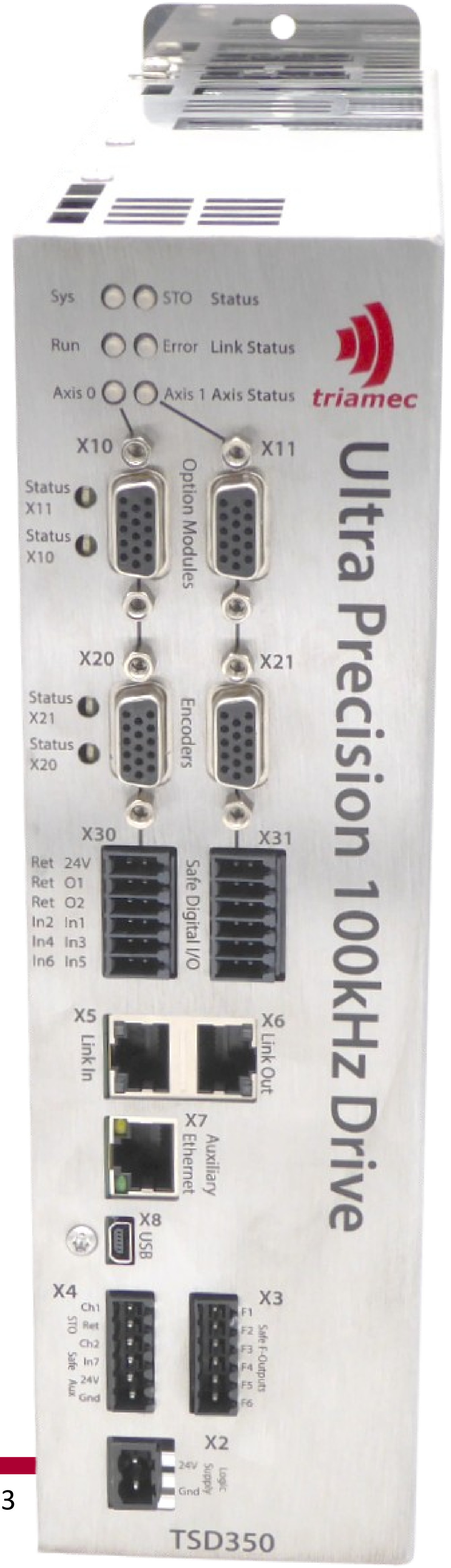
Controlling Triamec Drives through TAM API

Initial Training

- ***Control an axis***
  - Development Environment
  - Basic functions of the *TAM API*
- ***Background***
  - Topology, registers, configuration and simulation
  - Commissioning tool: *TAM System Explorer*
- ***Advanced tasks***
  - Move sequence
  - Measurement



Controlling Triamec Drives through TAM API

Initial Training – Part I

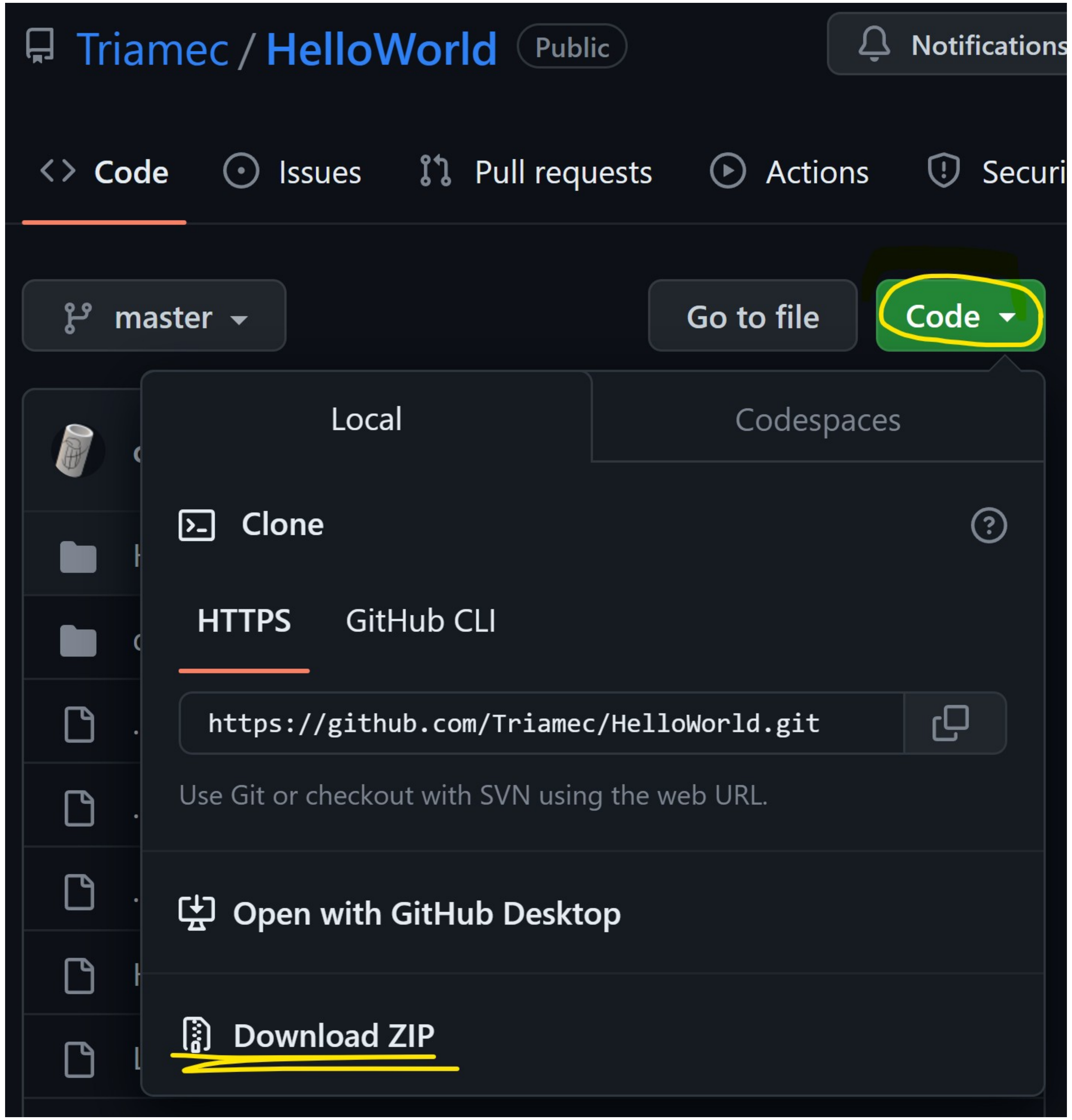
- ***Control an axis***
  - Development Environment
  - Basic functions of the *TAM API*
- ***Background***
  - Topology, registers, configuration and simulation
  - Commissioning tool: *TAM System Explorer*
- ***Advanced tasks***
  - Move sequence
  - Measurement

Development Environment

- *Microsoft Windows 10* – 64-bit recommended – or *Windows 11*
- One of those:
  - *Visual Studio 2017 Express* – free
    - Install the *.NET Framework 4.8 Developer pack*
  - *Visual Studio 2019* or *2022* – free for open source projects and small organizations
    - Select the *.NET desktop development* workload
    - Add the optional *.NET Framework 4.8 development tools*
- Install the *TAM Software*

Development Environment

- Browse to https://github.com/Triamec/HelloWorld
- Get the sample, for example as ZIP
  - Extract `.zip` to your documents



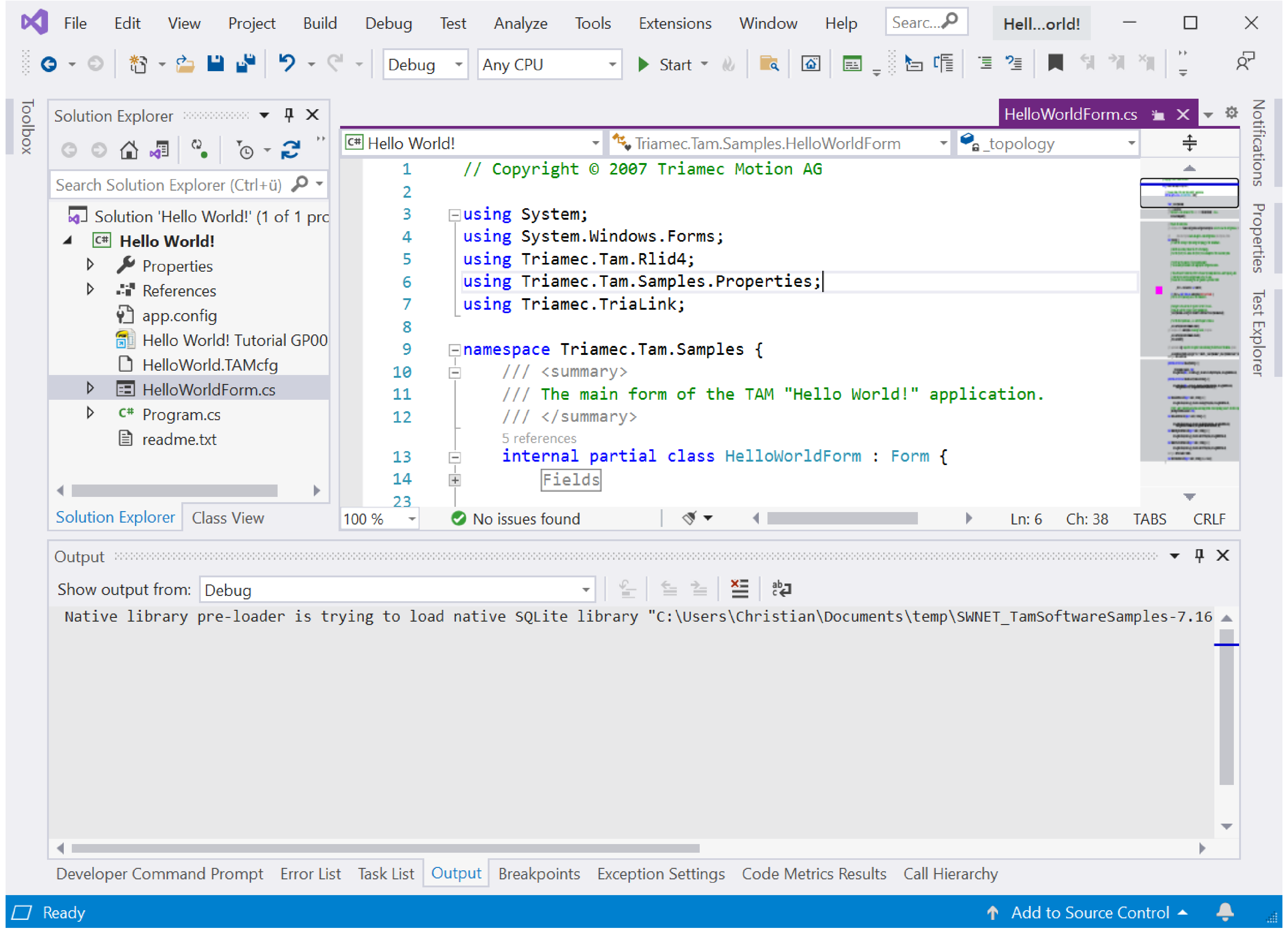
Triamec / HelloWorld Public

https://github.com/Triamec/HelloWorld.git

Use Git or checkout with SVN using the web URL.

