get the settings.
Delete	The method deletes a pair of settings by the specified	To remove comment settings when reconfiguring the analyzer.

Method	Method description	Description of use
	parameter. Input parameter: Idx (integer) - index of the pair of settings.	
Get_Count	The function returns the number of pairs of settings (integer).	To get the number of settings.

INCT_Brackets - a CAM system interface that implements the list of settings for determining brackets in the NC program text. The settings consist of pairs of characters: opening and closing. Brackets are recognized as separate tokens when the CanUseEqualToAssignRegister setting is on, but they are not used when determining the register value; the settings are added for future use.

Description of the interface methods:

Method	Method description	Description of use
Add	The function adds a pair of settings to the list and returns the index of the added pair in the list (integer). Input parameters: Open (character) - opening bracket character; Close (character) - closing bracket character.	To add bracket settings when setting up the analyzer.
Get	The method returns the bracket settings by index. Input parameter: Idx (integer) - index of the pair of settings. Output parameters: Open (character) - opening bracket character; Close (character) - closing bracket character.	To get the settings.
Delete	The method deletes a pair of settings by the specified parameter. Input parameter: Idx (integer) - index of the pair of settings.	To remove bracket settings when reconfiguring the analyzer.

Method	Method description	Description of use
Get_Count	The function returns the number of pairs of settings (integer).	To get the number of settings.

INCT_Quotes - a CAM system interface that implements the list of settings for determining strings. Each setting consists of the character with which a string value begins and ends. When an NC program block is analyzed, the characters between any pair of string characters are considered a string value.

Description of the interface methods:

Method	Method description	Description of use
Add	The function adds a setting to the list and returns the index of the added setting in the list (integer). Input parameter: Sym (character) - character with which a string value begins and ends.	To add string value settings when setting up the analyzer.
Get	The method returns a setting by index. Input parameter: Idx (integer) - setting index. Output parameters: Sym (character) - character with which a string value begins and ends.	To get the settings.
Delete	The method deletes a setting by the specified parameter. Input parameter: Idx (integer) - setting index.	To remove string value settings when reconfiguring the analyzer.
Get_Count	The function returns the number of settings (integer).	To get the number of settings.

INCT_UnionLineChars - a CAM system interface that implements the list of settings for joining the NC program text lines. Each setting consists of a line joining character. If an NC program block ends with any character from the list, the line is joined with the next one without the joining character.

Description of the interface methods:

Method	Method description	Description of use
Add	The function adds a setting to the list and returns the index of the added setting in the list (integer). Input parameter: Sym (character) - line joining character.	To add line joining character settings when setting up the analyzer.
Get	The method returns a setting by index. Input parameter: Idx (integer) - setting index. Output parameters: Sym (character) - line joining character.	To get the settings.
Delete	The method deletes a setting by the specified parameter. Input parameter: Idx (integer) - setting index.	To remove line joining character settings when reconfiguring the analyzer.
Get_Count	The function returns the number of settings (integer).	To get the number of settings.

INCT_EndOfLineChars - a CAM system interface that implements the list of settings for splitting the NC program text into lines. Each setting consists of a separator string. The NC program text is passed to the interpreter line by line, where each line is the content between separators.

Description of the interface methods:

Method	Method description	Description of use
Add	The function adds a setting to the list and returns the index of the added setting in the list (integer). Input parameter: Str (string) - separator.	To add line separator settings when setting up the analyzer.

Method	Method description	Description of use
Get	The method returns a setting by index. Input parameter: Idx (integer) - setting index. Output parameters: Str (string) - separator.	To get the settings.
Delete	The method deletes a setting by the specified parameter. Input parameter: Idx (integer) - setting index.	To remove line separator settings when reconfiguring the analyzer.
Get_Count	The function returns the number of settings (integer).	To get the number of settings.

Simplifying tool path construction

This section describes the CAM system interfaces that simplify the construction of the tool path.

Linear movements

INCT_LinearMotionFormer - a CAM system interface that simplifies forming linear movements. The interface is returned by the function INCT_CLData.CreateLinearMotionFormer.

Purpose:

Linear movements are formed from a block of the NC program as follows. The NC program text is translated, that is, split into registers with values. The registers with values are added to the list of objects for interpretation. The objects of the list are interpreted. When the objects are interpreted, a multi-axis movement command (INCT_CLData.OpenMultiMotion, AddMotion, CloseMultiMotion) is added to the tool path (INCT_CLData) according to the changed values of the registers linked to the machine axes of the kinematic scheme.

In more detail, the process consists of the following actions.

Before the NC program is interpreted:

1. *Get the list of registers linked to the machine axes in the kinematic scheme.*
2. *Store the list of registers with the axis indexes.*

Before a block is interpreted:

1. *Get the current tool position coordinates for all axes linked to the machine axes in the kinematic scheme.*

When registers are interpreted:

1. *Get the new values of the block registers linked to the machine axes in the kinematic scheme.*
2. *Set the new values for all changed block registers linked to the machine axes in the kinematic scheme.*

After the block is interpreted, or during the interpretation but after all registers are interpreted:

1. *Compare the new axis coordinates with the old ones to find changes.*
2. *If there was a change:*
3. *Call OpenMultiMotion.*
4. *Check each axis value for a change and, if the tool position along the axis has changed, call AddMotion.*
5. *Call CloseMultiMotion.*

To execute commands that require movements in the physical machine coordinates, the movement type must also be switched to gtmPhysic.

The actions from the list above that are *in italics* are implemented in the linear movement generator (INCT_LinearMotionFormer). Using the generator makes writing an interpreter easier.

We recommend creating one linear movement generator for the whole cycle of interpreting the NC program. Create it when the interpreter is initialized (INCT_Interpreter.Initialize).

Description of the interface methods:

Method	Method description	Description of use
SetGotoMode	The method sets the type of the formed movements. Input parameter: Mode (TNCGotoMode) - movement type, see the description of OpenMultiMotion. If the movement type is not set by the method, it is gtmTooltip. No output parameters.	The method can be used when interpreting G53 in ISO code (set the movement type to gtmPhysic, then return it to gtmTooltip).
Get_ClearGotoMode	The function returns the value of the ClearGotoMode parameter (boolean). No input parameters.	The ClearGotoMode parameter controls the automatic reset of the movement type (see SetGotoMode) to gtmTooltip. The movement type is reset in InitBlock. As a rule, the G53 command acts only in the current block, so it makes sense to set ClearGotoMode to True. Then, after a block with G53 is interpreted (where the movement type is set to gtmPhysic), the movement type returns to gtmTooltip by itself at the next InitBlock call.
Set_ClearGotoMode	The method sets the value of the ClearGotoMode parameter. Input parameter: Value (boolean) - value of the ClearGotoMode parameter. No output parameters.	

Method	Method description	Description of use
InitAxisIndex	The method adds an axis index to the list. Input parameters: Index (integer) - index of the axis added to the list. No output parameters.	Used to add the axes linked to the machine axes in the kinematic scheme. As a rule, it is called right after getting the interface (function INCT_CLData.CreateLinearMotionFormer) in the implementation of the interpreter method INCT_Interpreter.Initialize. The indexes of the registers linked to the machine axes can be obtained with the interfaces INCT_SysState, INCT_Equipment.
ClearAxesIndexes	The method clears the list of axis indexes.	Used when the tool is changed if the names of the axes controlling the tool coincide.
InitBlock	The method stores in the internal list the current tool coordinates for all axes from the list formed with InitAxisIndex. If the ClearGotoMode parameter is True, the movement type set in SetGotoMode is set to gtmTooltip. No input or output parameters.	Must be used before interpreting the objects linked to registers. It is recommended to use it in the implementation of the INCT_Interpreter.BeforeInterprete method.
SetAxisValue	The method stores in the internal list the new tool coordinate for the specified axis. Input parameters: Abs (boolean) - type of the new coordinate value, True - absolute value, False - relative value (relative to	Used to store the new value in the internal list to form a multi-axis movement command later. Used when interpreting in INCT_Interpreter.Interprete.

Method	Method description	Description of use
	the current coordinate, see InitBlock); Index (integer) - axis index; Value (real number) - new value. No output parameters.	
MakeWorkpath	The method compares the list of current coordinates (stored in InitBlock) with the new tool coordinates (stored in SetAxisValue). If the coordinates have changed, a multi-axis movement command along the changed axis coordinates is formed in the tool path (INCT_CLData.OpenMultiMotion, AddMotion, CloseMultiMotion).	To form a multi-axis movement command. Used when interpreting in INCT_Interpreter.Interprete after all registers are interpreted (add it to the end of the list of objects for interpretation), or in INCT_Interpreter.AfterInterprete.

INCT_GeomLinearMotionFormer - a CAM system interface that simplifies forming linear movements. The interface is returned by the function INCT_GeomCLData.CreateLinearMotionFormer.

Its purpose is the same as that of the INCT_LinearMotionFormer interface, but for movements in geometric axes. The movements are formed by the CutTo3d command. Therefore, to form the first movement, the coordinates of all three axes must be set.

Description of the interface methods:

Method	Method description	Description of use
ClearSpecifiedAxes	The method clears the list of axes set for the first movement. No input or output parameters.	Must be used at the beginning of each operation. It is recommended to use it in the implementation of the INCT_Interpreter.NextOperation method.
InitBlock	The method stores in the internal list the current values of	Must be used before interpreting the objects linked to registers. It is recommended to

Method	Method description	Description of use
	the 3 tool coordinates. No input or output parameters.	use it in the implementation of the INCT_Interpreter.BeforeInterprete method.
SetAxisValue	The method stores in the internal list the new tool coordinate for the specified axis. Input parameters: Abs (boolean) - type of the new coordinate value, True - absolute value, False - relative value (relative to the current coordinate, see InitBlock); Index (integer) - axis index (1-3); Value (real number) - new value. No output parameters.	Used to store the new value in the internal list to form a movement command later. Used when interpreting in INCT_Interpreter.Interprete.
MakeWorkpath	The method compares the list of current coordinates (stored in InitBlock) with the new tool coordinates (stored in SetAxisValue). If the coordinates have changed and the coordinates of all 3 axes are set, a movement command (INCT_GeomCLData.CutTo3d) is formed in the tool path.	To form a movement command. Used when interpreting in INCT_Interpreter.Interprete after all registers are interpreted (add it to the end of the list of objects for interpretation), or in INCT_Interpreter.AfterInterprete.

Circular movements

INCT_CircularMotionFormer - a CAM system interface that simplifies forming movements along an arc. The interface is returned by the functions INCT_CLData.CreateCircularMotionFormer, INCT_GeomCLData.CreateCircularMotionFormer.

Purpose:

Arc movements are formed from a block of the NC program as follows. The NC program text is translated, that is, split into registers with values. The registers with values are added to the list of objects for interpretation. The objects of the list are interpreted. When the objects are interpreted, an arc movement command (INCT_CLData.AddCircle/INCT_GeomCLData.ArcTo2d) is added to the tool path (INCT_CLData/INCT_GeomCLData) according to the changed values of the registers linked to the machine axes of the kinematic scheme.

In more detail, the process consists of the following actions.

Before a block is interpreted:

1. *Get the indexes of the X, Y, Z axes for the current tool or working object.*
2. *Store the axis indexes.*
3. *Get the current tool position coordinates for the stored axes, separately for the arc center point.*

When registers are interpreted:

1. Get the new values of the block registers linked to the arc center point, end point, radius or angle, and direction.
2. *Set the new values for all changed arc parameters.*

After the block is interpreted, or during the interpretation but after all registers are interpreted:

1. *Calculate the missing parameters for INCT_CLData.AddCircle/INCT_GeomCLData.ArcTo2d.*
2. *Call INCT_CLData.AddCircle/INCT_GeomCLData.ArcTo2d.*

The actions from the list above that are *in italics* are implemented in the arc movement generator (INCT_CircularMotionFormer). Using the generator makes writing an interpreter easier.

We recommend creating one arc movement generator for the whole cycle of interpreting the NC program. Create it when the interpreter is initialized (INCT_Interpreter.Initialize).

Description of the interface methods:

Method	Method description	Description of use
InitBlock	The method stores in the internal list the axis indexes and the current X, Y, Z coordinates of the current tool or working object. No input or output parameters.	Must be used before interpreting the objects linked to registers. It is recommended to use it in the implementation of the INCT_Interpreter.BeforeInterprete method.
InitCenterPoint	The method stores the current X, Y, Z coordinates of the current tool or working object as the center point. No input or output parameters.	Must be used before interpreting the objects linked to registers. It is recommended to use it in the implementation of the INCT_Interpreter.BeforeInterprete or INCT_Interpreter.Interprete method (if the circle center point is not set in the current block).
SetConstructMode	The method sets the arc construction mode. Input parameter: Mode (TNCConstructArcMode) - arc construction mode: caByCircleCenter - by the direction, end point and center of the arc; caByRadius - by the direction, end point and radius of the arc. No output parameters.	Must be used when interpreting registers in the implementation of the INCT_Interpreter.Interprete method.
SetMode	The method sets the arc direction. Input parameter: Mode (TNCCircularMode) - arc direction:	

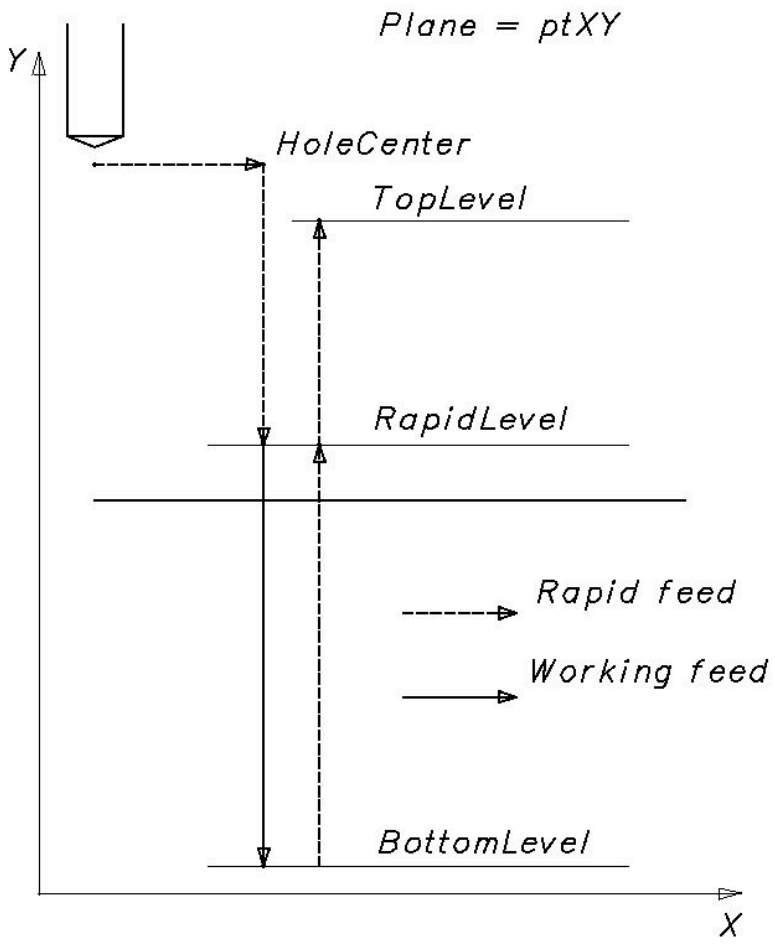
Method	Method description	Description of use
	cmCW - clockwise; cmCCW - counterclockwise. No output parameters.	
SetEndPoint	The method sets the coordinate of one axis of the end point. Input parameters: Abs (boolean) - type of the end point value, True - absolute value, False - relative value (relative to the current coordinate, see InitBlock); AxisType (TNCAxisType) - axis: axX - X axis, axY - Y axis, axZ - Z axis; Value (real number) - coordinate value. No output parameters.	
SetEndPointByAngle	The method sets the arc end point by the specified angle relative to the current point. Input parameters: Plane (TNCPlaneType) - current plane: ptXY - XY plane, ptYZ - YZ plane, ptZX - ZX plane; Angle (real number) - arc angle. No output parameters. Important! When a full circle is constructed (the arc angle is 360 degrees), the end point of the path must be forced to equal the start (current) point. Otherwise, because of a calculation error of hundredths	

Method	Method description	Description of use
	or thousandths in the end point coordinates, the next movement command may become incorrect. This is especially important for commands in which the start point is used to calculate auxiliary parameters of the path (for example, when the tool moves along a circular arc set by the coordinates of its center, the end point and the movement direction).	
SetCenterPoint	The method sets the coordinate of one axis of the arc center point. Input parameters: Abs (boolean) - type of the center point value, True - absolute value, False - relative value (relative to the current coordinate, see InitCenterPoint); AxisType (TNCAxisType) - axis: axX - X axis, axY - Y axis, axZ - Z axis; Value (real number) - coordinate value. No output parameters.	
SetRadius	The method sets the arc radius. Input parameter: Value (real number) - radius value. No output parameters.	

Method	Method description	Description of use
MakeWorkpath	The method calculates the arc end point if an angle has been set (SetEndPointByAngle). Depending on the arc construction mode (SetConstructMode), it calculates the missing parameters for AddCircle. It adds to the tool path an arc movement command (INCT_CLData.AddCircle/INCT_GeomCLData.ArcTo2d) with the calculated parameters. Input parameter: LastPlane (TNCPlaneType) - arc plane. No output parameters.	To form an arc movement command. Used when interpreting in INCT_Interpreter.Interprete after all registers are interpreted (add it to the end of the list of objects for interpretation), or in INCT_Interpreter.AfterInterprete.

Drilling cycle

INCT_DrillingCycleFormer - a CAM system interface that simplifies forming the movements of drilling cycles. The object implementing this interface is returned by the function INCT_CLData.CreateDrillingCycleFormer, INCT_GeomCLData.CreateDrillingCycleFormer when it is called with any of the parameters dctSimple, dctWithShift, dctPeck.



Purpose:

The movements in the various drilling cycles generally follow the same tool path on different controllers (Fanuc: G74, G81, G82, G84, G85, G86, G88, G89; Sinumerik: CYCLE81, CYCLE82, CYCLE85, CYCLE87, CYCLE88, CYCLE89; Heidenhain: CYCL DEF 201, CYCL DEF 206, CYCL DEF 207, etc.):

1. Positioning the tool above the center of the hole or of the next hole at rapid traverse.
2. Moving the tool to the rapid level at rapid traverse.
3. Moving the tool to the hole bottom level at working feed.
4. Returning to the rapid level at rapid traverse.
5. Returning to the retract level at rapid traverse.
6. If the number of repetitions is not reached, returning to step 1.

The generator forms the tool path from the specified parameters: initial position, rapid level, hole bottom level, number of repetitions, etc. It also allows adding to any section of the tool path the features required by a particular cycle (dwell, oriented spindle stop, tapping, etc.), changing the feed type or value, or skipping a tool movement prepared by the generator.

Using the generator:

We recommend creating a new drilling cycle generator for each new cycle in a block of the NC program, except when the cycle is repeated in a new block (like CYCL CALL for Heidenhain). Therefore, the interface of the old generator must be released when a new one is created or when the interpreter finishes its work.

To form the tool movement commands along the cycle path, do the following:

1. Get the generator interface (INCT_CLData.CreateDrillingCycleFormer, INCT_GeomCLData.CreateDrillingCycleFormer).
2. Set the indexes of the X, Y, Z axes along which the movements are made (Set_AxisIndexes).
3. Set the current plane (Set_Plane).
4. Set the hole center point (Set_HoleCenter).
5. Set the center point mode (Set_HoleCenterMode).
6. Set the rapid level (Set_RapidLevel).
7. Set the hole bottom level (Set_BottomLevel).
8. Set the retract level (Set_TopLevel).
9. Set the feed (Set_Feedrate) and the feed units (Set_MPM).
10. Set the number of cycle repetitions (Set_Count).
11. Assign a cycle event handler (Set_Events) if the cycle path needs to be changed.
12. Form the tool movement commands along the cycle path (MakeWorkpath).

Description of the interface methods:

Method	Method description
Set_AxisIndexes	The method sets the indexes of the X, Y, Z axes along which the movements are made when the cycle path is formed. Input parameters: Value (TNCDCAxisIndexes) - X (integer) - index of the X axis, Y (integer) - index of the Y axis, Z (integer) - index of the Z axis. No output parameters.

Method	Method description
Get_AxisIndexes	The function returns the indexes of the X, Y, Z axes along which the movements are made when the cycle path is formed (TNCDCAxisIndexes). No input parameters.
Set_HoleCenter	The method sets the coordinates of the hole center point. Input parameters: Value (TNC3DPoint), X, Y, Z - coordinate values for the corresponding axes. No output parameters.
Get_HoleCenter	The function returns the coordinates of the hole center point (TNC3DPoint). No input parameters.
Set_HoleCenterMode	The method sets the center point mode. Input parameter: Value (boolean) - center point mode. True - the coordinates of the hole center point (Set_HoleCenter) are absolute. False - the coordinates of the hole center point are relative to the current tool position. No output parameters.
Get_HoleCenterMode	The function returns the center point mode (boolean). No input parameters.
Set_Plane	The method sets the current plane. Input parameter: Value (TNCPlaneType) - current plane.

Method	Method description
	ptXY - XY plane, the hole is drilled along the Z axis. ptYZ - YZ plane, the hole is drilled along the X axis. ptZX - ZX plane, the hole is drilled along the Y axis. No output parameters.
Get_Plane	The function returns the current plane (TNCPlaneType). No input parameters.
Set_RapidLevel	The method sets the rapid level along the drilling axis. Input parameter: Value (real number) - rapid level. No output parameters.
Get_RapidLevel	The function returns the rapid level along the drilling axis (real number). No input parameters.
Set_BottomLevel	The method sets the hole bottom level along the drilling axis. Input parameter: Value (real number) - hole bottom level. No output parameters.
Get_BottomLevel	The function returns the hole bottom level along the drilling axis (real number). No input parameters.

Method	Method description
Set_TopLevel	The method sets the retract level along the drilling axis. Input parameter: Value (real number) - retract level. No output parameters.
Get_TopLevel	The function returns the retract level along the drilling axis (real number). No input parameters.
Set_Feedrate	The method sets the working feed. Input parameter: Value (real number) - feed value. No output parameters.
Get_Feedrate	The function returns the working feed (real number). No input parameters.
Set_MPM	The method sets the feed units flag. Input parameter: Value (boolean) - feed units flag. True - mm/min. False - mm/rev. No output parameters.
Get_MPM	The function returns the feed units flag (boolean). No input parameters.
Set_Count	The method sets the number of cycle repetitions. Input parameter: Value (integer) - number of cycle repetitions.

Method	Method description
	No output parameters. The cycle works differently depending on the hole center point mode (Set_HoleCenterMode): • if the number of repetitions = 0, only the tool positioning above the hole center is made; • if the number of repetitions = 1, one iteration of the cycle is made; • if the number of repetitions > 1, the cycle is repeated the specified number of times: at the same place if the coordinates of the hole center point are absolute; if they are relative, each iteration is shifted by the value set in Set_HoleCenter from the point where the previous iteration ended.
Get_Count	The function returns the number of cycle repetitions (integer). No input parameters.
Set_Events	The method assigns events that control the cycle behavior. Events allow adding to any section of the tool path the features required by a particular cycle (dwell, oriented spindle stop, tapping, etc.), changing the feed type or value, or skipping a tool movement prepared by the generator. Input parameter: Value (INCT_DrillingCycleEvents) - object implementing the events. See the detailed description below. No output parameters.
Get_Events	The function returns the object implementing the events that is linked to the cycle generator (INCT_DrillingCycleEvents).

Method	Method description
	No input parameters.
MakeWorkpath	The method forms the cycle path from the specified parameters.
Get_NextHole	The function returns the coordinates of the current hole point calculated for the current iteration (TNC3DPoint). The value can be used only inside the events (Set_Events).
Get_RapidPoint	The function returns the coordinates of the rapid level point calculated for the current iteration (TNC3DPoint). The value can be used only inside the events (Set_Events).
Get_TopPoint	The function returns the coordinates of the retract level point calculated for the current iteration (TNC3DPoint). The value can be used only inside the events (Set_Events).
Get_BottomPoint	The function returns the coordinates of the hole bottom point calculated for the current iteration (TNC3DPoint). The value can be used only inside the events (Set_Events).

INCT_DrillingCycleEvents - an interface implemented in the interpreter that allows controlling the behavior of a drilling cycle. The interface is a set of event methods executed during the tool movements inside the cycle path generator. With these methods, the features required by a particular cycle (dwell, oriented spindle stop, tapping, etc.) can be added to any section of the tool path, the feed type or value can be changed, or a tool movement prepared by the generator can be skipped.

If events are assigned to the cycle path generator, every tool movement inside the generator consists of 3 stages:

1. The INCT_DrillingCycleEvents.BeforeMove method is executed. It receives the values set by the cycle: movement type, coordinates of the movement point, feed type.
2. The INCT_DrillingCycleEvents.OnMove method is executed. It receives: movement type, coordinates of the movement point, feed type and the flag of whether the movement has been made. All parameters except the movement type can be changed in the method implementation. Depending on the flag set in OnMove, the tool moves to the specified point with the specified feed, or the movement is not made.

3. The INCT_DrillingCycleEvents.AfterMove method is executed. It receives the values returned by OnMove: movement type, coordinates of the movement point, feed type.

Description of the interface methods:

Method	Method description	Description of use
BeforeMove	The method is executed before each tool movement when the cycle path is formed. Input parameters: MoveType (TNCDrillingCycleMoveType) - movement type, see the description below; Point (TNC3DPoint) - coordinates of the movement point; Feed (TNCFeedType) - feed type. mtMoveToNextHole - movement to the first or the next hole; mtMoveToRapidLevel - movement to the rapid level; mtMoveToBottomLevel - movement to the hole bottom; mtReturnToRapidLevel - return to the rapid level; mtReturnToTopLevel - return to the retract level; mtPeck* - used when a cycle with chip breaking is formed. No output parameters.	If some commands must be added before a movement, add them to the method implementation. For example, to set clockwise spindle rotation before drilling the hole, call INCT_CLData.AddSpindleSpeedOnRPM with the appropriate parameters in the BeforeMove implementation under the condition MoveType = mtMoveToBottomLevel.
OnMove	The method is executed before each tool movement when the cycle path is formed. Input parameters: MoveType (TNCDrillingCycleMoveType) - movement type, see the description above; Input and output parameters: Point (TNC3DPoint) - coordinates of the movement	To override the movement point or the feed, or to skip the movement. For example, to change the feed type from rapid to working when returning after drilling, in the OnMove implementation set the returned Feed parameter to ffWorking under the condition MoveType =

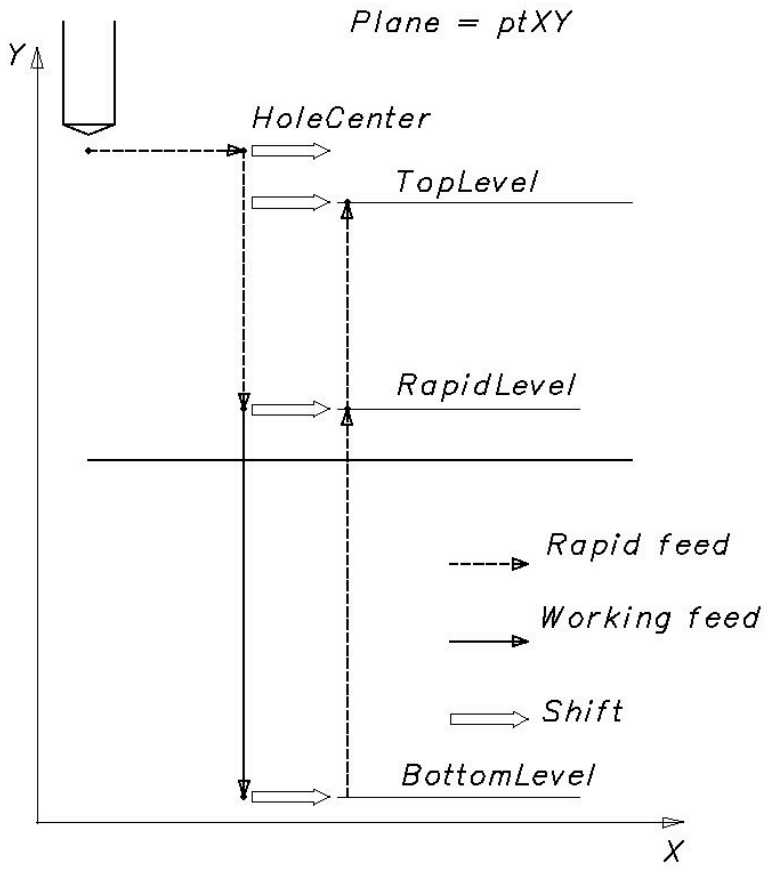
Method	Method description	Description of use
	point; Feed (TNCFeedType) - feed type; Handled (boolean) - flag of whether the movement has been made inside the implementation (True - the movement has been made, False - it has not). If Handled is True after the OnMove call, the movement has been made inside the OnMove implementation, and the cycle generator does not add a movement command to the tool path. If Handled is False after the OnMove call, the generator considers that the movement has not been made inside the OnMove implementation and adds to the tool path a command to move to the point Point with the feed type Feed.	mtReturnToRapidLevel (for the feed type, see OutStandardFeed). To move the tool to a point other than the one calculated by the cycle, change the Point parameter returned by OnMove. If the movement to the point calculated by the cycle is not needed, set the Handled parameter to True inside the OnMove implementation.
AfterMove	The method is executed after each tool movement when the cycle path is formed, even if the movement has been canceled in OnMove. Input parameters: MoveType (TNCDrillingCycleMoveType) - movement type, see the description above; Point (TNC3DPoint) - coordinates of the movement point; Feed (TNCFeedType) - feed type. No output parameters.	If some commands must be added after a movement, add them to the method implementation. For example, to add a dwell after drilling the hole, call INCT_CLData.AddDelay with the appropriate parameters in the AfterMove implementation under the condition MoveType = mtMoveToBottomLevel.

