EcoStruxure Machine Expert Twin

Creating Customized Catalogs

User Guide

EIO0000005040.00
03/2024

www.se.com

Legal Information

The information provided in this document contains general descriptions, technical characteristics and/or recommendations related to products/solutions.

This document is not intended as a substitute for a detailed study or operational and site-specific development or schematic plan. It is not to be used for determining suitability or reliability of the products/solutions for specific user applications. It is the duty of any such user to perform or have any professional expert of its choice (integrator, specifier or the like) perform the appropriate and comprehensive risk analysis, evaluation and testing of the products/solutions with respect to the relevant specific application or use thereof.

The Schneider Electric brand and any trademarks of Schneider Electric SE and its subsidiaries referred to in this document are the property of Schneider Electric SE or its subsidiaries. All other brands may be trademarks of their respective owner.

Schneider Electric does not grant any right or license for commercial use of the document or its content, except for a non-exclusive and personal license to consult it on an "as is" basis.

Schneider Electric reserves the right to make changes or updates with respect to or in the content of this document or the format thereof, at any time without notice.

To the extent permitted by applicable law, no responsibility or liability is assumed by Schneider Electric and its subsidiaries for any errors or omissions in the informational content of this document, as well as any non-intended use or misuse of the content thereof.

Table of Contents

Safety Information

Important Information

Read these instructions carefully, and look at the equipment to become familiar with the device before trying to install, operate, service, or maintain it. The following special messages may appear throughout this documentation or on the equipment to warn of potential hazards or to call attention to information that clarifies or simplifies a procedure.

The addition of this symbol to a “Danger” or “Warning” safety label indicates that an electrical hazard exists which will result in personal injury if the instructions are not followed.

This is the safety alert symbol. It is used to alert you to potential personal injury hazards. Obey all safety messages that follow this symbol to avoid possible injury or death.

⚠ DANGER

DANGER indicates a hazardous situation which, if not avoided, **will result in** death or serious injury.

⚠ WARNING

WARNING indicates a hazardous situation which, if not avoided, **could result in** death or serious injury.

⚠ CAUTION

CAUTION indicates a hazardous situation which, if not avoided, **could result** in minor or moderate injury.

NOTICE

NOTICE is used to address practices not related to physical injury.

Please Note

Electrical equipment should be installed, operated, serviced, and maintained only by qualified personnel. No responsibility is assumed by Schneider Electric for any consequences arising out of the use of this material.

A qualified person is one who has skills and knowledge related to the construction and operation of electrical equipment and its installation, and has received safety training to recognize and avoid the hazards involved.

About the Book

Document Scope

This document describes how to create a digital model of your mechatronic system as well as the corresponding catalog programming in C#.

Validity Note

This document has been created for the release of EcoStruxure Machine Expert Twin V2.0.

Related Documents

Document title	Reference
Cybersecurity Best Practices	CS-Best-Practices-2019-340
Cybersecurity Guidelines for EcoStruxure Machine Expert, Modicon and PacDrive Controllers and Associated Equipment	EIO0000004242
EcoStruxure Machine Expert DigitalTwinCommunication - Library Guide	EIO0000004735 (ENG)
EcoStruxure Machine Expert How to Emulate - User Guide	EIO0000004858 (ENG); EIO0000004859 (FRE); EIO0000004860 (GER); EIO0000004862 (SPA); EIO0000004861 (ITA); EIO0000004863 (CHS)
EcoStruxure Machine Expert Twin Getting Started - User Guide	EIO0000005022 (ENG)

To find documents online, visit the Schneider Electric download center (www.se.com/ww/en/download/).

Product Related Information

⚠WARNING

LOSS OF CONTROL

- Perform a Failure Mode and Effects Analysis (FMEA), or equivalent risk analysis, of your application, and apply preventive and detective controls before implementation.
- Provide a fallback state for undesired control events or sequences.
- Provide separate or redundant control paths wherever required.
- Supply appropriate parameters, particularly for limits.
- Review the implications of transmission delays and take actions to mitigate them.
- Review the implications of communication link interruptions and take actions to mitigate them.
- Provide independent paths for control functions (for example, emergency stop, over-limit conditions, and error conditions) according to your risk assessment, and applicable codes and regulations.
- Apply local accident prevention and safety regulations and guidelines.[1]
- Test each implementation of a system for proper operation before placing it into service.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

[1] For additional information, refer to NEMA ICS 1.1 (latest edition), *Safety Guidelines for the Application, Installation, and Maintenance of Solid State Control* and to NEMA ICS 7.1 (latest edition), *Safety Standards for Construction and Guide for Selection, Installation and Operation of Adjustable-Speed Drive Systems* or their equivalent governing your particular location.

⚠WARNING

UNINTENDED EQUIPMENT OPERATION

- Only use software approved by Schneider Electric for use with this equipment.
- Update your application program every time you change the physical hardware configuration.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

For reasons of Internet security, for those devices that have a native Ethernet connection, TCP/IP forwarding is disabled by default. Therefore, you must manually enable TCP/IP forwarding. However, doing so may expose your network to possible cyberattacks if you do not take additional measures to protect your enterprise. In addition, you may be subject to laws and regulations concerning cybersecurity.

⚠WARNING

UNAUTHENTICATED ACCESS AND SUBSEQUENT NETWORK INTRUSION

- Observe and respect any and all pertinent national, regional and local cybersecurity and/or personal data laws and regulations when enabling TCP/IP forwarding on an industrial network.
- Isolate your industrial network from other networks inside your company.
- Protect any network against unintended access by using firewalls, VPN, or other, proven security measures.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

Consult the Schneider Electric Cybersecurity Best Practices for additional information.

EcoStruxure Machine Expert Twin is a simulation and emulation software suite to create digital models of real machines to start the virtual design, virtual pre-commissioning, and to support co-development before building the machine – thus enabling parallel engineering of mechanical, electrical and controls work assignments.

The simulation, emulation and machine visualization functions of EcoStruxure Machine Expert Twin are intended to support you in developing your application and its configuration by simulating the behavior of the various machine or process components. These functions are not intended to substitute for, but to complement the processes of risk assessment, risk evaluation, validation, and commissioning as well as any ancillary processes, tasks, and obligations according to the applicable regulations and standards such as ISO/EN 13849 and IEC 62061. The product, though powerful, does not, nor can it, simulate every aspect of the application and its environment.

⚠WARNING

INSUFFICIENT TEST COVERAGE

- Do not use EcoStruxure Machine Expert Twin as the sole means for risk assessment, risk evaluation, validation, and commissioning as well as any ancillary processes, tasks, and obligations according to the applicable regulations and standards such as, but not limited to, ISO/EN 13849 and IEC 62061.
- Verify and validate your results on the intended equipment before placing your machine or process into service.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

Based on the system configuration and operation, a hazard and risk analysis must be conducted for the system (for example, according to ISO 12100 or ISO 13849-1) independent of the work with EcoStruxure Machine Expert Twin. The results of this analysis must be considered when designing the machine, and subsequently applying safety-related equipment and safety-related functions. The results of your analysis may deviate from any digital models of physical machines that you may create. For example, additional safety components may be required. In principle, the results from the hazard and risk analysis have priority.

⚠WARNING

NON-CONFORMANCE TO SAFETY FUNCTION REQUIREMENTS

- Specify the requirements and/or measures to be implemented in the risk analysis you perform.
- Verify that your safety-related application complies to applicable safety regulations and standards.
- Make certain that appropriate procedures and measures (according to applicable sector standards) have been established to help avoid hazardous situations when operating the machine.
- Use appropriate safety interlocks where personnel and/or equipment hazards exist.
- Validate the overall safety-related function and thoroughly test the application.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

Catalogs contain important data, parameters and operational aspects of the devices defined within. This information is subject to change over time for a variety of reasons. Therefore, it is necessary to maintain the relationship between the models you create and the catalogs you have used to do so. Version mismatches of catalogs may cause your models to operate in ways that are incongruent with the equipment they represent and may lead to errors in design and operation.

⚠WARNING

UNINTENDED EQUIPMENT OPERATION

- Impose a system of file name conventions that readily indicate the version of the catalogs you use and models you create.
- Create documentation that records catalog and model versions, as well as firmware versions of the equipment used in your models.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

In addition, changes to your underlying application (logic, data address, functions, I/O configurations, device types and configuration, etc.) can have serious impact on the models you have created.

⚠WARNING

UNINTENDED EQUIPMENT OPERATION

- Update your models every time you modify your application or change the physical hardware configuration.
- Verify that objects you have created in your models are coherent with the modifications and/or changes you have made to your application and that they are associated with the correct variables.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

It is also important to connect to the correct automation logic/motion controller in a networked, multi-controller environment.

⚠ WARNING
UNINTENDED EQUIPMENT OPERATION Verify that you have connected to the intended automation controller. **Failure to follow these instructions can result in death, serious injury, or equipment damage.**

It is important to manage the amount of data that is transmitted between your automation logic/motion controller and EcoStruxure Machine Expert Twin. Large amounts of data, or data that is not contiguous in the controller memory may impact performance of EcoStruxure Machine Expert Twin, the controller or both.

Information on Non-Inclusive or Insensitive Terminology

As a responsible, inclusive company, Schneider Electric is constantly updating its communications and products that contain non-inclusive or insensitive terminology. However, despite these efforts, our content may still contain terms that are deemed inappropriate by some customers.

Terminology Derived from Standards

The technical terms, terminology, symbols and the corresponding descriptions in the information contained herein, or that appear in or on the products themselves, are generally derived from the terms or definitions of international standards.