Development Environment

- Open solution file
  `Hello World!\Hello World!.sln`
- Build the solution (Menu **Build > Build Solution**)



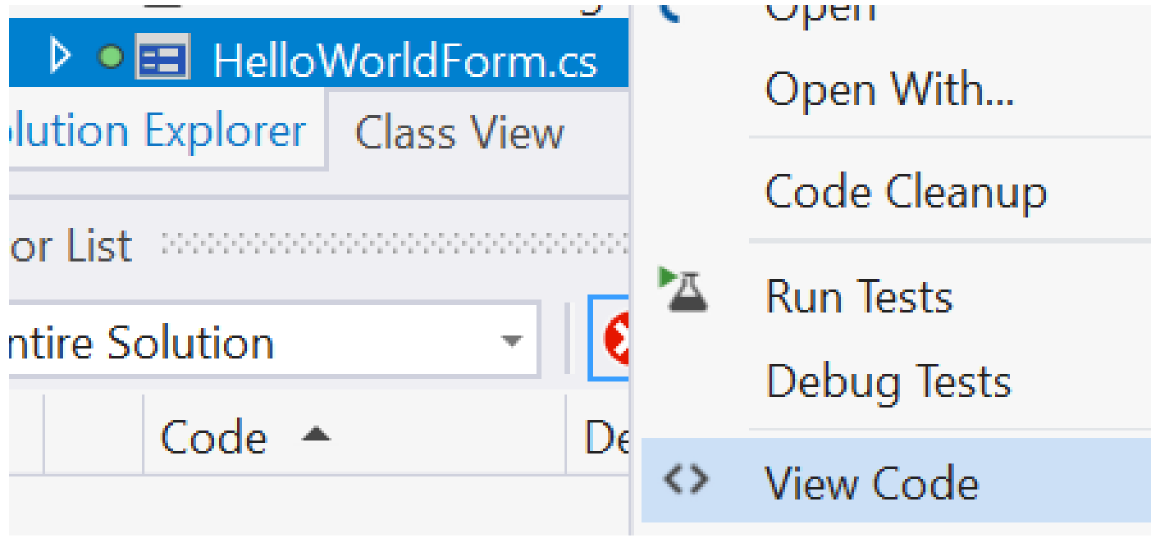
```
// Copyright © 2007 Triamec Motion AG

using System;
using System.Windows.Forms;
using Triamec.Tam.Rlid4;
using Triamec.Tam.Samples.Properties;
using Triamec.TriaLink;

namespace Triamec.Tam.Samples {
    /// <summary>
    /// The main form of the TAM "Hello World!" application.
    /// </summary>
    5 references
    internal partial class HelloWorldForm : Form {
        Fields
```

Show output from: Debug

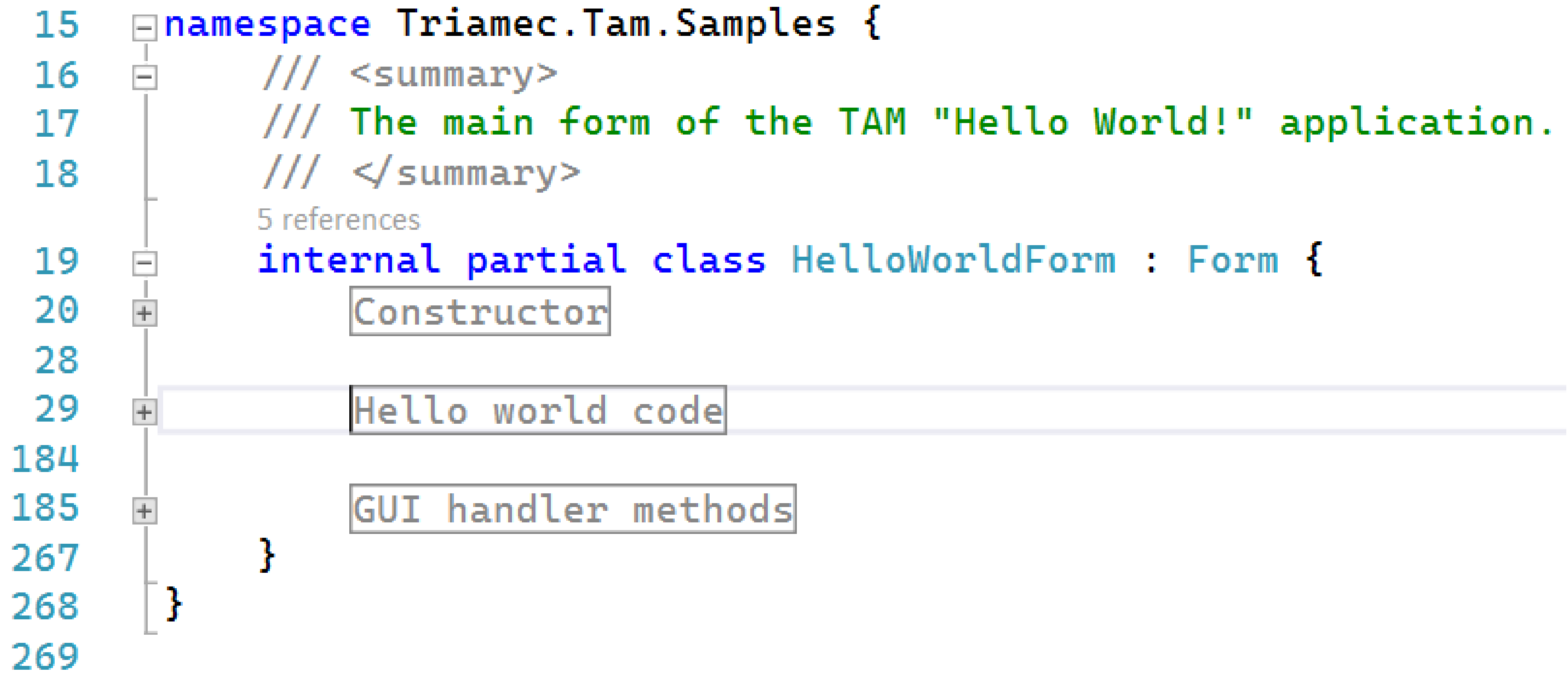
```
Native library pre-loader is trying to load native SQLite library "C:\Users\Christian\Documents\temp\SWNET_TamSoftwareSamples-7.16
```

Simulate the Drive

- Study `HelloWorldForm.cs` in Text Editor
  - Open using right click

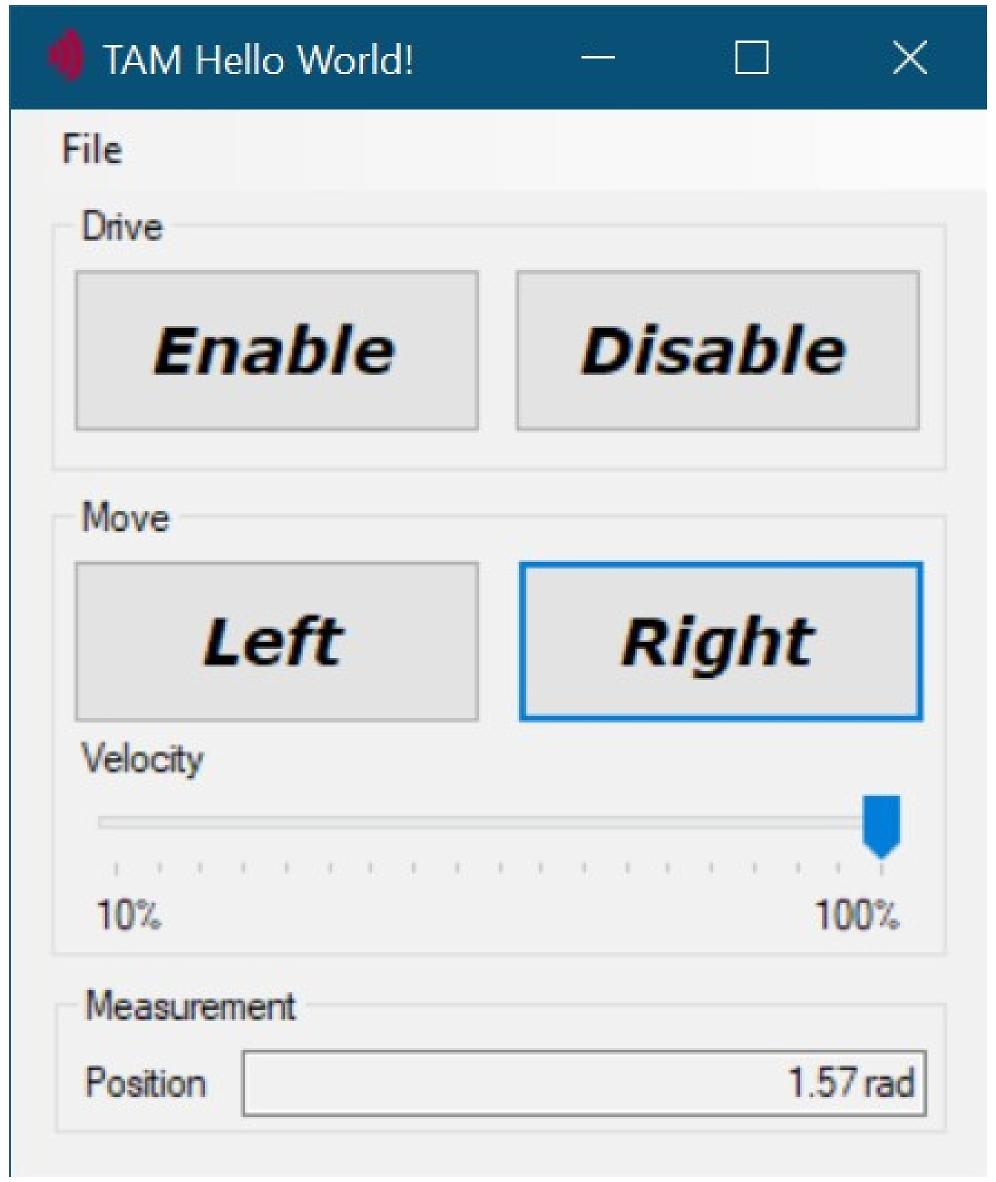


  - Expand the *Hello world code* region



```
namespace Triamec.Tam.Samples {
    /// <summary>
    /// The main form of the TAM "Hello World!" application.
    /// </summary>
    5 references
    internal partial class HelloWorldForm : Form {
        Constructor

        Hello world code

        GUI handler methods
    }
}
```

- Start the app with F5
  - A simulation of a drive is used
  - Set breakpoints to step through the code



Position 1.57 rad

Access Drive

- Create the root object of the structure

```
_topology = new TamTopology("Tutorial");
```

- Add local system to the structure

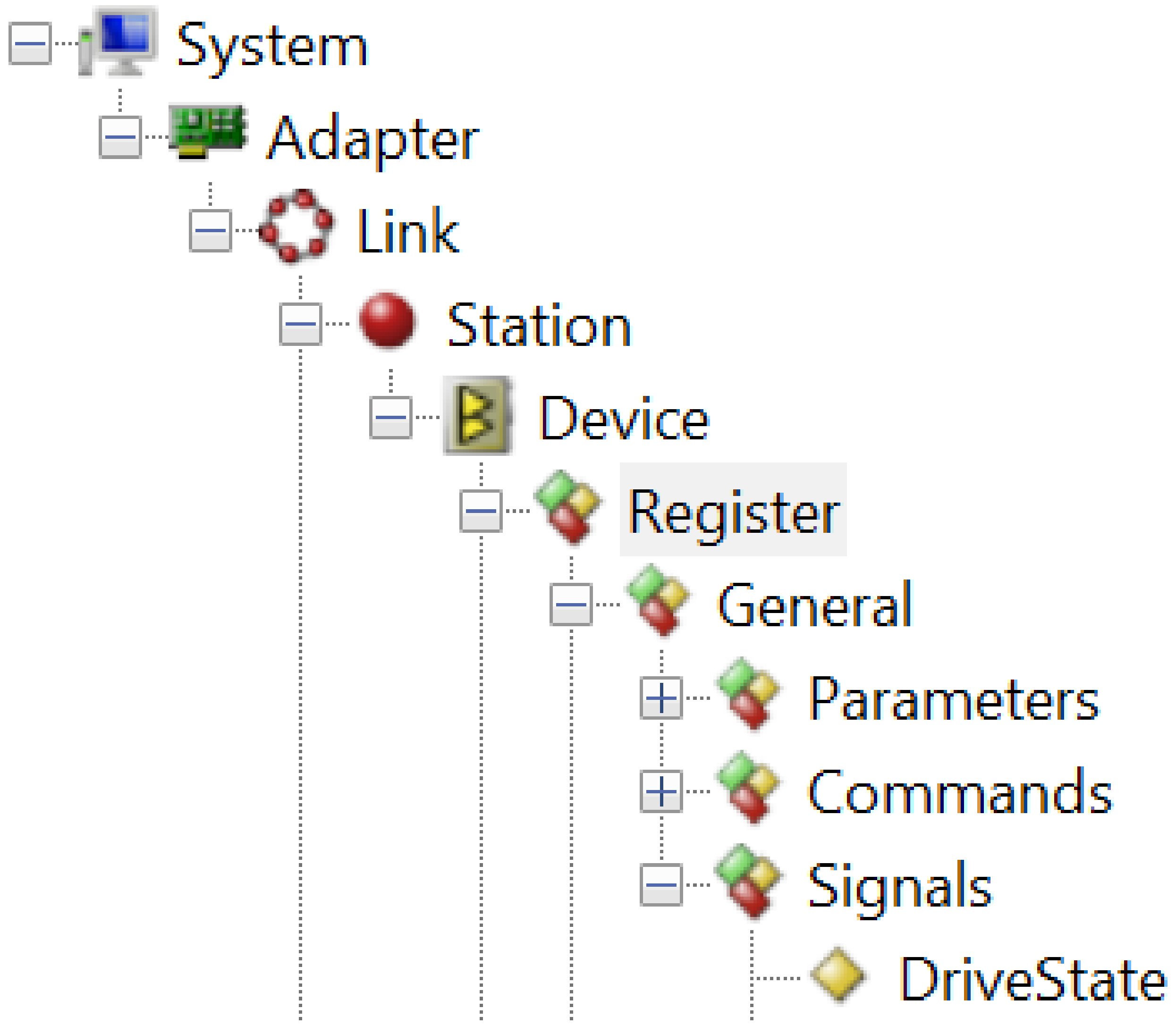
```
system = _topology.AddLocalSystem();
```