Drilling cycle with shift

INCT_DrillingCycleWithShiftFormer - a CAM system interface that simplifies forming the movements of drilling cycles with shift. When the function INCT_CLData.CreateDrillingCycleFormer, INCT_GeomCLData.CreateDrillingCycleFormer is called with the dctWithShift parameter, it returns an object that implements two interfaces: INCT_DrillingCycleFormer, INCT_DrillingCycleWithShiftFormer.

The INCT_DrillingCycleFormer interface sets the basic parameters of the drilling cycle with shift. Its use is described in the article Drilling cycle.

The INCT_DrillingCycleWithShiftFormer interface adds to the generator the ability to control the parameters of forming the tool path with shift.



Purpose:

The movements in the various drilling cycles with shift generally follow the same tool path on different controllers:

1. Positioning the tool above the center of the hole or of the next hole at rapid traverse.
2. Moving the tool to the rapid level at rapid traverse.
3. Moving the tool to the hole bottom level at working feed.
4. Returning to the rapid level at rapid traverse.
5. Returning to the retract level at rapid traverse.

6. If the number of repetitions is not reached, returning to step 1.

On one or several sections of such cycles the tool path is shifted along one or several axes. Examples of such cycles: Fanuc: G76, G87; Sinumerik: CYCLE86; Heidenhain: CYCL DEF 202, etc.

Using the generator:

We recommend creating a new drilling cycle generator for each new cycle in a block of the NC program, except when the cycle is repeated in a new block (like CYCL CALL for Heidenhain). Therefore, the interface of the old generator must be released when a new one is created or when the interpreter finishes its work.

The INCT_DrillingCycleFormer interface allows forming the tool path from the specified parameters: initial position, rapid level, hole bottom level, number of repetitions, etc. It also allows adding to any section of the tool path the features required by a particular cycle (dwell, oriented spindle stop, tapping, etc.), changing the feed type or value, or skipping a tool movement prepared by the generator.

The INCT_DrillingCycleWithShiftFormer interface allows setting a shift of the coordinates of the tool movement points at any stage of the cycle except the first one. At each of these stages the tool moves from the current point to the new point as follows:

- from the current point to the shifted current point;
- from the shifted current point to the shifted new point;
- from the shifted new point to the new point.

To form the tool movement commands along the cycle path, do the following:

1. Get the generator interfaces (INCT_CLData.CreateDrillingCycleFormer, INCT_GeomCLData.CreateDrillingCycleFormer with the dctWithShift parameter).
2. Set the indexes of the X, Y, Z axes along which the movements are made (Set_AxisIndexes).
3. Set the current plane (Set_Plane).
4. Set the hole center point (Set_HoleCenter).
5. Set the hole center point mode (Set_HoleCenterMode).
6. Set the rapid level (Set_RapidLevel).
7. Set the hole bottom level (Set_BottomLevel).
8. Set the retract level (Set_TopLevel).
9. Set the feed (Set_Feedrate) and the feed units (Set_MPM).
10. Set the number of cycle repetitions (Set_Count).
11. Assign a cycle event handler (Set_Events) if the cycle path needs to be changed.
12. Set the shift type of the tool movement points (SetShiftByOneAxis or SetShiftBySomeAxis).
13. Set the tool path sections where the shift is applied (Set_ShiftPathSet).
14. Form the tool movement commands along the cycle path (MakeWorkpath).

Description of the INCT_DrillingCycleWithShiftFormer interface methods:

Method	Method description
SetShiftByOneAxis	The method switches the shift calculation to “Calculation along one axis” and sets the shift value. In this case the shift is made at all stages along one axis depending on the plane set by Set_Plane: for ptXY along the X axis, for ptYZ along the Y axis, for ptZX along the Z axis. Input parameter: Value (real number) - shift value. No output parameters.
SetShiftBySomeAxis	The method switches the shift calculation to “Calculation along the selected axes” and sets the shift value for each axis. In this case the point at the hole bottom level is shifted along the selected axes by the specified values. The points of the other levels are shifted along the selected axes by the specified values too, except along the drilling axis. Input parameters: Axis (TNCShiftAxis.X, Y, Z - boolean) - axes along which the shift is made; Values (TNCShiftValues.X, Y, Z - real number) - shift values along the axes. No output parameters.
Set_ShiftPathSet	The method sets the stages of the tool path at which the generator makes the shift. Input parameter: Value (TNCShiftPathSet) - set of tool path stages (TNCShiftPath) at which the generator makes the shift. Several stages can be selected. spMoveToRapidLevel - when moving to the rapid level; spMoveToBottomLevel - when moving to the hole bottom level; spReturnToRapidLevel - when returning to the rapid level; spReturnToTopLevel - when returning to the retract level.

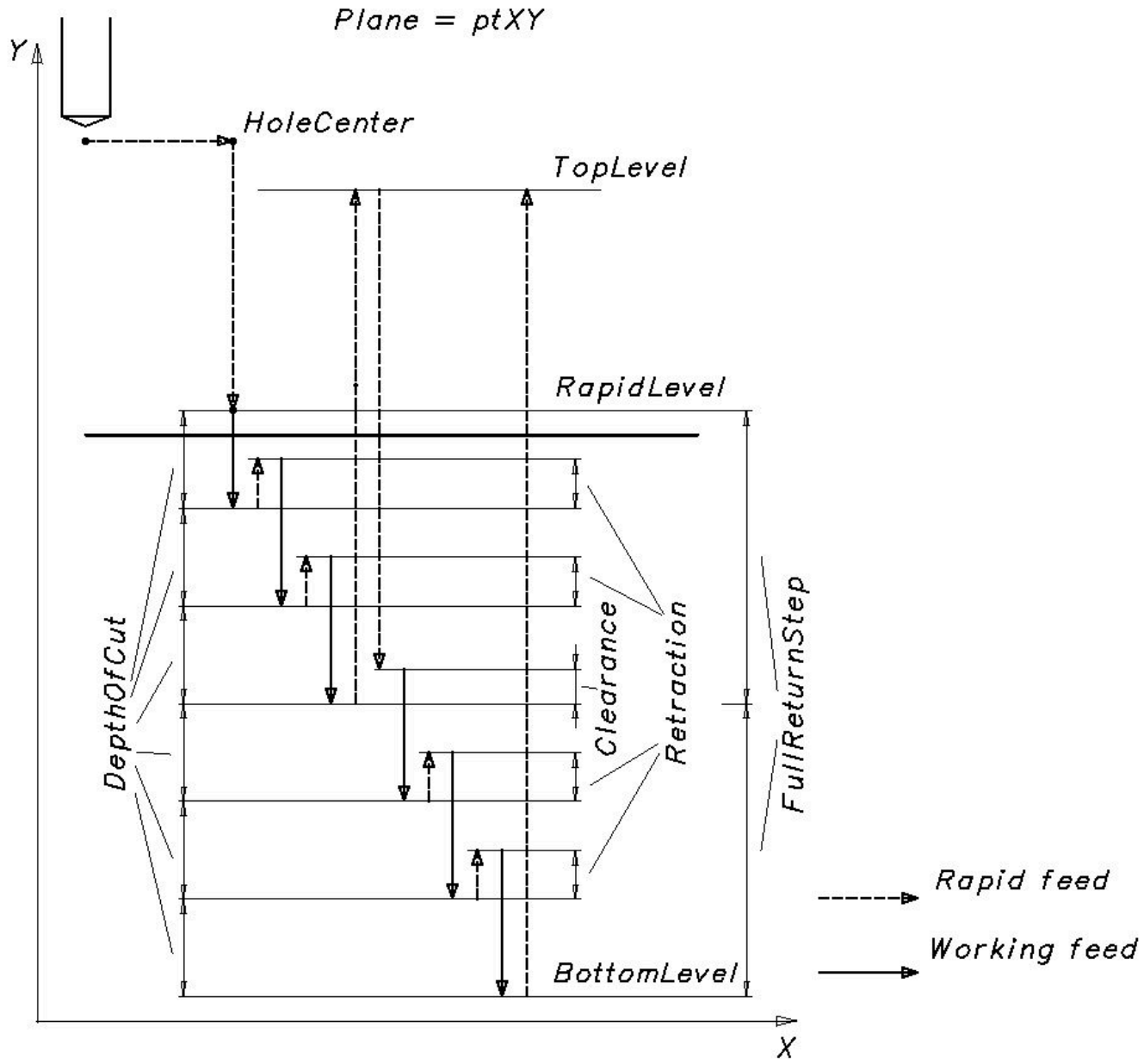
Method	Method description
	In the type library the type is called SetOfTNCShiftPath and is a byte; TNCShiftPathSet is its name in Delphi (set of TNCShiftPath). Each selected stage sets the bit whose number equals its TNCShiftPath value (bit masks: spMoveToRapidLevel = 1, spMoveToBottomLevel = 2, spReturnToRapidLevel = 4, spReturnToTopLevel = 8). No output parameters.
Get_ShiftPathSet	The function returns the set stages of the tool path at which the generator makes the shift (TNCShiftPathSet). No input parameters.
Get_ShiftNextHole	The function returns the coordinates of the shifted point of the current hole calculated for the current iteration (TNC3DPoint). The value can be used only inside the events (Set_Events). No input parameters.
Get_ShiftRapidPoint	The function returns the coordinates of the shifted point at the rapid level calculated for the current iteration (TNC3DPoint). The value can be used only inside the events (Set_Events). No input parameters.
Get_ShiftTopPoint	The function returns the coordinates of the shifted point at the retract level calculated for the current iteration (TNC3DPoint). The value can be used only inside the events (Set_Events). No input parameters.
Get_ShiftBottomPoint	The function returns the coordinates of the shifted point at the hole bottom level calculated for the current iteration (TNC3DPoint). The value can be used only inside the events (Set_Events). No input parameters.

Drilling cycle with chip breaking

INCT_PeckDrillingCycleFormer - a CAM system interface that simplifies forming the movements of drilling cycles with chip breaking. When the function INCT_CLData.CreateDrillingCycleFormer, INCT_GeomCLData.CreateDrillingCycleFormer is called with the dctPeck parameter, it returns an object that implements two interfaces: INCT_DrillingCycleFormer, INCT_PeckDrillingCycleFormer.

The INCT_DrillingCycleFormer interface sets the basic parameters of the drilling cycle with chip breaking. Its use is described in the article Drilling cycle.

The INCT_PeckDrillingCycleFormer interface adds to the generator the ability to control the parameters of forming the tool path with chip breaking.



Purpose:

The movements in the various drilling cycles with chip breaking generally follow the same tool path on different controllers:

1. Positioning the tool above the center of the hole or of the next hole at rapid traverse.
2. Moving the tool to the rapid level at rapid traverse.
3. Moving the tool to the hole bottom level at working feed.
4. Returning to the rapid level at rapid traverse.
5. Returning to the retract level at rapid traverse.
6. If the number of repetitions is not reached, returning to step 1.

In such cycles the hole is drilled (the stage of moving to the hole bottom level) not in one pass but in several iterations. Examples of such cycles: Fanuc: G73, G83; Sinumerik: CYCLE83; Heidenhain: CYCL DEF 200, 203, 205, etc.

Using the generator:

We recommend creating a new drilling cycle generator for each new cycle in a block of the NC program, except when the cycle is repeated in a new block (like CYCL CALL for Heidenhain). Therefore, the interface of the old generator must be released when a new one is created or when the interpreter finishes its work.

The INCT_DrillingCycleFormer interface allows forming the tool path from the specified parameters: initial position, rapid level, hole bottom level, number of repetitions, etc. It also allows adding to any section of the tool path the features required by a particular cycle (dwell, oriented spindle stop, tapping, etc.), changing the feed type or value, or skipping a tool movement prepared by the generator.

The INCT_PeckDrillingCycleFormer interface allows setting the parameters that control the number of iterations, the depth of cut per iteration, the type of tool retraction, etc.

To form the tool movement commands along the cycle path, do the following:

1. Get the generator interfaces (INCT_CLData.CreateDrillingCycleFormer, INCT_GeomCLData.CreateDrillingCycleFormer with the dctPeck parameter).
2. Set the indexes of the X, Y, Z axes along which the movements are made (Set_AxisIndexes).
3. Set the current plane (Set_Plane).
4. Set the hole center point (Set_HoleCenter).
5. Set the center point mode (Set_HoleCenterMode).
6. Set the rapid level (Set_RapidLevel).
7. Set the hole bottom level (Set_BottomLevel).
8. Set the retract level (Set_TopLevel).
9. Set the feed (Set_Feedrate) and the feed units (Set_MPM).
10. Set the number of cycle repetitions (Set_Count).
11. Assign a cycle event handler (Set_Events) if the cycle path needs to be changed.

12. Set the chip breaking iteration parameters (Set_StartLevel, Set_DepthOfCut, Set_DegressionType, Set_Degression, Set_FullReturn, etc., see the description below).
13. Form the tool movement commands along the cycle path (MakeWorkpath).

For the cycle with chip breaking, the INCT_DrillingCycleFormer.Set_Events method takes as input an object that implements two interfaces: INCT_DrillingCycleEvents and INCT_PeckDrillingCycleEvents. The INCT_DrillingCycleEvents events BeforeMove, OnMove, AfterMove receive as the movement type, besides the values described in the article Drilling cycle, the following values:

mtPeckMoveToNextDepthLevel - movement to the next depth of cut (next iteration);
mtPeckReturnToRapidLevel - return to the rapid level at the current iteration (chip removal);
mtPeckMoveToClearanceLevel - movement to the current cut level minus the clearance after the return from the rapid level;
mtPeckMoveToRetractionLevel - retraction and return to the current cut level (chip breaking).

Description of the INCT_PeckDrillingCycleFormer interface methods:

Method	Method description
Set_DepthOfCut	The method sets the depth of cut per iteration. The depth of cut is the distance the tool moves at each iteration when drilling the hole. Input parameter: Value (real number) - depth of cut. No output parameters.
Get_DepthOfCut	The function returns the depth of cut (real number). No input parameters.
Set_StartLevelType	The method sets the type of the “Start level” value. Input parameter: Value (TNCStartLevelType) - value type: slFromLevel - the first depth of cut is counted from the value set in Set_StartLevel. slFirstLevel - the value set in Set_StartLevel is the first depth of cut. No output parameters.
Get_StartLevelType	The function returns the type of the “Start level” value (TNCStartLevelType).

Method	Method description
	No input parameters.
Set_StartLevel	The method sets the level from which the first depth of cut is counted, or the first depth of cut itself, depending on the value set in Set_StartLevelType. Input parameter: Value (real number) - the value. No output parameters.
Get_StartLevel	The function returns the level from which the first depth of cut is counted, or the first depth of cut itself, depending on the value set in Set_StartLevelType (real number). No input parameters.
Set_Degression	The method sets the degression - the change of the depth of cut at each iteration. Input parameter: Value (real number) - degression value. The calculated degression depends on the degression type set in Set_DegressionType. No output parameters.
Get_Degression	The function returns the current degression value (real number). No input parameters.
Set_DegressionType	The method sets the degression type. Input parameter: Value (TNCDegressionType) - degression type. dtNone - no degression, the depth of cut does not change; dtValue - the depth of cut decreases by a constant value at each iteration (the value is set in Set_Degression); dtCoef - the depth of cut is multiplied by a factor at each iteration (the factor is set in Set_Degression). No output parameters.

Method	Method description
Get_DegressionType	The function returns the current degression type (TNCDegressionType). No input parameters.
Set_MinDepthOfCut	The method sets the minimum depth of cut. As a rule, it is used for cycles with degression. Input parameter: Value (real number) - minimum depth of cut. No output parameters.
Get_MinDepthOfCut	The function returns the current minimum depth of cut (real number). No input parameters.
Set_Retraction	The method sets the current retraction value. The retraction is the distance the tool rises to break the chip after reaching the current depth of cut at an iteration, if at this iteration there is no return to the rapid level (that is, the return flag in Set_FullReturn is False). Input parameter: Value (real number) - retraction value. The calculated retraction depends on the retraction type set in Set_RetractionType. No output parameters.
Get_Retraction	The function returns the current retraction value (real number). No input parameters.
Set_RetractionType	The method sets the retraction type. Input parameter: Value (TNCRetractionType) - retraction type. rtNone - no retraction; rtValue - retraction value (set in Set_Retraction). No output parameters.
Get_RetractionType	The function returns the current retraction type (TNCRetractionType). No input parameters.

Method	Method description
Set_RetractionCount	The method sets the number of retractions for chip breaking. Input parameter: Value (integer) - number of retractions for chip breaking. No output parameters.
Get_RetractionCount	The function returns the number of retractions for chip breaking (integer). No input parameters.
Set_ClearanceType	The method sets the clearance type. The clearance is the distance by which the tool stops short of the depth of cut of the current iteration after the return from the rapid level (that is, the return flag in Set_FullReturn is True). Input parameter: Value (TNCClearanceType) - clearance type. ctNone - no clearance; ctValue - value (set in Set_Clearance). No output parameters.
Get_ClearanceType	The function returns the current clearance type (TNCClearanceType). No input parameters.
Set_Clearance	The method sets the clearance value. Input parameter: Value (real number) - clearance value. The calculated clearance depends on the clearance type set in Set_ClearanceType. No output parameters.
Get_Clearance	The function returns the current clearance value (real number). No input parameters.
Set_FullReturn	The method sets the flag of return to the rapid level. If the flag is set, then after reaching the depth of cut at the current iteration the tool returns to the rapid level and then moves to the

Method	Method description
	cut level of the iteration taking the clearance into account (see Set_ClearanceType). Input parameter: Value (boolean) - return flag. True - with return to the rapid level; False - without return. No output parameters.
Get_FullReturn	The function returns the flag of return to the rapid level (boolean). No input parameters.
Set_FullReturnStep	The method sets the step of cut iterations at which the return to the rapid level is made. Example. The step of cut iterations is set to 3. In this case the return to the rapid level is made at every third iteration (iterations 3, 6, 9, etc.). At the other iterations there is no return (iterations 1, 2, 4, 5, 7, 8, etc.). Input parameter: Value (integer) - step of cut iterations. Only if Set_FullReturn is set to True (with return). No output parameters.
Get_FullReturnStep	The function returns the step of cut iterations at which the return to the rapid level is made (integer). No input parameters.
Set_SameLastSteps	The method sets the number of last equal cut steps. If the value is other than 0 or 1, the specified number of last iterations is made with the same depth of cut. Input parameter: Value (integer) - number of last equal cut steps. No output parameters.
Get_SameLastSteps	The function returns the current number of last equal cut steps (integer). No input parameters.

INCT_PeckDrillingCycleEvents - an interface implemented in the interpreter that allows getting and controlling the values calculated inside the drilling cycle with chip breaking. The interface is a set of event methods executed when particular values are calculated inside the cycle path generator. With these methods, the cut levels of the iterations, the flag of return to the rapid level at an iteration, and the clearance and retraction values can be obtained and changed while the cycle path is formed.

If events are assigned to the cycle path generator, the corresponding method is called for each calculated value:

1. The INCT_PeckDrillingCycleEvents.OnCalcNextDepth method is executed when the next calculated cut level is added to the internal list of the generator. The number of cut levels (the list size) is the number of iterations in which the tool reaches the hole bottom. The method receives the calculated cut level.
2. The INCT_PeckDrillingCycleEvents.OnGetNextFullReturn method is executed after the flag of return to the rapid level is calculated from the cycle parameters. The method receives the iteration number and the calculated flag.
3. The INCT_PeckDrillingCycleEvents.OnGetNextClearance method is executed after the clearance value is calculated. The method receives the iteration number and the calculated clearance.
4. The INCT_PeckDrillingCycleEvents.OnGetNextRetraction method is executed after the retraction value is calculated. The method receives the iteration number and the calculated retraction.

Description of the interface methods:

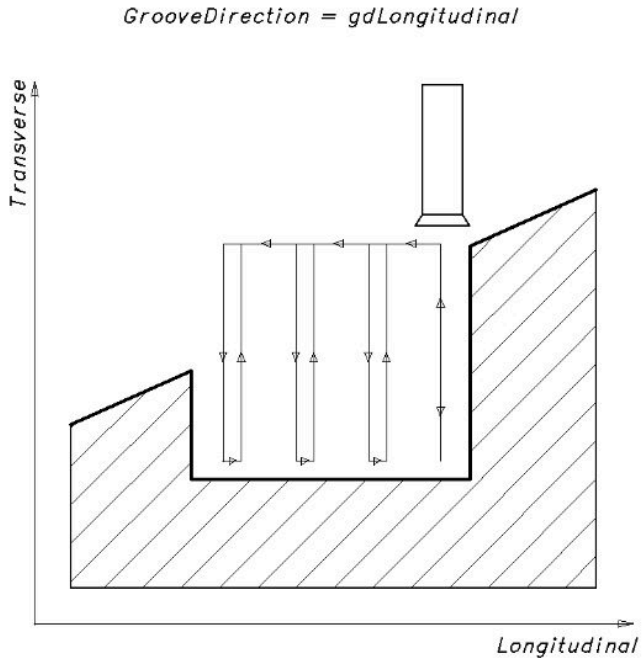
Method	Method description	Description of use
OnCalcNextDepth	The method is executed each time the next calculated cut level is added to the internal list of the generator. The size of the formed list (the number of elements) is the number of iterations in which the tool reaches the hole bottom. Input and output parameter: Depth (real number) - cut level for the iteration (absolute value).	1. To get the total number of iterations for recalculating the clearance, retraction or return to the rapid level when implementing calculation algorithms that the cycle does not provide. 2. To change the cut levels when implementing calculation algorithms that the cycle does not provide.
OnGetNextFullReturn	The method is executed after the flag of return to the rapid	To change the flag of return to the rapid level for the current

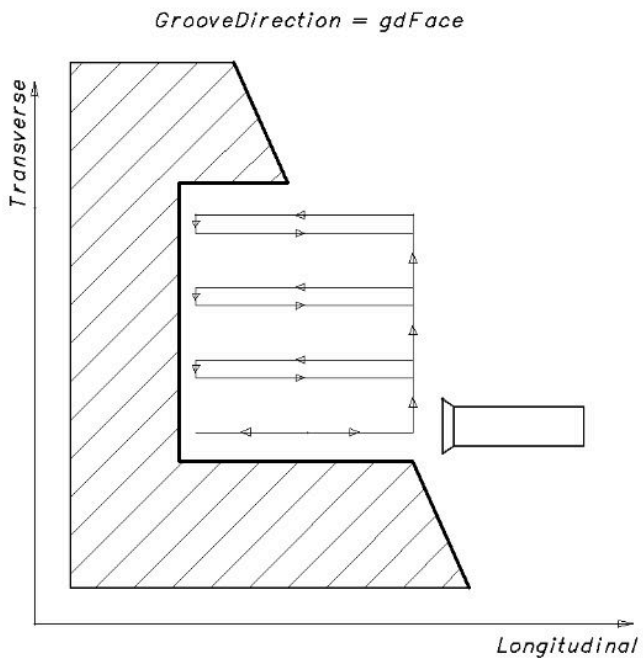
Method	Method description	Description of use
	level is calculated from the cycle parameters for the current iteration. Input parameters: Iteration (integer) - iteration number. Input and output parameter: FullReturn (boolean) - flag of return to the rapid level at the given iteration.	iteration when implementing calculation algorithms that the cycle does not provide.
OnGetNextClearance	The method is executed after the clearance value is calculated from the cycle parameters for the current iteration. It is called only if the flag of return to the rapid level is set for the iteration (FullReturn = True). Input parameters: Iteration (integer) - iteration number. Input and output parameter: Clearance (real number) - clearance value at the given iteration.	To change the clearance value for the current iteration when implementing calculation algorithms that the cycle does not provide.
OnGetNextRetraction	The method is executed after the retraction value is calculated from the cycle parameters for the current iteration. It is called only if the flag of return to the rapid level is not set for the iteration (FullReturn = False). Input parameters: Iteration (integer) - iteration number.	To change the retraction value for the current iteration when implementing calculation algorithms that the cycle does not provide.

Method	Method description	Description of use
	Input and output parameter: Retraction (real number) - retraction value at the given iteration.	

Turning grooving cycle

INCT_LatheGroovingCycleFormer - a CAM system interface that simplifies forming the movements of a turning grooving cycle. The object implementing this interface is returned by the function INCT_CLData.CreateLatheCycleFormer, INCT_GeomCLData.CreateLatheCycleFormer when it is called with the lctGrooving parameter.





