

Automation Software

Sysmac Studio

3D Simulation Function Operation Manual

SYSMAC-SE2□□□

SYSMAC-SA4□□L-64



NOTE

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, mechanical, electronic, photocopying, recording, or otherwise, without the prior written permission of OMRON.

No patent liability is assumed with respect to the use of the information contained herein. Moreover, because OMRON is constantly striving to improve its high-quality products, the information contained in this manual is subject to change without notice. Every precaution has been taken in the preparation of this manual. Nevertheless, OMRON assumes no responsibility for errors or omissions. Neither is any liability assumed for damages resulting from the use of the information contained in this publication.

Trademarks

- Sysmac and SYSMAC are trademarks or registered trademarks of OMRON Corporation in Japan and other countries for OMRON factory automation products.
- Microsoft, Windows, Excel, and Visual Basic are either registered trademarks or trademarks of Microsoft Corporation in the United States and other countries.
- EtherCAT® is registered trademark and patented technology, licensed by Beckhoff Automation GmbH, Germany.
- ODVA, CIP, CompoNet, DeviceNet, and EtherNet/IP are trademarks of ODVA.
- The SD and SDHC logos are trademarks of SD-3C, LLC.  
- NVIDIA, the NVIDIA logo, GeForce, and the GeForce logo are the trademarks or registered trademarks of NVIDIA Corporation in the USA and other countries.
- ATI™ and Radeon™ are the trademarks of Advanced Micro Devices, Inc.
- Celeron, Intel and Intel Core are trademarks of Intel Corporation in the U.S. and / or other countries.
- Git and the Git logo are either registered trademarks or trademarks of Software Freedom Conservancy, Inc., corporate home of the Git Project, in the United States and/or other countries.

Other company names and product names in this document are the trademarks or registered trademarks of their respective companies.

Copyrights

Microsoft product screen shots reprinted with permission from Microsoft Corporation.

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

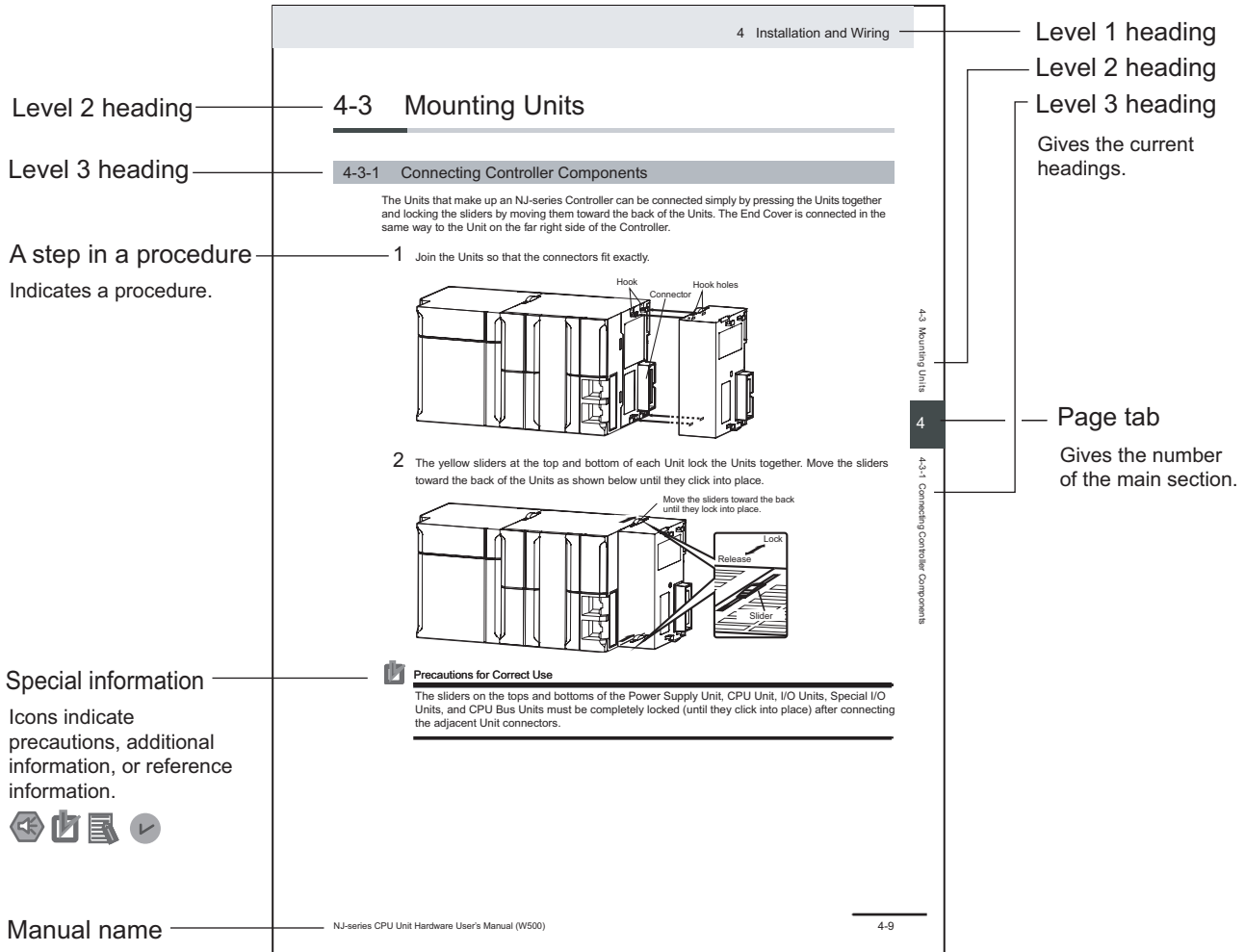
Part of the specifications and restrictions for the products are given in other manuals.

Refer to *Related Manuals* on page 17.

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

1	Features and Specifications of the 3D Simulation Function	A	Appendices	1	A
2	Software Setup	I	Index	2	I
3	3D Simulation Procedure			3	
4	Creating 3D Shape Data			4	
5	Working with the 3D Visualizer and 3D Editing Area			5	
6	Creating Settings and Scripts for Operating the 3D Shape Data			6	
7	Executing a 3D Simulation			7	
8	3D Simulation of Robot Integrated Systems			8	
9	Useful Functions			9	

CONTENTS

Introduction	1
Intended Audience	1
Applicable Products	1
Manual Structure.....	2
Page Structure	2
Special Information	2
Precaution on Terminology	3
Terms and Conditions Agreement.....	10
Safety Precautions.....	12
Precautions for Safe Use	13
Precautions for Correct Use	14
Regulations and Standards	15
Software Licenses and Copyrights	15
Versions	16
Related Manuals.....	17
Terminology.....	18
Revision History.....	20
Sections in this Manual	5

Section 1 Features and Specifications of the 3D Simulation Function

1-1 Features of the 3D Simulation	1-2
1-2 Specifications.....	1-3
1-2-1 Product Model Numbers	1-3
1-2-2 Supported Languages.....	1-3
1-2-3 Applicable Models	1-3
1-2-4 Applicable Computers	1-4

Section 2 Software Setup

2-1 Installing the Sysmac Studio	2-2
2-2 Registering Sysmac Studio Option License	2-3

Section 3 3D Simulation Procedure

3-1 Introduction to 3D Simulation.....	3-2
---	------------

3-2	Processes of 3D Simulation.....	3-4
-----	---------------------------------	-----

Section 4 Creating 3D Shape Data

4-1	Introduction to 3D Shape Data	4-2
4-1-1	Location of 3D Shape Data	4-2
4-1-2	Parent-child Relationship between 3D Shape Data	4-5
4-2	Adding Application Manager	4-6
4-3	Creating 3D Shape Data for the Mechanical Component.....	4-7
4-3-1	Types of Supported CAD Data Files	4-7
4-3-2	Preparations for CAD Data Files	4-7
4-3-3	CAD Data Import Procedure	4-10
4-3-4	Types of Mechanical Component Models	4-16
4-3-5	Mechanical Component Settings	4-24
4-4	Adding 3D Shape Data for the Part	4-29
4-4-1	Importing CAD Data	4-29
4-4-2	Adding a Box or a Cylinder	4-33
4-5	Adding the Part Detection Sensor.....	4-37
4-5-1	Adding the Virtual Part Detection Sensor.....	4-37
4-5-2	Virtual Part Detection Sensor Settings	4-37
4-6	3D Shape Data Placement Methods	4-40
4-6-1	Entering a Coordinate Directly	4-40
4-6-2	Dragging and Dropping the Data on the 3D Visualizer	4-40
4-6-3	Snapping to the Link Point	4-40

