Mitsubishi Electric
CNC Software Interface Module (MEL-SIM)

Version 1.1

MEAU – Engineering Group

Content

1 Introduction 1
2 Overview 1
3 The MEL-SIM API Architecture 2
4 The NCMachine Class 3
5 The NCMachine Class Properties 4
5.1 Machine Programs 4
5.1.1. Programs.Load Method 5
5.1.2. ExecutionStatus Property 5
5.1.3. Programs Indexer 5
5.2 BitSelection 6
5.3 Auxiliary Axes 7
5.4 Feed Rates 8
5.5 Machine Axes 9
5.5.1. Axis Servos 10
5.5.2. Axis Positions 11
5.6 Machine Functions 12
5.7 Controller I/O 13
5.7.1. Analog Input and Output 13
5.7.2. PLC I/O 14
5.8 Machine Operation 15
5.9 Machine Spindles 16
5.10 Machine Times 17
5.11 Machine Tool Offsets 18
5.12 Machine Variables 19
5.13 Workpiece Offsets 20
5.14 Reference Position Return Values 21
5.15 Simple Properties 21
5.15.1. CommTimeout 21
5.15.2. IPAddress 21
5.15.3. NCNumber 21
6 NCMachine Methods 22
6.1 Machine Version Information 22
6.2 Machine Alarms 23

6.3 Network Parameters 24
6.4 Other Machine Parameters 25
7 Machine File Management 26

Document Management Information

Date	Version No.	Document Name	Revision	Revised By
January 3, 2013	1.0	MELSIM	Release for internal review	MEAU Custom Solutions Center
April 8, 2013	1.1	MELSIM	Formal Release	MEAU Engineering Group

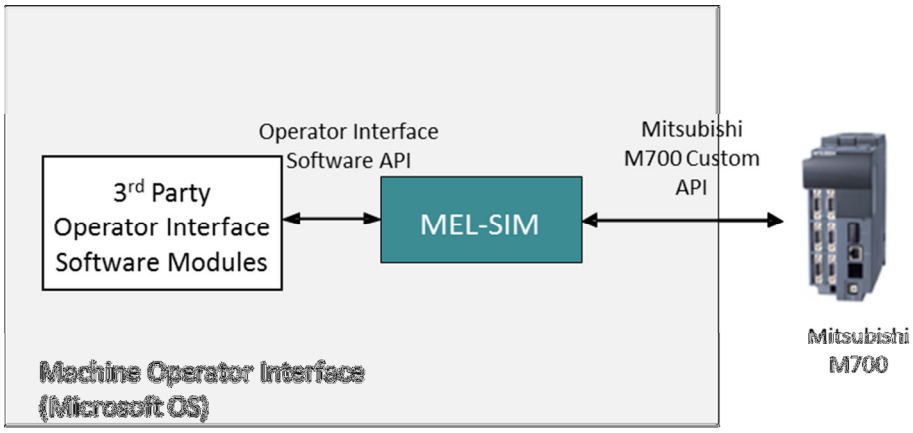
1 Introduction

Mitsubishi Electric CNC Software Interface Module (MEL-SIM) provides connectivity between a third-party operator interface frontend running on a Microsoft Windows operating system and a Mitsubishi Electric M70 or M700 CNC.

The key functions of MEL-SIM are:

- providing connectivity to Mitsubishi CNC models M70 or M700 through the use of Mitsubishi APIs in the M70/M700 Series Custom API Library;
- transferring commands from the third-party operator interface to the CNC for execution;
- providing CNC status, data and parameters to the third-party operator interface for display and other uses.

The objective is that any machine OEM who has a PC-based (i.e. running on Microsoft Windows Operating System using .NET and/or COM components) operator interface front end can utilize MEL-SIM to connect the front end to a Mitsubishi M70/M700 for machine applications.



The purpose of this document is to provide the description of the MEL-SIM .NET Application Programmer's Interface (API) for the Mitsubishi M700 Machine Controller so that a user can develop the necessary front end .NET application to interface with the Mitsubishi CNC for specific applications.

It is assumed that the users of MEL-SIM are familiar with the .NET environment and .NET application development as well as with CNC operations in general.

2 Overview

MEL-SIM is a Microsoft .NET application and is capable of running on Windows XP as well as 64 or 32 bit Windows 7 environment. MEL-SIM utilizes the Mitsubishi M70/M700 Custom NCAPI's to perform the actual low-level communicate with the CNC hardware and acts as a .NET wrapper to enable the Mitsubishi M700 Custom API Library to execute in the .NET environment.

Thus MEL-SIM will only function properly when the Mitsubishi M700 Custom API Library is present on the same computer where MEL-SIM is loaded. **The Mitsubishi M70/M700 Custom API Library will need to be installed separately** by the user.

MEL-SIM is a provided as a *dynamic-link library* file (i.e. a dll file). A user simply adds the dll file as a reference to his or her own application and deploys it with his or her completed application.

MEL-SIM contains a generic software interface with specific APIs (MEL-SIM APIs) that can be used by applications to retrieve information from and send commands to the CNC. The MEL-SIM API set isolates the customer's .NET applications from interfacing with the M700 Custom API Library directly and enables customers' existing Windows Applications to be used with minimal modifications.

MEL-SIM supports the following functions and capabilities:

- Configuring the CNC through MEL-SIM to perform desired functions;
- Transferring commands to the CNC to start and stop appropriate CNC operations;
- Receiving status and error data and/or messages and providing them to the OEM applications for processing and/or display.
- Downloading programs and profiles to the CNC controllers.

3 The MEL-SIM API Architecture

The MEL-SIM is an API centered on a single object, the NCMachine. A user application should create **one and only one NCMachine instance** and use the methods and properties of the instance for accessing the desired NC system.

Generally speaking, the NCMachine class exposes a wide array of properties to provide a logical, type-safe mapping to NC parameter values that are typically of interest to an application programmer. "Property Read" operations are translated to real-time (i.e. not cached) calls to the M70/M700 Custom APIs to read parameters and values from the CNC, and these parameters and values are rarely cached or pre-fetched.

IMPORTANT NOTE

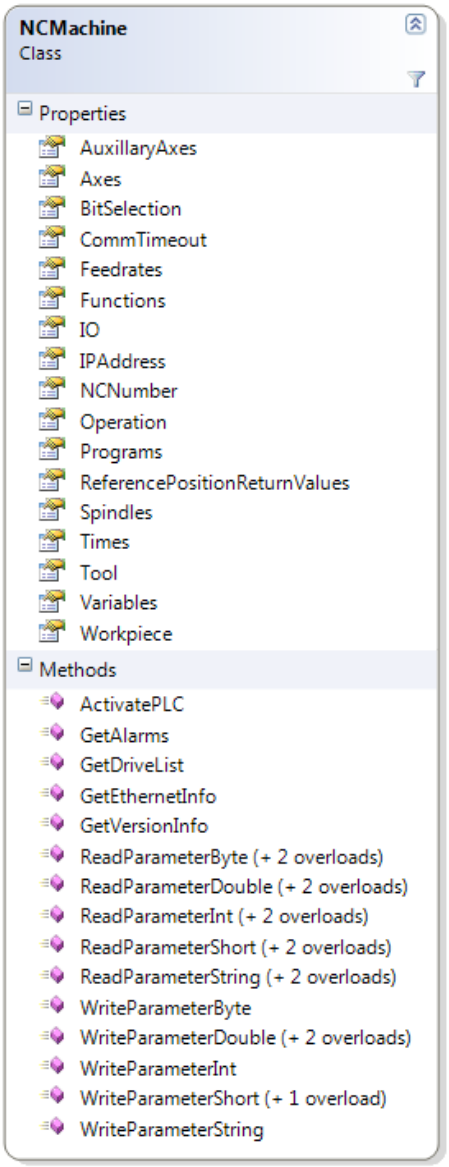
It is critical to understand the capabilities of the target NC machine and verify that all hardware is connected and operational before using the MEL-SIM API. Using the MEL-SIM API without ensuring proper connection and operation (e.g. querying servo properties with no physical servos connected) can cause the M700 Custom API subsystem to deadlock in some cases. This will require a reset of the NC, the PC running the application, or both.

4 The NCMachine Class

The NCMachine class is the main entry point to the MEL-SIM API. A single instance of the NCMachine should be created and this one instance should be used for all application access to the target NC. Having multiple instances in one program is not a tested or supported scenario.

