



# Machine Developer Help

## Table of Contents

Getting Started	1
Introduction	1-2
User Interface	2-6
Machine Definition Principles	7
How it works	7-8
Nc Code Creation	8-9
Modal Words	9-12
IntelliPrompt	12-14
Member List Icons	14-15
Compiling Machine Definition	16
Compilation	16
Runtime Error	16-17
User Configured Input	18-19
Number Formatting	20-21
Menu and Toolbar Commands	23
File	23
Save Machine Definition File	23
Save As	23
Print	23
Page Setup	23-24
Print Preview	24
Quit	24
Edit	24
Undo	24
Redo	24
Find/Replace	24-26
Goto Line Number	26
Cut	26
Copy	26-27
Paste	27

Delete	27
Comment Selection	27
Uncomment Selection	27-28
Indent	28
Outdent	28
View	28
Navigator	28
Nc Code	28
Error List	28-29
Tools	29
Compile	29
Toggle Bookmark	29
Previous Bookmark	29
Next Bookmark	29
Delete Bookmark	29-30
Outlining	30-31
Toggle All Outlining	31
Toggle Outlining Expansion	31
List Members	31
Parameter Info	31
Quick Info	31
Complete Word	31-32
Upgrade to IMachineDefinition_2_0	32
Debug	32
Show Error Line Number	32
Window	32
Remove Splits	32
Configure Machine Definition	33-34
Glossary	35

## Getting Started

### Introduction

#### Welcome to Machine Developer Help

The Machine Developer is used to create the Machine Definition; the Machine Definition controls all aspects relating to format and style of the NC code generated for the CNC control fitted to a machine tool. The Machine Definition is akin to the term Post Processor as used by other CAD/CAM systems.

The Machine Definition is a term used to describe a group of files, contained in a folder, that make up the complete Machine Definition:

- **Containing folder** - this folder is used to group all the files and folders listed below. The name must match the name of the Machine Definition.
- **Machine Definition File** - this file has a .mdf extension and manages all aspects of the Machine Definition.
- **Source code file** - this file has a .vb extension and contains the *Visual Basic .Net source code* that controls the format and style of the NC code generated. Its name must match the name of the Machine Definition File.
- **Configuration.xml** - this file is used to store the changes made when the user makes a change with the **Configure Machine Definition** dialog box. Do not change the name or extension of this file. If the Machine Definition is not configurable then this file will not be present.
- **Xxxxxxx.chm** - this Help file name must match the name of the Machine Definition. Is displayed when the user chooses the Help command from the Help menu in the Configure Machine Definition dialog box. Not necessarily present.
- **Cache folder** - this folder contains system files and should be of no concern. It can safely be deleted (you must close the Machine Developer first or you will get an error message preventing you from doing so), especially to reduce the overall size of the Machine Definition when e-mailing. It will be recreated the next time you start the Machine Developer.

The Source code file is compiled into memory each time a Part is created or opened, resulting in no additional files needing to be saved or managed.

#### Renaming a Machine Definition

The easiest way to rename a Machine Definition is to use the **Save As** command and use the newly created Machine Definition instead. Alternatively if you want to rename the existing Machine Definition, then you must rename the following:

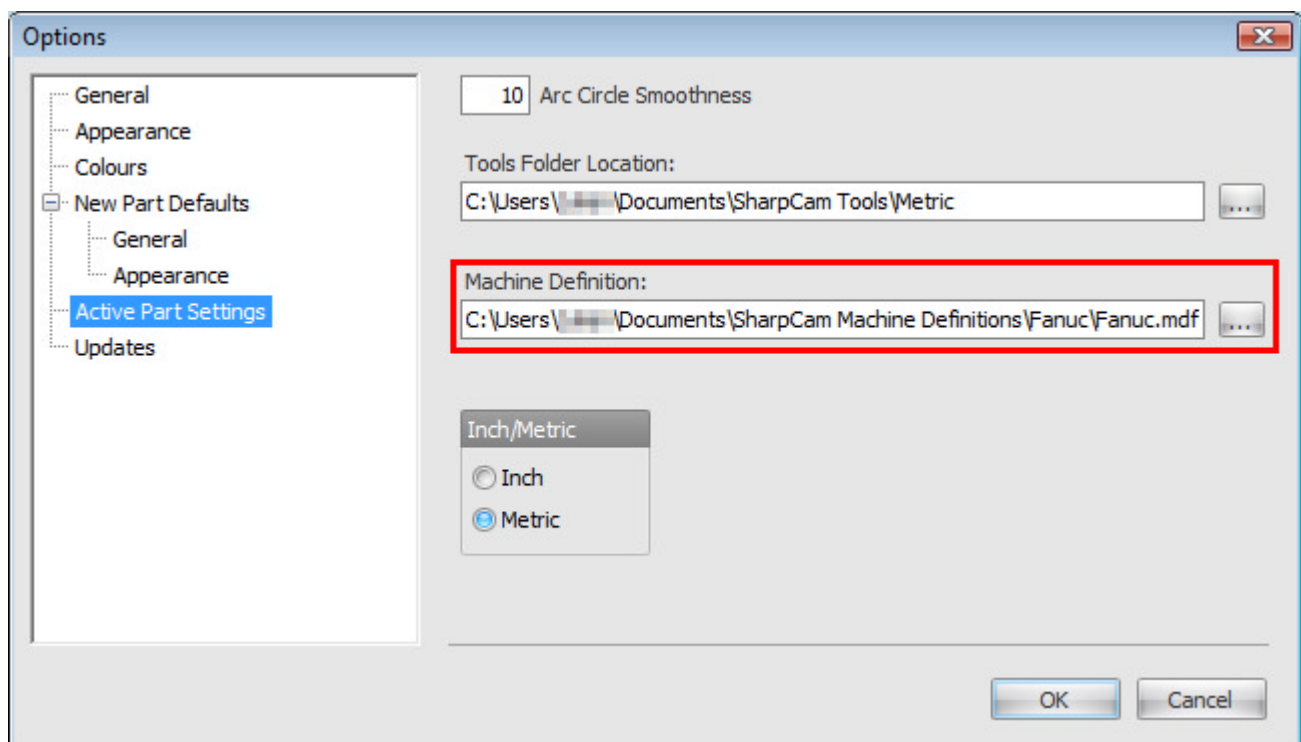
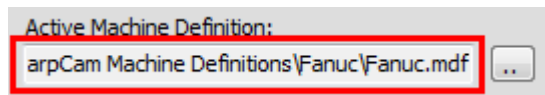
- The Containing folder
- The Machine Definition file
- The Source code file
- The Help file

Use the same name for each one, but do not change the extension of the files. Make a backup before attempting to rename.

## Starting the Machine Developer

The Machine Developer can only be started from within SharpCam using the menu command: Tools -> Machine Developer... This command is only available if a Part exists.

The Machine Definition associated with the active Part is the one that will be opened in the Machine Developer when it starts, as shown on the NC code Tab of the Part Manager or the Active Part Settings in the Options dialog box:

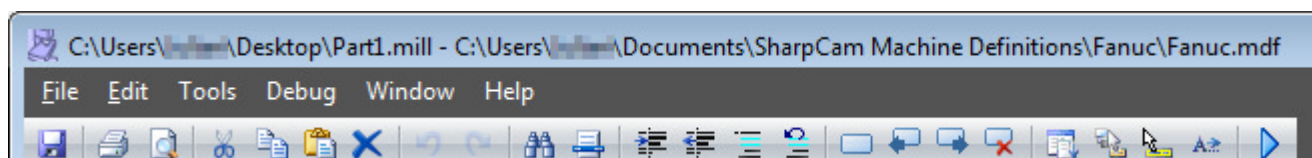


If required a Machine Developer can be started for each open Part, allowing the development of more than one Machine Definition at a time.

## Title Bar

The title bar text comprises of, on the left, the name and location of the Part with which the Machine Developer is associated. Any changes in the Machine Definition will affect this Part.