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

5-1	Displaying the 3D Visualizer and 3D Editing Area.....	5-2
5-1-1	Displaying the 3D Visualizer.....	5-2
5-1-2	Displaying the 3D Editing Area	5-3
5-2	Displayed Items in the 3D Visualizer and 3D Editing Area	5-5
5-2-1	Split Window	5-6
5-2-2	Selection and Edit	5-7
5-2-3	Translate	5-7
5-2-4	Rotate.....	5-8
5-2-5	Zoom	5-8
5-2-6	Projection Mode	5-9
5-2-7	Scene Graph	5-10
5-2-8	Measurement Ruler.....	5-12
5-2-9	Snap	5-13
5-2-10	Record.....	5-14
5-2-11	3D View Switching Tool.....	5-17
5-3	Operating 3D Shape Data in the 3D Visualizer.....	5-19
5-3-1	Moving 3D Shape Data	5-19
5-3-2	Rotating 3D Shape Data	5-20
5-3-3	Editing 3D Shape Data Simply.....	5-22
5-4	Positioning with a Mount Point or a Link Point	5-24
5-4-1	Outline of a Mount Point and a Link Point.....	5-24
5-4-2	Setting a Mount Point.....	5-26
5-4-3	Setting a Link Point	5-27
5-4-4	Offset Setting Methods.....	5-29
5-4-5	Using a Mount Point and a Link Point to Place 3D Shape Data	5-33

Section 6 Creating Settings and Scripts for Operating the 3D Shape Data

6-1	Outline of Settings and Scripts for Operating 3D Shape Data	6-2
6-1-1	Outline of a Shape Script	6-2
6-1-2	Execution Timing and Period of Shape Scripts and Programs	6-3
6-2	Setting Mechanical Component	6-7
6-2-1	Generating Virtual Output Scripts of Limit Switch	6-7
6-3	Creating Operation Scripts for the Part	6-9
6-3-1	Adding Shape Scripts	6-9
6-3-2	Setting the Execution of Shape Scripts	6-9
6-4	Configuring the Operation Settings for the Virtual Part Detection Sensor	6-13
6-5	Shape Script Editor	6-15
6-5-1	Shape Script Editor Window	6-15
6-5-2	Shape Script Programming	6-18

Section 7 Executing a 3D Simulation

7-1	Operating Procedures for a 3D Simulation	7-2
7-1-1	Executing a Controller Simulation	7-2
7-1-2	Executing the Operation Script for the Virtual Part Detection Sensor and the Virtual Output Script for the Mechanical Component	7-3
7-1-3	Executing the Shape Scripts for the Part	7-3
7-1-4	Checking Operations in the 3D Visualizer	7-4
7-1-5	Executing Operations for a 3D Simulation at Once	7-4
7-2	Debugging a Shape Script	7-7
7-2-1	Breakpoint	7-7
7-2-2	Step Execution	7-8
7-2-3	Trace Statement	7-9
7-2-4	Takt Time Measurement	7-9
7-3	Collision Detection Function	7-13
7-3-1	Collision Detection Target	7-13
7-3-2	Collision Detection Setting Procedure	7-13
7-3-3	How to Check Detected Collisions	7-16
7-3-4	How to Detect Collisions with Shape Scripts	7-17
7-3-5	Collision Filter Model Settings	7-24

Section 8 3D Simulation of Robot Integrated Systems

8-1	Outline of a 3D Simulation of Robot Systems with an Application Controller	8-2
8-1-1	Types of 3D Simulation	8-2
8-1-2	3D Simulation of Mechanical Components Controlled by IEC 61131-3 Languages	8-2
8-1-3	3D Simulation of Robot Applications	8-2
8-2	3D Simulation of Robot Systems	8-4
8-2-1	Using a Peripheral Device to Manipulate Parts That Were Manipulated by a Robot	8-4
8-2-2	Using a Robot to Manipulate Parts That Were Manipulated by a Peripheral Device	8-4
8-2-3	Detecting Collisions of Parts That Are Manipulated Only by a Robot	8-6

Section 9 Useful Functions

9-1	Updating All CAD Data	9-2
9-1-1	Procedure to Update All CAD Data	9-2

Appendices

A-1	Functions Used in Shape Scripts	A-2
A-1-1	Creating and Displaying Parts and Pallets	A-5
A-1-2	Conveying Parts and Pallets	A-13
A-1-3	Virtual Conveyor	A-15
A-1-4	Clamping Parts and Pallets	A-16
A-1-5	Placing Parts and Pallets	A-22
A-1-6	Changing and Detecting Status	A-27
A-1-7	Sharing Variables with the Controller	A-30
A-1-8	Cooperation with the Process Manager	A-39
A-2	Differences between the Simulator and the Physical Controller	A-43
A-2-1	Operation of Functions	A-43

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 3D Simulation Function Operation Manual	W618	SYSMAC-SE2□□□ 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 Drive Functions Operation Manual	I589	SYSMAC-SE2□□□ 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-□□□□ NX102-□□□□ NX1P2-□□□□ NJ501-□□□□ NJ301-□□□□ 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. <ul style="list-style-type: none"> • CPU Unit operation • CPU Unit features • Initial settings • Programming based on IEC 61131-3 language specifications
NA-series Programmable Terminal Software 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, a Virtual Part Detection Sensor, and a part, and the settings and scripts that define the operations of their 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.
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.
Part	This is the target object carried by a mechanical component 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. 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.
Shape Script	A program that defines the operations of a 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, or .iges 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,
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. The values that represent the position and pose (X, Y, Z, yaw, pitch, and roll) are called location elements. Refer to <i>Pose of 3D Shape Data</i> on page 4-3 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 <i>parent</i> and the 3D shape data for following the operation of the parent is called <i>child</i> . 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.

Term	Description
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, at which a tool such as the robot hand is mounted.

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-E1-03

↑ 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.

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	1-2
1-2	Specifications	1-3
1-2-1	Product Model Numbers.....	1-3
1-2-2	Supported Languages	1-3
1-2-3	Applicable Models	1-3
1-2-4	Applicable Computers	1-4

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 Intel® Core™ i7 9750H or equivalent or faster recommended
RAM	8 GB min. 16 GB min. recommended
Display	Full HD 1,920 × 1,080, 16 million colors min.
Video card	NVIDIA® GeForce® GTX1650 or higher

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	2-2
2-2	Registering Sysmac Studio Option License.....	2-3

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.

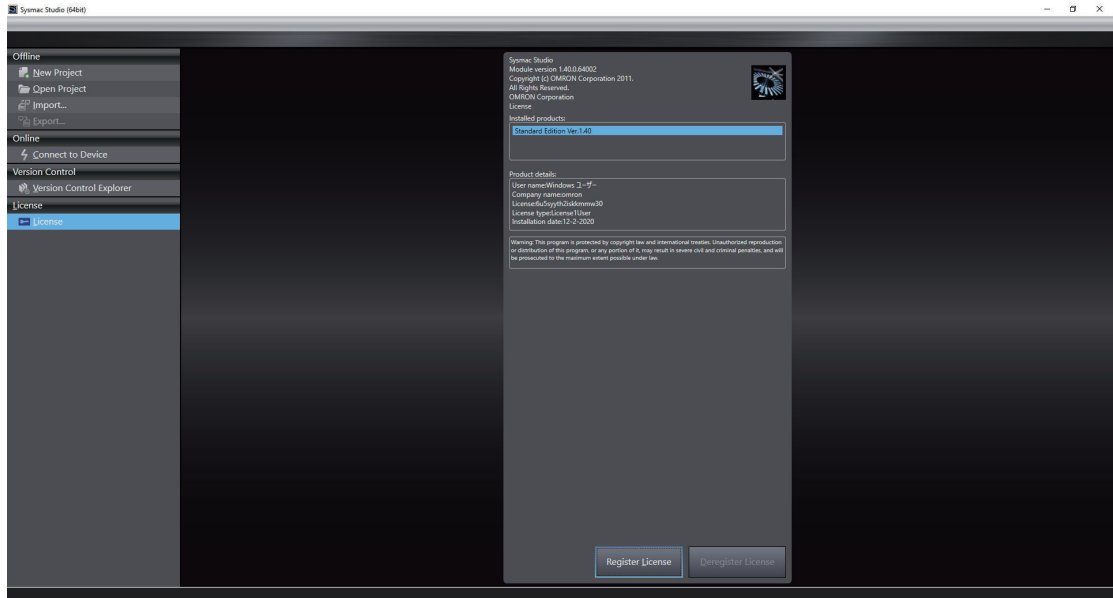
Use the following procedure to register an option license.