In the area of functional safety systems, drives and general automation, this may include, but is not limited to, terms such as *safety*, *safety function*, *safe state*, *fault*, *fault reset*, *malfunction*, *failure*, *error*, *error message*, *dangerous*, etc.

Among others, these standards include:

Standard	Description
IEC 61131-2:2007	Programmable controllers, part 2: Equipment requirements and tests.
ISO 13849-1:2023	Safety of machinery: Safety related parts of control systems. General principles for design.
EN 61496-1:2013	Safety of machinery: Electro-sensitive protective equipment. Part 1: General requirements and tests.
ISO 12100:2010	Safety of machinery - General principles for design - Risk assessment and risk reduction
EN 60204-1:2006	Safety of machinery - Electrical equipment of machines - Part 1: General requirements
ISO 14119:2013	Safety of machinery - Interlocking devices associated with guards - Principles for design and selection
ISO 13850:2015	Safety of machinery - Emergency stop - Principles for design
IEC 62061:2021	Safety of machinery - Functional safety of safety-related electrical, electronic, and electronic programmable control systems
IEC 61508-1:2010	Functional safety of electrical/electronic/programmable electronic safety-related systems: General requirements.
IEC 61508-2:2010	Functional safety of electrical/electronic/programmable electronic safety-related systems: Requirements for electrical/electronic/programmable electronic safety-related systems.
IEC 61508-3:2010	Functional safety of electrical/electronic/programmable electronic safety-related systems: Software requirements.

Standard	Description
IEC 61784-3:2021	Industrial communication networks - Profiles - Part 3: Functional safety fieldbuses - General rules and profile definitions.
2006/42/EC	Machinery Directive
2014/30/EU	Electromagnetic Compatibility Directive
2014/35/EU	Low Voltage Directive

In addition, terms used in the present document may tangentially be used as they are derived from other standards such as:

Standard	Description
IEC 60034 series	Rotating electrical machines
IEC 61800 series	Adjustable speed electrical power drive systems
IEC 61158 series	Digital data communications for measurement and control – Fieldbus for use in industrial control systems

Finally, the term *zone of operation* may be used in conjunction with the description of specific hazards, and is defined as it is for a *hazard zone* or *danger zone* in the *Machinery Directive* (*2006/42/EC*) and *ISO 12100:2010*.

NOTE: The aforementioned standards may or may not apply to the specific products cited in the present documentation. For more information concerning the individual standards applicable to the products described herein, see the characteristics tables for those product references.

Steps to Create a Digital Model of Your Mechatronic System

Overview

EcoStruxure Machine Expert Twin provides features for visualization, simulation, and emulation of machines and automation lines throughout the complete lifecycle. This user guide describes how to convert parts of your physical machine into digital models that can be made available in EcoStruxure Machine Expert Twin catalogs for use in projects.

Catalogs function as a form of libraries that provide different objects. You can drag these objects into the **Model** view of EcoStruxure Machine Expert Twin to use them as assemblies in your scene. Catalogs are handled as DLL files. The default catalogs are extended by the prefix *Experior.Catalog*, for example *Experior.Catalog.SchneiderElectric.Accessories.dll*.

To create customized catalogs of customized mechatronic systems for the EcoStruxure Machine Expert Twin Builder, the following prerequisites must be fulfilled:

- An EcoStruxure Machine Expert Twin Developer license is available.
- Microsoft Visual Studio is installed to develop customized catalogs in the C# programming language.

Steps

Perform the following steps to create a digital model of your mechatronic system:

1. Identify the different kinds of movements that are performed by your mechatronic system.

   Refer to Identifying the Movements of the Machine, page 14.
2. Split the overall CAD file of the mechatronic system into individual subsystems (using a CAD tool such as Solidworks or PTC Creo) according to the type of motion that is performed.

   To achieve this, identify the parts according to the movement they perform:

   - Movement in X direction.
   - Movement in Y direction.
   - Movement in Z direction.
   - Parts that perform other movements, as, for example, rotations.
   - Parts of the mechatronic system that are not moving. They form the ground, the frame itself.

   Save each subsystem as a separate **.dae* (Collada) file.

   Refer to Extracting CAD Files per Axis, page 15.
3. Identify the parent-child relationships that exist between the subsystems, which are represented by the different **.dae* files you created, and schematize them in a hierarchical concept.

   Refer to Creating a Hierarchical Concept of Parent-Child Relations, page 19.
4. Program the catalog in C# using Microsoft Visual Studio.

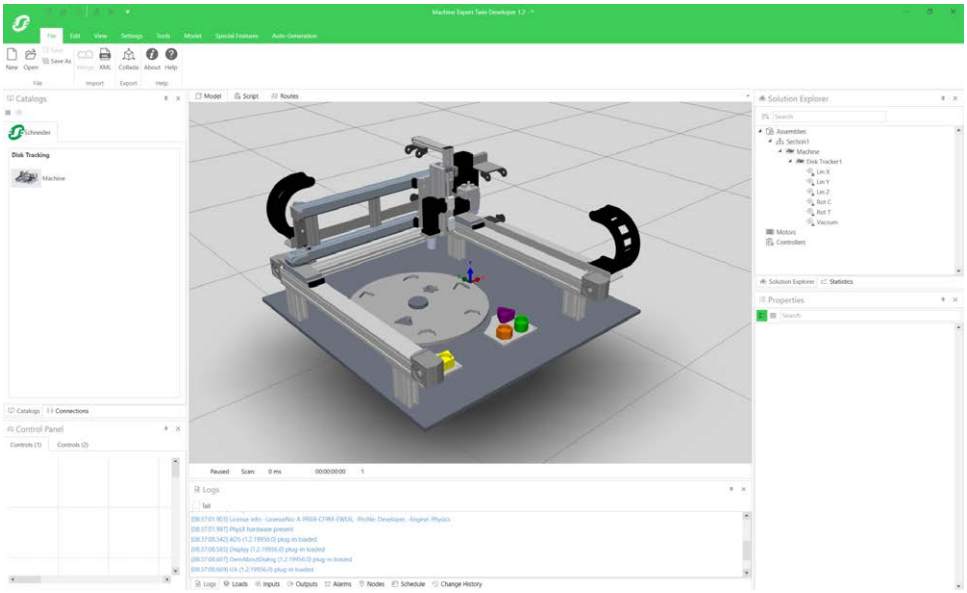
   Refer to Programming in Microsoft Visual Studio, page 22.
5. Create the parent-child relationships by kinematizing the individual CAD elements to define which parts of the mechatronic system are moving in relation to one another.

   Refer to Programming Relations, page 23.

Example of a Disk Tracking Machine

The photo displays the Schneider Electric disk tracking machine. It is used as an example in this user guide to describe the steps that need to be performed to create a digital model of this physical machine that can be made available in EcoStruxure Machine Expert Twin catalogs for use in projects.

The screenshot displays the goal of the process. After the steps explained in this user guide have been performed, the disk tracking machine is available as an object in an EcoStruxure Machine Expert Twin catalog and can be used in projects.



Identifying the Movements of the Machine

Overview

As a first step in creating a digital twin model, understand your mechatronic system and identify the different kinds of movements that are performed.

Example: Identifying the Movements of the Disk Tracking Machine

The disk tracking machine consists of the following components:

- Cartesian robot that performs motion in X, Y and Z direction.
- Rotating gripper
- Rotating disk

It thus contains five axes of movement.

Before you proceed with the next step, identify the following:

- Parts of the mechatronic system that belong to the same axis.
- Parent-child relationships that exist between the different axes.

Example of parent-child relationships:

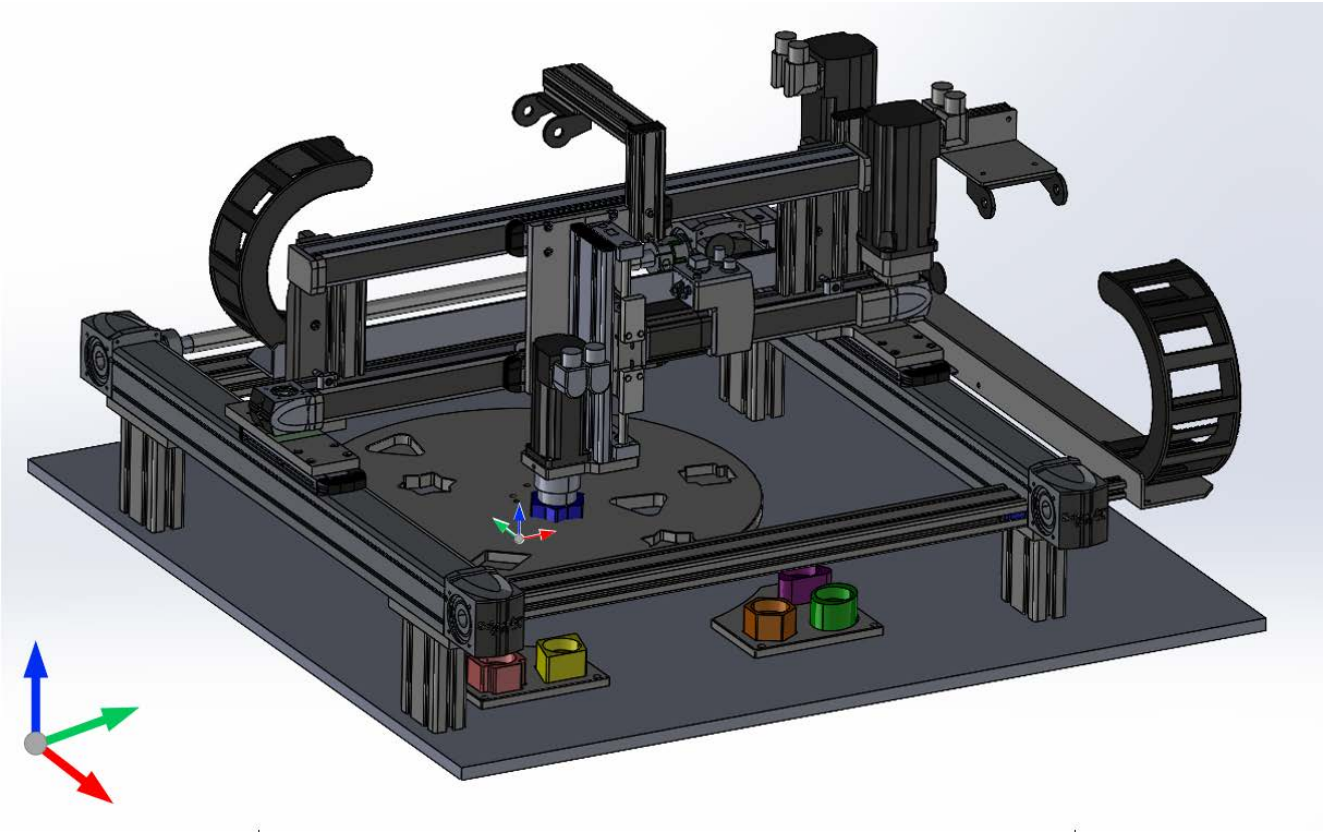
If the...	Then the...
X axis of the cartesian robot is moving,	Y and Z axes are moved together with the X axis.
Y axis is moving,	X axis is not affected but the Z axis is moved together with the Y axis.

Extracting CAD Files per Axis

Overview

As a second step in creating a digital model, split the overall CAD file of the mechatronic system into individual files according to the movement axis. To edit the overall CAD file, use a CAD tool such as Solidworks or PTC Creo.

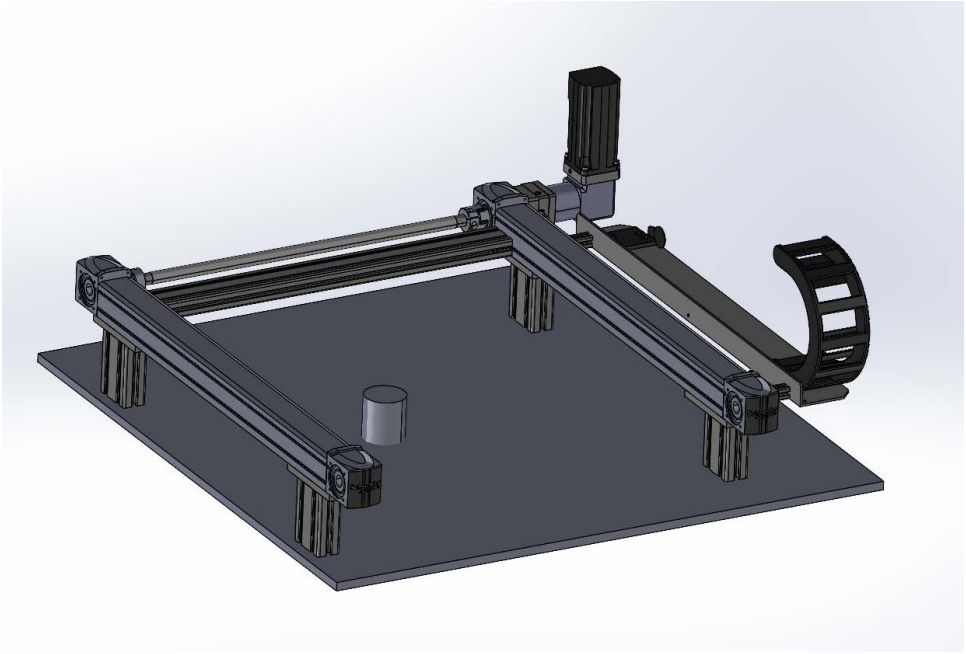
Overall CAD File of the Disk Tracking Machine



In the overall CAD file of the disk tracking machine, identify the parts that belong to the same axis of movement. Extract the selected parts and save each group to an individual CAD (Collada) file with the file extension *.dae. For a list of CAD files that result from this disk tracking machine example, refer to Resulting CAD Files, page 19.

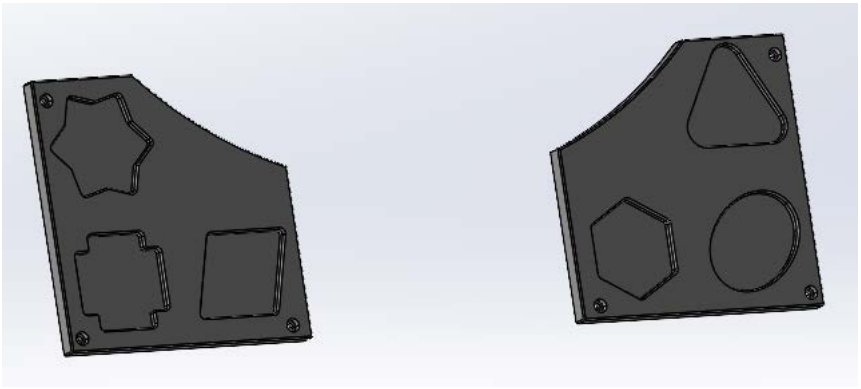
CAD File for the Ground

Identify the static components and save them as a separate file, in our example, *Ground.dae*.



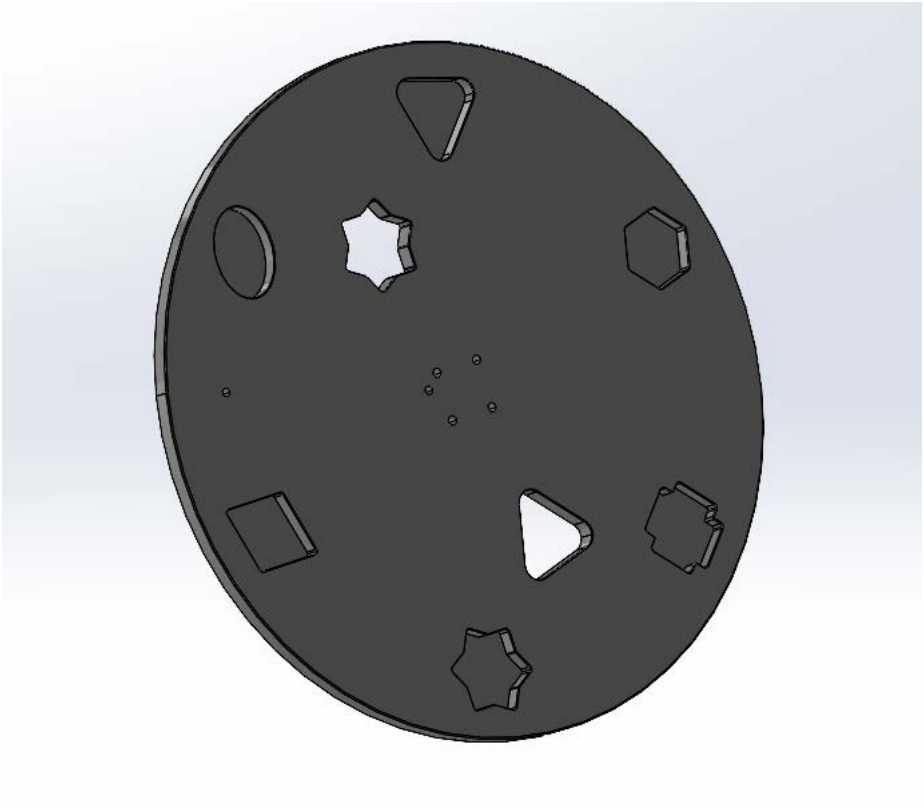
CAD Files for the Load Holders

Identify the components of the load holders that are not moving. They provide the space for loads that are not on the rotating disk. Save them as a separate file, in our example, *Holder.dae*.



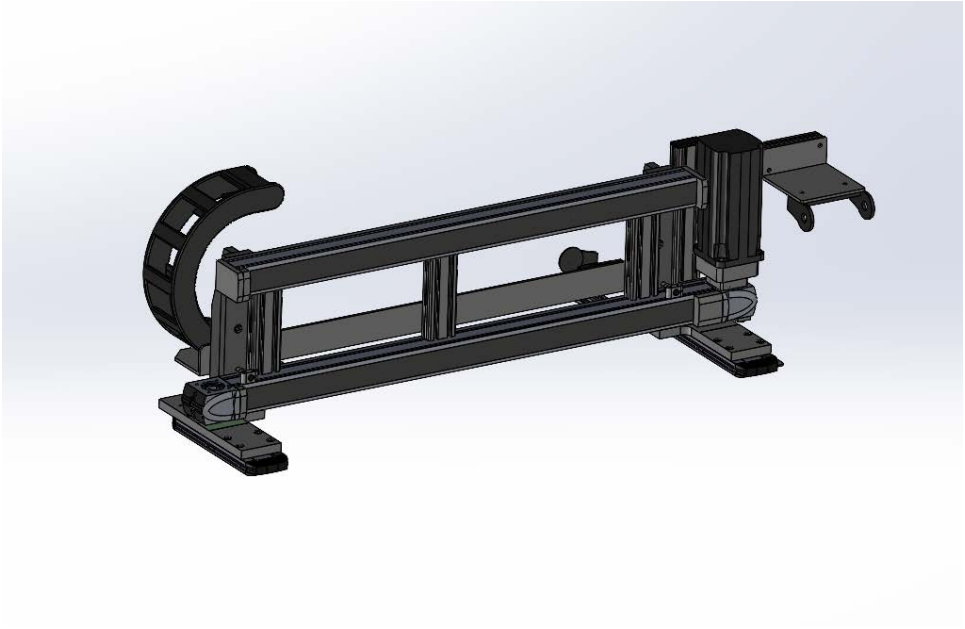
CAD File for the Rotating Disk

Identify components that are part of the rotating disk and save them as a separate file, in our example, *Disk.dae*.