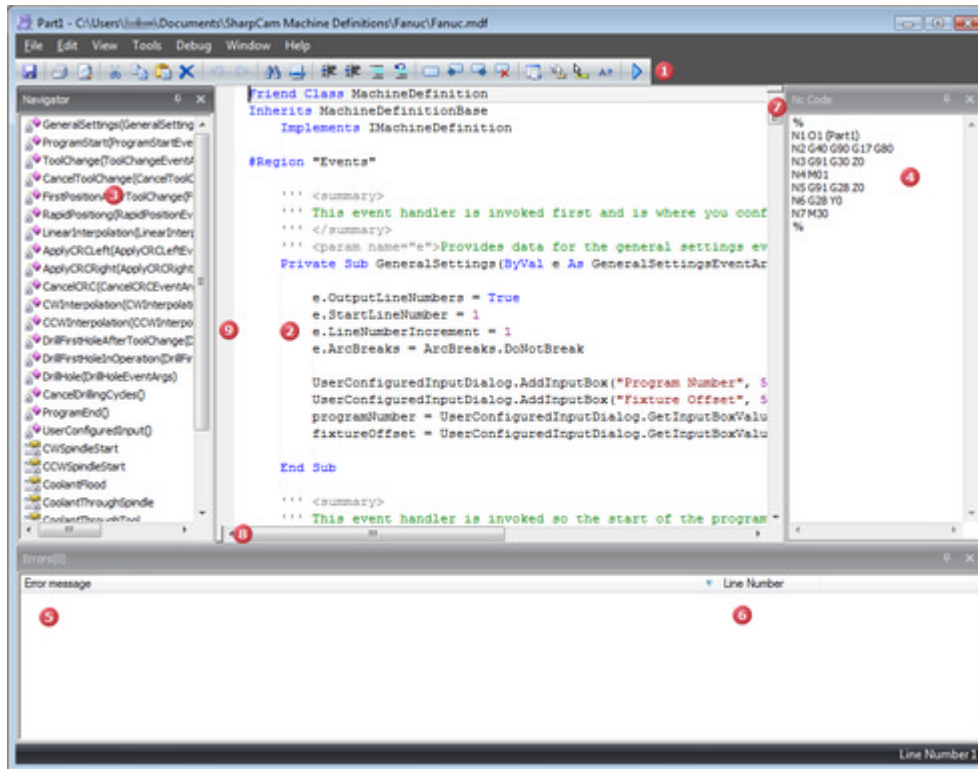
On the right is the name and location of the Machine Definition currently being developed.



## User Interface

## ► User Interface Layout

The various aspects of the Machine Developer interface are detailed below.



### 1 Menus and Toolbars

The menus and toolbars provide access to the Machine Developer commands.

### Code Editor

2 All editing of the *source code* that controls the format and style of the Nc Code takes place here.

### Navigator

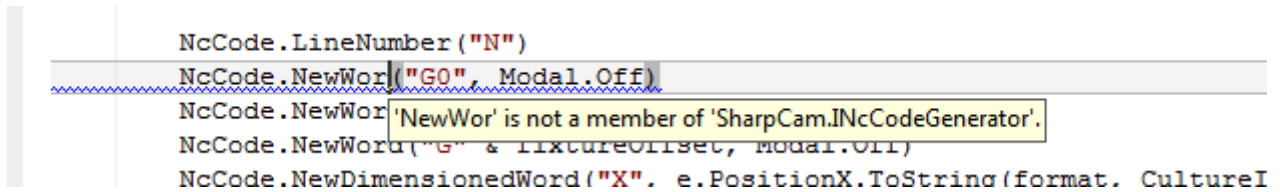
3 This is used to quickly find the Events and Properties that control the format and style of the Nc Code. Click on the relevant description and the Code Editor will display the correct position.

### Nc Code

4 Displays the Nc Code created by the Machine Definition. As you make changes and **re-compile** you can instantly see the effects of editing.

### Compiler Errors

5 Any errors that occur as a result of compiling the code are displayed in this list. Double click an error and the *caret* will be positioned in the Code Editor at the error location. A wavy line will also indicate the location of the error. If the mouse is hovered over the location of the error a tool tip is displayed:



```

NcCode.LineNumber("N")
NcCode.NewWor("GO", Modal.Off)
NcCode.NewWor("G" & FixtureOriset, Modal.Off)
NcCode.NewDimensionedWord("X", e.PositionX.ToString(format, CultureI

```

### 6 Compiler Error Line Number

Displays the line number at which the error occurred.

### Horizontal Splitter

7 Splits the Code Editor horizontally, allowing simultaneous views of different areas. Grab the splitter bar and drag to create a split view.

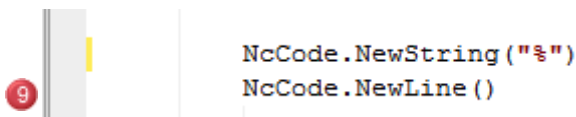
### Vertical Splitter

8 Splits the Code Editor vertically, allowing simultaneous views of different areas. Grab the splitter bar and drag to create a split view.

### Margin

Displays Line Modification Marking:

A yellow vertical bar indicates that the line(s) has been modified but not yet saved.

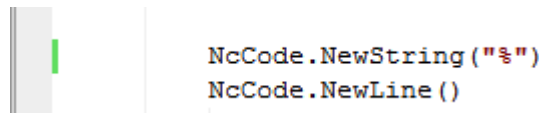


```

NcCode.NewString("%")
NcCode.NewLine()

```

A green vertical bar indicates that the line(s) has been modified and saved.



```

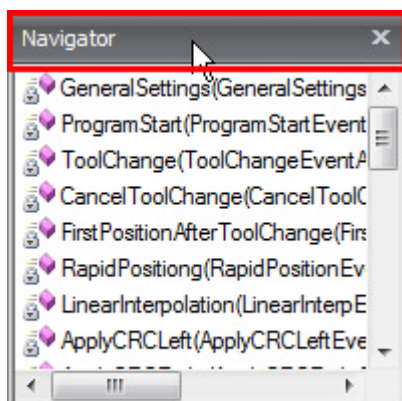
NcCode.NewString("%")
NcCode.NewLine()

```

## ► Organizing Dock Windows

The *Navigator*, *Nc Code* and *Compiler Error* Window can be docked anywhere, or left floating.

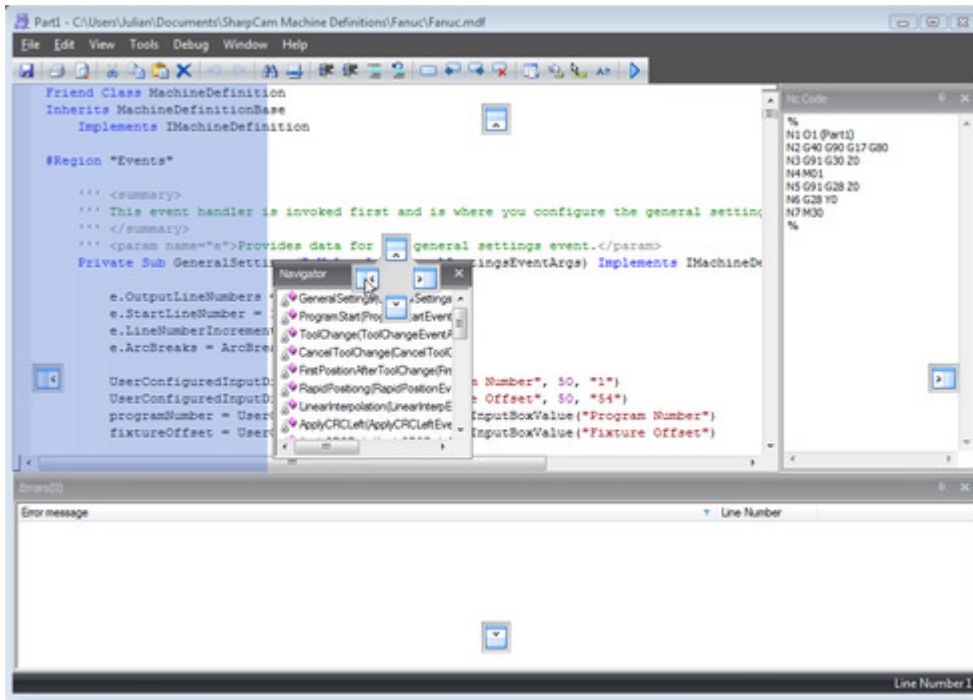
**To dock a Window:** Drag the window caption and move the mouse around the edges of the Machine Developer.



