

BECKHOFF New Automation Technology

Manual | EN

TF3820

TwinCAT 3 | Machine Learning Server

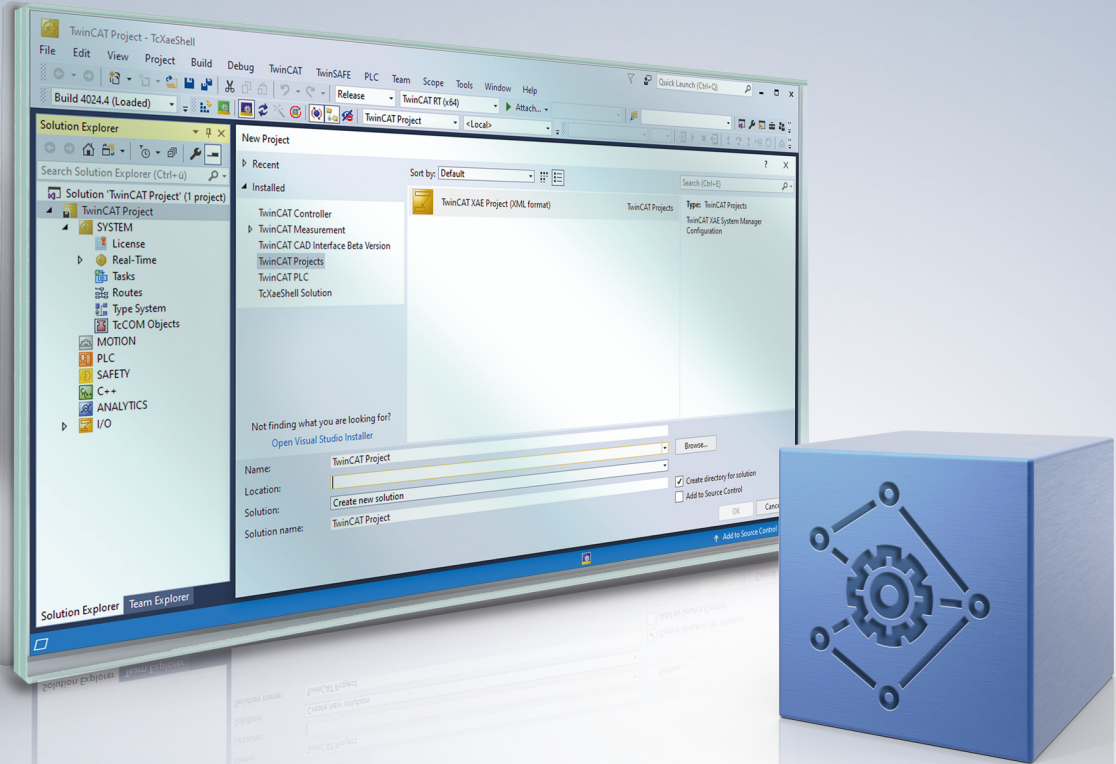


Table of contents

1 Foreword	5
1.1 Notes on the documentation	5
1.2 For your safety	5
1.3 Notes on information security.....	7
1.4 Documentation issue status	8
2 Overview	9
3 Installation	11
3.1 Licensing	12
3.2 Setting up an NVIDIA graphics card	14
4 Quickstart	16
5 Technical introduction	19
5.1 Workflow.....	19
5.1.1 Preparing ONNX for use with TwinCAT Machine Learning Server.....	19
5.1.2 Make model description files available on the Server Device	21
5.1.3 Configuring the server from the PLC client	22
5.1.4 Execute AI model.....	23
5.1.5 Updating the AI model.....	25
5.2 TwinCAT Machine Learning Model Manager.....	26
5.2.1 Graphical user interface.....	26
5.2.2 Python interface	28
5.2.3 Command Line Interface.....	29
5.3 ONNX Support	30
5.4 TcMIServer Service.....	31
5.4.1 Execution Provider.....	32
6 API	33
6.1 Function blocks	33
6.1.1 FB_MISvrPrediction	33
6.2 Data types	38
6.2.1 E_ExecutionProvider.....	38
6.2.2 ST_PredictionParameter.....	38
7 Samples	40
7.1 AI-based image processing.....	40
7.2 Custom attributes	42
8 Appendix	46
8.1 Error Codes.....	46
8.2 Log files.....	49
8.3 Third-party components	49
8.4 Support and Service.....	50

1 Foreword

1.1 Notes on the documentation

This description is intended exclusively for trained specialists in control and automation technology who are familiar with the applicable national standards.

For installation and commissioning of the components, it is absolutely necessary to observe the documentation and the following notes and explanations.

The qualified personnel is obliged to always use the currently valid documentation.

The responsible staff must ensure that the application or use of the products described satisfies all requirements for safety, including all the relevant laws, regulations, guidelines, and standards.

Disclaimer

The documentation has been prepared with care. The products described are, however, constantly under development.

We reserve the right to revise and change the documentation at any time and without notice.

No claims to modify products that have already been supplied may be made on the basis of the data, diagrams, and descriptions in this documentation.

Trademarks

Beckhoff®, TwinCAT®, TwinCAT/BSD®, TC/BSD®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® and XPlanar® are registered and licensed trademarks of Beckhoff Automation GmbH.

If third parties make use of designations or trademarks used in this publication for their own purposes, this could infringe upon the rights of the owners of the said designations.

Patents

The EtherCAT Technology is covered, including but not limited to the following patent applications and patents:

EP1590927, EP1789857, EP1456722, EP2137893, DE102015105702
and similar applications and registrations in several other countries.

EtherCAT®

EtherCAT® is registered trademark and patented technology, licensed by Beckhoff Automation GmbH, Germany

Copyright

© Beckhoff Automation GmbH & Co. KG, Germany.

The distribution and reproduction of this document as well as the use and communication of its contents without express authorization are prohibited.

Offenders will be held liable for the payment of damages. All rights reserved in the event that a patent, utility model, or design are registered.

1.2 For your safety

Safety regulations

Read the following explanations for your safety.

Always observe and follow product-specific safety instructions, which you may find at the appropriate places in this document.

Exclusion of liability

All the components are supplied in particular hardware and software configurations which are appropriate for the application. Modifications to hardware or software configurations other than those described in the documentation are not permitted, and nullify the liability of Beckhoff Automation GmbH & Co. KG.

Personnel qualification

This description is only intended for trained specialists in control, automation, and drive technology who are familiar with the applicable national standards.

Signal words

The signal words used in the documentation are classified below. In order to prevent injury and damage to persons and property, read and follow the safety and warning notices.

Personal injury warnings**⚠ DANGER**

Hazard with high risk of death or serious injury.

⚠ WARNING

Hazard with medium risk of death or serious injury.

⚠ CAUTION

There is a low-risk hazard that could result in medium or minor injury.

Warning of damage to property or environment**NOTICE**

The environment, equipment, or data may be damaged.

Information on handling the product

This information includes, for example:
recommendations for action, assistance or further information on the product.

1.3 Notes on information security

The products of Beckhoff Automation GmbH & Co. KG (Beckhoff), insofar as they can be accessed online, are equipped with security functions that support the secure operation of plants, systems, machines and networks. Despite the security functions, the creation, implementation and constant updating of a holistic security concept for the operation are necessary to protect the respective plant, system, machine and networks against cyber threats. The products sold by Beckhoff are only part of the overall security concept. The customer is responsible for preventing unauthorized access by third parties to its equipment, systems, machines and networks. The latter should be connected to the corporate network or the Internet only if appropriate protective measures have been set up.

In addition, the recommendations from Beckhoff regarding appropriate protective measures should be observed. Further information regarding information security and industrial security can be found in our <https://www.beckhoff.com/secguide>.

Beckhoff products and solutions undergo continuous further development. This also applies to security functions. In light of this continuous further development, Beckhoff expressly recommends that the products are kept up to date at all times and that updates are installed for the products once they have been made available. Using outdated or unsupported product versions can increase the risk of cyber threats.

To stay informed about information security for Beckhoff products, subscribe to the RSS feed at <https://www.beckhoff.com/secinfo>.

1.4 Documentation issue status

Version	Modifications
1.0.0	First release

2 Overview

Introduction

The TwinCAT Machine Learning Server enables the execution of AI models directly on the control IPC or on an Edge Device.

The TwinCAT Machine Learning Server consists of two components:

- The PLC function block as a client of the Machine Learning Server.
- The Machine Learning Server as a provider of services (loading, execution, ... of AI models).

These components provide asynchronous execution functionality for PLC programs. The concept of asynchronous calculation effectively decouples the AI model execution time from the cyclic operation of the PLC. The Machine Learning Server enables the execution of any sophisticated AI models on both CPUs and NVIDIA GPUs and is therefore particularly suitable for use with the C6043 Industrial PC. The Machine Learning Server is executed in the user mode of the operating system. This results in non-deterministic behavior that can only be partially mitigated by configuring the user mode components accordingly.

The TwinCAT Machine Learning Server loads AI models that are provided as ONNX files. All relevant AI frameworks, such as Tensor Flow, Pytorch, Scikit Learn, etc. support this interoperability standard. This decouples the training environment from the execution environment. Any training environment can be used to create AI models, which can then be executed with the TwinCAT Machine Learning Server.

Target groups and use cases

The Machine Learning Server is aimed at the following use cases, among others:

- Use of computationally intensive AI models where the expected reduction in computing time due to acceleration on a GPU overcompensates for the expected computing time fluctuations (jitter).
 - In particular, vision AI models for image classification, object recognition or segmentation should be mentioned here.
- Use of AI models in low-priority tasks that are only loosely coupled with the deterministic PLC program.
 - AI models whose results are not used by the control system, but are communicated to systems above the control level. For example, process analysis models in which the machine operator is informed, predictive maintenance models in which the service personnel are informed, etc.
 - AI models whose results are not required by the controller at a specific point in time. For example, AI models to provide optimized or adapted process parameters.

Differentiation and comparison with similar TwinCAT products

In addition to the TwinCAT Machine Learning Server, there are other TwinCAT products with similar functionality, i.e. the execution of AI models.

- TF3800 TwinCAT Machine Learning Inference Engine
- TF3810 TwinCAT Neural Network Inference Engine
- TF7800 TwinCAT Vision Machine Learning
- TF7810 TwinCAT Vision Neural Network

The main differences between the listed products and the TwinCAT Machine Learning Server are listed in the following table.

Table 1: Product properties in comparison

Deterministic AI: TF3800, TF3810, TF7810	Accelerated AI: TF3820
Deterministic AI execution in the TwinCAT process	Near-real-time execution in a separate process
Execution on standard x64 CPUs	Hardware acceleration on NVIDIA GPUs possible
Supports selected AI models and operators	Supports current ONNX Opset version and thus current and diverse AI models

Deterministic AI: TF3800, TF3810, TF7810	Accelerated AI: TF3820
Standard PLC function block for easy integration in TwinCAT	Standard PLC function block for easy integration in TwinCAT
Interoperability through ONNX support	Interoperability through ONNX support
License bundle: TF3810 includes TF3800, TF7800 and TF7810	Can also be used as a server in a network with several clients

3 Installation

To use the TwinCAT Machine Learning Server you need three components:

- **TF3820 | TwinCAT Machine Learning Server**
Install and license this workload either directly on your control IPC or on an Edge IPC.
 - Installs the [TcMIServer \[▶ 31\]](#) service on the system.
- **TF3830 | TwinCAT Machine Learning Server Client**
Install this workload on your engineering PC to use the required PLC library.
 - Installs the PLC library [Tc3_MIServer \[▶ 33\]](#) for the TwinCAT 3 XAE.
- **TF38xx | TwinCAT Machine Learning Model Manager**
Install this workload on your engineering PC to [prepare the ONNX file for use in TwinCAT \[▶ 19\]](#).

System requirements: TwinCAT Machine Learning Server (TF3820)

Technical data	Requirements
Operating system	Windows10
Target platform	x64
Minimum TwinCAT version	TwinCAT 3.1 Build 4026
Required TwinCAT setup level	TwinCAT 3 XAR
Required TwinCAT license	TC1000


System requirements: TwinCAT Machine Learning Server Client (TF3830)

Technical data	Requirements
Operating system	Windows10
Target platform	x64
Minimum TwinCAT version	TwinCAT 3.1 Build 4026
Required TwinCAT setup level	TwinCAT 3 XAE
Required TwinCAT license	TC1200

TwinCAT Package Manager: Installation (TwinCAT 3.1 Build 4026)

Detailed instructions on installing products can be found in the chapter [Installing workloads](#) in the [TwinCAT 3.1 Build 4026 installation instructions](#).

Install the following workload to be able to use the product:



TF3820 | TwinCAT Machine Learning Server
(Runtime: 3.2.1)

The TF3820 TwinCAT 3 Machine Learning Server is a high-performance service for executing trained AI models with the option of using hardware accelerators.

✓

TwinCAT Package Manager UI: TF3820 | TwinCAT 3 Machine Learning Server

TwinCAT Package Manager CLI:

```
tcpkg install TF3820.MachineLearningServer.XAR
```



TF3830 | TwinCAT Machine Learning Server Client (Engineering: 3.2.1)

TF3830 TwinCAT 3 Machine Learning Server Client provides PLC function blocks that can be used to configure and call up services supported by the TwinCAT 3 Machine Learning Server.

TwinCAT Package Manager UI: TF3830 | TwinCAT 3 Machine Learning Server Client

TwinCAT Package Manager CLI:

```
tcpkg install TF3830.MachineLearningServerClient.XAE
```



TF38xx | TwinCAT Machine Learning Model Manager (Engineering: 1.0.0)

Management tool for converting and generating AI model information

TwinCAT Package Manager UI: TF38xx | TwinCAT 3 Machine Learning Model Manager

TwinCAT Package Manager CLI:

```
tcpkg install TF38xx.MachineLearningModelManager.XAE
```

Installation on systems with TF3800 or TF3810 version 3.2.6 and later

i Potential conflict with old installations

There may be conflicts regarding the packages for the TwinCAT Machine Learning Model Manager. This conflict is quickly resolved by uninstalling.

If you have installed a version of the workload TF3800.MachineLearningInferenceEngine.XAE or TF3810.NeuralNetworkInferenceEngine.XAE in version 3.2.6 or lower, you should first uninstall these workloads:

```
tcpkg uninstall TF3800.MachineLearningInferenceEngine.XAE --include-dependencies
```

```
tcpkg uninstall TF3810.NeuralNetworkInferenceEngine.XAE --include-dependencies
```

Only then should you install the workloads in version 3.2.10 or higher.

3.1 Licensing

The **TF3820** TwinCAT Machine Learning Server license is requested by the server component (TcMLServer service). The license must be stored on the IPC on which the server process is executed. The TF3820 license includes the TF3830 license, so that local communication between the function block and server is possible without additional licenses.