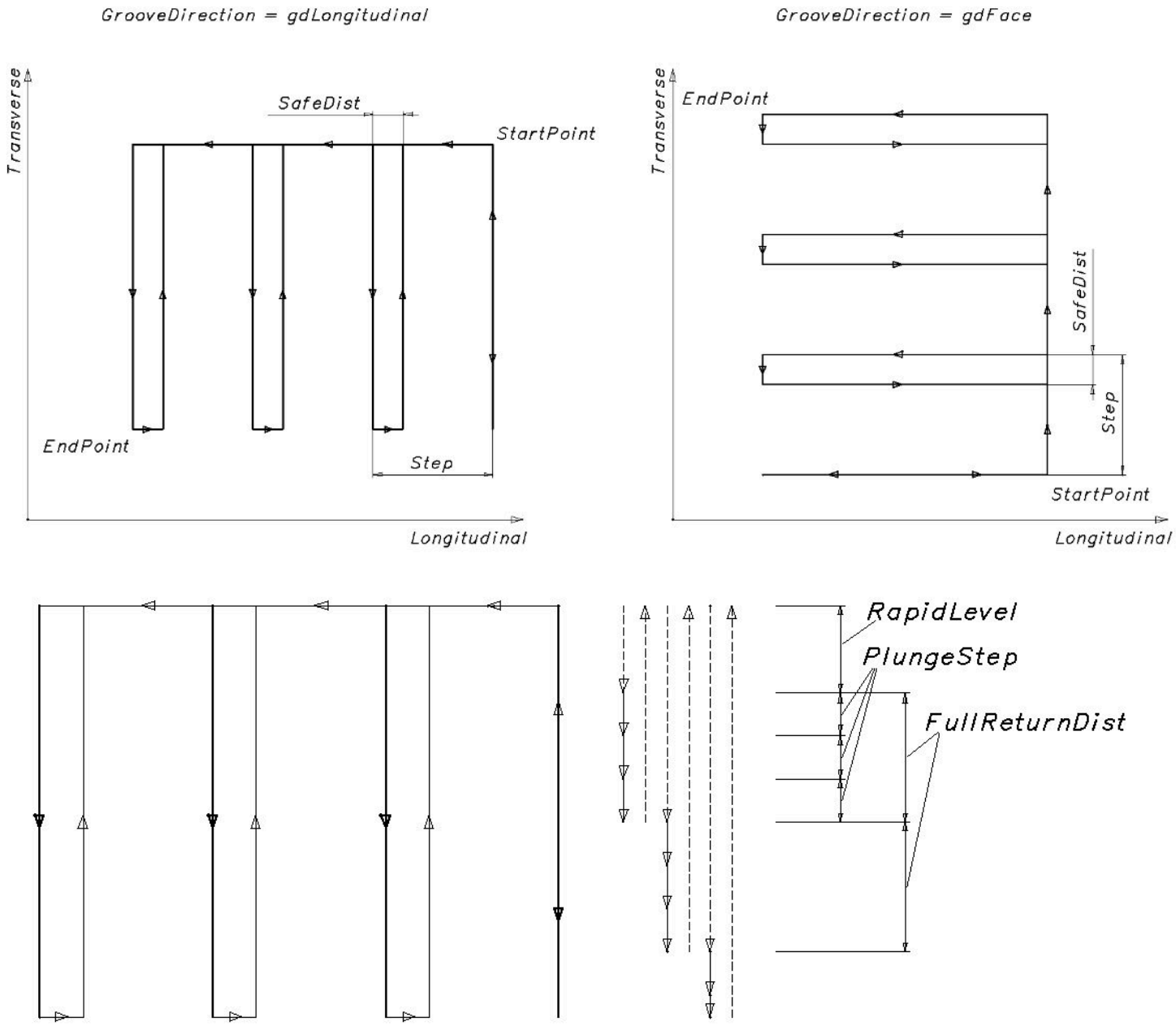
Purpose:

The movements in the various types of grooving cycles generally follow the same tool path on different controllers:

1. Moving to the start point or to the next plunge point at rapid traverse.
2. Plunging to the groove depth in one pass or in several steps by the depth of cut at working feed, with retraction for chip breaking.
3. Returning to the start position with or without retraction at transition feed.
4. If the groove width is not reached, returning to step 1.

Example of such a cycle: Sinumerik: CYCLE93.

The generator forms the tool path from the specified parameters: groove start and end points, step across the groove width, depth of cut, retraction distance, etc. It also allows adding to any section of the tool path the features required by a particular cycle, changing the feed type or value, or skipping a tool movement prepared by the generator.



Using the generator:

We recommend creating a new turning cycle generator for each new cycle in a block of the NC program, except when the cycle is repeated in a new block. Therefore, the interface of the old generator must be released when a new one is created or when the interpreter finishes its work.

To form the tool movement commands along the cycle path, do the following:

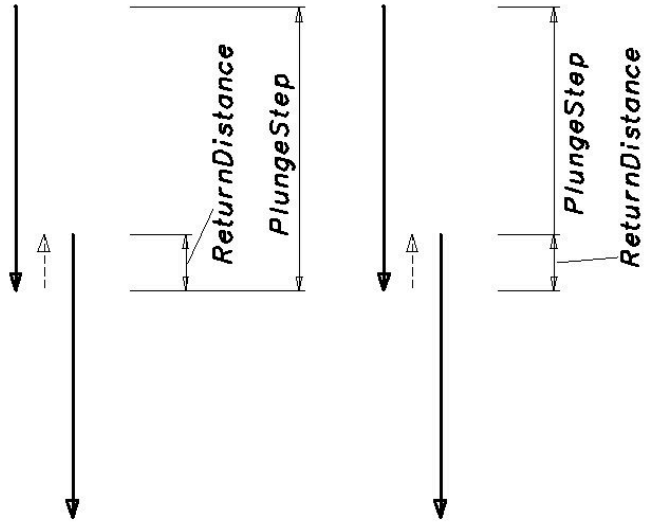
1. Get the generator interface (INCT_CLData.CreateLatheCycleFormer, INCT_GeomCLData.CreateLatheCycleFormer).
2. Set the indexes of the longitudinal and transverse axes along which the movements are made (Set_AxisIndexes).
3. Set the groove start point (Set_StartPoint).
4. Set the groove end point (Set_EndPoint).

5. Set the groove direction (Set_GrooveDirection).
6. Set the step across the groove width (Set_Step).
7. Set the retraction distance for the return (Set_SafeDist).
8. Set the depth of cut in the axial direction (Set_PlungeStep).
9. Set the retraction distance for chip breaking (Set_ReturnDistance).
10. Assign a cycle event handler (Set_Events) if the cycle path needs to be changed.
11. Form the tool movement commands along the cycle path (MakeWorkpath).

Description of the interface methods:

Method	Method description
Set_AxisIndexes	The method sets the indexes of the longitudinal and transverse axes along which the movements are made when the cycle path is formed. Input parameters: Value (TNCLCAxisIndexes) - Transverse (integer) - index of the transverse axis, Longitudinal (integer) - index of the longitudinal axis. No output parameters.
Get_AxisIndexes	The function returns the indexes of the longitudinal and transverse axes along which the movements are made when the cycle path is formed (TNCLCAxisIndexes). No input parameters.
Set_StartPoint	The method sets the coordinates of the groove start point. Input parameter: Value (TNC2DPoint) - coordinates of the start point (Transverse - transverse, Longitudinal - longitudinal). No output parameters.
Get_StartPoint	The function returns the coordinates of the groove start point (TNC2DPoint). No input parameters.
Set_EndPoint	The method sets the coordinates of the groove end point. Input parameter: Value (TNC2DPoint) -

Method	Method description
	coordinates of the end point. No output parameters.
Get_EndPoint	The function returns the coordinates of the groove end point (TNC2DPoint). No input parameters.
Set_GrooveDirection	The method sets the groove direction. Input parameter: Value (TNCGrooveDirection) - groove direction. gdLongitudinal - longitudinal groove; gdFace - face groove. No output parameters.
Get_GrooveDirection	The function returns the groove direction (TNCGrooveDirection). No input parameters.
Set_Step	The method sets the step across the groove width in the radial direction. Input parameter: Value (real number) - step. No output parameters.
Get_Step	The function returns the step across the groove width in the radial direction (real number). No input parameters.
Set_SafeDist	The method sets the side retraction distance before the tool exits. Input parameter: Value (real number) - side retraction distance. No output parameters.
Get_SafeDist	The function returns the side retraction distance before the tool exits (real number). No input parameters.
Set_PlungeStep	The method sets the step of plunging to the groove depth. Input parameter: Value (real number) - step. No output parameters.

Method	Method description
Get_PlungeStep	The function returns the step of plunging to the groove depth (real number). No input parameters.
Set_ReturnDistance	The method sets the upward retraction distance for chip breaking. Input parameter: Value (real number) - upward retraction distance. No output parameters.
Get_ReturnDistance	The function returns the upward retraction distance for chip breaking (real number). No input parameters.
Set_ReturnDistanceInsidePlungeStep	The method sets the flag “Retraction inside the plunge step”. True - the retraction is included in the plunge step, False - the retraction is added to the plunge step. Input parameter: Value (boolean) - flag value. No output parameters. 
Get_ReturnDistanceInsidePlungeStep	The function returns the flag “Retraction inside the plunge step”. No input parameters.

Method	Method description
Set_RapidLevel	The method sets the rapid level. Input parameter: Value (real number) - rapid level. No output parameters.
Get_RapidLevel	The function returns the rapid level. No input parameters.
Set_FullReturnDist	The method sets the retraction distance to the rapid level. Input parameter: Value (real number) - retraction distance to the rapid level. No output parameters.
Get_FullReturnDist	The function returns the retraction distance to the rapid level. No input parameters.
Set_Feedrate	The method sets the working feed. Input parameter: Value (real number) - working feed. No output parameters.
Get_Feedrate	The function returns the working feed. No input parameters.
Set_MPM	The method sets the working feed units. Input parameter: Value (boolean) - working feed units. True - mm/min, False - mm/rev. No output parameters.
Get_MPM	The function returns the working feed units. No input parameters.
Set_Events	The method assigns events that control the cycle behavior. Events allow adding to any section of the tool path the features required by a particular cycle, changing the feed type or value, or skipping a tool movement prepared by the generator. Input parameter: Value (INCT_LatheCycleEvents) - object implementing the events. See the detailed

Method	Method description
	description below. No output parameters.
Get_Events	The function returns the object implementing the events that is linked to the generator (INCT_LatheCycleEvents). No input parameters.
MakeWorkpath	The method forms the cycle path from the specified parameters.

INCT_LatheCycleEvents - an interface implemented in the interpreter that allows controlling the behavior of a turning cycle. The interface is a set of event methods executed during the tool movements inside the generator. With these methods, the features required by a particular cycle (dwell, oriented spindle stop, etc.) can be added to any section of the tool path, the feed type can be changed, or a tool movement prepared by the generator can be skipped.

If events are assigned to the generator, every tool movement inside the generator consists of 3 stages:

1. The INCT_LatheCycleEvents.BeforeMove method is executed. It receives the values set by the cycle: movement type, coordinates of the movement point, feed type.
2. The INCT_LatheCycleEvents.OnMove method is executed. It receives: movement type, coordinates of the movement point, feed type and the flag of whether the movement has been made. All parameters except the movement type can be changed in the implementation. Depending on the flag set in OnMove, the tool moves to the specified point with the specified feed, or the movement is not made.
3. The INCT_LatheCycleEvents.AfterMove method is executed. It receives the values returned by OnMove: movement type, coordinates of the movement point, feed type.

Description of the interface methods:

Method	Method description	Description of use
BeforeMove	The method is executed before each tool movement when the cycle path is formed. Input parameters: MoveType (TNCLatheCycleMoveType) - movement type, see the description below; Point (TNC2DPoint) - coordinates of	If some commands must be added before a movement, add them to the method implementation. For example, to turn on the machine coolant before grooving, call INCT_CLData.AddCoolant with

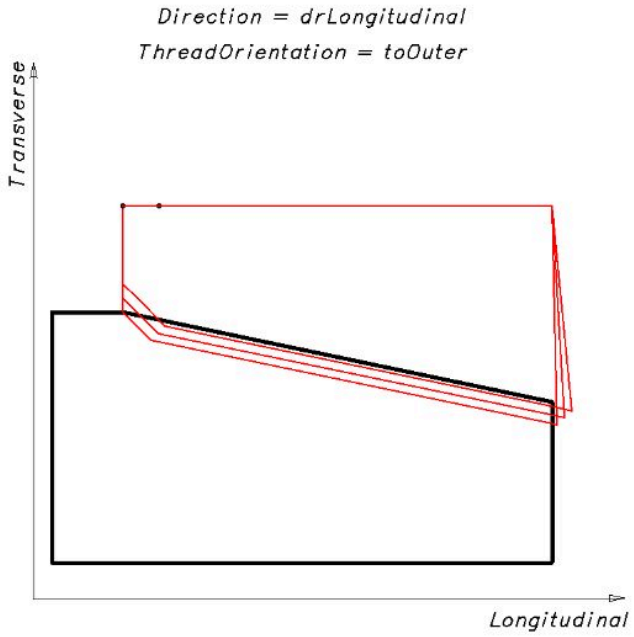
Method	Method description	Description of use
	the movement point (Transverse - transverse, Longitudinal - longitudinal); Feed (TNCFeedType) - feed type. ImMoveToStartPoint - movement to the groove start point; ImMoveToNextGroove - movement to the next groove step; ImMoveToNextLevel - movement to the next plunge depth; ImMoveToBottomLevel - movement to the groove bottom; ImChipBreaking - chip breaking; ImRetractSideways - side retraction before the tool exits; ImReturnToTopLevel - return to the initial level; ImReturnToStartPoint - return to the start point; ImCutTheThread - thread cutting (for the Turning threading cycle); ImApproach - approach (for the Turning stock removal cycle); ImRetract - retract (for the turning stock removal cycle); ImWorkingMove - working movement (for the turning stock removal cycle); ImOverlapContour - pass along the contour (for the turning stock removal cycle); ImReturnToSafeDistance - return to the safe distance (for the turning stock removal cycle); ImTtransition - transition (for the turning stock removal cycle); ImMoveToRapidLevel - return to	the appropriate parameters in the BeforeMove implementation under the condition MoveType = ImMoveToStartPoint.

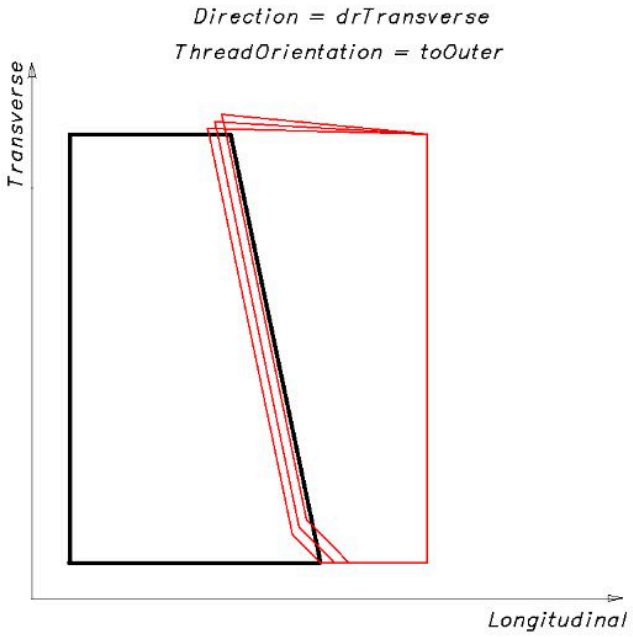
Method	Method description	Description of use
	the rapid level (tool exit). No output parameters.	
OnMove	The method is executed immediately before each tool movement when the cycle path is formed. Input parameters: MoveType (TNCLatheCycleMoveType) - movement type, see the description above; Input and output parameters: Point (TNC2DPoint) - coordinates of the movement point; Feed (TNCFeedType) - feed type; Handled (boolean) - flag of whether the movement has been made inside the implementation (True - the movement has been made, False - it has not). If Handled is True after the OnMove call, the movement has been made inside the OnMove implementation, and the generator does not add a movement command to the tool path. If Handled is False after the OnMove call, the generator considers that the movement has not been made and adds to the tool path a command to move to the point Point with the feed type Feed.	To override the movement point or the feed, or to skip the movement. For example, to switch to rapid traverse when returning to the initial level after grooving, in the OnMove implementation set the returned Feed parameter to ffRapid under the condition MoveType = lmReturnToTopLevel (for the feed type, see OutStandardFeed). To move the tool to a point other than the one calculated by the cycle, change the Point parameter returned by OnMove. If the movement to the calculated point is not needed, set the Handled parameter to True inside the OnMove implementation.
AfterMove	The method is executed after each tool movement when the cycle path is formed, even if the	If some commands must be added after a movement, add them to the method

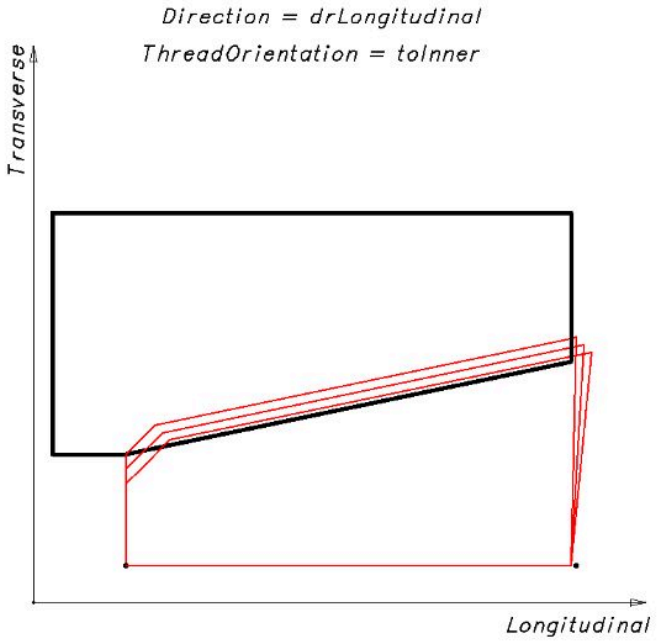
Method	Method description	Description of use
	movement has been canceled in OnMove. Input parameters: MoveType (TNCLatheCycleMoveType) - movement type, see the description above; Point (TNC2DPoint) - coordinates of the movement point; Feed (TNCFeedType) - feed type. No output parameters.	implementation. For example, to add a dwell at the groove bottom, call INCT_CLData.AddDelay with the appropriate parameters in the AfterMove implementation under the condition MoveType = lmMoveToBottomLevel.

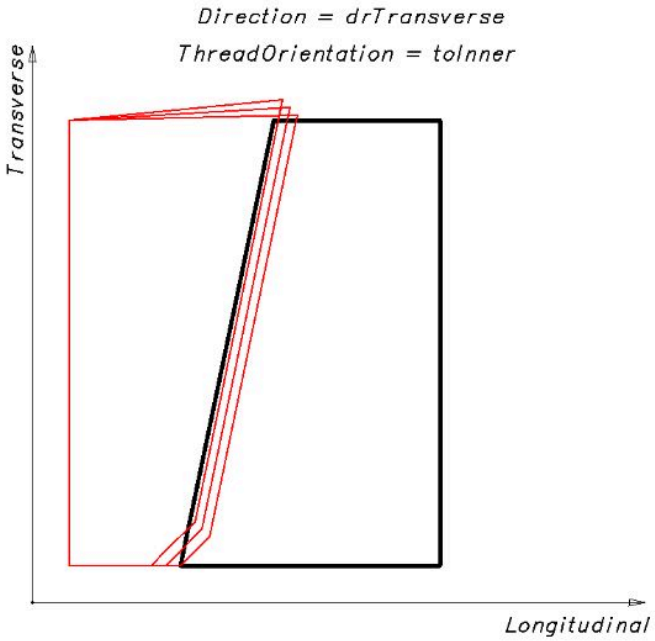
Turning threading cycle

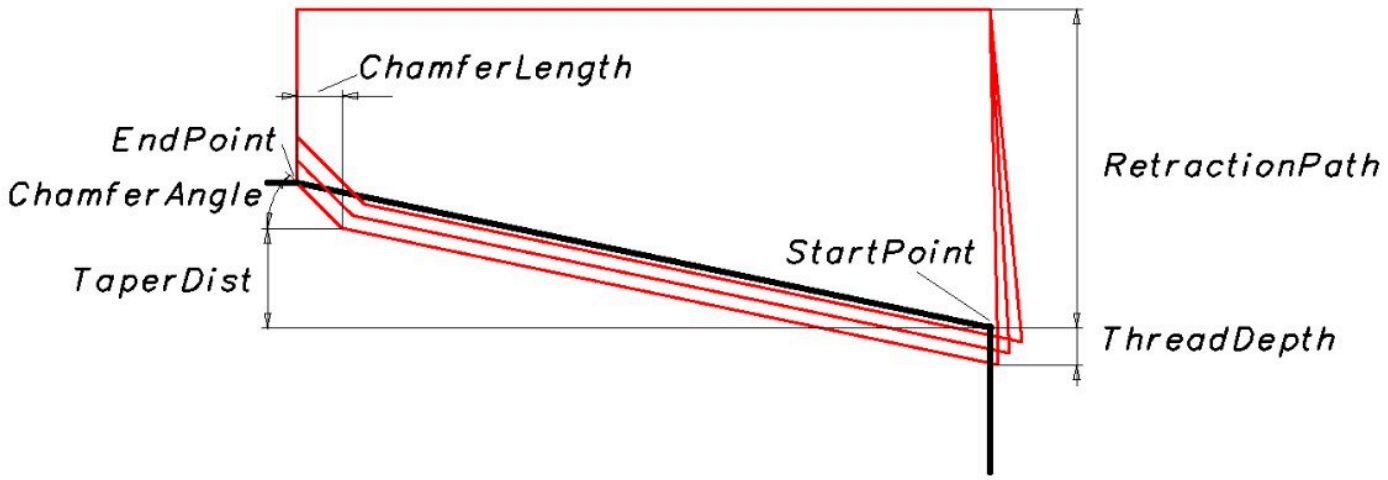
INCT_LatheThreadCuttingCycleFormer - a CAM system interface that simplifies forming the movements of a turning thread cutting cycle. The object implementing this interface is returned by the function INCT_CLData.CreateLatheCycleFormer, INCT_GeomCLData.CreateLatheCycleFormer when it is called with the lctThreadCutting parameter.











Purpose:

The movements in the various types of turning thread cutting cycles generally follow the same tool path on different controllers:

1. Approach of the tool to the start point or to the point of the next start in the retraction plane at rapid traverse.
2. Movement to the thread start point at working feed.
3. Thread cutting to the end point at working feed.
4. Return to the retraction plane at rapid traverse.
5. If the number of passes (roughing and finishing) is not reached, returning to step 1.

Example of such a cycle: Sinumerik: CYCLE97.

The generator forms the tool path from the specified parameters: thread start and end points, retraction plane, number of thread starts, infeed method, infeed angle, etc. It also allows adding to any section of the tool path the features required by a particular cycle, changing the feed type or value, or skipping a tool movement prepared by the generator.

Using the generator:

We recommend creating a new turning cycle generator for each new cycle in a block of the NC program, except when the cycle is repeated in a new block. Therefore, the interface of the old generator must be released when a new one is created or when the interpreter finishes its work.

To form the tool movement commands along the cycle path, do the following:

1. Get the generator interface (INCT_CLData.CreateLatheCycleFormer, INCT_GeomCLData.CreateLatheCycleFormer).
2. Set the indexes of the longitudinal and transverse axes along which the movements are made (Set_AxisIndexes).

3. Set the cycle parameters (Set_ThreadOrientation, Set_ThreadDepth, Set_ThreadStep, Set_ThreadStartAngle, Set_StartPoint, etc., see the description below).
4. Assign a cycle event handler (Set_Events) if the cycle path needs to be changed.
5. Form the tool movement commands along the cycle path (MakeWorkpath).

Description of the interface methods:

Method	Method description
Set_AxisIndexes	The method sets the indexes of the longitudinal and transverse axes along which the movements are made when the cycle path is formed. Input parameters: Value (TNCLCAxisIndexes) - Transverse (integer) - index of the transverse axis, Longitudinal (integer) - index of the longitudinal axis. No output parameters.
Get_AxisIndexes	The function returns the indexes of the longitudinal and transverse axes along which the movements are made when the cycle path is formed (TNCLCAxisIndexes). No input parameters.
Set_Direction	The method sets the direction of the thread cutting cycle. Input parameter: Value (TNCThreadCuttingDirection) - cycle direction: drLongitudinal - longitudinal, drTransverse - transverse. No output parameters.
Get_Direction	The function returns the cycle direction (TNCThreadCuttingDirection). No input parameters.
Set_ThreadOrientation	The method sets the thread orientation type. Input parameter: Value (TNCThreadOrientation, see INCT_CLData.AddSinglePassThread) - thread orientation type. No output parameters.

Method	Method description
Get_ThreadOrientation	The function returns the thread orientation type (TNCThreadOrientation). No input parameters.
Set_ThreadDepth	The method sets the thread depth. Input parameter: Value (real number) - thread depth. No output parameters.
Get_ThreadDepth	The function returns the thread depth (real number). No input parameters.
Set_ThreadStep	The method sets the thread pitch. Input parameter: Value (real number) - thread pitch. No output parameters.
Get_ThreadStep	The function returns the thread pitch (real number). No input parameters.
Set_ThreadStartAngle	The method sets the thread start angle. Input parameter: Value (real number) - thread start angle. No output parameters.
Get_ThreadStartAngle	The function returns the thread start angle (real number). No input parameters.
Set_ThreadStartCount	The method sets the number of thread starts. Input parameter: Value (integer) - number of thread starts. No output parameters.
Get_ThreadStartCount	The function returns the number of thread starts (integer). No input parameters.

Method	Method description
Set_StartPoint	The method sets the coordinates of the thread start point. Input parameter: Value (TNC2DPoint) - coordinates of the thread start point. No output parameters.
Get_StartPoint	The function returns the coordinates of the thread start point (TNC2DPoint). No input parameters.
Set_EndPoint	The method sets the coordinates of the thread end point. Input parameter: Value (TNC2DPoint) - coordinates of the thread end point. No output parameters.
Get_EndPoint	The function returns the coordinates of the thread end point (TNC2DPoint). No input parameters.
Set_TaperDist	The method sets the taper between the thread start and end points. Input parameter: Value (real number) - taper between the thread start and end points. No output parameters.
Get_TaperDist	The function returns the taper between the thread start and end points (real number). No input parameters.
Set_ChamferLength	The method sets the length of the chamfer for the tool runout at the thread end. Input parameter: Value (real number) - length of the chamfer for the tool runout at the thread end. No output parameters.
Get_ChamferLength	The function returns the length of the chamfer for the tool runout at the thread end (real number). No input parameters.

Method	Method description
Set_ChamferAngle	The method sets the chamfer angle in degrees. Input parameter: Value (real number) - chamfer angle in degrees. No output parameters.
Get_ChamferAngle	The function returns the chamfer angle in degrees (real number). No input parameters.
Set_RetractionPath	The method sets the level of the retraction plane. Input parameter: Value (real number) - level of the retraction plane. No output parameters.
Get_RetractionPath	The function returns the set level of the retraction plane (real number). No input parameters.
Set_RunInPath	The method sets the run-in path. Input parameter: Value (real number) - run-in path. No output parameters.
Get_RunInPath	The function returns the run-in path (real number). No input parameters.
Set_RunOutPath	The method sets the run-out path. Input parameter: Value (real number) - run-out path. No output parameters.
Get_RunOutPath	The function returns the run-out path (real number). No input parameters.
Set_ThreadAngle	The method sets the thread profile angle. Input parameter: Value (real number) - thread profile angle. No output parameters.
Get_ThreadAngle	The function returns the thread profile angle (real number).

Method	Method description
	No input parameters.
Set_InclinationAngle	The method sets the angle of one of the thread profile flanks relative to the vertical. Input parameter: Value (real number) - angle value. No output parameters.
Get_InclinationAngle	The function returns the angle of one of the thread profile flanks relative to the vertical (real number). No input parameters.
Set_SequenceType	The method sets the infeed method for roughing passes. Input parameter: Value (TNCThreadSequenceType) - infeed method for roughing passes. stRadial - radial; stFlank - flank; stModifiedFlank - modified flank (at an angle); stAlternateFlank - alternate flank. No output parameters.
Get_SequenceType	The function returns the infeed method for roughing passes (TNCThreadSequenceType). No input parameters.
Set_PlungeAngle	The method sets the infeed angle for the stModifiedFlank strategy. Input parameter: Value (real number) - infeed angle in degrees. No output parameters.
Get_PlungeAngle	The function returns the infeed angle for the stModifiedFlank strategy (real number). No input parameters.
Set_DepthIsEqual	The method sets the way the depth of cut is determined. Input parameter: Value (boolean) - way of determining the depth of cut. True - constant

Method	Method description
	depth, False - constant cutting area. No output parameters.
Get_DepthIsEqual	The function returns the way the depth of cut is determined (boolean). No input parameters.
Set_IsPassCountDefined	The method sets the method of determining the depth of cut. Input parameter: Value (boolean) - method of determining the depth of cut. True - by the number of passes, False - by the initial depth. No output parameters.
Get_IsPassCountDefined	The function returns the method of determining the depth of cut (boolean). No input parameters.
Set_MinCutDepth	The method sets the minimum possible depth of a roughing pass. Input parameter: Value (real number) - minimum possible depth of a roughing pass. No output parameters.
Get_MinCutDepth	The function returns the minimum possible depth of a roughing pass (real number). No input parameters.
Set_FirstCutDepth	The method sets the depth of the first roughing pass. Input parameter: Value (real number) - depth of the first roughing pass. No output parameters.
Get_FirstCutDepth	The function returns the depth of the first roughing pass (real number). No input parameters.
Set_RoughPassCount	The method sets the number of roughing passes. Input parameter: Value (integer) - number of

Method	Method description
	roughing passes. No output parameters.
Get_RoughPassCount	The function returns the number of roughing passes (integer). No input parameters.
Set_FinishPassCount	The method sets the number of finishing passes. Input parameter: Value (integer) - number of finishing passes. No output parameters.
Get_FinishPassCount	The function returns the number of finishing passes (integer). No input parameters.
Set_FinishPassDepth	The method sets the depth of the finishing passes. Input parameter: Value (real number) - depth of the finishing passes. No output parameters.
Get_FinishPassDepth	The function returns the depth of the finishing passes (real number). No input parameters.
Set_Feedrate	The method sets the working feed. Input parameter: Value (real number) - working feed. No output parameters.
Get_Feedrate	The function returns the working feed (real number). No input parameters.
Set_MPM	The method sets the working feed units. Input parameter: Value (boolean) - working feed units. True - mm/min, False - mm/rev. No output parameters.