Drop position indicators, illustrating where the dock window can be dropped, are displayed. Moving over the drop position indicators will display a blue shaded area,


# Machine Developer Help

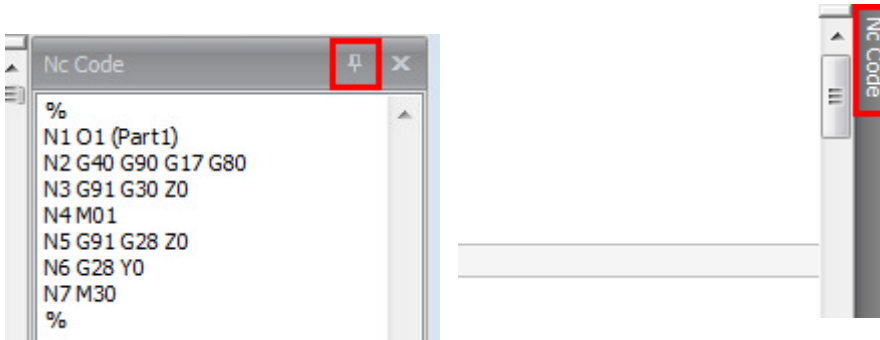
indicating the docking position. Let go of the mouse to dock the window.





There are many combinations of dock positions, experiment until you find a layout that suits you best.

**To float a Window:** Drag the window caption and drop in area that is not on a drop position indicator.

**To collapse a Window:** Click the pushpin button  on the Dock window caption.



To restore a collapsed Dock window, move the cursor over the collapsed Dock window caption. This will cause the Dock window to slide out to its expanded position. Clicking the pushpin button  again causes the Dock window to return to its previously docked position.

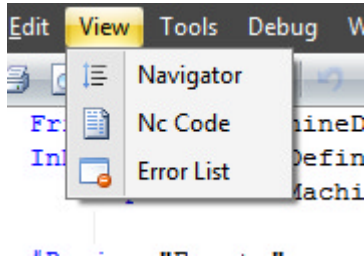
**To Close a Window:** Click the pushpin button  on the Dock window caption.



Once a window is closed the only way to show it again is by using the view menu



command and selecting the window to show:



## Machine Definition Principles

### How it works

#### Understanding how the Machine Definition works



- This topic is of a very technical nature. Users with a back ground in *Visual Basic .Net* programming will feel at home. Nevertheless a competent user should still be able to control a large proportion of the format and style of the Nc Code generated.
- The Visual Basic .Net programming language targeting the Microsoft .Net 2.0 Framework is used to create the Machine Definition. You have full access to the programming language, it is far beyond the scope of this Help file to document all features of this programming language. Features relevant to the Machine Definition will be detailed. There are many books on the subject as well as the internet for additional information.

Three criteria have to be satisfied for the Machine Definition to work:

1. A public class named MachineDefinition must exist: - `Public Class MachineDefinition`
2. This class must inherit from MachineDefinitionBase: - `Inherits MachineDefinitionBase`
3. This class must implement the interface IMachineDefinition\_1\_0 or IMachineDefinition\_2\_0: - `Implements IMachineDefinition_2_0`

The Machine definition works on an event driven principle. An event is when SharpCam calls a Sub procedure (also known as a *Method*). A Sub procedure is a series of Visual Basic statements enclosed by the `Sub` and `End Sub` statements. For example, see the code below, a Sub procedure called `ProgramStart` is defined.

```
''' <summary>
''' This event handler is invoked so the start of the program can be created.
''' </summary>
''' <param name="e">Provides data for the program start event.</param>
''' <remarks>Here is where you should create any preparatory code.</remarks>
Private Sub ProgramStart(ByVal e As ProgramStartEventArgs) Implements IMachineDef

End Sub
```

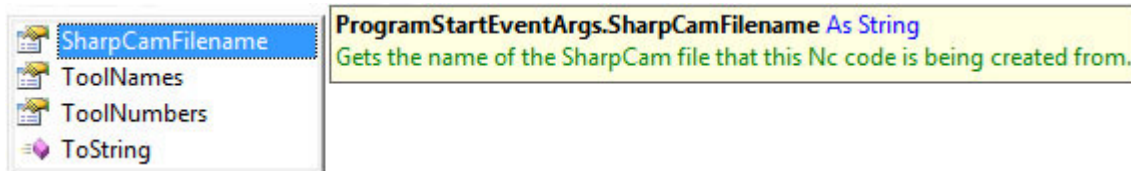
The above Sub procedure implements the **IMachineDefinition\_1\_0** interface as indicated by: `Implements IMachineDefinition_1_0.ProgramStart` It is possible to create Sub procedures that do not implement **IMachineDefinition\_1\_0**, these are never called directly by SharpCam during the Nc Code generation process, but can be used to add functionality to the Machine Definition. The Navigator Window will display all Sub procedures defined in the Machine Definition.

Each time the procedure is called statements are executed, starting with the first executable statement after the `Sub` statement and ending with the `End Sub`. A Sub procedure in this context is also called an Event Handler, because it is called by SharpCam (raises the event) and consequently handles the event. These statements are the instructions that control the format and style of the Nc Code and also output the Nc

Code proper to the Nc Code Tab of the Part manager.

Looking through the Machine Definition you will see many event handlers (that all implement the **IMachineDefinition\_1\_0** interface) which are raised in the appropriate order, depending on the Toolpaths present in the Operations.

Each event handler has an argument passed to it. In the `ProgramStart` event handler the argument is the letter 'e' and is of type **ProgramStartEventArgs**. This argument contains all the information pertinent to the `ProgramStart` event. To access this information type the letter 'e' followed by '.' (Period), then a list of members will be displayed. This feature is called IntelliPrompt Member Lists:



See **Member Lists** for more information.

## Properties

The Machine Definition also contains Properties. Properties allow the Machine Definition to get *String* values for pre defined items.

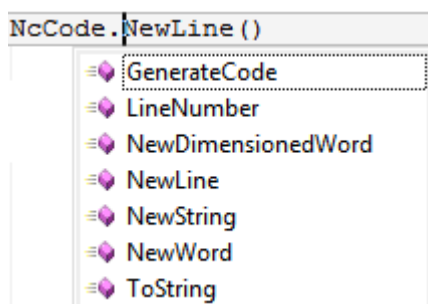
For example the M code for spindle counter-clockwise is shown below. The value to the right of 'Return' is what will appear in the Nc code when a spindle counter-clockwise is required.

```
''' <summary>
''' Gets the Nc code to be output for a clockwise spindle start.
''' </summary>
Private ReadOnly Property CWSpindleStart() As String Implements IMachineDefinition
    Get
        Return "M3"
    End Get
End Property
```

## Nc Code Creation

To actually output Nc Code to the Nc code tab the base class MachineDefinitionBase has a property called **NcCode**, which gives access to methods that create the Nc code.

The NcCode property can be used in any Sub procedure to create the Nc code. Type NcCode followed by a '.' (Period) and an IntelliPrompt Member List will be displayed showing all the methods available:



See topics below for details on each method.

**GenerateCode**

**LineNumber**

**NewDimensionedWord**

**NewLine**

**NewString**

**NewWord**

## Modal Words

### ▶ Modal Words

Most types of Nc code use the concept of modality. This is where, once an instruction has been read and processed, it is in effect until being replaced with another instruction.

An example would be G1, all subsequent moves will be in G1 until told otherwise, say with a G0. For example:

```
G1 X50 Y50
```

```
X100 Y50
```

```
X100 Y0
```

```
G0Z50
```

This code is exactly the same but uses G1 on every line. This is not very efficient.

```
G1 X50 Y50
```


```
G1 X100 Y50
```

```
G1 X100 Y0
```

```
G0Z50
```

SharpCam, if not instructed, will create output like this. There are a few steps required to ensure that the most efficient output is achieved.

### ▶ Modal Absolute Dimensioned Words

 This section refers to Nc instructions that are given in absolute coordinates from the datum.