The NCMachine instance is created by simply passing the target NC number into the NCMachine constructor.

```
NCMachine machine = new NCMachine(1);
```

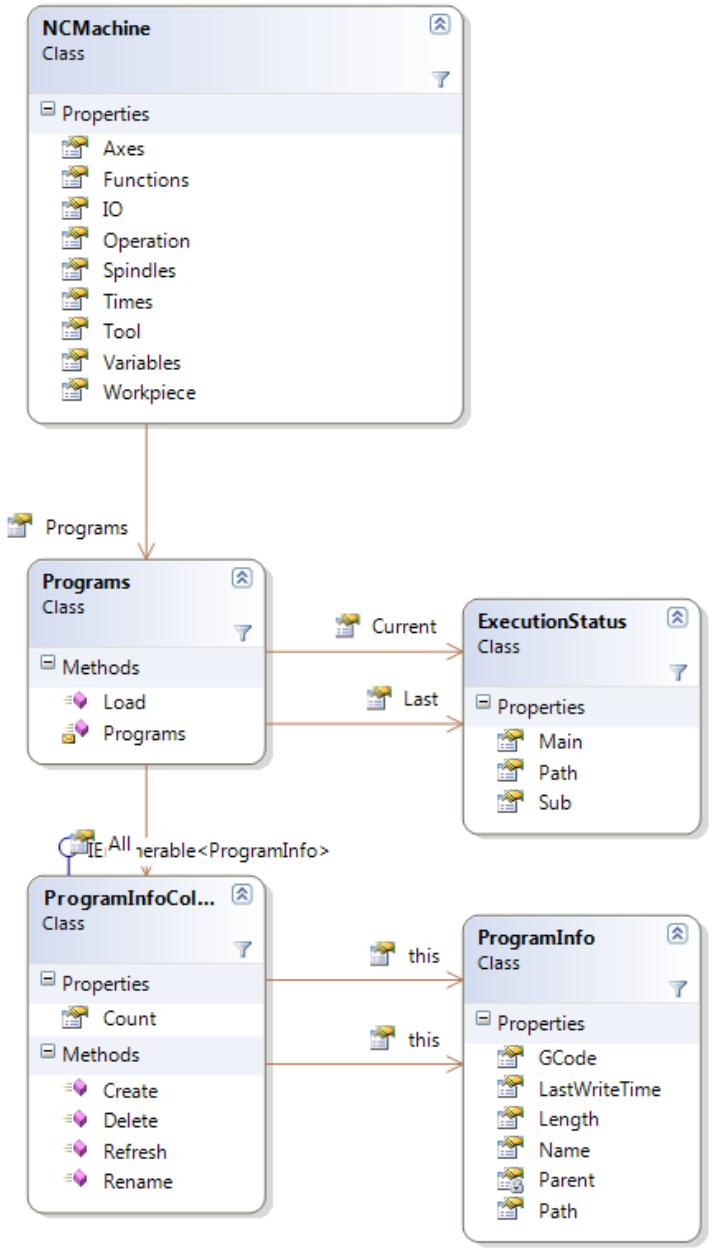


5 The NCMachine Class Properties

This section describes the Properties of the NCMachine class.

5.1 Machine Programs

The **NCMachine.Programs** Property provides access to the in-memory GCode programs of the controller. In-memory programs can be loaded, iterated and properties such as "last write time" and "size" can be queried. In addition, information such as the path, active main and active sub can be queried for the current and last program to be executed.



5.1.1. Programs.Load Method

The Programs property contains a Load method, which can be used to load new Programs into memory from the NC File System (covered in Section 7 in this document). Simply call Load with the full path to the program to load.

5.1.2. ExecutionStatus Property

The Programs Property contains two sub-properties: Current and Last. These properties contain information about the execution status of the respective program such as its main program, sub program and path.

5.1.3. Programs Indexer

The Programs list contains indexers by either index number or program name, allowing an application to query information about any given program in memory, such as the program's current GCode, Length, Name and Path. The indexer can also be used to Create, Delete, Rename or Refresh a given Program.

5.2 BitSelection

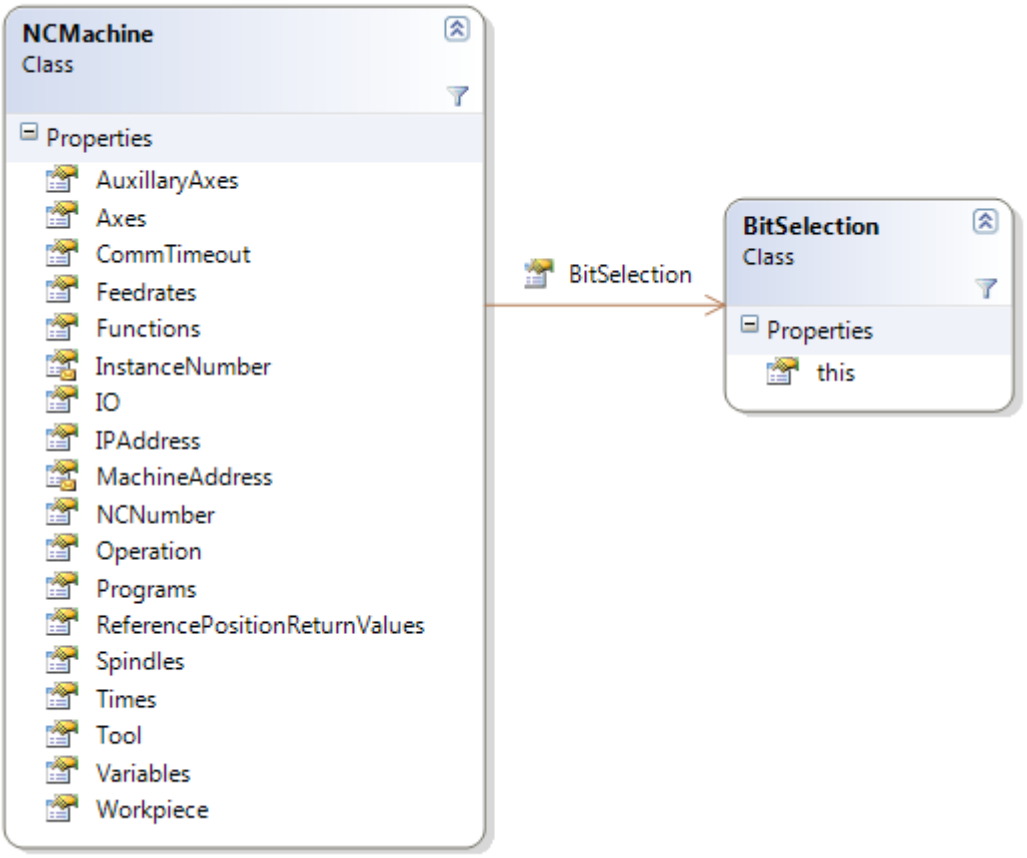
The **NCMachine.BitSelection** Property provides access to the NC Controller's Bit Selection parameters (6401 through 6448). Bit Selection parameters are accessed by either the parameter number or by zero-based index.

Following is an example of querying Parameter 6401 by Parameter number:

```
var value = machine.BitSelection[6401];
```

This is the functional equivalent of querying the same information by index zero:

```
var value = machine.BitSelection[0];
```

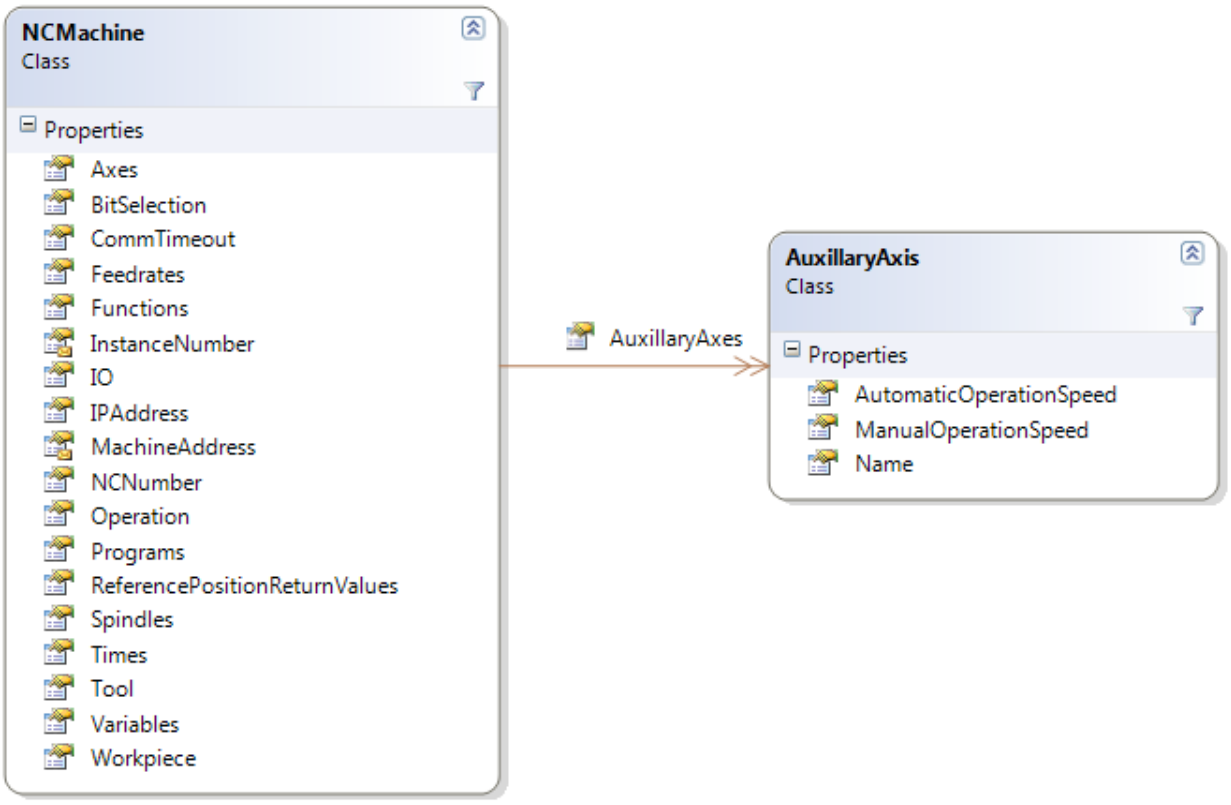


5.3 Auxiliary Axes

The **NCMachine.AuxiliaryAxes** Property provides access to all Auxiliary Axes of the Controller, if they exist. Auxiliary axes are more limited in the amount of information they can provide compares to an Axis in the Axes collection.

