# AMCore User Guide

Created on 5 December, 2019

Page intentionally left blank

# Table of Contents

This guide introduces you to ANCA Motion's AMCore software. It provides instructions for getting started, configuring and using AMCore.

In this user guide, the following instructional icons are used:

**Tip**

**Info**

**Note**

**Warning**

The information contained in this guide was correct at the time of writing, but is subject to change. Please ensure you always refer to the version of the guide corresponding to the AMCore version you are using.

# 1 Related documents

| Document | Location | Description |
|---|---|---|
| `CHANGELOG.md` | AMCore application directory | History of product changes |
| `Part Programmers Reference.pdf` | AMCore application directory | Guide to the extended part programming language |
| `PLC Programmers Reference.pdf` | AMCore application directory | Guide to the PLC language |
| `CNC Connect API.pdf` | SDK Application directory | Reference to the AMCore API: CNC Connect |

# 2 Terms and abbreviations

| | |
|---|---|
| **API** | Application Programming Interface |
| **Bundle** | A collection of installation packages that are chained together into a single user experience |
| **CNC** | Computer Numerical Control |
| **EPPL** | Extended Part Programming Language(see page 12) |
| **Feedrate** | The rate of movement of the end effector through the work-piece |
| **Home folder** | The folder where OEMs keep their files. More information: Home folder(see page 20) |
| **INtime** | Real-time operating system that runs alongside Microsoft Windows on ANCA Motion CNCs |
| **IO** | Input/Output |
| **LM** | Logical Machine(see page 13) |
| **MPG** | Manual Pulse Generator(see page 0) |
| **MUP** | Machine Update Period |
| **OEM** | Original Equipment Manufacturer |
| **OS** | Operating System |
| **PLC** | Programmable Logic Controller(see page 0) |
| **PLCL** | PLC Language |
| **RAM** | Random Access Memory |

# 3 What is AMCore?

AMCore is a flexible, high performance software solution for controlling the motion of machines. It is the heart of ANCA Motion's CNC system, which has been refined over 40 years.

The controller runs on the CNC's real-time operating system (alongside Windows) to achieve precise timing, allowing quick, deterministic responses to critical machine operations. It is able to control up to 20 axes and 10 spindles, executing up to 5000 NC instructions per second in up to 3 simultaneous part programs.

It accepts standards-compliant G-code, including all primary G-code commands such as arcs, circles and helices. It also accepts EPPL (Extended Part Programming Language), which is ANCA Motion's extension to G-code.

EPPL adds many features, including:

- Mnemonic names for G-codes
- Variables, if statements, for loops
- Spline interpolation
- User input and output prompts
- File operations

AMCore includes full acceleration and jerk management with look-ahead. This means the controller will look at future commands and plan its velocities and accelerations to deliver smooth acceleration and low jerk cornering. Look-ahead synergises with AMCore's dynamic path smoothing to achieve the best possible speed for the configured tool quality.

The MPG feed, retrace and active program edit features allow correction of points immediately during a dry-run without the need to restart. Soft axes allow complex axis combinations to be programmed as a single virtual axis.

AMCore has the following extension points:

- A PLC compiler is included, so anyone can write PLC code that AMCore compiles and executes.
- The CNC Connect API allows interaction with programs, parameters and variables from a C, C++, C# or VB.NET application.
- The OPC-UA server allows interaction with variables from an OPC-UA client.

# 4 Concepts

**In this section**



This section introduces many important AMCore concepts that are used throughout the guide. It also provides links to further reading.

## 4.1 Configuration properties

Configuration properties are settings that apply to the current AMCore session. You can set them by providing a configuration file to AMCore on startup.

You can override configuration properties from an existing configuration file, in order to customise an existing system.

All configuration properties are optional. If you don't set (or inherit) some properties, they will revert to their default values.

> To learn how to use configuration properties, go to the Understand configuration properties(see page 25) section.
> For a list of all available configuration properties, refer to the Configuration property reference(see page 62).

## 4.2 Data block mapping

Data block mapping provides a configurable way to access data in the EtherCAT process data map for an EtherCAT drive. A Data Block Map defines the source, destination, size and state of the data to be transferred from the drive.

## 4.3 Extended Part Programming Language (EPPL)

AMCore's Extended Part Programming Language (EPPL) is a superset of the ISO-6983 (G-Code) programming language. It includes familiar G-Codes, M-Codes, S-Codes, etc. but also includes many optional extensions which may be used to produce programs that are more powerful and easier to read. Some key features of EPPL are:

- Mnemonic names for G-codes

- Variables, if statements, for loops
- Spline interpolation
- Transformations (rotation, scaling, mirroring)

## 4.4 Fieldbus

Fieldbus is the name of a family of industrial computer network protocols used for real-time distributed control, standardized as IEC 61158. For this user guide, these protocols are SERCOS and/or EtherCAT.

## 4.5 Joint

A joint is an actuator of an axis. In AMCore, joints have an index from 1 to 20.

## 4.6 Kinematics

Kinematics refers to the relationship between axis space (commanded by part programs) and joint space (commanded by AMCore).

See the Set up your axes(see page 37) section to learn how to configure the kinematics in AMCore.

## 4.7 Logical machine

A logical machine (LM) is a grouping of programmable dimension words (e.g. X, Y, Z) that may be operated in synchronization. It is an indivisible, non-shareable resource. The allocation of dimension words to logical machines is static, made once during system start-up.

## 4.8 Manual Pulse Generator (MPG)

A Manual Pulse Generator (MPG) is the wheel component of ANCA Motion pendants. Turning the wheel generates position pulses that can be used to precisely move an axis. It can also be used to step through and rewind a part program.

## 4.9 Move

A move refers to an AMCore command that makes the machine move. You can command the following move types:

| Type | Description |
|---|---|
| Rapid | A move to a specified location *as quickly as possible*. |
| Linear | A move to a specified location at the configured feedrate in a straight line. |
| Helical | A move to a specified location at the configured feedrate by travelling in a circular arc. |

| Type | Description |
|---|---|
| Spline | A move to a specified location at the configured feedrate in such a way that causes the overall path to be smooth. |
| Joint | A special type of move in which you directly command the joints (rather than the axes). |

## 4.10 Parameters

Parameters are persistent settings that apply to AMCore. They have a key and a value, separated by a colon.

The key is used to identify the parameter. Many keys begin with a modifier (e.g. `1.smoothing_factor`) that can be used to set the parameter in different domains (logical machines, joints, etc.). Omitting the modifier sets the parameter in all domains. For example:

```
# Set the safe_velocity for all joints to 1960
*safe_velocity: 1960

# Change the safe_velocity for joints 4 and 5
*4.safe_velocity: 17640
*5.safe_velocity: 186.2
```

A *parameter file* is a text file containing one or more parameters. There are 6 parameter files (listed in priority order): test, user, oem, mspec, common and gen.

> ⓘ You can provide one (or more) of these files (except *gen*) via the Parameter files configuration properties(see page 62).

When AMCore queries a parameter, it searches the parameter files in priority order until it finds a match. This hierarchy is commonly referred to as "the database".

The *gen* parameter file contains default values for AMCore parameters, and is not configurable. For the list of AMCore parameters and their default values, refer to the Parameter reference(see page 65).

> ⓘ For a more detailed description of parameters and how to use them, see the Understand parameters(see page 30) section.

## 4.11 Part programs

Also referred to as "NC programs", part programs are used to program machine movement. They are written in EPPL(see page 12).

You can run part programs(see page 52) via a number of different interfaces - including via EPPL G-Codes and M-Codes, or from your PLC or application code.

You can choose which folders AMCore searches for part programs (and the order in which they are searched) by defining a defining a part program search hierarchy(see page 43).

> ⓘ To learn the basics of writing part programs, go to the Create part programs(see page 55) section. To learn more advanced programming, refer to the EPPL Guide(see page 9).

## 4.12 Programmable Logic Controller (PLC)

In AMCore, a programmable logic controller (PLC) controls the input and output state variables. For example, when you press an emergency stop button, this is detected by the PLC which tells AMCore to disable the drives.

## 4.13 Shared memory

Shared memory is precisely what it sounds like: memory that is shared. In particular, many AMCore processes use shared memory for inter-process communication. Effectively, variables in shared memory are global variables that anyone can read and write.

## 4.14 Variables

Commonly referred to as "shared memory variables", variables are named locations in shared memory that can be accessed by all parts of the system. They are typically used to monitor the state of the CNC or to change some aspects of machine behaviour until the next system restart.

> ⓘ For a list of variables available in AMCore, refer to the Variable reference(see page 80).

# 5 Get started

This section provides instructions to get AMCore up and running. Due to the highly configurable nature of AMCore, this can be a challenging task. So, we'll first get the system running without communicating with devices (as a simulator).

## 5.1 Prerequisites

Before you begin, make sure you have:

- Met the system requirements(see page 17),
- Downloaded the AMCore installer, and
- Received the reference project.

The reference project contains samples of the other components required by AMCore.

To get a copy of the reference project, contact your ANCA Motion representative. It will be more widely available in the future.

## 5.2 Topics

When you're ready, work through each of the following topics:

- Install AMCore(see page 18)
  - This section will guide you through the details of installing AMCore and describes the installation options. It also provides instructions to change your installed AMCore version, which can include installing multiple versions side-by-side.
- Set up your environment(see page 19)
  - In this section, you will learn about the other components of the system that are required to get AMCore running. Remember, AMCore is only the core of a motion control system; you will need to provide the rest.
- Activate your license(see page 21)
  - Provides simple steps to guide you through activating your license.
- Start AMCore(see page 22)
  - Provides details on launching AMCore, including descriptions of the different startup options.

## 5.3 Next steps

Now you've got AMCore running, you'll need to work through some more in-depth configuration to enable communication with your devices.

Continue with the section Configure AMCore(see page 24) for further instructions.

## 5.4 System requirements

### 5.4.1 Machine

| Component | Requirement |
|---|---|
| CNC | • Front Panel CNC<br>• AMC5<br>• AMC5 G2 |
| Operating System | • PMK 9<br>• PMK 10 |
| Drive | • 5DX<br>• AMD2000<br>• AMD5000<br>• AMD5X |
| IO device (SERCOS) | • Any 3DX IO device |
| IO device (USB) | • 5DX USB Relay Panel<br>• 5DX USB Safety Unit<br>• 5DX USB Front Panel<br>• 5DX USB Satellite Panel |
| IO device (EtherCAT) | • All ANCA Motion EtherCAT devices (unless otherwise specified with the device)<br>• Many third-party EtherCAT devices |

### 5.4.2 Simulator

| Component | Requirement |
|---|---|
| Computer and processor | Equivalent to or higher than a Pentium 4 2.0 GHz processor |
| Memory (RAM) | Minimum of 1 GB system memory |
| Hard Disk | Minimum of 1 GB free space |
| Operating System | • Windows 7<br>• Windows 8 (64-bit)<br>• Windows 8.1 (64-bit)<br>• Windows 10 (64-bit) |
| Windows Installer | 4.5 or higher |

## 5.5 Install AMCore

**In this section**



This section describes how to install AMCore via the Windows desktop or via the command line (allows silent unattended installs).

### 5.5.1 Install via Windows desktop

1. Double-click the installer file to start the installation process.
2. Click **Next**.
3. After agreeing to the license terms, click **Next**.
4. (*Optional*) Enable the SDK if you want to develop applications that interface with AMCore.
5. (*Optional*) Enable the OPC UA Server if you want to interface with AMCore using the OPC UA protocol.
6. Click **Install**.

### 5.5.2 Install via command line

Installing AMCore via the command line allows you to install, repair, or uninstall silently. To install AMCore via the command line use Microsoft's Msiexec utility[1].

> ⚠ Make sure you have administrator rights if installing via the command line.

Use the `/qn` option to install without showing the installer graphical interface. Select features using `ADDLOCAL`.

For example, below is sample syntax to silently install AMCore, the SDK and the OPC UA server:

```
msiexec /i AMCore-PCC-1.8.0.msi /qn ADDLOCAL=Core,SDK,OpcUaServer
```

1 https://docs.microsoft.com/en-us/windows/win32/msi/command-line-options

### 5.5.3 Change versions

The **minor version** of AMCore is the second number in the version. The **patch number** is the third. For example, AMCore 1.8.0 has a minor version number of 8 and a patch number of 0.

#### 5.5.3.1 Change minor version

AMCore supports **side-by-side installations** of minor versions. So, you can have AMCore 1.7 installed at the same time as AMCore 1.8.

To install another minor version, simply follow the usual installation steps. The existing minor version can be removed as required.

If you wish to keep both versions installed, you can select which version to run using the `-version` option in the AMCore command line interface(see page 22).

#### 5.5.3.2 Change patch version

Where `x` is the minor version number of AMCore you wish to use.

To upgrade from a previous version of AMCore with the same minor version number, simply follow the usual installation steps.

To downgrade, you must first uninstall the previous AMCore version before installing as usual.

## 5.6 Set up your environment

**In this section**



As it's name suggests, AMCore contains the core functionality of a complete motion control solution. This means additional files and configuration are required to get the system up and running.

In particular, you should provide:

- The locations of your files
- Custom PLC
- Parameter files
- *(Optional)* A user interface
- *(Optional)* A launcher

### 5.6.1 Set up the reference project

Before you continue, make sure you've completed the setup instructions provided with the reference project.