When an axis is already at a position given in the Nc code then no movement will take place. In the example below the Y50 in all moves after the first are not necessary:

```
G90 G1 X50 Y50
```

```
X100 Y50
```

```
X150 Y50
```

```
X200 Y50
```

```
X250 Y50
```

SharpCam, if not instructed, will create output like this. There are a few steps required to ensure that the most efficient output is achieved. This code is exactly the same but with the unnecessary Y50 removed:

G90 G1 X50 Y50


X100

X150

X200

X250

### ▶ Modal Incremental Dimensioned Words

 This section refers to Nc instructions that are given in incremental coordinates from current position.

When an incremental value of zero is instructed no movement will take place and is not necessary. The most common example of this is when creating arcs that use I and J to define the centre as used by Fanuc controls.

The J0 in the example below is not necessary:

G2 I50 J0

SharpCam, if not instructed, will create output like this. There are a few steps required to ensure that the most efficient output is achieved. This code is exactly the same but with the unnecessary J0 removed:

G2 I50

## Preventing unnecessary repeated Nc code

The Sub procedure **Modals** is where the code is created to prevent the above mentioned issues. All code samples below are in this Sub procedure.

### Modal Words

The concept of word groups is used. A group is a set of words (Nc code instructions) where only one can be active at any given time. The best example is G0, G1, G2, G3.

Firstly create a new group:

Visual Basic

```
Dim wordGroup1 As New ModalWordGroup()
```

Next add each word to the group:

Visual Basic

```
wordGroup1.Words.Add("G0")
wordGroup1.Words.Add("G1")
wordGroup1.Words.Add("G2")
wordGroup1.Words.Add("G3")
```

Finally add the word group to the WordGroups collection:

Visual Basic

```
e.WordGroups.Add(wordGroup1)
```

## Using the group

The group is now created. Any time a word from the group is required in the Nc code the **NcCode.NewWord** method must be used:

## Visual Basic

```
NcCode.NewWord("G0", Modal.On)
```

Now the 'G0' will only be output when required. Should you wish the 'G0' to definitely be output then use **Modal.Off**, do not be tempted to use:

## Visual Basic

```
NcCode.NewString("G0 ")
```

This will cause problems as SharpCam will loose track of state of the group.

## Absolute Dimensioned Words

To prevent duplicate positions and values being output, the address of the value must be added to **ModalEventArgs.AbsoluteDimensionedWords**. For example to add an X position:

## Visual Basic

```
e.AbsoluteDimensionedWords.Add(New ModalAbsoluteDimensionedWord("X"))
```

Any time an X position is required in the Nc code the **NcCode.NewDimensionedWord** method must be used:

## Visual Basic

```
NcCode.NewDimensionedWord("X", e.AbsX.ToString(format, CultureInfo), Modal.On)
```

## Incremental Dimensioned Words

To prevent duplicate positions and values being output, the address of the value must be added to

**ModalEventArgs.IncrementalDimensionedWords**. For example to add a J value:

## Visual Basic

```
e.IncrementalDimensionedWords.Add(ModalIncrementalDimensionedWord("J"))
```

Any time a J value is required in the Nc code the **NcCode.NewDimensionedWord** method must be used:

## Visual Basic

```
NcCode.NewDimensionedWord("J", e.IncrementalCentreY.ToString(format, CultureInfo),  
Modal.On)
```

## When not to use Modal Words

If the word is always required to be output, then there is no need to use any of the above techniques. For example when using G2/G3 on a Fanuc control the R word is always required, even though the previous one is the same. In this case use the **NewString** method:

## Visual Basic

```
NcCode.NewString("R" & e.Radius.ToString(format, CultureInfo) & " ")
```



### *Small arcs*


Problems can occur when the X and Y end point value of a small arc are not output

(assuming they are set as an absolute dimensioned word).

Internally SharpCam creates arcs that can have a length as small as 0.0001. If the X and Y values are rounded to 3 decimal places, SharpCam will not output the X and Y value.

If the Machine Definition creates arcs that use I and J, as on a Fanuc control, you can end up with a situation where the I and J remain but the X and Y values are not output, effectively creating a full 360° circle.

To prevent this problem be sure to set the **MinimumArcLength** property.

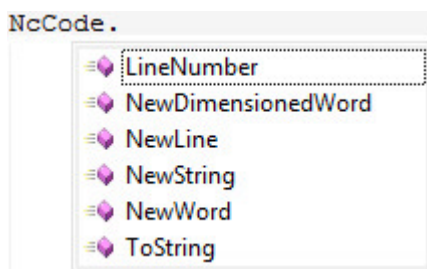
 In the *Visual Basic .NET* code samples the '&' (sometimes called Ampersand) symbol is used to join together the *strings* that appear either side.

## IntelliPrompt

The Machine Developer Code Editor helps you write code with fewer keystrokes and fewer errors by providing lists of the available keywords, variables, and members (methods, properties and events). The Machine Developer Code Editor also completes words as you type your code, you can get all the help you need right in the Code Editor, while you type your code.

### Member Lists

You can display a list of valid members from a type or namespace. When a member from the list is selected, you can press ENTER or TAB to insert that member into your code.




### To use the List Members option

1. Begin typing the name of your object, and then press CTRL+SPACE to use **Complete Word**, which displays the members list box if what you have typed has more than one possible match or no match at all. IntelliPrompt displays all valid members in a scrollable list. For example, you can scroll or use the arrow keys to navigate through the list, or, if you know the first few letters of the member name, begin typing to jump directly to the member in the list.
2. After the name of a class or structure, type the '.' (Period):
3. To insert the selected member in your code, do one of the following:
  1. Press ENTER, TAB, or double-click to insert just the member. If no item is selected in the drop-down menu, press ENTER to insert a blank new line.
  2. Type the character that follows the member, such as open parenthesis, comma, space, or others, to insert the selected member followed by the character that you have just typed.
4. Press ESC at any time to close the Member list.

# Machine Developer Help

When you select an item from the Member list, but before you insert it, you will get **Quick Info** on the item and any code comments for the item.

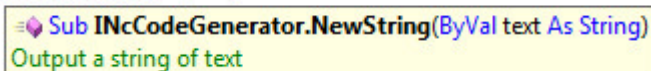
The icon to the left represents the type of the member, such as namespace, class, method, or variable. Please see **Member List Icons** for a listing of list icons.

The Member List can also be invoked from the menu command: Tools -> List Members or from the tool bar button .

## Quick Info

Hover your mouse over any identifier in the code to have the Code Editor show an informational, quick info, tooltip about what is under the mouse. Namespaces, types, members, parameters, and variables are all supported.

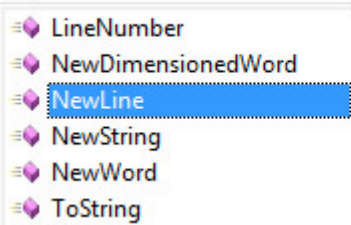
```
NcCode.NewString(" ")
```



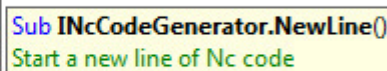
Sub INcCodeGenerator.NewString(ByVal text As String)  
Output a string of text

When you select a member from the Member Lists, Quick Info also appears.


```
NcCode.
```



- LineNumber
- NewDimensionedWord
- NewLine**
- NewString
- NewWord
- ToString



Sub INcCodeGenerator.NewLine()  
Start a new line of Nc code

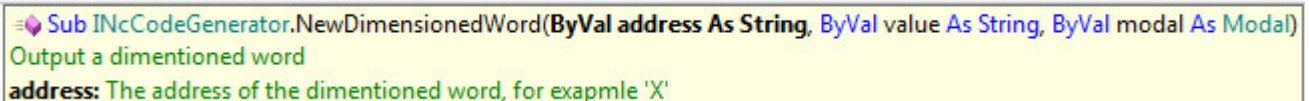
Quick Info can also be invoked from the menu command: Tools -> Quick Info or from the tool bar button .

## Parameter Info

The Parameter Info feature opens the Parameters list to give you information about the number, names, and types of parameters required by a *method*.