- Add connected drives to the structure

```
system.Identify();
```

- Locate axis in the structure (various strategies possible)

```
_axis = system.AsDepthFirstLeaves<TamAxis>()
              .FirstOrDefault(a => a.Name == AxisName);
```

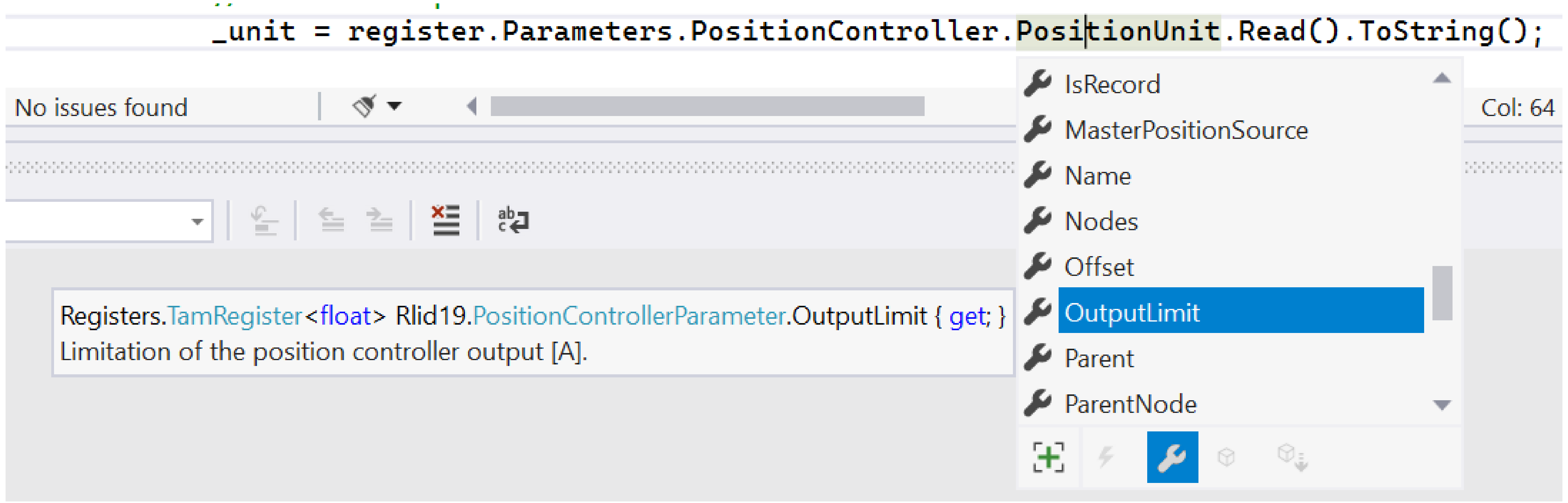


Read State

- Access the drive’s parametrization and state

```
    // Get the register layout of the axis
    // and cast it to the RLID-specific register layout.
    var register = (Axis)_axis.Register;

    // Read and cache the original velocity maximum value,
    // which was applied from the configuration file.
    _velocityMaximum = register.Parameters.PathPlanner.VelocityMaximum.Read();
```

- Let *code completion* assist you



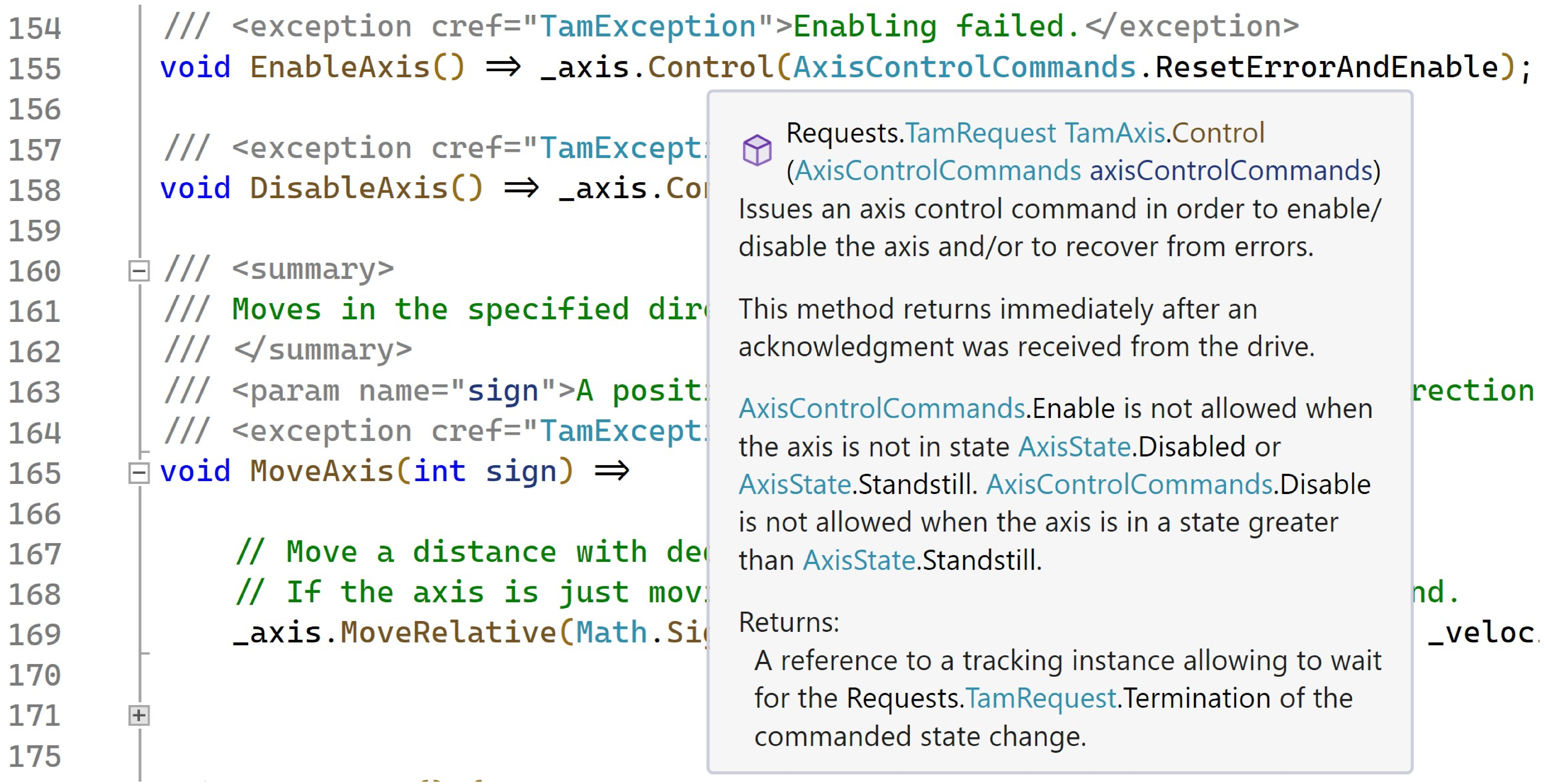
```
        _unit = register.Parameters.PositionController.PositionUnit.Read().ToString();
```

No issues found

Col: 64

Registers.TamRegister<float> Rlid19.PositionControllerParameter.OutputLimit { get; }
Limitation of the position controller output [A].

Enable Controller



```
/// <exception cref="TamException">Enabling failed.</exception>
void EnableAxis() => _axis.Control(AxisControlCommands.ResetErrorAndEnable);

/// <exception cref="TamExcept
void DisableAxis() => _axis.Co

/// <summary>
/// Moves in the specified dir
/// </summary>
/// <param name="sign">A posit                          rection
/// <exception cref="TamExcept
void MoveAxis(int sign) =>

    // Move a distance with de
    // If the axis is just mov                          nd.
    _axis.MoveRelative(Math.Si                          _veloc
```

Requests.TamRequest TamAxis.Control (AxisControlCommands axisControlCommands)

Issues an axis control command in order to enable/disable the axis and/or to recover from errors.

This method returns immediately after an acknowledgment was received from the drive.

AxisControlCommands.Enable is not allowed when the axis is not in state AxisState.Disabled or AxisState.Standstill. AxisControlCommands.Disable is not allowed when the axis is in a state greater than AxisState.Standstill.

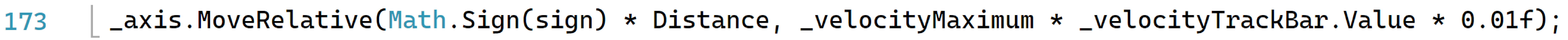
Returns:

A reference to a tracking instance allowing to wait for the Requests.TamRequest.Termination of the commanded state change.

- Take notice of inline documentation (shown when hovering or typing)

Command Movement

- Operate on the `TamAxis` object



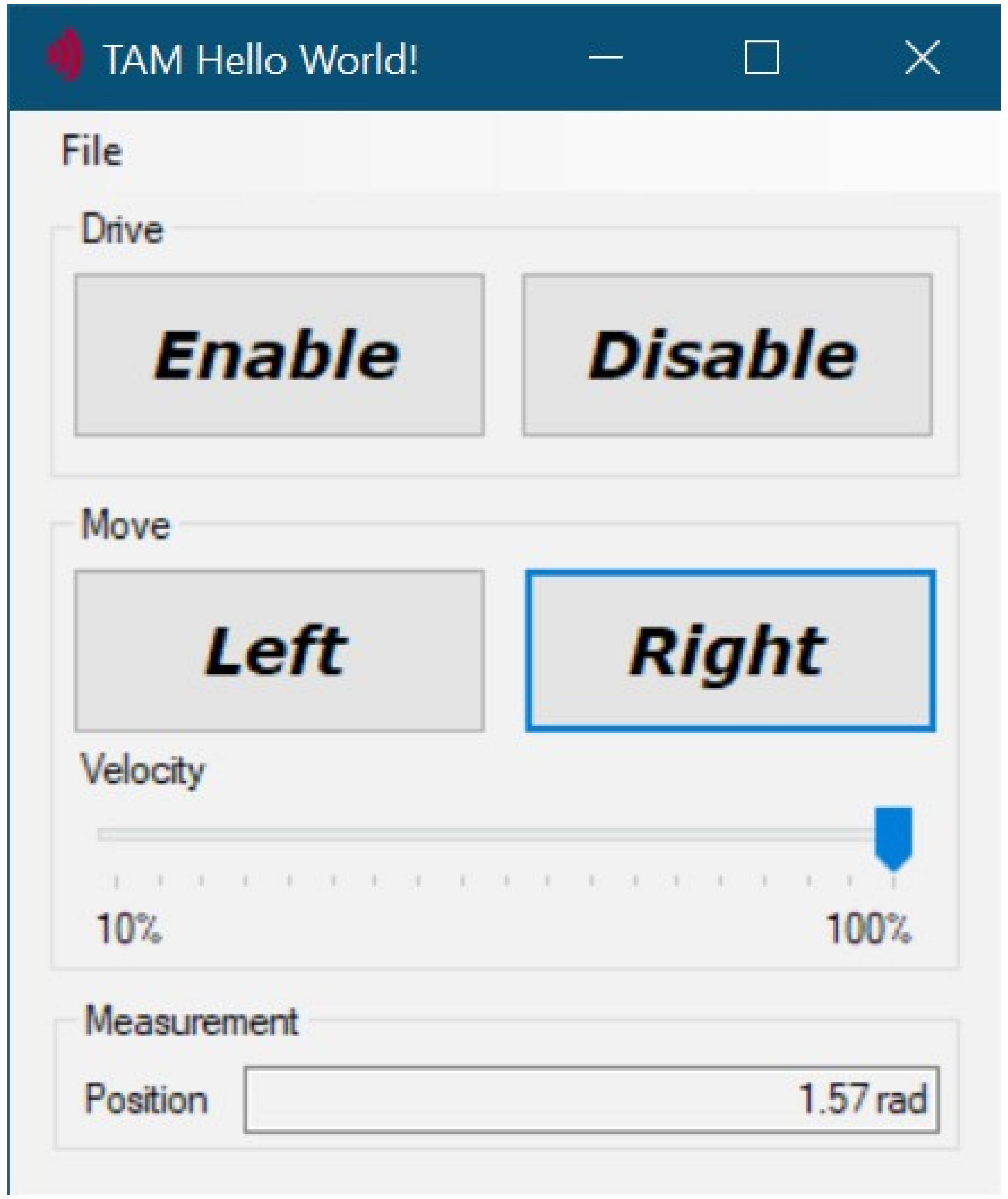
```
_axis.MoveRelative(Math.Sign(sign) * Distance, _velocityMaximum * _velocityTrackBar.Value * 0.01f);
```

- Often used methods:

– `MoveAbsolute`	– Move to position
– `MoveVelocity`	– Move with constant velocity
– `Stop`	– Return to stand still
– `SetPosition`	– Apply an offset to the position axis