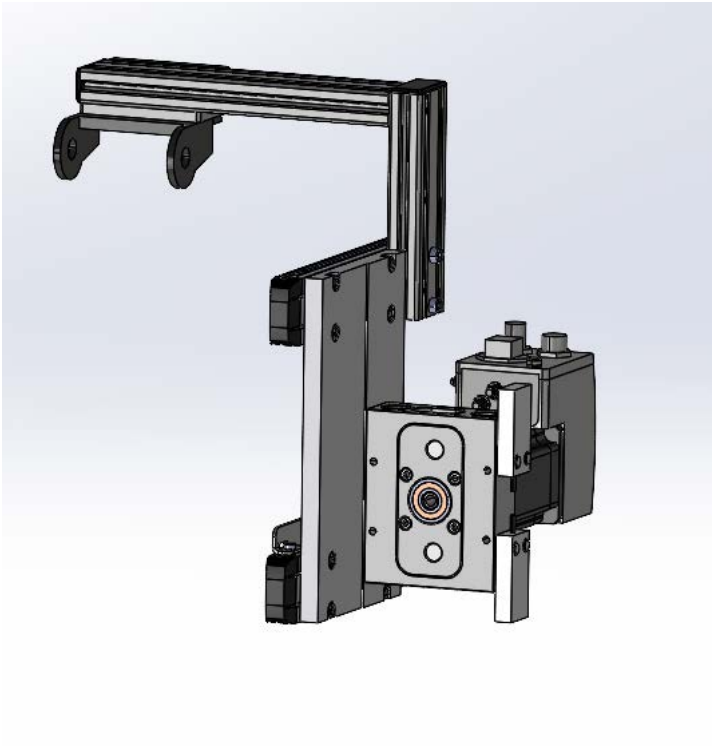
CAD File for the X Axis

Identify the X axis components of the cartesian robot and save them as a separate file, in our example, *Axis1.dae*.



CAD File for the Y Axis

Identify the Y axis components of the cartesian robot and save them as a separate file, in our example, *Axis2.dae*.



CAD File for the Z Axis

Identify the Z axis components of the cartesian robot and save them as a separate file, in our example, *Axis3.dae*.

Resulting CAD Files

In our example, we have created six individual CAD files. One for the ground and one for the load holders that are not moving and four for different axes of movement:

- *Ground.dae*
- *Holder.dae*
- *Disk.dae*
- *Axis1.dae* (of the cartesian portal)
- *Axis2.dae* (of the cartesian portal)
- *Axis3.dae* (of the cartesian portal)

To see how the CAD files are implemented in the programming code, refer to *DiskTracking.cs*, page 26.

Creating a Hierarchical Concept of Parent-Child Relations

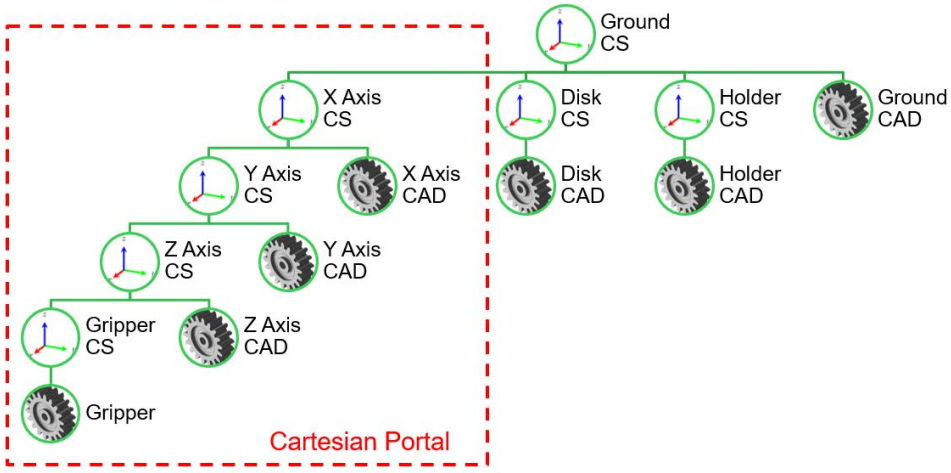
Overview

The parent-child relationships that exist in your mechatronic system between the different axes of movement are reflected in EcoStruxure Machine Expert Twin by coordinate systems. Use a coordinate system as a starting point that forms the ground, the frame itself, and assign one coordinate system as child. Proceed with adding further coordinate systems by creating more parent-child relationships.

Before you start programming your digital model in C#, create a hierarchical concept of the parent-child relationships representing your mechatronic system.

Hierarchy of the Disk Tracking Machine

The five axes of movement that have been identified for the disk tracking machine can be represented in the following hierarchy using coordinate systems for creating parent-child relations between the CAD files representing the components for the different directions of movement:



Catalog Programming in C#

Overview

This chapter lists the prerequisites and the actions that need to be performed. For the programming code of the disk tracking machine example, refer to the Appendix: Programming Code, page 26.

Prerequisites

For programming customized catalogs in C# programming language that represent your mechatronic system, the following prerequisites must be fulfilled:

- An EcoStruxure Machine Expert Twin Developer license must be available. It provides access to the EcoStruxure Machine Expert Twin suite and the underlying .NET framework. For developing catalogs in Microsoft Visual Studio, a C# project template is provided.
- Microsoft Visual Studio must be installed.

Catalog Template Configuration

Overview

Use the Machine Expert Twin Catalog Template that is provided with the EcoStruxure Machine Expert Twin installation, for programming customized catalogs.

Installing the Machine Expert Twin Catalog Template for Microsoft Visual Studio

Copy the *Machine Expert Twin Catalog Template.zip* file from the subfolder *Templates* of the EcoStruxure Machine Expert Twin installation folder and add it to the project templates of Microsoft Visual Studio by pasting it in the *Users \Documents* path in the following folder:

[...]\Documents\Visual Studio XXXX\Templates\Project Templates\Visual C#

Project Configuration in Microsoft Visual Studio

To configure a new project in Microsoft Visual Studio for programming your catalog, proceed as follows:

Step	Action
1	Start Microsoft Visual Studio and execute the command **Create a new project**.
2	In the **Recent project templates** step, browse for the Machine Expert Twin Catalog Template that you have saved to the Microsoft Visual Studio **Templates** folder as described in Installing the Machine Expert Twin Catalog Template for Microsoft Visual Studio, page 20. Select it and click **Next**.
3	In the **Configure your new project** step: • Enter a **Project name** that must be extended by the prefix *Experior.Catalog*, for example *Experior.Catalog*. • Browse to the **Location** where you want to save the project. • Click the **Create** button.
4	In the **Solution Explorer** view, right-click the **Experior.Catalog.DiskTrackingModel** project node and execute the command **Properties** from the contextual menu.
5	Select the **Application** tab of the project properties and perform the following actions: • Copy the content of the field **Default namespace** that contains the name you entered as **Project name** in step 3 (*Experior.Catalog.DiskTrackingModel* in this example) and paste it into the field **Assembly name**. As this is the name that will be displayed in EcoStruxure Machine Expert Twin for the catalog, page 25, ensure to replace the default name by a unique name. • From the **Target framework:** list, select the entry **.NET Framework 4.8.1**.
6	Select the **Build** tab of the project properties. In the **Output** section, click the **Browse...** button to browse to the **Output path**: *C:\Program Files\Schneider Electric\EcoStruxure Machine Expert Twin*
7	Select the **Debug** tab of the project properties. In the **Start action** section, click the **Browse...** button of the **Start external program** and browse to the following path: *C:\Program Files\Schneider Electric\EcoStruxure Machine Expert Twin \MachineExpertTwin.exe*

Importing CAD Files to Microsoft Visual Studio

Overview

To make the individual CAD files that you have created from your mechatronic system as described in Extracting CAD Files per Axis, page 15 available in Microsoft Visual Studio, proceed as follows:

Step	Action
1	In the **Solution Explorer**, right-click the **Mesh** folder and execute the command **Add > Existing Item...** from the contextual menu.
2	Select the option **All Files *.*** from the list on the left-hand side to display the Collada files with the file extension *.dae and browse to the folder where you stored the CAD files that you have created of your mechatronic system.
3	Select the *.dae file or files you want to import from this folder and click the **Add** button. **Result**: The *.dae files you selected are displayed as subitems of the **Mesh** folder in the **Solution Explorer**.
4	Select the *.dae file or files in the **Solution Explorer** and set the property **Advanced > Build Action** that is displayed in the lower part of the **Solution Explorer** to **Embedded Resource**.

