OMRON

Automation Software

# Sysmac Studio

## 3D Simulation Function Operation Manual

**SYSMAC-SE2□□□**
**SYSMAC-SA4□□L-64**


SYSMAC
always in control

W618-E1-11

# Introduction

Thank you for purchasing a Sysmac Studio 3D Simulation Option.
This manual contains information that is necessary to use the 3D Simulation Function with the Sysmac Studio 3D Simulation Option. Please read this manual and make sure you understand the functionality and performance of the Sysmac Studio 3D Simulation Option before you attempt to use it in a control system.
Keep this manual in a safe place where it will be available for reference during operation.

## Intended Audience

This manual is intended for the following personnel, who must also have knowledge of electrical systems (an electrical engineer or the equivalent).

- Personnel in charge of introducing FA systems.
- Personnel in charge of designing FA systems.
- Personnel in charge of installing and maintaining FA systems.
- Personnel in charge of managing FA systems and facilities.

For programming, this manual is intended for personnel who understand the programming language specifications in international standard IEC 61131-3 or Japanese standard JIS B 3503.

## Applicable Products

This manual covers the following products.

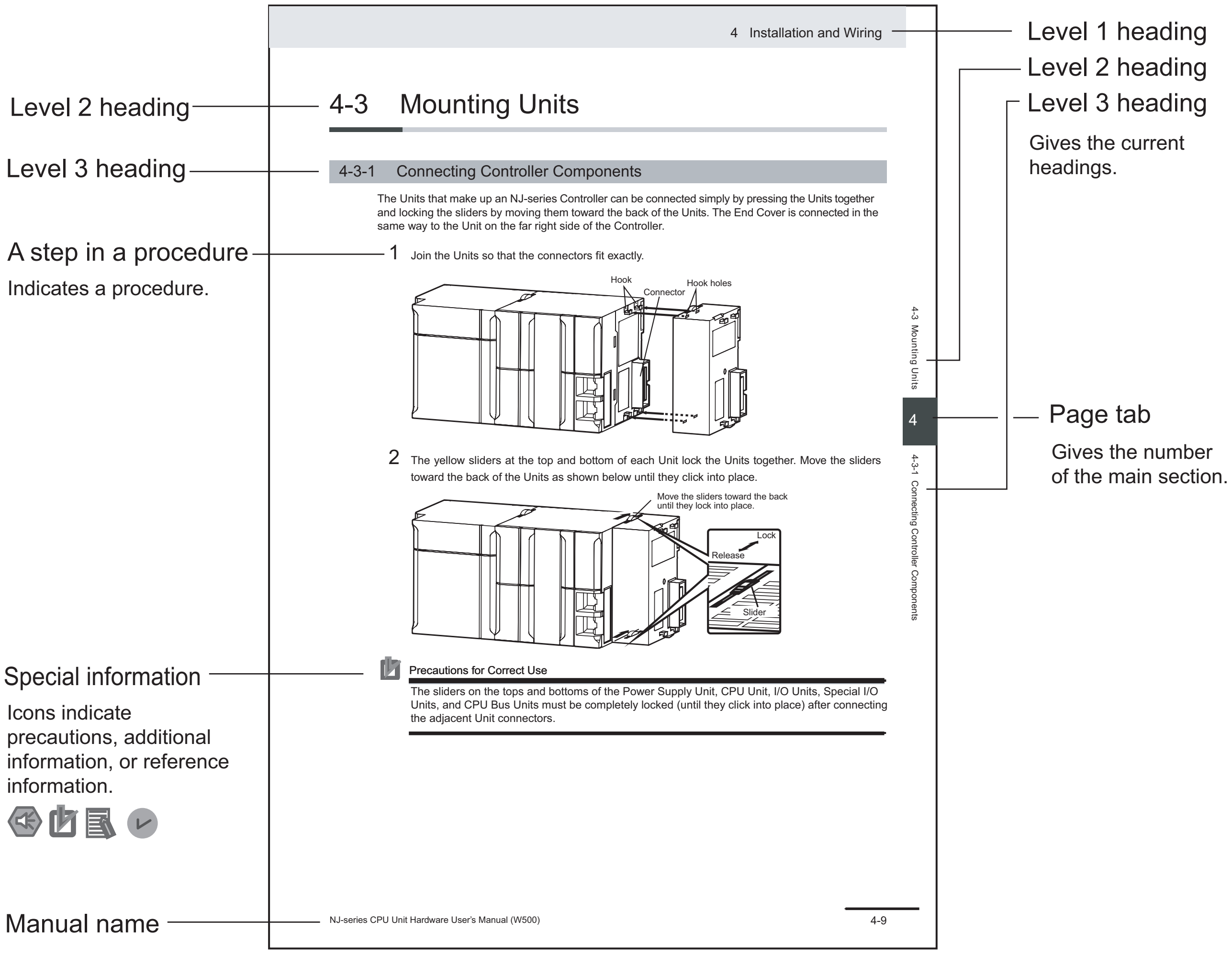
- Sysmac Studio 3D Simulation Option

Part of the specifications and restrictions for the products are given in other manuals.
Refer to *Related Manuals* on page 18.

# Manual Structure

## Page Structure

The following page structure is used in this manual.




This illustration is provided only as a sample. It may not literally appear in this manual.

## Special Information

Special information in this manual is classified as follows:

**Precautions for Safe Use**

Precautions on what to do and what not to do to ensure safe usage of the product.

**Precautions for Correct Use**

Precautions on what to do and what not to do to ensure proper operation and performance.

**Additional Information**

Additional information to read as required.
This information is provided to increase understanding or make operation easier.

**Version Information**

Information on differences in specifications and functionality for Controllers and Units with different unit versions and for different versions of Support Software is given.

## Precaution on Terminology

In this manual, *download* refers to transferring data from the Sysmac Studio to the physical Controller and *upload* refers to transferring data from the physical Controller to the Sysmac Studio.
For the Sysmac Studio, *synchronization* is used to both *upload* and *download* data. Here, *synchronize* means to automatically compare the data for the Sysmac Studio on the computer with the data in the physical Controller and transfer the data in the direction that is specified by the user.

# Sections in this Manual

# CONTENTS

## Section 4 Creating 3D Shape Data



## Section 5 Working with the 3D Visualizer and 3D Editing Area

## Section 8 Executing a 3D Simulation



## Section 9 3D Simulation of Robot Integrated Systems



## Section 10 Useful Functions



## Appendices

# Index

# Terms and Conditions Agreement

## WARRANTY

- The warranty period for the Software is one year from the date of purchase, unless otherwise specifically agreed.
- If the User discovers defect of the Software (substantial non-conformity with the manual), and return it to OMRON within the above warranty period, OMRON will replace the Software without charge by offering media or download from OMRON's website. And if the User discovers defect of media which is attributable to OMRON and return it to OMRON within the above warranty period, OMRON will replace defective media without charge. If OMRON is unable to replace defective media or correct the Software, the liability of OMRON and the User's remedy shall be limited to the refund of the license fee paid to OMRON for the Software.

## LIMITATION OF LIABILITY

- THE ABOVE WARRANTY SHALL CONSTITUTE THE USER'S SOLE AND EXCLUSIVE REMEDIES AGAINST OMRON AND THERE ARE NO OTHER WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO, WARRANTY OF MERCHANTABILITY OR FITNESS FOR PARTICULAR PURPOSE. IN NO EVENT, OMRON WILL BE LIABLE FOR ANY LOST PROFITS OR OTHER INDIRECT, INCIDENTAL, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF USE OF THE SOFTWARE.
- OMRON SHALL HAVE NO LIABILITY FOR DEFECT OF THE SOFTWARE BASED ON MODIFICATION OR ALTERNATION TO THE SOFTWARE BY THE USER OR ANY THIRD PARTY. OMRON SHALL NOT BE RESPONSIBLE AND/OR LIABLE FOR ANY LOSS, DAMAGE, OR EXPENSES DIRECTLY OR INDIRECTLY RESULTING FROM THE INFECTION OF OMRON PRODUCTS, ANY SOFTWARE INSTALLED THEREON OR ANY COMPUTER EQUIPMENT, COMPUTER PROGRAMS, NETWORKS, DATABASES OR OTHER PROPRIETARY MATERIAL CONNECTED THERETO BY DISTRIBUTED DENIAL OF SERVICE ATTACK, COMPUTER VIRUSES, OTHER TECHNOLOGICALLY HARMFUL MATERIAL AND/OR UNAUTHORIZED ACCESS.
- OMRON SHALL HAVE NO LIABILITY FOR SOFTWARE DEVELOPED BY THE USER OR ANY THIRD PARTY BASED ON THE SOFTWARE OR ANY CONSEQUENCE THEREOF.

## APPLICABLE CONDITIONS

USER SHALL NOT USE THE SOFTWARE FOR THE PURPOSE THAT IS NOT PROVIDED IN THE ATTACHED USER MANUAL.

## CHANGE IN SPECIFICATION

The software specifications and accessories may be changed at any time based on improvements and other reasons.

## ERRORS AND OMISSIONS

The information in this manual has been carefully checked and is believed to be accurate; however, no responsibility is assumed for clerical, typographical, or proofreading errors, or omissions.

# Safety Precautions

Refer to the *Sysmac Studio Version 1 Operation Manual (Cat. No. W504)* for safety precautions.

# Precautions for Safe Use

Refer to the *Sysmac Studio Version 1 Operation Manual (Cat. No. W504)* for precautions for safe use.

# Precautions for Correct Use

Refer to the *Sysmac Studio Version 1 Operation Manual (Cat. No. W504)* for precautions for correct use.

# Regulations and Standards

## Software Licenses and Copyrights

This product incorporates certain third party software. The license and copyright information associated with this software is available at http://www.fa.omron.co.jp/nj_info_e/.

# Versions

Hardware revisions and unit versions are used to manage the hardware and software in NJ/NX-series Units, NY-series Industrial PCs, and EtherCAT slaves.
Refer to *Sysmac Studio Version 1 Operation Manual (Cat. No. W504)* for details on versions.

# Related Manuals

The followings are the manuals related to this manual. Use these manuals for reference.

| Manual name | Cat. No. | Model numbers | Application | Description |
|---|---|---|---|---|
| Sysmac Studio<br>3D Simulation Function Operation Manual | W618 | SYSMAC-SE2□□□<br>SYSMAC-SA4□□□-64 | Learning about an outline of the 3D simulation function of the Sysmac Studio and how to use the function. | Describes an outline, execution procedures, and operating procedures for the 3D simulation function of the Sysmac Studio. |
| Sysmac Studio Version 1 Operation Manual | W504 | SYSMAC-SE2□□□ | Learning about the operating procedures and functions of the Sysmac Studio. | Describes the operating procedures of the Sysmac Studio. |
| Sysmac Studio<br>Robot Integrated System Building Function with Robot Integrated CPU Unit Operation Manual | W595 | SYSMAC-SE2□□□<br>SYSMAC-SE200D-64 | Learning about the operating procedures and functions of the Sysmac Studio to configure Robot Integrated System using Robot Integrated CPU Unit. | Describes the operating procedures of the Sysmac Studio for Robot Integrated CPU Unit. |
| Sysmac Studio<br>Robot Integrated System Building Function with IPC Application Controller Operation Manual | W621 | SYSMAC-SE2□□□<br>SYSMAC-SE200D-64 | Learning about the operating procedures and functions of the Sysmac Studio to configure Robot Integrated System using IPC Application Controller. | Describes the operating procedures of the Sysmac Studio for IPC Application Controller. |
| Sysmac Studio<br>Drive Functions Operation Manual | I589 | SYSMAC-SE2□□□<br>SYSMAC-DE□□□L | Learning about the Servo Drive related functions of the Sysmac Studio. | Describes the Servo Drive related operating procedures and functions among those of the Sysmac Studio. |
| NJ/NX-series CPU Unit Software User's Manual | W501 | NX701-□□□□<br>NX502-□□□□<br>NX102-□□□□<br>NX1P2-□□□□<br>NJ501-□□□□<br>NJ301-□□□□<br>NJ101-□□□□ | Learning how to program and set up an NJ/NX-series CPU Unit. Mainly software information is provided. | The following information is provided on a Controller built with an NJ/NX-series CPU Unit.<br>• CPU Unit operation<br>• CPU Unit features<br>• Initial settings<br>• Programming based on IEC 61131-3 language specifications |

| Manual name | Cat. No. | Model numbers | Application | Description |
|---|---|---|---|---|
| NJ-series<br>NJ Robotics CPU Unit<br>User’s Manual | W539 | NJ501-4□□□<br>NJ501-R□□□ | Controlling robots with NJ-series CPU Units. | Describes the functionality to control robots. |
| NJ-series<br>Robot Integrated CPU Unit<br>User’s Manual | O037 | NJ501-R□□□ | Using the NJ-series Robot Integrated CPU Unit. | Describes the settings and operation of the CPU Unit and programming concepts for OMRON robot control. |
| NA-series<br>Programmable Terminal<br>Software<br>User’s Manual | V118 | NA5-□W□□□□ | Learning about NA-series PT pages and object functions. | Describes the pages and object functions of the NA-series Programmable Terminals. |

# Terminology

The following describes the terms used in this manual.

| Term | Description |
|---|---|
| Virtual equipment model | An equipment model created in the Sysmac Studio in order to execute a 3D simulation. A virtual equipment model is made up of 3D shape data for a mechanical component, custom mechanics, parallel link model, Virtual Part Detection Sensor, and part, and the settings and scripts that define the operations of these 3D shape data. |
| 3D shape data | Data that represents the shape, size, and position of a Virtual equipment model in 3D space, which is created in the Sysmac Studio. 3D shape data is made up of CAD data, boxes, and cylinders. |
| 3D Visualization object | A generic term for data that manages the 3D shape data for a virtual equipment model used in a 3D simulation, settings for operating the Mechanical Component and Virtual Part Detection Sensor, and scripts that define the operations of the part.<br>You can add 3D visualization objects such as *Box*, *Virtual Part Detection Sensor*, *Shape Script*, and other data under **Configurations and Setup** – **3D Visualization** in the Multiview Explorer. |
| Mechanical component | A component driven by Servo, such as an X-Y table. You can operate a mechanical component by setting the operations of movable parts that make up the component and assigning axis variables and BOOL variables for the Controller. |
| Custom mechanics | A component that you can create by selecting desired parts and combining them. Use this component to realize operation that cannot be performed by standard mechanical components. This component enables you to realize operation of electric chucks and electric cylinders. |
| Parallel link model | A component that represents a parallel link robot. Three types of parallel link model that are supported by NJ-series NJ Robotics CPU Unit are available. |
| Part | This is the target object carried by a mechanical component, custom mechanics, or parallel link model in a 3D simulation. To realize part operations, such as displaying and moving a part, create scripts in the C# language. |
| Virtual Part Detection Sensor | A virtual sensor that detect a part in a 3D simulation. You can assign BOOL variables for the Controller as inputs to the Controller. |
| Application Manager | A logical device that manages the data and settings required to use the 3D Simulation Function, as well as the settings and programs of the IPC Application Controller that controls robot systems. In a 3D simulation, Application Manager manages the 3D shape data for a Virtual equipment model, settings for operating the Mechanical Component and Virtual Part Detection Sensor, and scripts that define the operations of the part.<br>Refer to the *Sysmac Studio Robot Integrated System Building Function with IPC Application Controller Operation Manual (Cat. No. W621)* for the Application Manager functions to control robot systems. |
| Shape Script | A program that defines the operations of 3D shape data such as a part and movable part. A Shape Script is written in the C# language. |
| Shape Script Sequence | A setting that defines the order of execution of Shape Scripts. A Shape Script is executed from a Shape Script Sequence. |
| CAD data | 3D CAD data for equipment or a part, which becomes the basis of 3D shape data. Use third party 3D CAD software to create CAD data. You can load CAD data files with a .stp, .step, .igs, .iges, .usd, .usda, .usdc, or .usdz extension. |
| Box | A box-shaped object that becomes the basis of 3D shape data. This object is provided as standard in the Sysmac Studio, and has height, width, depth, and color settings. |
| Cylinder | A cylinder-shaped object that becomes the basis of 3D shape data. This object is provided as standard in the Sysmac Studio, and has radius, height, and color settings, |

| Term | Description |
|---|---|
| Location | Information that represents the position and pose of 3D shape data. The position is represented by the coordinate components (X, Y, Z) of a right-handed coordinate system. The pose is represented by the rotation angle around an axis (yaw, pitch, and roll) that is centered at the origin of a local coordinate system.<br>The values that represent the position and pose (X, Y, Z, yaw, pitch, and roll) are called location elements. Refer to *Pose of 3D Shape Data* on page 4-4 for the detailed definitions of yaw, pitch, and roll. |
| World coordinate system | A coordinate system that represents absolute coordinates on the 3D Visualizer. The position of 3D shape data in a world coordinate system is represented by the coordinate components X, Y, and Z. |
| Local coordinate system | Individual 3D shape data has its own 3D shape data. Use this coordinate system when you use an offset to determine the center of rotation of 3D shape data, or set relative positions between two sets of 3D shape data. |
| Parent and child | Terms that represent the relationship between two sets of 3D shape data that are connected together. When more than one set of 3D shape data operates, the 3D shape data for the main operation is called *parent* and the 3D shape data for following the operation of the parent is called *child*. To set a parent-child relationship between two sets of 3D shape data, select the parent 3D shape data on the setup tab page for the child 3D shape data. |
| Mount point and link point | Points used to join two sets of 3D shape data that have a parent-child relationship on the 3D Visualization display. Set a mount point in the child 3D shape data and a link point in the parent 3D shape data. Then, join the mount point of the child 3D shape data with the link point of the parent 3D shape data. Thus, you can easily position two sets of 3D shape data. |
| TCP (Tool Center Point) | TCP is a point that specifies the position in the 3D shape data for the mechanical component, custom mechanics, or parallel link model, at which a tool such as a robot hand is mounted. |
| OpenUSD(Universal Scene Description) | An open-source file format developed by Pixar Animation Studio for transferring 3D scenes and data between different software applications. |
| USDPhysicsJoint | Physics joint information of the physics schema defined in OpenUSD. It is used to represent the relationship between two or more rigid bodies in physics simulations and enables the creation of complex interconnected structures and motions in a 3D environment. |
| Prim | A fundamental component in OpenUSD. Prims represent objects and entities in a scene, and come in various types such as Geometry, Lights, Cameras, and Groups. |
| Prim Path | A text string that identifies the position of a prim within a scene graph hierarchy. Each element of the path represents a nesting level within the hierarchy. It begins with the root (“/”), followed by the names of parent and child prims separated by slashes. |

# Revision History

A manual revision code appears as a suffix to the catalog number on the front and back covers of the manual.

**Cat. No. W618-E1-11**

Revision code

| Revision code | Date | Revised content |
|---|---|---|
| 01 | April 2020 | Original production |
| 02 | August 2020 | Revisions for an upgrade to Sysmac Studio version 1.42. |
| 03 | October 2020 | Revisions for an upgrade to Sysmac Studio version 1.43. |
| 04 | January 2021 | Revisions for an upgrade to Sysmac Studio version 1.44. |
| 05 | April 2021 | Revisions for an upgrade to Sysmac Studio version 1.45. |
| 06 | October 2021 | Revisions for an upgrade to Sysmac Studio version 1.47. |
| 07 | April 2022 | Revisions for an upgrade to Sysmac Studio version 1.49. |
| 08 | July 2022 | Revisions for an upgrade to Sysmac Studio version 1.50. |
| 09 | October 2022 | Revisions for an upgrade to Sysmac Studio version 1.52. |
| 10 | April 2023 | Revisions for an upgrade to Sysmac Studio version 1.54. |
| 11 | April 2025 | Revisions for an upgrade to Sysmac Studio version 1.62. |

1

1

# Features and Specifications of the 3D Simulation Function

This section provides an introduction and features of the Sysmac Studio *3D Simulation* Function.

# 1-1 Features of the 3D Simulation

The Sysmac Studio 3D Simulation Function allows you to visually see operations of the equipment controlled by an NJ/NX/NY-series Controllers and operation of the processed or assembled parts being carried, on a computer.
You can check on a computer how a part is carried by a conveyor belt and an X-Y table and manipulated by a gantry crane before you assemble the physical equipment.
The 3D Simulation Function has the following features.

## Reduced ROI (Return On Investment) Check Time

Because this function enables a visual check on the entire equipment, you can quickly verify the feasibility of the equipment requirements without purchasing a complete set of the actual equipment.
This leads to early decision-making on investment in equipment introduction.

## Reduced Design and Start-up Time

The function allows for creating and debugging programs while you check the entire system operation and the part movement on the same screen, which reduces the design time.
You can use the equipment's CAD data to check for interference between equipment objects.
This leads to the reduction of the on-site equipment start-up time because you can adjust the layout of equipment objects and parts in advance.

## Reduced Product Type Change Verification Time

In cases where you need to change the type of parts that the equipment processes, the function allows for changing and verifying the programs and settings required for the change without stopping the equipment.
It enables the verification and adjustment with an actual equipment in a short time and reduces the equipment stop time.

# 1-2 Specifications



## 1-2-1 Product Model Numbers

The Sysmac Studio 3D Simulation Function supports Sysmac Studio (64 bit) version 1.40 or higher.
To use the Sysmac Studio 3D Simulation Function, the following Sysmac Studio licenses is needed. In addition, to execute a 3D simulation of a mechanical component, the following Sysmac Studio option license is needed.
To install the Sysmac Studio (64 bit), the following DVD is needed.

### ● Sysmac Studio License

| Product name | Number of licenses | Model number |
|---|---|---|
| Sysmac Studio Standard Edition Ver.1.□□ | 1 license | SYSMAC-SE201L |
| | 3 licenses | SYSMAC-SE203L |
| | 10 licenses | SYSMAC-SE210L |
| | 30 licenses | SYSMAC-SE230L |
| | 50 licenses | SYSMAC-SE250L |

### ● Sysmac Studio Option License

| Product name | Number of licenses | Model number |
|---|---|---|
| Sysmac Studio 3D Simulation Option | 1 license | SYSMAC-SA401L-64 |
| | 3 licenses | SYSMAC-SA403L-64 |
| | 10 licenses | SYSMAC-SA410L-64 |
| | 30 licenses | SYSMAC-SA430L-64 |
| | 50 licenses | SYSMAC-SA450L-64 |

### ● DVD

| Product name | Media | Model number |
|---|---|---|
| Sysmac Studio Standard Edition Ver.1.□□ | 64-bit edition DVD | SYSMAC-SE200D-64 |

## 1-2-2 Supported Languages

The supported languages conform to the specifications of the Sysmac Studio. Refer to *Sysmac Studio Version 1 Operation Manual (Cat. No. W504)* for details.

## 1-2-3 Applicable Models

You can use the Sysmac Studio 3D Simulation Function with all the controllers supported by the Sysmac Studio. Refer to *Sysmac Studio Version 1 Operation Manual (Cat. No. W504)* for details.

## 1-2-4 Applicable Computers

The computer on which the Sysmac Studio (64 bit) can be installed. Refer to *Sysmac Studio Version 1 Operation Manual (Cat. No. W504)* for details.

To use the 3D Simulation Option functions, the following operating environment is needed.

| System requirement | Specification |
|---|---|
| CPU | DOS/V personal computers (IBM AT compatible machines) equipped with Intel® Core™ i5 8250U (1.60-3.40 GHz) or equivalent/faster processors<br>Intel® Core™ i7 9750H or equivalent or faster recommended |
| RAM | 8 GB min.<br>16 GB min. recommended |
| Display | Full HD 1,920 × 1,080, 16 million colors min. |
| Video card | Video card that supports DirectX11 (NVIDIA® GeForce®, AMD Radeon™, etc., NVIDIA recommended) |

# 2

# Software Setup

This section describes the procedures for setting up the software to use the Sysmac Studio 3D Simulation Function.

# 2-1 Installing the Sysmac Studio

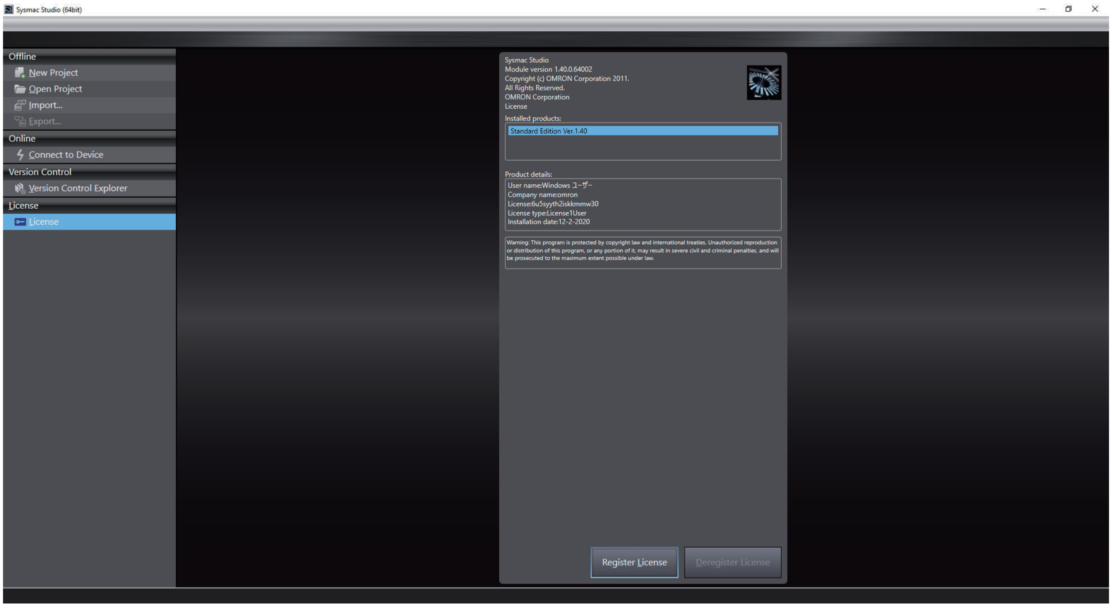
Install the Sysmac Studio from the DVD. For details of the installation procedure, refer to *Sysmac Studio Version 1 Operation Manual (Cat. No. W504)*.

# 2-2 Registering Sysmac Studio Option License

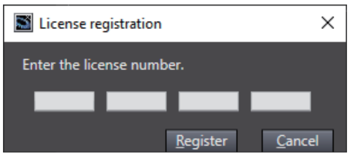
To execute a 3D simulation of a mechanical component, you must register the *Sysmac Studio 3D Simulation Option* license on the Sysmac Studio Standard Edition.
Use the following procedure to register an option license.

**1** Select **All Programs** - **OMRON** - **Sysmac Studio** - **Sysmac Studio** from the Windows Start menu.
The Sysmac Studio starts and the start page is displayed.

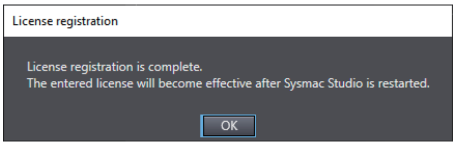

**2** Click **License** on the start page.
The licenses that are currently registered are displayed.




**3** Click the **Register License** button.
The **License Registration** dialog box is displayed.




**4** Enter the Sysmac Studio 3D Simulation Option license number, and then click the **Register** button.
If the license is registered normally, a message appears asking you to restart the software.




Restart the Sysmac Studio to complete registration.

# 3

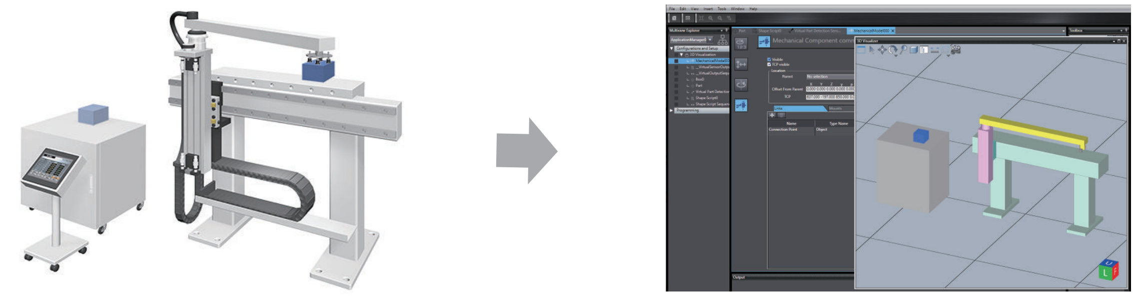
# 3D Simulation Procedure

3

This section describes the required preparations and outline of the procedure to execute a 3D simulation.

# 3-1 Introduction to 3D Simulation

To execute a Sysmac Studio 3D simulation, create a virtual equipment model that represents the equipment and part in the Sysmac Studio and simulate how the virtual equipment model operates.

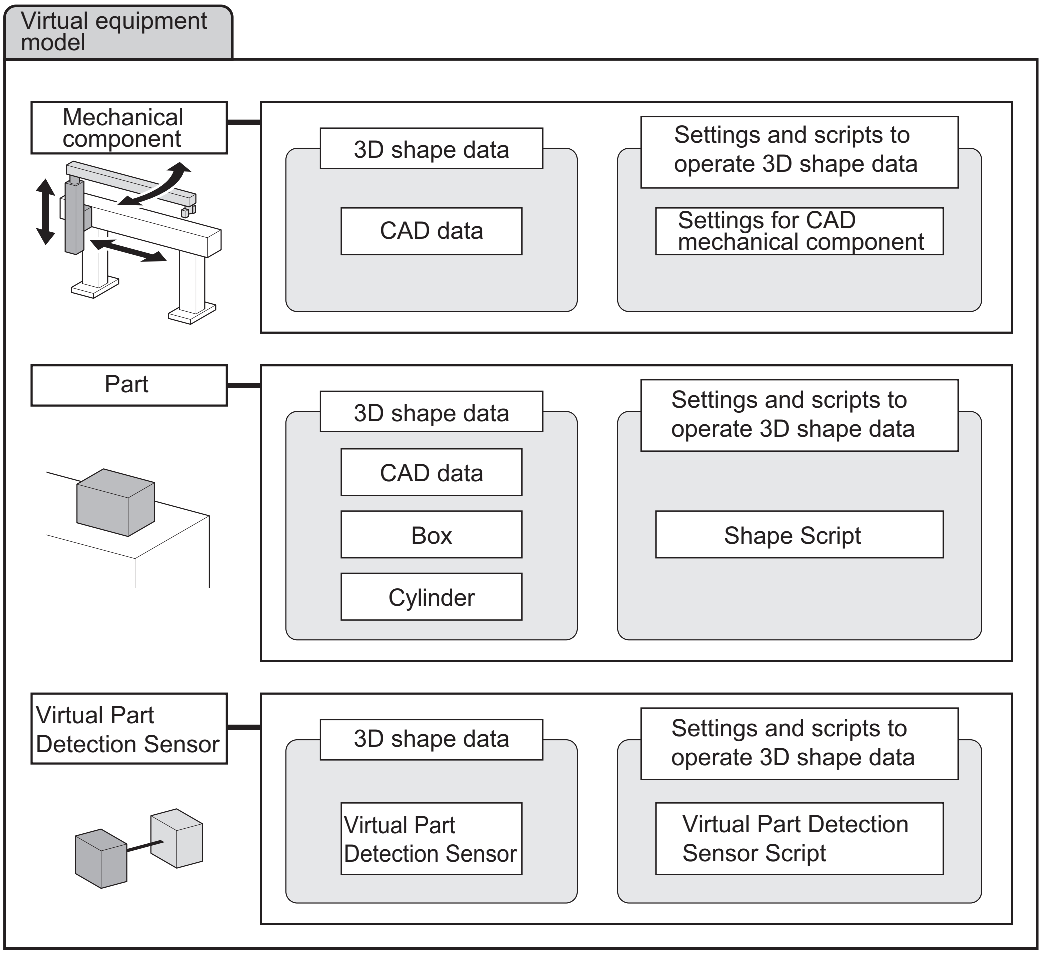


A virtual equipment model consists of a *mechanical component*[*1] that represents a movable part of the equipment, a *part* that represents the target to be carried, and a *Virtual Part Detection Sensor* that detects the part.

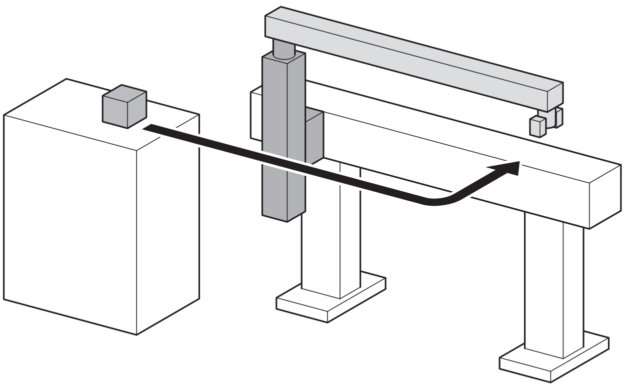
*1. Includes custom mechanics and parallel link models. Mechanical components, custom mechanics, and parallel link models are hereinafter collectively called *mechanical components* depending on the context.

The mechanical component, part, and Virtual Part Detection Sensor are made up of 3D shape data and the settings and scripts for operating the 3D shape data.
Create 3D shape data, and then set and create the settings and scripts for operating the 3D shape data.

The relationship among the virtual equipment model, 3D shape data, and settings and scripts for operating the 3D shape data is as shown below.




The following sections describe the operating procedures and functions to execute a 3D simulation, using a part carrying equipment as an example.



# 3-2 Processes of 3D Simulation

This section describes the processes of 3D simulation using the Sysmac Studio.
Refer to the corresponding section for details on each process.
To execute a 3D simulation, you must create a project in advance, and then create programs and register axes. Refer to *Sysmac Studio Version 1 Operation Manual (Cat. No. W504)* for the project creation, program creation, and axis registration procedures.

The following describes the processes of 3D simulation using a mechanical component as an example.

| Process | Description | Reference |
|---|---|---|
| Creating a Project<br>Creating Programs and Registering Axes | Create a Sysmac Studio project, and then create programs for controlling movable parts (mechanical components). In addition, register the axes for controlling movable parts. | *Sysmac Studio Version 1 Operation Manual (Cat. No. W504)* |
| ↓ | | |
| Creating 3D Shape Data | Add Application Manager to the project and create 3D shape data for the mechanical component, part, and Virtual Part Detection Sensor of the equipment. | --- |
| Adding Application Manager | Add Application Manager that manages the data and settings required to use the 3D Simulation Function. | *4-2 Adding Application Manager* on page 4-7 |
| ↓ | | |
| Adding the Mechanical Component | Load the CAD data for the target equipment and create 3D shape data for the mechanical component.<br>Depending on the configuration of the target equipment, select *Mechanical Component*, *Custom Mechanics*, or *Parallel Link Model*. | *4-3 Creating 3D Shape Data for the Mechanical Component* on page 4-10<br>*4-4 Creating 3D Shape Data for the Custom Mechanics* on page 4-33<br>*4-5 Creating 3D Shape Data for Parallel Link Model* on page 4-82 |
| ↓ | | |
| Adding the Part | Load the CAD data for the part and add 3D shape data for the part. Instead of the CAD data, you can add boxes and cylinders provided as standard in the Sysmac Studio to create 3D shape data for the part. | *4-6 Adding 3D Shape Data for the Part* on page 4-94 |
| ↓ | | |

| Process | | Description | | Reference |
|---|---|---|---|---|
| | Adding the Part Detection Sensor | Add 3D shape data for the *Virtual Part Detection Sensor*, which is a virtual sensor to detect the part. | | *4-7 Adding the Part Detection Sensor* on page 4-103 |

| | | | |
|---|---|---|---|
| Configuring Simulation Settings | Select whether to use scripts or Easy Part Simulation Configuration as the method of 3D simulation. | | *3-2-1 Selecting How to Configure Simulation Settings* on page 3-5 |
| | | How to configure simulation settings using scripts | *3-2-2 Configuring 3D Simulation Settings Using Scripts* on page 3-6 |
| | | How to configure simulation settings using Easy Part Simulation Configuration | *3-2-3 Configuring 3D Simulation Settings Using Easy Part Simulation Configuration* on page 3-8 |

| | | | |
|---|---|---|---|
| Executing a Simulation | Execute a 3D simulation to check how the virtual equipment model operates. | | --- |
| | | How to execute a 3D simulation using scripts | *8-1-2 Executing the Operation Script for the Virtual Part Detection Sensor and the Virtual Output Script for the Mechanical Component* on page 8-3 |
| | | How to execute a 3D simulation using Easy Part Simulation Configuration | *8-1-4 Executing a Simulation Using Behaviors Settings* on page 8-4 |

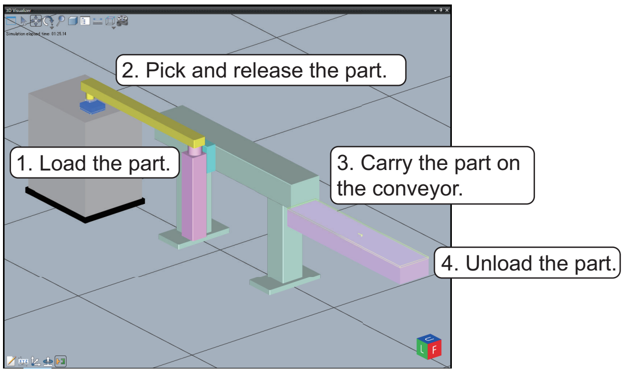
## 3-2-1 Selecting How to Configure Simulation Settings

There are two methods of 3D simulation, i.e., using scripts or using Easy Part Simulation Configuration. Select how to configure 3D simulation settings by considering the following features.

| Method of 3D simulation | Features |
|---|---|
| Using scripts | Enter scripts (C# language-based programs) in which the motions of the part are expressed by script functions based on the operation specifications of the mechanical component considered in the equipment design phase.<br>This method is flexible because you can handle all 3D shape data displayed on the 3D Visualizer using scripts. |
| Using Easy Part Simulation Configuration | Enter the mechanical component settings that define what motion the mechanical component will give to the part based on the operation specifications of the mechanical component considered in the equipment design phase. Unlike using scripts, this method allows for quick part simulation because it requires no consideration to create scripts and no entry of scripts.<br>However, since it is specialized for the simulation of part behaviors, the combined use of scripts may also be necessary for complex 3D visualization. |

## 3-2-2 Configuring 3D Simulation Settings Using Scripts

This section provides an overview of 3D simulation using scripts.




Operation specifications of mechanical components considered in the equipment design phase

| No. | Behavior of 3D Machine Model | Function | State (StepID) |
|---|---|---|---|
| 1 | Loads a part | LoadPart() | - |
| 2 | Clamps the part | ClampPart() | 1 |
| | Releases the part | ReleasePart() | 2 |
| 3 | Conveys the part | MoveObjectOnBelt() | - |
| 4 | Unloads the part | UnloadCollidingPart() | - |

Express the part behaviors with script functions

```
            // Please assign your shape.
            IBox partModel = (IBox) ace["/ApplicationManager0/Part"];
            this.CreateMultipleObjects(partModel, "Part", "PartGroup", 2, renderInfoList);

            return renderInfoList;
        }

        /// <summary>
        /// Called to render the Shape Script object
        /// </summary>
        /// <returns>The list of shapes to render</returns>
        public override void Render(IEnumerable<ShapeRenderInfo> renderInfoList) {
            this.InitializeObject(renderInfoList);

            IBox partModel = (IBox) ace["/ApplicationManager0/Part"];
            IBox handModel = (IBox) ace["/ApplicationManager0/HandBox"];
            IBox beltModel = (IBox) ace["/ApplicationManager0/Belt"];
            IBox pickPlaceModel = (IBox) ace["/ApplicationManager0/PickPlace"];
            IBox stopperModel = (IBox) ace["/ApplicationManager0/Stopper"];
            var controllerName = "new_Controller_0";
```

Enter scripts (C# language-based programs)

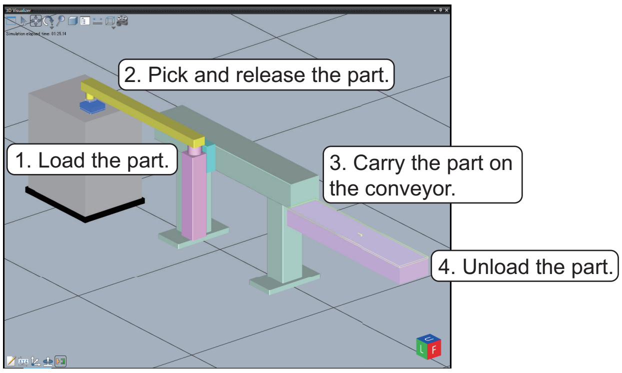
The table below describes the flow of configuring 3D simulation settings using scripts.

| Process | Description | Reference |
|---|---|---|
| Configuring Settings and Creating Scripts to Operate 3D Shape Data | Configure settings and create scripts for operating the 3D shape data for the equipment. | --- |
| Setting the Mechanical Component | Configure the operation settings for the mechanical component. | *6-2 Setting Mechanical Component* on page 6-10 |
| | | |
| Creating Operation Scripts for the Part | Create scripts that define the behaviors of the part. | *6-3 Creating Operation Scripts for the Part* on page 6-12 |
| | | |

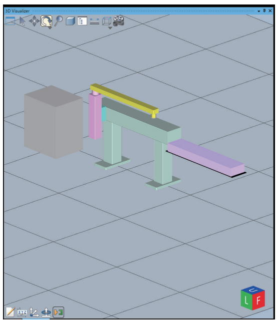
| | Process | Description | Reference |
|---|---|---|---|
| | Configuring the Operation Settings for the Part Detection Sensor | Configure the operation settings for the Virtual Part Detection Sensor. From the operation settings, generate scripts that define the operations. | *6-4 Configuring the Operation Settings for the Virtual Part Detection Sensor* on page 6-16 |

## 3-2-3 Configuring 3D Simulation Settings Using Easy Part Simulation Configuration

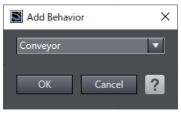
This section provides an overview of 3D simulation using Easy Part Simulation Configuration.



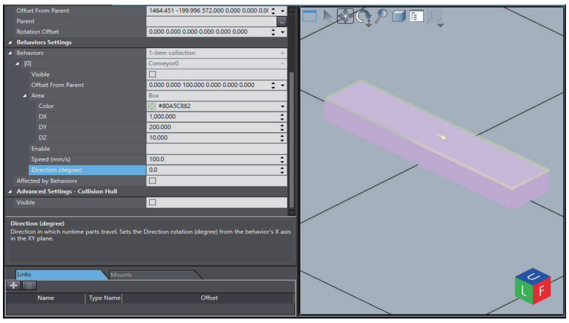

Operation specifications of mechanical components considered in the equipment design phase



(1) In the 3D Visualizer, double-click the 3D shape data for which to set *Conveyor*.




(2) Select **Conveyor** in the **Add Behavior** dialog box.



(3) Set the area, direction, and speed for carrying the part.

Enter the mechanical component settings that define what motion the mechanical component will give to the part.

The table below describes the flow of configuring 3D simulation settings using Easy Part Simulation Configuration.

| Process | Description | Reference |
|---|---|---|
| Configuring Settings to Operate the Part with 3D Shape Data | Configure settings to operate the part based on the operation of 3D shape data that make up the virtual equipment model, such as the mechanical component. | --- |
| Setting the Mechanical Component | Configure the operation settings for the mechanical component. | *7-2 Setting Mechanical Component* on page 7-3 |
| | | |
| Configuring Settings to Operate the Part | Configure settings to operate the part in 3D shape data that make up the virtual equipment model, such as the mechanical component. | *7-3 Behaviors Settings* on page 7-4 |

# 4

# Creating 3D Shape Data

To execute a 3D simulation in the Sysmac Studio, add an Application Manager, and create 3D shape data for the Mechanical Component, part, and Virtual Part Detection Sensor.

# 4-1 Introduction to 3D Shape Data

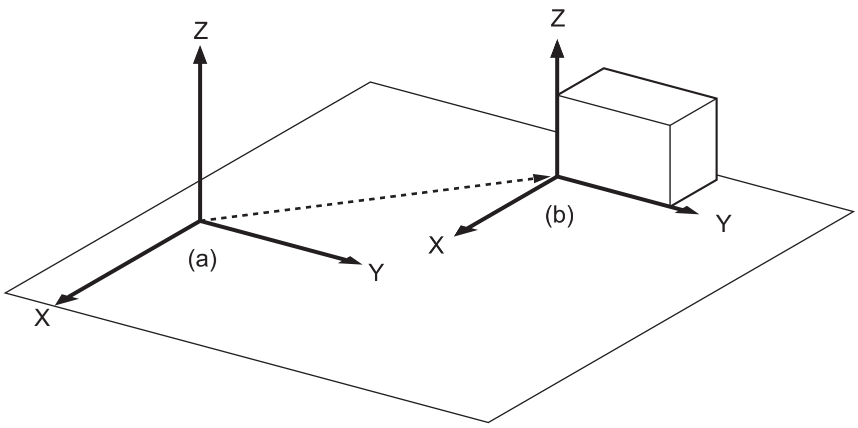
3D shape data is a set of data used in the 3D Simulation Function, which represents the shape, size, position, and pose of the movable parts, part, and Virtual Part Detection Sensor of the equipment. To create 3D shape data, use CAD data created with 3D CAD software, or box and cylinder data provided as standard in the Sysmac Studio.
In the Sysmac Studio 3D Visualizer and 3D editing area, the position of 3D shape data is represented by a coordinate value.
You can set a parent-child relationship between two sets of 3D shape data to realize the conveying operation of the part.

## 4-1-1 Location of 3D Shape Data

The position and pose of 3D shape data are represented by the location elements (X, Y, Z, yaw, pitch, and roll).
The position and pose are defined as follows.

### Position of 3D Shape Data

There are two coordinate systems, i.e. the world coordinate system and the local coordinate system, to represent the position of 3D shape data.
The position of 3D shape data is represented by the coordinate components *X, Y, and Z* of the world coordinate system or local coordinate system.

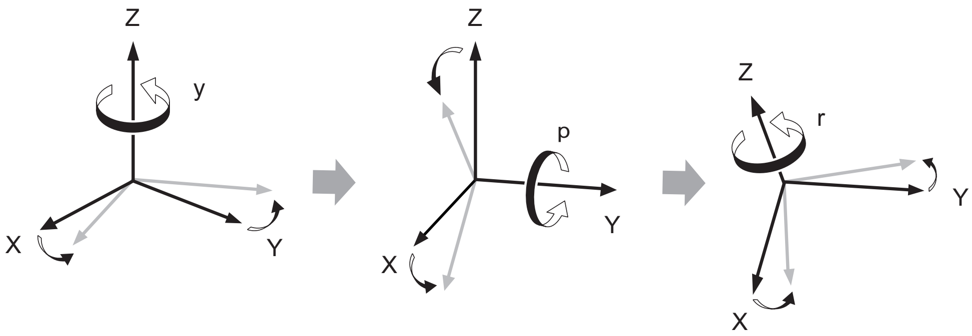



| | Coordinate system | Description |
|---|---|---|
| (a) | World coordinate system | This coordinate system represents absolute positions on the 3D Visualizer. |
| (b) | Local coordinate system | Each 3D shape data has its local coordinate system. Use this coordinate system when you configure an offset to determine the rotational center of 3D shape data, or set a relative position between two sets of 3D shape data. |

| Coordinate component | Description | Unit |
|---|---|---|
| X | Represents the position along the X axis. | mm |
| Y | Represents the position along the Y axis. | mm |
| Z | Represents the position along the Z axis. | mm |

## Pose of 3D Shape Data

The pose of 3D shape data is represented by *y (yaw), p (pitch), and r (roll)*, the rotation angles around axes centered at the origin of the local coordinate system. In the 3D Simulation Function, *y (yaw), p (pitch), and r (roll)* are defined as follows.



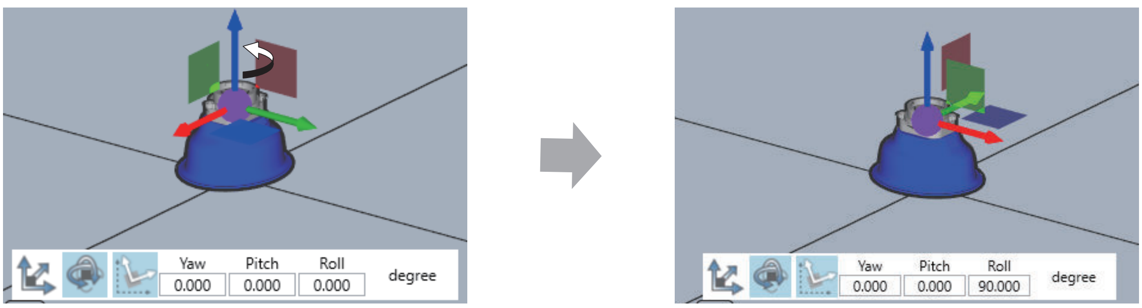

(a) (b) (c)

| | Rotation | Description | Unit |
|---|---|---|---|
| (a) | y (yaw) | Represents the rotation angle around the Z axis of the local coordinate system. | ° (degree) |
| (b) | p (pitch) | Represents the rotation angle around the Y axis of the local coordinate system based on the calculation of yaw. | ° (degree) |
| (c) | r (roll) | Represents the rotation angle around the Z axis of the local coordinate system based on the calculations of yaw and pitch. | ° (degree) |

This is similar to the concept of Z-Y-Z Euler angle. Set the y (yaw), p (pitch) and r (roll) values by rotating the 3D shape data around the Z, Y, and Z axes in this order.
Examples are shown below.

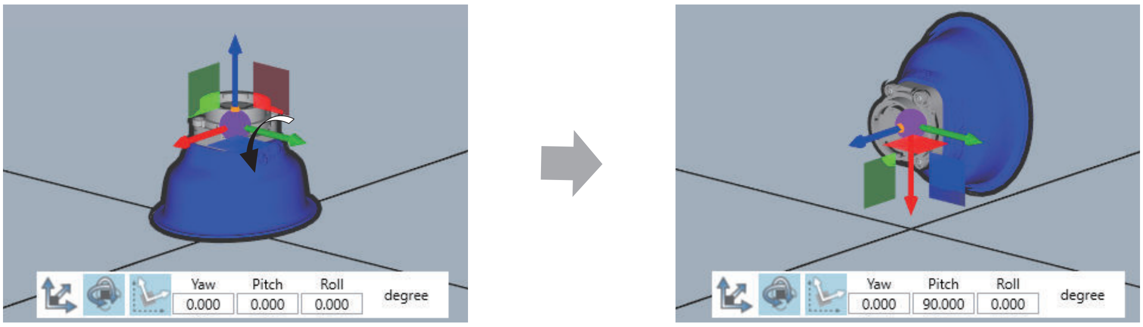
### ● Rotation around the Z Axis (Blue Arrow)

To rotate the 3D shape data 90 degrees around the Z axis, set the *y (yaw)* or *r (roll)* value to *90*. However, if you set the *y (yaw)* value to *90*, due to internal calculation, the *y (yaw)* and the *r (roll)* values are internally converted to *0* and *90*, respectively.




### ● Rotation around the Y Axis (Green Arrow)

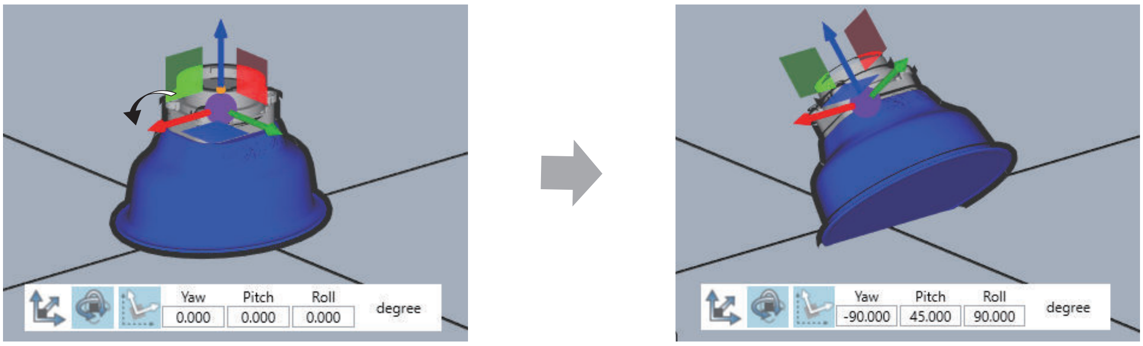
To rotate the 3D shape data 90 degrees around the Y axis, set the *p (pitch)* value to *90*.




## ● Rotation around the X Axis (Red Arrow)

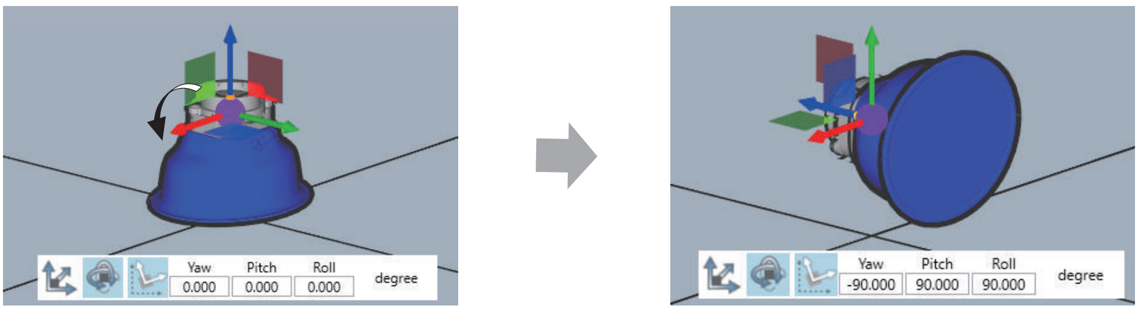
To rotate the 3D shape data around the X axis, unlike the Y and Z axes, you need to set values for two or more parameters.

- Rotating the 3D shape data 45 degrees around the X axis



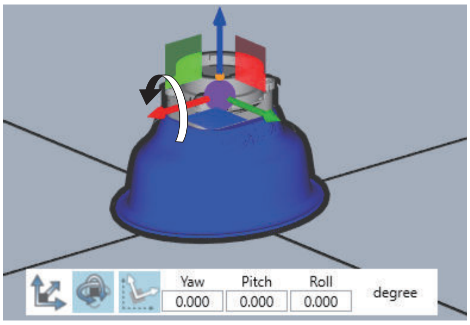

Set the *y (yaw)* value to *-90*, the *p (pitch)* value to *45*, and the *r (roll)* value to *90*.

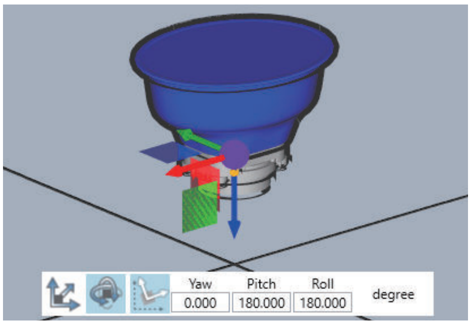
- Rotating the 3D shape data 90 degrees around the X axis




Set the *y (yaw)* value to *-90*, the *p (pitch)* value to *90*, and the *r (roll)* value to *90*.

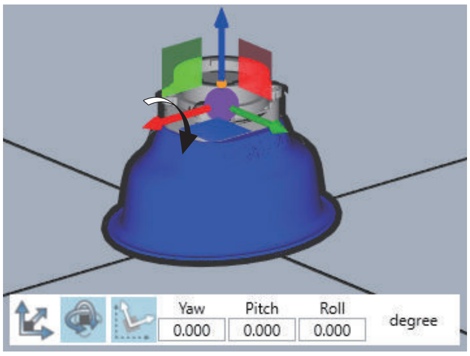
- Rotating the 3D shape data 180 degrees around the X axis

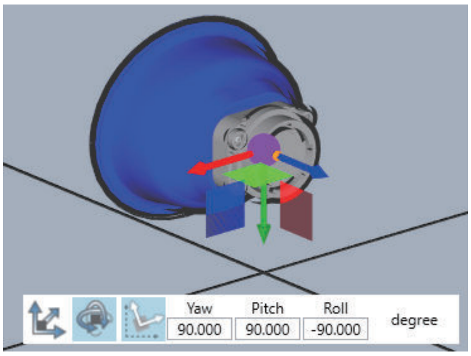



Set the *p (pitch)* value to *180* and the *r (roll)* value to *180*.

- Rotating the 3D shape data -90 degrees around the X axis

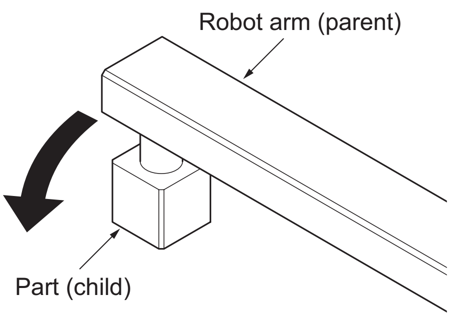



Set the *y (yaw)* value to *90*, *p (pitch)* value to *90*, and *r (roll)* value to *-90*.

## 4-1-2 Parent-child Relationship between 3D Shape Data

To achieve an operation in which one set of 3D shape data follows another set of 3D shape data as in the case of picking and then moving a part by a robot arm, set a parent-child relationship between the two sets of 3D shape data.
When more than one set of 3D shape data operates, the 3D shape data for the main operation is called *parent* and the 3D shape data for following the operation of the parent is called *child*.

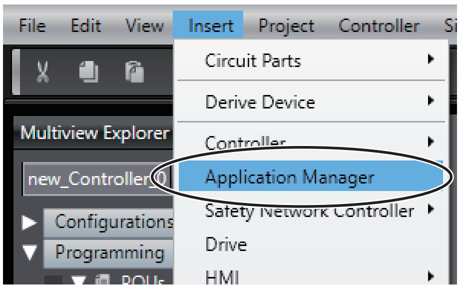



To set a parent-child relationship, select the parent 3D shape data on the setup tab page for the child 3D shape data.

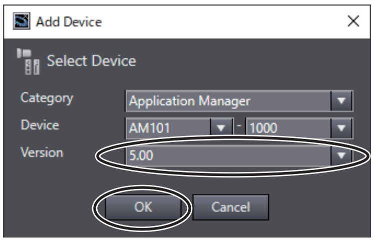
# 4-2 Adding Application Manager

Application Manager is a logical device that manages the data and settings required to use the 3D Simulation Function.
To create 3D shape data, add Application Manager to the project according to the following procedure.

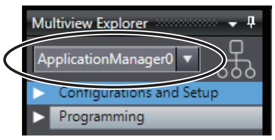
**1** Select **Application Manager** from the **Insert** menu.




The **Add Device** dialog box is displayed. In the dialog box, select the version to add from the **Version** list, and then click the **OK** button.

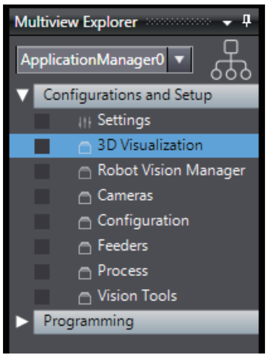



**Application Manager** is added to the device list in the Multiview Explorer.



**Additional Information**

Application Manager provides items for managing the IPC Application Controller settings and programs that control robot systems, in addition to items for managing the data and settings required to use the 3D Simulation Function.

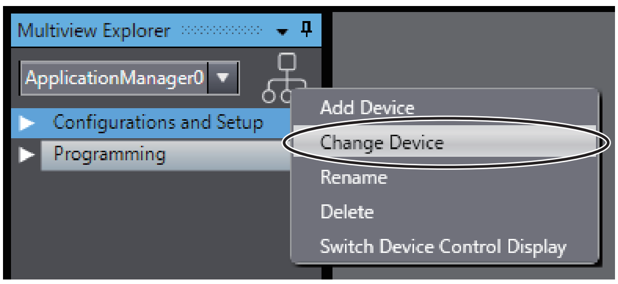



In **3D Visualization** under **Configurations and Setup**, manage the 3D shape data for virtual equipment models, settings for operating the mechanical components and Virtual Part Detection Sensor, and scripts that define the operations of the parts that you want to use for 3D simulations.
For simplicity, the following descriptions use only windows that you see when **3D Visualization** is selected.
Refer to the *Sysmac Studio Robot Integrated System Building Function with IPC Application Controller Operation Manual (Cat. No. W621)* for details on the items for managing the IPC Application Controller settings and programs.

## Checking and Changing the Application Manager Version

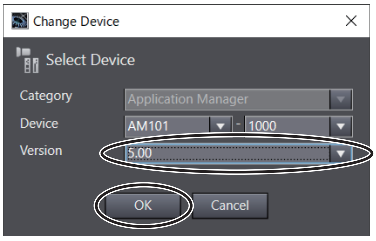
The procedure for checking and changing the version of the added Application Manager is as follows.

**1** Right-click the device icon for the project and select **Change Device** from the menu.




The **Change Device** dialog box is displayed. The **Version** value shown in the dialog box is the version of the Application Manager.
The **Change Device** dialog box also allows you to change the version. However, it is not possible to set the version lower than the current version.

**2** In the **Change Device** dialog box, select the version to change from the **Version** list, and then click the **OK** button.


Change Device
Select Device
Category
Application Manager
Device
AM101
1000
Version
5.00
OK
Cancel

# 4-3 Creating 3D Shape Data for the Mechanical Component

To create 3D shape data for a mechanical component, import CAD data for the movable parts of the equipment.
When you import the CAD data, select the applicable component from the component models provided in the Sysmac Studio and set how its movable parts operate.

**Additional Information**

Obtain the CAD data provided by the equipment and parts manufacturers in advance, or create CAD data with 3D CAD software.

## 4-3-1 Types of Supported CAD Data Files

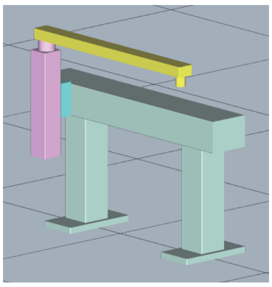
You can load the following types of CAD data files into the Sysmac Studio.

| CAD data type | Supported version |
|---|---|
| STEP | AP203, AP214, AP242 |
| IGES | Ver. 5.3 or later |
| OpenUSD | OpenUSD that supports OpenUSD 23.11 SDK |

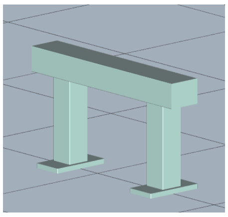
## 4-3-2 Preparations for CAD Data Files

To import CAD data files for mechanical component parts, create CAD data files for the individual parts that make up the component depending on a mechanical component.
Refer to *4-3-4 Types of Mechanical Component Models* on page 4-19 for details on types of mechanical components supported by the 3D Simulation Function.

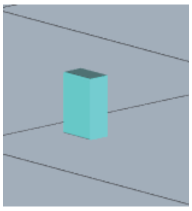
Example: Movable part as an X-Y table (XY Theta)



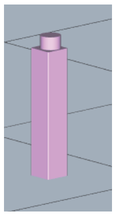
To import CAD data for movable part that consists of a base, an X-axis slider, a Z-axis slider, and an arm with a rotary axis as the mechanical component *X-Y table (XY Theta)*, create a CAD data file for each of the base, X-axis slider, Z-axis slider, and arm with a rotation axis. (In this example, Z axis is applied to the Y axis of the X-Y table (XY Theta).)



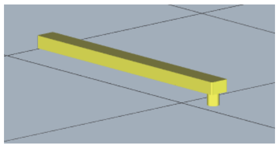
Base
(XYThetaTable_base.step)



X-axis slider
(XYThetaTable_X.step)



Z-axis slider
(XYThetaTable_Y.step)



Arm with a rotary axis
(XYThetaTable_theta.step)

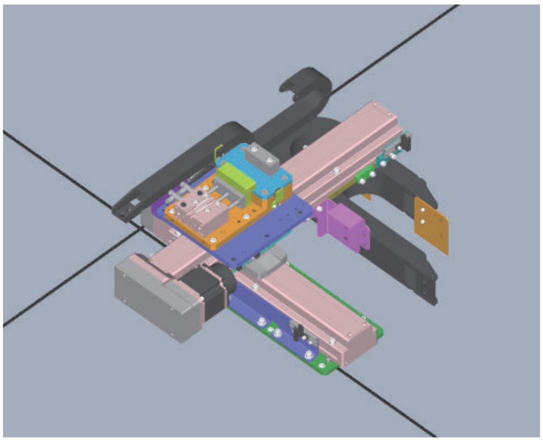
## Procedure to Output CAD Data Files with 3D CAD Software

A mechanical component consists of a base that serves as a foundation and movable parts that operate according to outputs from an axes or I/O. To import a mechanical component to the 3D Visualizer, you need to prepare CAD data files for the base and movable parts that are output from 3D CAD software.

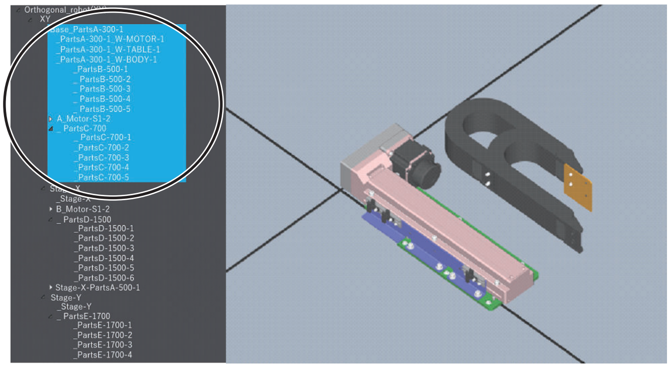
### Additional Information

- If there are CAD data files that are created in the mechanical design stage, use the 3D CAD software to create CAD data files for the mechanical component based on the CAD data files. Then, create CAD data files for the base and movable parts.
- When you open a CAD data file in 3D CAD software, the assembly information is displayed in a tree structure. In most 3D CAD software, you can select the assembly information needed for the base or movable parts from the tree structure to output CAD data files for the selected parts. Import these CAD data files as the base or movable parts of the mechanical component.

The following is an example of using 3D CAD software to output the CAD data files for an orthogonal robot (XY) to three CAD data files for the base, movable part *X stage*, and movable part *Y stage*.

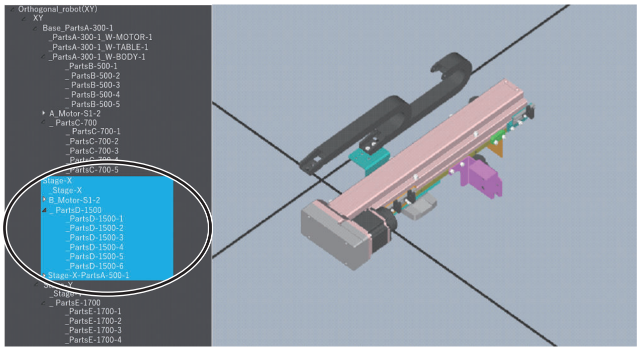


**1** From the assembly information displayed in the 3D CAD software, select the assembly information that corresponds to the base and output only the selected information to a CAD data file.
The following is an example of selecting assembly information to display only the selected assembly information in the 3D Visualizer.



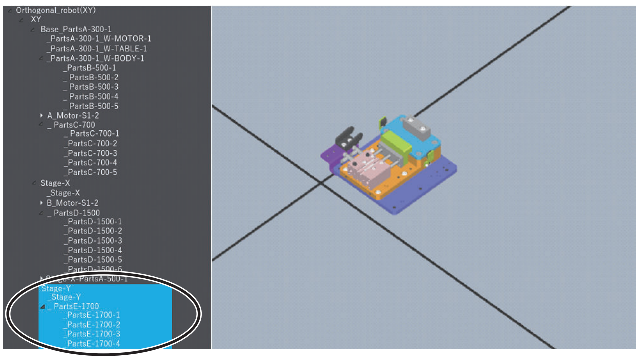
**2** Select the assembly information that corresponds to the movable part *X stage* and output only the selected information to a CAD data file.

The following is an example of selecting assembly information to display only the selected assembly information in the 3D Visualizer.



**3** Select the assembly information that corresponds to the movable part *Y stage* and output only the selected information to a CAD data file.

The following is an example of selecting assembly information to display only the selected assembly information in the 3D Visualizer.



### Precautions for Correct Use

Depending on the assembly structure of CAD data files, the assembly information may not properly correspond to the base or movable parts as shown in the example. Use 3D CAD software to select movable parts accurately and output the CAD data files for the base part or movable parts.

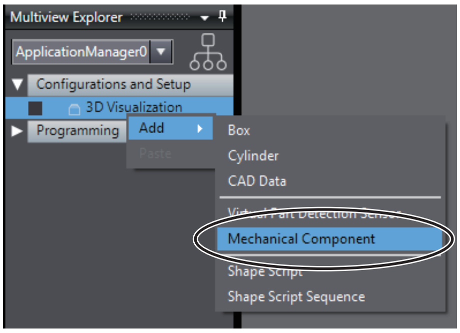
## 4-3-3 CAD Data Import Procedure

Import the CAD data file that you prepared. Use the following import procedure.

**Additional Information**

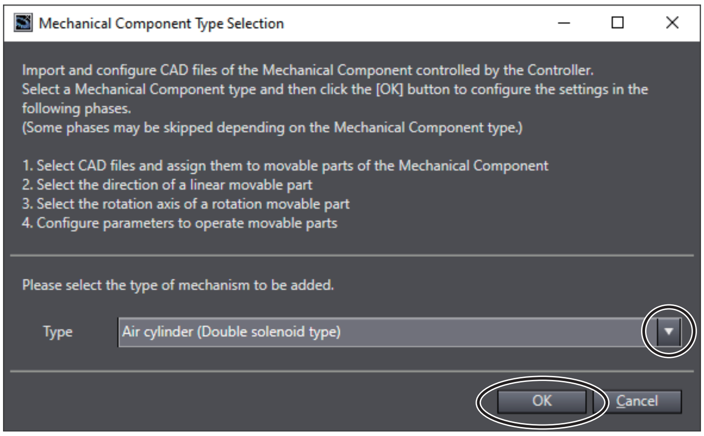
You can optionally set the mesh coarseness of CAD data at the time of import. Refer to the *Sysmac Studio Robot Integrated System Building Function with Robot Integrated CPU Unit Operation Manual (Cat. No. W595)* for details on the option settings.

**1** Right-click **3D Visualization** under **Configurations and Setup** and select **Add** - **Mechanical Component** from the menu.



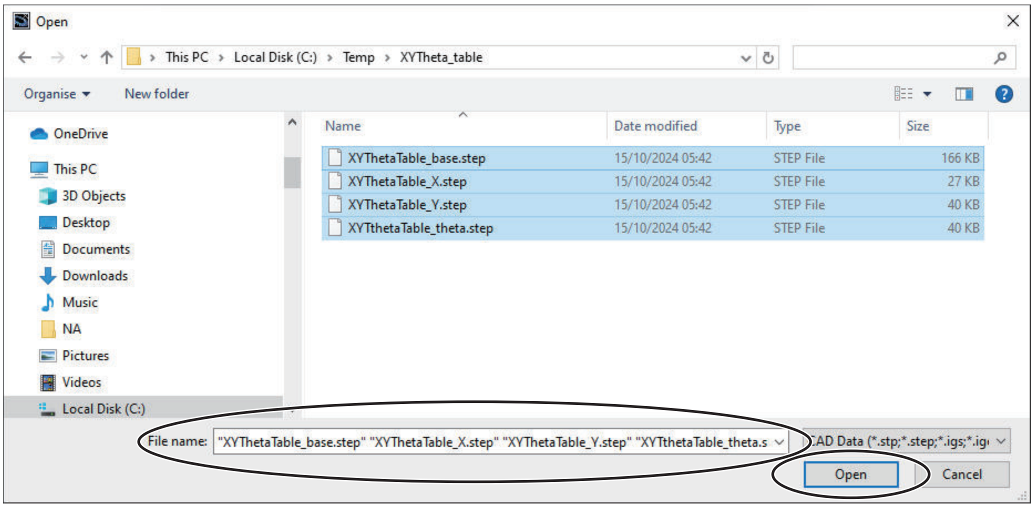

The **Mechanical Component Type Selection** dialog box is displayed.

**2** Select the type of the mechanical component, and then click the **OK** button.




If you specify a type other than **Conveyor**, the **Open** dialog box is displayed.

**3** Select a CAD file with a .stp, .step, .igs, .iges, .usd, .usda, .usdc, or .usdz file name extension, and then click the **Open** button.




The **Mechanical Component Adding Wizard** is displayed. The steps of the wizard are described in the following table.

| Step | Navigation | Description |
|---|---|---|
| 1 | Movable Parts Selection | Assign each CAD file to the movable parts that make up a mechanical component. |
| 2 | Linear Direction Selection | Set the moving direction of the linear parts that make up a mechanical component. |
| 3 | Rotate Axis Selection | Set the rotation direction of the rotation axis that makes up a mechanical component. |
| 4 | Parameter Setting | Configure the parameters of the movable parts that make up a mechanical component. |

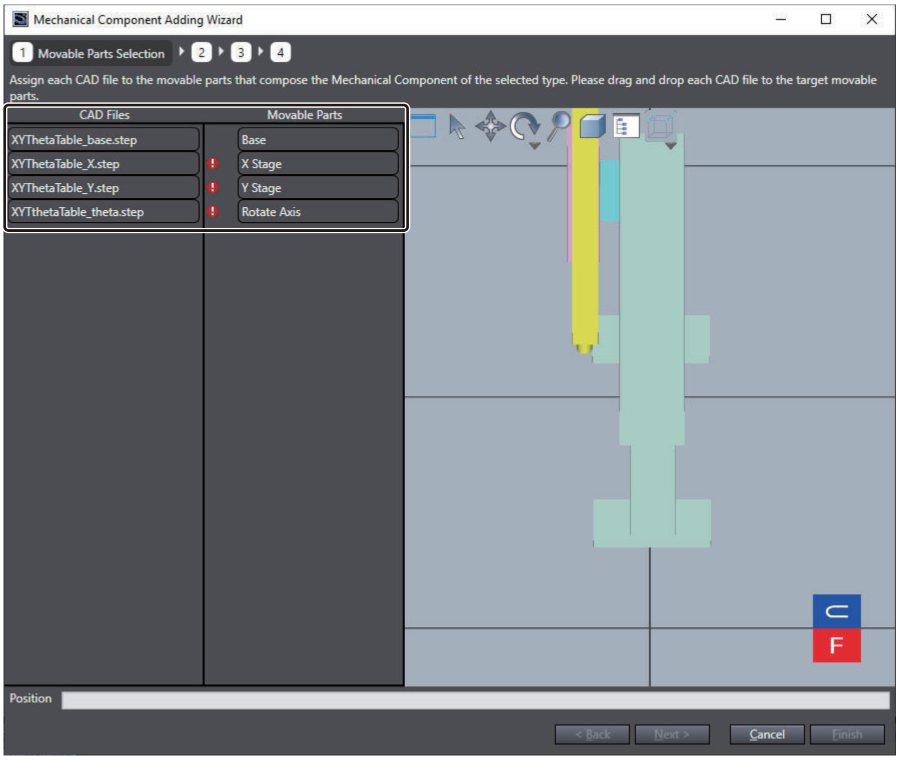
Depending on types of mechanical components, steps without any settings are omitted. Steps that are omitted are grayed out.

| Step | Navigation | Type of Mechanical Component for which the step is omitted |
|---|---|---|
| 1 | Movable Parts Selection | Conveyor |
| 2 | Linear Direction Selection | Motor rotation |
| 3 | Rotate Axis Selection | • Single axis position control<br>• Air cylinder (Single solenoid type)<br>• Air cylinder (Double solenoid type)<br>• Robot tool (Parallel switching 2-finger type chuck/single solenoid type)<br>• Robot tool (Parallel switching 2-finger type chuck/double solenoid type)<br>• Orthogonal robot (XY)<br>• Orthogonal robot (XYZ)<br>• Conveyor |
| 4 | Parameter Setting | None |

The operation in each step is described below, using the X-Y table (XY Theta) model as an example.

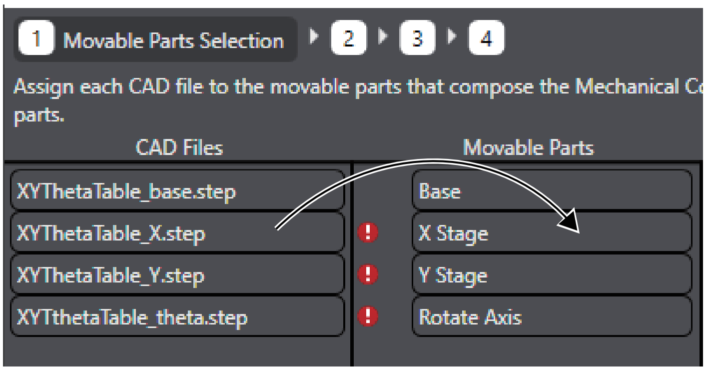
## Step 1: Movable Parts Selection

Assign each CAD file to the movable parts that make up a mechanical component.

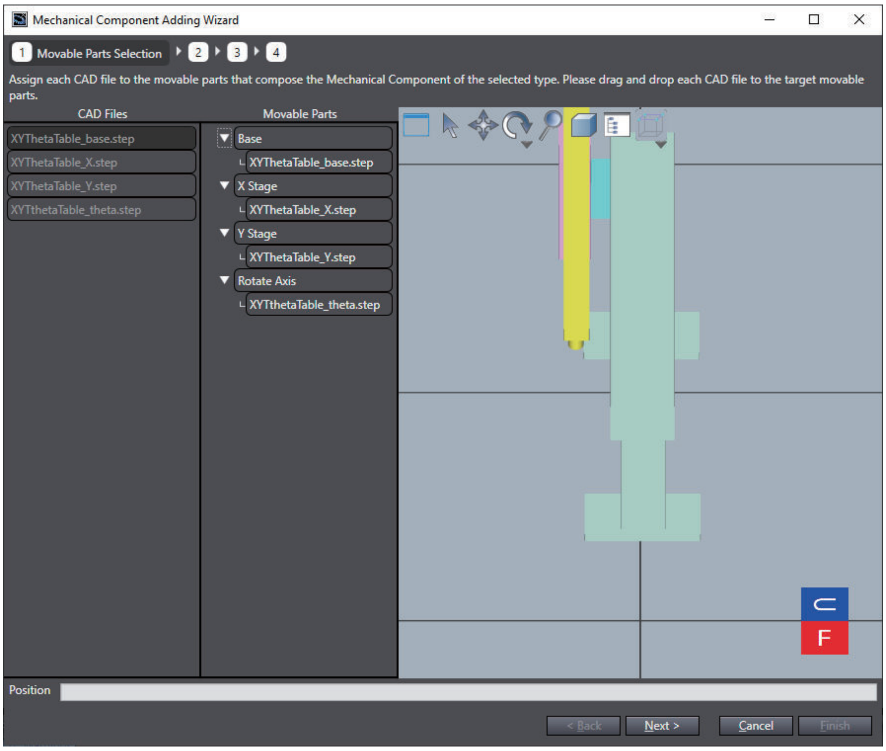



### ● Procedure 1: Assigning CAD Files to the Movable Parts

Drag and drop each CAD file to the target movable parts.




The CAD files are assigned and displayed under the movable parts.



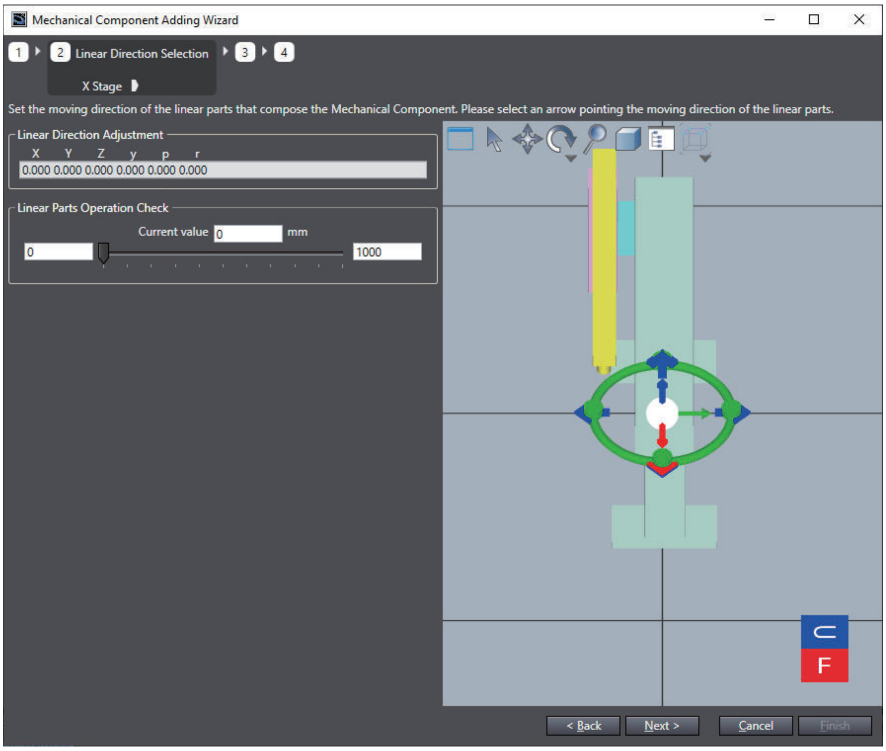

● **Procedure 2: Setting the Position of the Movable Parts**

In **Position**, enter the pose and position of the target movable parts. The location elements are X, Y, Z, y (yaw), p (pitch), and r (roll), from left to right.

After completion of all the settings, click the **Next** button.

## Step 2: Linear Direction Selection

Set the moving direction of the linear parts that make up a mechanical component.




- **Procedure 1: Setting the Linear Direction of the Movable Parts**

  Click one of the four blue arrow icons on the 3D Visualizer to select the linear direction of the movable parts. The selected icon turns red.

  For the conveyor, only the arrow in the X direction is displayed.

- **Procedure 2: Setting the Local Coordinate Home of the Movable Parts**

  Drag and move the white sphere on the 3D Visualizer to set the local coordinate origin of the movable parts. Moving the sphere updates the coordinate values in **Linear Direction Adjustment**. Alternatively, you can enter the coordinate values in **Linear Direction Adjustment**.

- **Procedure 3: Checking the Operation of the Linear Parts**

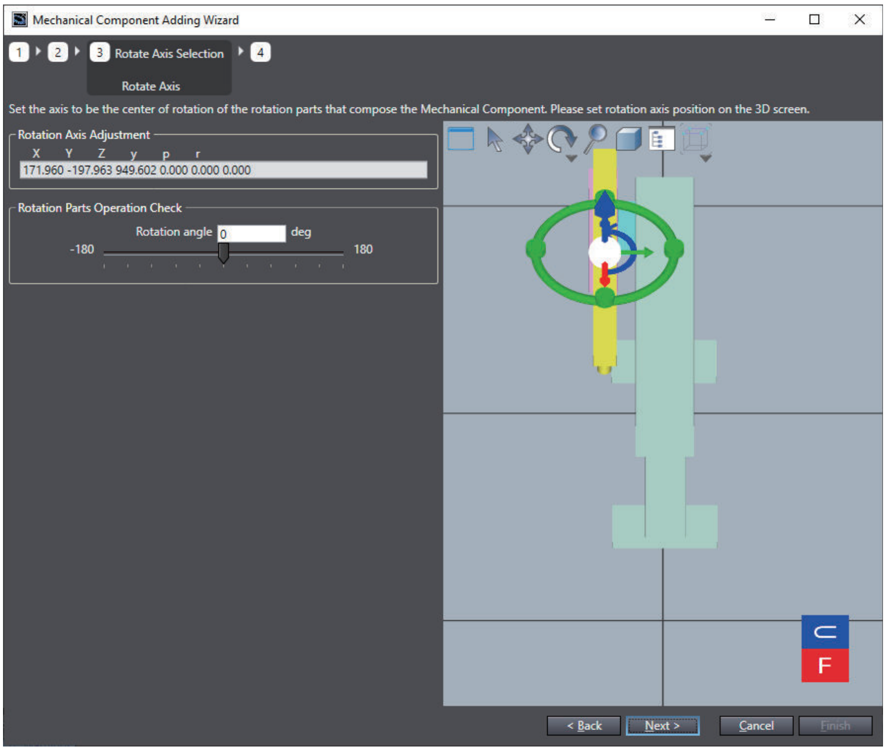
  Move the slider in **Linear Parts Operation Check** to check the operation of the linear parts on the 3D Visualizer.

  For the conveyor, the **Linear Parts Operation Check** items are not displayed.

  After completion of all the settings, click the **Next** button.

## Step 3: Rotate Axis Selection

Set the rotation direction of a rotation axis that makes up a mechanical component.




- **Procedure 1: Setting the Rotation Direction of the Movable Parts**

  In **Rotation Axis Adjustment**, enter the y, p, and r values to set the rotation axis direction. Entering the values changes the direction of the Edit Rotate Axis Direction icon on the 3D Visualizer. Alternatively, you can drag the Edit Rotate Axis Direction icon on the 3D Visualizer to set the rotation axis direction.

- **Procedure 2: Setting the Center of Rotation Axis of the Movable Parts**

  Drag and move the white sphere on the 3D Visualizer to set the center of rotation axis of a rotation part. Moving the sphere updates the coordinate values in **Rotation Axis Adjustment**. Alternatively, you can directly change and set the coordinate values in **Rotation Axis Adjustment**.

- **Procedure 3: Checking the Operation of the Linear Parts**

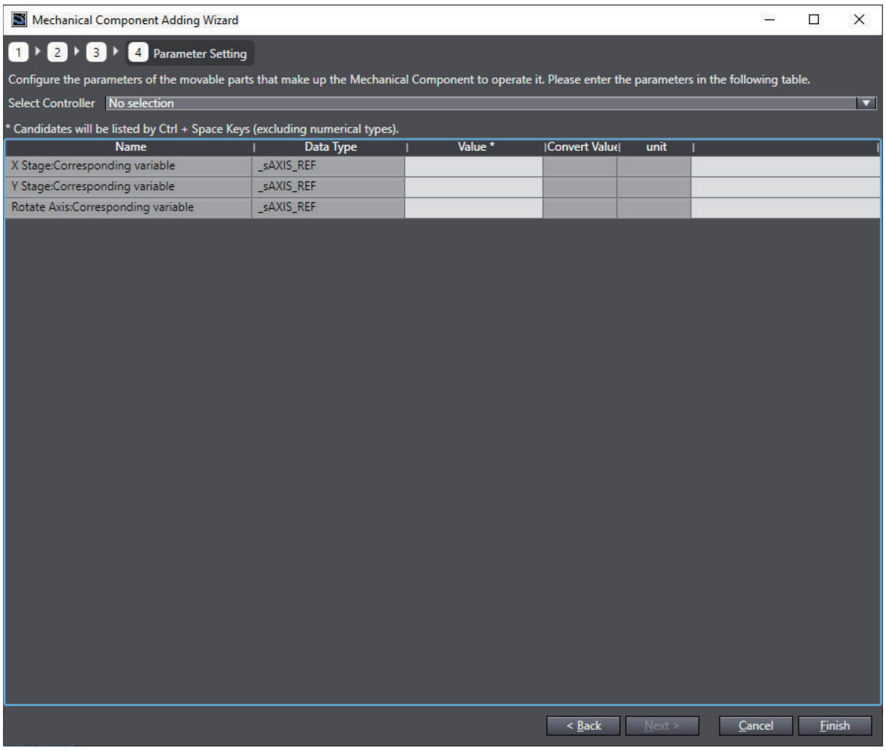
  Move the slider in **Rotation Parts Operation Check** to check the operation of the rotation parts on the 3D Visualizer.

  After completion of all the settings, click the **Next** button.

## Step 4: Parameter Setting

To operate a mechanical component, configure the parameters of the movable parts that make up the mechanical component. The parameters of the movable parts depend on the type of the mechanical component.

Refer to *4-3-4 Types of Mechanical Component Models* on page 4-19 for details on mechanical component parameters.

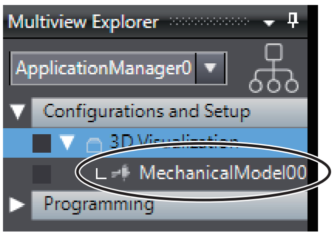



## ● Procedure 1: Setting the Variables According to the Data Type

In **Select Controller**, select the Controller to control the movable parts. Then, in the **Set value** text box for each variable, enter the corresponding axis variable. Enter a variable name directly into each text box or, with the text box enabled for entry, press the Ctrl + Space keys and select a variable name from the candidate list.

## ● Procedure 2: Setting the Convert Coefficients

If the **Convert coefficient** text boxes are enabled, enter the convert coefficients.
After completion of all the settings, click the **Finish** button.
**MechanicalModel000** is registered and displayed under **3D Visualization**.




# 4-3-4 Types of Mechanical Component Models

The following mechanical components are supported in the Sysmac Studio. To set a mechanical component, use the Mechanical Component Adding Wizard. Refer to *4-3-3 CAD Data Import Procedure* on page 4-13 for details.

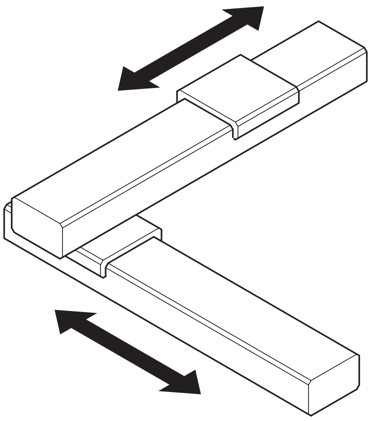
- Orthogonal robot (XY)
- Orthogonal robot (XYZ)
- X-Y-Z stage + rotation axis (upward direction)

- X-Y-Z stage + rotation axis (downward direction)
- X-Y table (XY Theta)
- X-Y table (Theta XY)
- Single axis position control
- Motor rotation
- Air cylinder (Single solenoid type)
- Air cylinder (Double solenoid type)
- Robot tool (Parallel switching 2-finger type chuck/single solenoid type)
- Robot tool (Parallel switching 2-finger type chuck/double solenoid type)
- Conveyor

The parameters that you set for each component are given in the following pages.

## Orthogonal Robot (XY)

Orthogonal robot (XY) refers to a component that can move to any position on a plane.



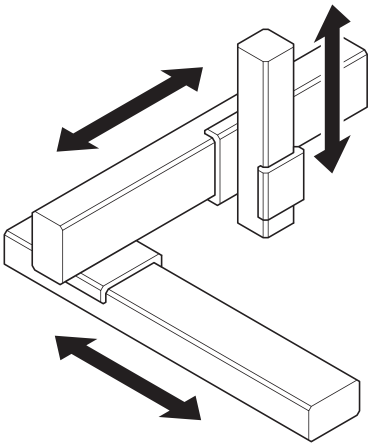
For the Orthogonal robot (XY) model, set the following parameters.
Set the corresponding variable for the axes group, or set the corresponding variables for the X stage and Y stage.

| Item | Description | Set value |
|---|---|---|
| Axes Group: Corresponding variable*1*2 | The axes group settings assigned to the Orthogonal robot (XY) model. | Variables of _sGROUP_REF data type |
| X Stage: Corresponding variable | Set the corresponding axis of the X stage. | Variables of _sAXIS_REF data type |
| Y Stage: Corresponding variable | Set the corresponding axis of the Y stage. | Variables of _sAXIS_REF data type |

*1. An error will occur if Axes Group Use in Axes Group is set to **Unused axes group**.
*2. When this parameter is set, the axis A0 is assigned to the X stage and the axis A1 to the Y stage in the axes groups.

## Orthogonal Robot (XYZ)

Orthogonal robot (XYZ) refers to a component that can move to any position in a 3D space.



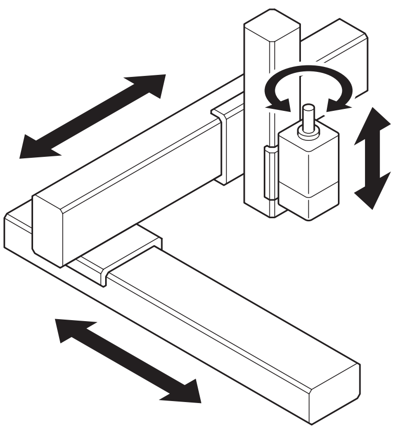
For the Orthogonal robot (XYZ) model, set the following parameters.
Set the corresponding variable for the axes group, or set the corresponding variables for the X stage, Y stage, and Z stage.

| Item | Description | Set value |
|---|---|---|
| Axes Group: Corresponding variable*1*2 | The axes group settings assigned to the Orthogonal robot (XYZ) model. | Variables of _sGROUP_REF data type |
| X Stage: Corresponding variable | Set the corresponding axis of the X stage. | Variables of _sAXIS_REF data type |
| Y Stage: Corresponding variable | Set the corresponding axis of the Y stage. | Variables of _sAXIS_REF data type |
| Z Stage: Corresponding variable | Set the corresponding axis of the Z stage. | Variables of _sAXIS_REF data type |

*1. An error will occur if Axes Group Use in Axes Group is set to **Unused axes group**.
*2. When this parameter is set, the axis A0 is assigned to the X stage, the axis A1 to the Y stage, and axis A2 to the Z stage in the axes groups.

## X-Y-Z Stage + Rotation Axis (Upward Direction)

X-Y-Z stage + rotation axis (upward direction) refers to a component in which an upward rotation axis is attached to the tip of the X-Y-Z stage.

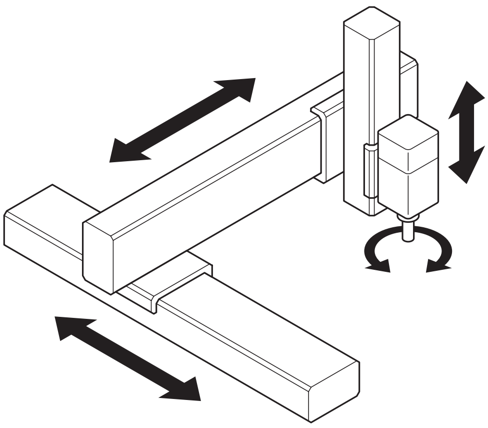


For the X-Y-Z stage + rotation axis (upward direction) model, set the following parameters.

| Item | Description | Set value |
|---|---|---|
| X Stage: Corresponding variable | Set the corresponding axis of the X stage. | Variables of _sAXIS_REF data type |
| Y Stage: Corresponding variable | Set the corresponding axis of the Y stage. | Variables of _sAXIS_REF data type |
| Z Stage: Corresponding variable | Set the corresponding axis of the Z stage. | Variables of _sAXIS_REF data type |
| Rotate Axis: Corresponding variable | Set the corresponding axis of the rotation axis. | Variables of _sAXIS_REF data type |

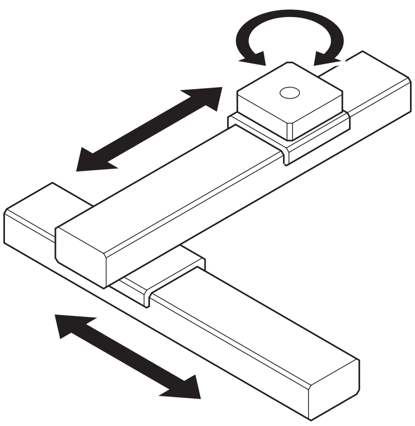
## X-Y-Z Stage + Rotation Axis (Downward Direction)

X-Y-Z stage + rotation axis (downward direction) refers to a component in which an downward rotation axis is attached to the tip of the X-Y-Z stage.
Set the same parameters as X-Y-Z stage + rotation axis (upward direction). Refer to *X-Y-Z Stage + Rotation Axis (Upward Direction)* on page 4-21.



## X-Y Table (XY Theta)

X-Y table (XY Theta) refers to an X-Y table that has a rotation component on the top.

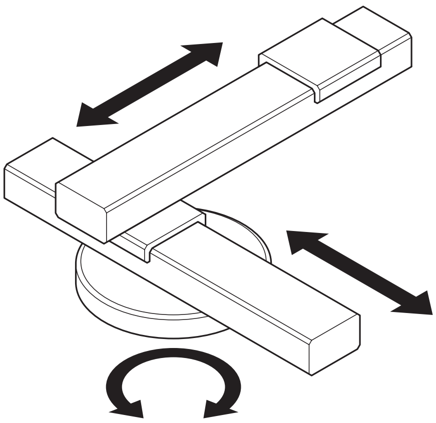


For the X-Y table (XY Theta) model, set the following parameters.

| Item | Description | Set value |
|---|---|---|
| X Stage: Corresponding variable | Set the corresponding axis of the X stage. | Variables of _sAXIS_REF data type |
| Y Stage: Corresponding variable | Set the corresponding axis of the Y stage. | Variables of _sAXIS_REF data type |
| Rotate Axis: Corresponding variable | Set the corresponding axis of the rotation axis. | Variables of _sAXIS_REF data type |

## X-Y Table (Theta XY)

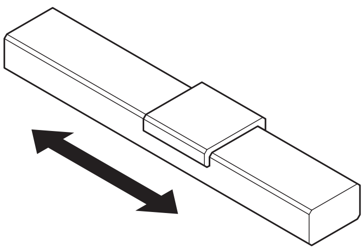
X-Y table (Theta XY) refers to a rotation component that has an X-Y table on the top.



Set the same parameters as X-Y table (XY Theta). Refer to *X-Y Table (XY Theta)* on page 4-22.

## Single Axis Position Control

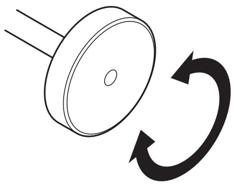
Single axis position control refers to a component that can move to any position in a straight line.
For the single axis position control model, set the following parameters.



| Item | Description | Set value |
|---|---|---|
| Y Stage: Corresponding variable | Set the corresponding axis of the Y stage. | Variables of _sAXIS_REF data type |

## Motor Rotation

Motor rotation refers to a component that visualizes the rotation direction of a motor.

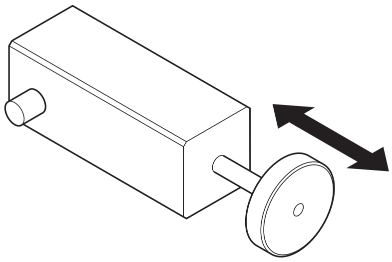


For the Motor rotation model, set the following parameters.

| Item | Description | Set value |
|---|---|---|
| Motor: Corresponding variable | Set the corresponding axis of the motor. | Variables of _sAXIS_REF data type |

## Air Cylinder (Single Solenoid Type)

Air cylinder (Single solenoid type) refers to a component whose piston moves according to a single input value.

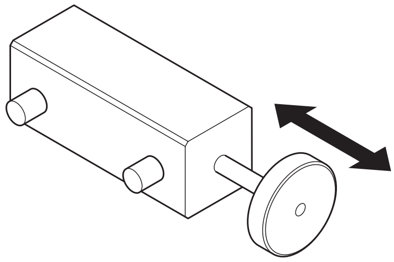


For the Air cylinder (Single solenoid type) model, set the following parameters.

| Item | Description | Set value | Initial value |
|---|---|---|---|
| Air cylinder: Corresponding variable | Set the I/O variable by which air injection to the air cylinder is started and stopped. | BOOL global variable | --- |
| Virtual output (Advance position detection): Corresponding variable | Set the I/O variable that is turned ON when the piston is completely extended. | BOOL global variable | --- |
| Virtual output (Return position detection): Corresponding variable | Set the I/O variable that is turned ON when the piston is completely returned. | BOOL global variable | --- |
| Air cylinder: Cylinder travel time | Set the cylinder travel time. (Unit: s) | 0.0 to 100.0 | 0.5 |
| Air cylinder: Cylinder length | Set the travel distance when the corresponding variable changes from ON to OFF or from OFF to ON. (Unit: mm) | 0 to 35,000,000,000,000 | 100 |
| Air cylinder: Cylinder type | Set the operation method of the air cylinder. | Advance/Return | Advance |

## Air Cylinder (Double Solenoid Type)

Air cylinder (Double solenoid type) refers to a component whose piston moves according to two input values.

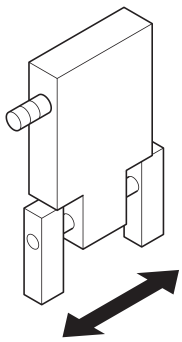


For the Air cylinder (Double solenoid type) model, set the following parameters.

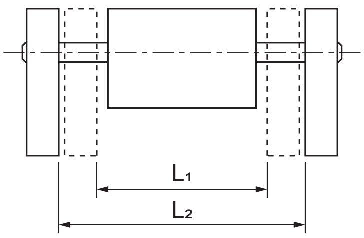
| Item | Description | Set value | Initial value |
|---|---|---|---|
| Advance switch: Corresponding variable | Set a variable by which the piston is extended when the air cylinder is returned. | BOOL global variable | --- |
| Return switch: Corresponding variable | Set a variable by which the piston is returned when the air cylinder is extended. | BOOL global variable | --- |
| Virtual output (Advance position detection): Corresponding variable | Set the I/O variable that is turned ON when the piston is completely extended. | BOOL global variable | --- |
| Virtual output (Return position detection): Corresponding variable | Set the I/O variable that is turned ON when the piston is completely returned. | BOOL global variable | --- |
| Air cylinder: Cylinder travel time | Set the Cylinder travel time. (Unit: s) | 0.0 to 100.0 | 0.5 |
| Air cylinder: Cylinder length | Set the travel distance when the corresponding variable changes from ON to OFF or from OFF to ON. (Unit: mm) | 0 to 35,000,000,000,000 | 100 |
| Air cylinder: Initial status of the piston | Set the initial status of the piston in the air cylinder. | Advance/Return | Return |

## Robot Tool (Parallel Switching 2-finger Type Chuck/Single Solenoid Type)

Robot tool (Parallel switching 2-finger type chuck/single solenoid type) refers to a component in which a chuck attached to the tip of a robot is operated by injecting air to enable the robot to pick parts.

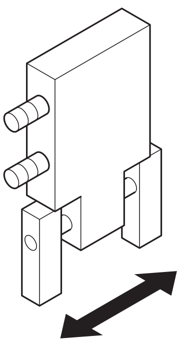


For the Robot tool (Parallel switching 2-finger type chuck/single solenoid type) model, set the following parameters.

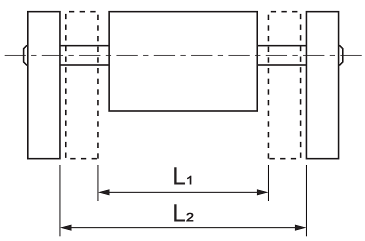
| Item | Description | Set value | Initial value |
|---|---|---|---|
| Chuck: Corresponding variable | Set the I/O variable by which air injection to the chuck is started and stopped. | BOOL global variable | --- |
| Virtual output (Open position detection): Corresponding variable | Set the I/O variable that is turned ON when the chuck is completely opened. | BOOL global variable | --- |
| Virtual output (Close position detection): Corresponding variable | Set the I/O variable that is turned ON when the chuck is completely closed. | BOOL global variable | --- |
| Chuck: Stroke time | Set the time until the chuck is completely opened/closed. (Unit: s) | 0.0 to 100.0 | 0.5 |
| Chuck: Open/close stroke width | Set the open/close stroke width of the chuck. (Unit: mm)<br>Set the value of L2 minus L1 in the figure below.<br><br> | 0 to 100 | 10 |
| Chuck: Operation | Set the operation method of the chuck. | Normally Opened/ Normally Closed | Normally Closed |

## Robot Tool (Parallel Switching 2-finger Type Chuck/Double Solenoid Type)

Robot tool (Parallel switching 2-finger type chuck/double solenoid type) refers to a component in which a chuck attached to the tip of a robot is opened and closed by injecting air to enable the robot to pick parts.

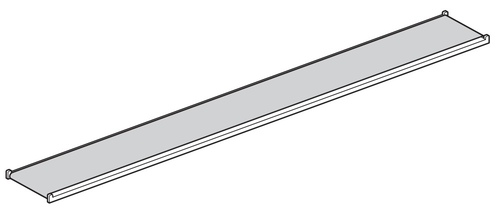


For the Robot tool (Parallel switching 2-finger type chuck/double solenoid type) model, set the following parameters.

| Item | Description | Set value | Initial value |
|---|---|---|---|
| Input to Open direction: Corresponding variable | Set the I/O variable by which air injection in the direction to open the chuck is started and stopped. | BOOL global variable | --- |
| Input to Close direction: Corresponding variable | Set the I/O variable by which air injection in the direction to close the chuck is started and stopped. | BOOL global variable | --- |
| Virtual output (Open position detection): Corresponding variable | Set the I/O variable that is turned ON when the chuck is completely opened. | BOOL global variable | --- |
| Virtual output (Close position detection): Corresponding variable | Set the I/O variable that is turned ON when the chuck is completely closed. | BOOL global variable | --- |
| Chuck: Stroke Time | Set the time until the chuck is completely opened/closed. (Unit: s) | 0.0 to 100.0 | 0.5 |
| Chuck: Open/Close Stroke Width | Set the open/close stroke width of the chuck. (Unit: mm)<br>Set the value of L2 minus L1 in the figure below.<br><br> | 0 to 100 | 10 |
| Chuck: Initial status | Set the initial status of the chuck. | Open/Close | Open |

## Conveyor

Conveyor is a component that you can assign to a motion axis in order to carry parts.



**Version Information**

The conveyor is supported by Application Manager version 5.0 or higher.
For the conveyor, set the following parameters.

| Item | Description | Set value | Initial value |
|---|---|---|---|
| Conveyor: Corresponding variable | Set the axis to assign to the conveyor. | _sAXIS_REF global variable | --- |
| Conveyor: Length | Set the length of the conveyor. (Unit: mm) | 100 to 1,000,000 | 1,800 |
| Conveyor: Width | Set the width of the conveyor. (Unit: mm) | 100 to 1,000,000 | 100 |

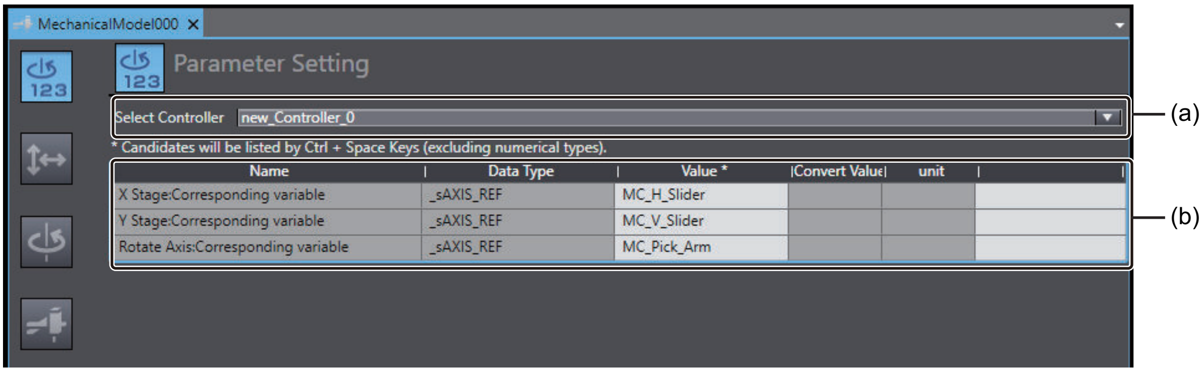
## 4-3-5 Mechanical Component Settings

Double-click the mechanical component that you added to display the setup tab page for the mechanical component.
The following settings are provided:

- Parameter Settings
- Linear Direction Settings
- Rotate Axis Settings
- Mechanical Component Common Settings

### Parameter Settings

Select the Controller that controls a mechanical component. Then assign axis variables to each corresponding variable.