The parameter in bold indicates the next parameter that is required as you type the *method*

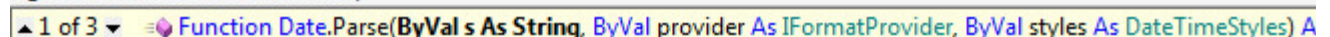
```
NcCode.NewDimensionedWord (
```




Sub INcCodeGenerator.NewDimensionedWord(**ByVal address As String**, ByVal value As String, ByVal modal As Modal)  
Output a dimentioned word  
**address:** The address of the dimentioned word, for exapmle 'X'

For overloaded methods, you can use the UP and DOWN arrow keys to view alternative parameter information for the *method* overloads.

```
System.DateTime.Parse (
```



1 of 3 Function Date.Parse(**ByVal s As String**, ByVal provider As IFormatProvider, ByVal styles As DateTimeStyles) A

Parameter Info can also be invoked from the menu command: Tools -> Parameter Info or from the tool bar button .


## Complete Word

The Complete Word option types the rest of a variable, command, or *method* name, once you have entered enough characters to disambiguate the term. Type the first few letters































of the name, and then press CTRL+SPACEBAR.

If what you have typed has more than one possible match, or no match at all, then Complete Word is invoked to display the pop-up **Member Lists** box, which you can then use to find the term and insert it into your code.

Complete Word can also be invoked from the menu command: Tools -> Complete Word Info or from the tool bar button .

## Member List Icons

-  Assembly
-  Enumeration Item
-  Folder
-  Generic Argument
-  Internal Class
-  Internal Constant
-  Internal Delegate
-  Internal Enumeration
-  Internal Event
-  Internal Extension Method
-  Internal Field
-  Internal Interface
-  Internal Method
-  Internal Property
-  Internal Standard Module
-  Internal Structure
-  Keyword
-  Namespace
-  Operator
-  Private Class
-  Private Constant
-  Private Delegate
-  Private Enumeration
-  Private Event
-  Private Extension Method
-  Private Field
-  Private Interface
-  Private Method


-  Private Property
-  Private Structure
-  Protected Class
-  Protected Constant
-  Protected Delegate
-  Protected Enumeration
-  Protected Event
-  Protected Extension Method
-  Protected Field
-  Protected Interface
-  Protected Method
-  Protected Property
-  Protected Structure
-  Public Class
-  Public Constant
-  Public Delegate
-  Public Enumeration
-  Public Event
-  Public Extension Method
-  Public Field
-  Public Interface
-  Public Method
-  Public Property
-  Public Standard Module
-  Public Structure

## Compiling Machine Definition

### Compilation

As you make changes to the *source code* in the **code editor**, no changes are made to the Nc code. To actually commit these changes a process called compiling has to be performed. This converts the source code into a language that the computer understands.

To perform the compilation do one of the following:

- Choose menu command: Tools -> Compile
- Click the  tool bar button
- Press the F5 key

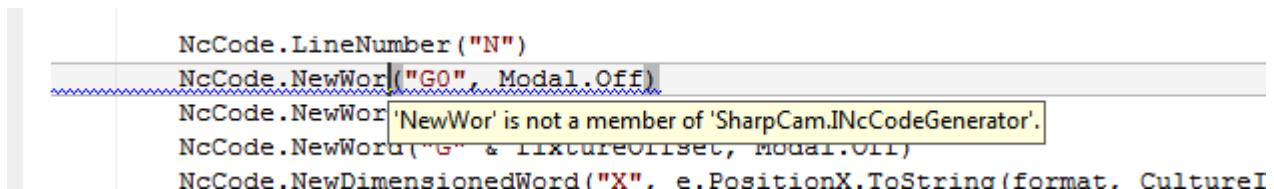
The Machine Definition is saved automatically before compiling.

### Compiler Errors

Any errors that occur as a result of compiling the code are displayed in the error list:

Errors(1)	
Error message	Line Number
'NewWor' is not a member of 'SharpCam.INcCodeGenerator'.	119

Double click an error and the *caret* will be positioned in the Code Editor at the error location:



```

NcCode.LineNumber ("N")
NcCode.NewWor("GO", Modal.Off)
NcCode.NewWor("NewWor' is not a member of 'SharpCam.INcCodeGenerator'."
NcCode.NewWor("G" & FixtureOffset, Modal.Off)
NcCode.NewDimensionedWord("X". e.PositionX.ToString(format, CultureInfo

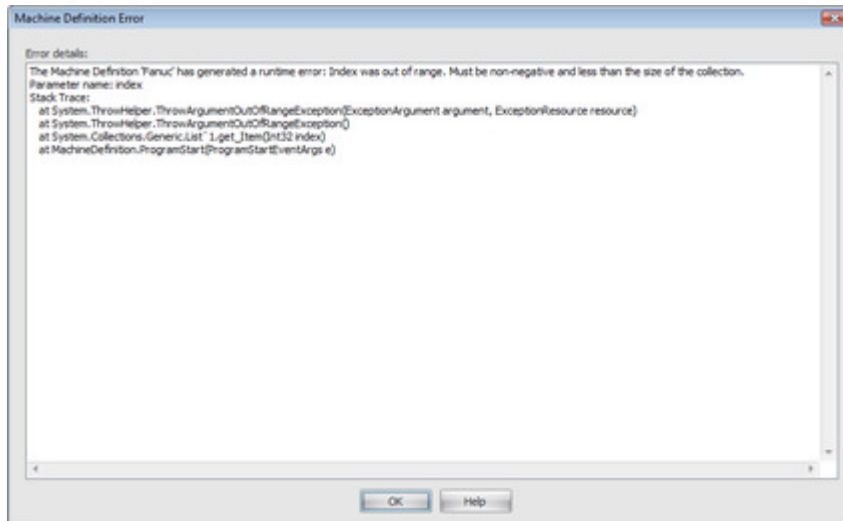
```

A wavy line will also indicate the location of the error. If the mouse is hovered over the location of the error a tool tip is displayed, showing the error message

### Runtime Error

Even after the compilation is successful it is possible for a runtime error to occur. This is when an error occurs during the execution of the instructions that make up the compiled code.

A run time error is displayed in a dialog box:



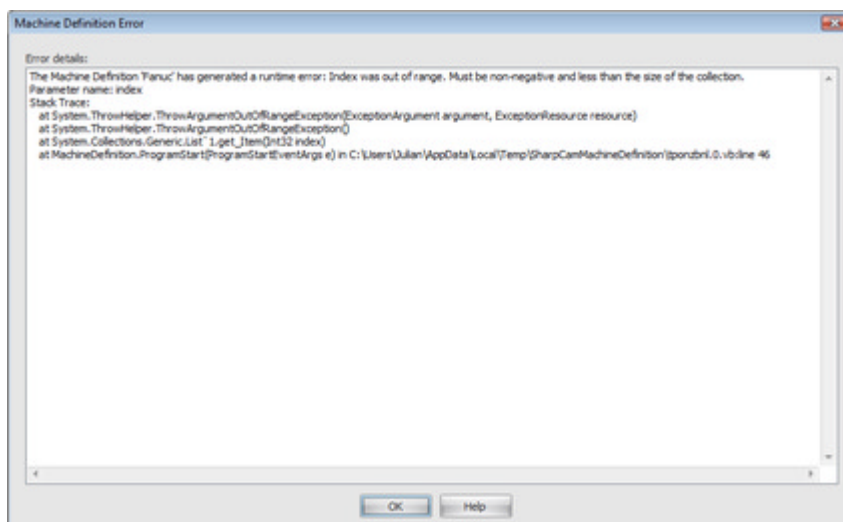
The message will indicate the cause of the error. In the above case the cause is 'Index was out of range. Must be non-negative and less than the size of the collection.'

These errors are created by the *.NET Framework* and therefore beyond the scope of this Help file. Microsoft published all its **documentation** (<http://msdn.microsoft.com/en-gb/default.aspx>) on the internet. Using Google will quickly help you find the cause.

The stack trace indicates the name of the member in the code where the error has occurred. The stack trace may include places not in your code, so look until you find a location that exists in your code.

To help find the exact place, use the menu command: Debug -> Show Error Line Number

Compile the code again and the stack trace will now include the line number on which the error occurred:

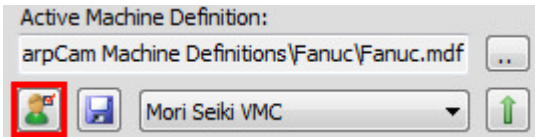


As can be seen the error occurred on line 46. When you have diagnosed the problem it is recommended to turn off the debug facility, as there is an overhead associated with it.