The **TF3830** TwinCAT Machine Learning Server Client license is queried by the function block of the PLC library Tc3_MIServer. The license is required on runtime systems that access the TwinCAT Machine Learning Server remotely. No separate TF3830 license is required for local communication between function block and server.

The TwinCAT 3 function can be activated as a full version or as a 7-day test version. Both license types can be activated via the TwinCAT 3 development environment (XAE).

i Cyclic license query of the TwinCAT Machine Learning Server

If no or only a 7-day test license is available for the TwinCAT Machine Learning Server, a license query is performed cyclically every 10 seconds.

Licensing the full version of a TwinCAT 3 Function

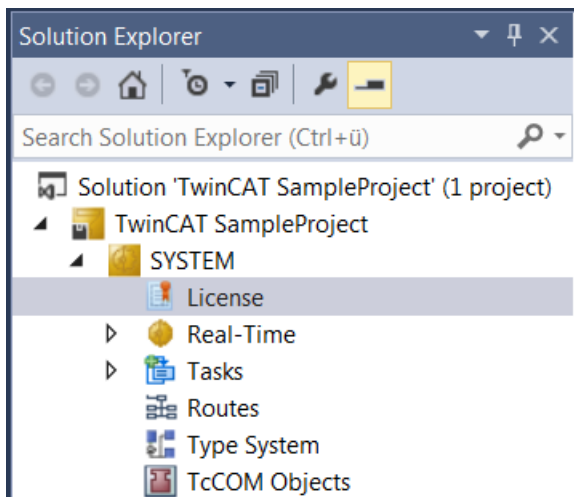
A description of the procedure to license a full version can be found in the Beckhoff Information System in the documentation "[TwinCAT 3 Licensing](#)".

Licensing the 7-day test version of a TwinCAT 3 Function



A 7-day test version cannot be enabled for a [TwinCAT 3 license dongle](#).

1. Start the TwinCAT 3 development environment (XAE).
2. Open an existing TwinCAT 3 project or create a new project.
3. If you want to activate the license for a remote device, set the desired target system. To do this, select the target system from the **Choose Target System** drop-down list in the toolbar.
 - ⇒ The licensing settings always refer to the selected target system. When the project is activated on the target system, the corresponding TwinCAT 3 licenses are automatically copied to this system.
4. In the **Solution Explorer**, double-click **License** in the **SYSTEM** subtree.



⇒ The TwinCAT 3 license manager opens.

5. Open the **Manage Licenses** tab.
6. In the **Add License** column, activate the check box for the license you want to add to your project (in this case "TF3820 TC3 Machine Learning Server" and/or "TF3830 TC3 Machine Learning Server Client"). **Please note that TF3820 includes the license TF3830 for the local system. If a TF3820 is on the local system, you do not need the TF3830 for the client. TF3830 is only required on remote clients.**

Order No	License	Add License
TF3685	TC3 Weighing	<input type="checkbox"/> cpu license
TF3710	TC3 Interface for LabVIEW	<input type="checkbox"/> cpu license
TF3800	TC3 Machine Learning Inference Engine	<input type="checkbox"/> cpu license
TF3810	TC3 Neural Network Inference Engine	<input type="checkbox"/> cpu license
TF3820	TC3 Machine Learning Server	<input checked="" type="checkbox"/> cpu license
TF3830	TC3 Machine Learning Server Client	<input checked="" type="checkbox"/> cpu license
TF3850	TC3 Machine Learning Creator	<input type="checkbox"/> cpu license

7. Open the **Order Information (Runtime)** tab.
 - ⇒ In the tabular overview of licenses, the previously selected license is displayed with the status "missing".

8. Click **7 Days Trial License...** to activate the 7 days trial license.

The screenshot shows the 'License Management' window with several sections:

- Order Information (Runtime)**: Includes tabs for 'Manage Licenses', 'Project Licenses', and 'Online Licenses'. Below are fields for 'License Device' (set to 'Target (Hardware Id)'), 'System Id' (2DB25408-B4CD-81DF-5488-6A3D9B49EF19), and 'Platform' (other (91)).
- License Request**: Includes a 'Provider' dropdown (Beckhoff Automation), 'License Id', 'Customer Id', and a 'Comment' field. A 'Generate File...' button is present.
- License Activation**: This section is highlighted with a red box and contains two buttons: '7 Days Trial License...' and 'License Response File...'.

⇒ A dialog box opens, prompting you to enter the security code displayed in the dialog.

The 'Enter Security Code' dialog box contains the following elements:

- Title: Enter Security Code
- Instruction: Please type the following 5 characters:
- Text box: Contains the code 'Kg8T4'.
- Input field: A two-character input field with a red border, currently empty.
- Buttons: 'OK' (highlighted with a red box) and 'Cancel'.

9. Enter the code exactly as it is displayed and confirm the entry.

10. Confirm the subsequent dialog, which indicates the successful activation.

⇒ In the tabular overview of licenses, the license status now indicates the expiry date of the license.

11. Restart the TwinCAT system.

⇒ The 7-day trial version is enabled.

3.2 Setting up an NVIDIA graphics card

If **no** Beckhoff hardware with the associated Beckhoff image is used for the GPU-accelerated AI model execution, the customer is independently responsible for creating the necessary framework conditions on his system for operating the graphics card.



If you are using a Beckhoff IPC with GPU and associated Beckhoff image, you can skip this section.

System requirements

System requirements are specific to NVIDIA components. The software versions listed below should not be newer than the specified versions (e.g. do not install CUDA 12.6), otherwise incompatibilities may occur.

NVIDIA drivers

- Normal NVIDIA GPU: Version 560.67
- Quadro/RTX GPU: Version 522.86

CUDA (Compute Unified Device Architecture)

- CUDA Version 12.5.1

cuDNN

- cuDNN Version 9.2.1.18 (as zip file, do not use as Windows Installer)
- Installation instructions can be found at NVIDIA: [Tarball Installation](#).
- It is necessary to add the path to the cuDNN libraries to the PATH variable of the system, not that of the user.

In general, the above measures meet the following system requirements:

- Availability of the cuda.dll library
- Availability of the nvml.dll library

Optimized runtime behavior

When using Beckhoff hardware, e.g. C6043 Industrial PC with GPU and associated Beckhoff image, the runtime behavior of the NVIDIA GPU is optimally designed for interaction with the TwinCAT real-time. When using third-party GPUs, considerable runtime fluctuations may occur during model execution, depending on the specific GPU and its settings.

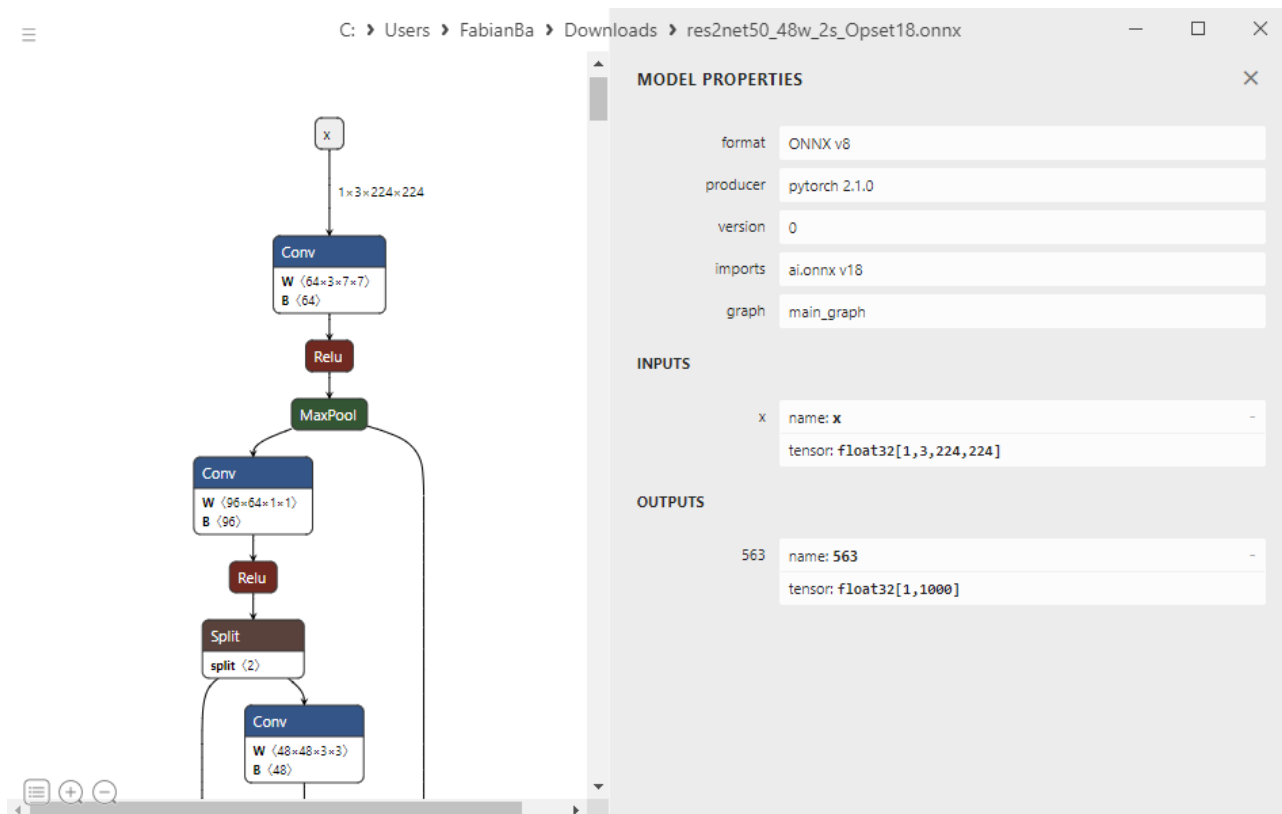
The TwinCAT Machine Learning Server informs about possible challenges of the used graphics card in the [log file \[▶ 49\]](#) immediately after starting the server.

4 Quickstart

Create or download ONNX file

If you do not have your own ONNX to hand for an initial test, you can use the [ONNX Model Zoo](#) on GitHub for tests, for example. The [ResNet50](#) from the ONNX Model Zoo is used as an example in the following.

Netron can be used to easily inspect whether the [requirements \[► 30\]](#) for execution with the TwinCAT Machine Learning Server are met.



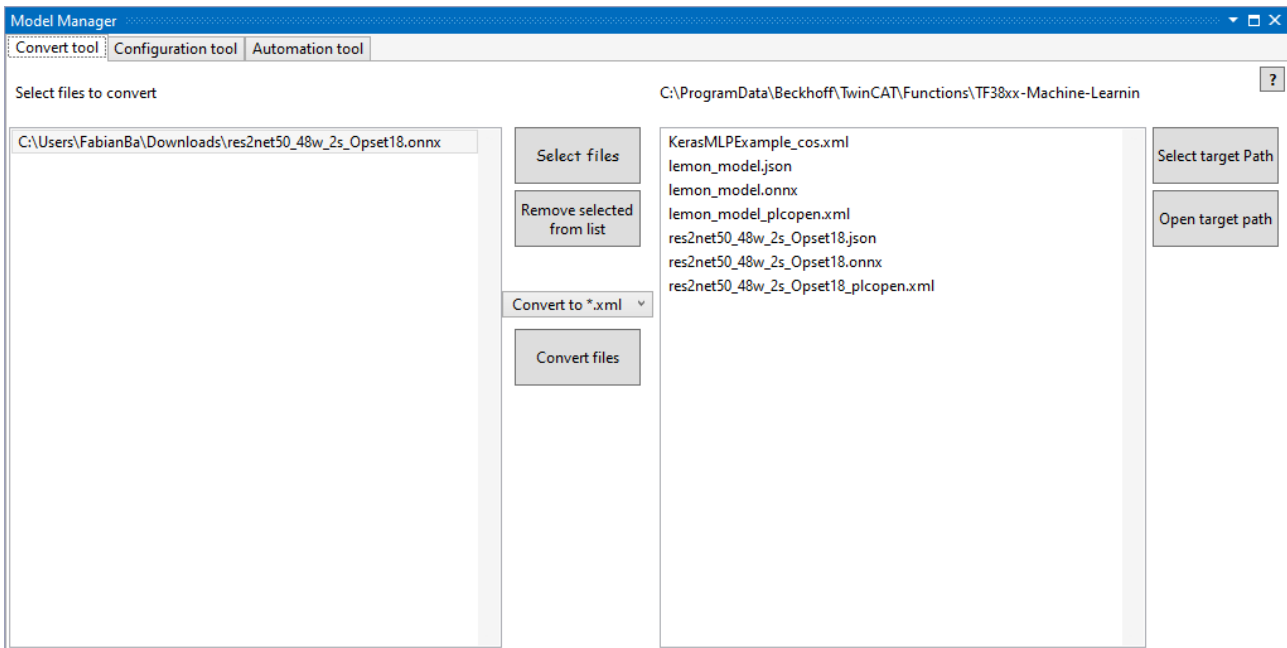
The Input Nodes are not dynamic and the ONNX Opset used is also supported.

Preparing ONNX file with TwinCAT Machine Learning Model Manager

Open TwinCAT XAE and navigate to TwinCAT > Machine Learning > [Machine Learning Model Manager \[► 26\]](#).

Load the downloaded ONNX with "Select files" and then select "Convert files". The ONNX and the associated JSON and PlcOpenXml created are now displayed in the target path.

Select "Open target path" to open the File Explorer on this path.



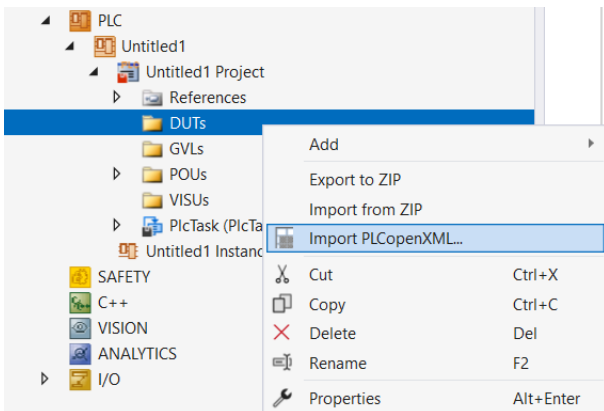
Making files available on the target system

In this Quickstart, it is assumed that the TwinCAT Machine Learning Server is operated on the same device as the PLC. Accordingly, the model files (res2net50_48w_2s_Opset18.onnx and res2net50_48w_2s_Opset18.json) are stored on the target device under the path C:\models.

Further information on this step can be found here: [Make model description files available on the Server Device \[▶ 21\]](#).

Writing source code

It starts with an empty PLC project. First import the created res2net50_48w_2s_Opset18_plcopen.xml by right-clicking on the DUTs folder and selecting "Import PLCopenXML".



Also add the PLC library Tc3_MIServer under References.

In the minimal sample, the code consists of two steps. First, a session is created on the TwinCAT Machine Learning Server and then the inference of the loaded model is executed.