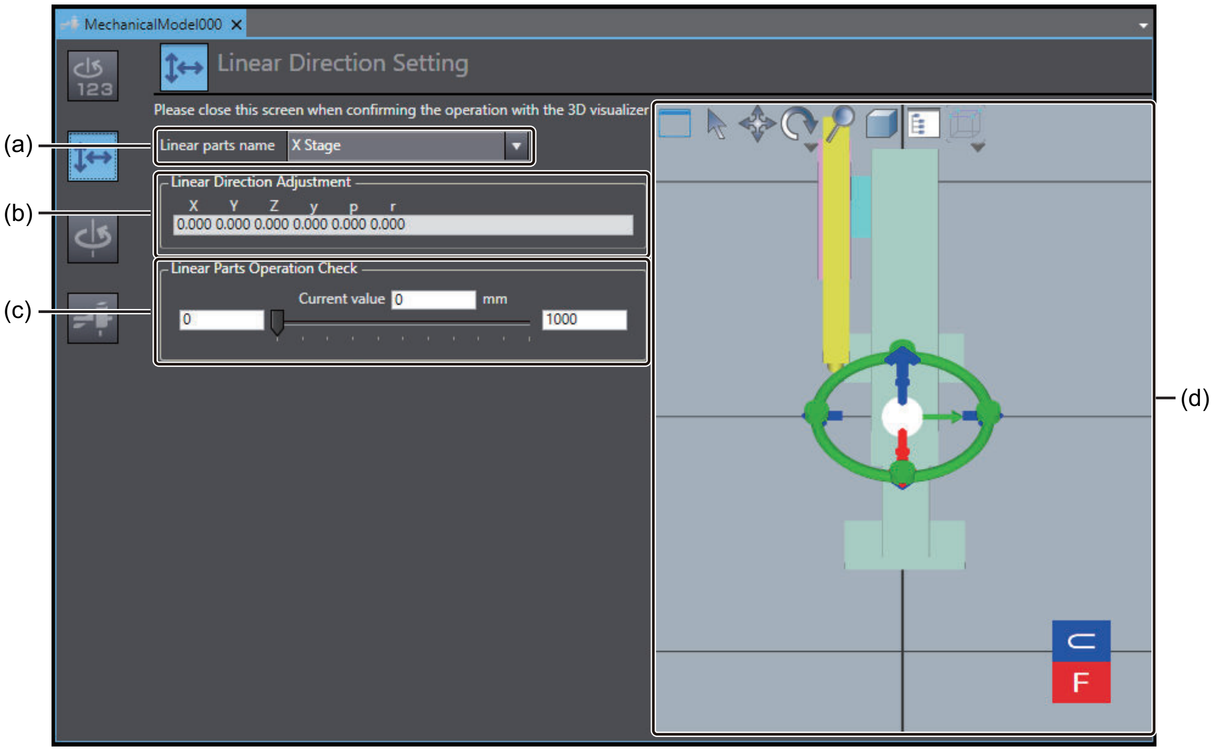



| | Item | Description | Set value | Initial value*1 |
|---|---|---|---|---|
| (a) | Select Controller | Select the Controller to control a mechanical component. | Controller name | No selection |
| (b) | Axis variable settings | Assign the axis variables to control the target movable parts of the mechanical component. | Axis variables for the selected Controller | None |

*1. The values that you set in step 4 of the **Mechanical Component Adding Wizard** are displayed.

## Linear Direction Settings

Among the movable parts of a mechanical component, set the position, pose, and operating range of the parts that make a linear operation.

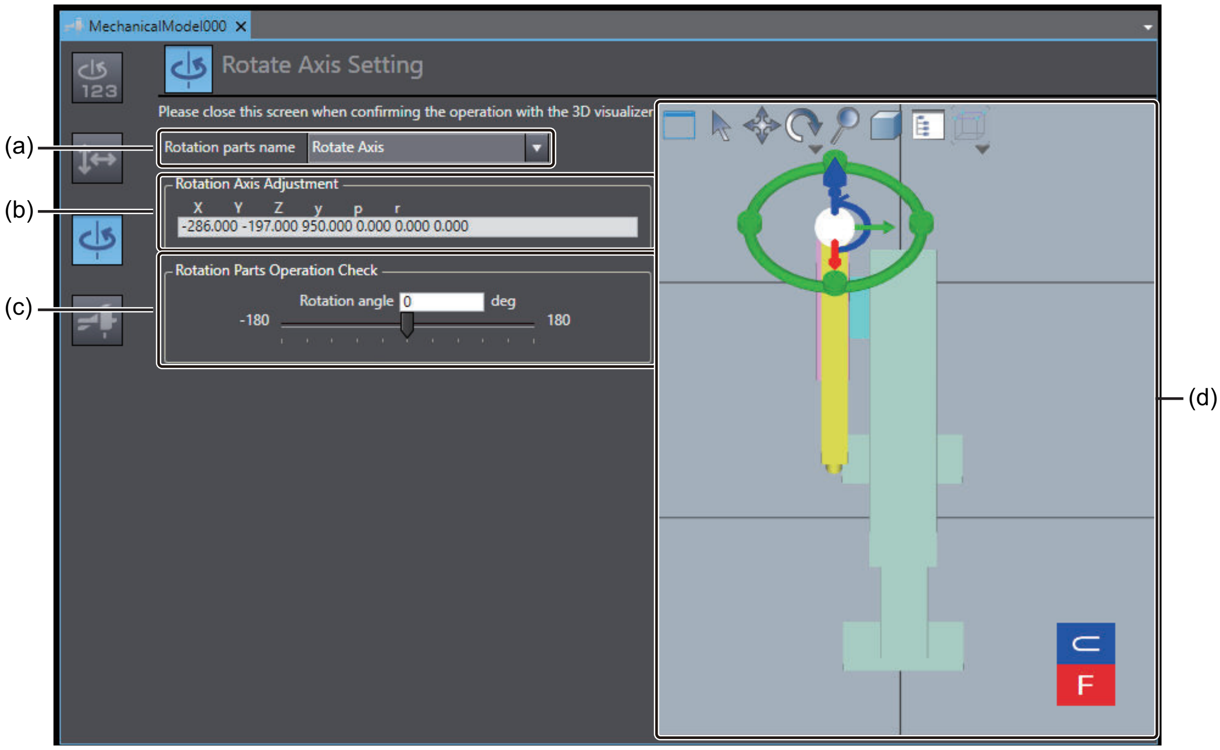



| | Item*1 | Description | Set value | Initial value*2 |
|---|---|---|---|---|
| (a) | Linear parts name | Among the movable parts of a mechanical component, select the parts that make a linear operation. | Target movable parts | --- |
| (b) | Linear Direction Adjustment | Set the start position of operation and posture of the linear parts. The location elements are X, Y, Z, y (yaw), p (pitch), and r (roll), from left to right. | X, Y, Z: For each, -1,000,000.000 to 1,000,000.000<br>y, r: For each, -180.000 to 180.000<br>p: 0.000 to 180.000 | 0.000 for all |
| (c) | Linear Parts Operation Check | Move the slider to check the operation of the linear parts. | -10,000,000,000,000 to 10,000,000,000,000 | Minimum value: 0<br>Maximum value: 1,000 |
| (d) | 3D editing area | Set the position and pose of the linear parts. The linear parts move according to the operation that you make in **Linear Parts Operation Check**. | --- | --- |

*1. For mechanical components without parts that make a linear operation, the items display no value.
*2. The values that you set in step 2 of the **Mechanical Component Adding Wizard** are displayed.

## Rotation Direction Settings

Among the movable parts of a mechanical component, set the position, pose, and operating range of the parts that make a rotational operation.

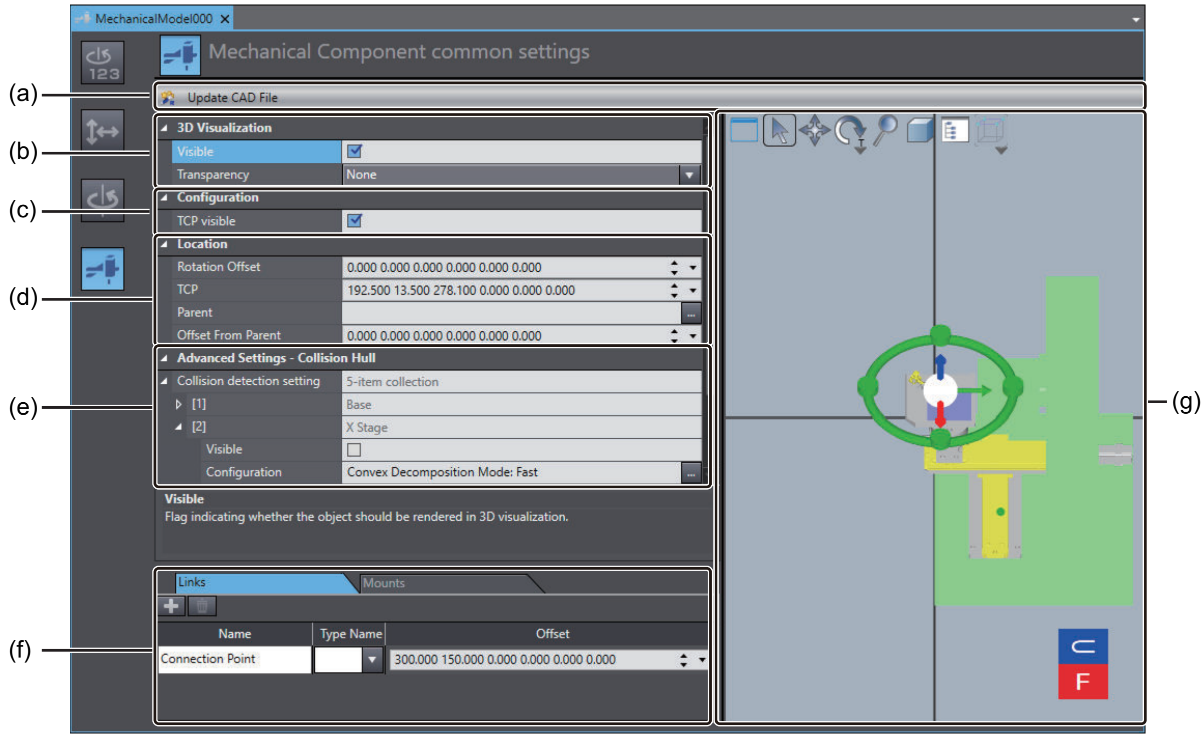



| | Item*1 | Description | Set value | Initial value*2 |
|---|---|---|---|---|
| (a) | Rotation parts name | Among the movable parts of a mechanical component, select the parts that make a rotational operation. | Target movable parts | --- |
| (b) | Rotation Axis Adjustment | Set the start position of operation and posture of the rotation parts. The location elements are X, Y, Z, y (yaw), p (pitch), and r (roll), from left to right. | For each, -1,000,000.000 to 1,000,000.000 | 0.000 for all |
| (c) | Rotation Parts Operation Check | Move the slider to check the operation of the rotation parts. | --- | --- |
| (d) | 3D editing area | Set the position and pose of the rotation parts. The rotation parts move according to the operation that you make in **Rotation Parts Operation Check**. | --- | --- |

*1. For mechanical components without parts that make a rotational operation, the items display no value.
*2. The values that you set in step 3 of the **Mechanical Component Adding Wizard** are displayed.

## Mechanical Component Common Settings

Configure the visibility, location, and mount point/link point settings for the 3D Visualizer.
The origin of the 3D shape data for a mechanical component is set to the origin defined in the imported CAD data file.




| | Item | | Description | Set value | Initial value |
|---|---|---|---|---|---|
| (a) | Update CAD File | | Update the CAD data file for a mechanical component object. | --- | --- |
| (b) | 3D Visualization | Visible | Set whether to make the 3D shape data for a mechanical component visible on the 3D Visualizer. Select this check box to make it visible on the 3D Visualizer. | Checked or unchecked | Checked |
| | | Transparency | Set the transparency of the 3D shape data. | None, Low, Medium, or High | None |
| (c) | Configuration | TCP visible | Set whether to make the TCP visible in the 3D editing area. Select this check box to make it visible. | Checked or unchecked | Checked |
| (d) | Location | Rotation Offset | Set the rotation offset from the origin of the local coordinate system for this 3D shape data. The position of the rotation axis will change based on this offset. The location elements are X, Y, Z, y (yaw), p (pitch), and r (roll), from left to right. | X, Y, Z: For each, -1,000,000.000 to 1,000,000.000<br>y, r: For each, -180.000 to 180.000<br>p: 0.000 to 180.000 | 0.000 for all |
| | | TCP | TCP is a set of coordinates of the point indicating the position where tools such as a robot hand are mounted on the 3D shape data of a mechanical component. Set the coordinates in the local coordinate system of the 3D shape data for the mechanical component. The location elements are X, Y, Z, y (yaw), p (pitch), and r (roll), from left to right. | X, Y, Z: For each, -1,000,000.000 to 1,000,000.000<br>y, r: For each, -180.000 to 180.000<br>p: 0.000 to 180.000 | 0.000 for all |

| | Item | | Description | Set value | Initial value |
|---|---|---|---|---|---|
| | | Parent | Select the 3D shape data to be the parent of the 3D shape data for a mechanical component. | Name of 3D shape data | No selection |
| | | Offset From Parent | Set the position and pose of the 3D shape data for a mechanical component. When the parent is not selected, set the coordinate in the world coordinate system of the 3D Visualizer. When the parent is selected, set the coordinate in the local coordinate system of the parent 3D shape data. The location elements are X, Y, Z, y (yaw), p (pitch), and r (roll), from left to right. | X, Y, Z: For each, -1,000,000.000 to 1,000,000.000<br>y, r: For each, -180.000 to 180.000<br>p: 0.000 to 180.000 | 0.000 for all |
| (e) | Advanced Settings – Collision Hull | Visible | Set whether to show collision hulls. | Checked or unchecked | Unchecked |
| | | Configuration | Configure the collision hull settings. Click the button at the right, and then configure the advanced collision hull settings. Refer to *Advanced Collision Hull Settings* on page 8-27 for details on how to configure the settings. | --- | Convex: True \| Multiple Hulls: False |
| (f) | Mount point or link point setup area | | Use this area to set mount points and link points for the 3D shape data for a mechanical component. Refer to *5-4 Positioning with a Mount Point or a Link Point* on page 5-25 for information on how to set the mount points and link points. | --- | --- |
| (g) | 3D editing area | | Use this area to display the position, pose, and mount point or link points of the 3D shape data for a mechanical component. Refer to *Section 5 Working with the 3D Visualizer and 3D Editing Area* on page 5-1 for details on how to make items visible and work with them in the 3D editing area. | --- | --- |

# 4-4 Creating 3D Shape Data for the Custom Mechanics

Custom mechanics are components that allow you to configure motions of desired movable parts. By combining movable parts for linear motion, rotation, and linear-rotational compound motion, you can realize components that are not supported by the standard mechanical components provided by Sysmac Studio.

## 4-4-1 Components That Custom Mechanics Realize

Custom mechanics allow you to create the following components by combining movable parts for linear motion, rotation, and linear-rotational compound motion.

| Components | Description | Motion to combine |
|---|---|---|
| Electric cylinder | Electric cylinder refers to a component whose piston is electrically driven according to a single input value. | Linear motion |
| Electric chuck (2-finger type, 3-finger type) | Electric chuck refers to a component that electrically drives a chuck attached to the tip of a robot to pick parts. | Linear motion |
| Air chuck (2-finger type, 3-finger type) | Air chuck refers to a component that drives by injecting air a chuck attached to the tip of a robot to pick parts. | Linear motion |
| Linear-rotational compound motion actuator | This refers to a component that makes a linear motion and rotation according to a single input value. Once the component receives an input, it makes a linear motion first, and then makes a rotation. | Linear motion and rotation |

**Precautions for Correct Use**

You cannot create the following components.
- Gantry
- H-Bot
- T-Bot
- Mechanical cam and electronic cam
- Gear
- Components that include screw joints

## 4-4-2 Procedure of Creating 3D Shape Data for the Custom Mechanics

Create 3D shape data for a custom mechanics by the following procedure.

<table>
<tr><th></th><th>Item</th><th>Description</th><th colspan="2">Reference</th></tr>
<tr><td>1</td><td>Adding a Custom Mechanics</td><td>Add a custom mechanics to the Application Manager.</td><td colspan="2">4-4-4 Custom Mechanics Addition Procedure on page 4-36</td></tr>
<tr><td rowspan="2">2</td><td rowspan="2">Adding Movable Parts by Importing CAD Data</td><td rowspan="2">Import CAD data for the movable parts of the custom mechanics and create the 3D shape data. You can register more than one movable part.<br>The operating procedure differs depending on the type of the CAD data to import.</td><td>OpenUSD files that include USDPhysicsJoint</td><td>4-4-6 Importing OpenUSD Files That Include USDPhysicsJoint on page 4-43</td></tr>
<tr><td>Other than the above</td><td>4-4-5 Adding Movable Parts and Importing CAD Data on page 4-39</td></tr>
<tr><td>3</td><td>Configuring the TCP</td><td>Select the movable part that becomes the tip of the custom mechanics and configure the TCP settings.</td><td colspan="2">4-4-7 Configuring the TCP on page 4-48</td></tr>
<tr><td>4</td><td>Configuring the Joint Motion</td><td>Configure the type of the joint between the movable parts. Select from the six types: Fixed Joint, Hinge Joint, Ball Joint, Sliding Joint, Rotational joint, and Set as parent and child.</td><td colspan="2">4-4-8 Configuring the Joint Motion on page 4-50</td></tr>
</table>

| | Item | Description | Reference |
|---|---|---|---|
| 5 | Configuring the Motion Direction Settings of the Movable Parts | Configure the motion directions of movable parts. Select from three types: *Linear*, *Rotation*, and *Linear and Rotation*. | *4-4-9 Configuring the Motion Direction Settings of the Movable Parts* on page 4-58 |
| 6 | Checking the Motion of the Custom Mechanics | Perform the test run of the movable parts of the custom mechanics to check that the settings of the movable parts and joint are correct. | *4-4-10 Checking the Motion of the Custom Mechanics* on page 4-63 |
| 7 | Configuring the Motion Settings of the Custom Mechanics | Define the motion of the custom mechanics in 3D simulation. | *4-4-11 Configuring the Motion Settings of the Custom Mechanics* on page 4-66 |

## 4-4-3 Supported OpenUSD Files

This section describes OpenUSD files supported by the Sysmac Studio. Refer to *4-3-1 Types of Supported CAD Data Files* on page 4-10 for CAD data.

### ● Supported OpenUSD File Formats

The following are the formats of OpenUSD files that can be loaded into the Sysmac Studio.

| File format | Description |
|---|---|
| USD | ASCII-based text format or binary format |
| USDA | ASCII-based text format |
| USDC | Binary format |
| USDZ | Archive format |

### ● Supported OpenUSD File Versions

The following are the supported versions of OpenUSD files that can be loaded into the Sysmac Studio.

| | Supported version |
|---|---|
| OpenUSD | OpenUSD that supports OpenUSD 23.11 SDK |

### ● Supported USDPhysicsJoint Types

The following are the types of USDPhysicsJoint that can be loaded in the Sysmac Studio.

| Joint information | Type |
|---|---|
| USDPhysicsJoint | Prismatic |
| | Revolute |
| | Fixed |

**Additional Information**

- The Sysmac Studio supports USDPhysicsJoint in OpenUSD, where body0 and body1 have a relationship of parent and child in the prim path.
  Definition example:

```
def PhysicsPrismaticJoint "SpecialJoint" {
  rel physics:body0 = </World/Base>
  rel physics:body1 = </World/Base/X>
```

- In cases where body1 of certain USDPhysicsJoint is the parent of body0, it is judged as invalid information and will not be imported.
  Definition example:

```
def PhysicsPrismaticJoint "InvalidJoint" {
  rel physics:body0 = </World/Base/X>
  rel physics:body1 = </World/Base>
```

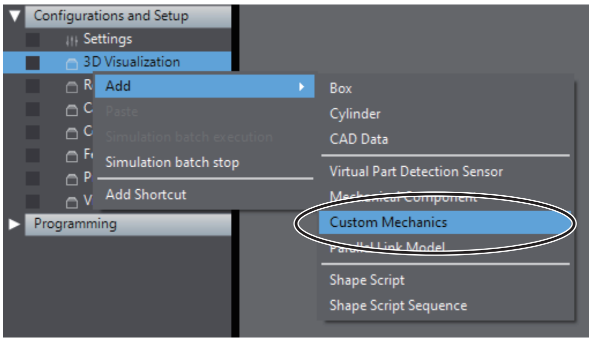
- In cases where a prim specified by body1 of certain USDPhysicsJoint is also specified as body1 by different USDPhysicsJoint, it will be judged as an invalid OpenUSD file and its import processing will be interrupted.

## 4-4-4 Custom Mechanics Addition Procedure

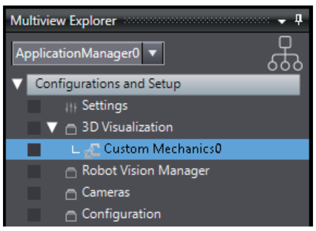
Add a custom mechanics to the Application Manager. Use the following procedure to add a custom mechanics.

### Operating Procedure

**1** Right-click **3D Visualization** under **Configurations and Setup** and select **Add** – **Custom Mechanics** from the menu.

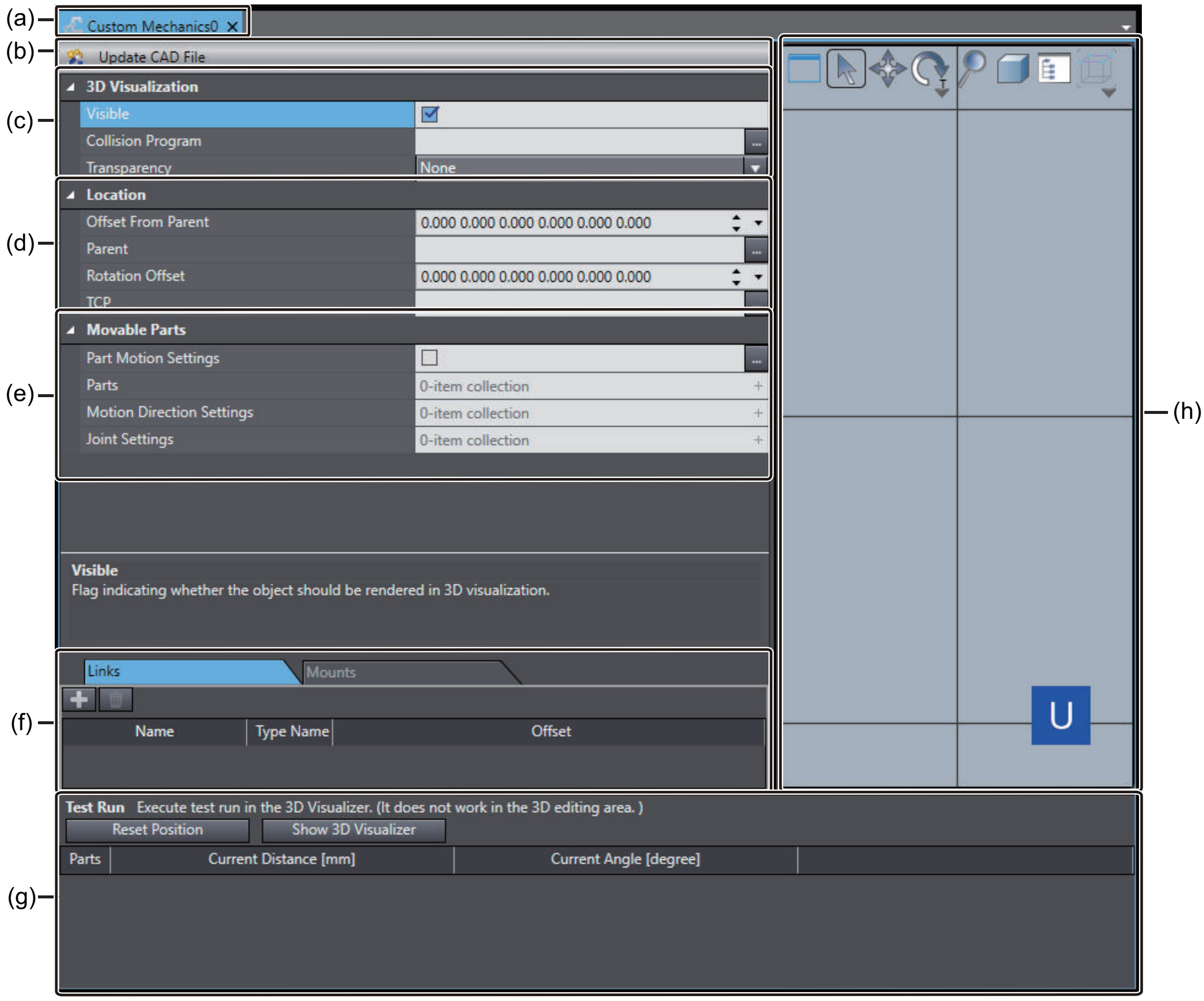



The selected component is registered and displayed under **3D Visualization**.



## Custom Mechanics Settings

Double-clicking the custom mechanics that you added displays the setup tab page for the custom mechanics. This section describes the settings common to all types of custom mechanics and the settings specific to each type.




| | Item | | Description | Set value | Initial value |
|---|---|---|---|---|---|
| (a) | Custom mechanics name | | Displays the custom mechanics name displayed in the Multiview Explorer. | --- | --- |
| (b) | Update CAD File | | Import the CAD Data to use as 3D shape data. Refer to *4-4-5 Adding Movable Parts and Importing CAD Data* on page 4-39 for details. | --- | --- |
| (c) | 3D Visualization | Visible | Set whether to make the 3D shape data for a custom mechanics visible on the 3D Visualizer. Select this check box to make it visible on the 3D Visualizer. | Checked or unchecked | Checked |

| | Item | | Description | Set value | Initial value |
|---|---|---|---|---|---|
| | | Collision Program | Specify the C# program to execute when a collision of the 3D shape data is detected. Click the button at the right, and then select the C# program from the list. You can write a C# program that outputs the time of collision, the target of collision, etc. to get information on the collision that occurred. | C# program name | None |
| | | Transparency | Set the transparency of the 3D shape data. | None, Low, Medium, or High | None |
| (d) | Location | Offset From Parent | Set the position and pose of the 3D shape data for a custom mechanics. When the parent is not selected, set the coordinates in the world coordinate system of the 3D Visualizer. When the parent is selected, set the coordinates in the local coordinate system of the parent 3D shape data. The location elements are X, Y, Z, y (yaw), p (pitch), and r (roll), from left to right. | X, Y, Z: For each, -1,000,000.000 to 1,000,000.000<br>y, r: For each, -180.000 to 180.000<br>p: 0.000 to 180.000 | 0.000 for all |
| | | Parent | Select the 3D shape data to be the parent of the 3D shape data for a custom mechanics. | Name of 3D shape data | No selection |
| | | Rotation Offset | Set the rotation offset from the origin of the local coordinate system for the 3D shape data. The position of the rotation axis will change based on this offset. The location elements are X, Y, Z, y (yaw), p (pitch), and r (roll), from left to right. | X, Y, Z: For each, -1,000,000.000 to 1,000,000.000<br>y, r: For each, -180.000 to 180.000<br>p: 0.000 to 180.000 | 0.000 for all |
| | | TCP | Configure the movable part for the TCP of the custom mechanics and the TCP position. Set the position in the local coordinate system of the selected movable part. Refer to *4-4-7 Configuring the TCP* on page 4-48 for details on how to configure the settings. | X, Y, Z: For each, -1,000,000.000 to 1,000,000.000<br>y, r: For each, -180.000 to 180.000<br>p: 0.000 to 180.000 | --- |
| (e) | Movable Parts | Part Motion Settings | Configure the motion settings of movable parts. Refer to *4-4-11 Configuring the Motion Settings of the Custom Mechanics* on page 4-66 for details on how to configure the settings. | --- | --- |
| | | Parts | Configure the settings of movable parts. Refer to *4-4-5 Adding Movable Parts and Importing CAD Data* on page 4-39 for details on how to configure the settings. | --- | --- |

| | Item | | Description | Set value | Initial value |
|---|---|---|---|---|---|
| | | Motion Direction Settings | Configure the motion direction settings of movable parts. Refer to *4-4-9 Configuring the Motion Direction Settings of the Movable Parts* on page 4-58 for details on how to configure the settings. | --- | --- |
| | | Joint Settings | Configure the type of the joint between the movable parts. Refer to *4-4-8 Configuring the Joint Motion* on page 4-50 for details. | --- | --- |
| (f) | Mount point or link point setup area | | Use this area to set mount points and link points for the 3D shape data for a custom mechanics. Refer to *5-4 Positioning with a Mount Point or a Link Point* on page 5-25 for information on how to set the mount points and link points. | --- | --- |
| (g) | Movable parts test run | | Operate the movable parts of the custom mechanics to check the motion settings of the movable parts and joint. Refer to *4-4-10 Checking the Motion of the Custom Mechanics* on page 4-63 for details. | --- | --- |
| (h) | 3D editing area | | Use this area to display the position, pose, and mount point or link points of the 3D shape data for a custom mechanics. Refer to *Section 5 Working with the 3D Visualizer and 3D Editing Area* on page 5-1 for details on how to make items visible and work with them in the 3D editing area. | --- | --- |

## 4-4-5 Adding Movable Parts and Importing CAD Data

Add movable parts to the custom mechanics and import the CAD Data to create the 3D shape data. You can register more than one movable part.

This section describes an example of loading OpenUSD format CAD data that does not include USD-PhysicsJoint definitions.

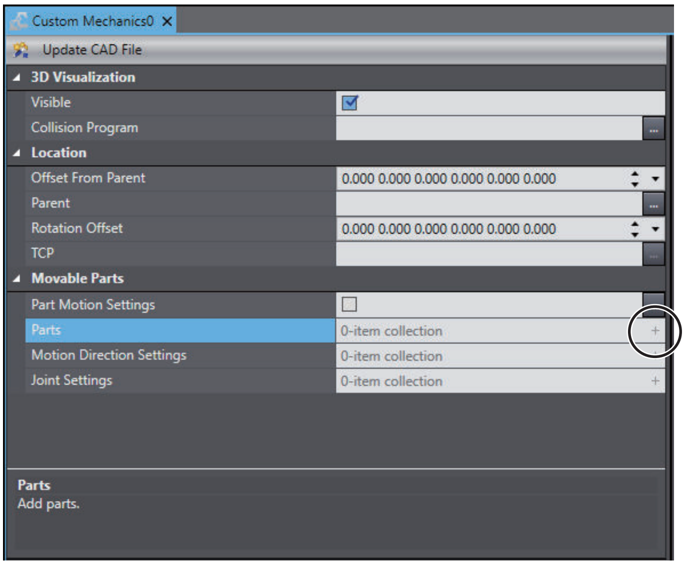
### Operating Procedure

Use the following procedure to add movable parts.

**Additional Information**

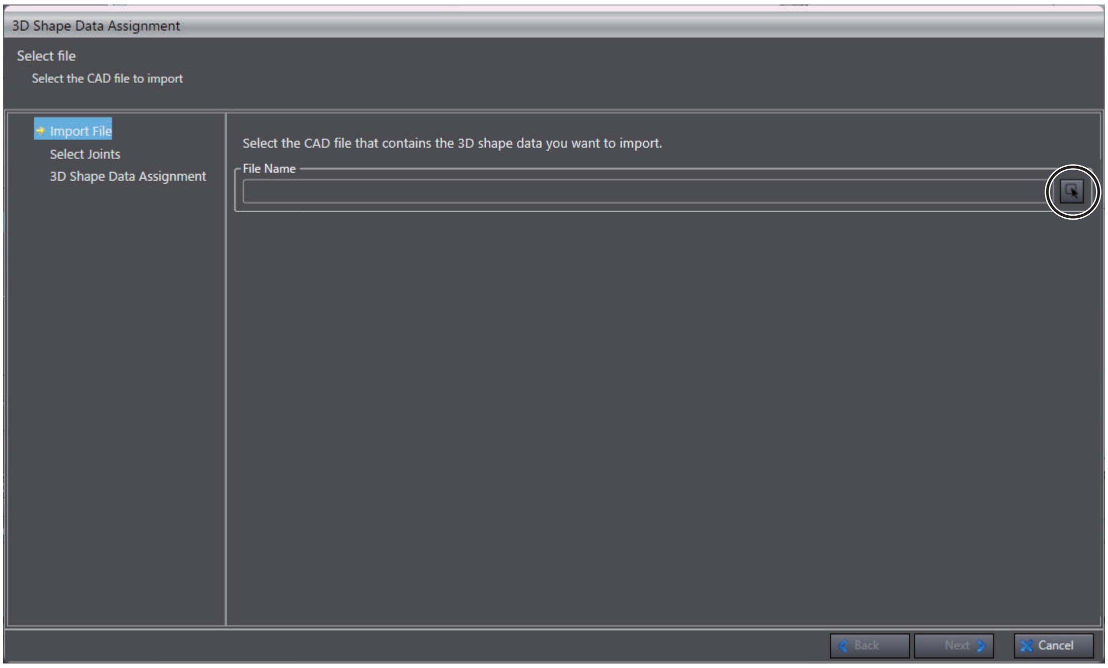
You can optionally set the mesh coarseness of CAD data at the time of import. Refer to the *Sysmac Studio Robot Integrated System Building Function with Robot Integrated CPU Unit Operation Manual (Cat. No. W595)* for details on the option settings.

1. In the setup tab page for the target custom mechanics, click the Add button at the right of **Parts**.



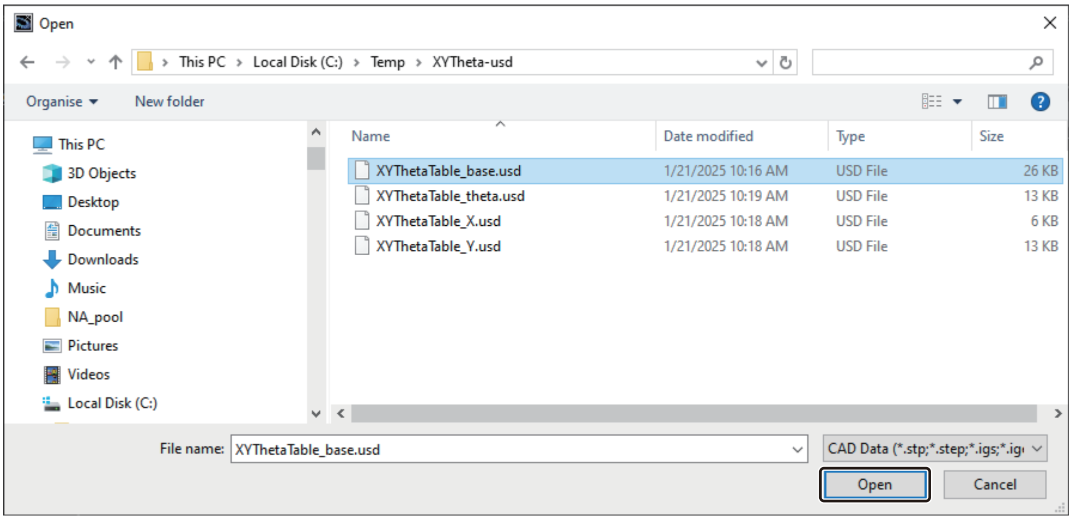

The wizard starts and the **3D Shape Data Assignment** page is displayed.

**2** In the **Select file** page, click the button at the right of **File Name**.

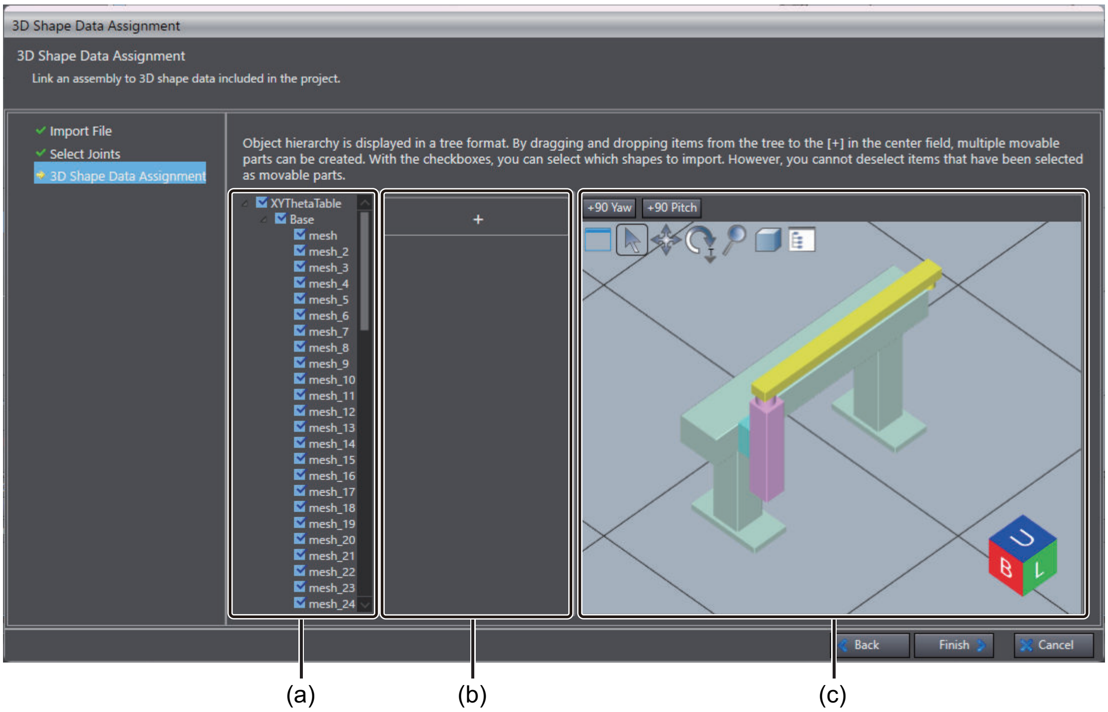



The **Open** dialog box is displayed.

**3** In the **Open** dialog box, select the STEP, IGES, or OpenUSD CAD data file to import, and then click the **Open** button.

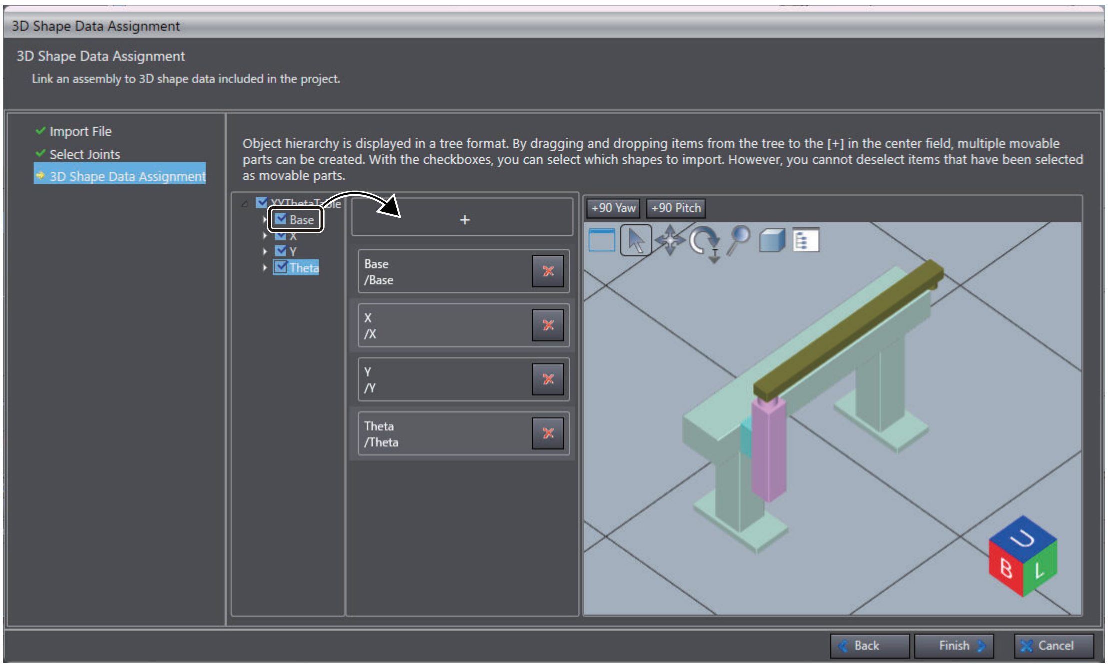



In the **Select file** dialog box, click the **Next** button. The **3D Shape Data Assignment** page is displayed.



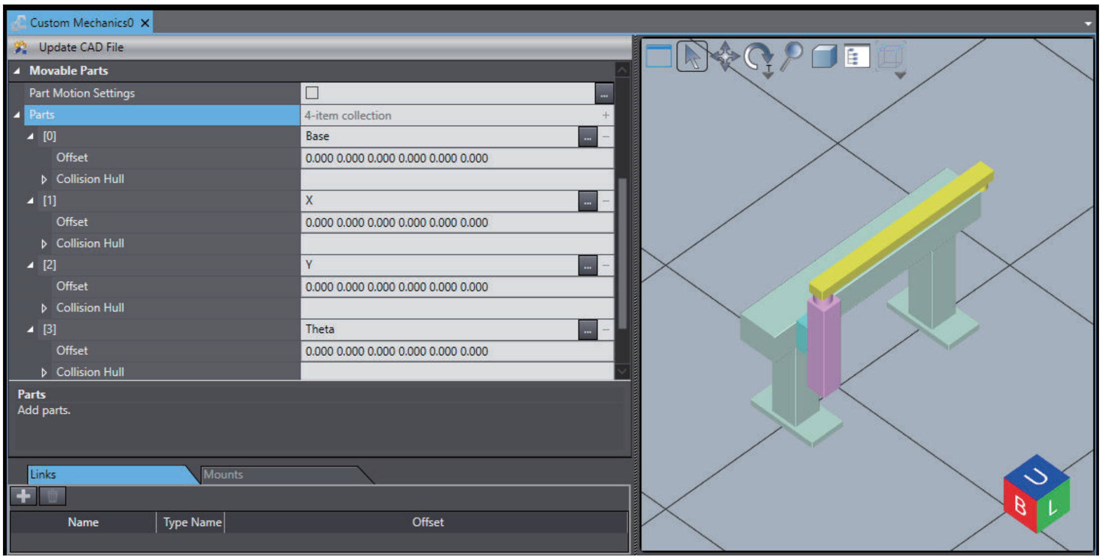

| | Item | Description |
|---|---|---|
| (a) | Imported OpenUSD prims | Imported OpenUSD prims are displayed in a hierarchy.<br>You can exclude 3D shape data that you do not want to import by clearing the check boxes. |
| (b) | List of movable part | Drag and drop each element from (a) to the **+** area to create movable parts.<br>It is not possible to create multiple movable parts from the same element.<br>You can delete the created movable parts by clicking the × button. |
| (c) | Imported CAD data in the 3D Visualizer | A 3D view of the imported CAD data is displayed. |

**4** Drag and drop each element to set as a movable part to the **+** area.




Repeat dragging and dropping as many elements as you want to set as movable parts.

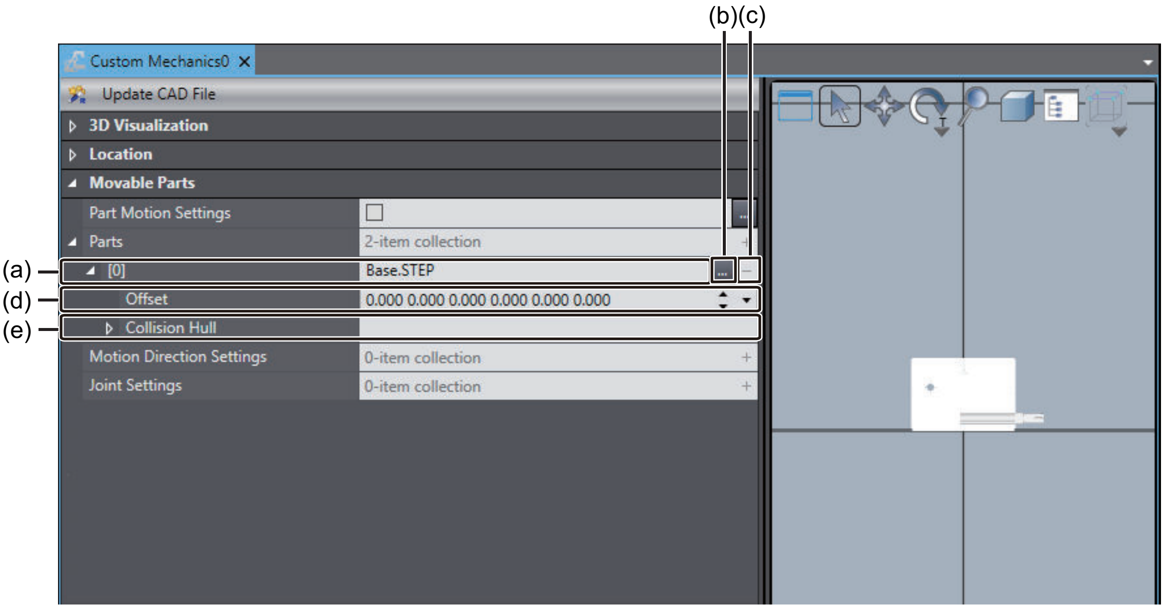
**5** After completion of assigning movable parts, click the **Finish** button.




The item names of the CAD data set as movable parts are added. Also, the data of the CAD file that you imported is displayed in the 3D editing area.

## Movable Parts Settings

The setting items of movable parts are as follows.




| | Item | Description | Set value | Initial value |
|---|---|---|---|---|
| (a) | CAD file name | Displays the name of the imported CAD file. | --- | The name of the imported CAD file |
| (b) | Import a CAD File button | Allows you to replace the CAD file of the movable part. Clicking this button displays the **Select file** dialog box. | --- | --- |
| (c) | Delete a Movable Part button | Clicking this button deletes the added movable part. | --- | --- |
| (d) | Offset | Specify the position of the movable part. Set the relative coordinates from the origin of the custom mechanics. The location elements are X, Y, Z, y (yaw), p (pitch), and r (roll), from left to right. | X, Y, Z: For each, -1,000,000.000 to 1,000,000.000<br>y, r: For each, -180.000 to 180.000<br>p: 0.000 to 180.000 | 0.000 for all |
| (e) | Collision Hull | Configure the collision hull related settings. Refer to *8-3-5 Collision Hull Settings* on page 8-26 for details. | --- | --- |

## 4-4-6 Importing OpenUSD Files That Include USDPhysicsJoint

Import OpenUSD files that include USDPhysicsJoint. Due to USDPhysicsJoint, this helps to reduce the amount of work required for assigning movable parts and setting motion directions when you create 3D shape data for custom mechanics, thereby reducing the time required to build a simulation environment.

### Importing Procedure

Use the following import procedure.

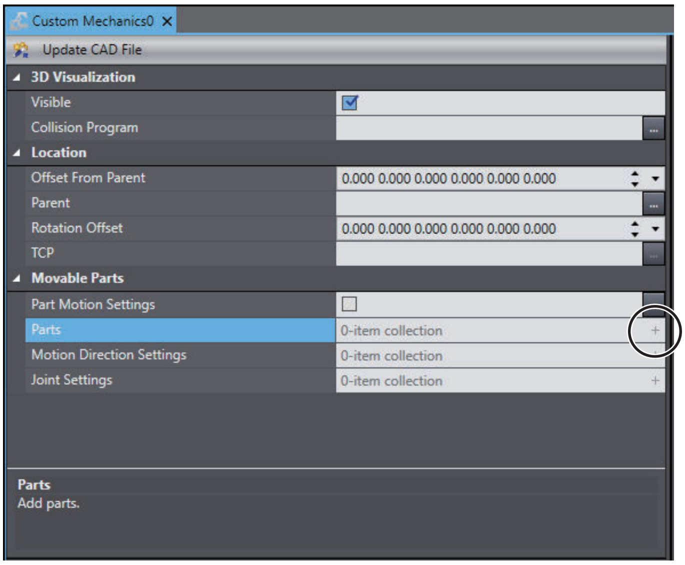
**Additional Information**

You can optionally set the mesh coarseness of CAD data at the time of import. Refer to the *Sysmac Studio Robot Integrated System Building Function with Robot Integrated CPU Unit Operation Manual (Cat. No. W595)* for details on the option settings.

**1** Add custom mechanics.

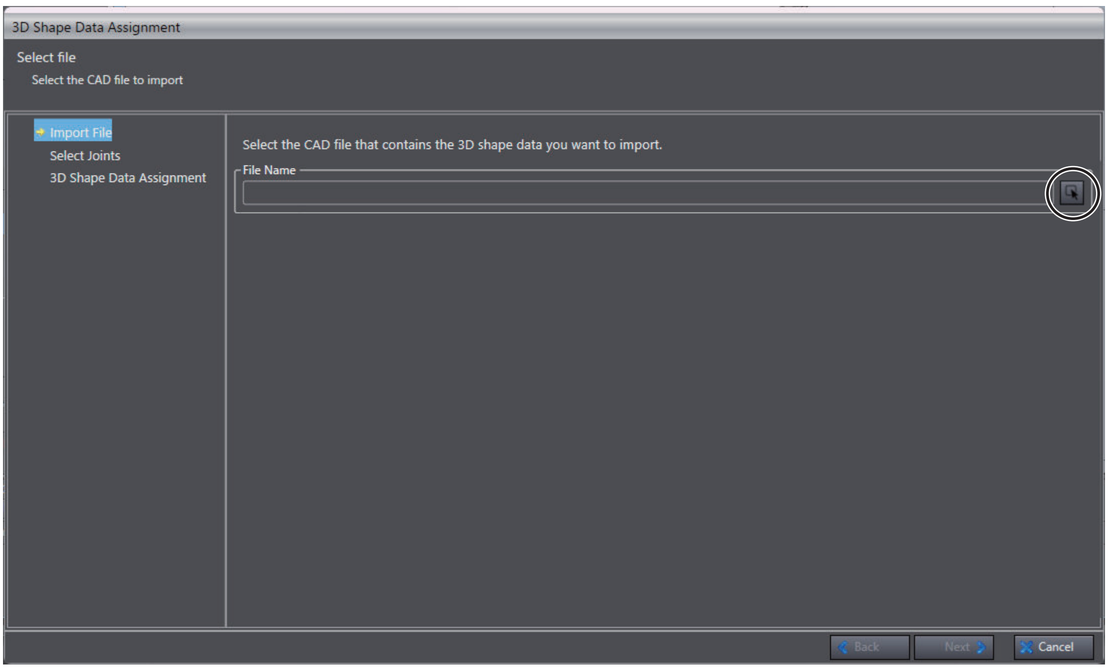
Refer to *4-4-4 Custom Mechanics Addition Procedure* on page 4-36 for the procedure for adding custom mechanics.

**2** In the setup tab page for the target custom mechanics, click the Add button at the right of **Parts**.



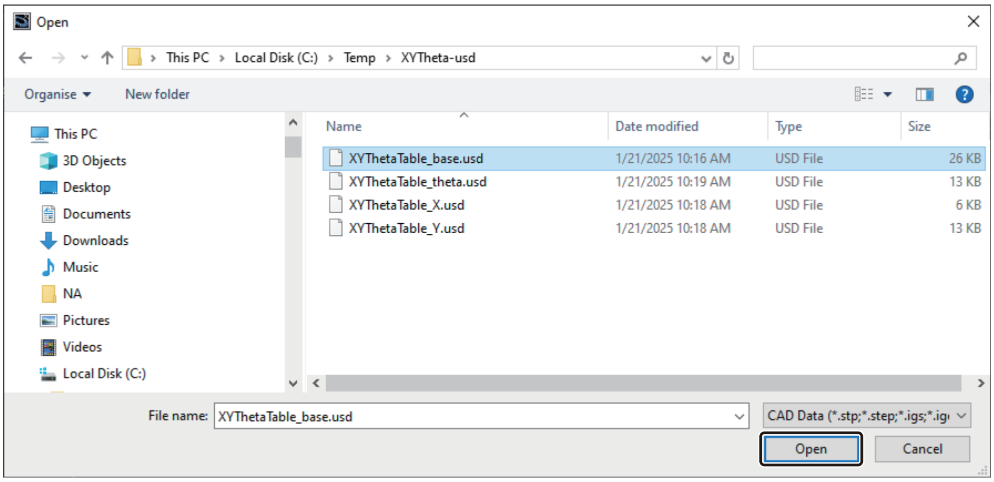

The wizard starts and the **3D Shape Data Assignment** page is displayed.

**3** In the **Select file** page, click the button at the right of **File Name**.

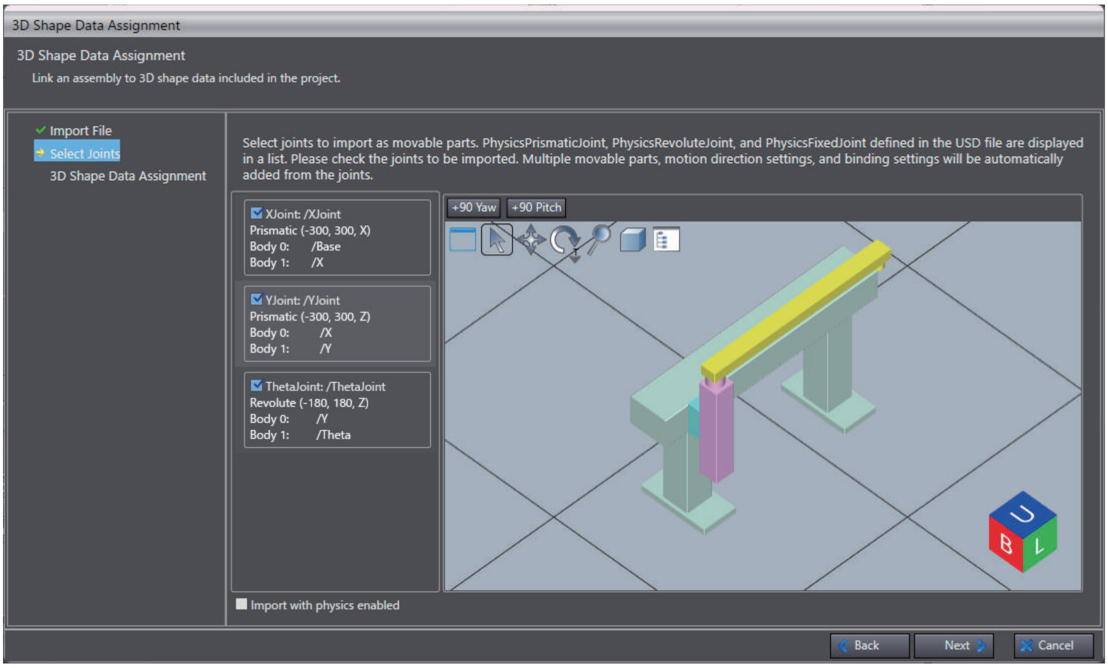


The **Open** dialog box is displayed.

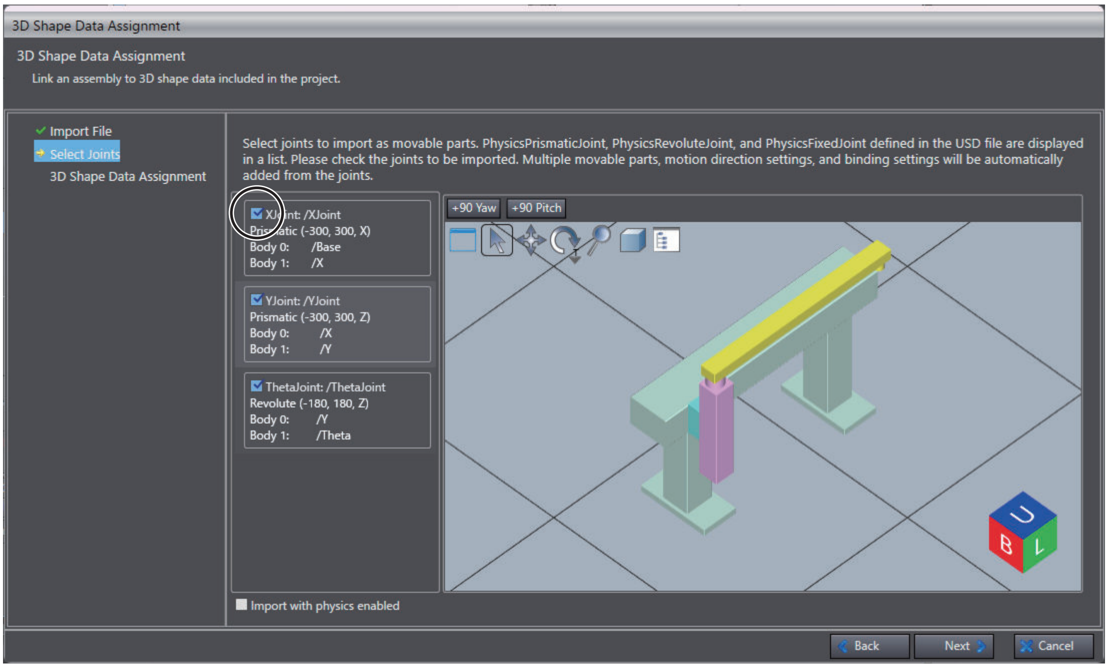
**4** In the **Open** dialog box, select the USD, USDA, USDC or USDZ format OpenUSD file to import, and then click the **Open** button.




In the **Select file** dialog box, click the **Next** button. The **Select Joints** page is displayed.



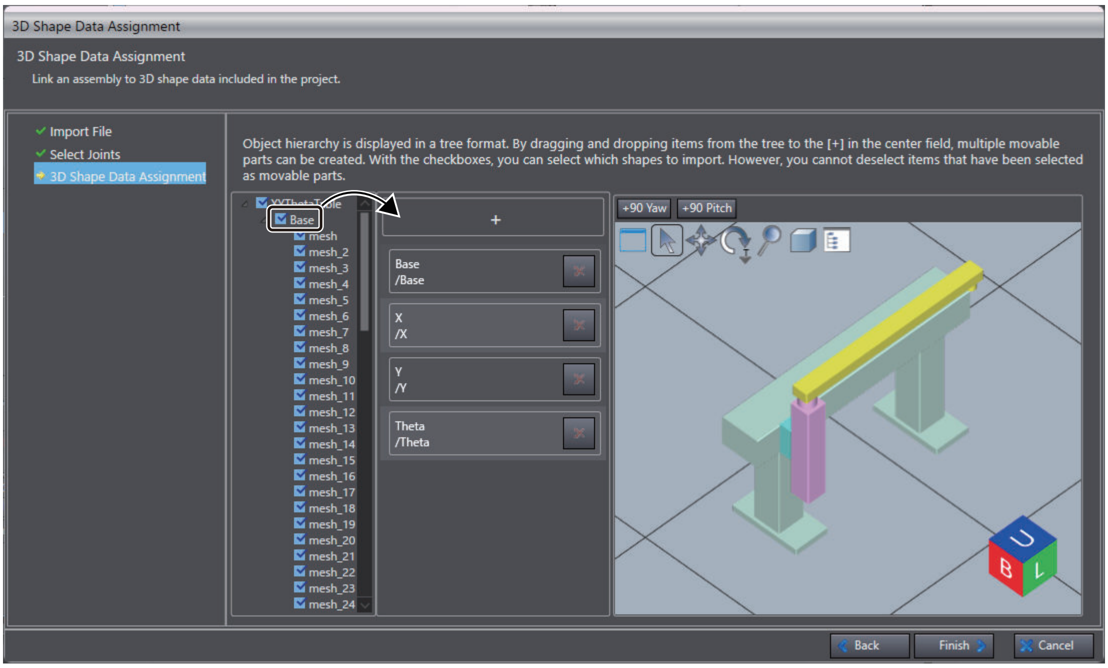

**5** In the **Select Joints** page, select the check boxes for the USDPhysicsJoint to import.




**6** If necessary, select the **Import with physics enabled** check box and then click the **Next** button.


Refer to ***Import with Physics Enabled*** on page 4-47 for details on the **Import with physics enabled** check box.

**7** The **3D Shape Data Assignment** page displays multiple movable parts that have been created based on the USDPhysicsJoint definitions.

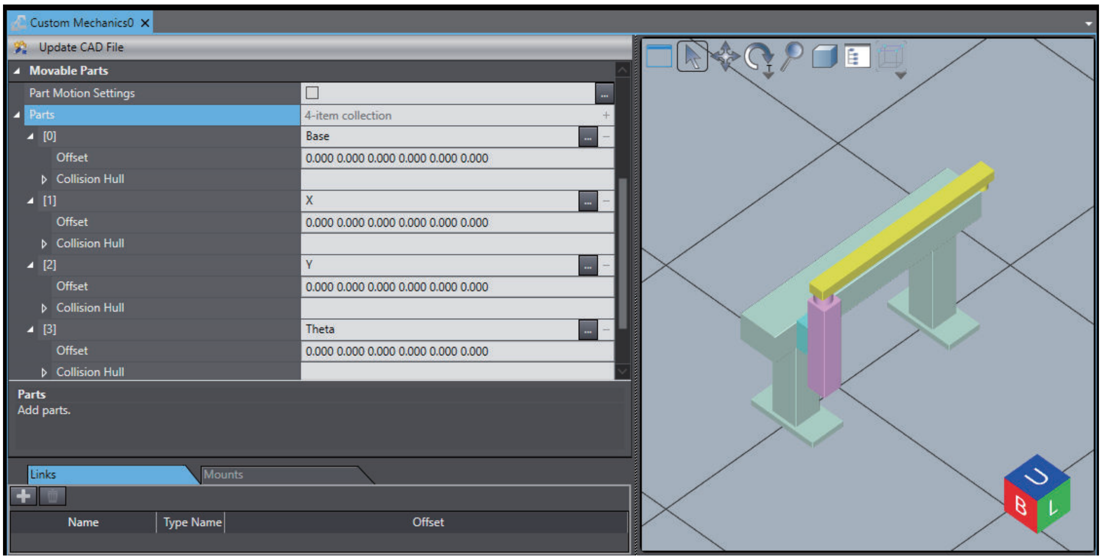


To add more movable parts, drag and drop the elements to set as movable parts to the **+** area. Repeat dragging and dropping as many elements as you want to set as movable parts, and then click the **Finish** button.

### Additional Information

- The level of elements displayed in the USD prim tree is determined by the option settings. Also, depending on the level setting, the **Select Joints** page may be skipped and you may not be able to import USDPhysicsJoint. Refer to the *Sysmac Studio Robot Integrated System Building Function with Robot Integrated CPU Unit Operation Manual (Cat. No. W595)* for the option settings.
- Refer to *4-4-5 Adding Movable Parts and Importing CAD Data* on page 4-39 for details on the items displayed on the **3D Shape Data Assignment** page.

**8** The movable parts are added to the setup tab page for the corresponding custom mechanics. Configure the motion settings for each movable part.




Refer to *4-4-11 Configuring the Motion Settings of the Custom Mechanics* on page 4-66 for details on the motion settings.

## Import with Physics Enabled

**Import with physics enabled** is a function that allows you to select whether to configure the joint settings according to the USDPhysicsJoint type or as parent and child when USDPhysicsJoint is imported.

When the **Import with physics enabled** check box is selected, the joint settings are configured according to the USDPhysicsJoint type so that each movable part is not controlled by its own axis, enabling physics simulations that reproduce the motion according to the operation of other movable parts, etc.

When the **Import with physics enabled** check box is cleared, the joint settings are configured as parent and child so that each movable part is controlled by its own axis, which is suitable for situations where physics simulations are unnecessary.

- When the **Import with physics enabled** check box is selected

When USDPhysicsJoint is imported, the following joint settings are configured in accordance with the USDPhysicsJoint type in OpenUSD, where the movable part specified by body0 is *Joint Target 1* and the movable part specified by body1 is *Joint Target 2*.

| USDPhysicsJoint in OpenUSD to be imported | Joint Settings in the Sysmac Studio |
|---|---|
| UsdPhysicsPrismaticJoint | Sliding Joint |
| UsdPhysicsRevoluteJoint | Rotational joint |
| UsdPhysicsFixedJoint | Fixed Joint |

The physics settings for the movable part set in *Joint Target 2* are enabled. Settings for the joint position are automatically configured based on the following axis direction and axis position definitions described in USDPhysicsJoint.

a) physics:axis
b) physics:localPos0
c) physics:localRot0
d) physics:localPos1
e) physics:localRot1

- When the **Import with physics enabled** check box is cleared
  When USDPhysicsJoint is imported, the joint settings are configured as parent and child, where the movable part specified by body0 is the *parent* and the movable part specified by body1 is the *child*.

  If movable parts generated by USDPhysicsJoint have a relationship of parent and child in the hierarchical structure in OpenUSD, the joint settings will be configured as parent and child. However, the relationship of parent and child will be configured only between movable parts that are closest in the hierarchy, rather than between all movable parts. In addition, if the joint settings are already configured by USDPhysicsJoint, the Import with physics enabled setting will be ignored.

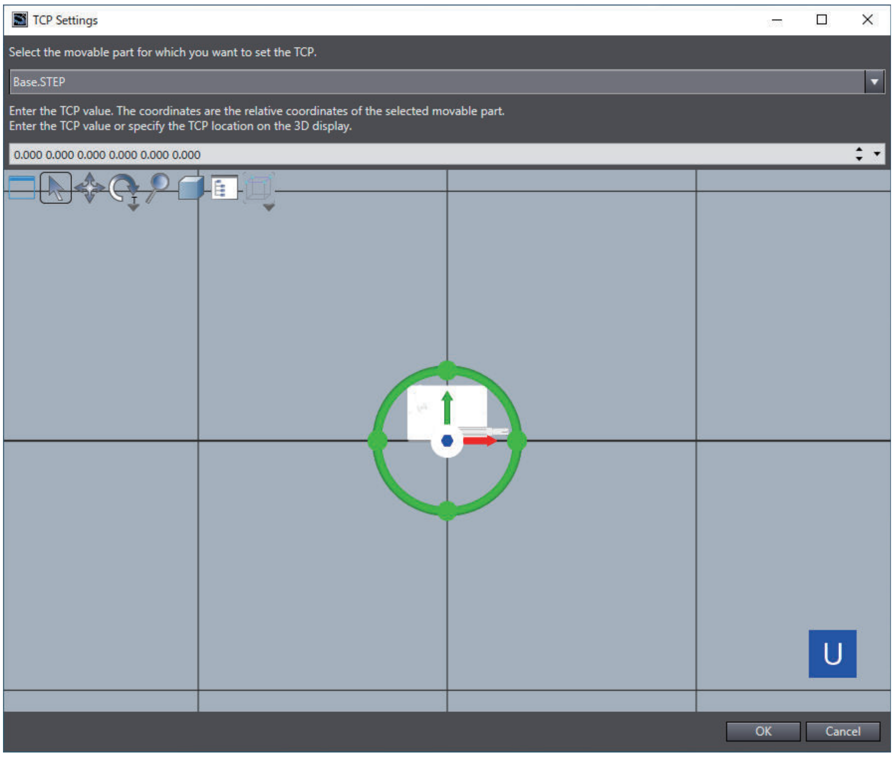
## 4-4-7 Configuring the TCP

Select the movable part that becomes the tip of the custom mechanics and configure the TCP settings.

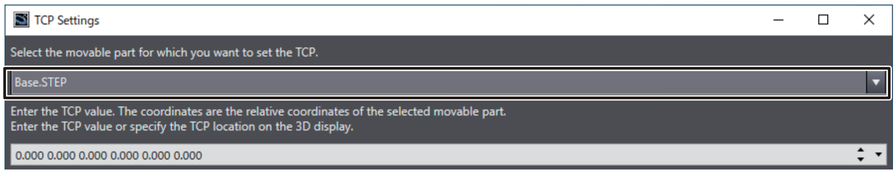
### Setting Procedure

Use the following procedure to configure the TCP settings.

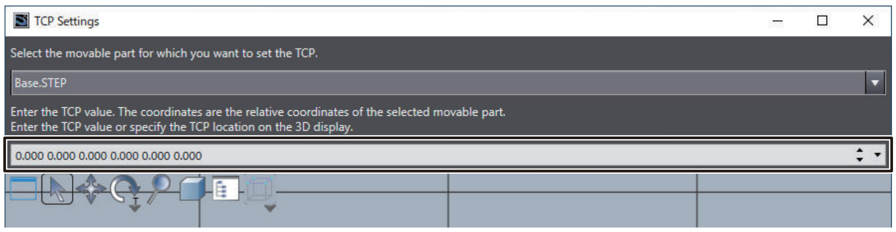
**1** In the setup tab page for the custom mechanics, click the button at the right of **TCP**.
The **TCP Settings** dialog box is displayed.




**2** Select the movable part for the tip of the custom mechanics.

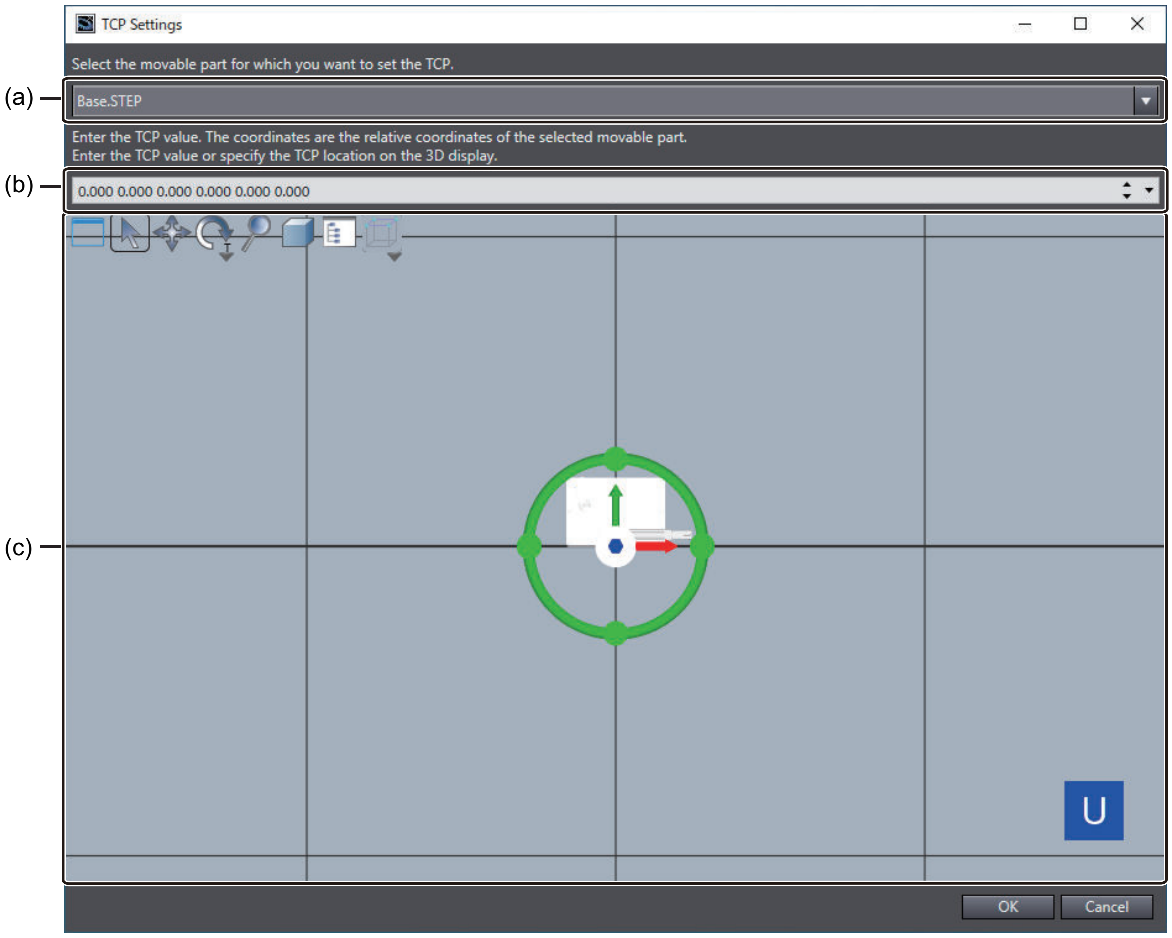



**3** Set the TCP position.




**4** Click the **OK** button.
The TCP of the custom mechanics is saved.
Refer to *TCP Settings* on page 4-50 for details on the TCP settings.

## TCP Settings




| | Item | Description | Set value | Initial value |
|---|---|---|---|---|
| (a) | Movable part selection | Select the movable part on which to set TCP. | --- | --- |
| (b) | TCP setting values | Set the TCP position. The position is represented by relative coordinates to the movable part that you selected in the movable part selection box. Changing the set values moves the TCP position in the 3D editing area according to the set values. The location elements are X, Y, Z, y (yaw), p (pitch), and r (roll), from left to right. | X, Y, Z: For each, -1,000,000.000 to 1,000,000.000<br>y, r: For each, -180.000 to 180.000<br>p: 0.000 to 180.000 | 0.000 for all |
| (c) | 3D editing area | Displays the custom mechanics and the TCP position. You can change the position and orientation of the TCP by operating the mouse. Changes made are reflected in the TCP setting values. | --- | --- |

## 4-4-8 Configuring the Joint Motion

Configure the type of the joint between the movable parts.
Setting the physics option of the movable parts enables the connection of the joint.

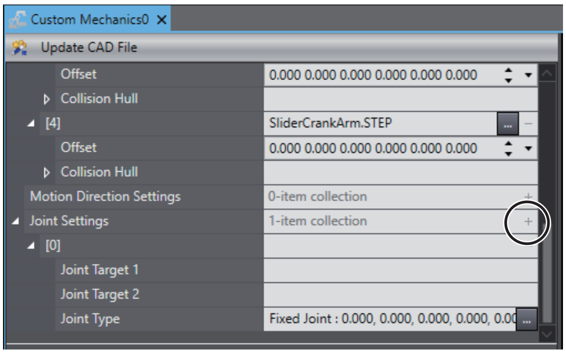
Select from the six types: *Fixed Joint*, *Hinge Joint*, *Ball Joint*, *Sliding Joint*, *Rotational joint*, and *Set as parent and child*.

| Connection method | Description |
|---|---|
| Fixed Joint | This method fixes two movable parts. When the position and orientation of one movable part are changed in the fixed state, another movable part will follow the part. |
| Hinge Joint | This method connects two movable parts with a hinge joint, which rotates at a fixed angle for the X-axis. |
| Ball Joint | This method connects two movable parts with a ball joint, which rotates with respect to Y and Z axes, and the X-axis is fixed. |
| Sliding Joint | This method connects two movable parts in the linear direction of the X-axis. |
| Rotational joint | With this method, the two movable parts rotate around the X-axis. |
| Set as parent and child | This method connects two movable parts as parent and child. The movable part set as child follows the movable part set as parent. |

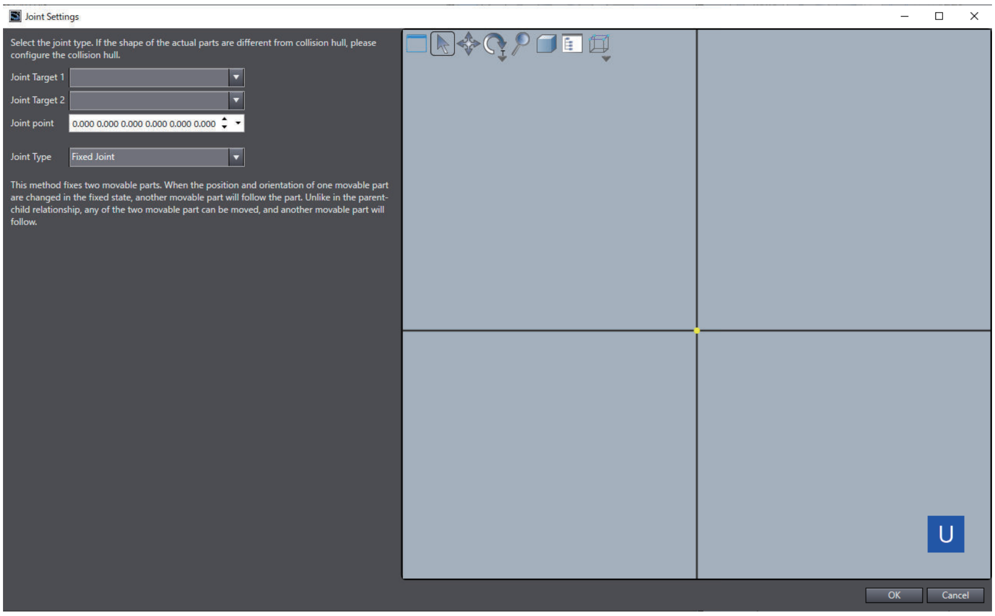
## Setting Procedure

Use the following procedure to configure the joint settings.

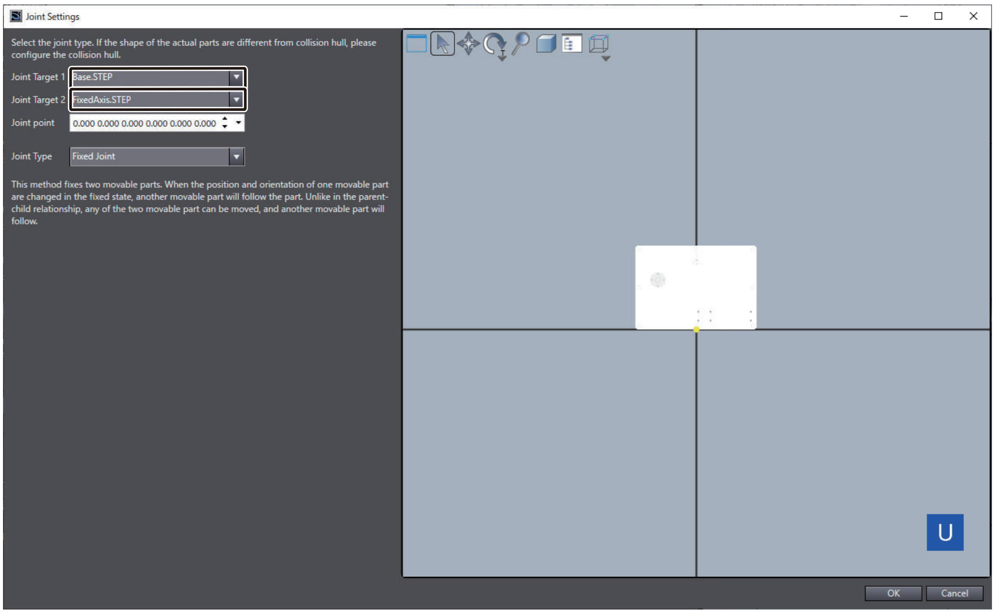
**1** In the setup tab page for the custom mechanics, click the button at the right of **Joint Settings**.




**2** The **Joint Settings** dialog box is displayed.




**3** In **Joint Target 1** and **Joint Target 2**, select two movable parts to connect.



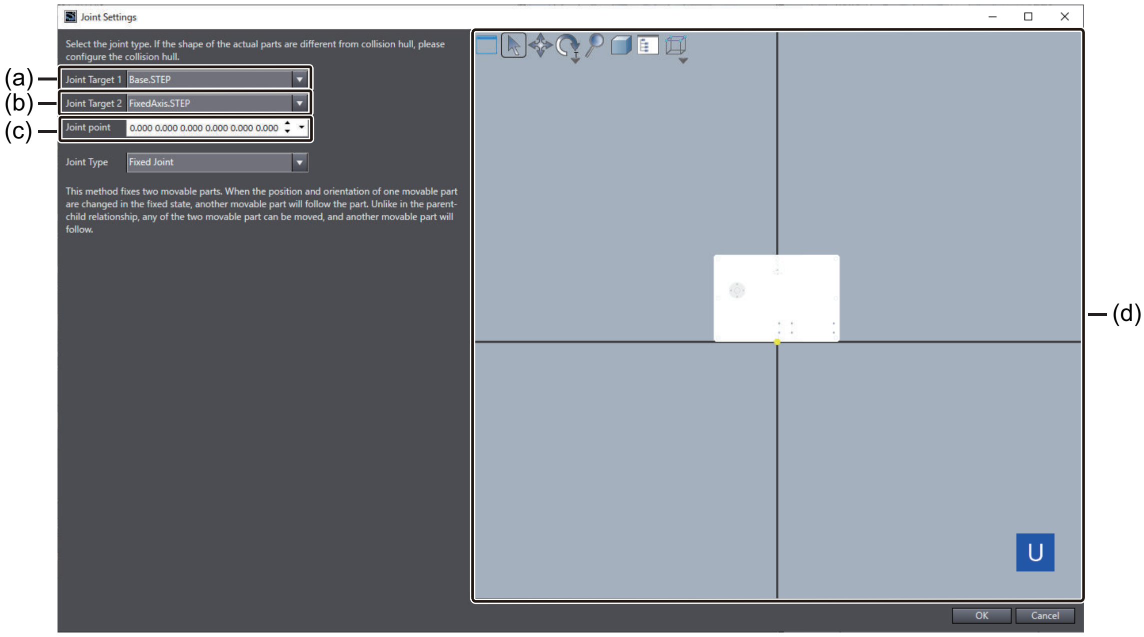

**4** In **Joint Type**, select a type from *Fixed Joint*, *Hinge Joint*, *Ball Joint*, *Sliding Joint*, *Rotational joint*, and *Set as parent and child*.
The setting items of the selected joint type is displayed under **Joint Type**. The contents displayed depend on the selected joint type.

**5** Configure the detailed settings of the joint type, and click the **OK** button.
The joint type settings are saved.
Refer to *Joint Settings* on page 4-53 for details on the setting items of each joint type.

## Joint Settings

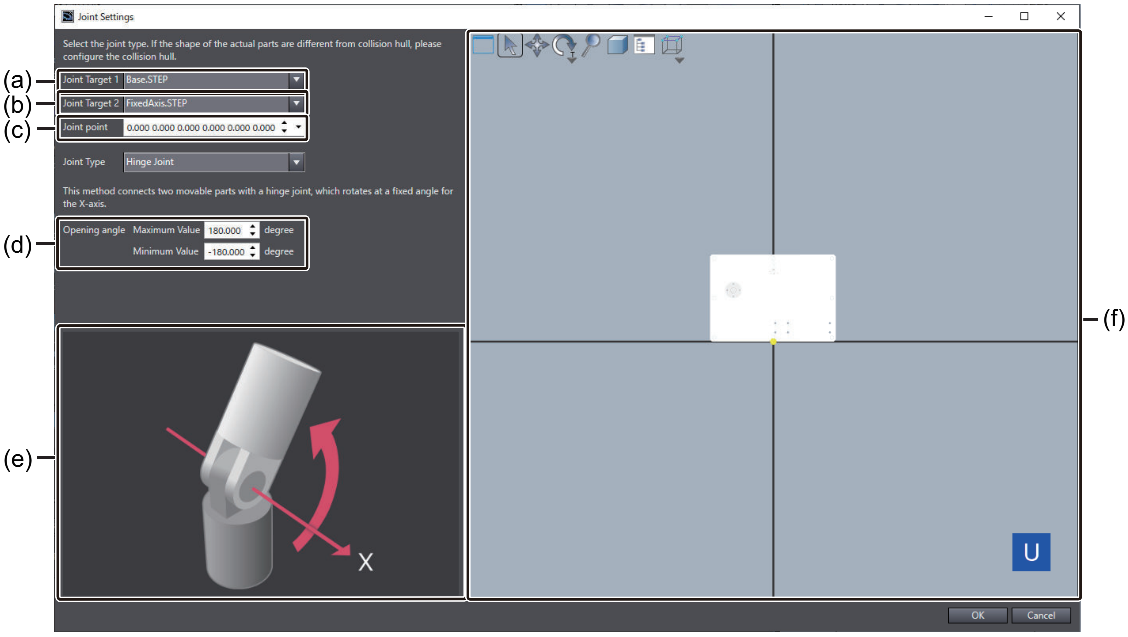
The setting items of each joint are as follows.

### ● *Fixed Joint*



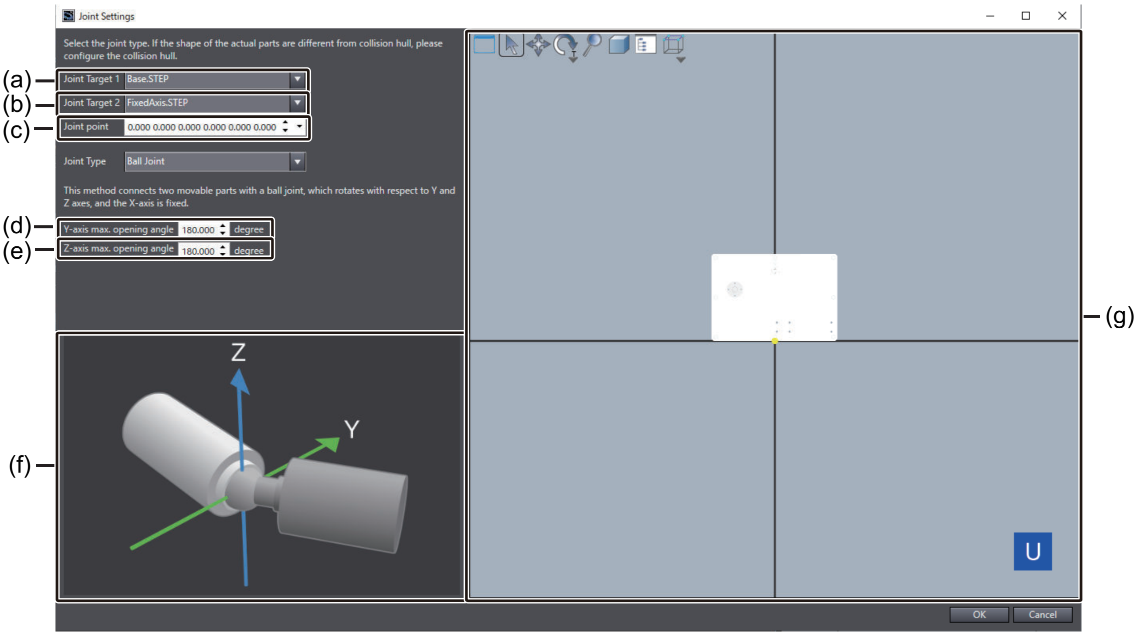

| | Item | Description | Set value | Initial value |
|---|---|---|---|---|
| (a) | Joint Target 1 | Select one of the two movable parts that are connected through the joint. | A movable part added to the custom mechanics | --- |
| (b) | Joint Target 2 | Select one of the two movable parts that are connected through the joint. | A movable part added to the custom mechanics | --- |
| (c) | Joint point | Set the coordinates of the joint point between the movable parts. The location elements are X, Y, Z, y (yaw), p (pitch), and r (roll), from left to right. | X, Y, Z: For each, -1,000,000.000 to 1,000,000.000<br>y, r: For each, -180.000 to 180.000<br>p: 0.000 to 180.000 | 0.000 for all |
| (d) | 3D editing area | Displays the movable parts to connect and the joint point.<br>You can move the joint point and snap it to the movable parts by operating the mouse. | --- | --- |

● ***Hinge Joint***



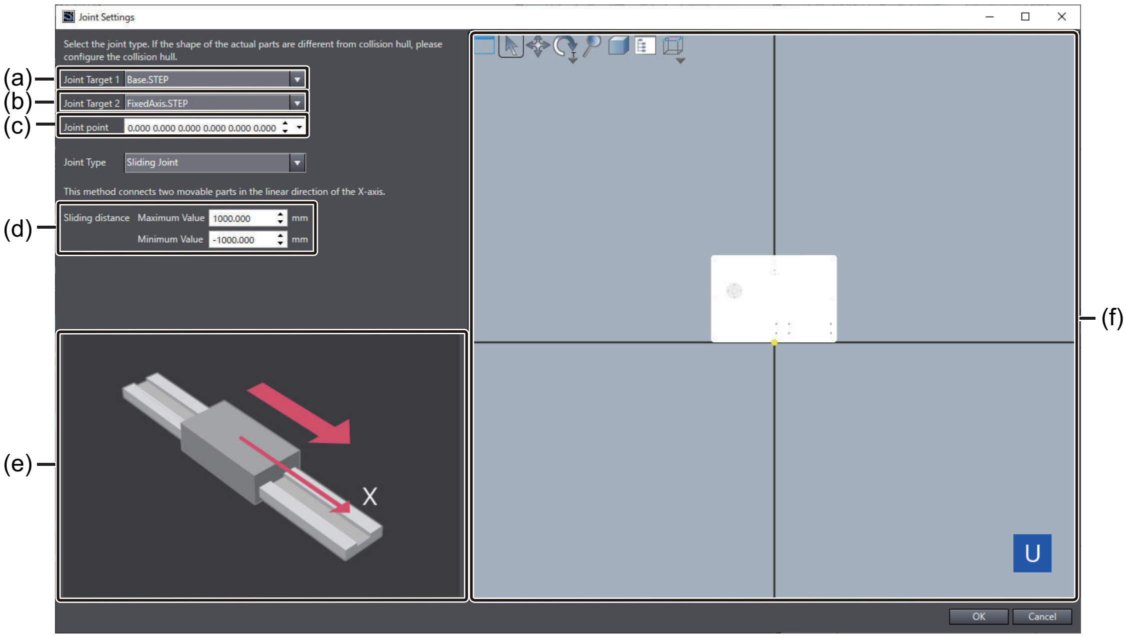

| | Item | | Description | Set value | Initial value |
|---|---|---|---|---|---|
| (a) | Joint Target 1 | | Select one of the two movable parts that are connected through the joint. | A movable part added to the custom mechanics | --- |
| (b) | Joint Target 2 | | Select one of the two movable parts that are connected through the joint. | A movable part added to the custom mechanics | --- |
| (c) | Joint point | | Set the coordinates of the joint point between the movable parts. The location elements are X, Y, Z, y (yaw), p (pitch), and r (roll), from left to right. | X, Y, Z: For each, -1,000,000.000 to 1,000,000.000<br>y, r: For each, -180.000 to 180.000<br>p: 0.000 to 180.000 | 0.000 for all |
| (d) | Opening angle | Maximum Value | Set the maximum value of the opening angle of the hinge joint. The unit is [degree]. | -179.000 to 180.000<br>and equal to or more than the minimum value | 180.000 |
| | | Minimum Value | Set the minimum value of the opening angle of the hinge joint. The unit is [degree]. | -180.000 to 179.000<br>and equal to or less than the maximum value | -180.000 |
| (e) | Hinge joint illustration | | Illustrates the hinge joint and the opening angle. | --- | --- |
| (f) | 3D editing area | | Displays the movable parts to connect and the joint point.<br>You can move the joint point and snap it to the movable parts by operating the mouse. | --- | --- |

### ● *Ball Joint*



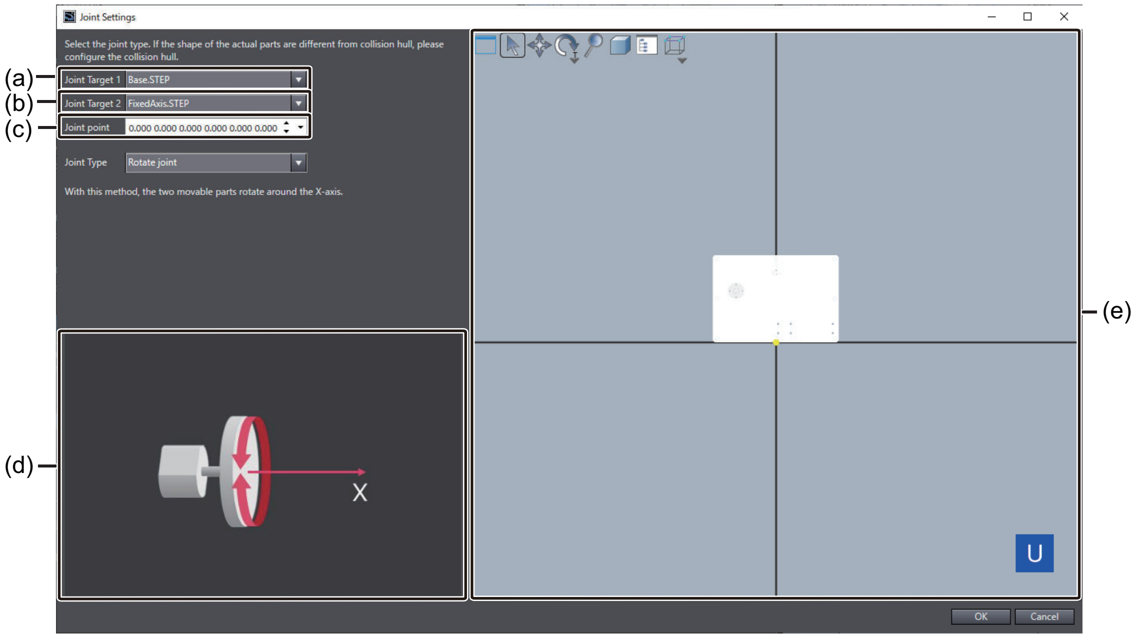

| | Item | Description | Set value | Initial value |
|---|---|---|---|---|
| (a) | Joint Target 1 | Select one of the two movable parts that are connected through the joint. | A movable part added to the custom mechanics | --- |
| (b) | Joint Target 2 | Select one of the two movable parts that are connected through the joint. | A movable part added to the custom mechanics | --- |
| (c) | Joint point | Set the coordinates of the joint point between the movable parts. The location elements are X, Y, Z, y (yaw), p (pitch), and r (roll), from left to right. | X, Y, Z: For each, -1,000,000.000 to 1,000,000.000<br>y, r: For each, -180.000 to 180.000<br>p: 0.000 to 180.000 | 0.000 for all |
| (d) | Y-axis max. opening angle | Set the maximum angle to which the ball joint opens in the Y-axis direction. The unit is [degree]. | 1.000 to 360.000 | 180.000 |
| (e) | Z-axis max. opening angle | Set the maximum angle to which the ball joint opens in the Z-axis direction. The unit is [degree]. | 1.000 to 360.000 | 180.000 |
| (f) | Ball joint illustration | Illustrates the ball joint and the Y- and Z-axis maximum opening angles. | --- | --- |
| (g) | 3D editing area | Displays the movable parts to connect and the joint point.<br>You can move the joint point and snap it to the movable parts by operating the mouse. | --- | --- |

## ● *Sliding Joint*



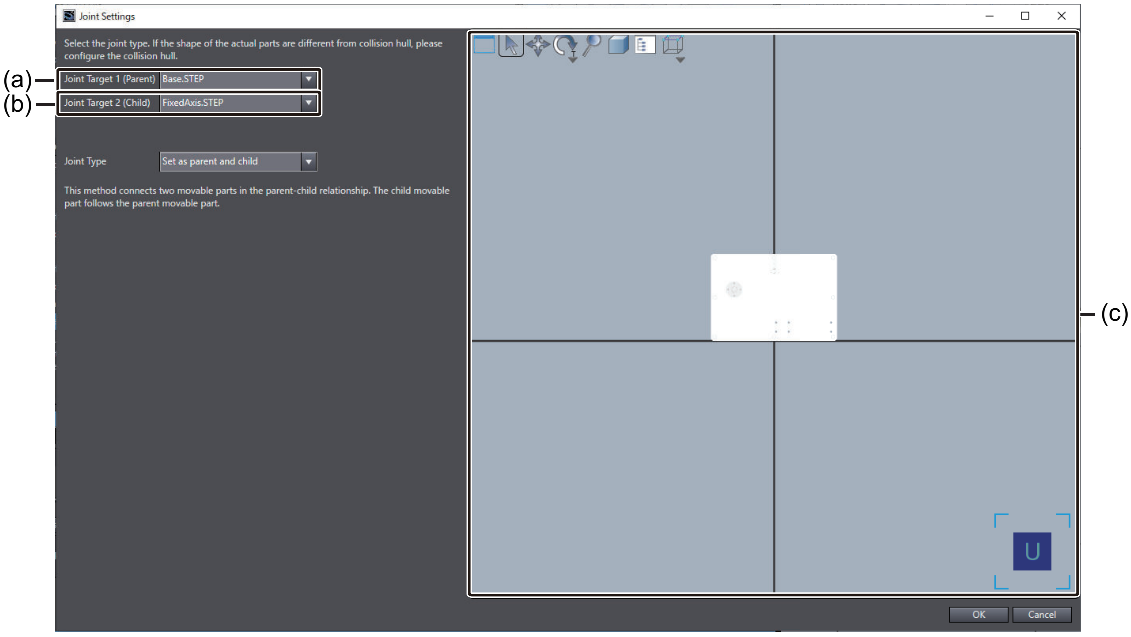

| | Item | | Description | Set value | Initial value |
|---|---|---|---|---|---|
| (a) | Joint Target 1 | | Select one of the two movable parts that are connected through the joint. | A movable part added to the custom mechanics | --- |
| (b) | Joint Target 2 | | Select one of the two movable parts that are connected through the joint. | A movable part added to the custom mechanics | --- |
| (c) | Joint point | | Set the coordinates of the joint point between the movable parts. The location elements are X, Y, Z, y (yaw), p (pitch), and r (roll), from left to right. | X, Y, Z: For each, -1,000,000.000 to 1,000,000.000<br>y, r: For each, -180.000 to 180.000<br>p: 0.000 to 180.000 | 0.000 for all |
| (d) | Sliding distance | Maximum Value | Set the maximum value of the sliding distance of the sliding joint. The unit is [mm]. | -3,999.000 to 4,000.000 and equal to or more than the minimum value | 1,000.000 |
| | | Minimum Value | Set the minimum value of the sliding distance of the sliding joint. The unit is [mm]. | -4,000.000 to 3,999.000 and equal to or less than the maximum value | -1,000.000 |
| (e) | Sliding joint illustration | | Illustrates the sliding joint and the sliding distance. | --- | --- |
| (f) | 3D editing area | | Displays the movable parts to connect and the joint point.<br>You can move the joint point and snap it to the movable parts by operating the mouse. | --- | --- |

## ● Rotational Joint




| | Item | Description | Set value | Initial value |
|---|---|---|---|---|
| (a) | Joint Target 1 | Select one of the two movable parts that are connected through the joint. | A movable part added to the custom mechanics | --- |
| (b) | Joint Target 2 | Select one of the two movable parts that are connected through the joint. | A movable part added to the custom mechanics | --- |
| (c) | Joint point | Set the coordinates of the joint point between the movable parts. The location elements are X, Y, Z, y (yaw), p (pitch), and r (roll), from left to right. | X, Y, Z: For each, -1,000,000.000 to 1,000,000.000<br>y, r: For each, -180.000 to 180.000<br>p: 0.000 to 180.000 | 0.000 for all |
| (d) | Rotational joint illustration | Illustrates the rotational joint. | --- | --- |
| (e) | 3D editing area | Displays the movable parts to connect and the joint point.<br>You can move the joint point and snap it to the movable parts by operating the mouse. | --- | --- |

### ● *Set as Parent and Child*




| | Item | Description | Set value | Initial value |
|---|---|---|---|---|
| (a) | Joint Target 1 (Parent) | Select a movable part to connect as parent. | A movable part added to the custom mechanics | --- |
| (b) | Joint Target 2 (Child) | Select a movable part to connect as child. | A movable part added to the custom mechanics | --- |
| (c) | 3D editing area | Displays the movable parts to connect. | --- | --- |

## 4-4-9 Configuring the Motion Direction Settings of the Movable Parts

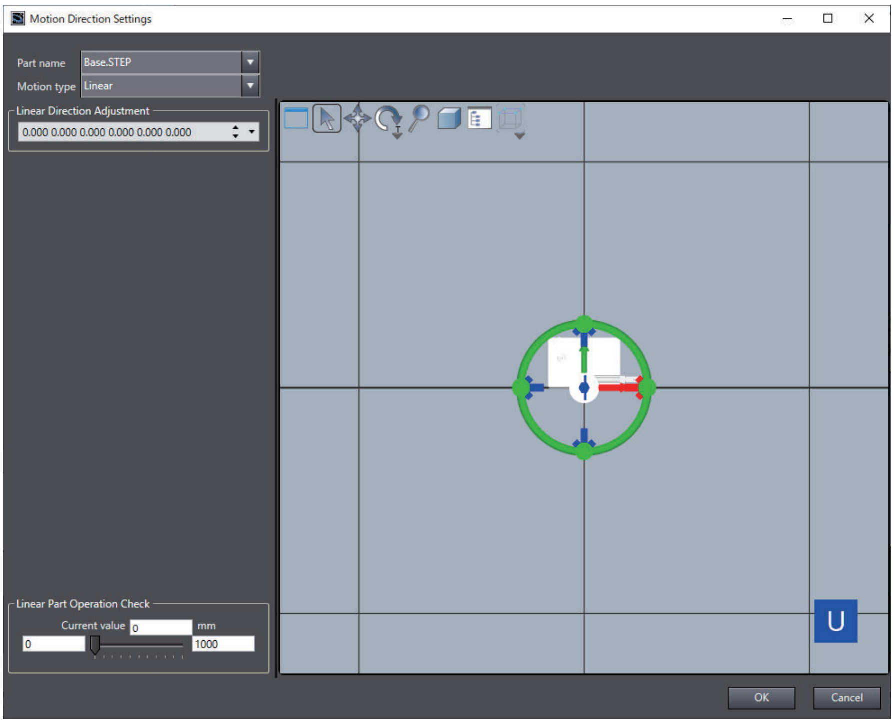
Configure the motion directions of movable parts. Select a motion type to set the motion direction from the three types: *Linear*, *Rotation*, and *Linear and Rotation*.

### Setting Procedure

Use the following procedure to configure the motion direction settings of movable parts.

**1** In the setup tab page for the custom mechanics, click the button at the right of **Motion Direction Settings**.

**2** The **Motion Direction Settings** dialog box is displayed.




**3** In **Part name**, select the movable part whose motion direction settings you want to configure.

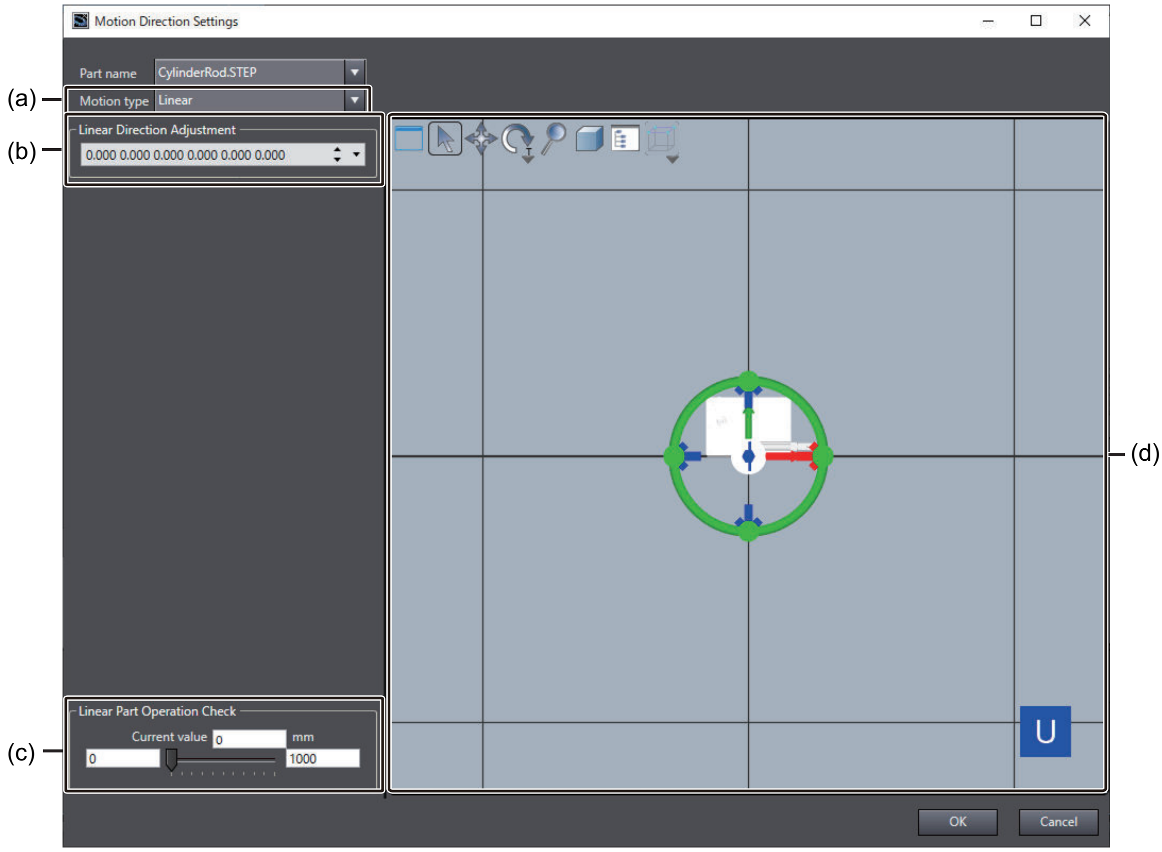
**4** In **Motion type**, select either of *Linear*, *Rotation*, or *Linear and Rotation*.
The setting items of the selected motion type are displayed under **Motion type**. The contents displayed depend on the motion type.

**5** Configure the motion direction settings, and click the **OK** button.
The motion direction settings of the movable part are saved.
Refer to *Motion Type Settings* on page 4-59 for details on the setting items of each motion type.

## Motion Type Settings

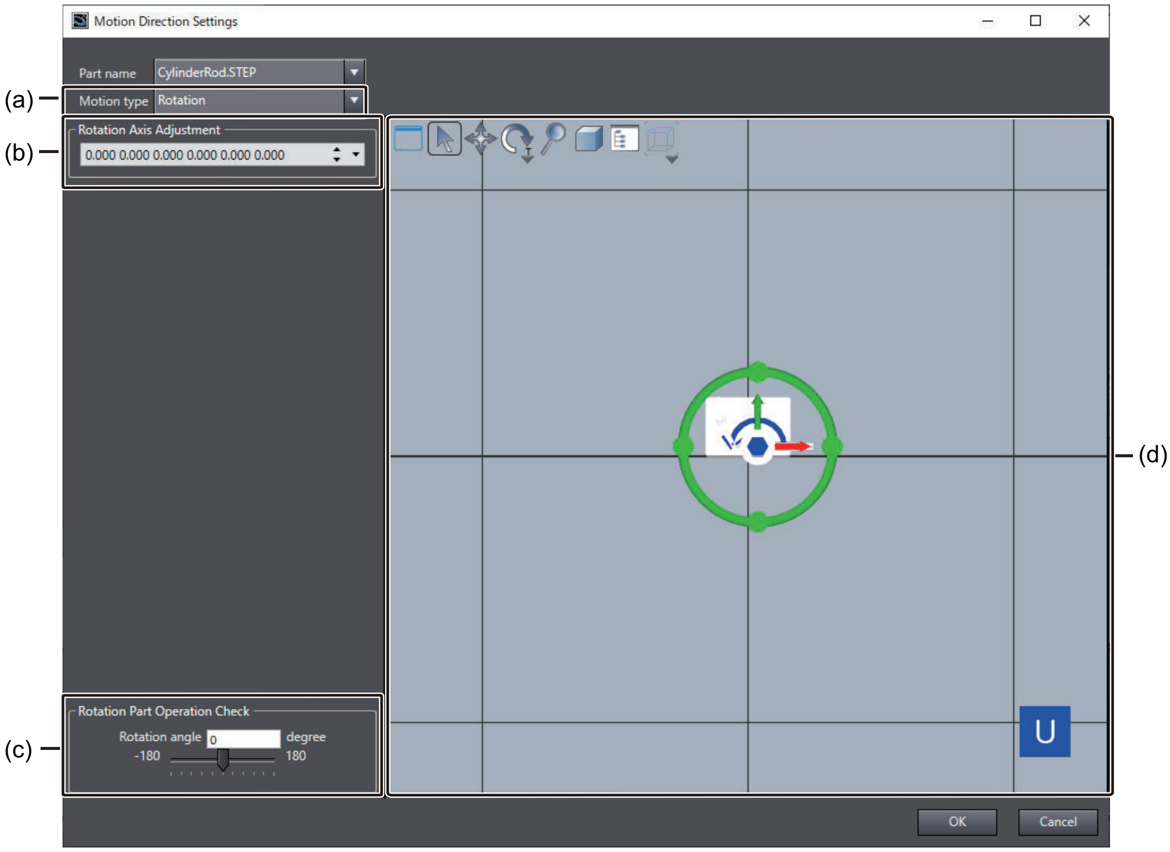
The setting items of each motion type are as follows.