- 1 Select **All Programs - OMRON - Sysmac Studio - Sysmac Studio** from the Windows Start menu.

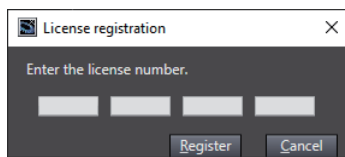
The Sysmac Studio starts and the start page is displayed.



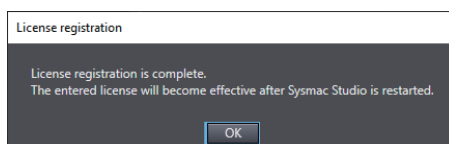
- 2 Click **License** on the start page.
The licenses that are currently registered are displayed.



- 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

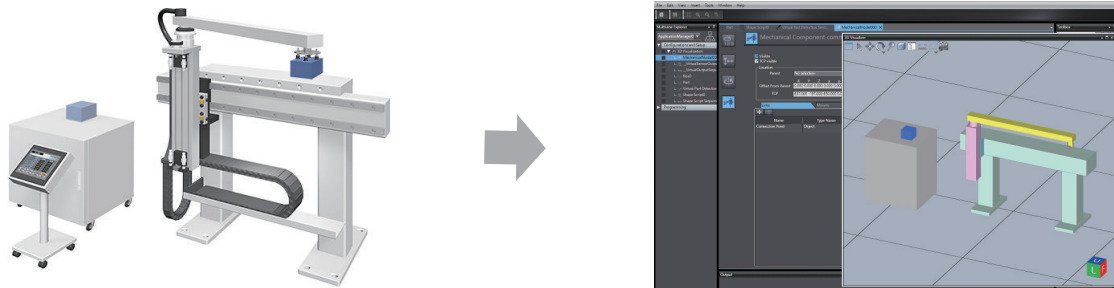
3D Simulation Procedure

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

3-1	Introduction to 3D Simulation	3-2
3-2	Processes of 3D Simulation	3-4

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.

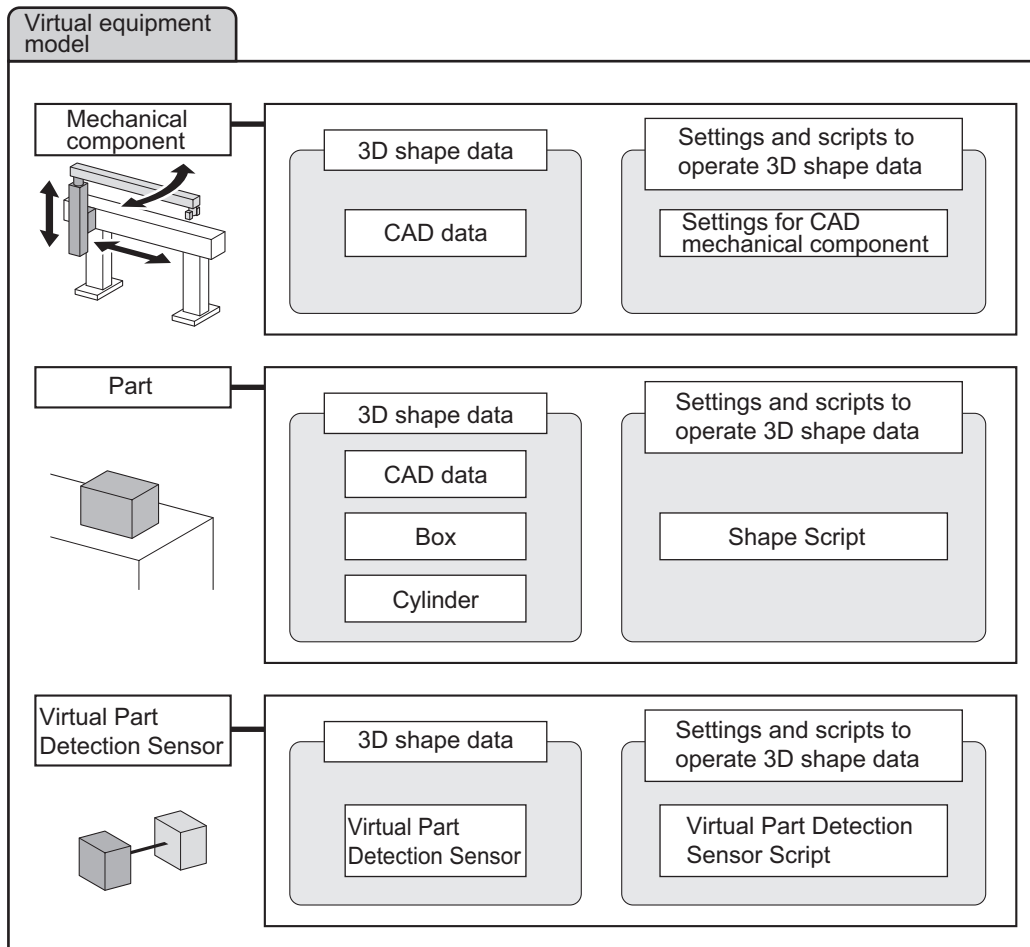


A virtual equipment model consists of a *mechanical component* that represents a movable part of the equipment, a *part* that represents the target to be carried, and a *Virtual Part Detection Sensor* that detects the part.

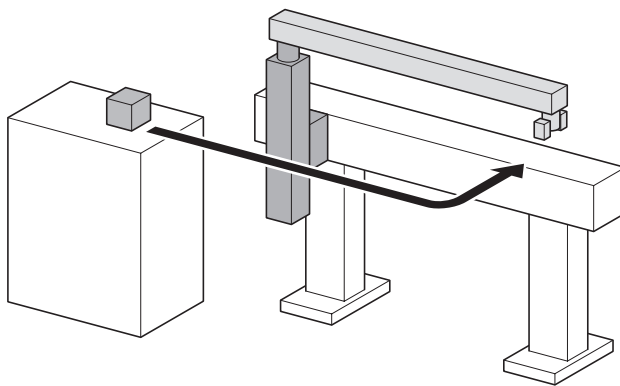
The mechanical component, part, and Virtual Part Detection Sensor are made up of 3D shape data and the settings and scripts for operating the 3D shape data.

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 to execute a 3D simulation in 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.