Run the App

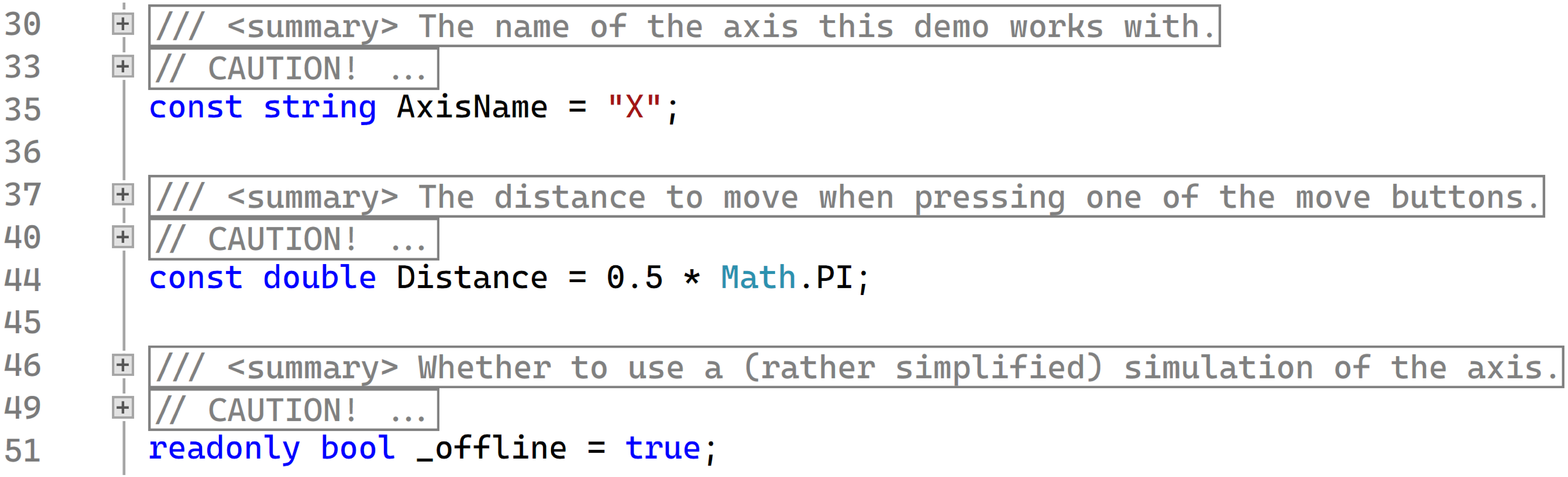
- Check whether
  - the controller can be enabled
  - the position changes
    - when moving
    - at a lower rate at 10% velocity
  - a move can be reprogrammed with a new move



Position 1.57 rad

Customize the App

- Modify the constants to adapt to your hardware



```
/// <summary> The name of the axis this demo works with.
// CAUTION! ...
const string AxisName = "X";

/// <summary> The distance to move when pressing one of the move buttons.
// CAUTION! ...
const double Distance = 0.5 * Math.PI;

/// <summary> Whether to use a (rather simplified) simulation of the axis.
// CAUTION! ...
readonly bool _offline = true;
```

Achievements

- Set up development environment
- Working code at disposal
- Control an already configured axis
- Examine the state of axis and drive
- Employ inline help to examine the API surface

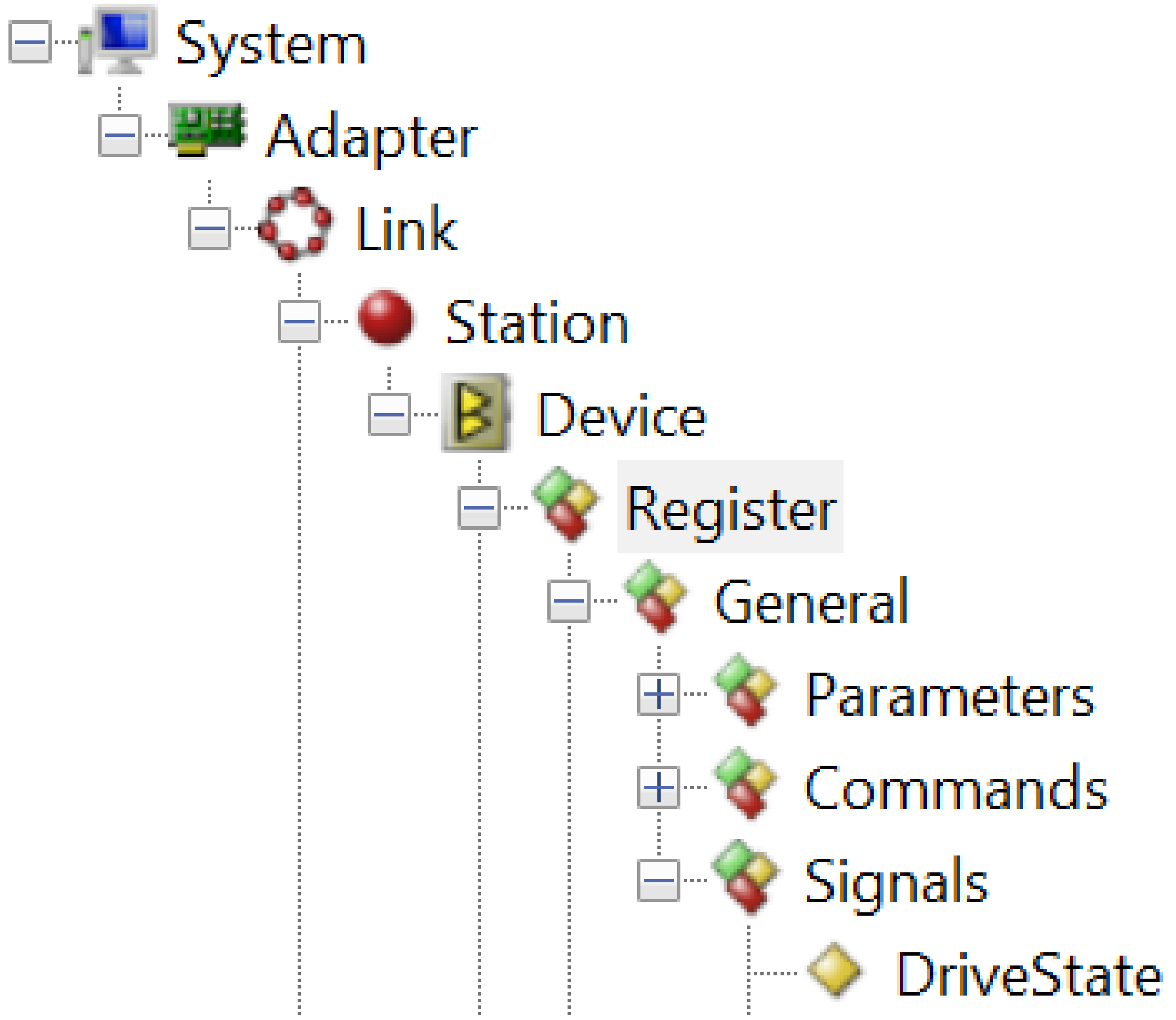
Controlling Triamec Drives through TAM API

Initial Training – Part II

- *Control an axis*
  - Development Environment
  - Basic functions of the *TAM API*
- ***Background***
  - Topology, registers, configuration and offline mode
  - Commissioning tool: *TAM System Explorer*
- *Advanced tasks*
  - Move sequence
  - Measurement

Communication Topology

- *Topology* Top of hierarchy (not shown in TAM System Explorer)
- *System* Represents the local or a remote computer
- *Adapter* Computer hardware used to access the drive
- *Link* Communication layer, represents the wire
- *Station* Addressable party within the link
- *Device* Represents the microprocessor of the drive
- *Register* Structured representation of parametrization, commands and visible state (signals) of the drive.

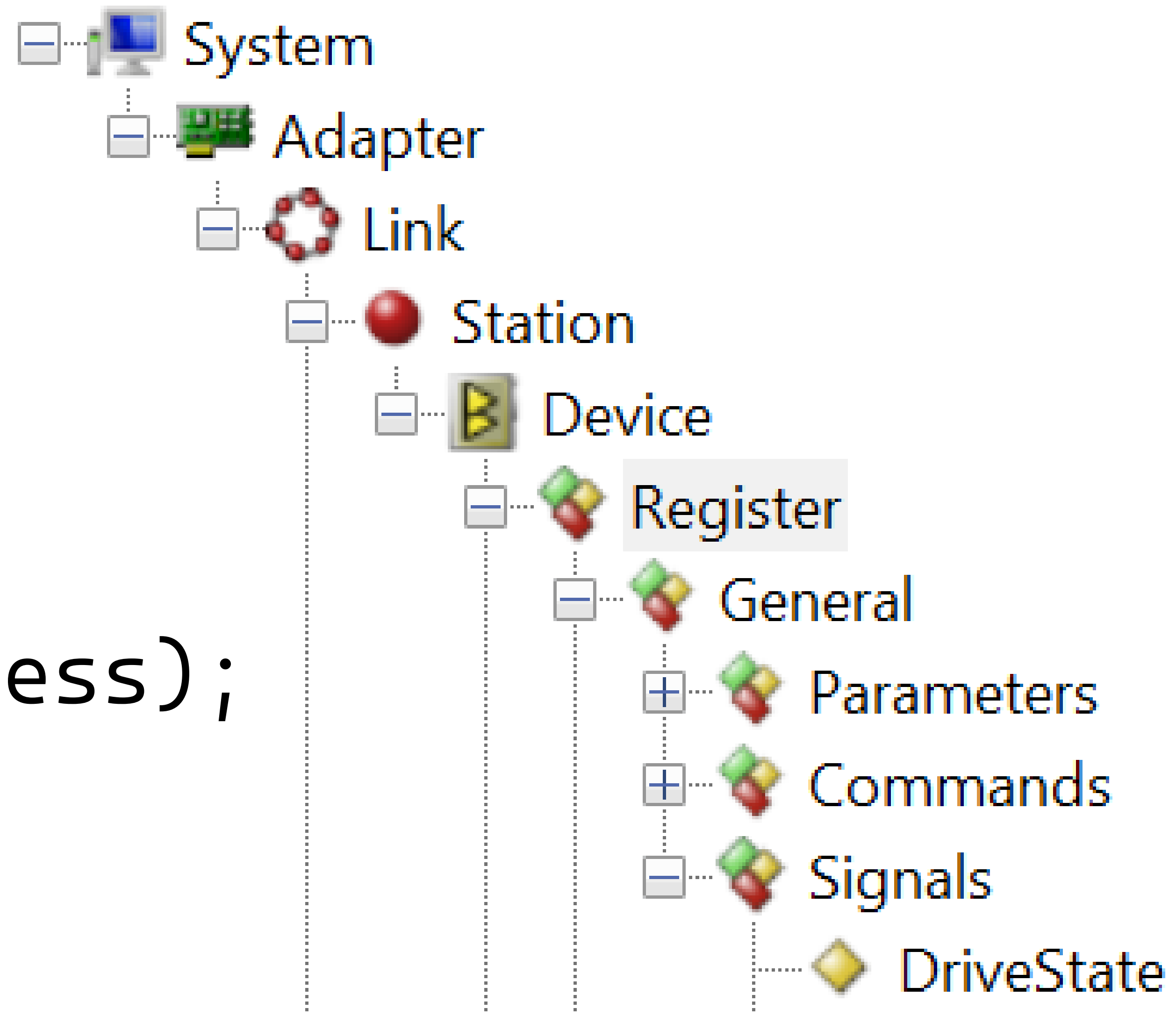


Navigate the Structure

- Top-down
  - Names `System.Adapters[0].Link.Stations[0]…`
  - Generic `System.Nodes[0].Nodes[0]…`
  - Search