## ● *Linear*



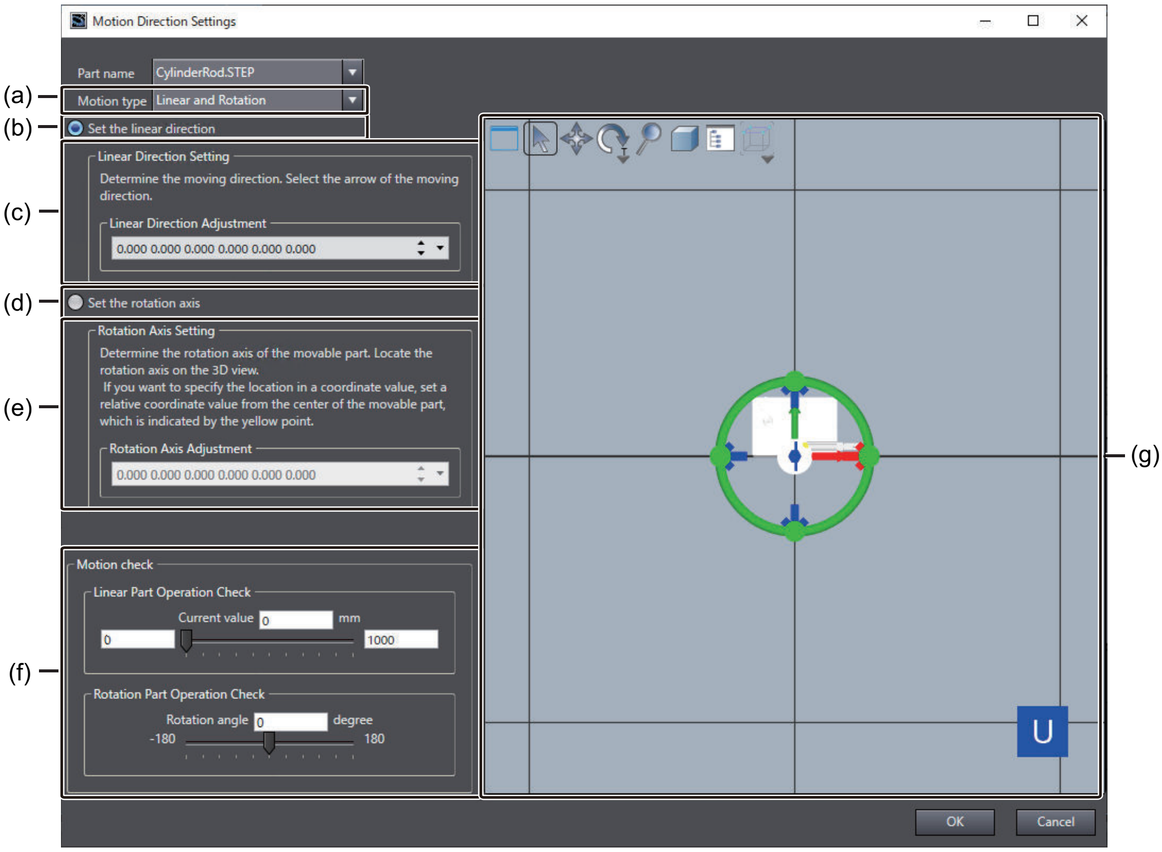

| | Item | Description | Set value | Initial value |
|---|---|---|---|---|
| (a) | Motion type | Set the motion type of the movable part.<br>Select *Linear* here. | Linear | None |
| (b) | Linear Direction Adjustment | Set the start position of operation and posture of the linear part. The location elements are X, Y, Z, y (yaw), p (pitch), and r (roll), from left to right. | X, Y, Z: For each, -3.402823E+38 to 3.402823E+38<br>y, r: For each, -180.000 to 180.000<br>p: 0.000 to 180.000 | 0.000 for all |
| (c) | Linear Part Operation Check | Move the slider to check the operation of the linear part. | --- | Minimum value: 0<br>Maximum value: 1,000 |
| (d) | 3D editing area | Set the position and pose of the linear part.<br>Click one of the six blue arrow icons on the 3D editing area to select the linear direction of the movable part. The selected arrow icon turns red.<br>The motion of the linear part is displayed according to the operation that you make in **Linear Part Operation Check**. | --- | --- |

## ● *Rotation*




| | Item | Description | Set value | Initial value |
|---|---|---|---|---|
| (a) | Motion type | Set the motion type of the movable part.<br>Select *Rotation* here. | Rotation | None |
| (b) | Rotation Axis Adjustment | Set the start position of operation and posture of the rotation part. The location elements are X, Y, Z, y (yaw), p (pitch), and r (roll), from left to right. | X, Y, Z: For each, -3.402823E+38 to 3.402823E+38<br>y, r: For each, -180.000 to 180.000<br>p: 0.000 to 180.000 | 0.000 for all |
| (c) | Rotation Part Operation Check | Move the slider to check the operation of the rotation part. | --- | --- |
| (d) | 3D editing area | Set the position and pose of the rotation part.<br>In **Rotation Axis Adjustment**, enter the X, Y, Z, y, p, and r values to set the rotation axis direction. Entering the values changes the direction of the rotation axis direction edit icon on the 3D editing area. Alternatively, you can drag the rotation axis direction edit icon on the 3D editing area to set the direction. | --- | --- |

## ● *Linear and Rotation*




| | Item | Description | Set value | Initial value |
|---|---|---|---|---|
| (a) | Motion type | Set the motion type of the movable part.<br>Select *Linear and Rotation* here. | Linear and Rotation | None |
| (b) | Set the linear direction | Select this button when you set the linear direction. | --- | --- |
| (c) | Linear Direction Setting | Set the start position of operation and posture of the linear part. The location elements are X, Y, Z, y (yaw), p (pitch), and r (roll), from left to right. | X, Y, Z: For each, -3.402823E+38 to 3.402823E+38<br>y, r: For each, -180.000 to 180.000<br>p: 0.000 to 180.000 | 0.000 for all |
| (d) | Set the rotation axis | Select this button when you set the rotation axis. | --- | --- |
| (e) | Rotation Axis Setting | Set the start position of operation and posture of the rotation part. The location elements are X, Y, Z, y (yaw), p (pitch), and r (roll), from left to right. | X, Y, Z: For each, -3.402823E+38 to 3.402823E+38<br>y, r: For each, -180.000 to 180.000<br>p: 0.000 to 180.000 | 0.000 for all |

| | Item | | Description | Set value | Initial value |
|---|---|---|---|---|---|
| (f) | Motion check | Linear Part Operation Check | Move the slider to check the operation of the linear part. | --- | Minimum value: 0<br>Maximum value: 1,000 |
| | | Rotation Part Operation Check | Move the slider to check the operation of the rotation part. | --- | --- |
| (g) | 3D editing area | | Set the position and pose of the linear part when you selected **Set the linear direction**. The motion of the linear part is displayed according to the operation that you make in **Linear Part Operation Check**.<br>Set the position and pose of the rotation part when you selected **Set the rotation axis**. The motion of the rotation part is displayed according to the operation that you make in **Rotation Part Operation Check**. | --- | --- |

## 4-4-10 Checking the Motion of the Custom Mechanics

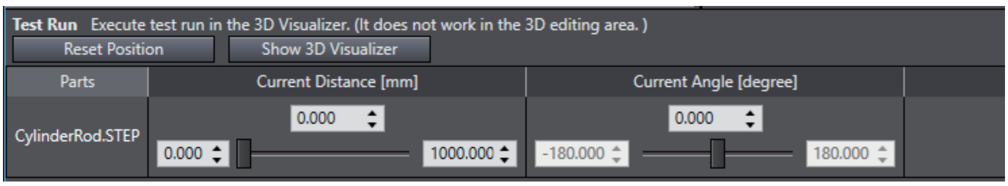
Operate the movable parts of the custom mechanics to check that the motion settings of the movable parts and joint are correct. To perform the check, use the Test Run pane for movable parts.

### Operating Procedure

Use the following procedure to operate the Test Run pane for movable parts.

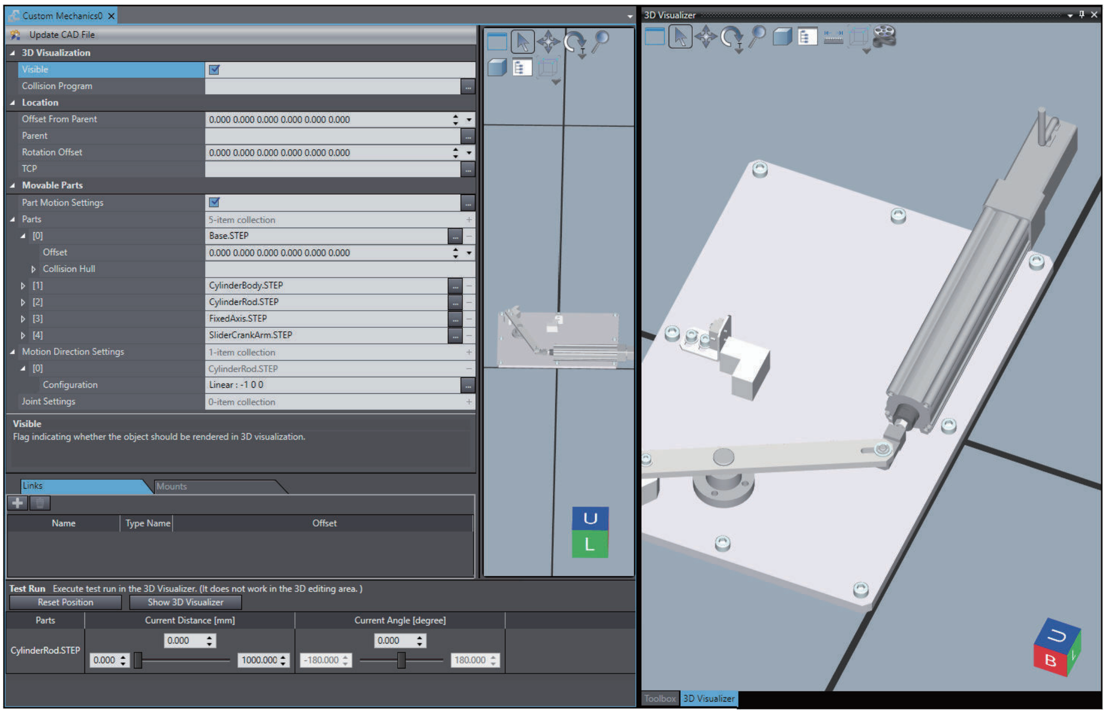
**1** In **3D Visualization** under **Configurations and Setup**, double-click the target custom mechanics.
The setup tab page for custom mechanics is displayed. In the **Test Run** pane at the bottom of the setup tab page for the custom mechanics, the movable parts with the *Linear*, *Rotation*, or *Linear and Rotation* setting are displayed.

**2** In the **Test Run** pane, set the travel distance and rotation angle of each movable part.

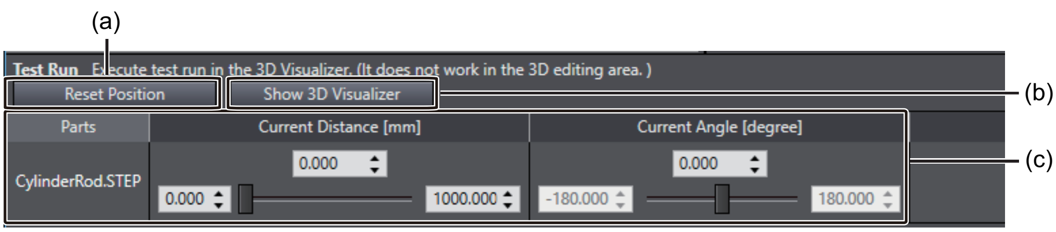



**3** To do so, click the sliders for the travel distance and rotation angle.

The target movable part in the 3D Visualizer moves by the amount of travel distance and rotation angle.



## Movable Parts Test Run Pane




| | Item | | Description | Set value | Initial value |
|---|---|---|---|---|---|
| (a) | **Reset Position** button | | Clicking the Reset Position button resets the position of the movable part that moved in a test run. | --- | --- |
| (b) | **Show 3D Visualizer** button | | Clicking this button displays the **3D Visualizer** in the same place of the toolbox pane. | --- | --- |
| (c) | Movable parts list | Parts | Displays a movable part with the *Linear*, *Rotation*, or *Linear and Rotation* setting. | --- | --- |
| | | Current Distance [mm] | Displays the travel distance and rotation angle of the movable part during the test run. You can also change the displayed values. This item is displayed when the motion type of the movable part is *Linear* or *Linear and Rotation*. | --- | --- |

| Item | | | Description | Set value | Initial value |
|---|---|---|---|---|---|
| | | Minimum value | Set the **current value** and the minimum value that can be set using the **slider**. If you attempt to set a value larger than the maximum value, the same value as the maximum value will be set. | -10000.000 to maximum value | 0.000 |
| | | Maximum value | Set the **current value** and the maximum value that can be set using the **slider**. If you attempt to set a value smaller than the minimum value, the same value as the minimum value will be set. | Minimum value to 10000.000 | 1000.000 |
| | | Current value | Displays the current value of the travel distance of the movable part. If you enter a value between the **minimum value** and the **maximum value** in the text box, the movable part will move to the position of that value. If the value is smaller than the minimum value or larger than the maximum value, the movable part will move to the position of the **minimum value** or **maximum value**, respectively. | Minimum value to maximum value | 0.000 |
| | | Slider | Operate this slider to change the current value of the travel distance between the minimum value and the maximum value. | Minimum value to maximum value | 0.000 |
| | Current Angle [degree] | | Displays the rotation angle of the movable part during the test run. You can also change the displayed values. This item is displayed when the motion type of the movable part is *Linear* or *Linear and Rotation*. | --- | --- |
| | | Minimum value | Displays the **current value** and the minimum value that can be set using the **slider**. | --- | -180 (fixed) |
| | | Maximum value | Displays the **current value** and the maximum value that can be set using the **slider**. | --- | 180 (fixed) |
| | | Current value | Displays the current value of the rotation angle of the movable part. If you enter a value between the **minimum value** and the **maximum value** in the text box, the movable part will rotate to the angle of that value. If the value is smaller than the minimum value or larger than the maximum value, the movable part will rotate to the angle of the **minimum value** or **maximum value**, respectively. | Minimum value to maximum value | 0.000 |

| | Item | | | Description | Set value | Initial value |
|---|---|---|---|---|---|---|
| | | | Slider | Operate this slider to change the current value of the rotation angle between the minimum value and the maximum value. | Minimum value to maximum value | 0.000 |

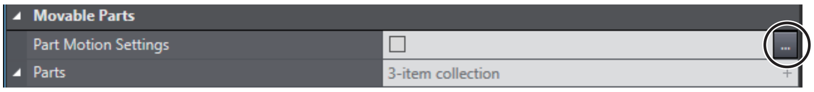
## 4-4-11 Configuring the Motion Settings of the Custom Mechanics

Configure the parameters that define the motions of movable parts of custom mechanics. There are two ways of positioning, i.e., positioning by specifying a motion number and positioning by specifying a variable value. You can move movable parts according to the set parameters by executing a 3D simulation.

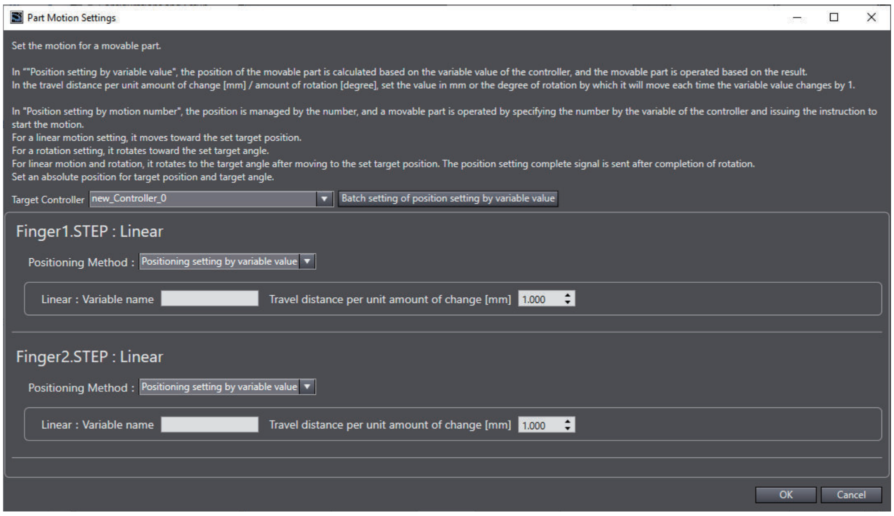
### Operating Procedure

#### ● For Positioning by Specifying a Motion Number

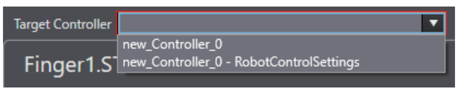
**1** Click the button at the right of **Part Motion Settings**.



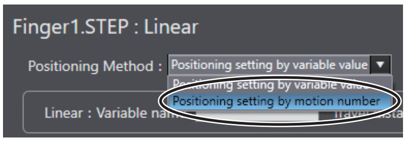

The Part Motion Settings dialog box for movable parts is displayed.




**2** Select the Controller to control the custom mechanics from the **Target Controller** pull-down.
You can select the Controller or sub-device of the Robot Integrated CPU Unit that has been added in the project.




**3** Select **Positioning setting by motion number**.



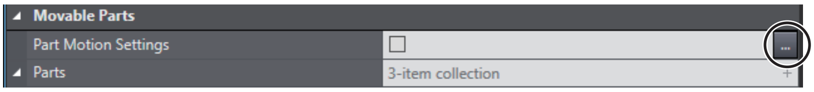

**4** Set the corresponding variable names to control the movable part to the **I/O variable (Start positioning)**, **I/O variable (Finish positioning)**, **Variable (Target motion number)**, **Variable (Current position)**.
Refer to *For Positioning Setting by Motion Number* on page 4-69 for details on the variables to set.

**5** Set the motion parameters to control the movable part.
Refer to *For Positioning Setting by Motion Number* on page 4-69 for details on the motion parameters to set.

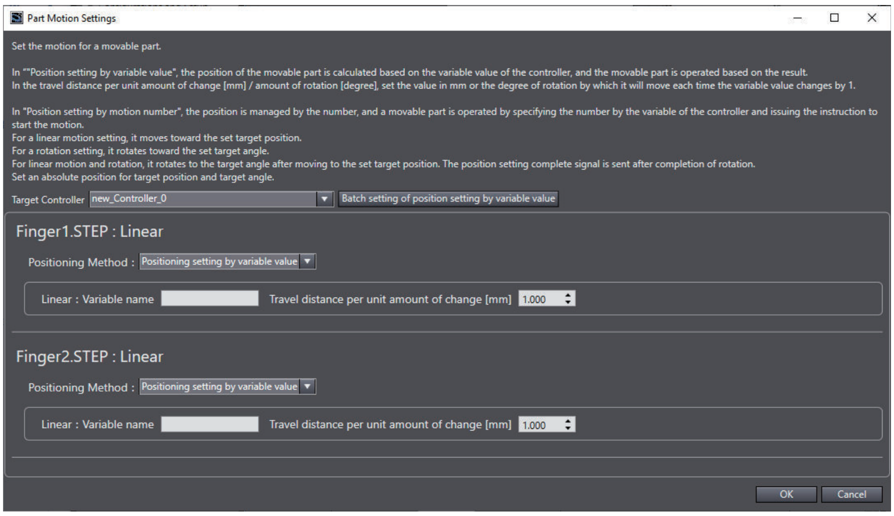
**6** Repeat steps 3 to 5 for all movable parts displayed on the Part Motion Settings dialog box.

### ● For Positioning by Specifying a Variable Value

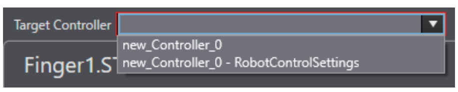
**1** Click the button at the right of **Part Motion Settings**.



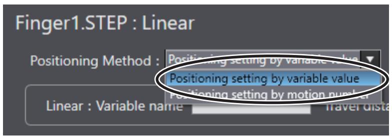

The Part Motion Settings dialog box for movable parts is displayed.




**2** Select the Controller to control the custom mechanics from the **Target Controller** pull-down.
You can select the Controller or sub-device of the Robot Integrated CPU Unit that has been added in the project.



**3** Select **Positioning setting by variable value**.

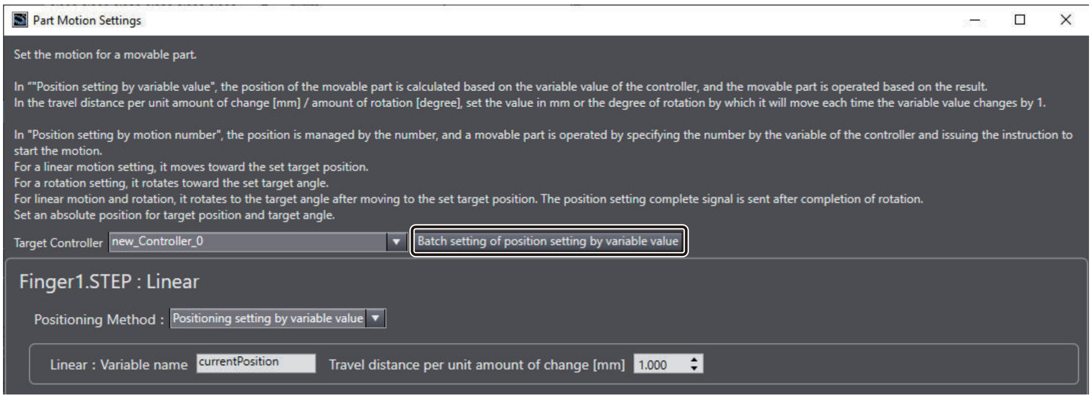



**4** For **Linear** and **Rotation**, set the **Variable name** and the **Travel distance per unit amount of change [mm]/Amount of rotation per unit amount of change [degree]**.
Refer to *For Positioning Setting by Variable Value* on page 4-71 for details on the parameters.

**5** Repeat steps 3 to 4 for all movable parts displayed on the Part Motion Settings dialog box.

### Additional Information

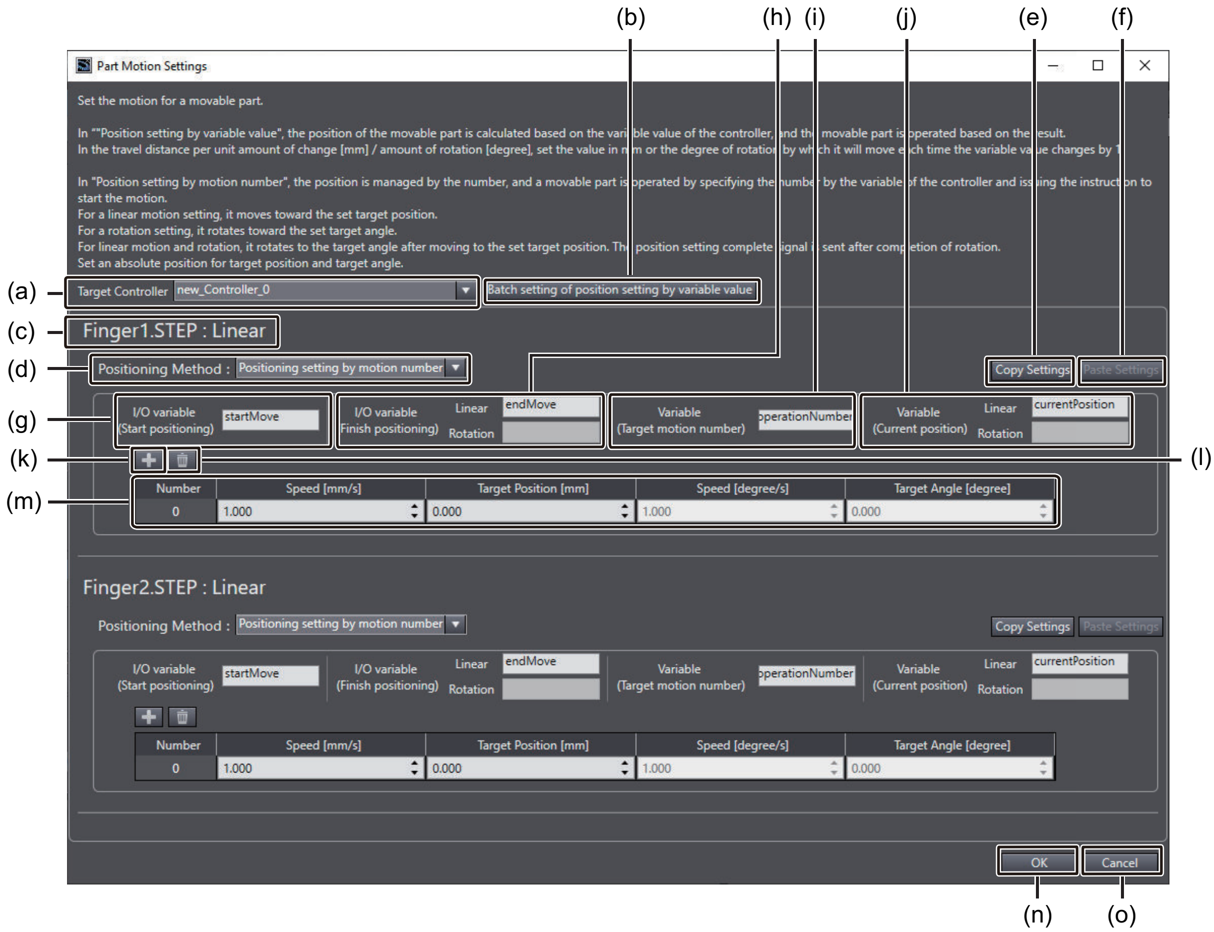
You can also set all movable parts whose positioning method is set to *Positioning setting by variable value* at once.
Click the **Batch setting of position setting by variable value** button in the Part Motion Settings dialog box.




Refer to *For Positioning Setting by Variable Value (Batch Setting)* on page 4-72 for the setting items for batch setting of position setting by variable value.

## Settings

### ● For Positioning Setting by Motion Number



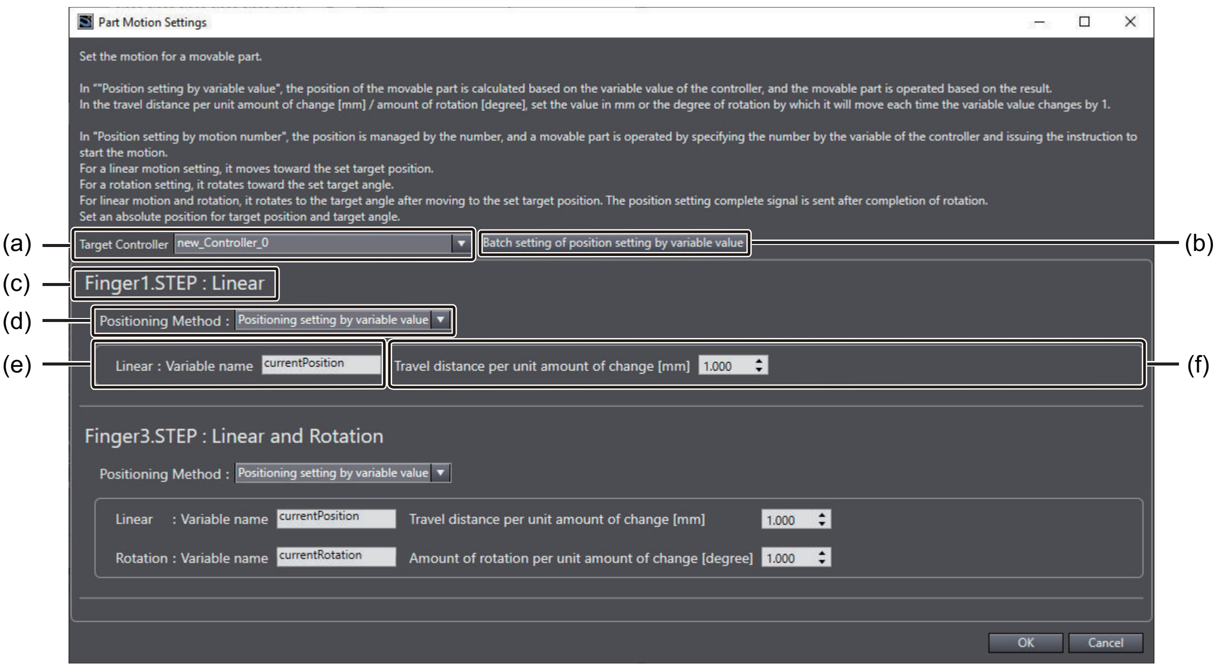

| | Item | Description | Set value | Initial value |
|---|---|---|---|---|
| (a) | Target Controller | Select the Controller to control a custom mechanics. | The Controller or sub-device of the Robot Integrated CPU Unit that has been added in the project. | --- |
| (b) | **Batch setting of position setting by variable value** button | Sets all movable parts whose positioning method is set to *Positioning setting by variable value* at once. Refer to *For Positioning Setting by Variable Value (Batch Setting)* on page 4-72 for the settings. | --- | --- |
| (c) | Movable part name | Displays the name of the movable part registered in Motion Direction Settings. | --- | --- |
| (d) | Positioning Method | Select the positioning method for the movable part. | Positioning setting by variable value/ Positioning setting by motion number | Positioning setting by variable value |

| | Item | | Description | Set value | Initial value |
|---|---|---|---|---|---|
| (e) | **Copy Settings** button | | Copies the settings of the I/O variable (Start positioning), I/O variable (Finish positioning), Variable (Target motion number), Variable (Current position), and motion parameters at once. | --- | --- |
| (f) | **Paste Settings** button | | Pastes at once the settings of another movable part name that were copied by the **Copy Settings** button. | --- | --- |
| (g) | I/O variable (Start positioning) | | Set a BOOL global variable that triggers positioning operation. When the Target Controller is a sub-device of a Robot Integrated CPU Unit, you can set a V+Digital I/O number. | A BOOL global variable or V+Digital I/O number | --- |
| (h) | I/O variable (Finish positioning) | Linear | Set a BOOL global variable in which the value of the linear motion positioning finish signal is written. When the Target Controller is a sub-device of a Robot Integrated CPU Unit, you can set a V+Digital I/O number. | A BOOL global variable or V+Digital I/O number | --- |
| | | Rotation | Set a BOOL global variable in which the value of the rotation positioning finish signal is written. When the Target Controller is a sub-device of a Robot Integrated CPU Unit, you can set a V +Digital I/O number. | A BOOL global variable or V+Digital I/O number | --- |
| (i) | Variable (Target motion number) | | Set a UDINT global variable that specifies the motion number. When the Target Controller is a sub-device of a Robot Integrated CPU Unit, you can set a REAL V+variable. | A UDINT global variable or REAL V+variable | --- |
| (j) | Variable (Current position) | Linear | Set an LREAL global variable in which the value of the current linear motion position is written. When the Target Controller is a sub-device of a Robot Integrated CPU Unit, you can set a REAL V+variable. | An LREAL global variable or REAL V+variable | --- |
| | | Rotation | Set an LREAL global variable in which the value of the current rotation position is written. When the Target Controller is a sub-device of a Robot Integrated CPU Unit, you can set a REAL V+variable. | An LREAL global variable or REAL V+variable | --- |
| (k) | Add button | | Adds a motion parameter line. When a parameter line is selected, a new line is added under the selected line. | --- | --- |
| (l) | Delete button | | Deletes a motion parameter line. | --- | --- |
| (m) | Motion parameters | Number | Specifies motion parameters. | --- | --- |

| | Item | | Description | Set value | Initial value |
|---|---|---|---|---|---|
| | | Speed [mm/s] | Set a velocity at which the movable part moves to the target position. If the speed value is positive, the movable part moves closer to the target position. If the value is negative, it moves away from the target position. | -10,000.000 to 10,000.000 | 1.000 |
| | | Target Position [mm] | Set a target position. If the speed value is positive, the movable part moves closer to the target position. If the value is negative, it moves away from the target position. | -4,000.000 to 4,000.000 | 0.000 |
| | | Speed [degree/s] | Set a target velocity at which the movable part rotates to the target angle. If the speed value is positive, the movable part moves closer to the target angle. If the value is negative, it moves away from the target angle. | -10,000.000 to 10,000.000 | 1.000 |
| | | Target Angle [degree] | Set a target angle. If the speed value is positive, the movable part moves closer to the target angle. If the value is negative, it moves away from the target angle. | -180.000 to 180.000 | 0.000 |
| (n) | **OK** button | | Confirms the motion settings of the movable part. | --- | --- |
| (o) | **Cancel** button | | Cancels the settings. | --- | --- |

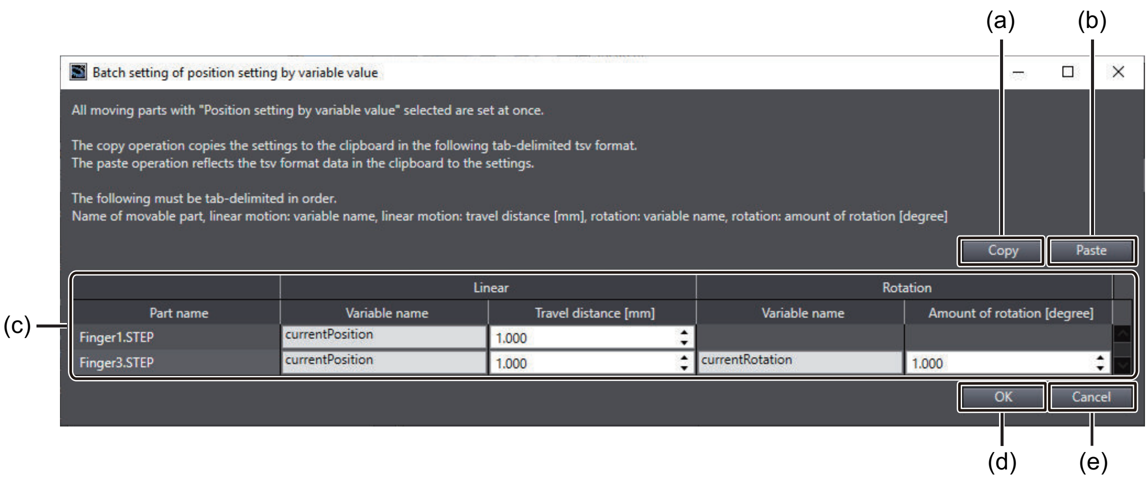
Refer to the *NJ-series Robot Integrated CPU Unit User's Manual (Cat. No. O037)* and *Sysmac Studio Robot Integrated System Building Function with Robot Integrated CPU Unit Operation Manual (Cat. No. W595)* for information on Robot Integrated CPU Units, V+Digital I/O numbers, and V+variables.

## ● For Positioning Setting by Variable Value



| | Item | Description | Set value | Initial value |
|---|---|---|---|---|
| (a) | Target Controller | Select the Controller to control a custom mechanics. | The Controller or sub-device of the Robot Integrated CPU Unit that has been added in the project. | --- |
| (b) | **Batch setting of position setting by variable value** button | Sets all movable parts whose positioning method is set to *Positioning setting by variable value* at once. Refer to *For Positioning Setting by Variable Value (Batch Setting)* on page 4-72 for the settings. | --- | --- |
| (c) | Movable part name | Displays the name of the movable part registered in Motion Direction Settings. | --- | --- |
| (d) | Positioning Method | Select the positioning method for the movable part. | Positioning setting by variable value/ Positioning setting by motion number | Positioning setting by variable value |
| (e) | Variable name | Set the name of the LREAL variable to get as the current position. Set this for both **Linear** and **Rotation**. | LREAL global variable | --- |
| (f) | Travel distance per unit amount of change [mm]/ Amount of rotation per unit amount of change [degree] | This is the unit of measure for calculating the current position of the movable part. The movable part will be moved to the position calculated by multiplying the value of the variable specified in **Variable name** by the value set in this item. Set this for both **Linear** and **Rotation**. | -10000.000 to 10000.000 | -1.0 |

## ● For Positioning Setting by Variable Value (Batch Setting)




| | Item | Description |
|---|---|---|
| (a) | **Copy** button | Copies the settings to the clipboard in tsv format. Table titles such as **Linear** and **Rotation** are not included as headers. |
| (b) | **Paste** button | Pastes the tsv format text from the clipboard back to the settings. Among the text lines, only those with matching movable part names are pasted. |

| | Item | Description |
|---|---|---|
| (c) | Settings | Lists the settings of movable parts whose positioning method is *Positioning setting by variable value*. Settings for unsupported motion types are not listed. |
| (d) | **OK** button | Accepts the settings and closes the **Batch setting of position setting by variable value** pane. |
| (e) | **Cancel** button | Cancels the settings and closes the **Batch setting of position setting by variable value** pane. |

## 4-4-12 Updating CAD Data

You can replace all CAD Data of registered movable parts at once.

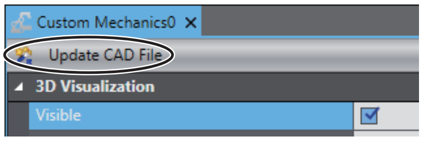
### Operating Procedure

Use the following procedure.

**Additional Information**

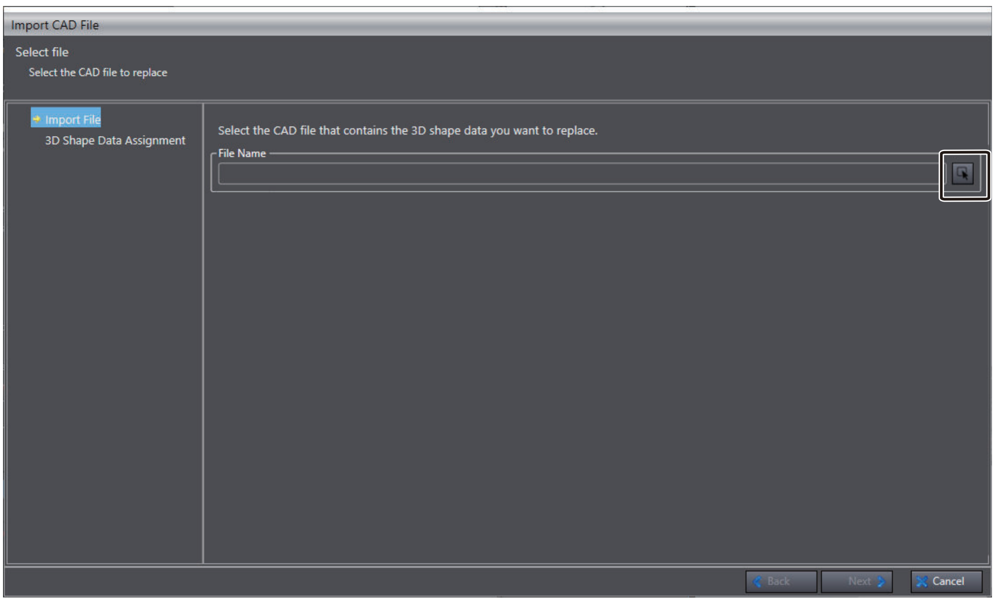
You can optionally set the mesh coarseness of CAD data at the time of import. Refer to the *Sysmac Studio Robot Integrated System Building Function with Robot Integrated CPU Unit Operation Manual (Cat. No. W595)* for details on the option settings.

**1** In the setup tab page for the target custom mechanics, click the **Update CAD File** button.



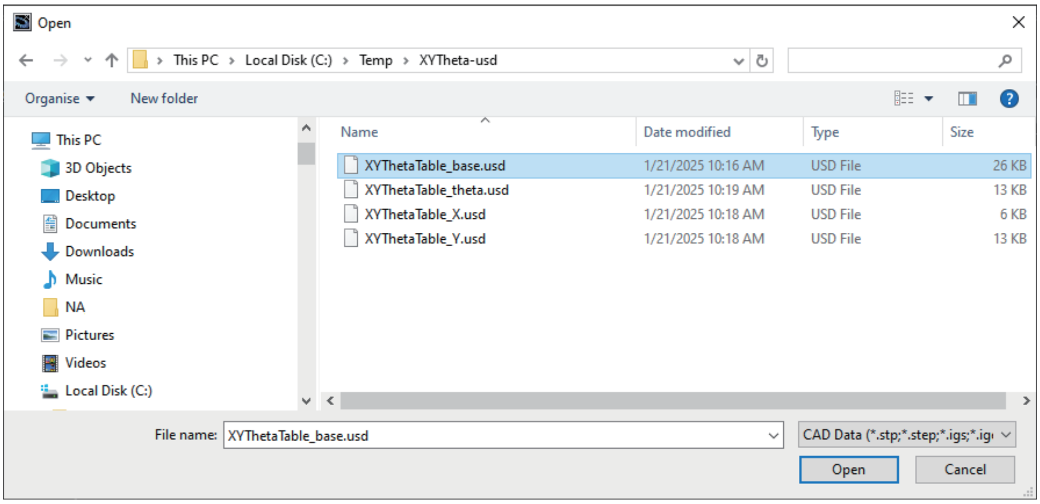

The wizard starts and the **Select file** page is displayed.

**2** In the **Select file** page, click the button at the right of **File Name**.




The **Open** dialog box is displayed.

**3** In the **Open** dialog box, select the STEP, IGES, or OpenUSD CAD data file for update, and then click the **Open** button.




In the **Select file** dialog box, click the **Next** button. The **3D Shape Data Assignment** page is displayed.

**4** In the **3D Shape Data Assignment** page, assign the CAD Data to a 3D shape data.
Refer to *10-1-1 Procedure to Update All CAD Data* on page 10-2 for the procedure of operating the page.

**Additional Information**

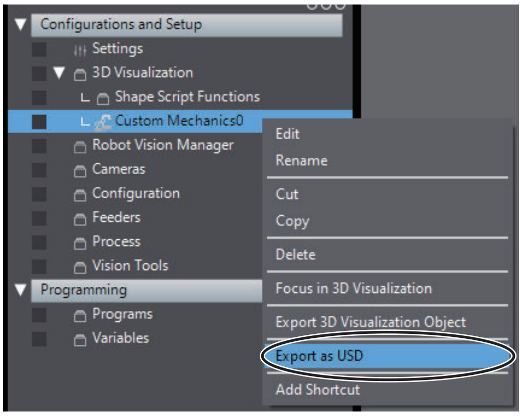
If you update existing 3D shape data that has the same prim path, the 3D shape data assignment operation will be done automatically.

## 4-4-13 Exporting OpenUSD Files

You can export registered custom mechanics in OpenUSD file format. OpenUSD files that include joint information can be exported only when the joint settings and motion direction settings are configured for each movable part.

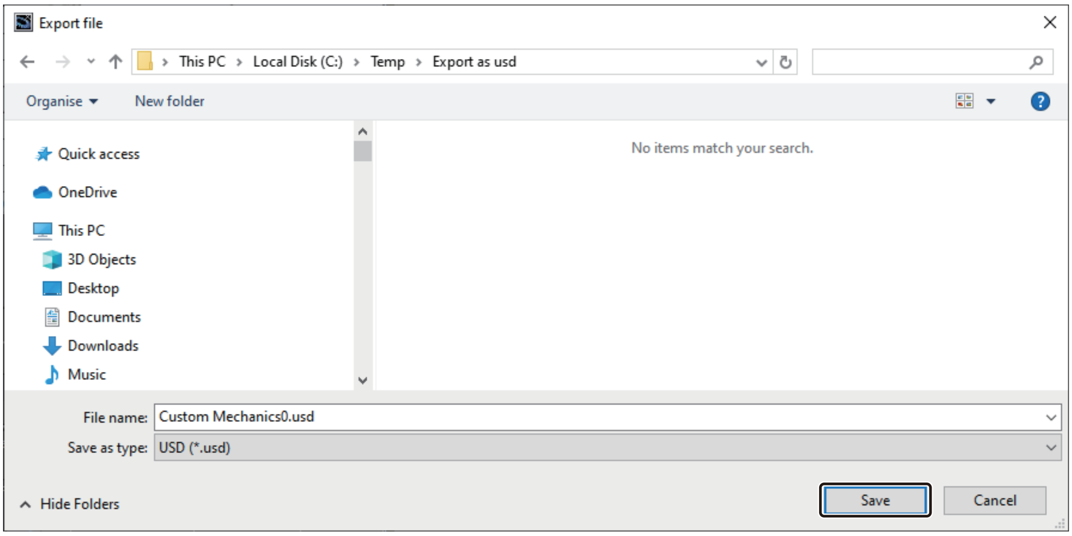
### Export Procedure

**1** Right-click *Custom Mechanics0* under **Configurations and Setup** – **3D Visualization** in the Multiview Explorer and select **Export as USD** from the menu.




The **Export file** dialog box is displayed.

**2** Enter the file name, select *USD*, *USDA*, or *USDC* for the Save as type, and then click the **Save** button.




The exported data will be saved in OpenUSD format files, one for each movable part of the custom mechanics and another for the configuration information for the movable parts.

## Types of Data Included in Exported OpenUSD Files

Types of data included in exported OpenUSD files are as follows.

- Geometry (UsdGeomMesh)
- Material (To define the material model)
- Joint (UsdPhysicsJoint)
  There are the following three joint types.
  UsdPhysicsRevoluteJoint/UsdPhysicsPrismaticJoint/UsdPhysicsFixedJoint

The table below shows the correspondence between the joint motion settings in the Sysmac Studio and the USDPhysicsJoint in OpenUSD to be exported.

| Joint motion settings in the Sysmac Studio | USDPhysicsJoint in the OpenUSD to be exported |
|---|---|
| Sliding Joint | UsdPhysicsPrismaticJoint |

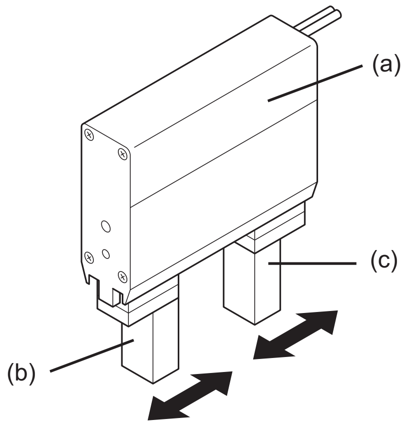
| Joint motion settings in the Sysmac Studio | USDPhysicsJoint in the OpenUSD to be exported |
|---|---|
| Rotational joint | UsdPhysicsRevoluteJoint |
| Fixed Joint | UsdPhysicsFixedJoint |

Refer to *4-4-8 Configuring the Joint Motion* on page 4-50 for details on the joint motion settings.

## 4-4-14 Example of Creating a Custom Mechanics

This section describes the procedure of creating custom mechanics and configuring the motion settings by taking an electric chuck (2-finger type) and electric cylinder as examples.

### Electric Chuck (2-finger Type)




Movable parts:

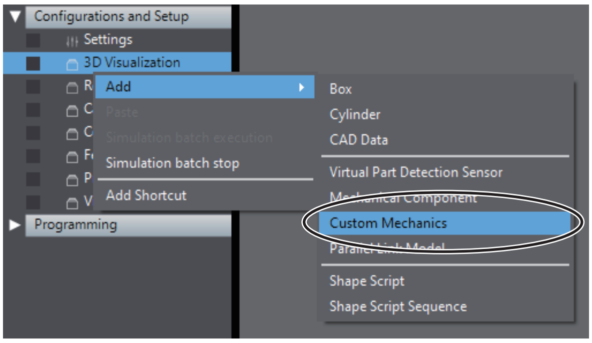
| | Movable part name (CAD Data parts) | Motion type |
|---|---|---|
| (a) | Base | None |
| (b) | Finger 1 | Linear |
| (c) | Finger 2 | Linear |

Joint: None

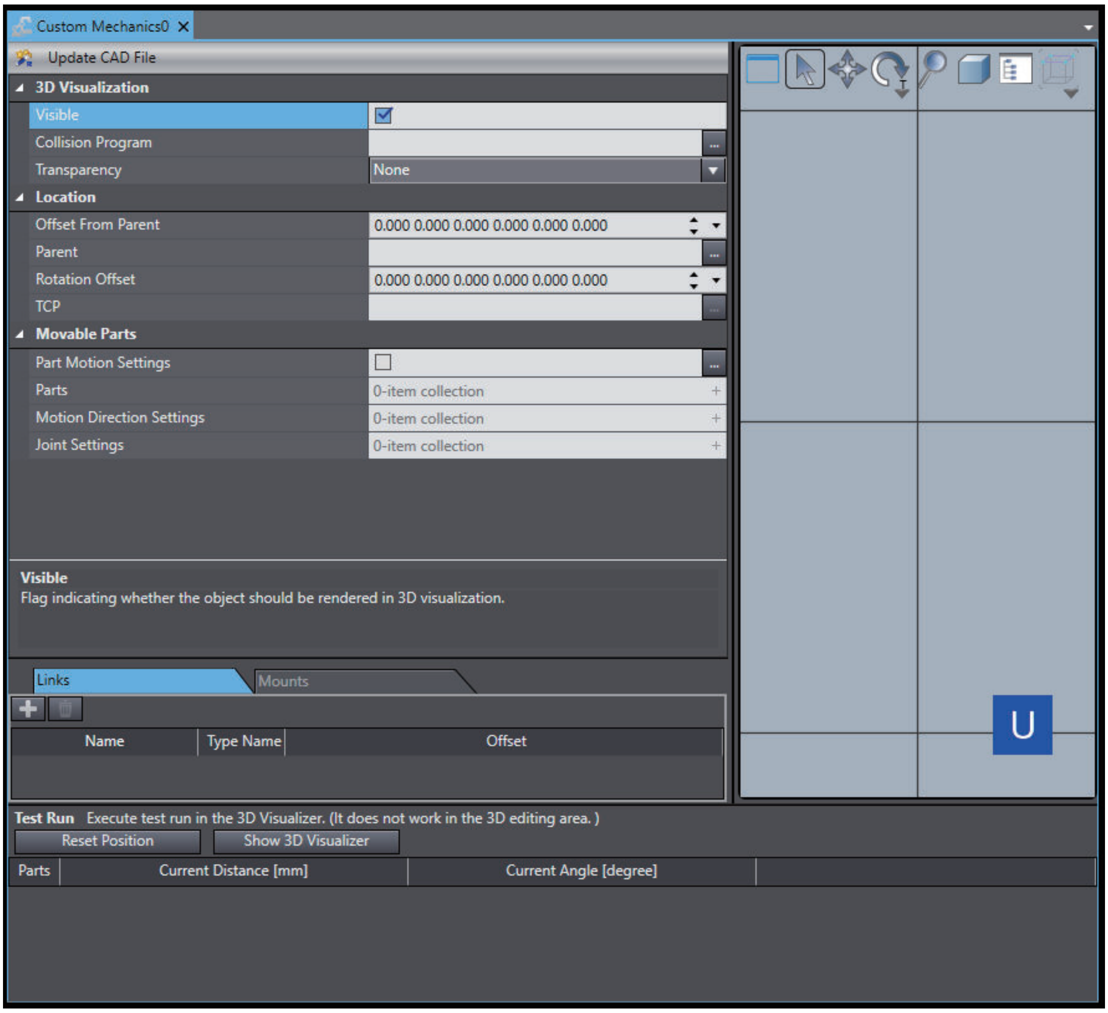
#### ● Creating Procedure

**1** Use 3D CAD software and save each movable part of the electric chuck (2-finger type) separately.
Refer to *4-3-2 Preparations for CAD Data Files* on page 4-10 for how to output CAD Data files with 3D CAD software.

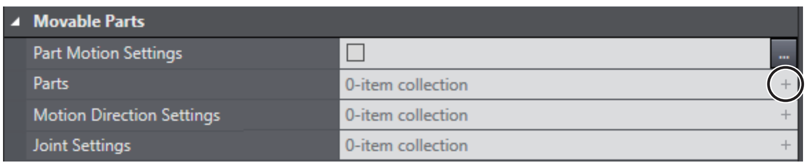
**2** Right-click **3D Visualization** under **Configurations and Setup** and select **Add** – **Custom Mechanics** from the menu to register a custom mechanics.




**3** Double-click the custom mechanics that you registered to display the setup tab page for the custom mechanics.

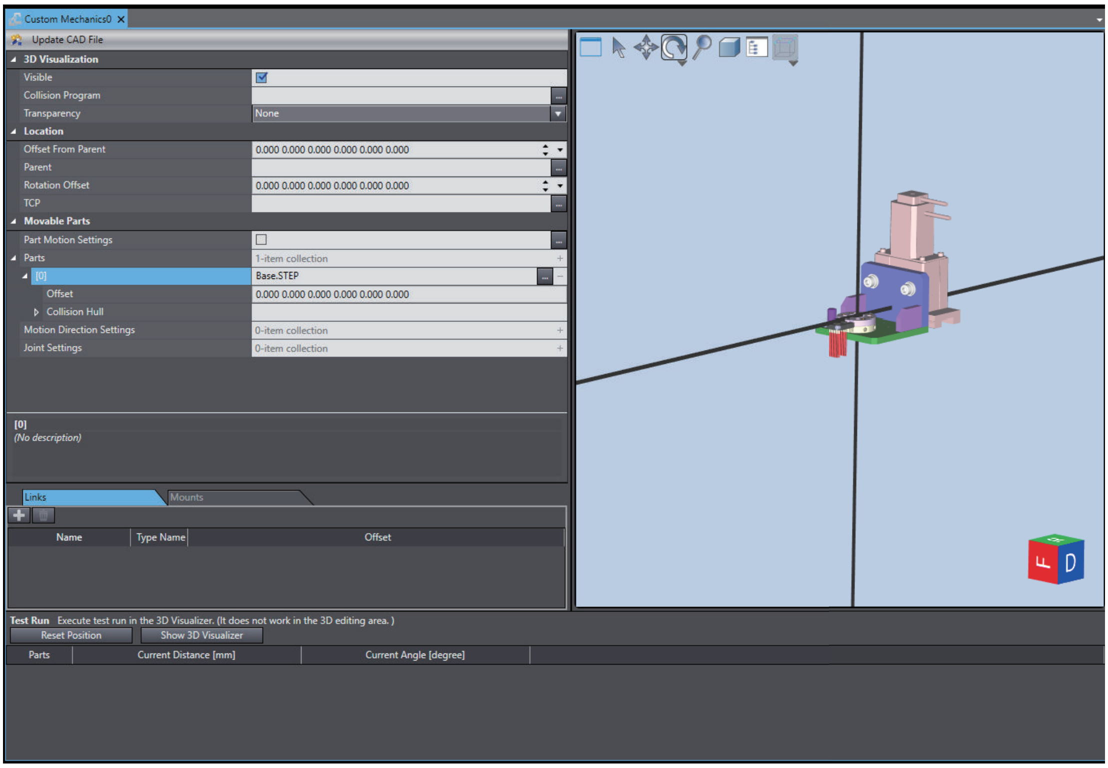



**4** Click the Add button at the right of **Parts**.

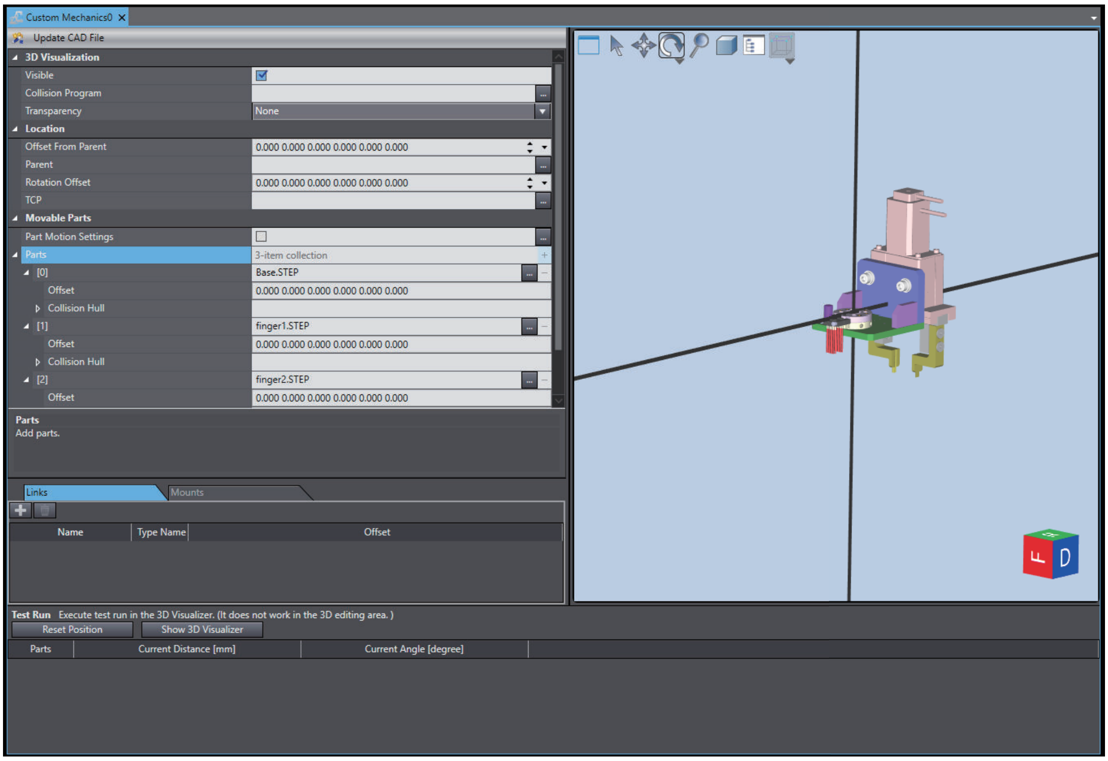



The **Open** dialog box is displayed.

**5** In the **Open** dialog box, select the CAD Data file of a movable part that you saved at step 1, and then click the **Open** button.
The item name of the selected CAD Data is added. Also, the data of the CAD file that you imported is displayed in the 3D editing area.

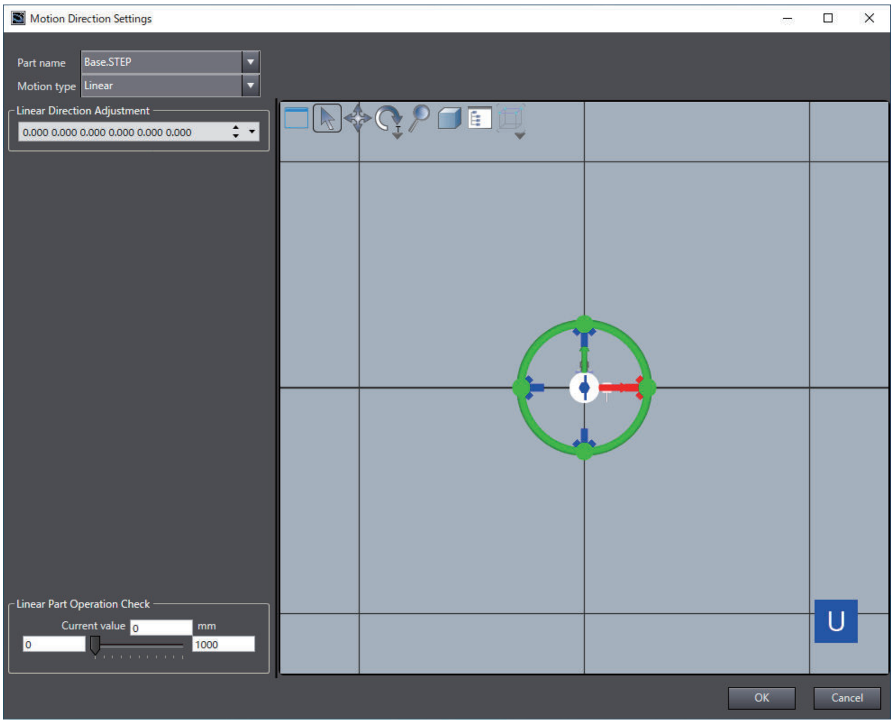


**6** Add the CAD Data of all movable parts in the same way.



**7** Click the button at the right of **Motion Direction Settings** to add a motion direction setting item.

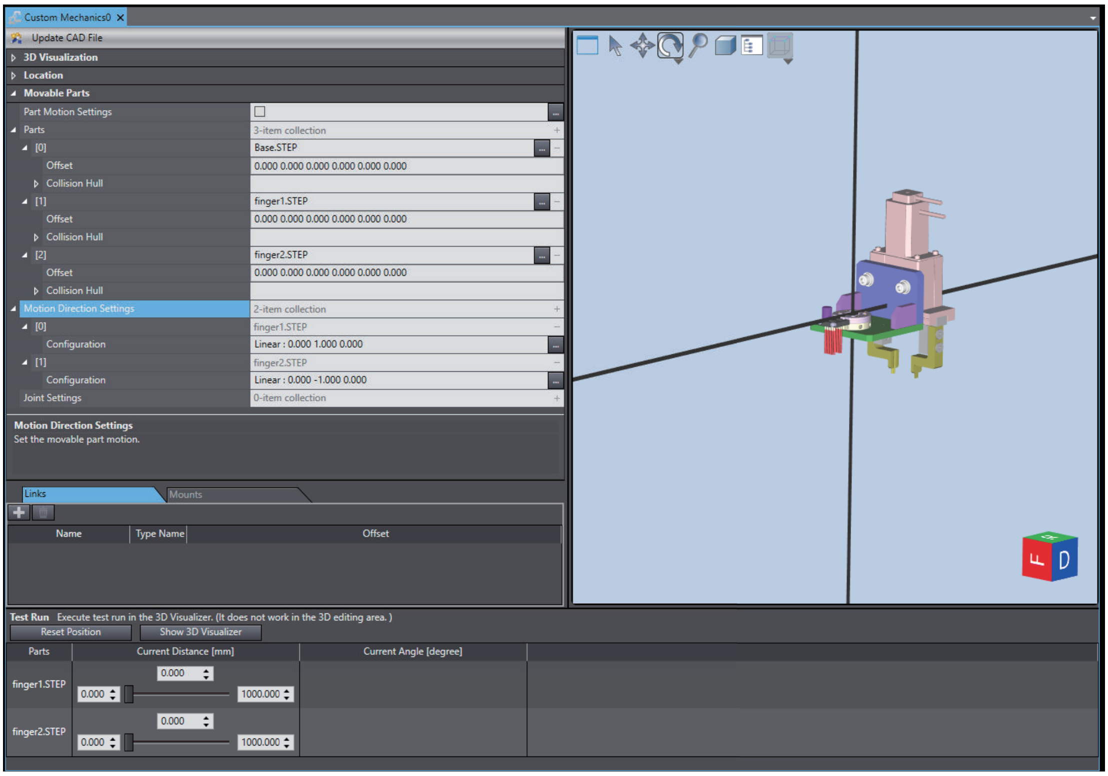

**8** The **Motion Direction Settings** dialog box is displayed.



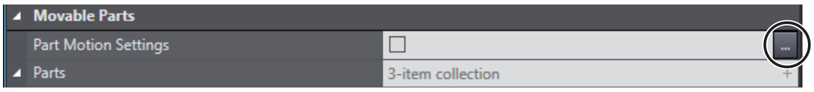

**9** Select the motion type of the movable part, configure the necessary settings, and then click the **OK** button.

The motion direction settings of the movable part are saved.

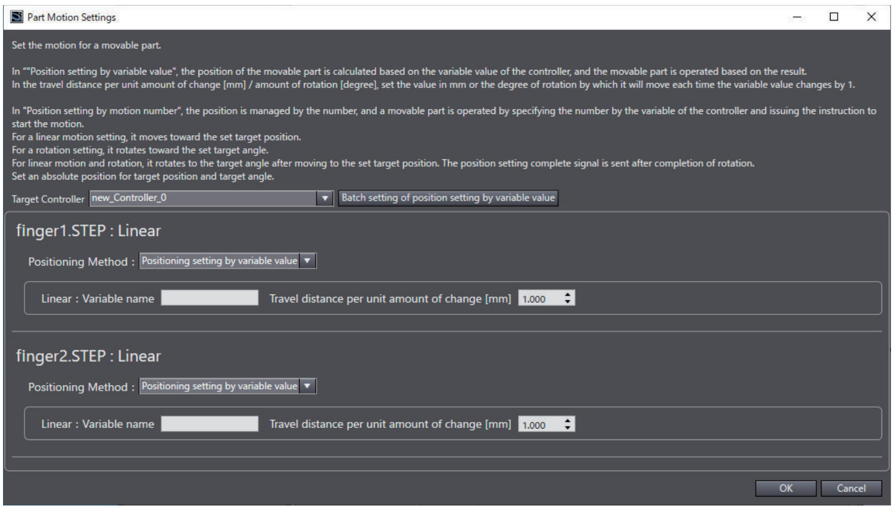
**10** Configure the motion direction settings of all movable parts in the same way.




**11** Click the button at the right of **Part Motion Settings**.



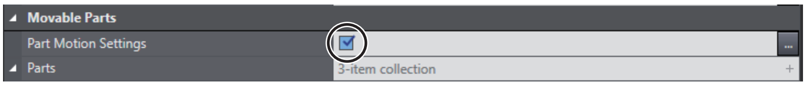

**12** The **Part Motion Settings** dialog box is displayed.




Refer to *4-4-11 Configuring the Motion Settings of the Custom Mechanics* on page 4-66 for details on the settings.

**13** Enter the setting values and click the **OK** button.

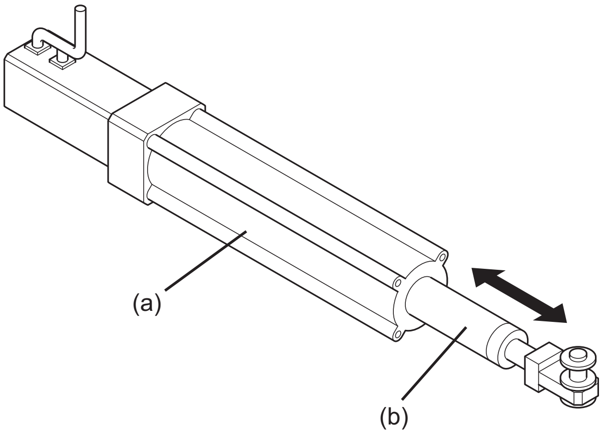
**14** Select the check box of **Part Motion Settings**.




This completes the procedure of creating the electric chuck (2-finger type) and configuring the motion settings.

## Electric Cylinder

Create the following electric cylinder as an example.




Movable parts:

| | Movable parts name (CAD Data parts) | Motion type |
|---|---|---|
| (a) | Cylinder | No motion |
| (b) | Piston rod | Linear |

### ● Creating Procedure

The creating procedure is the same as the electric chuck (2-finger type). Refer to *Electric Chuck (2-finger Type)* on page 4-76 for the creating procedure.

# 4-5 Creating 3D Shape Data for Parallel Link Model

To create a parallel link model, select a type of component from the types provided. You can select a type of component from the following types that are supported by NJ-series NJ Robotics CPU Unit.

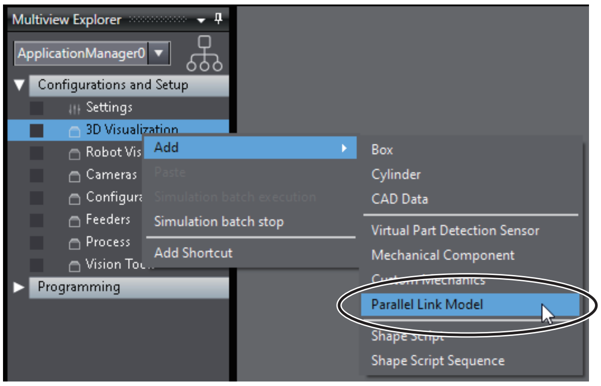
| Type | Description |
|---|---|
| Delta3 | A type of parallel link model that consists of three arms. |
| Delta3R | A parallel link model that is a combination of a Delta3 and a wrist that works as a rotation axis. |
| Delta2 | A type of parallel link model that consists of two arms. |

You can replace created 3D shape data for parallel link models by importing CAD Data.
Refer to the *NJ-series NJ Robotics CPU Unit User's Manual (Cat. No. W539)* for details on the components supported by the NJ-series NJ Robotics CPU Unit.

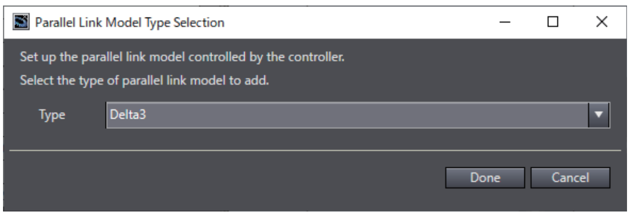
## 4-5-1 Parallel Link Model Addition Procedure

Use the following procedure to add a parallel link model.

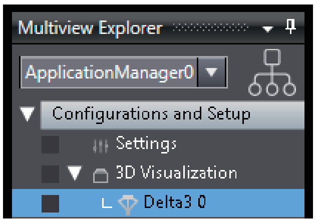
**1** Right-click **3D Visualization** under **Configurations and Setup** and select **Add** – **Parallel Link Model** from the menu.




The **Parallel Link Model Type Selection** dialog box is displayed.




**2** Select the type of the parallel link model to use from **Type**, and then click the **Done** button.
The selected component is registered and displayed under **3D Visualization**.

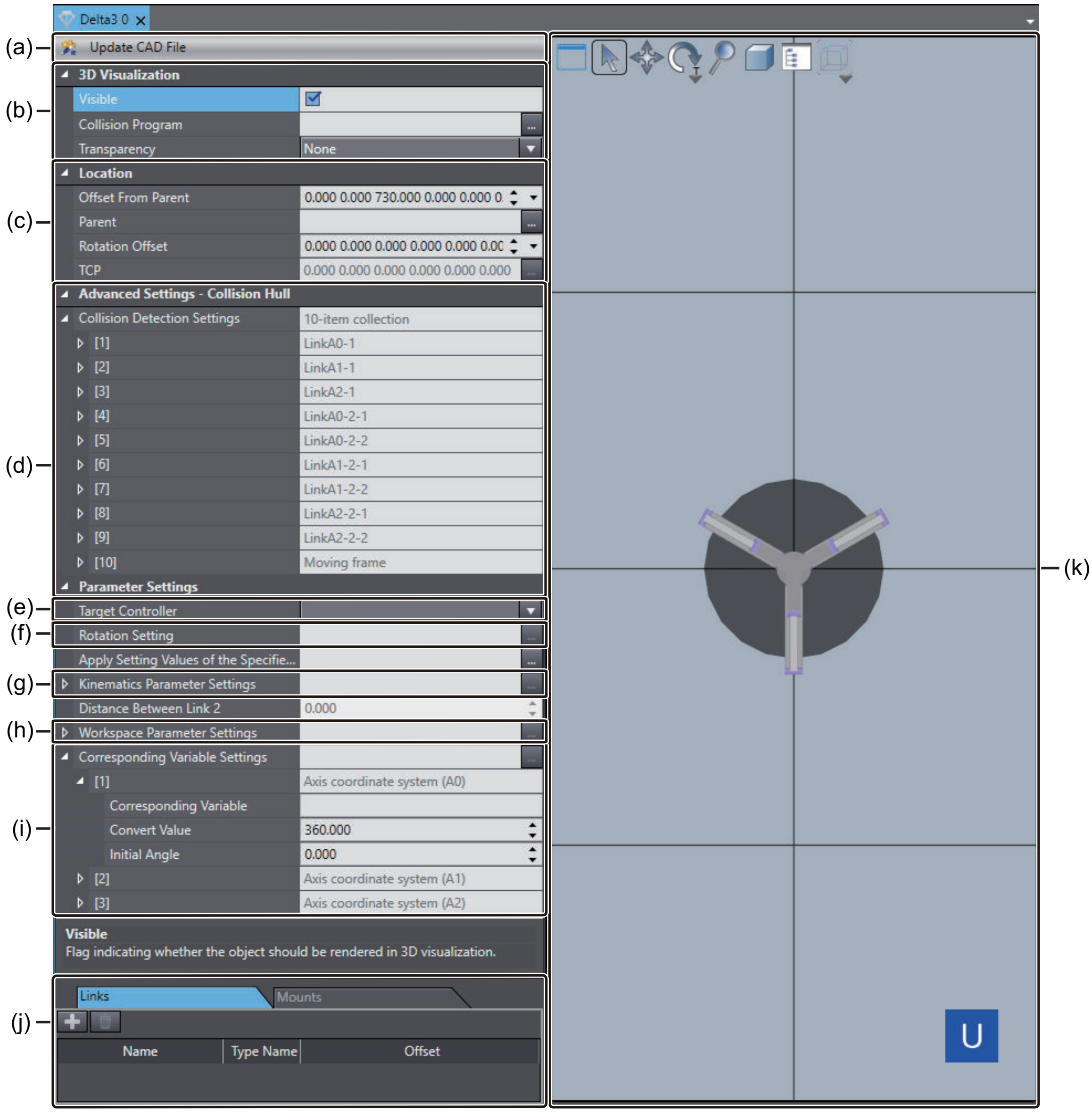



## 4-5-2 Parallel Link Model Settings

Double-clicking the parallel link model that you added displays the setup tab page for the parallel link model.

This section describes the settings common to all types of parallel link model and the settings specific to each type.

### Parallel Link Model Common Settings



| | Item | | Description | Set value | Initial value |
|---|---|---|---|---|---|
| (a) | Update CAD File | | Import the CAD Data to use as 3D shape data.<br>Refer to *4-5-3 Importing CAD Data* on page 4-91 for details. | --- | --- |
| (b) | 3D Visualization | Visible | Set whether to make the 3D shape data for the parallel link model visible on the 3D Visualizer. Select this check box to make it visible on the 3D Visualizer. | Checked or unchecked | Checked |
| | | Collision Program | Specify the C# program to execute when a collision of the parallel link model is detected. Click the button at the right, and then select the C# program from the list. You can write a C# program that outputs the time of collision, the target of collision, etc. to get information on the collision that occurred. | C# program name | None |
| | | Transparency | Set the transparency of the 3D shape data. | None, Low, Medium, or High | None |
| (c) | Location | Offset From Parent | Set the position and pose of the 3D shape data for the parallel link model. When the parent is not selected, set the coordinate in the world coordinate system of the 3D Visualizer. When the parent is selected, set the coordinate in the local coordinate system of the parent 3D shape data. The location elements are X, Y, Z, y (yaw), p (pitch), and r (roll), from left to right. | X, Y, Z: For each, -1,000,000.000 to 1,000,000.000<br>y, r: For each, -180.000 to 180.000<br>p: 0.000 to 180.000 | Other than Z: 0.000<br>Z: 730.000 |
| | | Parent | Select the 3D shape data to be the parent of the 3D shape data for the parallel link model. | Name of 3D shape data | No selection |
| | | Rotation Offset | Set the rotation offset from the origin of the local coordinate system for the 3D shape data. The position of the rotation axis will change based on this offset. The location elements are X, Y, Z, y (yaw), p (pitch), and r (roll), from left to right. | X, Y, Z: For each, -1,000,000.000 to 1,000,000.000<br>y, r: For each, -180.000 to 180.000<br>p: 0.000 to 180.000 | 0.000 for all |

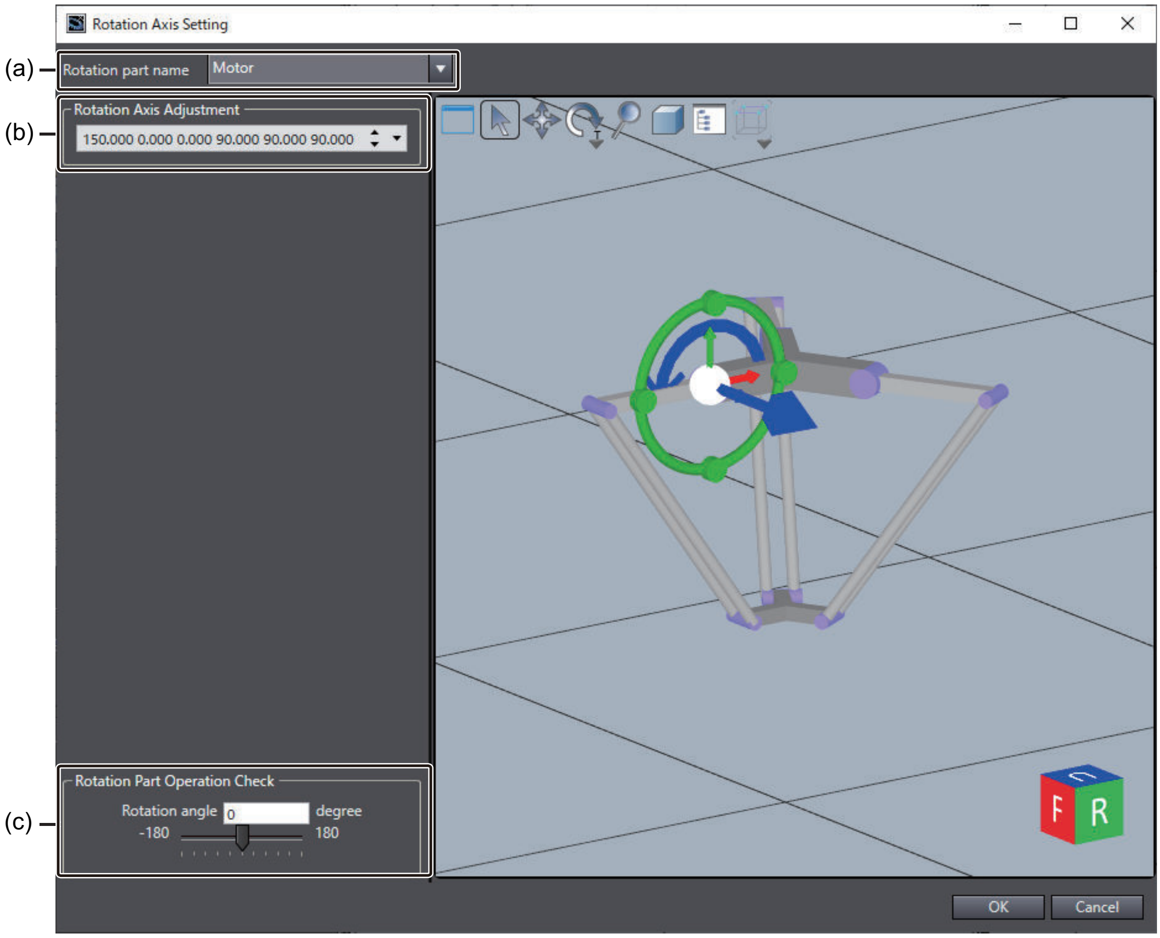
| | Item | | Description | Set value | Initial value |
|---|---|---|---|---|---|
| | | TCP | TCP is a set of coordinates of the point indicating the position where tools such as a robot hand are mounted on the 3D shape data of a parallel link model. This setting is available when the 3D shape data is an imported CAD Data. Click the button at the right, and then configure the coordinates. Refer to *TCP Settings* on page 4-88 for details on how to configure the settings.<br>Set the coordinates in the local coordinate system of the moving frame of this parallel link model. The location elements are X, Y, Z, y (yaw), p (pitch), and r (roll), from left to right. | X, Y, Z: For each, -1,000,000.000 to 1,000,000.000<br>y, r: For each, -180.000 to 180.000<br>p: 0.000 to 180.000 | Moving frame coordinates |
| (d) | Advanced Settings – Collision Hull | Visible | Set whether to show collision hulls. | Checked or unchecked | Unchecked |
| (e) | Target Controller | | Select the Controller that contains IEC programs to operate the parallel link model. You can select only the Controllers that can execute robotics instructions. | • NJ501-R□□□<br>• NJ501-4□□□ (Unit version 1.08 or later) | --- |
| (f) | Rotation Setting | | Set the center of the rotation axis when you use an imported CAD Data as the 3D shape data. Click the button at the right, and then configure the rotation axis setting. Refer to *Rotation Axis Setting* on page 4-86 for details on how to configure the setting. | --- | --- |
| (g) | Kinematics Parameter Settings | | Set the kinematics parameters. With the Target Controller set, click the button at the right, and then configure the parameter settings. Refer to *Kinematics Parameter Settings* on page 4-87 for details on how to configure the setting. | --- | --- |
| (h) | Workspace Parameter Settings | | Set the workspace parameters. With the Target Controller set, click the button at the right, and then configure the parameter settings. Refer to *Workspace Parameter Settings* on page 4-88 for details on how to configure the setting. | --- | --- |

| | Item | | Description | Set value | Initial value |
|---|---|---|---|---|---|
| (i) | Corresponding Variable Settings | | Set the variable corresponding to each axis for each axis coordinate system.<br>For each type, set the same number of variables as the number of axes below.<br>Delta3: Three axes<br>Delta3R: Four axes<br>Delta2: Two axes<br>With the Target Controller set, click the button at the right, and then select an axes group variable, which allows you to set all axis variables at once. Refer to *Corresponding Variable Settings* on page 4-88 for details on how to configure the setting. | --- | --- |
| | | Corresponding Variable | Set the variable corresponding to each axis. Enter the name of the corresponding axis variable directly. | --- | --- |
| | | Convert Value | When the unit of the axis variable set in **Corresponding Variable** is other than degree, set a convert value to convert the unit into degree. The unit is [unit of variable/rev]. | 0.000 to 4294967295.000 | 360.000 |
| | | Initial Angle | The initial angle of each axis. Set this item when link 1 is not in parallel with the XY plane. During operation check using the simulator or actual equipment, the angle of each axis is set to the sum of the axis variable value and the initial angle.<br>This item is not available for Axis coordinate system (A3) of Delta3R. | -180.000 to 180.000 | 0.000 |
| (j) | Mount point or link point setup area | | Use this area to set mount points or link points for the 3D shape data for the parallel link model. Refer to *5-4 Positioning with a Mount Point or a Link Point* on page 5-25 for information on how to set the mount points and link points. | --- | --- |
| (k) | 3D editing area | | Use this area to display the position, pose, and mount points or link points of the 3D shape data for the parallel link model. Refer to *Section 5 Working with the 3D Visualizer and 3D Editing Area* on page 5-1 for details on how to make items visible and work with them in the 3D editing area. | --- | --- |

### ● Rotation Axis Setting

Set the centers of rotation parts of a parallel link model.

Clicking the button at the right of **Rotation Setting** displays the **Rotation Axis Setting** dialog box.

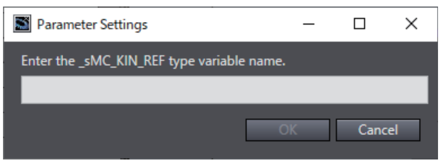



| | Item | Description | Set value | Initial value |
|---|---|---|---|---|
| (a) | Rotation part name | Displays a rotation parts name. You can select *Motor* for Delta2 and Delta3, and *Motor* or *Rotate Axis* for Delta3R. | Motor<br>Rotate Axis | Motor |
| (b) | Rotation Axis Adjustment | Adjust the position of the rotation axis. The location elements are X, Y, Z, y (yaw), p (pitch), and r (roll), from left to right. | X, Y, Z: For each, -1,000,000.000 to 1,000,000.000<br>y, r: For each, -180.000 to 180.000<br>p: 0.000 to 180.00 | 0.000 for all |
| (c) | Rotation Part Operation Check | Check the operation of the rotation part. The rotation axis rotates according to the slider position. | -180 to 180 | 0 |

## ● Kinematics Parameter Settings

Set the kinematics parameters.

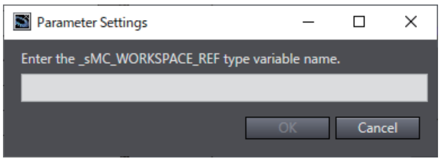
Clicking the button at the right of **Kinematics Parameter Settings** displays the **Parameter Settings** dialog box.



Enter the name of an *_sMC_KIN_REF* variable that contains setting values and click the **OK** button to apply the kinematics parameters contained in the members *KinParam* and *ExpansionParam* of the *_sMC_KIN_REF* variable.
Refer to the *NJ-series NJ Robotics CPU Unit User's Manual (Cat. No. W539)* for details on kinematics parameters.

### ● Workspace Parameter Settings

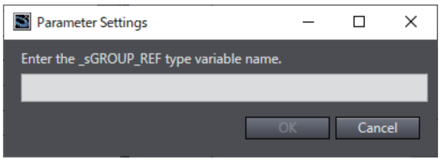
Set the workspace parameters.
Clicking the button at the right of **Workspace Parameter Settings** displays the **Parameter Settings** dialog box.




Enter the name of an *sMC_WORKSPACE_REF* variable that contains setting values and click the **OK** button to apply the workspace parameters contained in the member *WorkspaceParam* of the *sMC_WORKSPACE_REF* variable.
Refer to the *NJ-series NJ Robotics CPU Unit User's Manual (Cat. No. W539)* for details on workspace parameters.

### ● Corresponding Variable Settings

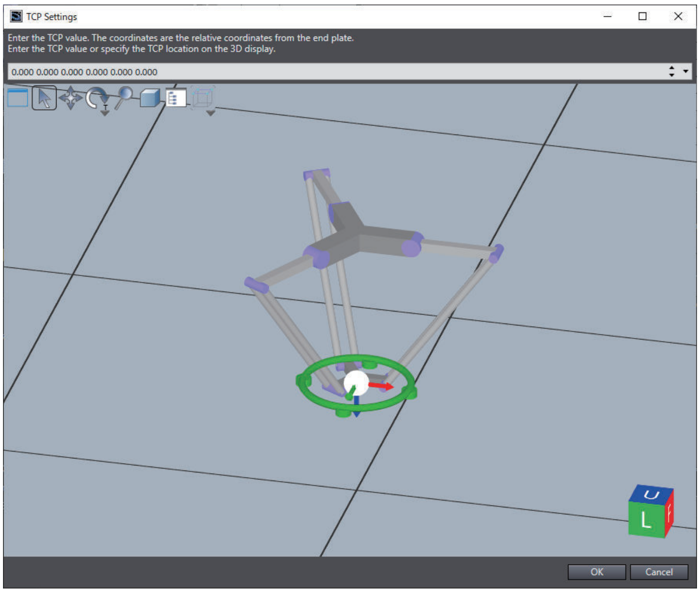
Set all corresponding axis variables at once.
Clicking the button at the right of **Corresponding Variable Settings** displays the **Parameter Settings** dialog box.




Enter the name of a corresponding axes group variable and then click the **OK** button to set all axis variables at once.
Refer to the *NJ-series NJ Robotics CPU Unit User's Manual (Cat. No. W539)* for details on axis variables and axes group variables.

### ● TCP Settings

Set the coordinates of TCP.
Clicking the button at the right of **TCP Settings** displays the **TCP Settings** dialog box.

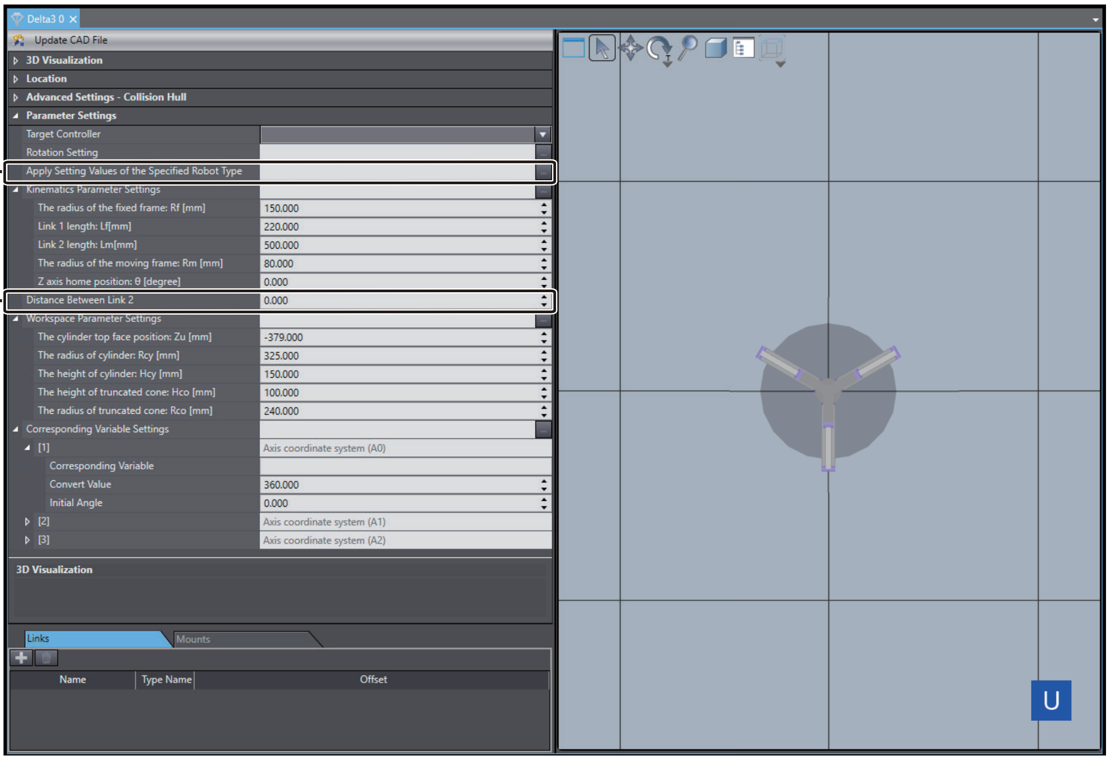



Enter the coordinate values or drag the TCP in the dialog box to specify the position. The TCP setting values are X, Y, Z, y (yaw), p (pitch), and r (roll), from left to right.

## Delta3 Settings

The parameter settings specific to Delta3 are as follows.

(a)
(b)



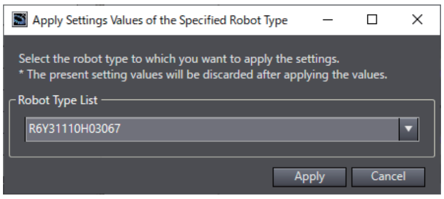
| | Item | Description | Set value | Initial value |
|---|---|---|---|---|
| (a) | Apply Setting Values of the Specified Robot Type | Set the kinematics parameters and workspace parameters automatically. Click the button at the right, and then configure the setting. Refer to *Apply Setting Values of the Specified Robot Type* on page 4-90 for details on how to configure the setting. | --- | --- |
| (b) | Distance Between Link 2 | The distance between the centers of the balls of the ball joints at both ends of the part that connects link 2. | 0.000 to 1000.000 | 0.000 |

### ● Apply Setting Values of the Specified Robot Type

Select the robot type.

Clicking the button at the right of **Apply Setting Values of the Specified Robot Type** displays the **Apply Settings Values of the Specified Robot Type** dialog box.

Select the robot type whose setting values you want to apply, and then click the **Apply** button.

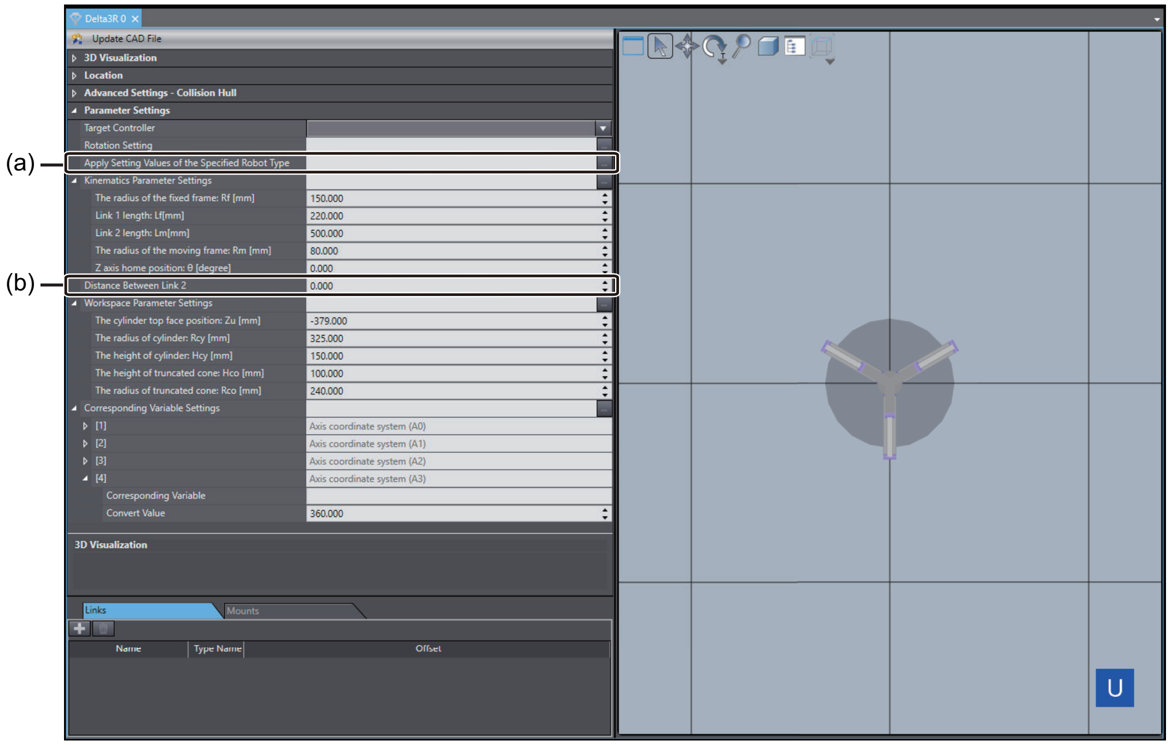



The items displayed in Robot Type List are as follows.

| Robot type | Description |
|---|---|
| R6Y31110H03067 | Mid-size parallel link robot (Operating range: φ1,100 mm, weight capacity: 3 kg, high inertia type with θ axis) |
| R6Y31110L03067 | Mid-size parallel link robot (Operating range: φ1,100 mm, weight capacity: 3 kg, low inertia type with θ axis) |
| R6Y30110S03067 | Mid-size parallel link robot (Operating range: φ1,100 mm, weight capacity: 3 kg) |
| R6Y31065H02067 | Small-size parallel link robot (Operating range: φ650 mm, weight capacity: 2 kg, high inertia type with θ axis) |
| R6Y31065L02067 | Small-size parallel link robot (Operating range: φ650 mm, weight capacity: 2 kg, low inertia type with θ axis) |
| R6Y30065S02067 | Small-size parallel link robot (Operating range: φ650 mm, weight capacity: 2 kg) |

## Delta3R Settings

The parameter settings specific to Delta3R are as follows.




| | Item | Description | Set value | Initial value |
|---|---|---|---|---|
| (a) | Apply Setting Values of the Specified Robot Type | Set the kinematics parameters and workspace parameters automatically. Click the button at the right, and then configure the setting. Refer to *Apply Setting Values of the Specified Robot Type* on page 4-90 for Delta3 for details on how to configure the settings. | --- | --- |
| (b) | Distance Between Link 2 | The distance between the centers of the balls of the ball joints at both ends of the part that connects link 2. | 0.000 to 1000.000 | 0.000 |

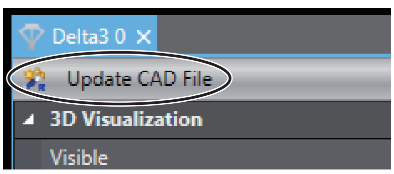
## 4-5-3 Importing CAD Data

To use a CAD Data as the 3D shape data for a parallel link model, import the CAD Data file into a registered parallel link model.

### Additional Information

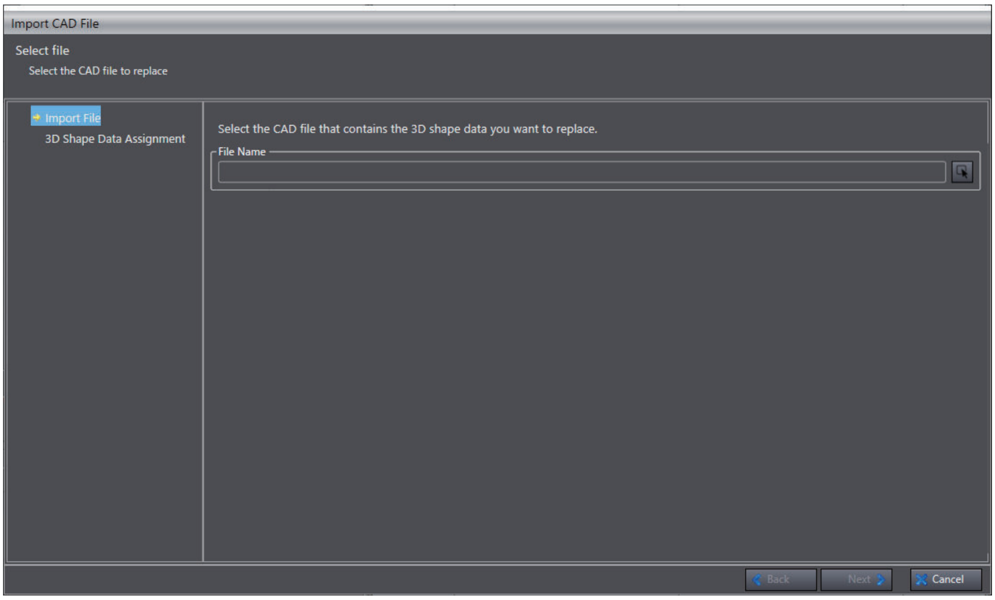
You can optionally set the mesh coarseness of CAD data at the time of import. Refer to the *Sysmac Studio Robot Integrated System Building Function with Robot Integrated CPU Unit Operation Manual (Cat. No. W595)* for details on the option settings.

**1** In the setup tab page for the target parallel link model, click the **Update CAD File** button.



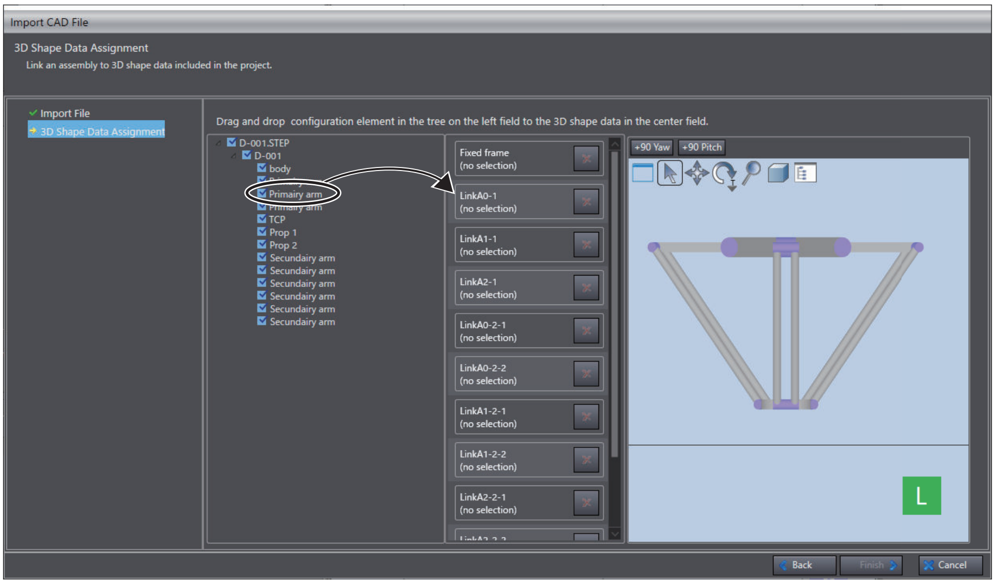
The wizard starts and the **Select file** page is displayed.

**2** Select the CAD Data file to import with the file selection button and then click the **Next** button.




The **3D Shape Data Assignment** page is displayed.

**3** Drag configuration elements of the CAD Data to target 3D shape data items.




The CAD Data are assigned to the target 3D shape data.
The 3D shape data are assigned to each link part of a parallel link model.

| Link part name | Delta3 | Delta3R | Delta2 |
|---|---|---|---|
| Fixed frame | ○ | ○ | ○ |
| LinkA0-1 | ○ | ○ | ○ |
| LinkA1-1 | ○ | ○ | ○ |
| LinkA2-1 | ○ | ○ | --- |
| LinkA0-2-1 | ○ | ○ | ○ |
| LinkA0-2-2 | ○ | ○ | ○ |

| Link part name | Delta3 | Delta3R | Delta2 |
|---|---|---|---|
| LinkA1-2-1 | ○ | ○ | ○ |
| LinkA1-2-2 | ○ | ○ | ○ |
| LinkA2-2-1 | ○ | ○ | --- |
| LinkA2-2-2 | ○ | ○ | --- |
| Moving frame | ○ | ○ | ○ |
| Rotation Axis Link A3-1 | --- | ○ | --- |
| Rotation Axis Link A3-2 | --- | ○ | --- |
| Rotation Axis End Effector | --- | ○ | --- |

Refer to the *NJ-series NJ Robotics CPU Unit User's Manual (Cat. No. W539)* for details on the link parts.

Assign CAD Data to all 3D shape data, and then click the **Finish** button.

### Additional Information

You can omit the assignment of CAD Data to the following link parts.

- Fixed frame
- Link2-A-2
- Link2-B-2
- Link2-C-2 (Delta3 and Delta3R only)

When you omit the assignment of CAD Data to the **Fixed frame**, the base is not displayed in the 3D Visualizer.

# 4-6 Adding 3D Shape Data for the Part

Add 3D shape data for the part. You can use CAD data, boxes, and cylinders as 3D shape data for the part.

## 4-6-1 Importing CAD Data

Load the CAD data used as 3D shape data for the part from a 3D CAD data file.

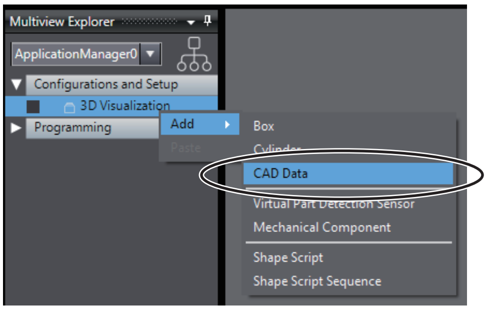
### CAD Data Import Procedure

Import CAD data to Application Manager as 3D shape data for the part. Use the following import procedure.

**Additional Information**

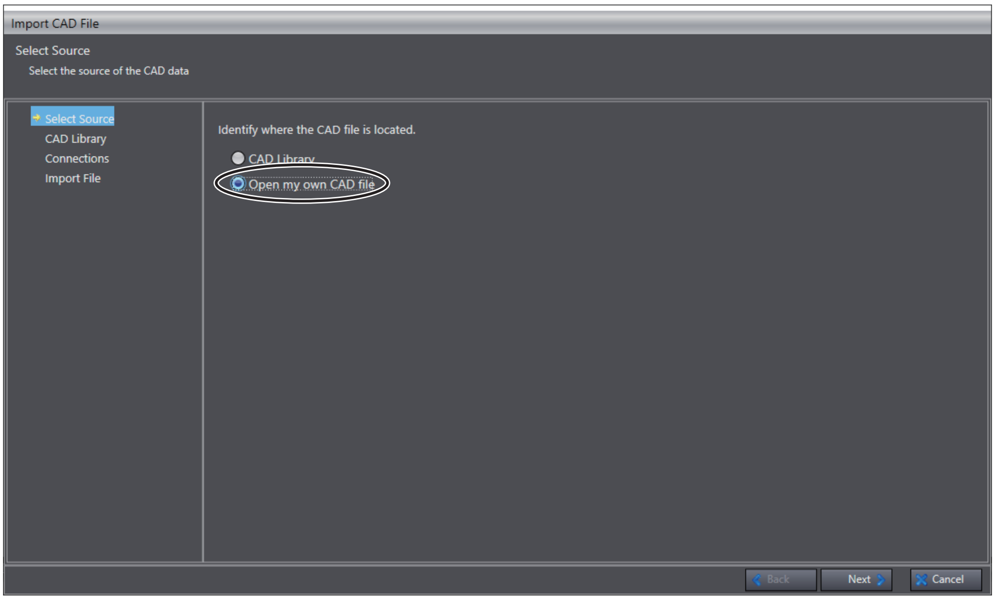
You can optionally set the mesh coarseness of CAD data at the time of import. Refer to the *Sysmac Studio Robot Integrated System Building Function with Robot Integrated CPU Unit Operation Manual (Cat. No. W595)* for details on the option settings.

**1** Right-click **3D Visualization** under **Configurations and Setup** and select **Add** - **CAD Data** from the menu.



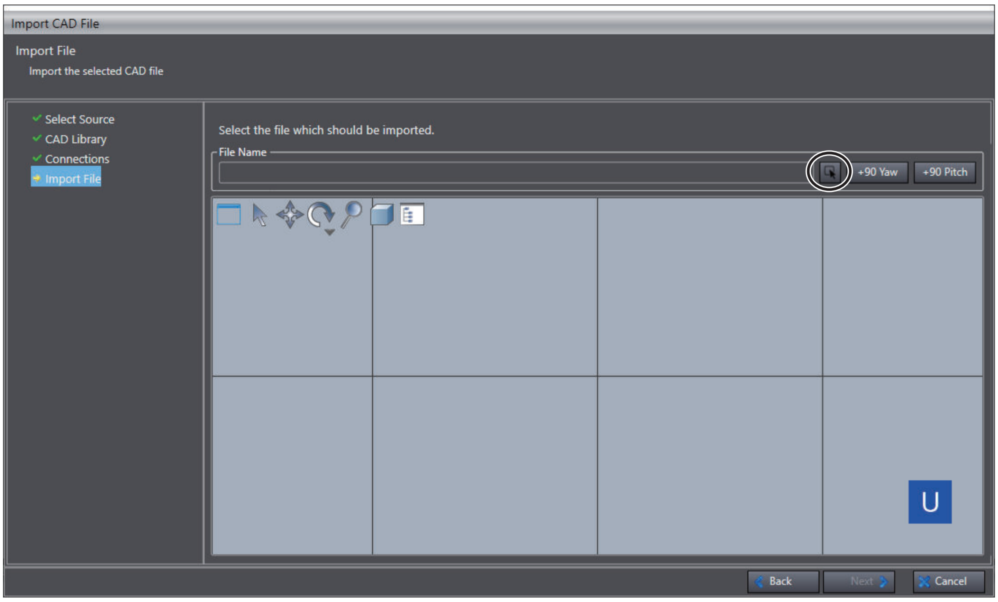

The **Import CAD File** wizard starts, and then the **Select Source** page is displayed.

**2** On the **Select Source** page, select **Open my own CAD file** and then click the **Next** button.



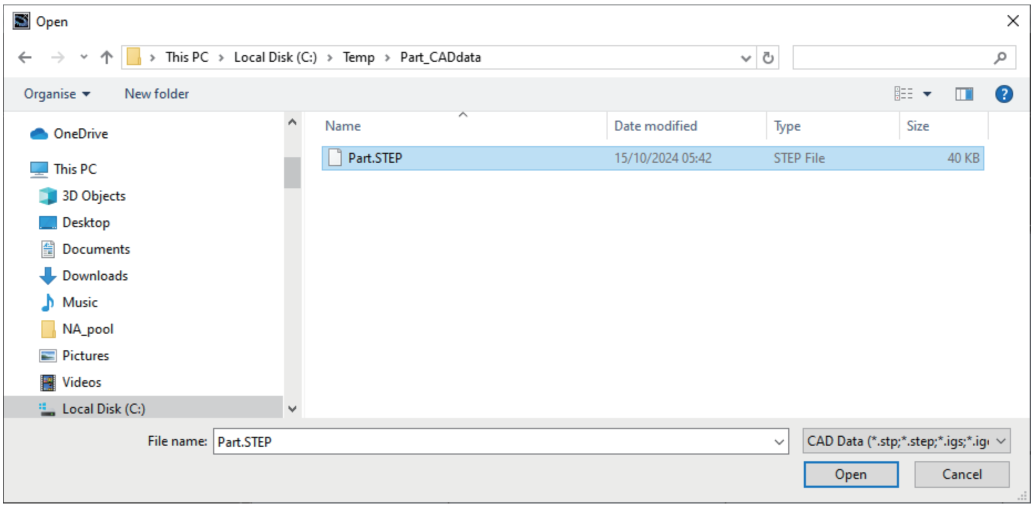

The **Import File** page is displayed.

**3** On the **Import File** page, click the **Import File** button.




The **Open** dialog box is displayed.

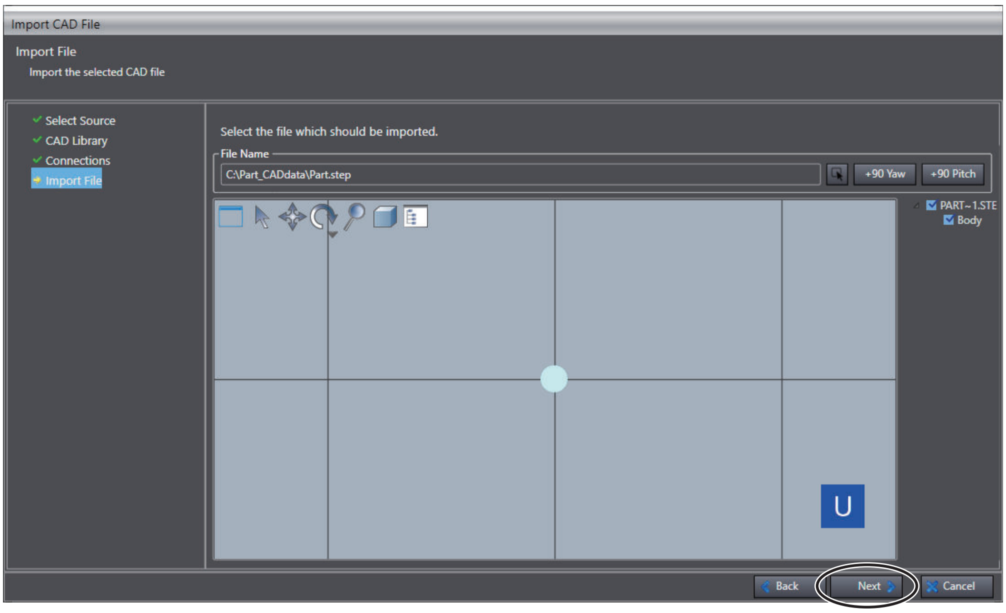
**4** Select a CAD file with a .stp, .step, .igs, .iges, .usd, .usda, .usdc, or .usdz file name extension, and then click the **Open** button.