Declaration

```
stModelInput : ST_res2net50_48w_2s_Opset18Input;
stModelOutput : ST_res2net50_48w_2s_Opset18Output;

fbMlSvr      : FB_MlSvrPrediction;
bConfigured : BOOL := FALSE;
bError      : BOOL := FALSE;

sSuccess      : T_MaxString;
nInferenceCount : UDINT := 0;
```

Code

```

IF NOT bConfigured AND NOT bError THEN

    fbMlSvr.stPredictionParameter.sMlModelFilePath := 'C:\models\res2net50_48w_2s_Opset18.json';
    fbMlSvr.stPredictionParameter.sMlSvrNetId := '127.0.0.1.1.1';
    fbMlSvr.stPredictionParameter.eExecutionProvider := E_ExecutionProvider.CPU;

    IF fbMlSvr.Configure(nTimeout := 10000, nPriority:=0) THEN
        IF fbMlSvr.nErrorCode <> 0 THEN
            bError := TRUE;
        ELSE
            bConfigured := TRUE;
        END_IF
    END_IF
END_IF

IF bConfigured AND NOT bError THEN
    IF fbMlSvr.Predict(
        pDataIn      := ADR(stModelInput),
        nDataInSize  := SIZEOF(stModelInput),
        pDataOut     := ADR(stModelOutput),
        nDataOutSize := SIZEOF(stModelOutput),
        nTimeout     := 1000,
        nPriority     := 0)
    THEN
        IF fbMlSvr.nErrorCode <> 0 THEN
            bError := TRUE;
        ELSE
            sSuccess := 'You made your first inference';
            nInferenceCount := nInferenceCount + 1;
            // use stModelOutput here
        END_IF
    END_IF
END_IF

```

Activating the configuration

Activate your configuration and start the PLC. The result is shown below. The counter value `nInferenceCount` increases and the variable `sSuccess` displays a success message.

The screenshot displays the TwinCAT software interface. The main window shows the PLC configuration for the project 'TwinCAT Project80'. The 'MAIN (PROG)' window displays the ladder logic code, which is the same code as shown in the previous block. The 'Error List' window at the bottom shows 0 errors and 0 warnings. The 'Watch' window at the bottom right shows the current values of the variables:

Expression	Type	Value	Prepar...	Address	Comm...
stModelInput	ST_res2net50_48w_2s...				
stModelOutput	ST_res2net50_48w_2s...				
fbMlSvr	FB_MlSvrPrediction				
bConfigured	BOOL	TRUE			
bError	BOOL	FALSE			
sSuccess	T_MaxString	'You made your first inference'			
nInferenceCount	UDINT	839			

5 Technical introduction

5.1 Workflow

The workflow consists of the following steps:

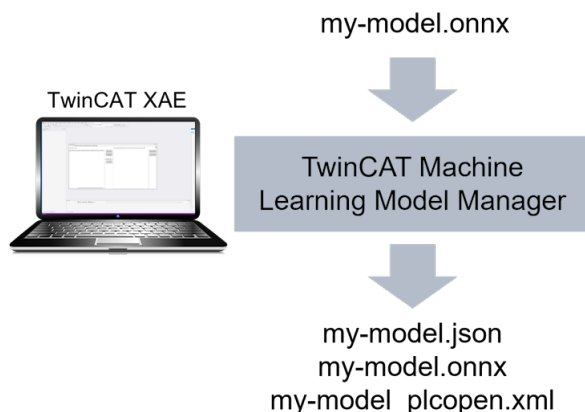
- Extract [PLC interface description \[▶ 19\]](#) from the ONNX file (with [the TwinCAT Machine Learning Model Manager \[▶ 26\]](#)).
 - A PlcOpenXml is created from the ONNX file, which provides the input and output data type of the model for the PLC.
 - A JSON file is generated which contains metadata about the model. Metadata is TwinCAT-specific on the one hand and application-specific from the user on the other.
- [Provide JSON file and ONNX file on the target system \[▶ 21\]](#).
- [Import PlcOpenXml in TwinCAT Engineering \[▶ 19\]](#) and incorporate it into the PLC program using [FB_MlSvrPrediction \[▶ 33\]](#).
- [Configure the TwinCAT Machine Learning Server from the PLC \[▶ 22\]](#) and load the AI model.
- [Call the loaded AI model asynchronously to the PLC task cycle \[▶ 23\]](#).
- [Exchange/update an AI model at runtime of the machine \[▶ 25\]](#).

5.1.1 Preparing ONNX for use with TwinCAT Machine Learning Server

Generate interface descriptions for the PLC

To be able to use an ONNX with the [FB_MlSvrPrediction \[▶ 33\]](#) in the TwinCAT PLC, interface information is required. These are generated by the [TwinCAT Machine Learning Model Manager \[▶ 26\]](#).

Information on the supported ONNX Opset and restrictions can be found here: [ONNX Support \[▶ 30\]](#).



JSON file

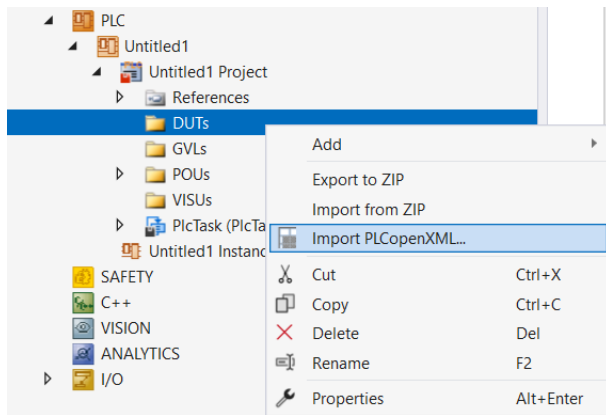
The JSON file contains metadata required by the function block `FB_MlSvrPrediction`. Users can also add their own metadata to the JSON file via the [Custom Attributes \[▶ 42\]](#). The JSON file is loaded by the function block, see [Configure method \[▶ 34\]](#). Both the JSON file and the ONNX file must be available for loading on the runtime PC, see [Make model description files available on the Server Device \[▶ 21\]](#).

The byte array of the header must not be changed!

The names of the elements of the STRUCT can be adjusted if required, but not their order in the STRUCT.

Import of a PlcOpenXml in TwinCAT 3

The generated PlcOpenXml can be transferred to the PLC project in the PLC by right-clicking on a folder (e.g. DUTs) via the "Import PLCOpenXML" field.



NOTICE

Verified signature: Operation of the Machine Learning Server only via generated input and output model types

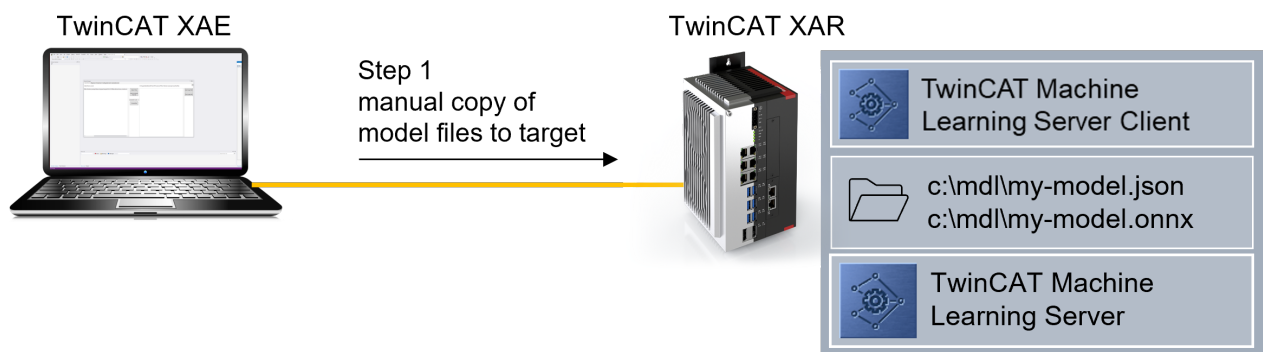
Note that the use of the input/output model types defined in the supplied PlcOpen file is mandatory. The types have a signature verified by the TcMIServer to ensure secure inference operations.

5.1.2 Make model description files available on the Server Device

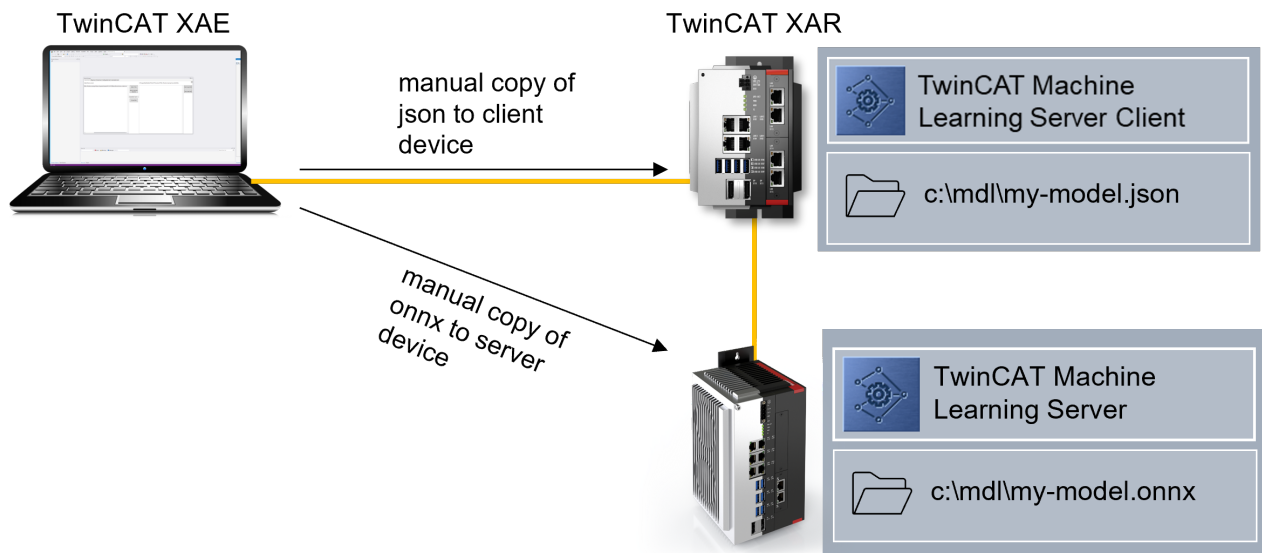
In order to be able to load models, they must be made known to the TwinCAT Machine Learning Server. This means that the server must know the location of the created JSON and ONNX file on the file system in order to successfully load the AI model. This announcement is made using the [Configure \[▶ 34\]](#) method. The method is passed the full path to the JSON file, which contains the interface description. The associated ONNX file must be stored in the same folder path. A hash is always used to validate the connection between JSON and ONNX.

The ONNX file is required on the device on which the TwinCAT Machine Learning Server is installed. The JSON file, in turn, is required on the client device. The storage path itself is arbitrary; but JSON and ONNX must have the same path.

Variante 1 - Client and server are installed on the same IPC: The JSON and ONNX file must be stored on the IPC in any path.



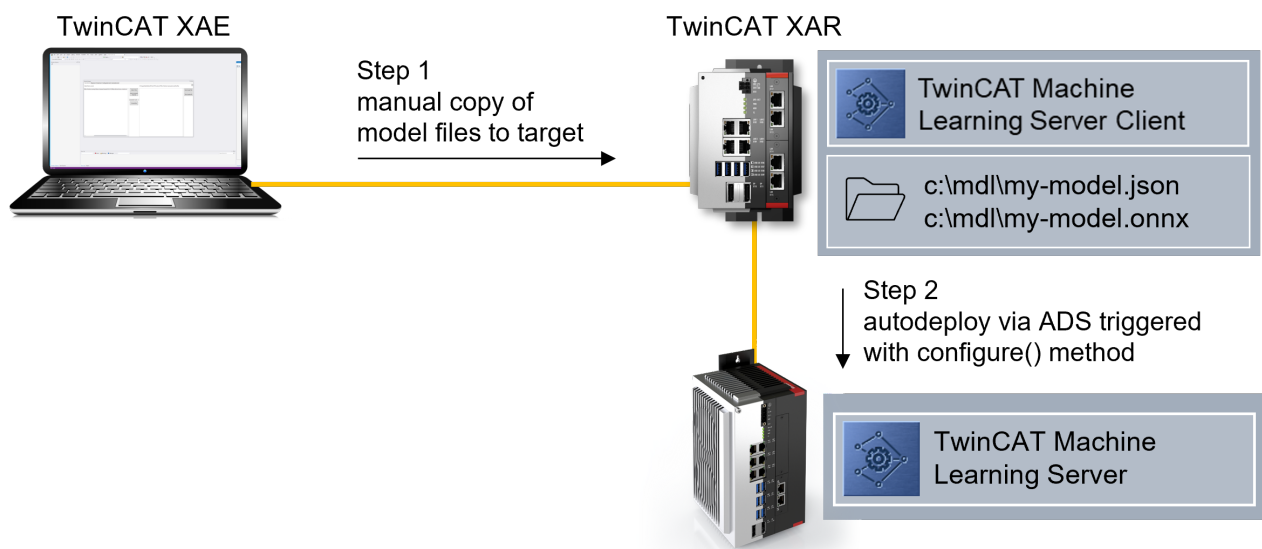
Variante 2 - Client and server are installed on different IPCs: The ONNX file can be stored directly on the server. The JSON file must then be stored on the client device under the same path.



Variante 3 - Client and server are installed on different IPCs: The model files (ONNX and JSON) can alternatively both be stored on the client. When the Configure method is called, the path on the Server Device is checked first. If the ONNX file is not found there, the path on the Client Device is checked. If the JSON and ONNX are found on the Client Device, the ONNX file is transferred to the server device via ADS and then loaded from the server. This variante is time-consuming and is only carried out once for this model. The model is then loaded directly from the Server Device.



The ADS router memory must be large enough to be able to send the model via ADS.



5.1.3 Configuring the server from the PLC client

Calling the method `configure` [▶ 34] instantiates a session for the respective instance of the function block `FB_MlSvrPrediction` in the `TcMIServer`. The configuration defined in the FB member `stPredictionParameter` [▶ 38] is used for instantiation. As a rule, each instance of `FB_MlSvrPrediction` is assigned its own session on the server. However, sessions can also be used together via the parameter `bExclusiveSession`.

During the configuration call, the following is defined in particular:

- Where is the TwinCAT Machine Learning Server?
 - Is specified via the AMS Net Id of the Server Device. The default value is "local".

- Which AI model should be loaded?
 - Is specified via the path to the corresponding JSON file.
- On which hardware should the AI model be executed?
 - Is defined via the Execution Provider ([E_ExecutionProvider](#) [▶ 38](#)) and optionally the DeviceId of the GPU.