## 5.6.2 Understand the reference project

Once you've set up the reference project, you'll be ready to go.

Before you continue with the next section, you should make sure you understand each of the components of the project and how they are integrated into AMCore. The rest of this section is dedicated to this.

### 5.6.2.1 Home folder

The home folder is the default folder used when a path is not configured. It is the combination of a registry entry and an environment variable.

The registry entry value is an absolute path to a common folder, which is commonly referred to as the "OEM path". The key for it is (for 64-bit systems):

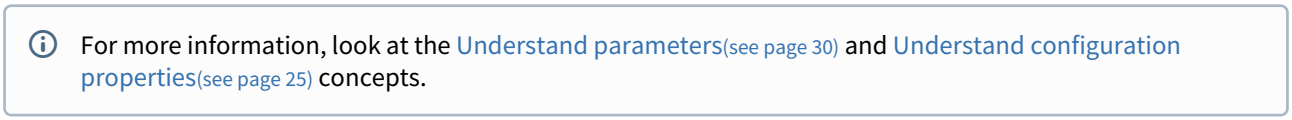
```
HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\ANCA\OEM\OemPATH
```

The environment variable value is the name of the final folder. It is defined by the environment variable "Target". Don't change it unless you know what you're doing.

### 5.6.2.2 Configuration

AMCore is configured via parameters. The reference project contains parameter files to set AMCore up as a simulator.

AMCore knows where these files are via configuration properties. In particular, the reference project includes a configuration file containing configuration properties that define the locations of these files. This configuration file is passed to AMCore by the launcher.



For more information, look at the Understand parameters(see page 30) and Understand configuration properties(see page 25) concepts.

### 5.6.2.3 Programmable logic controller

PLC is an essential component of AMCore. There is a base level of functionality included with the reference project, but you can extend this if you like. For more information, refer to the PLC Programmers Reference(see page 9).

AMCore knows the location of the PLC files via parameters.

### 5.6.2.4 User interface

The reference project contains a sample user interface to command AMCore. This is achieved through the AMCore API: CNC Connect. You can learn more about CNC Connect in the CNC Connect API(see page 9).

The user interface is run by the launcher.

#### 5.6.2.5 Launcher

Finally, the reference project contains a simple launcher that:

- Launches AMCore, passing the configuration file
- Launches the user interface, exposing some simple machine commands

## 5.7 Activate your license

Before you can run AMCore, you need to activate your license. You can do this online or offline. In both cases, you will need the activation link you received when you purchased your license.

> ⚠ The CodeMeter Runtime must be installed on your target system to activate your license. This component should be deployed by the installer, which is included in the reference project.

### 5.7.1 Online activation

The easiest way to activate a license is via the internet if your target system has internet access. Follow these steps to activate your target system:

1. Open the activation link on the system to be licensed.
2. Click **Activate License** and wait for the process to complete successfully.
3. Done. You can now launch AMCore!

### 5.7.2 Offline activation

In some cases, you may not have internet access on the target system. You will still need internet access on another system to facilitate the activation. Follow these steps to activate your target system offline:

1. Generate a context file(see page 61) on your target system, and transfer it to your system with internet access (e.g. with a USB drive).
2. Open the activation link on your system with internet access.
3. Click **Switch to offline activation**.
4. Click **Choose File**, select the context file created in step 1, and click **Upload Request**.

5. Click **Download License Update** and transfer the update file to your target system.

6. Apply the update file on your target system (see the `-u` option under am-license(see page 60)).

7. Done. You can now launch AMCore!

## 5.8 Start AMCore

**In this section**



In AMCore 1.5, a single point entry to AMCore was introduced, which provides simplified setup, startup, and stop procedures.

This entry point is an executable named **AMCore.exe** and is found in the folder defined by the environment variable **%AMCore%**.

To start AMCore, you can simply execute (from the command line):

```
"%AMCore%AMCore"
```

This will run setup (if required) and launch AMCore. If setup runs, a system restart may be required to launch AMCore. The quotation marks prevent spaces in the path (defined by %AMCore%) causing issues.

### 5.8.1 Command line interface

The following command line options are available for AMCore.exe:

| Option | Description |
|---|---|
| `-version VALUE` | Indicates which AMCore version to use. Can be shortened to `-v`. |
| `-start` | Starts AMCore. If base mode is running, complete startup. |
| `-start base` | Starts AMCore in base mode. This advanced option is a partial startup that allows manipulation of parameters and variables. |
| `-stop` | Stops AMCore. This option will override all other options. |
| `-setup` | Forces setup to run. |
| `-config FILEPATH` | Start AMCore with the configuration file located at FILEPATH. The path must not contain spaces, but can contain environment variables.<br>This option is available from AMCore 1.8 onward. |

| Option | Description |
|---|---|
| `-sim` | Run in windows only, do not use hardware. |
| `-nohw` | Simulation with INtime, without use of hardware. |
| `-forcedl` | Force drive code download (EtherCAT only). |
| `-plcc` | Force the PLC to compile. |
| `-options` | Allows additional arguments to be passed to internal processes. |

If no arguments are supplied, then `-start` is assumed. If arguments are provided, then you must also specify `-start` if you want AMCore to run after the other options are processed.

You don't need to specify the version in most cases. If you don't provide the version, the active version runs. If no version has been activated, the highest version runs.

AMCore.exe returns an exit code to indicate the success of the requested operation, as described in AMCore.exe exit codes(see page 86).

#### 5.8.1.1 Examples

To start AMCore 1.8 when you have multiple versions installed (and run setup first if required):

```
"%AMCore%AMCore" -start -v 1.8
```

To setup and then start AMCore 1.8 when you have multiple versions installed:

```
"%AMCore%AMCore" -start -v 1.8 -setup
```

### 5.8.2 Launch an application with AMCore

You can launch an application with AMCore by wrapping AMCore.exe in your own application launcher (e.g. a batch file).

When AMCore.exe is used to launch AMCore (normally or in base mode), it will not return until the relevant startup has completed. So, you can simply wait for AMCore to return before launching an application that depends on AMCore.

This only works in AMCore 1.8. For earlier versions, use the previous mechanism: the target-specific batch file (e.g. `PCC_start.bat`).

# 6 Configure AMCore

This section provides instructions to configure each of the subsystems of AMCore.

## 6.1 Prerequisites

Before completing this section, make sure you:

- Have AMCore in a working state (you can launch it).
- Understand how to configure AMCore using both configuration properties(see page 25) and parameters(see page 30).

If you don't have AMCore working yet, you should work through the get started(see page 16) section, which helps you get AMCore running as a simulator.

## 6.2 Topics

If you're setting up AMCore from a working simulator, you should work through the following topics.

- Set up your devices(see page 33)
  - Provides instructions on defining your drives and devices so that AMCore can establish communication with them. If you've already set up some devices, you'll be able to use this section to add another one.

- Set up your axes(see page 37)
  - This section helps understand and configure the kinematics. Essentially, you'll learn how to define axes and transform joint coordinates into axis coordinates.

- Configure machine motion(see page 39)
  - Teaches you how to control the commands generated by AMCore with precision. This is essential to prevent collisions, vibrations and other performance defects.

## 6.3 Additional topics

There are many other aspects of AMCore that you can configure. Choose from one of the following topics.

- Define a part program search hierarchy(see page 43)
  - Helps you to define an ordered list of folders in which AMCore will search for your part programs.

- Set up OPC UA(see page 47)
  - Helps you get AMCore's OPC-UA server up and running, allowing you to monitor your machine remotely. Includes examples demonstrating how to use a simple OPC-UA client and customize which parameters and variables are accessible.

- Customize look and feel(see page 49)
  - Lists each aspect of AMCore's appearance that can be changed, along with instructions on how to change it.

## 6.4 See also

- Use AMCore(see page 52)
- Parameter reference(see page 65)

## 6.5 Understand configuration properties

**In this section**



You can configure AMCore by providing a *configuration file* containing some *configuration properties*, which are settings that apply to the current AMCore session.

To do this, run **AMCore.exe** to start AMCore with the optional command-line argument `-config <filepath>`, where `<filepath>` is the path of your configuration file.

A configuration file is in JSON format and contains one or more configuration properties. Each configuration property is a name : value pair. Configuration properties are of 5 basic types: *boolean*, *integer*, *floating-point*, *string* or *array of strings*.

This section describes how to use configuration properties. For a list of all available configuration properties, refer to the Configuration property reference(see page 62).

### 6.5.1 Example: Use a *string* configuration property

You can specify a custom location for the *test* parameter file via the configuration property `parameters.test.path`, which has the data type *`string`*.

You can do this by starting AMCore as follows:

**console**

```
C:\TEST>"%AMCore%AMCore.exe" -start -config example1.json
```

where your configuration file `C:\TEST\example1.json` contains:

**C:\TEST\example1.json**

```
{
    "version":                  "1.0.0",

    "parameters.test.path":     "MyParametersFolder/myTestFile.db"
}
```

The property `parameters.test.path` specifies a custom filepath for the *test* parameter file: `C:\TEST\MyParametersFolder\myTestFile.db`.

### 6.5.2 Example: Use an *array of strings* configuration property

You can specify custom locations for part program files via the configuration property `programs.paths`, which has the data type *array of strings*.

You can do this by starting AMCore as follows:

**console**

```
C:\TEST>"%AMCore%AMCore.exe" -start -config example2.json
```

where your configuration file `C:\TEST\example2.json` contains:

**C:\TEST\example2.json**

```
{
    "version":                  "1.0.0",

    "programs.paths":           [
                                    "MyProgramFolder1/",
                                    "MyProgramFolder2/",
                                    "MyProgramFolder3/"
                                ]
}
```

The property `programs.paths` specifies a custom list of folder paths, which will be searched when activating part program files:

- `C:\TEST\MyProgramFolder1\`
- `C:\TEST\MyProgramFolder2\`
- `C:\TEST\MyProgramFolder3\`

### 6.5.3 Usage

The command-line path `<filepath>` must be a valid path to a configuration file. This path may take the following forms:

- May be an absolute or relative path.
- May contain environment variables.
- Must not contain spaces.
- Folders may be delimited with forward slashes `/` or backslashes `\`.

If the absolute path of your configuration file contains one or more spaces, you should change your working directory then specify `<filepath>` using a relative path, to ensure that `<filepath>` does not contain any spaces.
(Alternatively, you can specify `<filepath>` using an environment variable that contains the absolute path. However, you must ensure that the environment variable is not expanded before `<filepath>` is passed to AMCore.exe.)

Some configuration properties (such as `parameters.test.path` and `programs.paths`) specify custom paths for certain files or folders. These paths may take the following forms:

- May be absolute or relative paths.
- May contain environment variables.
- May contain spaces.
- Folders may be delimited with forward slashes `/` but not backslashes `\`.

Do not use backslashes `\` in your configuration file. Backslashes have special meaning in the JSON file format, so AMCore may be unable to correctly parse your file. You should delimit folders with forward slashes `/` to avoid such errors.

All configuration properties are optional. If you don't set (or inherit(see page 27)) some properties, they will revert to their default values. For example:

- If you don't supply a configuration file (by omitting the option `-config <filepath>`), all properties will revert to their default values.
- If you supply a configuration file that contains a subset of the available properties, any omitted properties will revert to their default values.

Configuration properties are set during AMCore startup and apply to the current AMCore session. Each time you start AMCore, you can specify different values for some or all of the properties, if desired.

Configuration properties don't retain values that were specified during previous AMCore startups. Upon a new startup, if you don't set (or inherit) a given property, that property will instead revert to its default value. (When AMCore is offline, properties will retain their values from the last AMCore session. See When AMCore is offline(see page 27).)

You should begin every configuration file with the following entry, which specifies the file's format:

**JSON**

```
    "version":                "1.0.0",
```

AMCore currently recognises the file format `1.0.0` (only). This is the file format used in all our examples. Other file formats may be introduced in the future.

### 6.5.4 Override configuration properties

Your configuration file can include another configuration file, to inherit any configuration properties from that included file. In fact, configuration files may be nested (to any desired depth) by including files within files, etc. Your properties (as specified in your configuration file) will take priority over any included properties (as specified in any nested included files).

#### 6.5.4.1 Example: Nested configuration files

You can override some configuration properties, using nested configuration files, by starting AMCore as follows:

**console**

```
C:\TEST>"%AMCore%AMCore.exe" -start -config example3A.json
```

where your configuration file `C:\TEST\example3A.json` contains:

**C:\TEST\example3A.json**

```
{
    "version":                "1.0.0",

    "include":                "example3B.json",

    "parameters.test.path":   "MyParametersFolder/myTestFile.db",

    "parameters.user.path":   "MyParametersFolder/myUserFile.db",

    "programs.paths":         [
                                  "MyProgramFolder1/",
                                  "MyProgramFolder2/"
                              ]
}
```

and its included file `C:\TEST\example3B.json` contains:

**C:\TEST\example3B.json**

```
{
    "version":                  "1.0.0",

    "parameters.test.path":     "MyParametersFolder/myOtherFile.db",

    "parameters.oem.path":      "MyParametersFolder/myOemFile.db",

    "programs.paths":           [
                                    "MyProgramFolder3/",
                                    "MyProgramFolder4/"
                                ]
}
```

In this case, the resultant values of the properties are:

- `parameters.test.path`: `C:\TEST\MyParametersFolder\myTestFile.db`
- `parameters.user.path`: `C:\TEST\MyParametersFolder\myUserFile.db`
- `parameters.oem.path`: `C:\TEST\MyParametersFolder\myOemFile.db`
- `programs.paths`:
    - `C:\TEST\MyProgramFolder1\`
    - `C:\TEST\MyProgramFolder2\`
    - `C:\TEST\MyProgramFolder3\`
    - `C:\TEST\MyProgramFolder4\`

### 6.5.4.2 Values of overridden configuration properties

The configuration properties specified in a configuration file (e.g. `example3A.json`) take priority over those specified in its included files (e.g. `example3B.json`). However, the mechanism for resolving these priorities depends on the data type of each property.