Process	Description	Reference
Creating a Project 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.	<i>Sysmac Studio Version 1 Operation Manual (Cat. No. W504)</i>



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.	<i>4-2 Adding Application Manager on page 4-6</i>
		
Adding the Mechanical Component	Load the CAD data for the target equipment and create 3D shape data for the mechanical component.	<i>4-3 Creating 3D Shape Data for the Mechanical Component on page 4-7</i>
		
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.	<i>4-4 Adding 3D Shape Data for the Part on page 4-29</i>
		
Adding the Part Detection Sensor	Add 3D shape data for the <i>Virtual Part Detection Sensor</i> , which is a virtual sensor to detect the part.	<i>4-5 Adding the Part Detection Sensor on page 4-37</i>



Configuring Settings and Creating Scripts to Operate 3D Shape Data	Configure settings and create scripts for operating the 3D shape data for the equipment.	---
--	--	-----

Process	Description	Reference
Setting the Mechanical Component	Configure the operation settings for the Mechanical Component.	6-2 <i>Setting Mechanical Component</i> on page 6-7
		
Creating Operation Scripts for the Part	Create scripts that define the operations of the part.	6-3 <i>Creating Operation Scripts for the Part</i> on page 6-9
		
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 <i>Configuring the Operation Settings for the Virtual Part Detection Sensor</i> on page 6-13
		
Executing a Simulation	Execute a 3D simulation to check how the virtual equipment model operates.	Section 7 <i>Executing a 3D Simulation</i> on page 7-1

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

4-1	Introduction to 3D Shape Data	4-2
4-1-1	Location of 3D Shape Data	4-2
4-1-2	Parent-child Relationship between 3D Shape Data	4-5
4-2	Adding Application Manager	4-6
4-3	Creating 3D Shape Data for the Mechanical Component	4-7
4-3-1	Types of Supported CAD Data Files	4-7
4-3-2	Preparations for CAD Data Files	4-7
4-3-3	CAD Data Import Procedure	4-10
4-3-4	Types of Mechanical Component Models	4-16
4-3-5	Mechanical Component Settings.....	4-24
4-4	Adding 3D Shape Data for the Part	4-29
4-4-1	Importing CAD Data	4-29
4-4-2	Adding a Box or a Cylinder.....	4-33
4-5	Adding the Part Detection Sensor	4-37
4-5-1	Adding the Virtual Part Detection Sensor	4-37
4-5-2	Virtual Part Detection Sensor Settings	4-37
4-6	3D Shape Data Placement Methods	4-40
4-6-1	Entering a Coordinate Directly	4-40
4-6-2	Dragging and Dropping the Data on the 3D Visualizer	4-40
4-6-3	Snapping to the Link Point	4-40

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

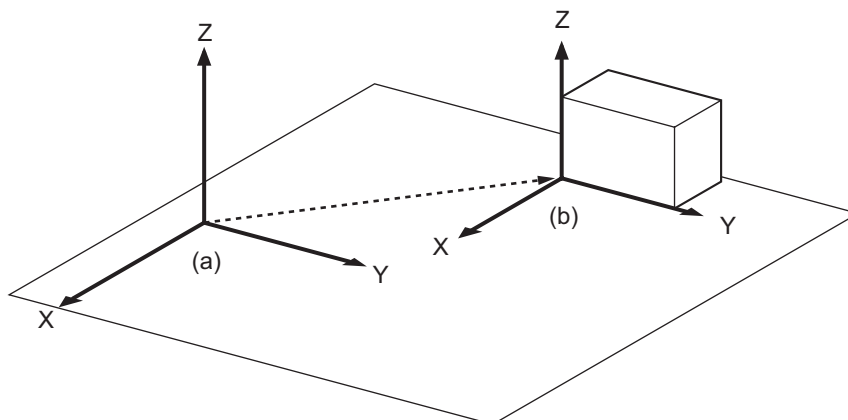
The position and pose of 3D shape data are represented by the location elements (X, Y, Z, yaw, pitch, and roll).

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.

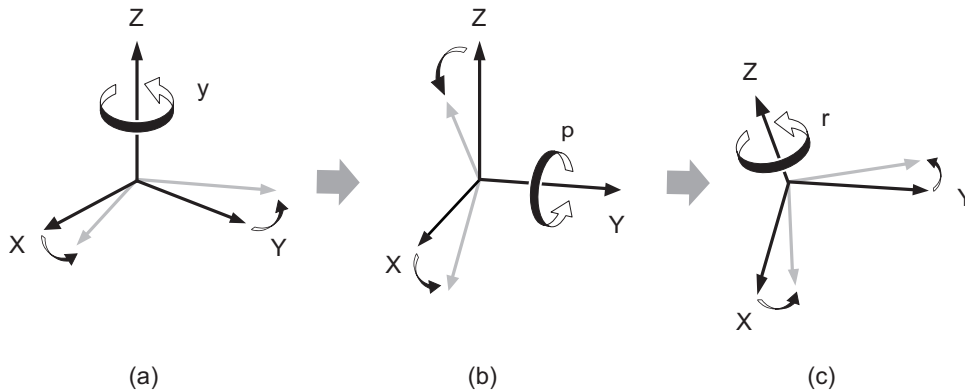


	Coordinate system	Description
(a)	World coordinate system	This coordinate system represents absolute positions on the 3D Visualizer.
(b)	Local coordinate system	Each 3D shape data has its local coordinate system. Use this coordinate system when you configure an offset to determine the rotational center 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.



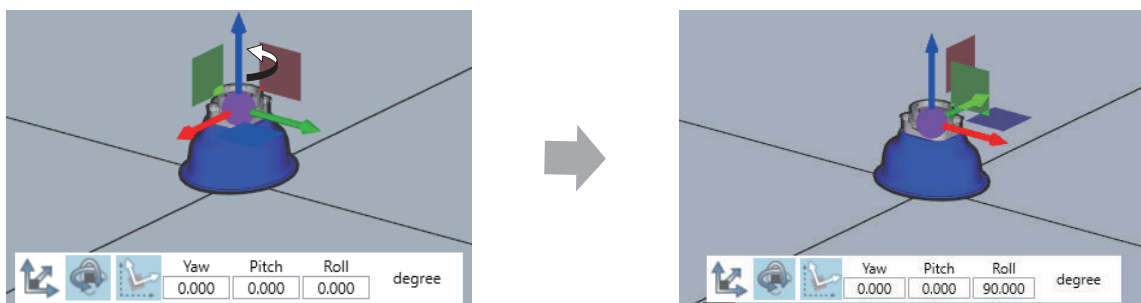
	Rotation	Description	Unit
(a)	y (<i>yaw</i>)	Represents the rotation angle around the Z axis of the local coordinate system.	° (degree)
(b)	p (<i>pitch</i>)	Represents the rotation angle around the Y axis of the local coordinate system based on the calculation of yaw.	° (degree)
(c)	r (<i>roll</i>)	Represents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

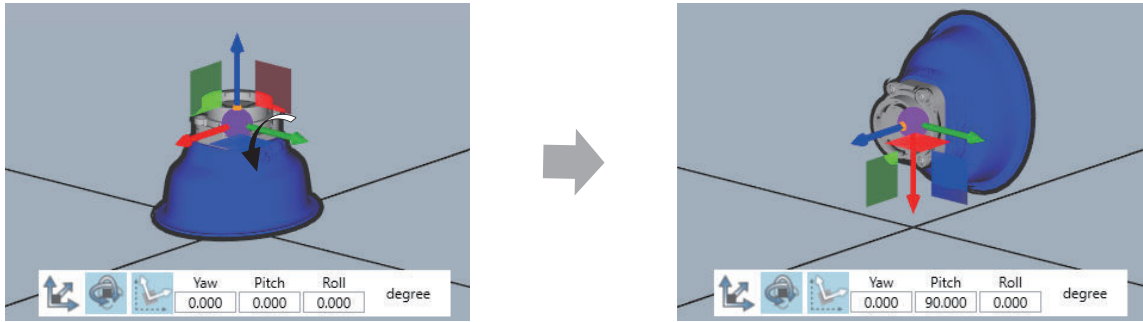
● Rotation around the Z Axis (Blue Arrow)

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)

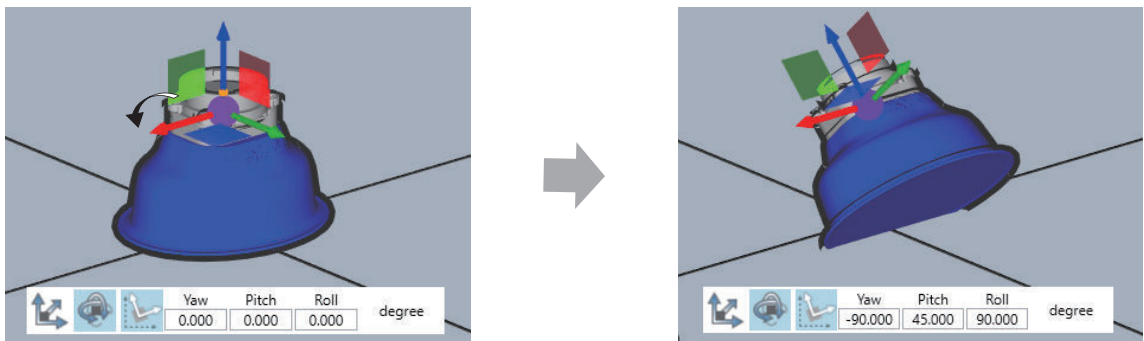
To rotate the 3D shape data 90 degrees around the Y axis, set the p (*pitch*) value to 90.