Following is an example of querying the manual operation speed of the first Auxiliary axis:

```
var speed = machine.AuxiliaryAxes[0].ManualOperationSpeed;
```

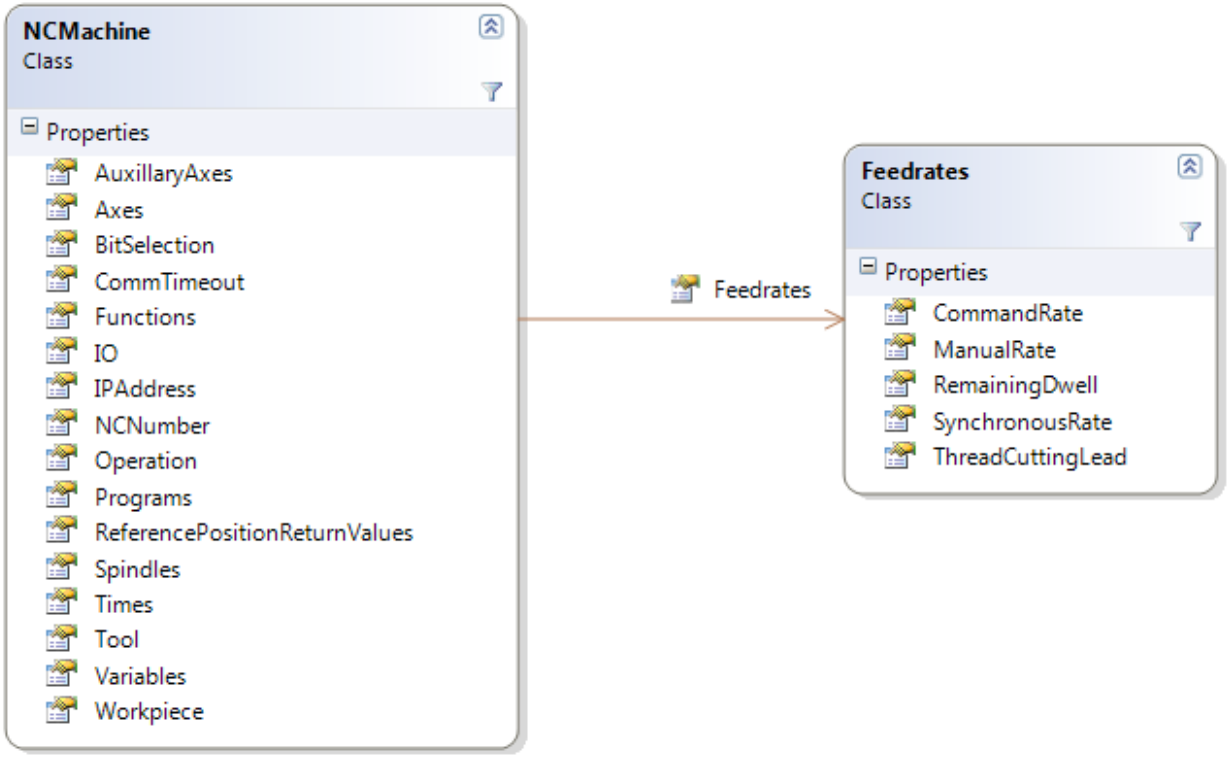


5.4 Feed Rates

The **NCMachine.Feedrates** Property provides read access to the feedrate parameters of the Controller.

Here is an example of querying the manual feedrate of the selected Controller:

```
var feedRate = m.Feedrates.ManualRate;
```

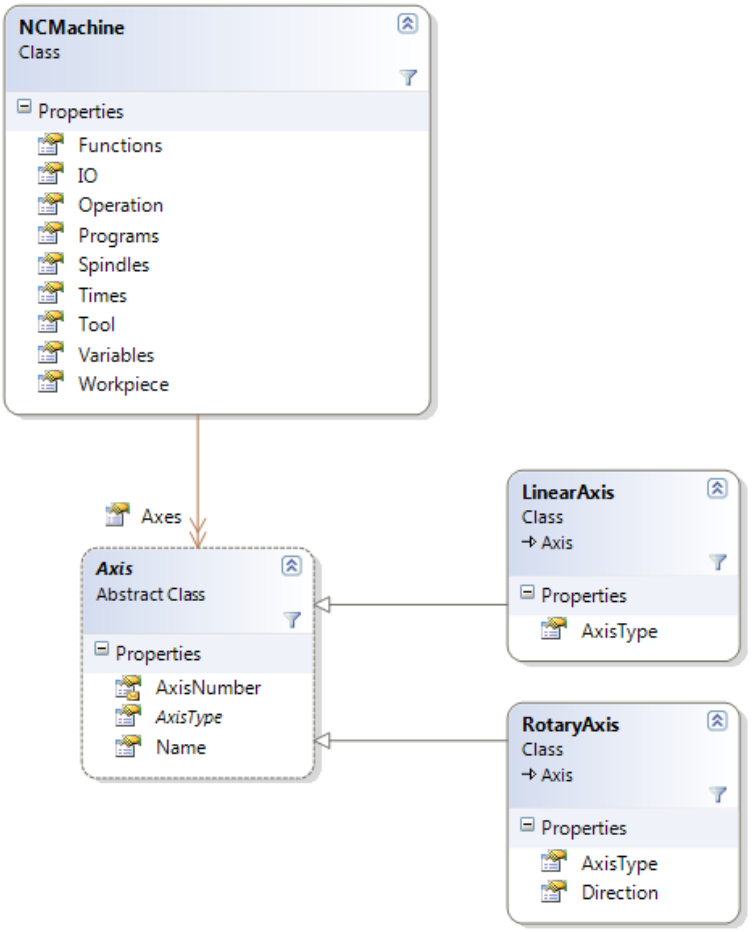


5.5 Machine Axes

The **NCMachine.Axes** collection Property and its sub-Properties provide access to a large amount of information on the axes of the controller. Each physical Axis has a corresponding LinearAxis or RotaryAxis in the collection that can be referenced by 0-based axis number, or by case-insensitive axis name.

Here is an example of querying a direct property of an Axis:

```
RotaryDirection direction = ((RotaryAxis)machine.Axes["X"]).Direction;
```