*Boolean*, *integer*, *floating-point* or *string*:

- If a property of type *boolean*, *integer*, *floating-point* or *string* is specified both in your configuration file and in an included configuration file, your value will override the included value.
- In Example 3, the final value of `parameters.test.path` is:
    - `C:\TEST\MyParametersFolder\myTestFile.db`
- This is because the value from `example3A.json` (`MyParametersFolder/myTestFile.db`) overrides the value from `example3B.json` (`MyParametersFolder/myOtherFile.db`).

*Array of strings*:

- If a property of type *array of strings* is specified both in your configuration file and in an included configuration file, your array will be prepended to the included array.
- In Example 3, the final value of `programs.paths` is:
    - `C:/TEST/MyProgramFolder1/`
    - `C:/TEST/MyProgramFolder2/`
    - `C:/TEST/MyProgramFolder3/`
    - `C:/TEST/MyProgramFolder4/`
- This is because the array from `example3A.json` (`[MyProgramFolder1/, MyProgramFolder2/]`) is prepended to the array from `example3B.json` (`[MyProgramFolder3/, MyProgramFolder4/]`).

#### 6.5.4.3 Configuration by multiple users

You can use a nested configuration file to override an AMCore configuration supplied by another user (without modifying their original files), in order to customise an existing system. Your configuration file would include their configuration file - to inherit their configuration properties, then optionally override some or all of those inherited properties.

A set of users may thus configure AMCore via a set of nested configuration files (one provided by each user). For example, a software application could specify its AMCore configuration by providing a configuration file containing some configuration properties. An OEM, who was customising that software application for their own use, could then alter or extend the AMCore configuration inherited from the application. The OEM would provide their own configuration file, which would include the application's configuration file - and would optionally override some properties and/or specify other properties.

### 6.5.5 When AMCore is offline

Some AMCore tools may be used while AMCore itself is not running, and may thus read some configuration properties while AMCore is offline. In this case, all properties will retain their values from the last AMCore session.

For example, AMCore's Diagnostic Tool is used to gather and package files into a diagnostic bundle, which may be sent back to ANCA Motion for analysis. When Diagnostic Tool runs, it reads properties of the form `parameters.*.path` , which specify user-configured locations for parameter files - and then copies these parameter files from their configured locations into its diagnostic bundle (along with various other files).

However, Diagnostic Tool is a standalone tool that may be run when AMCore is offline. In this case, the properties `parameters.*.path` will retain their values from the last AMCore session, so Diagnostic Tool will capture parameter files from their last configured locations.

> ⚠ If AMCore has been installed but has never been started, all configuration properties will be undefined. AMCore's standalone tools will thus be unable to read these properties, and may not function normally. For example, if Diagnostic Tool is run when AMCore has never been started, the properties `parameters.*.path` will be undefined, so Diagnostic Tool will be unable to capture any parameter files from user-configured locations.

## 6.6 Understand parameters

**In this section**



You can configure AMCore by providing a *parameter file* containing some *parameters*, which are persistent settings that apply to AMCore.

A parameter file is a text file containing one or more parameters. Each parameter is a key : value pair. Parameters are of 4 basic types: *boolean*, *integer*, *floating-point* or *string*.

AMCore supports a hierarchy of 6 parameter files, as shown in the following table.

| # | Parameter file | Writeable? | Supplied by | Notes |
|---|---|---|---|---|
| 5 | Test | Read-only | Users | Generally used only for testing purposes. |
| 4 | User | Writeable | Users | |
| 3 | Oem | Writeable | Users | |
| 2 | Mspec | Read-only | Users | |
| 1 | Common | Read-only | Users | |
| 0 | Gen | Read-only | AMCore | Not configurable. Contains default parameter values. |

The parameter files have a fixed priority order as listed above - from the *test* file (highest priority) to the *gen* file (lowest priority). When querying a parameter, AMCore will search these files, in the order listed above, and return the first match that is found.

The first 5 files (*test*, *user*, *oem*, *mspec*, *common*) are for user configuration of AMCore. You can configure AMCore by providing one or more of these files, containing your desired parameter values. However, each file is optional, and they do not need to be supplied.

The final file (*gen*) is part of AMCore and provides default values for many parameters. This file is not user-configurable, but is listed for completeness.

AMCore provides interfaces that allow run-time writing of parameters to the *user* or *oem* parameter files. The remaining files are treated as read-only and may not be written via these interfaces.

The parameter file hierarchy, as shown above, allows a set of users to configure AMCore via a set of individual parameter files (one provided by each user).

### 6.6.1 Configure locations for parameter files

You can customise the locations of the parameter files *test*, *user*, *oem*, *mspec* and *common*. You can specify the location (and name) of each parameter file via its corresponding configuration property(see page 25), as shown in the following table.

| # | Parameter file | Writeable? | Configuration property | Default (legacy) location |
|---|---|---|---|---|
| 5 | Test | Read-only | `parameters.test.path` | `<home folder>\misc\p_test.db` |
| 4 | User | Writeable | `parameters.user.path` | `<home folder>\misc\p_user.db` |
| 3 | Oem | Writeable | `parameters.oem.path` | `<home folder>\misc\p_oem.db` |
| 2 | Mspec | Read-only | `parameters.mspec.path` | `<home folder>\db\config\p_mspec.db` |

| # | Parameter file | Writeable? | Configuration property | Default (legacy) location |
|---|---|---|---|---|
| 1 | Common | Read-only | `parameters.common.path` | `<home folder>\db\config\p_common.db` |

If you want to add and/or modify some parameters, you should create a parameter file containing your custom parameter values, then set the corresponding configuration property (of data type: *`string`*) to the filepath of your parameter file. See Understand configuration properties(see page 25) for information on their usage.

Each parameter file has a default location, as shown in the table. If you don't customise the location of a parameter file (via its configuration property), AMCore will instead look for this file in its default location.

You should specify the locations of your parameter file(s) via their corresponding configuration propert(ies).
You should not rely on default locations for parameter files, since these exist primarily for legacy reasons.

If you specify a custom location for a parameter file (via a configuration property), a file must exist in the configured location.
AMCore will report an error (and fail to start) if a parameter file is missing from its configured custom location.

When AMCore's Diagnostic Tool is run, it will automatically capture each parameter file from its (last) configured custom location.

### 6.6.2 Automatic creation of writeable parameter files

You can use AMCore to write to the writeable parameter files (*user* and *oem*) without first creating these files, as AMCore will automatically create these files when required.

If you don't specify a custom location for a writeable parameter file (via its configuration property), and that file is absent from its default location (see table above), AMCore will automatically create a blank copy of that parameter file (in an AMCore-defined location). You can then use AMCore's standard interfaces to write parameters to (and subsequently read parameters from) that newly-created file.

This feature allows multiple software applications to write to a single parameter file, without first requiring an individual (pre-nominated) application to install a blank copy of that file. For example, a machine may run multiple software applications that each generate some machine configuration parameters, which they then save to the *user* parameter file. In this situation, it may be unclear who is responsible for creating the initial (blank) *user* file (and unclear when this should occur), so the applications could instead rely on AMCore to automatically generate this file. The individual applications could then simply write to the *user* file, when desired, without first needing to create the file.

Automatic creation applies only to the writeable parameter files. It would not be meaningful to automatically create a blank copy of a read-only parameter file, as all queries to this file would fail anyway.

## 6.7 Set up your devices

This section provides instructions to enable AMCore to communicate with and understand your EtherCAT devices.

### 6.7.1 Define your EtherCAT topology

Provide the path to your EtherCAT Network Information (ENI) file using the EtherCAT Network Information file(see page 66) parameter.

The contents of the ENI file must match the device configuration.

If you need help generating an ENI file, contact your ANCA Motion representative.

Once you've done this, you should be able to restart AMCore and establish device communication.

### 6.7.2 Define logical devices

A logical device is an ordered virtual device that can be linked to a physical device. The order of logical devices is independent from the order of physical devices.

In AMCore, you only configure logical devices. This means you can connect the physical devices in any order (i.e. to minimize cables) or modify the physical devices in your system without changing parameters that depend on devices.

It is up to you how to order your logical devices. You should define a logical device for each physical axis of your machine.

To define a logical device:

1. Set the Device EtherCAT address(see page 66) parameter to the corresponding physical address. The physical address of a device is the index (starting at 1) of the device in the EtherCAT topology.

2. Set an appropriate name by setting the Device name(see page 66) parameter. The name should describe the use of the device rather than the name of the product. For example, name a device "X-axis" rather than "AMD5x Drive". See the Set up your axes(see page 33) page to help determine the axis label you might use for the device.

3. Set it's control type using the Device control type(see page 66) parameter. A custom control type signifies that the device is an IO device. See Set up an IO device for more information.

4. *(Optional)* For a third party device, provide the product name by setting the Product name(see page 66) parameter. This will greatly improve the usefulness of many diagnostic error messages.

In other sections, the term device means logical device.

### 6.7.3 Set up a device

Depending on the type of device you're setting up, continue with one of the following topics:

### 6.7.4 Set up a drive

This section is a brief introduction to the configuration required for ANCA Motion drives.

#### 6.7.4.1 Change the firmware

You should specify the required firmware version for each drive, to ensure your configuration is appropriate for the version of firmware on the drive.

To do this, for each device:

1. Set the Firmware version(see page 71) parameter to the version you want to use on that device.
2. Provide a path to the firmware image using the Firmware root path(see page 71) and Firmware file(see page 71) parameters.
3. Set the Bootloader version(see page 71) parameter to the version you want to use on that device.
4. Provide a path to the bootloader image using the Bootloader root path(see page 71) and Bootloader file(see page 71) parameters.

#### 6.7.4.2 Set drive parameters

In almost all cases, the default drive firmware parameter values will not be appropriate for your application. You'll need to provide new values for the drive parameters that are loaded when AMCore starts.

To provide a new drive parameter value:

1. Add the parameter identification number (IDN) to the Phase 3 parameter list(see page 71) parameter.
2. Set the value of the drive parameter using the Operation data(see page 71) parameter.

For a complete list of the IDNs available for a particular firmware version, refer to the firmware documentation.

## 6.7.5 Set up an IO device

**In this section**



This sections provides steps to configure an IO device. You'll learn how to describe the structure of it's EtherCAT packet, and expose the IO to PLC.

### 6.7.5.1 Describe the EtherCAT packet

There are 3 different types of IO: Digital IO, Analog IO and String IO.

The data for each of these types are contained sequentially in an EtherCAT packet. By default, the IO is unpacked from it's packet in a default order: Digital → Analog → String. Inputs are unpacked before outputs.

This, along with the number and size of each input describe the structure of the EtherCAT packet.

So, for the device's custom control type, set the following parameters:

- Digital input count(see page 68)
- Digital output count(see page 68)
- Analog input count(see page 68)
- Analog input size(see page 68)
- Analog output count(see page 68)
- Analog output size(see page 68)
- String input count(see page 68)
- String input size(see page 68)
- String output count(see page 68)
- String output size(see page 68)

This allows AMCore to correctly read the inputs and outputs for any device that uses that control type.

You don't have to use all of the available IO of a device. The defined packet structure just has to describe the actual packet structure up until the bits that will are discarded.
For example: if a device has just 512 digital outputs available, you could set the digital output count as any number from 1 to 512.

The size of the defined packet structure should **not** exceed the available IO for the device!

#### Change the unpacking order

If your IO device uses a different packet structure, you can change the unpacking order by:

1. Defining a custom Device control type(see page 66) for each of it's IO types.

2. Use the Control type description(see page 68) parameter to change the order.

For example, let's consider an IO device with digital, analog and string IO.

For such a device, let's define a custom control type for each IO type:

```
sampledevice-digital.di : 2
sampledevice-digital.do : 1

sampledevice-analog.ai : 3
sampledevice-analog.1.ipi_size : 2
sampledevice-analog.2.ipi_size : 4
sampledevice-analog.3.ipi_size : 4
sampledevice-analog.ao : 2
sampledevice-analog.1.opi_size : 4
sampledevice-analog.2.opi_size : 2

sampledevice-string.si : 1
sampledevice-string.1.ips_size : 10
sampledevice-string.so : 2
sampledevice-string.1.ops_size : 5
sampledevice-string.2.ops_size : 7
```

Now let's change the unpacking order to String → Analog → Digital:

```
1.control_type : sampledevice
sampledevice.description : sampledevice-string+sampledevice-analog+sampledevice-
digital
```

### 6.7.5.2 Map IO to shared memory

You need to map device IO to shared memory so that you can access the state of the devices from PLC.