Each session that is opened allocates resources on the Server Device. The number of parallel sessions is not limited on the software side, but is restricted solely by the available hardware resources. If there is not enough memory (RAM or vRAM) available to open another session, the Configure command will fail.

The [deconfigure](#) [▶ 35](#) method can be used to close a session and thus release the resources.

If a client does not send a request to the server for a defined period of time, the so-called session time-out period, the server assumes that the client is no longer active. The session is automatically closed when the configured [session timeout](#) [▶ 38](#) is reached.

Sample

Declaration

```
fbMlSvr : FB_MlSvrPrediction();
```

Code

```
// configure session parameters
fbMlSvr.stPredictionParameter.sMlModelFilePath := 'C:\mdl\lemon_model.json';
fbMlSvr.stPredictionParameter.sMlSvrNetId := '127.0.0.1.1.1';
fbMlSvr.stPredictionParameter.eExecutionProvider := E_ExecutionProvider.CPU;

// Submit configuration request to the TcMlServer
// Provide a generous nTimeout, as the configuration can take a substantial amount of time
IF fbMlSvr.Configure(nTimeout := 1000, nPriority:=0) THEN
// check for error
// change state
END_IF;
```

5.1.4 Execute AI model

The execution of an AI model with the [FB_MlSvrPrediction](#) [▶ 33](#) is triggered via the [Predict](#) [▶ 35](#) method (or [PredictBatched](#) [▶ 36](#) in the case of a *batched* call), which is asynchronous to the PLC task cycle.

The input data type and the output data type (see [PlcOpenXml](#) in the [Machine Learning Model Manager](#) [▶ 26](#) section), as well as a timeout and a priority are passed to the method.

Send inference order:

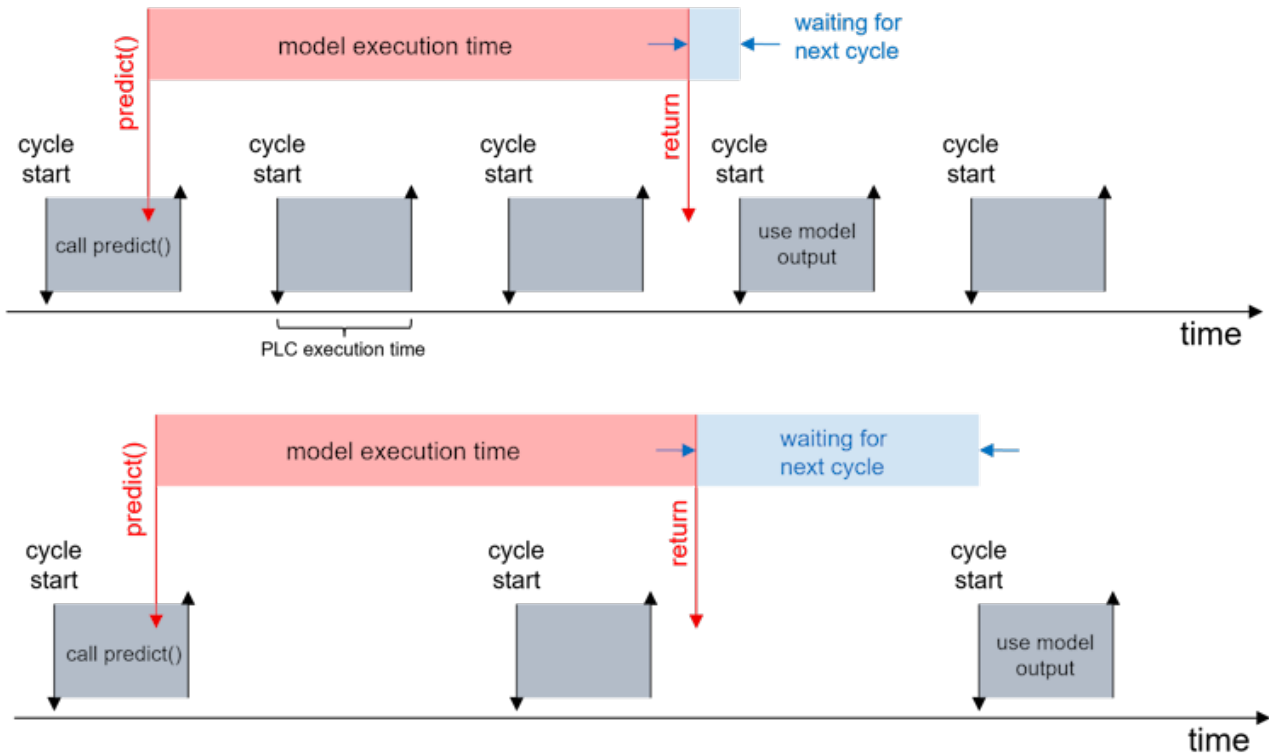
When the method is called, the input data area is sent to the server via ADS. The copying process required for this is carried out synchronously in the task cycle. Please note that larger amounts of data require more time. When transferring large amounts of data, such as large image data, the PLC task cycle time must be configured to prevent cycle timeouts.

Edit inference order:

The inference order is then accepted by the TwinCAT Machine Learning Server. If there are multiple requests that cannot be processed at the same time, a queue is created, with higher priorities moving up in the queue. CPU-based inference orders are always processed sequentially, i.e. the queue is particularly relevant here. GPU-based inferences, on the other hand, can be processed in parallel. Each FB instance can only ever have one outstanding request on the server, i.e. the maximum number of requests on the server is the number of active clients.

Request inference result:

It is important to note that the processing of the asynchronous request can only be monitored with the temporal granularity of the PLC task cycle time. It is therefore important that applications with a low delay budget have the lowest possible cycle times and correspondingly high sampling rates in order to minimize delays caused by time resolution. This is particularly important when interacting with other, computationally intensive and synchronously executed algorithms, e.g. when pre- or post-processing data.



Sample Predict()

Declaration

```
stModelInput : ST_LemonModelInput; // DUT from PlcOpenXml produced with TC ML Model Manager
stModelOutput : ST_LemonModelOutput; // DUT from PlcOpenXml produced with TC ML Model Manager
fbMlSvr : FB_MlSvrPrediction();
```

Code:

```
// Submission of an inference request at the TcMlServer
// and subsequent postprocessing of the inference result
// Submission of the asynchronous inference request to the TcMlServer
IF fbMlSvr.Predict (
    pDataIn      := ADR(stModelInput),
    nDataInSize  := SIZEOF(ST_LemonModelInput),
    pDataOut     := ADR(stModelOutput),
    nDataOutSize := SIZEOF(ST_LemonModelOutput),
    nTimeout     := 100,
    nPriority     := 0) THEN
    IF fbMlSvr.nErrorCode <> 0 AND NOT fbMlSvr.bConfigured THEN
        // If nErrorCode -1 is encountered, increase nTimeout
        eState := E_State.eError;
    ELSE
        // Postprocessing of the inference results
    END_IF
END_IF
```

Sample PredictBatched()

Declaration

```
stModelInputBatch : ARRAY[0..3] OF ST_LemonModelInput;
stModelOutputBatch : ARRAY[0..3] OF ST_LemonModelOutput;
```

Code


```
IF fbMlSvr.PredictBatched(  
    pDataIn      := ADR(stModelInputBatch),  
    nDataInSize  := SIZEOF(ST_LemonModelInput),  
    nBatchSize   := 4,  
    pDataOut     := ADR(stModelOutputBatch),  
    nDataOutSize := SIZEOF(ST_LemonModelOutput),  
    nTimeout     := 100,  
    nPriority     := 0) THEN
```

In the sample calls, the timeout for the inference request is set to 100 cycles in each case. In the figure above, the result is available after 3 cycles (top) or 2 cycles (bottom). The jitter that may be experienced can be queried via `fbMlSvr.nMaxInferenceDuration`. This shows the maximum number of PLC cycles that were required to execute an inference. Based on this value, the timeout value can usually be interpreted well.

5.1.5 Updating the AI model

AI models can be exchanged at runtime. The following describes two cases and the steps to follow.

Case 1: Model update without changing the model interface

Definition of the case:

In this case, the input and output interface to the AI model remains identical when the model is replaced. To do this, the input and output nodes must remain unchanged in their sequence (if there are several nodes) and in their shape.

It is recommended not to change the entire model architecture when updating the model, i.e. to carry out transfer training/fine-tuning on the existing AI model. As a result, the interfaces to the model do not change and the runtime behavior remains the same.

A model update can be carried out without a compile process and without a TwinCAT stop.

Steps to update the model:

- Create JSON and PlcOpenXml with the TwinCAT Machine Learning Model Manager.
- Make JSON and ONNX available on the relevant systems. If the full path has changed, make the new path known in a variable via ADS, for example.
- The hash of the input and output data types does not change. This means that **no** new PlcOpenXml needs to be read. The new PlcOpenXml remains unused.
- Change the state in the running PLC, e.g. set a corresponding variable via ADS, and call `Deconfigure [▶ 35]()` to close the current session.
- Call Configure to load the new JSON.

During the time from Deconfigure to the completion of the Configure method, no inference calls can be sent to the server with this function block.

Case 2: Model update with model interface change

Definition of the case:

In this case, the input and output interface to the AI model changes when the model is swapped. As a result, the input and output data types in the PLC no longer match the model. As a rule, several places in the source code have to be changed.

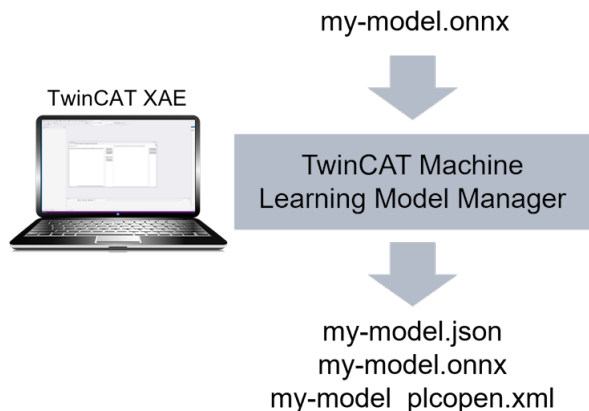
It is recommended to revise the source code in TwinCAT XAE, retest the project and only then load it onto the machine. Please note that the runtime behavior of the AI model may also have changed.

In this case, a model update is connected with a TwinCAT stop when the new, modified TwinCAT project is loaded.

5.2 TwinCAT Machine Learning Model Manager

The TwinCAT Machine Learning Model Manager is used to manage the ONNX model files and prepare them for use with TwinCAT. The functions of the TwinCAT Machine Learning Model Manager are summarized:

- Creation of a metadata file (JSON file).
- Creation of a PlcOpenXml, which describes a PLC data type description of the model input and output.
- Optional: Inserting application-specific metadata (Custom attributes). A user can add the Custom attributes to the JSON. The attributes can then be read out in the PLC at runtime using methods at [FB_MISvrPrediction \[▶ 33\]](#).



The TwinCAT Machine Learning Model Manager has three different interfaces:

1. A graphical user interface (Visual Studio plugin)
2. A Python interface (Python package)
3. A **Command Line Interface (CLI)**

These are described in more detail below.

5.2.1 Graphical user interface

The TwinCAT 3 Machine Learning Model Manager is the central UI for editing ONNX files. The tool is integrated in Visual Studio and can be opened via the menu bar under **TwinCAT > Machine Learning > Machine Learning Model Manager**.

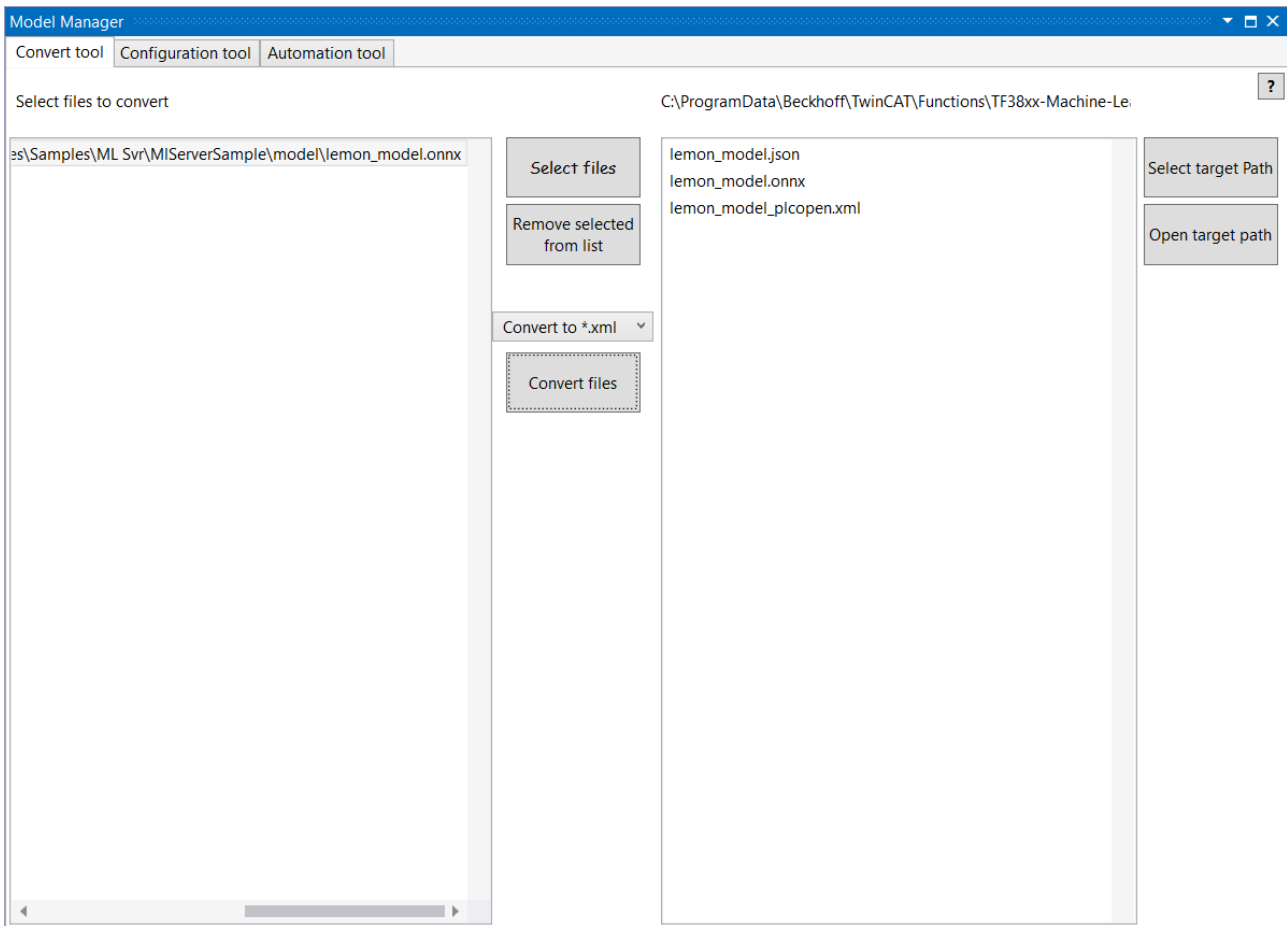
● Required Visual Studio version



The graphic interface of the TwinCAT 3 Machine Learning Model Manager is compatible with Visual Studio 2017, 2019, 2022 and the TcXaeShell.