Programming in Microsoft Visual Studio

Overview

It is a good practice to structure the programming code in the following regions:

- `Fields`

  Use this region for variable declarations.

- `Constructor`

  Use this region to construct the assembly and to add CAD files (meshes).

- `Public Properties`

  Use this region to configure the properties that are displayed in the **Properties** view of EcoStruxure Machine Expert Twin (for further information, refer to the EcoStruxure Machine Expert Twin Getting Started User Guide).

- `Public Methods`

  Use this region to configure the methods that are accessible both inside and outside the scope of your class.

- `Private Methods`

  Use this region to configure the methods that can only be accessed inside the scope of your class.

Variable Declarations

As a first programming step in Microsoft Visual Studio, declare variables for the CAD files, the coordinate systems and the loads in section:
`public class MyAssembly : Assembly`

- Variables for CAD files:

  For each CAD file you have imported to the **Mesh** folder as described in Importing CAD Files to Microsoft Visual Studio, page 21, declare a variable under the data type:

  ```
  private Experior.Core.Parts.Model
  ```

- Variables for coordinate systems:

  For each coordinate system you have identified in the step Creating a Hierarchical Concept of Parent-Child Relations, page 19, declare a variable under the data type:

  ```
  private Experior.Core.Assemblies.CoordinateSystem
  ```

  It is a good practice to use the same names as for the CAD files extended by a suffix such as `_cs`.

- Variables for loads:

  For each type of load (for example, boxes or cylinders), declare a variable under the data type:

  ```
  Experior.Core.Loads.Load
  ```

To see the implementation in the programming code, refer to *DiskTracking.cs*, page 26.

Programming Relations

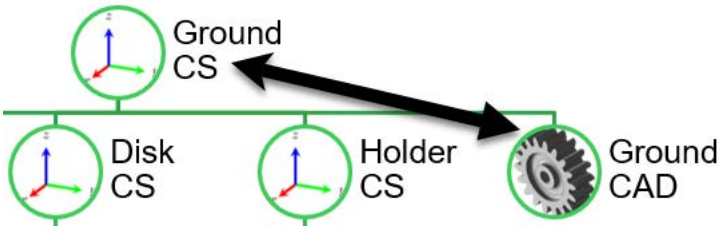
Overview

Program the relationships between variables and coordinate systems that have been identified in the step Creating a Hierarchical Concept of Parent-Child Relations, page 19.

To see the implementation in the programming code, refer to *DiskTracking.cs*, page 26.

Adding a CAD File as Sub Component to a Coordinate System

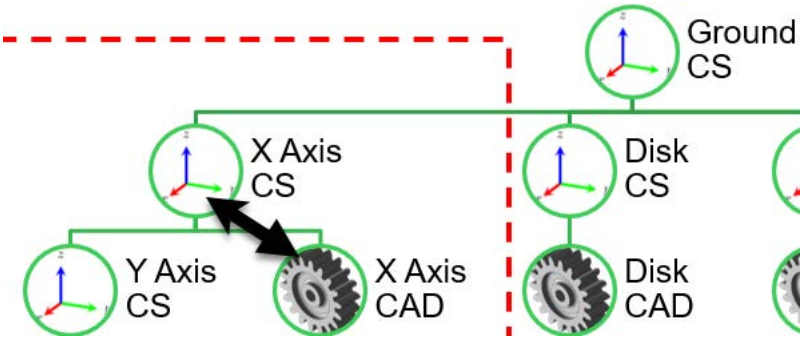
In the following example code, the CAD file *Ground.dae* is added as sub component to the ground coordinate system `ground_cs` that corresponds to the relationship marked in the section of the Hierarchy of the Disk Tracking Machine, page 19.



Example code:

```
ground_cs.Add(ground);
```

As a second example, the relationship between the `axis1_cs` and the CAD file *Axis1.dae* is created:



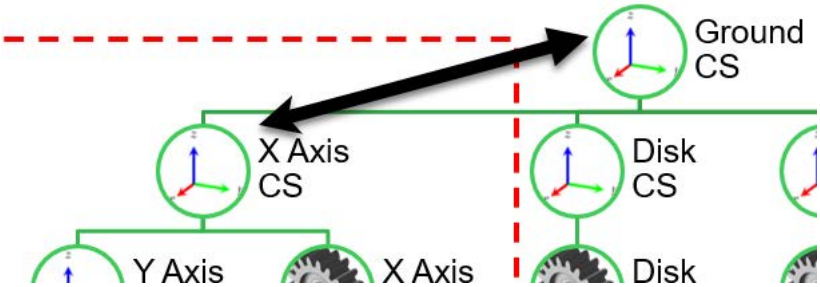
Example code:

```
axis1_cs.Add(axis1);
```

Proceed like this for all relationships between variables and coordinate systems that have been identified in the step Creating a Hierarchical Concept of Parent-Child Relations, page 19.

Adding a Coordinate System as Sub Component of a Coordinate System

In the following example code, the coordinate system `axis1_cs` is added as sub component to the ground coordinate system `ground_cs` that corresponds to the relationship marked in the section of the Hierarchy of the Disk Tracking Machine, page 19.



Example code:

```
ground_cs.AddSubSystem(axis1_cs, new System.Numerics.Vector3
(0,0,0));;
```

Use `AddSubSystem` and add the name of the second coordinate system. The 3-dimensional vector datatype `Vector3` is a mandatory parameter to configure an offset between the two coordinate systems. To define no offset, enter `(0,0,0)` for the `x`, `y` and `z` values.

Proceed like this for all relationships between two coordinate systems that have been identified in the step Creating a Hierarchical Concept of Parent-Child Relations, page 19.

For the programming example of the disk tracking machine, refer to the Appendix: Programming Code, page 26.

Building and Debugging the Catalog

Overview

After you have declared and configured the variables and programmed the relations, you can build the catalog and start EcoStruxure Machine Expert Twin in debugging mode. The debugging mode allows you to perform modifications in Microsoft Visual Studio and verifying the effects in EcoStruxure Machine Expert Twin. As a prerequisite, an EcoStruxure Machine Expert Twin Developer license must be available.

Building the Catalog Project

To build the project, proceed as follows:

Step	Action
1	Click the **Start** button in the toolbar of Microsoft Visual Studio. This leads to two results: • A new catalog DLL file is created in the EcoStruxure Machine Expert Twin installation folder. The file is named according to the string you entered for the parameters **Project name** and **Default namespace** in Project Configuration in Microsoft Visual Studio, page 21. In our example: *Experior.Catalog.DiskTrackingModel.dll* • EcoStruxure Machine Expert Twin starts and the **Select Catalog(s)** dialog box provides the new catalog Experior.Catalog.DiskTrackingModel for selection.
2	Select the check box of theExperior.Catalog.DiskTrackingModel in the **Select Catalog (s)** dialog box and click the **OK** button to continue starting EcoStruxure Machine Expert Twin in debug mode and providing this new catalog in the **Catalogs** view. **Result**: The model of the disk tracking machine is provided as an object of the new catalog.
3	To use it in your EcoStruxure Machine Expert Twin project, drag the object from the catalog and drop it to the scene.

For further information on the **Catalogs** view, refer to the *EcoStruxure Machine Expert Twin Getting Started User Guide*.

Debugging the Catalog

As EcoStruxure Machine Expert Twin has been started in debug mode, you can verify the content of the catalog in EcoStruxure Machine Expert Twin and return to Microsoft Visual Studio to perform modifications. Whenever you click the **Start** or the **Hot Reload** button in Microsoft Visual Studio, the modifications performed in Microsoft Visual Studio will become available in EcoStruxure Machine Expert Twin for you to verify.

Appendix: Programming Code

This appendix provides the programming code of the disk tracking machine in three separate code files:

- *DiskTracking.cs*, page 26

  The main class `DiskTracking` is defined as a partial class and is used in the *Load.cs* and *Motion.cs* code files.

- *Load.cs*, page 34

  Contains the C# script for handling loads, that are the methods to create and position loads.

- *Motion.cs*, page 37

  Contains the methods that handle the motion of the assemblies based on controller signals.

DiskTracking.cs

Overview

In the *DiskTracking.cs*, the main class `DiskTracking` is defined as a partial class. The `DiskTracking` partial class is then used in the *Load.cs* and *Motion.cs* code files.