For each device, choose the base addresses for the IO in shared memory. Set them using the following parameters:

- Digital input base address(see page 68)
- Digital output base address(see page 68)
- Analog input base address(see page 68)
- Analog output base address(see page 68)
- String input base address(see page 68)
- String output base address(see page 68)

It's a good idea to follow a convention when selecting shared memory addresses. For example, if you always map analog inputs to addresses in the range 300-399, it will be easier to tell at a glance that the shared memory address 304 relates to analog inputs.

## 6.8 Set up your axes

**In this section**



This section provides explanations and instructions to assist you in configuring the kinematics in AMCore to match that of your machine.

### 6.8.1 Understand the kinematics of AMCore

Before you continue, it is important that you understand the kinematic arrangement of AMCore. By default, AMCore includes a fixed kinematic mapping, which is described here.

To find out more about using alternate kinematic mappings, contact your ANCA Motion representative.

Joints are the actuators of the machine; think of the motor attached to a drive. They are numbered from 1 to 20. Each joint has a defined axis label and is listed below.

| **Joint** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Axis label** | X | Y | Z | U | V | W | A | B | C | X' | Y' | Z' | SLV | P | Q | R | A' | U' | V' | B' |

You can use any combination of these axis labels, with the only restrictions being:

- The X, Y and Z axes must be linear, perpendicular and follow the right-hand rule.
- The SLV axis is only to be used as the slave axis in a gantry.

### 6.8.2 Assign devices to axes

First, you need to decide which axis label you will use for each device. You should have set the Device name(see page 66) parameter to reflect this when you set up your devices(see page 33).

To map the device to the axis:

1. Find the joint number of the axis from the table above.
2. Determine the logical address of the device (it was set when you set up your devices(see page 33)).
3. Set the Joint logical device(see page 65) for the joint to the logical address of the device.

### 6.8.3 Set up logical machines

You can assign axes to different logical machines. All axes of a logical machine are synchronized. So, with multiple logical machines, you can control multiple machines asynchronously.

To map an axis to a logical machine:

1. Find the joint number of the axis from the table above.
2. Determine the logical machine number (1 to 3) of the machine that you want to control the axis.
3. Set the Joint logical machine(see page 65) parameter for the joint to the logical machine number.

### 6.8.4 (Optional) Enable plane selection

If any of your logical machines have 3 perpendicular linear axes, you can enable the plane selection functions of EPPL that use of the mirroring, scaling and rotation EPPL functions on these axes.

To do this:

1. Determine which of the axes will correspond to the X, Y and Z axes of the plane selection functions. Make sure they follow the right-hand rule to avoid confusion.
2. Set the Logical machine axes(see page 65) parameter for the logical machine. The ordinates `x1`, `x2` and `x3` shou ld be set to the axes you determined, respectively.

Now, you can select a plane on the logical machine and rotate, scale or mirror the commands. The axes X, Y and Z in the functions will correspond to the axes you configured, and will work analogously.

As a simple example, most machines will have the following parameters set:

```
1.x1.lm_map : X
1.x2.lm_map : Y
1.x3.lm_map : Z
```

This sets the ordinates for logical machine 1 to the X, Y and Z axes. So, selecting the XY plane in this logical machine will select the plane defined by the X and Y axes.

If the machine had another logical machine with 3 perpendicular, linear axes U, V and W:

```
2.x1.lm_map : U
2.x2.lm_map : V
2.x3.lm_map : W
```

Then, selecting the XY plane in logical machine 2 will select the plane defined by the U and V axes.

## 6.9 Configure machine motion

**In this section**



This section helps you configure the parameters that control the motion of the machine.

### 6.9.1 Constrain the joints

Before you can command a movement safely, you need to define the range of motion for each joint.

So, you need to set the following parameters for each joint:

- Joint lower soft limit(see page 74)
- Joint upper soft limit(see page 74)
- Joint soft limits deceleration limit(see page 74)
- Joint velocity limit(see page 74)
- Joint acceleration limit(see page 74)
- Joint deceleration limit(see page 74)
- Joint jerk limit(see page 74)

For the best performance, set the jerk limit to the largest jerk that the bandwidth of the joint allows, resulting in shorter execution times for rapid moves.

Once you've done this, you should be able to command each axis to change position. The commands AMCore generates for each joint will not exceed these limits.

This doesn't mean it is safe to move! Erroneous commands could result in a collision. Always use the feedrate override when uncertain, and keep your hand on the emergency stop!

It's important to note that configuring the above joint limits also configures the calculated feedrate of rapid moves. Rapid moves are moves that attempt to move the end effector as fast as possible (regardless of the *configured* feedrate), and are typically used when re-positioning the machine. In a rapid move, at least one joint involved will reach the velocity limit given sufficient time, and all the joints will be commanded at the acceleration, deceleration and jerk limits.

So, now that the joint limits are appropriately set, you should be able to command rapid moves.

### 6.9.2 Set nominal radii

A nominal radius for a rotary axis defines the effective radius of the axis. It is used to convert units from rotational to linear.

You have to set the nominal radii correctly so that the generated motion for combinations of linear and rotary joints will not exceed their motion limits.

To set the nominal radius for a rotary axis, set the Nominal radius(see page 65) parameter for the axis.

### 6.9.3 Constrain the end effector

Now, you should configure the limits that constrain the motion during normal operation.

To do this, set the following parameters:

- Velocity limit(see page 75)
- Tangential acceleration limit(see page 75)
- Tangential deceleration limit(see page 74)
- Radial acceleration limit(see page 75)
- Tangential jerk limit(see page 75)
- Radial jerk limit(see page 75)
- Transitional jerk limit(see page 75)

Many of these parameters have corresponding `ORIDE` variables that can help you quickly change the value without restarting AMCore. This allows you to quickly iterate to an appropriate value.

These limits will apply during all motion, excluding rapid moves.

Once you've set them appropriately, you should be able to run part programs and begin producing parts.

### 6.9.4 Set alternative velocity limits

Depending on your application, you can use the primary and secondary rapid-limits to programmatically switch to alternate velocity limits at run-time.

The primary rapid-limit takes effect when the variable `ILB_RAPID_LIMIT` is active, and the secondary rapid-limit takes effect when the variable `ILB_RAPID_LIMIT_2` is active.

To set your alternative velocity limits, set the following parameters:

- Joint rapid-limit velocity limit(see page 74)
- Rapid-limit velocity limit(see page 75)
- Secondary rapid-limit velocity limit(see page 75)

If a motion monitoring system is fitted to the machine, you should set the joint rapid-limit velocity limit such that there is some margin allowed for noise, etc.

### 6.9.5 Set error thresholds

You can define the amount of position lag which, when exceeded, causes an error. This means the machine will be disabled and an error message will appear.

The position lag is the distance between the commanded position and the measured position.

To do this, set the Joint position lag error threshold(see page 72). A suitable value is approximately 120% of the following lag (`G_SERVO_PE`) observed during a maximum velocity move.

You can also set a similar threshold in the velocity loop of the drive. This way, a potential error will be detected much quicker because:

- Drives inherently have a faster response time, and
- You don't have to wait for position error to accumulate.

To set this, use the drive parameter (32978) Velocity Following Error Threshold.

Refer to the firmware documentation(see page 9) for more information on this drive parameter.
See Set up a drive(see page 34) to learn how to set drive parameters in a parameter file.

### 6.9.6 Refine machine motion

The following parameters can be used to adjust the way that the tool path is processed and to fine tune the machine.

- Smoothing factor(see page 72)
- Tangency angle(see page 72)
- Corner rounding limit(see page 72)

#### 6.9.6.1 Smoothing factor

The smoothing factor determines the strength of the joint smoothing filter, which is a low pass filter that smooths the commanded position. Setting this factor to 0 disables the filter; higher values cause smoother position commands.

The filter is applied to each joint separately, so the tool path will be modified. It also adds some delay between the commanded position and the actual position.

For example, a smoothing factor of 10 will result in a deviation of a few microns (for a radial acceleration of 1000 $mm/s_2$) and a delay equal to 10 times the machine update period.

You can monitor the filtered position command using the variable `G_SERVO_FP<j>`. Compare this to `G_SERVO_CP<j>` to see precisely how the filter affects the position command.

#### 6.9.6.2 Tangency angle

Defines the angle between successive moves that, when exceeded, causes the machine to pause between the moves. This means the machine decelerates to zero velocity before executing the next move.

If the angle is lower than this parameter value, the machine won't stop, but may slow down depending on the angle, curvature difference, transition jerk limit and the corner tolerance.

This parameter does not affect rapid moves. The machine always stops before *and* after a rapid move.

Increasing the tangency angle prevents the machine from coming to a complete stop at corners (decreasing cycle time), but may reduce the sharpness of the corner.

#### 6.9.6.3 Corner rounding limit

When transitioning between moves (except splines), the corners may be rounded by an amount less than or equal to this parameter. The rounding only occurs when the tangency angle is not exceeded.

For rotary axes, the nominal radius is used (`nomrad`) to convert from degrees to millimetres.

Increasing the corner rounding limit will reduce the sharpness of the corner and can decrease the cycle time, since traversing a rounded corner can be performed at a higher feedrate without exceeding the motion limits.

### 6.9.7 Refine MPG motion

The MPG hand-wheel is an incredibly useful tool for manually controlling the machine.

It's behaviour is configured by these parameters:

- MPG velocity limit(see page 78)
- MPG gain(see page 78)
- MPG bias(see page 78)
- MPG position lag limit(see page 78)
- MPG axis position lag limit(see page 78)
- MPG input window(see page 78)
- MPG live offset input window(see page 78)

The default values will work for most situations. You may, however, need to adjust the position lag limit parameter, which defines how far the actual position of the axis can lag behind the generated position command.

The position lag limit only comes into effect when the velocity limit is active. This is because when the velocity is limited, the actual position begins to lag.

If the position lag limit is too high (and you command a move faster than the velocity limit), you will find that after you stop moving the MPG wheel, the axis keeps moving.

If the position lag limit is too low, you could find that commanding small movements with a precise number of pulses won't move as far as you expect. This is because to limit the lag, pulses that cause the lag limit to be exceeded are discarded. This indirectly limits the MPG velocity. You can determine this effective velocity limit (in mm/min) by calculating: $60 \times \text{jerk}^{1/3} \times \text{lag}^{2/3}$

You can lower the position lag limit for a single axis using the MPG axis position lag limit parameter. This is useful for small axes, for which the shared lag limit is large relative to the axis dimensions.

## 6.10 Define a part program search hierarchy

**In this section**



You can configure AMCore by defining a *part program search hierarchy*, which is an ordered list of folders in which AMCore will search for your part programs. This search hierarchy tells AMCore where to find your programs.

The main steps to use a part program search hierarchy are as follows:

1. Define a search hierarchy(see page 43). This is an ordered list of folders.
2. Store your part programs in these folders (including within their subfolders).
3. Run programs using the search hierarchy(see page 43). When you use a supported interface to run a part program, AMCore will search the configured hierarchy to locate the relevant program.

A part program search hierarchy provides the following benefits:

- You can run part programs via relative filepaths(see page 43) (relative to your specified hierarchy folders).
- You can override part programs(see page 43) supplied by other users (without modifying their original files) in order to customise an existing system.

You don't have to define a part program search hierarchy. However, in that case, you must run all part programs via absolute filepaths(see page 43), so AMCore knows where to find them.

### 6.10.1 Define a search hierarchy

You can define your part program search hierarchy via the configuration property(see page 25) `programs.paths` (data type: *`array of strings`*).

You should set this configuration property to a list of folder paths, in the order that you want them to be searched. See Understand configuration properties(see page 25) for information on their usage.

You can then store part programs in the specified folders (including within their subfolders) and use AMCore to run them.

You can extend another user's search hierarchy by creating a *configuration file* that includes their existing configuration file. See Override configuration properties(see page 28) for an example. Your `programs.paths` (which lists your part program folders) will be prepended to their existing `programs.paths` - creating a combined search hierarchy in which your folders take priority over their folders.

> ⓘ The part program search hierarchy may include any number of folders. The list may include 0 or more folders and does not have a maximum size.

> ⓘ The search order matches the order in which folders are listed in the configuration file(s). You don't specify a numbered position for each folder.

> ⓘ AMCore does not require the specified folders to exist. If a folder does not exist, it will simply be skipped when searching the hierarchy (just like a folder that exists but does not contain the relevant part program file).

> ⚠ AMCore's legacy part program folders (e.g. `PP`, `PPSYS` and `MISC`) and the corresponding parameters (`dirs.pp`, `dirs.cc`, `dirs.espp` and `dirs.espp_special`) are considered to be deprecated. AMCore does not require these parameters to be defined. You should define your part program folders via the part program search hierarchy, rather than relying on these legacy folders.

### 6.10.2 Run programs using the search hierarchy

You can use the following interfaces to run part programs using the configured search hierarchy:

- EPPL subprogram call (`CALLP`)
- PLCL sub-part-program devices (e.g. `SPPGG`)
- PP_RUN
- G-Codes
- M-Codes

See Run part programs(see page 44) for general information on using these interfaces.