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

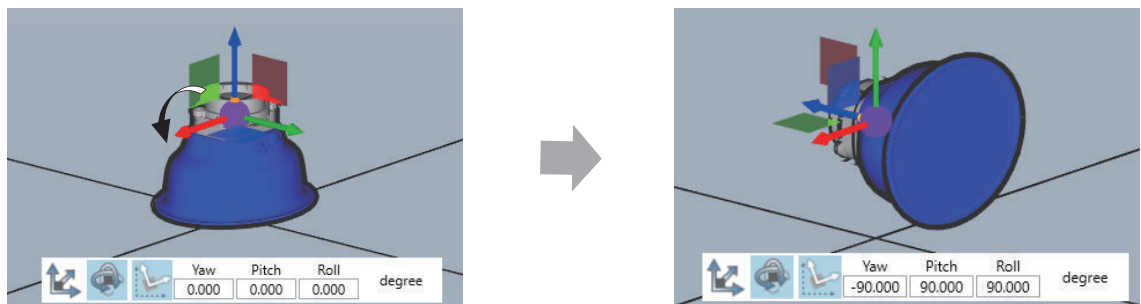
To rotate the 3D shape data around the X axis, unlike the Y and Z axes, you need to set values for two or more parameters.

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



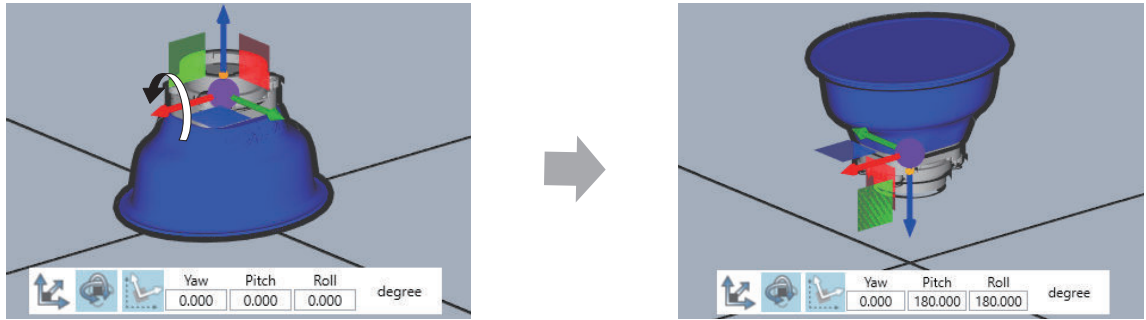
Set the *y* (*yaw*) value to -90, the *p* (*pitch*) value to 45, and the *r* (*roll*) value to 90.

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



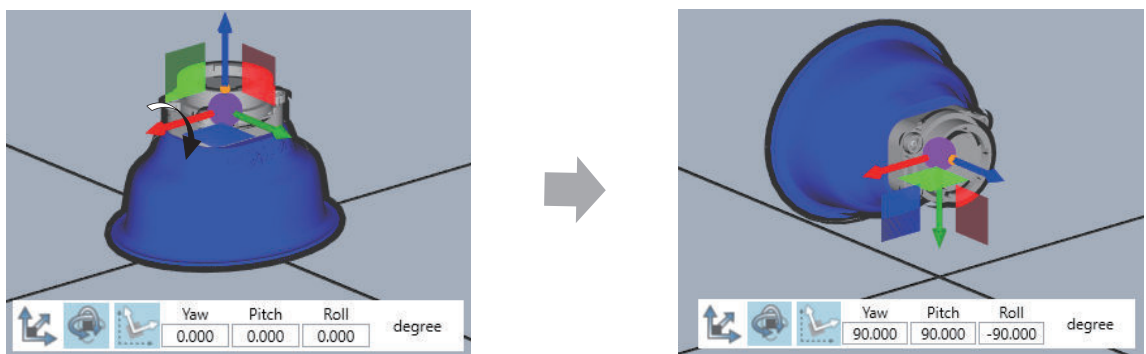
Set the *y* (*yaw*) value to -90, the *p* (*pitch*) value to 90, and the *r* (*roll*) value to 90.

- 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

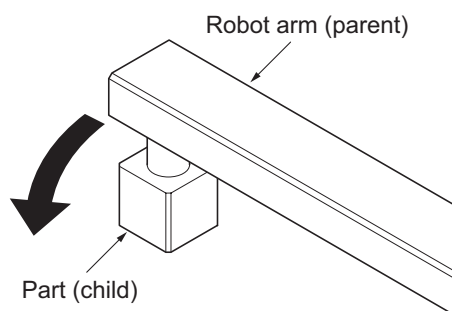


Set the y (*yaw*) value to 90, p (*pitch*) value to 90, and r (*roll*) value to -90.

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*.



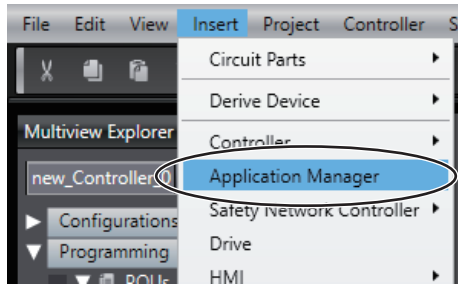
To set a parent-child relationship, select the parent 3D shape data on the setup tab page for the child 3D shape data.

4-2 Adding Application Manager

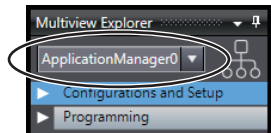
Application Manager is a logical device that manages the data and settings required to use the 3D Simulation Function.

To create 3D shape data, add Application Manager to the project according to the following procedure.

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



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



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.

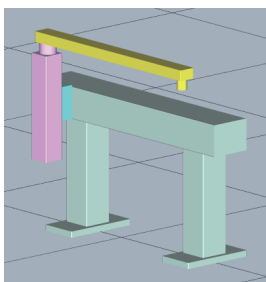
CAD data type	Supported version
STEP	AP203, AP214, AP242
IGES	Ver. 5.3 or later

4-3-2 Preparations for CAD Data Files

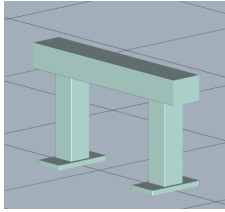
To import CAD data files for mechanical component parts, create CAD data files for the individual parts that make up the component depending on a mechanical component.

Refer to *4-3-4 Types of Mechanical Component Models* on page 4-16 for details on types of mechanical components supported by the 3D Simulation Function.

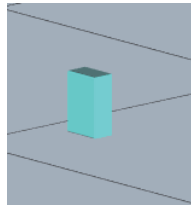
Example: Movable part as an X-Y table (XY Theta)



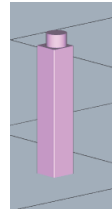
To import CAD data for movable part that consists of a base, an X-axis slider, a Z-axis slider, and an arm with a rotary axis as the mechanical component *X-Y table (XY Theta)*, create a CAD data file for each of the base, X-axis slider, Z-axis slider, and arm with a rotation axis. (In this example,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



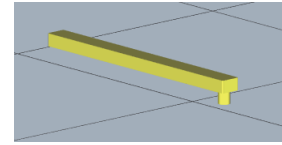
Base
(XYThetaTable_base.step)



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

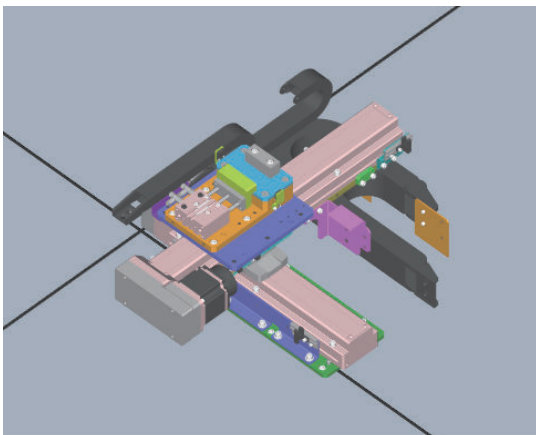
A mechanical component consists of a base that serves as a foundation and movable parts that operate according to outputs from an axes or I/O. To import a mechanical component to the 3D Visualizer, you need to prepare CAD data files for the base and movable parts that are output from 3D CAD software.