## User Configured Input

There are times when the user needs the ability to change the Nc code output without having to change the Machine Definition. A good example is the program number.

The Nc Code tab has a button called 'User Configured Input':



When this button is clicked, the **UserConfiguredInput** method is invoked. In this event handler any task can be performed.

### Getting information from the user

To make the task easier a property **UserConfiguredInputDialog** in the **MachineDefinitionBase** class is available.

This property is used to manage a dialog box, that allows the user to input values. These values are accessible in code to use as necessary.

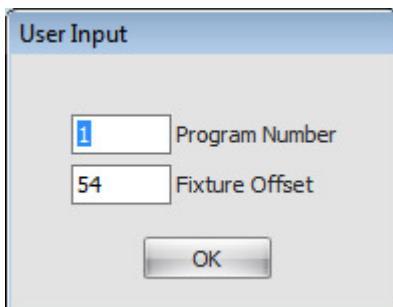
Using the **AddInputBox** method will create an input box in which the user can enter a value. Each call to **AddInputBox** will create a new box below the previous:

#### Visual Basic

```
UserConfiguredInputDialog.UserConfiguredInputDialog.DeleteAllInputBoxes()  
UserConfiguredInputDialog.AddInputBox("Program Number", 50, "1")
```

A call to **DeleteAllInputBoxes** is required before starting the creation of the input boxes.

A typical User Configured Input dialog box:



### Retrieving the data entered by the user

A call to the **GetInputBoxValue** method is required to get access to the value entered by the user:

#### Visual Basic



```
UserConfiguredInputDialog.GetInputBoxValue("Program Number")
```

This method will return the data entered by the user as a string. The string should be stored in a class level variable, so it can be accessed anywhere throughout the Machine Definition.

#### Visual Basic

```
Private programNumber As String
```

```
Private Sub UserConfiguredInput() Implements IMachineDefinition_2_0.UserConfiguredInput  
    UserConfiguredInputDialog.Display()  
    programNumber = UserConfiguredInputDialog.GetInputBoxValue("Program Number")  
    NcCode.GenerateCode()  
End Sub
```

-  A call to **NcCode.GenerateCode** is required to update the Nc code after the user has entered data.
-  When **Display** is called the execution of the Machine definition is halted until the user clicks the OK button.

## Number Formatting

Values such as X, Y and Z positions that are passed to the event handlers are of type *Double*. If you do not perform any formatting then the output to the Nc code will be an exact representation of that number, with many digits after the decimal place.

In order to output the number to the desired format the ToString() method of the number should be called:

### Visual Basic

```
Public Function ToString (format As String, provider As IFormatProvider) As String
```

### format

The format parameter controls how the number is converted to a *String* representation. The .NET Framework provides extensive formatting support. For more details see the **Microsoft Documentation** ([http://msdn.microsoft.com/en-us/library/26etazsy\(VS.80\).aspx](http://msdn.microsoft.com/en-us/library/26etazsy(VS.80).aspx)) on the subject.

There are two types of format specifiers 'standard numeric format strings' and 'custom numeric format string'

### Standard Numeric Format Strings

A standard numeric format string takes the form Axx, where A is an alphabetic character called the format specifier, and xx is an optional integer called the precision specifier.

The precision specifier indicates the desired number of decimal places. If the precision specifier is omitted, the default is used, this typically being 2.

See below for some examples:

Format specifier	Value	Type	String representation
"F"	17843	Double	17843.00
"F3"	-29541	Double	-29541.000
"F"	18934.1879	Double	18934.19
"F0"	18934.1879	Double	18934
"F1"	-1898300.2	Double	18934

### Custom Numeric Format String

A custom numeric format string, which you create and consists of one or more custom numeric format specifiers, defines how numeric data is formatted.

Format specifier	Name	Description
------------------	------	-------------

0		If the value being formatted has a digit in the position where the '0' appears in the format string, then that digit is copied to the result string; otherwise a '0' appears in the result string. The position of the leftmost '0' before the decimal point and the rightmost '0' after the decimal point determines the range of digits that are always present in the result string.
	Zero	The "00" specifier causes the value to be rounded to the nearest digit

placeholder preceding the decimal, where rounding away from zero is always used. For example, formatting 34.5 with "00" would result in the value 35.

The following example displays several values formatted using custom format strings that include zero placeholders.

If the value being formatted has a digit in the position where the '#' appears in the format string, then that digit is copied to the result string. Otherwise, nothing is stored in that position in the result string.

# Digit placeholder Note that this specifier never displays the '0' character if it is not a significant digit, even if '0' is the only digit in the string. It will display the '0' character if it is a significant digit in the number being displayed.

The "###" format string causes the value to be rounded to the nearest digit preceding the decimal, where rounding away from zero is always used. For example, formatting 34.5 with "###" would result in the value 35.

. Decimal point The first '.' character in the format string determines the location of the decimal separator in the formatted value; any additional '.' characters are ignored.

Format specifier	Value	Type	String representation
"00000"	123	Double	00123
"0.00"	1.2	Double	1.20
"00.00"	1.2	Double	01.20
"#.###"	1.2	Double	1.2
"#####"	123	Double	123
"0.0###"	1	Double	1.0
"0.0###"	1.11111111	Double	1.1112

## provider

An **IFormatProvider** (<http://msdn.microsoft.com/en-us/library/system.iformatprovider> (VS.80).aspx) that supplies culture-specific formatting information.

Depending on which language the operating system a PC uses has an effect on the output. One being the character used for the decimal separator. For example in the UK a '.' (period) is used but in Germany a ',' (comma) is used.

A comma is typically not what is required.

If you are using a PC that uses the '.' as a separator then you can omit this argument. Otherwise you must specify a culture that does use '.' as the separator. For convenience the **MachineDefinitionBase** base class has a property **CultureInfo** that sets the correct culture:

### Visual Basic

```
e.PositionX.ToString(format, CultureInfo)
```






## Menu and Toolbar Commands

### File

#### Save Machine Definition File


##### ▶ Accessing the command

Menu: File -> Save Machine Definition File

Toolbar button: 

Shortcut: Ctrl + S

Saves all changes since the last time the Machine Definition was saved.

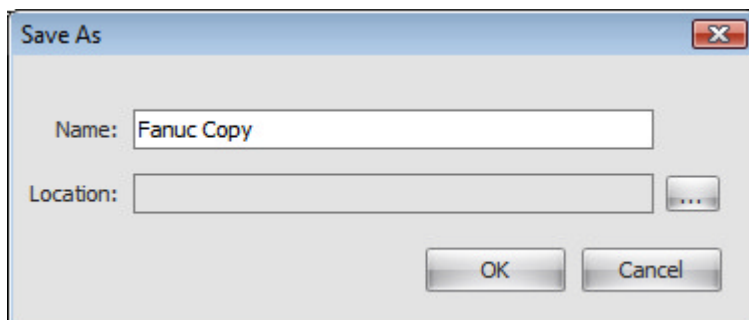
 The Machine Definition is automatically saved whenever it is compiled.


### Save As

##### ▶ Accessing the command

Menu: File -> Save As

Save the current Machine Definition in its entirety to a new Machine Definition with a different name. Choose the location and a new different name, then click OK.




 When a new Machine Definition is required, use this command to create one based on an existing Machine Definition. It is not possible to create an empty Machine Definition.

### Print

##### ▶ Accessing the command

Menu: File -> Print

Toolbar button: 

Print the *source code* for the Machine Definition.

### Page Setup

##### ▶ Accessing the command


Menu: File -> Page Setup

Displays the Page Setup dialog box, as seen in many Windows applications.

## Print Preview

### ▶ Accessing the command

Menu: File -> Print Preview

Toolbar button: 

Displays the Print Preview dialog box, as seen in many Windows applications.

## Quit

### ▶ Accessing the command

Menu: File -> Quit

Quits and closes the Machine Developer.

## Edit

## Undo

### ▶ Accessing the command

Menu: Edit -> Undo

Toolbar button: 


Shortcut: Ctrl + Z

Undoes the last change to the *source code*.