The following subsections describe AMCore's behaviour, including its searching of the configured search hierarchy, when running a part program via each of these interfaces.

In the following examples, we assume that the part program search hierarchy has been configured (via your configuration file and its nested included files (if any)) to a list of `N` *hierarchy folders* (in priority order):

- `[hierarchy path 1]` (highest priority, searched first)
- ...
- `[hierarchy path N]` (lowest priority, searched last)

You can use the part program search hierarchy to run part programs that are stored in the hierarchy folders themselves, or in subfolders of the hierarchy folders

#### 6.10.2.1 EPPL CALLP, PLC Devices and PP_RUN

If you run a part program via a **relative filepath**, AMCore will search the configured search hierarchy for the corresponding part program:

- You can use these interfaces to run a part program via a relative filepath: `"[subfolder path]/<filename>"`
- Here, `[subfolder path]` is optional and may have 0 or more levels. The filepath may optionally be specified using environment variables, and may optionally have a leading slash.
- In this case, AMCore will search the following locations for the relevant part program, and run the first program that it finds (if any):
  - `[hierarchy path 1]\[subfolder path]\<filename>`
  - ...
  - `[hierarchy path N]\[subfolder path]\<filename>`

PP_RUN is a command-line utility, but does not use its current working directory to resolve relative filepaths. Instead, it searches the configured search hierarchy for the corresponding part program, as described above.

Alternatively, if you run a part program via an **absolute filepath**, AMCore will run the exact part program that you have specified:

- You can use these interfaces to run a part program via an absolute filepath: `"<absolute path>"`
- The filepath may optionally be specified using environment variables.
- In this case, AMCore will run the specified part program (if it exists) - and will not search the configured search hierarchy.

You can use an absolute filepath to run a part program that resides either inside or outside the configured search hierarchy. This will prevent your specified program from being overridden by (other) programs in the search hierarchy.

#### 6.10.2.2 G-Codes

If your part program includes a (canned cycle) G-Code(see page 45), AMCore will search the configured search hierarchy for the corresponding part program:

- Your part program can include a (canned cycle) G-Code: `g# i0`
- Here, `#` is the G-Code number. The (canned cycle) G-Code (`g#`) must be programmed with at least one parameter (`i0`) or dimension word (`x0`) - or it will simply be ignored.
- In this case, AMCore will search the following locations for a *canned cycle* (a part program) named `g#.pp`, and run the first program that it finds (if any):
  - `[hierarchy path 1]\g#.pp`

    - ...
    - `[hierarchy path N]\g#.pp`
- You can thus customise a G-Code (`g#`) by creating a canned cycle (part program) named `g#.pp` in a (root) hierarchy folder from your configured search hierarchy.
- This applies only to G-Codes that trigger user-supplied canned cycles. Other G-Codes are *preparatory words* that are reserved by AMCore and do not trigger part programs.

#### 6.10.2.3 M-Codes

If your part program includes a (mode 4) M-Code(see page 46), AMCore will search the configured search hierarchy for the corresponding part program:

- Your part program can include a (mode 4) M-Code: `m#`
- Here, `#` is the M-Code number.
- In this case, AMCore will search the following locations for a part program named `m#.pp`, and run the first program that it finds (if any):
    - `[hierarchy path 1]\m#.pp`
    - ...
    - `[hierarchy path N]\m#.pp`
- You can thus customise an M-Code (`m#`) by creating a part program named `m#.pp` in a (root) hierarchy folder from your configured search hierarchy.
- This applies only to M-Codes that are configured to use mode 4 (trailing subprogram call), and are thus implemented via part programs. Other M-codes (modes 1-3 and 5-7) are instead implemented via the PLC and do not trigger part programs.

⚠ CNC Connect does not support the part program search hierarchy. You can only use the CNC Connect API to run a part program via an **absolute filepath** (which must not contain environment variables). In this case, AMCore will run the specified part program (if it exists) - and will not search the configured search hierarchy.
ANCA Motion's *Commander* software runs part programs via CNC Connect, and thus does not support the part program search hierarchy.

### 6.10.3 Override part programs

You can use the part program search hierarchy to override part programs supplied by other users (without modifying their original files) in order to customise an existing system. You can do this as follows:

1. *Inherit the existing search hierarchy*. Create a configuration file that includes the existing configuration file, to inherit the existing `programs.paths`.

2. *Extend the search hierarchy*. Add the configuration property `programs.paths`, which lists your hierarchy folder(s), to your configuration file.

3. *Override a part program*. Create a part program in your hierarchy folder, with the same filename (and subfolder structure) as the original part program, to implement your custom behaviour.

You can use the part program search hierarchy to override custom G-Codes and M-Codes supplied by other users.
In step 3, you should create a part program in your (root) hierarchy folder, with the same filename (`g#.pp` or `m#.pp`) as the original part program, to implement your custom behaviour.

You can arrange your part programs into "namespaces" by storing them in subfolders of your hierarchy folders. When inheriting and overriding part programs supplied by other users, we recommend using subfolders to avoid unintentional overrides (while still allowing deliberate overrides):

- You should create a uniquely-named subfolder (of your hierarchy folder) to contain your part programs. You should choose a subfolder name that hasn't been used by existing users (whom you are inheriting from).
- If you are creating a new part program (and not overriding an existing part program): You should save this part program in your uniquely-named subfolder. This minimises the risk of accidentally overriding part programs inherited from existing users (including any programs they may provide in the future, via updates to their software applications).
- If you are overriding an existing part program: You must match the subfolder and filename of the existing part program, as described above.

## 6.11 Set up OPC UA

**In this section**



This section helps you get AMCore's OPC UA server up and running.

### 6.11.1 Enable OPC UA

By default, the OPC UA server is not enabled. It also requires a valid certificate to start.

Set the configuration property `opcua.enable` to `true` to enable the OPC UA server.

You can either manually provide a certificate, or automatically generate one. Once you have enabled the server and provided a valid certificate, the OPC UA server starts and stops with AMCore.

#### 6.11.1.1 Manually provide a certificate

To provide a certificate:

1. Get a certificate. You can generate one via standard certificate generation software, or request a certificate from your IT administrator.

   The certificate must match the server configuration.

2. Add the certificate to the Windows Certificate Stores. For instructions on how to do this, refer to the Microsoft documentation.

3. Copy the public key certificate to `C:\ProgramData\ANCA Motion\UA backup\CertificateStores\UA Applications\certs`. Please ensure that the public key certificate is in DER format and the file name has an .der extension.

4. Provide the certificate by setting the `opcua.certificate.subject` configuration property to the subject of the certificate.

#### 6.11.1.2 Automatically generate a certificate

Alternatively, you can automatically generate a certificate:

1. Enable certificate generation by setting the configuration property `opcua.certificate.autogenerate` to `true`.

2. Set `opcua.certificate.validMonths` to the desired expiry of the certificate (in number of months).

Now, the next time the OPC UA server starts, it will generate a self-signed certificate and add it to the Windows Certificate Stores.

> The certificate will be generated to work with the configuration of the OPC UA server at the time of generation. If you further configure the server, you may need to regenerate the certificate.

## Connect to OPC UA

You'll need to connect to the OPC UA server via your OPC UA client. Refer to the documentation of your specific client to find out how to add a server.

When you're adding the server, you'll need to provide these details:

| | |
|---|---|
| **Name** | `ANCA Motion AMCore OPC UA Server` |
| **Endpoint Url** | `opc.tcp://<localhost>:51210/amcore` |

Where `<localhost>` is replaced by the name of the computer that is running AMCore.

#### 6.11.1.3 Accept your client certificate

The first time you connect your OPC UA client to AMCore's OPC UA server, the client certificate will be rejected and the connection will therefore fail.

Rejected certificates are placed in `C:\ProgramData\ANCA Motion\UA backup\CertificateStores\Rejected Certificates\certs`

Trusted certificates should be placed in `C:\ProgramData\ANCA Motion\UA backup\CertificateStores\UA Applications\certs`

To accept your client certificate, simply move it from the rejected directory into the trusted directory.

### 6.11.2 Add your own nodes

Once you're connected, you'll notice there are a few nodes that you can access. For example:

| Namespace | Node | Description | Node Id | Example Value |
|---|---|---|---|---|
| http://opcfoundation.org/UA/ | Server | Collection of nodes that are part of the OPC UA standard. | `2253` | - |
| urn:ancamotion:amcore | AMCore File Version | A string representation of the file version of AMCore that is running. | `{e2054d06-264c-4d05-90e8-656fee5ecdda}` | 1.8.211.0 |
| urn:ancamotion:amcore | AMCore Product Version | A string representation of the product version of AMCore that is running. | `{0e2cb1fb-18fd-4254-86e3-ac140f7ce4a3}` | 1.8.0 |
| urn:ancamotion:amcore | PMK Version | A string representation of the installed PMK version. | `{246dae8b-33dd-43b8-a26e-cf80bb112573}` | SW646-0-00-6000 |

You can add your own namespaces and nodes to access other variables using CNC Connect.

You'll need to use an application that interfaces with CNC Connect. Refer to the **Node Functions** section of the CNC Connect API Reference.

## 6.12 Customize look and feel

**In this section**



Some aspects of AMCore's appearance can be customised. This section lists each aspect and provides instructions to modify it.

### 6.12.1 Splash screen

The splash screen is the image that is displayed when AMCore is starting. You can change it by providing your own image.

To change the splash screen image:

1. Create a bitmap image file of the splash screen you would like. It must be 460 pixels wide, and 345 pixels high
2. Name the file `splash.bmp` and place it in a folder named `img` in the home folder.

### 6.12.2 Service icon

The service icon is the image that is displayed in the service prompt (before the splash screen) when AMCore is starting. You can change it by providing your own image.

To change the service icon:

1. Create a bitmap image file of the service icon you would like. It must be 55 pixels wide, and 16 pixels high
2. Name the file `splash_service.bmp` and place it in a folder named `img` in the home folder.

### 6.12.3 Axis image

The axis image is the image that appears in the status tool to help the end user understand the axis arrangement of the machine. You can change it by providing your own image.

> The status tool is not visible by default.

To change the axis image:

1. Create a bitmap image file of the axis image you would like. It must be 500 pixels wide, and 370 pixels high
2. Name the file `status_axes.bmp` and place it in a folder named `img` in the home folder.

### 6.12.4 Menu logo

The menu logo is the image that appears at the top of the ANCA Menu. You can change it by providing your own image.

The ANCA Menu (and the menu logo) is not visible by default.

To change the menu logo:

1. Create a bitmap image file of the menu logo you would like. It must be 55 pixels wide, and 16 pixels high
2. Name the file `menu_logo.bmp` and place it in a folder named `img` in the home folder.

# 7 Use AMCore

This section covers how to use AMCore on a machine or simulator.

## 7.1 In this section

- Run part programs(see page 52)
  - Lists all the ways you can run a part program.
- Create part programs(see page 52)
  - Provides an introduction to creating part programs. Includes simple instructions to help you write your first NC program and get your machine moving for the first time.

## 7.2 See also

- Tools(see page 52)
- Parameter reference(see page 52)
- Variable reference(see page 80)
- EPPL Guide(see page 9)
- PLCL Guide(see page 9)

## 7.3 Run part programs

**In this section**



You can run part programs (NC programs) via a number of different interfaces.

When a part program is run, it will be activated in the relevant part program processor, and execution may also commence (depending on the interface and the arguments you supply).

The following subsections provide a brief overview of some (but not all) of the interfaces through which you can run part programs. You must use the correct procedure and syntax for the relevant interface.

### 7.3.1 Using Active Program Display (APD)

You can use AMCore's Active Program Display (APD) application to run a part program:

1. Click **Program** → **Activate**.

2. Select your part program file, then click **Open**.

3. Use your main interface to run the part program, by clicking **Cycle Start**.

You can then use APD to deactivate the part program, by clicking **Program** → **Deactivate**.

### 7.3.2 From another part program (CALLP)

From a part program, you can use an EPPL subprogram call (`CALLP`) to run a part program:

**EPPL**

```
callp "<filepath>"
```

### 7.3.3 From PLC code (sub-part-program devices)

From PLC code, you can use a PLCL sub-part-program device (e.g. `SPPGG`) to run a part program:

**PLCL**

```
<subpp mnemonic> <subpp device number>
  ppp (<ppp number>)
  trigger (<condition>)
  sppname ("<filepath>");
```

where `<subpp mnemonic>` is one of `sppgg` (Go-Go), `sppgs` (Go-Stop), `sppsg` (Stop-Go) or `sppss` (Stop-Stop).

### 7.3.4 From the command prompt (PP_RUN)

From the command prompt, you can use the `pp_run` executable to run a part program:

**console**

```
pp_run <filepath>
```

### 7.3.5 From your application code (CNC Connect)

From your application code, you can call a CNC Connect function to run a part program. You may use one of the following functions:

**C**

```
CnccProgramRun("<filepath>", <ppp number>, <program type>, <timeout>)
```

**C**

```
CnccProgramRunAsync("<filepath>", <ppp number>, <program type>, <start>)
```

### 7.3.6 Using a G-Code

From a part program, you can use a (canned cycle) G-Code to trigger its corresponding part program:

**EPPL**

```
g# i0
```

where # is the G-Code number. This will run a *canned cycle* (a part program) named `g#.pp`, and is a convenient way to run an often-used part program.