Method	Method description
Get_MPM	The function returns the working feed units (boolean). No input parameters.
Set_Events	The method assigns events that control the cycle behavior. Events allow adding to any section of the tool path the features required by a particular cycle (dwell, oriented spindle stop, etc.), changing the feed type or value, or skipping a tool movement prepared by the generator. Input parameter: Value (interface) - object implementing the interfaces INCT_LatheCycleEvents, INCT_LatheThreadCuttingCycleEvents. See the detailed description below. No output parameters.
Get_Events	The function returns the object implementing the events that is linked to the cycle generator (INCT_LatheCycleEvents, INCT_LatheThreadCuttingCycleEvents). No input parameters.
MakeWorkpath	The method forms the cycle path from the specified parameters.

INCT_LatheThreadCuttingCycleFormer2 - a CAM system interface that simplifies forming the movements of a turning thread cutting cycle.

Description of the interface methods:

Method	Method description
Set_Approach	The method sets the approach type. Input parameter: Value (TNCThreadApproach) - approach type. taOutside - from outside (default); taIndise - from inside. No output parameters.

Method	Method description
Get_Approach	The function returns the approach type (TNCThreadApproach). No input parameters.
Get_SignLongitudinal	The function returns the sign of the movement along the longitudinal axis (integer).
Get_SignTransverse	The function returns the sign of the movement along the transverse axis (integer).

INCT_LatheCycleEvents - an interface implemented in the interpreter that allows controlling the behavior of a turning cycle. It is described in detail in the article Turning grooving cycle.

INCT_LatheThreadCuttingCycleEvents - an interface implemented in the interpreter that allows controlling thread cutting. It contains the event method OnThread, executed when a thread is cut inside the generator, and the events of calculating the passes OnCalcBaseDepth, OnCalcCount, OnCalcNextDepth, OnCalcPoint.

Description of the interface methods:

Method	Method description	Description of use
OnThread	The method is executed before each thread cut when the cycle path is formed. Its implementation allows changing the thread parameters for the selected iteration. Input parameters: Orient (TNCThreadOrientation) - thread orientation; LeadType (TNCThreadLeadType) - way of setting the thread pitch; Step (real number) - thread pitch; StartAngle (real number) - initial angular position of the spindle in degrees (used for multi-start threads). See INCT_CLData.AddSinglePassThread.	To replace the thread cutting parameters, or to set an algorithm of changing the thread at different passes that the generator does not implement.

Method	Method description	Description of use
	Input and output parameters: Handled (boolean) - flag of whether the movement has been made inside the implementation (True - the movement has been made, False - it has not). If Handled is True after the OnThread call, the thread has been cut inside the OnThread implementation, and the cycle generator does not add a thread cutting command to the tool path. If Handled is False after the OnThread call, the generator considers that the thread has not been cut inside the OnThread implementation and adds a thread cutting command to the tool path.	
OnCalcBaseDepth	The method is executed in the IsPassCountDefined = True mode after the base (first) depth is calculated. Its implementation allows changing the base depth. Input and output parameter: BaseDepth (real number) - base depth.	
OnCalcCount	The method is executed in the IsPassCountDefined = False mode after the number of passes is calculated. Its	

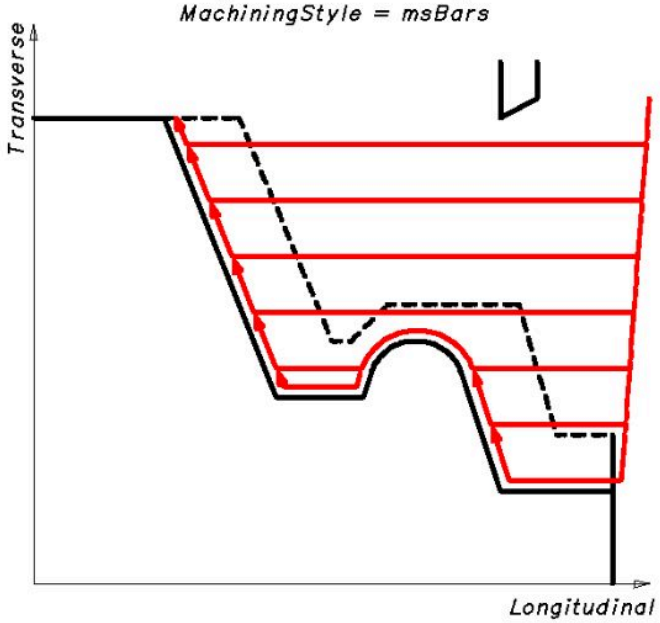
Method	Method description	Description of use
	implementation allows changing the number of passes. Input and output parameter: Count (integer) - number of passes.	
OnCalcNextDepth	The method is executed when the next depth is calculated (if the number of passes is one or more). Its implementation allows changing the depth for the iteration. Input parameter: Iteration (integer) - iteration (pass) number; Input and output parameter: Depth (real number) - depth at the iteration.	
OnCalcPoint	The method is executed when the coordinates of the tool position for the next movement are calculated. Its implementation allows changing the coordinates for the iteration. Input parameter: Iteration (integer) - iteration (pass) number; Input and output parameter: Point (TNC2DPoint) - coordinates of the tool position at the iteration.	

Turning stock removal cycle

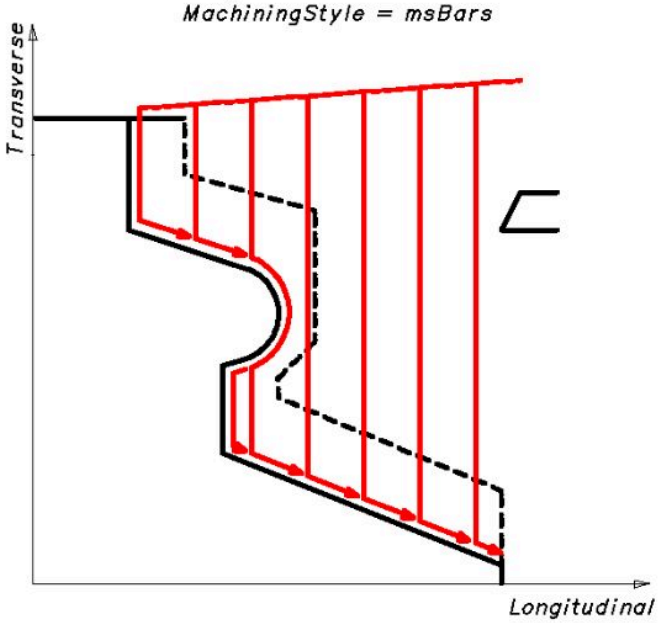
INCT_LatheStockRemovalCycleFormer - a CAM system interface that simplifies forming the movements of a turning cycle along a specified contour. The object implementing this interface is returned by the function INCT_CLData.CreateLatheCycleFormer, INCT_GeomCLData.CreateLatheCycleFormer when it is called with the lctStockRemoval parameter.

The INCT_LatheStockRemovalCycleFormer2 interface extends the capabilities of the turning cycle generator along a specified contour. Its new properties and methods are described below. The object implementing this interface is returned by the function CreateLatheCycleFormer (as for INCT_LatheStockRemovalCycleFormer).

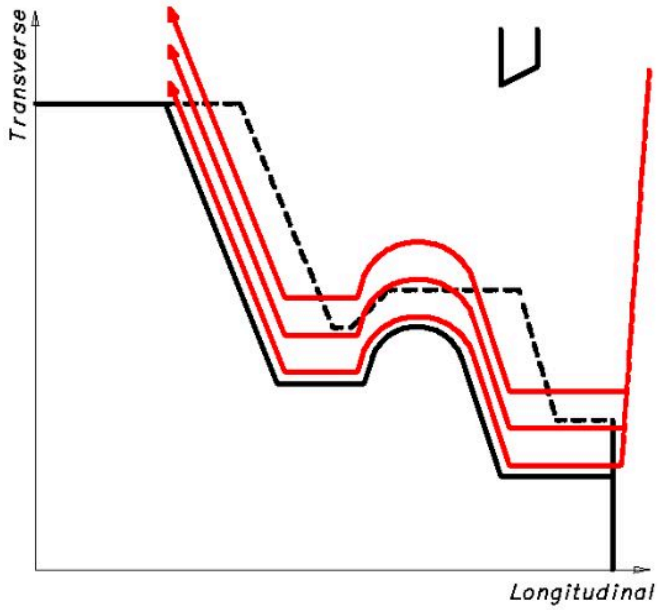
MachiningProcessing = mpRoughing
MachiningDirection = mdLongitudinal
MachiningStyle = msBars



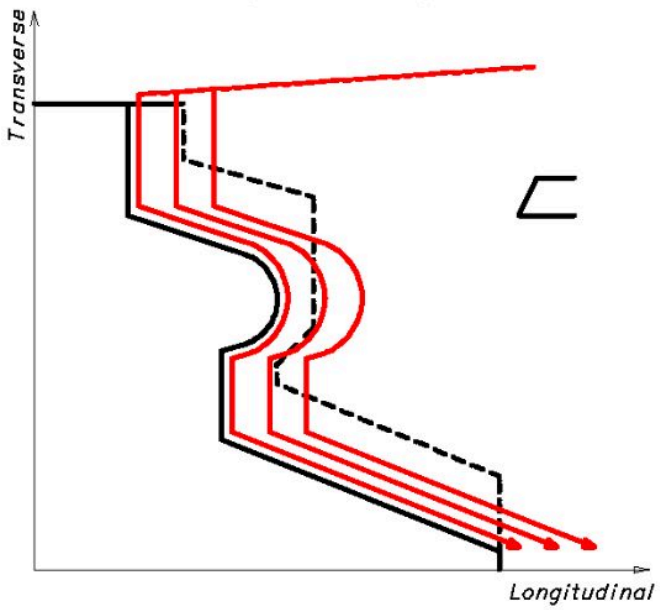
MachiningProcessing = mpRoughing
MachiningDirection = mdTransverse
MachiningStyle = msBars

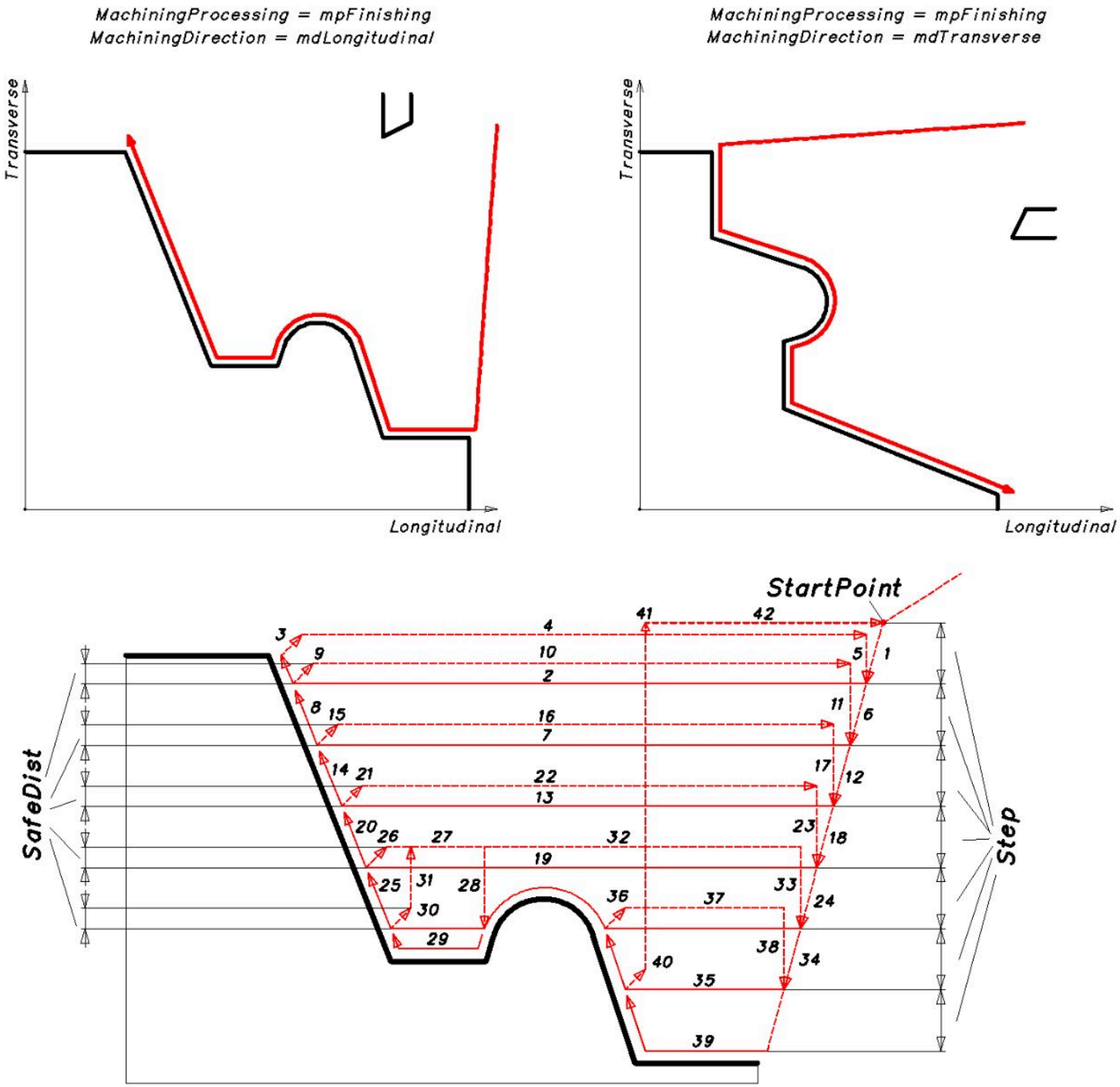


MachiningProcessing = mpRoughing
MachiningDirection = mdLongitudinal
MachiningStyle = msCopyContour



MachiningProcessing = mpRoughing
MachiningDirection = mdTransverse
MachiningStyle = msCopyContour





Purpose:

The movements in the various types of turning cycles along a specified contour generally follow the same tool path on different controllers:

1. Approach of the tool to the start point or to the point of the next pass at rapid traverse.
2. Approach to the level of the roughing pass at rapid traverse.
3. Movement to the point of the roughing pass parallel to the longitudinal axis at working feed (roughing pass).
4. Return parallel to the contour, along the contour with the finishing allowance, to the last point of the roughing pass at working feed (end of the roughing pass).

5. Return to the retraction plane at rapid traverse.
6. Returning to step 1 if the machining depth is not reached.
7. Finishing of the contour (approach to the start point at rapid traverse, finishing along the contour at working feed, retraction at rapid traverse).

Example of such a cycle: Sinumerik: CYCLE95.

The generator forms the tool path from the specified parameters: contour path, retraction plane, machining step, machining type and direction, etc. It also allows adding to any section of the tool path the features required by a particular cycle, changing the feed type or value, or skipping a tool movement prepared by the generator.

Using the generator:

We recommend creating a new turning cycle generator for each new cycle in a block of the NC program, except when the cycle is repeated in a new block. Therefore, the interface of the old generator must be released when a new one is created or when the interpreter finishes its work.

To form the tool movement commands along the cycle path, do the following:

1. Get the generator interface (INCT_CLData.CreateLatheCycleFormer, INCT_GeomCLData.CreateLatheCycleFormer).
2. Set the indexes of the longitudinal and transverse axes along which the movements are made (Set_AxisIndexes).
3. Set the cycle parameters (Set_NCProgram, Set_MachiningType, Set_MachiningDirection, Set_Overlap, etc., see the description below).
4. Assign a cycle event handler (Set_Events) if the cycle path needs to be changed.
5. Form the tool movement commands along the cycle path (MakeWorkpath).

Description of the interface methods:

Method	Method description
Set_AxisIndexes	The method sets the indexes of the longitudinal and transverse axes along which the movements are made when the cycle path is formed. Input parameters: Value (TNCLCAxisIndexes) - Transverse (integer) - index of the transverse axis, Longitudinal (integer) - index of the longitudinal axis. No output parameters.

Method	Method description
Get_AxisIndexes	The function returns the indexes of the longitudinal and transverse axes along which the movements are made when the cycle path is formed (TNCLCAxisIndexes). No input parameters.
Set_Plane	The method sets the current plane for the NC program of the contour. If the plane is not set, the current plane at the moment of getting the contour geometry is used. Input parameters: Value (TNCPlaneType) - current plane for the NC program of the contour. No output parameters.
Get_Plane	The function returns the current plane for the NC program of the contour. No input parameters.
Set_NCProgram	The method passes to the generator the interface implementing the text of the contour NC program. Input parameter: Value (INCT_Program) - interface implementing the text of the contour NC program. The contour text can be obtained using the INCT_ProgramManager.GetPartByIndex, INCT_ProgramManager.GetPartByLabels methods. No output parameters.

Method	Method description
Get_NCProgram	The function returns the interface implementing the text of the contour NC program (INCT_Program). No input parameters.
Set_RecalcContourGeometry	The method sets the flag of whether the contour NC program must be interpreted. Input parameter: Value (boolean) - False - do not interpret the contour NC program if it has already been interpreted, True - interpret the contour NC program in any case.
Get_RecalcContourGeometry	The function returns the flag of whether the contour NC program must be interpreted. No input parameters.
Set_MachiningProcessing	The method sets the machining kind. Input parameter: Value (TNCMachiningProcessing) - machining kind. mpRoughing - roughing (several passes are made taking into account the specified allowance, step, etc.); mpFinishing - finishing (only one pass along the contour without allowance is made). No output parameters.
Get_MachiningProcessing	The function returns the machining kind. No input parameters.
Set_MachiningType	The method sets the machining type.

Method	Method description
	Input parameter: Value (TNCLatheMachiningType) - machining type. mtInside - inside; mtOutside - outside. No output parameters.
Get_MachiningType	The function returns the machining type (TNCLatheMachiningType). No input parameters.
Set_MachiningDirection	The method sets the machining direction. Input parameter: Value (TNCLatheMachiningDirection) - machining direction. mdLongitudinal - longitudinal; mdTransverse - transverse. No output parameters.
Get_MachiningDirection	The function returns the machining direction (TNCLatheMachiningDirection). No input parameters.
Set_MachiningStyle	The method sets the contour machining style for roughing. Input parameters: Value (TNCLatheMachiningStyle) - machining style. msBars - at each iteration the machining is parallel to the axis (longitudinal or transverse depending on MachiningDirection); msCopyContour - at each iteration the machining is parallel to the contour (contour copy).

Method	Method description
	No output parameters.
Get_MachiningStyle	The function returns the contour machining style for roughing. No input parameters.
Set_Overlap	The method sets the overtravel type. Input parameter: Value (boolean) - overtravel type. True - with return along the contour, False - with direct return. No output parameters.
Get_Overlap	The function returns the overtravel type (boolean). No input parameters.
Set_AxialStock	The method sets the axial allowance on the contour. Input parameter: Value (real number) - axial allowance on the contour. No output parameters.
Get_AxialStock	The function returns the axial allowance on the contour (real number). No input parameters.

Method	Method description
Set_RadialStock	The method sets the radial allowance on the contour. Input parameter: Value (real number) - radial allowance on the contour. No output parameters.
Get_RadialStock	The function returns the radial allowance on the contour (real number). No input parameters.
Set_ProfileStock	The method sets the equidistant allowance on the contour. Input parameter: Value (real number) - equidistant allowance on the contour. No output parameters.
Get_ProfileStock	The function returns the equidistant allowance on the contour (real number). No input parameters.
Set_StartPointMode	The method sets the mode of setting the contour start point. Input parameters: Value (TNCLatheStartPointMode) - mode of setting the contour start point. spmCalcPoint - the start point is calculated by the cycle; spmSetPoint - the start point is set (see StartPoint). No output parameters.
Get_StartPointMode	The function returns the mode of setting the contour start point.

Method	Method description
	No input parameters.
Set_StartPoint	The method sets the cycle start point (for the mode StartPointMode = spmSetPoint). Input parameters: Value (TNC2DPoint) - cycle start point. No output parameters.
Get_StartPoint	The function returns the cycle start point. No input parameters.
Set_EachReliefcutSeparately	The method sets the flag of machining the relief cuts. Input parameter: Value (boolean) - True - each relief cut is machined separately, False - all relief cuts are machined in one pass. No output parameters.
Get_EachReliefcutSeparately	The function returns the flag of machining the relief cuts. No input parameters.
Set_DoReliefcutsInReverseOrder	The method sets the flag of the order of machining the relief cuts. Input parameters: Value (boolean) - flag of the order of machining the relief cuts. True - machine the relief cuts in reverse order. False - machine the relief cuts in direct order. No output parameters.

Method	Method description
Get_DoReliefcutsInReverseOrder	The function returns the flag of the order of machining the relief cuts. No input parameters.
Set_StepType	The method sets the type of calculating the roughing step. Input parameter: Value (TNCLatheStepType) - type of calculating the roughing step. stConst - at each pass the machining depth equals the machining step (Set_Step); stCalcEvenly - at each stage (relief cut) the generator calculates the number of steps from the machining step (Set_Step) and machines in equal steps (depth of cut / number of steps). No output parameters.
Get_StepType	The function returns the type of the roughing step (TNCLatheStepType). No input parameters.
Set_Step	The method sets the machining step. Input parameter: Value (real number) - machining step. No output parameters.
Get_Step	The function returns the machining step (real number). No input parameters.
Set_SafeDist	The method sets the retraction for the return (the side retraction before the tool exits).

Method	Method description
	Input parameter: Value (real number) - retraction value. No output parameters.
Get_SafeDist	The function returns the retraction for the return (real number). No input parameters.
Set_LengthChipBreaking	The method sets the travel length for chip breaking. Input parameter: Value (real number) - travel length for chip breaking. No output parameters.
Get_LengthChipBreaking	The function returns the travel length for chip breaking (real number). No input parameters.
Set_Feedrate	The method sets the working feed. Input parameter: Value (real number) - working feed. No output parameters.
Get_Feedrate	The function returns the working feed. No input parameters.

Method	Method description
Set_MPM	The method sets the working feed units. Input parameter: Value (boolean) - working feed units. True - mm/min, False - mm/rev. No output parameters.
Get_MPM	The function returns the working feed units. No input parameters.
Set_UseContourFeedrate	The method sets the flag of using the feed set in the NC program of the contour. Input parameters: Value (boolean) - flag of using the feed. True - the feed set in the contour NC program is used when machining the same contour sections. False - the feed set in the contour is ignored. No output parameters.
Get_UseContourFeedrate	The function returns the flag of using the feed set in the NC program of the contour. No input parameters.
Set_Events	The method assigns events that control the cycle behavior. Events allow adding to any section of the tool path the features required by a particular cycle (dwell, oriented spindle stop, etc.), changing the feed type or value, or skipping a tool movement prepared by the generator. Input parameter: Value (interface) - object implementing the interfaces INCT_LatheCycleEvents (described in detail in the article Turning grooving cycle) and INCT_LatheCycleArcEvents (described below), as

Method	Method description
	well as INCT_LatheStockRemovalCycleEvents, INCT_LatheStockRemovalCycleEvents2, INCT_LatheStockRemovalCycleEvents3 (described below). The generator requests all these interfaces from the passed object; the object may implement only the ones it needs. No output parameters.
Get_Events	The function returns the object implementing the events that is linked to the cycle generator (INCT_LatheCycleEvents, INCT_LatheCycleArcEvents). No input parameters.
Get_SignLongitudinal	The function returns the sign of the longitudinal machining (integer). 1 - the direction coincides with the longitudinal axis, 0 - the direction is not defined, -1 - the direction is opposite to the longitudinal axis. No input parameters.
Get_SignTransverse	The function returns the sign of the transverse machining (integer). 1 - the direction coincides with the transverse axis, 0 - the direction is not defined, -1 - the direction is opposite to the transverse axis. No input parameters.
MakeWorkpath	The method forms the cycle path from the specified parameters. The tool path is formed as follows: 1. Getting the current state of the interpreter (method INCT_InterpreterState.Save) if the

Method	Method description
	interpreter supports the INCT_InterpreterState interface (described below). 2. Creating and initializing a new interpreter instance. 3. Restoring the current state for the new interpreter instance (INCT_InterpreterState.Load). 4. Running the new interpreter on the NC program of the cycle contour text to form the contour path. 5. After the new interpreter finishes, the tool path is formed from the specified parameters and the obtained contour path.

INCT_LatheStockRemovalCycleFormer2 extends the capabilities of the turning cycle generator along a specified contour.

Method	Method description
Set_PassReturnMethod	The method sets the order of the return to the pass entry point. Input parameter: Value (TNCPassReturnMethod) - order of the return to the pass entry point. prmStepwise - at each cycle iteration the return is made first normal to the feed axis and then normal to the machining axis; prmDirect - at each cycle iteration the return is made directly to the entry point of the next pass. No output parameters.
Get_PassReturnMethod	The function returns the order of the return to the pass entry point (TNCPassReturnMethod). No input parameters.
Set_CutDepthCalcMethod	The method sets the way the depth of cut is determined. Input parameter: Value (TNCCutDepthCalcMethod) - way of determining the depth of cut. cdcByStep - the depth of each cut is equal to or (at the last

Method	Method description
	cycle iteration) less than the specified machining step; cdcByCount - the depth of each cut is calculated from the specified number of cycle repetitions. No output parameters.
Get_CutDepthCalcMethod	The function returns the way the depth of cut is determined (TNCCutDepthCalcMethod). No input parameters.
Set_Count	The method sets the number of cycle repetitions (for the way of determining the depth of cut by the number of repetitions, cdcByCount). Input parameter: Value (integer) - number of repetitions. No output parameters.
Get_Count	The function returns the number of cycle repetitions (for the way of determining the depth of cut by the number of repetitions, cdcByCount). No input parameters.
Set_Offset	The method sets the offset of the contour machining start point at each pass. Input parameter: Value (real number) - offset of the contour start point. No output parameters.
Get_Offset	The function returns the offset of the contour machining start point at each pass. No input parameters.
Get_ContourStartPoint	The function returns the coordinates of the contour start point taking into account the allowance set in the generator (TNC2DPoint).
Get_ContourEndPoint	The function returns the coordinates of the contour end point taking into account the allowance set in the generator (TNC2DPoint).

Method	Method description
Set_AxialStockDependsOnContour	The method sets how the axial and radial allowances (Set_AxialStock, Set_RadialStock) are taken into account. Input parameter: Value (boolean). True (default) - the allowances are taken as absolute values, and the shift direction is determined by the contour and the machining direction; False - the allowances are treated as shifts along the axes with the specified sign. No output parameters.
Get_AxialStockDependsOnContour	The function returns how the axial and radial allowances are taken into account (boolean). No input parameters.
Set_StartPointContainsSafeDist	The method sets whether the retraction value (Set_SafeDist) is taken into account when the cycle start point is calculated (for the mode StartPointMode = spmCalcPoint). Input parameter: Value (boolean). True (default) - the start point is moved away from the contour by the allowance and the retraction value; False - by the allowance only. No output parameters.
Get_StartPointContainsSafeDist	The function returns whether the retraction value is taken into account when the cycle start point is calculated (boolean). No input parameters.