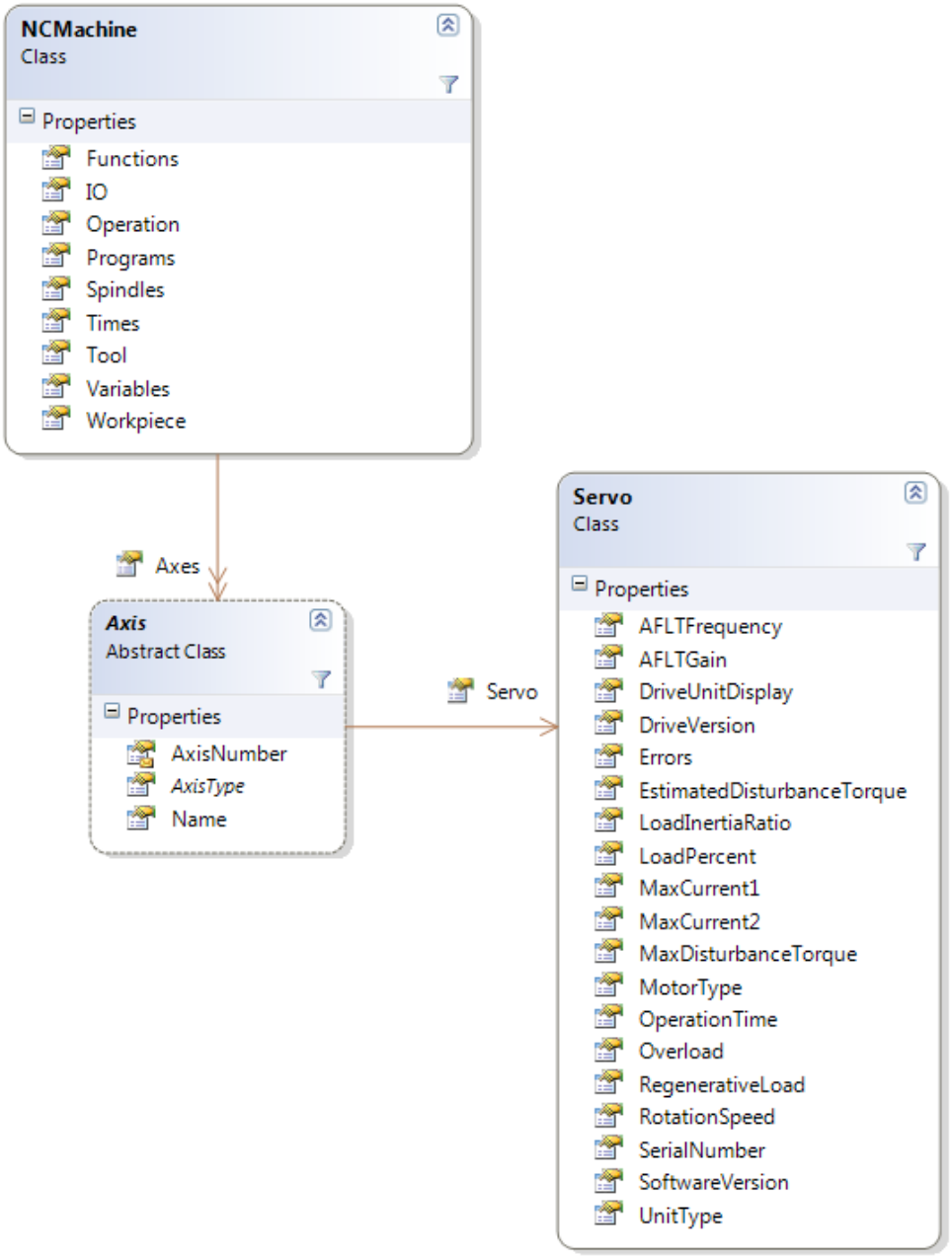


5.5.1. Axis Servos

Each Axis in the Axes collection contains a Servo class that provides information specific to the physical servo for that Axis.

Here is an example of querying an **Axis Servo** Property:

```
var servoLoad = machine.Axes["A"].Servo.LoadPercent;
```

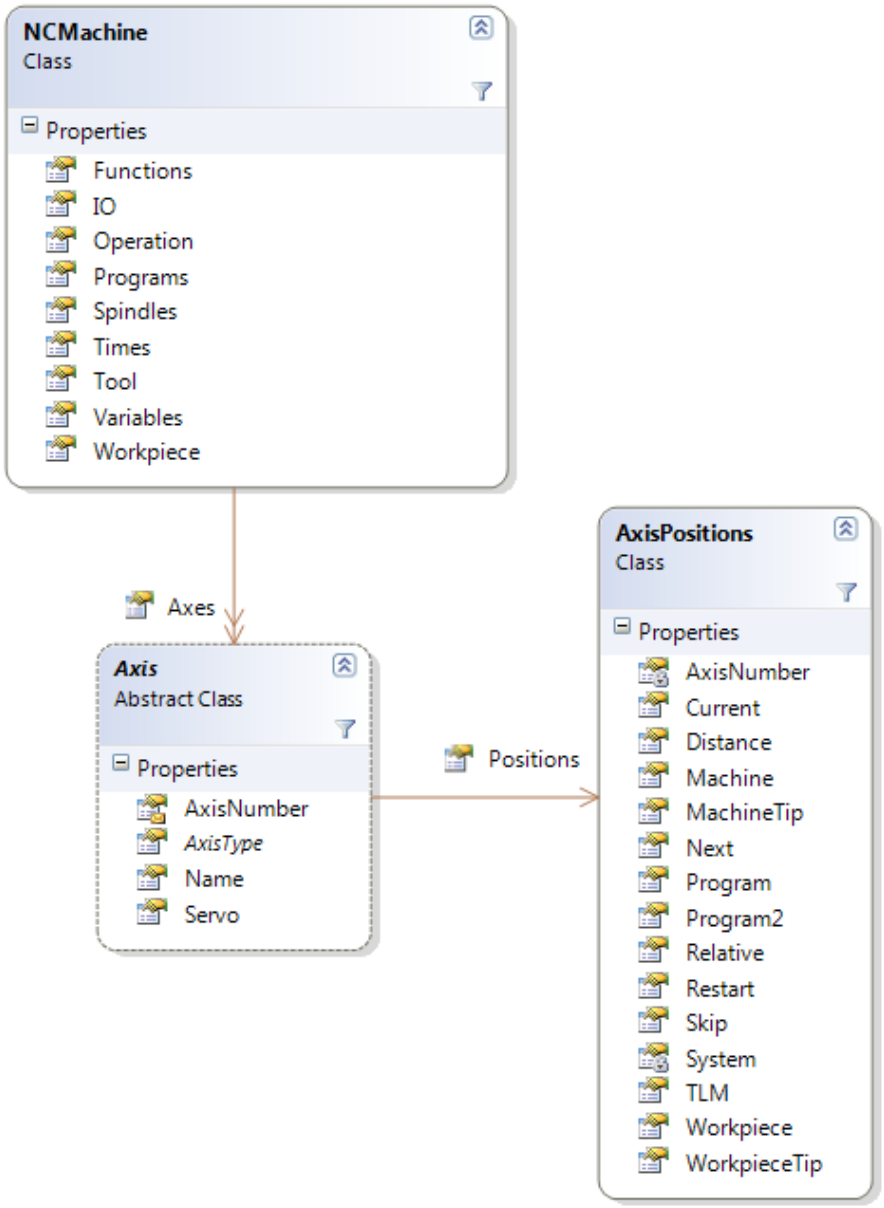


5.5.2. Axis Positions

Each Axis in the Axes collection contains a Positions class that provides positional information specific to that Axis

Here is an example of querying an **Axis Position** Property:

```
var currentPosition = machine.Axes["A"].Positions.Current;
```

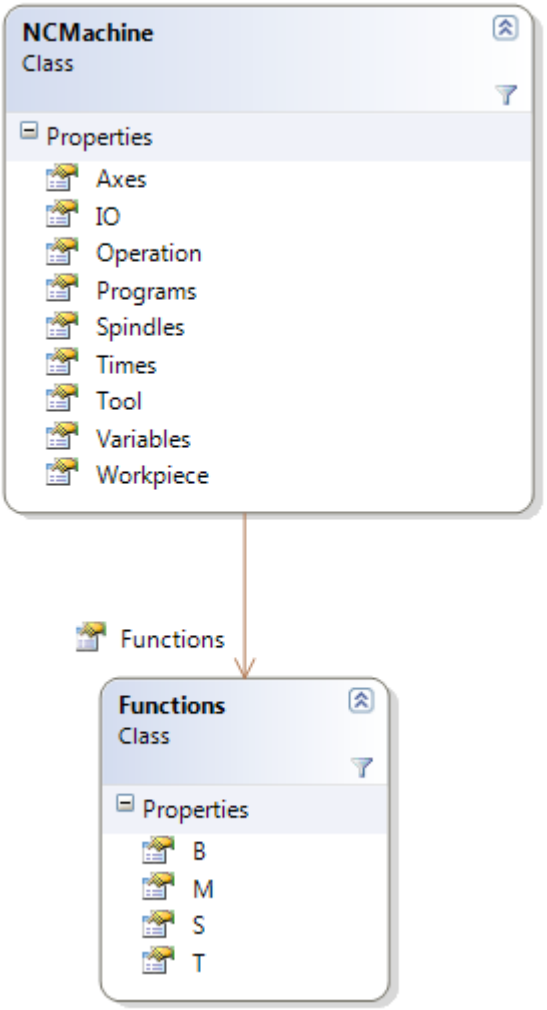


5.6 Machine Functions

The **NCMachine.Functions** Property contains MSTB information about the NC.