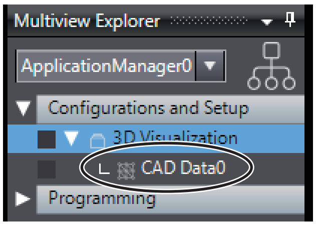

The CAD file is imported.
If necessary, click the **+90 Yaw** or **+90 Pitch** button to change the orientation of the CAD data to import.

| Button | Description |
|---|---|
| +90 Yaw | Rotates the CAD data +90° around the Z axis of the CAD data coordinate system. At this time, the CAD data coordinate system is rotated together. |
| +90 Pitch | Rotates the CAD data +90° around the Y axis of the CAD data coordinate system. At this time, the CAD data coordinate system is rotated together. |

**5** Click the **Next** button.

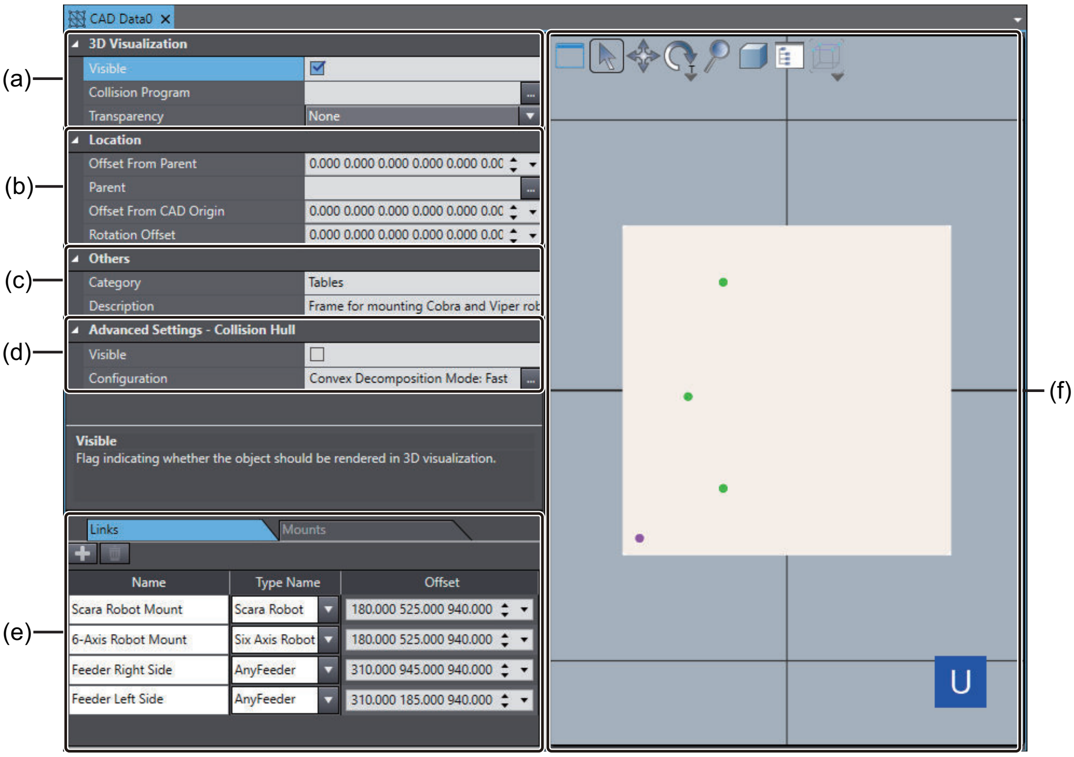



The imported CAD data is added and displayed under **3D Visualization** in the Multiview Explorer.




## CAD Data Settings

Click the imported CAD data to display the CAD DATA setup tab page.
Configure the placement and mount point/link point settings for the CAD data.
The CAD data settings are listed in the following table.




| | Item | | Description | Set value | Initial value |
|---|---|---|---|---|---|
| (a) | 3D Visualization | Visible | Set whether to make this 3D shape data visible on the 3D Visualizer. Select this check box to make it visible on the 3D Visualizer. | Checked or unchecked | Checked |
| | | Collision Program | Specify the C# program to execute when a collision of the CAD data is detected. Click the button at the right, and then select the C# program from the list. You can write a C# program that outputs the time of collision, the target of collision, etc. to get information on the collision that occurred. | C# program name | None |
| | | Transparency | Set the transparency of the 3D shape data. | None, Low, Medium, or High | None |

| | Item | | Description | Set value | Initial value |
|---|---|---|---|---|---|
| (b) | Location | Parent | Select the 3D shape data to be the parent of this 3D shape data. Click the button at the right, and then select the parent 3D shape data from the list. | Name of 3D shape data | None |
| | | Offset From Parent | Set the position and pose of this 3D shape data.<br>When the parent is not selected, set the coordinate in the world coordinate system of the 3D Visualizer. When the parent is selected, set the coordinate in the local coordinate system of the parent 3D shape data. The location elements are X, Y, Z, y (yaw), p (pitch), and r (roll), from left to right. | X, Y, Z: For each, -1,000,000.000 to 1,000,000.000<br>y, r: For each, -180.000 to 180.000<br>p: 0.000 to 180.000 | 0.000 for all |
| | | Rotation Offset | Set the rotation offset from the origin of the local coordinate system for this 3D shape data.<br>The position of the rotation axis will change based on this offset. The location elements are X, Y, Z, y (yaw), p (pitch), and r (roll), from left to right. | X, Y, Z: For each, -1,000,000.000 to 1,000,000.000<br>y, r: For each, -180.000 to 180.000<br>p: 0.000 to 180.000 | 0.000 for all |
| | | Offset From CAD Origin | Set the offset from the CAD origin of the local coordinate system for this 3D shape data. This allows you to correct the origin that is set with 3D CAD software.<br>The origin of the 3D shape data will change based on this offset. The location elements are X, Y, Z, y (yaw), p (pitch), and r (roll), from left to right. | X, Y, Z: For each, -1,000,000.000 to 1,000,000.000<br>y, r: For each, -180.000 to 180.000<br>p: 0.000 to 180.000 | 0.000 for all |
| (c) | Others | Category/ Description | Describe the explanation of this 3D shape data as required. | Text string | None |
| (d) | Advanced Settings – Collision Hull | Visible | Set whether to show collision hulls. | Checked or unchecked | Unchecked |
| | | Configuration | Configure the collision hull settings. Click the button at the right, and then configure the advanced collision hull settings. Refer to *Advanced Collision Hull Settings* on page 8-27 for details on how to configure the settings. | --- | Convex: True \| Multiple Hulls: False |
| (e) | Mount point or link point setup area | | Use this area to set mount points or link points for this 3D shape data. Refer to *5-4 Positioning with a Mount Point or a Link Point* on page 5-25 for information on how to set mount points and link points. | --- | --- |

| | Item | Description | Set value | Initial value |
|---|---|---|---|---|
| (f) | 3D editing area | Use this area to display the position, pose, and mount points or link points of the 3D shape data. Refer to *Section 5 Working with the 3D Visualizer and 3D Editing Area* on page 5-1 for details on how to make items visible and work with them in the 3D editing area. | --- | --- |

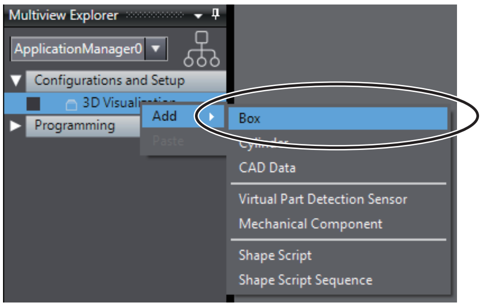
## 4-6-2 Adding a Box or a Cylinder

This section describes the procedures to add a box or a cylinder.

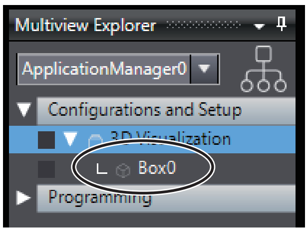
### Box or Cylinder Addition Procedure

Use the following procedure to add a box or a cylinder.

**1** Right-click **3D Visualization** under **Configurations and Setup** and select **Add** - **Box** or **Cylinder** from the menu.




A box or a cylinder is added and displayed under **3D Visualization** in the Multiview Explorer.

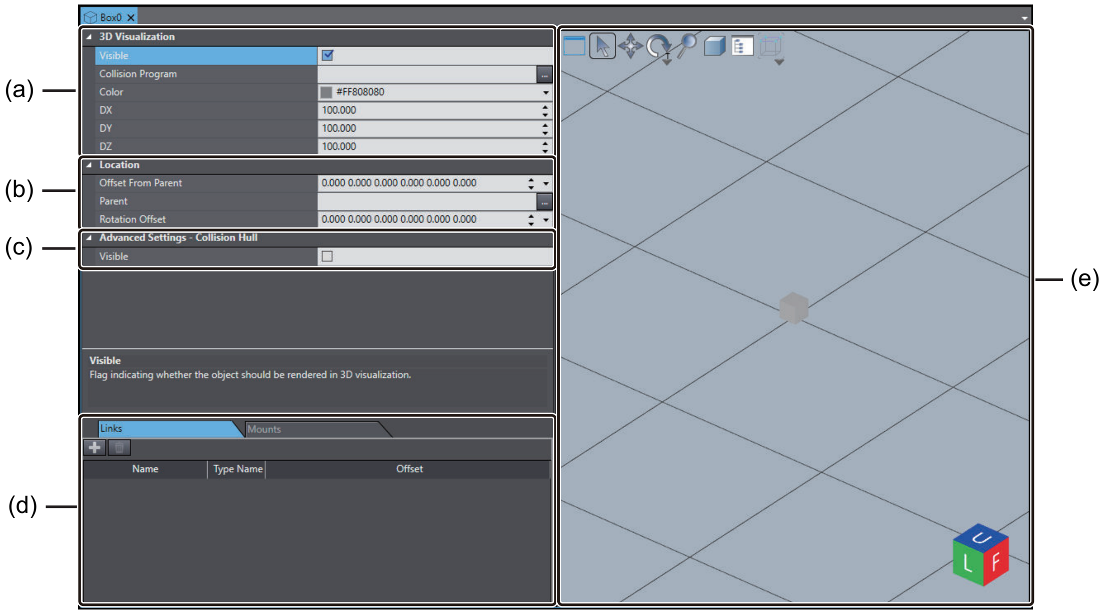



### Box Settings

Configure the size, placement, and mount point/link point settings for a box.

The origin of the 3D shape data for a box is set to the center of gravity of the bottom face of the box.

The box settings are listed in the following table.

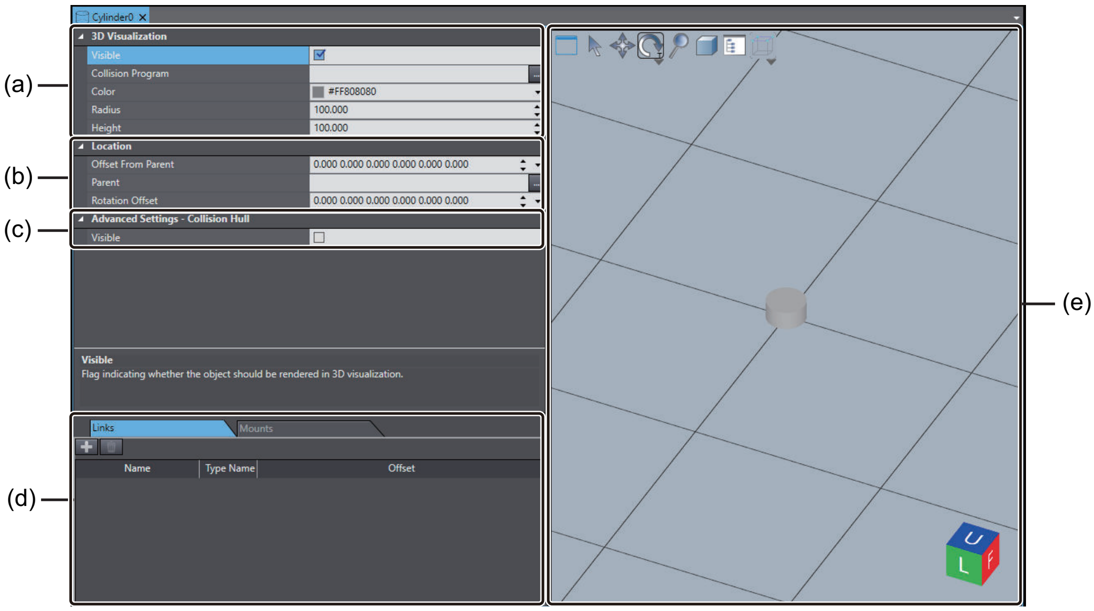



| | Item | | Description | Set value | Initial value |
|---|---|---|---|---|---|
| (a) | 3D Visu-alization | Visible | Set whether to make a box visible on the 3D Visualizer. Select this check box to make it visible on the 3D Visualizer. | Checked or un-checked | Checked |
| | | Color | Set the color of a box. | Transparency (0 to #FF) and RGB (0 to #FF, 0 to #FF, 0 to #FF) | #FF808080 |
| | | DX | Set the length of a box along the X axis. (Unit: mm) | 0 to 5,000 | 100 |
| | | DY | Set the length of a box along the Y axis. (Unit: mm) | 0 to 5,000 | 100 |
| | | DZ | Set the length of a box along the Z axis. (Unit: mm) | 0 to 5,000 | 100 |
| (b) | Location | Offset From Parent | Set the position and pose of a box.<br>When the parent is not selected, set the co-ordinate in the world coordinate system of the 3D Visualizer. When the parent is se-lected, set the coordinate in the local coordi-nate system of the parent 3D shape data.<br>The location elements are X, Y, Z, y (yaw), p (pitch), and r (roll), from left to right. | X, Y, Z: For each, -1,000,000.000 to 1,000,000.000<br>y, r: For each, -180.000 to 180.000<br>p: 0.000 to 180.000 | 0.00 for all |
| | | Parent | Select the 3D shape data to be the parent of a box.<br>Click the button at the right, and then select the parent 3D shape data from the list. | Name of 3D shape da-ta | None |
| | | Rotation Offset | Set the rotation offset from the origin of the local coordinate system for a box.<br>The position of the rotation axis will change based on this offset. The location elements are X, Y, Z, y (yaw), p (pitch), and r (roll), from left to right. | X, Y, Z: For each, -1,000,000.000 to 1,000,000.000<br>y, r: For each, -180.000 to 180.000<br>p: 0.000 to 180.000 | 0.00 for all |

| | Item | | Description | Set value | Initial value |
|---|---|---|---|---|---|
| (c) | Advanced Settings – Collision Hull | Visible | Set whether to show collision hulls. | Checked or unchecked | Unchecked |
| (d) | Mount point or link point setup area | | Use this area to set mount points or link points for the Box. Refer to *5-4 Positioning with a Mount Point or a Link Point* on page 5-25 for information on how to set mount points and link points. | --- | --- |
| (e) | 3D editing area | | Use this area to display the position, pose, and mount points or link points of a box. Refer to *Section 5 Working with the 3D Visualizer and 3D Editing Area* on page 5-1 for details on how to make items visible and work with them in the 3D editing area. | --- | --- |

## Cylinder Settings

Configure the size, placement, and mount point/link point settings for a cylinder.
The origin of the 3D shape data for a cylinder is set to the center of the bottom face of the cylinder.
The cylinder settings are listed in the following table.




| | Item | | Description | Set value | Initial value |
|---|---|---|---|---|---|
| (a) | 3D Visualization | Visible | Set whether to make a cylinder visible on the 3D Visualizer. Select this check box to make it visible on the 3D Visualizer. | Checked or unchecked | Checked |
| | | Color | Set the color of the Cylinder. | Transparency (0 to #FF) and RGB (0 to #FF, 0 to #FF, 0 to #FF) | #FF808080 |
| | | Radius | Set the radius of a cylinder. (Unit: mm) | 0 to 5,000 | 100 |
| | | Height | Set the height of a cylinder. (Unit: mm) | 0 to 5,000 | 100 |

4-6 Adding 3D Shape Data for the Part

4

4-6-2 Adding a Box or a Cylinder

| | Item | | Description | Set value | Initial value |
|---|---|---|---|---|---|
| (b) | Location | Offset From Parent | Set the position and pose of a cylinder. When the parent is not selected, set the coordinate in the world coordinate system of the 3D Visualizer. When the parent is selected, set the coordinate in the local coordinate system of the parent 3D shape data. The location elements are X, Y, Z, y (yaw), p (pitch), and r (roll), from left to right. | X, Y, Z: For each, -1,000,000.000 to 1,000,000.000<br>y, r: For each, -180.000 to 180.000<br>p: 0.000 to 180.000 | 0.00 for all |
| | | Parent | Select the 3D shape data to be the parent of a cylinder.<br>Click the button at the right, and then select the parent 3D shape data from the list. | Text string | None |
| | | Rotation Offset | Set the rotation offset from the origin of the local coordinate system for a cylinder.<br>The position of the rotation axis will change based on this offset. The location elements are X, Y, Z, y (yaw), p (pitch), and r (roll), from left to right. | X, Y, Z: For each, -1,000,000.000 to 1,000,000.000<br>y, r: For each, -180.000 to 180.000<br>p: 0.000 to 180.000 | 0 for all |
| (c) | Advanced Settings – Collision Hull | Visible | Set whether to show collision hulls. | Checked or unchecked | Unchecked |
| (d) | Mount point or link point setup area | | Use this area to set mount points or link points for a cylinder. Refer to *5-4 Positioning with a Mount Point or a Link Point* on page 5-25 for information on how to set mount points and link points. | --- | --- |
| (e) | 3D editing area | | Use this area to display the position, pose, and mount points or link points of a cylinder. Refer to *Section 5 Working with the 3D Visualizer and 3D Editing Area* on page 5-1 for details on how to make items visible and work with them in the 3D editing area. | --- | --- |

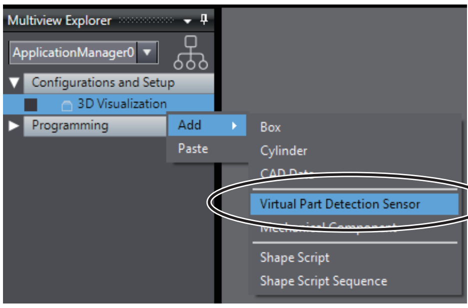
# 4-7 Adding the Part Detection Sensor

Add the *Virtual Part Detection Sensor*, which is a virtual sensor to detect the position of the part in a 3D simulation. Installing the Virtual Part Detection Sensor in a specified position enables the detection of part travel. This makes it unnecessary to create debugging programs that detect the part.

## 4-7-1 Adding the Virtual Part Detection Sensor

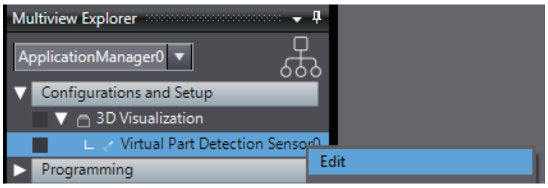
Add the Virtual Part Detection Sensor to the Application Manager.

**1** Right-click **3D Visualization** under **Configurations and Setup** and select **Add** - **Virtual Part Detection Sensor** from the menu.




The Virtual Part Detection Sensor is added.

**2** To set up the Virtual Part Detection Sensor, double-click the target Virtual Part Detection Sensor in the Multiview Explorer. Or, right-click it and select **Edit** from the menu.

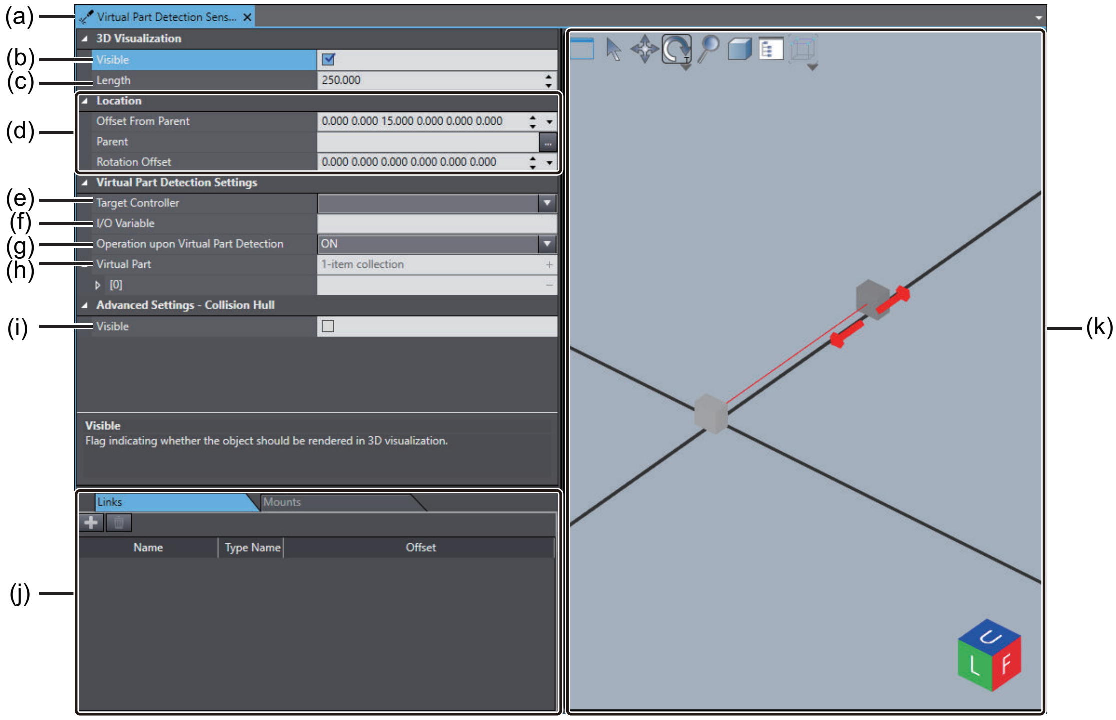



The Virtual Part Detection Sensor setup tab page is displayed.

## 4-7-2 Virtual Part Detection Sensor Settings

Set the size, placement, operation upon virtual part detection, and mount points and link points for the Virtual Part Detection Sensor.

The following describes the items that you can set on the Virtual Part Detection Sensor setup tab page.




| | Item | Description | Set value | Initial value |
|---|---|---|---|---|
| (a) | Virtual Part Detection Sensor name | The name of the Virtual Part Detection Sensor is displayed. | --- | --- |
| (b) | Visible | Set whether to make the Virtual Part Detection Sensor visible on the 3D Visualizer. | Checked or unchecked | Checked |
| (c) | Length | Set the length of the Virtual Part Detection Sensor.<br>You cannot change the thickness. | 0.001 to 8,000 mm | 250 mm |
| (d) | Location | Configure the location settings.<br>The settings are the same as those for CAD data, boxes, and cylinders. | --- | --- |

| | Item | | Description | Set value | Initial value |
|---|---|---|---|---|---|
| (e) | Virtual Part Detection Settings | Target Controller | Select the Controller to detect the part.<br>With Sysmac Studio version 1.42 or higher, you can select robot sub-devices in addition to the Controller. | Controllers registered in the project | None |
| (f) | | I/O Variable | The I/O variable represents the status that the Virtual Part Detection Sensor has detected the part.<br>Set a BOOL global variable for the target Controller.<br>If the target is a robot sub-device, set the V+Digital I/O. | Text string | None |
| (g) | | Operation upon Virtual Part Detection | Set how the I/O variable operates when the part is detected by the Virtual Part Detection Sensor.<br>If it changes to TRUE when the part is detected, set *ON upon detection*. If it changes to FALSE when the part is detected, set *OFF upon detection*. | ON upon detection or OFF upon detection | ON upon detection |
| (h) | | Virtual Part | Set the name of the target part detected by the Virtual Part Detection Sensor.<br>You can set a part name that is included in the same Application Manager. | Text string | None |
| (i) | Advanced Settings – Collision Hull | Visible | Set whether to show collision hulls. | Checked or unchecked | Unchecked |
| (j) | Mount point or link point setup area | | Set mount points and link points for the Virtual Part Detection Sensor.<br>Refer to *5-4 Positioning with a Mount Point or a Link Point* on page 5-25 for information on how to set mount points and link points. | --- | --- |
| (k) | 3D editing area | | Use this area to display the position, pose, and mount points or link points of the Virtual Part Detection Sensor. Refer to *Section 5 Working with the 3D Visualizer and 3D Editing Area* on page 5-1 for details on how to make items visible and work with them in the 3D editing area. | --- | --- |

# 4-8 3D Shape Data Placement Methods

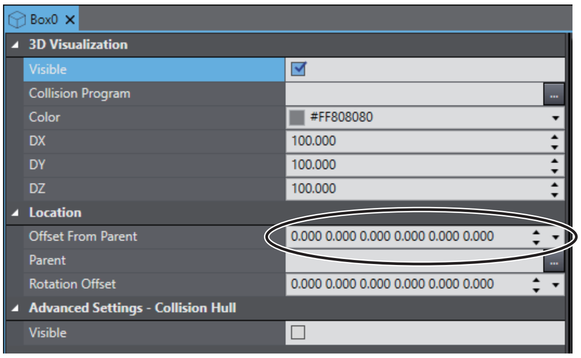
To set the placement of 3D shape data, the following methods are available.

- Entering a coordinate directly
- Dragging and dropping the data on the 3D Visualizer
- Snapping to the link point position

## 4-8-1 Entering a Coordinate Directly

**1** Display the 3D shape data setup tab page.

**2** In **Offset From Parent**, directly enter the six location elements (X, Y, Z, y, p, r) that determine the position and pose in the Cartesian coordinate space.




## 4-8-2 Dragging and Dropping the Data on the 3D Visualizer

Drag and drop the 3D shape data to the target position on the 3D Visualizer. Refer to *5-3-1 Moving 3D Shape Data* on page 5-20 for details.

## 4-8-3 Snapping to the Link Point

You can use this method when the 3D shape data comes in contact with the point specified for other 3D shape data: the case that the part is placed on the specified position on a pallet, for example.
Refer to *5-4 Positioning with a Mount Point or a Link Point* on page 5-25 for the setting and operating procedures.

# 4-9 Exporting and Importing 3D Visualization Objects

You can export the 3D Visualization objects that you created and reuse them in another project. This eliminates the need for reconfiguring the 3D Visualization object settings, resulting in reduced design and start-up time.

## 4-9-1 Types of 3D Visualization Objects That Can be Exported and Imported

3D Visualization objects that can be exported and imported depend on the version of the Application Manager. The table below shows the types of 3D Visualization objects that can be exported and imported and the versions of the Application Manager.

| Type of 3D Visualization object | Application Manager version |
|---|---|
| Default Functions | Supported by Application Manager version 5.0 or higher. |
| User Functions | Supported by Application Manager version 5.0 or higher. |
| Box | All versions |
| Cylinder | All versions |
| CAD Data | All versions |
| Virtual Part Detection Sensor | All versions |
| Mechanical Component | Other than Conveyor: All versions<br>Conveyor: Supported by Application Manager version 5.0 or higher |
| Custom Mechanics | Supported by Application Manager version 4.0 or higher. |
| Parallel Link Model | Supported by Application Manager version 4.0 or higher. |
| Shape Script | All versions |
| Shape Script Sequence | All versions |

If the Application Manager version in the import destination is lower than the version shown above, the import processing will be suspended.

## 4-9-2 Access Level Required for Export and Import Operations

You can perform export and import operations when the access level for robot system operation authority verification is Engineer. Refer to the *Sysmac Studio Robot Integrated System Building Function with Robot Integrated CPU Unit Operation Manual (Cat. No. W595)* for information on access levels for robot system operation authority verification.

## 4-9-3 Export of 3D Visualization Objects

Use the following procedure to export 3D Visualization objects.

### Export Target Data

All data of 3D Visualization objects except the reference information below is the export target.

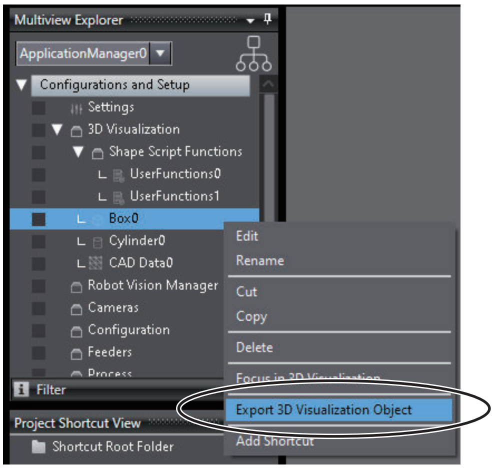
- Parent of each 3D Visualization object

- Virtual Parts in the Virtual Part Detection Sensor
- Shape Scripts in the Shape Script Sequence

## Export Procedure

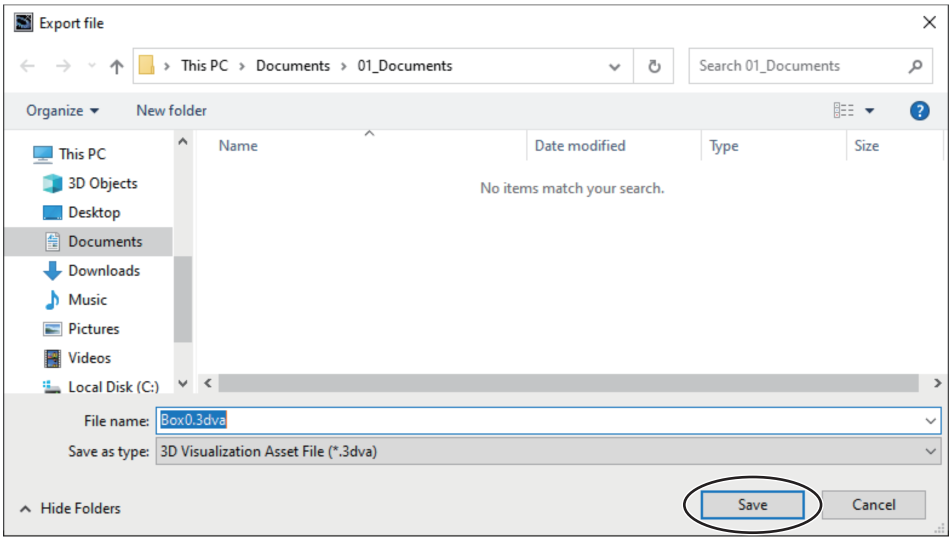
The following is an example of a box.

**1** Right-click *Box0* under **Configurations and Setup** – **3D Visualization** in the Multiview Explorer and select **Export 3D Visualization Object** from the menu.




The **Export file** dialog box is displayed.

**2** Enter the file name, and then click the **Save** button.




The export target 3D Visualization object is saved in a 3dva format file.

## 4-9-4 Batch Export of 3D Visualization Objects

Use the following procedure to export multiple 3D Visualization objects at once.

## Export Target Data

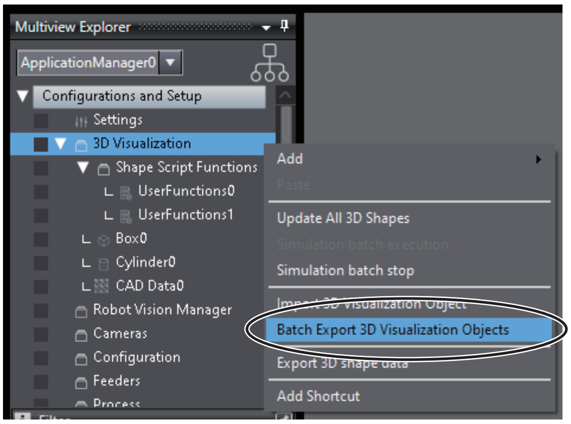
All data of 3D Visualization objects is the export target. However, if you do not select together the following data, which is referenced by each 3D Visualization object, the associated reference information is not the export target.

- Parent of each 3D Visualization object
- Virtual Parts in the Virtual Part Detection Sensor
- Shape Scripts in the Shape Script Sequence

If reference information is not the export target, the set values will be empty after the 3D Visualization object is imported.

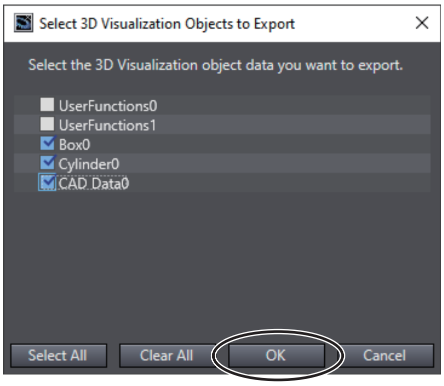
## Batch Export Procedure

**1** Right-click **3D Visualization** under **Configurations and Setup** and select **Batch Export 3D Visualization Objects** from the menu.



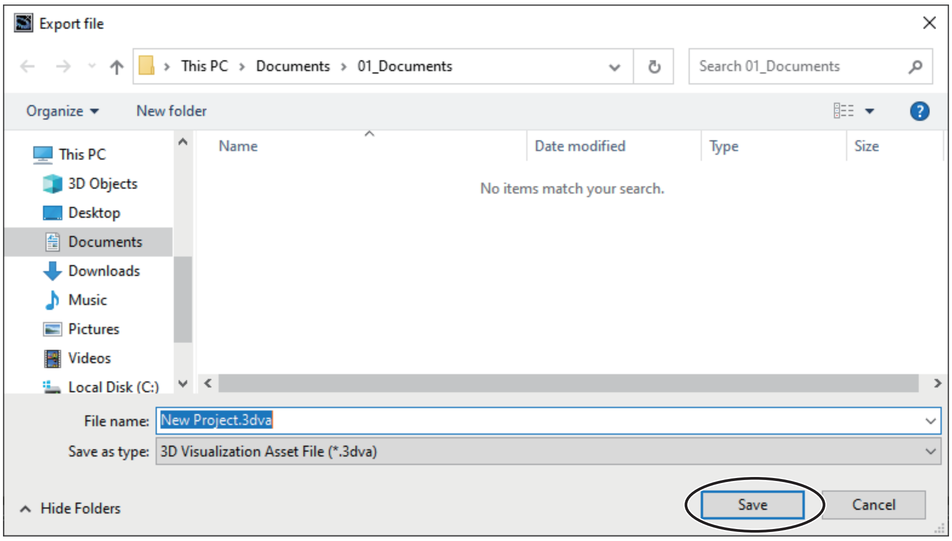

The **Select 3D Visualization Objects to Export** dialog box is displayed.

**2** Select the check boxes for the 3D Visualization objects to export, and then click the **OK** button.




The **Export file** dialog box is displayed.

**3** Enter the file name, and then click the **Save** button.

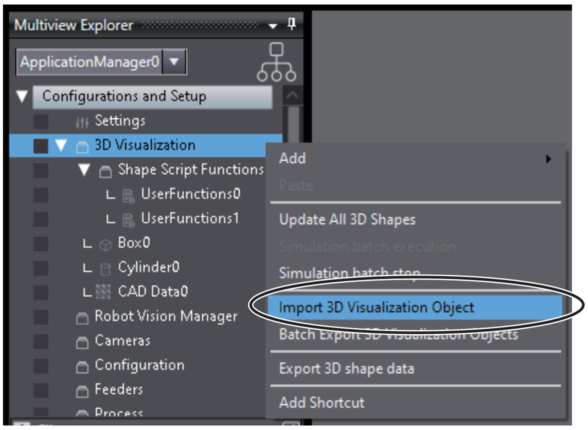



The export target 3D Visualization object is saved in a 3dva format file.

## 4-9-5 Import Procedure for 3D Visualization Objects

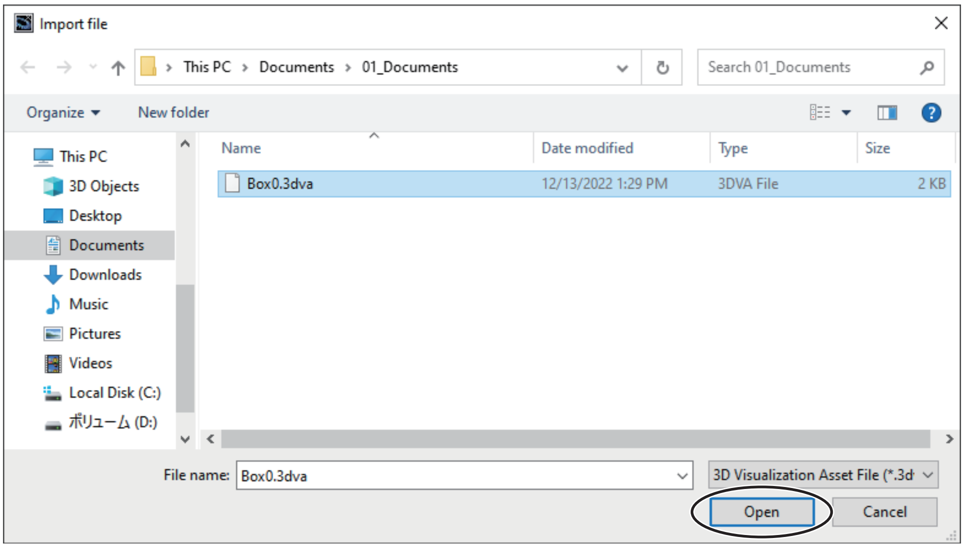
Use the following procedure to import 3D Visualization objects.
The following is an example of a box.

**1** Right-click **3D Visualization** under **Configurations and Setup** in the Multiview Explorer and select **Import 3D Visualization Object** from the menu.

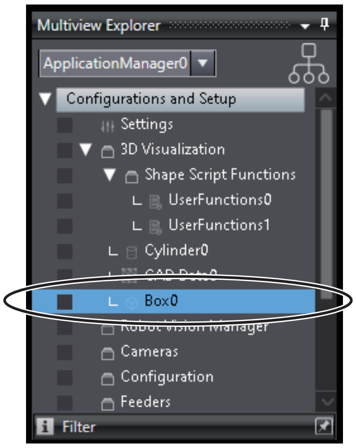



The **Import file** dialog box is displayed.

**2** Select the .3dva file to import, and then click the **Open** button.




*Box0* is added and displayed under **Configurations and Setup** – **3D Visualization** in the Multiview Explorer.




If there is a 3D Visualization object with the same name in the import destination, the source 3D Visualization object will be added with the name *[3D Visualization object name]_1*.

### Precautions for Correct Use

If reference information is not the export target, the set values will be empty after the 3D Visualization object is imported.

### Additional Information

If you import a .3dva file generated in batch export, all 3D Visualization objects contained in the .3dva file will be added.

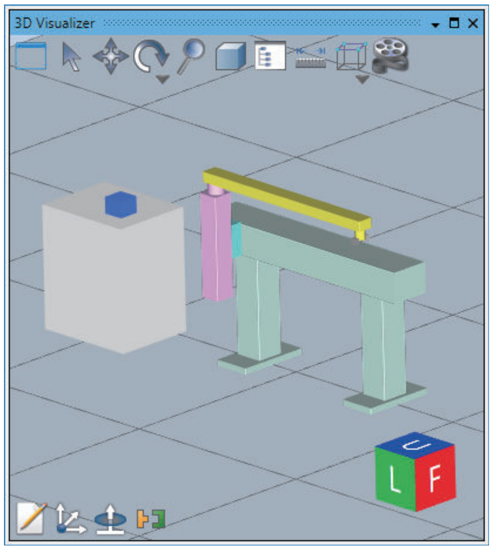
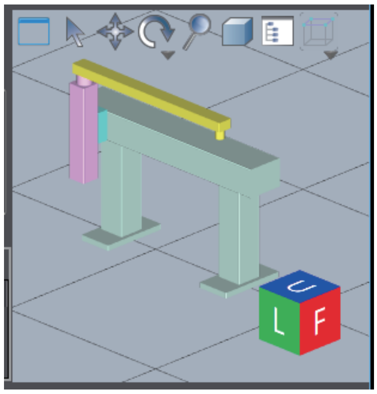
# 5

# Working with the 3D Visualizer and 3D Editing Area

This section describes the displayed items and operation methods in the 3D Visualizer and 3D editing area.

# 5-1 Displaying the 3D Visualizer and 3D Editing Area

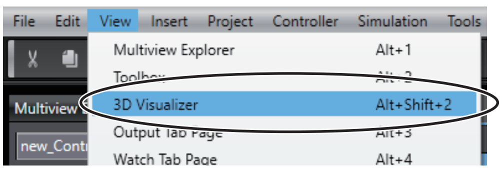
The 3D Visualizer and 3D editing area both display the 3D shape data that you added.
However, the 3D Visualizer and 3D editing area are different as described below.

| | Description | |
|---|---|---|
| 3D Visualizer | All the 3D shape data that is registered in a project is displayed. You can check the operations of a mechanical component, part, and Virtual Part Detection Sensor during the execution of a 3D simulation. |  |
| 3D editing area | Use the 3D editing area when you set the position and pose of the target 3D shape data. It is displayed as part of the setup tab page for each 3D shape data. |  |

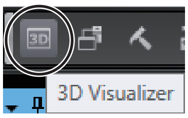
## 5-1-1 Displaying the 3D Visualizer

To display the 3D Visualizer, use the following procedure.

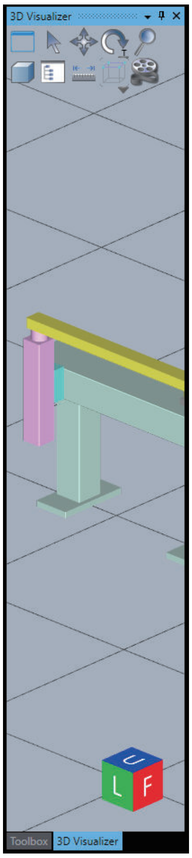
**1** Select **3D Visualizer** from the **View** menu.




Or, click the **3D Visualizer** button in the toolbar.



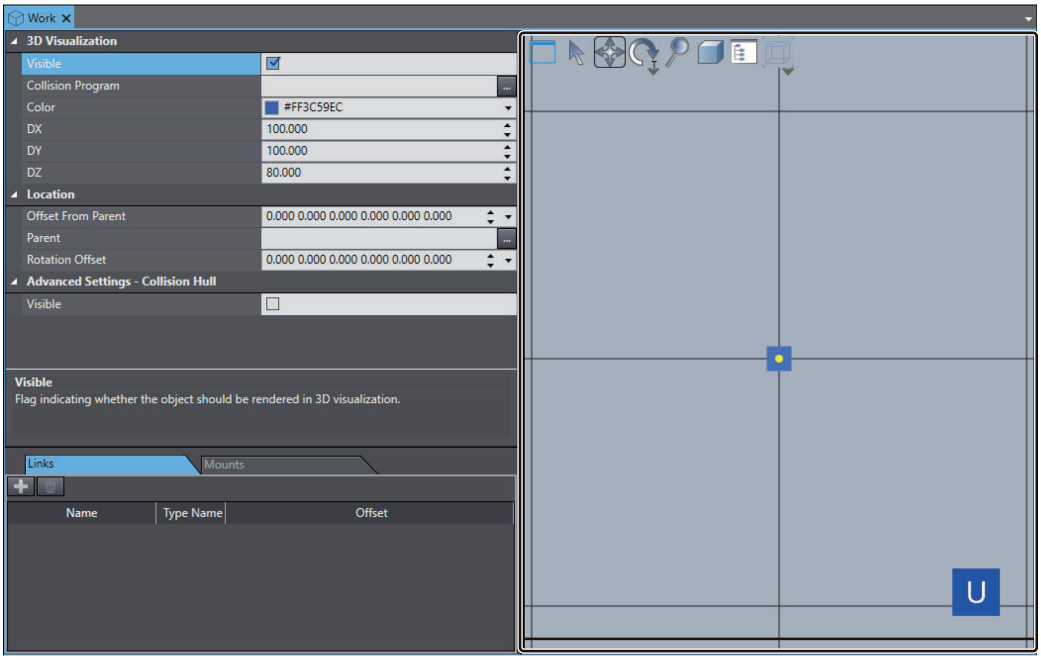

The **3D Visualizer** is displayed in the same place of the toolbox pane.




You can display the 3D Visualizer as a floating window and change the window size to make it easy to view. Refer to *Sysmac Studio Version 1 Operation Manual (Cat. No. W504)* for how to change the location of the window.

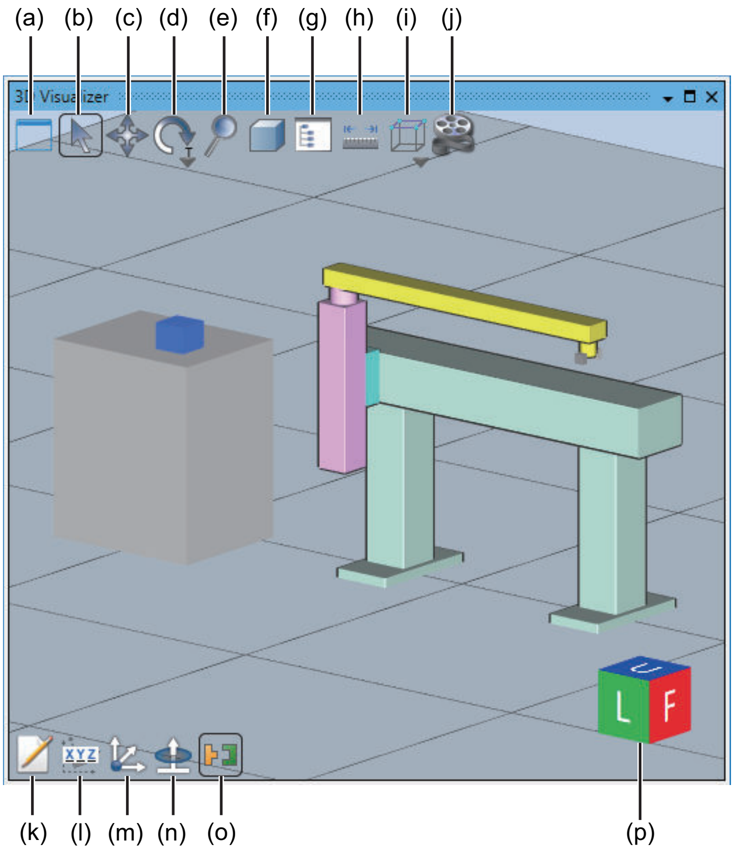
## 5-1-2 Displaying the 3D Editing Area

The 3D editing area is displayed as part of the setup tab page for individual 3D shape data. Refer to the procedures to add individual 3D shape data for the display method. An example of displaying the box 3D shape data is given below.


Work
3D Visualization
Visible
Collision Program
Color
#FF3C59EC
DX
100.000
DY
100.000
DZ
80.000
Location
Offset From Parent
0.000 0.000 0.000 0.000 0.000 0.000
Parent
Rotation Offset
0.000 0.000 0.000 0.000 0.000 0.000
Advanced Settings - Collision Hull
Visible
Visible
Flag indicating whether the object should be rendered in 3D visualization.
Links
Mounts
Name
Type Name
Offset
U

# 5-2 Displayed Items in the 3D Visualizer and 3D Editing Area

This section describes the displayed items and icons in the 3D Visualizer and 3D editing area. Some of the following functions are available only in the 3D Visualizer.




| Letter | Name | Description | Reference |
|---|---|---|---|
| (a) | Split Window | Splits the 3D Visualizer or 3D editing area. | *5-2-1 Split Window* on page 5-6 |
| (b) | Selection | Selects the 3D shape data to edit. | *5-2-2 Selection and Edit* on page 5-7 |
| (c) | Translate | Translates the point of view in the 3D Visualizer or 3D editing area. | *5-2-3 Translate* on page 5-7 |
| (d) | Rotate | Rotates the point of view in the 3D Visualizer or 3D editing area. | *5-2-4 Rotate* on page 5-8 |
| (e) | Zoom | Zooms in or out the 3D Visualizer or 3D editing area. | *5-2-5 Zoom* on page 5-8 |
| (f) | Projection Mode | Changes the projection modes in the 3D Visualizer and the 3D editing area: parallel projection or perspective projection. | *5-2-6 Projection Mode* on page 5-9 |
| (g) | Scene Graph | Opens the Scene Graph dialog box to configure the visibility and collision detection settings for 3D shape data. | *5-2-7 Scene Graph* on page 5-10 |
| (h)*1 | Measurement Ruler | Measures the distance between 3D shape data in the 3D Visualizer. | *5-2-8 Measurement Ruler* on page 5-12 |

| Letter | Name | Description | Reference |
|---|---|---|---|
| (i) | Snap | Moves 3D shape data, or a mount point or link point of 3D shape data, to a specified point. | *5-2-9 Snap* on page 5-13 |
| (j)[*1] | Record | Captures a simulation executed in the 3D Visualizer on video. | *5-2-10 Record* on page 5-15 |
| (k)[*1] | Edit | Displays the settings for the selected 3D shape data. | *5-2-2 Selection and Edit* on page 5-7 |
| (l)[*1] | Direct Position Edit | Allows you to enter the values to change the position, orientation, and size of the selected 3D shape data. | *5-3-3 Editing 3D Shape Data Simply* on page 5-23 |
| (m)[*1] | Edit Workspace Position | Moves the position of the selected 3D shape data. | *5-3-1 Moving 3D Shape Data* on page 5-20 |
| (n)[*1] | Edit Workspace Orientation | Changes the orientation of the selected 3D shape data. | *5-3-2 Rotating 3D Shape Data* on page 5-21 |
| (o)[*1] | Show/Hide Mount Points | Shows or hides mount points. | *5-4 Positioning with a Mount Point or a Link Point* on page 5-25 |
| (p) | 3D View Switching Tool | Switches the display direction of 3D shape data in the 3D Visualizer or 3D editing area. | *5-2-11 3D View Switching Tool* on page 5-18 |

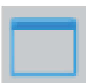
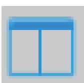
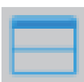
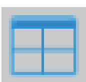
*1. They are not displayed in the 3D editing area.

Refer to *5-3 Operating 3D Shape Data in the 3D Visualizer* on page 5-20 for editing 3D shape data with the Edit, Direct Position Edit, Edit Workspace Position, and Edit Workspace Orientation icons.

## 5-2-1 Split Window

Use this icon to split the 3D Visualizer or 3D editing area.
Click the icon to open the following window. To close the window, click **X**.

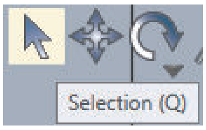
The functions of icons that you can select in the window are as follows.

| Icon | Name | Function |
|---|---|---|
|  | Not split | Does not split the 3D Visualizer or 3D editing area. |
|  | Split into two (Vertical) | Splits the 3D Visualizer or 3D editing area vertically into two sections. |
|  | Split into two (Horizontal) | Splits the 3D Visualizer or 3D editing area horizontally into two sections. |
|  | Split into four | Splits the 3D Visualizer or 3D editing area into four sections. |

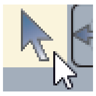
## 5-2-2 Selection and Edit

Use these icons to select 3D shape data and change the settings specific to it.

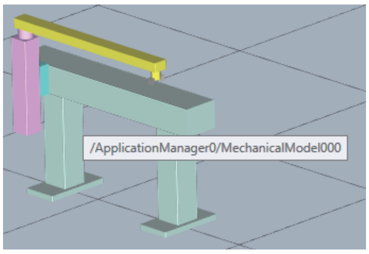
**1** Click the **Selection** icon in the 3D Visualizer or 3D editing area. Or, press the Q key.



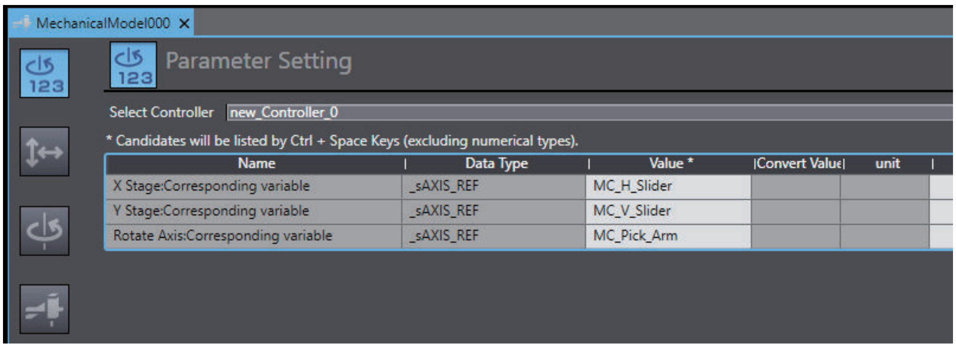

The cursor changes to an arrow that indicates that you are about to select 3D shape data.



**2** Move the cursor to the 3D shape data to select.
The name of the 3D shape data at the cursor position is displayed.




**3** To edit the 3D shape data in the 3D Visualizer, double-click the 3D shape data. Or, click the **Edit** icon.
The setup tab page for the 3D shape data is displayed.

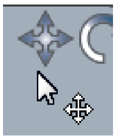



## 5-2-3 Translate

Use this icon to translate the point of view in the 3D Visualizer or 3D editing area.

**1** Click the **Translate** icon in the 3D Visualizer or 3D editing area. Or, press the W key.


The cursor changes to an arrow that indicates that you are about to translate the point of view in the 3D Visualizer or 3D editing area.



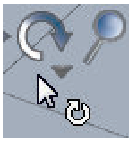
**2** Press and hold the left mouse button, and then drag the mouse in the direction to translate.

## 5-2-4 Rotate

Use this icon to rotate the point of view in the 3D Visualizer or 3D editing area.

**1** Click the **Rotate** icon in the 3D Visualizer or 3D editing area. Or, press the E key.


The cursor changes to an arrow that indicates that you are about to rotate the point of view in the 3D Visualizer or 3D editing area.



**2** Press and hold the left mouse button, and then drag the mouse in the direction to rotate.
Dragging up: Moves the point of view up.
Dragging down: Moves the point of view down.
Dragging right: Moves the point of view to the right.
Dragging left: Moves the point of view to the left.

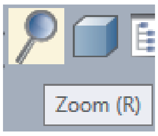
There are two rotation modes as follows.

| Mode | Description |
|---|---|
| Tumbler rotation | 3D shape data can be viewed from any angle. |
| Turntable rotation | The point of view can be rotated clockwise or counterclockwise around the Z axis of the world coordinate system. 3D shape data can be viewed in the range of ±90° vertically. |

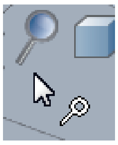
## 5-2-5 Zoom

Use this icon to zoom in or out the 3D Visualizer or 3D editing area.

**1** Click the **Zoom** icon in the 3D Visualizer or 3D editing area. Or, press the R key.




The cursor changes to an arrow that indicates that you are about to perform zooming.



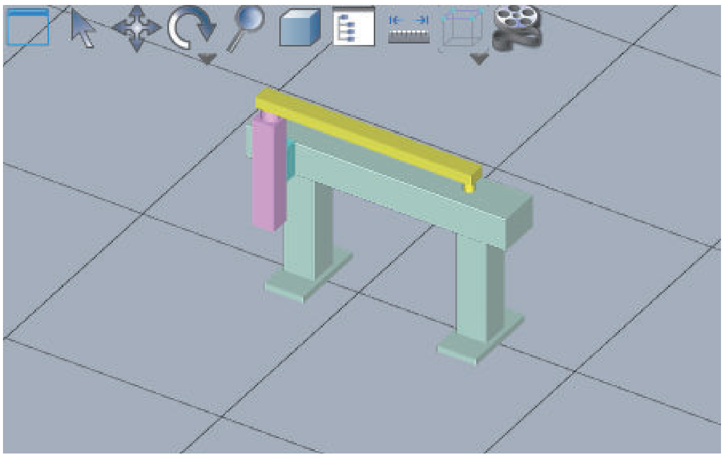
**2** Press and hold the left mouse button, and then drag the mouse.
Dragging up: Zoom-in
Dragging down: Zoom-out

## 5-2-6 Projection Mode

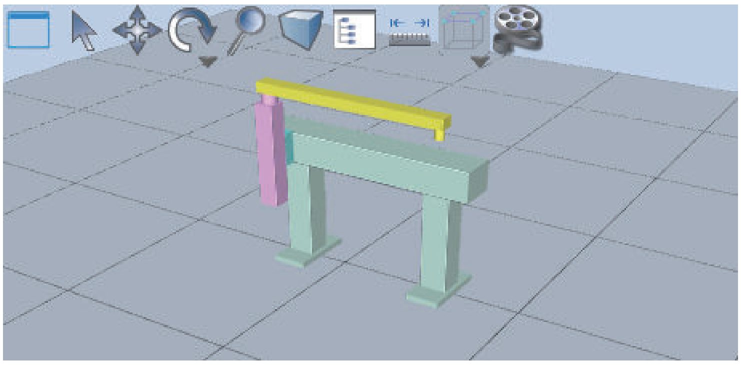
Use this icon to change the projection mode in the 3D Visualizer or 3D editing area between parallel projection and perspective projection.

| Icon | Name | Description |
|---|---|---|
|  | Parallel projection | In this projection mode, the projection lines are connected in parallel between every point on the object and the point of view.<br>It has a characteristic that an object is displayed in its true size regardless of the distance from the viewer. This projection method is suitable when you compare the sizes of objects that are placed. |
|  | Perspective projection | A method of projecting an object based on the law of perspective.<br>It has a characteristic that farther away an object from the viewer, smaller it appears, and closer the object to the viewer, larger it appears. This projection method is suitable when you display objects approximately in their size in the real world. |

Example of parallel projection



Example of perspective projection



## 5-2-7 Scene Graph

Use this icon to configure the visibility and collision detection settings for 3D shape data.

The setting items are as follows.

### Visibility Tab Page

In the list displayed on the tab page, set whether to show or hide the tiles, ruler, and 3D shape data in the 3D Visualizer or 3D editing area.
To display an item, select the corresponding check box.

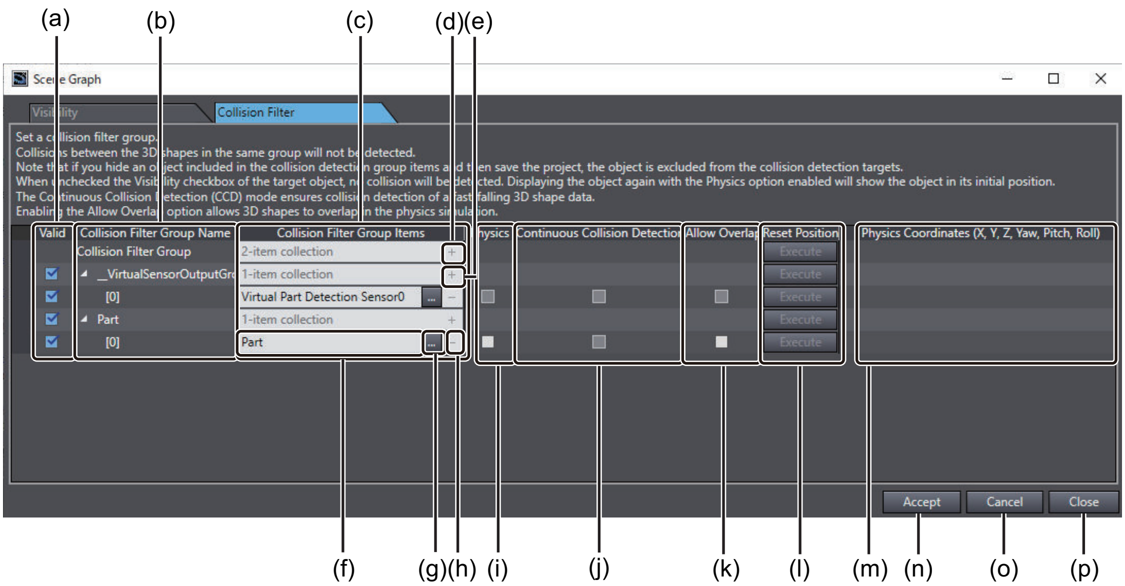

| | Item | Description | Set value | Initial value |
|---|---|---|---|---|
| (a) | Tiles | Select whether to display the XY plane with a 1,000-mm mesh when Z=0 in the 3D Visualizer or 3D editing area. | Checked or unchecked | Checked |
| (b) | Ruler | Select whether to display a ruler when you measure the distance between 3D shape data in the 3D Visualizer. This is the same as the function of the Measurement Ruler icon. | Checked or unchecked | Unchecked |
| (c) | Devices | Select whether to show or hide in the 3D Visualizer the 3D shape data registered in the project by device. [*1] | Checked or unchecked | Checked |

| | Item | Description | Set value | Initial value |
|---|---|---|---|---|
| (d) | 3D shape data | Select whether to show or hide in the 3D Visualizer the 3D shape data registered in the project by 3D shape data. *1 | Checked or unchecked | Checked |
| (e) | Object Visibility | | | |

*1. In the 3D editing area, 3D shape data is always displayed regardless of whether the check box is selected or cleared.

## Collision Filter Tab Page

The Collision Filter tab is displayed only in the 3D Visualizer.




| | Item | Description | Set value | Initial value |
|---|---|---|---|---|
| (a) | Valid check box | Select whether to enable the item in each collision filter group as a collision detection target. Select the check box to enable the item as a collision detection target. | Checked or unchecked | Checked |
| (b) | Collision Filter Group Name list | Manage the items in the Collision Filter Group list.<br>The text box displays **-itemcollection (* is the collision filter group number)*. Click the **Add Collision Filter Group** button to add a collision filter group.<br>The group name to be added is *Group* (* is the number of collision filter groups)*. | Text string | 0-item collection (No collision filter group in the initial status) |
| (c) | Collision Filter Group Items list | Manage the items in the Collision Filter Group Items list.<br>**-itemcollection (* is the number of collision filter group items)* is displayed in the list. Click the **Add Collision Filter Group Items** button to add a collision filter group item. | Text string | 0-item collection (No collision filter group item in the initial status) |
| (d) | Add Collision Filter Group button | Adds a collision filter group.<br>Clicking this button adds a collision filter group to the list. | --- | --- |

| | Item | Description | Set value | Initial value |
|---|---|---|---|---|
| (e) | Add Collision Filter Group Item button | Adds a collision filter group item.<br>Clicking this button adds a collision filter group item setting row to the list. | --- | --- |
| (f) | Collision Filter Group Item name | Set the target to add to the collision filter group.<br>Refer to *8-3-1 Collision Detection Target* on page 8-16 for the targets that you can set. | Text string | --- |
| (g) | Collision Filter Group Item Selection button | Displays a dialog box in which you can select the target to add to the collision filter group. | --- | --- |
| (h) | Delete Collision Filter Group Item button | Deletes a collision filter group item from the collision filter group.<br>Clicking this button deletes the item. | --- | --- |
| (i) | Physics | Selecting this check box and clicking the **Accept** button enables the physics simulation of the target 3D shape data.<br>This allows you to check how the falling 3D shape data collides with other 3D shape data and how the movable parts of custom mechanics move. | Checked or unchecked | Unchecked |
| (j) | Continuous Collision Detection | Enabling Continuous Collision Detection (CCD) makes it possible to certainly detect the collision of 3D shape data falling at high speed. Enabling this function makes sure of detection of collisions but affects the drawing performance of 3D Visualizer. | Checked or unchecked | Unchecked |
| (k) | Allow Overlap | Enabling Allow Overlap allows the target 3D shape data of physics simulation to overlap with each other. | Checked or unchecked | Unchecked |
| (l) | Reset Position | Resets the target 3D shape data of physics simulation to the original position.<br>When the **Physics** check box is selected, the target 3D shape data falls from the original position or the movable parts of a custom mechanics move from the original position each time you click the **Execute** button. | --- | --- |
| (m) | Physics Coordinates (X, Y, Z, Yaw, Pitch, Roll) | Displays the coordinate values of the 3D shape data during physics simulation. | --- | --- |
| (n) | **Accept** button | Accepts the changes. | --- | --- |
| (o) | **Cancel** button | Cancels the changes and then, closes the Scene Graph dialog box. | --- | --- |
| (p) | **Close** button | Closes the Scene Graph dialog box. | --- | --- |

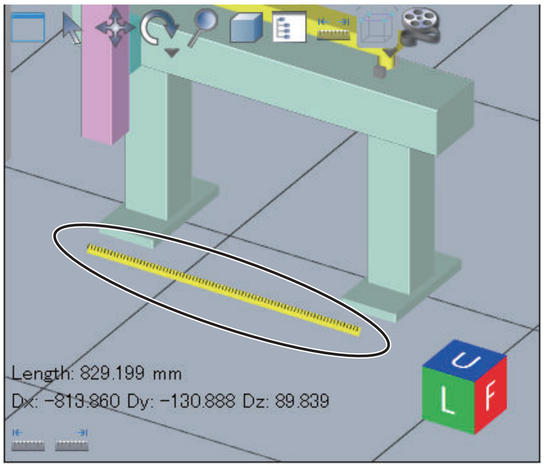
Refer to *8-3 Collision Detection Function* on page 8-16 for details on how to configure the collision detection settings.

## 5-2-8 Measurement Ruler

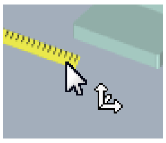
Use this icon to measure the distance between 3D shape data in the 3D Visualizer.

**1** In the 3D Visualizer, click the **Measurement Ruler** icon.


A yellow ruler (with black graduation) is displayed.




**2** Move the mouse cursor to one end of the ruler.
The mouse cursor changes.

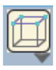


**3** Drag the end of the ruler to the end of the interval to be measured.

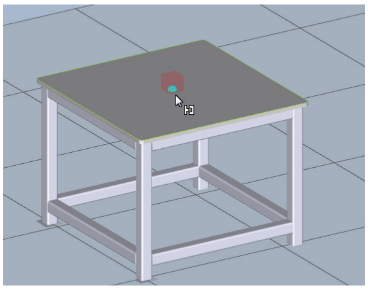
## 5-2-9 Snap

Use this icon to set an offset for a mount point or a link point of 3D shape data.

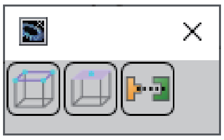
**1** Select the target 3D shape data, and then click the **Snap** icon or press the T key.



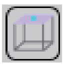
**2** Move the cursor to where you want to snap it to display candidate points of the snap destination. Click one of the candidate points to snap it to that position.

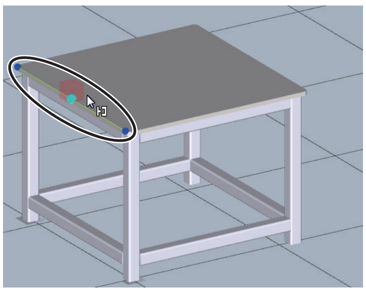


To change the mode of snapping, click the icon and select its function in the following window. Or, press the T + number keys. To close the window, click ×.

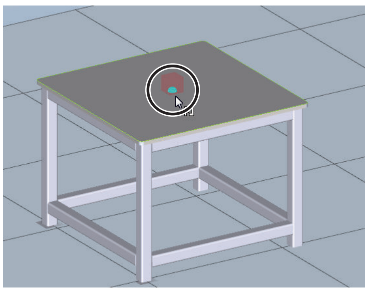


The functions of icons that you can select in the window are as follows.

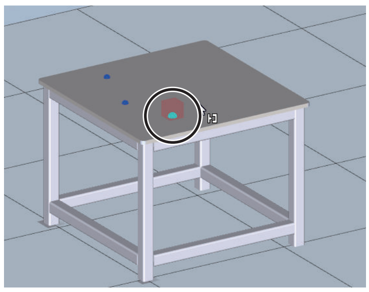
| Icon | Name | Function | Shortcut keys |
|---|---|---|---|
|  | Snap to Edge | Moving the cursor near a mount point or link point that is set in the 3D shape data displays a preview of the snap position.<br>Either both ends or the center of the highlighted edge is emphasized, and the mount point or the link point can be snapped to the position. | T + 1 |
|  | Snap to Face | Moving the cursor near a mount point or link point that is set in the 3D shape data displays a preview of the snap position.<br>The center of gravity of the highlighted face is emphasized, and the mount point or the link point can be snapped to the position. | T + 2 |
|  | Snap to Link | Moving the cursor near a mount point that is set in the 3D shape data displays a preview of the snap position.<br>Any link point that is present on the highlighted face is displayed, and the mount point can be snapped to the position.<br>This function is available only in the 3D Visualizer. | T + 5 |



Snap to Edge



Snap to Face



Snap to Link

**Additional Information**

- A preview of the snap position will be displayed for the following objects.
  a) Box
  b) Cylinder
  c) CAD data
  d) Mechanical component
  e) Custom mechanics
  f) Parallel link model
  When an end-effector mounted on a robot is snapped, a preview of the above objects a) to f) that have the end-effector set as the parent will be displayed.
- You can change the highlight color of the snap position. Change it in the option settings. Refer to the *Sysmac Studio Robot Integrated System Building Function with Robot Integrated CPU Unit Operation Manual (Cat. No. W595)* for details.

## 5-2-10 Record

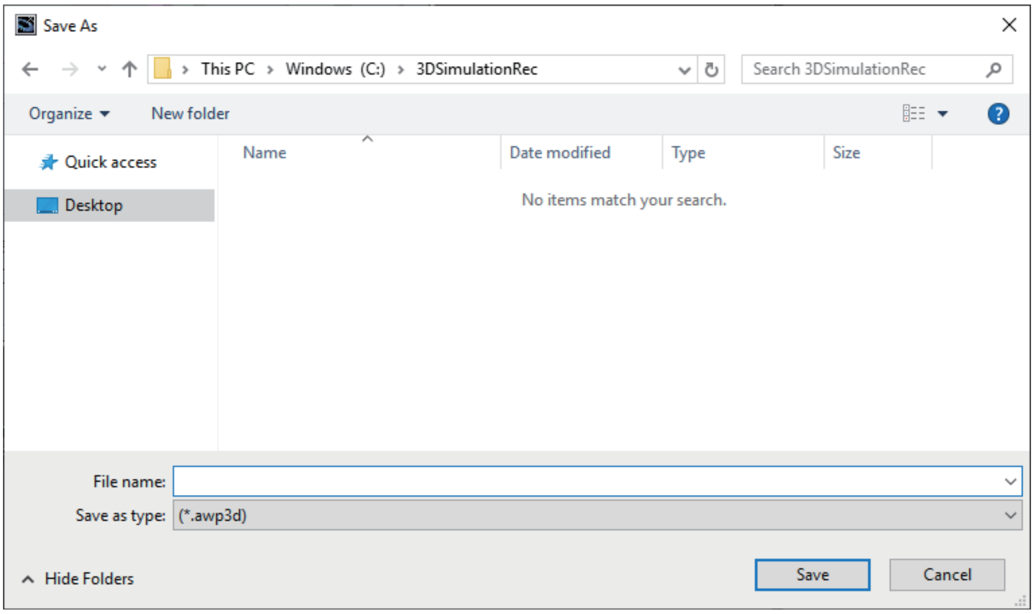
Use this icon to capture a simulation executed in the 3D Visualizer on video.

### Recording Video

**1** Click the **Record** icon during the execution of a 3D simulation.


The icon changes and starts flashing. This indicates that video recording is in progress.

**2** To stop the recording, click the **Record** icon.
The **Save As** dialog box is displayed.

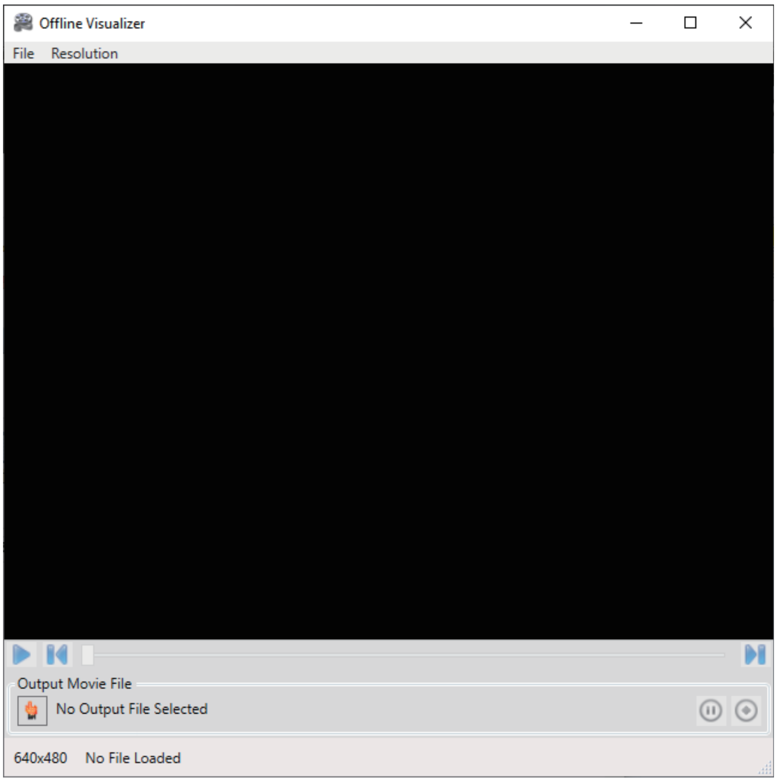



**3** Enter the file name, and then click the **Save** button.
The video is saved to a file.

## Playing Back Video

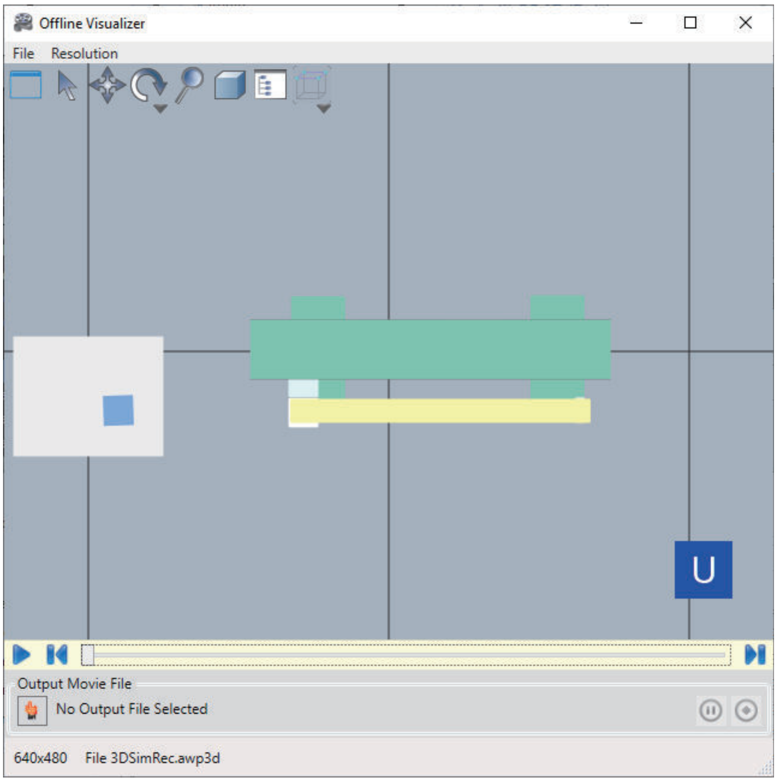
Use the Offline Visualizer to play back video.

**1** Select **All Programs** - **OMRON** - **Sysmac Studio** - **Tools** - **Offline Visualizer** from the Windows Start menu.
The **Offline Visualizer** starts.




**2** Select **Open** from the **File** menu.
The **Open** dialog box is displayed.

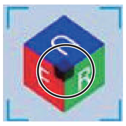
**3** Select the record file (with a .awp3d extension) to play back and then click the **Open** button.
The selected record file is opened.



**4** Click the **Play** button.
The video is played back.

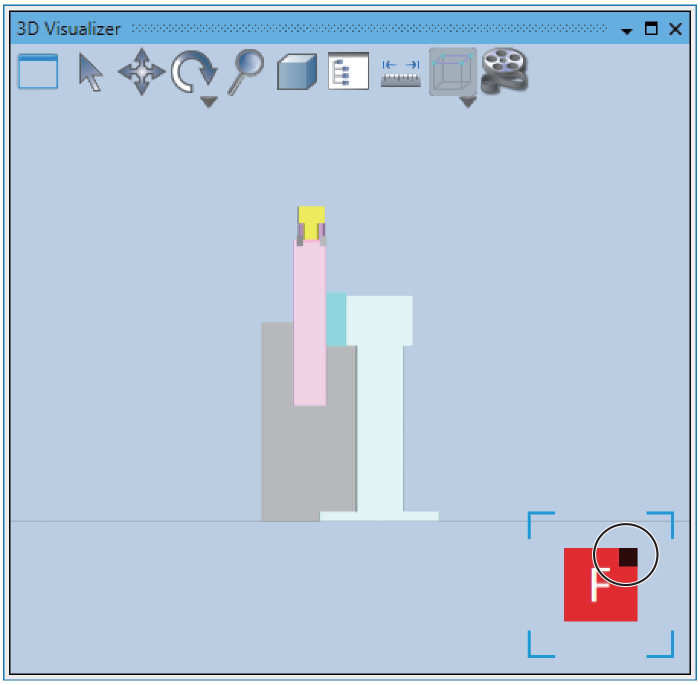
## 5-2-11 3D View Switching Tool

Use this icon to switch the display direction of 3D shape data in the 3D Visualizer or 3D editing area. The 3D View Switching Tool is made up of three elements, i.e., Face, Corner, and Edge. Place the mouse cursor over an element and, when it turns black, click it. Then, the view is switched so that the portion that you clicked is the front face. Accordingly, the display direction of 3D shape data in the 3D Visualizer or 3D editing area is switched.

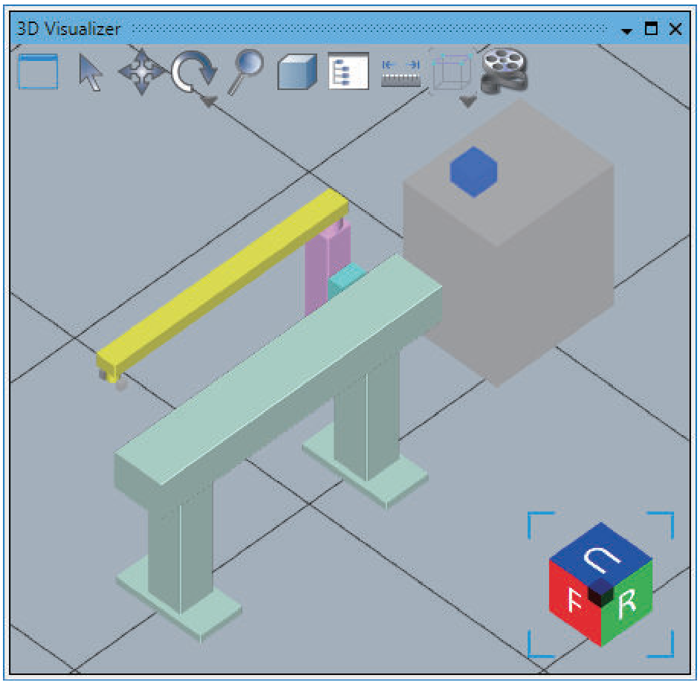
| Configuration element | Name | Description | Shortcut keys |
|---|---|---|---|
|  | Face | Represents a face. A face is indicated with one of the following symbols. | --- |
| | | F (Front): The front face when 3D shape data faces the yz plane | Ctrl + 1 |
| | | B (Back): The face parallel to the F face | Ctrl + 2 |
| | | U (Up): The upper orthogonal face to the F face | Ctrl + 3 |
| | | D (Down): The lower orthogonal face to the F face | Ctrl + 4 |
| | | L (Left): The left side face to the F face | Ctrl + 5 |
| | | R (Right): The right side face to the F face | Ctrl + 6 |
|  | Corner | Represents a corner. | --- |
|  | Edge | Represents an edge. | Ctrl + 7 (The edge between Up and Front) |

The operating procedure when you select Corner of the 3D View Switching Tool is given below, as an example.

**1** In the 3D Visualizer or 3D editing area, click the upper right corner of the Face icon in the 3D Visualizer.




The icon changes so that the selected corner faces front, with the view of the 3D shape data in the 3D Visualizer or 3D editing area switched.




**Additional Information**

To reset the scale and display position of 3D visualization to the initial status, place the mouse cursor in the 3D Visualizer or 3D editing area, and then press the Ctrl + 8 keys.

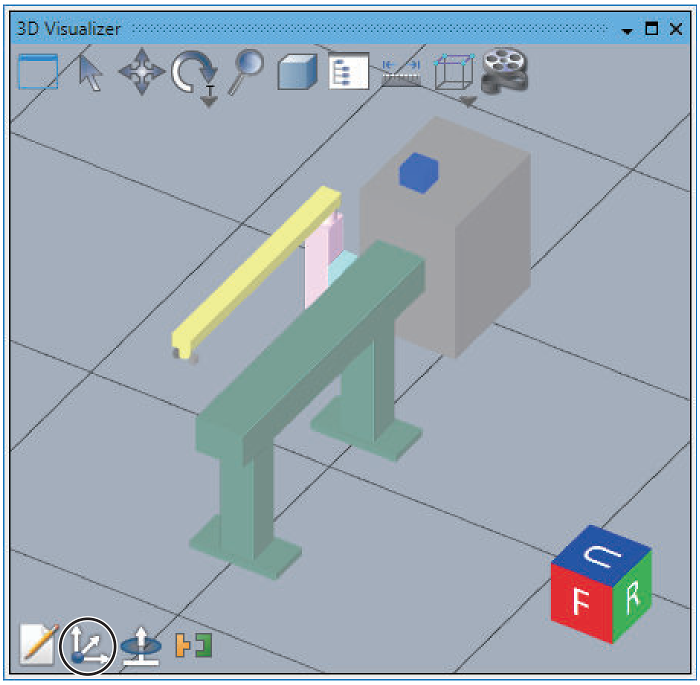
# 5-3 Operating 3D Shape Data in the 3D Visualizer

This section describes the operating procedures such as moving 3D shape data in the 3D Visualizer.

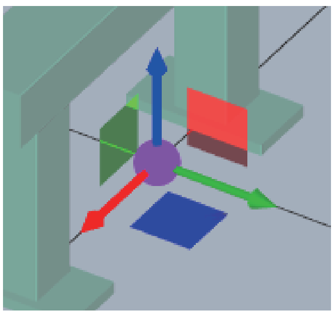
## 5-3-1 Moving 3D Shape Data

The 3D Visualizer allows you to edit 3D shape data while you are checking the positional relationship between 3D shape data.

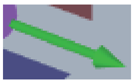
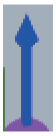
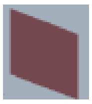
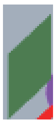
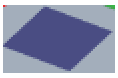
**1** In the 3D Visualizer, select the 3D shape data to move with the mouse cursor.
The **Edit Workspace Position** icon is displayed in the 3D Visualizer.




**2** Click the **Edit Workspace Position** icon.
The **Move** icon is displayed on the 3D shape data.



**3** Drag the icon to move the 3D shape data.

| Icon | Name | Function |
|---|---|---|
|  | Translate in X Direction | Dragging the icon translates the 3D shape data along the X axis of the local coordinate system. |
|  | Translate in Y Direction | Dragging the icon translates the 3D shape data along the Y axis of the local coordinate system. |
|  | Translate in Z Direction | Dragging the icon translates the 3D shape data along the Z axis of the local coordinate system. |
|  | Move on YZ Plane | Dragging the icon moves the 3D shape data on the YZ plane of the local coordinate system. |
|  | Move on XZ Plane | Dragging the icon moves the 3D shape data on the XZ plane of the local coordinate system. |
|  | Move on XY Plane | Dragging the icon moves the 3D shape data on the XY plane of the local coordinate system. |

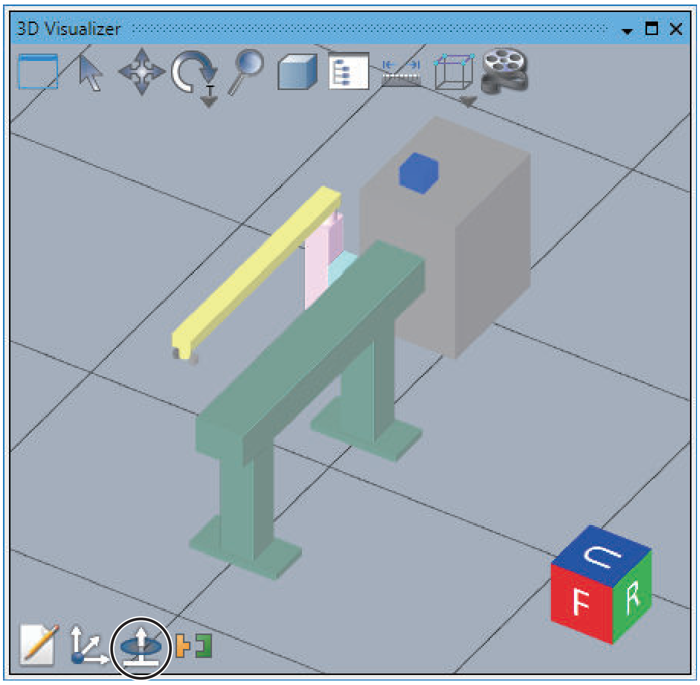
**Additional Information**

You can directly edit the **Location** values on the setup tab page for the target 3D shape data to move 3D shape data.

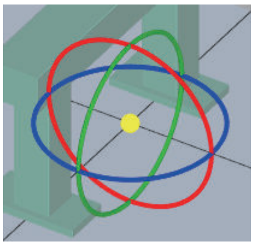
## 5-3-2 Rotating 3D Shape Data

The 3D Visualizer allows you to edit 3D shape data while you are checking the positional relationship between 3D shape data.

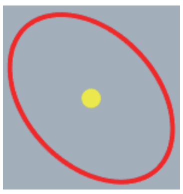
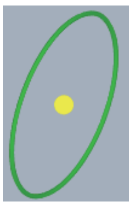
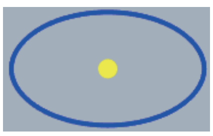
**1** In the 3D Visualizer, select the 3D shape data to rotate with the mouse cursor.
The **Edit Workspace Orientation** icon is displayed in the 3D Visualizer.




**2** Click the **Edit Workspace Orientation** icon.
The **Rotate** icon is displayed on the 3D shape data.



**3** Drag the icon to rotate the 3D shape data.

| Icon | Name | Function |
|---|---|---|
|  | Rotation around the X axis | Dragging the handle of the icon rotates the 3D shape data around the X axis with the point that you set in **Rotation Offset** in the 3D shape data setup tab page as the center of rotation. |
|  | Rotation around the Y axis | Dragging the handle of the icon rotates the 3D shape data around the Y axis with the point that you set in **Rotation Offset** in the 3D shape data setup tab page as the center of rotation. |
|  | Rotation around the Z axis | Dragging the handle of the icon rotates the 3D shape data around the Z axis with the point that you set in **Rotation Offset** in the 3D shape data setup tab page as the center of rotation. |

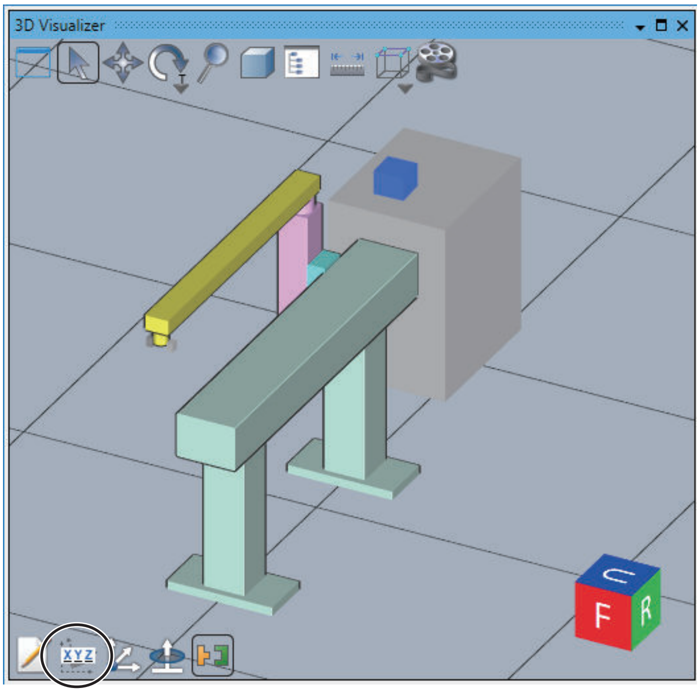
**Additional Information**

- You can directly edit the **Location** values on the setup tab page for the target 3D shape data to rotate 3D shape data.
- As you rotate the 3D shape data around each axis, its local coordinate system rotates together.

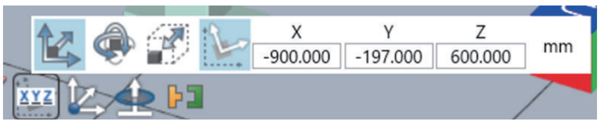
## 5-3-3 Editing 3D Shape Data Simply

The 3D Visualizer allows you to edit 3D shape data by directly entering values while you are checking the positional relationship between the 3D shape data.

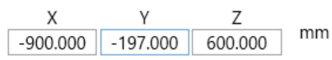
**1** In the 3D Visualizer, select the 3D shape data to edit with the mouse cursor.
The **Direct Position Edit** icon is displayed in the 3D Visualizer.

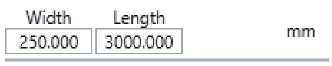
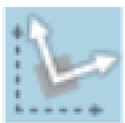
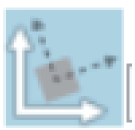



**2** Click the **Direct Position Edit** icon.
The value input fields for the 3D shape data are displayed in the 3D Visualizer.




**3** Select an icon in the **Direct Position Edit** to enable the value input fields for the 3D shape data.

| Icon | Name | Function |
|---|---|---|
|  | Edit Workspace Position | Enter X, Y, and Z values that specify the position of the 3D shape data.   |

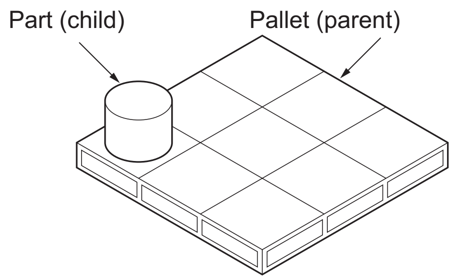
| Icon | Name | Function |
|---|---|---|
|  | Edit Workspace Orientation | Enter Yaw, Pitch, and Roll values that specify the orientation of the 3D shape data.<br> |
|  | Edit Size | Enter a value that specifies the size of the 3D shape data. The values that you can enter change depending on the 3D shape data.<br>**Box**<br><br>**Cylinder**<br><br>**Virtual Part Detection Sensor**<br><br>**Belt**<br> |
|  | Local Coordinate System | Select the coordinate system used for **Edit Workspace Position** or **Edit Workspace Orientation**. The coordinate system toggles every time you click. The initial setting is the local coordinate system. |
|  | World Coordinate System | |

# 5-4 Positioning with a Mount Point or a Link Point

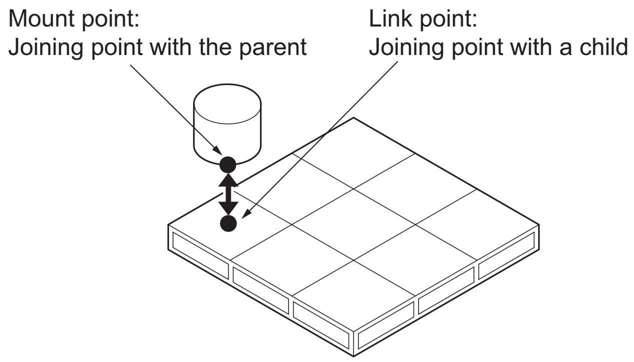
If you position two sets of 3D shape data in contact with each other, such as when you place the part on a specific pallet, you can use a mount point or a link point to position them easily.
This section describes the procedures to set a mount point or a link point and the operating procedure to place 3D shape data with the mount point and the link point.

## 5-4-1 Outline of a Mount Point and a Link Point

A mount point and a link point are the points used to join two sets of 3D shape data that have a parent-child relationship. The child-side joining point with a parent is called mount point, while the parent-side joining point with a child is called link point.




The pallet is the parent and the part is a child.



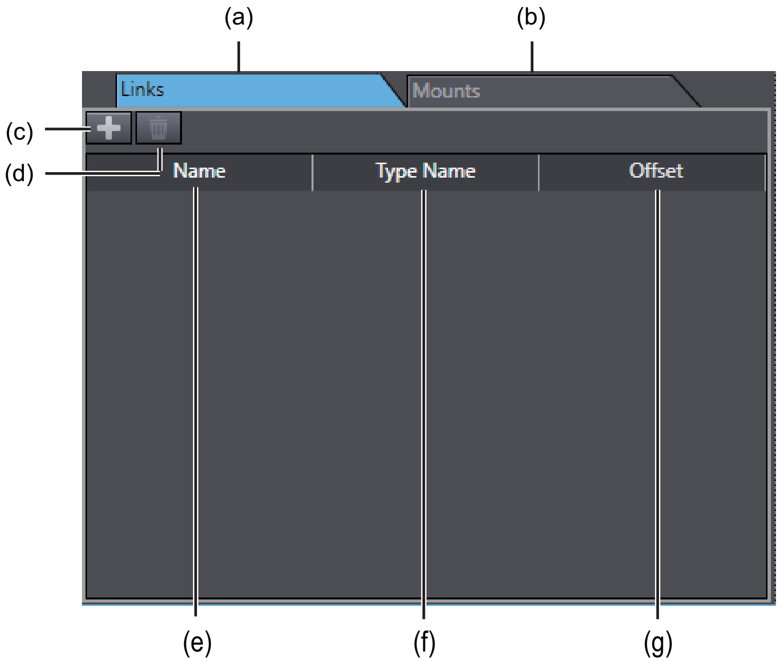

Example of setting the pallet as a mount point and the part as a link point

**Additional Information**

On the 3D Visualizer, it is not easy to use the mouse to correctly align the position of two 3D shape data that are in contact with each other.
However, you can position them easily on the 3D Visualizer by setting in advance a mount point and a link point on the target 3D shape data.

### Mount Point or Link Point Setup Tab Pages

Add a mount point or a link point for 3D shape data in the mount point or link point setup area on the tab page.

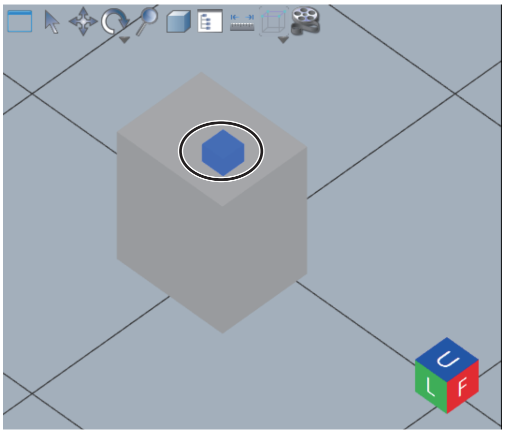



| | Item | Description | Set value | Initial value |
|---|---|---|---|---|
| (a) | Links tab page | Use this tab page to set a link point for the 3D shape data.<br>A link point is the point at which a child 3D shape data is joined to the parent 3D shape data. | None | Empty |
| (b) | Mounts tab page | Use this tab page to set a mount point for the 3D shape data.<br>A mount point is the point at which the child 3D shape data is joined to a parent 3D shape data. | None | Empty |
| (c) | Add Mount Point/Link Point button | Clicking this button adds a mount point or a link point to the table in the mount point or link point setting area. | --- | --- |
| (d) | Delete Mount Point/Link Point button | Clicking this button with the row to delete selected deletes the mount point or a link point in the selected row. | --- | --- |
| (e) | Name | The name of each mount point or link point is displayed.<br>You can set any name. | Text string | Connection Point |
| (f) | Type Name | The type of each mount point or link point is displayed.<br>When the same type name is set for mount points and link points of two sets of 3D shape data, selecting **Snap to Link** causes only link points with the same type name to be displayed in the 3D Visualizer. | Text string | Object |
| (g) | Offset | Set the coordinate of each mount point or link point.<br>Set the coordinate in the local coordinate system of the 3D shape data. The corresponding mount point or link point is displayed in the 3D editing area. Changing an offset value also changes the position of the mount point or link point displayed in the 3D editing area. The location elements are X, Y, Z, y (yaw), p (pitch), and r (roll), from left to right. | X, Y, Z: For each, -1,000,000.000 to 1,000,000.000<br>y, r: For each, -180.000 to 180.000<br>p: 0.000 to 180.000 | 0.000 for all |

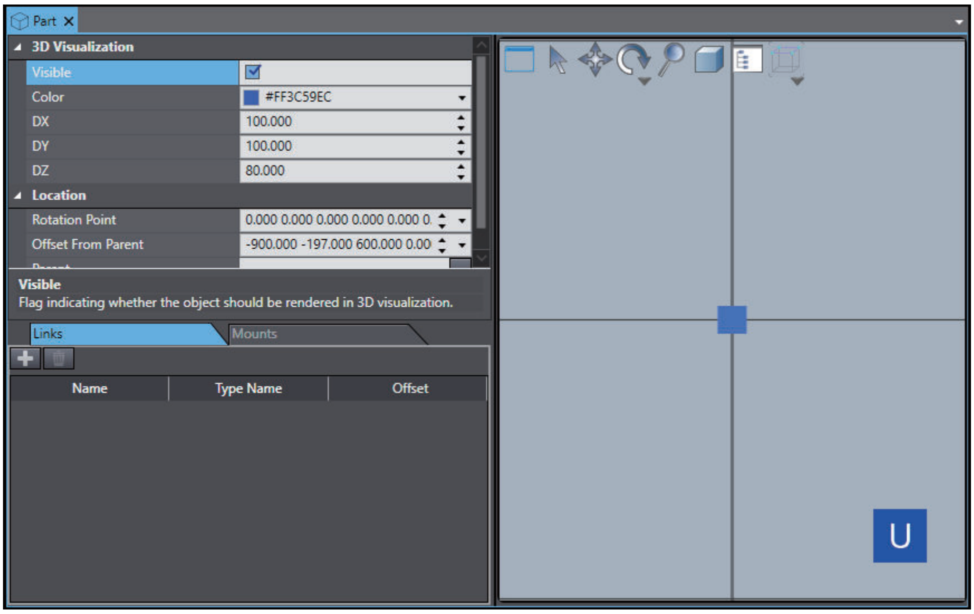
## 5-4-2 Setting a Mount Point

Use the following procedure to set a mount point.

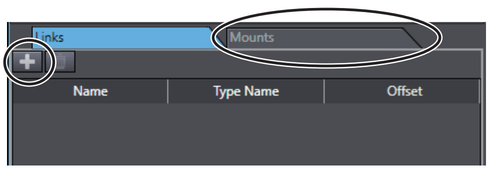
**1** In the 3D Visualizer or Multiview Explorer, double-click the 3D shape data for which to set a mount point.



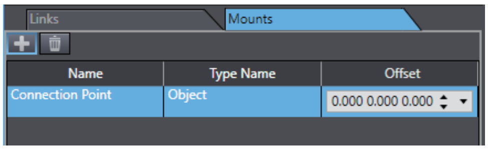
The setup tab page for the target 3D shape data is displayed.



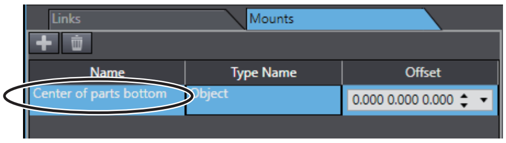

**2** Click the **Mounts** tab, and then click the **Add** button.



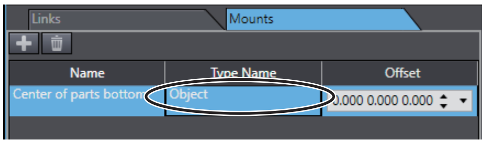

A row that contains a new mount point is added to the list.




**3** Enter a name for this mount point, and then press the Enter key.




**4** Enter a type for the mount point, and then press the Enter key.
Here, *Object* is set.

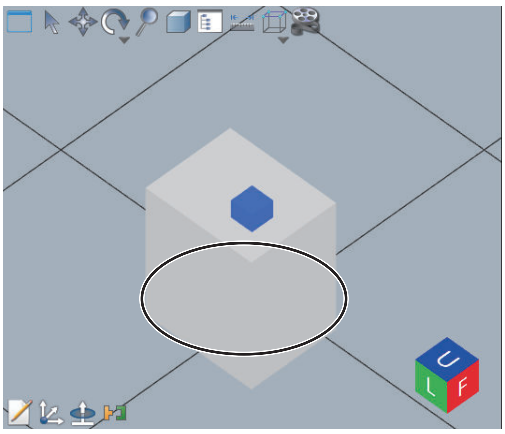



**5** Set an offset for the mount point.
Change the **Offset** value directly, or move the mount point in the 3D editing area. Refer to *5-4-4 Offset Setting Methods* on page 5-30 for the setting procedure in the 3D editing area.

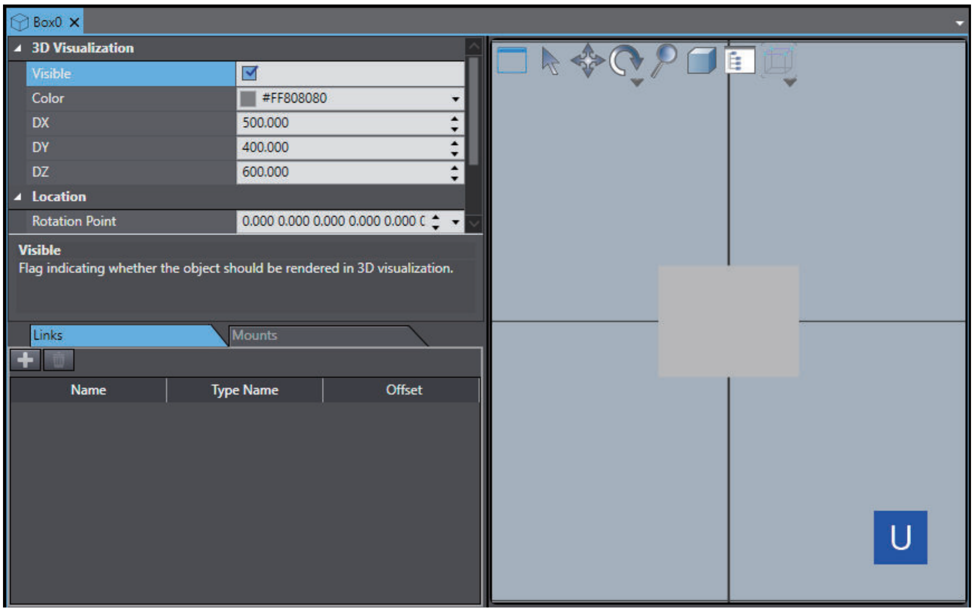
## 5-4-3 Setting a Link Point

Use the following procedure to set a link point.

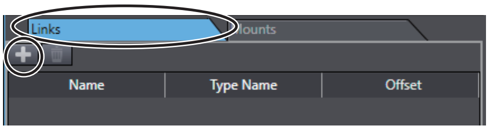
**1** In the 3D Visualizer or Multiview Explorer, double-click the 3D shape data for which to set a link point.



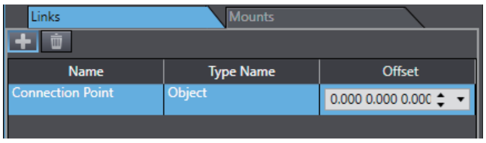
The setup tab page for the target 3D shape data is displayed.



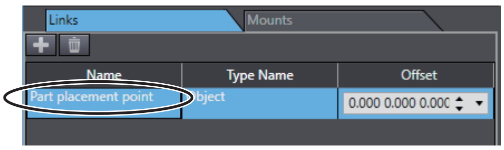

**2** Click the Links tab, and then click the **Add** button.



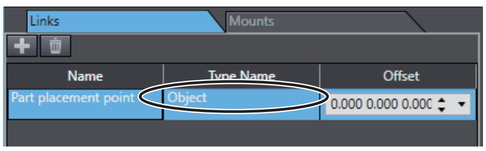

A row that contains a new link point is added to the list.




**3** Enter a name for this link point, and then press the Enter key.




**4** Enter a type for the link point, and then press the Enter key.
Here, *Object* is set.




**5** Set an offset for the link point.
Change the **Offset** value directly, or move the link point in the 3D editing area. Refer to *5-4-4 Offset Setting Methods* on page 5-30 for the setting procedure in the 3D editing area.

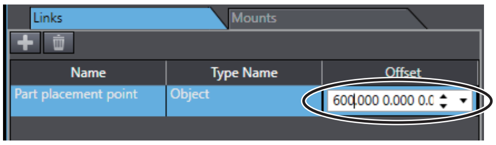
## 5-4-4 Offset Setting Methods

There are three ways of setting an offset for a mount point or a link point in the 3D editing area, as follows.

- Entering an offset coordinate directly
- Using icons
- Using the Snap function

### Entering an Offset Coordinate Directly

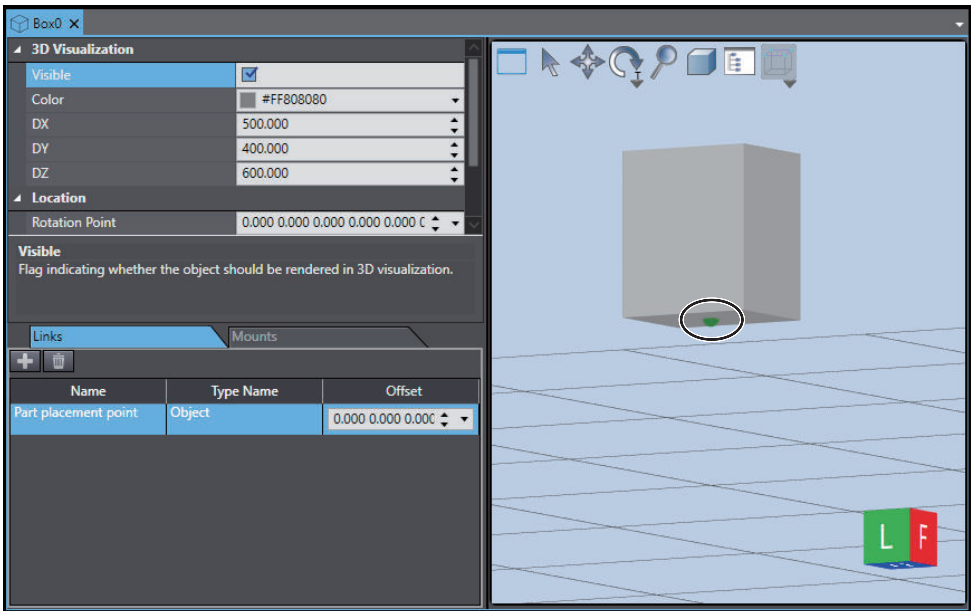
Enter the **offset** coordinate of each mount point or link point directly.



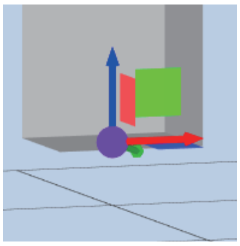

### Using Icons to Set an Offset

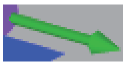
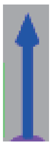
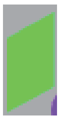
The procedure to set an offset with a link point is given below, as an example. The same procedure also applies when you set an offset for a mount point.

**1** Click the green link point icon.
Normally, it is located at the origin (X=0, Y=0, Z=0).

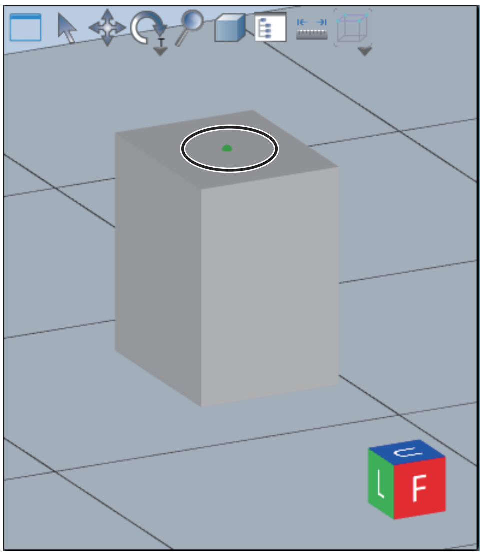



The following icon is displayed to move the link point.



| Icon | Name | Function |
|---|---|---|
|  | Translate in X Direction | Dragging the icon translates a link point in the X direction. |
|  | Translate in Y Direction | Dragging the icon translates a link point in the Y direction. |
|  | Translate in Z Direction | Dragging the icon translates a link point in the Z direction. |
|  | Move on YZ Plane | Dragging the icon moves a link point on the YZ plane. |
|  | Move on XZ Plane | Dragging the icon moves a link point on the XZ plane. |
|  | Move on XY Plane | Dragging the icon moves a link point on the XY plane. |

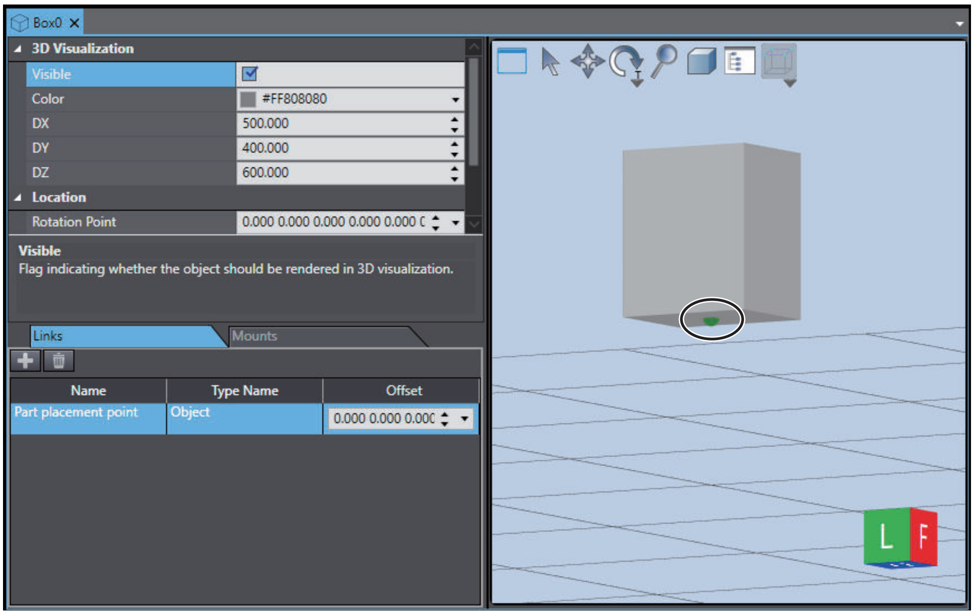
**2** Drag the icon to move the link point to the target position.