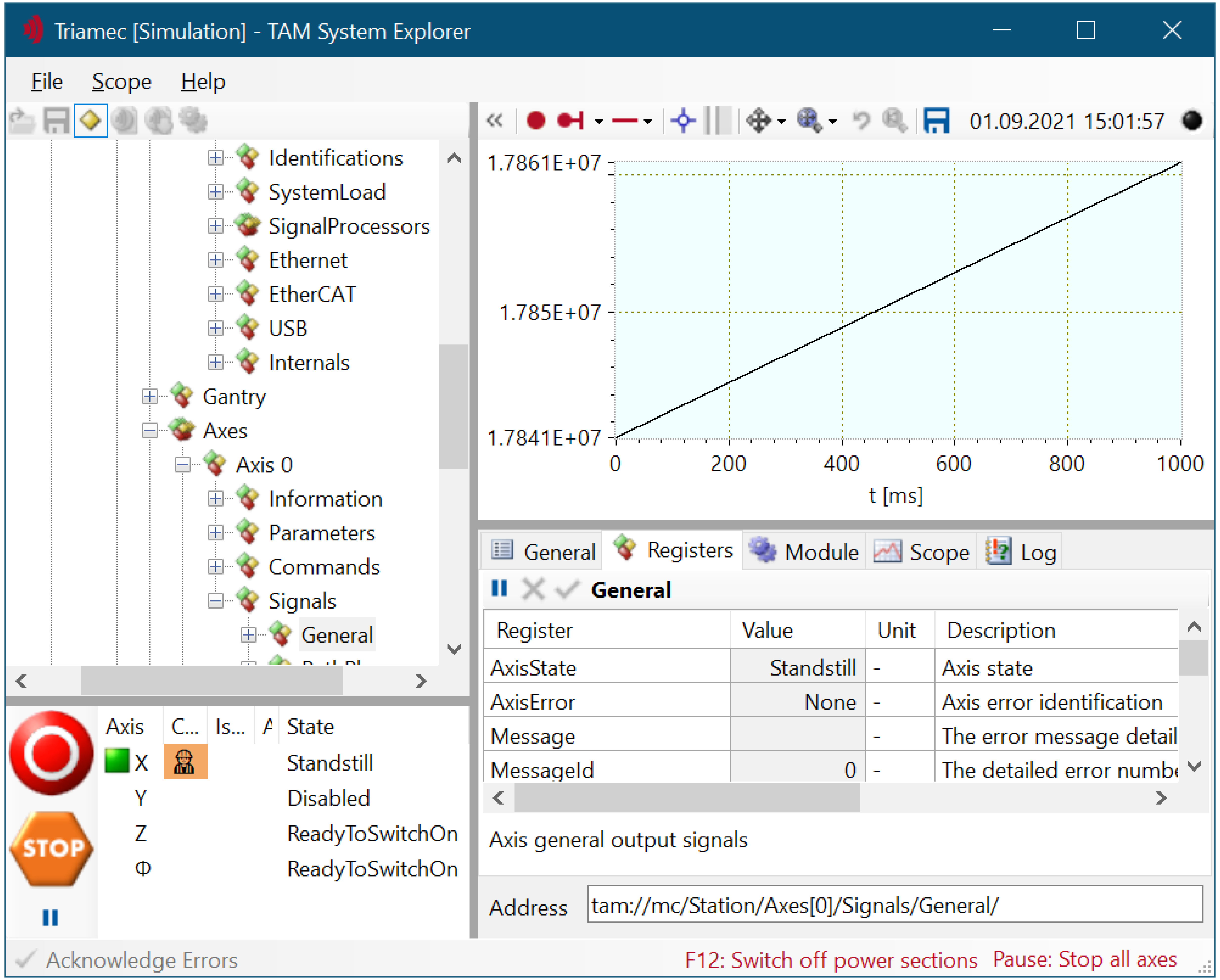
    ```
    var address = new Uri("tam://mc/X");
    station =
       (TamStation)system.FindTamNode(address);
    ```

  - Cast `ITamDevice` to `ITamDrive`
  - Access `TamAxis` instances with `ITamDrive.Axes`
- Bottom-up
  - Step by step `device.Station.Link.Adapter.System`
  - Generic `device.ParentNode.ParentNode…`
  - Go to root `device.NavigateToRoot()`



Commissioning

- Use the *TAM System Explorer* to
  - commission the axis
  - tune the controller
  - manage configuration
  - diagnose and measure
  - run side-by-side with app:
    - Tria-Link and USB: exclusive!
    - Ethernet: possible
- Homepage
- Servo Drive Setup Guide



Axis	C...	Is...	A	State
🟩 X				Standstill
Y				Disabled
Z				ReadyToSwitchOn
Φ				ReadyToSwitchOn

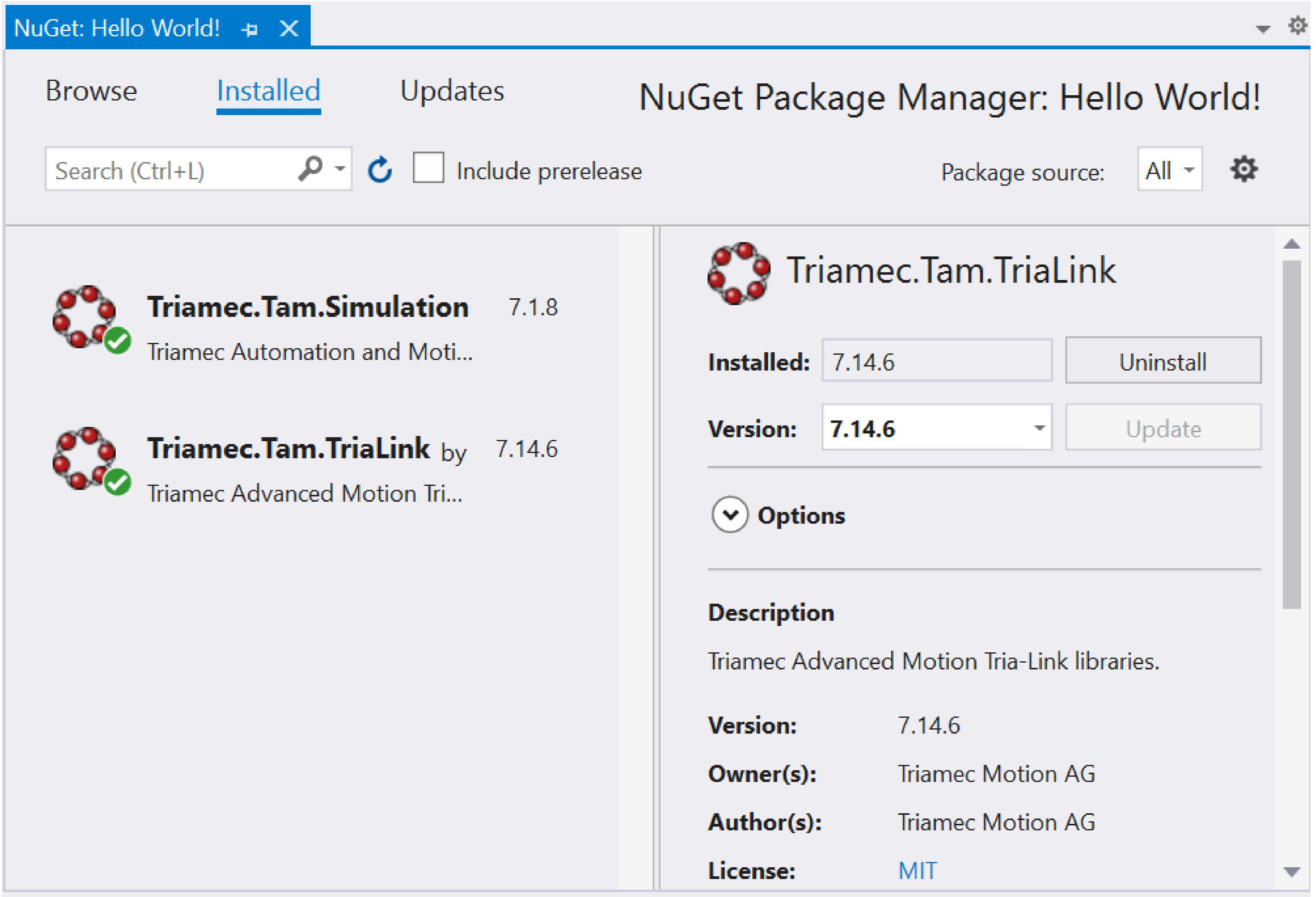
Register	Value	Unit	Description
AxisState	Standstill	-	Axis state
AxisError	None	-	Axis error identification
Message		-	The error message detail
MessageId	0	-	The detailed error numb

Axis general output signals

Address tam://mc/Station/Axes[0]/Signals/General/

Development Environment

- Manage dependencies with *NuGets*
  - Menu **Project > Manage NuGet Packages…**
  - Application runs independently from any installed TAM Software, apart from drivers.
  - Update to a newer version of the TAM Software with ease.
- When to use which NuGets:
  - *Triamec.Tam.TriaLink*: Applications
  - *Triamec.Tools.TamaCompiler* Projects with Tama programs
  - *Triamec.Tam.UI* Integrate TAM System Explorer
  - *Triamec.Tam.Simulation* NuGet not needed in production.



NuGet Package Manager: Hello World!

☐ Include prerelease

Package source: All

Triamec.Tam.Simulation 7.1.8
Triamec Automation and Moti...

Triamec.Tam.TriaLink by 7.14.6
Triamec Advanced Motion Tri...

Triamec.Tam.TriaLink

Installed: 7.14.6

Version: 7.14.6

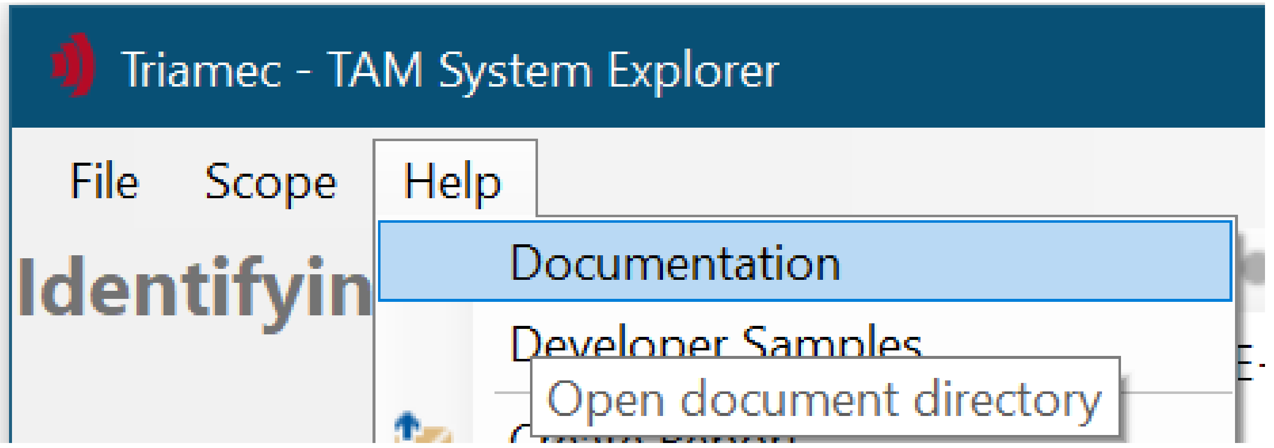
Description

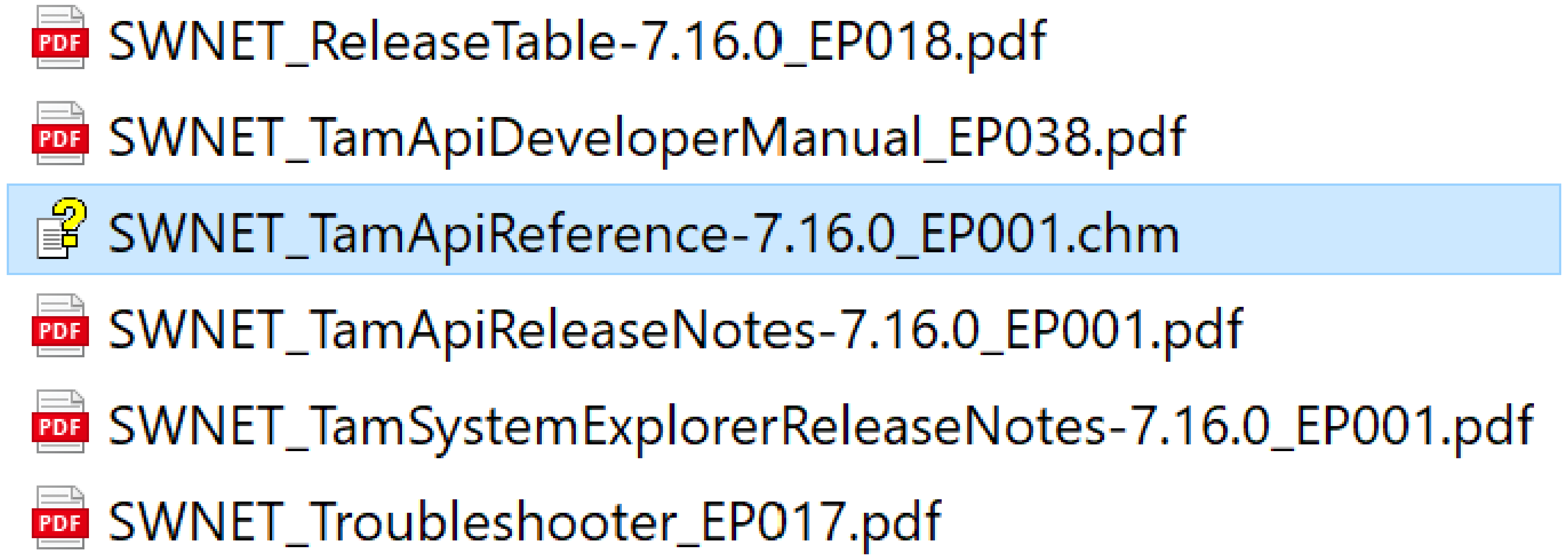
Triamec Advanced Motion Tria-Link libraries.