Additional Information

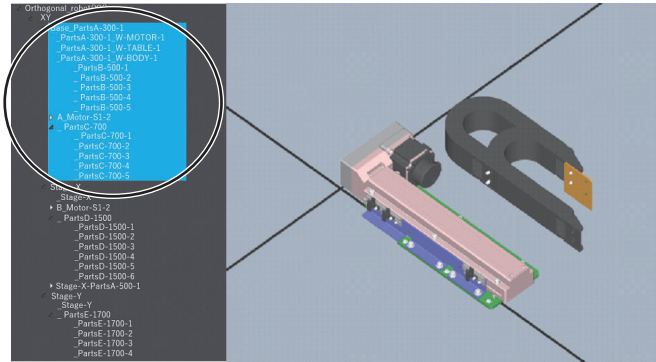
- If there are CAD data files that are created in the mechanical design stage, use the 3D CAD software to create CAD data files for the mechanical component based on the CAD data files. Then, create CAD data files for the base and movable parts.
- When you open a CAD data file in 3D CAD software, the 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*.

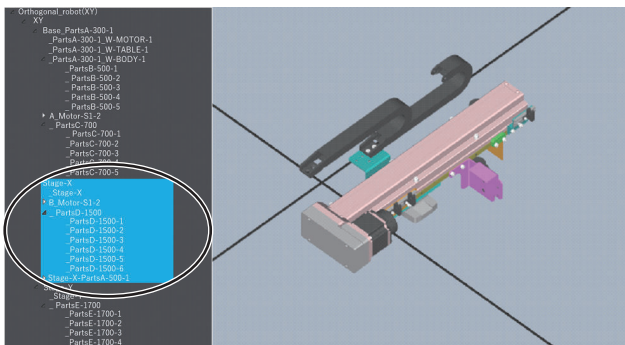


- 1 From the assembly information displayed in the 3D CAD software, select the assembly information that corresponds to the base and output only the selected information to a CAD data file.

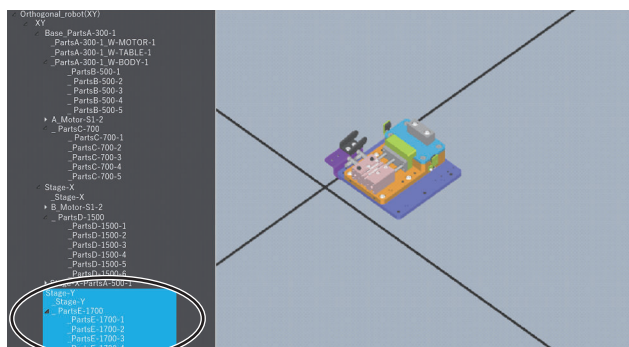
The following is an example of selecting assembly information to display only the selected assembly information in the 3D Visualizer.



- 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.



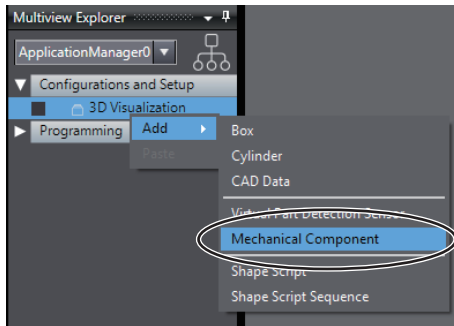
Precautions for Correct Use

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

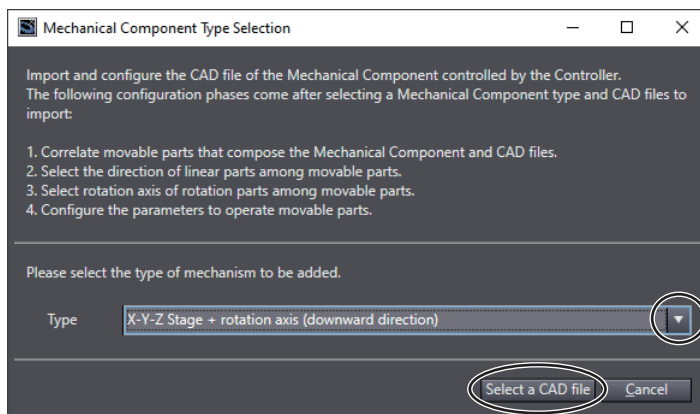
Import the CAD data file that you prepared. Use the following import procedure.

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



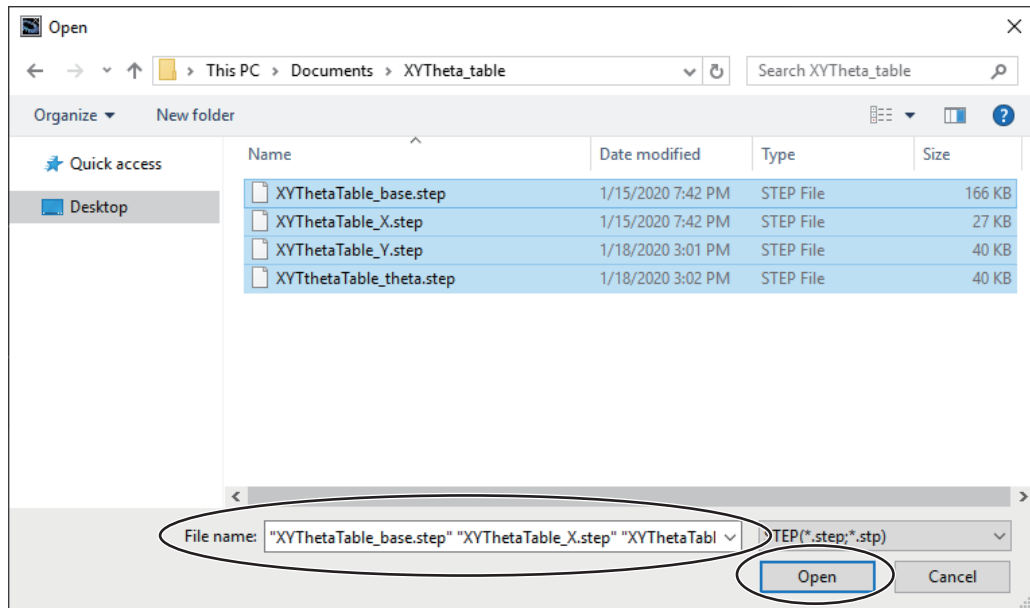
The **Mechanical Component Type Selection** dialog box is displayed.

- 2 Select the type of the mechanical component, and then click the **Select a CAD file** button.



The **Open** dialog box is displayed.

- 3 Select a CAD file with a .stp, .step, .igs, or .iges 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.