Here is an example of querying MSTB information from the connected NC:

```
double m3 = machine.Functions.M[3];
```



5.7 Controller I/O

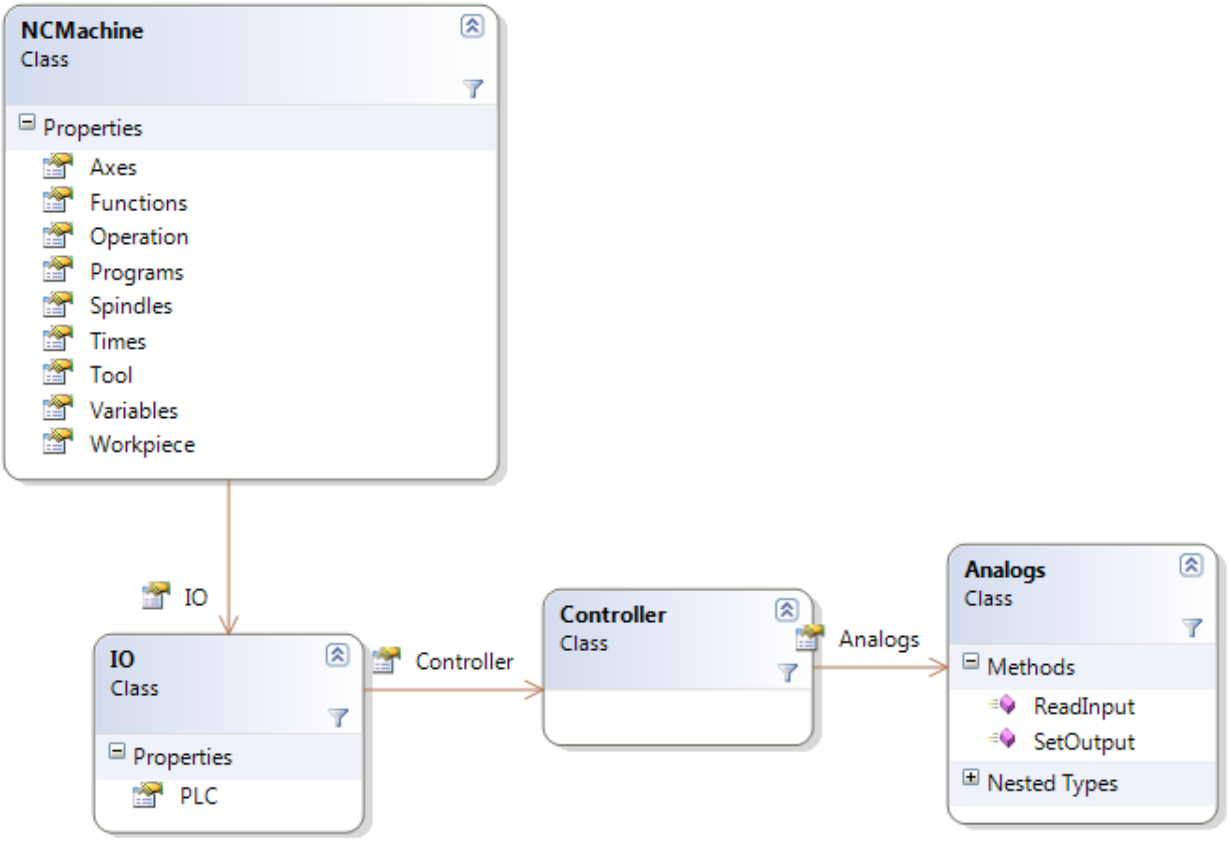
5.7.1. Analog Input and Output

Access to Controller analogs, both input and output, is done through methods on the **NCMachine.IO.Controler.Analogs** Property.

Here are examples of both reading and writing Controller analogs:

```
double ai1 = machine.IO.Controller.Analogs.ReadInput(Analogs.InputPort.AI1);

machine.IO.Controller.Analogs.SetOutput(Analogs.OutputPort.AO2, 3.14);
```



5.7.2. PLC I/O

Access to PLC I/O, both input and output, is done through methods and properties on the **NCMachine.IO.PLC** Property.

Here is an example of reading a PLC switch:

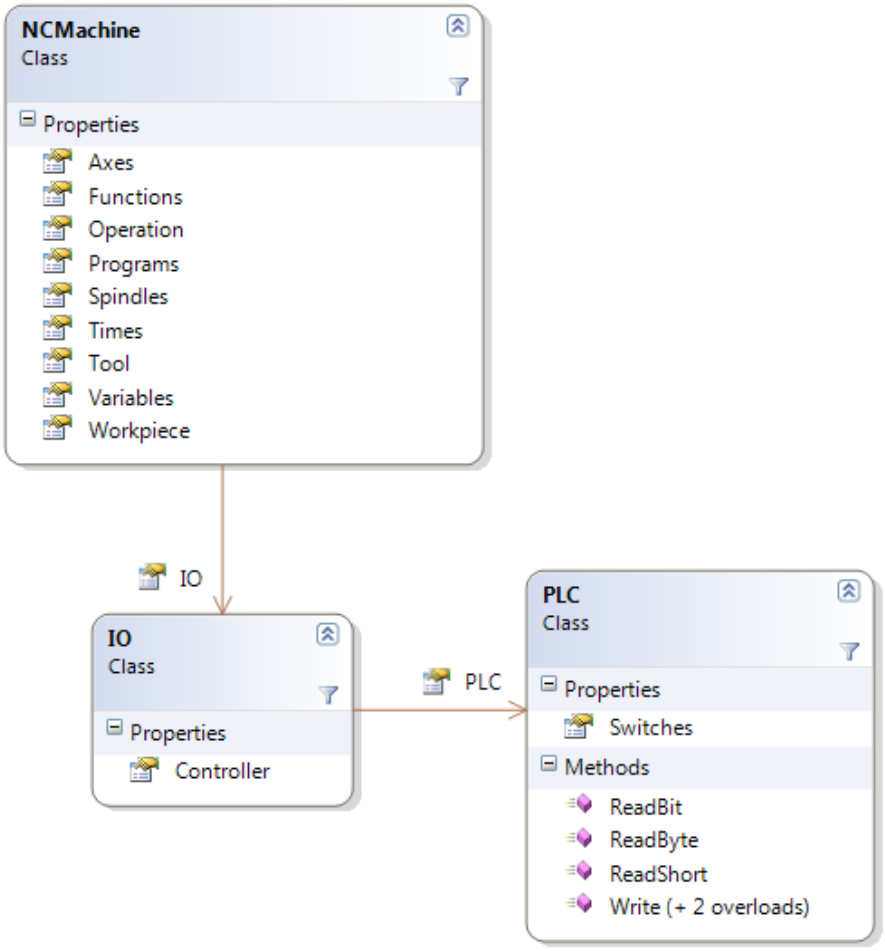
```
bool switch16isOn = machine.IO.PLC.Switches[16];
```

Here is an example of writing a 16-bit value to register 8 of Device D:

```
machine.IO.PLC.Write(IODevice.D, 8, (short)3);
```

Here is an example of reading an 8-bit value from register 1 of Device M:

```
machine.IO.PLC.ReadByte(IODevice.M, 1);
```

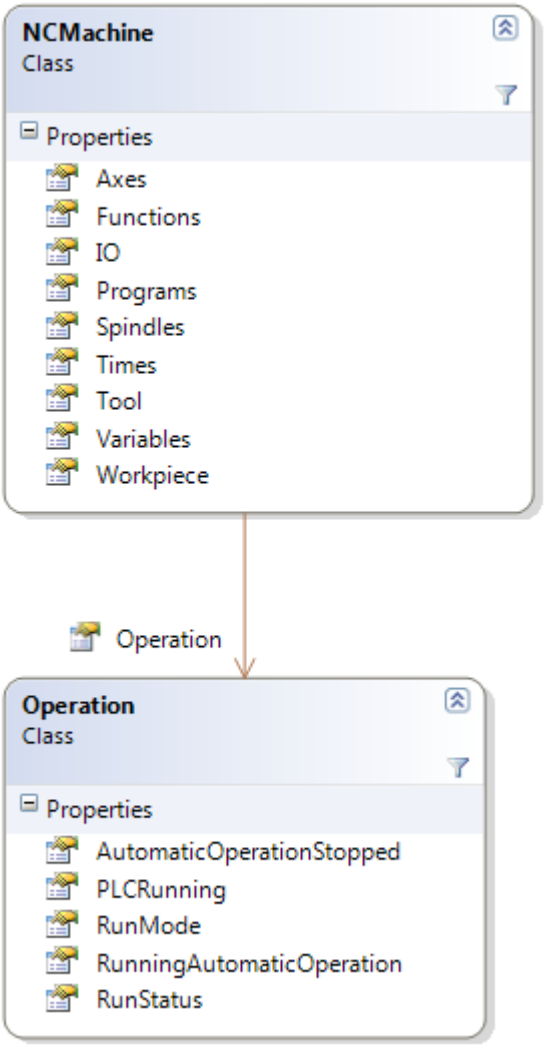


5.8 Machine Operation

Machine operation status values can be queried through sub-properties of the **NCMachine.Operation** Property.