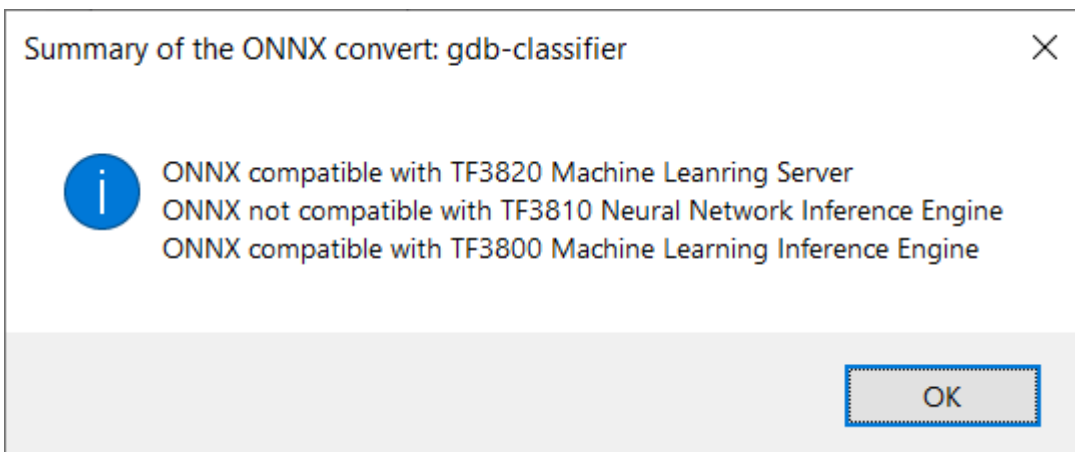
Creating the JSON and PlcOpenXml

1. Open the **Convert Tool** tab.
 - ⇒ Click **Select files** to open the file browser.
2. Select ONNX files (multi-select possible by Ctrl-clicking).
 - ⇒ Selected ONNX files are listed on the left-hand side with their path and file name.
3. If required, you can select files and remove them from the list again by clicking the **Remove selected from list** button.
4. Click on **Convert files** to create the required JSON and PlcOpenXml.
 - ⇒ The files are stored in the converted file path. The default path is <TwinCATPath>\Functions\TF38xx-Machine-Learning\ConvertToolFiles.
5. Click on **Open target path** to open the converted file path in the file browser.
 - ⇒ The path can be changed with **Select target path**. The change is retained even after restarting the PC.



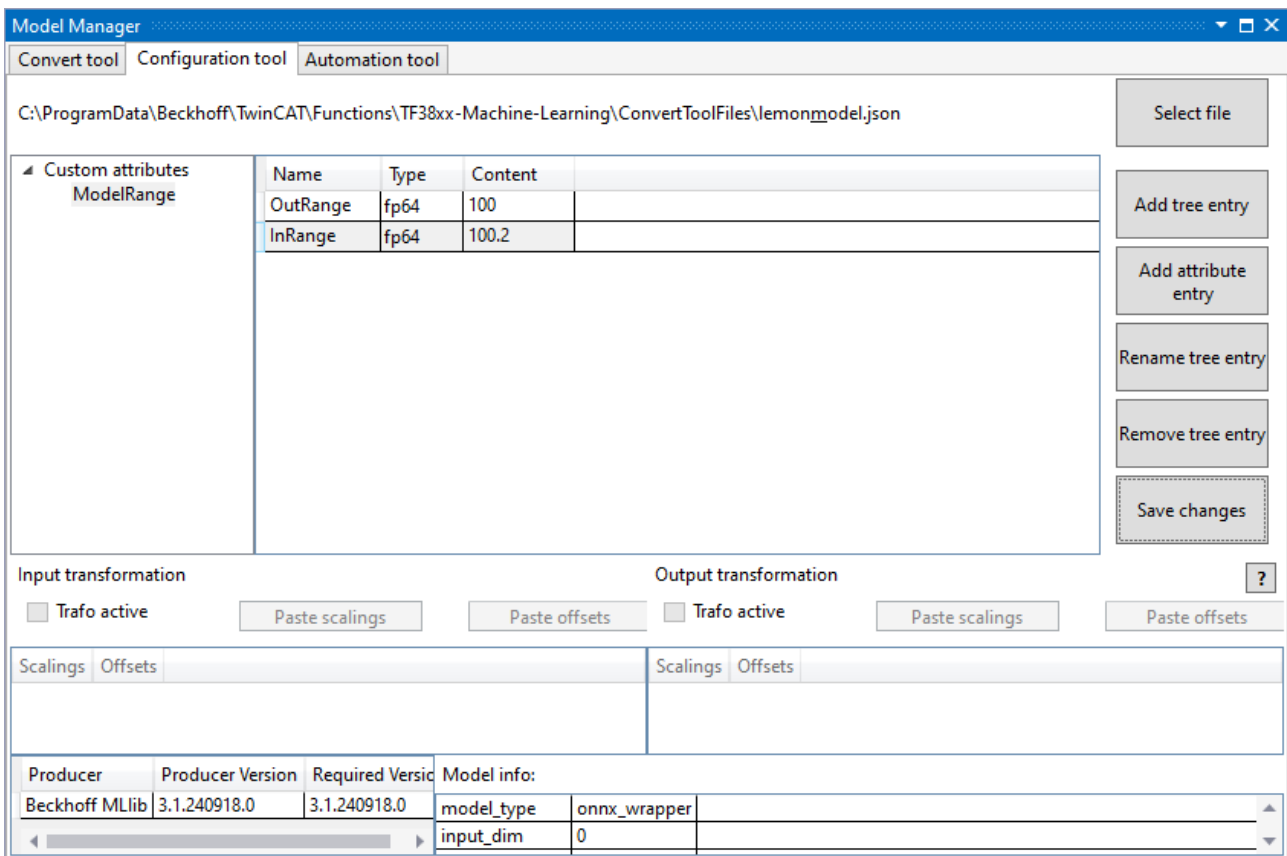
The tool described here is also the central tool for managing ONNX files for the TwinCAT Machine Learning Inference Engine and TwinCAT Neural Network Inference Engine. For this reason, description files for the optional execution of the AI model are also generated with these products if the ONNX is compatible. The Convert to *.xml drop-down menu is relevant for these products. This is not relevant for the TwinCAT Machine Learning Server.

A pop-up window indicates the TwinCAT product with which the ONNX is compatible for each converted ONNX.



Creating Custom attributes

A JSON can be selected via **Select File** and then edited. After editing, the original file is overwritten using **Save changes**.



The Custom attributes are edited using the buttons:

- **Add attribute entry:** Adds an attribute to the selected Tree item. The Tree item must be selected in the left-hand list.
 - If an attribute is created, then the name, the data type and the value or values respectively must be specified.
 - The attributes are deleted by selecting the attribute and pressing the Delete button.
- **Add tree entry:** Adds a Tree item under the selected Tree item (as a Sub Tree Item).
- **Rename tree entry:** The selected Tree item can be renamed.
- **Remove tree entry:** The selected Tree item incl. the Sub Tree Items is deleted.

5.2.2 Python interface

Installation of the Python package

The Python package is stored as a whl file in the folder <TwinCatInstallDir>\Functions\TF38xx-Machine-Learning\Utilities\ModelManagerAPI\PythonPackage.

To install the package, use `pip install <TwinCatInstallDir>\Functions\TF38xx-Machine-Learning\Utilities\ModelManagerAPI\PythonPackage\<whl-file-name>`. The folder may contain different versions of the package (only if you have installed a new setup on top of an old TwinCAT Machine Learning Setup).

Make sure you always use the current version.

Using the API

The package is loaded with

```
import beckhoff.toolbox as tb
```

Creating a JSON and PicoOpenXml from an ONNX file

```
tb.onnxprep("C:\\PathTo\\myONNX.onnx")
```

Display of model information

```
tb.info("C:\\PathTo\\myONNX.json")
```

Add Custom Attributes

```
new_ca = { 'nID' : -34234, 'bTested' : True, 'fNum' : 324.3E-12, 'AnotherTreeItem' : { 'fPi' : 3.134
12, 'bFalseFlag' : False }}
tb.modify_ca("C:\\PathTo\\myONNX.json", "C:\\PathTo\\myONNX_ca.json", new_ca)
```

Add Model Description (name, version, author, etc. of a model)

```
model_description = {
    "new_version" : "2.3.1.0",
    "new_name" : "CurrentPreControlAxis42",
    "new_desc": "This is the most awesome model to control Axis42",
    "new_author": "Max",
    "new_tags": "awesome, ingenious, astounding",
}
tb.modify_md("C:\\PathTo\\myONNX.json", "C:\\PathTo\\myONNX_md.json",
**model_description)
```

5.2.3 Command Line Interface

The Model Manager can also be used via the Command Prompt. The `mllib_toolbox.exe` is available for this purpose.

The Executable is located in `<TwinCatInstallDir>\\Functions\\TF38xx-Machine-Learning\\Utilities\\ModelManagerAPI`.

Help is displayed by running the exe without arguments.

```

Command Prompt
C:\ProgramData\Beckhoff\TwinCAT\Functions\TF38xx-Machine-Learning\Utilities\ModelManagerAPI>mllib_toolbox.exe

=====
Beckhoff MLLib Toolbox
=====
ERROR: Need at least one command.

General Help / Command Line arguments:
-----
mllib_toolbox <command> <param1> ... <paramN> [--opt1] ... [--opt2]

* hwinfo
  Shows detailed info about the computers hardware capabilities.

* info <input file>
  Shows detailed info about the model contained in the specified file.

* store <input file> [output file]
  Stores the input file to the destination file and format, which is
  specified using the file extension, for example 'dst.xml' or 'dst.bml'.

* onnximport <input ONNX file> [output file]
  Imports an ONNX neural network file to the native MLLib file formats.

* convert <input file> [output file] --output_model='dst_model_type'
  Attempts to convert the model stored in the input file to the
  specified target machine learning model type.

```

Using the API

Creating a JSON and PlcOpenXml from an ONNX file

```
mllib_toolbox.exe onnxprep C:\PathTo\mymodel.onnx
```

Display of model information

```
mllib_toolbox.exe info C:\PathTo\mymodel.json
```

The creation of Custom attributes is not supported in the CLI.

5.3 ONNX Support

Supported ONNX opset version

The TwinCAT Machine Learning Server, more precisely the TcMIServer service, supports ONNX Opset version 21. Backward compatibility with more recent ONNX Opset versions is generally provided by ONNX.

The Opset version used is normally listed in the ONNX file under *imports*. In the image below, visualized with Netron, ONNX Opset version 18 as an example.

Restrictions on supported ONNX properties

No dynamic input or output shapes

Dynamic input or output shapes are not supported. Only the leading batchsize parameter may be a dynamic value (see next section).

Permitted are for example:

```
float32[244, 244, 1]
```

```
float32[1, 3, 244, 244]
```

```
float32[5, 244, 244, 3]
```

Not permitted are:

```
float32[244, 244, ?]
```

```
float32[244, height, 3]
```

```
float32[?, 244, 244, ?]
```

```
float32[1, 244, 244, unknown]
```

You can quickly fix the shape of ONNX nodes, e.g. using the `onnxruntime` package in Python, see [Make dynamic input shape fixed | onnxruntime](#).

The shape...

```
float32[-1, 3, ?, ?]
```

becomes with...

```
python -m onnxruntime.tools.make_dynamic_shape_fixed --input_name x --input_shape 1,3,960,960
model.onnx model.fixed.onnx
```

```
float32[1, 3, 960, 960]
```

Batchsize must be leading

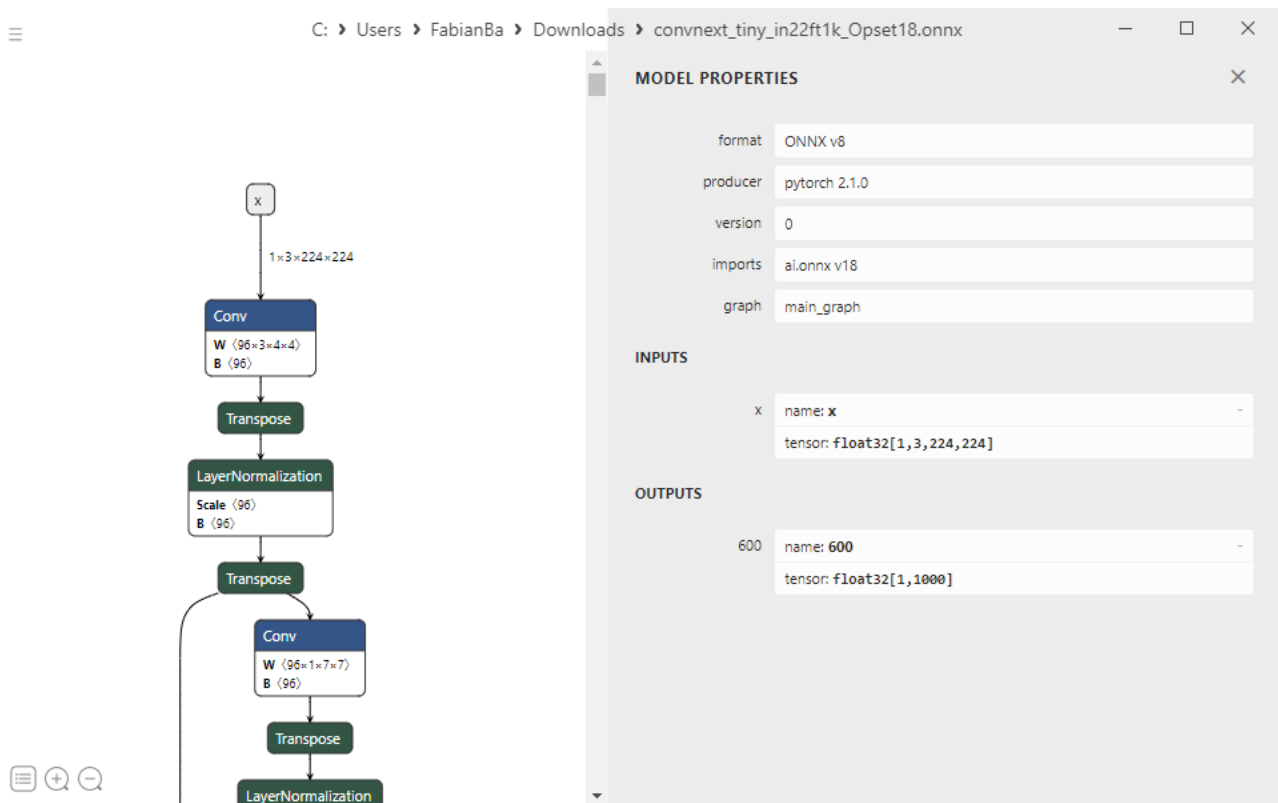
The TwinCAT Machine Learning Server only supports models with Batchsize as the leading parameter of the input node. The batchsize parameter may be dynamic (unlike all other parameters). Permitted are for example:

```
float32[batch, 3, 244, 244]
```

```
float32[?, 3, 244, 244]
```

```
float32[unknown, 244, 244, 3]
```

The `PredictBatched()` [\[▶ 36\]](#) method can only be used if the model contains a dynamic batchsize as the leading parameter.