INCT_LatheCycleArcEvents - an interface implemented in the interpreter that allows controlling the behavior of a turning cycle when arcs are formed. The interface consists of one event method executed when the tool moves along an arc inside the cycle path generator. With this method, the features required by a particular cycle (dwell, oriented spindle stop, etc.) can be added to any section of the tool path, the feed type can be changed, or a tool movement prepared by the generator can be skipped.

Description of the interface method:

Method	Method description	Description of use
OnArc	The method is executed before each tool movement along an arc when the cycle path is formed. Input parameters: MoveType (TNCLatheCycleMoveType) - movement type, see Turning grooving cycle; Input and output parameters: Center (TNC2DPoint) - coordinates of the arc center; Point (TNC2DPoint) - coordinates of the arc end point; Radius (real number) - arc radius; Feed (TNCFeedType) - feed type; Handled (boolean) - flag of whether the movement has been made inside the implementation (True - the movement has been made, False - it has not). If Handled is True after the OnArc call, the movement has been made inside the OnArc implementation, and the cycle generator does not add an arc movement command to the tool path. If Handled is False after the OnArc call, the generator considers that the arc movement has not been made inside the OnArc implementation and adds to the tool path a command to move along an arc to the point Point with the feed type Feed.	To override the arc movement or the feed, or to skip the movement. For example, to change the feed type for the pass along the contour, in the OnArc implementation change the returned Feed parameter under the condition MoveType = lmOverlapContour (for the feed type, see OutStandardFeed). To move the tool to a point other than the one calculated by the cycle, change the Center, Point, Radius parameters returned by OnArc. If the movement to the point calculated by the cycle is not needed, set the Handled parameter to True inside the OnArc implementation.

INCT_LatheStockRemovalCycleEvents - an interface implemented in the interpreter that allows controlling the behavior of the turning stock removal cycle. The interface consists of one event method executed after the contour geometry is obtained and before the cycle path is calculated. With this method and the INCT_InterpreterState interface, the contour parameters can be obtained from the NC program text and passed to the cycle path generator for correct forming (for example, the machining direction if it is set by a command in the contour path).

Description of the interface methods:

Method	Method description	Description of use
OnCalcContourGeometry	The method is executed after the contour geometry is obtained and before the generator calculates the cycle path. No parameters.	With this method and the INCT_InterpreterState interface, the contour parameters can be obtained from the NC program text and passed to the cycle path generator for correct forming (for example, the machining direction if it is set by a command in the contour path).

INCT_LatheStockRemovalCycleEvents2 - an interface implemented in the interpreter that allows controlling the behavior of the turning stock removal cycle. It extends the INCT_LatheStockRemovalCycleEvents interface.

Description of the interface methods:

Method	Method description	Description of use
OnCalcNextStep	The method is executed when the tool path is calculated for each pass. It is intended for changing the step and the offset used to calculate the next start point of the pass along the contour and the return point after the pass along the contour for the current iteration. Input parameters: Iteration (integer) - pass number starting from 1; PointType	With this method, your own algorithm of calculating the start points of the passes along the contour and the return points after them can be set for each pass by operating the step and the offset at the current iteration.

Method	Method description	Description of use
	(TNCLSRPointType) - type of the contour point: ptProfilePoint - start point of the pass along the contour; ptReturnPoint - return point after the pass along the contour. Input and output parameters: Step (real number) - step for calculating the current pass (infeed axis); Offset (real number) - offset for calculating the current pass (the other axis, perpendicular to the infeed axis).	
OnCalcNextPoint	The method is executed when the tool path is calculated for each pass. It is intended for changing the coordinates of the start point of the pass along the contour and of the return point after the pass, calculated for the current iteration. Input parameters: Iteration (integer) - pass number starting from 1; PointType (TNCLSRPointType) - type of the contour point: ptProfilePoint - start point of the pass along the contour; ptReturnPoint - return point after the pass along the contour. Input and output parameters: Point (TNC2DPoint) - point coordinates.	With this method, your own algorithm of calculating the start points of the passes along the contour and the return points after them can be set for each pass by operating the point coordinates at the current iteration.

INCT_LatheStockRemovalCycleEvents3 - an interface implemented in the interpreter that allows controlling the behavior of the turning cycle. It extends the INCT_LatheStockRemovalCycleEvents2 interface.

Description of the interface methods:

Method	Method description	Description of use
OnCalcSafeCut	The method is executed when the tool path is calculated. It is intended for changing the retraction values for the return calculated inside the cycle (the side retraction before the tool exits). Input and output parameters: Point (TNC2DPoint) - point coordinates (relative).	Before the method is called, the cycle calculates the retraction from Set_SafeDist along both axes away from the contour. With this method, your own retraction value can be set, for example, retraction along one axis only. The new value is used for the approach to the machining start, for the return to the safe distance after a pass and for calculating the levels of the passes.

INCT_InterpreterState - an interface implemented by the interpreter (by the same object that implements the INCT_Interpreter interface) that allows saving the current state of the interpreter and loading it into a newly created interpreter instance for correct forming of the contour path.

For example, the current plane must be known to form the contour path. For this, the interpreter must create an object, save the current plane in it (method Save), and, when a new interpreter instance is created, restore the current plane from the object passed to it (method Load).

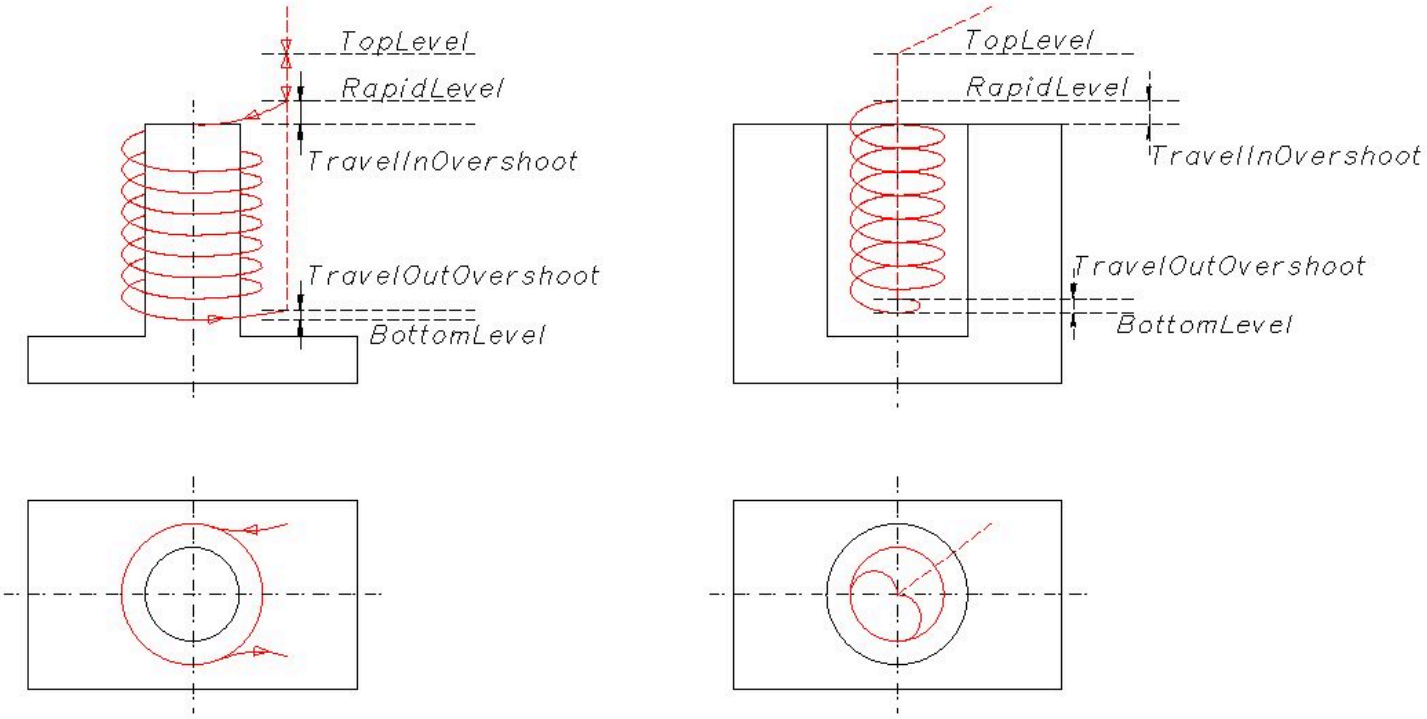
Description of the interface methods:

Method	Method description	Description of use
Save	The method must create an object implementing its own interface, save the data of the current state in the object and pass the interface of the object in the output parameter. No input parameters. Output parameter: State - IUnknown interface.	To form the contour path, the current plane set in the NC program text must be known. In the implementation of the Save method, save the current plane in the object created inside the method. The CAM system kernel passes the reference to the object to the new interpreter instance by calling the Load method. In its implementation,

Method	Method description	Description of use
Load	The method must get its own interface from the object passed in the input parameter and get the current state of the interpreter required to form the contour path. Input parameter: State - IUnknown interface. No output parameters.	get the current plane from the passed object.

Thread milling cycle

INCT_ThreadMillingCycleFormer - a CAM system interface that simplifies forming the movements of a thread milling cycle. The object implementing this interface is returned by the function INCT_CLData.CreateMillingCycleFormer when it is called with the mctThreadMilling parameter.



Purpose:

The movements in the various types of thread milling cycles generally follow the same tool path on different controllers:

1. Approach of the tool to the initial level at rapid traverse.
2. Approach of the tool to the travel-in overshoot level at rapid traverse.
3. Approach along an arc or a helix to the milling start point at working feed (overshoot).
4. Thread milling with the specified parameters (center, diameter, pitch) to the milling end level.
5. Retraction along an arc or a helix at working feed at the travel-out overshoot level (overshoot).
6. Return to the initial level at rapid traverse.

Example of such a cycle: Sinumerik: CYCLE90.

The generator forms the tool path from the specified parameters. It also allows adding to any section of the tool path the features required by a particular cycle, changing the feed type or value, or skipping a tool movement prepared by the generator.

Using the generator:

We recommend creating a new generator for each new cycle in a block of the NC program. Therefore, the interface of the old generator must be released when a new one is created or when the interpreter finishes its work.

To form the tool movement commands along the cycle path, do the following:

1. Get the generator interface (INCT_CLData.CreateMillingCycleFormer).
2. Set the indexes of the axes along which the movements are made (Set_AxisIndexes).
3. Set the current plane (Set_Plane).
4. Set the working feed (Set_Feedrate).
5. Set the thread type (Set_ThreadType).
6. Set the thread direction (Set_ThreadDirection).
7. Set the thread pitch (Set_ThreadStep).
8. Set the thread diameter (Set_Diameter).
9. Set the thread center point (Set_CenterPoint).
10. Set the approach level (Set_TopLevel).
11. Set the safe plane level (Set_RapidLevel).
12. Set the milling end level (Set_BottomLevel).
13. Set the travel-in overshoot level (Set_TravelInOvershoot).
14. Set the travel-out overshoot level (Set_TravelOutOvershoot).
15. Assign a cycle event handler (Set_Events) if the cycle path needs to be changed.
16. Form the tool movement commands along the cycle path (MakeWorkpath).

Description of the interface methods:

Method	Method description
Set_AxisIndexes	The method sets the indexes of the X, Y, Z axes along which the movements are made when the cycle path is formed. Input parameters: Value (TNCDCAxisIndexes) - X (integer) - index of the X axis, Y (integer) - index of the Y axis, Z (integer) - index of the Z axis. No output parameters.
Get_AxisIndexes	The function returns the indexes of the X, Y, Z axes along which the movements are made when the cycle path is formed (TNCDCAxisIndexes). No input parameters.

Method	Method description
Set_Plane	The method sets the current plane. Input parameter: Value (TNCPlaneType) - current plane. ptXY - XY plane, the thread is milled along the Z axis. ptYZ - YZ plane, the thread is milled along the X axis. ptZX - ZX plane, the thread is milled along the Y axis. No output parameters.
Get_Plane	The function returns the current plane (TNCPlaneType). No input parameters.
Set_ThreadType	The method sets the thread type. Input parameter: Value (TNCMillingThreadType) - thread type. mttInternal - internal; mttExternal - external. No output parameters.
Get_ThreadType	The function returns the thread type (TNCMillingThreadType). No input parameters.
Set_ThreadDirection	The method sets the thread direction. Input parameter: Value (TNCCircularMode) - thread direction. cmCW - clockwise; cmCCW - counterclockwise. No output parameters.
Get_ThreadDirection	The function returns the thread direction (TNCCircularMode). No input parameters.
Set_ThreadStep	The method sets the thread pitch. Input parameter: Value (real number) - thread

Method	Method description
	pitch. No output parameters.
Get_ThreadStep	The function returns the thread pitch (real number). No input parameters.
Set_Diameter	The method sets the thread diameter. Input parameter: Value (real number) - thread diameter. No output parameters.
Get_Diameter	The function returns the thread diameter (real number). No input parameters.
Set_CenterPoint	The method sets the thread center point. Input parameter: Value (TNC2DPoint) - thread center point. No output parameters.
Get_CenterPoint	The function returns the thread center point (TNC2DPoint). No input parameters.
Set_TopLevel	The method sets the approach level. Input parameter: Value (real number) - approach level. No output parameters.
Get_TopLevel	The function returns the approach level (real number). No input parameters.
Set_RapidLevel	The method sets the safe plane level. Input parameter: Value (real number) - safe plane level. No output parameters.
Get_RapidLevel	The function returns the safe plane level (real number).

Method	Method description
	No input parameters.
Set_BottomLevel	The method sets the milling end level. Input parameter: Value (real number) - milling end level. No output parameters.
Get_BottomLevel	The function returns the milling end level (real number). No input parameters.
Set_TravelInOvershoot	The method sets the travel-in overshoot level. Input parameter: Value (real number) - travel-in overshoot level. No output parameters.
Get_TravelInOvershoot	The function returns the travel-in overshoot level (real number). No input parameters.
Set_TravelOutOvershoot	The method sets the travel-out overshoot level. Input parameter: Value (real number) - travel-out overshoot. No output parameters.
Get_TravelOutOvershoot	The function returns the travel-out overshoot level (real number). No input parameters.
Set_Feedrate	The method sets the working feed. Input parameter: Value (real number) - working feed. No output parameters.
Get_Feedrate	The function returns the working feed (real number). No input parameters.
Set_MPM	The method sets the working feed units. Input parameter: Value (boolean) - working feed

Method	Method description
	units. True - mm/min, False - mm/rev. No output parameters.
Get_MPM	The function returns the working feed units (boolean). No input parameters.
Set_Events	The method assigns events that control the cycle behavior. Events allow adding to any section of the tool path the features required by a particular cycle, changing the feed type or value, or skipping a tool movement prepared by the generator. Input parameter: Value (INCT_MillingCycleEvents) - object implementing the events. See the detailed description below. No output parameters.
Get_Events	The function returns the object implementing the events that is linked to the cycle generator (INCT_MillingCycleEvents). No input parameters.
MakeWorkpath	The method forms the cycle path from the specified parameters.

INCT_MillingCycleEvents - an interface implemented in the interpreter that allows controlling the behavior of the thread milling cycle. The interface is a set of event methods executed during the tool movements inside the cycle path generator. With these methods, the features required by a particular cycle (dwell, oriented spindle stop, etc.) can be added to any section of the tool path, the feed type can be changed, or a tool movement prepared by the generator can be skipped.

If events are assigned to the cycle path generator, every tool movement inside the generator consists of 3 stages:

1. The INCT_MillingCycleEvents.BeforeMove method is executed. It receives the values set by the cycle: movement type, coordinates of the movement point, feed type.
2. The INCT_MillingCycleEvents.OnMove or OnArc method is executed, depending on the movement type: linear or along an arc.
   OnMove receives: movement type, coordinates of the movement point, feed type and the flag of whether the movement has been made. All parameters except the movement type can be changed in the method implementation. Depending on the flag set in OnMove, the tool moves to the specified

point with the specified feed, or the movement is not made.
OnArc receives: movement type, coordinates of the arc center and of the movement point, radius, feed type and the flag of whether the movement has been made. All parameters except the movement type can be changed in the method implementation. Depending on the flag set in OnArc, the tool moves to the specified point with the specified feed, or the movement is not made.

3. The INCT_MillingCycleEvents.AfterMove method is executed. It receives the values returned by OnMove or OnArc: movement type, coordinates of the movement point, feed type.

Description of the interface methods:

Method	Method description	Description of use
BeforeMove	The method is executed before each tool movement when the cycle path is formed. Input parameters: MoveType (TNCMillingCycleMoveType) - movement type, see the description below; Point (TNC3DPoint) - coordinates of the movement point; Feed (TNCFeedType) - feed type. mmMoveToTopLevel - movement to the start point at the TopLevel level; mmMoveToRapidLevel - movement to the approach start level; mmTravelInMovement - approach to the thread along an arc; mmThreadMilling - thread milling; mmTravelOutMovement - retraction from the thread to the center along an arc; mmReturnToTopLevel - return to TopLevel. No output parameters.	If some commands must be added before a movement, add them to the method implementation. For example, to turn on the machine coolant before thread milling, call INCT_CLData.AddCoolant with the appropriate parameters in the BeforeMove implementation under the condition MoveType = mmMoveToTopLevel.

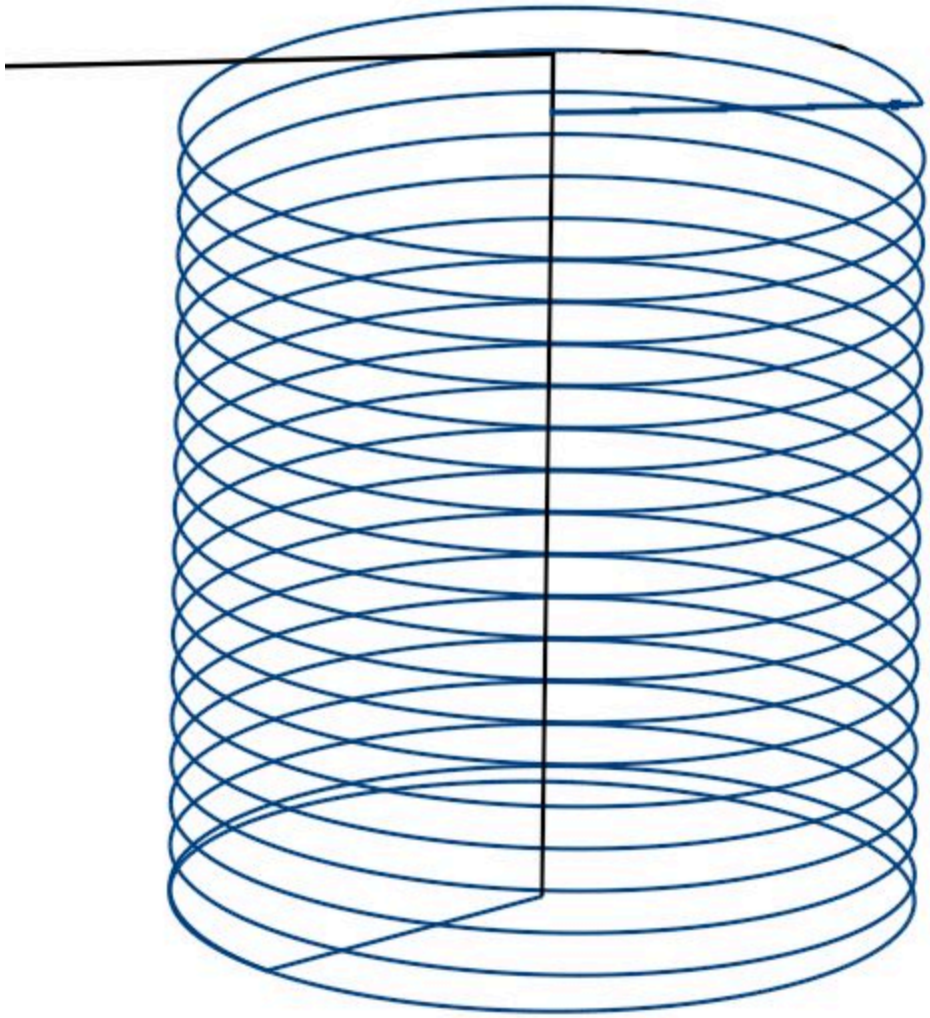
Method	Method description	Description of use
OnMove	The method is executed immediately before each linear tool movement when the cycle path is formed. Input parameters: MoveType (TNCMillingCycleMoveType) - movement type, see the description above; Input and output parameters: Point (TNC3DPoint) - coordinates of the movement point; Feed (TNCFeedType) - feed type; Handled (boolean) - flag of whether the movement has been made inside the implementation (True - the movement has been made, False - it has not). If Handled is True after the OnMove call, the movement has been made inside the OnMove implementation, and the cycle generator does not add a movement command to the tool path. If Handled is False after the OnMove call, the generator considers that the movement has not been made inside the OnMove implementation and adds to the tool path a command to move to the point Point with the feed type Feed.	To override the movement point or the feed, or to skip the movement. For example, to switch to rapid traverse when returning to the initial level after milling, in the OnMove implementation set the returned Feed parameter to ffRapid under the condition MoveType = mmReturnToTopLevel (for the feed type, see OutStandardFeed). To move the tool to a point other than the one calculated by the cycle, change the Point parameter returned by OnMove. If the movement to the point calculated by the cycle is not needed, set the Handled parameter to True inside the OnMove implementation.
OnArc	The method is executed immediately before each tool movement along an arc when the cycle path is formed. Input parameters: MoveType	To override the movement point or the feed, or to skip the movement.

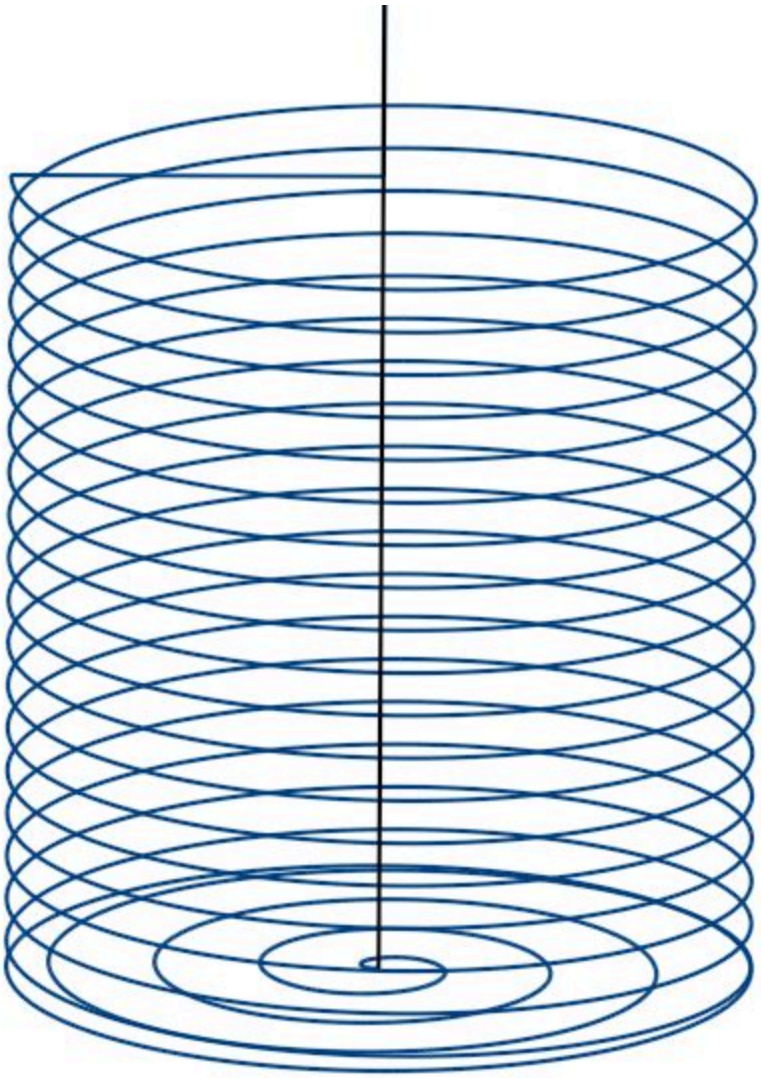
Method	Method description	Description of use
	(TNCMillingCycleMoveType) - movement type, see the description above; Input and output parameters: Center, Point (TNC3DPoint) - coordinates of the center and of the movement point; Radius (real number) - arc radius; Feed (TNCFeedType) - feed type; Handled (boolean) - flag of whether the movement has been made inside the implementation (True - the movement has been made, False - it has not). If Handled is True after the OnArc call, the movement has been made inside the OnArc implementation, and the cycle generator does not add a movement command to the tool path. If Handled is False after the OnArc call, the generator considers that the movement has not been made inside the OnArc implementation and adds to the tool path a command to move along an arc to the point Point with the feed type Feed.	
AfterMove	The method is executed after each tool movement when the cycle path is formed, even if the movement has been canceled in OnMove or OnArc. Input parameters: MoveType (TNCMillingCycleMoveType) - movement type, see the	If some commands must be added after a movement, add them to the method implementation.

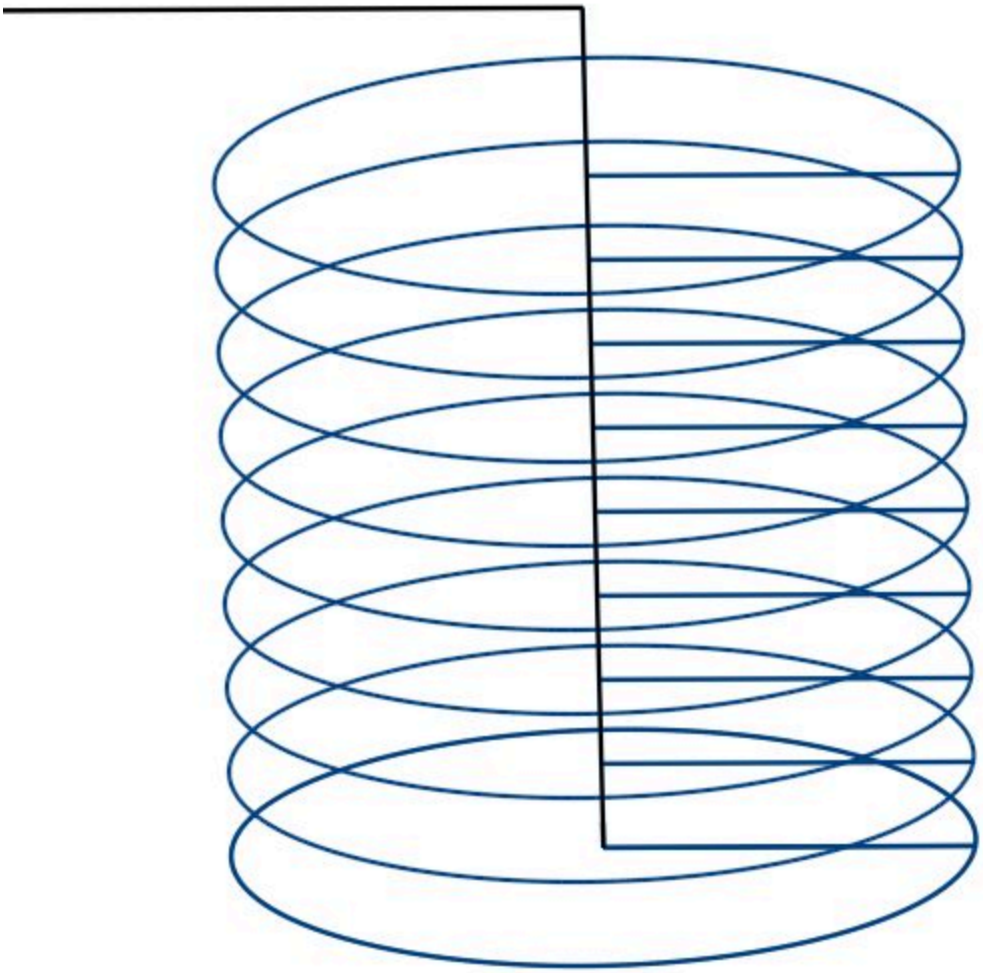
Method	Method description	Description of use
	description above; Point (TNC3DPoint) - coordinates of the movement point; Feed (TNCFeedType) - feed type. No output parameters.	