Code Example

```
using Experior.Core.Assemblies;
using Experior.Core.Communication.PLC;
using Experior.Core.Mathematics;
using Experior.Core.Parts;
using Experior.Core.Parts.Sensors;
using Experior.Core.Properties;
using Experior.Interfaces;
using System;
using System.ComponentModel;
using System.Numerics;
using System.Windows.Media;
using System.Xml.Serialization;
namespace Experior.Catalog.SchneiderElectric.DiskTracking.Assemblies
{
    public partial class DiskTracking : Assembly
    {
        #region Fields
        private DiskTrackingInfo info;

        //Declare the mesh variables
        private Experior.Core.Parts.Model ground, axis1, axis2, axis3, axis4,
disk, holder, dummyDisk, dummyHolder;

        //private Experior.Core.Parts.Cylinder dummyDisk;

        //Declare coordinate system variables
        private Experior.Core.Assemblies.CoordinateSystem ground_cs, axis1_cs,
axis2_cs, axis3_cs, axis4_cs, disk_cs;

        // Declare load variables
        Experior.Core.Loads.Load star, square, triangle, hexagon, oval,
parallelogram, load_1;
```

```
        // Declare the gripper variables
        private Experior.Core.Parts.Sensors.Box gripper;
        private bool GripperActive = false;


        #region Global Variables
        // offset values (zero positions with respect to coordinates)
        // Values calculated for the CAD model
        public double x_offset = 0.265f;
        public double y_offset = 0.053f;
        public double z_offset = 0.23009f;
        public double c_offset = 0.0f;
        public double t_offset = 8.2f;


        public double targetPositionX = 0;
        private double targetPositionY = 0;
        private double targetPositionZ = 0;
        private double targetPositionC = 0;
        public double targetPositionT = 0;
        #endregion

        #endregion

        #region Constructor
        public DiskTracking(DiskTrackingInfo info): base(info)
        {
            this.info = info;

            // Define the mesh variables and add the meshes(machine parts) to the
environment
            ground = new Model(Common.EmbeddedResourceLoader.Get("Ground.dae"));
            Add(ground);

            axis1 = new Model(Common.EmbeddedResourceLoader.Get("Axis1.dae"));
            Add(axis1);

            axis2 = new Model(Common.EmbeddedResourceLoader.Get("Axis2.dae"));
            Add(axis2);

            axis3 = new Model(Common.EmbeddedResourceLoader.Get("Axis3.dae"));
            Add(axis3);

            axis4 = new Model(Common.EmbeddedResourceLoader.Get("Gripper.dae"));
            Add(axis4);

            disk = new Model(Common.EmbeddedResourceLoader.Get("Disk.dae")) {
Color = Colors.DarkGray, Rigid = true };// disk is set to be rigid, to use the
attach properts of loads
            Add(disk);
            disk.Visible = false;

            dummyDisk = new Model(Common.EmbeddedResourceLoader.Get("Disk.dae")) {
Color =Colors.DarkGray, Rigid = false };//new Experior.Core.Parts.Cylinder(Color =
Colors.Red, 0.01f, 0.25f, 1000) { Rigid = true };
            Add(dummyDisk);
            dummyDisk.Visible = true;
            //dummyDisk.Color = Colors.;

            holder = new Model(Common.EmbeddedResourceLoader.Get("Holder.dae")) {
Rigid = true }; // holder is set to be rigid, to use the attach properts of loads
            Add(holder);
            holder.Visible = false;

            dummyHolder = new Model(Common.EmbeddedResourceLoader.Get("Holder.
dae")) { Rigid = false }; // holder is set to be rigid, to use the attach properts
of loads
            Add(dummyHolder);
```

```
            dummyHolder.Visible = true;


            // creating coordinate systems
            ground_cs = new CoordinateSystem();     // define the coordinate
system
            Add(ground_cs);                         // ADD Coordinate system to
the scene
            ground_cs.Add(ground);                  // add the ground mesh to the
coordinate system

            axis1_cs = new CoordinateSystem();
            Add(axis1_cs);                          // ADD Coordinate system to
the scene

            axis2_cs = new CoordinateSystem();
            Add(axis2_cs);                          // ADD Coordinate system to
the scene

            axis3_cs = new CoordinateSystem();
            Add(axis3_cs);                          // ADD Coordinate system to
the scene

            axis4_cs = new CoordinateSystem();
            Add(axis4_cs);                          // ADD Coordinate system to
the scene

            disk_cs = new CoordinateSystem();
            Add(disk_cs);                           // ADD Coordinate system to
the scene

            // set visibility of cs
            ground_cs.VisibleAxes = CoordinateSystem.VisibleAxesTypes.NotVisible;
            axis1_cs.VisibleAxes = CoordinateSystem.VisibleAxesTypes.NotVisible;
            axis2_cs.VisibleAxes = CoordinateSystem.VisibleAxesTypes.NotVisible;
            axis3_cs.VisibleAxes = CoordinateSystem.VisibleAxesTypes.NotVisible;
            axis4_cs.VisibleAxes = CoordinateSystem.VisibleAxesTypes.NotVisible;
            disk_cs.VisibleAxes = CoordinateSystem.VisibleAxesTypes.NotVisible;

            // Creating the parent child relationship
            // adding one coordinate system as subsystem of another one
            ground_cs.AddSubSystem(axis1_cs, new Vector3(0.0f, 0.0f, 0.0f));
// axis1_cs is added as a subsystem of ground_cs
            axis1_cs.AddSubSystem(axis2_cs, new Vector3(0.0f, 0.0f, 0.0f));
// axis2_cs is added as a subsystem of axis1_cs
            axis2_cs.AddSubSystem(axis3_cs, new Vector3(0.0f, 0.0f, 0.0f));
            axis3_cs.AddSubSystem(axis4_cs, new Vector3(0.0f, 0.0f, 0.0f));
            ground_cs.AddSubSystem(disk_cs, new Vector3(0.0f, 0.0f, 0.0f));

            // Linking the meshes to the repective coordinate systems
            ground_cs.Add(ground);
            ground_cs.Add(dummyHolder, new Vector3(0.0f,0.0f, 0.010f));
            ground_cs.Add(holder, new Vector3(0.0f, 0.0f, 0.005f));

            axis1_cs.Add(axis1);
            axis2_cs.Add(axis2);
            axis3_cs.Add(axis3);

            axis4_cs.Add(axis4, new Vector3(0.0f, -0.05589f, 0.07f));      //
offset calculated from CAD model
            axis4.LocalRoll = -(float)Math.PI / 2;
            disk_cs.Add(dummyDisk, new Vector3(0.0f, 0.0f, -0.025f));
            disk_cs.Add(disk, new Vector3(0.0f,0.0f, -0.037f)) ;


            // Creating the gripper and adding it to the scene
            gripper = new Experior.Core.Parts.Sensors.Box(Colors.LightGray, 0.05f,
0.05f, 0.05f) {  Collision = Collisions.Loads }; // old radius 0.021 0.031
```

```
            Add(gripper);
            axis4_cs.Add(gripper, new Vector3(0, -0.05589f, 0.0305f));      //
offset calculated from CAD model
            gripper.LocalPitch = -(float)Math.PI / 2;
            gripper.Visible = false;

            #region PLC Output Variables

            if (info.inputposition_x == null)
                info.inputposition_x = new Input() { DataSize = DataSize.LREAL,
Description = "Reference Position X",SymbolName = "Lin X"  };

            if (info.inputposition_y == null)
                info.inputposition_y = new Input() { DataSize = DataSize.LREAL,
Description = "Reference Position Y", SymbolName = "Lin Y" };

            if (info.inputposition_z == null)
                info.inputposition_z = new Input() { DataSize = DataSize.LREAL,
Description = "Reference Position Z", SymbolName = "Lin Z" };

            if (info.inputposition_c == null)
                info.inputposition_c = new Input() { DataSize = DataSize.LREAL,
Description = "Rotation of Gripper around Z axis", SymbolName = "Rot C" };

            if (info.inputposition_t == null)
                info.inputposition_t = new Input() { DataSize = DataSize.LREAL,
Description = "Rotation of Disk", SymbolName = "Rot T" };

            if (info.vaccum == null)
                info.vaccum = new Input() { DataSize = DataSize.BOOL, Description
= "Gripper Vaccum", SymbolName = "Vaccum" };

            Add(info.inputposition_x);
            Add(info.inputposition_y);
            Add(info.inputposition_z);
            Add(info.inputposition_c);
            Add(info.inputposition_t);
            Add(info.vaccum);

            // subscribe to the event to detect change of inputs
            info.inputposition_x.OnReceived += InputPositionX_OnReceived;
            info.inputposition_y.OnReceived += InputPositionY_OnReceived;
            info.inputposition_z.OnReceived += InputPositionZ_OnReceived;