ONNX file size

Currently the size limit for a loadable ONNX file is 2 GB. Contact Beckhoff (see [Support and Service](#) [► 50]) if this limit is a challenge for your application.

5.4 TcMIServer Service

The installation of TF3820 can be verified by checking the presence of the TcMIServer.exe service. Note that the TcMIServer service offers a delayed autonomous start on restart. Please note the delayed start time here. The TcMIServer allocates ADS port 19900 on the system at startup.

System requirements

Apart from a 64-bit Windows and a TC1000 TwinCAT 3 ADS installation (workload TC1000.ADS.XAR), the TcMIServer does not formally place any further requirements on the system apart from a hard disk requirement of approx. 500 MB. However, the highest possible number of UM cores is recommended for high-performance operation of the TcMIServer.

License query

When starting, the TcMIServer checks whether the TF3820 license is available. If the license is not initially available, the license status is queried every ten seconds. It may therefore take a short time for a license to become effective. A corresponding error code is issued for requests if the TcMIServer is not correctly licensed.

Installation path and log files:

C:\ProgramData\Beckhoff\TwinCAT\Functions\TF38xx-Machine-Learning\TcMIServer

In support cases, log files with any recorded messages are available in the logs folder.

The logs folder has the following structure:

After each restart, the TcMIServer creates a new directory whose name is made up of the creation date and time. The logs are stored in JSON file format in the directory and regularly written by the TcMIServer, if available. In the event of an error, there may therefore be several JSON files whose names specify an interval of the errors or warnings they contain.

5.4.1 Execution Provider

The following ExecutionProviders are currently available:

- CPU
- CUDA

The [ExecutionProvider \[▶ 38\]](#) is transferred with the [Configure \[▶ 34\]](#) method of the [FB_MISvrPrediction \[▶ 33\]](#).

CPU

The loaded AI model is executed on the CPU resources of the IPC. If a TwinCAT runtime is active on the same device, only the CPU resources that are not used by TwinCAT can be used (isolated cores are only used by TwinCAT, shared cores can only be used for a limited time). The operating system takes over the parallelization of the calculation on all available threads.

If several clients create a session with Execution Provider CPU on a server, the inference requests are processed one after the other. Please note the `Priority` parameter when calling `Predict`.

CUDA

The loaded AI model is executed on the GPU resources of the IPC. Several sessions can be run in parallel on one GPU if the GPU resources are sufficient.

The `nDeviceId` field allows you to distinguish between multiple graphics cards that may be installed. For computers with a maximum of one graphics card, the value can be left at the default value, otherwise the field corresponds to the CUDA compute index of the graphics cards. If a computer with several graphics cards is used, the following **system environment variable** should be set:

```
CUDA_DEVICE_ORDER='PCI_BUS_ID'
```

Furthermore, the `TcMIServer` allows sharing of inference resources between different FBs that provide the same specification for their inference session. The use of a shared inference engine (`bExclusiveSession = FALSE`) can occur in cases where a bottleneck - for example in memory on graphics cards - would otherwise be expected. For stateful models (e.g. recurrent models), however, such a configuration should be avoided.

6 API

6.1 Function blocks

6.1.1 FB_MlSvrPrediction

FB_MlSvrPrediction is a TcMIServer client that provides asynchronous and optionally hardware-accelerated AI model inference for the PLC.

The function block is located in the **PLC library Tc3_MIServer**.

Syntax

Declaration:

```
fbMlSvr : FB_MlSvrPrediction;
```

Definition:

```
FUNCTION_BLOCK FB_FTR_IIRCoeff
VAR_INPUT
    stPredictionParameter : ST_PredictionParameter;
END_VAR
VAR_OUTPUT
    bError                : BOOL;
    nErrorCode            : HRESULT;
    bConfigured           : BOOL;
    nMaxInferenceDuration : UDINT;
END_VAR
```

Inputs

Name	Type	Description
stPredictionParameter ▶ 38	ST_PredictionParameter	Configuration structure of the Machine Learning Server session

Outputs

Name	Type	Description
bError	BOOL	TRUE if the currently pending asynchronous request to the TcMLServer was canceled with an error.
nErrorCode	HRESULT	Return code ▶ 46 for the currently pending asynchronous request to the TcMIServer.
bConfigured	BOOL	TRUE if the configuration was successful. Indicates whether the function block has a valid session with the TcMIServer. If FALSE, "configure" must be called. Please note: bConfigured can change to FALSE due to an error during the execution of the request at the TcMIServer.
nMaxInferenceDuration	UDINT	Maximum number of PLC cycles required to execute an inference. See also Execute AI model ▶ 23 .

Methods

Name	Definition location	Description
Configure ▶ 34	Local	Create a session for the FB instance on the TcMIServer according to the configuration defined in stPredictionParameter ▶ 38 .
Deconfigure ▶ 35	Local	End a session of the function block on the TcMIServer. The allocated resources on the server are released again.

Name	Definition location	Description
GetCustomAttributeArray [▶ 37]	Local	Get custom attributes of type ARRAY of the AI model
GetCustomAttributeBool [▶ 37]	Local	Get custom attributes of type BOOL of the AI model
GetCustomAttributeFp64 [▶ 37]	Local	Get custom attributes of type LREAL of the AI model
GetCustomAttributeInt64 [▶ 37]	Local	Get custom attributes of type LINT of AI model
GetCustomAttributeStr [▶ 38]	Local	Get custom attributes of type STRING of the AI model
Predict [▶ 35]	Local	Transmission of an asynchronous inference request to the TcMIServer
PredictBatched [▶ 36]	Local	Transmission of an asynchronous, bundled inference request to the TcMLServer

6.1.1.1 Asynchronous methods

The asynchronous methods of `FB_MlSvrPrediction` are characterized in their signature by the fact that they accept a *timeout* parameter and return a *bool* indicating the execution state of the asynchronous request. The PLC program must therefore wait for the asynchronous methods to finish executing and configure the *timeout* according to the requirements of the application so that the application can respond appropriately to any delays in the TcMIServer. Note that the specified *timeout* is given in the unit PLC task cycles.

It is also important to note that the processing of the asynchronous request can of course only be monitored with the temporal granularity of the **PLC task cycle time**. It is therefore important that applications with a low delay budget have the lowest possible cycle times and correspondingly high sampling rates in order to minimize delays caused by time resolution. This is particularly important when interacting with other, computationally intensive and synchronously executed algorithms, e.g. when pre- or post-processing data.

However, a restriction regarding the minimum PLC task cycle time is imposed by the amount of data that has to be transported from the client to the server. When transferring large amounts of data, such as large image data, the PLC task cycle time must be configured to prevent cycle timeouts.

6.1.1.1.1 Configure()

Calling the method *configure* instantiates a session for the respective instance of the function block `FB_MlSvrPrediction` in the TcMIServer. The configuration defined in the FB member `stPredictionParameter` is used for instantiation.

The instantiation of a session can take a considerable amount of time, in particular because several inferences are made to temper the inference engine (*model warm-up*). This fact should be taken into account when selecting the timeout parameter of the method call.

It should also be taken into account that calling the method when configuring a CUDA accelerated session temporarily requires exclusive access to the GPU. The configuration of such a session can therefore interfere considerably with the inference performance of other FBs operating in parallel.

After the method indicates the completion of the processing of the asynchronous instantiation call by returning TRUE, the result can be evaluated via the FB members `bError` and, in the event of an error, `nErrorCode`. Once an inference session has been successfully instantiated, the FB will set the member `bConfigured` to TRUE.

See also [Configuring the server from the PLC client](#) [[▶ 22](#)].

	Parameter	Type	Default	Description
INPUT	nTimeout	ULINT		Number of PLC task cycles before the timeout error is returned.

	Parameter	Type	Default	Description
INPUT	nPriority	UDINT	0	Priority of the request. Bigger means higher priority.
OUTPUT	Configure	BOOL		Return value. TRUE as soon as the result of the asynchronous call is available. The result of the call can then be checked using the 'bError' and 'nErrorCode' properties.

6.1.1.1.2 Deconfigure()

The Deconfigure method closes the inference session of the FB in the TcMIServer. After a successful call to Deconfigure, the FB can set up a new inference session using the Configure method.

After a successful configuration of an inference session, all calls to Configure are ignored until a call to Deconfigure has been made.

	Parameter	Type	Default	Description
INPUT	nTimeout	ULINT		Number of PLC task cycles before the timeout error is returned.
OUTPUT	Deconfigure	BOOL		Return value. TRUE as soon as the result of the asynchronous call is available. The result of the call can then be checked using the 'bError' and 'nErrorCode' properties.

6.1.1.1.3 Predict()

The Predict method submits an asynchronous inference order to the TcMIServer.

The method expects the provision of input data according to the specification of the ONNX file, see [Preparing ONNX for use with TwinCAT Machine Learning Server](#) [► 19].

The provided pointer to the output data must be valid and point to an instance of the output data type according to the created PlcOpenXml. Once the asynchronous inference has been successfully completed, the data in the transferred output memory area is valid and released for further processing.

See also [Execute AI model](#) [► 23].

	Parameter	Type	Default	Description
INPUT	pDataIn	PVOID		Pointer to the instance of the input data type
INPUT	nDataInSize	UDINT	0	Size of the input data type
INPUT	pDataOut	PVOID		Pointer to the instance of an output data type
INPUT	pDataOutSize	UDINT		Size of the output data type
INPUT	nTimeout	ULINT		Number of PLC task cycles before the timeout error is returned.

	Parameter	Type	Default	Description
INPUT	nPriority	UDINT	0	Priority of the request. Bigger means higher priority.
OUTPUT	Predict	BOOL		Return value. TRUE as soon as the result of the asynchronous call is available. The result of the call can then be checked using the 'bError' and 'nErrorCode' properties.

6.1.1.1.4 PredictBatched()

The PredictBatched method submits an asynchronous batch inference order to the TcMIServer.

The method expects the provision of an array of the input data type of the model according to the specification of the ONNX file, see [Preparing ONNX for use with TwinCAT Machine Learning Server](#) [► 19].

The provided pointer to the output data must be valid and point to an instance of an array of output data types according to the created PlcOpenXml. Once the asynchronous inference has been successfully completed, the data in the transferred output memory area is valid and released for further processing.

See also [Execute AI model](#) [► 23].

	Parameter	Type	Default	Description
INPUT	pDataIn	PVOID		Pointer to the instance of an array of input data types
INPUT	nDataInSize	UDINT	0	Size of the input data type (size of an element, not of the array)
INPUT	nBatchSize	UINT		Size of the batch
INPUT	pDataOut	PVOID		Pointer to the instance of an output data type
INPUT	pDataOutSize	UDINT		Size of the output data type
INPUT	nTimeout	ULINT		Number of PLC task cycles before the timeout error is returned.
INPUT	nPriority	UDINT	0	Priority of the request. Bigger means higher priority.
OUTPUT	PredictBatched	BOOL		Return value. TRUE as soon as the result of the asynchronous call is available. The result of the call can then be checked using the 'bError' and 'nErrorCode' properties.

6.1.1.2 Synchronous methods

6.1.1.2.1 GetCustomAttribute_array

	Parameter	Type	Description
INPUT	sCustomAttributeName	T_MaxString	Name of the Custom attribute
INPUT	fmtAttributeDataType	Reference to ETcMIIDataType	Data format of the Custom attribute
INPUT	pDataBuffer	PVOID	Destination data buffer into which the user-defined attribute is copied.
INPUT	nDataBufferLen	UDINT	Length of the destination data buffer in bytes
INPUT	nArrayLength	Reference To UDINT	Number of data type elements (i.e. number of fp32 values) found in the user-defined attribute.
INPUT	pnBytesWritten	Pointer To UDINT	Returns the number of bytes written to the destination buffer.
OUTPUT	GetCustomAttribute_array	BOOL	TRUE if an error occurred in the method.

6.1.1.2.2 GetCustomAttribute_bool

	Parameter	Type	Description
INPUT	sCustomAttributeName	T_MaxString	Name of the Custom attribute
INPUT	nDataOut	Reference To BOOL	Output value Custom attributes of type Bool
OUTPUT	GetCustomAttribute_bool	BOOL	TRUE if an error occurred in the method.

6.1.1.2.3 GetCustomAttribute_fp64

	Parameter	Type	Description
INPUT	sCustomAttributeName	T_MaxString	Name of the Custom attribute
INPUT	nDataOut	Reference To LREAL	Output value of the Custom attribute fp64
OUTPUT	GetCustomAttribute_fp64	BOOL	TRUE if an error occurred in the method.

6.1.1.2.4 GetCustomAttribute_int64

	Parameter	Type	Description
INPUT	sCustomAttributeName	T_MaxString	Name of the Custom attribute
INPUT	nDataOut	Reference To LINT	Output value of the Custom attribute int64
OUTPUT	GetCustomAttribute_int64	BOOL	TRUE if an error occurred in the method.

6.1.1.2.5 GetCustomAttribute_str

	Parameter	Type	Description
INPUT	sCustomAttributeName	T_MaxString	Name of the Custom attribute
INPUT	nDataOut	Reference To T_MaxString	Output value of the Custom attribute of type String
OUTPUT	pnStringLen	Pointer To UDINT	(Optional) Actual length of the string attribute
OUTPUT	GetCustomAttribute_string	BOOL	TRUE if an error occurred in the method.

6.2 Data types

6.2.1 E_ExecutionProvider

Enum describes all supported execution modes of the TcMIServer.

Name	Type	Value	Description
CPU	USINT	0	Execution on CPU
CUDA	USINT	1	Execution on CUDA-capable NVIDIA GPUs

6.2.2 ST_PredictionParameter

Configuration options for an inference session on the TcMIServer.