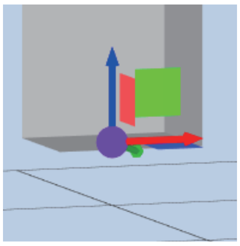
## Using the Snap Function to Set an Offset

The procedure to set an offset with a link point is given below, as an example. The same procedure also applies when you set an offset for a mount point.

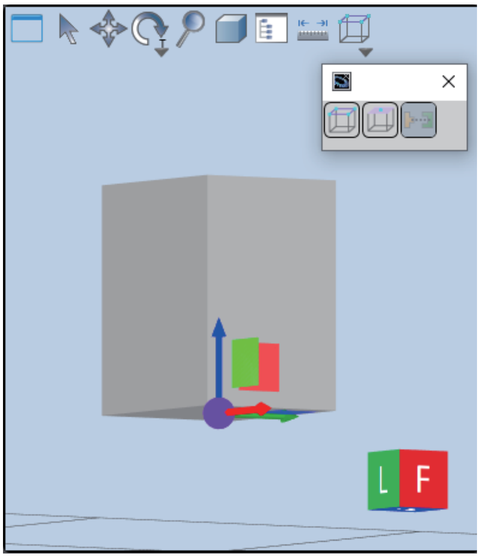
**1** Click the green link point icon.
Normally, it is located at the origin (X=0, Y=0, Z=0).



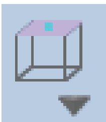

The following icon is displayed to move the link point.



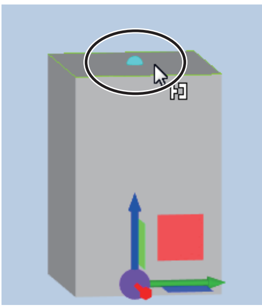
**2** Click the Snap icon, and then click the Snap to Face button.



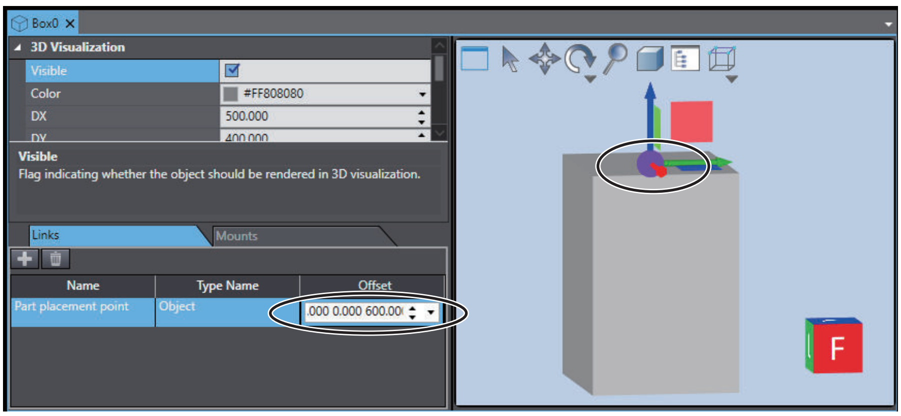

The icon changes.



**3** Move the cursor to around the center of gravity of the face where you want to set a link point.
The face of the 3D shape data turns blue, with its center of gravity shown as a light blue dot.

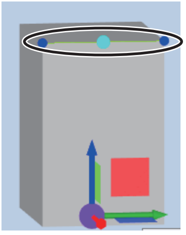


**4** Click the light blue dot.
The link point moves in the 3D editing area and an offset value is set in **Offset**.



**Additional Information**

If you use the Snap to Edge button, the end points and the intermediate point of the line segment are displayed as candidates of the link point.

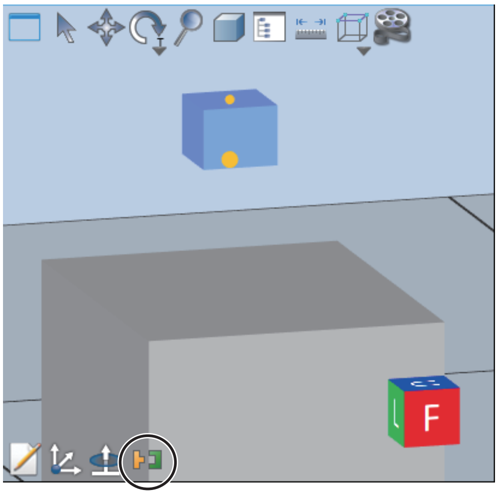


## 5-4-5 Using a Mount Point and a Link Point to Place 3D Shape Data

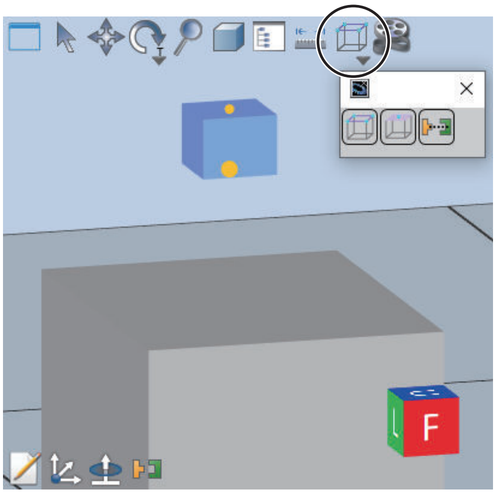
To place two sets of 3D shape data that has a mount point and a link point respectively, use the Snap function in the 3D Visualizer.
Select a mount point in the child 3D shape data, and then snap it to a link point in the parent 3D shape data.

**1** In the 3D Visualizer, select the 3D shape data to move.

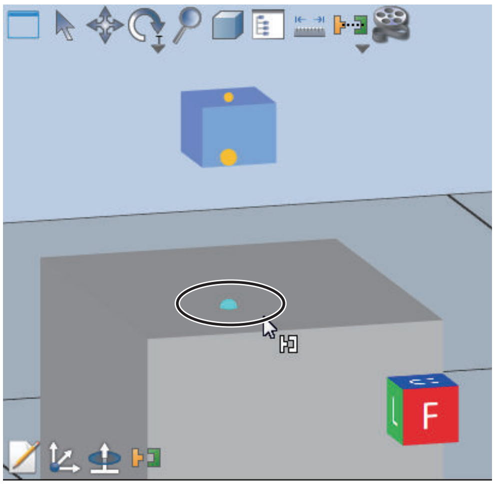
**2** Click the **Show/Hide Mount Points** icon to display mount points and select one in the 3D Visualizer.



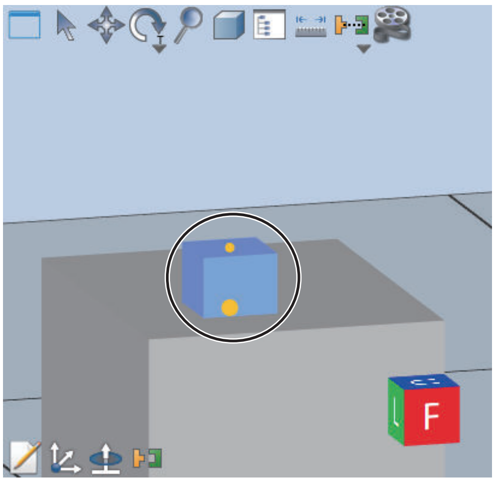
**3** Click the **Snap** icon, and then select **Snap to Link**.



**4** Select a link point.



The 3D shape data moves to the link point.



# 6

# Creating Settings and Scripts for Operating the 3D Shape Data

This section describes how to create settings and scripts for operating 3D shape data, such as the part and the mechanical component in a 3D simulation.

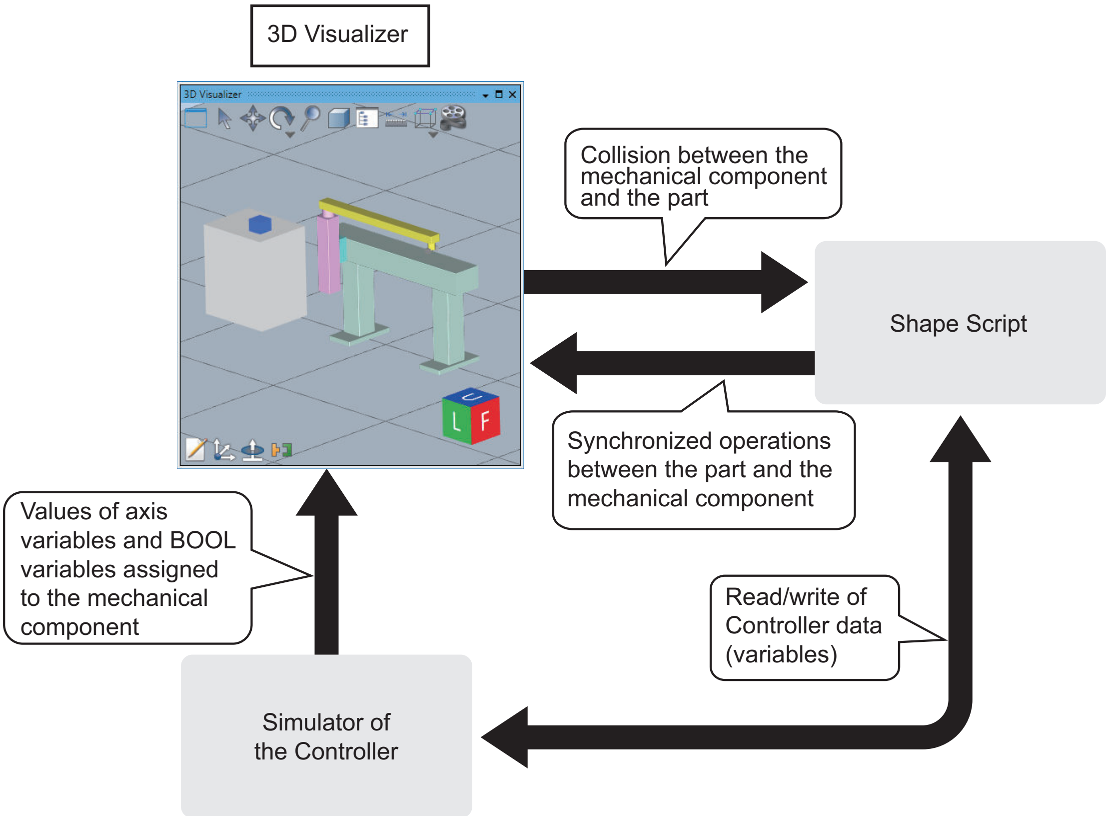
# 6-1 Outline of Settings and Scripts for Operating 3D Shape Data

This section describes how to create programs to realize the motions of the part and equipment in a 3D simulation.
Create a program for operating the part as a *Shape Script* that conforms to the C# language specifications.
For the following items, you can automatically generate scripts that define how they should operate.

- Virtual Output Scripts of Limit Switch
- Operation scripts for the Virtual Part Detection Sensor
- With Application Manager version 5.0 or higher, you can use *Shape Script Functions* as Shape Script function libraries.
  Refer to *6-6 Shape Script Functions* on page 6-24 for information on how to create Shape Script Functions and how to call them from a Shape Script.

## 6-1-1 Outline of a Shape Script

A Shape Script is a program that defines the operations of the part synchronizing with the operations of the mechanical component in the virtual equipment model displayed in the 3D Visualizer.
When you execute a Controller program during a 3D simulation, the values of axis variables and BOOL variables assigned to the mechanical component change. Then, in the 3D Visualizer, the mechanical component operates according to the changes in the values of the variables.
You can also display the operation of the part in the 3D Visualizer by executing the Shape Scripts for it concurrently with the execution of the Controller simulation.
A Shape Script uses the collision detection function to detect a collision between the mechanical component and the part in the 3D Visualizer and, based on it, generates an operation of the part that is synchronized with the operation of the mechanical component. This enables the 3D Visualizer to display synchronized movement of the part with the operation of the mechanical component.




**Additional Information**

Shape Scripts do not affect the operation of the actual equipment. They are programs written for a 3D simulation.

## 6-1-2 Execution Timing and Period of Shape Scripts and Programs

To execute Shape Scripts, assign them to a Shape Script Sequence. Refer to *6-3-2 Setting the Execution of Shape Scripts* on page 6-12 for the execution settings for Shape Scripts.
When you execute a Shape Script, the script written in the Render() function of the Shape Script is executed periodically. You can change the execution timing om the execution settings for the Shape Script Sequence. There are two types of execution timing settings, i.e., *asynchronous execution* that does not set the target Controller and *synchronous execution* that sets the target Controller. The following describes the execution timing in asynchronous execution and the execution period in synchronous execution.

The differences between asynchronous execution and synchronous execution are as follows.

| Item | Asynchronous execution | Synchronous execution |
|---|---|---|
| Execution speed of the Simulator of the Controller | Fast | Slow |
| Probability of missing collisions | Higher than synchronous execution | Lower as the execution count decreases |

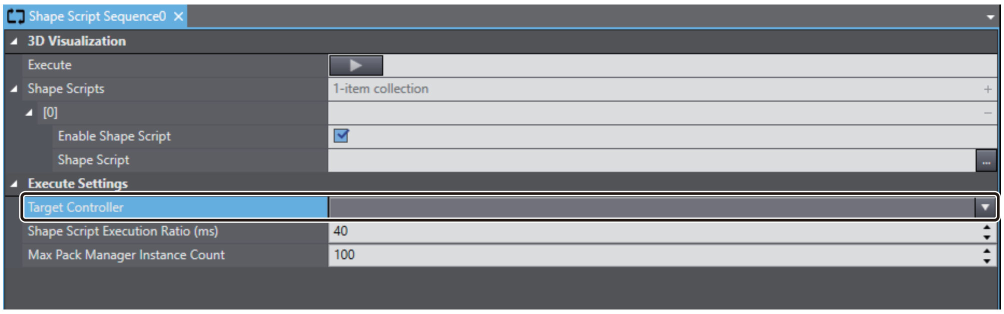
| Item | Asynchronous execution | Synchronous execution |
|---|---|---|
| Timing of sending and receiving signals and variables between the Simulator of the Controller and Shape Scripts | Signals and variables are sent and received between the Simulator of the Controller and Shape Scripts at an execution interval that is based on the *Shape Script Execution Ratio* setting for the Shape Script Sequence.<br>Because the execution interval is measured regardless of the internal time of the Simulator of the Controller, the signals and variables are not synchronized between the Simulator of the Controller and the Shape Script. | Signals and variables are sent and received between the Simulator of the Controller and Shape Scripts at an execution interval that is based on the *Execution Count* setting for the Shape Script Sequence.<br>Because the execution interval is measured based on the internal time of the Simulator of the Controller, the signals and variables are synchronized between the Simulator of the Controller and the Shape Script. |

The following describes in detail the execution timing in asynchronous execution and the execution period in synchronous execution.

## Asynchronous Execution and Execution Timing of Shape Scripts and Programs

Asynchronous execution refers to that the Simulator of the Controller and a Shape Script run independently, not synchronized. The Simulator runs faster in the asynchronous execution as it does not wait for an execution of Shape Script. However, signal transmissions and receptions, and 3D shape data collision detection timings in the 3D Visualizer, are imprecision in the asynchronous execution. Note that the Shape Script continues to run even if the Simulator of the Controller is paused.

To execute Shape Scripts asynchronously, set **Target Controller** for the Shape Script Sequence to **None**.




In asynchronous execution, the Render() function in a Shape Script is executed at the interval set in **Shape Script Execution Ratio** for the Shape Script Sequence. If you set **Shape Script Execution Ratio** to 50 ms, the Render() function in the Shape Script is executed once every 50 ms. To execute it in a shorter cycle, set a smaller value. To execute it in a longer cycle, set a larger value.

The following explains the difference between when the Shape Script Execution Ratio is set to 50 ms and when it is set to 200 ms.
The following is an example of a script that controls a part that moves at 100 mm/s on a conveyor belt.

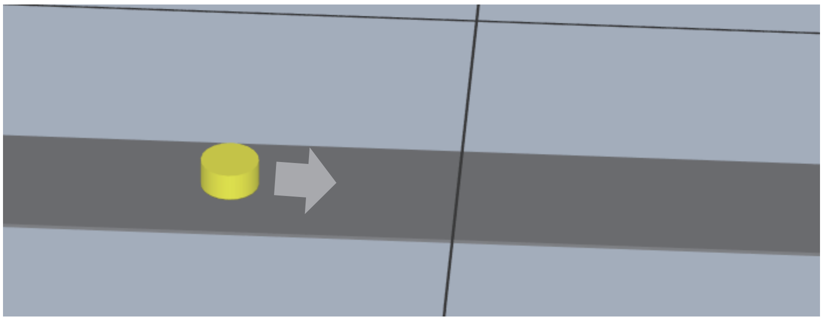
```
private DateTime previousTime;

/// <summary>
/// Called to render the Shape Script object
/// </summary>
/// <returns>The list of shapes to render</returns>
public override void Render(IEnumerable<ShapeRenderInfo> renderInfoList) {
    int speed = 100;        // 100mm/s
    DateTime currentTime = this.GetCurrentControllerTime("new_Controller_0");

    if (this.IsInitialized)
    {
        var delta = currentTime - this.previousTime;
        var distance = delta.TotalMilliseconds * speed / 1000;
        IShapeBase workpiece = (IShapeBase) ace["/ApplicationManager0/Workpiece"];
        workpiece.OffsetFromParent = new Transform3D(
            workpiece.OffsetFromParent.DX,
            workpiece.OffsetFromParent.DY + distance,
            workpiece.OffsetFromParent.DZ);
    }

    this.previousTime = currentTime;

    this.InitializeObject(renderInfoList);
}
```

Get the current time from the Simulator of the Controller.

Calculate the elapsed time since the previous execution of the Render() function to obtain the travel distance of the part.

Move the part.

Executing the script causes the part on the 3D Visualizer to move on the conveyor belt.



If you set the execution interval to 50 ms, the conveyor belt speed is 100 mm/s, so the part is drawn to move 5 mm on the 3D Visualizer.
On the other hand, if you set the interval to 200 ms, the part is drawn to move 20 mm on the 3D Visualizer. This means that the drawing is coarser than a drawing in the execution interval of 50 ms. Within that travel of 20 mm, no collision is detected even if the part collides with an obstacle.

In summary, setting a shorter execution interval provides smoother drawing and is more likely to detect collisions with obstacles. However, the greater the execution count, the greater the load on the computer, and the execution of the Simulator of the Controller may become slower.
Setting a long execution interval results in a smaller execution count, which means a smaller load on the computer and a lower risk of slow execution in the Simulator of the Controller. However, it is more likely to miss collisions with obstacles due to the coarse drawing.

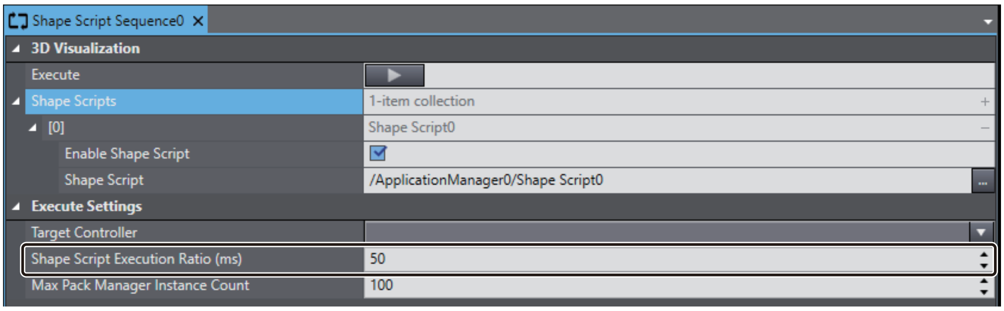
In addition, if **Shape Script Execution Ratio** is not set to an appropriate value, or if the execution time of the Shape Script exceeds the set execution ratio, the following phenomena may occur.

- The values of the I/O variables corresponding to the Virtual Part Detection Sensor change after the part passes through the position of the Virtual Part Detection Sensor on the 3D Visualizer.
- Executing a function that updates the display state or position of a part as a controller variable value changes, such as the LoadPart() template function, takes more time than expected to update the display state or part position after a variable value has changed.

- Executing a function that grasps a part, such as the ClampPart() template function, causes the part to start following the mechanical component after the position of the mechanical component changes, rather than when the mechanical component and the part come into contact.
- Executing a function for a part on a conveyor belt, such as the MoveObjectOnBelt() template function, results in a coarse drawing of the part.

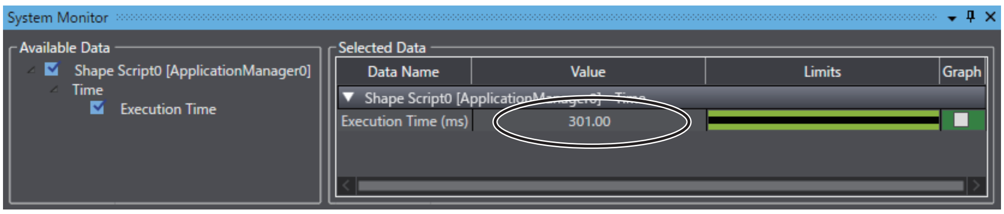
If any of these phenomena occur, use the following procedure to solve the issue.

**1** Set **Shape Script Execution Ratio** to a lower value.



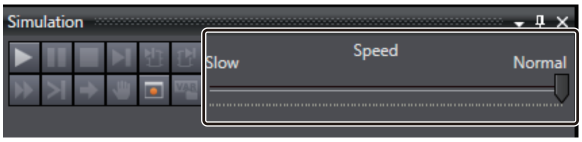

Setting **Shape Script Execution Ratio** to a lower value results in a reduced time until a phenomenon is processed after it is detected. For example, a Shape Script that changes the I/O variables corresponding to the Virtual Part Detection Sensor will be executed earlier after the part comes into contact with the Virtual Part Detection Sensor. Thus, you can change the values of the I/O variables at the expected time.

**2** If setting **Shape Script Execution Ratio** to a lower value does not improve the issue, the execution time of the Shape Script may have exceeded the set execution ratio. If so, modify the Shape Script so that its execution time will be reduced. Then, in the System Monitor of the Application Manager, check the execution time.




If the Shape Script contains a number of time-consuming operations such as reading/writing values from/to Controller variables, or if its execution time is long due to the specifications or operating conditions of the computer, the execution time of the Shape Script may exceed the setting of **Shape Script Execution Ratio**. If the Shape Script's execution time exceeds the set value in **Shape Script Execution Ratio**, modify the Shape Script so that its execution time is reduced.
If the Shape Script contains a number of operations to read/write values to Controller variables, modify it so that multiple variables will be read or written together, not one by one.

**3** If modifying the Shape Script does not improve the issue, or if changing the Shape Script itself is difficult, reduce the execution speed of the Simulator of the Controller. Drag the *Simulation Speed Slider* in the **Simulation** pane to change the execution speed of the Simulator.



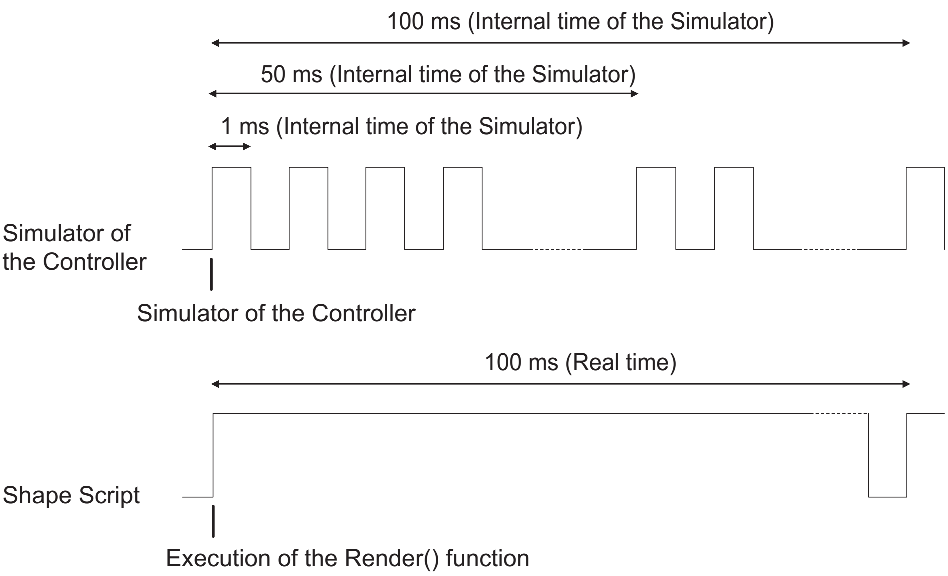

Changing the execution speed of the Simulator of the Controller does not change the execution time of the Shape Script.

Assume that the **Shape Script Execution Ratio** setting is 50 ms and the execution time of the Shape Script is 100 ms in the example of a conveyor mentioned above. When the execution time of Simulator is the actual speed, the part will be redrawn whenever it moves 10 mm forward.

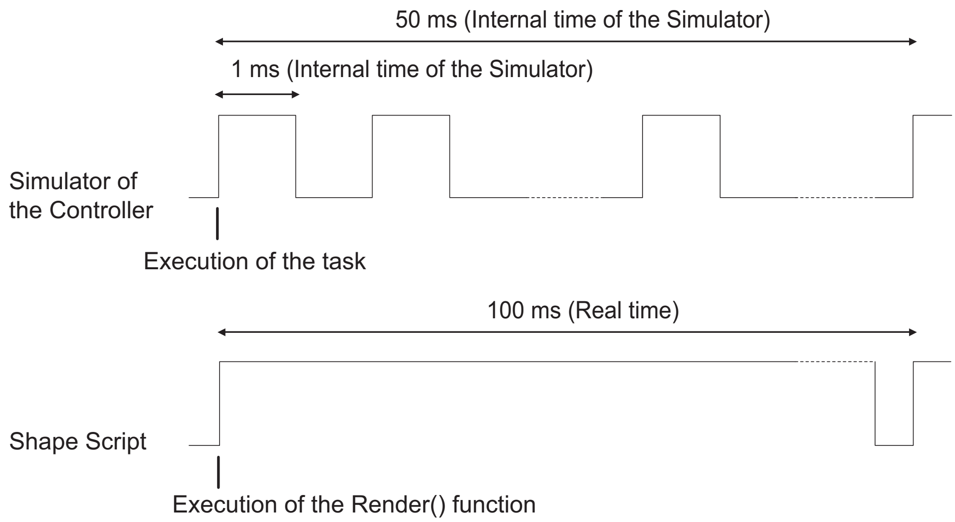
On the other hand, if you change the execution speed of the Simulator of the Controller to 0.5 times the actual speed, the Shape Script will be executed after a lapse of 50 ms in the internal time of the Simulator. In other words, with the internal time of the Simulator halved, the Shape Script will be executed in the same interval of 50 ms as the setting of **Shape Script Execution Ratio**. Since the part is redrawn whenever it moves 5 mm forward at this time, the movement of the part on the 3D Visualizer is smoother than when the execution speed of the Simulator is the actual speed.

These two cases are shown in timing charts as follows.

- **The Execution Speed of the Simulator of the Controller Is the Actual Speed**



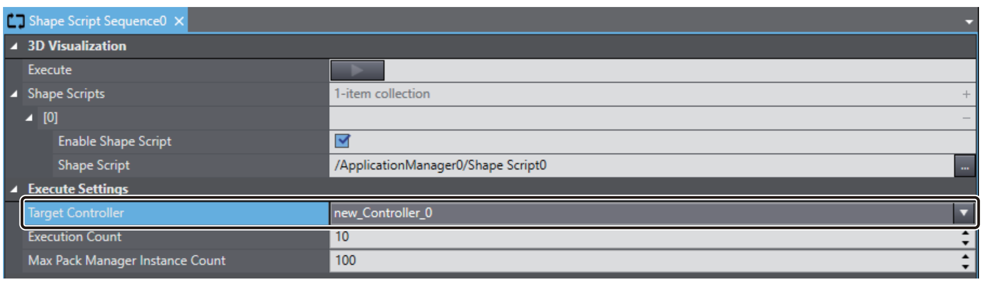
- **The Execution Speed of the Simulator of the Controller Is 0.5 Times the Actual Speed**




## Synchronous Execution and Execution Period of Shape Scripts and Programs

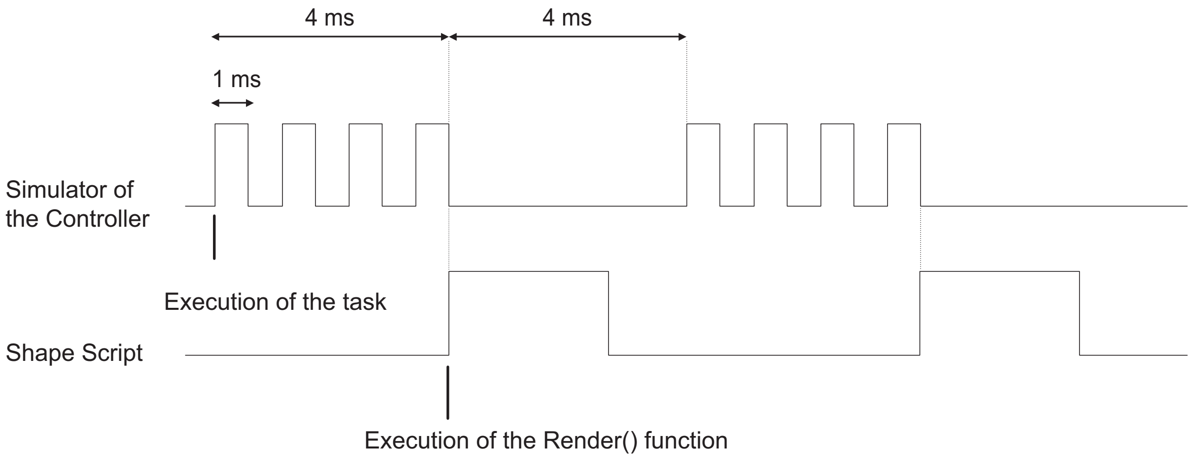
Synchronous execution refers to an execution in which the execution time of the Simulator of the Controller and that of a Shape Script are managed to achieve synchronicity. A Shape Script is executed after completion of the task that is running in the Simulator of the Controller. The Simulator of the Controller also waits for the Shape Script to complete before it executes a task. Since both of them wait for each other's completion before execution, the timing of sending and receiving signals each other and the timing of detecting collisions between 3D shape data on the 3D Visualizer are accurate, but the execution speed of the Simulator is slow due to this waiting. In addition, due to the waiting, the execution of the Shape Script also stops if the Simulator of the Controller is paused. Conversely, if the Shape Script is paused at a breakpoint, the Simulator of the Controller also stops.

To use synchronous execution between the Simulator of the Controller and a Shape Script, in **Target Controller** for the Shape Script Sequence, set the Controller that is registered in the project to synchronize.




In synchronous execution, the Render() function in a Shape Script is executed at the interval set in **Execution Count** for the Shape Script Sequence. For the execution count, a task period of 4 ms in the Simulator of the Controller is counted as 1. With the executions count set to 1, the Render() function in the Shape Script is executed once when the task in the Simulator of the Controller is executed for 4 ms. If the execution count is set to 2, the Render() function in the Shape Script is executed once when the task in the Simulator of the Controller is executed for 2 × 4 ms, or 8 ms.

When the task period is set to 1 ms for the Simulator of the Controller and the **Execution Count** is set to 1 for the Shape Script Sequence, the execution timing of the Simulator of the Controller and that of a Shape Script are as follows.




As shown in the figure, the Render() function in the Shape Script is executed when the task in the Simulator of the Controller is executed for 4 ms. Even if the execution time of the Render() function in the Shape Script is shorter than 4 ms, the next 4 ms of the task in the Simulator of the Controller is executed 4 ms after the execution of the Render() function. If the execution of the Render() function exceeds 4 ms, the next 4 ms of the task in the Simulator of the Controller is executed after completion of the execution.

**Additional Information**

You cannot select a Robot Integrated CPU Unit in **Target Controller** for the Shape Script Sequence.

# 6-2 Setting Mechanical Component

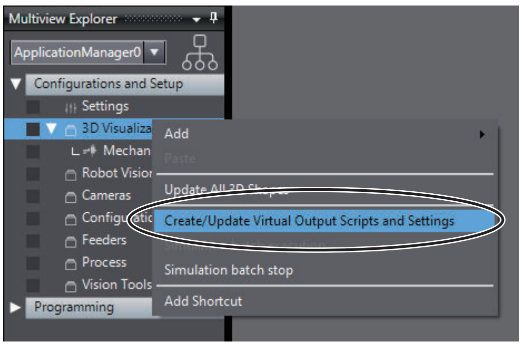
Configure the operation settings for a mechanical component.
Display the Mechanical Component tab page and, on the **Parameter Settings** screen, check that the target Controller and variables are assigned.
Refer to *4-3-5 Mechanical Component Settings* on page 4-28 for details on the settings.

## 6-2-1 Generating Virtual Output Scripts of Limit Switch

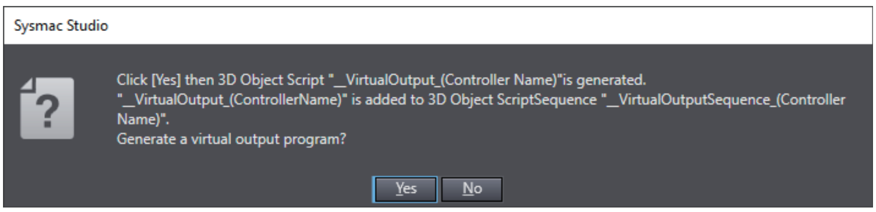
For specific mechanical components, you can automatically create operation scripts that reproduce the operations of a limit switch that the mechanical components perform in a 3D simulation.

| Mechanical Component | Type of virtual output | Description |
|---|---|---|
| • Air cylinder (Single solenoid type)<br>• Air cylinder (Double solenoid type) | Advance position detection | Detects the extended position of the piston. The output is turned ON when the piston is completely extended. |
| | Return position detection | Detects the return position of the piston. The output is turned ON when the piston is completely returned. |
| • Robot tool (Parallel switching 2-finger type chuck/single solenoid type)<br>• Robot tool (Parallel switching 2-finger type chuck/double solenoid type) | Open position detection | Detects the position of the chuck when it opens. The output is turned ON when the chuck is completely opened. |
| | Close position detection | Detects the position of the chuck when it closes. The output is turned ON when the chuck is completely closed. |

**1** Right-click **3D Visualization** under **Configurations and Setup** in the Multiview Explorer and select **Create/Update Virtual Output Scripts and Settings** from the menu.

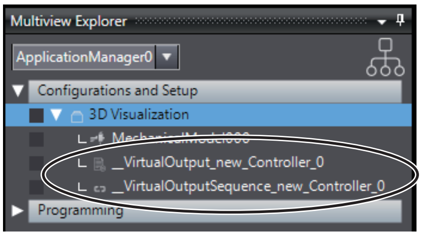



A script generation/update confirmation dialog box is displayed.



**2** Click the **Yes** button.

A Shape Script named **__VirtualOutput_(Controller name)** and a Shape Script Sequence named **__VirtualOutputSequence_(Controller name)** are generated and displayed under 3D Visualization. If **__VirtualOutput_(Controller name)** and **__VirtualOutputSequence_(Controller name)** are already present, the Shape Script and Shape Script Sequence are updated.




Refer to *8-1 Operating Procedures for a 3D Simulation* on page 8-2 for how to execute generated scripts.

### Precautions for Correct Use

- If you use the following types of mechanical components, do not turn ON both virtual outputs at the same time. Doing so may cause the mechanical components to fail.
  - Air Cylinder (Double Solenoid Type): *Advance position detection* and *Return position detection*
  - Robot Tool (Parallel Switching 2-finger Type Chuck/Double Solenoid Type): *Open position detection* and *Close position detection*
- If any of the following operations is performed, right-click and select **Create/Update Virtual Output Scripts and Settings** from the menu again. Selecting **Create/Update Virtual Output Scripts and Settings** updates the registered **__VirtualOutput** and **__VirtualOutputSequence** values and applies the changes to the settings.
  - Changing the settings of the target mechanical component
  - Changing the target Controller, times of execution, or Shape Script execution interval setting of the Shape Script Sequence **__VirtualOutputSequence**

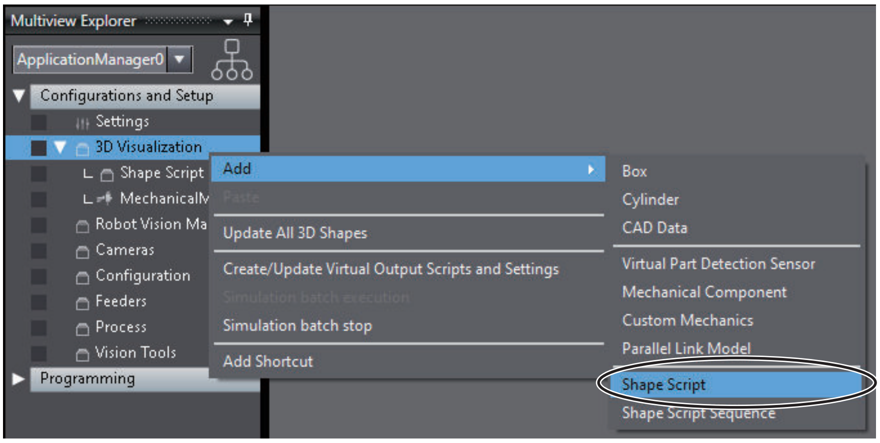
# 6-3 Creating Operation Scripts for the Part

Add Shape Scripts that define the operations of the part, and then create programs in C# language. Then, add a Shape Script Sequence and set the order of execution of the Shape Scripts.

## 6-3-1 Adding Shape Scripts

The procedure to add Shape Scripts is given below.

**1** Right-click **3D Visualization** under **Configurations and Setup** in the Multiview Explorer and select **Add** - **Shape Script** from the menu.




**Shape Script0** is registered and displayed under **3D Visualization**.
**Default Functions** is registered under **Shape Script Functions**.
For this registration, the following conditions must be met.

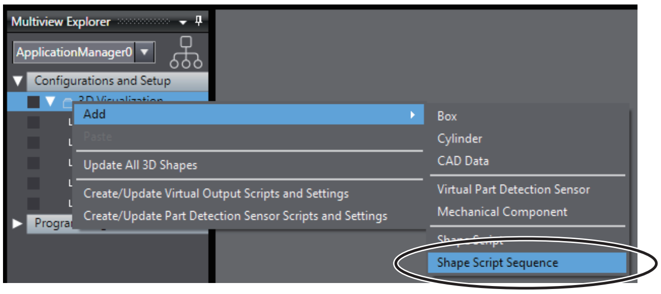
- Application Manager version 5.0 or higher
- Initial registration of a Shape Script

**2** Double-click **Shape Script0**. Or, right-click and select **Edit** from the menu.
The Shape Script Editor window is displayed.
In the Shape Script Editor, create programs that define the operations of the part. Refer to *6-5 Shape Script Editor* on page 6-18 for details on the Shape Script Editor window.

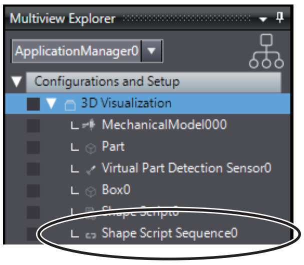
## 6-3-2 Setting the Execution of Shape Scripts

To execute the Shape Scripts that define the operations of the part, assign them to a Shape Script Sequence.
Use the following procedure to execute the Shape Scripts.

**1** Right-click **3D Visualization** under **Configurations and Setup** and select **Add** - **Shape Script Sequence** from the menu.

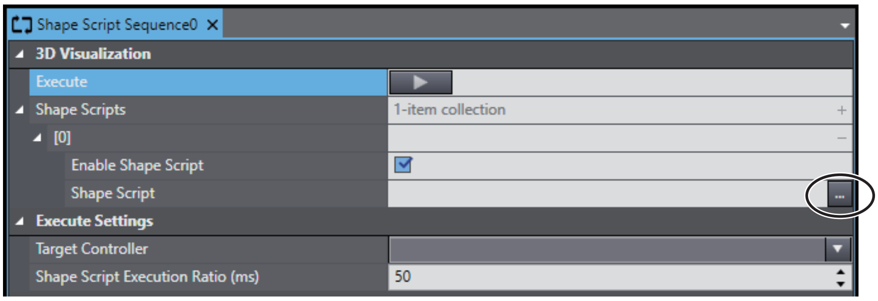



**Shape Script Sequence0** is registered and displayed under 3D Visualization.

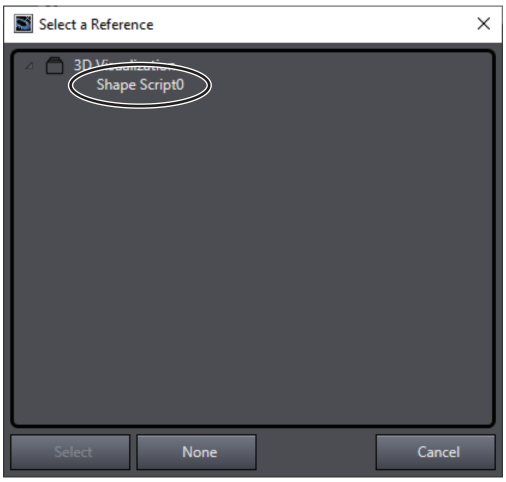



**2** Double-click the Shape Script Sequence.
The Shape Script Sequence setup tab page is displayed.

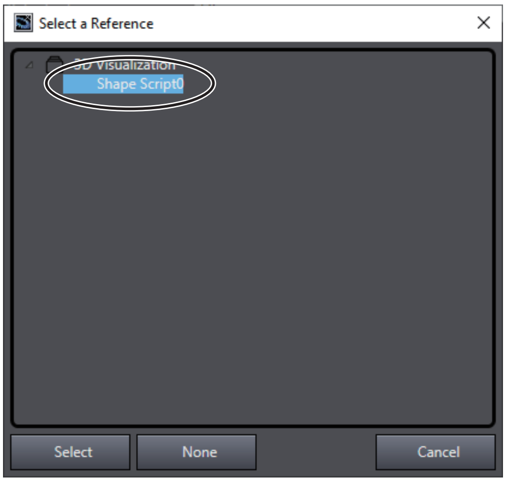
**3** Click the Browse button for the Shape Script.



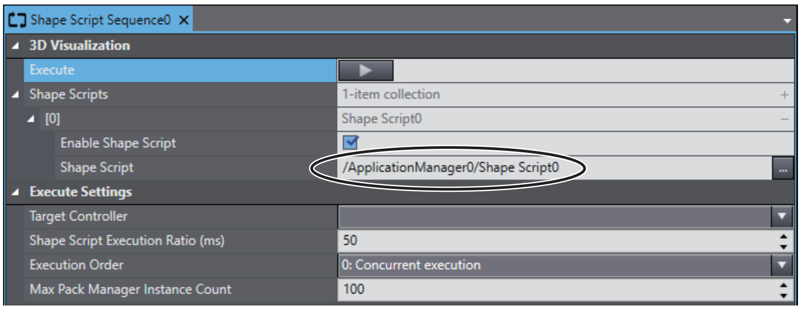

A list of registered Shape Scripts is displayed.



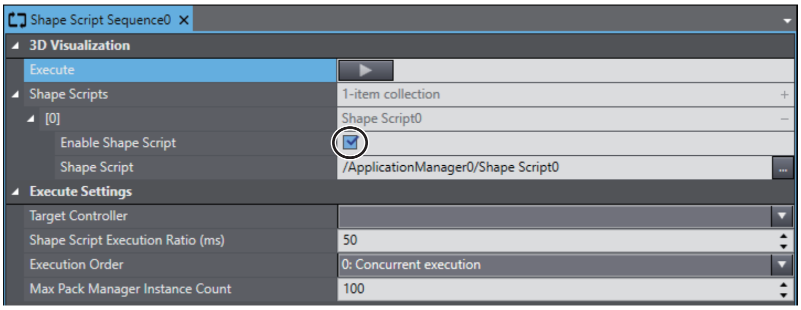

**4** Select the Shape Scripts to execute, and then click the **Select** button.




The Shape Scripts are assigned to the Shape Script Sequence.

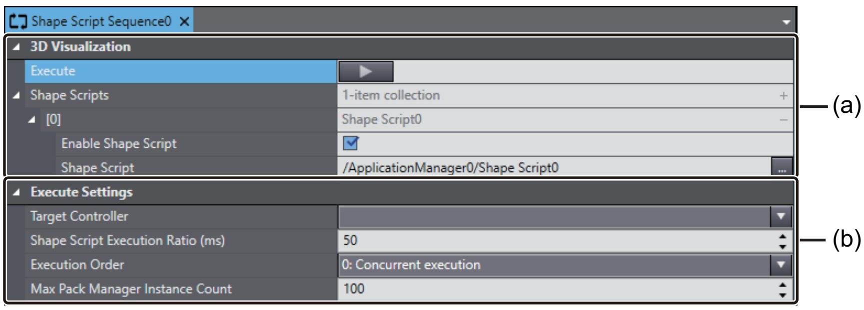



**5** Select the **Enable Shape Script** check box.




This completes the procedure to set the Shape Scripts for execution.

The setting items for the Shape Script Sequence are listed in the following table.



| | Item | | Description | Set value | Initial value |
|---|---|---|---|---|---|
| (a) | 3D Visualization | Execute | Executes the Shape Scripts. Select the **Enable Shape Script** check box, and then click this button to execute the registered Shape Scripts. | --- | --- |
| | | Shape Scripts | The number of registered Shape Scripts assigned to the Shape Script Sequence is displayed. | --- | --- |
| | | Index | The order of execution of Shape Scripts during the execution of the Shape Script Sequence is displayed. *1 | --- | --- |
| | | + (Add) button | Adds the Shape Script to assign to the Shape Script Sequence. | --- | --- |
| | | - (Unassign) button | Unassigns the selected Shape Script. | --- | --- |
| | | Enable Shape Script | Select whether to execute the Shape Script. | Checked or unchecked | Checked |
| | | Shape Script | The name of the assigned Shape Script is displayed. Click the button at the right, and then select the Shape Script to execute. | Name of Shape Script | None |
| (b) | Execute Settings | Target Controller | Select the Controller to execute the Shape Script Sequence. | Controllers registered in the project | None |
| | | Execution Count*2 | Set the number of times of user program execution in the primary periodic task executed in the Controller per execution of the Shape Scripts. | 1 to 1,000 | 10 |
| | | Shape Script Execution Ratio (ms)*2 | Set the execution period of the Shape Script. (unit: ms) | 0 or higher integer value | 50 |
| | | Execution Order | Select the order of execution of Shape Scripts during the simultaneous execution of multiple Shape Script Sequences. | 0: Execution independent of order, 1 to Number of Shape Script Sequences registered in project: Execution in order of selection | 0 |

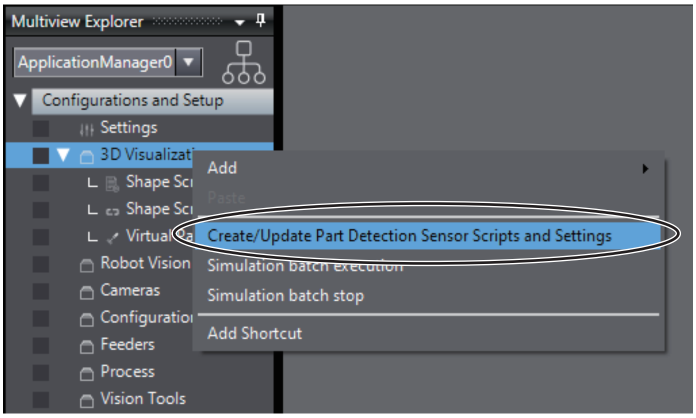
*1. To change the order of execution of a Shape Script, right-click the Shape Script and select **Move Up** or **Move Down** from the menu.

*2. **Execution Count** is displayed when the target Controller is selected, while **Shape Script Execution Ratio (ms)** is displayed when the target Controller is not selected.

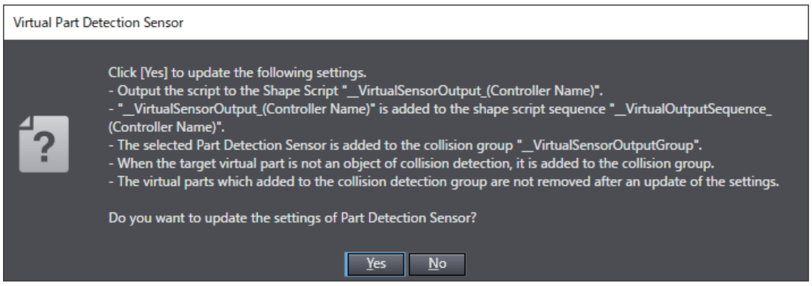
# 6-4 Configuring the Operation Settings for the Virtual Part Detection Sensor

Define the operations that the Virtual Part Detection Sensor performs to detect the part in a *Virtual Part Detection Sensor script*.
Use the following procedure to add the Virtual Part Detection Sensor script.

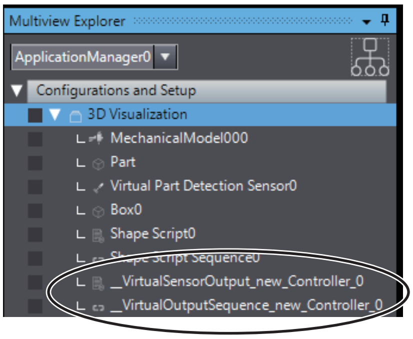
**1** Right-click **3D Visualization** under **Configurations and Setup** in the Multiview Explorer and select **Create/Update Part Detection Sensor Scripts and Settings** from the menu.




A script generation/update confirmation dialog box is displayed.




**2** Click the **Yes** button.
A Shape Script named **__VirtualSensorOutput_(Controller name)** and a Shape Script Sequence named **__VirtualOutputSequence_(Controller name)** are generated and displayed under 3D Visualization. If **__VirtualSensorOutput_(Controller name)** and **__VirtualOutputSequence_(Controller name)** are already present, the Shape Script and Shape Script Sequence are updated.




Refer to *8-1 Operating Procedures for a 3D Simulation* on page 8-2 for how to execute generated scripts.

### Precautions for Correct Use

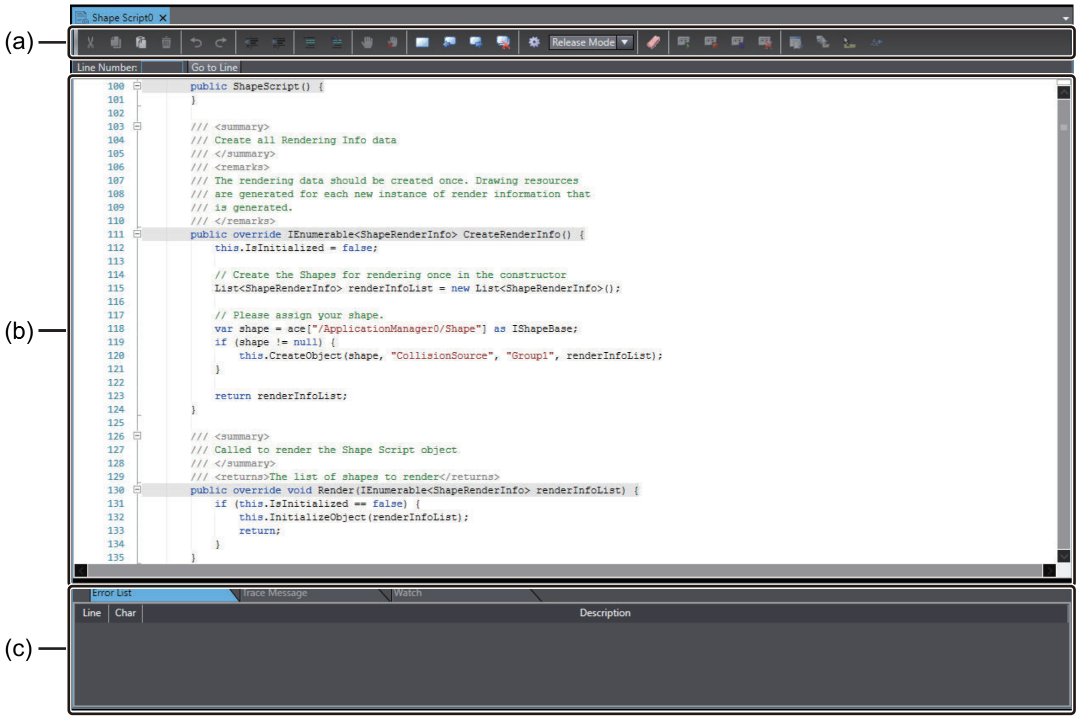
If any changes are made to the Virtual Part Detection Sensor settings, right-click and select **Create/Update Part Detection Sensor Scripts and Settings** from the menu again.
Selecting **Create/Update Part Detection Sensor Scripts and Settings** updates the registered **__VirtualSensorOutput** and **__VirtualOutputSequence** values and applies the changes to the settings.

# 6-5 Shape Script Editor

The Shape Script Editor allows you to write *Shape Scripts* that define the operations of the part and other 3D shape data.
As for the language specifications of Shape Scripts, the editor uses Microsoft's C# version 6.0. Its variables, data types, constructs, functions, operators, and so on conform to the language specifications of the C# language.
For the language specifications of the C# language, refer to the references and guides provided by Microsoft, or commercially available technical books for the C# language.
Refer to *A-1-1 Function List* on page A-2 for an explanation of functions for Shape Scripts that operate 3D shape data.

## 6-5-1 Shape Script Editor Window

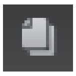
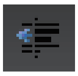
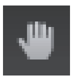
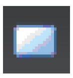
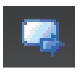
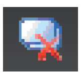
The functions of the Shape Script Editor window are given below.

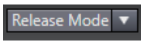
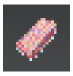
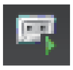
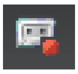
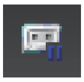
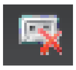
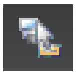
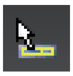
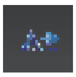



| | Item | Description |
|---|---|---|
| (a) | Toolbar | The function buttons used for script creation are displayed. Refer to *Toolbar in the Shape Script Editor* on page 6-18 for details. |
| (b) | Editor | Use this editor to create scripts. Refer to *6-5-2 Shape Script Programming* on page 6-21 for details. |
| (c) | Tab page | Errors, trace messages, and variable values are displayed. Refer to *Tab Pages of the Shape Script Editor* on page 6-20 for details. |

### Toolbar in the Shape Script Editor

You can perform the following operations with the buttons in the toolbar.

| Button | Operation (short-cut) | Function |
|---|---|---|
|  | Cut (Ctrl + X) | Cuts and saves the selected text to the clipboard. |
|  | Copy (Ctrl + C) | Copies and saves the selected text to the clipboard. |
|  | Paste (Ctrl + V) | Pastes the text saved in the clipboard to the cursor position. |
|  | Delete (Del) | Deletes the selected text. |
|  | Undo (Ctrl + Z) | Undoes the previous edit. |
|  | Redo (Ctrl + Y) | Redoes the previous undo. |
|  | Outdent (Shift + Tab) | Outdents the selected line. |
|  | Indent (Tab) | Indents the selected line. |
|  | Comment Selection | Comments out the selected line. |
|  | Uncomment Selec-tion | Uncomments the selected line. |
|  | Toggle Breakpoint | Sets or clears a breakpoint, which is a stop point of the script, at the current cursor position only in DEBUG mode. |
|  | Clear Breakpoints | Clears all breakpoints only in DEBUG mode. |
|  | Toggle Bookmark | Sets or clears bookmarks on the selected line. |
|  | Previous Bookmark | Jumps to the previous bookmark. |
|  | Next Bookmark | Jumps to the next bookmark. |
|  | Clear Bookmarks | Clears all bookmarks. |

| Button | Operation (short-cut) | Function |
|---|---|---|
|  | Compile | Checks the script for errors. Error information will be displayed on the Error List tab page. |
|   | Mode Selection | Allows the selection of **RELEASE mode** or **DEBUG mode**. Use DEBUG mode to debug scripts. |
|  | Erase Trace-Messages | Erases the information displayed on the Trace Messages tab page. Refer to *8-2-3 Trace Statement* on page 8-10 for trace messages. |
|  | Run Recorded Macro | Runs the recorded macro. Each time you click this button, a sequence of keystrokes recorded in the macro is executed. |
|  | Record Macro | Records a macro, which is a recording of a sequence of keystrokes that you performed in the Shape Script Editor. Click the button to start recording, and then click the button again to stop recording. |
|  | Pause Recording | Pauses the macro recording. To resume recording the macro, click the button again. |
|  | Cancel Recording | Cancels the macro recording. |
|  | Display an Object Member List | Displays a candidate list of C# object members. Place the cursor after an entered C# object, and then click this button. |
|  | Display Parameter Info | Displays an explanation of the C# object parameter at the cursor position. |
|  | Display Quick Info | Displays information on an error at the cursor position as a tool-tip. |
|  | Display Word Completion | Displays the candidates of C# objects from characters that are in the middle of entry. |
|   | Go to Line button | Jumps to the entered line number. |
|  | Step Over | Displayed only in DEBUG mode. Executes the script step by step, not stopping at a code in a function. |
|  | Step Into | Displayed only in DEBUG mode. A script is executed step by step and the processing will stop at a code in a function. |
|  | Go | Displayed only in DEBUG mode. Restarts the stopped script. |

## Tab Pages of the Shape Script Editor

The following tab pages are available.

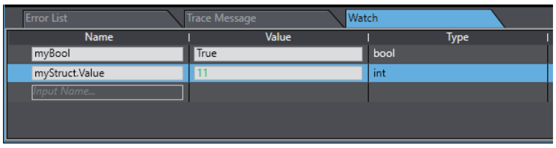
| Item | Description |
|---|---|
| Error List | Displays a list of compile errors. The line number, error location, and error message are displayed for each error. |
| Trace Message | Displays trace messages (text included in Trace.WriteLine() calls) and warning information. |
| Watch | Displays the values of the variables in the Shape Script Editor. Refer to *The **Watch** Tab Page for Shape Scripts* on page 6-21 for details. |

## ● The Watch Tab Page for Shape Scripts

The **Watch** tab page enables you to register variables used in Shape Scripts and to display and change the values of variables in a Shape Script whose execution is paused.

**1** In DEBUG mode, execute a Shape Script with breakpoints set in it or perform the step execution to pause the execution of the Shape Script.

**2** In the **Watch** tab page, register a variable name.




The value of the variable is displayed in the **Value** column.

### Additional Information

- Variable values can be displayed and changed only when the execution of a Shape Script is paused at a breakpoint or during step execution in DEBUG mode. In other situations, the values are grayed out.
- To change the value of a variable, enter a value directly in the **Value** column and press the Enter key.

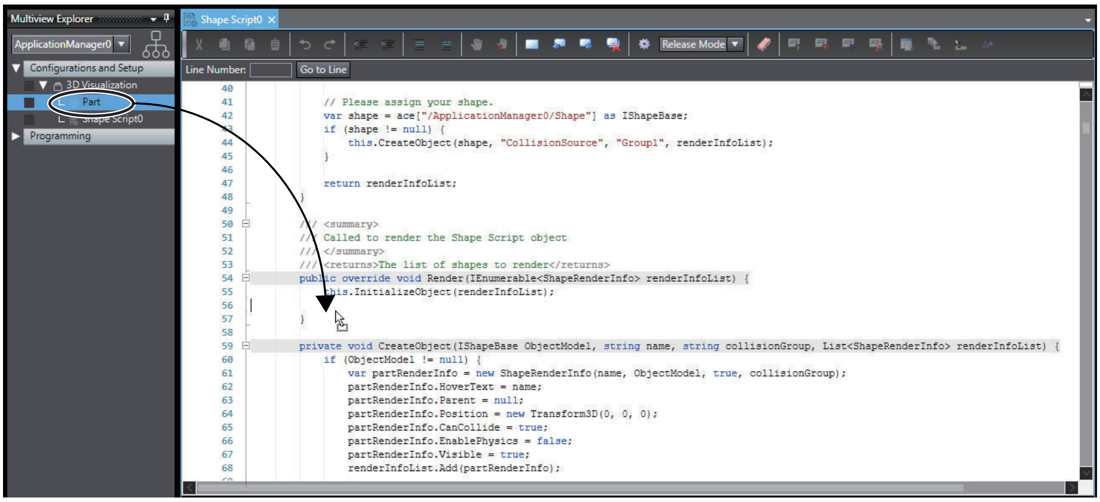
## 6-5-2 Shape Script Programming

The operating procedures for programming with the Shape Script Editor are described below.
In the Shape Script Editor, you can write a script by dragging-and-dropping data from the Multiview Explorer or functions registered in the Toolbox.

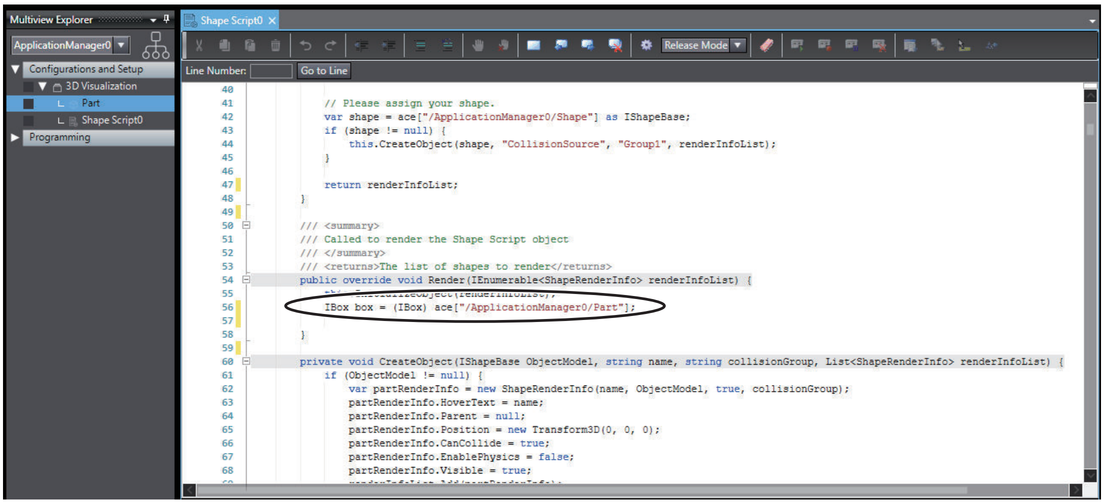
### Creating a Variable Declaration for Variables that Represent 3D Shape Data in a Shape Script

To use variables that represent 3D shape data such as the part in a Shape Script, you must declare the variables.
You can drag a 3D shape data item from the Multiview Explorer into the Shape Script Editor to create a variable declaration.
Here, the operating procedure to create a variable declaration for the part is given as an example.

**1** Select **Part** under **3D Visualization** under **Configurations and Setup** in the Multiview Explorer and drag it to the point of insertion.



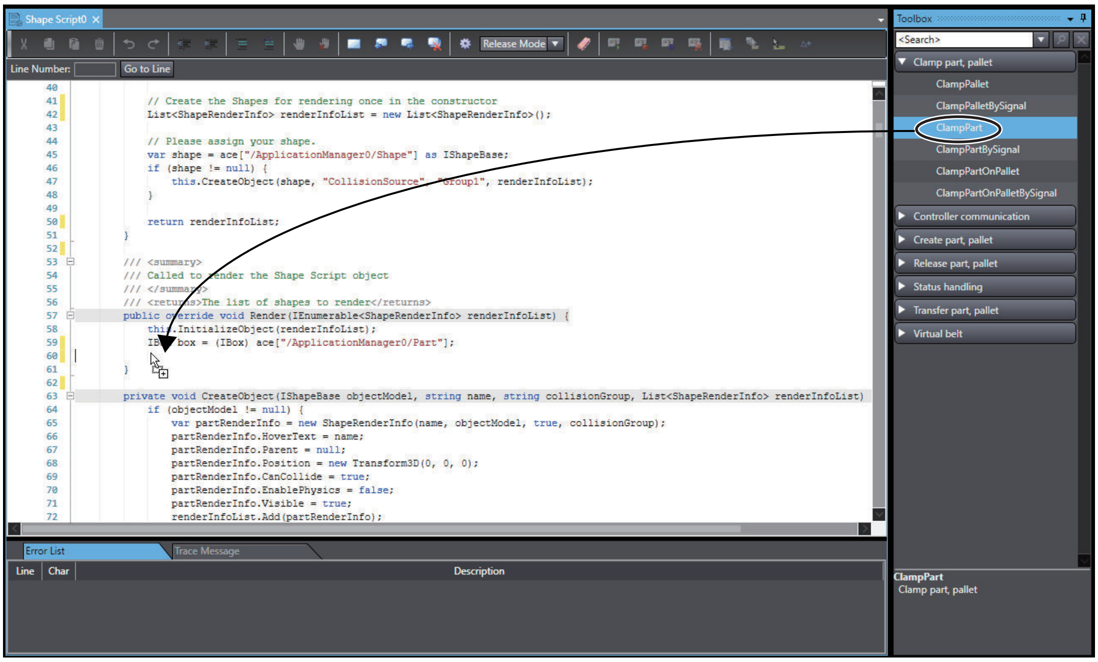
The variable declaration for the part is inserted into the point at which you drop it.



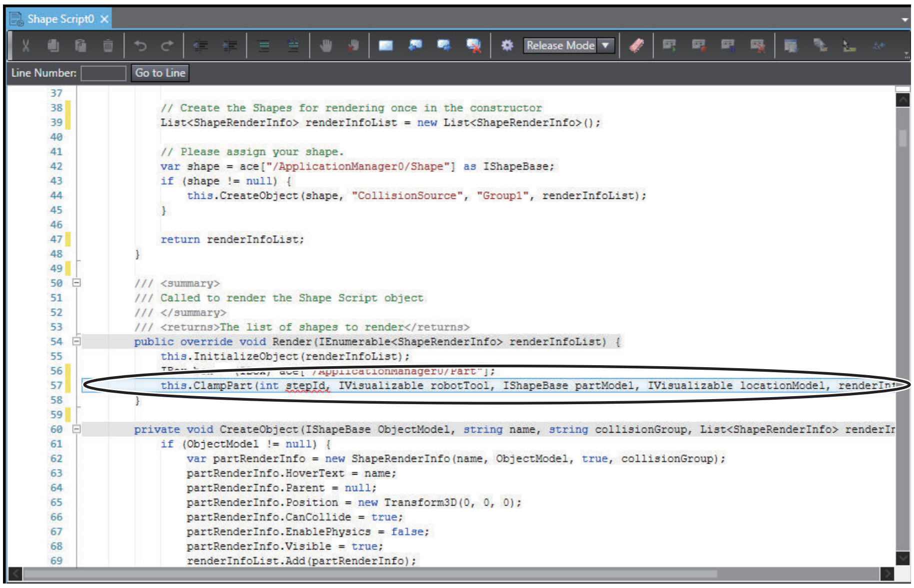
## Inserting a Function from the Toolbox into a Shape Script

Insert a function from the Toolbox to a Shape Script.
Here, the operating procedure to insert a function that processes the clamping of the part is given as an example.

**1** Select **ClampPart** under **Clamp part, pallet** in the Toolbox and drag it to the point of insertion.




The function is inserted into the point at which you drop it.




In the inserted function, specify appropriate variables for arguments.

This completes the insertion of the function.

Refer to *A-1-1 Function List* on page A-2 for details on the functions for Shape Scripts provided in the Sysmac Studio.

# 6-6 Shape Script Functions

Shape Script Functions are libraries of C# language functions for Shape Scripts.
There are two types of Shape Script Functions as follows.

1. Default Functions
   This is a library of functions that are used as standard in Shape Scripts. When you create a Shape Script, **Default Functions** is automatically created. In addition to the automatically created **Default Functions**, you can also register any Default Functions. Upgrading the Sysmac Studio to a higher version may add new functions to the Default Functions or enhance the existing functions. These changes are managed by the version of the Default Functions. The version of the Default Functions to be registered is the latest version available at the time.
   Refer to *A-1 Functions Used in Shape Scripts* on page A-2 for information on functions used in Shape Scripts.
2. User Functions
   This is a library of user-defined functions. You can register more than one set of User Functions.

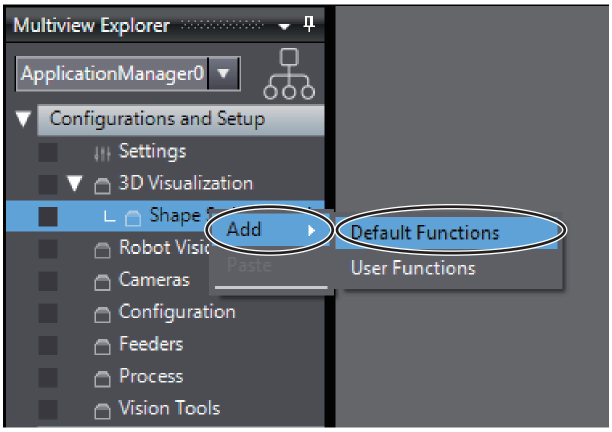
## 6-6-1 Adding Shape Script Functions

This section describes the procedure for registering Default Functions and User Functions.

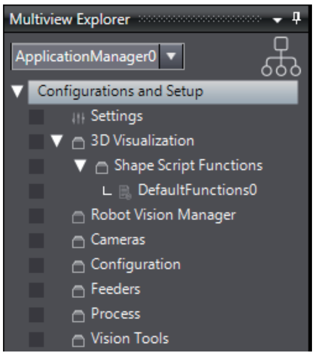
### Adding Default Functions

Use the following procedure to add a set of Default Functions.

**1** Right-click **Shape Script Functions** under **Configurations and Setup** – **3D Visualization** in the Multiview Explorer and select **Add** – **Default Functions** from the menu.



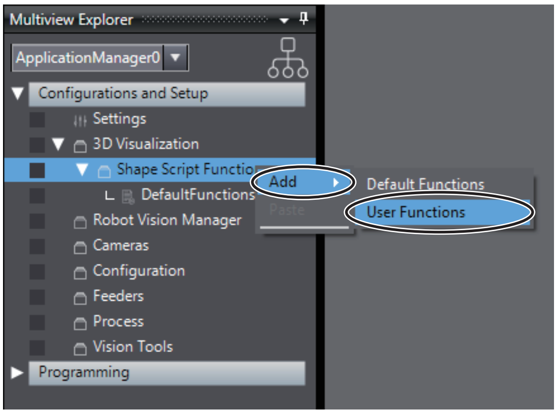

**DefaultFunctions0** is registered under **Shape Script Functions** under **3D Visualization**.



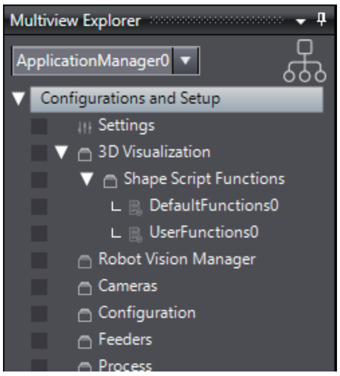

## Adding User Functions

This section describes the procedure for adding a set of User Functions.

1. Right-click **Shape Script Functions** under **Configurations and Setup** – **3D Visualization** in the Multiview Explorer and select **Add** – **User Functions** from the menu.




**UserFunctions0** is registered under **Shape Script Functions** under **3D Visualization**.




## 6-6-2 Referencing Default Functions and User Functions

This section describes how to reference Default Functions and User Functions from a Shape Script.

## Referencing Default Functions

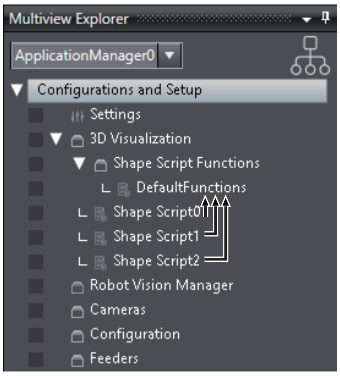
The procedure to reference the Default Functions from a Shape Script is given below.
Added Shape Scripts automatically reference any function in the **Default Functions**.

Default functions name.Function name (Arguments);

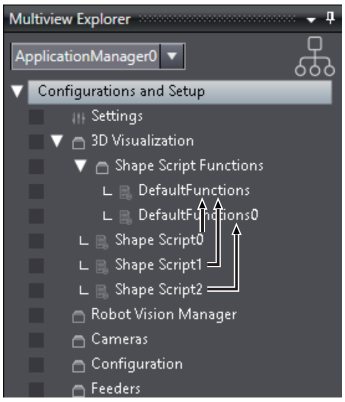
Example:

```
DefaultFunctions.CreateObject(this, shape, "CollisionSource", "Group1", renderInfoL
ist);
```

If there are multiple Shape Scripts, you can reference functions in the **Default Functions** to use them commonly in more than one Shape Script.




You can also specify functions in any Default Functions other than the automatically created **Default Functions** to reference them in the Shape Scripts.




## Referencing User Functions

The procedure to reference User Functions from a Shape Script is given below.
Specify any function in an added set of User Functions as follows.

User functions name.Function name (Arguments);

Example:

```
UserFunctions0.MySelectPalletFunction(this, stepId, controllerName, variableName, p
alletModel);
```

You can define functions that are commonly used in more than one Shape Script.

# 7

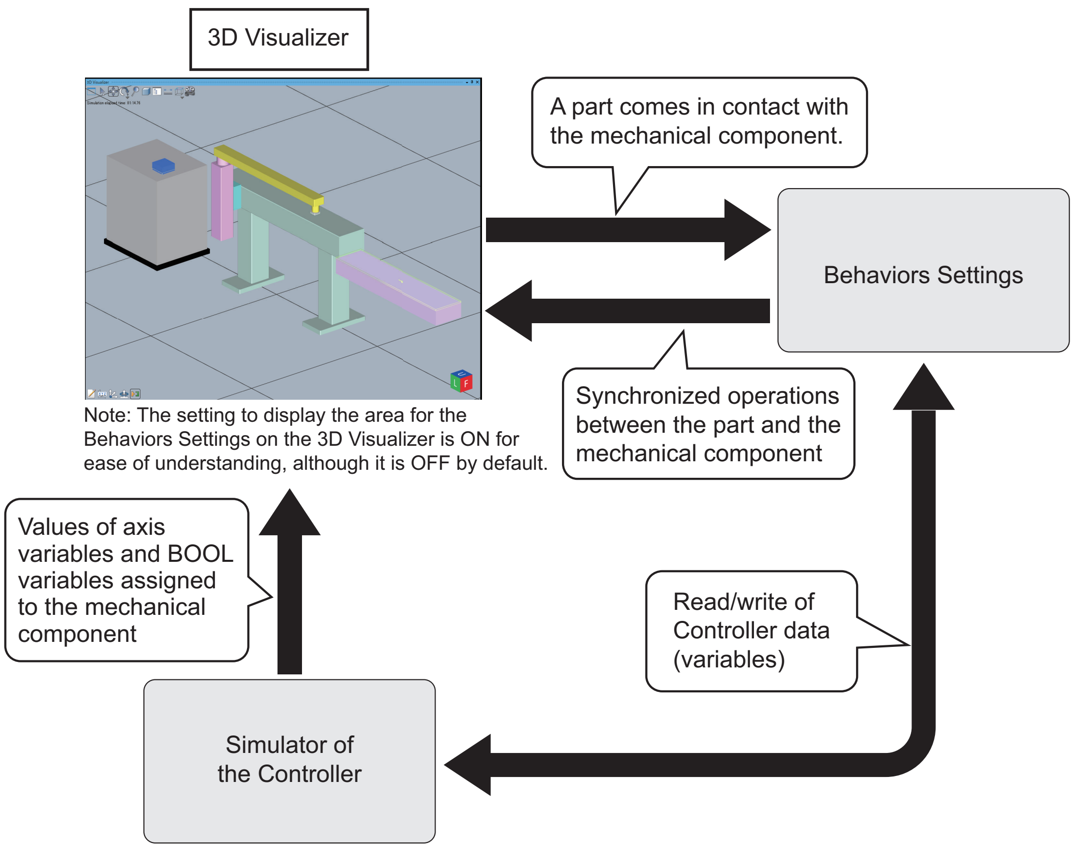
# Easy Part Simulation Configuration

This section describes a function that achieves the operations of the part in 3D simulation only by settings.

# 7-1 Overview of Easy Part Simulation Configuration

This section describes the settings to reproduce the motions of the part in 3D simulation.

Easy Part Simulation Configuration enable the motions of the part to be synchronized with those of the mechanical component in the virtual equipment model displayed in the 3D Visualizer. Unlike the simulation based on Shape Scripts described in *Section 6 Creating Settings and Scripts for Operating the 3D Shape Data* on page 6-1, this function realizes simulation of the part without programming.
For each mechanical component that will be in contact with the part, configure the operation settings of the part that will move in synchronization with the mechanical component. These operation settings of the part are called “Behaviors Settings.”
When you execute a Controller program during a Controller simulation, the values of axis variables and BOOL variables assigned to the mechanical component change. Then, in the 3D Visualizer, the mechanical component and 3D shape data move according to the changes in the values of the variables.
However, during a simulation using Easy Part Simulation Configuration, the part moves based on the Behaviors Settings configured for the mechanical component that is in contact with the part. This enables the 3D Visualizer to display synchronized movement of the part with the operation of the mechanical component.



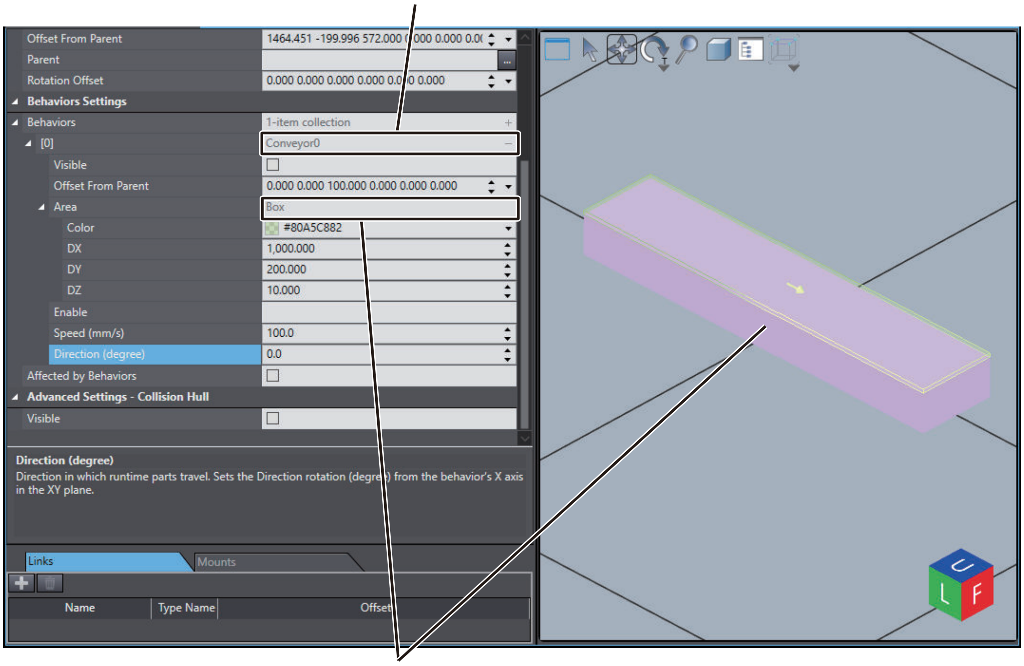
# 7-2 Setting Mechanical Component

Configure the operation settings for a mechanical component.
Display the Mechanical Component tab page and, on the **Parameter Settings** screen, check that the target Controller and variables are assigned.
Refer to *4-3-5 Mechanical Component Settings* on page 4-28 for details on the settings.

# 7-3 Behaviors Settings

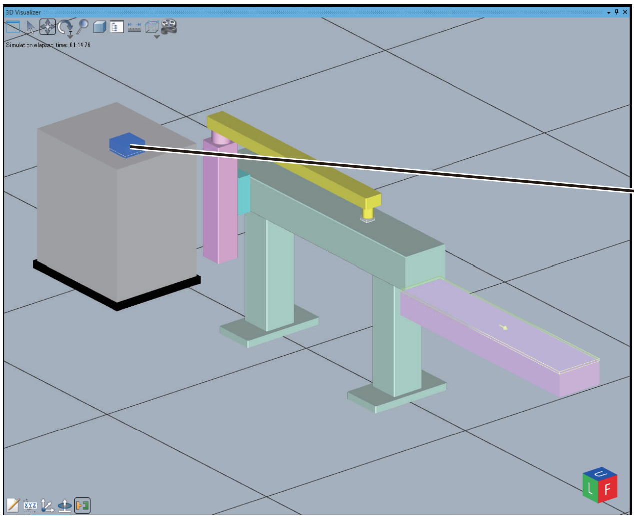
Behaviors Settings define how a part should be operated when it comes in contact with a mechanical component or other 3D shape data.
The following is an example of 3D shape data for part-carrying operation.
In the setup tab page for the 3D shape data, there is an item named Behaviors Settings, under which you configure the area and other settings to specify where and how the part should be operated when it comes in contact with the 3D shape data. Refer to *7-3-1 List of Behaviors Settings* on page 7-6 for details on Behaviors Settings.

Setting for how parts should be operated (e.g., Conveyor)



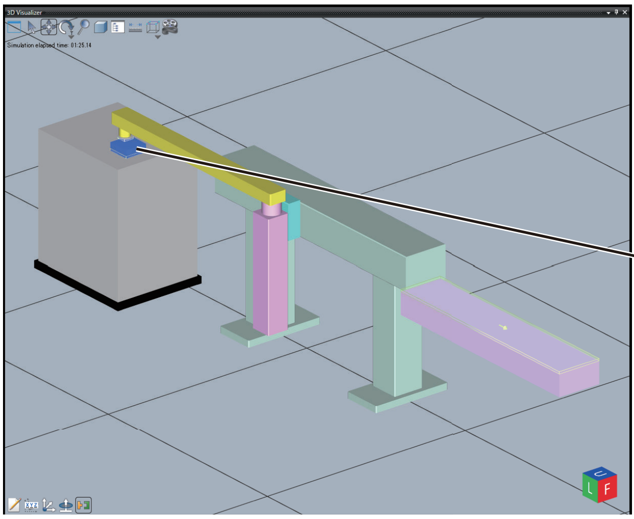

Area where a part should be operated when it comes in contact with the 3D shape data

In simulation using Easy Part Simulation Configuration, you can simulate how a part will move in a virtual equipment model.
The following is an example of a sequence of operations: a box simulating the part exists on a box simulating the loader; the XYθ slider picks the part by the tip, moves, and releases the part on a box simulating the conveyor; and the conveyor carries the part.

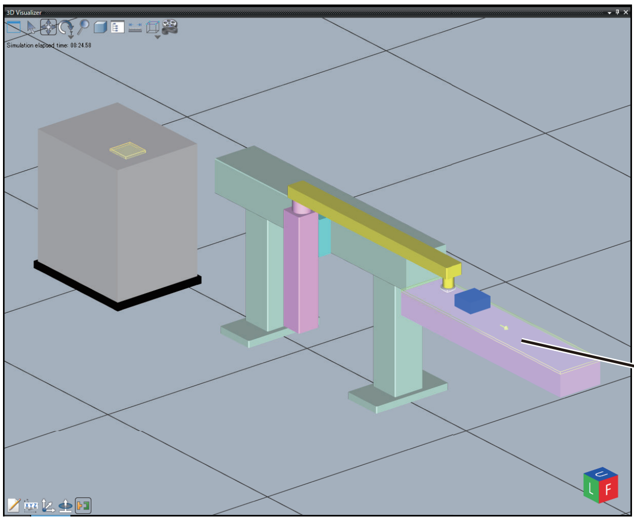


A part exists.

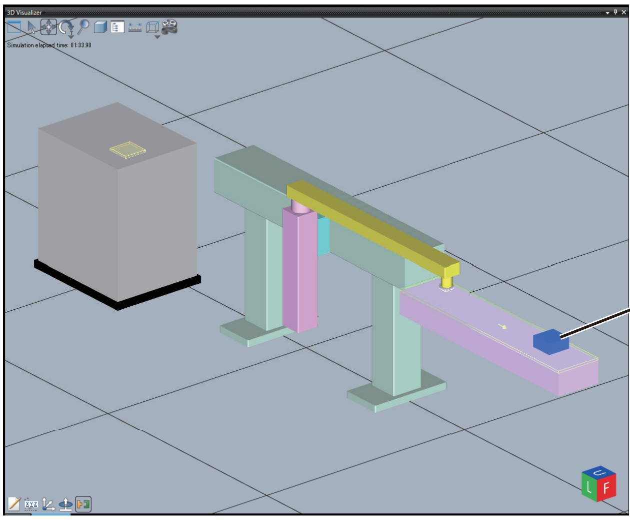
Note: The setting to display the area for the Behaviors Settings on the 3D Visualizer is ON for ease of understanding, although it is OFF by default.



The slider picks a box simulating the part by the tip.



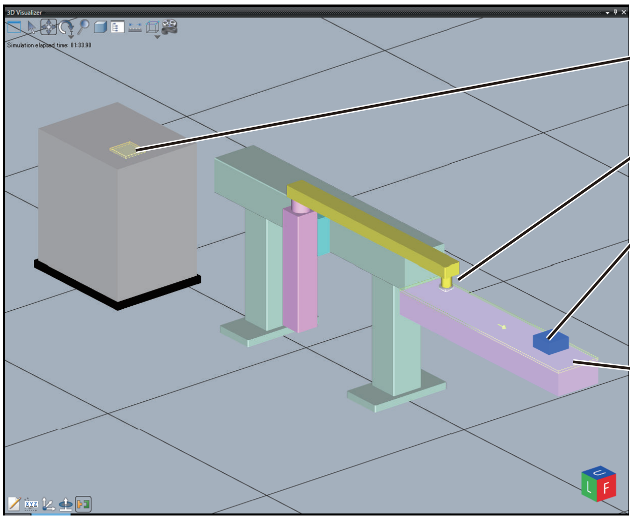
The slider moves and releases the part on a box simulating the conveyor.



A box simulating the conveyor carries the part.

In this virtual equipment model, consider what motions the 3D Visualization data will give to the part that comes in contact with it.

The operations involved in this example are: (1) the loader loads a part; (2) the XYθ slider picks the part by the tip; (3) the conveyor carries the part; and (4) the unloader unloads the part.



(1) The loader loads the part.

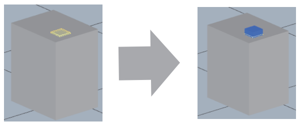
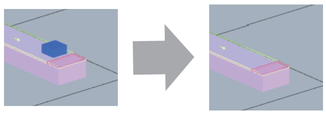
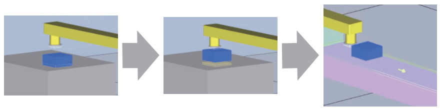
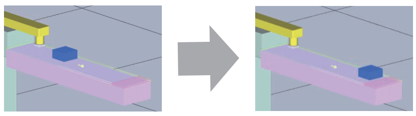
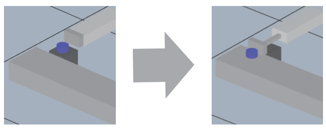
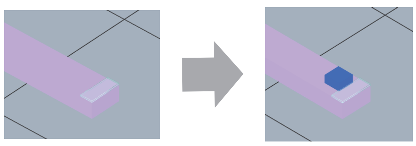
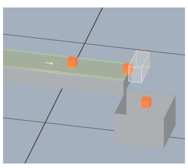
(2) The XYθ slider picks the part by the tip and carries the part.

(3) The conveyor carries the part.

(4) The unloader unloads the part.

## 7-3-1 List of Behaviors Settings

The table below shows the behaviors settings.

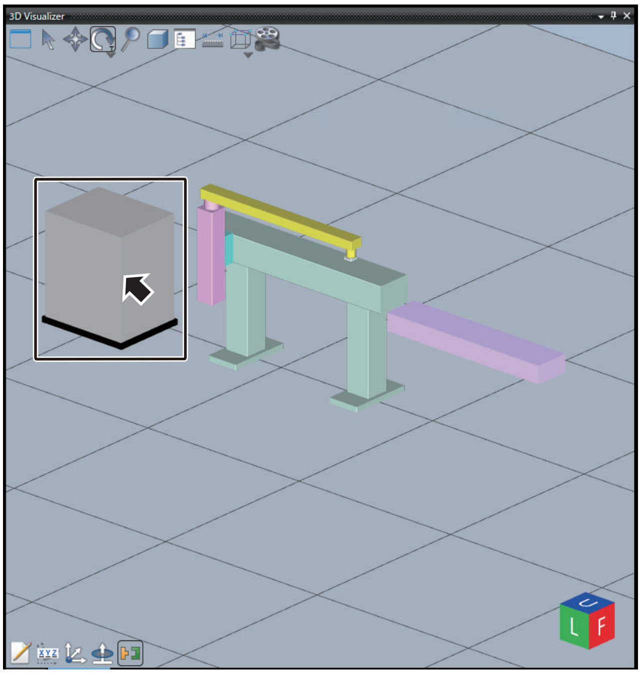
| Category | Behaviors Settings | Operation image | Description |
|---|---|---|---|
| Part loading and hiding | Loader |  | Displays the specified 3D shape data as the part to be configured in the Behaviors Settings.<br>You can set parameters to change the loading interval and load the part in a random position.<br>You can control the loading timing of part by specifying a BOOL variable in the Controller. |
| | Unloader |  | Hides the part when it comes in contact with the specified area. |
| Part carrying | Clamp |  | Joins the part that comes in contact with the 3D shape data to simulate the picking motion of the mechanical component.<br>The part will be released from the 3D shape data when one of the following conditions is met.<br>• The part comes in contact with the area set in the Behaviors Settings for other 3D shape data (e.g., the area with the setting of Clamp in the Behaviors Settings for the 3D shape data).<br>• The BOOL variable in the specified Controller changes to FALSE. |
| | Conveyor |  | Carries the part in the specified direction and at the specified speed.<br>You can control the start and stop of carrying the part by specifying a BOOL variable in the Controller. |
| | Push |  | Pushes the part. |
| Part detection | Sensor |  | Changes the specified Controller variable (BOOL) to TRUE when the part comes in contact with the specified area.<br>No operation will be given to the part. |
| Physics | Physics |  | Causes the part to move based on a simple physics simulation when the part comes in contact with the specified area.<br>You can simulate falling and sliding. |

## 7-3-2 Loader

Use the following procedure to set *Loader* in the Behaviors Settings for 3D shape data registered in the Application Manager, such as a mechanical component.

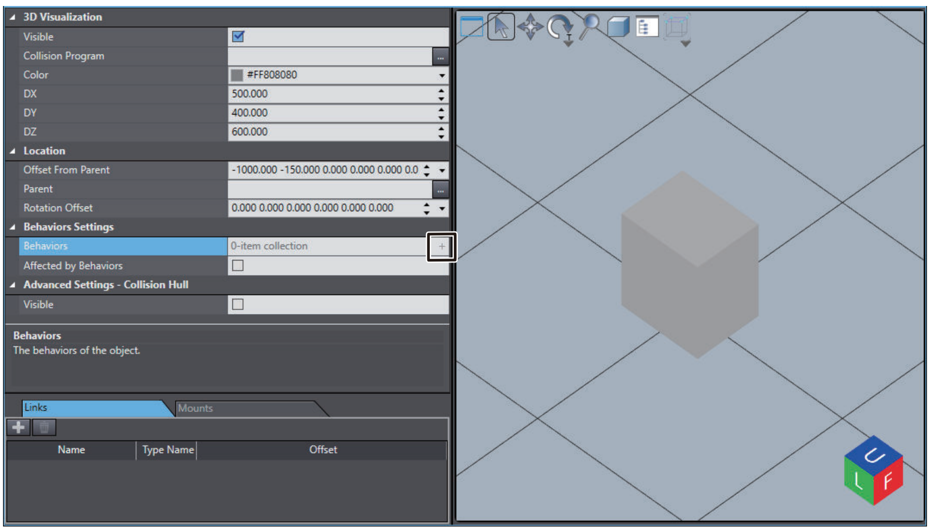
### Operating Procedure

**1** In the 3D Visualizer, double-click the 3D shape data for which to set *Loader*.



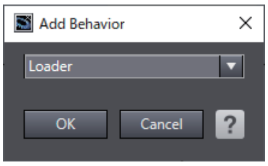

The setup tab page for the 3D shape data is displayed in the edit pane.

**2** In the setup tab page for the 3D shape data, click the button on the right side of **Behaviors** under **Behaviors Settings**. For mechanical components, **Behaviors Settings** are located in the Mechanical Component common settings.

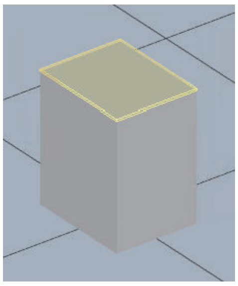


The **Add Behavior** dialog box is displayed.

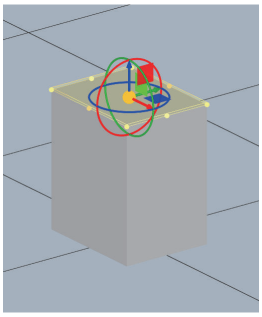
**3** Select **Loader**, and then click the **OK** button.



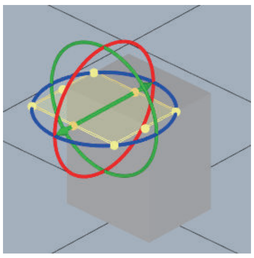

**4** In the 3D editing area, set the area in which to load the part.



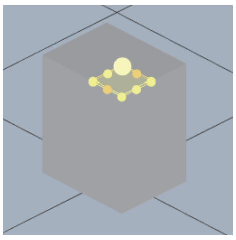
Click the center point of the area displayed on the 3D shape data to display the **Move** and **Rotate** icons.



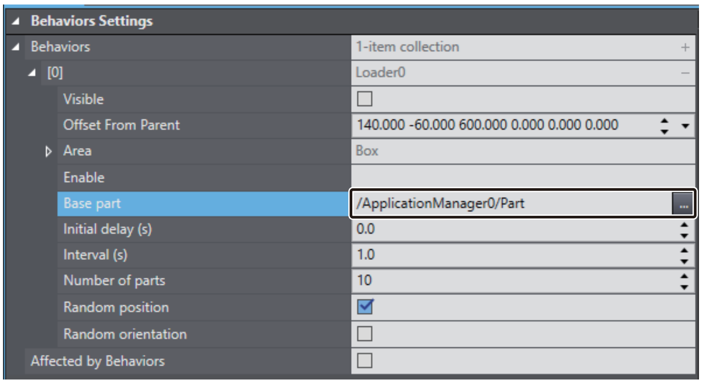
Drag each icon to adjust the position and pose of the area.



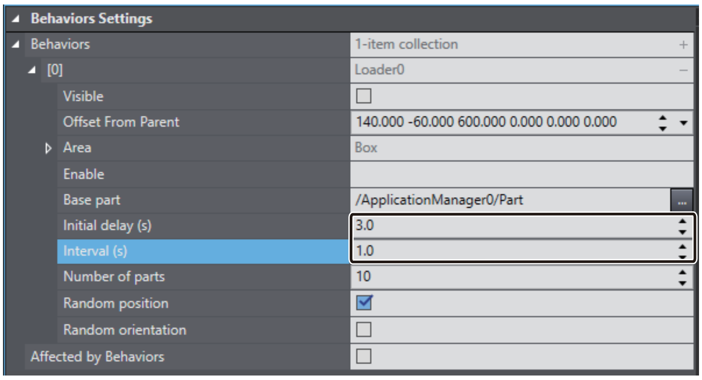
Drag the dots displayed in the area set on the 3D shape data to adjust the size of the area.



**5** Set the 3D shape data to display as the part. In **Base part** in the **Behaviors Settings**, select the name of the 3D shape data registered with the 3D Visualization object.

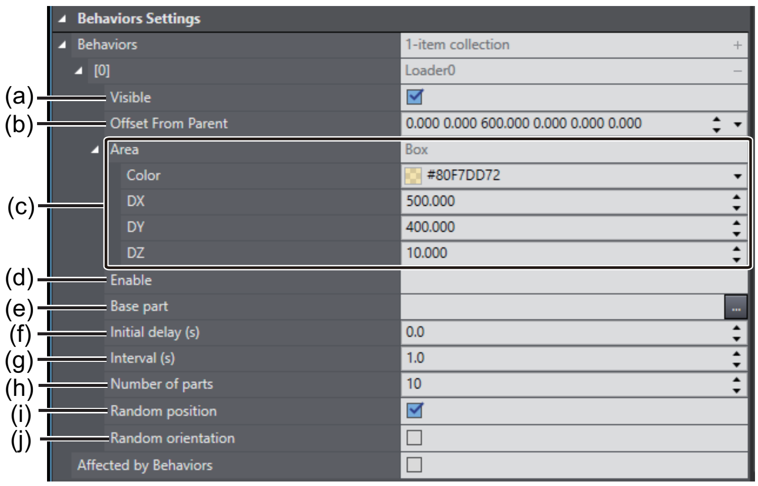



**6** Set the conditions for generating the part. Here, set the time from the start of the part simulation until the first part is loaded and the time interval for loading the part.




## Setting Items of *Loader*

The *Loader* Behaviors Settings have the following setting items.



| | Item | | Description | Set value | Initial value |
|---|---|---|---|---|---|
| (a) | Visible | | Shows or hides the setting area set in the 3D Visualizer. | Selected or unselected | Unselected |
| (b) | Offset From Parent | | Sets the position and pose of the area.<br>Uses the coordinates in the local coordinate system of the 3D shape data.<br>The location elements are X, Y, Z, y (yaw), p (pitch), and r (roll), from left to right. | X, Y, Z: For each, -1,000,000.000 to 1,000,000.000<br>y, r: For each, -180.000 to 180.000<br>p: 0.000 to 180.000 | Depending on the operation at setting the area |
| (c) | Area | Color | Sets the color of the area. | Transparency (0 to #FF) and RGB (0 to #FF, 0 to #FF, 0 to #FF) | #80F7DD72 |
| | | DX | Sets the X-axis length of the area. (Unit: mm) | 0.1 to 10,000 | Depending on the operation at setting the area |
| | | DY | Sets the Y-axis length of the area. (Unit: mm) | 0.1 to 10,000 | Depending on the operation at setting the area |
| | | DZ | Sets the Z-axis length of the area. (Unit: mm) | 0.1 to 10,000 | 10 |
| (d) | Enable | | Enables or disables the Behaviors Settings for *Loader*.<br>You can control the timing of loading a part using a Controller variable by specifying the name of a BOOL variable in the Controller.<br>Blank: The *Loader* Behaviors Settings are always enabled.<br>True: The *Loader* Behaviors Settings are always enabled.<br>False: The *Loader* Behaviors Settings are always disabled.<br>ControllerName.ControllerVariableName(BOOL): The *Loader* Behaviors Settings are enabled when the Controller variable is TRUE.<br>Example: The following setting loads the part every time the variable *PartGenerate* of the Controller name *new_Controller0* changes to True.<br>Enable: new_Controller0.PartGenerate<br>Initial delay (s): 0<br>Interval (s): 0<br>Number of parts: 1 | True<br>False<br>ControllerName.ControllerVariableName(BOOL) | Blank |
| (e) | Base part | | Specifies the 3D shape data to load as the part. | Name of 3D shape data | Blank |
| (f) | Initial delay (s) | | Sets the duration from when the part simulation is started and the *Loader* Behaviors Settings are enabled until the first part is loaded. (Unit: s) | 0.0 to 1,000 | 0 |
| (g) | Interval (s) | | Sets the time interval for loading the part. (Unit: s) | 0.1 to 100 | 1 |
| (h) | Number of parts | | Sets the number of parts to load. | 1 to 1,000 | 10 |

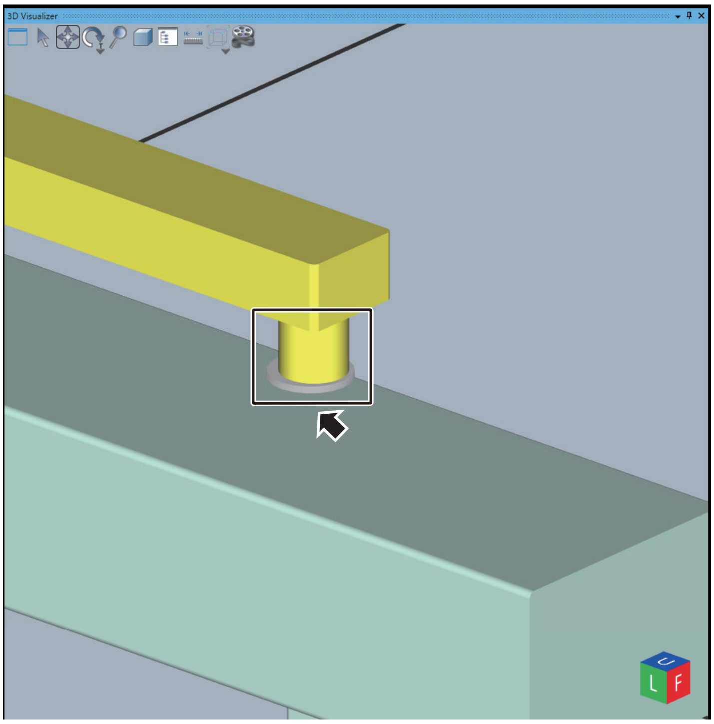
| | Item | Description | Set value | Initial value |
|---|---|---|---|---|
| (i) | Random position | Sets this to load the part in a random position within the area. | Selected or unselected | Selected |
| (j) | Random orientation | Sets this to load the part in a random pose. | Selected or unselected | Unselected |

## 7-3-3 Clamp

Use the following procedure to set *Clamp* in the Behaviors Settings for 3D shape data that is registered in the Application Manager, such as a mechanical component.

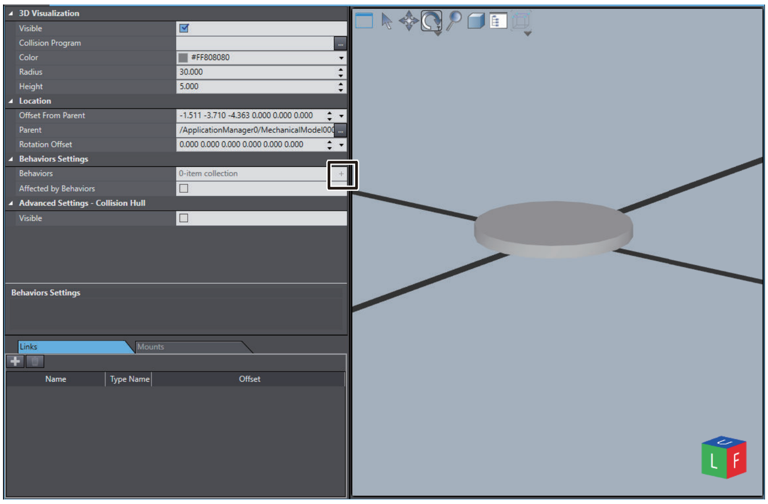
### Operating Procedure

**1** In the 3D Visualizer, double-click the 3D shape data for which to set *Clamp*.



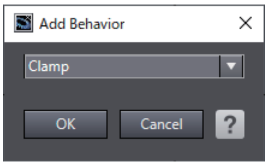

The setup tab page for the 3D shape data is displayed in the edit pane.

**2** In the setup tab page for the 3D shape data, click the button on the right side of **Behaviors** under **Behaviors Settings**. For mechanical components, **Behaviors Settings** are located in the Mechanical Component common settings.

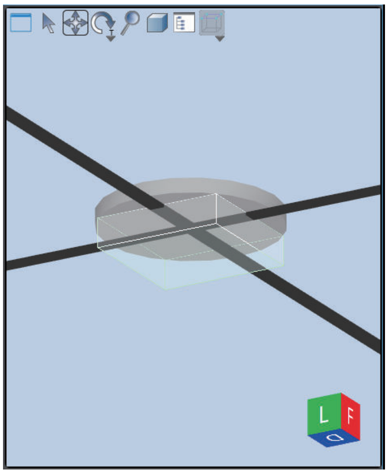



The **Add Behavior** dialog box is displayed.

**3** Select **Clamp**, and then click the **OK** button.

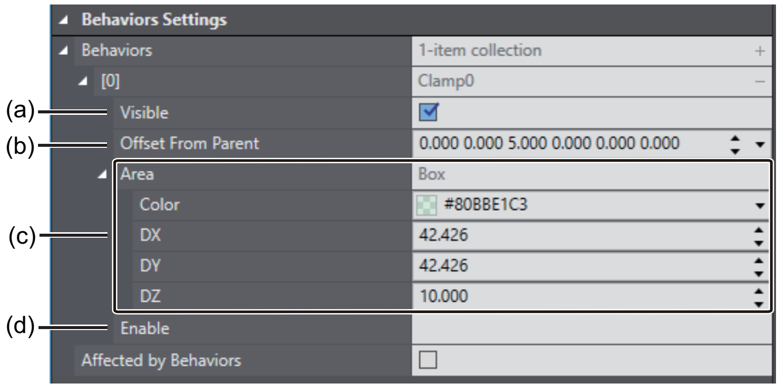



**4** Set the area in which to pick the part on the 3D shape data displayed on the right side of the 3D shape data edit pane.




## Setting Items of *Clamp*

The *Clamp* Behaviors Settings have the following setting items.




| | Item | | Description | Set value | Initial value |
|---|---|---|---|---|---|
| (a) | Visible | | Shows or hides the setting area in the 3D Visualizer. | Selected or unselected | Unselected |
| (b) | Offset From Parent | | Sets the position and pose of the area.<br>Uses the coordinates in the local coordinate system of the 3D shape data.<br>The location elements are X, Y, Z, y (yaw), p (pitch), and r (roll), from left to right. | X, Y, Z: For each, -1,000,000.000 to 1,000,000.000<br>y, r: For each, -180.000 to 180.000<br>p: 0.000 to 180.000 | Depending on the operation at setting the area |
| (c) | Area | Color | Sets the color of the area. | Transparency (0 to #FF) and RGB (0 to #FF, 0 to #FF, 0 to #FF) | #80BBE1C3 |
| | | DX | Sets the X-axis length of the area. (Unit: mm) | 0.1 to 10,000 | Depending on the operation at setting the area |
| | | DY | Sets the Y-axis length of the area. (Unit: mm) | 0.1 to 10,000 | Depending on the operation at setting the area |
| | | DZ | Sets the Z-axis length of the area. (Unit: mm) | 0.1 to 10,000 | 10 |

| | Item | Description | Set value | Initial value |
|---|---|---|---|---|
| (d) | Enable | Optionally enables or disables the *Clamp* Behaviors Settings.<br>You can control the timing of part clamping using a Controller variable by specifying the name of a BOOL variable in the Controller.<br>Blank: The *Clamp* Behaviors Settings are always enabled.<br>True: The *Clamp* Behaviors Settings are always enabled.<br>False: The *Clamp* Behaviors Settings are always disabled.<br>ControllerName.ControllerVariableName(BOOL): The *Clamp* Behaviors Settings are enabled when the Controller variable is True.<br>Example: If the Controller variable changes to True when the 3D shape data is in contact with the part, the 3D shape data joins the part. Then, if the Controller variable changes to False, the 3D shape data releases the part. | True<br>False<br>ControllerName.ControllerVariableName(BOOL) | Blank |

**Additional Information**

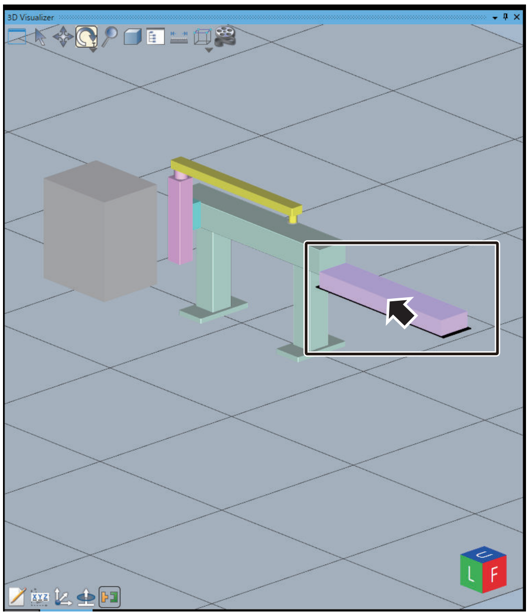
When 3D shape data joins a part based on the *Clamp* Behaviors Settings, the part is not joined with 3D shape data that has the *Conveyor* Behaviors Settings even if the part comes in contact with the 3D shape data that has the *Conveyor* Behaviors Settings. In such a case, prepare a Controller variable that controls the picking and releasing of the part as in the actual control and assign the Controller variable to the *Enable* setting in the *Clamp* Behaviors Settings. This will enable the *Clamp* Behaviors Settings when that Controller variable changes to TRUE, which allows the picking of the part. When the Controller variable changes to FALSE, the 3D shape data with the *Conveyor* Behaviors Settings controls the part.