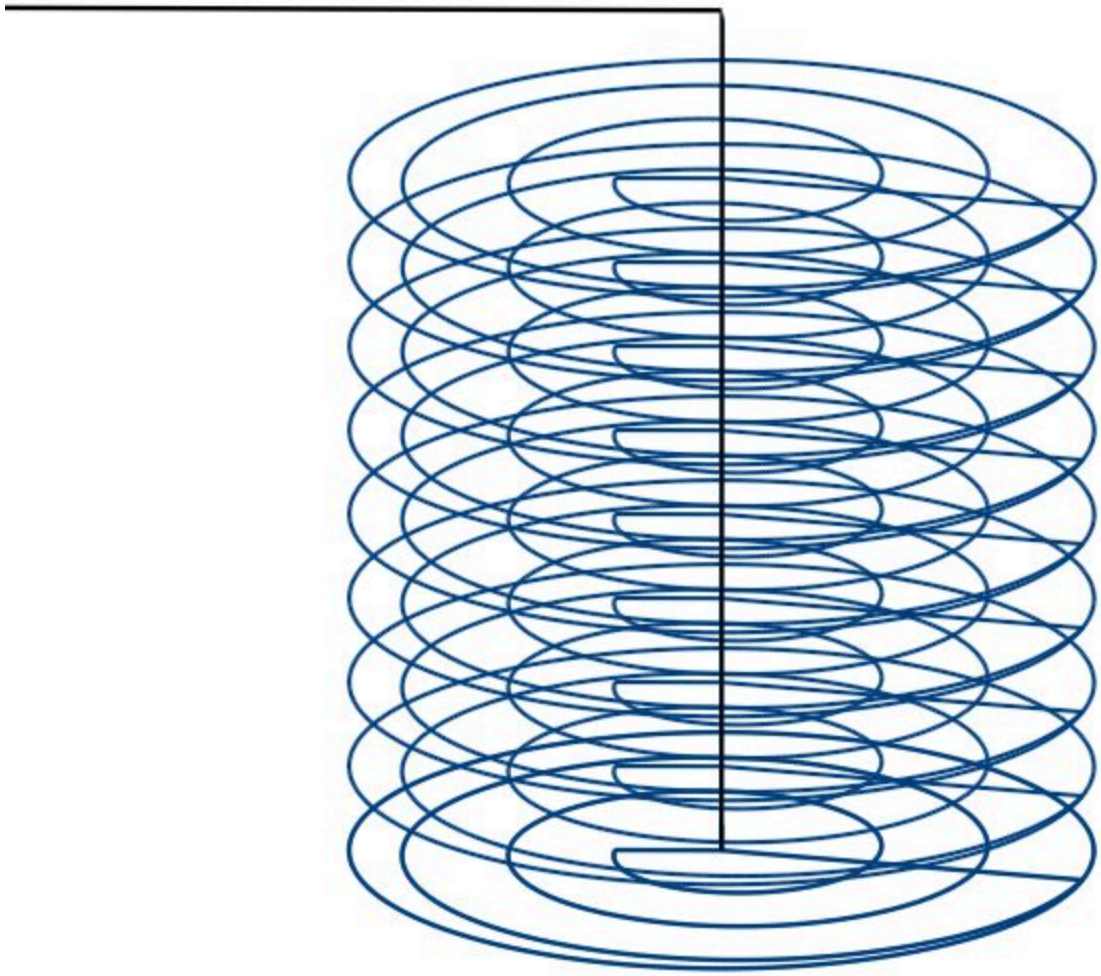
Circular pocket milling cycle

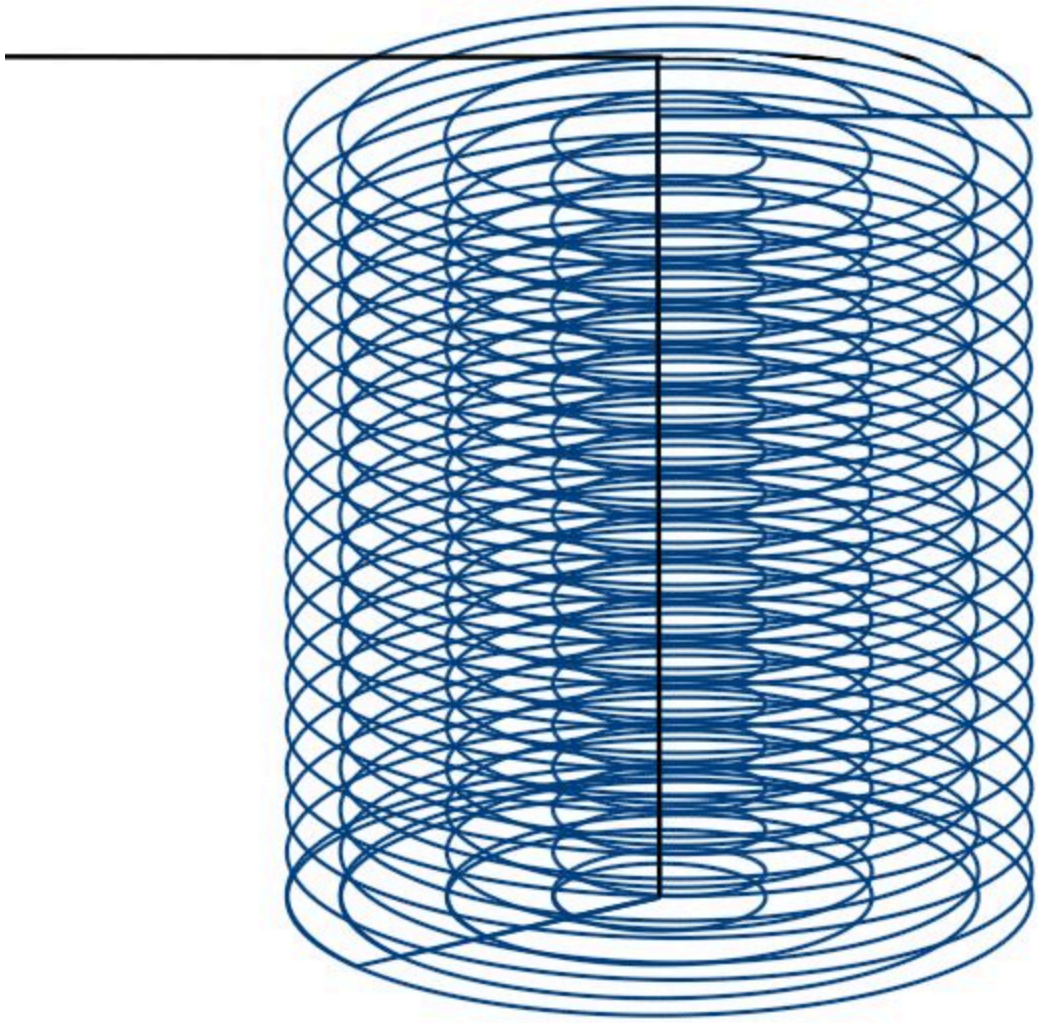
INCT_PocketCycleFormer - a CAM system interface that simplifies forming the movements of a circular pocket milling cycle. The object implementing this interface is returned by the method INCT_CLData_PocketCycleFormer.CreatePocketCycleFormer.

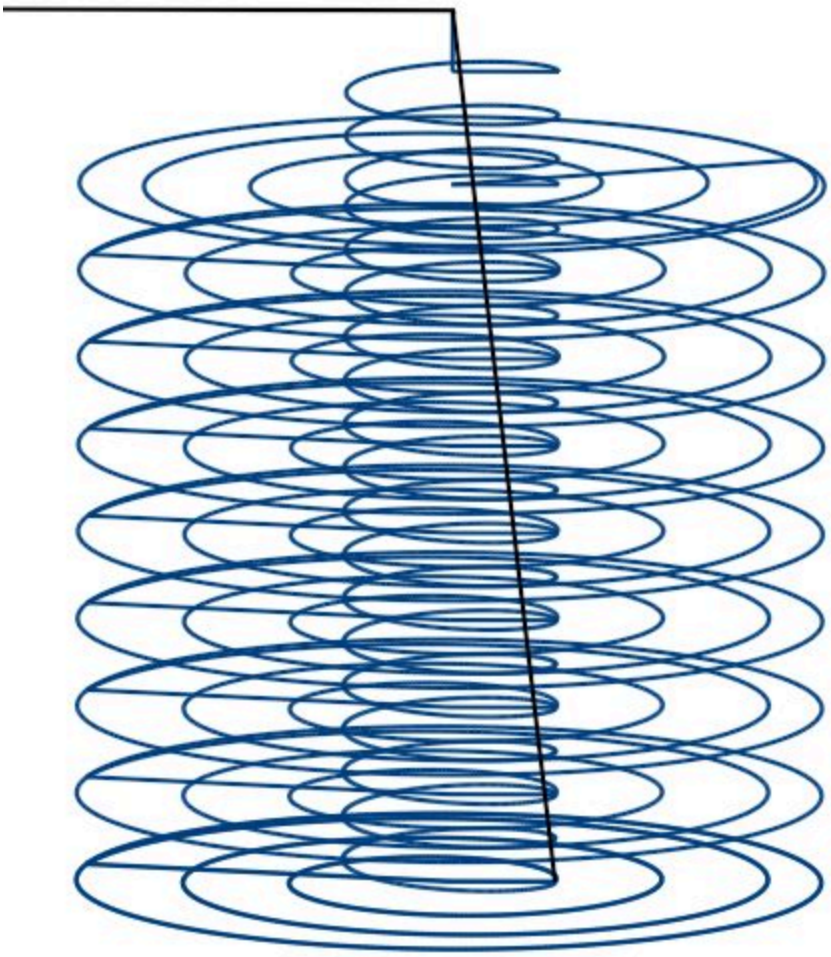












Purpose:

The movements in the various types of circular pocket milling cycles generally follow the same tool path on different controllers:

1. Approach to the pocket center at the initial level.
2. Approach to the safe distance.
3. Approach to the base point.
4. Plunging (helical or linear) to the specified depth of cut or to the full depth.
5. Retraction to the safe distance and repetition down to the pocket depth; in finishing, the tool goes directly to the pocket depth.
6. Return to the pocket center at the level where the machining started, here at the initial level.

Example of such a cycle: Sinumerik: POCKET4.

The generator forms the tool path from the specified parameters. It also allows adding to any section of the tool path the features required by a particular cycle, changing the feed type or value, or skipping a tool movement prepared by the generator.

Using the generator:

We recommend creating a new generator for each new cycle in a block of the NC program. Therefore, the interface of the old generator must be released when a new one is created or when the interpreter finishes its work.

1. Get the generator interface (INCT_CLData_PocketCycleFormer.CreatePocketCycleFormer).
2. Set the indexes of the axes along which the movements are made (Set_AxisIndexes).
3. Set the current plane (Set_Plane).
4. Set the pocket center (Set_HoleCenter).
5. Set the safe distance (Set_SafeLevel).
6. Set the pocket start (Set_BaseLevel).
7. Set the pocket bottom level (Set_BottomLevel).
8. Set the pocket diameter (Set_Diameter).
9. Set the flat spiral step (Set_FlatSpiralStep).
10. Set the flat spiral step type (Set_FlatSpiralStepType).
11. Set the flat spiral direction (Set_FlatSpiralDirection).
12. Set the helix direction (Set_SpiralDirection).
13. Set the helix pitch (Set_SpiralStep).
14. Set the machining type (Set_TypeOfHoleProcessing).
15. Set the depth of cut per pass (Set_Degression).
16. Set the plunge radius (Set_PlungeRadius).
17. Set the allowance at the pocket edge (Set_EdgeStock).
18. Set the allowance at the pocket bottom (Set_BottomStock).
19. Assign a cycle event handler (Set_Events) if the cycle path needs to be changed.
20. Form the tool movement commands along the cycle path (MakeWorkpath).

Description of the interface methods:

Method	Method description
Set_AxisIndexes	The method sets the indexes of the X, Y, Z axes along which the movements are made when the cycle path is formed. Input parameters: Value (TNCDCAxisIndexes) - X (integer) - index of the X axis, Y (integer) - index of the Y axis, Z (integer) - index of the Z axis. No output parameters.

Method	Method description
Get_AxisIndexes	The function returns the indexes of the X, Y, Z axes along which the movements are made when the cycle path is formed (TNCDCAxisIndexes). No input parameters.
Set_Plane	The method sets the current plane. Input parameter: Value (TNCPlaneType) - current plane. ptXY - XY plane, the pocket is machined along the Z axis; ptYZ - YZ plane, the pocket is machined along the X axis; ptZX - ZX plane, the pocket is machined along the Y axis. No output parameters.
Get_Plane	The function returns the current plane (TNCPlaneType). No input parameters.
Set_HoleCenter	The method sets the pocket center point. Input parameter: Value (TNC3DPoint) - pocket center point. No output parameters.
Get_HoleCenter	The function returns the pocket center point (TNC3DPoint). No input parameters.
Set_SafeLevel	The method sets the safe distance. Input parameter: Value (real number) - safe distance. No output parameters.
Get_SafeLevel	The function returns the safe distance (real number). No input parameters.
Set_BaseLevel	The method sets the pocket start. Input parameter: Value (real number) - pocket

Method	Method description
	start. No output parameters.
Get_BaseLevel	The function returns the pocket start (real number). No input parameters.
Set_Diameter	The method sets the pocket diameter. Input parameter: Value (real number) - pocket diameter. No output parameters.
Get_Diameter	The function returns the pocket diameter (real number). No input parameters.
Set_FlatSpiralStep	The method sets the flat spiral step. Input parameter: Value (real number) - flat spiral step. No output parameters.
Get_FlatSpiralStep	The function returns the flat spiral step (real number). No input parameters.
Set_BottomLevel	The method sets the pocket bottom level. Input parameter: Value (real number) - pocket bottom level. No output parameters.
Get_BottomLevel	The function returns the pocket bottom level (real number). No input parameters.
Set_FlatSpiralStepType	The method sets the flat spiral step type. Input parameter: Value (TNCFlatSpiralStepType) - flat spiral step type. fsByMillimeters - the spiral step is calculated in millimeters; fsByPersentOfToolDiameter - the spiral step is

Method	Method description
	calculated as a percentage of the tool diameter. No output parameters.
Get_FlatSpiralStepType	The function returns the flat spiral step type (TNCFlatSpiralStepType). No input parameters.
Set_FlatSpiralDirection	The method sets the flat spiral direction. Input parameter: Value (TNCSpiralDirection) - flat spiral direction. sdCW - clockwise; sdCCW - counterclockwise. No output parameters.
Get_FlatSpiralDirection	The function returns the flat spiral direction (TNCSpiralDirection). No input parameters.
Set_SpiralDirection	The method sets the helix direction. Input parameter: Value (TNCSpiralDirection) - helix direction. sdCW - clockwise; sdCCW - counterclockwise. No output parameters.
Get_SpiralDirection	The function returns the helix direction (TNCSpiralDirection). No input parameters.
Set_TypeOfHoleProcessing	The method sets the pocket machining type. Input parameter: Value (TNCTypeOfProcessing) - pocket machining type. tpHelicalPlungeRoughing - roughing, helical plunge, machining with a flat spiral; tpLinearPlungeRoughing - roughing, linear plunge, machining with a flat spiral; tpOnlyHelicalRoughing - roughing, fully helical machining; tpOnlyHelicalFinishing - finishing, helical machining along the edge;

Method	Method description
	tpOnlyCircularFinishing - finishing, machining along the edge with circles; tpOnlyHelicalFinishingAndBottom - finishing, helical machining along the edge, machining of the bottom with a flat spiral. No output parameters.
Get_TypeOfHoleProcessing	The function returns the pocket machining type (TNCTypeOfProcessing). No input parameters.
Set_SpiralStep	The method sets the helix pitch. Input parameter: Value (real number) - helix pitch. No output parameters.
Get_SpiralStep	The function returns the helix pitch (real number). No input parameters.
Set_Degression	The method sets the depth of cut per pass. Input parameter: Value (real number) - depth of cut per pass. No output parameters.
Get_Degression	The function returns the depth of cut per pass (real number). No input parameters.
Set_DepthOfCut	The method stores a depth of cut value. The generator does not use it when forming the tool path; the depth of cut per pass is set by Set_Degression. Input parameter: Value (real number) - depth of cut. No output parameters.
Get_DepthOfCut	The function returns the value stored by Set_DepthOfCut (real number). No input parameters.
Set_PlungeRadius	The method sets the plunge radius: the lateral offset from the pocket center before helical

Method	Method description
	plunging (machining type tpHelicalPlungeRoughing). Input parameter: Value (real number) - plunge radius. No output parameters.
Get_PlungeRadius	The function returns the plunge radius (real number). No input parameters.
Set_EdgeStock	The method sets the allowance at the pocket edge for roughing. Input parameter: Value (real number) - allowance at the pocket edge. No output parameters.
Get_EdgeStock	The function returns the allowance at the pocket edge for roughing (real number). No input parameters.
Set_BottomStock	The method sets the allowance at the pocket bottom for roughing. Input parameter: Value (real number) - allowance at the pocket bottom. No output parameters.
Get_BottomStock	The function returns the allowance at the pocket bottom for roughing (real number). No input parameters.
Set_Events	The method assigns events that control the cycle behavior. Events allow adding to any section of the tool path the features required by a particular cycle, changing the feed type or value, or skipping a tool movement prepared by the generator. Input parameter: Value (INCT_PocketCycleEvents) - object implementing the events. See the detailed description below. No output parameters.

Method	Method description
Get_Events	The function returns the object implementing the events that is linked to the generator (INCT_PocketCycleEvents). No input parameters.
MakeWorkpath	The method forms the cycle path from the specified parameters.

INCT_PocketCycleEvents - an interface implemented in the interpreter that allows controlling the behavior of the circular pocket milling cycle. The interface is a set of event methods executed during the tool movements inside the generator. With these methods, the features required by a particular cycle (dwell, oriented spindle stop, etc.) can be added to any section of the tool path, the feed type can be changed, or a tool movement prepared by the generator can be skipped.

If events are assigned to the generator, every tool movement inside the generator consists of 3 stages:

1. The INCT_PocketCycleEvents.BeforeMove method is executed. It receives the values set by the cycle: movement type, coordinates of the movement point, feed type.
2. The INCT_PocketCycleEvents.OnMove method is executed. It receives: movement type, coordinates of the movement point, feed type and the flag of whether the movement has been made. All parameters except the movement type can be changed in the implementation. Depending on the flag set in OnMove, the tool moves to the specified point with the specified feed, or the movement is not made.
3. The INCT_PocketCycleEvents.AfterMove method is executed. It receives the values returned by OnMove: movement type, coordinates of the movement point, feed type.

Description of the interface methods:

Method	Method description	Description of use
BeforeMove	The method is executed before each tool movement when the cycle path is formed. Input parameters: MoveType (TNCPocketCycleMoveType) - movement type, see the description below; Point (TNC3DPoint) - coordinates of the movement point; Feed (TNCFeedType) - feed type.	If some commands must be added before a movement, add them to the method implementation. For example, to turn on the machine coolant before milling the pocket, call INCT_CLData.AddCoolant with the appropriate parameters in the BeforeMove implementation

Method	Method description	Description of use
	pcmMoveToHoleCenter - movement to the pocket center; pcmMoveToSafeLevel - movement to the safe distance; pcmMoveToCutOfDepthByLine - movement to the next cut level along a line; pcmMoveToCutOfDepthByHelic - movement to the next cut level along a helix; pcmMoveToAside - movement from the center toward the pocket edge; pcmMoveToHoleEdge - movement along the flat spiral to the pocket edge; pcmMoveOnHoleEdge - movement along the pocket edge; pcmMoveToTopLevel - movement to the start point; pcmMoveFromHoleEdgeToCenter - movement from the edge to the pocket center. No output parameters.	under the condition MoveType = pcmMoveToSafeLevel.
OnMove	The method is executed immediately before each tool movement when the cycle path is formed. Input parameters: MoveType (TNCPocketCycleMoveType) - movement type, see the description above; Input and output parameters: Point (TNC3DPoint) - coordinates of the movement point; Feed (TNCFeedType) - feed type; Handled (boolean) -	To override the movement point or the feed, or to skip the movement. For example, to switch to rapid traverse when returning to the initial level after milling, in the OnMove implementation set the returned Feed parameter to ffRapid under the condition MoveType = pcmMoveToTopLevel (for the feed type, see OutStandardFeed).

Method	Method description	Description of use
	flag of whether the movement has been made inside the implementation (True - the movement has been made, False - it has not). If Handled is True after the OnMove call, the movement has been made inside the OnMove implementation, and the generator does not add a movement command to the tool path. If Handled is False after the OnMove call, the generator considers that the movement has not been made and adds to the tool path a command to move to the point Point with the feed type Feed.	To move the tool to a point other than the one calculated by the cycle, change the Point parameter returned by OnMove. If the movement to the calculated point is not needed, set the Handled parameter to True inside the OnMove implementation.
AfterMove	The method is executed after each tool movement when the cycle path is formed, even if the movement has been canceled in OnMove. Input parameters: MoveType (TNCPocketCycleMoveType) - movement type, see the description above; Point (TNC3DPoint) - coordinates of the movement point; Feed (TNCFeedType) - feed type. No output parameters.	If some commands must be added after a movement, add them to the method implementation.

Interpreter interaction log

The process of interpreting the NC program text, including the interaction with the interpreter, is recorded to the log when a separate log option is on. The log is managed in the System setup window: <Additional> page, <Logs> group, <Use logs> and <Use log interaction with the interpreter> options. The second option is available only when the first one is on. In ENCY Tuner the log is turned on in the same way.

If both options are on, the interpretation of the NC program text is recorded in the log for calculations in the “Simulation of machining by NC program” mode and for calculations of “G-code based operations”: the kernel's messages about the words read and the lines the interpreter writes.

The log is kept in one file per running program process: all calculations made while the program runs are written to this file. File location:

- ENCY - `%APPDATA%\ENCY Software\ENCY\Version 3\Log\ency-<process ID>.log`;
- ENCY Tuner - `%APPDATA%\ENCY Software\ENCY Tuner\Version 3\Log\ltuner-<process ID>.log`.

When the log file exceeds 20 MB, its previous content is moved to the `RotatedLogs` subfolder, and writing continues to a file with the same name.

The log is intended for creating your own interpreters.

AI-assisted interpreter development

This section explains how to develop and maintain an interpreter with the help of an AI assistant: how to set up the tools, what to ask the assistant for, what it needs from you, and how to check the result. It applies to interpreters written in C# from the templates of the CAM system, described in Developing a custom interpreter in (.NET) C#, and to native interpreters written in Delphi or C++ (see Interpreter library).

By default the assistant develops an interpreter in C# (.NET). It creates a native library in Delphi or C++ only if you explicitly ask for one or if you are changing an existing native interpreter.

What the work looks like

You write in the chat, in your own words:

> Programs for our machine write the rotary axis as B90, but the machine calls it B1. A program with B90 is not simulated. Fix it.

Then you watch. The assistant reads the NC programs and the machine of the project, finds where the interpreter reads axis words, decides what kind of difference it is (a setting or a change in the code), makes the change, builds the interpreter, runs the NC programs through it in the CAM system and shows you what changed in the simulation and in the log.

You read the result and answer the same way — "right, but the same machine also writes B=DC(30)" — and it goes round again.

You do not have to write the code yourself, although you can look at every change and edit it by hand whenever you prefer. What cannot be handed over is the judgment: whether the simulation matches what your machine does is decided by you. Everything this section says about checking the result exists for exactly that.

What the assistant needs from you

- The NC programs the interpreter must read, and the CAM project with the machine they are written for.
- The programming manual of the controller for the words and cycles in question.
- The installed CAM system or ENCY Tuner the interpreter must work in.
- The expected result: what the machine does with the block that is simulated wrongly.

The more real programs you give, the better: the assistant takes the spellings of the words from them, not from memory.

What it costs

The documentation and the skills are published free of charge, but the model that answers you is not part of them. An AI assistant is a service of whoever provides the model, paid for by the amount of text processed or by subscription. You will need an account with an AI service provider or with the developer of the AI client; the work runs on their terms and at their prices.

If your data must stay inside your network, the model can be deployed on your own hardware, with a client that supports it pointed at that model. Keep in mind that small local models handle multi-step work guided by skills noticeably worse: before relying on such a model, test it on a real task. Nothing else in this section changes — the CAM system, the interpreter and the NC programs are on your computer anyway.

Reading order

Page	What it covers
How AI-assisted development works	What parts the work consists of, what each of them does, what the assistant can and cannot do
Setting up the tools	The short path: install an AI client, download the skills, check with two requests
Organizing the workspace	The working folder as the assistant's memory: NC programs, machine project, manual, notes, change history
.NET workflow	The default path: a C# interpreter from a template — creation, build, debugging, changes
Native interpreter workflow	Only on explicit request or for an interpreter that is already native: Delphi, C++
Checking and confirming the result	Running all NC programs before and after a change, verification levels, preparing an interpreter for hand-over
Troubleshooting	Typical symptoms, their causes and what to do
Advanced setup	The full setup procedure: skills in different clients, build and debugging tools, the log, instruction file, several product versions, a separate workplace, automatic checks

If you are just starting, read the pages in order up to the .NET workflow; read the page on the native interpreter only if you need one. You do not have to read Advanced setup yourself: the assistant reads it when it sets up the tools.

Skills for the assistant

A skill is an instruction in plain language: what to do, in what order, what to check, and when the work counts as finished. There is no code in it — it is a text file the assistant reads and follows, much like a briefing given to a new colleague. Anyone who can explain a way of working in words can write or correct one.

The skills are in the `skills` folder of this section: each skill is a folder with a `SKILL.md` file. All the skills download from this website as one archive: download the skills (`ai-skills.zip`, with the six skill folders at the root of the archive). The skills are written in English. You can talk to the assistant in any language. Together they guide the whole work on an interpreter: the assistant starts with the general skill and turns to the others as the work goes on.

Skill (folder)	What it is for	Section page
`interpreter-development`	The general skill: what to find out first, in what order to work, and when the work is finished. By default an interpreter is developed in C# (.NET)	How AI-assisted development works
`inspect-nc-programs`	Collecting the NC programs and the machine data (axes, channels, M-functions), and listing every word the programs use	Organizing the workspace
`develop-dotnet-interpreter`	Creating, configuring, building and debugging a C# interpreter from a template — the default path	.NET workflow
`develop-native-interpreter`	A native interpreter library in Delphi or C++ — only when a native interpreter is really needed	Native interpreter workflow

Skill (folder)	What it is for	Section page
`nc-interpreter-foundations`	How the CAM system reads a block and when the interpreter's code is called	How AI-assisted development works
`verify-interpretation`	Running all NC programs before and after a change, comparing the results, and preparing an interpreter for hand-over	Checking and confirming the result

Each skill can also be read on this site through the link in the table. If the archive is not available, download the source file `SKILL.md` from the skill page and put it in a folder named after the skill, for example `skills/interpreter-development/SKILL.md`. Copy all the skills at once: they refer to each other. How to add them to your AI client is described in setting up the tools, in detail — in advanced setup.