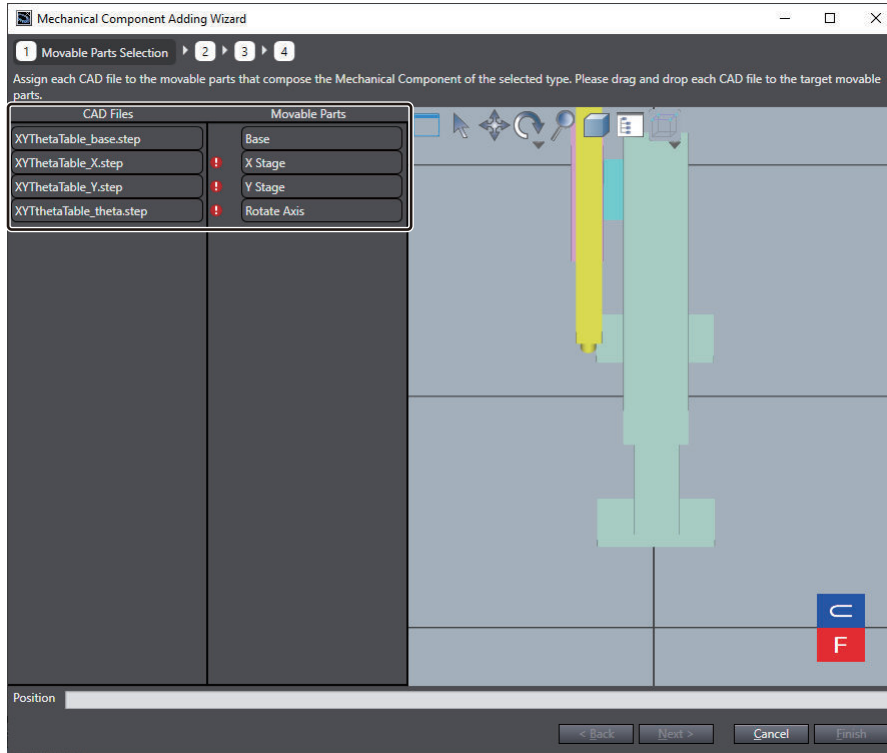
Depending on types of mechanical components, steps without any settings are omitted. Steps that are omitted are grayed out.

Step	Navigation	Type of Mechanical Component for which the step is omitted
1	Movable Parts Selection	None
2	Linear Direction Selection	Motor rotation
3	Rotate Axis Selection	<ul style="list-style-type: none"> • Single axis position control • 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)
4	Parameter Setting	None

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

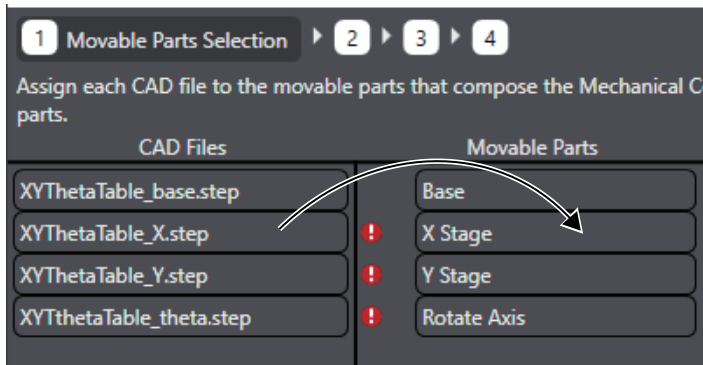
Step 1: Movable Parts Selection

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

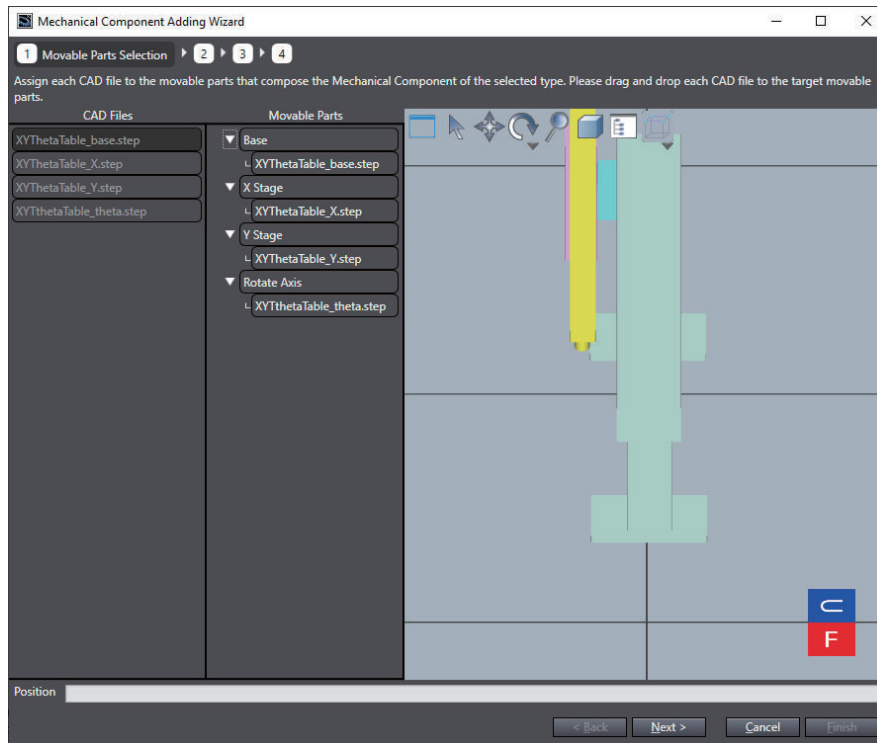


● **Procedure 1: Assigning CAD Files to the Movable Parts**

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



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



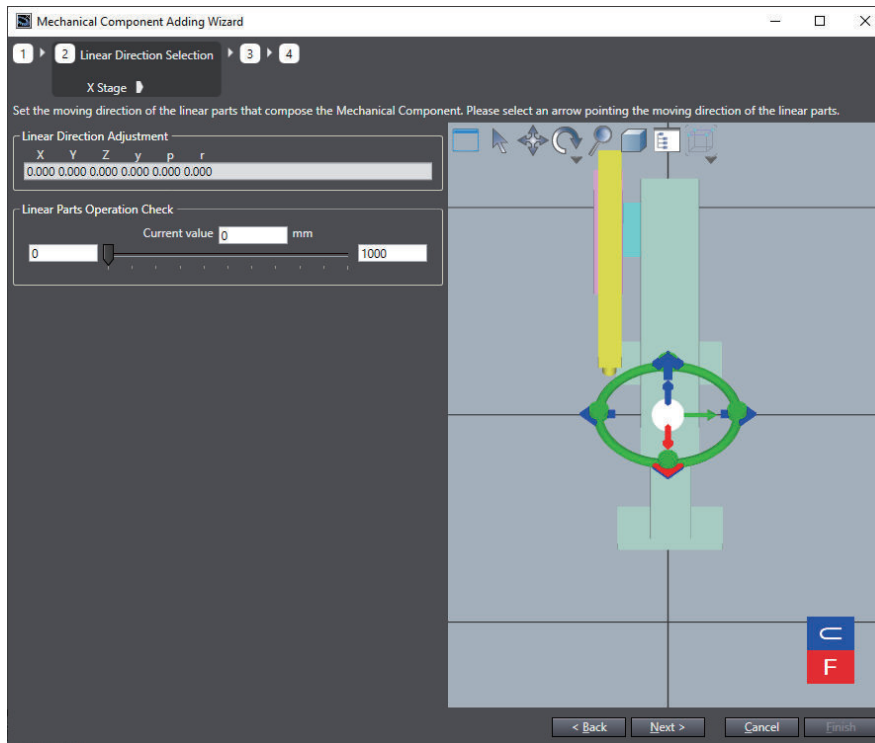
● Procedure 2: Setting the Position of the Movable Parts

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.

● 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**.