## Redo

### ▶ Accessing the command

Menu: Edit -> Redo

Toolbar button: 

Shortcut: Ctrl + Y

Redoes the last Undo.

## Find/Replace

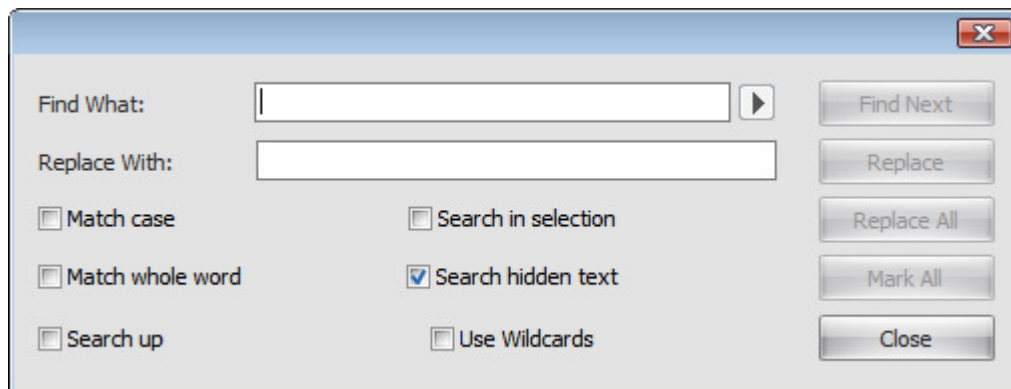
### ▶ Accessing the command

Menu: Edit -> Find/Replace

Toolbar button: 

Shortcut: Ctrl + F

The Find/Replace dialog supports the standard find and replace found in Word processors.



## ► Match case

Check this box to find text that is an exact match, including the case.

## ► Match whole word

Check this box to find text that is a whole word and not just part of a word.

## ► Search up

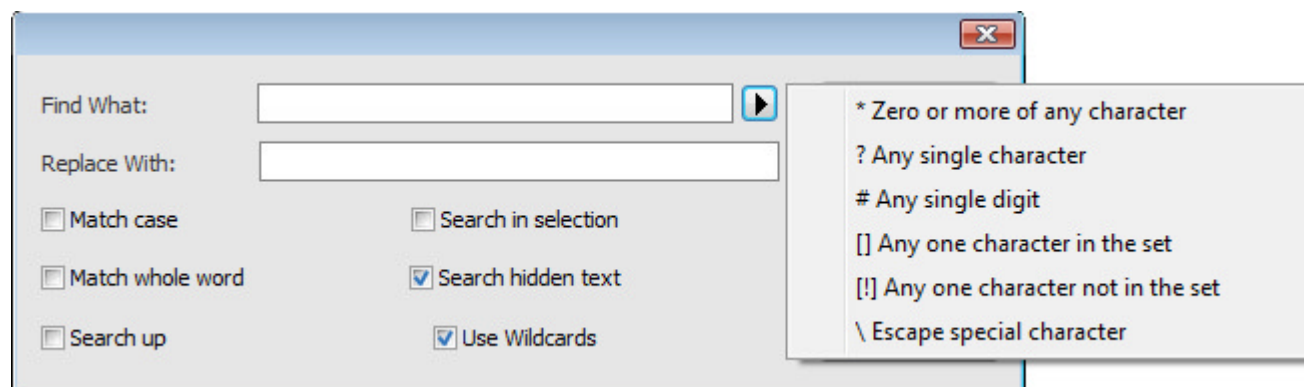
Check this box to search in an up direction as opposed to a down direction.

## ► Use Wildcards

Finds patterns of text.

Check this box to use wildcards when searching. The triangular Expression Builder button next to the 'Find What' box then becomes available. Click this button to display a list of the available wildcards. When you choose any item from the Expression Builder, it is inserted into the 'Find What' string.

Wildcards can also be entered directly into the 'Find What' box if required.



Expression	Syntax	Description
Zero or more of any character	*	Matches zero or more characters. For example, new* matches any text that includes "new", such as newfile.
Any single character	?	Matches any single character.
Any single digit	#	Matches any single digit. For example, 7# matches numbers that include 7

Any one character in the set	[ ]	followed by another number, such as 71, but not 17.
Any one character not in set	[! ]	Matches any one character specified in the set.
Escape special character	\	Matches any one character not specified in the set.
		Matches the character that follows the back slash (\) as a literal. This allows you to find the characters used in wildcard notation, such as * and #.

#### ► Search in selection

Check this box to search only in selected text.

#### ► Search hidden text

Check this box to search in collapsed text when **Outlining** is enabled.

#### ► Find

Finds the text specified in the 'Find What' box.

#### ► Replace

Replaces the text that was found using Find with the text in the 'Replace With' box.

#### ► Replace All

Replaces all occurrences of the text that was found using Find with the text in the 'Replace With' box.

#### ► Mark All


Marks all occurrences of the text that was found using Find with a **bookmark**.

💡 To remove text completely, replace the found text with nothing (Leave the 'Replace With' box empty).

## Goto Line Number

#### ► Accessing the command

Menu: Edit -> Goto Line Number

Toolbar button: 

Shortcut: Ctrl + G

Displays a dialog box that is used to position the *caret* at the given line number.

## Cut

#### ► Accessing the command

Menu: Edit -> Cut

Toolbar button: 


Shortcut: Ctrl + X

As in a standard Word processor, cuts the selected text to the clipboard.

## Copy

### ▶ Accessing the command

Menu: Edit -> Copy

Toolbar button: 


Shortcut: Ctrl + C

As in a standard Word processor, copies the selected text to the clipboard.

## Paste

### ▶ Accessing the command

Menu: Edit -> Paste

Toolbar button: 

Shortcut: Ctrl + V

As in a standard Word processor, pastes text from the clipboard.

## Delete

### ▶ Accessing the command

Menu: Edit -> Delete


Toolbar button: 

Deletes the selected text.

## Comment Selection

### ▶ Accessing the command

Menu: Edit -> Comment Selection

Toolbar button: 

Adds comment symbols to a block of code by selecting one or more lines of code and choosing the Comment command.




As you read the code examples, you often encounter the comment symbol (''). This symbol tells the Visual Basic compiler to ignore the text, or the comment, following it. Comments are brief explanatory notes added to code for the benefit of those reading it.

## Uncomment Selection

### ▶ Accessing the command

Menu: Edit -> Uncomment Selection

Toolbar button: 

Removes comment symbols from a block of code by selecting one or more lines of code and choosing the Uncomment command.



As you read the code examples, you often encounter the comment symbol (''). This symbol tells the Visual Basic compiler to ignore the text, or the comment, following it. Comments are brief explanatory notes added to code for the benefit of those reading it.

## Indent

### ▶ Accessing the command

Menu: Edit -> Indent

Toolbar button: 

Indents the selected lines one tab character. The selected lines are moved to the right.

## Outdent

### ▶ Accessing the command

Menu: Edit -> Outdent

Toolbar button: 

Outdents the selected lines one tab character. The selected lines are moved to the left.

## View

## Navigator

### ▶ Accessing the command

Menu: View -> Navigator

Displays the **Navigator** Window.



Use this command to re-show the Window when it has been closed.

## Nc Code

### ▶ Accessing the command

Menu: View -> Nc Code

Displays the **Nc Code** Window.



Use this command to re-show the Window when it has been closed.

## Error List

## ▶ Accessing the command

Menu: View -> Error List

Displays the **Error List Window**.

💡 Use this command to re-show the Window when it has been closed.

## Tools

### Compile

## ▶ Accessing the command

Menu: Tools -> Compile

Toolbar button: 

Shortcut: F5

See **Compiling the Machine Definition** topic.

### Toggle Bookmark

## ▶ Accessing the command

Menu: Tools -> Toggle Bookmark


Toolbar button: 

Displays a *bookmark* in the left margin next to where the *caret* is positioned. If one already exists then it is removed.

### Previous Bookmark

## ▶ Accessing the command

Menu: Tools -> Previous Bookmark

Toolbar button: 

Moves the *caret* to the previous Bookmark.

### Next Bookmark

## ▶ Accessing the command

Menu: Tools -> Next Bookmark

Toolbar button: 


Moves the *caret* to the next Bookmark.