Note the following:

- Some G-Codes are *preparatory words* that are reserved by AMCore and implement standard functionality.
- Other G-Codes trigger user-supplied canned cycles that may implement custom behaviour.
- A (canned cycle) G-Code (`g#`) must be programmed with at least one parameter (`i0`) or dimension word (`x0`) - or it will simply be ignored.

### 7.3.7 Using an M-Code

From a part program, you can use a (mode 4) M-Code to trigger its corresponding part program:

**EPPL**

```
{ A mode-4 M-Code }
m#
```

where # is the M-Code number. This will run a part program named `m#.pp`, and is a convenient way to run an often-used part program.

Note the following:

- Each M-Code may be configured (via relevant parameters) to use a specified mode (1-7). The mode specifies how the M-Code is implemented.

- Mode 4 (trailing subprogram call): These M-Codes are implemented via part programs. Upon processing such an M-Code, AMCore will trigger the corresponding part program.
- Modes 1-3 and 5-7 (PLC): These M-Code are implemented via PLC code. Upon processing such an M-Code, AMCore will instead trigger the PLC (by setting relevant bits).

## 7.4 Create part programs

In this section, you'll learn the basics of creating part programs.

### 7.4.1 Examples

- Write your first program(see page 55)
  - This example walks you through the process for getting an introductory (Hello World) program running.

- Program simple movements(see page 56)
  - Once you know about the environment for running a program, it's time to get moving. Work through this example to learn some simple concepts and to get your machine moving for the first time!

### 7.4.2 Next steps

You can continue to learn more advanced concepts by reading through the EPPL Guide(see page 9).

### 7.4.3 Write your first program

In this example, we'll use the Active Program Display (APD) to activate and run a simple "Hello World" program in two ways.

#### 7.4.3.1 Quickly run "Hello World!"

First, let's quickly get a program running using Manual Data Input (MDI).

1. In APD, click **MDI** in the menu bar.
2. Enter the following code:

```
write("Hello World!")
dwell x5
```

3. Click **Activate**.
4. Use your main interface to run the program by clicking **Cycle Start**.

The "Hello World" program will now run. It should display a "Read/Write" window that reads "Hello World!". After 5 seconds, the program should finish, which closes the window.

Now, the next time you open MDI, you can press **Recall** to quickly bring up and modify the previous code you wrote.

Keep in mind that MDI is only intended as a quick way to test some code - the code you write won't be saved! Let's look at a better way to write programs.

#### 7.4.3.2 Save and run "Hello World!"

Part programs are simply text files written in EPPL. So, to create and run a program:

1. Open your favourite text editor.
2. Enter the following text:

   ```
   write("Hello World!")
   dwell x5
   ```

   Make sure you end the program with an empty line.
3. Save the file using a .pp file extension.
4. In APD, click **Program** → **Activate**.
5. Select the file you saved in Step 1, then click **Open**.
6. Use your main interface to run the program by clicking **Cycle Start**.

This time, after the program runs, the program remains activated. Clicking **Cycle Start** again will run it again. You can deactivate it by clicking **Program** → **Deactivate**.

This method also let's you keep the program for later use. Much better!

### 7.4.4 Program simple movements

In this section, you'll learn some basic EPPL concepts that allow you to create a program that performs some simple machine movements.

#### 7.4.4.1 Prerequisites

Before you continue with this tutorial, make sure:

- You can enable the machine
- You've set up the motion constraints(see page 39)

Make sure your end effector has sufficient clearance (at least 50 mm) to run these programs without collision.

### 7.4.4.2 Command a move

First, let's get an axis to perform a small movement. We'll go with a 10 mm straight line in the positive X-direction at a feedrate of 1000 mm/min:

```
metric
relative
linear x10 f1000
```

There's a little bit to unpack here.

- the `metric` statement causes AMCore to operate in metric units of measurement.
- The `relative` statement causes any subsequent move to be commanded relative to the position of the end effector at the time of the move.
    - It's good practice to always specify that you're using either `absolute` or `relative` positions at the beginning of each program you write.
- `linear` sets the move mode to linear, causing the move to be a straight line.
- `f1000` means "set the feedrate to 1000 mm/min".
- `x10` means "move the X-axis to 10 mm". Remember that it will be relative to the current position.

The default unit of measurement is `metric` and thus we can and will omit this statement from other examples.

f you're familiar with G-codes, `linear` is the same as a `G1` command, and `relative` is the same as a `G91`.

So, overall, the program reads something like "use relative positions; move in a line to an X position of 10 mm at a feedrate of 1000 mm/min".

You can run the program now. You should see the program perform the move and then finish.

### 7.4.4.3 Move back

Now, let's move back to where we started. Comment out the move and add a new line that does the opposite move:

```
relative
{ linear x10 f1000 }
linear x-10 f1000
```

Note the following:

- Lines surrounded by curly brackets in EPPL are comments and won't be executed.
- The X-axis command is now -10 mm, since the movements are relative to the position of the axis at the time the command is executed.

Run the program and you should see the X-axis move back to it's starting position and then finish.

#### 7.4.4.4 Create a loop

Let's make it repeat these two moves indefinitely. Uncomment the first move and add a simple loop:

```
relative
n1
linear x10 f1000
linear x-10 f1000
goto n1
```

`n1` is a label that you can jump to with the `goto` statement.

Before you run this program, note that it will run indefinitely. You'll need to abort the program (the main interface should have a button for this).

Aborting the program is likely to result in the machine not ending at the starting point, so you may need to readjust the position of the machine if you're working in a tight space.

#### 7.4.4.5 Control the loop

Now, let's make it repeat the movement each time we press the "r" key.

```
relative
n1
readkey(&key)
if key="r" then
        linear x10 f1000
        linear x-10 f1000
        sync
        goto n1
ifend
```

Note the following:

- The indentation is not required, but is recommended to improve readability.
- The `readkey` statement waits for you to press a key and assigns the key you press to a variable named `key`.
- The `if` statement checks the key you pressed - if you pressed "r", it will perform the moves and return to read another key.
- The `sync` statement ensures the moves are executed before continuing. Learn about "look-ahead" to understand this better.

Run this program. Each time you press the "r" key in the "Read/Write" window, the machine will repeat the small movement. If you press any other key, the program will exit.

#### 7.4.4.6 Next steps

This concludes this introduction to creating programs in AMCore. To continue learning, refer to the EPPL Guide(see page 9).

# 8 Tools

This section provides documentation for the tools that come with AMCore.

## 8.1 In this section

## 8.2 am-license

**In this section**



am-license is a tool that allows you to manage your AMCore license offline. To manage your license online, please visit https://license.anca.com[2]

You can only interact with am-license via the command line. After you install AMCore, you'll find it in the following location:

```
%amcore_home%\3DX\Bin\Licensing
```

### 8.2.1 Command line options

| Option | Shorthand | Description |
|---|---|---|
| `--help` | `-h, -?` | Provides help for am-license, showing information about each of these commands. |
| `--information` | `-i` | Prints information about your licenses. |
| `--create-container` | `-c` | Creates an ANCA Group software container. |
| `--create-context=FILEPATH` | `-x` | Writes information about the container to a file (we call this a context file) located at FILEPATH.<br>A software container will be created if one does not exist. |

2 https://license.anca.com/index.php

| Option | Shorthand | Description |
|---|---|---|
| `--serial-number=SERIAL` | `-s` | Specifies the SERIAL number of the container when creating a context file.<br><br>⚠ A SERIAL number does not need to be specified if there is none or only one container on the target system. |
| `--update-license=FILEPATH` | `-u` | Updates the license using the provided update file, located at FILEPATH. |

### 8.2.2 Example: Generate a context file

If you're activating a license offline, you'll need to generate a context file. To do this:

1. Open command prompt.
2. Run this command:

```
"%amcore_home%\3DX\Bin\Licensing\am-license" --create-context="My Context"
```

> ⓘ If there is more than one container on your target system, you will need to additionally provide its serial number by adding the `-s` option to the above command. You can find out your serial numbers using the `-i` option.

After you've done this, a context file will be generated named `My Context` in the same directory as am-license (since a relative path was used).

3. Retrieve the context file from the `%amcore_home%\3DX\Bin\Licensing` directory.

# 9 Reference

This section provides a reference to quickly look up a configuration item or variable.

## 9.1 In this section

## 9.2 Configuration property reference

**In this section**



You can configure AMCore by providing a *configuration file* containing some *configuration properties*, which are settings that apply to the current AMCore session. See Understand configuration properties(see page 25) for details.

A configuration file is in JSON format and contains one or more configuration properties. Each configuration property is a name : value pair. Configuration properties are of 5 basic types: *boolean*, *integer*, *floating-point*, *string* or *array of strings*.

The following tables list all AMCore configuration properties. You can provide a configuration file containing some or all of these properties.

All configuration properties are optional.  If you don't set (or inherit) some properties, they will revert to their default values.

File and folder locations may be specified using absolute or relative paths. Environment variables may be used. Folders should be delimited with forward slashes `/`.

A configuration file may include special entries(see page 62) alongside its configuration properties. `version` should be used to specify the file format. `include` may be used to inherit configuration properties from a nested configuration file.

### 9.2.1 Parameters

These configuration properties specify custom locations for *parameter files*, which contain parameters(see page 30) to configure AMCore. The parameter files have a fixed priority order as listed here.

| Name | Type | Default (legacy) location | Units | Description |
|---|---|---|---|---|
| `parameters.test.path` | `String` | `<home folder>\misc\p_test.db` | - | Path of the *test* parameter file. |
| `parameters.user.path` | `String` | `<home folder>\misc\p_user.db` | - | Path of the *user* parameter file. |

| Name | Type | Default (legacy) location | Units | Description |
|---|---|---|---|---|
| `parameters.oem.path` | `String` | `<home folder>\misc\p_oem.db` | – | Path of the *oem* parameter file. |
| `parameters.mspec.path` | `String` | `<home folder>\db\config\p_mspec.db` | – | Path of the *mspec* parameter file. |
| `parameters.common.path` | `String` | `<home folder>\db\config\p_common.db` | – | Path of the *common* parameter file. |

### 9.2.2 Programs

| Name | Type | Default | Units | Description |
|---|---|---|---|---|
| `programs.paths` | `Array of strings` | `[]` | – | Part program search hierarchy.<br>This is an ordered list of folder paths, in which AMCore will search for part programs. If a folder doesn't exist, it is skipped. |

### 9.2.3 OPC UA

| Name | Type | Default | Units | Description |
|---|---|---|---|---|
| `opcua.enable` | `Boolean` | `false` | – | Whether the OPC UA server is enabled.<br>Set to `true` to enable the server. |
| `opcua.applicationUri` | `String` | `urn:localhost:ancamotion:amcore` | – | Application instance URI of the OPC UA server.<br>This value uniquely identifies the server and should match the URL value in the certificate subject alternative name. Note that the OPC UA server will automatically convert `localhost` to the host name. |

| Name | Type | Default | Units | Description |
|---|---|---|---|---|
| opcua.certificate.autogenerate | Boolean | false | – | Whether to automatically generate a certificate.<br>Set to false if you are providing your own certificate.<br>Set to true to have AMCore automatically generate a self-signed certificate. The lifetime of this certificate is specified via opcua.certificate.validMonths. |
| opcua.certificate.subject | String | CN=ANCA Motion AMCore OPC UA Server | – | Subject field of certificate.<br>This value must match the subject field of the certificate. |
| opcua.certificate.validMonths | Integer | 0 | Months | Lifetime of automatically generated certificate.<br>The number of months for which the generated certificate is valid. |

### 9.2.4 Tools

| Name | Type | Default (legacy) location | Units | Description |
|---|---|---|---|---|
| tools.backup.configuration.path | String | <home folder>\misc\user_backup_list.txt | – | Path of a configuration file for the User Backup tool.<br>Configuration file contains a list of paths to backup. |

### 9.2.5 Special Entries

A configuration file may include these *special entries* alongside its configuration properties.

| Name | Type | Default | Units | Description |
|---|---|---|---|---|
| version | String | 1.0.0 | – | Semantic version string specifying the configuration file format.<br>Should be included in every configuration file. The current file format is 1.0.0. |

| Name | Type | Default | Units | Description |
|---|---|---|---|---|
| include | String | - | - | Path of an included configuration file - to inherit any configuration properties from that included file.<br>Your properties (in your configuration file) will take priority over any included properties (from any nested included files). |

## 9.3 Parameter reference

**In this section**



Parameters are persistent settings that configure the behaviour of AMCore. This section lists the key parameters used by AMCore.

See the Parameters(see page 14) concept for more information on parameters.