            info.inputposition_c.OnReceived += InputPositionC_OnReceived;
            info.inputposition_t.OnReceived += InputPositionT_OnReceived;

            info.vaccum.On += Vaccum_On;
            info.vaccum.Off += Vaccum_On;

            #endregion

            #region Delegate functions
            // subscribe to the event; collision of loads with gripper(loads
entering gripper)
            gripper.OnEnter += Magnet_OnEnter;

            gripper.OnLeave += Gripper_OnLeave;

            // subscribe to the event; collision of loads with disk
            disk.OnContact += Disk_OnContact;

            //
            holder.OnContact += Holder_OnContact;


            #endregion
            // Reset used for initializing the model
```

```
        Reset();
    }

    #endregion

    #region PLC Output Signals
    [Category("PLC Output Signals")]
    [DisplayName(@"X Position")]
    [PropertyOrder(0)]
    public Input InputPosition_X
    {
        get => info.inputposition_x;
        set => info.inputposition_x = value;
    }

    [Category("PLC Output Signals")]
    [DisplayName(@"Y Position")]
    [PropertyOrder(1)]
    public Input InputPosition_Y
    {
        get => info.inputposition_y;
        set => info.inputposition_y = value;
    }

    [Category("PLC Output Signals")]
    [DisplayName(@"Z Position")]
    [PropertyOrder(2)]
    public Input InputPosition_Z
    {
        get => info.inputposition_z;
        set => info.inputposition_z = value;
    }

    [Category("PLC Output Signals")]
    [DisplayName(@"C Position")]
    [PropertyOrder(3)]
    public Input InputPosition_C
    {
        get => info.inputposition_c;
        set => info.inputposition_c = value;
    }

    [Category("PLC Output Signals")]
    [DisplayName(@"T Position")]
    [PropertyOrder(4)]
    public Input InputPosition_T
    {
        get => info.inputposition_t;
        set => info.inputposition_t = value;
    }

    [Category("PLC Output Signals")]
    [DisplayName(@"Gripper Vaccum")]
    [PropertyOrder(4)]
    public Input Vaccum
    {
        get => info.vaccum;
        set => info.vaccum = value;
    }

    #endregion

    #region Public Properties

    [Browsable(false)]
    public override float Yaw { get => base.Yaw; set => base.Yaw = 0; }

    [Browsable(false)]
```

```
        public override float Pitch { get => base.Pitch; set => base.Pitch = 0; }

        [Browsable(false)]
        public override float Roll { get => base.Roll; set => base.Roll = 0; }

        #endregion

        #region Public Methods

        public override void Step(float deltatime)
        {
            base.Step(deltatime);

        }
        public override void Reset()
        {
            Experior.Core.Environment.InvokeIfRequired(() =>
            {
                axis1_cs.LocalPosition = new Vector3(0, (float)x_offset, 0);
                axis2_cs.LocalPosition = new Vector3((float)y_offset, 0, 0);
                axis3_cs.LocalPosition = new Vector3(0, 0, (float)z_offset);

                gripper.LocalYaw = 0;
                //disk.LocalYaw = 0;

                disk.LocalYaw = (float)t_offset;
                dummyDisk.LocalYaw = disk.LocalYaw;

                targetPositionX = 0;
                targetPositionY = 0;
                targetPositionZ = 0;
                targetPositionC = 0;
                targetPositionT = 0;

                base.Reset();

                // insert the loads back into the scene
                // done by invoking the function InsertLoad(), from the partial
class Load.cs
                Experior.Core.Environment.Invoke(InsertLoad);

            });
        }
        public override void Dispose()
        {
            // unsubscribe from the events on deleting the model
            info.inputposition_x.OnReceived -= InputPositionX_OnReceived;
            info.inputposition_y.OnReceived -= InputPositionY_OnReceived;
            info.inputposition_z.OnReceived -= InputPositionZ_OnReceived;

            info.inputposition_c.OnReceived -= InputPositionC_OnReceived;
            info.inputposition_t.OnReceived -= InputPositionT_OnReceived;

            info.vaccum.On -= Vaccum_On;
            info.vaccum.Off -= Vaccum_On;

            gripper.OnEnter -= Magnet_OnEnter;

            gripper.OnLeave -= Gripper_OnLeave;

            disk.OnContact -= Disk_OnContact;

            holder.OnContact -= Holder_OnContact;

            base.Dispose();

            // removing all the loads from the scene on deleting the machine
```

```
            // done by invoking the function DisposeLoads(), from the partial
class Load.cs
            Experior.Core.Environment.Invoke(DisposeLoads);
        }

        public override string Category { get; } = "Machine";

        public override ImageSource Image { get; } = Common.EmbeddedImageLoader?.
Get("machine");

        #endregion

        #region Private Methods
        private void InputPositionX_OnReceived(Input sender, object value)
        {
            // Position unit received: mm
            targetPositionX = (float)x_offset - (System.Convert.ToDouble
(InputPosition_X.Value)) / 1000;

            Experior.Core.Environment.Invoke(Move_Linear_X);
        }
        private void InputPositionY_OnReceived(Input sender, object value)
        {
            // Position unit received: mm
            targetPositionY = (float)y_offset - (System.Convert.ToDouble
(InputPosition_Y.Value)) / 1000;

            Experior.Core.Environment.Invoke(Move_Linear_Y);
        }
        private void InputPositionZ_OnReceived(Input sender, object value)
        {
            // Position unit received: mm
            targetPositionZ = (float)z_offset - (System.Convert.ToDouble
(InputPosition_Z.Value)) / 1000;

            Experior.Core.Environment.Invoke(Move_Linear_Z);
        }

        private void InputPositionC_OnReceived(Input sender, object value)
        {
            targetPositionC = (float)c_offset - (System.Convert.ToDouble
(InputPosition_C.Value));

            Experior.Core.Environment.Invoke(Move_C);
        }

        private void InputPositionT_OnReceived(Input sender, object value)
        {
            targetPositionT = (float)t_offset - (System.Convert.ToDouble
(InputPosition_T.Value));

            Experior.Core.Environment.Invoke(Move_T);
        }

        private void Vaccum_On(Input sender)
        {
            if (Vaccum.Active)
            {
                GripperActive = true;
                ActivateGripper(load_1);
            }
            else
            {
                GripperActive = false;
                // checking if any loads are attached to the gripper
                if (gripper.Attached.Count >= 1)
                {
                    // identifying the first load attached to the gripper
```

```
                var load = gripper.Attached[0];
                gripper.UnAttach();
            }
        }
    }
    private void Magnet_OnEnter(Sensor sensor, object trigger)
    {
        load_1 = trigger as Experior.Core.Loads.Load;
        if (GripperActive)
        {
            ActivateGripper(load_1);
        }
        else
            return;
    }

    private void ActivateGripper(Experior.Core.Loads.Load load)
    {
        if (GripperActive)
             gripper.Attach(load);

        if (gripper.Attached.Count >= 1)
            load_1 = null;

    }
    private void Gripper_OnLeave(Sensor sensor, object trigger)
    {
        load_1 = null;
    }

    private bool Disk_OnContact(Static sender, Core.Loads.Load load)
    {
        if (sender.Attached.Contains(load))
            return false;
        else
        {
            AttachLoad_Disk(load);
            return true;
        }
    }
    private bool Holder_OnContact(Static sender, Core.Loads.Load load)
    {
        if (sender.Attached.Contains(load))
            return false;
        else
        {
            AttachLoad_Holder(load);
            return true;
        }
    }

    #endregion
}

[Serializable, XmlInclude(typeof(DiskTrackingInfo)), XmlType(TypeName =
"Experior.Catalog.SchneiderElectric.DiskTracking.Assemblies.DiskTrackingInfo")]
public class DiskTrackingInfo : Experior.Core.Assemblies.AssemblyInfo
{
    public double xpos = 0;
    public double ypos = 0;
    public double zpos = 0;

    public Input inputposition_x = null;
    public Input inputposition_y = null;
    public Input inputposition_z = null;
    public Input inputposition_c = null;
    public Input inputposition_t = null;
    public Input vaccum = null;
```

```
    }
}
```