## Delete Bookmark

### ▶ Accessing the command

Menu: Tools -> Delete all Bookmarks

Toolbar button: 

Deletes all Bookmarks.

## Outlining

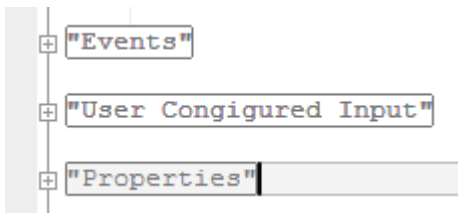
### ▶ Accessing the command

Menu: Tools -> Outlining

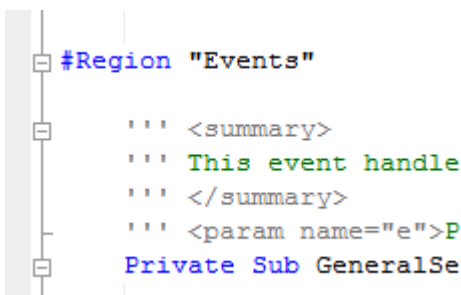
Toggles Outlining on and off.

## Outlining Overview

By default, all text is displayed in the **Code Editor**, but you can choose to hide some code from view by choosing the Outlining command. The Code Editor allows you to select a region of code and make it collapsible, so that it appears under a plus sign (+).



Click the plus sign (+) next to the symbol to expand or hide the region. Outlined code is not deleted, it is hidden from view.



Certain regions are Outlined automatically, for example all Sub Procedures.

It is also possible to create your own regions with the `#Region` keyword.

### Visual Basic

```
#Region "identifier_string"
#End Region
```

### *identifier\_string*

Required. String that acts as the title of a region when it is collapsed.

### **#End Region**

Terminates the `#Region` block.

## Toggle All Outlining

### ▶ Accessing the command

Menu: Tools -> Toggle All Outlining

Toggles all outlining by collapsing or expanding all regions. Only in effect if **Outlining** is in use.

## Toggle Outlining Expansion

### ▶ Accessing the command


Menu: Tools -> Toggle Outlining Expansion

Toggles the outlining by collapsing or expanding the region that the *caret* is in. Only in effect if **Outlining** is in use.

## List Members

### ▶ Accessing the command

Menu: Tools -> List Members

Toolbar button: 

See **Member Lists**.

## Parameter Info

### ▶ Accessing the command

Menu: Tools -> Parameter Info

Toolbar button: 


Shortcut: Ctrl + P

See **Parameter Info**.

## Quick Info

### ▶ Accessing the command

Menu: Tools -> Quick Info


Toolbar button: 

See **Quick Info**.

## Complete Word

► **Accessing the command**

Menu: Tools -> Complete Word

Toolbar button: 

See **Complete Word**.

## Upgrade to IMachineDefinition\_2\_0

► **Accessing the command**

Menu: Tools -> Upgrade to IMachineDefinition\_2\_0

Upgrades the current Machine Definition to version IMachineDefinition\_2\_0.

Version IMachineDefinition\_2\_0 adds 'Sub Routines' functionality.

## Debug

### Show Error Line Number

► **Accessing the command**

Menu: Debug -> Show Error Line Number

See **Runtime Error**.

## Window

### Remove Splits

► **Accessing the command**

Menu: Window -> Remove Splits

Removes window splits created by **Vertical Splitter** and **Horizontal Splitter**.

## Configure Machine Definition

When the user chooses the 'Configure Machine Definition' command from the Tools menu a number of properties are displayed allowing the user to make changes to the Machine Definition without using the **Machine Developer**.

This is primarily intended for internal use by SharpCam but it is still possible to take advantage of this technology yourself.

### Requirements

In order for the Machine Definition to be configurable the following members are required:

- A private field:

Visual Basic

```
Private _configuration As Configuration
```

- A public class:

Visual Basic

```
Public Class Configuration
```

```
End Class
```

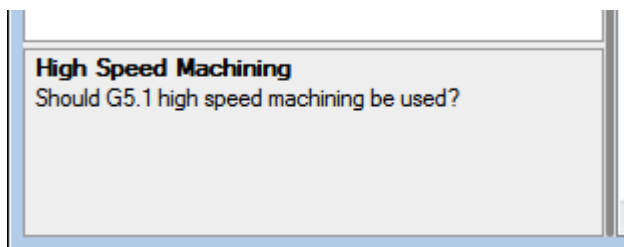
The Configure Machine Definition dialog box will display any public properties contained in the *Configuration* class. A private field of the same type is also required:

Visual Basic

```
Private _highSpeedMachining As Boolean
Public Property HighSpeedMachining() As Boolean
Get
Return _highSpeedMachining
End Get
Set
_highSpeedMachining = value
End Set
End Property
```

### Decorating the Properties with Attributes

When the user clicks each property, help can be displayed in the description pane:



Attributes are use for this purpose:

Visual Basic

```
<System.ComponentModel.Description("Should G5.1 high speed machining be used?")
```

The display name can also be set:

### Visual Basic

```
System.ComponentModel.DisplayName("High Speed Machining")>
```

## How it works

When the Machine Definition is compiled, SharpCam uses the **Configuration.xml** file to instantiate the *\_configuration* field with the values stored in the file. Therefore it is important that you do not set any properties in the Machine Definition.

When the user makes a change to a property, the *\_configuration* object is immediately updated and is written to the Configuration.xml file.

Every thing is taken care of by SharpCam and all you need to concern yourself with, is using the value of the properties of the *\_configuration* object to make the necessary changes to the Nc Code created by the Machine Definition.

You are encouraged to study existing Machine Definitions to help understand the process.

## Versioning

When the Version command from the Help menu is chosen a field call *version* is searched for in the Machine Definition, if found it is displayed in a Message Box.

### Visual Basic

```
Private version As System.Version = New System.Version("1.0")
```

## Glossary

### &

#### &

Joins two strings together.

### .

#### **.NET Framework**

The .NET framework is part of Windows and provides a controlled environment for developing and running applications.

### B

#### **Bookmark**

A blue rectangle in the left hand margin that is used to mark a specific place in the source code.

### C

#### **Caret**

A vertical, flashing bar used as a pointer for entering text.

### D

#### **Double**

Represents a double-precision floating-point number.

### M

#### **Method**

A Method is a series of statements that are executed when called. Methods allow us to handle code in a simple and organized fashion. There are two types of methods in VB .NET: those that return a value (Functions) and those that do not return a value (Sub Procedures).

### S

#### **source code**

A collection of statements or declarations written in a human-readable computer programming language.

#### **String**

Represents text as a series of Unicode characters.

### V

#### **Visual Basic .NET**

An object-oriented computer language implemented on the Microsoft .NET framework.

## Index

- Comment Selection, 27
- Compilation, 16
- Compile, 29
- Complete Word, 31-32
- Configure Machine Definition, 33-34
- Copy, 26-27
- Ctrl, 23
- Cut, 26
- Delete, 27
- Delete Bookmark, 29-30
- Error List, 28-29
- Find/Replace, 24-26
- Goto Line Number, 26
- How it works, 7-8
- IMachineDefinition, 7-8
- Indent, 28
- IntelliPrompt, 12-14
- Introduction, 1-2
- List Members, 31
- Member List Icons, 14-15
- Modal Words, 9-12
- Navigator, 28
- Nc Code, 28
- Nc Code Creation, 8-9
- Next Bookmark, 29
- Number Formatting, 20-21
- Outdent, 28
- Outlining, 30-31
- Page Setup, 23-24
- Parameter Info, 31
- Paste, 27
- Previous Bookmark, 29
- Print, 23

Print Preview, 24  
Quick Info, 31  
Quit, 24  
Redo, 24  
Remove Splits, 32  
Runtime Error, 16-17  
Save As, 23  
Save Machine Definition File, 23  
Show Error Line Number, 32  
Toggle All Outlining, 31  
Toggle Bookmark, 29  
Toggle Outlining Expansion, 31  
Uncomment Selection, 27-28  
Undo, 24  
Upgrade to IMachineDefinition\_2\_0, 32  
User Configured Input, 18-19  
User Interface, 2-6