### 9.3.1 Kinematics

| Name | Key | Class | Type | Default | Units | Description |
|---|---|---|---|---|---|---|
| Joint logical device | <j>.device | Joint.Device | Integer | - | - | Allocates a logical device to joint <j>. |
| Joint logical machine | <j>.lm_num | Joint | Integer | - | - | Allocates a logical machine to joint <j>. |

| Name | Key | Class | Type | Default | Units | Description |
|---|---|---|---|---|---|---|
| Logical machine axes | <lm>.x<i>.lm_map | Lm_num.Ord.Lm_map | String | - | - | Allocates an axis to the <i>th ordinate of logical machine <lm>. The value must match an axis label. |
| Nominal radius | <a>.nomrad | Dim.Nomrad | Float | - | mm | Defines the nominal radius of axis <a>. |

## 9.3.2 EtherCAT

### 9.3.2.1 General

| Name | Key | Class | Type | Default | Description |
|---|---|---|---|---|---|
| Device EtherCAT address | <d>.ethercat_address | Device.Address | Integer | - | Allocates an EtherCAT address to logical device <d>. An address of 0 indicates that there is no device present. |
| Device name | <d>.ds_device_name | Device.Name | String | - | Name of logical device <d>. The name should indicate the use of the device, i.e. "X-axis". |

| Name | Key | Class | Type | Default | Description |
|---|---|---|---|---|---|
| Device control type | `<d>.control_type` | `Device.Control` | String | NOT_USED | Defines the control type of logical device `<d>`. If the value is one of the specific drive values, the device is interpreted as a drive (of that type). Other values define custom control types, which are interpreted as IO devices.<br>Drive values: position, position_rotational, position_and_velocity, position_spindle, velocity |
| EtherCAT Network Information file | `ethercat.File` | `ethercat.File` | String | - | Specifies the path to the master XML (ENI) file defining the fieldbus devices. |
| Product name | `vendor_<vid>.product_<pid>.name` | `Vendor_id.Product_code.Name` | String | Unknown device | Defines the product name of an EtherCAT slave device. Default values are appropriate for ANCA Motion devices.<br>The vendor ID `<vid>` and product ID `<pid>` must be defined as integers (not hex codes). |

#### 9.3.2.2 IO devices

| Name | Key | Class | Type | Default | Description |
|---|---|---|---|---|---|
| Control type description | `<c>.description` | `Device.Control` | Integer | - | Optionally used to define a list of control type blocks separated by a "+" character, allowing device IO to be unpacked in a non-default order. |
| Control type digital input count | `<c>.di` | `Type.Attr` | Integer | 0 | Number of digital inputs to unpack into shared memory IPB space for control type `<c>`. |
| Control type digital output count | `<c>.do` | `Type.Attr` | Integer | 0 | Number of digital outputs to unpack into shared memory OPB space for control type `<c>`. |
| Control type analog input count | `<c>.ai` | `Type.Attr` | Integer | 0 | Number of analog inputs to unpack into shared memory IPI space for control type `<c>`. |
| Control type analog input size | `<c>.<i>.ipi_size` | `Type.Attr` | Integer | 2 | Number of bytes (1-4) for analog input index `<i>` (starting at 1) in control type `<c>`. |
| Control type analog output count | `<c>.ao` | `Type.Attr` | Integer | 0 | Number of analog outputs to unpack into shared memory OPI space for control type `<c>`. |

| Name | Key | Class | Type | Default | Description |
|---|---|---|---|---|---|
| Control type analog output size | <c>.<i>.opi_size | Type.Attr | Integer | 2 | Number of bytes (1-4) for analog output index <i> ( starting at 1) in control type <c>. |
| Control type string input count | <c>.si | Type.Attr | Integer | 0 | Number of string inputs to unpack into shared memory IPS space for control type <c>. |
| Control type string input size | <c>.<i>.ips_size | Type.Attr | Integer | - | Number of bytes for string input index <i> (starting at 1) in control type <c>. |
| Control type string output count | <c>.so | Type.Attr | Integer | 0 | Number of string outputs to unpack into shared memory OPS space for control type <c>. |
| Control type string output size | <c>.<i>.ops_size | Type.Attr | Integer | - | Number of bytes for string output index <i> (starting at 1) in control type <c>. |
| Device digital input base address | <d>.ipb_start | Device.Attr | Integer | 0 | Starting index of shared memory IPB space for digital inputs for device <d>. The first input is available at the starting index plus 1. |

| Name | Key | Class | Type | Default | Description |
|---|---|---|---|---|---|
| Device digital output base address | `<d>.opb_start` | `Device.Attr` | Integer | 0 | Starting index of shared memory OPB space for digital outputs for device <d>. The first output is available at the starting index plus 1. |
| Device analog input base address | `<d>.ipi_start` | `Device.Attr` | Integer | 0 | Starting index of shared memory IPI space for analog inputs for device <d>. The first input is available at the starting index plus 1. |
| Device analog output base address | `<d>.opi_start` | `Device.Attr` | Integer | 0 | Starting index of shared memory OPI space for analog outputs for device <d>. The first output is available at the starting index plus 1. |
| Device string input base address | `<d>.ips_start` | `Device.Attr` | Integer | 0 | Starting index of shared memory IPS space for string inputs for device <d>. The first input is available at the starting index plus 1. |
| Device string output base address | `<d>.ops_start` | `Device.Attr` | Integer | 0 | Starting index of shared memory OPS space for string outputs for device <d>. The first output is available at the starting index plus 1. |

#### 9.3.2.3 Drives

| Name | Key | Class | Type | Default | Description |
|---|---|---|---|---|---|
| Firmware file | <d>.Firmware.FileName | Firmware.FileName | String | - | Specifies the path to the firmware image file for device <d>, relative to the firmware root path. |
| Firmware root path | <d>.Firmware.RootPath | Firmware.RootPath | String | - | Specifies the absolute path to the folder that contains firmware images. |
| Firmware version | <d>.dsd_firmware_version | dsd_firmware_version | String | - | The firmware version number required for device <d>. |
| Bootloader file | <d>.Bootloader.FileName | Bootloader.FileName | String | - | Specifies the path to the bootloader image file for device <d>, relative to the bootloader root path. |
| Bootloader root path | <d>.Bootloader.RootPath | Bootloader.RootPath | String | - | Specifies the absolute path to the folder that contains bootloader images. |
| Bootloader version | <d>.dsd_bootloader_version | dsd_bootloader_version | String | - | The bootloader version number required for device <d>. |
| Phase 2 parameter list | <d>.ds_phase2_parameter_list | Type.Device.Ds | Integer Array | [] | Array of IDNs that are applied to device <d> during phase 2 of drive initialisation. |

| Name | Key | Class | Type | Default | Description |
|---|---|---|---|---|---|
| Phase 3 parameter list | `<d>.ds_phase3_parameter_list` | `Type.Device.Ds` | Integer Array | [] | Array of IDNs that are applied to device `<d>` during phase 3 of drive initialisation. |
| Operation data | `<d>.<idn>.ds_opdata` | `Device.Value` | Integer | - | The value to set for IDN `<idn>` in device `<d>`. The value will have custom SERCOS scaling applied before being written to the drive. |

### 9.3.3 Motion control

#### 9.3.3.1 General

| Name | Key | Class | Type | Default | Units | Description |
|---|---|---|---|---|---|---|
| Joint position lag error threshold | `<j>.xs_servo_err_tol` | `Joint.Tol` | Float | - | mm or deg | Defines the amount of position lag for joint `<j>` which, when exceeded, causes an error. |
| Corner rounding limit | `<lm>.corner_tol` | `LM.Fillet` | Float | 0 | mm | When transitioning between moves (except splines) on logical machine `<lm>`, the corners may be rounded by an amount less than or equal to this value. |

| Name | Key | Class | Type | Default | Units | Description |
|---|---|---|---|---|---|---|
| Smoothing factor | `<lm>.smoothing_factor` | `LM.Filter` | Integer | 0 | - | Determines the strength of the joint smoothing filter (a low pass filter that smooths the commanded position) for logical machine `<lm>`.<br>Variable: `G_SMOOTHING_FACTOR` (see page 72) |
| Tangency angle | `<lm>.tangency_angle` | `LM.Angle` | Float | 15 | deg | Defines the angle between successive moves (on logical machine `<lm>`) that, when exceeded, cause the machine to pause between the moves.<br>Variable: `G_TANGENCY_ANGLE` (see page 72) |
| Dry run velocity | `dry_run_velocity` | `Cnc` | Float | - | mm/min | Feedrate of the machine when performing a dry run. |

| Name | Key | Class | Type | Default | Units | Description |
|---|---|---|---|---|---|---|
| Pitch compensation folder | pcomp_folder | Pcomp | String | - | - | Absolute path of a folder containing pitch compensation data files. Can contain environment variables.<br>Default location (if parameter undefined): <home folder>\misc |

#### 9.3.3.2 Joint limits

| Name | Key | Class | Type | Default | Units | Description |
|---|---|---|---|---|---|---|
| Joint soft limits active | <j>.soft_limit_used | Joint.Soft_limit_used | Boolean | on | - | Defines whether or not the soft limits are active for joint <j>. |
| Joint soft limits deceleration | <j>.soft_limit_max_decel | Joint.Decel | Float | 1000 | mm/s$^2$ or deg/s$^2$ | Deceleration of joint <j> when approaching a soft limit. |
| Joint lower soft limit | <j>.soft_limit_minus | Joint.Limit | Float | - | mm or deg | Minimum position of joint <j>.<br>Variable: G_SERVO_SLM<j-1> (see page 74) |

| Name | Key | Class | Type | Default | Units | Description |
|---|---|---|---|---|---|---|
| Joint upper soft limit | <j>.soft_limit_plus | Joint.Limit | Float | - | mm or deg | Maximum position of joint <j>.<br>Variable: G_SERVO_SLP<j-1> (see page 80) |
| Joint velocity limit | <j>.max_velocity | Joint.Velocity | Float | 2000 | mm/min or deg/min | Maximum velocity of joint <j>. |
| Joint rapid-limit velocity limit | <j>.safe_velocity | Joint.SafeVel | Float | 1960, except Joint 4: 17640 Joint 5: 186.2 | mm/min or deg/min | Maximum velocity of joint <j> when the rapid-limit is active. |
| Joint acceleration limit | <j>.max_accel | Joint.Accel | Float | 600 | $mm/s^2$ or deg $/s^2$ | Maximum acceleration of joint <j>. |
| Joint deceleration limit | <j>.max_decel | Joint.Decel | Float | - | $mm/s^2$ or deg $/s^2$ | Maximum deceleration of joint <j>. |
| Joint jerk limit | <j>.max_jerk | Joint.Jerk | Float | 100000 | $mm/s^3$ or deg $/s^3$ | Maximum jerk of joint <j>. |

#### 9.3.3.3 Motion limits

| Name | Key | Class | Type | Default | Units | Description |
|---|---|---|---|---|---|---|
| Velocity limit | contour_rapid_velocity | Cnc | Float | 10000 | mm/min | Maximum feedrate of the end effector. Doesn't affect rapid moves. |

| Name | Key | Class | Type | Default | Units | Description |
|---|---|---|---|---|---|---|
| Rapid-limit velocity limit | `contour_rapid_limit_velocity` | `Cnc` | Float | 2000 | mm/min | Maximum feedrate of the end effector when the rapid-limit is active. Doesn't affect rapid moves. |
| Rapid-limit rapid velocity limit | `rapid_limit_velocity` | `Cnc` | Float | 2000 | mm/min | Maximum feedrate of the end effector when the rapid-limit is active.<br>Setting this higher than 2000 will not take effect and the default will still apply. |
| Secondary rapid-limit velocity limit | `rapid_limit_2_velocity` | `Cnc` | Float | 2000 | mm/min | Maximum feedrate of the end effector when the secondary rapid-limit is active.<br>Unlike the primary rapid-limit velocity limit, this can be higher than 2000 mm/min. |
| Tangential acceleration limit | `<lm>.acceleration` | `Lm.Accel` | Float | 600 | $mm/s^2$ | Maximum acceleration of the end effector of logical machine `<lm>` in the direction it is moving.<br>Variable: `G_ACCEL` (see page 80) |

| Name | Key | Class | Type | Default | Units | Description |
|---|---|---|---|---|---|---|
| Tangential deceleration limit | <lm>.deceleration | Lm.Decel | Float | 600 | mm/s$^2$ | Maximum deceleration of the end effector of logical machine <lm> in the direction it is moving.<br>Variable: G_DECEL(see page 80) |
| Radial acceleration limit | <lm>.radial_acceleration_limit | Lm.Accel | Float | 750 | mm/s$^2$ | Maximum acceleration of the end effector of logical machine <lm> in the direction perpendicular to it's movement.<br>Variables: G_RADIAL_ACCEL_LIMIT(see page 80), G_RADIAL_ACCEL_ORIDE(see page 80) |
| Tangential jerk limit | <lm>.jerk | Lm.Jerk | Float | 100000 | mm/s$^3$ | Maximum jerk of the end effector of logical machine <lm> in the direction it is moving.<br>Variables: G_JERK(see page 80), G_JERK_ORIDE(see page 80) |

| Name | Key | Class | Type | Default | Units | Description |
|---|---|---|---|---|---|---|
| Radial jerk limit | <lm>.radial_jerk_limit | Lm.Jerk | Float | 100000 | $mm/s^3$ | Maximum jerk of the end effector of logical machine <lm> in the direction perpendicular to it's movement.<br>Variables: G_RADIAL_JERK_LIMIT(see page 80), G_RADIAL_JERK_ORIDE(see page 80) |
| Transitional jerk limit | <lm>.transition_jerk_limit | Lm.Jerk | Float | 200000 | $mm/s^3$ | Maximum jerk of the end effector of logical machine <lm> during the transition from one move type to another.<br>Variables: G_TRANSITION_JERK_LIMIT(see page 80), G_TRANSITION_JERK_ORIDE(see page 80) |