Version: 7.14.6
Owner(s): Triamec Motion AG
Author(s): Triamec Motion AG
License: MIT

TAM API

- Homepage
- Access offline help via TAM System Explorer
  - Browse to *Software* folder
    - Developer Manual
    - TAM API Reference



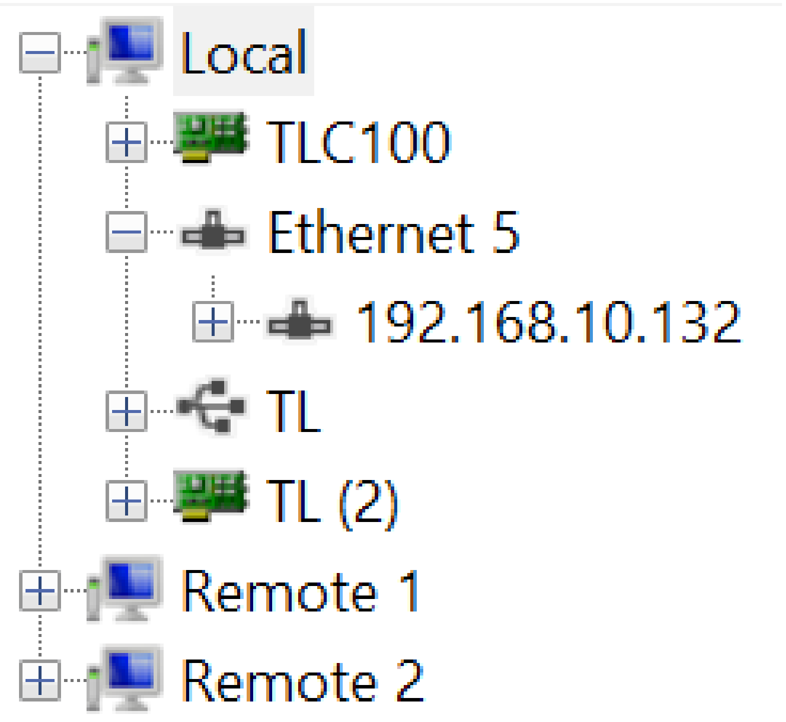

SWNET_ReleaseTable-7.16.0_EP018.pdf
SWNET_TamApiDeveloperManual_EP038.pdf
SWNET_TamApiReference-7.16.0_EP001.chm
SWNET_TamApiReleaseNotes-7.16.0_EP001.pdf
SWNET_TamSystemExplorerReleaseNotes-7.16.0_EP001.pdf
SWNET_Troubleshooter_EP017.pdf

Set Up Communication

- Force specific communication channel

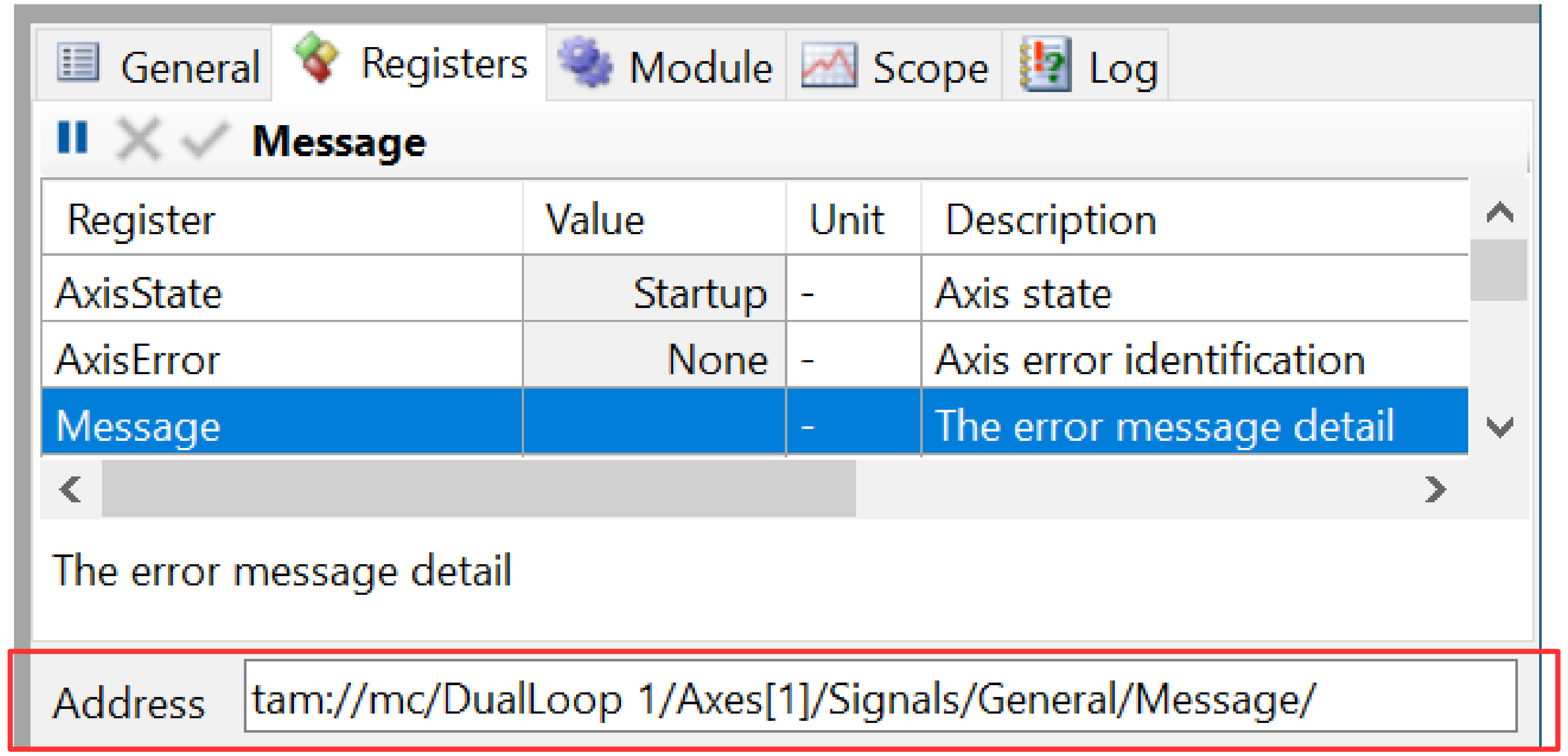
```
// Access the drive via Auxiliary Ethernet. Consult application note AN123 for correct setup. In particular,
// make sure to take into account the firewall. If you can connect to the drive but not acquire data, this
// is likely due to the firewall.
//var access = DataLinkLayers.Network;

// Access the drive via PCI card (not recommended when TwinCAT runs on the same system)
//var access = DataLinkLayers.TriaLink;

// Connect the drive with a USB cable to the PC (not recommended with harsh electromagnetic environments)
// Also works with a Tria-Link PCI adapter connected via USB to the measuring PC.
var access = DataLinkLayers.TriaLinkUsb;

var system = _topology.AddLocalSystem(access);

// Scan the Tria-Link in order to learn about connected stations.
system.Identify();
```

- Specifying the channel speeds up start-up.



Navigate to Registers

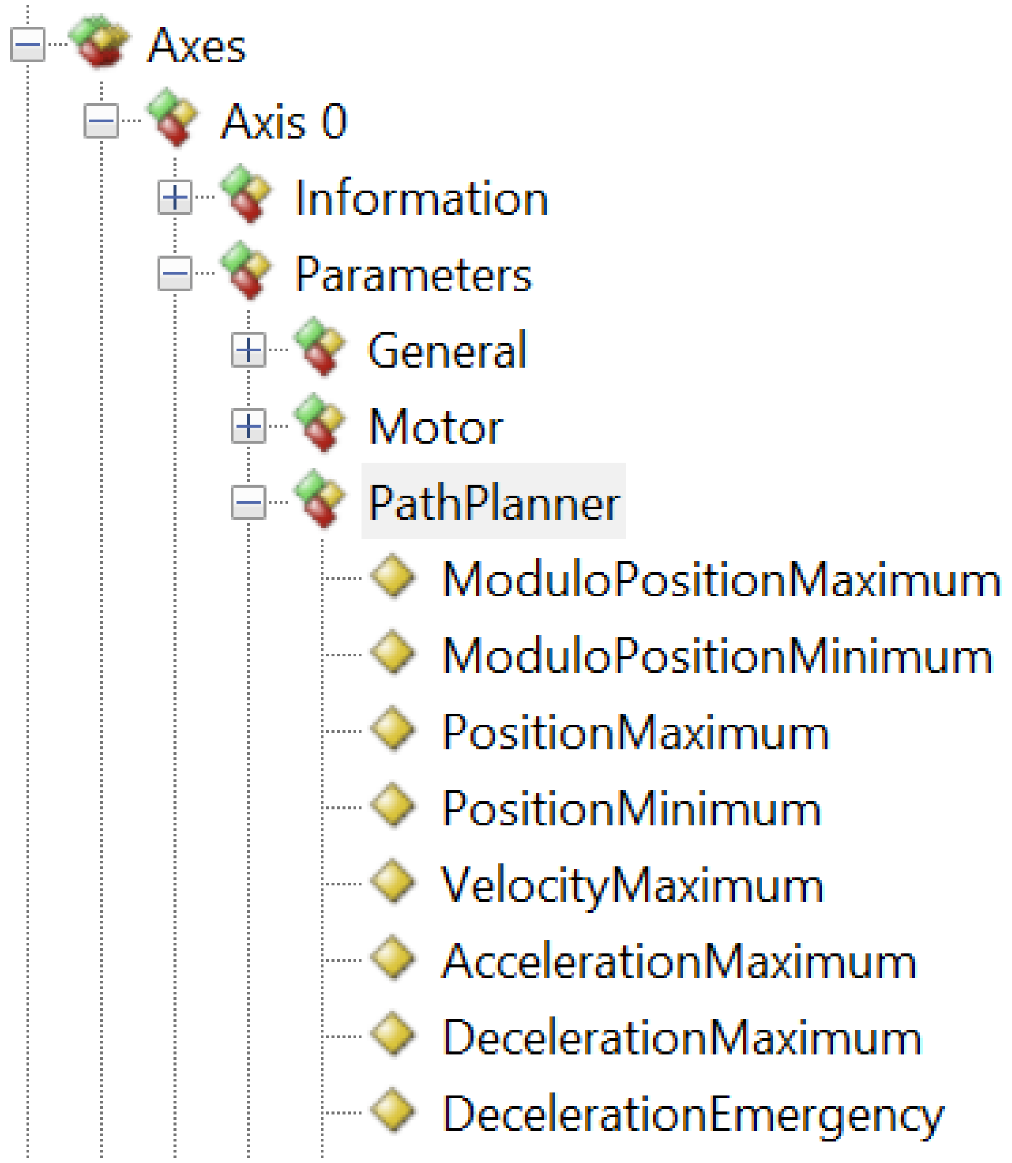
- Import Namespace `using Triamec.Tam.Rlid19;`
- Start at Drive `((Register)drive.Register).General…`
- Start at Axis `((Axis)axis.Register).Parameters…`
- Use TAM System Explorer when writing register access code
  - Visualize the structure
  - Copy & Paste from Address bar at bottom of register grid



Register	Value	Unit	Description
AxisState	Startup	-	Axis state
AxisError	None	-	Axis error identification
Message		-	The error message detail