Here is an example of determining if the PLC is currently running:

```
var plcIsRunning = machine.Operation.PLCRunning;
```



5.9 Machine Spindles

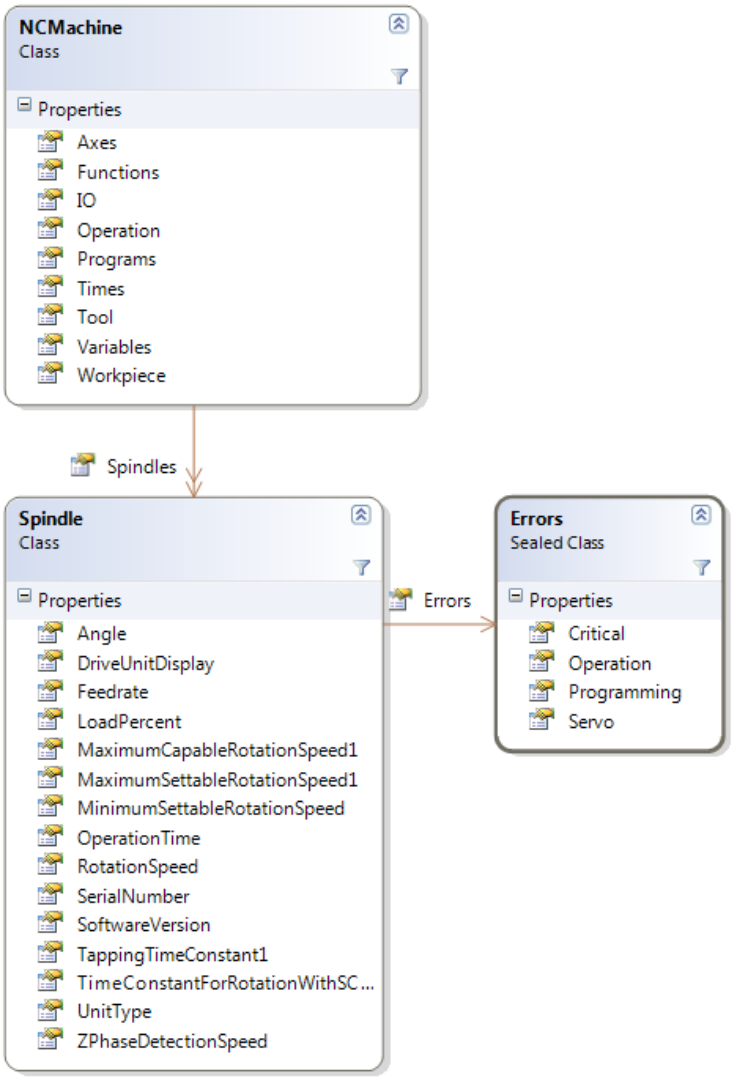
The **NCMachine.Spindles** collection Property provides access to information related to machine spindles. Each physical spindle has a corresponding Spindle in the collection that can be referenced by 0-based spindle number.

Here is an example of querying a direct property of a Spindle:

```
var firstSpindleFeedrate = machine.Spindles[0].Feedrate;
```

Spindle errors are available through the Errors property of the NCMachine.Spindle. Here is an example of reading a Spindle error:

```
string error = machine.Spindles[0].Errors.Critical;
```

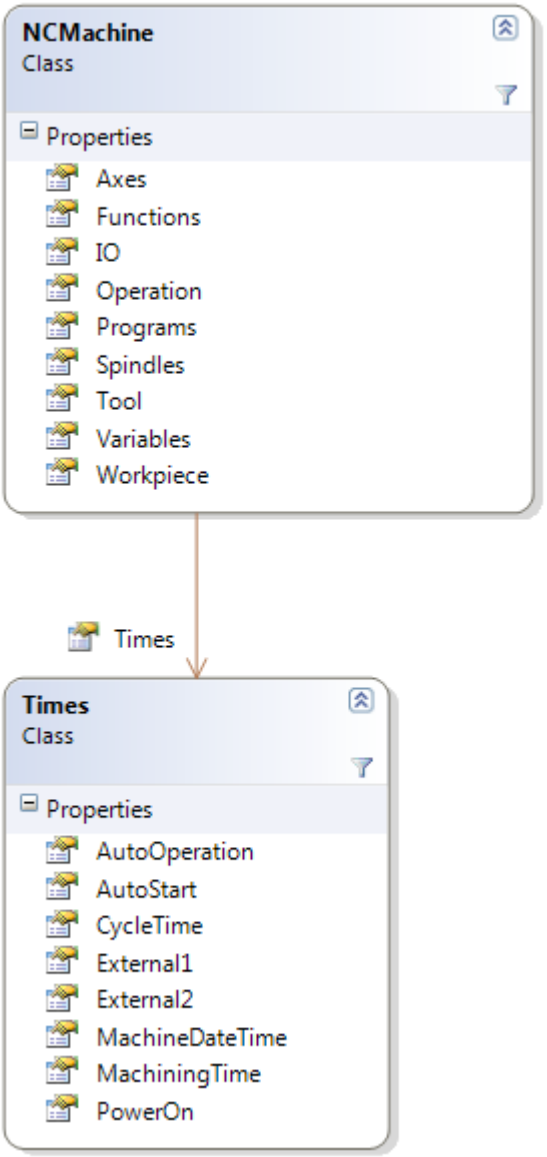


5.10 Machine Times

Machine-related times can be queried through sub-properties of the **NCMachine.Times** property of the NCMachine.

Here is an example of querying the current machine time (clock):

```
var machineTime = machine.Times.MachineDateTime;
```

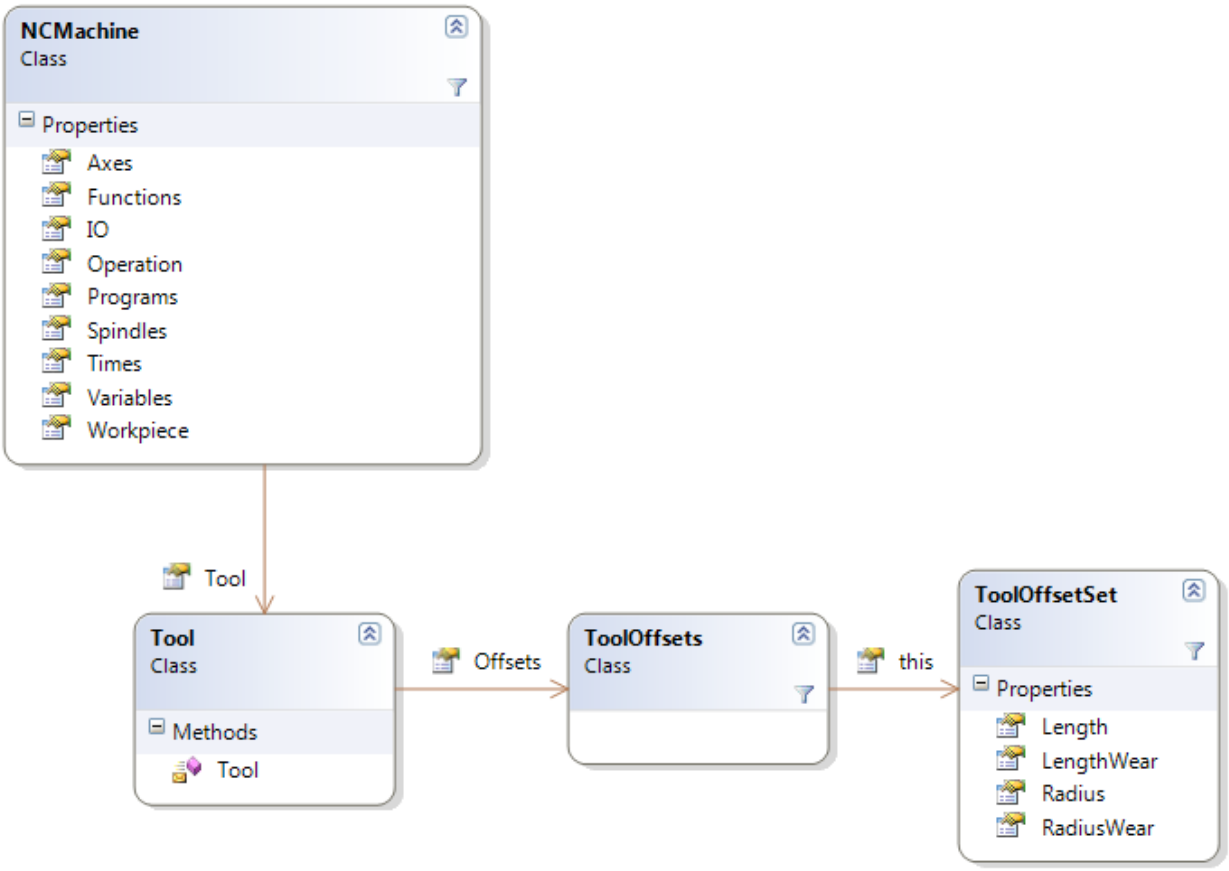


5.11 Machine Tool Offsets

Machine tool offsets are available through sub-properties of the **NCMachine.Tool** Property.