Load.cs

Overview

The *Load.cs* contains the C# script for handling loads, that are the methods to create and position loads.

Code Example

```
using Experior.Core.Mathematics;
using Experior.Interfaces;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Numerics;
using System.Text;
using System.Threading.Tasks;
namespace Experior.Catalog.SchneiderElectric.DiskTracking.Assemblies
{
    public partial class DiskTracking
    {
        #region Load functions
        public void InsertLoad()
        {
            // Defining/ creating the load from mesh
            star = Experior.Core.Loads.Load.Create(Common.EmbeddedResourceLoader.
Get("Star.dae"));
            star.Rigid = Rigids.Box;

            square = Experior.Core.Loads.Load.Create(Common.
EmbeddedResourceLoader.Get("Square.dae"));
            square.Rigid = Rigids.Box;

            parallelogram = Experior.Core.Loads.Load.Create(Common.
EmbeddedResourceLoader.Get("Parallelogram.dae"));
            parallelogram.Rigid = Rigids.Box;

            hexagon = Experior.Core.Loads.Load.Create(Common.
EmbeddedResourceLoader.Get("Hexagon.dae"));
            hexagon.Rigid = Rigids.Box;

            triangle = Experior.Core.Loads.Load.Create(Common.
EmbeddedResourceLoader.Get("Triangle.dae"));
            triangle.Rigid = Rigids.Box;

            oval = Experior.Core.Loads.Load.Create(Common.EmbeddedResourceLoader.
Get("Oval.dae"));
            oval.Rigid = Rigids.Box;

            PositionStar();
            PositionSquare();
            PositionParallelogram();
            PositionHexagon();
            PositionTriangle();
            PositionOval();

            //PositionStar_Disk();
            //PositionSquare_Disk();
            //PositionParallelogram_Disk();
```

```
            //PositionHexagon_Disk();
            //PositionTriangle_Disk();
            //PositionOval_Disk();
        }
        public void DisposeLoads()
        {
            // Removing loads from the scene
            star.Dispose();
            square.Dispose();
            parallelogram.Dispose();
            triangle.Dispose();
            oval.Dispose();
            hexagon.Dispose();
        }

        #region Attach loads to holder
        public void PositionStar()
        {
            Experior.Core.Mathematics.Trigonometry.LocalToGobal(new Vector3
(-0.21045f, -0.22428f, -0.0425f), Matrix4x4.Identity, holder.Position, holder.
Orientation, out var temp_pos, out var temp_ori);
            star.Position = temp_pos;
            star.Orientation = Matrix4x4.CreateFromYawPitchRoll(Yaw , Pitch , Roll
+ Trigonometry.Angle2Rad(30));
            holder.Attach(star);
        }
        public void PositionSquare()
        {
            Experior.Core.Mathematics.Trigonometry.LocalToGobal(new Vector3
(-0.2106f, -0.30539f, -0.0425f), Matrix4x4.Identity, holder.Position, holder.
Orientation, out var temp_pos, out var temp_ori);
            square.Position = temp_pos;
            square.Orientation = Matrix4x4.CreateFromYawPitchRoll(Yaw , Pitch,
Roll);
            holder.Attach(square);
        }
        public void PositionParallelogram()
        {
            Experior.Core.Mathematics.Trigonometry.LocalToGobal(new Vector3
(-0.12972f, -0.30539f, -0.0425f), Matrix4x4.Identity, holder.Position, holder.
Orientation, out var temp_pos, out var temp_ori);
            parallelogram.Position = temp_pos;
            parallelogram.Orientation = Matrix4x4.CreateFromYawPitchRoll(Yaw ,
Pitch, Roll + Trigonometry.Angle2Rad(90));
            holder.Attach(parallelogram);
        }
        public void PositionHexagon()
        {
            Experior.Core.Mathematics.Trigonometry.LocalToGobal(new Vector3
(0.1296f, -0.30518f, -0.0425f), Matrix4x4.Identity, holder.Position, holder.
Orientation, out var temp_pos, out var temp_ori);
            hexagon.Position = temp_pos;
            hexagon.Orientation = Matrix4x4.CreateFromYawPitchRoll(Yaw , Pitch,
Roll + Trigonometry.Angle2Rad(30));
            holder.Attach(hexagon);
        }
        public void PositionTriangle()
        {
            Experior.Core.Mathematics.Trigonometry.LocalToGobal(new Vector3
(0.20923f, -0.22539f, -0.0425f), Matrix4x4.Identity, holder.Position, holder.
Orientation, out var temp_pos, out var temp_ori);
            triangle.Position = temp_pos;
            triangle.Orientation = Matrix4x4.CreateFromYawPitchRoll(Yaw , Pitch,
Roll + Trigonometry.Angle2Rad(90));
            holder.Attach(triangle);
        }
        public void PositionOval()
        {
```

```
            Experior.Core.Mathematics.Trigonometry.LocalToGobal(new Vector3
(0.2101f, -0.30489f, -0.0425f), Matrix4x4.Identity, holder.Position, holder.
Orientation, out var temp_pos, out var temp_ori);
            oval.Position = temp_pos;
            oval.Orientation = Matrix4x4.CreateFromYawPitchRoll(Yaw , Pitch, Roll
+ Trigonometry.Angle2Rad(90));
            holder.Attach(oval);
        }
        public void AttachLoad_Holder(Core.Loads.Load load)
        {
            if (load == null)
                return;
            else if (load == star)
                PositionStar();
            else if (load == square)
                PositionSquare();
            else if (load == parallelogram)
                PositionParallelogram();
            else if (load == hexagon)
                PositionHexagon();
            else if (load == triangle)
                PositionTriangle();
            else if (load == oval)
                PositionOval();
        }

        #endregion

        #region Attach Load to Disk
        public void PositionStar_Disk()
        {
            Experior.Core.Mathematics.Trigonometry.LocalToGobal(new Vector3
(0.0684f, -0.18794f, 0.0325f), Matrix4x4.Identity, disk.Position, disk.
Orientation, out var temp_pos, out var temp_ori);
            star.Position = temp_pos;
            star.Orientation = temp_ori;
            star.Yaw = star.Yaw + Trigonometry.Angle2Rad(40.0f);
            disk.Attach(star);
        }
        public void PositionSquare_Disk()
        {
            Experior.Core.Mathematics.Trigonometry.LocalToGobal(new Vector3
(0.19696f, -0.03473f, 0.0325f), Matrix4x4.Identity, disk.Position, disk.
Orientation, out var temp_pos, out var temp_ori);
            square.Position = temp_pos;
            square.Orientation = temp_ori;
            square.Yaw = square.Yaw + Trigonometry.Angle2Rad(10.0f);
            disk.Attach(square);
        }
        public void PositionParallelogram_Disk()
        {
            Experior.Core.Mathematics.Trigonometry.LocalToGobal(new Vector3
(-0.12856f, -0.15321f, 0.0325f), Matrix4x4.Identity, disk.Position, disk.
Orientation, out var temp_pos, out var temp_ori);
            parallelogram.Position = temp_pos;
            parallelogram.Orientation = temp_ori;
            parallelogram.Yaw -= Trigonometry.Angle2Rad(20.0f);
            disk.Attach(parallelogram);
        }
        public void PositionHexagon_Disk()
        {
            Experior.Core.Mathematics.Trigonometry.LocalToGobal(new Vector3
(0.12856f, 0.15321f, 0.0325f), Matrix4x4.Identity, disk.Position, disk.
Orientation, out var temp_pos, out var temp_ori);
            hexagon.Position = temp_pos;
            hexagon.Orientation = temp_ori;
            hexagon.Yaw = hexagon.Yaw + Trigonometry.Angle2Rad(45.0f);
            disk.Attach(hexagon);
```

```
        }
        public void PositionTriangle_Disk()
        {
            Experior.Core.Mathematics.Trigonometry.LocalToGobal(new Vector3
(-0.0684f, 0.18794f, 0.0325f), Matrix4x4.Identity, disk.Position, disk.
Orientation, out var temp_pos, out var temp_ori);
            triangle.Position = temp_pos;
            triangle.Orientation = temp_ori;
            triangle.Yaw = triangle.Yaw + Trigonometry.Angle2Rad(40.0f);
            disk.Attach(triangle);
        }
        public void PositionOval_Disk()
        {
            Experior.Core.Mathematics.Trigonometry.LocalToGobal(new Vector3
(-0.19696f, 0.03473f, 0.0325f), Matrix4x4.Identity, disk.Position, disk.
Orientation, out var temp_pos, out var temp_ori);
            oval.Position = temp_pos;
            oval.Orientation = temp_ori;
            oval.Yaw = oval.Yaw + Trigonometry.Angle2Rad(90.0f);
            disk.Attach(oval);
        }
        public void AttachLoad_Disk(Core.Loads.Load load)
        {
            if (load == null)
                return;
            else if (load == star)
                PositionStar_Disk();
            else if (load == square)
                PositionSquare_Disk();
            else if (load == parallelogram)
                PositionParallelogram_Disk();
            else if (load == hexagon)
                PositionHexagon_Disk();
            else if (load == triangle)
                PositionTriangle_Disk();
            else if (load == oval)
                PositionOval_Disk();
        }

        #endregion

        #endregion
    }
}
```

Motion.cs

Overview

The *Motion.cs* contains the methods that handle the motion of the assemblies based on controller signals.

Code Example

```
using Experior.Core.Mathematics;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Numerics;
using System.Text;
using System.Threading.Tasks;
namespace Experior.Catalog.SchneiderElectric.DiskTracking.Assemblies
```

```
{
    public partial class DiskTracking
    {
        #region Functions
        private void Move_Linear_X()
        {
            axis1_cs.LocalPosition = new Vector3(0, (float)targetPositionX, 0);
        }
        private void Move_Linear_Y()
        {
            axis2_cs.LocalPosition = new Vector3((float)targetPositionY, 0, 0);
        }
        private void Move_Linear_Z()
        {
            axis3_cs.LocalPosition = new Vector3(0, 0, (float)targetPositionZ);
        }
        private void Move_C()
        {

            gripper.LocalYaw = Trigonometry.Angle2Rad((float)targetPositionC);
        }
        private void Move_T()
        {
            disk.LocalYaw =  Trigonometry.Angle2Rad((float)targetPositionT);
            dummyDisk.LocalYaw = disk.LocalYaw;
        }

        #endregion

    }
}
```

Glossary

D

digital twin:

A digital twin refers to a virtual representation or digital replica of a physical object, system, or process. It is a digital counterpart that simulates the behavior, characteristics, and performance of its physical counterpart in real-time or historical contexts. The concept of a digital twin allows for the integration of the physical and digital worlds, enabling organizations to monitor, analyze, and optimize the performance of their assets or processes.

EcoStruxure Machine Expert Twin provides features for visualization, simulation, and emulation of machines and automation lines throughout the complete lifecycle.

E

emulation:

Based on the *ISO 24765-2017 International Standard - Systems and software engineering--Vocabulary*, emulation is defined as the use of a data processing system to imitate another data processing system, so that the imitating system accepts the same data, executes the same programs, and achieves the same results as the imitated system.

M

Model view: In EcoStruxure Machine Expert Twin, the **Model** view provides the graphical representation of the scene.

P

physical simulation: The physical simulation is a software library that is designed to simulate and model physical systems in a computer-generated environment. It is used to create realistic and dynamic animations and simulations of objects, environments, and interactions between them. In EcoStruxure Machine Expert Twin the physical simulation uses mathematical algorithms to simulate physical phenomena, such as gravity, friction, and collision detection.

project: An EcoStruxure Machine Expert Twin project file is saved with the extension **.experior*. It contains the information about assemblies, connections, loads, settings.

S

scene: In the EcoStruxure Machine Expert Twin context, a scene is a representation of a set of assemblies interacting with loads.

simulation:

Based on the *ISO 24765-2017 International Standard - Systems and software engineering--Vocabulary*, simulation describes two concepts:

- A model that behaves or operates like a given system when provided a set of controlled inputs.
- The use of a data processing system to represent selected behavioral characteristics of a physical or abstract system.

In the context of this manual, the term simulation is used whenever it is referred to modeling physical systems in EcoStruxure Machine Expert Twin.

U

URDF: (unified robotics description format) A special type of eXtensible Markup Language (XML) file that includes the physical description of a robot and contains information on the mechanical structure, joints, 3-D modelling graphics, motors and colliders. URDF files are provided by numerous robotic manufacturers for download. EcoStruxure Machine Expert Twin allows importing URDF files for integrating third-party robots into a project without manual programming.

Index

Schneider Electric
35 rue Joseph Monier
92500 Rueil Malmaison
France

+ 33 (0) 1 41 29 70 00

www.se.com

As standards, specifications, and design change from time to time, please ask for confirmation of the information given in this publication.

EIO0000005040.00