Name	Type	Default	Description
sMIModelFilePath	STRING(255)		Fullpath to the created JSON file (see Model Manager [▶ 26])
eExecutionProvider	E_ExecutionProvider [▶ 38]	ExecutionProvider.CPU	The AI model named under sMIModelFilePath is to be executed on this specified hardware.
nDeviceId	UDINT	0	Index of the desired GPU device when using the "CUDA" ExecutionProvider. The index corresponds to the CUDA Compute Index and is only relevant for IPCs with multiple GPUs.
bExclusiveSession	BOOL	TRUE	Determines the exclusivity of the inference session that is created for the FB instance on the TcMIServer. If TRUE, the TcMIServer creates an exclusive session, which is necessary for state-dependent models (e.g. RNNs) in order to avoid interference. If FALSE, the session can be shared with other FB instances

Name	Type	Default	Description
			that request the same configuration, which can reduce the memory load.
nSessionTimeout	ULINT	72	Duration of inactivity in hours after which the FB session on the TcMIServer expires. The server then releases the allocated resources of the relevant client again.
sMISvrNetId	T_AmsSvrNetId	'127.0.0.1.1.1'	AMS Net Id of the device on which the TcMIServer service is accessible. Default is 'local'.

7 Samples

7.1 AI-based image processing

This sample demonstrates how to:

- Use TwinCAT Vision to load image data from the local hard disk and make it available in the PLC.
- Pre-process images with the TwinCAT Vision library.
- Use FB_MISvrPrediction to start a session on the locally installed TwinCAT Machine Learning Server and load an AI model (classification model).
- Execute the inference on the TwinCAT Machine Learning Server.
- Continue to use the result in the PLC.

Download and overview of the files

You can download the project here: https://infosys.beckhoff.com/content/1033/TF3820_TC3_Machine_Learning_Server/Resources/17328164363.zip

- The ZIP contains a **tnzip**, which you can open in TwinCAT XAE via *File > Open > Open Solution from Archive....*
- The models folder contains an **ONNX** and the already created **JSON** and PlcOpenXml.
- The dataset folder contains **sample images** that are to be processed.

Requirements

Install the following workloads:

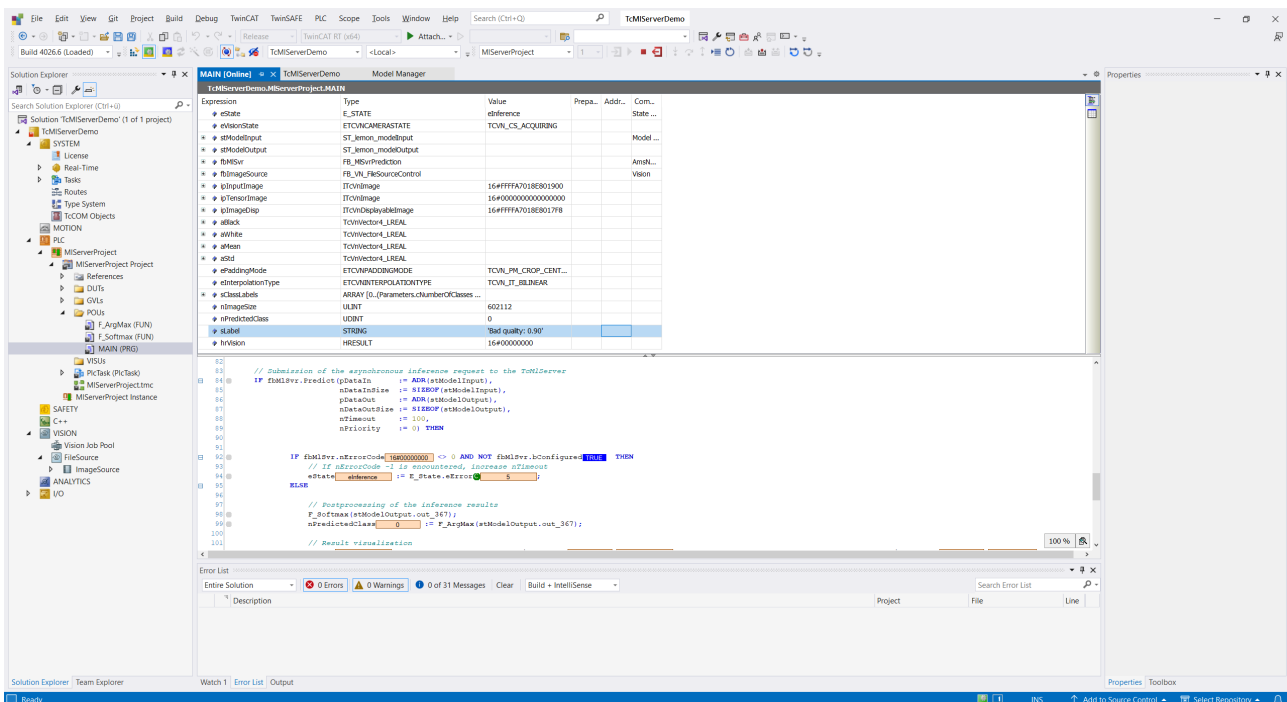
- TwinCAT Standard
- TF3820 | TwinCAT Machine Learning Server
- TF3830 | TwinCAT Machine Learning Server Client
- TF7xxx | TwinCAT 3 Vision

Setting up the project

- Open the tnzip and save your project.
- Set up the FileSpource:
 - In the System Manager, select the Tree Item *Vision > FileSource > ImageSource*
 - Add the images from the models folder.
 - Set the Cycle Time of the ImageSource to 100 ms.
- Name the FullPath to the model JSON in line 9 of the MAIN. The ONNX must be in the same folder.
- Make sure that all software licenses are available at least in the 7-day trial license:
 - TC1200 TwinCAT PLC
 - TF3820 TwinCAT Machine Learning Server
 - TF7100 TwinCAT Vision Base

Executing the project

Start the application with Activate Configuration on your target system. If all points are set correctly, after a short time the `eState` should be set to `eInference` and the variable `sLabel` should display the result of the current inference.



If an error has occurred, the eState is set to Error. You can then open the fbMlSvr instance and read out the error code. Use the [table of error codes \[▶ 46\]](#) to narrow down the problem.

Excerpts from the PLC program

Declaration

In the declaration, the main points concerning the handling of the TwinCAT Machine Learning Server are the input and output data types of the AI model and the instance of the clients to the Machine Learning Server.

```

stModelInput : ST_lemon_modelInput; //model input datatype, imported via PlcOpenXml
stModelOutput : ST_lemon_modelOutput; //model output datatype, imported via PlcOpenXml
fbMlSvr : FB_MlSvrPrediction(); // Instance of Client to TcMlServer

```

The data types have already been read into the TwinCAT project via the PlcOpenXml (see models folder). The description can be found in the DUTs folder.

Configuration of the session

In this sample, the client opens a session on the TwinCAT Machine Learning Server in the E_State.eMlSvrConfiguration state. This specifies the system on which the server is accessible, which model is loaded in the session and on which hardware the model is to be executed.

```

fbMlSvr.stPredictionParameter.sMlModelFilePath := 'C:\models\lemon_model.json'; // fullpath to model
fbMlSvr.stPredictionParameter.sMlSvrNetId := '127.0.0.1.1.1'; //
Server on local system
fbMlSvr.stPredictionParameter.eExecutionProvider := E_EXECUTIONPROVIDER.CPU;
// CPU execution

// Submit configuration request to the TcMlServer
// Provide a generous nTimeout, as the configuration can take a substantial amount of time
IF fbMlSvr.Configure(nTimeout := 1000, nPriority:=0) THEN
  IF fbMlSvr.nErrorCode <> 0 THEN
    // If nErrorCode -1 is encountered, increase nTimeout
    eState := E_State.eError;
  ELSE
    eState := E_State.eImageAcquisition;
  END_IF
END_IF

```

Calling the method `Configure()` sends the request to open a session to the server. The call is asynchronous to the PLC task and is acknowledged with a `TRUE` when the session setup has been successfully completed.

Executing the model

In the state `E_State.eInference`, the `Predict` call is sent to the Machine Learning Server. This call is also asynchronous to the PLC task. The method returns `TRUE` if the result is available.

In this sample, the image of type `ITcVnImage` is copied to the model input data type using the `F_VN_ExportImage` function before the inference call.

```
F_VN_ExportImage(ipTensorImage, ADR(stModelInput.in_input1), nImageSize, hrVision);
// Submission of the asynchronous inference request to the TcMlServer
IF fbMlSvr.Predict(pDataIn      := ADR(stModelInput),
                 nDataInSize   := SIZEOF(stModelInput),
                 pDataOut      := ADR(stModelOutput),
                 nDataOutSize  := SIZEOF(stModelOutput),
                 nTimeout      := 100,
                 nPriority      := 0) THEN
  IF fbMlSvr.nErrorCode <> 0 AND NOT fbMlSvr.bConfigured THEN
    // If nErrorCode -1 is encountered, increase nTimeout
    eState := E_State.eError;
  ELSE

    // Postprocessing of the inference results
    F_Softmax(stModelOutput.out_367);
    nPredictedClass := F_ArgMax(stModelOutput.out_367);
```

The result of the inference can be used after a successful error check.

7.2 Custom attributes

With the Custom attributes, it is possible to provide the AI model with metadata that is to be taken into account in the PLC at runtime. The metadata concept also exists similarly in ONNX, but these cannot be evaluated in the PLC at runtime.

In principle, you can use both metadata areas at the same time. Distinguish between information required at runtime in the PLC (Custom attribute area) and information required by the PLC programmer only at the engineering stage (ONNX metadata or Custom attributes).

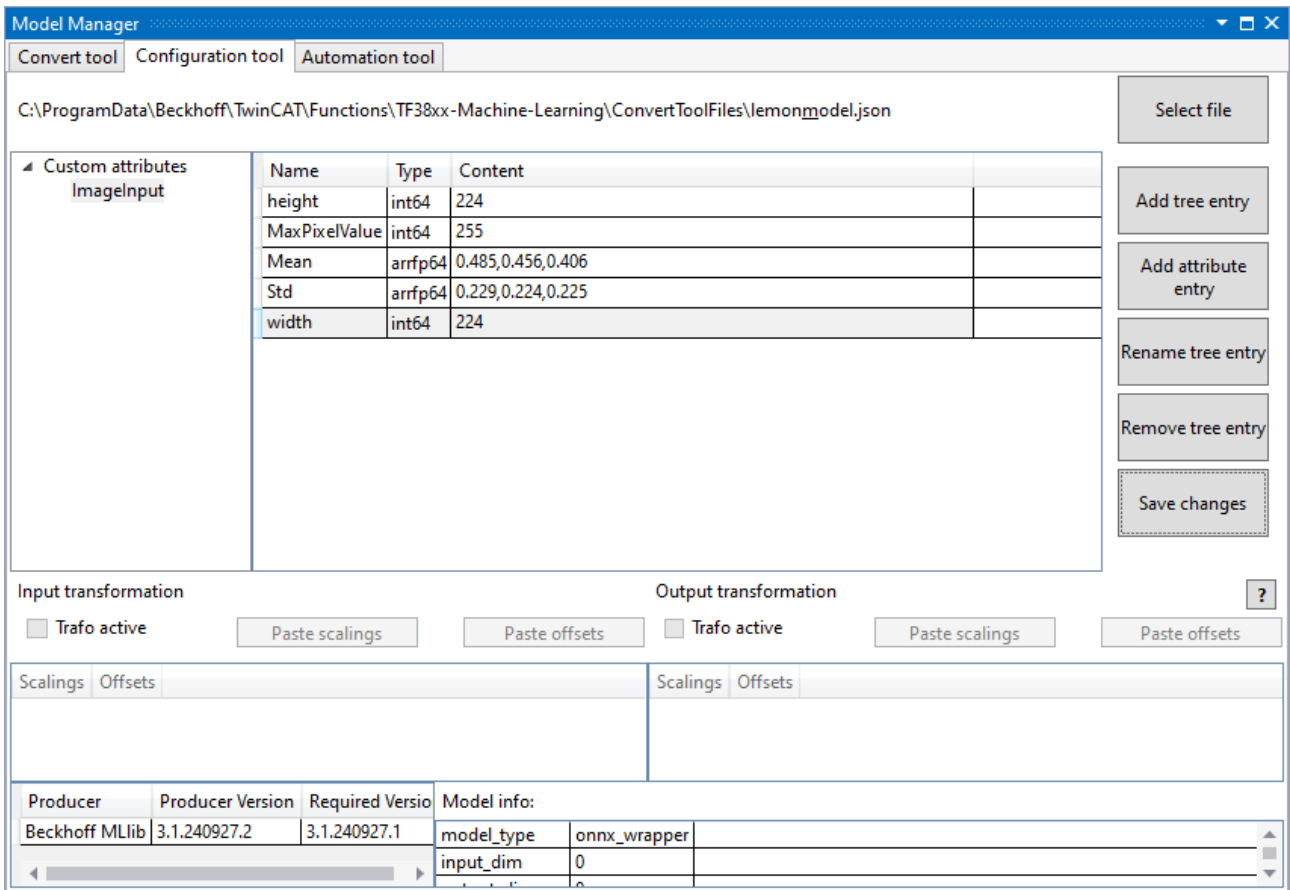
In any case, the aim is for the creator of an ONNX to be able to pass on enough information to the consumer of the ONNX so that the consumer can use the AI model correctly and efficiently.

Custom attributes for image preprocessing

In the following, the sample [AI-based image processing \[▶ 40\]](#) is extended. Information about the input image is added. The aim is to adapt the image pre-processing according to the metadata.

Implementation in the TwinCAT Machine Learning Model Manager

The Configuration Tool can be used in the graphical environment to enter the Custom attributes.



Implementation in Python

From a workflow perspective, it is often more convenient to enter all required Custom attributes directly in Python after model training.

```
new_ca = { 'ImageInput' : {'nwidth' : 244, 'nheight' : 244, 'nMaxPixelValue' : 255, 'fMean' : [0.485, 0.456, 0.40], 'fStd' : [0.229, 0.224, 0.225]} }
tb.modify_ca("C:\\models\\lemon_model.json", "C:\\models\\lemon_model.json", new_ca)
```

The result can be traced in plain text in the JSON.

```

1  {
2      "MLlib_JSON_File": {
3          "MachineLearningModel": {
4              "str_modelName": "onnx_wrapper",
5              "CustomAttributes": {
6                  "ImageInput": {
7                      "int64_height": 224,
8                      "int64_MaxPixelValue": 255,
9                      "fp64_Mean": [0.485, 0.456, 0.406],
10                     "fp64_Std": [0.229, 0.224, 0.225],
11                     "int64_width": 224
12                 }
13             },
14             "AuxiliarySpecifications": {
15                 "PTI": {
16                     "str_producer": "Beckhoff MLlib",
17                     "str_producerVersion": "3.1.240927.2",
18                     "str_requiredVersion": "3.1.240927.1"
19                 }
20             },
21             "Configuration": {
22                 "str_onnxHash": "1NhwyT0/h40XAvlgHBf2PUE8AC7p/SN1KeShHcawoCg=",
23                 "Inputs": {
24                     "TensorDesc0": {
25                         "str_name": "input.1",
26                         "str_dt": "fp32",
27                         "int32_shape": [1, 3, 224, 224]
28                     }
29                 },
30                 "Outputs": {
31                     "TensorDesc0": {
32                         "str_name": "367",
33                         "str_dt": "fp32",
34                         "int32_shape": [1, 3]
35                     }
36                 }
37             }
38         }
39     }
40 }

```

JSON file length: 867 lines: 40 Ln: 11 Col: 38 Pos: 296 Unix (LF) UTF-8 INS

Reading the Custom attributes in TwinCAT

The PLC project from the sample is expanded below.