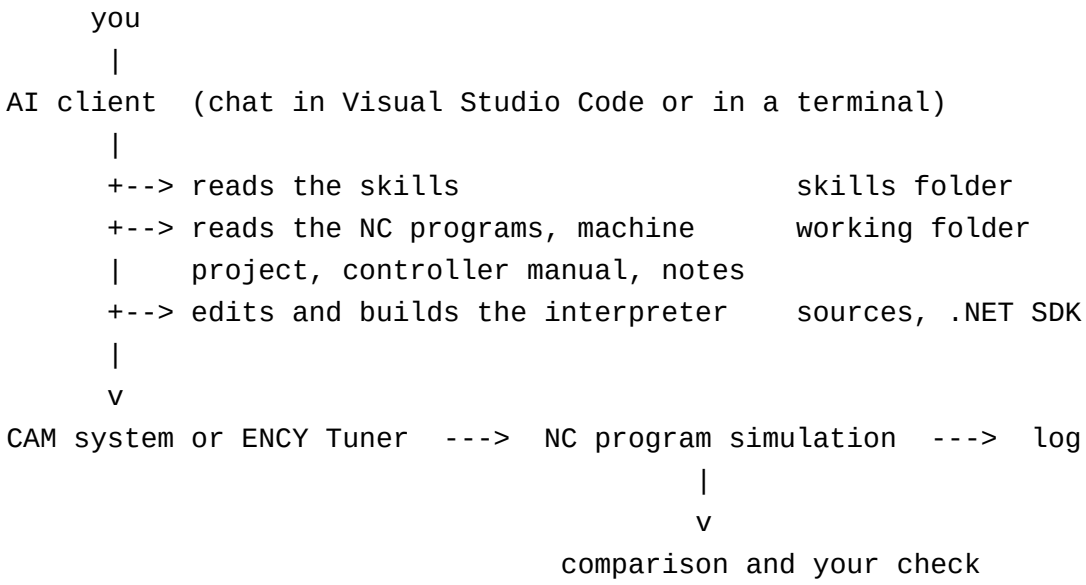
You can keep your own skills alongside them. They are plain Markdown: the practices your team accumulates — the controllers you support, the variations of your customers' programs — take the same form. See organizing the workspace.

What stays with you

Treat every answer as a draft. The assistant sees only what you gave it: it does not know undocumented behavior of the controller, and a plausible-looking simulation can be wrong. Before an interpreter is given to a user, run all the NC programs you have through it, read the warnings in the interpreter interaction log, and check the simulation of the blocks that were changed. Do not put passwords, customer archives or data that must not leave your company into the chat.

How AI-assisted development works

You talk to an AI client — a chat in a code editor or in a terminal. The client reads the skills and the files of your working folder, edits the interpreter sources, builds the interpreter and starts the CAM system or ENCY Tuner, in which the interpreter simulates the NC programs. No special servers are needed, and the only Visual Studio Code extensions you need are the chat itself and the C# extension for building and debugging: the assistant works with the same files and programs as you do.



What it consists of

Part	What it is	What it gives you
AI client	A chat extension of Visual Studio Code or a terminal program	A conversation with an assistant that is allowed to read files, edit them and run commands in the working folder
Skills	`SKILL.md` text files	A way of working written in words, so the cycle is the same every time: facts first, then the change, then the check. The work is led by the general `interpreter-development` skill, which brings in the others as needed

Part	What it is	What it gives you
Working folder	An ordinary folder on your computer	NC programs, machine project, controller manual and your notes — everything the assistant knows about your task
Interpreter sources	A C# project from a template or a native project	What the assistant changes
.NET SDK and Visual Studio Code with the C# extension	Build and debugging tools	Building the interpreter and running it under the debugger with the `F5` key
CAM system or ENCY Tuner	The installed product	Loads the interpreter and simulates NC programs in a G-code based operation
Interpreter interaction log	The product's log file	What the kernel and the interpreter did with each block: words read, warnings, errors

How the CAM system reads a block

The work is driven by the CAM system kernel, not by the interpreter: the kernel reads the NC program text, splits a block into words using the interpreter's register table, queues them and calls the interpreter's code to execute each word. Which functions are called and in what order is described in G-code based program interpretation; the `nc-interpreter-foundations` skill retells this for the assistant together with the traps that cause the most mistakes: the first movement of an operation, coordinates in a rotated coordinate system, spindles, cycles.

The meaning of each word is defined by the interpreter, and the same word means different things on different controllers. That is why the assistant always starts from the actual programs and the controller manual, not from the name of a word. The interpreter simulates what the controller would do, not what the CAM operation intended.

C# (.NET) by default

The assistant creates a new interpreter in C# from a template of the CAM system. Such an interpreter is built with one command, is debugged in Visual Studio Code and is easy to read for people who know the machine better than programming. A native interpreter in Delphi or C++ — only when you explicitly asked for one or when you are changing an interpreter that is already native. In detail — .NET workflow and Native interpreter workflow.

What the assistant does well and what it cannot do

It is good at listing every word that occurs in your programs, finding the place in the interpreter where a word is read, explaining unfamiliar code, preparing a small change in the style of the surrounding code, running all NC programs before and after the change and comparing the results block by block.

It cannot know controller behavior that is neither in the manual nor in the programs, see the machine, judge whether movements are safe, or take responsibility for an interpreter delivered to a customer. Why a successful build and a simulation without errors are not enough, and how to check the result, is described in Checking and confirming the result.

Setting up the tools

The assistant does the setup itself. Your part is small: install an AI client, download the archive with the skills and give the assistant the instructions. The rest — putting the skills in place, checking the build and debugging tools, finding the installed CAM system — the assistant does by itself and reports the result. Only permissions and actions in program windows need you: permission to install missing programs; restarting the AI client if that is required; in the CAM system or ENCY Tuner — creating the interpreter in the Create G-code interpreter window, clicking the Set ENCY file path button in that window (in ENCY Tuner the button shows its own name), then signing out of Windows and signing in again, and turning on the log in the System setup window. The assistant will tell you about each of these actions when it comes to it.

1. Install an AI client

It is most convenient to work in Visual Studio Code: the interpreter is built and debugged in it as well (see Interpreter debugging in Visual Studio Code). A chat is an extension like any other: open the Extensions panel (`Ctrl+Shift+X`), find the one you need by name and install it. Any client will do, as long as you can allow it to read and edit files in the working folder and run commands there — for example, Kilo Code or Claude Code. Terminal-based clients have their own installation procedure.

You will need an account with an AI service provider — see what it costs.

A C# interpreter also needs Visual Studio Code with the C# extension from Microsoft and the .NET SDK of the .NET version your product version runs on (see Product versions). You do not have to install them in advance: in step 3 the assistant checks what is missing and offers to install it — with your permission.

2. Download the skills

Download the documentation archive and unpack it. All the content of the archive is in its root folder `public-docs-Technical-Interpreters-en-v3`, and the skills are in the `src/ai-assisted-development/skills/` folder inside it: six folders, each with a `SKILL.md` file.

3. Hand it over to the assistant

Open the working folder in Visual Studio Code (see organizing the workspace), open the chat, allow it to run commands in the working folder and give it the task. The skills are copied into your user folder (`%USERPROFILE%`), outside the working folder, so the client will ask for permission for that as well. Correct the paths (`<archive>` is the full path to the `public-docs-Technical-Interpreters-en-v3` folder of the unpacked archive) and paste:

```
Set up the tools for interpreter development following the instructions in
<archive>\src\ai-assisted-development\advanced-setup.md
```

```
(the same page on the website:
https://docs.encycam.com/technical/Interpreters/3/en/src/ai-assisted-
development/advanced-setup.html)

Unpacked documentation archive: <archive>
The CAM system or ENCY Tuner is installed in: <installation folder>

Copy all the skills from the archive into this client's skills folder
for the user, not for the working folder,
check the .NET SDK and Visual Studio Code with the C# extension;
name whatever is missing and offer to install it.
Then tell me exactly what you changed and which skills you now see.
```

Clients pick up new skills in different ways: Claude Code — at once (if there was no skills folder before — after the `/reload-skills` command), Kilo Code — in a new session or after the `/reload` command, other clients — according to their documentation. If the client needs a restart, the assistant will tell you, since it cannot do that for you.

4. Check that everything worked

Two requests in the chat are enough. Both are safe: the assistant only reads and changes nothing.

Checking the skills. Write:

> Which skills for interpreter development do you see now, and which one will you start with?

A good answer lists six skills by their folder names and names `interpreter-development` first. If the assistant talks about "typical" skills or asks you to send files, the skills are not connected — go back to step 3 or to advanced setup.

Checking program reading. Put one or two real NC programs into the `programs` folder of the working folder and write:

> Read the NC programs in the `programs` folder and list which words occur in them and how they are spelled. Do not change anything.

A good answer is a list of words from your files: addresses, G- and M-functions, axis spellings such as `B90` and `B1=90`. If the answer contains words that are not in the programs, the assistant is reasoning from memory: point it to the files explicitly.

5. Updates

The skills are updated together with the documentation. When a new version is released, download the archive again and ask the assistant to replace the skill folders completely.

If something does not work

Troubleshooting collects typical symptoms. Advanced setup gives the whole procedure in detail — it is the very page you gave to the assistant, so you can perform any step by hand and check what it did.

Organizing the workspace

The assistant starts every session without memory of the previous one. It does not remember what you agreed on yesterday, which controller caused trouble last month, or why the interpreter reads a particular word exactly this way. It knows what is written in your working folder; what stayed in the chat does not exist for it.

That is why the working folder is not order for the sake of order but the assistant's memory. A few hours spent on it once save the same explanations in every following session.

What to keep in it

One folder, opened in Visual Studio Code:

```
work\
├── AGENTS.md          what the assistant must know every time
├── interpreters\      interpreter sources, one folder per interpreter
├── programs\          real NC programs of the machine and expected results
├── projects\          CAM projects with the machine
├── manuals\           controller programming manual
└── notes\             your practices and analyzed cases
```

The names do not matter; the separation does: sources apart from test data, your knowledge apart from the official documentation.

Programs and the machine

Put several real NC programs of this machine into `programs\`, not just one: generated by its postprocessor, written by hand, from other CAM systems. The assistant takes the spellings of the words from them: the same axis can appear as `B90`, `B=90` and `B1=90`, and the interpreter must understand every spelling that occurs in your programs. If there is a postprocessor that generates these programs, put it there too: it shows which variants of the words will appear.

Keep the CAM project with this machine (`*.stcp`) next to them and, if you have it, the expected result: what the machine does with the program. The controller manual in `manuals\` is the main source of what the words mean; without it the assistant will guess.

Ask the assistant to make a list of the words in all programs — this is what the `inspect-nc-programs` skill does. The list shows which words the interpreter already understands and which it does not, and serves as the list of what to check after every change.

AGENTS.md — a short permanent memo

This file is read at the start of every session, so keep it short — no more than a page — and move the details to `notes\`. What deserves a place in it:

- the machine, the controller and its version, which interpreter serves it;
- the versions of the CAM system and ENCY Tuner the interpreter must work in;
- where things are: programs, projects, manual, notes;
- working rules — read the programs and the manual first, keep a change minimal, after every change run all programs and compare with the previous result;
- restrictions you have already formulated once — written down the way you said them. Otherwise, a month later the assistant will again propose an idea you rejected for good reason.

Ask the assistant to draft it and correct the draft — that is faster than starting from an empty file.

notes — where your experience accumulates

After every analyzed case — a short file: what was wrong, how it was found, what helped and why exactly that. It is the "why" that saves time later.

- One topic — one file, not one growing file: a small file is easy to find and cheap to read.
- Describe the current state. Remarks such as "it used to be different" go stale over time; if the folder is under version control, the history is there anyway.

Also keep a written list of the requests and decisions for the current task in `notes\`. In a long conversation the assistant compresses earlier messages, and details get lost; a written list keeps them. Before reporting "done", ask the assistant to go through it item by item and show the evidence for each.

When a way of working starts repeating — the same check, the same sequence for a family of controllers — write it down as your own skill: a folder with a `SKILL.md` file, like the supplied skills. A client finds skills only in its skills folder, so put your skill where the others are installed (see advanced setup); a file in `notes\` is read by the assistant only if you point to it. A skill that names customers or the specifics of your particular machines must not be delivered with the interpreter or published.

Two ways to keep history

Folders and copies — you keep the history yourself

Before a significant change you copy the interpreter folder, keep the results of earlier runs next to the programs, and dated folders serve as the history. This is a perfectly workable approach. Its price is that the history relies on your memory: forget to make a copy, and the previous state is gone.

Git — the tool keeps the history

The same folder, but the past is the responsibility of a tool made for exactly that. It also removes the main worry of working with an assistant: every state is saved, so a failed edit costs a minute to roll back,

not a working interpreter.

You do not have to learn git for this. Ask in words, and the assistant does it:

- "commit, axis B1 is read" — it commits the change with a message saying why;
- "compare with what we had on Friday" — it shows what has changed;
- "find when the G83 cycle stopped being simulated" — it searches the history;
- "put it back as it was before the last edit" — it restores from the history instead of retyping.

If git is not installed, ask the assistant to install it — the official download is at git-scm.com. Ask it at the same time to exclude the build output (`bin`, `obj`) and logs from the history.

When the open folder must be the interpreter folder

For a C# interpreter, Visual Studio Code must be opened on the folder of the interpreter itself — that is how building and debugging with the `F5` key work. The memo and the notes then end up above the open folder. Two techniques are enough:

- Keep everything in one repository. Let the whole `work\` folder be the repository, with the interpreter folders inside it. Git works from any nested folder, and there is still one history.
- Make the memo visible from the interpreter folder. Claude Code reads instruction files in the folders above the open one by itself and needs nothing. Kilo Code reads only the open folder: point to the memo in its settings, or put a short two-line `AGENTS.md` into the interpreter folder: rules — in `..\..\AGENTS.md`, notes — in `..\..\notes\`. In detail — in advanced setup.

Install the skills for the user rather than for the working folder: then they are available with any folder open, and it is enough to put them in place once.

What must not be in it

Passwords, tokens, license files, customer archives unrelated to the interpreter, and unrelated repositories. The assistant may read everything that is in the working folder, and if you keep a history, it stays in the history. If a customer's program is needed only as an example, give a shortened one: a few blocks with the word being checked and the program header.

.NET workflow

A C# interpreter is a .NET project that references the interpreter SDK package and is built into a `*.dll` library; next to it lies the `*.snci` settings file. The CAM system or ENCY Tuner loads the library and calls it when simulating NC programs. This is the default path: the assistant chooses it for every new interpreter unless you asked for something else.

How to create an interpreter from a template is described in Developing a custom interpreter in (.NET) C#, how to run it under the debugger — in Interpreter debugging in Visual Studio Code. Details for the assistant are in the `develop-dotnet-interpreter` skill.

Where to start

Most often not from scratch. If there is an interpreter for the same or a similar controller whose behavior has been checked on real programs, copy it and refine it. The copy needs new identifiers: the GUID returned by `GetIdentifier` (it is also `LibraryID` in the `*.snci` file) and the GUID in the `ID` field. The CAM system cannot tell apart two interpreters with the same identifiers.

If you need a new one, create it with the Create interpreter item in the Utilities list of the CAM system or with the Interpreter manager item in ENCY Tuner — both open the same Create G-code interpreter window (the interpreter creator). The template the interpreter is created from is designed for one family of controllers; the assistant checks and fixes what the interpreter creator does not do by itself: fills in `ID` in the `*.snci` file, sets the released version of the SDK package, removes handlers of words that mean something else on your controller. For a descendant interpreter it also fixes the reference to the parent: `ParentID` must match the parent's `ID`, and `ParentFileName` must name the parent's `*.snci` file.

An interpreter can inherit another one — see Interpreter inheritance. Before taking a supplied interpreter as a parent, check its purpose in the list of interpreters: some of them are marked "Tool path import only" — they do not link NC program blocks to movements.

In any case, start from a known state:

1. Put the interpreter under version control before changing anything.
2. Build it without changes. If it does not build as it is, further diagnostics are pointless.
3. Run all your NC programs through it and save the result as the baseline for comparison.

Product versions

Each product version has its own .NET version: version 3 of the CAM system and ENCY Tuner runs on .NET 10, version 2 runs on .NET 8.

The interpreter references the `EncySoftware.NCInterpreterSDK.Net` package. The assistant determines its version as follows:

1. It reads which package version the installed product was built with: the `Bin64\STNCInterpreterSDKProject.stbuildresult.json` file, the `build_results.versions` section. An ENCY Tuner installation may not have this file; then the version is taken from the CAM system of the same version (ENCY 3 for ENCY Tuner 3, ENCY 2 for ENCY Tuner 2). Where ENCY Tuner has the file, its number is set by the build of ENCY Tuner, which does not publish the SDK packages, so a package with exactly that number may not exist on the package source.
2. It sets the newest released version of the package (without `-dev`, `-rc` and similar suffixes) with the same major number and not newer than the version from step 1. For ENCY Tuner, this is the number from its file or, if there is no file, from the file of the CAM system of the same version. The number from the ENCY Tuner file is only an upper limit.

The released packages are published on nuget.org, which the .NET SDK uses by default. A version that a nightly or test build of the product was built with cannot be downloaded by the customer, so it does not get into a project that leaves your computer.

If the interpreter must work in several product versions, one project is built for each of them; it must be checked in each.

The cycle

1. Formulate the task as a visible difference in the simulation: which program, which block, what should happen.
2. Ask the assistant to find the place where the interpreter reads this word and to explain what it does now.
3. Decide together with it what kind of change this is:
   - a variation — another axis name, a machine M-function, a word without effect on the tool path. Its place is the `InterpreterSettings.ini` file next to the library, not the code: then the next such case can be solved by a support engineer without a build. The first time, the setting key is added to the code; after that it is a setting;
   - new behavior — a cycle, a frame, a feed mode. This is a change in the code, in the file of the role it belongs to;
   - a kernel or SDK limitation — the required call does not exist. The assistant describes the workaround and its limits and writes down what would have to change in the product.
4. Make one small change and build the project. Build warnings are information about your change, not noise.
5. Run all NC programs and compare with the previous result — see Checking and confirming the result.

Debugging

The `F5` key in Visual Studio Code builds the interpreter and starts the configuration selected in the Run and Debug panel: Launch CAM starts the CAM system, Launch Tuner starts ENCY Tuner; a breakpoint is

hit when an NC program is simulated. Before debugging, turn on the log as described in advanced setup — otherwise the log will contain neither the kernel's messages about the words read nor the lines the interpreter writes.

To simulate a program, create a G-code based operation, paste the program text, select the interpreter in the Interpreter file field on the Strategy tab of the operation parameters and calculate the operation.

Keep building and debugging the way the template made them: a colleague must be able to open the folder, press `F5` and stop at a breakpoint. The assistant does not replace them with its own scripts and environment variables.

What to check in the code

- The changes contain what was asked for and nothing more: no reformatting, no unrelated rewrites.
- The line endings and encoding of the template files are preserved — otherwise the history shows every line as changed.
- Numbers in the program are read regardless of the Windows language: the dot is always the decimal separator.
- The interpreter's state is stored in its object, not in static fields: the CAM system creates a new instance for each operation calculation, and separate ones for the parent and for the cycle contour, and a static field is not shared between them; ENCY Tuner, on the contrary, keeps one instance per channel — from the first operation of the channel to the last — and the modal state and a static field carry over from one operation to the next: for example, a `G0` from the end of one operation is still in force at the start of the next.
- A word the interpreter does not simulate does not disappear silently: it is either listed as read without effect on the tool path, or gives one warning per program that a machine operator can understand.

Rules that do not depend on the kernel — parsing numbers and expressions, cycle parameters, rules for axes and spindles, reading the settings file — the assistant can cover with unit tests. Tests complement running real programs but do not replace it.

Native interpreter workflow

A native interpreter is a `*.dll` library built in Delphi, C++ or another language that compiles to machine code. The assistant takes on a native interpreter only if you explicitly asked for it or are changing an existing native interpreter; a new interpreter is created in C# by default — see .NET workflow.

Reference information on the library is in Interpreter library, Interpreter settings file and Interpreter interface. Details for the assistant are in the `develop-native-interpreter` skill.

What the assistant needs

- The interpreter sources and the project it is built with — or an explicit request to create a new one.
- A compiler: RAD Studio for Delphi, a C++ compiler or another one.
- The SDK interface declarations. They come as NuGet packages from the same source as the SDK package for C# (see Product versions), with the same version: for Delphi — `EncySoftware.NCInterpreterSDK.bpl.x64` (`IDL.<Name>.pas` units and `*.bpl`, `*.dcp` packages), for C++ and other languages — `EncySoftware.NCInterpreterSDK.tlb` (`*.tlb` type libraries, `*.h` headers and `*.idl` descriptions). Both packages pull in one package per interface, each with its own version number: `STNCInterpreter`, `STNCAnalyzer`, `STNCFormers`, `STNCCustomInterpreter` and `STNCConnect` (the protection interface used by protected interpreters). The assistant takes the declarations from them instead of inventing them. If the package of the required version is not available in the source, contact support.
- The `*.snci` file the interpreter is loaded with.

What the library must have

- The library exports two functions: `NCT_GetInterface` and `NCT_FreeMemory`. If either of them is missing, the product will not load the library.
- The string returned by `GetIdentifier` equals `LibraryID` in the `*.snci` file. Both identifiers, `LibraryID` and `ID`, are unique.
- For current products a 64-bit library is built; its name is given in `LibraryFileName_x64`. A 32-bit one, named in `LibraryFileName_x32`, is needed only for an old 32-bit product version.
- A descendant interpreter receives its parent through `SetInterface` and calls it according to the rules of interpreter inheritance.

Registers can be defined either in the `*.snci` file or in the library itself. It is more convenient to declare them in the library together with their handlers: then a new word is one line in the register list and one handler.

The cycle

1. Build the sources without changes and check that the library loads and simulates your NC programs. Save the result as the baseline.

2. Before looking for an error in the simulation, check the function exports, the bitness, the identifiers and the pair of `*.snci` and `*.dll`: most "it does not work" cases start here.
3. Make one small change, build, run all NC programs and compare with the previous result — see Checking and confirming the result.
4. Debug under your compiler's debugger by attaching to the CAM system or ENCY Tuner, and read the interpreter interaction log.

Protected interpreter

An interpreter that checks its data through the protection interface is described in Protected interpreters. Check it from the container in which it will be delivered: from an ordinary folder it receives no protection data. Without its data, a protected interpreter must refuse to simulate. How it reports this is up to its author: the assistant makes the interpreter show a clear message, but an existing one may silently produce an empty tool path — in that case, check the protection data first.

What not to do

- The kernel and the product libraries are not changed to make the interpreter work.
- Delphi implementation details are not part of the binary interface: in C++ and other languages the interface is checked against the type library and a working interpreter.
- Compilation is not the end of the work: the library must load and simulate your programs.

Checking and confirming the result

AI assistance changes how a draft is produced. It does not change how an interpreter is accepted. A successful build and a simulation without errors prove that the programs ran — and nothing more.

State how far the check has gone

Use one of four states and keep the evidence for it. The point is that an unfinished check stays visible, not that everything is called "done".

State	What it means
Not verified	There is a plan, changes or a build, but no NC programs have been run through the interpreter
Technically verified	The interpreter loaded, the programs ran, the log and the simulated commands are saved, but they have not been compared or analyzed
Verified on reference programs	All programs have been run before and after the change, every difference is explained, the log warnings have been read, all product versions the interpreter is intended for have been checked
Accepted for the machine	You or the customer confirmed the simulation for the particular machine

Only a person assigns the last state. Never describe a result as finished on the strength of an exit code or a plausible tool path.

Running the programs

Before any run, turn on the log as described in advanced setup: without it there is nothing to compare the warnings with.

One program can be checked by hand: create a G-code based operation, select the interpreter, calculate the operation, then look at the simulation and the log. Doing this for the whole set after every change takes too long. The assistant can make a small run program that, without your clicks, opens the project, simulates each NC program with the interpreter and saves the simulated commands and the log of that run. It may be built only on the public programming interface of the CAM system for external applications, CAMIPC: it is described in the CAM API documentation, and the library for connecting to the CAM system comes with its installation. Where the documentation does not show a step (for example, how to load the NC program text into an operation or select its interpreter), and in ENCY Tuner, which has no

documented interface of this kind, the run is manual: you calculate the operations, and the assistant prepares the list of programs and steps, then collects and compares the results. Keep the run program in the interpreter repository and never give it to the customer.

Before trusting the run program, test it on a known error: give it an NC program with an error and with an unknown word and make sure it counted both. A counter that reads the wrong field of the log shows "0 errors" on anything.

Run the programs one at a time: runs in one process of the CAM system write to one log file, and parallel processes may fail to start the kernel at all.

Comparison

1. Compare the summaries of all programs before and after. A changed number of warnings or commands in a program the change did not concern is the first thing to explain.
2. For each changed program, start with the first difference and trace it to the block and the word that caused it. An unintended difference matters more than an intended one.
3. Read the warnings themselves, not only their number: each one names a word, an axis or a cycle.
4. Check the edge cases that are not in the programs with a short program: the word in all spellings from the list, at the limits, omitted, repeated. Build it from the header of a real program of the same operation and replace only the blocks being checked — without the header the program may produce no movement at all.

A difference caused by the check itself rather than by the interpreter — a different coordinate system of the test operation, a different machine in the project — is not a result: name it and exclude it.

Checking through the postprocessor

If you have the customer's postprocessor and the CAM project, close the loop: generate NC programs from the project's operations, run them through the interpreter and compare the simulation with the tool path of the operation itself. A discrepancy may turn out to be a postprocessor error. The interpreter simulates what the controller would do with the program, not what the operation intended: report such an error to the author of the postprocessor instead of hiding it in the interpreter.

Safety check

Before hand-over, review every change for cases where the simulation looks safer than the machine: a missing retract, a guessed start point, a cycle that is not simulated but leaves the operation looking finished. Check each such case with a short program. After every round of edits based on review comments, run the whole set of programs again.

Preparing for hand-over

The customer receives a folder for each product version: the `*.snci` file, the interpreter library and `InterpreterSettings.ini`, plus the `*.deps.json` file only if the interpreter does not load without it (this is checked on an unmodified installation). The assistant copies the required files by name rather than archiving the whole build folder: it contains debug files and service files of the SDK package. With them goes a README in the customer's language: what is supported, the limitations, the settings, which folder is for which product version, whether the product needs a license option to load a custom interpreter, and which simplifications were made on purpose, so that they are not taken for errors.

Before sending, check that:

- the files contain no traces of your computer: user names, home folders, internal servers. Build the delivered library without debug information or with path mapping: a normal build writes the path of the `*.pdb` file on your computer into it;
- the files contain no names that do not belong to this customer: names of other customers, of your computer and its users, of internal servers and package feeds — neither in file names, paths, settings, launch configurations or SDK package references, nor in the README. Search the content of every file, in UTF-8 and UTF-16 encodings, not only the file names; the same applies to everything sent together with the package;
- the project, if the customer asks for the sources, builds from a clean copy with the released SDK package;
- the interpreter works on an unmodified product installation.

Build the package from sources committed to the history, and record what was sent, when and to whom.

What the assistant must not do

It must not assign the "Accepted for the machine" state to a result, change the reference programs, the machine or the project to make the comparison match, claim a check of a product version it did not run, or send the package without your permission.

Troubleshooting

The assistant does not see the skills

Ask which skills it has now. If the six interpreter skills are not among them, check that the skill folders are in the folder your client actually reads, that each of them contains a `SKILL.md` file, and that the client has picked them up: Claude Code sees new skills at once, or after the `/reload-skills` command if there was no skills folder before copying; Kilo Code — in a new session or after the `/reload` command; other clients — according to their documentation, often only after a restart. Where each client looks for skills — see advanced setup.

The assistant answers from memory instead of the programs

This happens when the request names no files or when the answer is not required to refer to the data. Name the programs and the manual explicitly, ask which file and which block it took the word from, and ask it to follow the skills — they require reading the programs first.

The assistant reports success without running anything

A build is not a result. Ask which programs were run, where the log of that run is, and how the result differs from the previous one. If there is none of this, the state is Not verified, see Checking and confirming the result.

The interpreter does not load

First read which Interpreter file is selected on the Strategy tab of the operation parameters, and the CAM system log. If the interpreter is not loaded, every word of the program will be an error.

The CAM system describes any failure to load a C# interpreter the same way — with the message "External library don't support NCT interface:" followed by the path to the library; a native library without the required exports gets the same message. The real cause does not get into the message. Most often it is one of these:

- the library named in the `*.snci` file, or its dependencies, is not next to that file;
- the library is built for a different .NET version than this product version needs;
- an exception occurred when the interpreter object was created — usually when the registers were created.

Ask the assistant to check the files and to create the interpreter object in a small console program with the same SDK package — this shows the real error message.

For a native interpreter, check the bitness of the library, the export of `NCT_GetInterface` and `NCT_FreeMemory`, and that `GetIdentifier` matches `LibraryID` — see Native interpreter workflow.