The error message detail

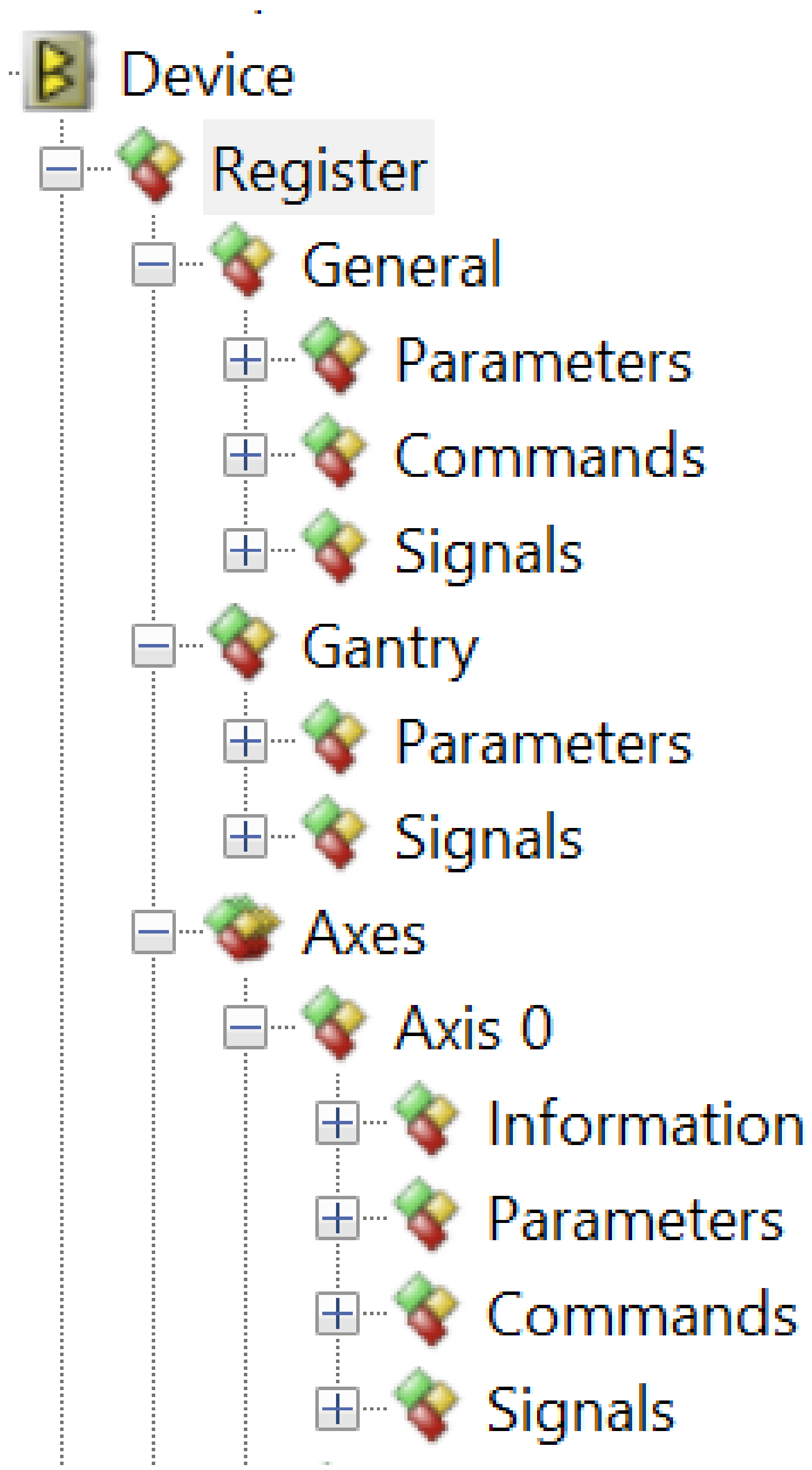
Address tam://mc/DualLoop 1/Axes[1]/Signals/General/Message/



- Axes
  - Axis 0
    - Information
    - Parameters
      - General
      - Motor
      - PathPlanner
        - ModuloPositionMaximum
        - ModuloPositionMinimum
        - PositionMaximum
        - PositionMinimum
        - VelocityMaximum
        - AccelerationMaximum
        - DecelerationMaximum
        - DecelerationEmergency

Register Classes

- *Parameters*
  Configure the drive for the intended application
- *Commands*
  Change state, for example to set a digital output
- *Signals*
  Observe public state in real-time
- *Information*
  Document properties of an axis. Not affecting behavior of the drive in any way

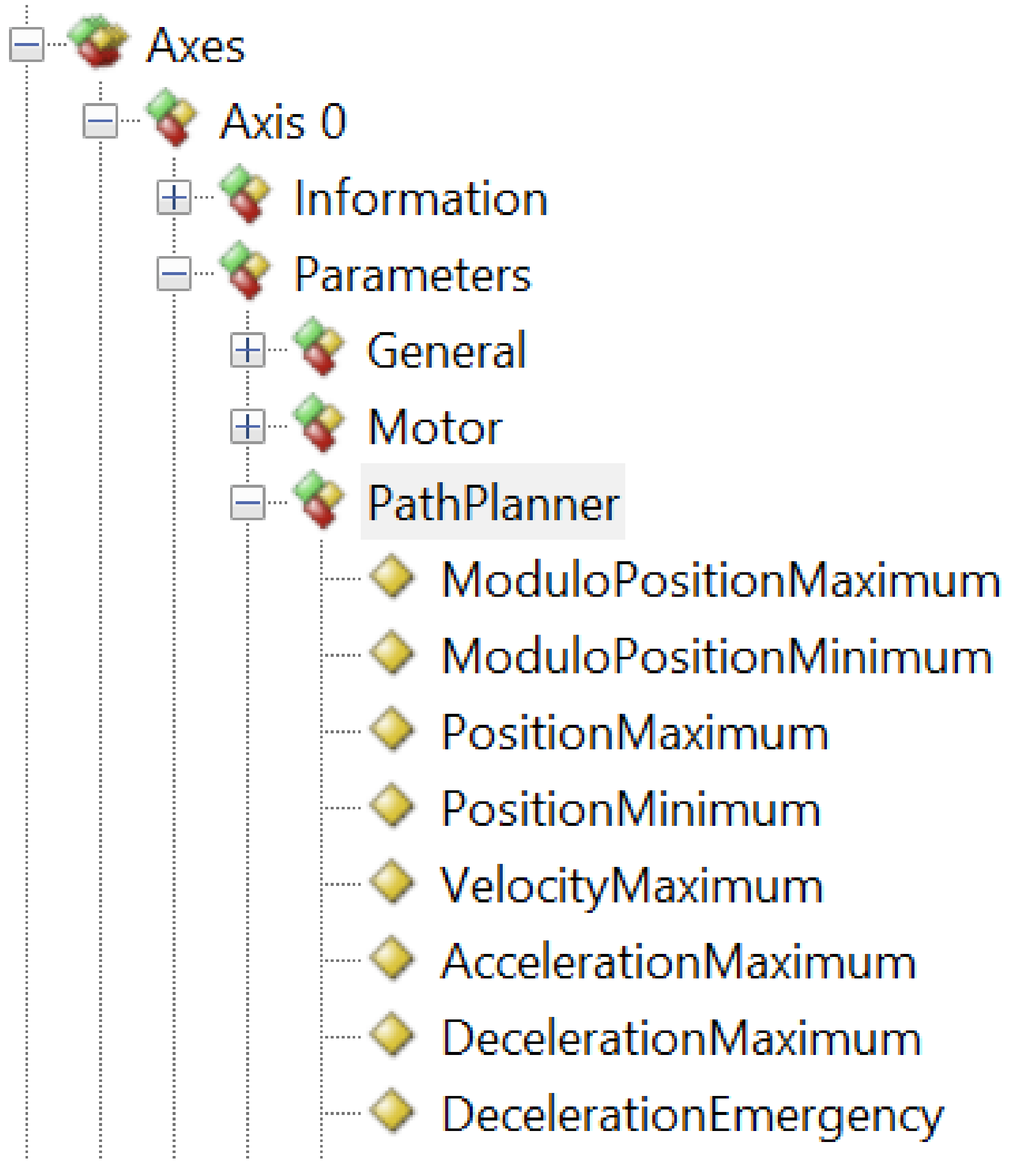


Operate on Leaf Registers

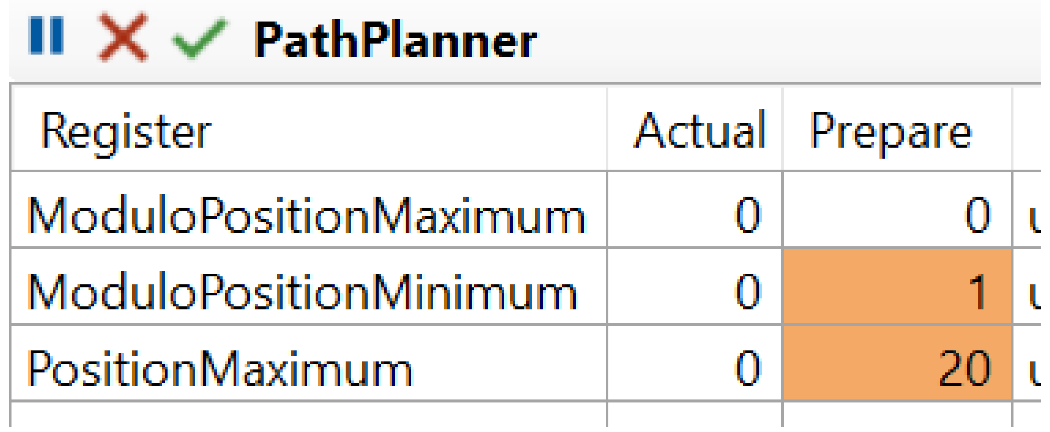
- Get value `int value = register1.Read();`
- Set value `register2.Write(32f);`
- Apply *parameters* after having written a set of them.

```
parameterRegister1.Write(x);
parameterRegister2.Write(y);
parameterRegister3.Write(z);
parameterRegister1.Commit();
```

  - One `Commit` call on any of these parameters suffices
  - Ensures atomic change
  - Commit returns as soon as the drive has applied parametrization
  - Registers apart from parameters don’t need to be committed



- Axes
  - Axis 0
    - Information
    - Parameters
      - General
      - Motor
      - PathPlanner
        - ModuloPositionMaximum
        - ModuloPositionMinimum
        - PositionMaximum
        - PositionMinimum
        - VelocityMaximum
        - AccelerationMaximum
        - DecelerationMaximum
        - DecelerationEmergency



PathPlanner

Register	Actual	Prepare
ModuloPositionMaximum	0	0
ModuloPositionMinimum	0	1
PositionMaximum	0	20

Configuration

- The *TAM Configuration* (`.TAMcfg`) is an XML text file containing the parametrization of all drives in a system.
- Applied to the system during commissioning and persisted on the drives
- It follows that for many scenarios, an app doesn’t need to configure the drive.
- In a system with multiple drives, drives are matched by the *General.Parameters.DeviceName* parameter.
  Despite that name, the *DeviceName* is correlated with the `TamStation`’s name!

Offline Environment

- Examine a machine offline by means of its TAM Configuration (with TAM System Explorer)
  - Number of drives, their type and other properties
  - Parametrization
  - Whole register interface
  - Bode tuning
- Simplified path planner
- Data acquisition
- ***Not* simulated**: hardware, controllers, most signals
- Only use for early prototyping of your application while no hardware is available

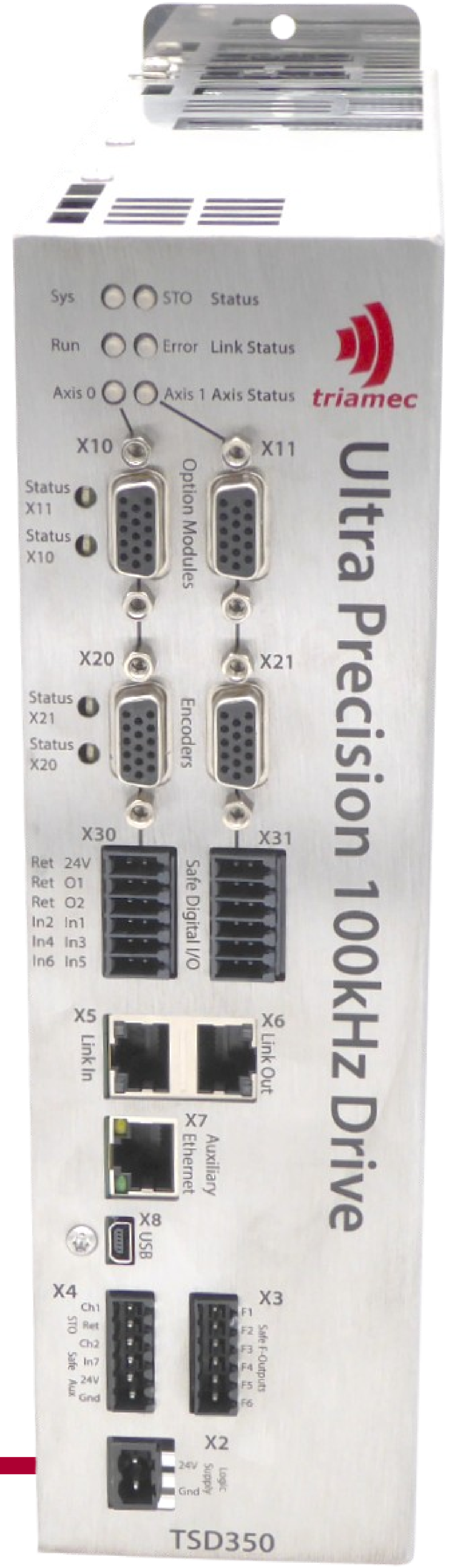
Achievements

- Find help & documentation for TAM API and TAM System Explorer
- Quickly access the drive from your app
- Parametrize drive
- Gain some insight in the C# project setup
- Observe and change drive state
- Know when a TAM Configuration file is needed