Once a session has been successfully configured on the TwinCAT Machine Learning Server, the stored Custom attributes are read out.

```

IF fbMLsvr.Configure(nTimeout := 1000, nPriority:=0) THEN
  IF fbMLsvr.nErrorCode <> 0 THEN
    // If nErrorCode -1 is encountered, increase nTimeout
    eState := E_State.eError;
  ELSE

    // read all relevant meta information from custom attributes
    IF fbMLsvr.GetCustomAttribute_int64('ImageInput/height', nHeight) THEN
      // check fbMLsvr.nErrorCode for further reference
      eState := E_State.eError;
    END IF
    fbMLsvr.GetCustomAttribute_int64('ImageInput/width', nWidth);
    fbMLsvr.GetCustomAttribute_int64('ImageInput/MaxPixelValue', nMaxPixelValue);
    fbMLsvr.GetCustomAttribute_array('ImageInput/
Mean', dtypeCustomAttribute, ADR(aMean), SIZEOF(aMean), nLength, ADR(nBytes));
    fbMLsvr.GetCustomAttribute_array('ImageInput/

```

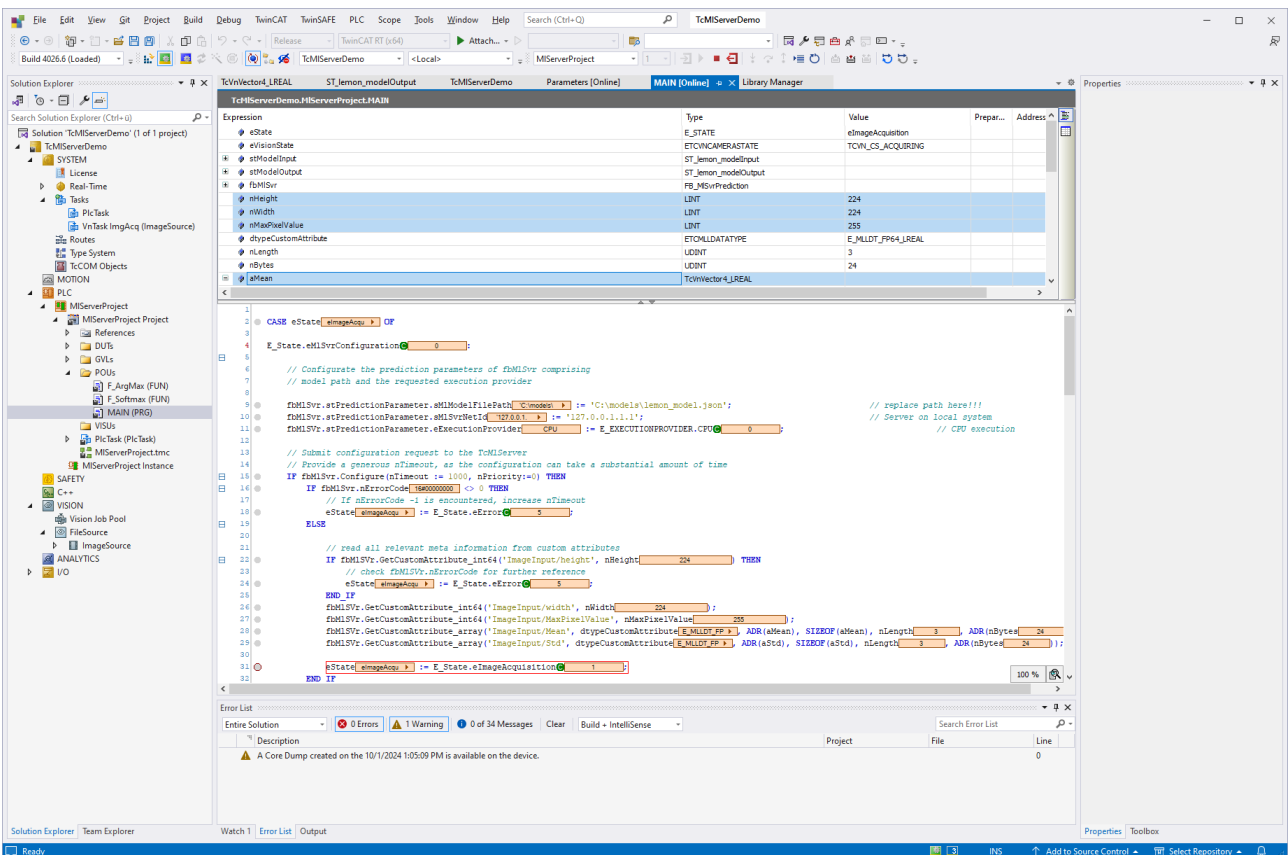
```
Std', dtypeCustomAttribute, ADR(aStd), SIZEOF(aStd), nLength, ADR(nBytes));

    eState := E_State.eImageAcquisition;
END_IF
END_IF
```

The Custom attributes stored in PLC variables are then used in the pre-processing pipeline. Essentially, all that needs to be considered here are the necessary type conversions.

```
// Adjust the dimensions of the input image to match the model input requirements
hrVision := F_VN_ResizeImageExp(ipInputImage, ipTensorImage, LINT_TO_UDINT(nWidth), LINT_TO_UDINT(nHeight), eInterpolationType, ePaddingMode, aBlack, hrVision);
// Convert the image to type REAL and scale to the range [0.0, 1.0]
hrVision := F_VN_ConvertElementTypeExp(ipTensorImage, ipTensorImage, TCVN_ET_REAL, 1.0 / LINT_TO_REAL(nMaxPixelValue), 0, hrVision);
// Normalization
hrVision := F_VN_SubtractVectorFromImage(ipTensorImage, aMean, ipTensorImage, hrVision);
hrVision := F_VN_DivideImageByVector(ipTensorImage, aStd, ipTensorImage, hrVision);
```

Once the configuration has been activated, you can use the Online View to check that the values have been applied correctly.



8 Appendix

8.1 Error Codes

Error Code	Description	Hints
0	Operation successful	-
-1	ML server timeout	Increase the timeout argument of your request. Check, that the TcMIServer service is running.
-3	ML server malformed data	-
-5	PLC protocol error	Make sure to call the methods of the FB in a canonical order, i.e. make sure the FB is configured before making inference calls.
-8	Invalid AMS Net ID	Check the syntactical correctness of the provided AmsNetId.
-9	Input pointer is null	Check the validity of the input data pointer provided to the predict methods.
-11	Output pointer is null	Check the validity of the output data pointer provided to the predict methods.
-13	Batch size is zero	Provide a valid batch size (≥ 1).
-15	Input size is zero	Provide a valid input data size (>0).
-17	Output size is zero	Check your installation of TF3830.
-18	Driver instantiation failed	Check your installation of TF3830.
-20	Driver state propagation to state OP failed	Check the validity and integrity of your model file (existing path, valid file format).
-22	Driver loading failed	Check your installation of TF3830.
-24	Driver state depropagation failed	Check your installation of TF3830.
-26	Driver parameter configuration failed	TwinCAT-internal error
-28	Could not instantiate file accessor	TwinCAT-internal error
-30	Could not propagate file accessor state	TwinCAT-internal error
-32	Could not access model file	Check the validity of your model file (existing path, valid file format).
-34	Mismatch in read model file bytes	TwinCAT-internal error
-36	Could not parse model file	Check the integrity of your model file.
-38	String buffer overflow	Check the integrity of your model file, especially the length of the contained model hash.
-40	Could not find model config in file	Check the integrity of your model file, especially the defined model configuration.
-42	Could not retrieve model attributes	Check the integrity of your model file, especially the availability of a model hash.
-43	ADS server connection error	ADS messages could not be sent. Check integrity of AmsRouter.
-44	Engine mismatch	The model file you intended to load is not designated for the TcMIServer.
-45	Invalid CST attribute name	Check the name of the attribute in your model file or your query. The name must not be empty.
-47	CST attribute retrieval failed	The queried attribute could not be found. Check the queried attribute name in query and model file.
-49	CST attribute invalid buffer	Check the destination pointer provided to the retrieval function. Must not be a null pointer.
-50	Versioned model could not be resolved	Check the validity of the model version you are querying.

Error Code	Description	Hints
-51	If a PLC online change is detected, a running predict is discarded with the error	
-101	Invalid PLC request variant	Internal error
-102	Invalid device index	Verify the device index provided in prediction parameters of the FB.
-103	Invalid request data	Internal error
-201	Invalid data	Internal error
-202	Invalid model type	Requested tensor of unsupported datatype. See log files.
-206	Invalid model	The provided model file is not supported. check especially, that no dimension parameters are left except for an optional batch dimension.
-300	Invalid execution provider	The requested execution provider is invalid. Make sure to select an EP from the enum E_ExecutionProvider.
-302	Unsupported execution provider	The requested execution provder is valid but not supported on your system (for instance due to a lack of GPU support).
-400	Session config file not found	Internal error
-402	Environment config file not found	Internal error
-404	Run config file not found	Internal error
-406	Server config file not found	Internal error
-408	Device config files not found	Internal error
-500	Invalid session config file	Internal error
-502	Invalid environment config file	Internal error
-504	Invalid run config file	Internal error
-506	Invalid server config file	Internal error
-508	Invalid device config files	Internal error
-600	Targeted inference provider unavailable	Internal error
-602	Inference provider session timeout	Increase the session timeout in the prediction parameters.
-701	Dynamic loaded library not found	Internal error
-703	Symbol not loadable from dynamic library	Internal error
-801	Unknown job execution error	Internal error
-803	Unknown error	Internal error
-901	Failed backend execution	Internal error
-1100	License server connection error	Internal licensing error
-1102	License server ADS error	Internal licensing error
-1104	License ADS error	Internal licensing error
-1106	License invalid	Make sure, that a license TF3820 is available.
-1108	License invalid unknown	Internal licensing error
-1201	Expected tensor collection type	Make sure, that only types defined in the PLCopen files generated by the Beckhoff model management products is used. Do not manipulate the contained header
-1203	Mismatching tensor collection hashes	Make sure, that only types defined in the PLCopen files generated by the Beckhoff model management products is used. Do not manipulate the contained header

Error Code	Description	Hints
-1205	Invalid tensor collection header	Make sure, that only types defined in the PLCopen files generated by the Beckhoff model management products is used. Do not manipulate the contained header
-1207	Tensor collection offset feature not supported	Make sure, that only types defined in the PLCopen files generated by the Beckhoff model management products is used. Do not manipulate the contained header
-1209	Internal allocation special type header	Internal error
-1300	Server startup cannot build log directory	No Logs directory available and could not be generated.
-1801	NVIDIA NVML library function failure	Make sure the Nvidia nvml.dll is available.
-1803	NVIDIA NVML library extraction failure	Make sure the Nvidia nvml.dll is available.
-1901	NVIDIA CUDA library function failure	Make sure the Nvidia cuda.dll is available.
-1903	NVIDIA CUDA library extraction failure	Make sure the Nvidia cuda.dll is available.
-2000	Model registry file storage error	Internal error
-2002	Model registry file deletion error	Internal error
-2003	Model registry invalid reference	Internal error
-2004	Model registry entry failed to load model	Internal error
-2006	Model registry inconsistent model hashes	Internal error
-2008	ADS could not open remote file	Make sure, that an .onnx file with the same name as your .json model description file is located in the same directory.
-2010	ADS could not retrieve size of remote file	Internal error
-2012	ADS could not load remote file	Make sure, that your AMS router has sufficient memory for your .onnx file.
-2014	ADS could not close remote file	Internal error
-2018	ADS could not open client port	Make sure your TwinCAT installation is valid.
-2201	Buffer base is null	Internal error
-2203	Not enough memory for head	Internal error
-2205	Buffer out of memory	Internal error
-2207	Buffer cannot read from null	Internal error
-2209	Buffer cannot write to null	Internal error
-2211	Buffer corrupted with invalid string	Internal error
-2020	Insufficient AMS Router Memory on Client	Increase Router Memory on the Client system
-2022	Insufficient AMS Router Memory on Server	Increase Router Memory on the Server system
-2024	Could not read router memory on client	Internal error
-2026	Could not read router memory on server	Internal error
-2301	Invalid CUDA setup detected	Check your CUDA and cuDNN installation.

8.2 Log files

TwinCAT Machine Learning Model Manager Logs: C:\ProgramData\Beckhoff\TwinCAT\Functions\TF38xx-Machine-Learning\Logs

TwinCAT Machine Learning Server Logs: C:\ProgramData\Beckhoff\TwinCAT\Functions\TF38xx-Machine-Learning\TcMIServer\Logs

8.3 Third-party components

This software contains third-party components.

Please refer to the license file provided in the following folder for further information:

C:\Program Files (x86)\Beckhoff\Legal\TwinCAT-XAR-MIServer

C:\Program Files (x86)\Beckhoff\Legal\TwinCAT-XAE-ModelManagerCore

8.4 Support and Service

Beckhoff and their partners around the world offer comprehensive support and service, making available fast and competent assistance with all questions related to Beckhoff products and system solutions.

Download finder

Our [download finder](#) contains all the files that we offer you for downloading. You will find application reports, technical documentation, technical drawings, configuration files and much more.

The downloads are available in various formats.

Beckhoff's branch offices and representatives

Please contact your Beckhoff branch office or representative for [local support and service](#) on Beckhoff products!

The addresses of Beckhoff's branch offices and representatives round the world can be found on our internet page: www.beckhoff.com

You will also find further documentation for Beckhoff components there.

Beckhoff Support

Support offers you comprehensive technical assistance, helping you not only with the application of individual Beckhoff products, but also with other, wide-ranging services:

- support
- design, programming and commissioning of complex automation systems
- and extensive training program for Beckhoff system components

Hotline: +49 5246 963-157
e-mail: support@beckhoff.com

Beckhoff Service

The Beckhoff Service Center supports you in all matters of after-sales service:

- on-site service
- repair service
- spare parts service
- hotline service

Hotline: +49 5246 963-460
e-mail: service@beckhoff.com

Beckhoff Headquarters

Beckhoff Automation GmbH & Co. KG

Huelshorstweg 20
33415 Verl
Germany

Phone: +49 5246 963-0
e-mail: info@beckhoff.com
web: www.beckhoff.com

More Information:
www.beckhoff.com/TF3820

Beckhoff Automation GmbH & Co. KG
Hülshorstweg 20
33415 Verl
Germany
Phone: +49 5246 9630
info@beckhoff.com
www.beckhoff.com