## 7-3-4 Conveyor

Use the following procedure to set *Conveyor* in the Behaviors Settings for 3D shape data that is registered in the Application Manager, such as a mechanical component.

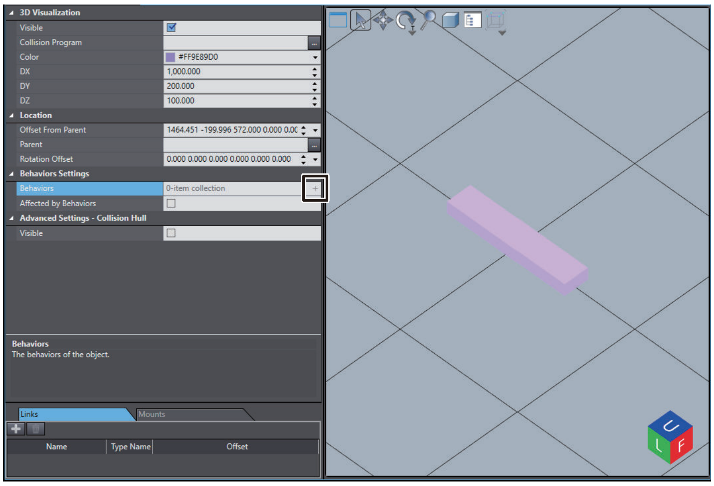
### Operating Procedure

**1** In the 3D Visualizer, double-click the 3D shape data for which to set *Conveyor*.



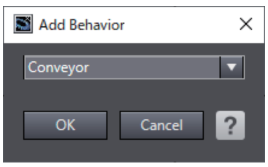

The setup tab page for the 3D shape data is displayed in the edit pane.

**2** In the setup tab page for the 3D shape data, click the button on the right side of **Behaviors** under **Behaviors Settings**. For mechanical components, **Behaviors Settings** are located in the Mechanical Component common settings.

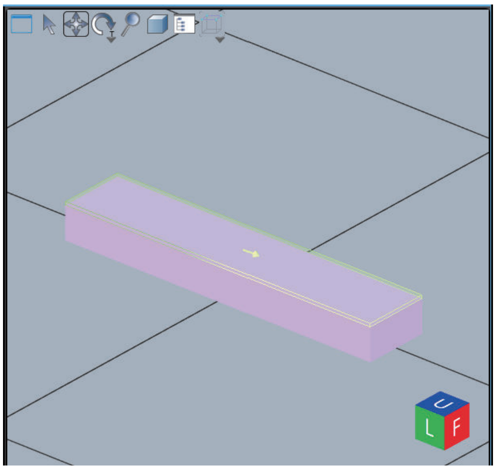



The **Add Behavior** dialog box is displayed.

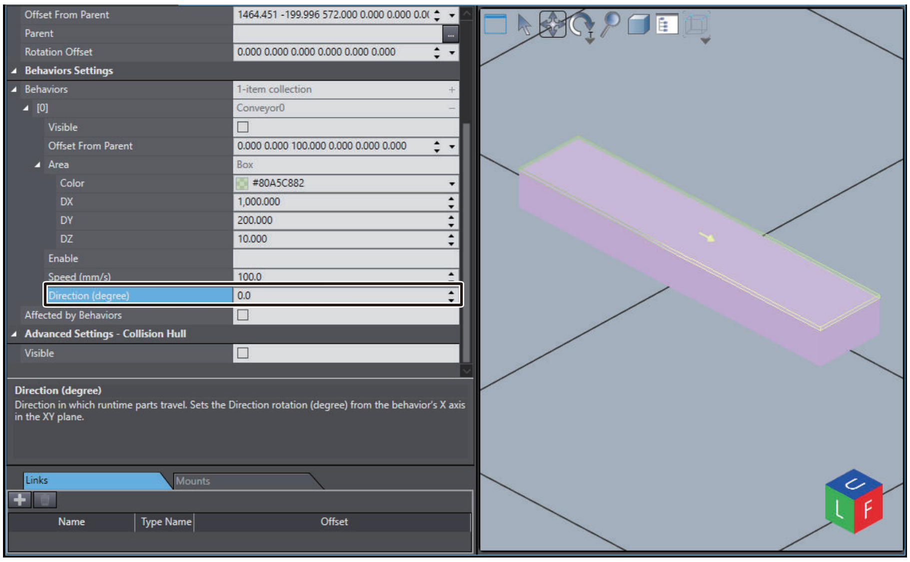
**3** Select **Conveyor**, and then click the **OK** button.



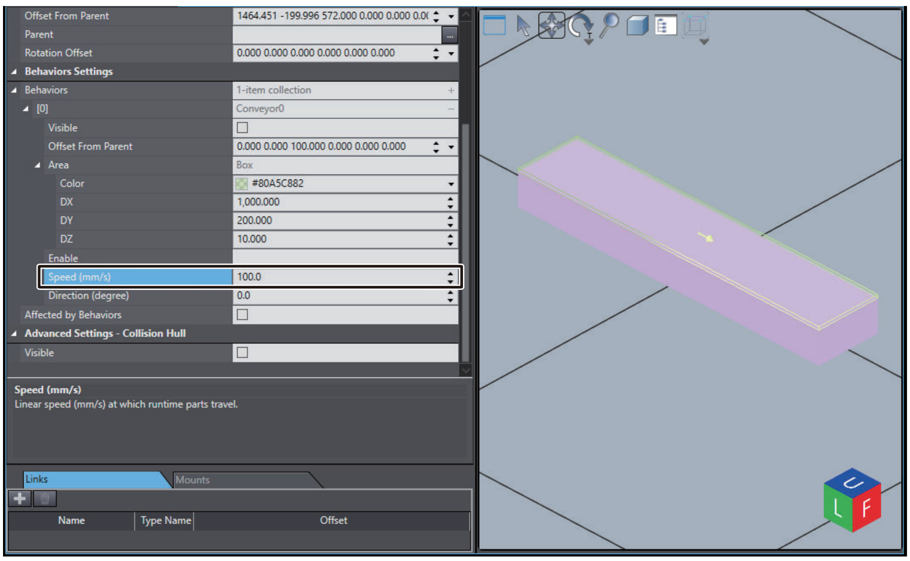

**4** Set the area in which to carry the part on the 3D shape data displayed on the right side of the 3D shape data edit pane.



**5** Set the direction in which to carry the part.

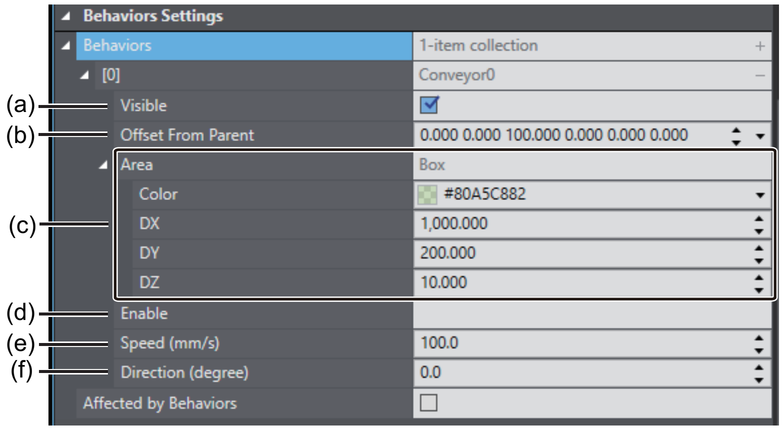



**6** Set the speed at which to carry the part.



## Setting Items of *Conveyor*

The *Conveyor* Behaviors Settings have the following setting items.




| | Item | | Description | Set value | Initial value |
|---|---|---|---|---|---|
| (a) | Visible | | Shows or hides the setting area set in the 3D Visualizer. | Selected or unselected | Unselected |
| (b) | Offset From Parent | | Sets the position and pose of the area. Uses the coordinates in the local coordinate system of the 3D shape data. The location elements are X, Y, Z, y (yaw), p (pitch), and r (roll), from left to right. | X, Y, Z: For each, -1,000,000.000 to 1,000,000.000<br>y, r: For each, -180.000 to 180.000<br>p: 0.000 to 180.000 | Depending on the operation at setting the area |
| (c) | Area | Color | Sets the color of the area. | Transparency (0 to #FF) and RGB (0 to #FF, 0 to #FF, 0 to #FF) | #80A5C882 |
| | | DX | Sets the X-axis length of the area. (Unit: mm) | 0.1 to 10,000 | Depending on the operation at setting the area |
| | | DY | Sets the Y-axis length of the area. (Unit: mm) | 0.1 to 10,000 | Depending on the operation at setting the area |
| | | DZ | Sets the Z-axis length of the area. (Unit: mm) | 0.1 to 10,000 | 10 |

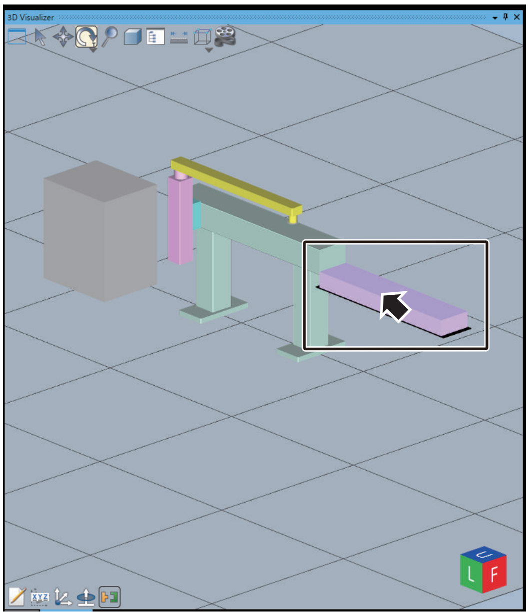
| | Item | Description | Set value | Initial value |
|---|---|---|---|---|
| (d) | Enable | Optionally enables or disables the *Conveyor* Behaviors Settings.<br>You can control the timing of part carrying using a Controller variable by specifying the BOOL variable name of the Controller.<br>Blank: The *Conveyor* Behaviors Settings are always enabled.<br>True: The *Conveyor* Behaviors Settings are always enabled.<br>False: The *Conveyor* Behaviors Settings are always disabled.<br>ControllerName.ControllerVariable-Name(BOOL): The Behaviors Settings are enabled when the Controller variable is True.<br>Example: If the Controller variable changes to True when the 3D shape data is in contact with the part, the part is carried. Then, if the Controller variable changes to False, the carrying of the part stops. | True<br>False<br>ControllerName.Con-trollerVariable-Name(BOOL) | Blank |
| (e) | Speed (mm/s) | Sets the speed at which to carry the part. (Unit: mm/s) | 1 to 10,000 | 100 |
| (f) | Direction (de-gree) | Sets the direction in which to carry the part, with the direction along the X-axis of the area being 0 degrees. | -180 to 180 | 0 |

## 7-3-5 Unloader

Use the following procedure to set *Unloader* in the Behaviors Settings for 3D shape data that is registered in the Application Manager, such as a mechanical component.

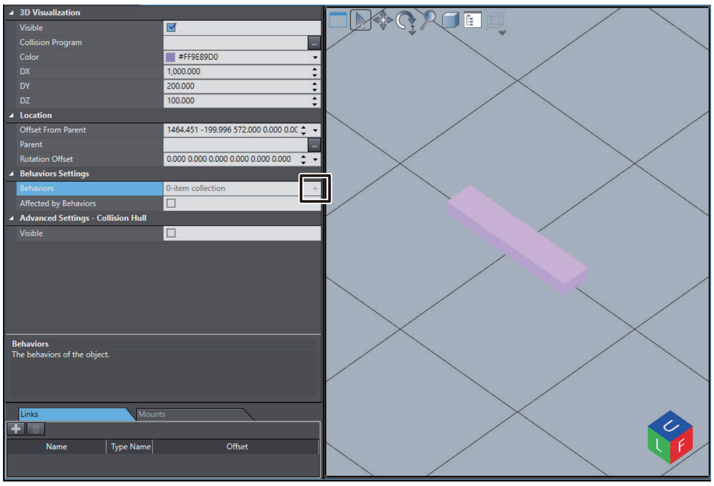
### Operating Procedure

**1** In the 3D Visualizer, double-click the 3D shape data for which to set *Unloader*.



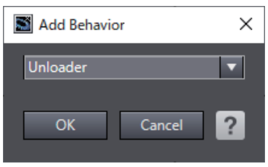
The setup tab page for the 3D shape data is displayed in the edit pane.

**2** In the setup tab page for the 3D shape data, click the button on the right side of **Behaviors** under **Behaviors Settings**. For mechanical components, **Behaviors Settings** are located in the Mechanical Component common settings.

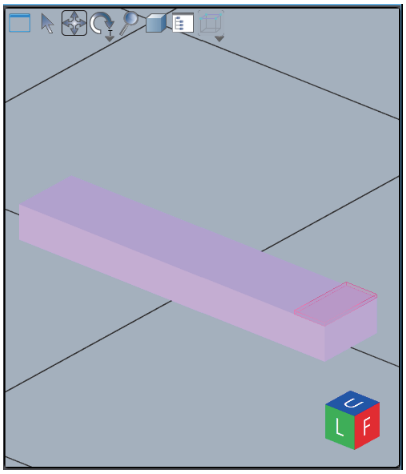



The **Add Behavior** dialog box is displayed.

**3** Select **Unloader**, and then click the **OK** button.

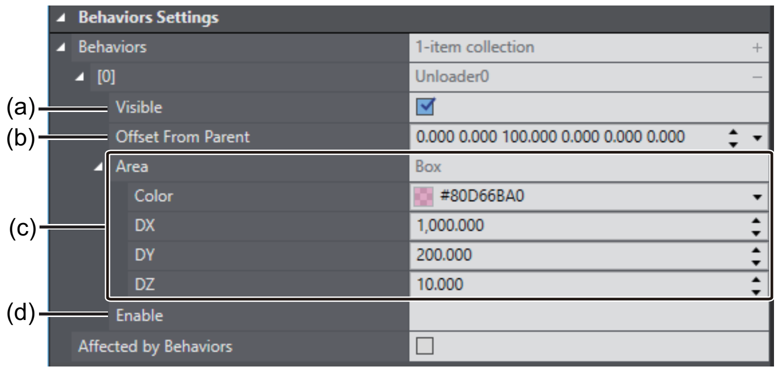



**4** Set the area in which to hide the part on the 3D shape data displayed on the right side of the 3D shape data edit pane.



## Setting Items of *Unloader*

The *Unloader* Behaviors Settings have the following setting items.




| | Item | | Description | Set value | Initial value |
|---|---|---|---|---|---|
| (a) | Visible | | Shows or hides the setting area in the 3D Visualizer. | Selected or unselected | Unselected |
| (b) | Offset From Parent | | Sets the position and pose of the area.<br>Uses the coordinates in the local coordinate system of the 3D shape data.<br>The location elements are X, Y, Z, y (yaw), p (pitch), and r (roll), from left to right. | X, Y, Z: For each, -1,000,000.000 to 1,000,000.000<br>y, r: For each, -180.000 to 180.000<br>p: 0.000 to 180.000 | Depending on the operation at setting the area |
| (c) | Area | Color | Sets the color of the area. | Transparency (0 to #FF) and RGB (0 to #FF, 0 to #FF, 0 to #FF) | #80D66BA0 |
| | | DX | Sets the X-axis length of the area. (Unit: mm) | 0.1 to 10,000 | Depending on the operation at setting the area |
| | | DY | Sets the Y-axis length of the area. (Unit: mm) | 0.1 to 10,000 | Depending on the operation at setting the area |
| | | DZ | Sets the Z-axis length of the area. (Unit: mm) | 0.1 to 10,000 | 10 |

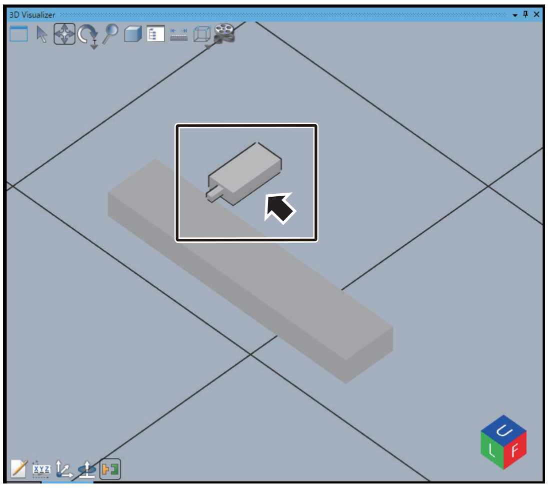
| | Item | Description | Set value | Initial value |
|---|---|---|---|---|
| (d) | Enable | Optionally enables or disables the *Unloader* Behaviors Settings.<br>You can control the timing of part unloading using a Controller variable by specifying the name of a BOOL variable in the Controller.<br>Blank: The *Unloader* Behaviors Settings are always enabled.<br>True: The *Unloader* Behaviors Settings are always enabled.<br>False: The *Unloader* Behaviors Settings are always disabled.<br>ControllerName.ControllerVariable-Name(BOOL): The *Unloader* Behaviors Settings are enabled when the Controller variable is True. | True<br>False<br>ControllerName.ControllerVariable-Name(BOOL) | Blank |

## 7-3-6 Push

Use the following procedure to set *Push* in the Behaviors Settings for 3D shape data that is registered in the Application Manager, such as a mechanical component.

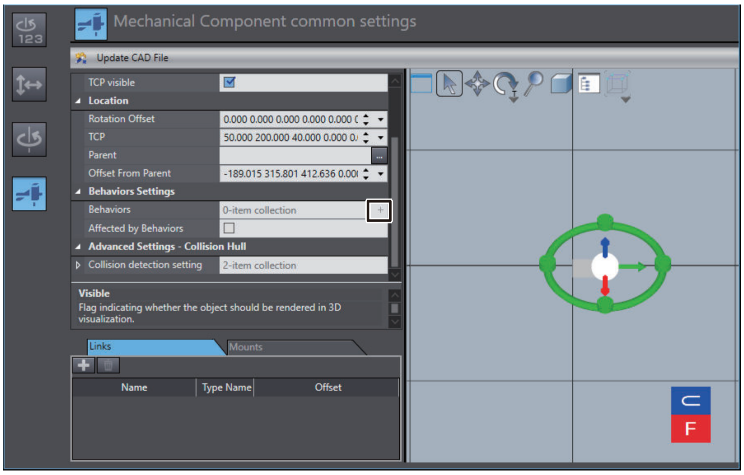
### Operating Procedure

**1** In the 3D Visualizer, double-click the 3D shape data for which to set *Push*.



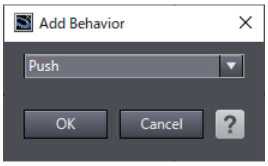
The setup tab page for the 3D shape data is displayed in the edit pane.

**2** In the setup tab page for the 3D shape data, click the button on the right side of **Behaviors** under **Behaviors Settings**. For mechanical components, **Behaviors Settings** are located in the Mechanical Component common settings.

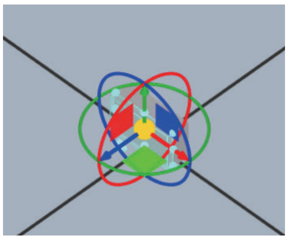



The **Add Behavior** dialog box is displayed.

**3** Select **Push**, and then click the **OK** button.

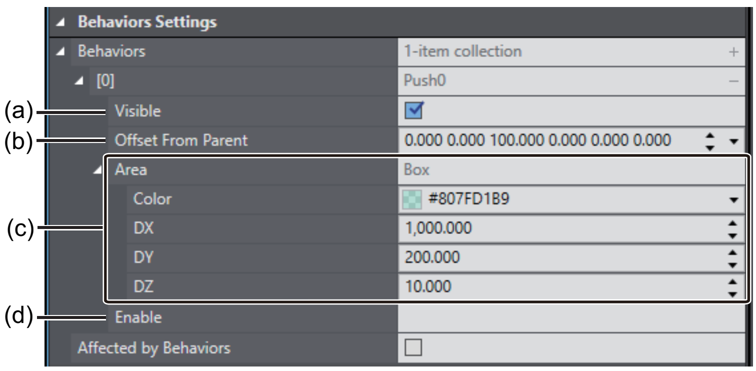



**4** Set the area in which to push the part on the surface of the 3D shape data displayed on the right side of the 3D shape data edit pane.



## Setting Items of *Push*

The *Push* Behaviors Settings have the following setting items.



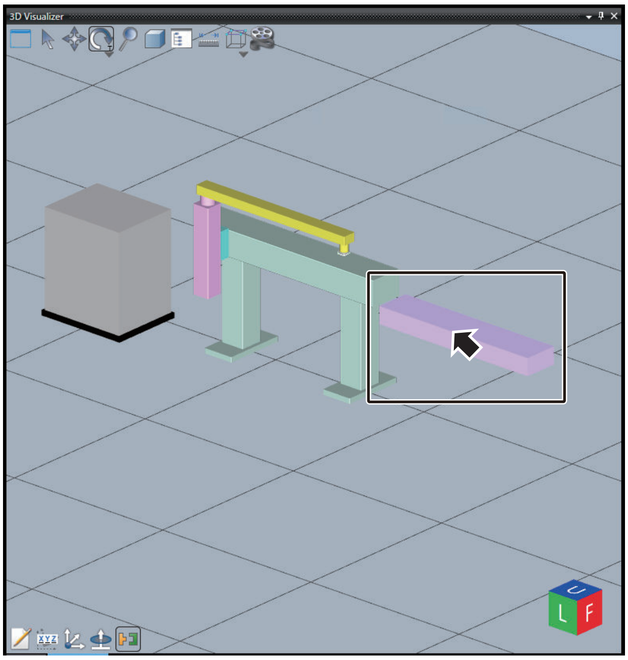
| | Item | | Description | Set value | Initial value |
|---|---|---|---|---|---|
| (a) | Visible | | Shows or hides the setting area in the 3D Visualizer. | Selected or unselected | Unselected |
| (b) | Offset From Parent | | Sets the position and pose of the area. Uses the coordinates in the local coordinate system of the 3D shape data. The location elements are X, Y, Z, y (yaw), p (pitch), and r (roll), from left to right. | X, Y, Z: For each, -1,000,000.000 to 1,000,000.000<br>y, r: For each, -180.000 to 180.000<br>p: 0.000 to 180.000 | Depending on the operation at setting the area |
| (c) | Area | Color | Sets the color of the area. | Transparency (0 to #FF) and RGB (0 to #FF, 0 to #FF, 0 to #FF) | #807FD1B9 |
| | | DX | Sets the X-axis length of the area. (Unit: mm) | 0.1 to 10,000 | Depending on the operation at setting the area |
| | | DY | Sets the Y-axis length of the area. (Unit: mm) | 0.1 to 10,000 | Depending on the operation at setting the area |
| | | DZ | Sets the Z-axis length of the area. (Unit: mm) | 0.1 to 10,000 | 10 |
| (d) | Enable | | Optionally enables or disables the *Push* Behaviors Settings.<br>You can control the timing of part pushing using a Controller variable by specifying the name of a BOOL variable in the Controller.<br>Blank: The *Push* Behaviors Settings are always enabled.<br>True: The *Push* Behaviors Settings are always enabled.<br>False: The *Push* Behaviors Settings are always disabled.<br>ControllerName.ControllerVariableName(BOOL): The *Push* Behaviors Settings are enabled when the Controller variable is True. | True<br>False<br>ControllerName.ControllerVariableName(BOOL) | Blank |

## 7-3-7 Sensor

Use the following procedure to set *Sensor* in the Behaviors Settings for 3D shape data that is registered in the Application Manager, such as a mechanical component.

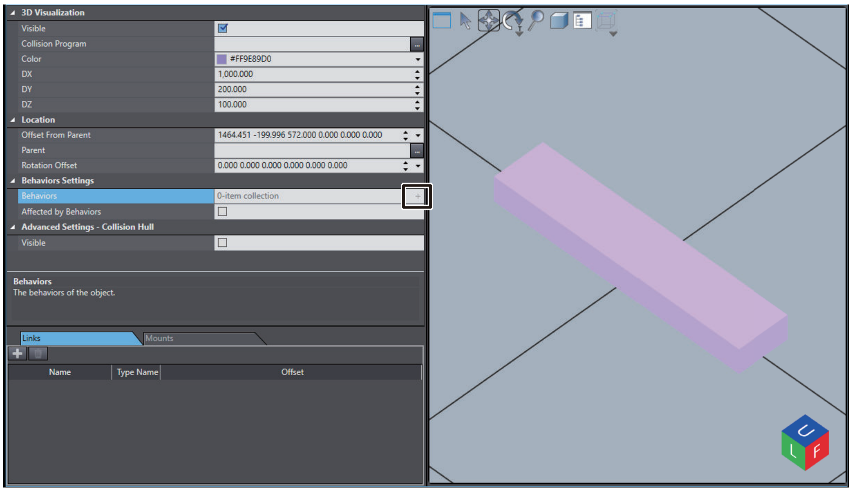
### Operating Procedure

**1** In the 3D Visualizer, double-click the 3D shape data for which to set *Sensor*.



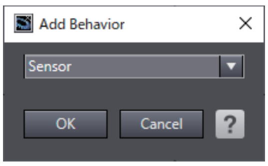

The setup tab page for the 3D shape data is displayed in the edit pane.

**2** In the setup tab page for the 3D shape data, click the button on the right side of **Behaviors** under **Behaviors Settings**. For mechanical components, **Behaviors Settings** are located in the Mechanical Component common settings.

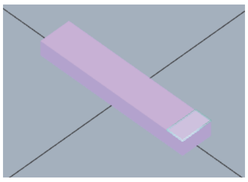



The **Add Behavior** dialog box is displayed.

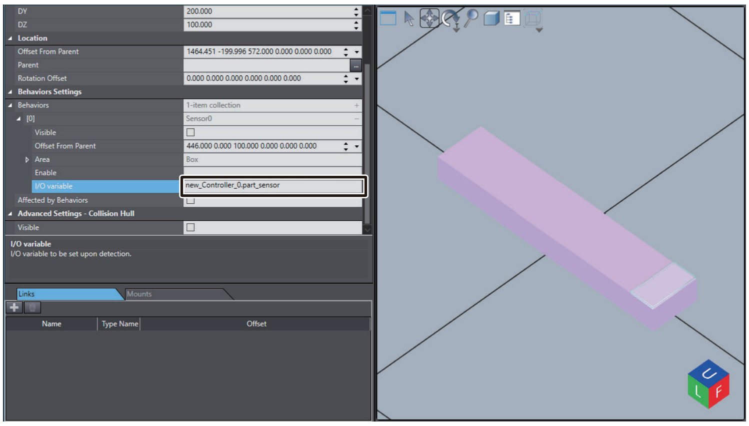
**3** Select **Sensor**, and then click the **OK** button.




**4** Set the area in which to detect the part on the surface of the 3D shape data displayed on the right side of the 3D shape data edit pane.

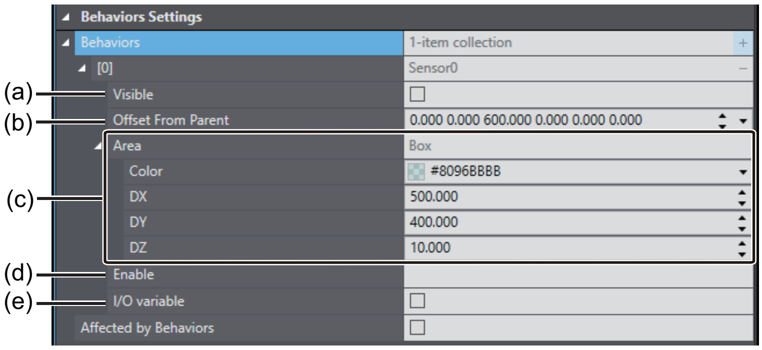


**5** Set a Controller variable that changes the current value to TRUE when a part is detected.



## Setting Items of *Sensor*

The *Sensor* Behaviors Settings have the following setting items.




| | Item | Description | Set value | Initial value |
|---|---|---|---|---|
| (a) | Visible | Show or hides the setting area in the 3D Visualizer. | Selected or unselected | Unselected |
| (b) | Offset From Parent | Sets the position and pose of the area.<br>Uses the coordinates in the local coordinate system of the 3D shape data.<br>The location elements are X, Y, Z, y (yaw), p (pitch), and r (roll), from left to right. | X, Y, Z: For each, -1,000,000.000 to 1,000,000.000<br>y, r: For each, -180.000 to 180.000<br>p: 0.000 to 180.000 | Depending on the operation at setting the area |

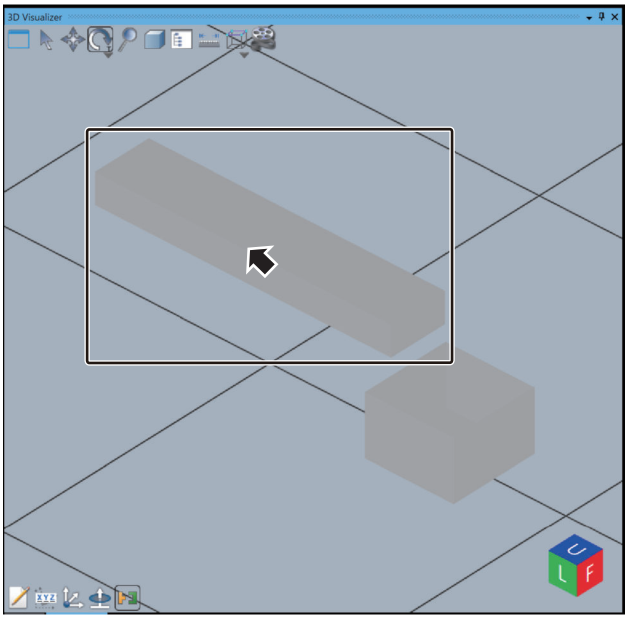
| | Item | | Description | Set value | Initial value |
|---|---|---|---|---|---|
| (c) | Area | Color | Sets the color of the area. | Transparency (0 to #FF) and RGB (0 to #FF, 0 to #FF, 0 to #FF) | #8096BBBB |
| | | DX | Sets the X-axis length of the area. (Unit: mm) | 0.1 to 10,000 | Depending on the operation at setting the area |
| | | DY | Sets the Y-axis length of the area. (Unit: mm) | 0.1 to 10,000 | Depending on the operation at setting the area |
| | | DZ | Sets the Z-axis length of the area. (Unit: mm) | 0.1 to 10,000 | 10 |
| (d) | Enable | | Optionally enables or disables the *Sensor* Behaviors Settings.<br>The *Sensor* Behaviors Settings can be enabled/disabled by specifying the name of a BOOL variable in the Controller.<br>Blank: The *Sensor* Behaviors Settings are always enabled.<br>True: The *Sensor* Behaviors Settings are always enabled.<br>False: The *Sensor* Behaviors Settings are always disabled.<br>ControllerName.ControllerVariableName(BOOL): The *Sensor* Behaviors Settings are enabled when the Controller variable is True. | True<br>False<br>ControllerName.ControllerVariableName(BOOL) | Blank |
| (e) | I/O variable | | Sets the name of the BOOL variable in the Controller that changes the current value to TRUE when the part enters the area. | ControllerName.ControllerVariableName(BOOL) | Blank |

## 7-3-8 Physics

The behavior, Physics, makes the part to move based on a simple physics simulation when the part comes in contact with the specified area. You can simulate falling and sliding.
Use the following procedure to set *Physics* in the Behaviors Settings for 3D shape data that is registered in the Application Manager, such as a mechanical component.

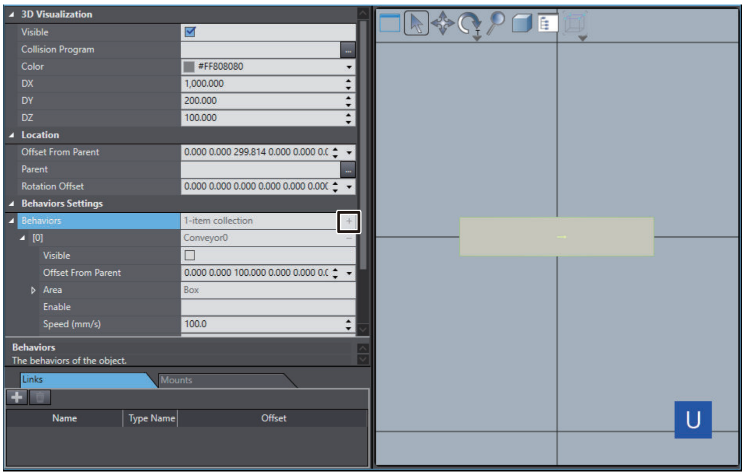
### Operating Procedure

**1** In the 3D Visualizer, double-click the 3D shape data for which to set *Physics*.



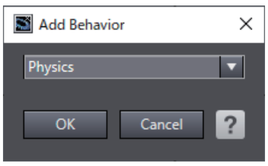

The setup tab page for the 3D shape data is displayed in the edit pane.

**2** In the setup tab page for the 3D shape data, click the button on the right side of **Behaviors** under **Behaviors Settings**. For mechanical components, **Behaviors Settings** are located in the Mechanical Component common settings.

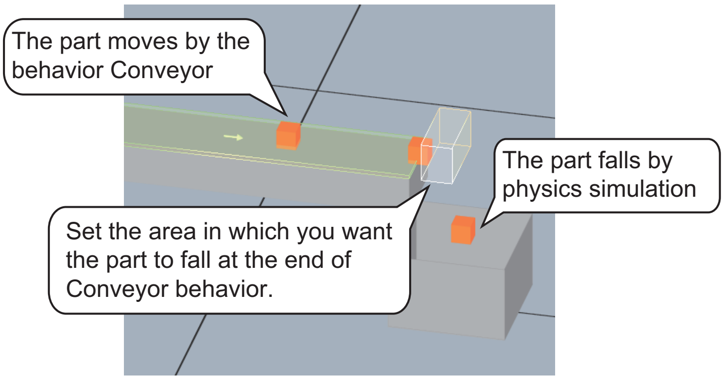



The **Add Behavior** dialog box is displayed.

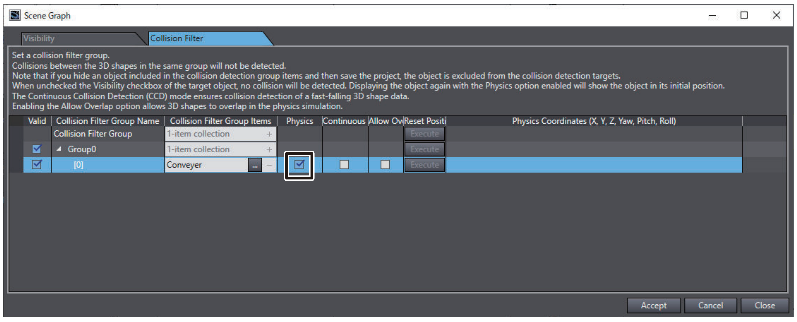
**3** Select **Physics**, and then click the **OK** button.




**4** Set the area in which to detect the part on the 3D shape data displayed on the right side of the 3D shape data edit pane.

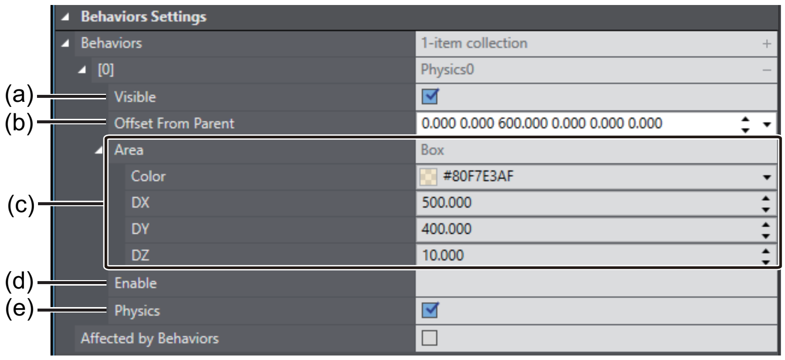



**5** In 3D shape data that the part will come in contact with after the physics simulation, set the collision detection filter to allow the part to continue its motion. If you do not set this, the part dropped in the physics simulation will pass through even when it collides with the 3D shape data.




## Setting Items of *Physics*

The *Physics* Behaviors Settings have the following setting items.

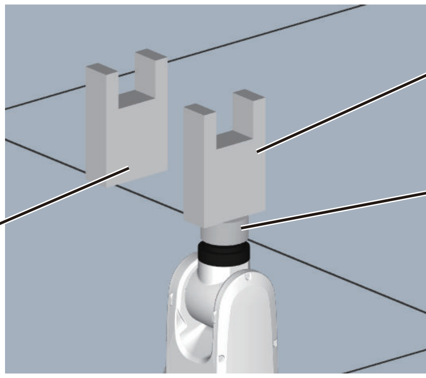



| | Item | Description | Set value | Initial value |
|---|---|---|---|---|
| (a) | Visible | Shows or hides the setting area in the 3D Visualizer. | Selected or unselected | Unselected |

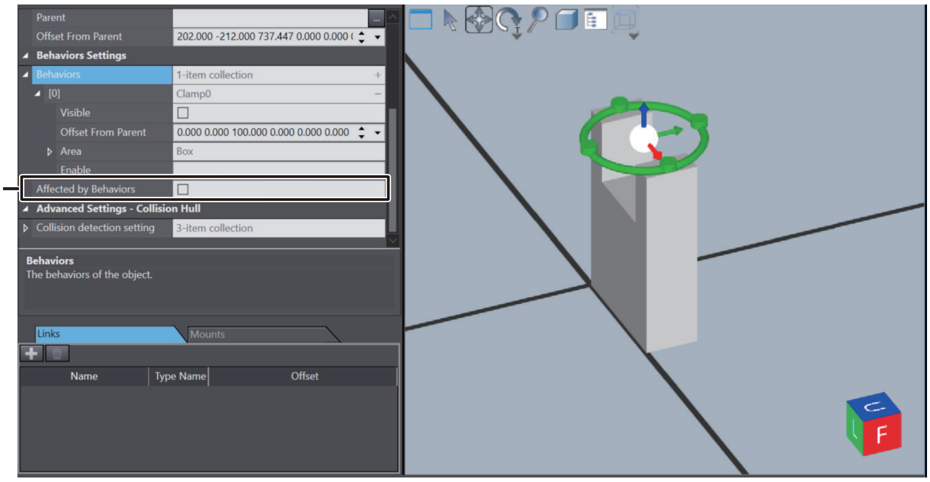
| | Item | | Description | Set value | Initial value |
|---|---|---|---|---|---|
| (b) | Offset From Parent | | Sets the position and pose of the area. Uses the coordinates in the local coordinate system of the 3D shape data.<br>The location elements are X, Y, Z, y (yaw), p (pitch), and r (roll), from left to right. | X, Y, Z: For each, -1,000,000.000 to 1,000,000.000<br>y, r: For each, -180.000 to 180.000<br>p: 0.000 to 180.000 | Depending on the operation at setting the area |
| (c) | Area | Color | Sets the color of the area. | Transparency (0 to #FF) and RGB (0 to #FF, 0 to #FF, 0 to #FF) | #80F7E3AF |
| | | DX | Sets the X-axis length of the area. (Unit: mm) | 0.1 to 10,000 | Depending on the operation at setting the area |
| | | DY | Sets the Y-axis length of the area. (Unit: mm) | 0.1 to 10,000 | Depending on the operation at setting the area |
| | | DZ | Sets the Z-axis length of the area. (Unit: mm) | 0.1 to 10,000 | 10 |
| (d) | Enable | | Optionally enables or disables the *Physics* Behaviors Settings.<br>The *Physics* Behaviors Settings can be enabled/disabled by specifying the name of the BOOL variable in the Controller.<br>Blank: The *Physics* Behaviors Settings are always enabled.<br>True: The *Physics* Behaviors Settings are always enabled.<br>False: The *Physics* Behaviors Settings are always disabled.<br>ControllerName.ControllerVariableName(BOOL): The *Physics* Behaviors Settings are enabled when the Controller variable is True. | True<br>False<br>ControllerName.ControllerVariableName(BOOL) | Blank |
| (e) | Physics | | Enables or disables physics simulation when the part enters the area. | Selected or unselected | Selected |

## 7-3-9 Common Behaviors Settings

“Behaviors Settings” give motions to parts that are shown by the *Loader* Behaviors Settings. Enabling *Affected by Behaviors* allows you to use 3D shape data that is not a part shown by the *Loader* Behaviors Settings in the same way as the part. The following is a configuration example for enabling a tool changer to switch the tool between robot tool 1 and robot tool 2. Set *Clamp* for the tool changer and enable *Affected by Behaviors* for robot tool 1 and robot tool 2.




**Affected by Behaviors** is located in **Behaviors** under **Behaviors Settings** in the setup tab page for the 3D shape data.




| | Item | Description | Set value | Initial value |
|---|---|---|---|---|
| (a) | Affected by Behaviors | Makes a part move according to the Behaviors Settings of the 3D shape data that the part contacts. | Selected or unselected | Unselected |

# 8

# Executing a 3D Simulation

This section describes the procedures to execute a 3D simulation and the operation check methods.

# 8-1 Operating Procedures for a 3D Simulation

The procedure of 3D simulation differs depending on whether you use scripts or Easy Part Simulation Configuration.

## ● Using Scripts

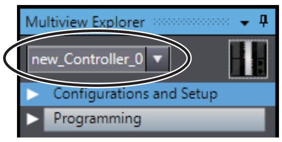
1. Refer to *8-1-1 Executing a Controller Simulation* on page 8-2.
2. Refer to *8-1-2 Executing the Operation Script for the Virtual Part Detection Sensor and the Virtual Output Script for the Mechanical Component* on page 8-3.
3. Refer to *8-1-3 Executing the Shape Scripts for the Part* on page 8-3.
4. Check operations in the 3D Visualizer

## ● Using Easy Part Simulation Configuration

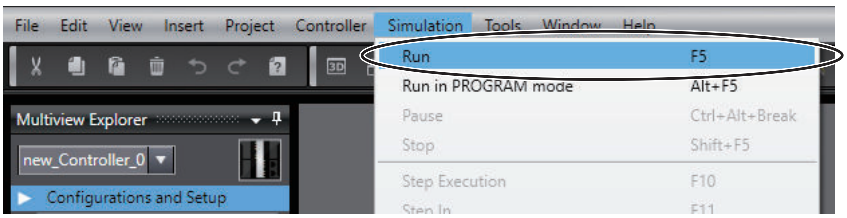
1. Refer to *8-1-1 Executing a Controller Simulation* on page 8-2.
2. Refer to *8-1-4 Executing a Simulation Using Behaviors Settings* on page 8-4.
3. Check operations in the 3D Visualizer

## 8-1-1 Executing a Controller Simulation

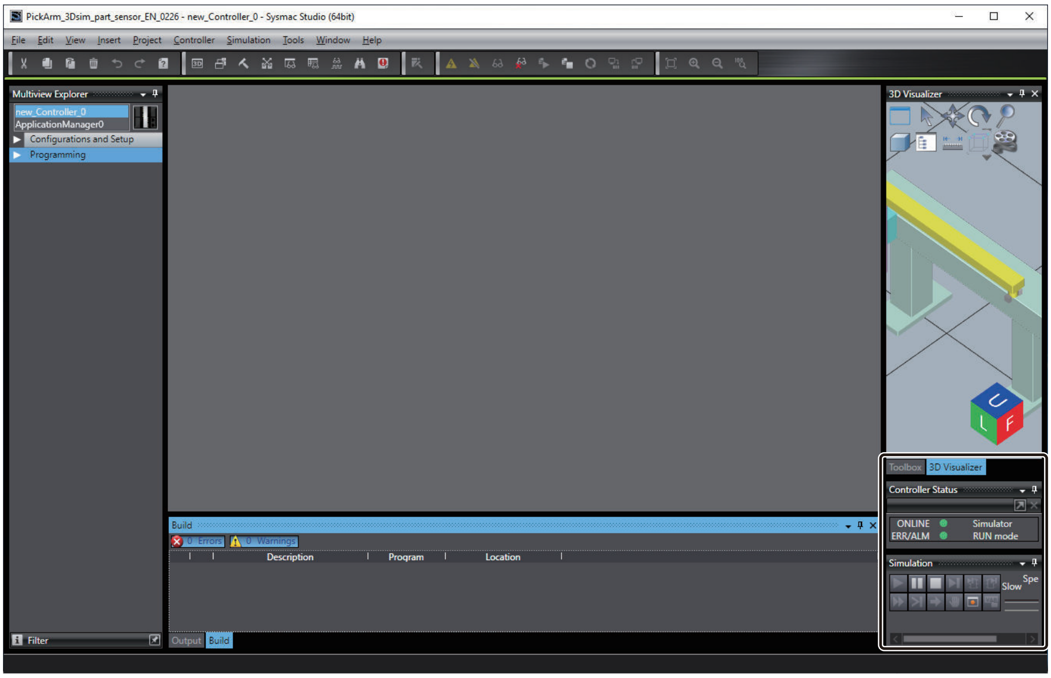
**1** Select a Controller in the device list.
The Controller is selected as the device.




**2** Select **Run** from the **Simulation** menu.




A connection is established with Simulator of the Controller.



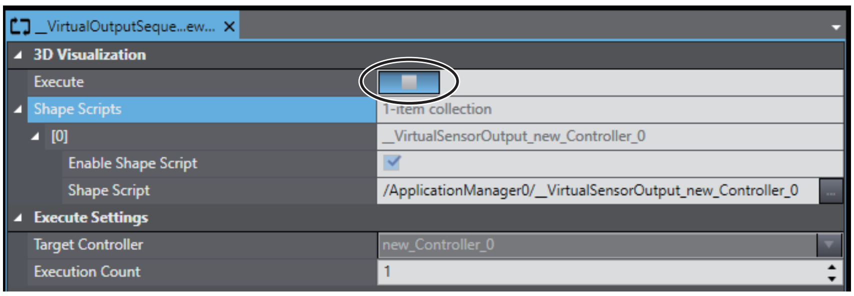

## 8-1-2 Executing the Operation Script for the Virtual Part Detection Sensor and the Virtual Output Script for the Mechanical Component

**1** Select an Application Manager in the device list, and then double-click **__VirtualOutputSequence_(Controller name)** under **3D Visualization** in the Multiview Explorer.

The **__VirtualOutputSequence_(Controller name)** setup tab page is displayed.

**2** Select the **Enable Shape Script** check box under **Shape Scripts**, and then click the **Execute** button under **3D Visualization**.

This starts the execution of the operation script for the Virtual Part Detection Sensor and the virtual output script for a mechanical component.



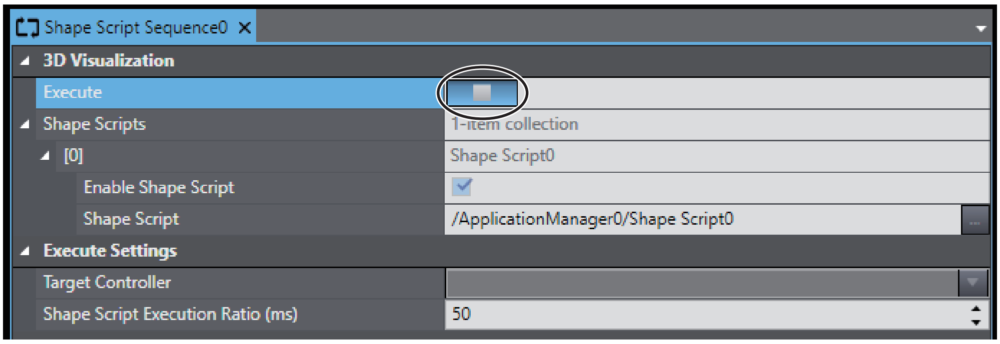

## 8-1-3 Executing the Shape Scripts for the Part

**1** Select an Application Manager in the device list, and then double-click Shape Script Sequence under **3D Visualization** in the Multiview Explorer.

The Shape Script Sequence setup tab page is displayed.

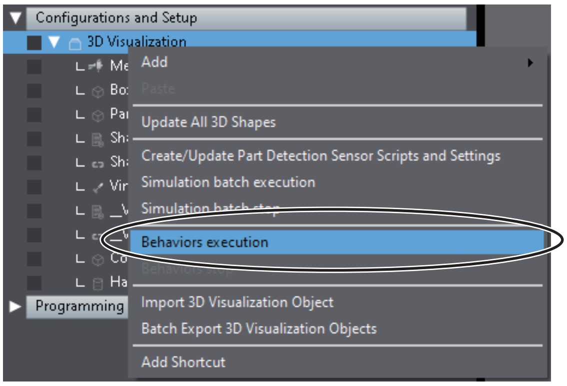
**2** Select the **Enable Shape Script** check box under **Shape Scripts**, and then click the **Execute** button under **3D Visualization**.

This starts the execution of the Shape Scripts.




## 8-1-4 Executing a Simulation Using Behaviors Settings

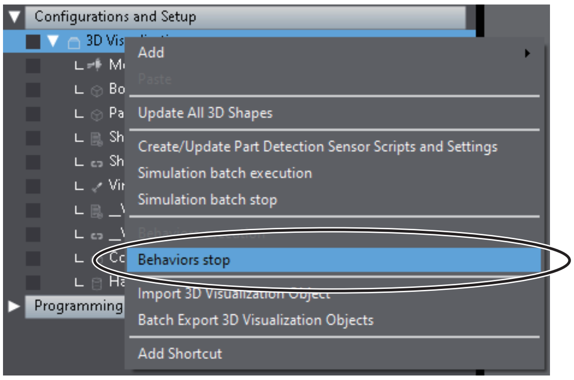
**1** Select an Application Manager from the device list, right-click **3D Visualization** in the Multiview Explorer, and select **Behaviors execution** from the menu.




This starts a simulation using Behaviors Settings.

## 8-1-5 Stopping a Simulation Using Behaviors Settings

**1** Select an Application Manager from the device list, right-click **3D Visualization** in the Multiview Explorer, and select **Behaviors stop** from the menu.




This stops the simulation using Behaviors Settings.

## 8-1-6 Checking Operations in the 3D Visualizer

**1** In the 3D Visualizer, perform operations such as Translate, Rotate, Zoom, and other operations to check how the part and Mechanical Component operate.
Refer to *5-1 Displaying the 3D Visualizer and 3D Editing Area* on page 5-2 for information on Translate, Rotate, Zoom, and other operations.
Refer to *8-2 Debugging a Shape Script* on page 8-8 for information on how to use the Shape Script Editor functions that you can use for checking and debugging the operations of the part.
To debug the control program for a mechanical component, use the debugging functions provided in the Ladder Editor or ST Editor. Refer to Sysmac Studio Version 1 Operation Manual (Cat. No. W504) for details on the operating procedure.

### Additional Information

- Depending on the system requirement for your computer, the Virtual equipment model in the 3D Visualizer may operate slower than the actual equipment. To have the Virtual equipment model in the 3D Visualizer operate at the same speed as the actual equipment, execute a Controller simulation in *Execution Time Estimation Mode*. Refer to *Sysmac Studio Version 1 Operation Manual (Cat. No. W504)* for the functions and execution procedures in Execution Time Estimation Mode.
- You can debug the control program for a mechanical component and the Shape Script at the same time. To do so, while you debug the control program by executing a data trace, switch the device to the Application Manager and debug the Shape Script.

### Simulation Elapsed Time

When you execute a Controller simulation, the 3D Visualizer displays the elapsed time since the start of execution of the Controller simulation.

| Item | Units |
|---|---|
| Simulation elapsed time | minutes:seconds.centiseconds |

**Simulation elapsed time** indicates the time during which the simulation is executed.
The Virtual equipment model in the 3D Visualizer may temporarily slow down due to changes in the load on your computer, which makes it difficult to grasp the operation timing and elapsed time of the model. Since the simulation elapsed time is linked to the movement of the Virtual equipment model on the display, you can check the operation timing and elapsed time of the model regardless of the load on the computer.

**Additional Information**

- When you pause the simulation, the display of the simulation elapsed time stops temporarily.
- When the simulation is completed, the simulation elapsed time disappears.

## 8-1-7 Executing Operations for a 3D Simulation at Once

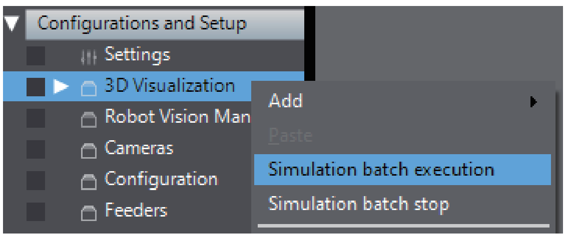
Although 3D simulation requires that you execute the following operations in order, it is also possible to perform these operations at once.

| Order | Operation | Reference |
|---|---|---|
| 1 | Executing a Controller Simulation | *8-1-1 Executing a Controller Simulation* on page 8-2 |
| 2 | Executing the Operation Script for the Virtual Part Detection Sensor and the Virtual Output Script for the Mechanical Component | *8-1-2 Executing the Operation Script for the Virtual Part Detection Sensor and the Virtual Output Script for the Mechanical Component* on page 8-3 |
| 3 | Executing the Shape Scripts for the Part | *8-1-3 Executing the Shape Scripts for the Part* on page 8-3 |
| 4 | Executing a Simulation Using Behaviors Settings | *8-1-4 Executing a Simulation Using Behaviors Settings* on page 8-4 |

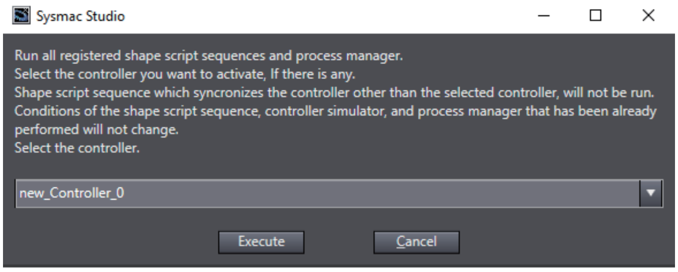
This section describes how to execute operations required for a 3D simulation at once.

### Executing a 3D Simulation

**1** Right-click **3D Visualization** under **Configurations and Setup** in the Multiview Explorer and select **Simulation batch execution** from the menu.



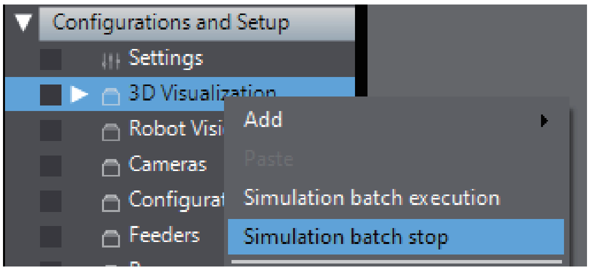

The Controller selection dialog box is displayed.




**2** Select the Controller to execute a simulation, and then click the **Execute** button.
The Controller simulation runs, and the operation script for the Virtual Part Detection Sensor, virtual output script for a mechanical component, Shape Script, and simulation based on the Behaviors Settings start running.

## Stopping a Simulation

**1** Right-click **3D Visualization** under **Configurations and Setup** in the Multiview Explorer and select **Simulation batch stop** from the menu.




The Simulator of the Controller stops, and the operation script for the Virtual Part Detection Sensor, virtual output script for a mechanical component, Shape Script, and the simulation based on the Behaviors Settings stop.

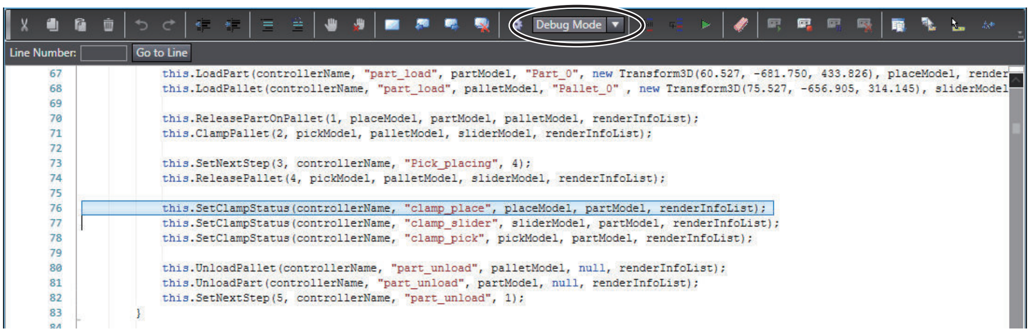
# 8-2 Debugging a Shape Script

To debug a Shape Script, use the debugging functions provided in the Shape Script Editor. Double-click the target Shape Script in the Multiview Explorer to display the Shape Script Editor in advance.

## 8-2-1 Breakpoint

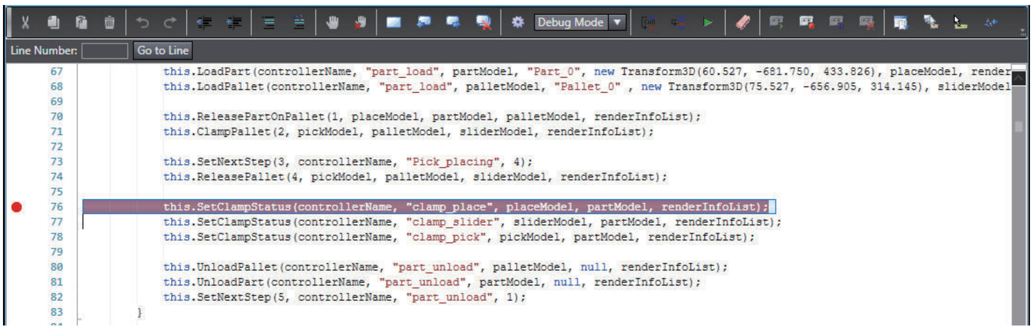
Use a breakpoint to pause the execution of a Shape Script on any line so that you can check the present value of a local variable at that point.
The execution of a Shape Script pauses at the beginning of the line on which a break point is set. However, it does not execute the line on which the break point is set.

**1** In the Shape Script Editor, change the **Mode Selection** in the toolbar from *RELEASE mode* to *DEBUG mode*.



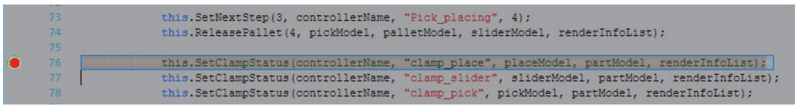

In DEBUG mode, you can click the **Toggle Breakpoint** and **Clear Breakpoints** buttons.

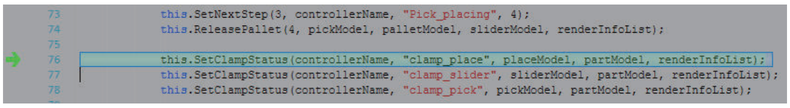
**2** Move the cursor to the line on which to pause the execution of the Shape Script, and then click the **Toggle Breakpoint** button.
A breakpoint is set on the line with the cursor.



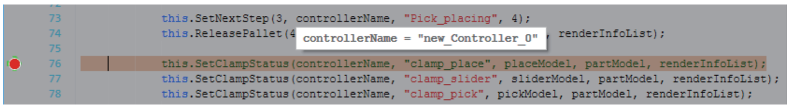

**3** Execute the target Shape Script in the Shape Script Sequence.

The color of the line with the breakpoint changes and, to the left of the line number, an arrow icon that indicates the execution position is superimposed on the icon that indicates the line with a breakpoint.




**4** Move the mouse cursor to any local variable above the executed line.
The present value of the local variable is displayed as a tooltip.



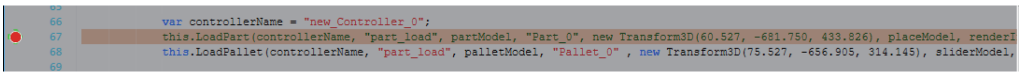

### Additional Information

The **Watch** tab page allows you to check multiple variable values at once. Refer to *The **Watch** Tab Page for Shape Scripts* on page 6-21 for details on the **Watch** tab page.

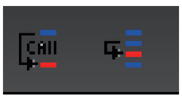
## 8-2-2 Step Execution

Use step execution to pause the execution of a Shape Script at the beginning of any line on which a breakpoint is set, and then execute it again line by line so that you can check the present value of a variable at that point.

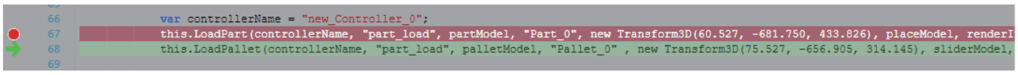
**1** Set a breakpoint, execute the target Shape Script in the Shape Script Sequence, and then stop the execution of the Shape Script before it reaches the point at which to perform step execution.




**2** Click the **Step Over** or **Step Into** button.



The execution position moves. Clicking the **Step Over** button moves the execution position to the beginning of the next function. Clicking the **Step Into** button moves the execution position to inside the target function.




**3** Click the **Step Over** or **Step Into** button again and again.
The execution position changes according to the execution result.

**Additional Information**

The **Watch** tab page allows you to check multiple variable values at once. Refer to *The **Watch** Tab Page for Shape Scripts* on page 6-21 for details on the **Watch** tab page.

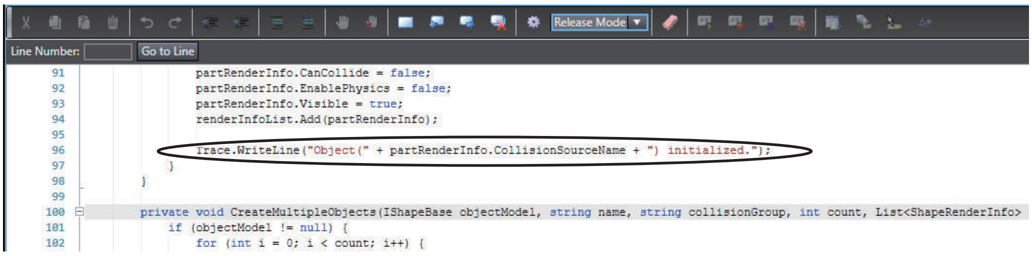
## 8-2-3 Trace Statement

Inserting a trace statement in any line of a Shape Script allows you to check which point the Shape Script is processed to. The text string specified in a trace statement will be output to a log and displayed on the **Trace Message** tab page during the execution of the trace statement in the Shape Script.

**1** Open the Shape Script Editor, and then insert the following trace statements in any positions.
Format:
Trace.WriteLine (Contents);

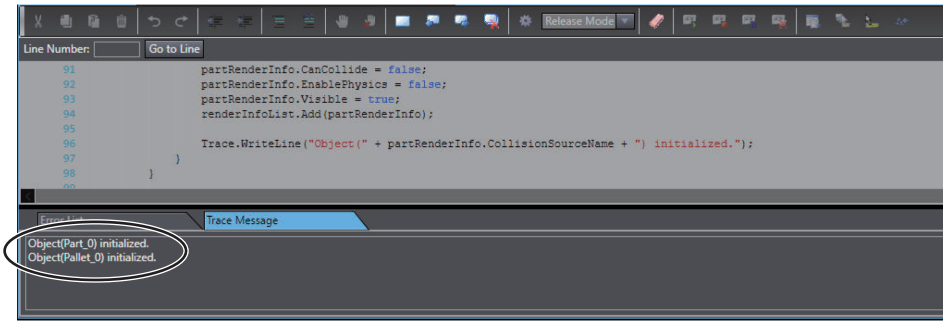
Enter "text strings" or members inside the parentheses. To list two or more text strings or members, use + between each entry.

Notation example: Trace.WriteLine("Object(" + partRenderInfo.CollisionSourceName + ") initialized.");
*Trace.WriteLine("Script Starting");*
*Trace.WriteLine("partDetectionSensor = "+ partDetectionSensor);*




The above example means to display the value of *partRenderInfo.CollisionSourceName* to leave the executed result of the trace statement in a log.

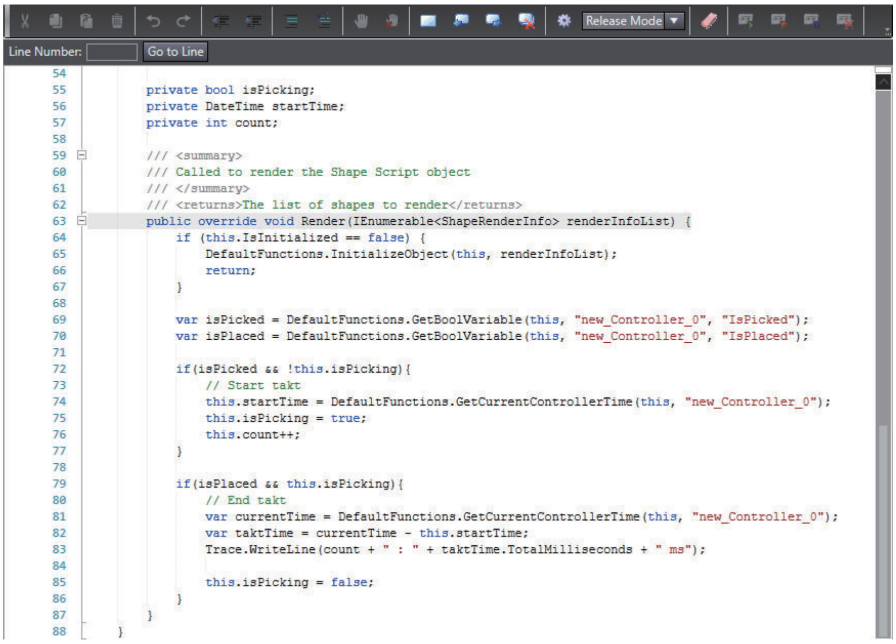
**2** Execute the target Shape Script in the Shape Script Sequence.
The execution results of the trace statements are displayed on the **Trace Message** tab page of the Shape Script Editor.




## 8-2-4 Takt Time Measurement

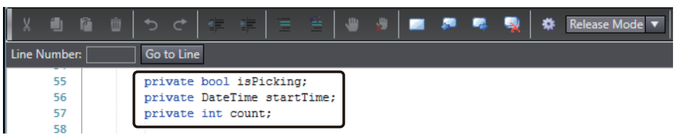
To measure the takt time, get the internal simulation time of the Controller during the execution of the Shape Script.

The following describes this procedure for an application where a part is picked up and placed, as an example.




**1** Define the following variables.

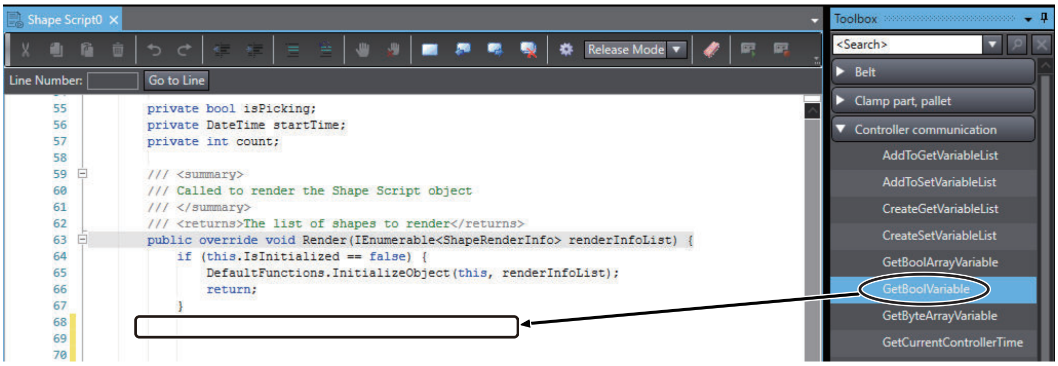
- isPicking (BOOL): A BOOL variable that is TRUE while the part is picked up. Use this variable to determine whether to process the part or not at the start or end of the cycle.
- startTime (DATETIME): A variable that records the time at the start of the cycle.
- count (INT): A variable that records the number of the cycle.




**2** Get the Controller's variables in the Render function of the Shape Script.
Assume that the Controller has the following variables.

- IsPicked (BOOL): A BOOL variable that is TRUE while the part is picked up.
- IsPlaced (BOOL): A BOOL variable that is TRUE while the part is placed, until the picking of the next part starts.

Get the above variables in the Render function of the Shape Script.
To get the BOOL type variable from the Controller, add the GetBoolVariable function.
To add the variable, select **ControllerCommunication** in the Toolbox and drag **GetBoolVariable** into the Shape Script Editor window. The GetBoolVariable function call is now inserted. Set the parameters.

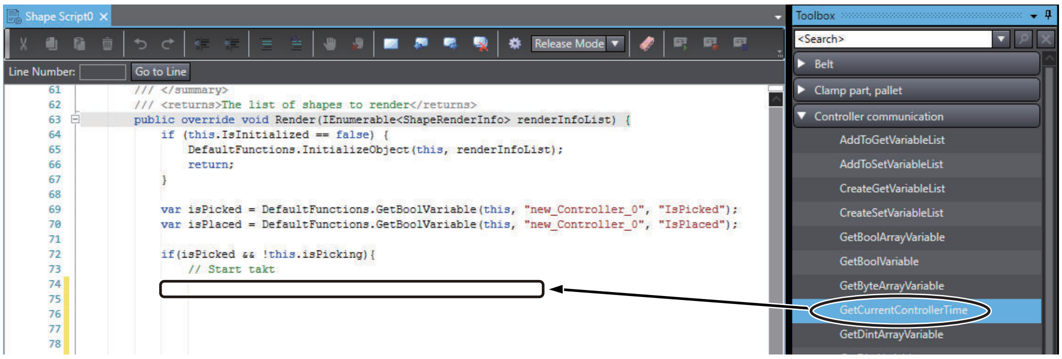



For **GetBoolVariable**, specify the Controller name *new_Controller_0* as the first argument and the Controller variable's name *IsPicked* as the second argument.
In addition, define the variable to which to assign the value that you got. Here, define the variable *IsPicked*.
Similarly, define *IsPlaced*, to which to assign the Controller's variable name *IsPlaced*.

```
public override void Render(IEnumerable<ShapeRenderInfo> renderInfoList) {
    if (this.IsInitialized == false) {
        DefaultFunctions.InitializeObject(this, renderInfoList);
        return;
    }

    var isPicked = DefaultFunctions.GetBoolVariable(this, "new_Controller_0", "IsPicked");
    var isPlaced = DefaultFunctions.GetBoolVariable(this, "new_Controller_0", "IsPlaced");
```

**3** Describe the processing to execute at the start of the cycle.
The start of the cycle means the point of time at which the picking of the part starts. It is the point at which the variable *IsPicked* changes to TRUE.
At this point, get the internal simulation time of the Controller and assign it to a variable *startTime*. The function to get the internal simulation time is as follows.

To get the internal simulation time of the Controller, add the GetCurrentControllerTime function.
To add the function, select **ControllerCommunication** in the Toolbox and drag **GetCurrentControllerTime** into the Shape Script Editor window. The GetCurrentController-Time function call is now inserted. Set the parameters.




At the start of the cycle, assign TRUE to the variable *isPicking*. This prevents the same processing from being passed for subsequent Render function calls.
Add the variable *count* by 1. The results are displayed in the **Trace Message** tab page.

```
        var isPicked = DefaultFunctions.GetBoolVariable(this, "new_Controller_0", "IsPicked");
        var isPlaced = DefaultFunctions.GetBoolVariable(this, "new_Controller_0", "IsPlaced");

        if(isPicked && !this.isPicking){
            // Start takt
            this.startTime = DefaultFunctions.GetCurrentControllerTime(this, "new_Controller_0");
            this.isPicking = true;
            this.count++;
        }
```

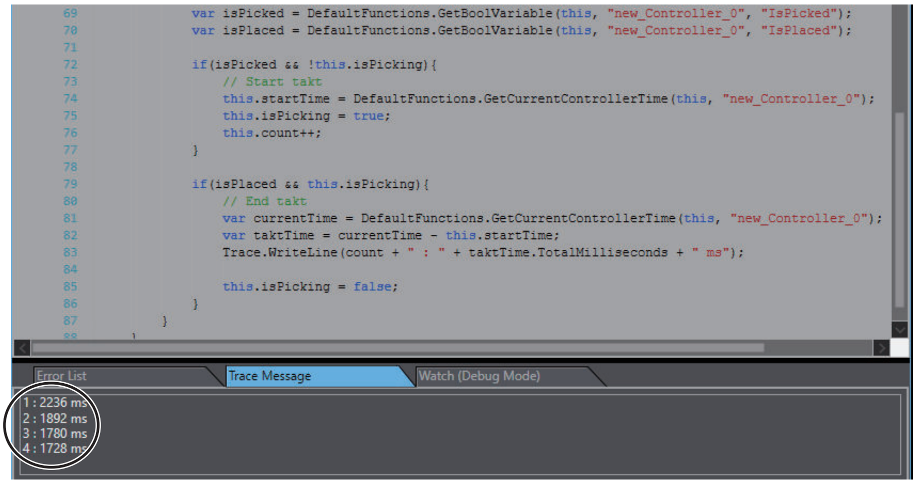
**4** Describe the processing to execute at the end of the cycle.

The end of the cycle means the point of time at which the part is placed. It is the point at which the variable *isPlaced* changes to TRUE.
At this point, get the internal simulation time of the Controller and assign it to another variable *currentTime*.
Assign the difference between the variable *currentTime* and the variable *startTime* with the internal time set at the start of the cycle to the variable *taktTime*.
Use the Trace.WriteLine function to display the takt time that you got in the **Trace Message** tab page.
Assign FALSE to the variable *isPicking*. This causes the processing to execute at the start of the cycle to be executed at the start of the next picking of the part.

```
            this.isPicking = true;
            this.count++;
        }

        if(isPlaced && this.isPicking){
            // End takt
            var currentTime = DefaultFunctions.GetCurrentControllerTime(this, "new_Controller_0");
            var taktTime = currentTime - this.startTime;
            Trace.WriteLine(count + " : " + taktTime.TotalMilliseconds + " ms");

            this.isPicking = false;
        }
    }
  }
}
```

**5** Execute the target Shape Script in the Shape Script Sequence.

The takt time is displayed in the **Trace Message** tab page.
The difference in the internal simulation time between when the Controller's variable *IsPicked* changes to TRUE and when the variable *IsPlaced* changes to TRUE is displayed.



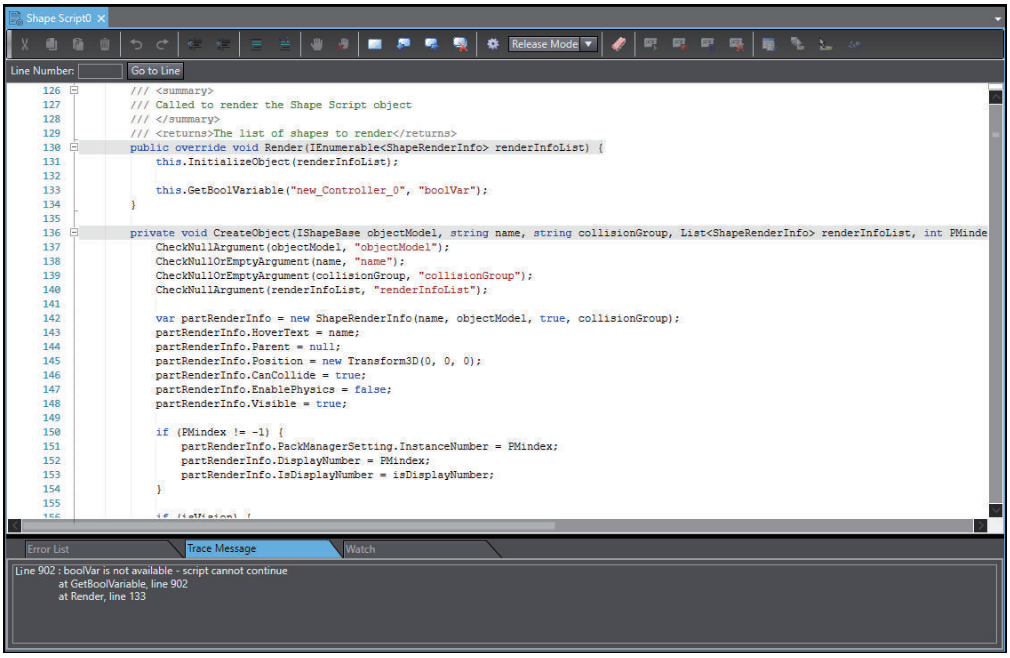

**Additional Information**

Using Stopwatch constructs allows you to measure execution times of desired sections in Shape Scripts. Refer to *A-3-2 Measuring Execution Time with Stopwatch Constructs* on page A-71 for details.

## 8-2-5 Displaying Errors during Execution of Shape Scripts

When you execute a Shape Script from a Shape Script Sequence, and the execution fails due to an error, the location where the error occurred is displayed on the **Trace Message** tab page.
The **Trace Message** tab page displays a function where an error occurred with the line number of the error location, under which the function that called the function where the error occurred is displayed with the line number in which the function where the error occurred is called. If an error occurs in a function that is called through multiple functions, the function names and the line numbers of the function calls are displayed in order of the last called function to the first called function. The Render() function and the line number of the related function call are displayed at the bottom of the function call history.

The following example shows a message that is displayed when an attempt to get a Controller variable fails because the variable is not found.




In this example, the Render() function executes the GetBoolVariable() function where an error occurs. The GetBoolVariable() function is executed with the parameters passed from the Render() function. If a parameter passed from the Render() function is invalid, the GetBoolVariable() function returns an error. In this example, the line calling the GetBoolVariable() function in the Render() function must be modified.

This example shows that the error occurred in line 902 in the GetBoolVariable() function. This example also shows that the GetBoolVariable() function was called from line 133 in the Render() function. Thus, line 133 must be modified. In this example, the name of the Controller variable of the arguments is invalid.

# 8-3 Collision Detection Function

This function detects whether contacts will occur between 3D shape data such as parts that make up an equipment model and the part during operation.
Checking the presence of contacts and making necessary corrections beforehand in a 3D simulation prevents unexpected collisions between parts during operation.

## 8-3-1 Collision Detection Target

Collision detection is possible for the following 3D shape data.

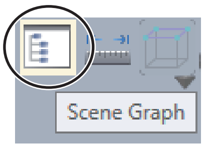
| Type of model | Collision detection target |
|---|---|
| Mechanical Component | Movable parts or the entire Mechanical Component |
| Part | All 3D shape data included in the part settings |

## 8-3-2 Collision Detection Setting Procedure

The procedure to set the collision detection function is described below.
To detect collisions, register target objects that could cause a collision between them in different *Collision Filter Groups*.
Collisions between 3D shape data in different Collision Filter Groups will be detected. Collisions between 3D shape data in the same Collision Filter Group will not be detected.
This section describes the setting procedure for collision detection between the mechanical component and the part, using an example case where a part of the mechanical component comes in contact with the part.
Register 3D shape data for the following mechanical component and part in advance.

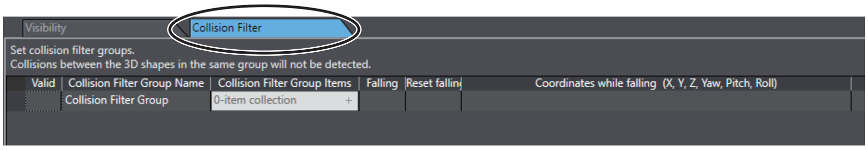
Name of mechanical component: MechanicalModel000
Name of part: Part

**1** In the 3D Visualizer, click the **Scene Graph** icon.

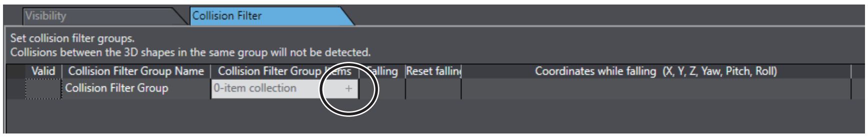



The **Scene Graph** dialog box is displayed.

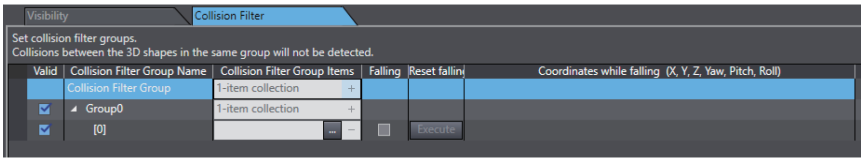

**2** Click the **Collision Filter** tab.
The display changes to the **Collision Filter** tab page.



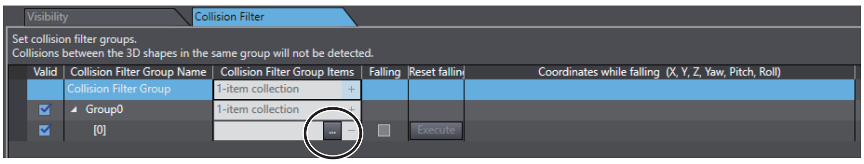

**3** Click the **+** button for Collision Filter Group.



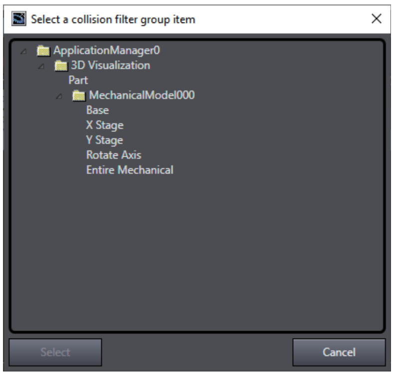

A Collision Filter Group is added.



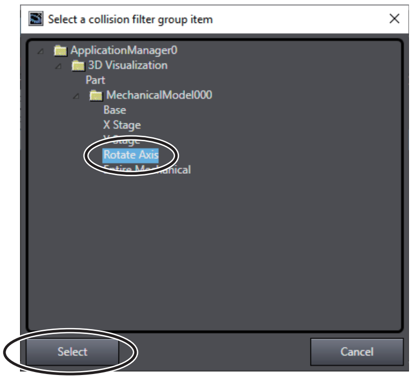

**4** Click the Item Selection button for the added Collision Filer Group.



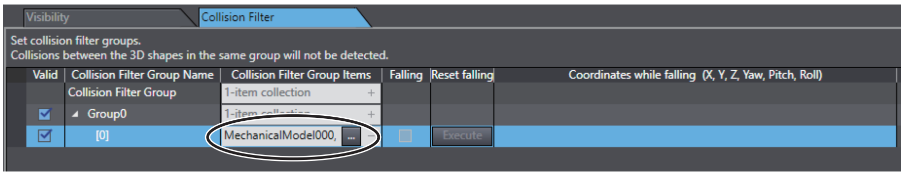

A dialog box is displayed for you to register the target 3D shape data in the Collision Filer Group.



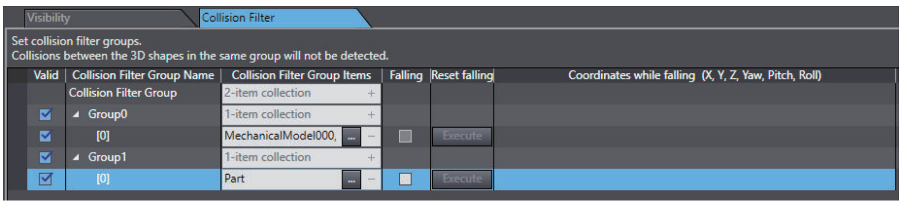

**5** Select the 3D shape data for the target Mechanical Component to register in the Collision Filer Group, and then click the **Select** button.




The selected 3D shape data is registered in the Collision Filer Group.




**6** In the same way, register 3D shape data for the part.
Be sure to set this 3D shape data in a different Collision Filer Group from that for the 3D shape data for the mechanical component.




**7** Check that the **Valid** check boxes for the 3D shape data and groups between which to detect collisions, and then click the **OK** button.

**Precautions for Correct Use**

Adding a Virtual Part Detection Sensor and then setting the detection target part registers __*VirtualSensorOutputGroup* and *a part name* in the Collision Filter Group. Be careful not to delete these settings because they are necessary for the Virtual Part Detection Sensor to operate.

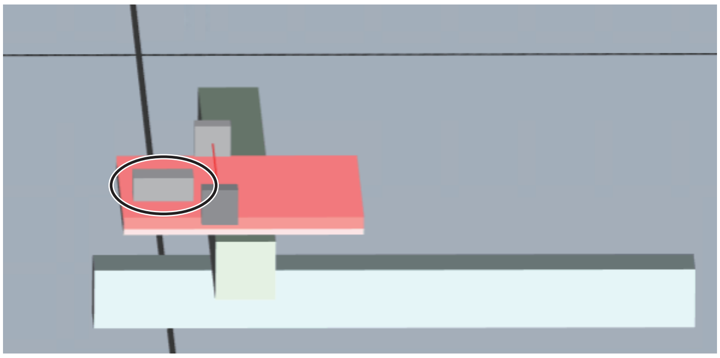
## 8-3-3 How to Check Detected Collisions

Use the following method to check whether any collision is detected during the execution of a 3D simulation.

Perform this check for detected collisions in the 3D Visualizer.

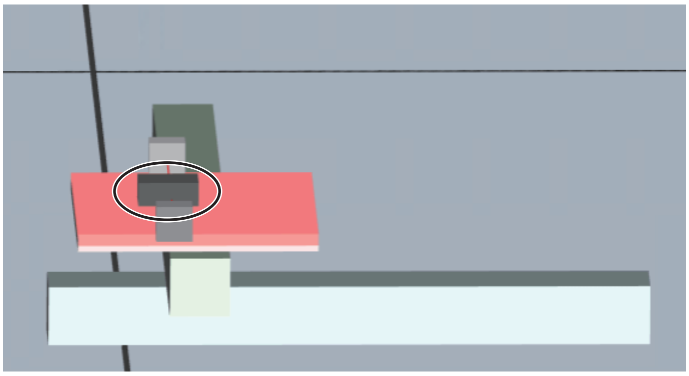
### 3D Shape Data That Did Not Cause a Collision

The 3D shape data is shown in the color specified in its setup tab page.



### 3D Shape Data That Caused a Collision

The 3D shape data is shown in a darker color than the color specified in its setup tab page.



If the target 3D shape data causes a collision, the value of the relevant 3D shape data property changes in the Shape Script Editor. You can check the occurrence of a collision by outputting this property value in a trace statement.

If an unexpected collision occurs, review the following elements to prevent the collision.

- Position of 3D shape data that caused a collision
- Shape of 3D shape data that caused a collision

- Control program and parameters related to the operations of 3D shape data that caused a collision

## 8-3-4 How to Detect Collisions with Shape Scripts

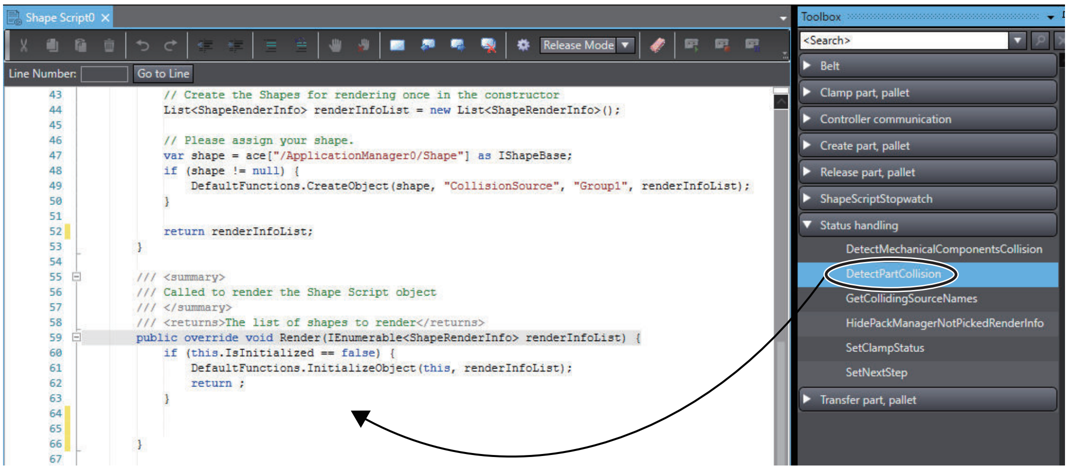
This section describes how to use Shape Scripts to detect collisions and how to get the name of the 3D shape data that caused the collisions.

### Detecting a Collision between a Part or Pallet and a Mechanical Component

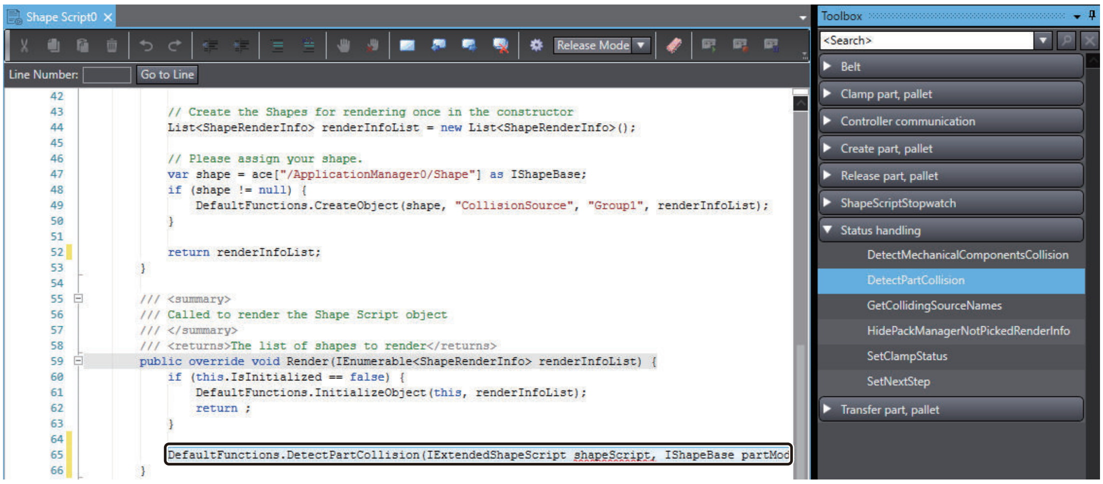
The following describes the procedure to detect a collision between a part or pallet and a mechanical component.

**1** Add the mechanical component for which to detect a collision to the Collision Filter Group.
Refer to *8-3-2 Collision Detection Setting Procedure* on page 8-16 for information on how to add a mechanical component to the Collision Filter.

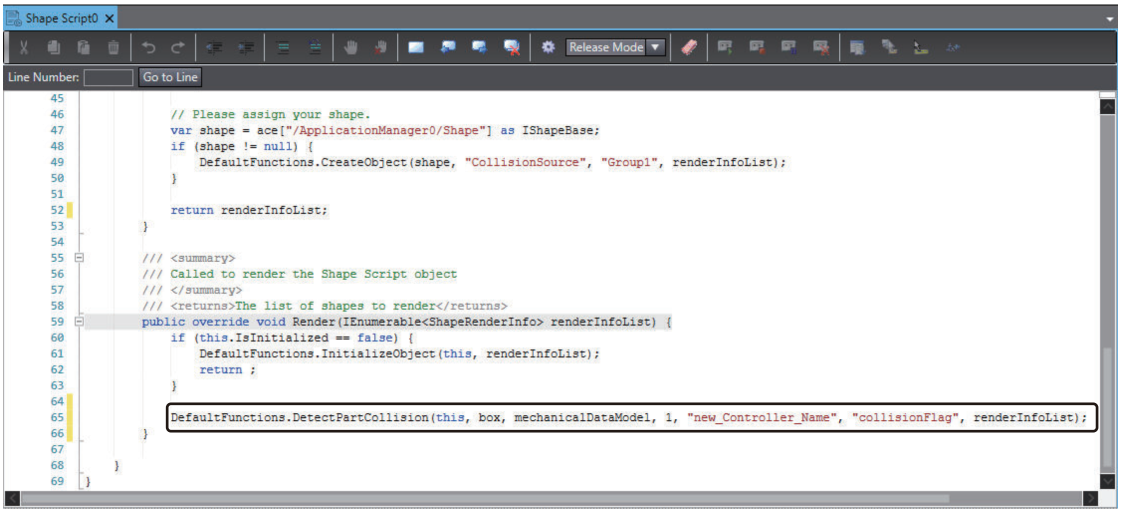
**2** Add the DetectPartCollision() function to the Render() function in the Shape Script.
Select **DetectPartCollision** under **StatusHandling** in the Toolbox and drag it to the Render() function.




The DetectPartCollision() function is inserted into the point at which you drop it.

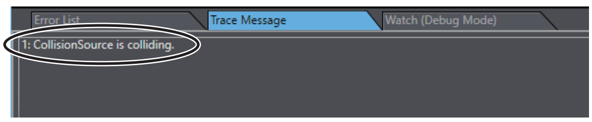



**3** Specify appropriate variables as arguments to the DetectPartCollision() function.
Refer to *DetectPartCollision* on page A-37 for details on the DetectPartCollision() function.
The following is an example where the BOOL global variable *collisionFlag* in the Controller *new_Controller_0* changes to TRUE when the part instance *box* and the mechanical component instance *mechanicalDataModel* come into contact.




**4** Start the Simulator of the Controller.

**5** Add a Shape Script that executes the DetectPartCollision() function to the Shape Script Sequence and execute the Shape Script.
Refer to *6-3-2 Setting the Execution of Shape Scripts* on page 6-12 for information on how to add a Shape Script. Refer to *8-1-3 Executing the Shape Scripts for the Part* on page 8-3 for information on how to execute a Shape Script.
Executing the Shape Script causes a log to be displayed in the **Trace Message** tab page for the Shape Script if a collision is detected.



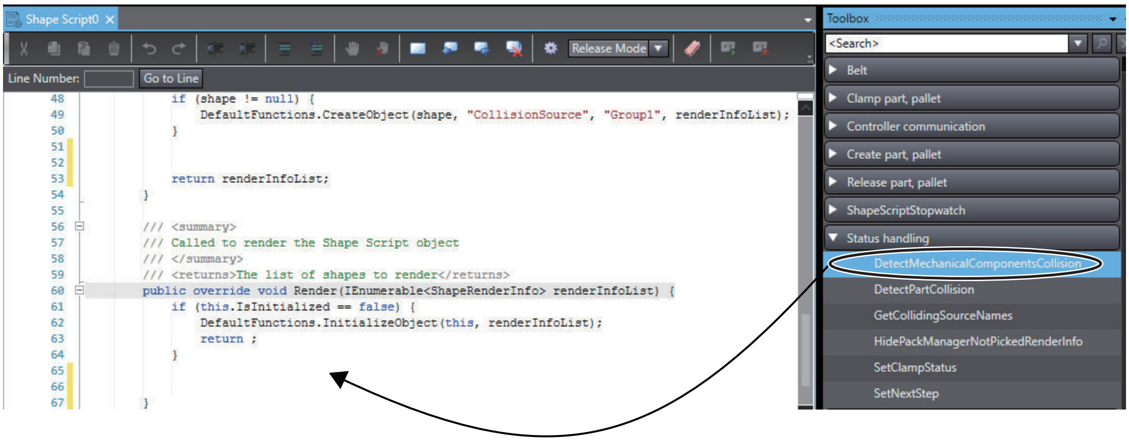
In the example in step 3, the BOOL global variable *collisionFlag* in the Controller *new_Controller_0* changes to TRUE.

## Detecting a Collision between Mechanical Components

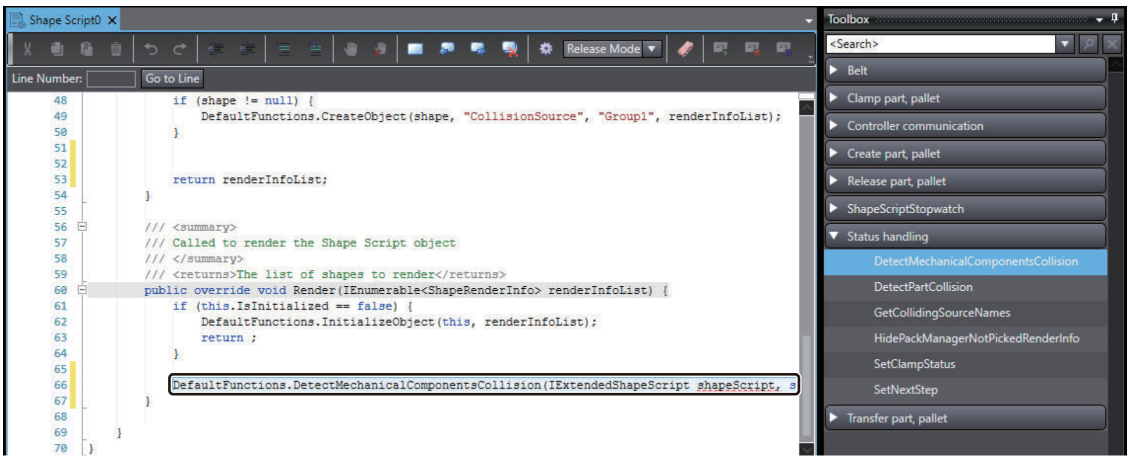
The following describes the procedure to detect a collision between mechanical components.

**1** Add the mechanical components for which to detect a collision to their Collision Filter Groups.
Refer to *8-3-2 Collision Detection Setting Procedure* on page 8-16 for information on how to add a mechanical component to the Collision Filter.

**2** Add the DetectMechanicalComponentsCollision() function to the Render() function in the Shape Script.
Select **DetectMechanicalComponentsCollision** under **StatusHandling** in the Toolbox and drag it to the Render() function.

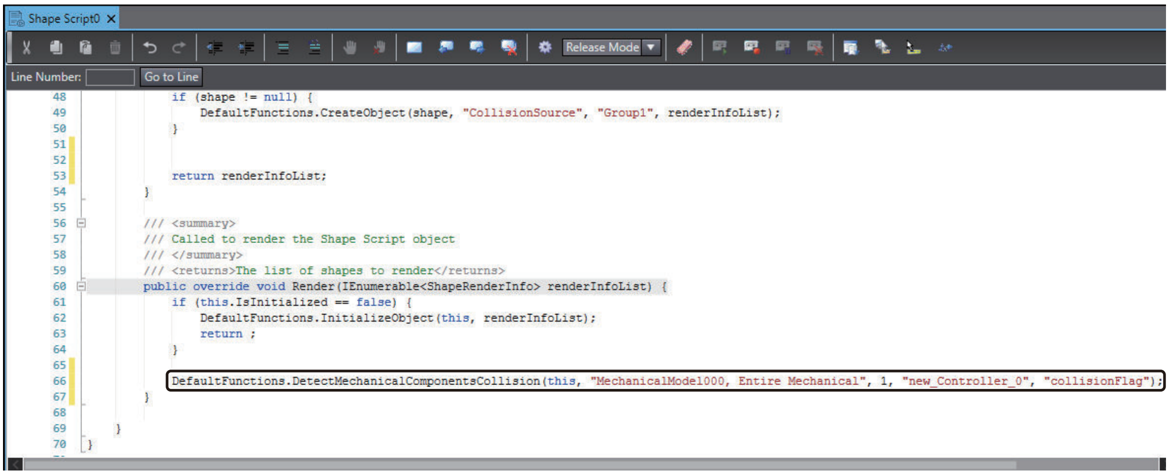



The DetectMechanicalComponentsCollision() function is inserted into the point at which you drop it.



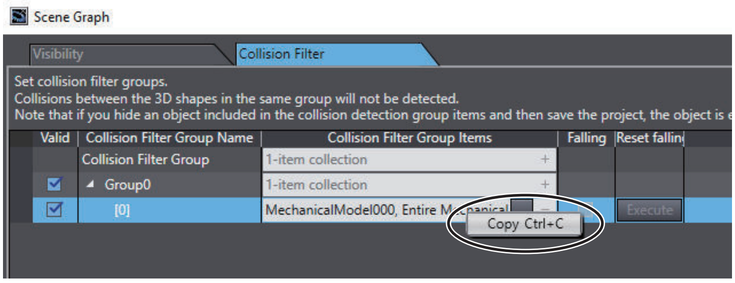

**3** Specify appropriate variables as arguments to the DetectMechanicalComponentsCollision() function.
Refer to *DetectMechanicalComponentsCollision* on page A-37 for details on the DetectMechanicalComponentsCollision() function.

The following is an example where the BOOL global variable *collisionFlag* in the Controller *new_Controller_0* changes to TRUE when the mechanical component *MehanicalModel000* collides with another mechanical component.



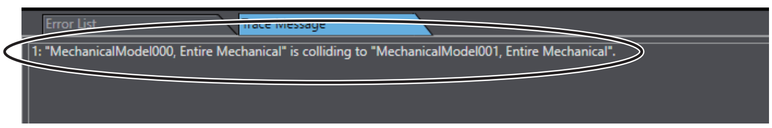

**Additional Information**

You can get the first argument of the DetectMechanicalComponentsCollision() function, i.e., the *Name which is registered in Collision Filter*, from the Collision Filter. In the **Collision Filter** tab page, copy the Collision Filter Group Items for the target mechanical component. You can paste the *Name which is registered in Collision Filter* into the Shape Script.




**4** Start the Simulator of the Controller.

**5** Add a Shape Script that executes the DetectMechanicalComponentsCollision() function to the Shape Script Sequence and execute the Shape Script.
Refer to *6-3-2 Setting the Execution of Shape Scripts* on page 6-12 for information on how to add a Shape Script. Refer to *8-1-3 Executing the Shape Scripts for the Part* on page 8-3 for information on how to execute a Shape Script.
Executing the Shape Script causes a log to be displayed in the **Trace Message** tab page for the Shape Script if a collision is detected.



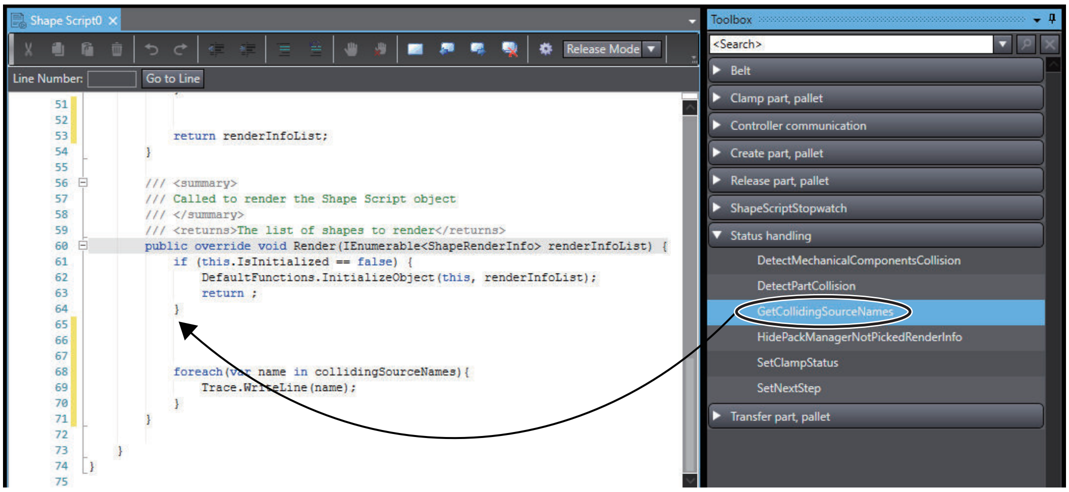
### Additional Information

If the Controller’s variables need not be changed when a collision is detected, set the arguments *Controller name* and *BOOL global variable name of the Controller* of the DetectMechanicalComponentsCollision() function to *string.Empty (blank character)*.

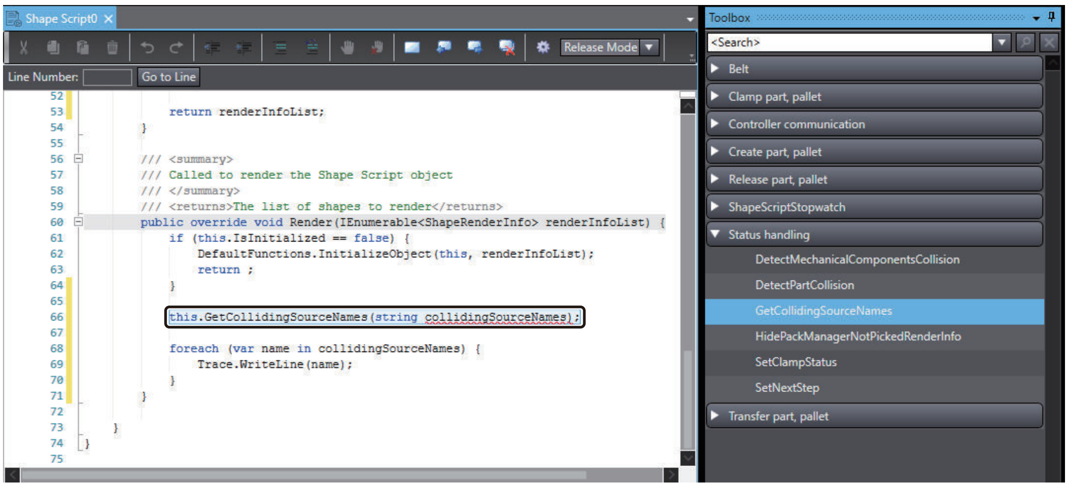
## Getting the Name of 3D Shape Data That Caused a Collision

The following describes how to get the name of 3D shape data that caused the collision.

**1** Add the GetCollidingSourceNames() function to anywhere in the Shape Script.
Select **GetCollidingSourceNames** under **StatusHandling** in the Toolbox and drag it to any point.

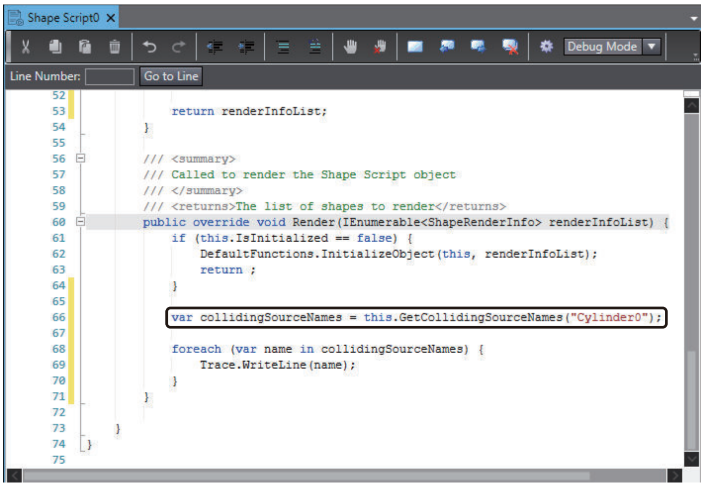


The GetCollidingSourceNames() function is inserted into the point at which you drop it.



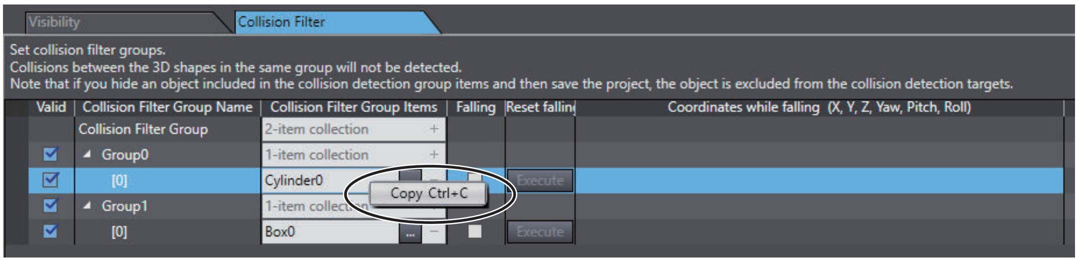
**2** Specify appropriate variable as arguments to the GetCollidingSourceNames() function.
Refer to *GetCollidingSourceNames* on page A-39 for details on the GetCollidingSourceNames() function.

The following is an example of getting the *Name of the 3D shape data* that caused a collision with *Cylinder0* and assigning it to the variable *collidingSourceNames*.



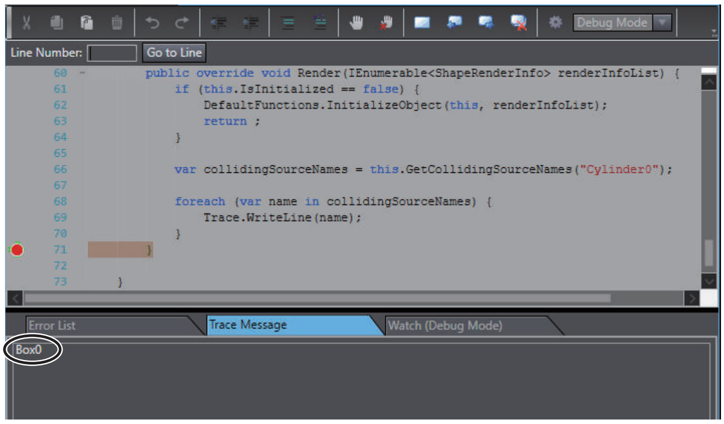

## Additional Information

You can get the argument *Name of the 3D shape data* of the GetCollidingSourceNames() function from the Collision Filter. In the **Collision Filter** tab page, copy the target Collision Filter Group Items.