Controlling Triamec Drives through TAM API

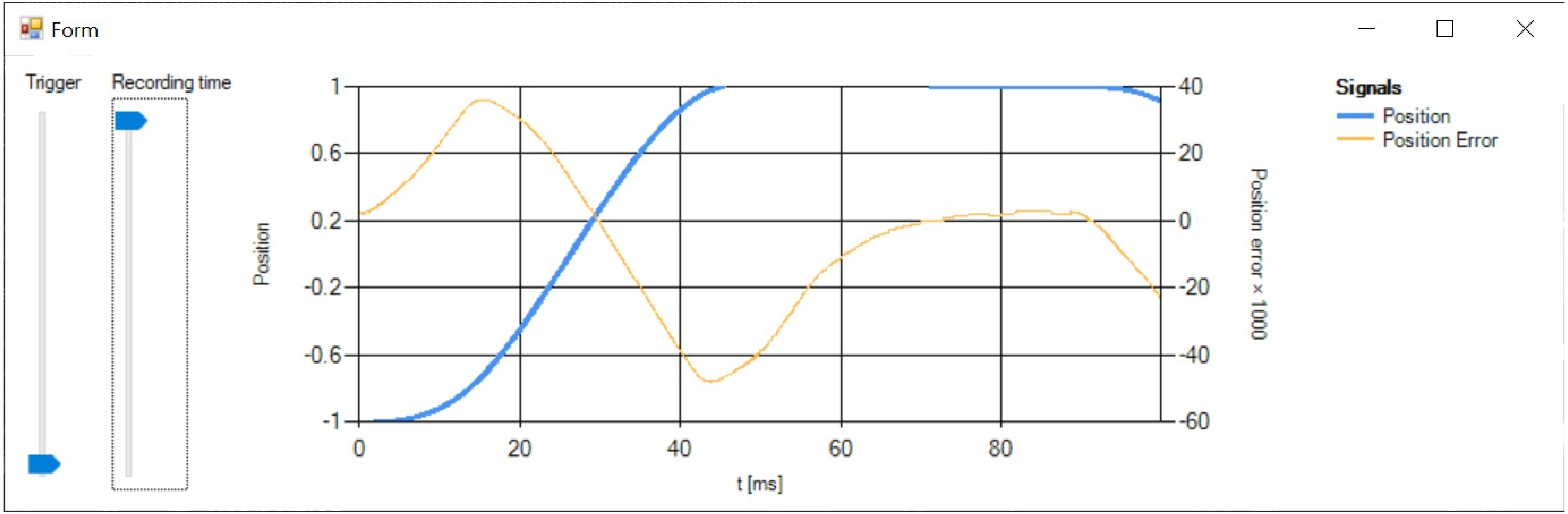
Initial Training – Part III

- *Control an axis*
  - Development Environment
  - Basic functions of the *TAM API*
- *Background*
  - Topology, registers, configuration and simulation
  - Commissioning tool: *TAM System Explorer*
- ***Advanced tasks***
  - Move sequence
  - Measurement



Acquisition Sample

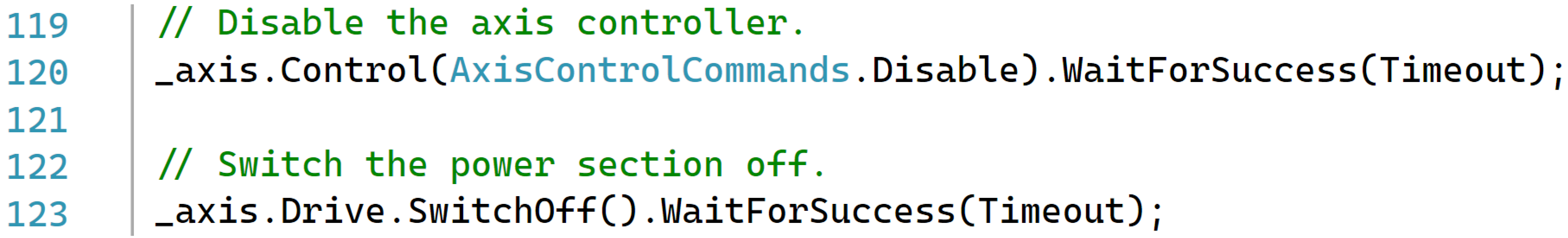
- `Acquisition.sln` demonstrates move sequence and measurement



  - Example screenshots in this presentation are taken from this sample code

Move Sequences

- Programming a sequence requires waiting for the move to end.
- Some API calls return a `TamRequest` instance for this sake, for example
  - `TamAxis.MoveAbsolute`
  - `TamAxis.Control`
- Chain `WaitForSuccess` or `WaitForSuccessAsync` to the command as follows:



```
// Disable the axis controller.
_axis.Control(AxisControlCommands.Disable).WaitForSuccess(Timeout);

// Switch the power section off.
_axis.Drive.SwitchOff().WaitForSuccess(Timeout);
```

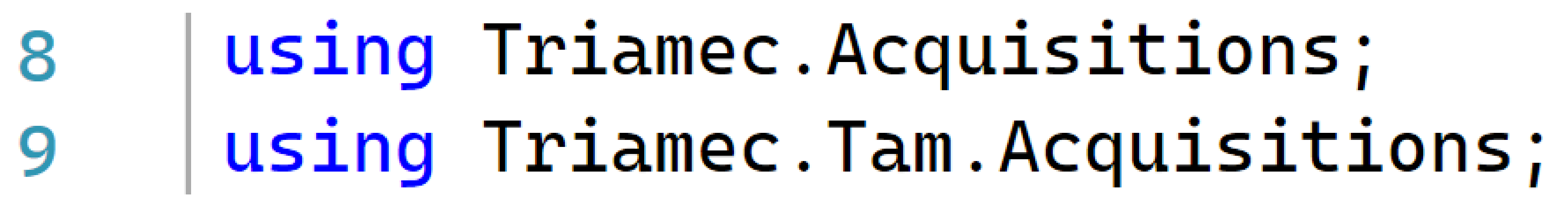
- Needs preparation: `ITamDevice.AddStateObserver` must be called at startup
- When calling `WaitForSuccess` on the result of `TamAxis.MoveVelocity`, another thread needs to reprogram the axis eventually, for example using `TamAxis.Stop`

TamRequest Features

- Wait… methods – see above
- CurrentState: Pending | Executing | Terminated
- Termination: None | Completed | Superseded | …
- Can fail due to
  - Timeout
  - The axis is in the wrong state
  - The axis was reprogrammed before the move ended
  - The firmware doesn’t support the command
  - Programming error
- Observe using
  - Termination event
  - Transition event on ITamDevice and TamAxis instances

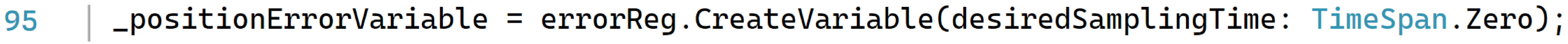
Acquire Real-Time data

- Import namespaces



```
using Triamec.Acquisitions;
using Triamec.Tam.Acquisitions;
```

- Create an object to hold data from any register



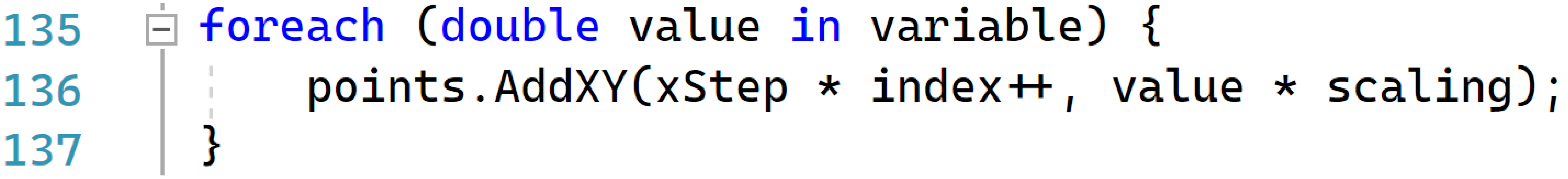
```
_positionErrorVariable = errorReg.CreateVariable(desiredSamplingTime: TimeSpan.Zero);
```

  - Sampling time zero means fastest possible (100 kHz)
  - Often 10 kHz – `TimeSpan.FromTicks(TimeSpan.TicksPerMillisecond / 10)` – suffices

- Acquire data

```
_positionErrorVariable.Acquire(TimeSpan.FromSeconds(1));
```

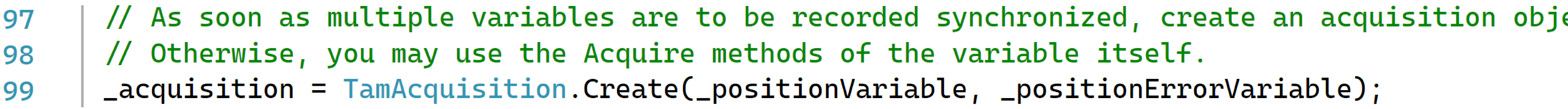
- Get data out of the buffer



```
foreach (double value in variable) {
    points.AddXY(xStep * index++, value * scaling);
}
```

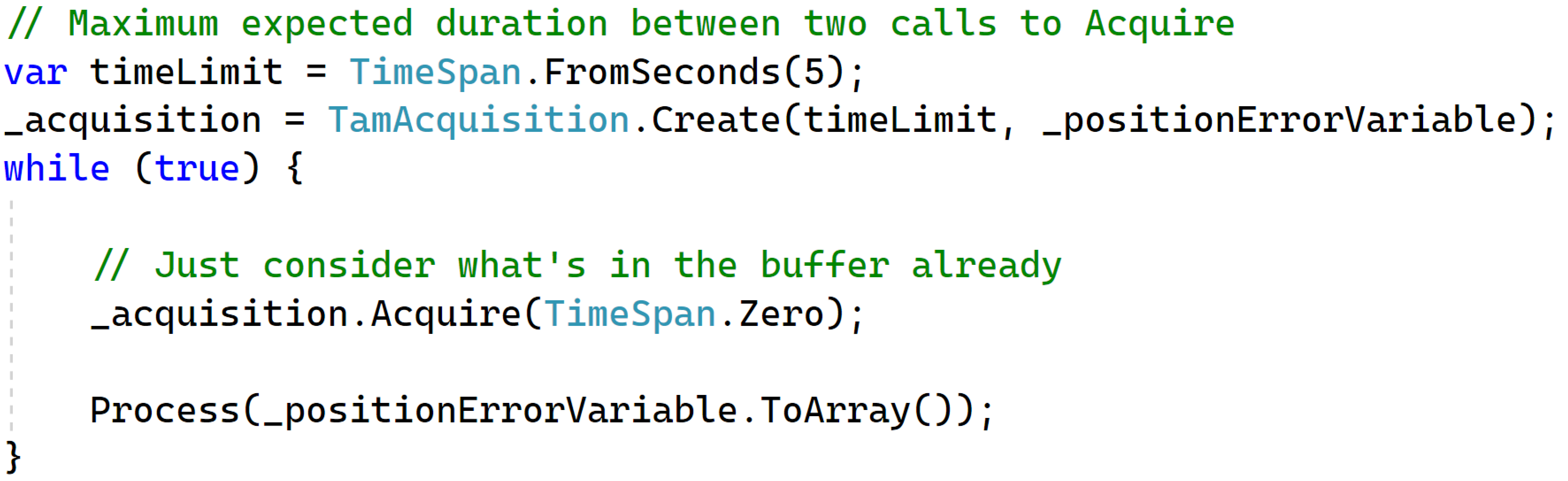
Synchronized Data and Continuous Measurement

- Create an acquisition object



```
// As soon as multiple variables are to be recorded synchronized, create an acquisition obje
// Otherwise, you may use the Acquire methods of the variable itself.
_acquisition = TamAcquisition.Create(_positionVariable, _positionErrorVariable);
```

- Sample data repeatedly, contiguous and without delay



```
// Maximum expected duration between two calls to Acquire
var timeLimit = TimeSpan.FromSeconds(5);
_acquisition = TamAcquisition.Create(timeLimit, _positionErrorVariable);
while (true) {

    // Just consider what's in the buffer already
    _acquisition.Acquire(TimeSpan.Zero);

    Process(_positionErrorVariable.ToArray());
}
```

Asynchronous Acquisition

- Use asynchronous Task workflow

```
await _acquisition.AcquireAsync(duration, null);

Fill(_chart.Series["Position"], _positionVariable, 1);

Fill(_chart.Series["Position Error"], _positionErrorVariable, 1E3);
```

Cleaning Up

- What to do before the application ends
  - Ensure the axis is in a safe place.
  - Disable controller `axis.Control(AxisControlCommands.Disable).WaitForSuccess(Timeout);`
  - Remove any state observer `drive.RemoveStateObserver(this);`
  - Dispose the root object `_topology.Dispose();`

Achievements

- Detect end of movement and other long-running tasks on the drive
- Obtain real-time data from the drive in the way that fits best in your application
- Leave system in a sane state