#### 9.3.3.4 MPG

| Name | Key | Class | Type | Default | Units | Description |
|---|---|---|---|---|---|---|
| MPG velocity limit | <m>.mpg_velocity_limit | MPG_Velocity_Limit | Float | - | mm or deg | Maximum feedrate that MPG <m> can generate. If no value is specified, the velocity limit of the end effector is used. |
| MPG gain | <m>.mpg_gain | Mpg.Gain | Float | 0.13 | $MUP^{-1}$ | The feedrate of moves of MPG <m> is determined by multiplying the position lag by this parameter and adding the bias. |
| MPG bias | <m>.mpg_bias | Mpg.Bias | Float | 2 | mm/min | The feedrate of moves of MPG <m> is determined by adding this parameter to the gain multiplied by the position lag. |

| Name | Key | Class | Type | Default | Units | Description |
|---|---|---|---|---|---|---|
| MPG position lag limit | `<m>.mpg_max_lag` | `Mpg.Lag` | Float | 3 | mm or deg | Maximum position lag for MPG `<m>`. Pulses that cause this limit to be exceeded are discarded.<br>For moves that involve rotary joints, the nominal radius is used. If the move is a single rotary joint, the units are degrees. |
| MPG axis position lag limit | `<a>.mpg_max_lag_for_axis` | `Mpg.Lag` | Float | 0 | mm or deg | This parameter is the same as the MPG position lag limit, except it is applied to the axis `<a>` instead of an MPG.<br>For moves that involve rotary joints, the nominal radius is used. If the move is a single rotary joint, the units are degrees. |
| MPG input window | `<m>.mpg_input_window` | `Mpg.Window` | Float | 50 | ms | The input for MPG `<m>` is averaged over time before commanding the axes. This parameter defines the time period over which the input is averaged.<br>Smaller values feel more responsive, higher values feel smoother. |
| MPG live offset input window | `<m>.lo_mpg_input_window` | `Mpg.Window` | Float | 3 | MUP | This parameter is almost the same as the input window, except it only applies to live offset commands and has slightly different units. |

### 9.3.4 Tools

| Name | Key | Class | Type | Default | Units | Description |
|---|---|---|---|---|---|---|
| Params configuration file | `params_file` | `Params` | String | - | - | Absolute path of a configuration file for the Params tool.<br>Path may contain environment variables. File contains a list of parameters to display.<br>Default location (if parameter undefined): `<home folder>\db\config\param.db` |

## 9.4 Variable reference

**In this section**



### 9.4.1 Motion control

A `sync` is required before a change to any of these variables takes effect.

| Name | Scope | Access | Type | Units | Description |
|---|---|---|---|---|---|
| `G_SMOOTHING_FACTOR` | `LM` | Read-Write | Integer | - | The strength of the joint smoothing filter (a low pass filter that smooths the commanded position).<br>Parameter: Smoothing factor(see page 72) |

| Name | Scope | Access | Type | Units | Description |
|---|---|---|---|---|---|
| G_TANGENCY_ANGLE | LM | Read-Write | Float | deg | Defines the angle between successive moves that, when exceeded, cause the machine to pause between the moves.<br>Parameter: Tangency angle(see page 72) |
| G_ACCEL | LM | Read-Write | Float | $mm/s^2$ | Maximum acceleration of the end effector in the direction it is moving.<br>Parameter: Tangential acceleration limit(see page 75) |
| G_DECEL | LM | Read-Write | Float | $mm/s^2$ | Maximum deceleration of the end effector in the direction it is moving.<br>Parameter: Tangential deceleration limit(see page 75) |
| G_RADIAL_ACCEL_LIMIT | LM | Read-Write | Float | $mm/s^2$ | Maximum acceleration of the end effector in the direction perpendicular to it's movement.<br>Parameter: Radial acceleration limit(see page 75) |
| G_RADIAL_ACCEL_ORIDE | LM | Read-Write | Float | $mm/s^2$ | Scale factor applied to G_RADIAL_ACCEL_LIMIT (see page 80). |
| G_JERK | LM | Read-Write | Float | $mm/s^3$ | Maximum jerk of the end effector in the direction it is moving.<br>Parameter: Tangential jerk limit(see page 75) |
| G_JERK_ORIDE | LM | Read-Write | Float | $mm/s^3$ | Scale factor applied to G_JERK(see page 80). |

| Name | Scope | Access | Type | Units | Description |
|---|---|---|---|---|---|
| G_RADIAL_JERK_LIMIT | LM | Read-Write | Float | $mm/s^3$ | Maximum jerk of the end effector in the direction perpendicular to it's movement.<br>Parameter: Radial jerk limit(see page 75) |
| G_RADIAL_JERK_ORIDE | LM | Read-Write | Float | $mm/s^3$ | Scale factor applied to G_RADIAL_JERK_LIMIT (see page 80). |
| G_TRANSITIONAL_JERK_LIMIT | LM | Read-Write | Float | $mm/s^3$ | Maximum jerk of the end effector during the transition from one move type to another.<br>Parameter: Transitional jerk limit(see page 75) |
| G_TRANSITIONAL_JERK_ORIDE | LM | Read-Write | Float | $mm/s^3$ | Scale factor applied to G_TRANSITIONAL_JERK_LIMIT(see page 80). |
| G_SERVO_SLM<j-1> | CNC | Read-Write | Float | $mm/s^3$ | Minimum position of the joint.<br>Must be within the new limit *and* capable of stopping before reaching it, otherwise an error will occur and the limit won't be applied.<br>Parameter: Joint lower soft limit(see page 74) |
| G_SERVO_SLP<j-1> | CNC | Read-Write | Float | $mm/s^3$ | Maximum position of the joint.<br>Must be within the new limit *and* capable of stopping before reaching it, otherwise an error will occur and the limit won't be applied.<br>Parameter: Joint upper soft limit(see page 74) |

### 9.4.2 System health

These variables will not be updated if you are running AMCore as a simulator.

| Variable | Scope | Access | Type | Units | Description |
|---|---|---|---|---|---|
| G_SYSMON_TEMP_CPU | CNC | Read-Only | Float | °C | Temperature of the CPU.<br>Measured by a designated CPU temperature sensor on the motherboard. |
| G_SYSMON_TEMP_SYS | CNC | Read-Only | Float | °C | Temperature of the main system. |
| G_SYSMON_TEMP_AUX | CNC | Read-Only | Float | °C | Temperature of the auxiliary system. |
| G_SYSMON_TEMP_HDD | CNC | Read-Only | Float | °C | Temperature of the hard disk. |
| G_SYSMON_TEMP_CPU_ZERO | CNC | Read-Only | Float | °C | Temperature of core 0 of the CPU.<br>Only updated for CPUs that measure their core temperatures. |
| G_SYSMON_TEMP_CPU_ONE | CNC | Read-Only | Float | °C | Temperature of core 1 of the CPU.<br>Only updated for CPUs that measure their core temperatures. |
| G_SYSMON_TEMP_CPU_TWO | CNC | Read-Only | Float | °C | Temperature of core 2 of the CPU.<br>Only updated for CPUs that measure their core temperatures. |
| G_SYSMON_TEMP_CPU_THREE | CNC | Read-Only | Float | °C | Temperature of core 3 of the CPU.<br>Only updated for CPUs that measure their core temperatures. |

| Variable | Scope | Access | Type | Units | Description |
|---|---|---|---|---|---|
| G_SYSMON_FAN_RPM_CPU | CNC | Read-Only | Float | RPM | Fan speed of the CPU. |
| G_SYSMON_FAN_RPM_SYS | CNC | Read-Only | Float | RPM | Speed of the system fan.<br>Only updated when a system fan is present. |
| G_SYSMON_FAN_RPM_AUX | CNC | Read-Only | Float | RPM | Speed of the auxiliary fan.<br>Only updated when a auxiliary fan is present. |
| G_SYSMON_VOLT_CPU_VCORE | CNC | Read-Only | Float | V | Core voltage of the CPU |
| G_SYSMON_VOLT_ANLG_THREE_THREE_V | CNC | Read-Only | Float | V | Voltage of the 3.3V analog source of the motherboard. |
| G_SYSMON_VOLT_FIVE_V_RAIL | CNC | Read-Only | Float | V | Voltage of the 5V rail. |
| G_SYSMON_HDD_USED | CNC | Read-Only | Float | % | Percentage of hard disk space used. |
| G_SYSMON_RAM_MEM | CNC | Read-Only | Float | % | Percentage of total RAM used. |
| G_SYSMON_RAM_USED | CNC | Read-Only | Integer | GB | Amount of RAM used. |
| G_SYSMON_RAM_AVAILABLE | CNC | Read-Only | Integer | GB | Amount of RAM available. |
| G_SYSMON_CPU_LOAD | CNC | Read-Only | Float | % | Percentage of total CPU utilization. |
| G_SYSMON_CPU_LOAD_ZERO | CNC | Read-Only | Float | % | Percentage of CPU core 0 utilization. |
| G_SYSMON_CPU_LOAD_ONE | CNC | Read-Only | Float | % | Percentage of CPU core 1 utilization. |
| G_SYSMON_CPU_LOAD_TWO | CNC | Read-Only | Float | % | Percentage of CPU core 2 utilization. |
| G_SYSMON_CPU_LOAD_THREE | CNC | Read-Only | Float | % | Percentage of CPU core 3 utilization. |

| Variable | Scope | Access | Type | Units | Description |
|---|---|---|---|---|---|
| G_SYSMON_CPU_BUS_SPEED | CNC | Read-Only | Float | MHz | Clock speed of the CPU.<br>The value is constant. |
| G_SYSMON_FAN_SYS_PRESENT | CNC | Read-Only | Boolean | - | A value indicating whether or not the system fan is present. |
| G_SYSMON_FAN_AUX_PRESENT | CNC | Read-Only | Boolean | - | A value indicating whether or not the auxiliary fan is present. |
| G_SYSMON_CPU_ONE_PRESENT | CNC | Read-Only | Boolean | - | A value indicating whether or not CPU core 1 is present. |
| G_SYSMON_CPU_TWO_PRESENT | CNC | Read-Only | Boolean | - | A value indicating whether or not CPU core 2 is present. |
| G_SYSMON_MONITORING_ACTIVE | CNC | Read-Only | Boolean | - | A value indicating whether or not the system monitoring is running. |

# 10 Troubleshoot

## 10.1 AMCore.exe exit codes

**In this section**



**AMCore.exe** is AMCore's entry point. It is used to setup, startup, and stop AMCore, as described in Start AMCore(see page 22).

AMCore.exe returns an exit code to indicate the success of the requested operation. When an error occurs, this exit code may indicate the nature of the error.

Different operations (setup, startup, etc.) have different sets of exit codes.

### 10.1.1 Setup exit codes

| Exit code | Description |
|---|---|
| 0 | Setup succeeded. |
| 1 | Setup failed. |

### 10.1.2 Startup exit codes

| Exit code | Description |
|---|---|
| 0 | Startup succeeded. |
| 1 | Startup failed for an unknown reason. |
| 10001 | Processing of configuration properties failed for an unknown reason. |
| 10002 | Failed to parse command-line arguments. |
| 10003 | Failed to launch AMCore for an unknown reason. |
| 10004 | Failed to resolve a configuration file's path to an absolute filepath. |
| 10005 | Failed to open a configuration file. |

| Exit code | Description |
|---|---|
| 10006 | Failed to parse a configuration file. |
| 10007 | A configuration file has an invalid version. |
| 10008 | A configuration file contains a configuration property that is unrecognised. |
| 10009 | A configuration file contains a configuration property of an incorrect type. |
| 10010 | A configuration file contains a configuration property that cannot be resolved to an absolute filepath. |
| 10011 | Failed to update a configuration property for a parameter file. |
| 10012 | A parameter file is missing from its configured custom location. |
| 10013 | Failed to create a writeable parameter file. |
| 10014 | Failed to set AMCore's configuration properties. |
| 10015 | AMCore is already running. |

In AMCore 1.7 and earlier, the only startup exit codes are 0 (startup succeeded) or 1 (startup failed).

# 11 Contact Information

## 11.1 General Enquires

https://motion.anca.com/Contact

## 11.2 ANCA Motion Pty. Ltd.

1 Bessemer Road,
Bayswater North,
VIC 3153,
Australia

Telephone: +61 3 9751 8900

Fax: +61 3 9751 8901

Email: sales.au@ancamotion.com[3]

## 11.3 ANCA Motion Taiwan

4F, No. 63, Jingke Central Road, Nantun District,
Taichung City 40852,
Taiwan

Telephone: +886 4 2359 0082

Fax: +886 4 2359 0067

Email: sales.tw@ancamotion.com[4]

## 11.4 ANCA Motion (Tianjin) Co., Ltd.

No. 102, Building F1,
XEDA Emerging Industrial Park,
Xiqing Economic-technological Development Area,
Tianjin, P.R.China

Telephone: +86 22 5965 3760

Fax: +86 22 5965 3761

Email: sales.cn@ancamotion.com[5]

3 mailto:sales.au@ancamotion.com
4 mailto:sales.tw@ancamotion.com
5 mailto:sales.cn@ancamotion.com