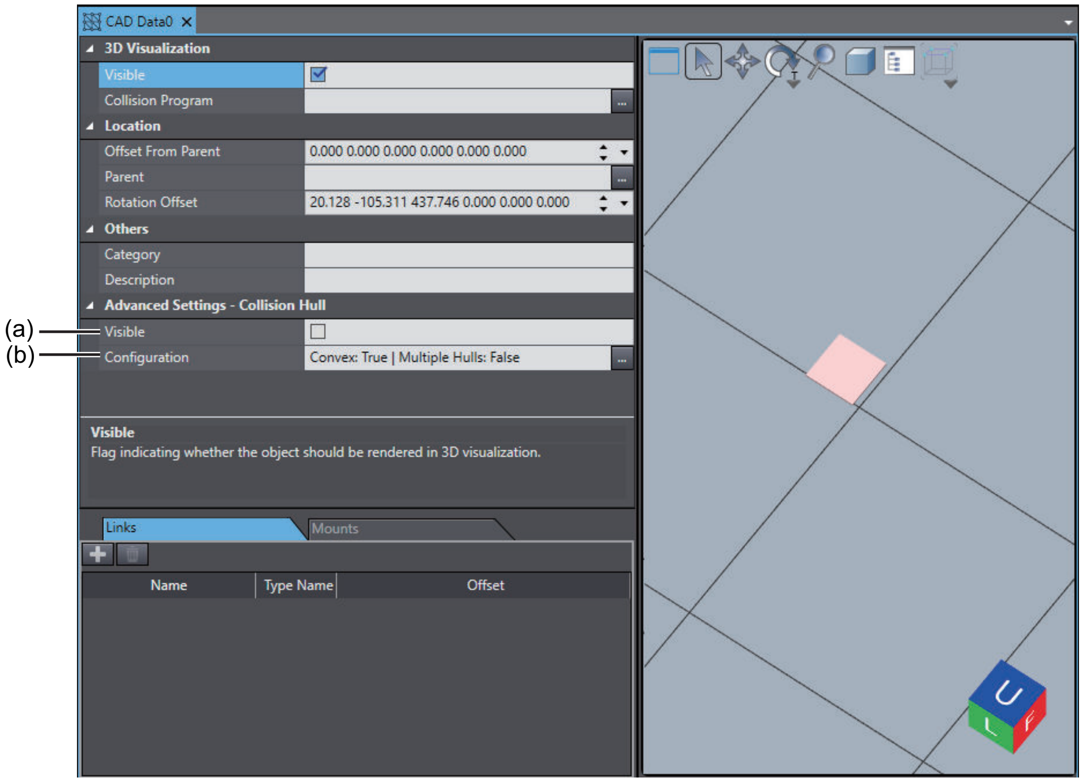

**3** Start the Simulator of the Controller.

**4** Add a Shape Script that executes the GetCollidingSourceNames() function to the Shape Script Sequence and execute the Shape Script.
Refer to *6-3-2 Setting the Execution of Shape Scripts* on page 6-12 for information on how to add a Shape Script. Refer to *8-1-3 Executing the Shape Scripts for the Part* on page 8-3 for information on how to execute a Shape Script.
Executing the Shape Script causes a log to be displayed in the **Trace Message** tab page, with *Box0* detected as the target of the collision with *Cylinder0*.




## 8-3-5 Collision Hull Settings

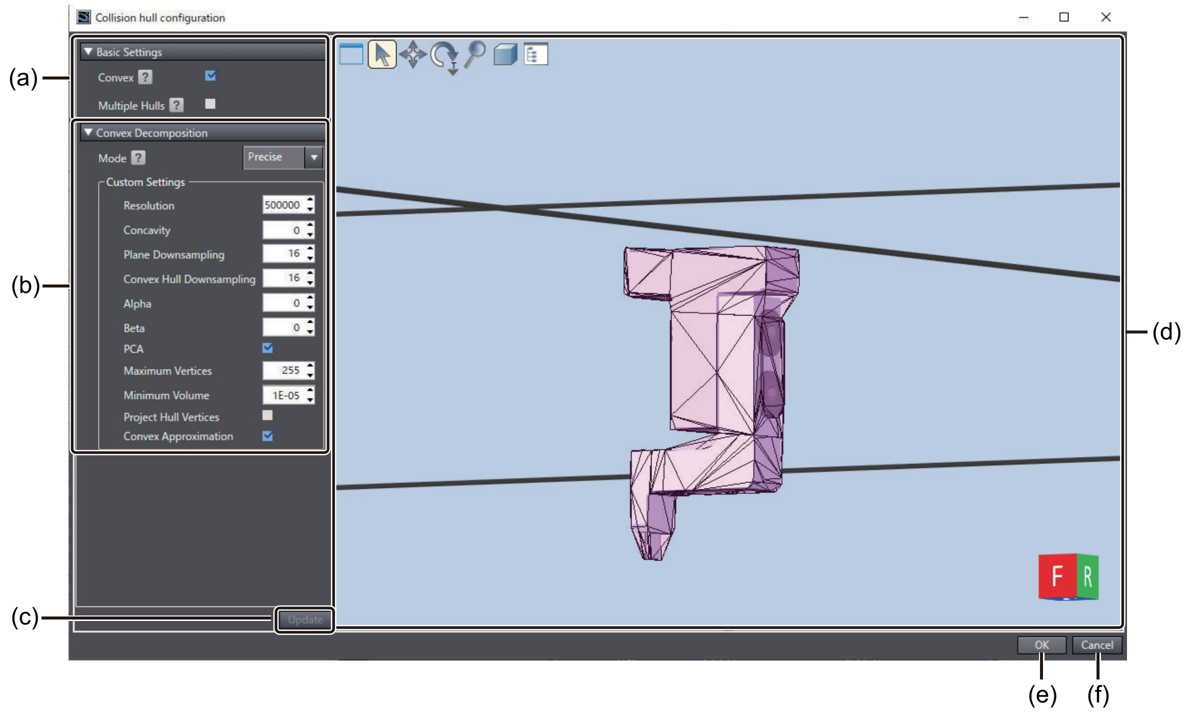
A collision hull is an aggregation of triangular pyramids that mimics 3D shape data for collision detection between 3D shape data that operates on the 3D Visualizer.
You can adjust collision hulls that affect the accuracy of collision detection by changing these settings depending on the shape and complexity of the 3D shape data.




| | Item | Description | Set value | Initial value |
|---|---|---|---|---|
| (a) | Visible | Set whether to show collision hulls. | Checked or unchecked | Unchecked |
| (b) | Configuration | Configure the collision hull settings. Click the button at the right, and then configure the advanced collision hull settings. | --- | Convex: True \| Multiple Hulls: False |

## Advanced Collision Hull Settings

Clicking the button at the right of **Configuration** under **Advanced Settings – Collision Hull** opens the **Collision hull configuration** dialog box, in which you can configure advanced collision hull settings.
The collision hull settings are listed in the following table.



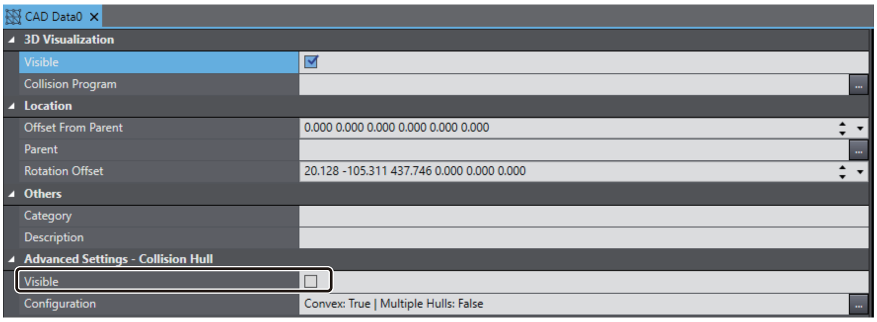

| | Item | | Description | Set value | Initial value |
|---|---|---|---|---|---|
| (a) | Basic Settings | Convex | Set whether to simplify the shapes of collision hulls. Select this check box to simplify collision hulls.<br>Simplifying collision hulls reduces the accuracy of the collision detection, but improves the speed at which images are redrawn on the 3D Visualizer. Be sure to simplify collision hulls, except for static 3D shape data such as belts and tables. Not simplifying collision hulls may prevent them from being displayed correctly.<br>If you clear this check box to disable the function to simplify collision hulls, the collision detection operates as follows.<br>• Collisions between objects are not detected.<br>• For objects that do not have simplified collision hulls, only surface collisions are detected. Internal collisions are not detected.<br>• The Falling option cannot be set. | Checked or unchecked | Checked |

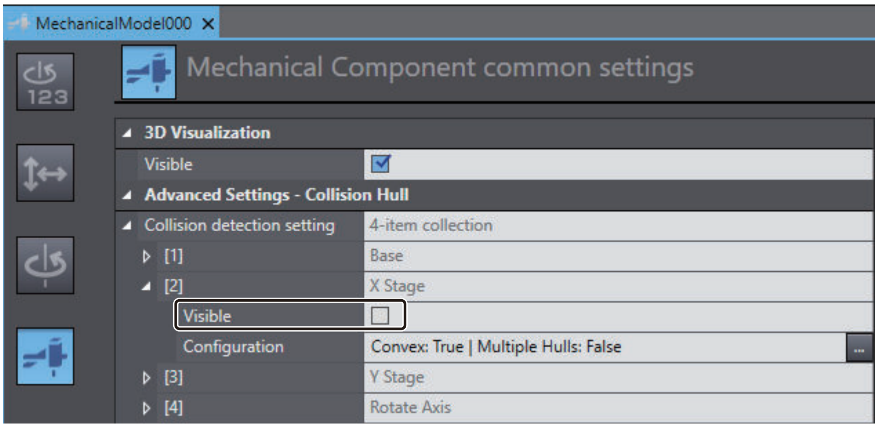
| | Item | | Description | Set value | Initial value |
|---|---|---|---|---|---|
| | | Multiple Hulls | Set whether to split the 3D shape data into multiple pieces to generate collision hulls.<br>Splitting the 3D shape data decreases the speed at which images are redrawn on the 3D Visualizer, but improves the accuracy of the collision detection.<br>You cannot set this in any of the following cases.<br>• The 3D shape data consists of a single part.<br>• The 3D shape data is generated with a Sysmac Studio version that does not support collision hull generation. | Checked or unchecked | Unchecked |
| (b) | Convex Decomposition | Mode | Set the accuracy of collision hulls. Selecting **Custom** allows you to adjust the following parameters.<br>To check the accuracy of collision hulls, change the settings and then click the **Update** button at the bottom to display the collision hulls in the 3D editing area on the right. | None, Fast, Balanced, Precise, or Custom | None |
| | | Resolution | Set the maximum number of voxels to be generated during collision hull voxelization. | 10,000 to 64,000,000 | 100,000 |
| | | Concavity | Set the largest concave surface. | 0.0 to 1.0 | 0.0 |
| | | Plane Downsampling | Set the value that controls the search granularity of the best clipping plane. | 1 to 16 | 4 |
| | | Convex Hull Downsampling | Set the value that controls the accuracy of the convex hull generation process during clipping plane selection. | 1 to 16 | 4 |
| | | Alpha | Set the value that controls the clipping bias along the symmetry plane. | 0.0 to 1.0 | 0 |
| | | Beta | Set the value that controls the clipping bias along the rotation axis. | 0.0 to 1.0 | 0 |
| | | PCA | Set whether to normalize the meshes in 3D shape data. Select this check box to normalize the meshes. | Checked or unchecked | Unchecked |
| | | Maximum Vertices | Set the maximum number of vertices in a single collision hull. | 5 to 1024 | 255 |
| | | Minimum Volume | Set the value that controls the adaptive sampling of the generated collision hulls. | 0.0 to 0.01 | 0.00001 |
| | | Project Hull Vertices | Set whether to project 3D shape data vertices on collision hulls. Select this check box to project vertices. Projecting vertices improves the accuracy of collision hulls. | Checked or unchecked | Unchecked |
| | | Convex Approximation | Set whether to approximate collision hulls. Select this check box to approximate collision hulls. Approximation means the loss of floating point precision. | Checked or unchecked | Unchecked |

| | Item | Description | Set value | Initial value |
|---|---|---|---|---|
| (c) | **Update** button | Reflect the settings on the collision hull displayed in the 3D editing area. This button is enabled after you change any settings. | --- | --- |
| (d) | 3D editing area | The area to display collision hulls. Click the **Update** button to update collision hulls. | --- | --- |
| (e) | **OK** button | Accept the settings and close the collision hull configuration dialog box. | --- | --- |
| (f) | **Cancel** button | Cancel the settings and close the collision hull configuration dialog box. | --- | --- |

## Displaying Collision Hulls

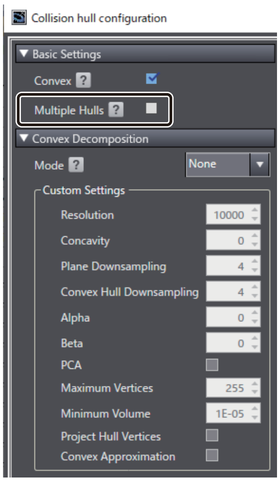
To reduce the processing load for detecting collisions, the 3D shape data is simplified.
As a result, a collision may be evaluated as not a collision although it is a collision on the 3D Visualizer, and vice versa. To check the simplified shape, open the setup tab page for the target 3D shape data and select the **Visible** check box in **Advanced Settings – Collision Hull**. Selecting this check box causes a simplified collision hull to be displayed on the 3D Visualizer.



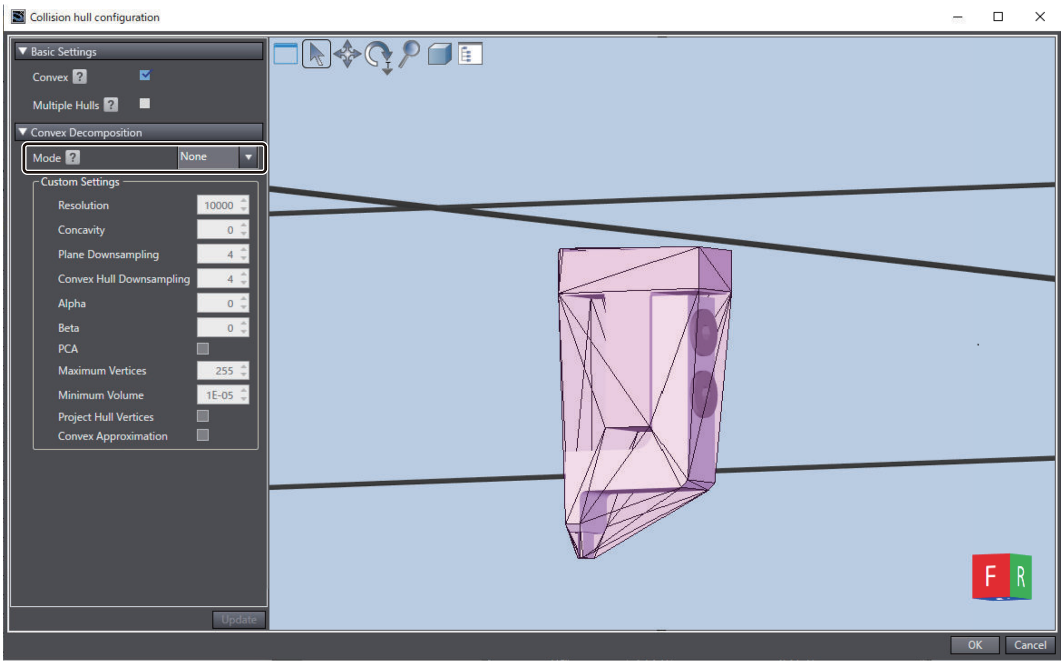

## Accuracy Adjustment for Collision Hulls

To use a collision hull that looks closer to the displayed 3D shape data for collision detection, select the **Multiple Hulls** check box in the Collision hull configuration dialog box. Selecting this check box causes the 3D shape data to be analyzed and internally split into multiple parts. Collision hulls are then generated for each of the split parts.

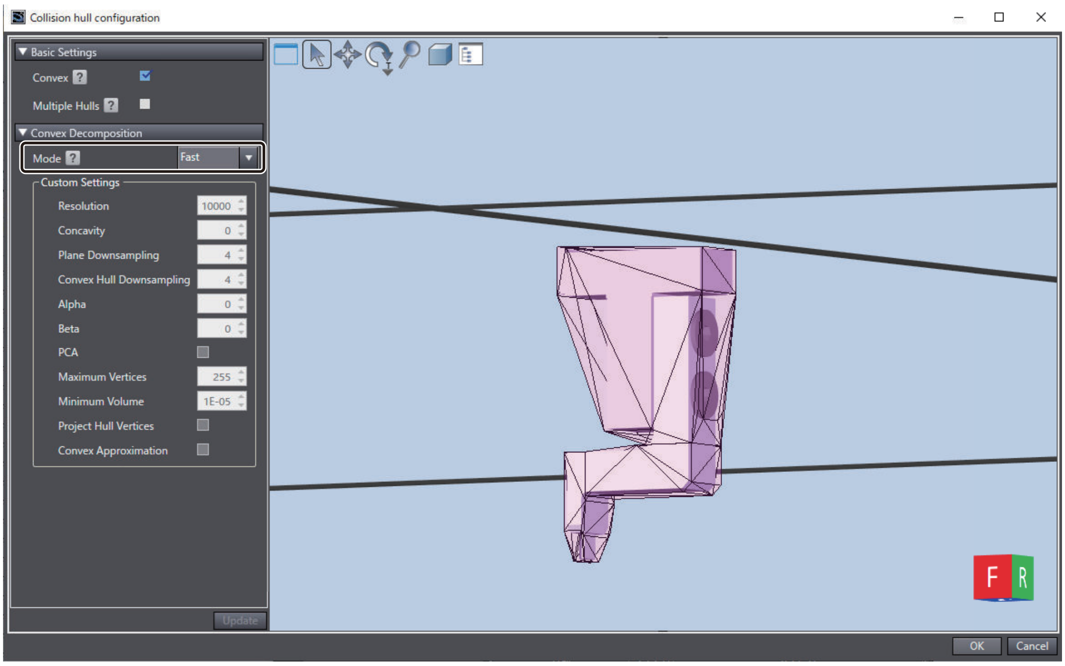



However, when a 3D shape data is made up of a single or a few objects as shown in the following figure, selecting the **Multiple Hulls** check box is not very effective. To increase the accuracy of collision detection, change the accuracy of collision hulls in the Collision hull configuration dialog box.

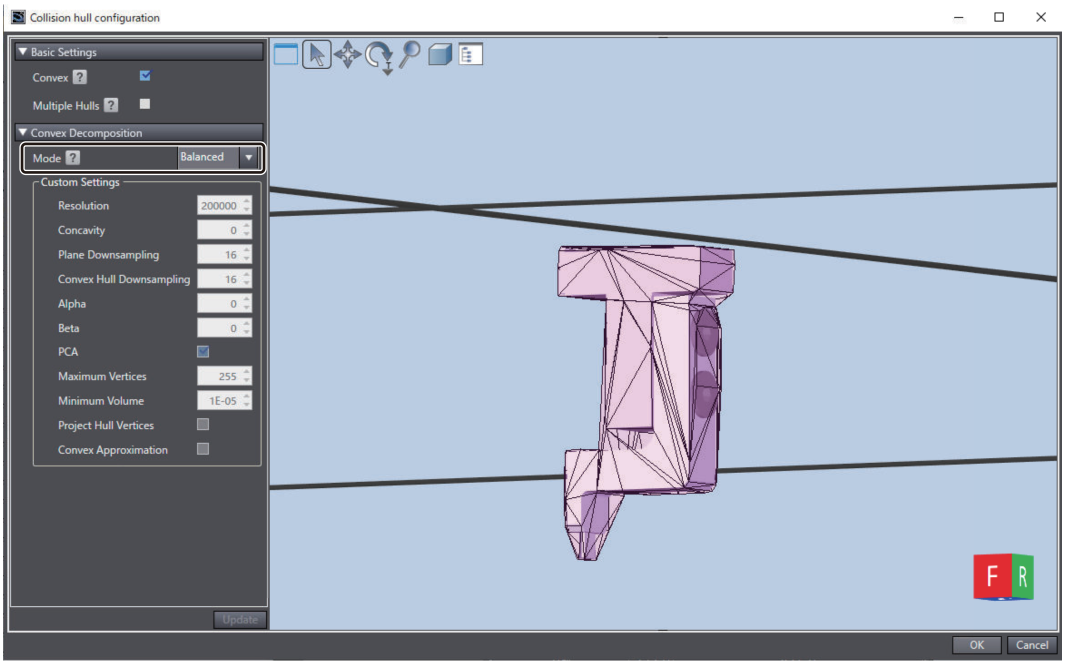
● **Mode: *None***



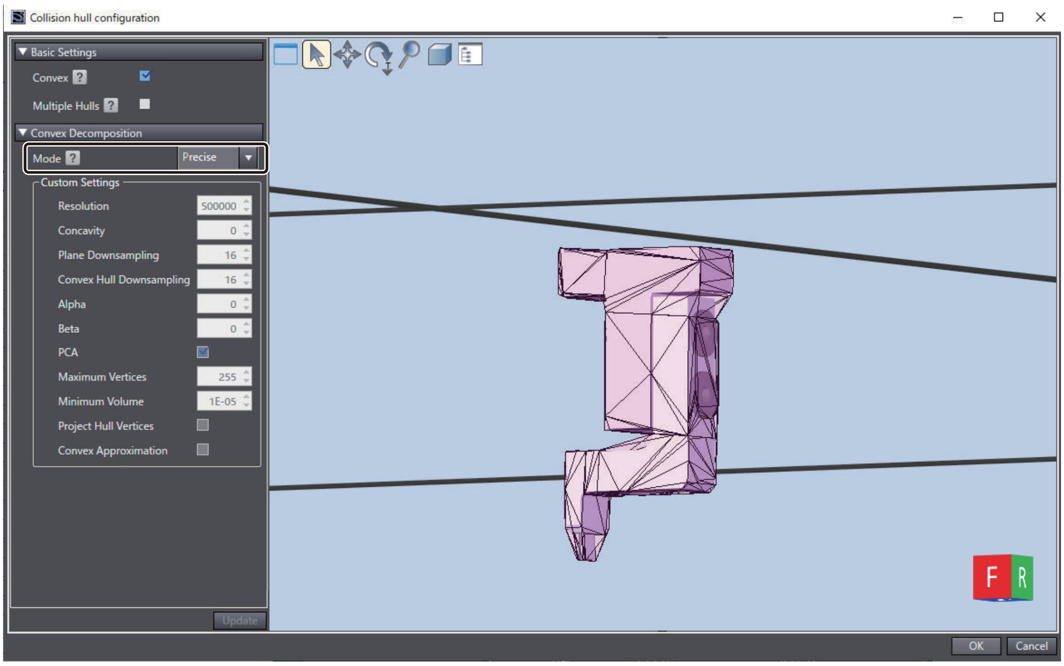
## ● Mode: *Fast*




## ● Mode: *Balanced*



## ● Mode: *Precise*



9

# 3D Simulation of Robot Integrated Systems

This section provides an overview and use of functions that are available in a 3D simulation of robot systems with an Application Controller.

# 9-1 Outline of a 3D Simulation of Robot Systems with an Application Controller

This section provides an overview of a 3D simulation of robot systems with an Application Controller that controls not only robots, but also peripheral devices (such as mechanical components).

## 9-1-1 Types of 3D Simulation

There are the following two types of 3D simulation.

- 3D Simulation of Mechanical Components Controlled by IEC 61131-3 Languages
  Create a Shape Script and use a Shape Script Sequence to create and manipulate parts and palettes, as described in *Section 6 Creating Settings and Scripts for Operating the 3D Shape Data* on page 6-1. For the shapes of the part and palette to manipulate, specify 3D shape data in the Shape Script.

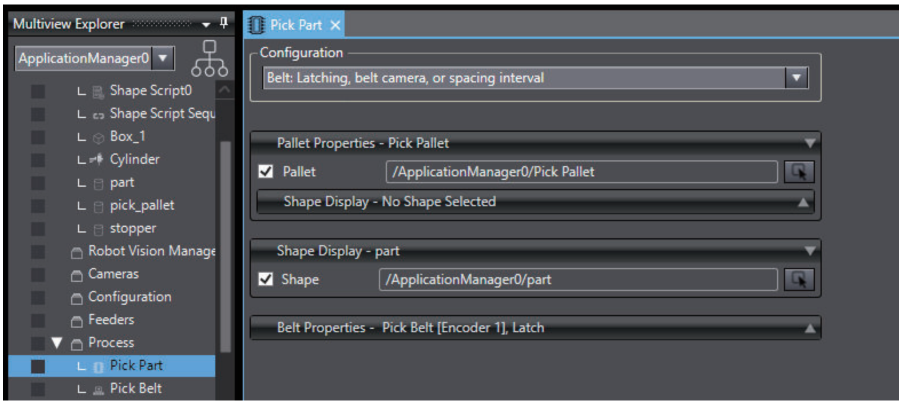
- 3D Simulation of Robot Applications
  In an Application Manager, use the emulation function of the Process Manager to operate the robot and create and manipulate parts and pallets. For the shapes of the part and palette to manipulate, specify 3D shape data with Process Manager parameters.

## 9-1-2 3D Simulation of Mechanical Components Controlled by IEC 61131-3 Languages

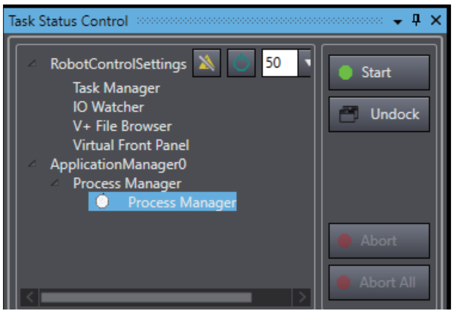
If the scope of 3D simulation is to verify the manipulation of parts with a mechanical component or robot that is controlled by IEC 61131-3 languages, use a Shape Script and a Shape Script Sequence to execute the 3D simulation.
In the Application Manager, right-click **3D Visualization** and select **Add** – **Shape Script** from the context menu to add a Shape Script. In the same way, select **Add** – **Shape Script Sequence** to add a Shape Script Sequence. Write a program with the Shape Script that you added, register it to the Shape Script Sequence, and execute the Shape Script Sequence.
Refer to *Section 6 Creating Settings and Scripts for Operating the 3D Shape Data* on page 6-1 for details on the operating procedure.

## 9-1-3 3D Simulation of Robot Applications

If the scope of 3D simulation is to verify the operation of a robot to pick up and place parts with an Application Manager, use the emulation function of the Process Manager to execute the 3D simulation.
To use 3D shape data for the part and palette that you added to **3D Visualization** in the Application Manager, select **Pick Part** and **Place Part Target** under **Process** in the Application Manager and register the shape data in **Palette Properties** and **Shape Display** on the respective tab pages.




Select **Task Status Control** from the **View** menu and, in the Task Status Control window, select the target Process Manager and click the **Start** button to start a 3D simulation.



# 9-2 3D Simulation of Robot Systems

Using a Shape Script, a Shape Script Sequence, and the emulation function of the Process Manager enables you to execute a 3D simulation of a robot and peripheral devices (mechanical components, etc.) including the part.

## 9-2-1 Using a Peripheral Device to Manipulate Parts That Were Manipulated by a Robot

If the scope of 3D simulation is to verify the operation of a robot to pick up and place parts with an Application Manager and manipulate the parts with a peripheral device (mechanical component, etc.) that is controlled by IEC 61131-3 languages, or to place parts that were picked up by a robot on a conveyor belt and move them on the conveyor, do the following.

- Add methods to the Shape Script and create in advance the parts to generate with the emulation function of the Process Manager as many as you need. In the CreateRenderInfo method, the CreateMultipleObjectsForNonCameraLatch method is used to generate parts that are used in the Process Manager.
- To allow the Shape Script to manipulate parts that were manipulated in the Process Manager, add a method for *part handover processing* to the Shape Script. In the Render method, the UpdatePartDataFromPackManager method is used to allow the Shape Script to manipulate parts that were manipulated in the Process Manager.

Program the Shape Script as follows.

```
Line Number: [    ] Go to Line
        /// The rendering data should be created once. Drawing resources
        /// are generated for each new instance of render information that
        /// is generated.
        /// </remarks>
        public override IEnumerable<ShapeRenderInfo> CreateRenderInfo() {
            this.IsInitialized = false;

            // Create the Shapes for rendering once in the constructor
            List<ShapeRenderInfo> renderInfoList = new List<ShapeRenderInfo>();

            // Please assign your shape.
            ICadData partModel = (ICadData) ace["/ApplicationManager0/HeadASSY"] as ICadData;
            DefaultFunctions.CreateMultipleObjectsForNonCameraLatch(this, partModel, "part", "partGroup", 10, renderInfoList);

            return renderInfoList;
        }

        /// <summary>
        /// Called to render the Shape Script object
        /// </summary>
        /// <returns>The list of shapes to render</returns>
        public override void Render(IEnumerable<ShapeRenderInfo> renderInfoList) {
            if (this.IsInitialized == false) {
                DefaultFunctions.InitializeObject(this, renderInfoList);
                return ;
            }

            ICadData partModel = (ICadData) ace["/ApplicationManager0/HeadASSY"];
            IBox place1Model = (IBox) ace["/ApplicationManager0/Box1"];
            IBox beltModel = (IBox) ace["/ApplicationManager0/Belt"];
            MechanicalDataModel xyzTableModel = (MechanicalDataModel) ace["/ApplicationManager0/XYZTable"];
            var controller = "new_Controller_0";

            DefaultFunctions.UpdatePartDataFromPackManager(partModel, place1Model, renderInfoList);

            DefaultFunctions.ClampPart(this, 1, xyzTableModel, partModel, place1Model, renderInfoList);
            DefaultFunctions.ReleasePart(this, 2, xyzTableModel, partModel, beltModel, renderInfoList);
            DefaultFunctions.MoveObjectOnBelt(this, controller, "move_belt", beltModel, new Transform3D(100, 0, 0), renderInfoList);
        }
```

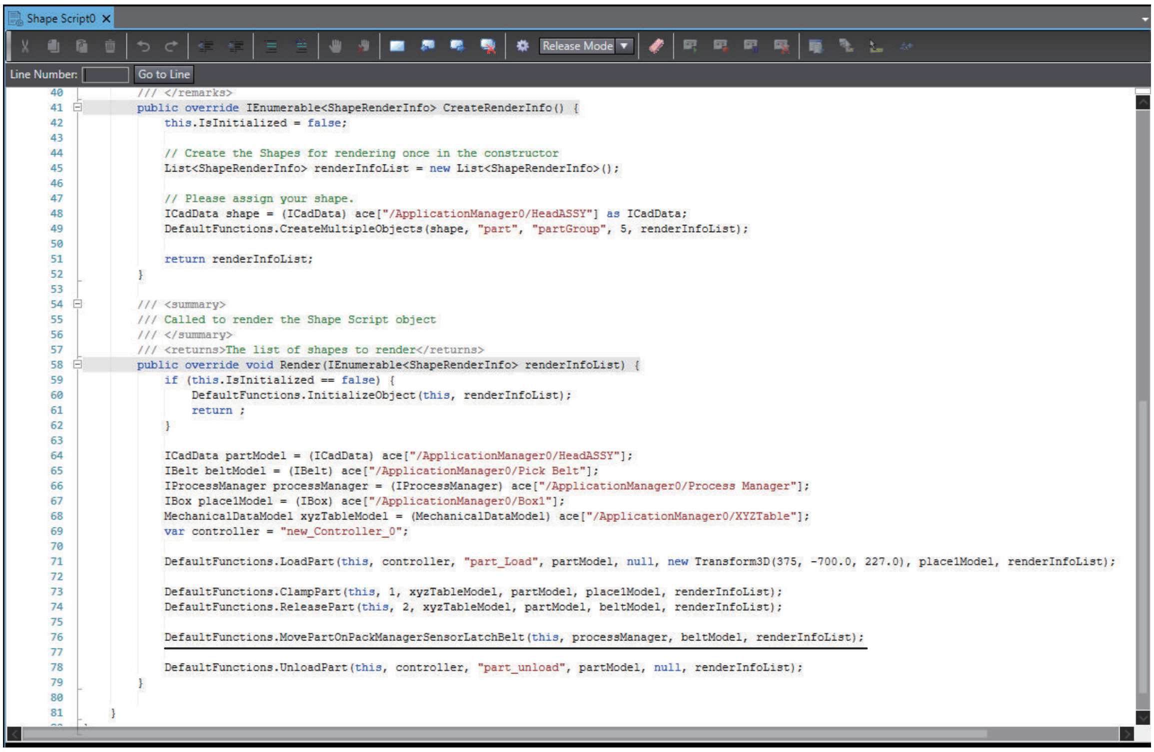
Refer to *A-1-1 Function List* on page A-2 for details on the methods.

## 9-2-2 Using a Robot to Manipulate Parts That Were Manipulated by a Peripheral Device

If the scope of 3D simulation is to verify the manipulation of parts with a peripheral device (mechanical component, etc.) that is controlled by IEC 61131-3 languages and you want to use the Process Manager to place them on a conveyor belt and use an Application Manager to pick up and place them, do the following.

- Add a method to perform the *part handover processing* to the Process Manager, which you created and operated with the Shape Script, to the Shape Script. In the Render method, use the MovePartOnPackManagerSensorLatchBelt method to move the parts that were placed on the belt by the ClampPart and ReleasePart methods to the latched position to allow the Process Manager to manipulate them.

Program the Shape Script as follows.




```
    /// </remarks>
    public override IEnumerable<ShapeRenderInfo> CreateRenderInfo() {
        this.IsInitialized = false;

        // Create the Shapes for rendering once in the constructor
        List<ShapeRenderInfo> renderInfoList = new List<ShapeRenderInfo>();

        // Please assign your shape.
        ICadData shape = (ICadData) ace["/ApplicationManager0/HeadASSY"] as ICadData;
        DefaultFunctions.CreateMultipleObjects(shape, "part", "partGroup", 5, renderInfoList);

        return renderInfoList;
    }

    /// <summary>
    /// Called to render the Shape Script object
    /// </summary>
    /// <returns>The list of shapes to render</returns>
    public override void Render(IEnumerable<ShapeRenderInfo> renderInfoList) {
        if (this.IsInitialized == false) {
            DefaultFunctions.InitializeObject(this, renderInfoList);
            return ;
        }

        ICadData partModel = (ICadData) ace["/ApplicationManager0/HeadASSY"];
        IBelt beltModel = (IBelt) ace["/ApplicationManager0/Pick Belt"];
        IProcessManager processManager = (IProcessManager) ace["/ApplicationManager0/Process Manager"];
        IBox place1Model = (IBox) ace["/ApplicationManager0/Box1"];
        MechanicalDataModel xyzTableModel = (MechanicalDataModel) ace["/ApplicationManager0/XYZTable"];
        var controller = "new_Controller_0";

        DefaultFunctions.LoadPart(this, controller, "part_Load", partModel, null, new Transform3D(375, -700.0, 227.0), place1Model, renderInfoList);

        DefaultFunctions.ClampPart(this, 1, xyzTableModel, partModel, place1Model, renderInfoList);
        DefaultFunctions.ReleasePart(this, 2, xyzTableModel, partModel, beltModel, renderInfoList);

        DefaultFunctions.MovePartOnPackManagerSensorLatchBelt(this, processManager, beltModel, renderInfoList);

        DefaultFunctions.UnloadPart(this, controller, "part_unload", partModel, null, renderInfoList);
    }

}
```

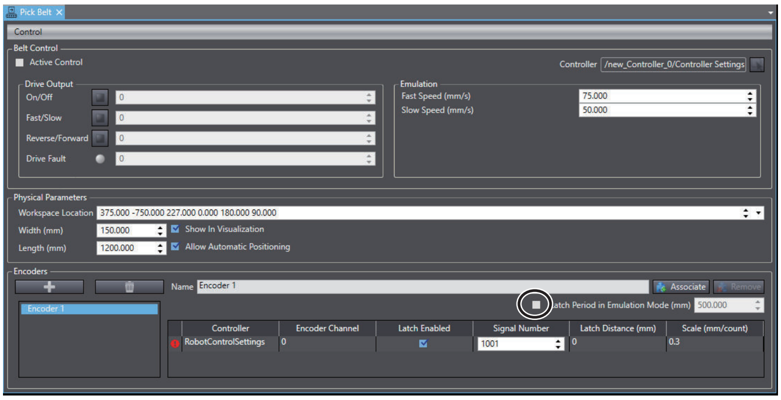
Refer to *A-1-1 Function List* on page A-2 for details on the methods.

In addition, clear the **Latch Period in Emulation Mode (mm)** check box in the Process Manager's belt settings to stop the Process Manager from generating parts.

9-2 3D Simulation of Robot Systems

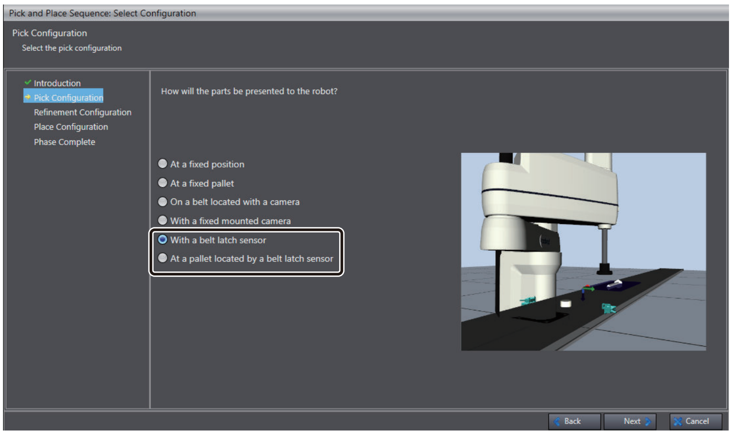
9

9-2-2 Using a Robot to Manipulate Parts That Were Manipulated by a Peripheral Device




### Precautions for Correct Use

You can use a robot application to continue to manipulate the parts and pallets that were created and manipulated by a Shape Script only if you selected **Application Sample** – **Pack Manager Sample** from the **Insert** menu and, in the **Pick and Place Sequence: Select Configuration** page displayed, select the **With a belt latch sensor** or **At a pallet located by a belt latch sensor** option in the **Pick Configuration** step.



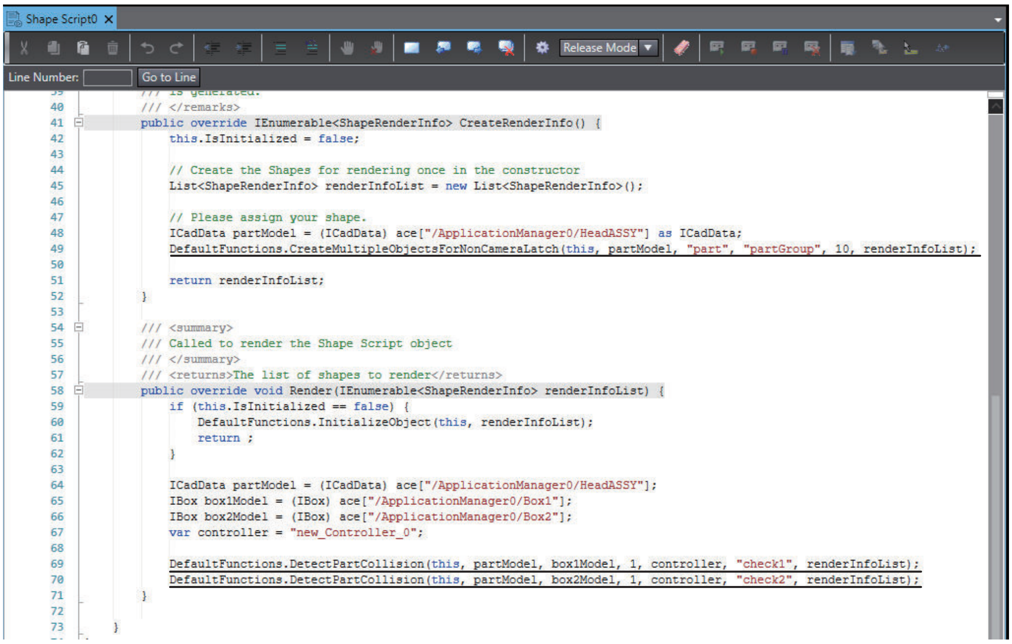

## 9-2-3 Detecting Collisions of Parts That Are Manipulated Only by a Robot

Even if the scope of 3D simulation is only to verify the operation of a robot to pick up and place parts with an Application Manager and you do not perform a 3D simulation of peripheral devices (mechanical component, etc.) that are controlled by IEC 61131-3 languages, you still need to create a Shape Script and execute a Shape Script Sequence to detect collisions between the parts.

- Add methods to the Shape Script and create in advance the parts to generate with the emulation function of the Process Manager as many as you need. In the CreateRenderInfo method, the CreateMultipleObjectsForNonCameraLatch method is used to generate parts that are used in the Process Manager.
- Add a method to allow you to recognize a collision, either by a trace message or by a change in the value of a Controller variable, to the Shape Script if a part collides with the target 3D shape data. In the Render method, the DetectPartCollision method is used to detect whether the part manipulated in the Process Manager collides with the target 3D shape data. To use a camera to handle more

than one type of part, use the CreateMultipleObjectsForCameraLatch method instead of the Create-MultipleObjectsForNonCameraLatch method.

Program the Shape Script as follows.




Refer to *A-1-1 Function List* on page A-2 for details on the methods.

10

# Useful Functions

This section describes useful 3D simulation functions.

# 10-1 Updating All CAD Data

You can update one set of CAD data that contains 3D shape data with another set of CAD data at once.
By updating CAD data in preliminary design created for feasibility check with CAD data in detailed design, you can use programs and scripts that you created without making major changes.

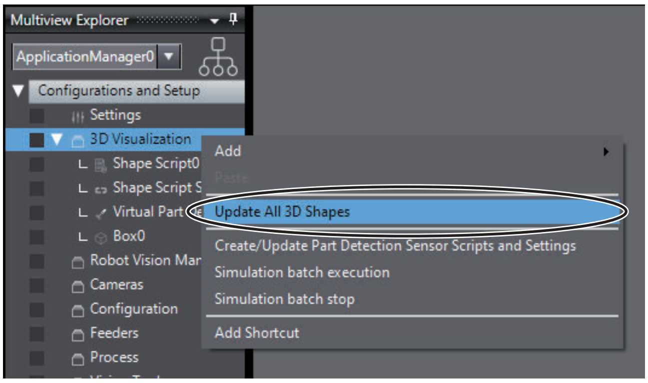
## 10-1-1 Procedure to Update All CAD Data

Use the following procedure to update all CAD data.

**Additional Information**

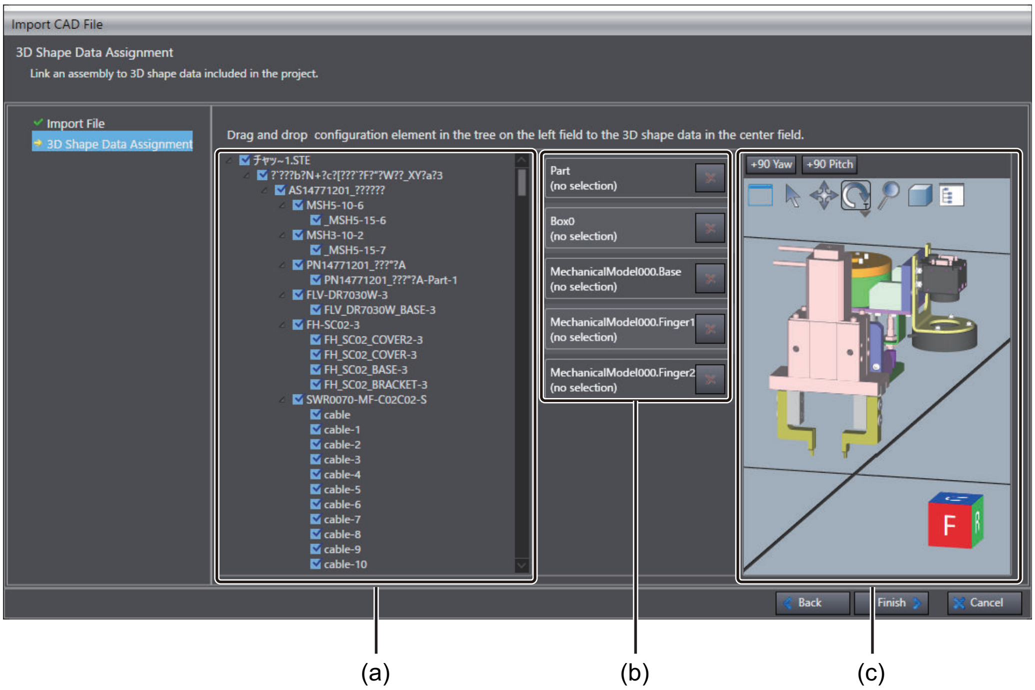
You can optionally set the mesh coarseness of CAD data at the time of import. Refer to the *Sysmac Studio Robot Integrated System Building Function with Robot Integrated CPU Unit Operation Manual (Cat. No. W595)* for details on the option settings.

**1** Right-click **3D Visualization** in the Multiview Explorer and select **Update All 3D Shapes** from the menu.



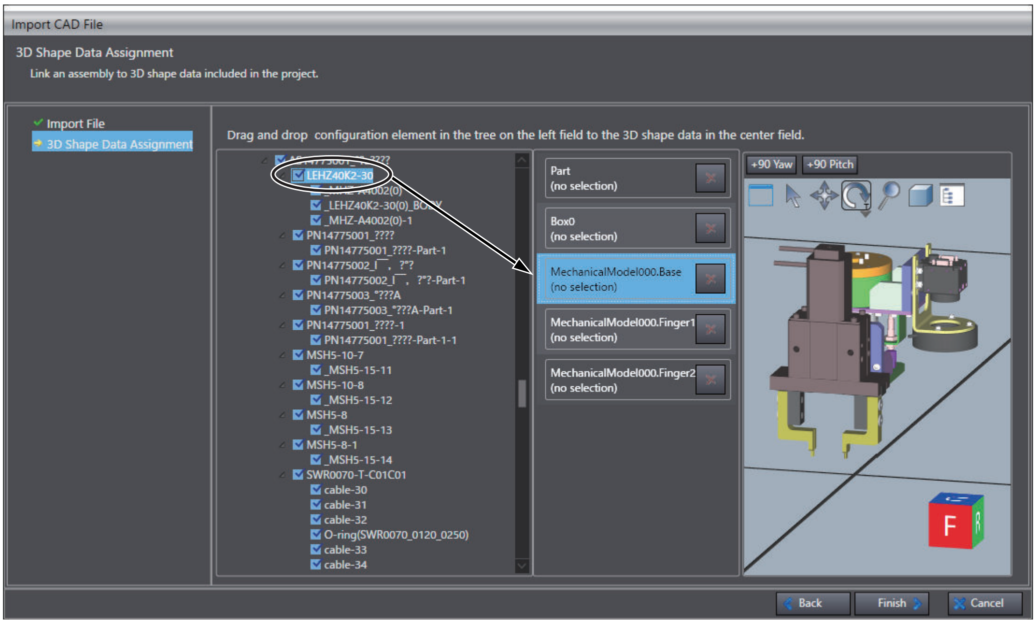

The Import CAD File wizard starts and the Select file page is displayed.
In the Select file page, you can select a single CAD file. Refer to *4-3-1 Types of Supported CAD Data Files* on page 4-10 for the types of supported CAD files.

**2** On the Select file page, select the source CAD file, and then click the **Next** button.
The **3D Shape Data Assignment** page is displayed.

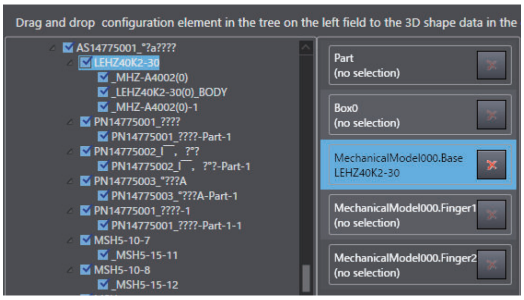



| | Item | Description |
|---|---|---|
| (a) | Source CAD data | Data in the selected CAD file is displayed. This represents the structure of parts that make up the CAD data. |
| (b) | Target 3D shape data | 3D shape data in the project is listed. |
| (c) | Source CAD data in the 3D Visualizer | A 3D view of the source CAD data is displayed. |

**3** Drag any part in the source CAD data to a target 3D shape data item.




The source CAD data is assigned to the target 3D shape data.




To cancel a selection, click the × button at the right of the name of the assigned CAD data.

**4** Assign all the necessary source CAD data, and then click the **Finish** button.
The target 3D shape data is updated with the CAD data that you specified.

# A



# Appendices

This section provides the supplemental information for main contents, such as descriptions about functions used in Shape Script.

# A-1 Functions Used in Shape Scripts

Functions enabling basic operations or processing on a part are provided for Shape Script that defines behaviors of the part.
Refer to *6-5-2 Shape Script Programming* on page 6-21 for how to input functions.

**Precautions for Correct Use**

Functions available in Shape Scripts are defined in Shape Scripts. When a Shape Script is created, all functions are defined in the Shape Script. Keep the following points in mind when you edit a Shape Script.

- Do not delete or modify the definitions of functions used in the Shape Script. Otherwise, malfunction of the related function may result. If you deleted or changed a definition by mistake, create a Shape Script, and copy and paste the code describing the behavior of the part to the new Shape Script.
- To use in an existing Shape Script the functions added or updated through an upgrade of Sysmac Studio, you must use a Shape Script that provides the new definitions of the functions. Create a Shape Script with the upgraded Sysmac Studio, and copy the code describing the behavior of the part from the existing Shape Script and paste the code in the new Shape Script.
- If you use Default Functions under Shape Script Functions, you can use Update Default Functions to update the functions. Refer to *A-1-11 Updating Functions Used in Shape Scripts* on page A-65 for the procedure.

## A-1-1 Function List

Functions available in Shape Script are listed below.

| Category | Variable name | Description | Reference |
|---|---|---|---|
| Creating and displaying parts and pallets | CreateObject | Create a part or a pallet. | *CreateObject* on page A-8 |
| | CreateMultipleObjects | Create multiple parts or pallets. | *CreateMultipleObjects* on page A-9 |
| | CreateMultipleObjects-ForCameraLatch | Create multiple parts or palettes that cooperate with the emulation function of the Process Manager. | *CreateMultipleObjectsFor-CameraLatch* on page A-9 |
| | CreateMultipleObjects-ForNonCameraLatch | Create multiple parts or palettes that are used by the emulation function of the Process Manager. | *CreateMultipleObjectsFor-NonCameraLatch* on page A-12 |
| | LoadPart | Display parts. | *LoadPart* on page A-13 |
| | LoadPallet | Display pallets. | *LoadPallet* on page A-15 |
| | LoadPartOnPallet | Display parts on a pallet. | *LoadPartOnPallet* on page A-16 |
| | UnloadPart | Hide the part by entering the BOOL variable of the Controller. | *UnloadPart* on page A-17 |
| | UnloadPallet | Hide the pallet by entering the BOOL variable of the Controller. | *UnloadPallet* on page A-18 |
| | UnloadCollidingPart | Hide the part after colliding with the designated 3D shape data. | *UnloadCollidingPart* on page A-19 |
| | UnloadCollidingPallet | Hide the pallet after colliding with the designated 3D shape data. | *UnloadCollidingPallet* on page A-19 |
| Conveying parts and pallets | PushPart | Push the part by the cylinder. | *PushPart* on page A-20 |
| | PushPallet | Push the pallet by the cylinder. | *PushPallet* on page A-21 |
| Conveyor | MoveObjectOnBelt | Move the part or pallet on the conveyor. | *MoveObjectOnBelt* on page A-22 |
| | InitializeMechanicalCon-veyor | Initialize the mechanical component *Conveyor*. | *InitializeMechanicalCon-veyor* on page A-23 |
| | MoveObjectOnMechani-calConveyor | Move the parts or pallets on the mechanical component *Conveyor*. | *MoveObjectOnMechani-calConveyor* on page A-23 |

| Category | Variable name | Description | Reference |
|---|---|---|---|
| Clamping parts and pallets | ClampPart | Pick up the part. | *ClampPart* on page A-24 |
| | ClampPallet | Pick up the pallet. | *ClampPallet* on page A-25 |
| | ClampPartOnPallet | Pick up the part on the pallet. | *ClampPartOnPallet* on page A-26 |
| | ClampPartBySignal | Pick up the part on a grasping signal. | *ClampPartBySignal* on page A-27 |
| | ClampPalletBySignal | Pick up the pallet on a grasping signal. | *ClampPalletBySignal* on page A-28 |
| | ClampPartOnPalletBySignal | Pick up the part on the pallet on a grasping signal. | *ClampPartOnPalletBySignal* on page A-29 |
| Placing parts and pallets | ReleasePart | Place the part picked up. | *ReleasePart* on page A-30 |
| | ReleasePallet | Place the pallet picked up. | *ReleasePallet* on page A-31 |
| | ReleasePartOnPallet | Place the part which picked up to the pallet. | *ReleasePartOnPallet* on page A-32 |
| | ReleasePartBySignal | Release (place) the part according to a grasping signal of the robot hand. | *ReleasePartBySignal* on page A-32 |
| | ReleasePalletBySignal | Release (place) the pallet according to a grasping signal of the robot hand. | *ReleasePalletBySignal* on page A-34 |
| | ReleasePartOnPalletBySignal | Release (place) the part to the pallet according to a grasping signal of the robot hand. | *ReleasePartOnPalletBySignal* on page A-35 |
| Changing and detecting status | SetNextStep | Change the value of stepId. | *SetNextStep* on page A-36 |
| | DetectPartCollision | Detect that the part or pallet collides with the mechanical component. | *DetectPartCollision* on page A-37 |
| | DetectMechanicalComponetsCollision | Detect that the mechanical component collides with another mechanical component. | *DetectMechanicalComponentsCollision* on page A-37 |
| | SetClampStatus | Write the grasping (colliding) status of the robot hand to the variable of the Controller. | *SetClampStatus* on page A-38 |
| | GetCollidingSourceNames | Get the names of all 3D shape data that collided with the specified 3D shape data. | *GetCollidingSourceNames* on page A-39 |
| Sharing variables with the Controller | GetBoolVariable | Read the value of a BOOL variable from the specified Controller. | *GetBoolVariable* on page A-40 |
| | SetBoolVariable | Write values of BOOL variables of the specified Controller. | *SetBoolVariable* on page A-40 |

| Category | Variable name | Description | Reference |
|---|---|---|---|
| | GetIntegerVariable | Read the value of a DINT variable from the specified Controller. | *GetIntegerVariable* on page A-41 |
| | SetIntegerVariable | Write values to DINT variables of the specified Controller. | *SetIntegerVariable* on page A-41 |
| | Get**Variable | Read the value of an integer variable from the specified Controller. For "**" of the function name, specify the data type of the Controller variable. | *Get**Variable* on page A-42 |
| | Set**Variable | Write values to integer variables of the specified Controller. For "**" of the function name, specify the data type of the Controller variable. | *Set**Variable* on page A-43 |
| | GetByteArrayVariable | Read the values of a BYTE array variable from the specified Controller. | *GetByteArrayVariable* on page A-44 |
| | SetByteArrayVariable | Write values to a BYTE array variable in the specified Controller. | *SetByteArrayVariable* on page A-44 |
| | GetBoolArrayVariable | Read the values of a BOOL array variable from the specified Controller. | *GetBoolArrayVariable* on page A-45 |
| | SetBoolArrayVariable | Write values to a BOOL array variable in the specified Controller. | *SetBoolArrayVariable* on page A-45 |
| | GetIntegerArrayVariable | Read the values of a DINT array variable from the specified Controller. | *GetIntegerArrayVariable* on page A-46 |
| | SetIntegerArrayVariable | Write values to a DINT array variable in the specified Controller. | *SetIntegerArrayVariable* on page A-46 |
| | GetWordArrayVariable | Read the values of a WORD array variables from the specified Controller. | *GetWordArrayVariable* on page A-47 |
| | SetWordArrayVariable | Write values to a WORD array variable in the specified Controller. | *SetWordArrayVariable* on page A-47 |
| | GetDwordArrayVariable | Read the values of a DWORD array variable from the specified Controller. | *GetDwordArrayVariable* on page A-48 |
| | SetDwordArrayVariable | Write values to a DWORD array variable in the specified Controller. | *SetDwordArrayVariable* on page A-48 |

| Category | Variable name | Description | Reference |
|---|---|---|---|
| | GetLwordArrayVariable | Read the values of an LWORD array variable from the specified Controller. | *GetLwordArrayVariable* on page A-49 |
| | SetLwordArrayVariable | Write values to an LWORD array variable in the specified Controller. | *SetLwordArrayVariable* on page A-50 |
| | GetRealArrayVariable | Read the values of a REAL array variable from the specified Controller. | *GetRealArrayVariable* on page A-50 |
| | SetRealArrayVariable | Write values to a REAL array variable in the specified Controller. | *SetRealArrayVariable* on page A-51 |
| | GetLrealArrayVariable | Read the values of an LREAL array variable from the specified Controller. | *GetLrealArrayVariable* on page A-51 |
| | SetLrealArrayVariable | Write values to an LREAL array variable in the specified Controller. | *SetLrealArrayVariable* on page A-52 |
| | GetStringArrayVariable | Read the values of a STRING array variable from the specified Controller. | *GetStringArrayVariable* on page A-52 |
| | SetStringArrayVariable | Write values to a STRING array variable in the specified Controller. | *SetStringArrayVariable* on page A-53 |
| | Get**ArrayVariable | Read the values of an integer array variable from the specified Controller. For "**" of the function name, specify the data type of the Controller variable. | *Get**ArrayVariable* on page A-53 |
| | Set**ArrayVariable | Write values to an integer array variable in the specified Controller. For "**" of the function name, specify the data type of the Controller variable. | *Set**ArrayVariable* on page A-54 |
| | CreateGetVariableList | Create a new read variable list, which is used to read multiple variables from the Controller. | *CreateGetVariableList* on page A-55 |
| | AddToGetVariableList | Add the variable to read from the Controller to the read variable list. | *AddToGetVariableList* on page A-56 |
| | GetVariableValues | Read the values of multiple variables from the Controller at a time. | *GetVariableValues* on page A-57 |

| Category | Variable name | Description | Reference |
|---|---|---|---|
| | GetValueFromVariableValueList | Get the value of a variable that you read with the GetVariableValues() function. | *GetValueFromVariableValueList* on page A-57 |
| | CreateSetVariableList | Create a new write variable list, which is used to write multiple variables from the Controller. | *CreateSetVariableList* on page A-58 |
| | AddToSetVariableList | Add the variable and value to write to the Controller to the write variable list. | *AddToSetVariableList* on page A-59 |
| | SetVariableValues | Write values to multiple Controller variables at a time. | *SetVariableValues* on page A-60 |
| | GetCurrentControllerTime | Read the internal simulation time of the specified Controller. | *GetCurrentControllerTime* on page A-60 |
| Cooperation with the Process Manager | MovePartOnPackManagerSensorLatchBelt | Move the part that is present on the conveyor in the Process Manager to the latched position to allow it to be continuously manipulated by the emulation function of the Process Manager. | *MovePartOnPackManagerSensorLatchBelt* on page A-61 |
| | MovePalletOnPackManagerSensorLatchBelt | Move the pallet that is present on the conveyor in the Process Manager to the latched position to allow it to be continuously manipulated by the emulation function of the Process Manager. | *MovePalletOnPackManagerSensorLatchBelt* on page A-61 |
| | SetPackManagerPartPosition | Write the position of the part that is manipulated in the Process Manager to a Controller variable. | *SetPackManagerPartPosition* on page A-62 |
| | UpdatePartDataFromPackManager | Allow the Shape Script to continue to manipulate the part that was picked up and placed in the Process Manager. | *UpdatePartDataFromPackManager* on page A-63 |
| | UpdatePartDataOnPalletFromPackManager | Allow the Shape Script to continue to manipulate the part that was picked up and placed on the pallet in the Process Manager. | *UpdatePartDataOnPalletFromPackManager* on page A-64 |

| Category | Variable name | Description | Reference |
|---|---|---|---|
| Measuring Execution Time of Shape Scripts | StartStopwatch | Start measurement of execution time of a Shape Script. | *StartStopwatch* on page A-64 |
| | StopStopwatch | Stop measurement of execution time of a Shape Script. | *StopStopwatch* on page A-65 |

## A-1-2 Creating and Displaying Parts and Pallets

### CreateObject

Create a part or a pallet.

#### ● Function Call

| | |
|---|---|
| **Function** | CreateObject( IShapeBase objectModel, string name, string collisionGroup, List<ShapeRenderInfo> renderInfoList ) |
| **Used in** | CreateRenderInfo() |

#### ● Arguments

| Argument | Description | Notation example |
|---|---|---|
| IShapeBase objectModel | Ahead of this function, define variables to which the CAD data model (e.g. ICylinder for a cylinder) and the 3D shape data registered in the project (the device name and the 3D shape data name) have been assigned in advance. | ICylinder objectModel = (ICylinder) ace["/ApplicationManager0/Part"]; |
| string name | Specify an instance name used in the Render function by string. | "Part" |
| string collisionGroup | Specify the name of Collision Filter Group to be registered by string. | "PartGroup" |
| List<ShapeRenderInfo> renderInfoList | List structure consists of the 3D shape data name, instance name, Collision Filter Group name mentioned above. It is called Part/Pallet Instance List. | --- |

#### ● Description

Instantiate the 3D shape data, which designated as *objectModel*, in the name of *name* to handle the data as a part.

If you use the part instantiated through this function in the Render function, it is necessary to specify names of the part's 3D shape data and Part/Pallet Instance List. Some functions require to specify the instance name.

**Version Information**

The CreateObject() function causes a compiling error in Sysmac Studio version 1.40 if it is included in a Shape Script that is newly created in Sysmac Studio version 1.42 or higher.

## CreateMultipleObjects

Create multiple parts or pallets.

### ● Function Call

| | |
|---|---|
| **Function** | CreateMultipleObjects( IShapeBase objectModel, string name, string collisionGroup, int count, List<ShapeRenderInfo> renderInfoList ) |
| **Used in** | CreateRenderInfo() |

### ● Arguments

| Argument | Description | Notation example |
|---|---|---|
| IShapeBase object-Model | Ahead of this function, define variables to which the CAD data model (e.g. IBox for a box) and the 3D shape data registered in the project (the device name and the 3D shape data name) have been assigned in advance. | ICylinder objectModel = (ICylinder) ace["/ApplicationManager0/Part"]; |
| string name | Specify an instance name used in the Render function by string. | "Part" |
| string collisionGroup | Specify the name of Collision Filter Group to be registered by string. | "PartGroup" |
| int count | Specify the number of parts or pallets to create. | --- |
| List<ShapeRenderIn-fo> renderInfoList | List structure consists of the 3D shape data name, instance name, Collision Filter Group name mentioned above. It is called Part/Pallet Instance List. | --- |

### ● Description

Instantiate the 3D shape data, which designated as *objectModel*, in the name of *name* to handle the data as multiple parts.
If you use the part instantiated through this function in the Render function, it is necessary to specify names of the part's 3D shape data and Part/Pallet Instance List. Some functions require to specify the instance name.

Register the 3D shape data in the Part/Pallet Instance List as many as the number specified by *count*. Names to be registered in the Part/Pallet Instance List consists of the string specified by *name*, and "_N" (N=0,1,2, ..., count-1).
Example: *name* is "Part", and N = 3
"Part_0"
"Part_1"
"Part_2"

## CreateMultipleObjectsForCameraLatch

Create multiple parts or palettes that cooperate with the emulation function of the Process Manager. When Vision Sensors are used in the part function of the Process Manager, you can use this function to generate a part to enable cooperation.

### Function Call

| | |
|---|---|
| **Function** | CreateMultipleObjectsForCameraLatch(IExtendedShapeScript shapeScript, IShapeBase objectModel, string name, string collisionGroup, int count, List<ShapeRenderInfo> renderInfoList, string objectName, bool isDisplayNumber = false) |
| **Used in** | CreateRenderInfo() |

### Arguments

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shapeScript | Specify the Shape Script to use in this function. | --- |
| IShapeBase objectModel | Ahead of this function, define variables to which the CAD data model (e.g. IBox for a box) and the 3D shape data registered in the project (the device name and the 3D shape data name) have been assigned in advance. | ICylinder objectModel = (ICylinder) ace["/ApplicationManager0/Part"]; |
| string name | Specify an instance name used in the Render function by string. | "Part" |
| string collisionGroup | Specify the name of Collision Filter Group to be registered by string. | "PartGroup" |
| int count | Specify the number of parts or pallets to create. | 10 |
| List<ShapeRenderInfo> renderInfoList | List structure consists of the 3D shape data name, instance name, Collision Filter Group name mentioned above. It is called Part/Pallet Instance List. | --- |
| string objectName | Set which part to link to, among the parts controlled by the Process Manager. | "Part0" |
| bool isDisplayNumber = false | Set whether to display the instance numbers of the parts generated by the Process Manager. The numbers are not always displayed in the order of generation. This argument can be omitted. If it is omitted, false is set. | true |

### Description

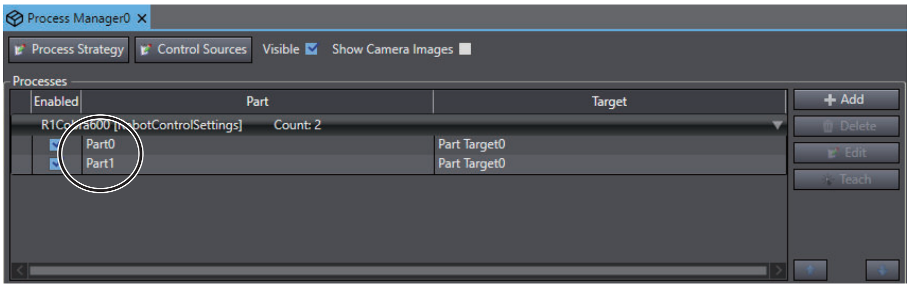
Instantiate the 3D shape data that is designated as *objectModel* with the name of *name* so that the data can be handled as a part by the emulation function.
Register the 3D shape data in the Part/Pallet Instance List as many as the number specified by *count*. Names to be registered in the Part/Pallet Instance List consists of the string specified by *name*, and "_N" (N=0,1,2, ..., count-1).
Example: *name* is "Part", and N = 3
"Part_0"
"Part_1"
"Part_2"

To generate multiple types of parts with different 3D shape data, execute this function for each type of part.
The following shows the sample code when **Belt: Latching, belt camera, or spacing interval** is set in the configuration of "Part0" in the Process Manager.

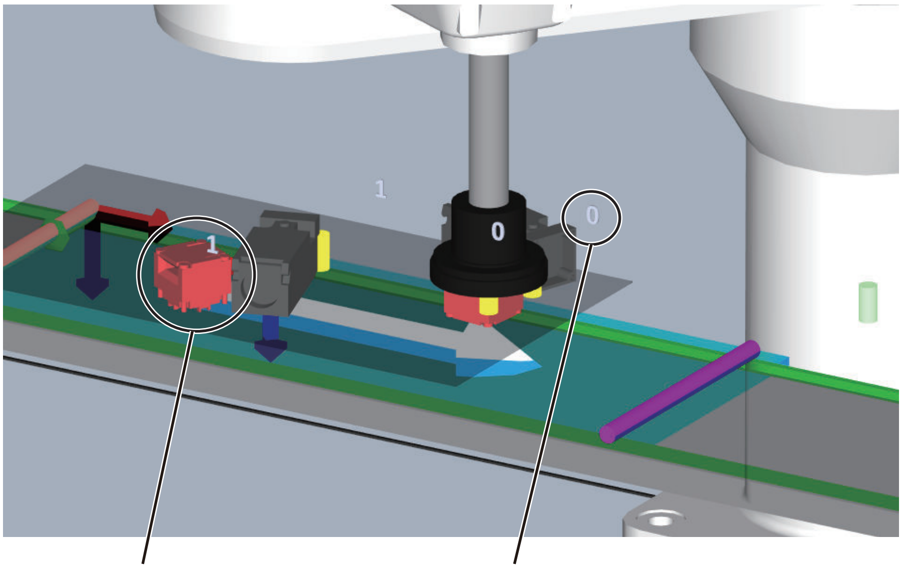
```
40      public override IEnumerable<ShapeRenderInfo> CreateRenderInfo() {
41          this.IsInitialized = false;
42
43          // Create the Shapes for rendering once in the constructor
44          List<ShapeRenderInfo> renderInfoList = new List<ShapeRenderInfo>();
45
46          var caddata0 = (ICadData) ace["/ApplicationManager0/Part0"];
47          var caddata1 = (ICadData) ace["/ApplicationManager0/Part1"];
48
49          DefaultFunctions.CreateMultipleObjectsForCameraLatch(this, cadData0, "Part0", "Group1", 50, renderInfoList, "Part0", true);
50          DefaultFunctions.CreateMultipleObjectsForCameraLatch(this, cadData1, "Part1", "Group1", 50, renderInfoList, "Part1", true);
51
52          return renderInfoList;
53      }
```

In this example, two types of parts are shown. First, define the 3D shape data of the part to use in lines 46 to 47. Next, in lines 49 to 50, use the CreateMultipleObjectsForCameraLatch() function to generate the part.

The arguments are described in the following table.

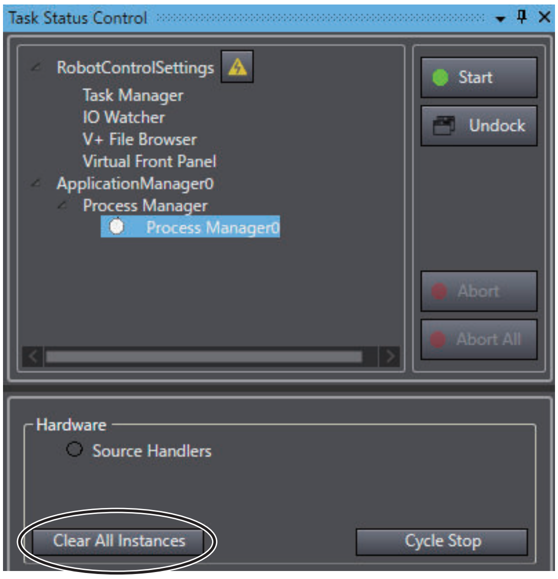
| Argu-ment | Description |
|---|---|
| caddata0 | Specify the 3D shape data defined in lines 46 to 47. |
| "Part0" | Specify the instance name. To define multiple types of parts as shown in the example, part names must not be duplicated. Parts with duplicated names will not be displayed. |
| "Group1" | Specify the name of the Collision Filter Group. To detect collisions of the parts generated in lines 49 and 50, specify a different group name for each of them. |
| 50 | Specify the number of instances of the part to generate. |
| renderIn-foList | Specify the renderInfoList. |
| "Part0" | Specify the part name defined in the Process Manager settings.<br><br> |
| true | Specify whether to display the part number. When this is set to "true", the part number will be displayed on the 3D Visualizer. This argument can be omitted. If it is omitted, the part number will not be displayed. |

The following is a display example of the 3D Visualizer when you execute the Shape Script shown in this example. Two types of parts are displayed corresponding to the generated instances.




**Precautions for Correct Use**

This function requires that the instance numbers of the parts generated by the CreateMultipleObjectsForCameraLatch() function must be the same as those of parts generated by the Process Manager. To match the instance numbers, clear all instances in the Process Manager of the **Task Status Control** window before you execute the Shape Script.




## CreateMultipleObjectsForNonCameraLatch

Create multiple parts or palettes that are used by the emulation function of the Process Manager. This function cannot be used if you use a camera to recognize more than one type of part at a time.

## ● Function Call

| | |
|---|---|
| **Function** | CreateMultipleObjectsForNonCameraLatch(IExtendedShapeScript shapeScript, IShapeBase objectModel, string name, string collisionGroup, int count, List<ShapeRenderInfo> renderInfo-List, bool isDisplayNumber = false) |
| **Used in** | CreateRenderInfo() |

## ● Arguments

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shapeScript | Specify the Shape Script to use in this function. | --- |
| IShapeBase objectMo-del | Ahead of this function, define variables to which the CAD da-ta model (e.g. IBox for a box) and the 3D shape data regis-tered in the project (the device name and the 3D shape data name) have been assigned in advance. | ICylinder objectMo-del =(ICylin-der)ace["/Applica-tionManager0/ Part"]; |
| string name | Specify an instance name used in the Render function by string. | "Part" |
| string collisionGroup | Specify the name of Collision Filter Group to be registered by string. | "PartGroup" |
| int count | Specify the number of parts or pallets to create. | 10 |
| List<ShapeRenderIn-fo> renderInfoList | List structure consists of the 3D shape data name, instance name, Collision Filter Group name mentioned above. It is called Part/Pallet Instance List. | --- |
| bool isDisplayNumber = false | Set whether to display the instance numbers of the parts generated by the Process Manager. The numbers are not al-ways displayed in the order of generation. This argument can be omitted. If it is omitted, false is set. | true |

## ● Description

Instantiate the 3D shape data, which is designated as *objectModel*, in the name of *name* to handle the data as multiple parts with the emulation function of the Application Manager.
Register the 3D shape data in the Part/Pallet Instance List as many as the number specified by *count*. Names to be registered in the Part/Pallet Instance List consists of the string specified by *name*, and "_N" (N=0,1,2, ..., count-1).
Example: *name* is "Part", and N = 3
"Part_0"
"Part_1"
"Part_2"

# LoadPart

Display parts.

## ● Function Call

| | |
|---|---|
| **Function** | LoadPart(IExtendedShapeScript shapeScript, string controllerName, string variableName, IShapeBase partModel, string name, Transform3D worldCoordinate, IVisualizable locationModel, IEnumerable<ShapeRenderInfo> renderInfoList, IDictionary<string, object> variableValues = null) |
| **Used in** | Render() |

## ● Arguments

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shapeScript | Specify the Shape Script to use in this function. | --- |
| string controllerName | Specify the Controller name by string. | "new_controller_0" |
| string variableName | Specify a BOOL global variable name of the Controller by string. | "LoadPart" |
| IShapeBase partModel | Specify a variable that designates the 3D shape data for a part registered in *renderInfoList* | "partModel" |
| string name | Specify the instance name of the 3D shape data. | "Part" |
| Transform3D worldCoordinate | Specify the coordinate on the 3D Visualizer to place the part. | new Transform3D(527.211, -81.746, 135.503) |
| IVisualizable locationModel | Specify the variable corresponding to the 3D shape data to be the parent of the displayed part. | placeModel |
| IEnumerable<ShapeRenderInfo> renderInfoList | Specify the name of the Part/Pallet Instance List designated by CreateRenderInfo. | --- |
| IDictionary<string, object> variableValues = null | Specify the list of variables that you got by GetVariableValues. When you get multiple variables from the Controller, using GetVariableValues to get the values of the multiple variables in advance may improve the processing speed. This argument can be omitted. If it is omitted, null is set. | --- |

## ● Description

When BOOL global variable whose Controller name is *controllerName* and global variable name is *variableName* is ON, search the part's 3D shape data name *partModel* and instance name *name* registered in the Part/Pallet Instance List, and then place the part at the specified world coordinate *worldCoordinate*.

The 3D shape data designated as *locationModel* becomes the parent.

## ● Restrictions

The LoadPart() function displays a part when the global variable *variableName* changes from OFF to ON. Although you can set the argument *Name* to null or string.Empty (blank character), the change from OFF to ON may be detected only when the LoadPart() function that specifies *variableName* is called the first time.

The following is an example where the part is not displayed correctly when the LoadPart() function is executed.

```
public override void Render(IEnumerable<ShapeRenderInfo> renderInfoList) {
    if (this.IsInitialized == false) {
        DefaultFunctions.InitializeObject(this, renderInfoList);
        return ;
    }

    ICadData partModel1 = (ICadData) ace["/ApplicationManager0/CAD Data0"];
    ICadData partModel2 = (ICadData) ace["/ApplicationManager0/CAD Data1"];
    IShapeBase parent = (IShapeBase) ace["/ApplicationManager0/parent"];

    DefaultFunctions.LoadPart("new_Controller_0", "var_LoadPart", partModel1, null, new Transform3D(0, 0, 0), parent, renderInfoList);
    DefaultFunctions.LoadPart("new_Controller_0", "var_LoadPart", partModel2, null, new Transform3D(0, 0, 0), parent, renderInfoList);
}
```

When line 69 is executed, the program checks whether the variable *var_LoadPart* changed from OFF to ON. If the variable has changed from OFF to ON, among parts that are registered in the *renderInfoList*, the part corresponding to the *partModel1* is displayed.
Then, the part corresponding to *partModel2* is not displayed even if line 70 is executed because the program determines that the variable *var_LoadPart* has already changed from OFF to ON.

There are the following two ways to display the part correctly.

1. Set the value that is set for the argument *Name* to the CreateObject() function as the value of the argument *Name* to the LoadPart() function. In the following example, *CollisionSource1* or *CollisionSource2* that is set for the argument *Name* to the CreateObject() function as the value of the argument *Name* to the LoadPart() function.

```
40      public override IEnumerable<ShapeRenderInfo> CreateRenderInfo() {
41          this.IsInitialized = false;
42
43          // Create the Shapes for rendering once in the constructor
44          List<ShapeRenderInfo> renderInfoList = new List<ShapeRenderInfo>();
45
46          ICadData partModel1 = (ICadData) ace["/ApplicationManager0/CAD Data0"];
47          ICadData partModel2 = (ICadData) ace["/ApplicationManager0/CAD Data1"];
48
49          DefaultFunctions.CreateObject(partModel1, "CollisionSource1", "Group1", renderInfoList);
50          DefaultFunctions.CreateObject(partModel2, "CollisionSource2", "Group1", renderInfoList);
51
52          return renderInfoList;
53      }
54
55      /// <summary>
56      /// Called to render the Shape Script object
57      /// </summary>
58      /// <returns>The list of shapes to render</returns>
59      public override void Render(IEnumerable<ShapeRenderInfo> renderInfoList) {
60          if (this.IsInitialized == false) {
61              DefaultFunctions.InitializeObject(this, renderInfoList);
62              return ;
63          }
64
65          ICadData partModel1 = (ICadData) ace["/ApplicationManager0/CAD Data0"];
66          ICadData partModel2 = (ICadData) ace["/ApplicationManager0/CAD Data1"];
67          IShapeBase parent = (IShapeBase) ace["/ApplicationManager0/parent"];
68
69          DefaultFunctions.LoadPart("new_Controller_0", "var_LoadPart", partModel1, "CollisionSource1", new Transform3D(0, 0, 0), parent, renderInfoList);
70          DefaultFunctions.LoadPart("new_Controller_0", "var_LoadPart", partModel2, "CollisionSource2", new Transform3D(0, 0, 0), parent, renderInfoList);
71      }
```

2. Create a BOOL global variable other than *var_LoadPart* on the Controller. Then, create a program that changes the newly created global variable from OFF to ON at the same time when *var_LoadPart* changes from OFF to ON. The newly added global variable must be set as an argument in line 70. You can also display the part by setting null or string.Empty (blank character) for the argument *name*.

```
59      public override void Render(IEnumerable<ShapeRenderInfo> renderInfoList) {
60          if (this.IsInitialized == false) {
61              DefaultFunctions.InitializeObject(this, renderInfoList);
62              return ;
63          }
64
65          ICadData partModel1 = (ICadData) ace["/ApplicationManager0/CAD Data0"];
66          ICadData partModel2 = (ICadData) ace["/ApplicationManager0/CAD Data1"];
67          IShapeBase parent = (IShapeBase) ace["/ApplicationManager0/parent"];
68
69          DefaultFunctions.LoadPart("new_Controller_0", "var_LoadPart", partModel1, null, new Transform3D(0, 0, 0), parent, renderInfoList);
70          DefaultFunctions.LoadPart("new_Controller_0", "var_LoadPart1", partModel2, null, new Transform3D(0, 0, 0), parent, renderInfoList);
71      }
```

## LoadPallet

Display pallets.

### Function Call

| | |
|---|---|
| **Function** | LoadPallet(IExtendedShapeScript shapeScript, string controllerName, string variableName, IShapeBase palletModel, string name, Transform3D worldCoordinate, IVisualizable locationModel, IEnumerable<ShapeRenderInfo> renderInfoList, IDictionary<string, object> variableValues = null) |
| **Used in** | Render() |

### Arguments

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shapeScript | Specify the Shape Script to use in this function. | --- |
| string controllerName | Specify the Controller name by string. | "new_controller_0" |
| string variableName | Specify a BOOL global variable name of the Controller by string. | "LoadPallet" |
| IShapeBase palletModel | Specify a variable that designates the 3D shape data for a part registered in *renderInfoList* | "palletModel" |
| string name | Specify the instance name of the 3D shape data. | "Pallet" |
| Transform3D worldCoordinate | Specify the coordinate on the 3D Visualizer to place the pallet. | new Transform3D(527.211, -81.746, 135.503) |
| IVisualizable locationModel | Specify the variable corresponding to the 3D shape data to be the parent of the displayed pallet. | placeModel |
| IEnumerable<ShapeRenderInfo> renderInfoList | Specify the name of the Part/Pallet Instance List designated by CreateRenderInfo. | --- |
| IDictionary<string, object> variableValues = null | Specify the list of variables that you got by GetVariableValues. When you get multiple variables from the Controller, using GetVariableValues to get the values of the multiple variables in advance may improve the processing speed. This argument can be omitted. If it is omitted, null is set. | --- |

### Description

When BOOL global variable whose Controller name is *controllerName* and global variable name is *variableName* is ON, search the pallet's 3D shape data name *palletModel* and instance name *name* registered in the Part/Pallet Instance List, and then place the pallet at the specified world coordinate *worldCoordinate*.
The 3D shape data designated as *locationModel* becomes the parent.

### Restrictions

As with the LoadPart() function, the LoadPartOnPallet() function displays the pallet when the global variable *variableName* changes from OFF to ON. Refer to Restrictions in *LoadPart* on page A-13 for the restrictions.

## LoadPartOnPallet

Display parts on a pallet.

### ● Function Call

| | |
|---|---|
| **Function** | LoadPartOnPallet(IExtendedShapeScript shapeScript, string controllerName, string variable-Name, IShapeBase partModel, string partName, Transform3D worldcoordinate, IShapeBase pal-letModel, string palletName, IEnumerable<ShapeRenderInfo> renderInfoList, IDictionary<string, object> variableValues = null) |
| **Used in** | Render() |

### ● Arguments

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shapeScript | Specify the Shape Script to use in this function. | --- |
| string controllerName | Specify the Controller name by string. | "new_controller_0" |
| string variableName | Specify a BOOL global variable name of the Controller by string. | "LoadPart" |
| IShapeBase partModel | Specify a variable that designates the 3D shape data for a part registered in *renderInfoList* | "partModel" |
| string partName | Specify the instance name of the part 3D shape data. | "Part" |
| Transform3D world-Coordinate | Specify the coordinate on the 3D Visualizer to place the part. | new Trans-form3D(527.211, -81.746, 135.503) |
| IShapeBase palletMo-del | Specify a variable that designates the 3D shape data for a pallet registered in *renderInfoList* | palletModel |
| string palletName | Specify the instance name of the pallet 3D shape data. | "Pallet" |
| IDictionary<string, ob-ject> variableValues = null | Specify the list of variables that you got by GetVariable-Values. When you get multiple variables from the Con-troller, using GetVariableValues to get the values of the multiple variables in advance may improve the process-ing speed. This argument can be omitted. If it is omitted, null is set. | --- |

### ● Description

When BOOL global variable whose Controller name is *controllerName* and global variable name is *variableName* is ON, search the part's 3D shape data name *partModel* and instance name *partName*, and the pallet's 3D shape data name *palletModel* and instance name *palletName* regis-tered in the Part/Pallet Instance List. Then place the part at the specified world coordinate *worldCoordinate*.
The 3D shape data designated to 3D shape data name *palletModel* and instance name *palletName* becomes the parent.

## UnloadPart

Hide the part by entering the BOOL variable of the Controller.

### ● Function Call

| | |
|---|---|
| **Function** | UnloadPart(IExtendedShapeScript shapeScript, string controllerName, string variableName, IShapeBase partModel, string name, IEnumerable<ShapeRenderInfo> renderInfoList, IDiction-ary<string, object> variableValues = null) |
| **Used in** | Render() |

## ● Arguments

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shapeScript | Specify the Shape Script to use in this function. | --- |
| string controllerName | Specify the Controller name by string. | "new_controller_0" |
| string variableName | Specify a BOOL global variable name of the Controller by string. | "UnloadPart" |
| IShapeBase partModel | Specify a variable that designates the 3D shape data for a part registered in *renderInfoList* | partModel |
| string name | Specify the instance name of the part 3D shape data. | "Part" |
| IEnumerable<ShapeRenderInfo> renderInfoList | Specify the name of the Part/Pallet Instance List designated by CreateRenderInfo. | --- |
| IDictionary<string, object> variableValues = null | Specify the list of variables that you got by GetVariableValues. When you get multiple variables from the Controller, using GetVariableValues to get the values of the multiple variables in advance may improve the processing speed. This argument can be omitted. If it is omitted, null is set. | --- |

## ● Description

When BOOL global variable whose Controller name is *controllerName* and global variable name is *variableName* is ON, search the part's 3D shape data name *partModel* and instance name *name* registered in the Part/Pallet Instance List, and then hide the part on the 3D Visualizer.

# UnloadPallet

Hide the pallet by entering the BOOL variable of the Controller.

## ● Function Call

| | |
|---|---|
| **Function** | UnloadPallet(IExtendedShapeScript shapeScript, string controllerName, string variableName, IShapeBase palletModel, string name, IEnumerable<ShapeRenderInfo> renderInfoList, IDictionary<string, object> variableValues = null) |
| **Used in** | Render() |

## ● Arguments

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shapeScript | Specify the Shape Script to use in this function. | --- |
| string controllerName | Specify the Controller name by string. | "new_controller_0" |
| string variableName | Specify a BOOL global variable name of the Controller by string. | "UnloadPart" |
| IShapeBase palletModel | Specify a variable that designates the 3D shape data for a pallet registered in *renderInfoList* | palletModel |
| string name | Specify the instance name of the pallet 3D shape data. | "Pallet" |
| IDictionary<string, object> variableValues = null | Specify the list of variables that you got by GetVariableValues. When you get multiple variables from the Controller, using GetVariableValues to get the values of the multiple variables in advance may improve the processing speed. This argument can be omitted. If it is omitted, null is set. | --- |

- **Description**

When BOOL global variable whose Controller name is *controllerName* and global variable name is *variableName* is ON, search the pallet's 3D shape data name *palletModel* and instance name *name* registered in the Part/Pallet Instance List, and then hide the pallet on the 3D Visualizer. Also hide the child part of the pallet.

## UnloadCollidingPart

Hide the part after colliding with the designated 3D shape data.

- **Function Call**

| | |
|---|---|
| **Function** | UnloadCollidingPallet(IExtendedShapeScript shapeScript, IShapeBase palletModel, IVisualizable targetModel, IEnumerable<ShapeRenderInfo> renderInfoList ) |
| **Used in** | Render() |

- **Arguments**

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shapeScript | Specify the Shape Script to use in this function. | --- |
| IShapeBase partModel | Specify a variable that designates the 3D shape data for a part registered in *renderInfoList* | partModel |
| IVisualizable targetModel | Specify the variable corresponding to the 3D shape data that collides with the part. | targetModel |
| IEnumerable<ShapeRenderInfo> renderInfoList | Specify the name of the Part/Pallet Instance List designated by CreateRenderInfo. | --- |

- **Description**

Search the part 3D shape data name *partModel* registered in the Part/Pallet Instance List. Then hide the part *partModel* after *partModel* collides with the 3D shape data specified by *targetModel*.

## UnloadCollidingPallet

Hide the pallet after colliding with the designated 3D shape data.

- **Function Call**

| | |
|---|---|
| **Function** | UnloadCollidingPallet(IExtendedShapeScript shapeScript, IShapeBase palletModel, IVisualizable targetModel, IEnumerable<ShapeRenderInfo> renderInfoList ) |
| **Used in** | Render() |

- **Arguments**

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shapeScript | Specify the Shape Script to use in this function. | --- |
| IShapeBase palletModel | Specify a variable that designates the 3D shape data for a pallet registered in *renderInfoList* | palletModel |

| Argument | Description | Notation example |
|---|---|---|
| IVisualizable targetModel | Specify the variable corresponding to the 3D shape data that collides with the pallet. | targetModel |
| IEnumerable<ShapeRenderInfo> renderInfoList | Specify the name of the Part/Pallet Instance List designated by CreateRenderInfo. | --- |

### ● Description

Search the pallet 3D shape data name *partModel* registered in the Part/Pallet Instance List. Then hide the pallet *palletModel* after *palletModel* collides with the 3D shape data specified by *targetModel*.
Also hide the child part of the pallet.

## A-1-3 Conveying Parts and Pallets

## PushPart

Push the part by the cylinder.

### ● Function Call

| | |
|---|---|
| **Function** | PushPart(IExtendedShapeScript shapeScript, int stepId, string controllerName, string variableName, IVisualizable pushModel, IShapeBase partModel, IVisualizable locationModel, IEnumerable<ShapeRenderInfo> renderInfoList, IDictionary<string, object> variableValues = null) |
| **Used in** | Render() |

### ● Arguments

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shapeScript | Specify the Shape Script to use in this function. | --- |
| int stepId | Specify part/pallet state number. *1 | 1 |
| string controllerName | Specify the Controller name by string. | "new_controller_0" |
| string variableName | Specify a BOOL global variable for the Controller by string. | "PushPart" |
| IVisualizable pushModel | Specify a variable corresponding to the 3D shape data to be the parent of the displayed part. | pushModel |
| IShapeBase partModel | Specify a variable that designates the 3D shape data for a part registered in *renderInfoList* | partModel |
| IVisualizable locationModel | Specify a variable corresponding to the 3D shape data that becomes the parent after colliding with the moved part. | placeModel |
| IDictionary<string, object> variableValues = null | Specify the list of variables that you got by GetVariableValues. When you get multiple variables from the Controller, using GetVariableValues to get the values of the multiple variables in advance may improve the processing speed. This argument can be omitted. If it is omitted, null is set. | --- |

*1. Part/Pallet State Number determines the execution order of serially performed functions in the category of *Conveying Parts and Pallets*.
1 is set for the number when a Shape Script is performed for the first time. A function in the *Conveying Parts and Pallets* category runs when *stepId*, which specified to the first parameter of itself, matches with the number. If the parent-child relationship between a part/pallet and 3D shape data has been changed

due to the processing, the part/pallet state number increments by 1. Any number can be designated to the part/pallet state number with the SetNextStep() function. Through the methods above, you can configure the execution order of *Conveying Parts and Pallets* functions.

### ● Description

When the BOOL global variable whose part status number is *stepId*, Controller name is *controllerName* and global variable name is *variableName* is ON, search the part's 3D shape data name *partModel* registered in the Part/Pallet Instance List, and then, in the 3D Visualizer, operate the part in tandem with the cylinder *pushModel*.
After processed, if the parent-child relationship has been changed, the part/pallet state number is incremented by 1.

## PushPallet

Push the pallet by the cylinder.

### ● Function Call

| | |
|---|---|
| **Function** | PushPallet(IExtendedShapeScript shapeScript, int stepId, string controllerName, string variableName, IVisualizable pushModel, IShapeBase palletModel, IVisualizable locationModel, IEnumerable<ShapeRenderInfo> renderInfoList, IDictionary<string, object> variableValues = null) |
| **Used in** | Render() |

### ● Arguments

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shapeScript | Specify the Shape Script to use in this function. | --- |
| int stepId | Specify part/pallet state number. *1 | 1 |
| string controllerName | Specify the Controller name by string. | "new_controller_0" |
| string variableName | Specify a BOOL global variable name of the Controller by string. | "PushPallet" |
| IVisualizable pushModel | Specify a variable corresponding to the 3D shape data to be the parent of the displayed part. | pushModel |
| IShapeBase palletModel | Specify a variable that designates the 3D shape data for a part registered in *renderInfoList* | palletModel |
| IVisualizable locationModel | Specify a variable corresponding to the 3D shape data that becomes the parent after colliding with the moved part. | placeModel |
| IEnumerable<ShapeRenderInfo> renderInfoList | Specify the name of the Part/Pallet Instance List designated by CreateRenderInfo. | --- |
| IDictionary<string, object> variableValues = null | Specify the list of variables that you got by GetVariableValues. When you get multiple variables from the Controller, using GetVariableValues to get the values of the multiple variables in advance may improve the processing speed. This argument can be omitted. If it is omitted, null is set. | --- |

*1. Part/Pallet State Number determines the execution order of serially performed functions in the category of *Conveying Parts and Pallets*.
1 is set for the number when a Shape Script is performed for the first time. A function in the *Conveying Parts and Pallets* category runs when *stepId*, which specified to the first parameter of itself, matches with

the number. If the parent-child relationship between a part/pallet and 3D shape data has been changed due to the processing, the part/pallet state number increments by 1. Any number can be designated to the part/pallet state number with the SetNextStep() function. Through the methods above, you can configure the execution order of *Conveying Parts and Pallets* functions.

● **Description**

When the BOOL global variable whose part state number is *stepId*, Controller name is *controllerName* and global variable name is *variableName* is ON, search the part’s 3D shape data name *palletModel* registered in the Part/Pallet Instance List, and then, in the 3D Visualizer, operate the pallet and the part on it in tandem with the cylinder *pushModel*.
After processed, if the parent-child relationship has been changed, the part/pallet state number is incremented by 1.

## A-1-4 Conveyor

### MoveObjectOnBelt

Move the part or pallet on the conveyor.

● **Function Call**

| | |
|---|---|
| **Function** | MoveObjectOnBelt(IExtendedShapeScript shapeScript, string controllerName, string variableName, IVisualizable beltModel, Transform3D distancePerSecond, IEnumerable<ShapeRenderInfo> renderInfoList, IDictionary<string, object> variableValues = null) |
| **Used in** | Render() |

● **Arguments**

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shapeScript | Specify the Shape Script to use in this function. | --- |
| string controllerName | Specify the Controller name by string. | "new_controller_0" |
| string variableName | Specify a BOOL global variable name of the Controller by string. | "MoveBelt" |
| IVisualizable beltModel | Specify the variable corresponding to the 3D shape data for the conveyor. | beltModel |
| Transform3D distancePerSecond | Specify the direction and travel distance of the part on the conveyor by 3D vector. The unit is mm/sec. | Transform3D(-2, 0, 0) Means -2 mm/sec in the X axis direction. |
| IEnumerable<ShapeRenderInfo> renderInfoList | Specify the name of the Part/Pallet Instance List designated by CreateRenderInfo. | --- |
| IDictionary<string, object> variableValues = null | Specify the list of variables that you got by GetVariableValues. When you get multiple variables from the Controller, using GetVariableValues to get the values of the multiple variables in advance may improve the processing speed. This argument can be omitted. If it is omitted, null is set. | --- |

### Description

When BOOL global variable whose Controller name is *controllerName* and global variable name is *variableName* is ON, operate the registered 3D shape data in the Part/Pallet Instance List that is the child of the conveyor 3D shape data *beltModel*, in the direction and travel distance specified by *distancePerSecond*.

3D shape data that collides with the 3D shape data except *beltModel* does not operate.

The MoveObjectOnBelt() function gets the system variable *_CurrentTime*, which is a system-defined variable of type DATE_AND_TIME that stores the internal time of the Controller, in addition to the variable specified by *variableName*. This system variable is used to calculate the travel distance of belts. If the GetVariableValues() function is used to get *variableValues*, you can speed up the processing of the Shape Script by also getting the value of the variable *_CurrentTime*.

## InitializeMechanicalConveyor

Initialize the mechanical component *Conveyor*.

### Function Call

| | |
|---|---|
| **Function** | InitializeMechanicalConveyor( IMechanicalConveyor mechanicalConveyor) |
| **Used in** | CreateRenderInfo() or Render() |

### Arguments

| Argument | Description | Notation example |
|---|---|---|
| IMechanicalConveyor mechanicalConveyor | Specify the mechanical component (Conveyor) object. | --- |

### Description

Call the *InitializeConveyorMoveStatus* of *IMechanicalConveyor* and internally set the *IsMoveConveyor* flag to FALSE.

Note that this is called and used only once during execution of the Shape Script.

## MoveObjectOnMechanicalConveyor

Move the parts or pallets on the mechanical component *Conveyor*.

### Function Call

| | |
|---|---|
| **Function** | MoveObjectOnMechanicalConveyor(IExtendedShapeScript shapeScript, IMechanicalConveyor mechanicalConveyor, IEnumerable<ShapeRenderInfo>renderInfoList, IDictionary<string, object>variableValues = null) |
| **Used in** | Render() |

### Arguments

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shapeScript | Specify the Shape Script to use in this function. | --- |

| Argument | Description | Notation example |
|---|---|---|
| IMechanicalConveyor mechanicalConveyor | Specify the mechanical component (Conveyor) object. | --- |
| IEnumerable<ShapeRenderInfo> renderInfoList | Specify the name of the Part/Pallet Instance List specified in CreateRenderInfo. | --- |
| IDictionary<string, object> variableValues = null | Specify the list of variables that you got by GetVariableValues. When you get multiple variables from the Controller, using GetVariableValues to get the values of the multiple variables in advance may improve the processing speed. This argument can be omitted. If it is omitted, null is set. | --- |

### ● Description

For the first call, the actual position *Act.Pos* of the axis assigned to the mechanical component at that time is recorded in *ConveyorPrevAxisValue* of the mechanical component object *IMechanicalConveyor*. Also, *IsMoveConveyor* is set to TRUE.
For the second and later calls, the difference between the previous axis position stored in *ConveyorPrevAxisValue* of the mechanical component object *IMechanicalConveyor* and the current axis position is calculated. Then, the displayed (placement) coordinates of the parts or pallets are moved based on the calculated value and the linear moving direction of the mechanical component. At this time, the parts or pallets whose parent is the target mechanical component will be moved. However, the parts or pallets will not move if they collide with something other than the target mechanical component.

## A-1-5 Clamping Parts and Pallets

## ClampPart

Pick up the part.

### ● Function Call

| | |
|---|---|
| **Function** | ClampPart(IExtendedShapeScript shapeScript, int stepId, IVisualizable robotTool , IShapeBase partModel, IVisualizable locationModel, IEnumerable<ShapeRenderInfo> renderInfoList ) |
| **Used in** | Render() |

### ● Arguments

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shapeScript | Specify the Shape Script to use in this function. | --- |
| int stepId | Specify part/pallet state number. [*1] | 1 |
| IVisualizable robotTool | Specify the variable corresponding to the 3D shape data that picks up the part. | pickModel |
| IShapeBase partModel | Specify a variable that designates the 3D shape data for a part registered in *renderInfoList* | partModel |

| Argument | Description | Notation example |
|---|---|---|
| IVisualizable locationModel | Specify the variable corresponding to the 3D shape data of the parent in advance of a part collision. | placeModel |
| IEnumerable<ShapeRenderIn-fo> renderInfoList | Specify the name of the Part/Pallet Instance List designat-ed by CreateRenderInfo. | --- |

*1. Part/Pallet State Number determines the execution order of serially performed functions in the category of *Conveying Parts and Pallets*.
1 is set for the number when a Shape Script is performed for the first time. A function in the *Conveying Parts and Pallets* category runs when *stepId*, which specified to the first parameter of itself, matches with the number. If the parent-child relationship between a part/pallet and 3D shape data has been changed due to the processing, the part/pallet state number increments by 1. Any number can be designated to the part/pallet state number with the SetNextStep() function. Through the methods above, you can con-figure the execution order of *Conveying Parts and Pallets* functions.

### ● Description

When the part state number is *stepId* and the parent of the part whose 3D shape name is *partModel* registered in the Part/Pallet Instance List is the 3D shape data specified by *locationModel*, change the parent of the part to the 3D shape data specified by *robotTool*, after *partModel* collides with the 3D shape data designated by *robotTool*.
After processed, if the parent-child relationship has been changed, the part/pallet state number is incremented by 1.

## ClampPallet

Pick up the pallet.

### ● Function Call

| | |
|---|---|
| **Function** | ClampPallet(IExtendedShapeScript shapeScript, int stepId, IVisualizable robotTool , IShapeBase palletModel, IVisualizable locationModel, IEnumerable<ShapeRenderInfo> renderInfoList ) |
| **Used in** | Render() |

### ● Arguments

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shape-Script | Specify the Shape Script to use in this function. | --- |
| int stepId | Specify part/pallet state number.*1 | 1 |
| IVisualizable robotTool | Specify the variable corresponding to the 3D shape data that picks up the pallet. | pickModel |
| IShapeBase palletModel | Specify a variable that designates the 3D shape data for a pallet registered in *renderInfoList* | palletModel |
| IVisualizable locationModel | Specify the variable corresponding to the 3D shape data of the parent in advance of a pallet collision. | placeModel |
| IEnumerable<ShapeRenderIn-fo> renderInfoList | Specify the name of the Part/Pallet Instance List designat-ed by CreateRenderInfo. | --- |

*1. Part/Pallet State Number determines the execution order of serially performed functions in the category of *Conveying Parts and Pallets*.

1 is set for the number when a Shape Script is performed for the first time. A function in the *Conveying Parts and Pallets* category runs when *stepId*, which specified to the first parameter of itself, matches with the number. If the parent-child relationship between a part/pallet and 3D shape data has been changed due to the processing, the part/pallet state number increments by 1. Any number can be designated to the part/pallet state number with the SetNextStep() function. Through the methods above, you can configure the execution order of *Conveying Parts and Pallets* functions.

### ● Description

When the part state number is *stepId* and the parent of the pallet whose 3D shape name is *palletModel* registered in the Part/Pallet Instance List is the 3D shape data specified by *locationModel*, change the parent of the pallet to the 3D shape data specified by *robotTool*, after *palletModel* collides with the 3D shape data designated by *robotTool*.
After processed, if the parent-child relationship has been changed, the part/pallet state number is incremented by 1.

## ClampPartOnPallet

Pick up the part on the pallet.

### ● Function Call

| | |
|---|---|
| **Function** | ClampPartOnPallet(IExtendedShapeScript shapeScript, int stepId, IVisualizable robotTool , IShapeBase partModel, IShapeBase palletModel, IEnumerable<ShapeRenderInfo> renderInfoList ) |
| **Used in** | Render() |

### ● Arguments

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shapeScript | Specify the Shape Script to use in this function. | --- |
| int stepId | Specify part/pallet state number. *1 | 1 |
| IVisualizable robotTool | Specify the variable corresponding to the 3D shape data that picks up the part. | pickModel |
| IShapeBase partModel | Specify a variable that designates the 3D shape data for a part registered in *renderInfoList* | partModel |
| IShapeBase palletModel | Specify a variable that designates the 3D shape data that is registered in *renderInfoList* and is the parent of the part. | palletModel |
| IEnumerable<ShapeRenderInfo> renderInfoList | Specify the name of the Part/Pallet Instance List designated by CreateRenderInfo. | --- |

*1. Part/Pallet State Number determines the execution order of serially performed functions in the category of *Conveying Parts and Pallets*.
1 is set for the number when a Shape Script is performed for the first time. A function in the *Conveying Parts and Pallets* category runs when *stepId*, which specified to the first parameter of itself, matches with the number. If the parent-child relationship between a part/pallet and 3D shape data has been changed due to the processing, the part/pallet state number increments by 1. Any number can be designated to the part/pallet state number with the SetNextStep() function. Through the methods above, you can configure the execution order of *Conveying Parts and Pallets* functions.

### Description

When the part state number is *stepId* and the parent of the part whose 3D shape name is *partModel* registered in the Part/Pallet Instance List is the 3D shape data specified by *palletModel*, change the parent of the part to the 3D shape data specified by *robotTool*, after *partModel* collides with the 3D shape data designated by *robotTool*.
After processed, if the parent-child relationship has been changed, the part/pallet state number is incremented by 1.

## ClampPartBySignal

Pick up the part on a grasping signal.

### Function Call

| | |
|---|---|
| **Function** | ClampPartBySignal(IExtendedShapeScript shapeScript, int stepId, string controllerName, string variableName, IVisualizable robotTool , IShapeBase partModel, IVisualizable locationModel, IEnumerable<ShapeRenderInfo> renderInfoList, IDictionary<string, object> variableValues = null) |
| **Used in** | Render() |

### Arguments

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shapeScript | Specify the Shape Script to use in this function. | --- |
| int stepId | Specify part/pallet state number. *1 | 1 |
| string controllerName | Specify the Controller name by string. | "new_controller_0" |
| string variableName | Specify a BOOL global variable name of the Controller by string. | "GraspPart" |
| IVisualizable robotTool | Specify the variable corresponding to the 3D shape data that picks up the part. | pickModel |
| IShapeBase partModel | Specify a variable that designates the 3D shape data for a part registered in *renderInfoList* | partModel |
| IVisualizable locationModel | Specify the variable corresponding to the 3D shape data of the parent in advance of a part collision. | placeModel |
| IEnumerable<ShapeRenderInfo> renderInfoList | Specify the name of the Part/Pallet Instance List designated by CreateRenderInfo. | --- |
| IDictionary<string, object> variableValues = null | Specify the list of variables that you got by GetVariableValues. When you get multiple variables from the Controller, using GetVariableValues to get the values of the multiple variables in advance may improve the processing speed. This argument can be omitted. If it is omitted, null is set. | --- |

*1. Part/Pallet State Number determines the execution order of serially performed functions in the category of *Conveying Parts and Pallets*.
1 is set for the number when a Shape Script is performed for the first time. A function in the *Conveying Parts and Pallets* category runs when *stepId*, which specified to the first parameter of itself, matches with the number. If the parent-child relationship between a part/pallet and 3D shape data has been changed due to the processing, the part/pallet state number increments by 1. Any number can be designated to the part/pallet state number with the SetNextStep() function. Through the methods above, you can configure the execution order of *Conveying Parts and Pallets* functions.

● **Description**

When the BOOL global variable whose part state number is *stepId*, Controller name is *controllerName*, and global variable name is *variableName* is ON, the parent of the part whose 3D shape name is *partModel* registered in the Part/Pallet Instance List is the 3D shape data specified by *locationModel*, change the parent of the part to the 3D shape data specified by *robotTool*, after *partModel* collides with the 3D shape data designated by *robotTool*.
After processed, if the parent-child relationship has been changed, the part/pallet state number is incremented by 1.

## ClampPalletBySignal

Pick up the pallet on a grasping signal.

● **Function Call**

| | |
|---|---|
| **Function** | ClampPalletBySignal(IExtendedShapeScript shapeScript, int stepId, string controllerName, string variableName, IVisualizable robotTool , IShapeBase palletModel, IVisualizable locationModel, IEnumerable<ShapeRenderInfo> renderInfoList, IDictionary<string, object> variableValues = null) |
| **Used in** | Render() |

● **Arguments**

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shapeScript | Specify the Shape Script to use in this function. | --- |
| int stepId | Specify part/pallet state number. *1 | 1 |
| string controllerName | Specify the Controller name by string. | "new_controller_0" |
| string variableName | Specify a BOOL global variable name of the Controller by string. | "GraspPallet" |
| IVisualizable robotTool | Specify the variable corresponding to the 3D shape data that picks up the pallet. | pickModel |
| IShapeBase palletModel | Specify a variable that designates the 3D shape data for a pallet registered in *renderInfoList* | palletModel |
| IVisualizable locationModel | Specify the variable corresponding to the 3D shape data of the parent in advance of a pallet collision. | placeModel |
| IEnumerable<ShapeRenderInfo> renderInfoList | Specify the name of the Part/Pallet Instance List designated by CreateRenderInfo. | --- |
| IDictionary<string, object> variableValues = null | Specify the list of variables that you got by GetVariableValues. When you get multiple variables from the Controller, using GetVariableValues to get the values of the multiple variables in advance may improve the processing speed. This argument can be omitted. If it is omitted, null is set. | --- |

*1. Part/Pallet State Number determines the execution order of serially performed functions in the category of *Conveying Parts and Pallets*.
1 is set for the number when a Shape Script is performed for the first time. A function in the *Conveying Parts and Pallets* category runs when *stepId*, which specified to the first parameter of itself, matches with the number. If the parent-child relationship between a part/pallet and 3D shape data has been changed due to the processing, the part/pallet state number increments by 1. Any number can be designated to

the part/pallet state number with the SetNextStep() function. Through the methods above, you can configure the execution order of *Conveying Parts and Pallets* functions.

### ● Description

When the BOOL global variable whose part state number is *stepId*, Controller name is *controllerName*, and global variable name is *variableName* is ON, the parent of the pallet whose 3D shape name is *palletModel* registered in the Part/Pallet Instance List is the 3D shape data specified by *locationModel*, change the parent of the part to the 3D shape data specified by *robotTool*, after *palletModel* collides with the 3D shape data designated by *robotTool*.
After processed, if the parent-child relationship has been changed, the part/pallet state number is incremented by 1.

## ClampPartOnPalletBySignal

Pick up the part on the pallet on a grasping signal.