Here is an example of querying the Length offset for index 0:

```
var lengthOffset0 = machine.Tool.Offsets[0].Length;
```

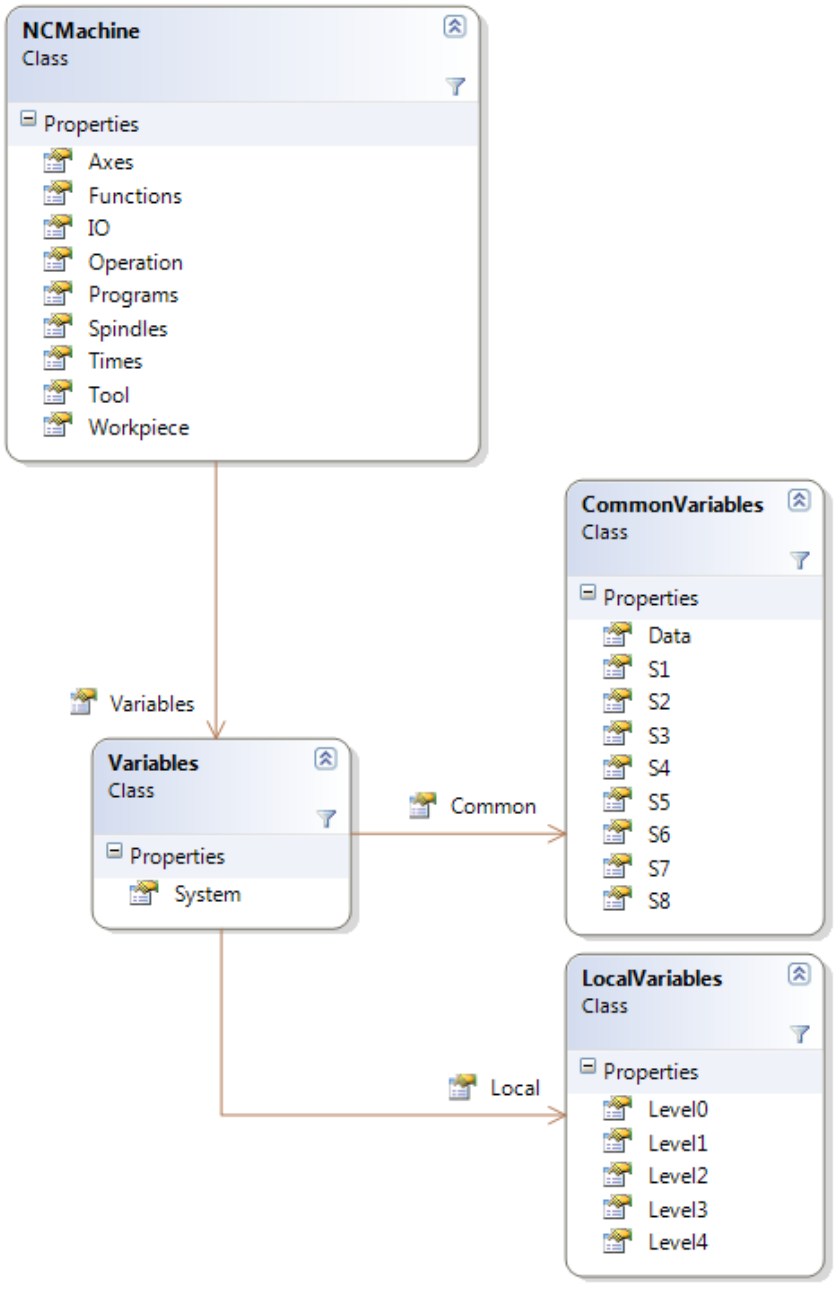


5.12 Machine Variables

Read access to Local, System and Common variables is available through sub-properties of the **NCMachine.Variables** property.

Here is an example of reading system variable 101:

```
double system101 = machine.Variables.System[101];
```

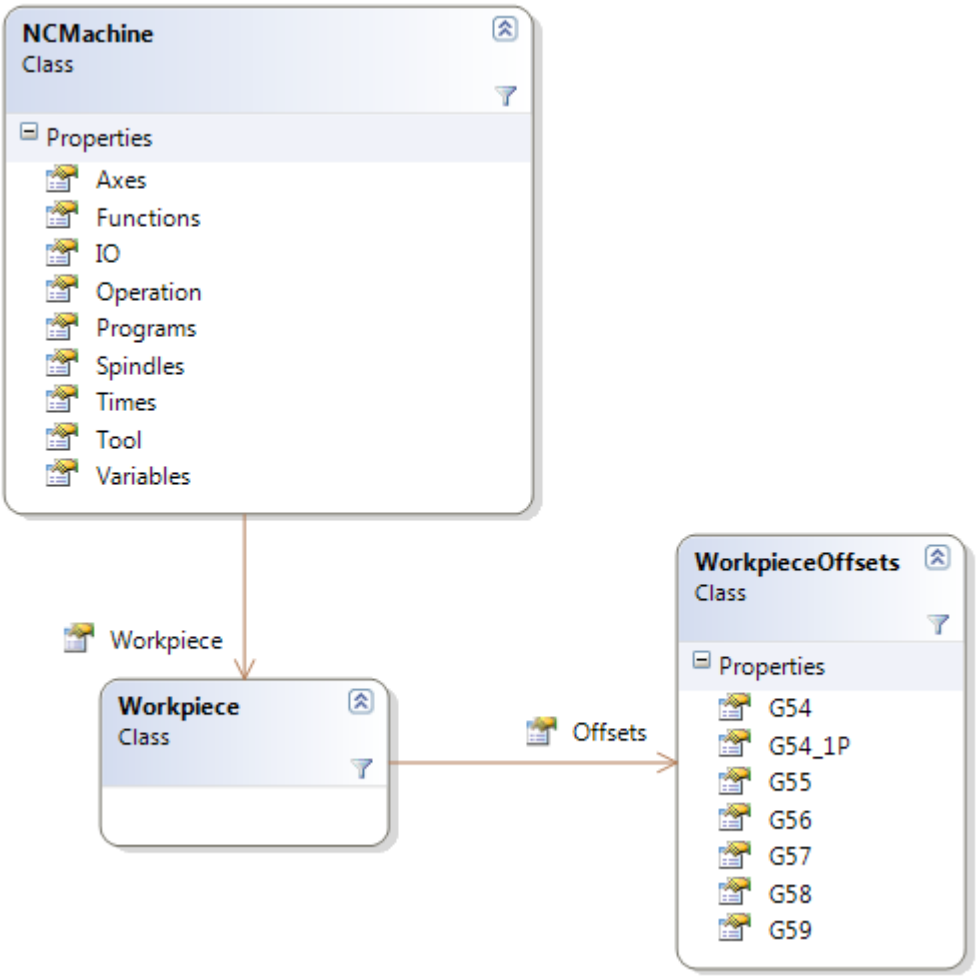


5.13 Workpiece Offsets

Workpiece offsets can be accessed through sub-properties of the **NCMachine.Workpiece** property. Offsets are accessed by offset name and axis name.

Here is an example of reading the G55 Workpiece offset on the X axis:

```
double xOffsetG55 = machine.Workpiece.Offsets.G55["X"];
```

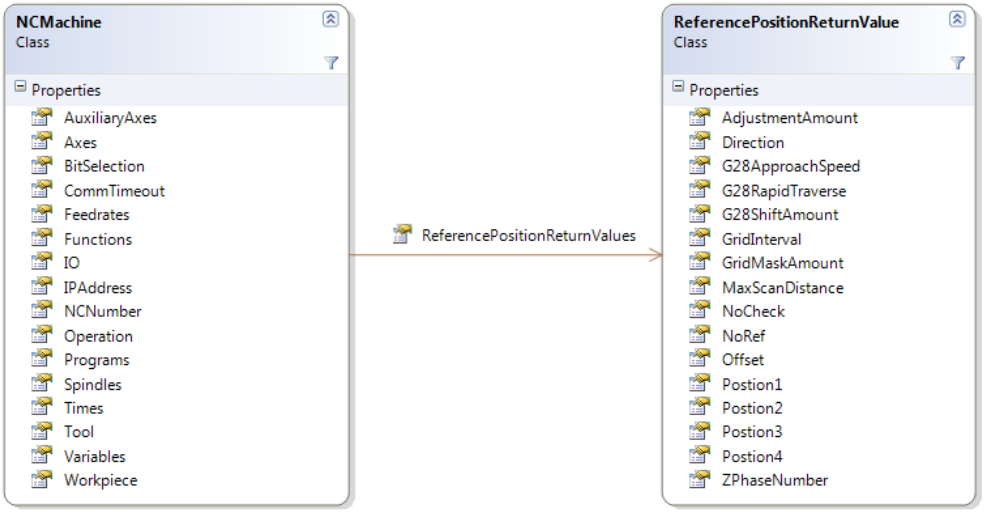


5.14 Reference Position Return Values

Common reference position return values can be accessed through sub-properties of the **NCMachine.ReferencePositionReturnValues** property.

Here is an example of reading the reference position offset:

```
var offset = machine.ReferencePositionReturnValue.Offset;
```



5.15 Simple Properties

The NCMachine class also contains some simple properties, meaning they are properties that do not contain sub-properties or methods, but instead provide direct information about the NCMachine itself.