Two interpreters get mixed up

The CAM system tells interpreters apart by the identifiers in the `*.snci` file. A copied interpreter with the same `LibraryID` and `ID` is indistinguishable from the original: sometimes one simulates, sometimes the other. Give the copy new identifiers.

A breakpoint is not hit

Check in order: there is a `*.pdb` file next to the library (a build without debug information makes any breakpoint "dead"); the operation uses your interpreter; the build before the launch succeeded — open the terminal panel and read its result; the CAM system was given the `*.snci` file from the folder of the version you started.

Settings from InterpreterSettings.ini have no effect

The templates have known errors in reading `InterpreterSettings.ini`: the file may not be read at all, and the log contains only an information line starting with `Error reading settings file`. A log filtered by errors will not show it. Ask the assistant to check the template the interpreter was created from — the `develop-dotnet-interpreter` skill lists these errors and their fixes.

An axis is simulated in the wrong place or not at all

The SDK package helper that the .NET templates use to find the machine axes warns about an unknown axis name once and then silently skips its values. Check that the axis name in the program matches the axis address in the machine schema, and that the name mapping works for all spellings from the word list: `B90`, `B=90`, `B1=90`.

The second channel moves the axes of the first or is not simulated

The interpreter must move the axes the CAM system gives it for the channel of its program: for the second channel, for example, `X2`, `Y2`, `Z2`. If the second channel moves the axes of the first, or its tool path shrinks to a point, compare these axes with the axes the interpreter passes to its motion commands. A mismatch is a defect of the interpreter or of its `*.snci` file, not of the CAM system.

Check the synchronization of channels in the CAM system: depending on its version and license, ENCY Tuner may simulate the channels one after another instead of at the same time, and then the wait points hold nothing back.

Empty tool path without an error

Common causes:

- the program lacks the header the postprocessor outputs it with: a test program made only of the blocks being checked may produce no movement at all — build it from the header of a real program of the same operation;
- the parent interpreter is intended only for importing a tool path (in the list of interpreters such interpreters are marked "Tool path import only");
- the axis name in the program does not match the axis address in the machine schema (see the section above);
- a protected interpreter did not receive its protection data — for example, it was started from an ordinary folder instead of the container — and is written so that it then silently simulates nothing. See Protected interpreters.

The result differs from what another product version gives

Each version of the CAM system and ENCY Tuner has its own .NET version and its own version of the SDK package (see Product versions). Check which product version the run was in and which one the library was built for — see Several product versions on one computer.

The documentation the assistant reads is out of date

A local copy is a snapshot: download the archive again when a new version is released, and record which version a decision is based on. Version-dependent details — the SDK package version, interface names — check against the installed product, not against the copy.

Advanced setup

This page is the full setup procedure and a reference for checking what has been done. Give it to the assistant as described in setting up the tools, follow it by hand, or read it when your team needs a reproducible result.

What this is about

This section is addressed to the assistant reading this page in order to set up the tools: here is the domain it is working in.

A controller executes an NC program — a text of blocks; each block consists of words: addresses with numbers, G- and M-functions, cycle calls. The CAM system can simulate such a program: show the tool path and the machine movements it will cause. For this it needs an interpreter — a library that explains to the CAM system kernel what each word means on a particular controller. Each controller and its dialect needs its own interpreter, and refining the interpreter is the work of the person the assistant is helping. Usually this is a process engineer, a support engineer or a tester who knows the machine much better than programming.

An interpreter consists of two files: the `*.snci` settings file and the `*.dll` library. The library comes in two kinds:

- .NET — a C# project created from a template of the CAM system — the default path;
- native — a library in Delphi, C++ or another compiled language — only on the user's explicit request or for an interpreter that is already native.

What the tools are set up for: reading real NC programs and the controller manual, editing and building the interpreter, running programs through it in the CAM system or ENCY Tuner, and comparing the result with the previous one. Whether the simulation is correct is decided by that person; the assistant prepares the grounds on which they judge.

The setup consists of three parts, in this order: skills, build and debugging tools, working folder. A problem report states the versions of the CAM system or ENCY Tuner, the .NET SDK, the AI client and the skills: most "it worked yesterday" reports are explained by a difference in versions.

Skills in different AI clients

The skills are published with this documentation: the skills archive `ai-skills.zip` is downloaded from the link on the AI-assisted interpreter development page, and the six skill folders are at its root. The documentation archive contains the same skills, in its `src/ai-assisted-development/skills/` folder inside the archive's root folder `public-docs-Technical-Interpreters-en-v3`. A skill is a folder named after the skill with a `SKILL.md` file inside. Copy all six folders at once: the skills refer to each other by name.

Client	Where to copy
Kilo Code	`.kilo/skills/` of the working folder, or `%USERPROFILE%\.kilo\skills\` to have the skills in every project; a new session picks them up, the `/reload` command does it without a restart
Claude Code	`.claude/skills/` of the working folder, or `%USERPROFILE%\.claude\skills\` to have the skills in every project; new skills are visible at once, and if there was no skills folder before — after the `/reload-skills` command
Other clients	Their own skills folder of the working folder or of the user, an entry in the agent configuration, or a settings item — according to the client's documentation

If the client has no skill mechanism, paste the content of the `SKILL.md` files you need into the chat or reference their files from the client's instruction file (`AGENTS.md`, `CLAUDE.md` and similar).

However you connected them, check that they are loaded: ask the assistant which skills it sees now. Skills installed for the user are available with any folder open; skills installed for the working folder — only in it.

The skills are updated together with the documentation. Replace the folders completely instead of editing them in place: keep your own additions as separate skills next to them (see organizing the workspace).

Build and debugging tools

A C# interpreter needs:

- .NET SDK of the .NET version used by the required product version (see Product versions). Check with the `dotnet --list-sdks` command.
- Visual Studio Code with the C# extension from Microsoft: it contains the debugger the launch configurations of the templates are designed for. The XML Language Support by Red Hat extension is convenient for editing `*.snci` files but not required.
- The path to the product program for debugging. The template starts the CAM system or ENCY Tuner from the folder stored in the environment variables `NCKernelDir` (CAM system) and `TunerDir` (ENCY Tuner). The Create G-code interpreter window (the interpreter creator below) writes one of them — the variable of the product it was opened from — when the Set ENCY file path button (in ENCY Tuner the button shows its own name) is clicked in it and the product's executable file is selected; after

that you need to sign out of Windows and sign in again, or at least start Visual Studio Code again from the Start menu. If there is no variable, the path can be written into `launch.json` — the standard product installation folder, not a folder that exists only on your computer. The name of the CAM system's executable file is put into `launch.json` by the interpreter creator; a template copied by hand, without the creator, keeps `${env:CAMName}`, which nothing sets — write the file name there instead. In detail — Interpreter debugging in Visual Studio Code.
- Log: in the System setup window, on the Additional page, in the Logs group, the Use logs and Use log interaction with the interpreter options are turned on (the second one is available only when the first one is on); the same in ENCY Tuner. Without them the log contains neither the kernel's messages about the words read nor the lines the interpreter writes. In detail — see Interpreter interaction log.

A native interpreter needs its compiler and the packages with the SDK interface declarations — see Native interpreter workflow.

Whatever is missing from this list must be named, not replaced with a guess; missing programs are installed only with the user's permission.

Working folder instruction file

Put requirements that always apply where the assistant always reads: into the instruction file of the working folder (`AGENTS.md` or the file your client reads) or into skills. What to keep in it is described in organizing the workspace.

For a C# interpreter the open folder must be the folder of the interpreter itself, and clients look for instruction files in different ways. Claude Code reads `CLAUDE.md` (or `AGENTS.md` if there is none) both in the open folder and in all folders above it, so the memo of a working folder that lies above the interpreter folder is visible to it anyway. Kilo Code reads `AGENTS.md` only in the open folder, therefore:

- in Kilo Code you can list additional instruction files in the global `instructions` setting of the `kilo.jsonc` file (usually `%USERPROFILE%\.config\kilo\kilo.jsonc`) — put the full path to the `AGENTS.md` of the working folder there, and it will be read in any project;
- if the client cannot do this, put a short `AGENTS.md` into the interpreter folder with a reference to the shared file and the notes folder.

In practice, the following rules pay off:

- first read the real programs and the controller manual, and name where each word was taken from;
- C# (.NET) by default, a native interpreter — only on explicit request;
- variations — into `InterpreterSettings.ini`, not into the code;
- one change at a time, then a build, a run of all programs and a comparison with the previous result — before any report that the work is done;
- do not change the installed product, its templates and libraries;

- do not send the interpreter to the customer without permission.

Several product versions on one computer

An interpreter works in the product version it is built for: each version has its own .NET and its own version of the SDK package (see Product versions). Decide which versions the interpreter is developed for, and run the programs in each of them. One project can be built for several versions — one output folder and one launch configuration for each. If a version is not installed on this computer, the check in it cannot be considered done: say so.

The `NCKernelDir` and `TunerDir` variables point to one installation. If a released version and a beta version are installed side by side, check which one the variable points to before comparing results: a run in another version finishes successfully and simply gives a different result.

A separate workplace for the assistant

- A copy or a branch of the interpreter, not the files the customer already has.
- A separate folder for run results, outside the source tree.
- Running from a copy of the build output, so that a new build does not replace the library in the middle of a set of programs.
- Shortened programs and projects instead of full customer archives, where that is enough.
- No passwords, tokens, license files, customer archives or unrelated repositories in the working folder and in the instruction files — see What must not be in it.

Automatic checks

A run program that simulates all NC programs without clicks and saves the simulated commands and the log makes the check reproducible: fixed programs, a fixed project, a fixed interpreter version. The assistant makes it following the `verify-interpretation` skill on the public programming interface of the CAM system; the rules without which it cannot be trusted, and the cases where the run stays manual, are in Running the programs. Automation can report that the simulation has changed, but never that the change is correct: the differences are analyzed by a person.

Appendix

See the child pages.

List of interpreters

Interpreters for the following CNC systems are currently available.

Machine group	CNC system	Comment	Note	List of implemented commands* *(Commands may be implemented incompletely, see the note)*
Milling	APT	Tool path import only		FEDRAT: MMPM, MMPR; RAPID, MULTAX, GOTO, ARCSLP XY, YZ, ZX, CLW, CCLW, CIRCLE, TOOLNO, LOADTL, SPINDL, COOLNT, CUTCOM, LENGTH, PLUS, RADIUS, RIGHT, LEFT
	Apt_Simplify_3D	Tool path import only		FEDRAT: MMPM, MMPR; M101, M103, RAPID, GOTO
	ISO	Tool path import only		G0, G01, G02, G03, G4, G90, G91, G17, G18, G19, X, Y, Z, I, J, K, R, G20, G21, S, M3, M4, M5, M13, M14, M0, M1, M2, M30, M7, M8, M9, T, F, G43, G44, G49,

Machine group	CNC system	Comment	Note	List of implemented commands* *(Commands may be implemented incompletely, see the note)*
				H, G40, G41, G42, D, G53, G10, G54..59, G73, G74, G76, G80, G81, G82, G83, G84, G85, G86, G87, G88, G89, G98, G99, P, G94, G95
	Global control	Tool path import only	An additional license is required	G0, G1, G2, G3, G90, G91, G161, G162, G70, G71, G04, X, Y, Z, I, J, G74, M0, M2, M30, F, G40, G41, G42, D
	Fanuc 30i	For simulation and tool path import		N, G0, G01, G02, G03, G90, G91, G17, G18, G19, G20, G21, X, Y, Z, A, B, C, I, J, K, R, S, M3, M4, M5, M0, M1, M2, M30, M8, M9, M6, T, F, G43, G43.4, G44, G49, H, G41, G42, D, G53, G92, G54..G59, G73, G74, G76, G80,

Machine group	CNC system	Comment	Note	List of implemented commands* *(Commands may be implemented incompletely, see the note)*
				G81, G82, G83, G84, G85, G86, G87, G88, G89, G98, G99, G94, G95, G96, G97, G68.2, G69, G53.1, M98, M99, O, G4, P, L, G52
	Fanuc 0i EmcoMill E600	For simulation and tool path import	An additional license is required	Fanuc 30i commands (descendant interpreter), plus M13, M14 (spindle on with coolant), G65 (executed as G83)
	Fanuc 0i MF	For simulation and tool path import	An additional license is required	Fanuc 30i commands (descendant interpreter)
	Fanuc Mill Custom	For simulation and tool path import	Customizable interpreter	N, G0, G01, G02, G03, G90, G91, G17, G18, G19, G20, G21, X, Y, Z, A, B, C, I, J, K, R, S, M3, M4, M5, M0, M1, M2, M30, M8, M9,

Machine group	CNC system	Comment	Note	List of implemented commands* *(Commands may be implemented incompletely, see the note)*
				M6, T, F, G43, G43.4, G44, G49, H, G40, G41, G42, D, G53, G92, G54..G59, G73, G74, G76, G80, G81, G82, G83, G84, G85, G86, G87, G88, G89, G98, G99, G94, G95, G96, G97, G68.2, G69, G53.1, M98, M99, O, P, L, G4, G52; the codes for tool change, coolant, spindle on with coolant, axis brakes and auxiliary codes are set in the interpreter settings
	Haas VF-2	For simulation and tool path import		G0, G01, G02, G03, G04, G90, G91, G17, G18, G19, X, Y, Z, A, B, C, I, J, K, R, S, M3, M4, M5, M0, M1, M2, M30, M8, M9, T, F,

Machine group	CNC system	Comment	Note	List of implemented commands* *(Commands may be implemented incompletely, see the note)*
				G43, G44, G49, G40, G41, G42, D, G53, G92, G54..59, G73, G74, G76, G77, G80, G81, G82, G83, G84, G85, G86, G89, G98, G99, G52
	Heidenhain iTNC 530	For simulation and tool path import		L, FMAX, F, X, Y, Z, IX, IY, IZ, A, B, LC, IA, IB, IC, CC, C, CP, CR, R, DR-, DR+, IPA, MM, INCH, M91, S, M8, M9, M3, M4, M5, M13, M14, M0, M1, M2, M30, DL+, DR2+, RL, RR,R0, CYCL CALL, M99, M89, CALL LBL, Q, CYCL DEF200, 201, 202, 203, 205, 206, 207, 7, 9, 13, ORIENT, ANGLE
	HNC-8	For simulation and tool path import	An additional license is required	N, G0, G01, G02, G03, G90, G91, G17, G18, G19,

Machine group	CNC system	Comment	Note	List of implemented commands* *(Commands may be implemented incompletely, see the note)*
				G20, G21, X, Y, Z, A, B, C, I, J, K, R, S, M3, M4, M5, M0, M1, M2, M30, M7, M8, M9, M6, T, F, G43, G43.4, G44, G49, H, G41, G42, D, G53, G92, G54..G59, G73, G74, G76, G80, G81, G82, G83, G84, G85, G86, G87, G88, G89, G98, G99, G94, G95, G96, G97, G68.2, G69, G53.1, M98, M99, O, G4, P, L, M19, M12, M12.1, M13, M13.1
	Mazatrol SmoothG	For simulation and tool path import	An additional license is required	G00, G01, G02, G03, X, Y, Z, G90, G91, F, I, J, K, R, G17, G18, G19, G53, G28, M30, M0, M1, M5, M6, M3, M4, S, M8, M9, G40, G41, G42, G4, G94,

Machine group	CNC system	Comment	Note	List of implemented commands* *(Commands may be implemented incompletely, see the note)*
				G95, G50, G52, G97, T, G98, G99, G80, G82, G81, G88, G89, G85, G87, G83, G84, G73, G86, G76, G74, G54..G59
	Mitsubishi M80a	For simulation and tool path import	An additional license is required	N, G0, G01, G02, G03, G90, G91, G17, G18, G19, G20, G21, X, Y, Z, A, B, C, I, J, K, R, S, M3, M4, M5, M0, M1, M2, M30, M8, M9, M6, T, F, G43, G43.4, G44, G49, H, G41, G42, D, G53, G92, G54..G59, G73, G74, G76, G80, G81, G82, G83, G84, G85, G86, G87, G88, G89, G98, G99, G94, G95, G96, G97, G68.2, G69, G53.1, M98, M99, O, G4, P, L, M19

Machine group	CNC system	Comment	Note	List of implemented commands* *(Commands may be implemented incompletely, see the note)*
	NC210	For simulation and tool path import	An additional license is required	G0, G01, G02, G03, G4, G9, TMR, G90, G91, G17, G18, G19, X, Y, Z, A, B, C, I, J, K, R, M3, M4, M5, M13, M14, M19, M0, M1, M2, M30, M7, M8, M9, T, G94, G95, F, G40, G41, G42, G79, G80, G81, G82, G83, G84
	Sinumerik 840D	For simulation and tool path import		N, G0, G01, G02, G03, G90, G91, G17, G18, G19, X, Y, Z, A, B, C, I, J, K, CR, TURN, F, S, T, M6, M3, M4, M5, M0, M1, M2, M30, M8, M9, G40, G41, G42, G53, G153, SUPA, G54..G57, G70, G71, G700, G710, G58, G59, TRANS, ATRANS, G4, G94, G95, MCALL, M17,

Machine group	CNC system	Comment	Note	List of implemented commands* *(Commands may be implemented incompletely, see the note)*
				TRAORI, TRAFOOF, CYCLE800, AC, IC, CYCLE81, CYCLE82, CYCLE83, CYCLE84, CYCLE840, CYCLE85, CYCLE86, CYCLE87, CYCLE88, CYCLE89, A3, B3, C3, CALL(Subprogram), POCKET4
	Sinumerik 828D EmcoMill E350	For simulation and tool path import	An additional license is required	Sinumerik 840D commands (descendant interpreter), plus M13, M14 (spindle on with coolant)
	Sinumerik 840D MasterMill HPC3	For simulation and tool path import	An additional license is required	Sinumerik 840D commands (descendant interpreter), plus M80, M82 (head selection), M101, H, RTCPON, RTCPOF

Machine group	CNC system	Comment	Note	List of implemented commands* *(Commands may be implemented incompletely, see the note)*
	Tormach PCNC Mach3	For simulation and tool path import		G4, P, G53, G92, G0, G1, G2, G3, X, Y, Z, A, B, C, I, J, K, R, G73, G80, G81, G82, G83, G84, G85, G86, G87, G88, G89, G17, G18, G19, G90, G91, G93, G94, G95, G20, G21, G40, G41, G42, D, G43, G44, G49, H, G98, G99, G54..G59, M0, M1, M2, M30, M3, M4, M5, M8, M9, F, S, T, M98, M99, O, P, L
	Tormach PCNC PathPilot	For simulation and tool path import		G4, P, G53, G92, G0, G1, G2, G3, X, Y, Z, A, B, C, I, J, K, R, G73, G80, G81, G82, G83, G84, G85, G86, G87, G88, G89, G17, G18, G19, G90, G91, G93, G94, G95, G20, G21, G40, G41,

Machine group	CNC system	Comment	Note	List of implemented commands* *(Commands may be implemented incompletely, see the note)*
				G42, D, G43, G49, H, G98, G99, G54..G59, M0, M1, M2, M30, M3, M4, M5, M8, M9, F, S, T, M98, M99, O, P, L
Turning	HNC-8	For simulation and tool path import	An additional license is required	G0, G01, G02, G03, G04, G90, G91, G17,G18, G19, G20, G21, X, Y, Z, U, W, A, B, C, I, J, K, R, S, M3, M4, M5, M0, M1, M2, M30, M7, M8, M9, M6, T, F, G43, G43.4, G44, G49, H, G40, G41, G42, D, G53, G92, G54..G59, G80, G83, G84, G98, G99, G94, G95, G96, G97, G68.2, G69, G53.1, M98, M99, O, G4, M19, G12, G12.1, G13, G13.1, G32, G52, G70, G71, G72,

Machine group	CNC system	Comment	Note	List of implemented commands* *(Commands may be implemented incompletely, see the note)*
				G73, G74, G75, G76
	Mazatrol SmoothC	For simulation and tool path import	An additional license is required	G00, G01, G02, G03, X, Z, U, W, I, K, G17, G18, G19, F, R, G98, G99, G4, S, G96, G97, G50, M3, M4, M5, M9, M0, M1, M30, M8, G52, G54..G59, G53, G28, T, G80, G83, G84, G85, G70, G71, G72, G73, G74, G75, G76, G32
Turn-milling	Fanuc 21i	For simulation and tool path import	An additional license is required	G4, G50, G52, G53, G92, G70, G71, G72, G73, G74, G75, G76, G92, G0, G01, G02, G03, G12.1, G13.1, G112, G113, G7.1, G107, X, Y, Z, B, C, U, V, W, H, I, J, K, R, G32, G96, G97, G98, G99, G20, G21, G40, G41, G42, G80,

Machine group	CNC system	Comment	Note	List of implemented commands* *(Commands may be implemented incompletely, see the note)*
				G83, G84, G85, G87, G88, G89, G28, M50, M51, G54..G59, G17, G18, G19, M0, M1, M2, M30, M3, M33, M4, M34, M5, M35, M8, M9, M98, P, M99, O
	Fanuc 0i EmcoTurn E45	For simulation and tool path import	An additional license is required	Fanuc 21i commands (descendant interpreter) with different numbers: G70, G71 (units), G72..G78 (multiple repetitive cycles), G21 (thread cutting cycle), G94, G95 (feed type), G192 (spindle speed limit), M303, M304, M305 (instead of M33, M34, M35)
	Fanuc 0i Schaublin SA-125-CN	For simulation and tool path import	An additional license is required	Fanuc 21i commands (descendant

Machine group	CNC system	Comment	Note	List of implemented commands* *(Commands may be implemented incompletely, see the note)*
				interpreter) with different numbers: G70, G71 (units), G72..G78 (multiple repetitive cycles), G21 (thread cutting cycle), G94, G95 (feed type), G92 (spindle speed limit), M203, M204, M105 (instead of M33, M34, M35); Q in the G83 cycle is set in micrometers
	FS-8	For simulation and tool path import	An additional license is required	N, G0, G01, G02, G03, G90, G91, G17, G18, G19, X, Y, Z, A, B, C, I, J, K, CR, TURN, F, S, T, G40, G41, G42, G53, G153, SUPA, G54..G57, G70, G71, G700, G710, G58, G59, TRANS, ATRANS, G4, G94, G95, MCALL, M17, TRAORI, TRAFOOF,

Machine group	CNC system	Comment	Note	List of implemented commands* *(Commands may be implemented incompletely, see the note)*
				CYCLE800, AC, IC, CYCLE81, CYCLE82, CYCLE83, CYCLE84, CYCLE840, CYCLE85, CYCLE86, CYCLE87, CYCLE88, CYCLE89, A3, B3, C3, CALL(Subprogram), DIAM90, DIAMON, DIAMOF, LIMS
	HAAS ST20Y	For simulation and tool path import	An additional license is required	G0, G01, G02, G03, G04, G17, G18, G19, X, Y, Z, I, J, K, L, R, C, G40, G41, G42, M19, M3, M133, M4, M134, G96, G97, S, P, M5, M135, M8, M9, G98, G99, G50, M1, M30, G54..G59, G28, G53, T, N, M154, M155, U, V, W, G4, G112, G113,

Machine group	CNC system	Comment	Note	List of implemented commands* *(Commands may be implemented incompletely, see the note)*
				G70, G71, G72, G73, G74, G75, G76, G80, G82, G81, G88, G89, G85, G87, G83, G84, G86, G186, G184, G95, G195, G196, G241, G242, G243
	NC220	For simulation and tool path import	An additional license is required	DFP, EPF, UCG, FIL, SPA, SPP, CLP, G0, G1, G2, G3, G4, G9, TMR, G90, G91, G17, G18, G19, G70, G71, X, Y, Z, A, B, C, I, J, K, R, S, M3, M4, M5, M13, M14, M19, G96, G97, SSL, M41, M42, M43, M0, M1, M2, M30, M7, M8, M9, M6, T, G94, G95, F, G40, G41, G42, G79, G33, G80, G81, G82, G83, G84

Machine group	CNC system	Comment	Note	List of implemented commands* *(Commands may be implemented incompletely, see the note)*
	Okuma OSP-P300	For simulation and tool path import	An additional license is required	G0, G01, G02, G03, G101, G102, G103, G132, G133, G90, G91, G17, G18, G19, G137, G138, G119, G136, X, Y, Z, C, I, J, K, L, F, S, SB, T, M3, M4, M13, M14, M5, M12, M0, M1, M2, M30, M8, M9, M174, M175, M262, M263, M41, M42, M43, M44, M146, M147, M109, M110, G94, G95, G96, G97, G50, G33, G4, G77, G78, G178, G179, G180, G181, G182, G183, G184, M136, G71, G72, G73, G74, G85, G86, G87, M22, M23, M32, M33, M34, M73, M74,

Machine group	CNC system	Comment	Note	List of implemented commands* *(Commands may be implemented incompletely, see the note)*
				M75, G80, G81, G82, E
	Sinumerik 840D	For simulation and tool path import		N, G0, G01, G02, G03, G90, G91, G17, G18, G19, X, Y, Z, A, B, C, I, J, K, CR, TURN, F, S, S2, S3, S4, T, M1, M2, M3, M4, M5, M8, M9, M30, M23, M24, M25, M33, M34, M35, M43, M44, M45, G40, G41, G42, D, G53, G153, SUPA, G54..G57, G70, G71, G700, G710, G58, G59, TRANS, ATRANS, G4, G94, G95, G96, G97, LIMS, DIAM90, DIAMON, DIAMOF, G63, G33, SF, TRAFOOF, MCALL, STOPRE, AC, IC, CYCLE81, CYCLE82, CYCLE83, CYCLE84,

Machine group	CNC system	Comment	Note	List of implemented commands* *(Commands may be implemented incompletely, see the note)*
				CYCLE85, CYCLE86, CYCLE87, CYCLE88, CYCLE89, CYCLE840, CYCLE93, CYCLE95, CYCLE97, CYCLE90, SETMS, SPCO, SPCOF, SPOS, TRANSMIT
Robot	ABB robot	For simulation and tool path import		MoveAbsJ, \Cons, \WObj, MoveLDO, MoveL, \Corr, MoveC, PERS, v5, v10⋯.V7000
	Estun	For simulation and tool path import	An additional license is required	A1..A6, E1, J, MovL P, MovL V, MovC P, MovC V, MovJ J, MovJ V, End, Start
	Fanuc robot (R-30iB controller)	For simulation and tool path import		Joint, Linear, Circular, J1..J6, E1..E3, UFRAME, UTOOL_NUM, J, L, C, FINE, CNT, ACC, X, Y, Z, W, P, R, /POS, /EN, UF,

Machine group	CNC system	Comment	Note	List of implemented commands* *(Commands may be implemented incompletely, see the note)*
				UT, F, N, U, D, T, B, deg, mm, GP1, GP2, /END, /MN, %, sec, msec, mm/sec, cm/min, inch/min, A1..A6
	Kuka robot	For simulation and tool path import		LIN, C_DIS, CIRC, PTP, $VEL.CP, DEF, END, $TOOL, TOOL_DATA, $BASE, BASE_DATA
	Motoman robot	For simulation and tool path import		/JOB, S, L, U, R, B, T, E1..E6, E, ///COMM, ///TOOL, ///USER, //POS, //INST, ///POSTYPE, ///PULSE, ///RECTAN, C, EC, MOVL, MOVJ, MOVC, SMOVL, EIMOVL, EIMOVC, UF, V, VJ, +MOVJ
	Nachi robot (AW Format)	For simulation and tool path import	An additional license is required	MOVEJ L, MOVE C, MOVE L, S, A, H, A1..A6, REM, CALL, STOP, END

Note: All interpreters implement only the list of commands generated by the corresponding postprocessors from the CAM system distribution kit.

"Tool path import only" interpreters do not support matching between an NC program line and the tool path.

> WARNING
>
> When selecting an interpreter, pay attention to its purpose (the Purpose field in the preview pane).
>
> The selected interpreter must be intended for simulation by the NC program. Otherwise, the tool path may be incorrect (offset relative to the workpiece coordinate system, duplicated approaches/retracts, a wrong start position, etc.).

> WARNING
>
> Some commands may be implemented partially and may not have all the parameters needed for operation. Before you start working with the interpreter, please make sure that the command in its current implementation really suits you. If you have any questions, contact your service provider.