### ● Function Call

| | |
|---|---|
| **Function** | ClampPartOnPalletBySignal(IExtendedShapeScript shapeScript, int stepId, string controllerName, string variableName, IVisualizable robotTool , IShapeBase partModel, IShapeBase palletModel, IEnumerable<ShapeRenderInfo> renderInfoList, IDictionary<string, object> variableValues = null) |
| **Used in** | Render() |

### ● Arguments

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shapeScript | Specify the Shape Script to use in this function. | --- |
| int stepId | Specify part/pallet state number. *1 | 1 |
| string controllerName | Specify the Controller name by string. | "new_controller_0" |
| string variableName | Specify a BOOL global variable name of the Controller by string. | "GraspPart" |
| IVisualizable robotTool | Specify the variable corresponding to the 3D shape data that picks up the part. | pickModel |
| IShapeBase partModel | Specify a variable that designates the 3D shape data for a part registered in *renderInfoList* | partModel |
| IShapeBase palletModel | Specify a variable that designates the 3D shape data for a pallet registered in *renderInfoList* | palletModel |
| IVisualizable locationModel | Specify the variable corresponding to the 3D shape data of the parent in advance of a part collision. | placeModel |
| IEnumerable<ShapeRenderInfo> renderInfoList | Specify the name of the Part/Pallet Instance List designated by CreateRenderInfo. | --- |
| IDictionary<string, object> variableValues = null | Specify the list of variables that you got by GetVariableValues. When you get multiple variables from the Controller, using GetVariableValues to get the values of the multiple variables in advance may improve the processing speed. This argument can be omitted. If it is omitted, null is set. | --- |

*1. Part/Pallet State Number determines the execution order of serially performed functions in the category of *Conveying Parts and Pallets*.

1 is set for the number when a Shape Script is performed for the first time. A function in the *Conveying Parts and Pallets* category runs when *stepId*, which specified to the first parameter of itself, matches with the number. If the parent-child relationship between a part/pallet and 3D shape data has been changed due to the processing, the part/pallet state number increments by 1. Any number can be designated to the part/pallet state number with the SetNextStep() function. Through the methods above, you can configure the execution order of *Conveying Parts and Pallets* functions.

### ● Description

When the BOOL global variable whose part state number is *stepId*, Controller name is *controllerName*, and global variable name is *variableName* is ON, the parent of the part whose 3D shape name is *partModel* registered in the Part/Pallet Instance List is the 3D shape data specified by *palletModel*, change the parent of the part to the 3D shape data specified by *robotTool*, after *partModel* collides with the 3D shape data designated by *robotTool*.
After processed, if the parent-child relationship has been changed, the part/pallet state number is incremented by 1.

## A-1-6 Placing Parts and Pallets

### ReleasePart

Place the part picked up.

### ● Function Call

| | |
|---|---|
| **Function** | ReleasePart(IExtendedShapeScript shapeScript, int stepId, IVisualizable robotTool, IShapeBase partModel, IVisualizable locationModel, IEnumerable<ShapeRenderInfo> renderInfoList ) |
| **Used in** | Render() |

### ● Arguments

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shapeScript | Specify the Shape Script to use in this function. | --- |
| int stepId | Specify part/pallet state number.*1 | 1 |
| IVisualizable robotTool | Specify a variable equivalent to the 3D shape data picking the part with this function. | pickModel |
| IShapeBase partModel | Specify a variable that designates the 3D shape data for a part registered in *renderInfoList*. | partModel |
| IVisualizable locationModel | Specify a variable equivalent to the 3D shape data placing the part. | placeModel |
| IEnumerable<ShapeRenderInfo> renderInfoList | Specify the name of the Part/Pallet Instance List specified in CreateRenderInfo. | --- |

*1. Part/Pallet State Number determines the execution order of serially performed functions in the category of *Conveying Parts and Pallets*.
1 is set for the number when a Shape Script is performed for the first time. A function in the *Conveying Parts and Pallets* category runs when *stepId*, which specified to the first parameter of itself, matches with the number. If the parent-child relationship between a part/pallet and 3D shape data has been changed due to the processing, the part/pallet state number increments by 1. Any number can be designated to the part/pallet state number with the SetNextStep() function. Through the methods above, you can configure the execution order of *Conveying Parts and Pallets* functions.

● **Description**

When the part state number is *stepId* and the part whose 3D shape name is *partModel* registered in the Part/Pallet Instance List is the child of the 3D shape data specified by *robotTool*, change the parent of the part to the 3D shape data specified by *locationModel* after colliding with the 3D shape data designated by *locationModel*.
After processed, if the parent-child relationship has been changed, the part/pallet state number is incremented by 1.

## ReleasePallet

Place the pallet picked up.

● **Function Call**

| | |
|---|---|
| **Function** | ReleasePallet(IExtendedShapeScript shapeScript, int stepId, IVisualizable robotTool, IShapeBase palletModel, IVisualizable locationModel, IEnumerable<ShapeRenderInfo> renderInfoList ) |
| **Used in** | Render() |

● **Arguments**

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shapeScript | Specify the Shape Script to use in this function. | --- |
| int stepId | Specify part/pallet state number.*1 | 1 |
| IVisualizable robotTool | Specify a variable equivalent to the 3D shape data picking the pallet with this function. | pickModel |
| IShapeBase palletModel | Specify a variable that indicates the pallet 3D shape data registered in *renderInfoList*. | palletModel |
| IVisualizable locationModel | Specify a variable equivalent to the 3D shape data placing the pallet. | placeModel |
| IEnumerable<ShapeRenderInfo> renderInfoList | Specify the name of the Part/Pallet Instance List specified in CreateRenderInfo. | --- |

*1. Part/Pallet State Number determines the execution order of serially performed functions in the category of *Conveying Parts and Pallets*.
1 is set for the number when a Shape Script is performed for the first time. A function in the *Conveying Parts and Pallets* category runs when *stepId*, which specified to the first parameter of itself, matches with the number. If the parent-child relationship between a part/pallet and 3D shape data has been changed due to the processing, the part/pallet state number increments by 1. Any number can be designated to the part/pallet state number with the SetNextStep() function. Through the methods above, you can configure the execution order of *Conveying Parts and Pallets* functions.

● **Description**

When the part state number is *stepId* and the pallet whose 3D shape name is *palletModel* registered in the Part/Pallet Instance List is the child of the 3D shape data specified by *robotTool*, change the parent of the pallet to the 3D shape data specified by *locationModel* after colliding with the 3D shape data designated by *locationModel*.
After processed, if the parent-child relationship has been changed, the part/pallet state number is incremented by 1.

## ReleasePartOnPallet

Place the part which picked up to the pallet.

### ● Function Call

| | |
|---|---|
| **Function** | ReleasePartOnPallet(IExtendedShapeScript shapeScript, int stepId, IVisualizable robotTool, IShapeBase partModel, IShapeBase palletModel, IEnumerable<ShapeRenderInfo> renderInfoList ) |
| **Used in** | Render() |

### ● Arguments

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shapeScript | Specify the Shape Script to use in this function. | --- |
| int stepId | Specify part/pallet state number.*1 | 1 |
| IVisualizable robotTool | Specify a variable equivalent to the 3D shape data picking the part with this function. | pickModel |
| IShapeBase partModel | Specify a variable that designates the 3D shape data for a part registered in *renderInfoList*. | partModel |
| IShapeBase palletModel | Specify a variable that indicates the pallet 3D shape data registered in *renderInfoList*. | palletModel |
| IEnumerable<ShapeRenderInfo> renderInfoList | Specify the name of the Part/Pallet Instance List specified in CreateRenderInfo. | --- |

*1. Part/Pallet State Number determines the execution order of serially performed functions in the category of *Conveying Parts and Pallets*.
1 is set for the number when a Shape Script is performed for the first time. A function in the *Conveying Parts and Pallets* category runs when *stepId*, which specified to the first parameter of itself, matches with the number. If the parent-child relationship between a part/pallet and 3D shape data has been changed due to the processing, the part/pallet state number increments by 1. Any number can be designated to the part/pallet state number with the SetNextStep() function. Through the methods above, you can configure the execution order of *Conveying Parts and Pallets* functions.

### ● Description

When the part state number is *stepId* and the part whose 3D shape name is *partModel* registered in the Part/Pallet Instance List is the child of the 3D shape data specified by *robotTool*, change the parent of the part to the 3D shape data specified by *palletModel* after colliding with the 3D shape data designated by *palletModel*.
After processed, if the parent-child relationship has been changed, the part/pallet state number is incremented by 1.

## ReleasePartBySignal

Release (place) the part according to a grasping signal of the robot hand.

## ● Function Call

| | |
|---|---|
| **Function** | ReleasePartBySignal(IExtendedShapeScript shapeScript, int stepId, string controllerName, string variableName, IVisualizable robotTool, IShapeBase partModel, IVisualizable locationModel, IEnumerable<ShapeRenderInfo> renderInfoList, IDictionary<string, object> variableValues = null) |
| **Used in** | Render() |

## ● Arguments

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shapeScript | Specify the Shape Script to use in this function. | --- |
| int stepId | Specify part/pallet state number.*1 | 1 |
| string controllerName | Specify the Controller name by string. | "new_controller_0" |
| string variableName | Specify a BOOL global variable for the Controller by string. | "GraspPart" |
| IVisualizable robotTool | Specify a variable equivalent to the 3D shape data picking the part with this function. | pickModel |
| IShapeBase partModel | Specify a variable that designates the 3D shape data for a part registered in *renderInfoList*. | partModel |
| IVisualizable locationModel | Specify a variable equivalent to the 3D shape data placing the part. | placeModel |
| IEnumerable<ShapeRenderInfo> renderInfoList | Specify the name of the Part/Pallet Instance List specified in CreateRenderInfo. | --- |
| IDictionary<string, object> variableValues = null | Specify the list of variables that you got by GetVariableValues. When you get multiple variables from the Controller, using GetVariableValues to get the values of the multiple variables in advance may improve the processing speed. This argument can be omitted. If it is omitted, null is set. | --- |

*1. Part/Pallet State Number determines the execution order of serially performed functions in the category of *Conveying Parts and Pallets*.
1 is set for the number when a Shape Script is performed for the first time. A function in the *Conveying Parts and Pallets* category runs when *stepId*, which specified to the first parameter of itself, matches with the number. If the parent-child relationship between a part/pallet and 3D shape data has been changed due to the processing, the part/pallet state number increments by 1. Any number can be designated to the part/pallet state number with the SetNextStep() function. Through the methods above, you can configure the execution order of *Conveying Parts and Pallets* functions.

## ● Description

When the BOOL global variable whose part state number is *stepId*, Controller name is *controllerName*, and global variable name is *variableName* is OFF and the part whose 3D shape name *partModel* is registered in the Part/Pallet Instance List is the child of the 3D shape data specified by *robotTool*, release the part and then, hide the part’s parent.
After the part collides with the 3D shape data specified by *locationModel*, the 3D shape data specified by *locationModel* becomes the parent of the part.
After processed, if the parent-child relationship has been changed, the part/pallet state number is incremented by 1.

## ReleasePalletBySignal

Release (place) the pallet according to a grasping signal of the robot hand.

### ● Function Call

| | |
|---|---|
| **Function** | ReleasePalletBySignal(IExtendedShapeScript shapeScript, int stepId, string controllerName, string variableName, IVisualizable robotTool, IShapeBase palletModel, IVisualizable locationModel, IEnumerable<ShapeRenderInfo> renderInfoList, IDictionary<string, object> variableValues = null) |
| **Used in** | Render() |

### ● Arguments

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shapeScript | Specify the Shape Script to use in this function. | --- |
| int stepId | Specify part/pallet state number.*1 | 1 |
| string controllerName | Specify the Controller name by string. | "new_controller_0" |
| string variableName | Specify a BOOL global variable for the Controller by string. | "GraspPallet" |
| IVisualizable robotTool | Specify a variable equivalent to the 3D shape data picking the pallet with this function. | pickModel |
| IShapeBase palletModel | Specify a variable that indicates the pallet 3D shape data registered in *renderInfoList*. | palletModel |
| IVisualizable locationModel | Specify a variable equivalent to the 3D shape data placing the pallet. | placeModel |
| IEnumerable<ShapeRenderInfo> renderInfoList | Specify the name of the Part/Pallet Instance List specified in CreateRenderInfo. | --- |
| IDictionary<string, object> variableValues = null | Specify the list of variables that you got by GetVariableValues. When you get multiple variables from the Controller, using GetVariableValues to get the values of the multiple variables in advance may improve the processing speed. This argument can be omitted. If it is omitted, null is set. | --- |

*1. Part/Pallet State Number determines the execution order of serially performed functions in the category of *Conveying Parts and Pallets*.
1 is set for the number when a Shape Script is performed for the first time. A function in the *Conveying Parts and Pallets* category runs when *stepId*, which specified to the first parameter of itself, matches with the number. If the parent-child relationship between a part/pallet and 3D shape data has been changed due to the processing, the part/pallet state number increments by 1. Any number can be designated to the part/pallet state number with the SetNextStep() function. Through the methods above, you can configure the execution order of *Conveying Parts and Pallets* functions.

### ● Description

When the BOOL global variable whose part state number is *stepId*, Controller name is *controllerName*, and global variable name is *variableName* is OFF and the pallet whose 3D shape name *palletModel* is registered in the Part/Pallet Instance List is the child of the 3D shape data specified by *robotTool*, release the part and then, hide the pallet's parent.
After the pallet collides with the 3D shape data specified by *locationModel*, the 3D shape data specified by *locationModel* becomes the parent of the pallet.

After processed, if the parent-child relationship has been changed, the part/pallet state number is incremented by 1.

## ReleasePartOnPalletBySignal

Release (place) the part to the pallet according to a grasping signal of the robot hand.

### ● Function Call

| | |
|---|---|
| **Function** | ReleasePartOnPalletBySignal(IExtendedShapeScript shapeScript, int stepId, string controllerName, string variableName, IVisualizable robotTool, IShapeBase partModel, IShapeBase palletModel, IEnumerable<ShapeRenderInfo> renderInfoList, IDictionary<string, object> variableValues = null) |
| **Used in** | Render() |

### ● Arguments

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shapeScript | Specify the Shape Script to use in this function. | --- |
| int stepId | Specify part/pallet state number.*1 | 1 |
| string controllerName | Specify the Controller name by string. | "new_controller_0" |
| string variableName | Specify a BOOL global variable for the Controller by string. | "GraspPallet" |
| IVisualizable robotTool | Specify a variable equivalent to the 3D shape data picking the part with this function. | pickModel |
| IShapeBase partModel | Specify a variable that designates the 3D shape data for a part registered in *renderInfoList*. | partModel |
| IShapeBase palletModel | Specify a variable that indicates the pallet 3D shape data registered in *renderInfoList*. | palletModel |
| IEnumerable<ShapeRenderInfo> renderInfoList | Specify the name of the Part/Pallet Instance List specified in CreateRenderInfo. | --- |
| IDictionary<string, object> variableValues = null | Specify the list of variables that you got by GetVariableValues. When you get multiple variables from the Controller, using GetVariableValues to get the values of the multiple variables in advance may improve the processing speed. This argument can be omitted. If it is omitted, null is set. | --- |

*1. Part/Pallet State Number determines the execution order of serially performed functions in the category of *Conveying Parts and Pallets*.
1 is set for the number when a Shape Script is performed for the first time. A function in the *Conveying Parts and Pallets* category runs when *stepId*, which specified to the first parameter of itself, matches with the number. If the parent-child relationship between a part/pallet and 3D shape data has been changed due to the processing, the part/pallet state number increments by 1. Any number can be designated to the part/pallet state number with the SetNextStep() function. Through the methods above, you can configure the execution order of *Conveying Parts and Pallets* functions.

### ● Description

When the BOOL global variable whose part state number is *stepId*, Controller name is *controllerName*, and global variable name is *variableName* is OFF and the part whose 3D shape

name *partModel* is registered in the Part/Pallet Instance List is the child of the 3D shape data specified by *robotTool*, release the part and then, hide the part's parent.
After the part collides with the 3D shape data specified by *palletModel*, the 3D shape data specified by *palletModel* becomes the parent of the part.
After processed, if the parent-child relationship has been changed, the part/pallet state number is incremented by 1.
This function is used to place the part to the pallet.

## A-1-7 Changing and Detecting Status

### SetNextStep

Change the value of stepId.

#### ● Function Call

| | |
|---|---|
| **Function** | SetNextStep(IExtendedShapeScript shapeScript, int stepId, string controllerName, string variableName, int value, IDictionary<string, object> variableValues = null) |
| **Used in** | Render() |

#### ● Arguments

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shapeScript | Specify the Shape Script to use in this function. | --- |
| int stepId | Specify part/pallet state number.*1 | 1 |
| string controllerName | Specify the Controller name by string. | "new_controller_0" |
| string variableName | Specify a BOOL global variable for the Controller by string. | "NextStep" |
| int value | Specify the Part/Pallet status number to be changed. | 2 |
| IDictionary<string, object> variableValues = null | Specify the list of variables that you got by GetVariableValues. When you get multiple variables from the Controller, using GetVariableValues to get the values of the multiple variables in advance may improve the processing speed. This argument can be omitted. If it is omitted, null is set. | --- |

*1. Part/Pallet State Number determines the execution order of serially performed functions in the category of *Conveying Parts and Pallets*.
1 is set for the number when a Shape Script is performed for the first time. A function in the *Conveying Parts and Pallets* category runs when *stepId*, which specified to the first parameter of itself, matches with the number. If the parent-child relationship between a part/pallet and 3D shape data has been changed due to the processing, the part/pallet state number increments by 1. Any number can be designated to the part/pallet state number with the SetNextStep() function. Through the methods above, you can configure the execution order of *Conveying Parts and Pallets* functions.

#### ● Description

When the BOOL global variable whose part status number is *stepId*, Controller name is *controllerName* and global variable name is *variableName* is ON, change the part state number to the value of *value*.

## DetectPartCollision

Detect if a part or a pallet comes into contact with Mechanical Component.

### ● Function Call

| | |
|---|---|
| **Function** | DetectPartCollision(IExtendedShapeScript shapeScript, IShapeBase partModel, IVisualizable mechanicalModel, int targetStep , string controllerName, string variableName, IEnumerable<ShapeRenderInfo> renderInfoList, IList<Tuple<string, object>> setVariableList = null) |
| **Used in** | Render() |

### ● Arguments

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shapeScript | Specify the Shape Script to use in this function. | --- |
| IShapeBase partModel | Specify a variable that indicates the Part/Pallet 3D shape data registered in *renderInfoList*. | partModel |
| IVisualizable mechanicalModel | Specify the 3D shape data name for Mechanical Component. | XYTable |
| int targetStep | Specify part/pallet state number. | 1 |
| string controllerName | Specify the Controller name by string. | "new_controller_0" |
| string variableName | Specify a BOOL global variable for the Controller by string. | "GraspPallet" |
| IEnumerable<ShapeRenderInfo> renderInfoList | Specify the name of the Part/Pallet Instance List specified in CreateRenderInfo. | --- |
| IList<Tuple<string, object>> setVariableList = null | Specify a write variable list that you created with the CreateSetVariableList() function.<br>Executing the DetectPartCollision() function adds the names and values of the variables to the write variable list. To write multiple variable values to the Controller, you can add the variable names and values to this write variable list and then set the write variable list as an argument to the SetVariableValues() function to speed up the processing. This argument can be omitted. If it is omitted, null is set. If null is set, the value specified for *variableName* in the DetectPartCollision() function is written to the Controller. | --- |

### ● Description

When the part whose state number is *targetStep*, 3D shape data name that is registered in the Part/Pallet Instance List is *partModel*, and instance name is *name* is colliding with the mechanical component specified by *mechanicalModel*, show the world coordinate in a trace message.
When the BOOL global variable whose Controller name is *controllerName* and global variable name is *variableName* is set, the global variable turns ON.

## DetectMechanicalComponentsCollision

Detect that the mechanical component collides with another mechanical component.

### Function Call

| | |
|---|---|
| **Function** | DetectMechanicalComponentsCollision(IExtendedShapeScript shapeScript, string csName, int targetStep, string controllerName, string variableName, IList<Tuple<string, object>> setVariableList = null) |
| **Used in** | Render() |

### Arguments

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shapeScript | Specify the Shape Script to use in this function. | --- |
| string csName | Set the name(CollisionSourceName) , which is registered in Collision Filter, to *csName*. | "Slider, the overall Mechanical Component" |
| int targetStep | Specify part/pallet state number. | 1 |
| string controllerName | Specify the Controller name by string. | "new_controller_0" |
| string variableName | Specify a BOOL global variable for the Controller by string. | "GraspPallet" |
| IList<Tuple<string, object>> setVariableList = null | Specify a write variable list that you created with the CreateSetVariableList() function.<br>Executing the DetectMechanicalComponetsCollision() function adds the names and values of the variables to the write variable list. To write multiple variable values to the Controller, you can add the variable names and values to this write variable list and then set the write variable list as an argument to the SetVariableValues() function to speed up the processing. This argument can be omitted. If it is omitted, null is set. If null is set, the value specified for *variableName* in the DetectMechanicalComponetsCollision() function is written to the Controller. | --- |

### Description

When the mechanical component whose state number is *targetStep* and specified by *csName* makes a collision, an collided object is displayed in a trace message by the name registered in Collision Filter Group.
When the BOOL global variable whose Controller name is *controllerName* and global variable name is *variableName* is set, the global variable turns ON.

## SetClampStatus

Write the grasping (colliding) status of the robot hand to the variable of the Controller.

### Function Call

| | |
|---|---|
| **Function** | SetClampStatus(IExtendedShapeScript shapeScript, string controllerName, string variableName, IVisualizable robotTool, IShapeBase partModel, IEnumerable<ShapeRenderInfo> renderInfoList, IList<Tuple<string, object>> setVariableList = null) |
| **Used in** | Render() |

## ● Arguments

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shapeScript | Specify the Shape Script to use in this function. | --- |
| string controllerName | Specify the Controller name by string. | "new_controller_0" |
| string variableName | Specify a BOOL global variable for the Controller by string. | "GraspPart" |
| IVisualizable robotTool | Specify a variable equivalent to the 3D shape data picking the part. | pickModel |
| IShapeBase partModel | Specify a variable that indicates the Part/Pallet 3D shape data registered in *renderInfoList*. | partModel |
| IEnumerable<ShapeRenderInfo> renderInfoList | Specify the name of the Part/Pallet Instance List specified in CreateRenderInfo. | --- |
| IList<Tuple<string, object>> setVariableList = null | Specify a write variable list that you created with the CreateSetVariableList function.<br>Executing the SetClampStatus() function adds the names and values of the variables to the write variable list. To write multiple variable values to the Controller, you can add the variable names and values to this write variable list and then set the write variable list as an argument to the SetVariableValues() function to speed up the processing. This argument can be omitted. If it is omitted, null is set. If null is set, the value specified for *variableName* in the SetClampStatus() function is written to the Controller. | --- |

## ● Description

When the 3D shape data designated by *robotTool* has the child (part/pallet) specified by *partModel*, the BOOL global variable whose Controller name is *controllerName* and global variable name is *variableName* is turned ON.

# GetCollidingSourceNames

Get the names of all 3D shape data that collided with the specified 3D shape data.

## ● Function Call

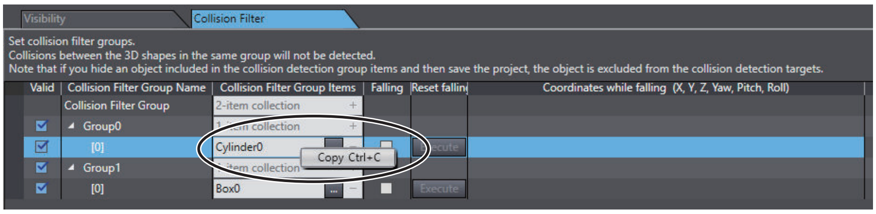
| | |
|---|---|
| **Function** | IEnumerable<string>GetCollidingSourceNames(string collisionSourceName) |
| **Used in** | Render() |

## ● Arguments

| Argument | Description | Notation example |
|---|---|---|
| string collisionSourceName | Specify the name of the 3D shape data as a string. | "Cylinder0" |

## ● Description

Get the names of all 3D shape data that collided with the 3D shape data specified in an argument. The value is the return value of this function.
You can get the *Name of the 3D shape data* specified in the argument from the Collision Filter. In the **Collision Filter** tab page, you can copy the target Collision Filter Group Item by right-clicking it.




## A-1-8 Sharing Variables with the Controller

### GetBoolVariable

Read the value of a BOOL variable from the specified Controller.

#### ● Function Call

| | |
|---|---|
| **Function** | Bool GetBoolVariable(IExtendedShapeScript shapeScript, string controllerName, string variable-Name) |
| **Used in** | Render() |

#### ● Arguments

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shape-Script | Specify the Shape Script to use in this function. | --- |
| string controllerName | Specify the Controller name by string. | "new_controller_0" |
| string variableName | Specify a BOOL global variable for the Controller by string. | --- |

#### ● Description

Read the BOOL global variable whose Controller name is *controllerName* and global variable name is *variableName*. The value is the return value of this function.

### SetBoolVariable

Write values of BOOL variables of the specified Controller.

#### ● Function Call

| | |
|---|---|
| **Function** | SetBoolVariable(IExtendedShapeScript shapeScript, string controllerName, string variableName, bool value) |
| **Used in** | Render() |

#### ● Arguments

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shape-Script | Specify the Shape Script to use in this function. | --- |
| string controllerName | Specify the Controller name by string. | "new_controller_0" |

| Argument | Description | Notation example |
|---|---|---|
| string variableName | Specify a BOOL global variable for the Controller by string. | --- |
| bool value | Specify TRUE or FALSE. | --- |

### ● Description

Write a value to the BOOL global variable whose Controller name is *controllerName* and global variable name is *variableName*.

## GetIntegerVariable

Read the value of a DINT variable from the specified Controller.

### ● Function Call

| | |
|---|---|
| **Function** | int GetIntegerVariable(IExtendedShapeScript shapeScript, string controllerName, string variableName) |
| **Used in** | Render() |

### ● Arguments

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shapeScript | Specify the Shape Script to use in this function. | --- |
| string controllerName | Specify the Controller name by string. | "new_controller_0" |
| string variableName | Specify the DINT global variable name for the Controller with a character string. | --- |

### ● Description

Read the DINT global variable whose Controller name is *controllerName* and global variable name is *variableName*. The value is the return value of this function.

## SetIntegerVariable

Write INT values to DINT variables of the specified Controller.

### ● Function Call

| | |
|---|---|
| **Function** | SetIntegerVariable(IExtendedShapeScript shapeScript, string controllerName, string variableName, int value) |
| **Used in** | Render() |

### ● Arguments

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shapeScript | Specify the Shape Script to use in this function. | --- |
| string controllerName | Specify the Controller name by string. | "new_controller_0" |
| string variableName | Specify the DINT global variable name for the Controller with a character string. | --- |

| Argument | Description | Notation example |
|---|---|---|
| int value | Specify the INT data type value. | --- |

## ● Description

Write a value to the DINT global variable whose Controller name is *controllerName* and global variable name is *variableName*.

# Get**Variable

Read the value of an integer variable from the specified Controller. For "**" of the function name, specify the data type of the Controller variable. Refer to *Function Call* on page A-42 below for the data types of variables and specific function names that you can specify.

## ● Function Call

| | |
|---|---|
| **Function** | sbyte GetSintVariable(IExtendedShapeScript shapeScript, string controllerName, string variableName)<br>short GetIntVariable(IExtendedShapeScript shapeScript, string controllerName, string variableName)<br>int GetDintVariable(IExtendedShapeScript shapeScript, string controllerName, string variableName)<br>long GetLintVariable(IExtendedShapeScript shapeScript, string controllerName, string variableName)<br>byte GetUsintVariable(IExtendedShapeScript shapeScript, string controllerName, string variableName)<br>ushort GetUintVariable(IExtendedShapeScript shapeScript, string controllerName, string variableName)<br>uint GetUdintVariable(IExtendedShapeScript shapeScript, string controllerName, string variableName)<br>ulong GetUlintVariable(IExtendedShapeScript shapeScript, string controllerName, string variableName) |
| **Used in** | Render() |

## ● Arguments

| Item | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shapeScript | Specify the Shape Script to use in this function. | --- |
| string controllerName | Specify the Controller name by string. | "new_controller_0" |
| string variableName | Specify the integer variable name of the Controller by string. | --- |

## ● Description

Read the integer variable whose Controller name is *controllerName* and variable name is *variableName*. The value is the return value of this function.

## Set**Variable

Write values to integer variables of the specified Controller. For "**" of the function name, specify the data type of the Controller variable. Refer to *Function Call* on page A-43 below for the data types of variables and specific function names that you can specify.

### ● Function Call

| | |
|---|---|
| **Function** | SetSintVariable(IExtendedShapeScript shapeScript, string controllerName, string variableName, object value)<br>SetIntVariable(IExtendedShapeScript shapeScript, string controllerName, string variableName, object value)<br>SetDintVariable(IExtendedShapeScript shapeScript, string controllerName, string variableName, object value)<br>SetLintVariable(IExtendedShapeScript shapeScript, string controllerName, string variableName, object value)<br>SetUsintVariable(IExtendedShapeScript shapeScript, string controllerName, string variableName, object value)<br>SetUintVariable(IExtendedShapeScript shapeScript, string controllerName, string variableName, object value)<br>SetUdintVariable(IExtendedShapeScript shapeScript, string controllerName, string variableName, object value)<br>SetUlintVariable(IExtendedShapeScript shapeScript, string controllerName, string variableName, object value) |
| **Used in** | Render() |

### ● Arguments

| Item | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shapeScript | Specify the Shape Script to use in this function. | --- |
| string controllerName | Specify the Controller name by string. | "new_controller_0" |
| string variableName | Specify the integer variable name of the Controller by string. | --- |
| object value | Specify the value to set. Set a value that can be converted to the following data types in the Shape Script.<br>SetSintVariable: sbyte<br>SetIntVariable: short<br>SetDintVariable: int<br>SetLintVariable: long<br>SetUsintVariable: byte<br>SetUintVariable: ushort<br>SetUdintVariable: uint<br>SetUlintVariable: ulong | --- |

### ● Description

Write a value to the integer variable whose Controller name is *controllerName* and variable name is *variableName*. If you specify a value that cannot be written, an error message is displayed in the trace message.

## GetByteArrayVariable

Read the values of a BYTE array variable from the specified Controller.

### ● Function Call

| | |
|---|---|
| **Function** | byte[] GetByteArrayVariable(IExtendedShapeScript shapeScript, string controllerName, string variableName) |
| **Used in** | Render() |

### ● Arguments

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shapeScript | Specify the instance name of the Shape Script. | --- |
| string controllerName | Specify the Controller name by string. | "new_controller_0" |
| string variableName | Specify the BYTE array global variable name for the Controller by string. | --- |

### ● Description

Read the BYTE array global variable whose Controller name is *controllerName* and global variable name is *variableName*. The read values are the return values of this function.

**Version Information**

This function causes an execution error in Sysmac Studio version 1.51 or lower.

## SetByteArrayVariable

Write values to a BYTE array variable in the specified Controller.

### ● Function Call

| | |
|---|---|
| **Function** | SetByteArrayVariable(IExtendedShapeScript shapeScript, string controllerName, string variableName, byte[] values) |
| **Used in** | Render() |

### ● Arguments

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shapeScript | Specify the instance name of the Shape Script. | --- |
| string controllerName | Specify the Controller name by string. | "new_controller_0" |
| string variableName | Specify the BYTE array global variable name for the Controller by string. | --- |
| byte[] values | Specify the BYTE array values. | --- |

### ● Description

Write values to the BYTE array global variable whose Controller name is *controllerName* and global variable name is *variableName*.

**Version Information**

This function causes an execution error in Sysmac Studio version 1.51 or lower.

## GetBoolArrayVariable

Read the values of a BOOL array variable from the specified Controller.

### ● Function Call

| | |
|---|---|
| **Function** | bool[] GetBoolArrayVariable(IExtendedShapeScript shapeScript, string controllerName, string variableName) |
| **Used in** | Render() |

### ● Arguments

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shape-Script | Specify the instance name of the Shape Script. | --- |
| string controllerName | Specify the Controller name by string. | "new_controller_0" |
| string variableName | Specify the BOOL array global variable name for the Controller by string. | --- |

### ● Description

Read the BOOL array global variable whose Controller name is *controllerName* and global variable name is *variableName*. The read values are the return values of this function.

**Version Information**

This function causes an execution error in Sysmac Studio version 1.51 or lower.

## SetBoolArrayVariable

Write values to a BOOL array variable in the specified Controller.

### ● Function Call

| | |
|---|---|
| **Function** | SetBoolArrayVariable(IExtendedShapeScript shapeScript, string controllerName, string variable-Name, bool[] values) |
| **Used in** | Render() |

### ● Arguments

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shape-Script | Specify the instance name of the Shape Script. | --- |
| string controllerName | Specify the Controller name by string. | "new_controller_0" |
| string variableName | Specify the BOOL array global variable name for the Controller by string. | --- |
| bool[] values | Specify the BOOL array values. | --- |

### Description

Write values to the BOOL array global variable whose Controller name is *controllerName* and global variable name is *variableName*.

**Version Information**

This function causes an execution error in Sysmac Studio version 1.51 or lower.

## GetIntegerArrayVariable

Read the values of a DINT array variable from the specified Controller.

### Function Call

| | |
|---|---|
| **Function** | int[] GetIntegerArrayVariable(IExtendedShapeScript shapeScript, string controllerName, string variableName) |
| **Used in** | Render() |

### Arguments

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shapeScript | Specify the instance name of the Shape Script. | --- |
| string controllerName | Specify the Controller name by string. | "new_controller_0" |
| string variableName | Specify the DINT array global variable name for the Controller by string. | --- |

### Description

Read the DINT array global variable whose Controller name is *controllerName* and global variable name is *variableName*. The read values are the return values of this function.

**Version Information**

This function causes an execution error in Sysmac Studio version 1.51 or lower.

## SetIntegerArrayVariable

Write int array variable values to a DINT array variable in the specified Controller.

### Function Call

| | |
|---|---|
| **Function** | SetIntegerArrayVariable(IExtendedShapeScript shapeScript, string controllerName, string variableName, int[] values) |
| **Used in** | Render() |

### Arguments

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shapeScript | Specify the instance name of the Shape Script. | --- |
| string controllerName | Specify the Controller name by string. | "new_controller_0" |

| Argument | Description | Notation example |
|---|---|---|
| string variableName | Specify the DINT array global variable name for the Controller by string. | --- |
| int[] values | Specify the int array values. | --- |

### Description

Write values to the DINT array global variable whose Controller name is *controllerName* and global variable name is *variableName*.

**Version Information**

This function causes an execution error in Sysmac Studio version 1.51 or lower.

## GetWordArrayVariable

Read the values of a WORD array variables from the specified Controller.

### Function Call

| | |
|---|---|
| **Function** | ushort[]GetWordArrayVariable(IExtendedShapeScript shapeScript, string controllerName, string variableName) |
| **Used in** | Render() |

### Arguments

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shape-Script | Specify the instance name of the Shape Script. | --- |
| string controllerName | Specify the Controller name by string. | "new_controller_0" |
| string variableName | Specify the WORD array global variable name for the Controller by string. | --- |

### Description

Read the WORD array global variable whose Controller name is *controllerName* and global variable name is *variableName*. The read values are the return values of this function.

**Version Information**

This function causes an execution error in Sysmac Studio version 1.51 or lower.

## SetWordArrayVariable

Write ushort array variable values to a WORD array variable in the specified Controller.

### Function Call

| | |
|---|---|
| **Function** | SetWordArrayVariable(IExtendedShapeScript shapeScript, string controllerName, string variableName, ushort[] values) |
| **Used in** | Render() |

- **Arguments**

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shape-Script | Specify the instance name of the Shape Script. | --- |
| string controllerName | Specify the Controller name by string. | "new_controller_0" |
| string variableName | Specify the WORD array global variable name for the Controller by string. | --- |
| ushort[] values | Specify the ushort array values. | --- |

- **Description**

Write values to the WORD array global variable whose Controller name is *controllerName* and global variable name is *variableName*.

**Version Information**

This function causes an execution error in Sysmac Studio version 1.51 or lower.

## GetDwordArrayVariable

Read the values of a DWORD array variable from the specified Controller.

- **Function Call**

| | |
|---|---|
| **Function** | uint[]GetDwordArrayVariable(IExtendedShapeScript shapeScript, string controllerName, string variableName) |
| **Used in** | Render() |

- **Arguments**

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shape-Script | Specify the instance name of the Shape Script. | --- |
| string controllerName | Specify the Controller name by string. | "new_controller_0" |
| string variableName | Specify the DWORD array global variable name for the Controller by string. | --- |

- **Description**

Read the DWORD array global variable whose Controller name is *controllerName* and global variable name is *variableName*. The read values are the return values of this function.

**Version Information**

This function causes an execution error in Sysmac Studio version 1.51 or lower.

## SetDwordArrayVariable

Write uint array variable values to a DWORD array variable in the specified Controller.

● **Function Call**

| | |
|---|---|
| **Function** | SetDwordArrayVariable(IExtendedShapeScript shapeScript, string controllerName, string variableName, uint[] values) |
| **Used in** | Render() |

● **Arguments**

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shapeScript | Specify the instance name of the Shape Script. | --- |
| string controllerName | Specify the Controller name by string. | "new_controller_0" |
| string variableName | Specify the DWORD array global variable name for the Controller by string. | --- |
| uint[] values | Specify the uint array values. | --- |

● **Description**

Write values to the DWORD array global variable whose Controller name is *controllerName* and global variable name is *variableName*.

**Version Information**

This function causes an execution error in Sysmac Studio version 1.51 or lower.

## GetLwordArrayVariable

Read the values of an LWORD array variable from the specified Controller.

● **Function Call**

| | |
|---|---|
| **Function** | ulong[]GetLwordArrayVariable(IExtendedShapeScript shapeScript, string controllerName, string variableName) |
| **Used in** | Render() |

● **Arguments**

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shapeScript | Specify the instance name of the Shape Script. | --- |
| string controllerName | Specify the Controller name by string. | "new_controller_0" |
| string variableName | Specify the LWORD array global variable name for the Controller by string. | --- |

● **Description**

Read the LWORD array global variable whose Controller name is *controllerName* and global variable name is *variableName*. The read values are the return values of this function.

**Version Information**

This function causes an execution error in Sysmac Studio version 1.51 or lower.

## SetLwordArrayVariable

Write ulong array variable values to an LWORD array variable in the specified Controller.

### ● Function Call

| | |
|---|---|
| **Function** | SetLwordArrayVariable(IExtendedShapeScript shapeScript, string controllerName, string varia-bleName, ulong[] values) |
| **Used in** | Render() |

### ● Arguments

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shapeScript | Specify the instance name of the Shape Script. | --- |
| string controllerName | Specify the Controller name by string. | "new_controller_0" |
| string variableName | Specify the LWORD array global variable name for the Controller by string. | --- |
| ulong[] values | Specify the ulong array values. | --- |

### ● Description

Write values to the LWORD array global variable whose Controller name is *controllerName* and global variable name is *variableName*.

**Version Information**

This function causes an execution error in Sysmac Studio version 1.51 or lower.

## GetRealArrayVariable

Read the values of a REAL array variable from the specified Controller.

### ● Function Call

| | |
|---|---|
| **Function** | float[]GetRealArrayVariable(IExtendedShapeScript shapeScript, string controllerName, string variableName) |
| **Used in** | Render() |

### ● Arguments

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shape-Script | Specify the instance name of the Shape Script. | --- |
| string controllerName | Specify the Controller name by string. | "new_controller_0" |
| string variableName | Specify the REAL array global variable name for the Controller by string. | --- |

### ● Description

Read the REAL array global variable whose Controller name is *controllerName* and global variable name is *variableName*. The read values are the return values of this function.

**Version Information**

This function causes an execution error in Sysmac Studio version 1.51 or lower.

## SetRealArrayVariable

Write float array variable values to a REAL array variable in the specified Controller.

### ● Function Call

| | |
|---|---|
| **Function** | SetRealArrayVariable(IExtendedShapeScript shapeScript, string controllerName, string variable-Name, float[] values) |
| **Used in** | Render() |

### ● Arguments

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shape-Script | Specify the instance name of the Shape Script. | --- |
| string controllerName | Specify the Controller name by string. | "new_controller_0" |
| string variableName | Specify the REAL array global variable name for the Controller by string. | --- |
| float[] values | Specify the float array values. | --- |

### ● Description

Write a value to the REAL array global variable whose Controller name is *controllerName* and glob-al variable name is *variableName*.

**Version Information**

This function causes an execution error in Sysmac Studio version 1.51 or lower.

## GetLrealArrayVariable

Read the values of an LREAL array variable from the specified Controller.

### ● Function Call

| | |
|---|---|
| **Function** | double[]GetLrealArrayVariable(IExtendedShapeScript shapeScript, string controllerName, string variableName) |
| **Used in** | Render() |

### ● Arguments

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shape-Script | Specify the instance name of the Shape Script. | --- |
| string controllerName | Specify the Controller name by string. | "new_controller_0" |
| string variableName | Specify the LREAL array global variable name for the Controller by string. | --- |

### Description

Read the LREAL array global variable whose Controller name is *controllerName* and global variable name is *variableName*. The read values are the return values of this function.

**Version Information**

This function causes an execution error in Sysmac Studio version 1.51 or lower.

## SetLrealArrayVariable

Write double array variable values to an LREAL array variable in the specified Controller.

### Function Call

| | |
|---|---|
| **Function** | SetLrealArrayVariable(IExtendedShapeScript shapeScript, string controllerName, string variableName, double[] values) |
| **Used in** | Render() |

### Arguments

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shapeScript | Specify the instance name of the Shape Script. | --- |
| string controllerName | Specify the Controller name by string. | "new_controller_0" |
| string variableName | Specify the LREAL array global variable name for the Controller by string. | --- |
| double[] values | Specify the double array values. | --- |

### Description

Write values to the LREAL array global variable whose Controller name is *controllerName* and global variable name is *variableName*.

**Version Information**

This function causes an execution error in Sysmac Studio version 1.51 or lower.

## GetStringArrayVariable

Read the values of a STRING array variable from the specified Controller.

### Function Call

| | |
|---|---|
| **Function** | string[] GetStringArrayVariable(IExtendedShapeScript shapeScript, string controllerName, string variableName) |
| **Used in** | Render() |

### Arguments

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shapeScript | Specify the instance name of the Shape Script. | --- |

| Argument | Description | Notation example |
|---|---|---|
| string controllerName | Specify the Controller name by string. | "new_controller_0" |
| string variableName | Specify the STRING array global variable name for the Controller by string. | --- |

### ● Description

Read the STRING array global variable whose Controller name is *controllerName* and global variable name is *variableName*. The read values are the return values of this function.

**Version Information**

This function causes an execution error in Sysmac Studio version 1.51 or lower.

## SetStringArrayVariable

Write values to a STRING array variable in the specified Controller.

### ● Function Call

| | |
|---|---|
| **Function** | SetStringArrayVariable(IExtendedShapeScript shapeScript, string controllerName, string variableName, string[] values) |
| **Used in** | Render() |

### ● Arguments

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shapeScript | Specify the instance name of the Shape Script. | --- |
| string controllerName | Specify the Controller name by string. | "new_controller_0" |
| string variableName | Specify the STRING array global variable name for the Controller by string. | --- |
| string[] values | Specify the string array values. | --- |

### ● Description

Write values to the STRING array global variable whose Controller name is *controllerName* and global variable name is *variableName*.

**Version Information**

This function causes an execution error in Sysmac Studio version 1.51 or lower.

## Get**ArrayVariable

Read the values of an integer array variable from the specified Controller. For "**" of the function name, specify the data type of the Controller variable. Refer to *Function Call* on page A-54 below for the data types of variables and specific function names that you can specify.

### ● Function Call

| | |
|---|---|
| **Func-tion** | sbyte[] GetSintArrayVariable(IExtendedShapeScript shapeScript, string controllerName, string variableName)<br>short[] GetIntArrayVariable(IExtendedShapeScript shapeScript, string controllerName, string variableName)<br>int[] GetDintArrayVariable(IExtendedShapeScript shapeScript, string controllerName, string variableName)<br>long[] GetLintArrayVariable(IExtendedShapeScript shapeScript, string controllerName, string variableName)<br>byte[] GetUsintArrayVariable(IExtendedShapeScript shapeScript, string controllerName, string variableName)<br>ushort[] GetUintArrayVariable(IExtendedShapeScript shapeScript, string controllerName, string variableName)<br>uint[] GetUdintArrayVariable(IExtendedShapeScript shapeScript, string controllerName, string variableName)<br>ulong[] GetUlintArrayVariable(IExtendedShapeScript shapeScript, string controllerName, string variableName) |
| **Used in** | Render() |

### ● Arguments

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shape-Script | Specify the instance name of the Shape Script. | --- |
| string controllerName | Specify the Controller name by string. | "new_controller_0" |
| string variableName | Specify the integer variable name for the Controller by string. | --- |

### ● Description

Read the integer array variable whose Controller name is *controllerName* and variable name is *variableName*. The read values are the return values of this function.

**Version Information**

This function causes an execution error in Sysmac Studio version 1.51 or lower.

## Set**ArrayVariable

Write values to an integer array variable in the specified Controller. For "**" of the function name, specify the data type of the Controller variable. Refer to *Function Call* on page A-55 below for the data types of variables and specific function names that you can specify.

### Function Call

| | |
|---|---|
| **Func-tion** | SetSintArrayVariable(IExtendedShapeScript shapeScript, string controllerName, string variable-Name, object value)<br>SetIntArrayVariable(IExtendedShapeScript shapeScript, string controllerName, string variable-Name, object value)<br>SetDintArrayVariable(IExtendedShapeScript shapeScript, string controllerName, string variable-Name, object value)<br>SetLintArrayVariable(IExtendedShapeScript shapeScript, string controllerName, string variable-Name, object value)<br>SetUsintArrayVariable(IExtendedShapeScript shapeScript, string controllerName, string variable-Name, object value)<br>SetUintArrayVariable(IExtendedShapeScript shapeScript, string controllerName, string variable-Name, object value)<br>SetUdintArrayVariable(IExtendedShapeScript shapeScript, string controllerName, string variable-Name, object value)<br>SetUlintArrayVariable(IExtendedShapeScript shapeScript, string controllerName, string variable-Name, object value) |
| **Used in** | Render() |

### Arguments

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShape-Script shapeScript | Specify the instance name of the Shape Script. | --- |
| string controller-Name | Specify the Controller name by string. | "new_controller_0" |
| string variable-Name | Specify the integer array variable name for the Controller by string. | --- |
| object value | Specify the value to set. Set a value that can be converted to the following data types in the Shape Script.<br>SetSintArrayVariable: sbyte[]<br>SetIntArrayVariable: short[]<br>SetDintArrayVariable: int[]<br>SetLintArrayVariable: long[]<br>SetUsintArrayVariable: byte[]<br>SetUintArrayVariable: ushort[]<br>SetUdintArrayVariable: uint[]<br>SetUlintArrayVariable: ulong[] | --- |

### Description

Write values to the integer array variable whose Controller name is *controllerName* and variable name is *variableName*. If you specify a value that cannot be written, an error message is displayed in the Trace Message tab page.

**Version Information**

This function causes an execution error in Sysmac Studio version 1.51 or lower.

## CreateGetVariableList

Create a new read variable list, which is used to read multiple variables from the Controller.

### ● Function Call

| Function | IList<Tuple<string, Type>> CreateGetVariableList() |
|---|---|
| Used in | Render() |

### ● Arguments

None

### ● Description

Create a new read variable list, which you specify as an argument to the GetVariableValues() function. The newly created empty read variable list is the return value of the function. Use this function in combination with the AddToGetVariableList() or GetVariableValues() function.

The following example shows how to get at a time the BOOL variables "boolVar1", "boolVar2", and "boolVar3" from the Controller "new_Controller_0".

```
62      public override void Render(IEnumerable<ShapeRenderInfo> renderInfoList) {
63          if (this.IsInitialized == false) {
64              DefaultFunctions.InitializeObject(this, renderInfoList);
65              return ;
66          }
67
68          var getVariableList = DefaultFunctions.CreateGetVariableList();
69          DefaultFunctions.AddToGetVariableList("boolVar1", "BOOL", getVariableList);
70          DefaultFunctions.AddToGetVariableList("boolVar2", "BOOL", getVariableList);
71          DefaultFunctions.AddToGetVariableList("boolVar3", "BOOL", getVariableList);
72
73          var variableValueList = DefaultFunctions.GetVariableValues(this, "new_Controller_0", getVariableList);
74      }
```

Create a new read variable list in line 68. Next, in lines 69 to 71, use the AddToGetVariableList() function to add the variables to read to the read variable list. Then, in line 73, use the GetVariable-Values() function to get the variable list that you read from the Controller.

## AddToGetVariableList

Add the variable to read from the Controller to the read variable list.

### ● Function Call

| Function | bool AddToGetVariableList(string variableName, string iecDataType, IList<Tuple<string, Type>> getVariableList, bool isShowErrorMessage = true) |
|---|---|
| Used in | Render() |

### ● Arguments

| Item | Description | Notation example |
|---|---|---|
| string variableName | Specify the Controller variable name by string. | --- |
| string iecDataType | Specify the data type of the variable by string. | "BOOL" |
| IList<Tuple<string, Type>> getVariableList | Specify a reference to the read variable list. | --- |
| bool isShowErrorMessage | Set whether to display an error message in the trace message if an addition to the read variable list fails. This argument can be omitted. If it is omitted, an error message is displayed in the trace message. | true |

### ● Description

Add the variable name *variableName* and the data type *iecDataType* to the read variable list *getVariableList*. When the variable is successfully added, the return value is TRUE. If *iecDataType* is set to an illegal value or the variable to add is already added, the return value is FALSE. In addition, if *isShowErrorMessage* is TRUE, an error message is displayed in the trace message.

## GetVariableValues

Read the values of multiple variables from the Controller at a time.

### ● Function Call

| | |
|---|---|
| **Function** | IDictionary<string, object> GetVariableValues(IExtendedShapeScript shapeScript, string controllerName, IList<Tuple<string, Type>> getVariableList) |
| **Used in** | Render() |

### ● Arguments

| Item | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shapeScript | Specify the Shape Script to use in this function. | --- |
| string controllerName | Specify the Controller name by string. | "new_controller_0" |
| IList<Tuple<string, Type>> getVariableList | Specify a reference to the read variable list. | --- |

### ● Description

Read the values of the Controller variables based on the variable names added to the read variable list. The combination of the variable names and values of the read variables is the return value of this function.

**Version Information**

This function causes a compiling error in Sysmac Studio version 1.40.

## GetValueFromVariableValueList

Get the value of a variable that you read with the GetVariableValues() function.

### ● Function Call

| | |
|---|---|
| **Function** | object GetValueFromVariableValueList(string variableName, IDictionary<string, object> variableValueList) |
| **Used in** | Render() |

### ● Arguments

| Item | Description | Notation example |
|---|---|---|
| string variableName | Specify the Controller variable name by string. | --- |
| IDictionary<string, object> variableValueList | Specify the variable value list that you read with the GetVariableValues() function. | --- |

### Description

Get the value of a variable that you read with the GetVariableValues() function. The variable value is the return value of this function. If the value that you obtained does not correspond to the specified variable name, the return value is null.

The type of the return value depends on the data type of the Controller variable. The following shows the correspondence between the data types of Controller variables and the data types of Shape Script variables.

| Data type of Controller variable | Data type of Shape Script variable |
|---|---|
| BOOL | bool |
| SINT | sbyte |
| USINT | byte |
| INT | short |
| UINT | ushort |
| DINT | int |
| UDINT | uint |
| LINT | long |
| ULINT | ulong |
| BYTE | byte |
| WORD | ushort |
| DWORD | uint |
| LWORD | ulong |
| REAL | float |
| LREAL | double |
| TIME | TimeSpan |
| DATE | DateTime |
| TIME_OF_DAY | DateTime |
| DATE_AND_TIME | DateTime |
| STRING | string |

The following example shows how to get the value of the variable "boolVar1" that you read.

```
62      public override void Render(IEnumerable<ShapeRenderInfo> renderInfoList) {
63          if (this.IsInitialized == false) {
64              DefaultFunctions.InitializeObject(this, renderInfoList);
65              return ;
66          }
67
68          var getVariableList = DefaultFunctions.CreateGetVariableList();
69          DefaultFunctions.AddToGetVariableList("boolVar1", "BOOL", getVariableList);
70          DefaultFunctions.AddToGetVariableList("boolVar2", "BOOL", getVariableList);
71          DefaultFunctions.AddToGetVariableList("boolVar3", "BOOL", getVariableList);
72
73          var variableValueList = DefaultFunctions.GetVariableValues(this, "new_Controller_0", getVariableList);
74          var result = (bool)DefaultFunctions.GetValueFromVariableValueList("boolVar1", variableValueList);
75      }
```

In line 73, use the GetVariableValues() function to get the variable list that you read from the Controller. In line 74, get only the value of the variable to read from the variable value list. Cast the obtained value to the data type of the Shape Script variable depending on the data type of the Controller variable.

## CreateSetVariableList

Create a new write variable list, which is used to write multiple variables from the Controller.

## ● Function Call

| Function | IList<Tuple<string, object>> CreateSetVariableList() |
|---|---|
| Used in | Render() |

## ● Arguments

None

## ● Description

Create a new write variable list, which you specify as an argument to the SetVariableValues() function. The newly created empty write variable list is the return value of the function. Use this function in combination with the AddToSetVariableList() or SetVariableValues() function.

The following example shows how to write at a time "1", "2", and "3" to the INT variables "intVar1", "intVar2", and "intVar3" of the Controller "new_Controller_0", respectively.

```
62 public override void Render(IEnumerable<ShapeRenderInfo> renderInfoList) {
63     if (this.IsInitialized == false) {
64         DefaultFunctions.InitializeObject(this, renderInfoList);
65         return ;
66     }
67
68     var setVariableList = DefaultFunctions.CreateSetVariableList();
69     DefaultFunctions.AddToSetVariableList("intVar1", "INT", 1, setVariableList);
70     DefaultFunctions.AddToSetVariableList("intVar2", "INT", 2, setVariableList);
71     DefaultFunctions.AddToSetVariableList("intVar3", "INT", 3, setVariableList);
72
73     DefaultFunctions.SetVariableValues(this, "new_Controller_0", setVariableList);
74 }
```

Create a new write variable list in line 68. Next, in lines 69 to 71, use the AddToSetVariableList() function to add the values to write to the write variable list. Then, in line 73, use the SetVariableValues() function to write the values to the Controller variables.

# AddToSetVariableList

Add the variable and value to write to the Controller to the write variable list.

## ● Function Call

| Function | bool AddToSetVariableList(string variableName, string iecDataType, object setValue, IList<Tuple<string, object>> setVariableList, bool isShowErrorMessage = true) |
|---|---|
| Used in | Render() |

## ● Arguments

| Item | Description | Notation example |
|---|---|---|
| string variableName | Specify the Controller variable name by string. | --- |
| string iecDataType | Specify the data type of the variable by string. | "BOOL" |
| object setValue | Specify the value to write to the Controller variable. | --- |
| IList<Tuple<string, object>> setVariableList | Specify a reference to the write variable list. | --- |
| bool isShowErrorMessage | Set whether to display an error message in the trace message if an addition to the read variable list fails. This argument can be omitted. If it is omitted, an error message is displayed in the trace message. | true |

● **Description**

Add the variable name *variableName* and the set value *setValue* to the write variable list *setVariableList*.

When the variable is successfully added, the return value is TRUE. In the following cases, the return value changes to FALSE and the set value is not written to the write variable list.

- *iecDataType* is set to an illegal value.
- *setValue* has an illegal value.
- The variable to add is already added.

In addition, if *isShowErrorMessage* is TRUE, an error message is displayed in the trace message.

## SetVariableValues

Write values to multiple Controller variables at a time.

● **Function Call**

| | |
|---|---|
| **Function** | SetVariableValues(IExtendedShapeScript shapeScript, string controllerName, IList<Tuple<string, object>> setVariableList) |
| **Used in** | Render() |

● **Arguments**

| Item | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shapeScript | Specify the Shape Script to use in this function. | --- |
| string controllerName | Specify the Controller name by string. | "new_controller_0" |
| IList<Tuple<string, object>> setVariableList | Specify a reference to the write variable list. | --- |

● **Description**

Write values to the Controller variables based on the variable names added to the read variable list.

**Version Information**

This function causes a compiling error in Sysmac Studio version 1.40.

## GetCurrentControllerTime

Read the internal simulation time of the specified Controller.

● **Function Call**

| | |
|---|---|
| **Function** | DateTime GetCurrentControllerTime(IExtendedShapeScript shapeScript, string controllerName) |
| **Used in** | Render() |

● **Arguments**

| Item | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shapeScript | Specify the Shape Script to use in this function. | --- |
| string controllerName | Specify the Controller name by string. | "new_controller_0" |

### Description

Read the internal simulation time of the Controller whose name is *controllerName*. The value is the return value of this function.

## A-1-9 Cooperation with the Process Manager

### MovePartOnPackManagerSensorLatchBelt

Move the part that is present on the conveyor in the Process Manager to the latched position to allow it to be continuously manipulated by the emulation function of the Process Manager.
This is effective if you use a belt latch sensor in the configuration of the Process Manager.

#### Function Call

| | |
|---|---|
| **Function** | MovePartOnPackManagerSensorLatchBelt(IExtendedShapeScript shapeScript, IProcessManager processManager, IBelt beltModel, IEnumerable<ShapeRenderInfo> renderInfoList) |
| **Used in** | Render() |

#### Arguments

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shapeScript | Specify the Shape Script to use in this function. | --- |
| IProcessManager processManager | Ahead of this function, define a variable to which the Process Manager registered in the Application Manager was assigned in advance. | IProcessManager processManager=(IProcessManager)ace["/ApplicationManager0/Process Manager"]; |
| IBelt beltModel | Define a variable to which the conveyor (belt) registered in the Process Manager was assigned. | IBelt beltModel=(IBelt)ace["/ApplicationManager0/Pick Belt"]; |
| IEnumerable<ShapeRenderInfo> renderInfoList | Specify the name of the Part/Pallet Instance List specified in CreateRenderInfo. | --- |

#### Description

Among the part’s 3D shape data registered in the Part/Pallet Instance List, move one that is the child of the conveyor’s 3D shape data *beltModel* to the latched position according to the movement of the conveyor (belt) operated by the emulation function of the Process Manager. The part that was moved to the latched position is then manipulated by the Process Manager.

**Version Information**

This function causes a compiling error in Sysmac Studio version 1.40.

### MovePalletOnPackManagerSensorLatchBelt

Move the pallet that is present on the conveyor in the Process Manager to the latched position to allow it to be continuously manipulated by the emulation function of the Process Manager.
This is effective if you use a belt latch sensor in the configuration of the Process Manager.

● **Function Call**

| | |
|---|---|
| **Function** | MovePalletOnPackManagerSensorLatchBelt(IExtendedShapeScript shapeScript, IProcessManager processManager, IBelt beltModel, IEnumerable<ShapeRenderInfo> renderInfoList) |
| **Used in** | Render() |

● **Arguments**

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shapeScript | Specify the Shape Script to use in this function. | --- |
| IProcessManager processManager | Ahead of this function, define a variable to which the Process Manager registered in the Application Manager was assigned in advance. | IProcessManager processManager=(IProcessManager)ace["/ApplicationManager0/Process Manager"]; |
| IBelt beltModel | Define a variable to which the conveyor (belt) registered in the Process Manager was assigned. | IBelt beltModel=(IBelt)ace["/ApplicationManager0/Pick Belt"]; |
| IEnumerable<ShapeRenderInfo> renderInfoList | Specify the name of the Part/Pallet Instance List specified in CreateRenderInfo. | --- |

● **Description**

Among the pallet's 3D shape data registered in the Part/Pallet Instance List, move one that is the child of the conveyor's 3D shape data *beltModel* to the latched position according to the movement of the conveyor (belt) operated by the emulation function of the Process Manager. This also moves the part that is placed on the pallet. The pallet that was moved to the latched position is then manipulated by the Process Manager.

**Version Information**

This function causes a compiling error in Sysmac Studio version 1.40.

## SetPackManagerPartPosition

Write the position of the part that is manipulated in the Process Manager to a Controller variable.

● **Function Call**

| | |
|---|---|
| **Function** | SetPackManagerPart Position(IExtendedShapeScript shapeScript, string controllerName, string variableDX, string variableDY, string variableDZ, IShapeBase partModel, IEnumerable<ShapeRenderInfo> renderInfoList) |
| **Used in** | Render() |

● **Arguments**

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shapeScript | Specify the Shape Script to use in this function. | --- |
| string ControllerName | Specify the Controller name by string. | "new_Controller_0" |
| string variableDX | Specify the LREAL global array variable name of the Controller that stores the X position of the part by string. | "PositionX" |

| Argument | Description | Notation example |
|---|---|---|
| string variableDY | Specify the LREAL global array variable name of the Controller that stores the Y position of the part by string. | "PositionY" |
| string variableDZ | Specify the LREAL global array variable name of the Controller that stores the Z position of the part by string. | "PositionZ" |
| IShapeBase partModel | Specify a variable that designates the 3D shape data for a part registered in *renderInfoList*. | partModel |
| IEnumerable<ShapeRenderInfo> renderInfoList | Specify the name of the Part/Pallet Instance List specified in CreateRenderInfo. | --- |

### ● Description

Write the X, Y, and Z positions of the part that is manipulated in the Process Manager to LREAL global array variables of the Controller. The part that is manipulated in the Process Manager is assigned an ID number that starts from 0, which is used as a subscript in the array. In other words, the X position of a part with an ID of 0 is written to PositionX[0] and the X position of a part with an ID of 1 is written to PositionX[1]. If you set the variable name to null, the position value is not written anywhere.
In this case, the part must be created by the Shape Script.

**Version Information**

This function causes a compiling error in Sysmac Studio version 1.40.

## UpdatePartDataFromPackManager

Allow the Shape Script to continue to manipulate the part that was picked up and placed in the Process Manager.

### ● Function Call

| | |
|---|---|
| **Function** | UpdatePartDataFromPackManager(IShapeBase partModel, IVisualizable locationModel, IEnumerable<ShapeRenderInfo> renderInfoList) |
| **Used in** | Render() |

### ● Arguments

| Argument | Description | Notation example |
|---|---|---|
| IShapeBase partModel | Specify a variable that designates the 3D shape data for a part registered in *renderInfoList*. | partModel |
| IVisualizable locationModel | Specify a variable that designates the 3D shape data in which the Process Manager placed the part. | locationModel |
| IEnumerable<ShapeRenderInfo> renderInfoList | Specify the name of the Part/Pallet Instance List specified in CreateRenderInfo. | --- |

### ● Description

Allow the Shape Script to manipulate again the created part after the part is picked up and placed in the Process Manager.

**Version Information**

This function causes a compiling error in Sysmac Studio version 1.40.

## UpdatePartDataOnPalletFromPackManager

Allow the Shape Script to continue to manipulate the part that was picked up and placed on the pallet in the Process Manager.

### ● Function Call

| | |
|---|---|
| **Function** | UpdatePartDataOnPalletFromPackManager(IExtendedShapeScript shapeScript, IShapeBase partModel, IShapeBase palletModel, IVisualizable locationModel, IEnumerable<ShapeRenderInfo> renderInfoList) |
| **Used in** | Render() |

### ● Arguments

| Argument | Description | Notation example |
|---|---|---|
| IExtendedShapeScript shapeScript | Specify the Shape Script to use in this function. | --- |
| IShapeBase partModel | Specify a variable that designates the 3D shape data for a part registered in *renderInfoList*. | partModel |
| IShapeBase palletModel | Specify a variable that indicates the pallet 3D shape data registered in *renderInfoList*. | palletModel |
| IVisualizable locationModel | Specify a variable that designates the 3D shape data that is the parent of the pallet. | locationModel |
| IEnumerable<ShapeRenderInfo> renderInfoList | Specify the name of the Part/Pallet Instance List specified in CreateRenderInfo. | --- |

### ● Description

Allow the Shape Script to manipulate again the created pallet and part on it together after the part is picked up and placed in the Process Manager.

**Version Information**

This function causes a compiling error in Sysmac Studio version 1.40.

# A-1-10 Measuring Execution Time of Shape Scripts

## StartStopwatch

Start measurement of execution time of a Shape Script. This function is used with the StopStopwatch() function to measure the time that the computer spends executing the Shape Script code between the StartStopwatch() function and StopStopwatch() function.

### ● Function Call

| | |
|---|---|
| **Function** | void StartStopwatch(string name) |

| Used in | Render() |
|---|---|

● **Arguments**

| Argument | Description | Notation example |
|---|---|---|
| string name | Set the name to identify measurement of execution time of a Shape Script. | "Stopwatch0" |

● **Description**

Start measurement of execution time of the Shape Script specified by the name set in the argument. This function measures the execution time of the Shape Script code between this function and the StopStopwatch() function that has the same argument. The measured time is displayed in the System Monitor. In the System Monitor, you can check execution time on a cycle basis.

## StopStopwatch

Stop measurement of execution time of a Shape Script. This function is used with the StartStopwatch() function to measure the time that the computer spends executing the Shape Script code between the StartStopwatch() function and StopStopwatch() function.

● **Function Call**

| Function | void StopStopwatch(string name) |
|---|---|
| Used in | Render() |

● **Arguments**

| Argument | Description | Notation example |
|---|---|---|
| string name | Set the name to identify measurement of execution time of a Shape Script. | "Stopwatch0" |

● **Description**

Stop measurement of execution time of the Shape Script specified by the name set in the argument.

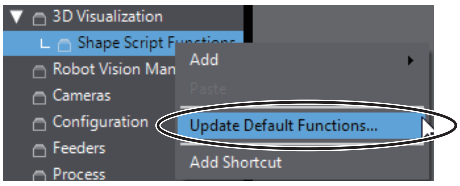
# A-1-11 Updating Functions Used in Shape Scripts

Upgrading the Sysmac Studio to a higher version may add new functions to the Shape Scripts or enhance the existing functions.
*Update Default Functions* updates existing Shape Scripts functions used in a project if any change has been made for Shape Scripts functions.
This feature targets the functions in Default Functions under Shape Script Functions.
You can perform *Update Default Functions* only when the Shape Scripts in the Shape Script Sequence have not yet been executed.

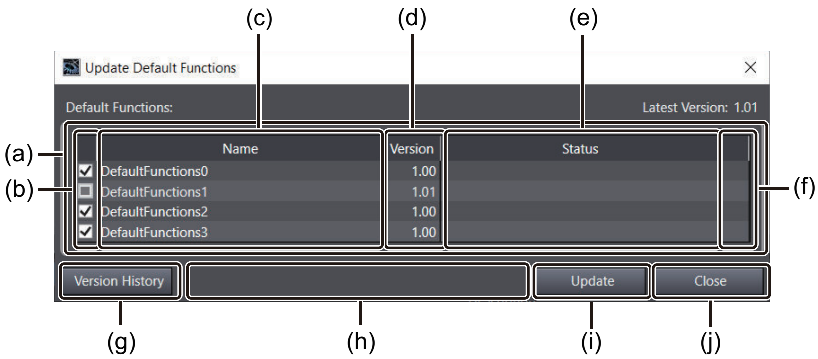
## Procedure to Update Default Functions

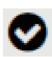
Use the following procedure to update Default Functions.

**1** Right-click **Shape Script Functions** under **Configurations and Setup** – **3D Visualization** in the Multiview Explorer and select **Update Default Functions** from the menu.




The **Update Default Functions** dialog box is displayed.



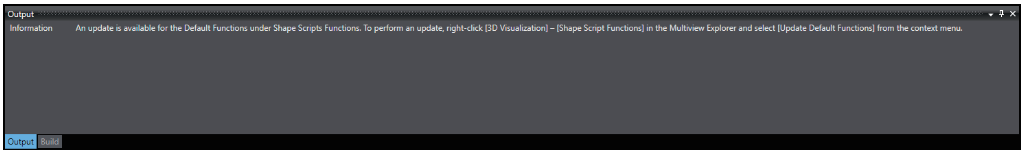

<table>
<tr><th></th><th>Item</th><th>Description</th></tr>
<tr><td>(a)</td><td>Default Functions list</td><td>All Default Functions are listed.<br>Default Functions whose Version is the latest version or later will be grayed out.<br>You can click the column heading Name or Version to sort the list.</td></tr>
<tr><td>(b)</td><td>Check boxes</td><td>Use these check boxes to set whether to update Default Functions that are not grayed out.<br>Checked (default): The Default Functions will be updated.<br>Unchecked: The Default Functions will not be updated.</td></tr>
<tr><td>(c)</td><td>Name</td><td>The names of Default Functions are displayed.</td></tr>
<tr><td>(d)</td><td>Version</td><td>The versions of Default Functions are displayed.</td></tr>
<tr><td>(e)</td><td>Status</td><td>The status of the update processing for Default Functions that you set to update using the check boxes in (b) is displayed.
<table>
<tr><th>Displayed message</th><th>Status</th></tr>
<tr><td>Blank</td><td>Waiting for execution</td></tr>
<tr><td>Updating</td><td>Executing</td></tr>
<tr><td>Updated to latest template.</td><td>Execution completed normally</td></tr>
<tr><td>Changes detected. Not updated. *1</td><td>Execution completed abnormally</td></tr>
</table></td></tr>
<tr><td>(f)</td><td>Status icon</td><td>When the update processing of Default Functions is completed, execution result icons appear.
<table>
<tr><th>Displayed icon</th><th>Status</th></tr>
<tr><td></td><td>Execution completed normally</td></tr>
<tr><td></td><td>Execution completed abnormally</td></tr>
</table></td></tr>
</table>

| | Item | Description |
|---|---|---|
| (g) | **Version History** button | Click this button to display a change history list for all versions of Default Functions that were updated in the past. |
| (h) | Progress bar | This shows the execution status of the update processing. |
| (i) | **Update** button | Click this button to execute update processing. *2 |
| (j) | **Close** button | Click this button to close the dialog box. |

*1. Default Functions will not be updated if they are manually edited.

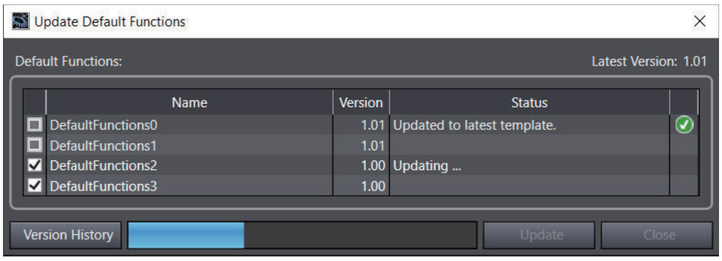
*2. This button is enabled under the following conditions.

- The Output tab page displays the following message when a Sysmac Studio project is opened.

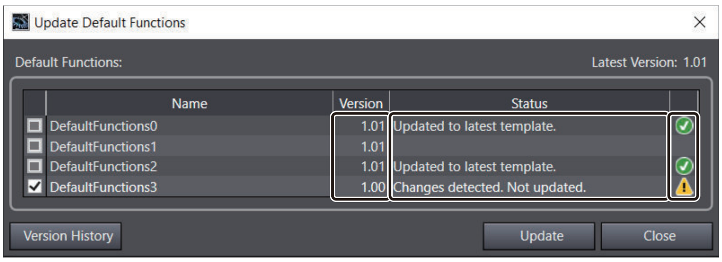



- One or more check boxes are selected in (b).

**2** Click the **Update** button in the **Update Default Functions** dialog box.
The update processing starts.




All target Default Functions are updated sequentially.
In the **Status** column, the progress of the update processing is displayed. When the update processing is completed, status icons appear. If the update processing is completed normally, the **Version** column will display the updated versions.

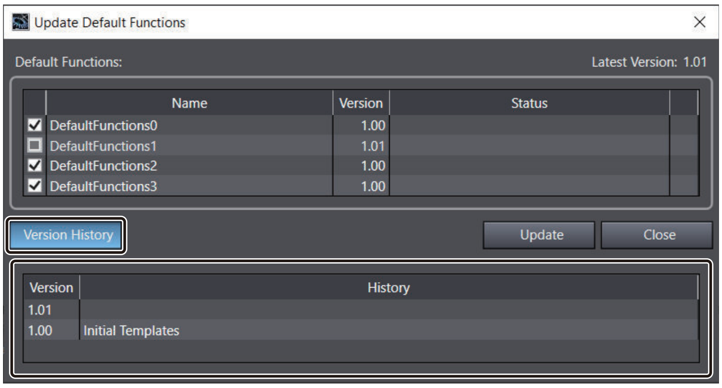



### Precautions for Correct Use

The CreateRenderInfo() and Render() functions will not be updated.

### Additional Information

Clicking the **Version History** button in the **Update Default Functions** dialog box displays a change history list for all versions of Default Functions that were updated in the past. Use this information as a reference in selecting the Default Functions to update.



# A-2 Differences between the Simulator and the Physical Controller

## A-2-1 Operation of Functions

The Simulator has the following functional differences in comparison with the physical Controller.

| Item | | Differences between Simulator and physical Controller |
|---|---|---|
| Mechanical component | Motion on the 3D Visualizer while the Simulator is running | In the Simulator, a mechanical component moves according to the command values for each axis. Physical elements such as weight, gravity, and inertia are not considered. |
| Robot | Motion on the 3D Visualizer while the Simulator is running | In the Simulator, a robot moves according to the command values for each axis. Physical elements such as weight, gravity, and inertia are not considered. |
| Belt | Latch function | When **Latch** is selected in the belt properties of a part, it is possible to generate an external trigger at a specified interval during simulation in the Process Manager. It is also possible to generate a trigger from the CreateLatch() function in the Shape Script. |
| | Belt encoder function | In the Simulator, the drive output for belt control can be turned ON and OFF to operate an encoder. |
| Part detection sensor | Limitations of part detection sensor | The function of a part detection sensor is to detect the passage of a part on a 3D Visualizer. Although it looks like a photoelectric sensor, the following functions are not provided. Therefore, do not use it to adjust the layout of photoelectric sensors, etc.<br>• Thickness adjustment for the optical axis<br>• Simulation of beam reflection<br>• Simulation of hysteresis |

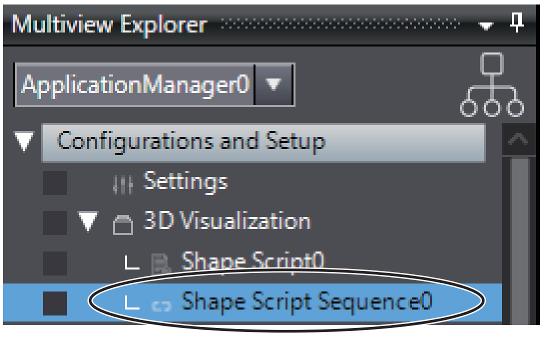
# A-3 Execution Time of Shape Scripts

The execution time of a Shape Script depends on the script's content. If you write a script that takes a long processing time, its execution time may exceed the **Shape Script Execution Ratio** setting for the Shape Script Sequence.
This section describes the procedures to check and reduce the execution time of a Shape Script.

## A-3-1 Checking the Execution Time with the System Monitor

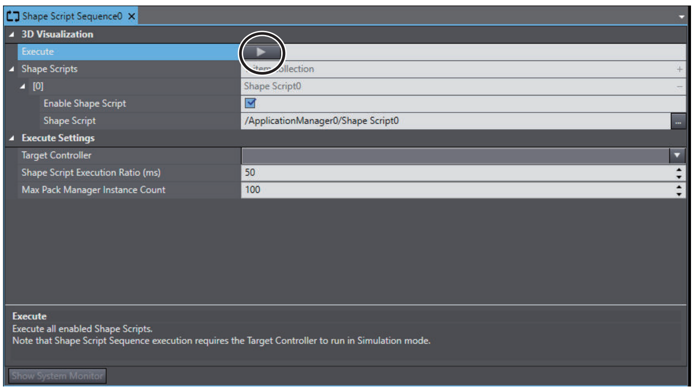
You can check the execution time of a Shape Script in the System Monitor. The following describes the procedure to check the execution time of a Shape Script in the System Monitor.

**1** Double-click the Shape Script Sequence for which to check the execution time under **Configurations and Setup** in the Multiview Explorer.



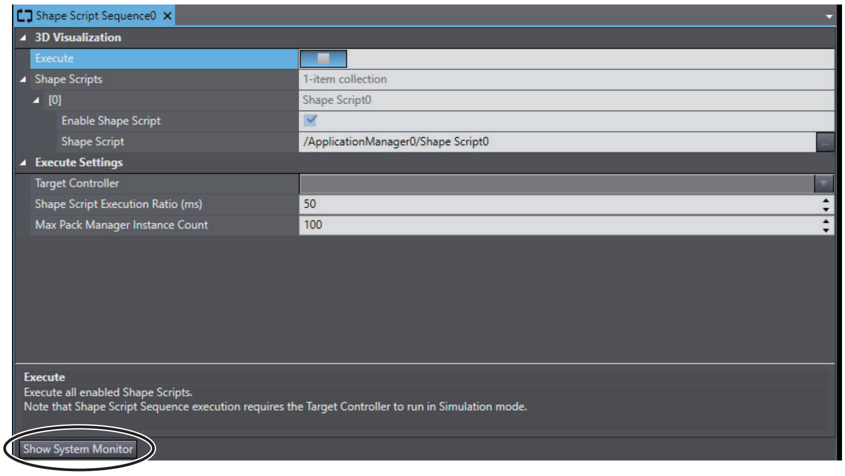

The Shape Script Sequence setup tab page is displayed.

**2** Click the **Execute** button for the Shape Script Sequence.

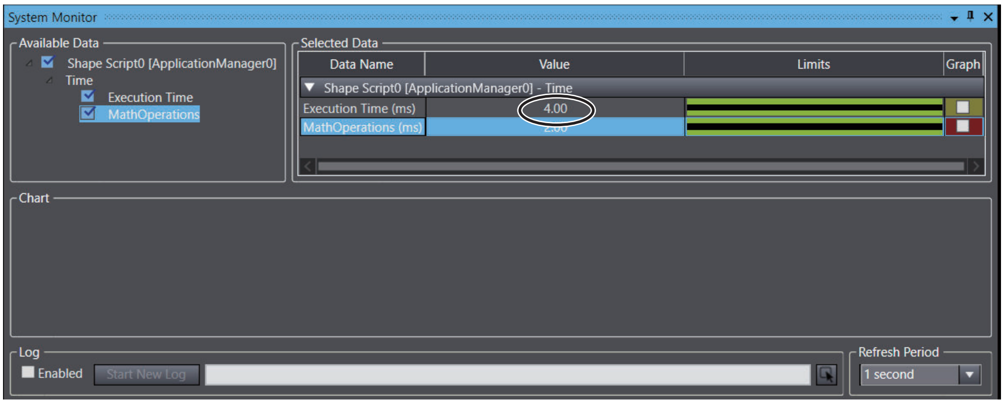



This executes the Shape Script Sequence.

**3** Click the **Show System Monitor** button for the Shape Script.




The System Monitor is displayed. You can now check the execution time of the Shape Script.




### Additional Information

Using Stopwatch constructs allows you to check execution time of specific sections in Shape Scripts.
Refer to *A-3-2 Measuring Execution Time with Stopwatch Constructs* on page A-71 for details on the operating procedure.

## A-3-2 Measuring Execution Time with Stopwatch Constructs

Writing a Stopwatch construct in a Shape Script allows you to measure the time that the computer spends executing a desired code section in the Shape Script. The measured time is displayed in the System Monitor.

Identifying and optimizing code sections that take long execution times allows you to reduce the execution time of the Shape Script.

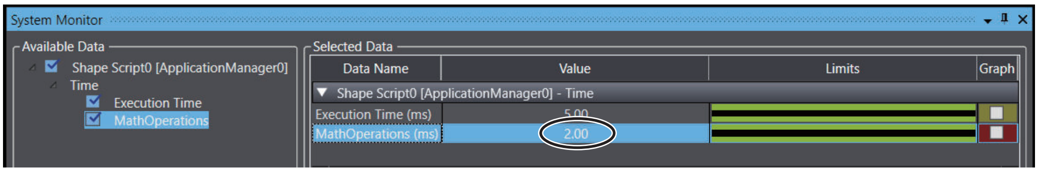
The procedure is given below with an example of a Shape Script that executes math operation.

**1** Enclose the desired code section in the Render() function in the Shape Script with **StartStopwatch("MathOperations")** and **StopStopwatch("MathOperations")**.
Enter a desired string in the *MathOperations*, which is the name to identify the section to measure. The name entered here is displayed in the System Monitor.

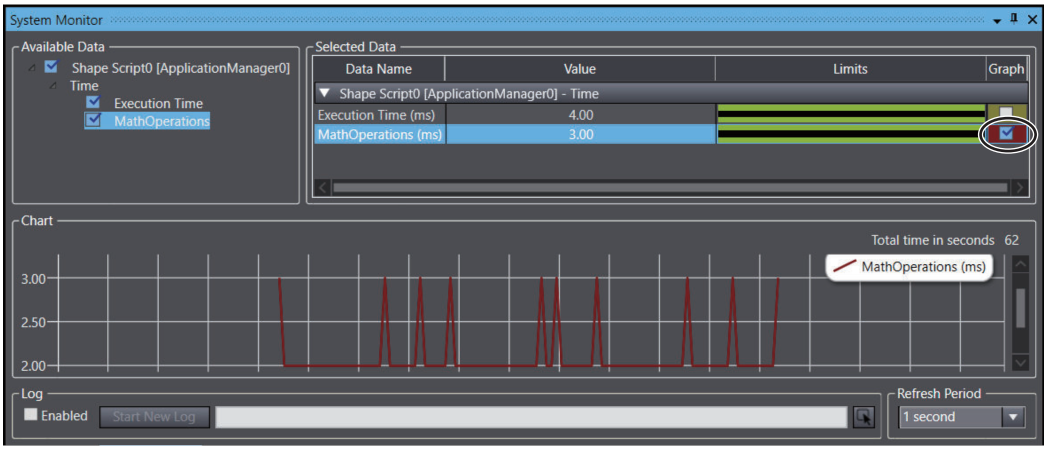
```
            StartStopwatch("MathOperations");
            // Move in circle
            Thread.Sleep(2);

            var x = renderInfo.Position.DX;
            var y = renderInfo.Position.DY;
            var z = renderInfo.Position.DZ;
            var r = Math.Sqrt(x*x + y*y);
            double phi;
            if (x == 0 && y == 0) phi = 0;
            else if (x >= 0) phi = Math.Atan(y / x);
            else phi = -Math.Asin(y / r) + Math.PI;

            phi += speed;
            x = r * Math.Cos(phi);
            y = r * Math.Sin(phi);

            renderInfo.Position = new Transform3D(x, y, z);
            StopStopwatch("MathOperations");
        }
```

**2** Execute the target Shape Script.

**3** From **Shape Script Sequence**, click the **Show System Monitor** button.
The System Monitor is displayed, and the execution time of the target Shape Script code section is displayed in the *MathOperations* value.




Selecting the check box of *MathOperations* in the **Graph** column displays the execution time of the *MathOperations* for each cycle in the graph.




Refer to *StartStopwatch* on page A-64 and *StopStopwatch* on page A-65 for details on the StartStopwatch() function and StopStopwatch() function.

## A-3-3 Speeding Up the Processing of Shape Scripts

Reading/writing values from/to Controller variables is one of the time-consuming operations of script processing. When a Shape Script reads/writes values from/to Controller variables, the reading or writing of the values is executed for the Simulator of the Controller through the internal network of the computer, which requires a long time to process network communications.

If a Shape Script contains multiple variables to read or write in the Render() function, reading or writing them one by one results in communications as many as the number of times of processing. On the other hand, reading or writing multiple variables together at once results in a reduced number of communications, and thus the execution time of the Shape Script can be reduced.
This section describes how to create Shape Scripts for reading and writing multiple variables at once.

## Reading Multiple Controller Variables

First, create a new read variable list to read multiple Controller variables at once. To create a read variable list, use the CreateGetVariableList() and AddToGetVariableList() functions.
Next, use the read variable list to read multiple Controller variables at once. To read multiple Controller variables at once, use the GetVariableValues() function. The return values of the GetVariableValues() function are the variables and their values you want to get.
To get a specific variable value from the variables and values, use the GetValueFromVariableValueList() function. The return value of the GetValueFromVariableValueList() function is the variable value to get.

The following is an example of a Shape Script that reads multiple Controller variables at once.

```
62      public override void Render(IEnumerable<ShapeRenderInfo> renderInfoList) {
63          if (this.IsInitialized == false) {
64              DefaultFunctions.InitializeObject(this, renderInfoList);
65              return ;
66          }
67
68          var getVariableList = DefaultFunctions.CreateGetVariableList();
69          DefaultFunctions.AddToGetVariableList("boolVar1", "BOOL", getVariableList);
70          DefaultFunctions.AddToGetVariableList("boolVar2", "BOOL", getVariableList);
71          DefaultFunctions.AddToGetVariableList("boolVar3", "BOOL", getVariableList);
72
73          var variableValueList = DefaultFunctions.GetVariableValues(this, "new_Controller_0", getVariableList);
74          var result = (bool)DefaultFunctions.GetValueFromVariableValueList("boolVar1", variableValueList);
75      }
```

In this example, the Shape Script reads the BOOL variables *boolVar1*, *boolVar2*, and *boolVar3* at once from the Controller *new_Controller_0*, and then gets the value of *boolVar1*.
In line 68, use the CreateGetVariableList() function to create a new read variable list.
Next, in lines 69 to 71, use the AddToGetVariableList() function to add the variables to read to the read variable list.
Then, in line 73, use the GetVariableValues() function to get the variable list that you read from the Controller. In line 74, get only the value of the variable to read from the variable value list.

**Version Information**

Reading from multiple variables is not available in Sysmac Studio version 1.40.

Some functions used in Shape Scripts read Controller variables from inside the functions. Even when these functions are used, reading multiple Controller variables at once can reduce the execution time of the Shape Script.
The following is a list of functions that read Controller variables.

| List of functions that read Controller variables |
|---|
| LoadPart |
| LoadPallet |
| LoadPartOnPallet |
| UnloadPart |

| List of functions that read Controller variables |
|---|
| UnloadPallet |
| SetNextStep |
| PushPart |
| PushPallet |
| MoveObjectOnBelt |
| ClampPartBySignal |
| ClampPalletBySignal |
| ClampPartOnPalletBySignal |
| ReleasePartBySignal |
| ReleasePalletBySignal |
| ReleasePartOnPalletBySignal |

For all of these functions, you can set the argument *variableValues*. You can omit the communications processing when the Shape Script reads Controller variables by setting the return values of the GetVariableValues() function in *variableValues*. Omitting the communications processing allows for reducing the execution time of the Shape Script.

The following is an example of a Shape Script that sets the return values of the GetVariableValues() function as function arguments.

```
62      public override void Render(IEnumerable<ShapeRenderInfo> renderInfoList) {
63          if (this.IsInitialized == false) {
64              DefaultFunctions.InitializeObject(this, renderInfoList);
65              return ;
66          }
67
68          var getVariableList = DefaultFunctions.CreateGetVariableList();
69          DefaultFunctions.AddToGetVariableList("boolVar1", "BOOL", getVariableList);
70          DefaultFunctions.AddToGetVariableList("boolVar2", "BOOL", getVariableList);
71          DefaultFunctions.AddToGetVariableList("boolVar3", "BOOL", getVariableList);
72
73          var variableValues = DefaultFunctions.GetVariableValues(this, "new_Controller_0", getVariableList);
74
75          ICadData partModel1 = (ICadData) ace["/ApplicationManager0/partModel1"];
76          ICadData partModel2 = (ICadData) ace["/ApplicationManager0/partModel2"];
77          ICadData table = (ICadData) ace["/ApplicationManager0/table"];
78
79          DefaultFunctions.LoadPart(this, "new_Controller_0", "boolVar1", partModel1, string.Empty, new Transform3D(0, 0, 0), table, renderInfoList, variableValues);
80          DefaultFunctions.LoadPart(this, "new_Controller_0", "boolVar2", partModel2, string.Empty, new Transform3D(100, 0, 0), table, renderInfoList, variableValues);
81      }
```

In this example, the Shape Script reads the BOOL variables *boolVar1*, *boolVar2*, and *boolVar3* at once from the Controller *new_Controller_0*. The read results are set in *variableValues* in line 73.
In line 79, *variableValues* is set as an argument to the LoadPart() function. This allows the Shape Script to get the value of *boolVar1* from *variableValues* in the LoadPart() function.
Similarly, in line 80, the Shape Script gets *boolVar2* from *variableValues* in the LoadPart() function.
Thus, you can get the values of Controller variables in each LoadPart() function quickly without using the network.

**Version Information**

The argument *variableValues* is not available in Sysmac Studio version 1.42 or lower.

## Writing Values to Multiple Controller Variables

First, create a write variable list to write values to multiple Controller variables at once. To create a write variable list, use the CreateSetVariableList() and AddToSetVariableList() functions.
Next, use the write variable list to write values to multiple Controller variables at once. To write values to multiple Controller variables together, use the SetVariableValues() function.

The following is an example of a Shape Script that writes values to multiple Controller variables at once.

```
62      public override void Render(IEnumerable<ShapeRenderInfo> renderInfoList) {
63          if (this.IsInitialized == false) {
64              DefaultFunctions.InitializeObject(this, renderInfoList);
65              return ;
66          }
67
68          var setVariableList = DefaultFunctions.CreateSetVariableList();
69          DefaultFunctions.AddToSetVariableList("intVar1", "INT", 1, setVariableList);
70          DefaultFunctions.AddToSetVariableList("intVar2", "INT", 2, setVariableList);
71          DefaultFunctions.AddToSetVariableList("intVar3", "INT", 3, setVariableList);
72
73          DefaultFunctions.SetVariableValues(this, "new_Controller_0", setVariableList);
74      }
```

In this example, the Shape Script writes *1*, *2*, and *3* to the INT variables *intVar1*, *intVar2*, and *intVar3* of the Controller *new_Controller_0*, respectively.
In line 68, use the CreateSetVariableList() function to create a new write variable list.
Next, in lines 69 to 71, use the AddToSetVariableList() function to add the values to write to the write variable list.
Then, in line 73, use the SetVariableValues() function to write the values to the Controller variables.

**Version Information**

Writing values to multiple variables is not available in Sysmac Studio version 1.40.

Some functions used in Shape Scripts write values to Controller variables from inside the functions. Even when these functions are used, writing values to multiple Controller variables at once can reduce the execution time of the Shape Script.
The following is a list of functions that write values to Controller variables.

| List of functions that write values to Controller variables |
|---|
| SetClampStatus |
| DetectPartCollision |
| DetectMechanicalComponentsCollision |

Setting the return value of the CreateSetVariableList() function as the argument *setVariableList* to these functions adds the Controller variable names and values to *setVariableList*. You can set the *setVariableList* as an argument to the SetVariableValues() function at the end of processing to write the values to the Controller variables at once. Thus, you can omit the communications processing when the Shape Script writes values to Controller variables. Omitting the communications processing allows for reducing the processing time of the Shape Script.

The following is an example of a Shape Script that sets the return values of the CreateSetVariableList() function as function arguments and writes the values to Controller variables at once.

```
110 public override void Render(IEnumerable<ShapeRenderInfo> renderInfoList) {
111     if (this.IsInitialized == false) {
112         DefaultFunctions.InitializeObject(this, renderInfoList);
113         return ;
114     }
115
116     var getVariableList = DefaultFunctions.CreateGetVariableList();
117     DefaultFunctions.AddToGetVariableList("boolVar1", "BOOL", getVariableList);
118     DefaultFunctions.AddToGetVariableList("boolVar2", "BOOL", getVariableList);
119     DefaultFunctions.AddToGetVariableList("boolVar3", "BOOL", getVariableList);
120
121     var variableValues = DefaultFunctions.GetVariableValues(this, "new_Controller_0", getVariableList);
122
123     var setVariableList = DefaultFunctions.CreateSetVariableList();
124
125     ICadData partModel1 = (ICadData) ace["/ApplicationManager0/partModel1"];
126     ICadData partModel2 = (ICadData) ace["/ApplicationManager0/partModel2"];
127     ICadData table = (ICadData) ace["/ApplicationManager0/table"];
128     MechanicalDataModel mechanicalDataModel = (MechanicalDataModel) ace["/ApplicationManager0/MechanicalModel"];
129
130     DefaultFunctions.LoadPart(this, "new_Controller_0", "boolVar1", partModel1, string.Empty, new Transform3D(0, 0, 0), table, renderInfoList, variableValues);
131     DefaultFunctions.LoadPart(this, "new_Controller_0", "boolVar2", partModel2, string.Empty, new Transform3D(100, 0, 0), table, renderInfoList, variableValues);
132
133     DefaultFunctions.DetectPartCollision(this, partModel1, mechanicalDataModel, 1, "new_Controller_0", "boolVar4", renderInfoList, setVariableList);
134     DefaultFunctions.DetectPartCollision(this, partModel2, mechanicalDataModel, 1, "new_Controller_0", "boolVar5", renderInfoList, setVariableList);
135
136     if(setVariableList.Count > 0){
137         DefaultFunctions.SetVariableValues(this, "new_Controller_0", setVariableList);
138     }
139 }
```

In this example, the CreateSetVariableList() function in line 123 creates the write variable list *setVariableList*. *setVariableList* is set as arguments to the DetectPartCollision() functions in lines 133 and 134. In each DetectPartCollision() function, the values to set in the Controller variables *boolVar4* and *boolVar5* are added to *setVariableList*.

In line 136, the Shape Script checks if the values are set in *setVariableList*. If the values are set, in line 137, the Shape Script writes the values to the Controller variable at once. Thus, you can write the values to Controller variables in each DetectPartCollision() function quickly without using the network.

**Version Information**

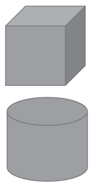
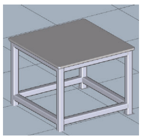
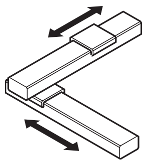
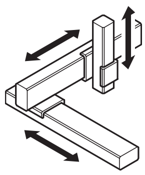
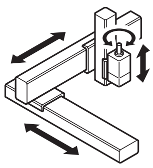
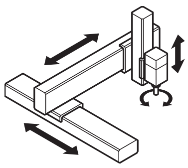
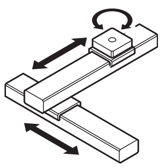
The argument *setVariableList* is not available in Sysmac Studio version 1.43 or lower.

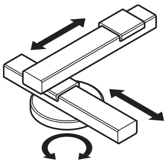
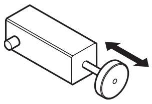
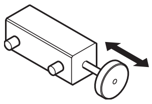
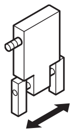
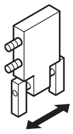
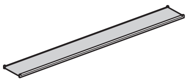
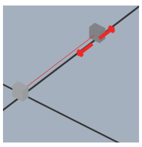
# A-4 Behaviors Settings

## A-4-1 Availability of Behaviors Settings for 3D Shape Data

The availability of Behaviors Settings depends on the 3D shape data. The table below provides the details.

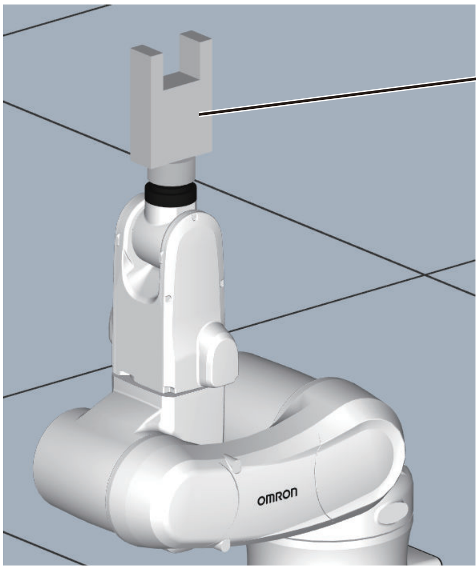
○: Available, ×: Not available

| Category | Image | Components | Behaviors Settings | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | Load-er | Con-veyor | Clamp | Push | Sen-sor | Phys-ics | Un-loader |
| Basic shape |  | Box and cylinder | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| CAD data |  | CAD data | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| Mechanical component |  | Orthogonal robot (XY) | × | × | ○ | ○ | ○ | ○ | × |
| |  | Orthogonal robot (XYZ) | × | × | ○ | ○ | ○ | ○ | × |
| |  | X-Y-Z stage + rotation axis (upward direction) | × | × | ○ | ○ | ○ | ○ | × |
| |  | X-Y-Z stage + rotation axis (downward direc-tion) | × | × | ○ | ○ | ○ | ○ | × |
| |  | X-Y table (XY Theta) | × | × | ○ | ○ | ○ | ○ | × |

| Category | Image | Components | Behaviors Settings | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | Load-er | Con-veyor | Clamp | Push | Sen-sor | Phys-ics | Un-loader |
| |  | X-Y table (Theta XY) | × | × | ○ | ○ | ○ | ○ | × |
| |  | Air cylinder (Single solenoid type) | × | × | ○ | ○ | ○ | ○ | × |
| |  | Air cylinder (Double solenoid type) | × | × | ○ | ○ | ○ | ○ | × |
| |  | Robot tool (Parallel switching 2-finger type chuck/single solenoid type) | × | × | ○ | ○ | ○ | ○ | × |
| |  | Robot tool (Parallel switching 2-finger type chuck/double solenoid type) | × | × | ○ | ○ | ○ | ○ | × |
| |  | Conveyor | × | ○ | × | × | ○ | ○ | × |
| Sensor |  | Virtual Part Detection Sensor | × | × | × | × | ○ | ○ | ○ |

**Additional Information**

Although robots and custom mechanics do not support Behaviors Settings, you can set the desired operations by setting 3D shape data (e.g., robot hand) at the point of contact with the part and then configuring the Behaviors Settings for the 3D shape data.




Set *Clamp* in the Behaviors Settings for the robot hand.

## A-4-2 How to Use Shape Scripts and Behaviors Settings Simultaneously

You can use Shape Scripts to control parts generated by *Loader* Behaviors Settings. It is also possible to use the Behaviors Settings to control parts generated by Shape Scripts. The following describes how to control parts using these methods.

### How to Use Shape Scripts to Control Parts Shown by *Loader* Behaviors Settings

You can control the motion of parts that are loaded based on the *Loader* Behaviors Settings and displayed in the 3D Visualizer in a Shape Script. To do so, cast the objects in world.Objects, which means all objects in the 3D Visualizer, to the RuntimePart data type in the Shape Script, extract the required objects using the GetComponent("Hierarchy") method, and update their properties.
The example below shows code in the Render function of a Shape Script, which moves the parts loaded by the Behaviors Settings by 1 mm in the Z direction.

```
using Ace.Server.Core.Sim3d.PartDetection;
using Ace.Server.Core.Sim3d.MechanicalComponent;
using Ace.Server.Core.Sim3d.Behaviors.Runtime;
using Ace.Visualization.Hierarchies;
using Ace.Server.Xpert.PackXpert;
```

Declare the use of Ace.Server.Core.Sim3d.Behaviors.Runtime and Ace.Visualization.Hierarchies in the Using statement. (Not required when registering a new Shape Script)

```
    public override void Render(IEnumerable<ShapeRenderInfo> renderInfoList) {
        if (this.IsInitialized == false) {
            DefaultFunctions.InitializeObject(this, renderInfoList);

            foreach(var obj in world.Objects){
                var runtimeObject = obj as RuntimePart;
                if (runtimeObject == null) {
                continue;
                }

                var hierarchyComponent = runtimeObject.GetComponent("Hierarchy") as HierarchyComponent;
                if (hierarchyComponent == null) {
                continue;
                }

                hierarchyComponent.OffsetFromParent = hierarchyComponent.OffsetFromParent.Shift(0, 0, 1);
                }

            return;
        }
    }
```

Search for RuntimePart-type objects in world.Object and assign them to runtimeObject.

Move the object hierarchyComponent corresponding to the part identified in the above processing by 1mm in the Z direction.

Get “Hierarchy”, which is an attribute of a part object in RuntimeObject, as a key and assign it to hierarchyComponent.

## How to Use Behaviors Settings to Control Parts Generated by Shape Scripts

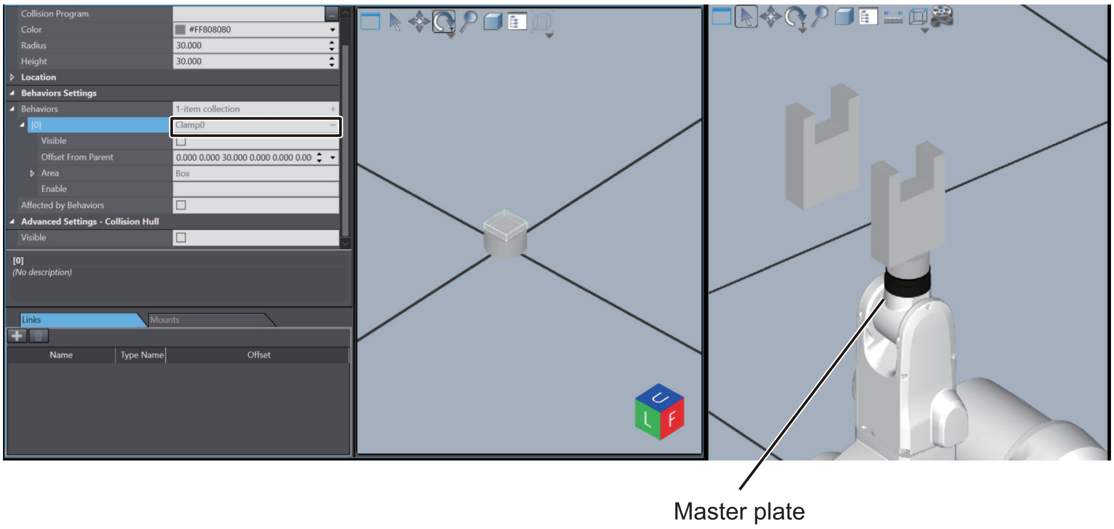
You can use Behaviors Settings to control parts loaded by a Shape Script by calling the EnableCanBeAffectedByBehaviors() method. The following example shows code in the CreateRenderInfo() function of the Shape Script that assigns a part to a variable named renderInfo0 from renderInfoList and calls the EnableCanBeAffectedByBehaviors() method in registering a cylinder named Part in the Multiview Explorer as a part in the renderInfoList to allow Part to be controlled by Behaviors Settings. You can also call the DisableCanBeAffectedByBehaviors() method to disable the control by Behaviors Settings. By default, the control by Behaviors Settings is disabled.

```
    public override IEnumerable<ShapeRenderInfo> CreateRenderInfo() {
        this.IsInitialized = false;

        // Create the Shapes for rendering once in the constructor
        List<ShapeRenderInfo> renderInfoList = new List<ShapeRenderInfo>();

        // Please assign your shape.
        ICylinder Part = (ICylinder) ace["/ApplicationManager0/Part"];
        if (Part != null) {
            DefaultFunctions.CreateObject(Part, "CollisionSource", "Group1", renderInfoList);

            var renderInfo0 = renderInfoList[0];
            renderInfo0.EnableCanBeAffectedByBehaviors();
        }

        return renderInfoList;
    }
```

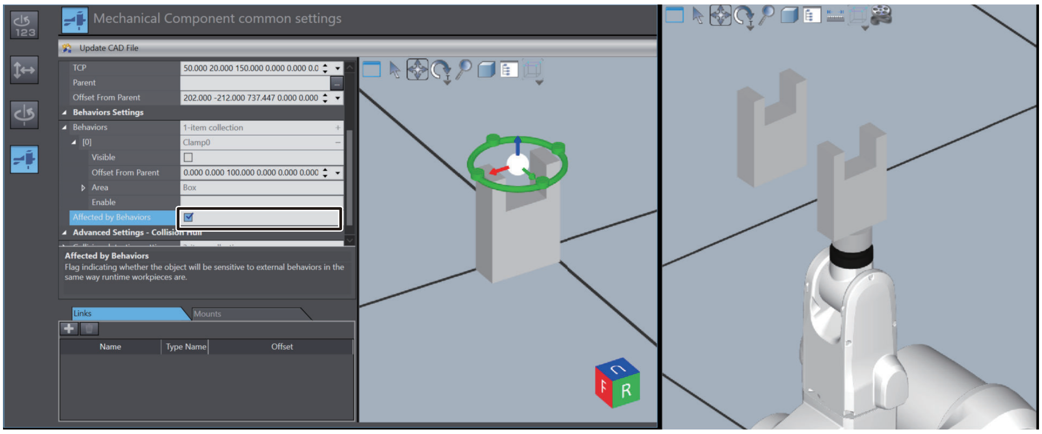
# A-4-3 How to Represent a Robot Hand Tool Changer

You can use Behaviors Settings to represent a robot hand tool changer. Use the following procedure.

1. Add the *Clamp* Behaviors Settings to the master plate on which to mount tools.




**2** Open the edit pane for the tool to mount and select the *Affected by Behaviors* check box.



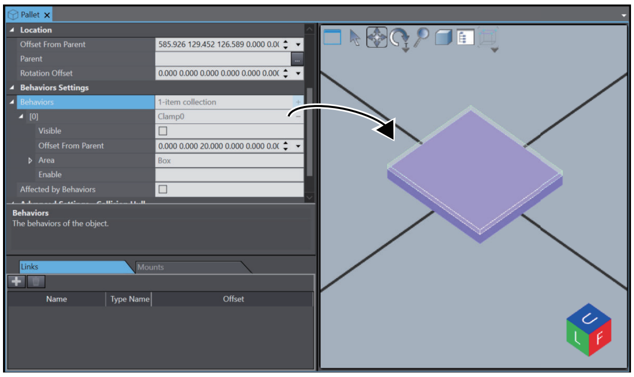
The tool will operate by following the master plate when it enters the area of the master plate set by the *Clamp* Behaviors Settings and the *Clamp* Behaviors Settings are enabled.

## A-4-4 How to Represent a Pallet

You can use Behaviors Settings to represent a pallet. Use the following procedure.

**1** Add CAD data that will become the palette in the Multiview Explorer.

**2** Add *Clamp* Behaviors Settings to the CAD data that you added in step 1.



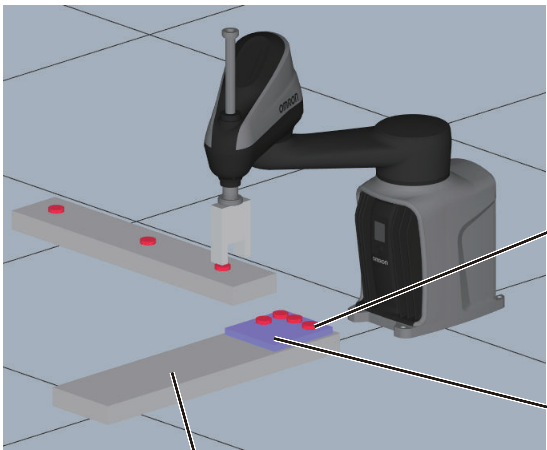

**3** Add *Loader* Behaviors Settings to 3D shape data that is different from that in step 1, and then configure the settings to generate the CAD data mentioned in step 1.

**4** Add *Loader* Behaviors Settings to generates parts.

**5** Configure other settings for picking parts, carrying them with the pallet, etc.

**6** Execute a simulation using the Behaviors Settings.

The *Loader* Behaviors Settings are available for the CAD data loaded by *Clamp* Behaviors Settings. Therefore, the pallet can *pick* parts while the parts are in the pallet's area of *Clamp* Behaviors Settings and the *Clamp* Behaviors Settings are enabled.
In addition, you can operate the pallet using *Conveyor* Behaviors Settings and *Clamp* Behaviors Settings since *Loader* Behaviors Settings have loaded the pallet. In this case, the parts *picked* by the pallet also follow the pallet.



The parts that come in contact with the pallet are "picked" by the pallet and follow the pallet's movement.

The pallet shown by the *Loader* Behaviors Settings uses the *Clamp* Behaviors Settings to pick the parts.

The 3D shape data carries the pallet based on the *Conveyor* Behaviors Settings.

I

# Index

I

# Index

**OMRON Corporation** **Industrial Automation Company**
**Kyoto, JAPAN** **Contact : www.ia.omron.com**

***Regional Headquarters***

**OMRON EUROPE B.V.**
Wegalaan 67-69, 2132 JD Hoofddorp
The Netherlands
Tel: (31) 2356-81-300 Fax: (31) 2356-81-388

**OMRON ELECTRONICS LLC**
2895 Greenspoint Parkway, Suite 200
Hoffman Estates, IL 60169 U.S.A.
Tel: (1) 847-843-7900 Fax: (1) 847-843-7787

**OMRON ASIA PACIFIC PTE. LTD.**
438B Alexandra Road, #08-01/02 Alexandra
Technopark, Singapore 119968
Tel: (65) 6835-3011 Fax: (65) 6835-3011

**OMRON (CHINA) CO., LTD.**
Room 2211, Bank of China Tower,
200 Yin Cheng Zhong Road,
PuDong New Area, Shanghai, 200120, China
Tel: (86) 21-6023-0333 Fax: (86) 21-5037-2388

**Authorized Distributor:**



**Cat. No. W618-E1-11** 0425