5.15.1. CommTimeout

The CommTimeout is the current setting for the communication timeout for the NC API. This can be set or read. Following is an example illustrating the use of CommTimeout.

```
var timeout = machine.CommTimeout;
timeout += 10;
machine.CommTimeout = timeout;
```

5.15.2. IPAddress

The IPAddress is the current TCP/IP address of the current NCMachine. It can be set or read. Following is an example illustrating the use of IPAddress.

```
var address = IPAddress.Parse("192.168.10.200");
machine.IPAddress = address;
Debug.WriteLine(machine.IPAddress.ToString());
```

5.15.3. NCNumber

The NCNumber is the NC index number passed into the construction of the NCMachine class instance (typically a value of '1'). This property is read-only. Following is an example illustrating the use of NCNumber.

```
var machine = new NCMachine(1);
var machineNumber = machine.NCNumber;
```

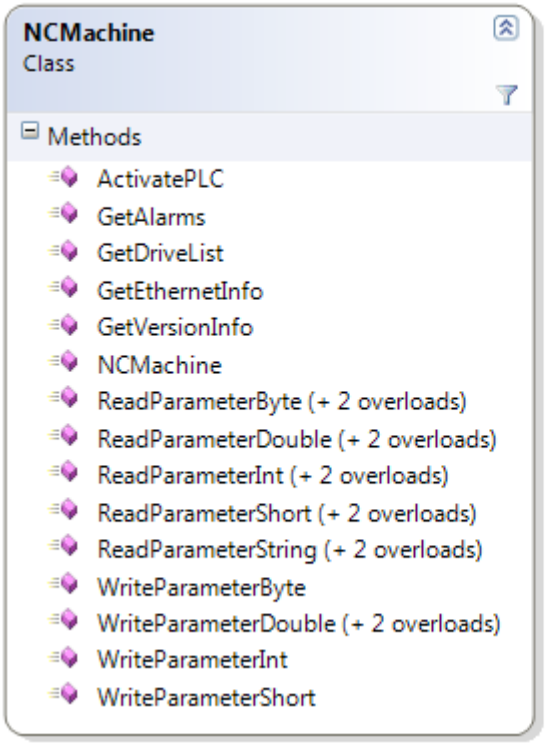
6 NCMachine Methods

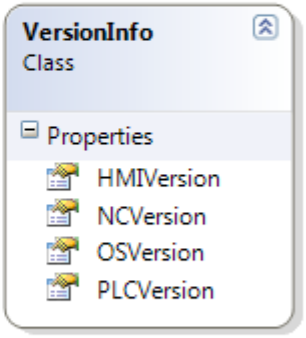
6.1 Machine Version Information

Version information about machine subsystems can be queried by calling the **GetVersionInfo** method of the NCMachine.

Here is an example of reading the PLC version:

```
var plcVersion = machine.GetVersionInfo().PLCVersion;
```



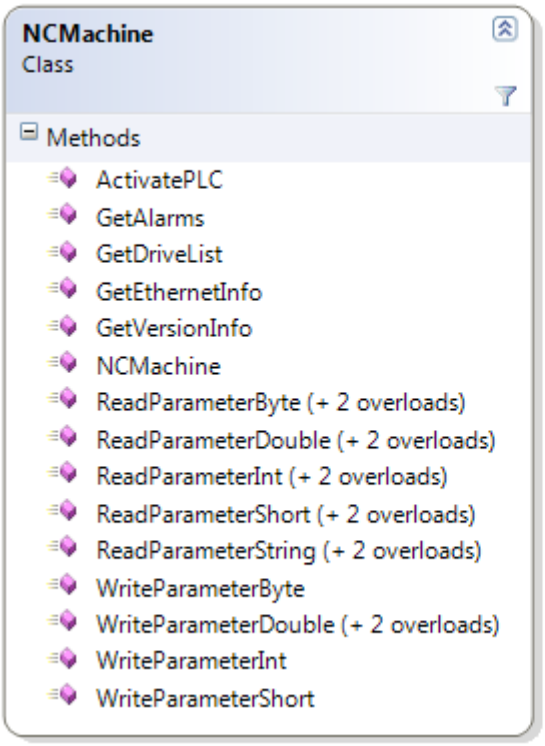


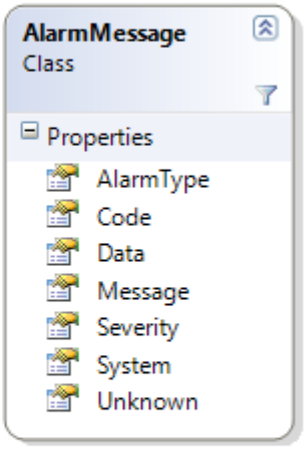
6.2 Machine Alarms

General NC Alarm information can be queried using the **NCMachine.GetAlarms** method. GetAlarms returns an array of all current alarms, in order of alarm priority.

Here is an example of retrieving the message for the first (highest priority) alarm in the system.

```
var firstAlarmMessage = machine.GetAlarms()[0].Message;
```



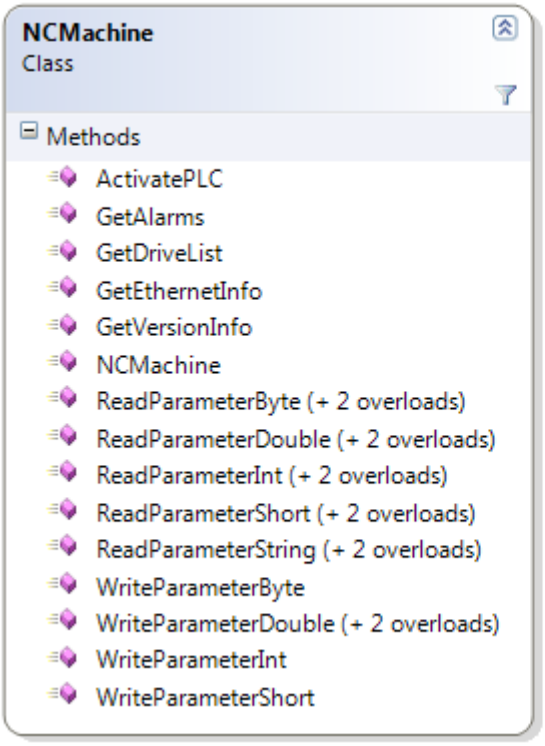


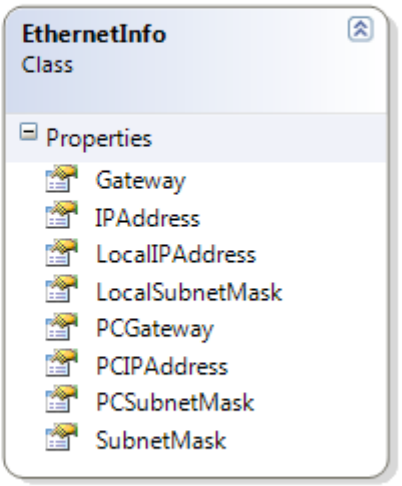
6.3 Network Parameters

Machine network parameters can be read by calling the **GetEthernetInfo** method of the NCMachine class.

Here is an example of querying the machine's IP Address:

```
var ncIP = machine.GetEthernetInfo().LocalIPAddress;
```





6.4 Other Machine Parameters

The CNC API has type-safe named coverage for a large number of commonly accessed machine parameters, but it does not directly cover all parameters supported by all machines.

However, any parameter not directly covered by a property beneath the NCMachine class can still be accessed using the MEL-SIM API using the one of the NCMachine **ReadParameterXxx** and/or **WriteParameterXxx** methods. These methods follow the following general form.

For a read operation:

```
public int ReadParameterXxx(Section section, int subsection, int axisMask, int system)
```

For a write operation:

```
public void WriteParameterInt(Section section, int subsection, int value, bool forceWrite)
```

Where:

- "Xxx" is one of:
  - Byte (8-bit)
  - Short (16-bit)
  - Int (32-bit)
  - Double (floating point) or String
- **section** is the enumeration value corresponding to the parameter Section outlined in the M700 Custom API Library Functions document.
- **subsection** is the numeric value corresponding to the parameter Section outlined in the M700 Custom API Library Functions document.
- **axisMask** is a *mask* of the axis to be affected (1 << axisNumber) as described in the M700 Custom API Library Functions document.
- **system** is a value of '1'
- **value** is the value to write to the parameter (for write operations)
- **forcedWrite** determines if the write is considered a "forced" write as described in the M700 Custom API Library Functions document.

7 Machine File Management

File management functions are managed by using the **NCFile** class. The file management functions are beyond, but still including in-memory Programs covered by the NCMachine.Programs property. The NCFile is structured to follow the same conventions as standard .NET file I/O and should be familiar to any developer familiar with standard file manipulation in .NET.

Stream-based operations are available through the NCStream returned by calls to the NCFile.Open method:

```
public void NCFileCopyByStream()
{
    var machine = new NCMachine(1);
    var buffer = new byte[10000];
    int pos = 0;
    using (var stream = NCFile.Open(machine, "\\PRG\\USER\\111", FileMode.Open, FileAccess.Read))
    {
        int read = 0;
        do
        {
            read = stream.Read(buffer, pos, 512);
            pos += read;
        } while (read > 0);
    }

    using (var stream = NCFile.Open(machine, "\\PRG\\USER\\111y", FileMode.Create, FileAccess.ReadWrite))
    {
        stream.Write(buffer, 0, pos);
        stream.Flush();
    }
}
```

Simplified methods for bulk reads and writes, just like in the .NET File class, are also available:

```
public void NCFileCopy()
{
    var machine = new NCMachine(1);
    var contentA = NCFile.ReadAllText(machine, "\\PRG\\USER\\111");

    NCFile.WriteAllText(machine, "\\PRG\\USER\\111tmp", contentA);
}
```