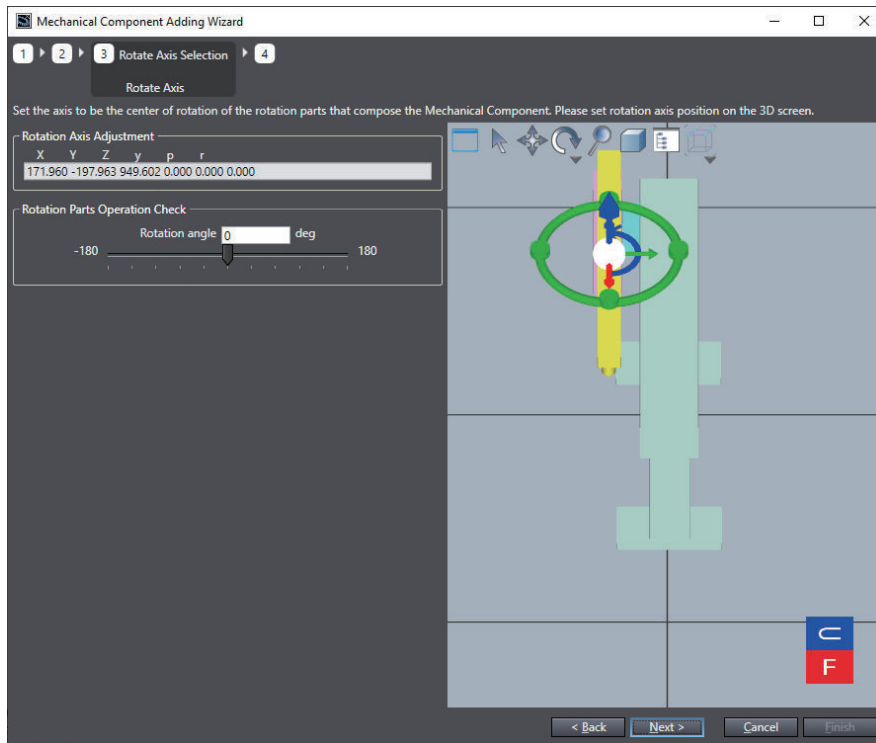
● Procedure 3: Checking the Operation of the Linear Parts

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

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

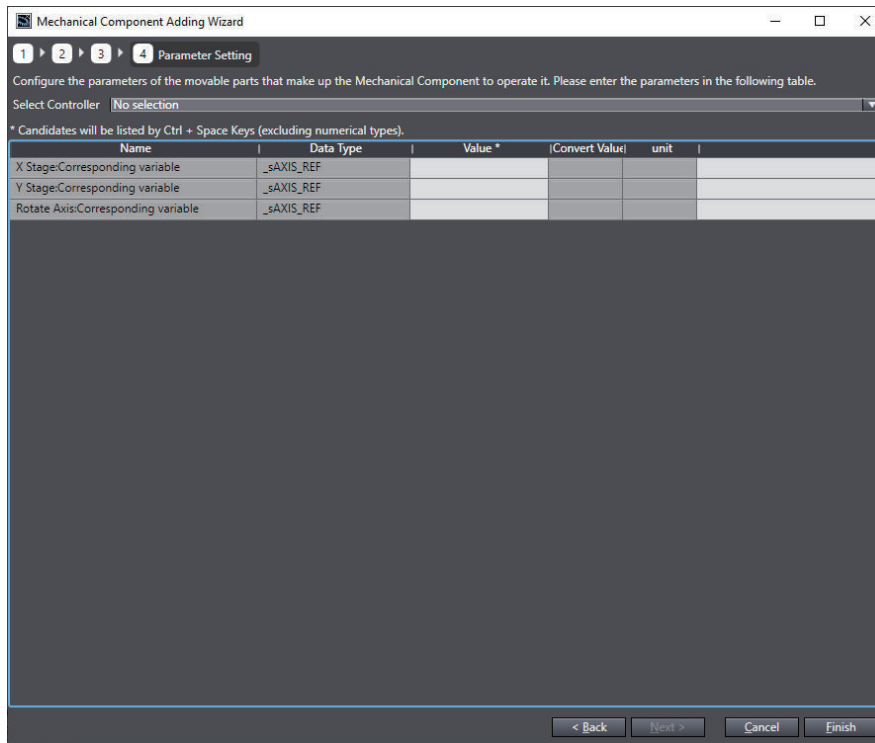
Move the slider in **Rotation Parts Operation Check** to check the operation of the rotation parts on the 3D Visualizer.

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 4-16 for details on mechanical component parameters.



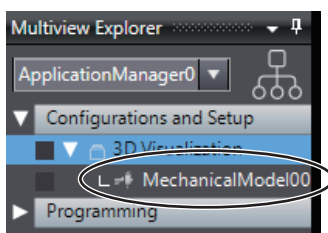
● Procedure 1: Setting the Variables According to the Data Type

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 4-10 for details.

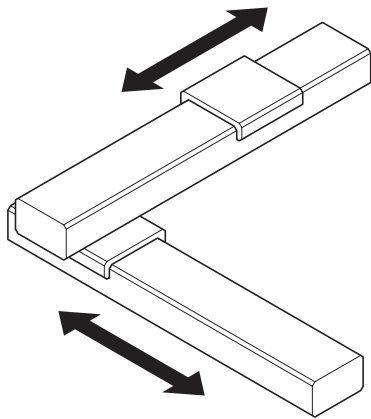
- Orthogonal robot (XY)
- Orthogonal robot (XYZ)
- X-Y-Z stage + rotation axis (upward direction)

- 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)

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.

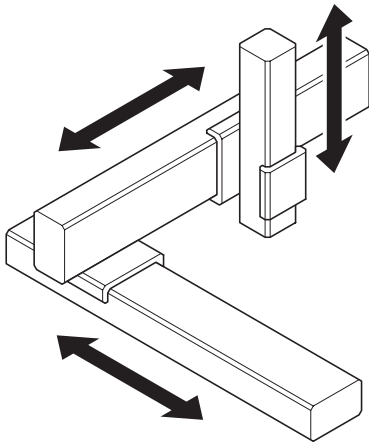
Item	Description	Set value
Axes Group: Corresponding variable*1*2	The axes group settings assigned to the Orthogonal robot (XY) model.	Variables of _sGROUP_REF data type
X Stage: Corresponding variable	Set the corresponding axis of the X stage.	Variables of _sAXIS_REF data type
Y Stage: Corresponding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

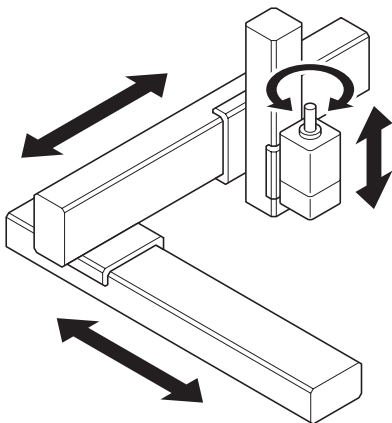
Item	Description	Set value
Axes Group: Corresponding variable ^{*1*2}	The axes group settings assigned to the Orthogonal robot (XYZ) model.	Variables of _sGROUP_REF data type
X Stage: Corresponding variable	Set the corresponding axis of the X stage.	Variables of _sAXIS_REF data type
Y Stage: Corresponding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



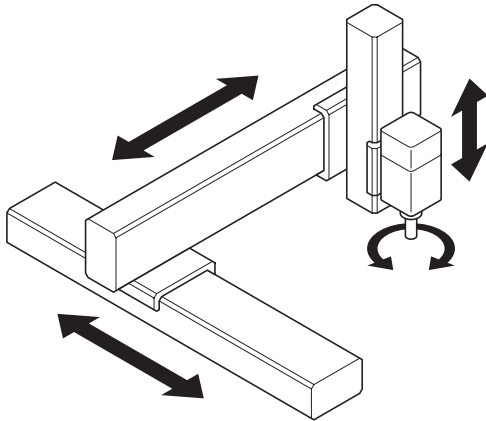
For the X-Y-Z stage + rotation axis (upward direction) model, set the following parameters.

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)

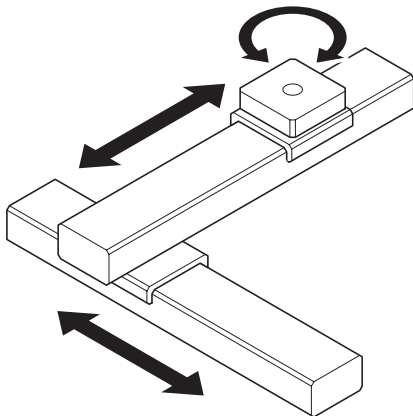
X-Y-Z stage + rotation axis (downward direction) refers to a component in which an downward rotation axis is attached to the tip of the X-Y-Z stage.

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 4-18.



X-Y Table (XY Theta)

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

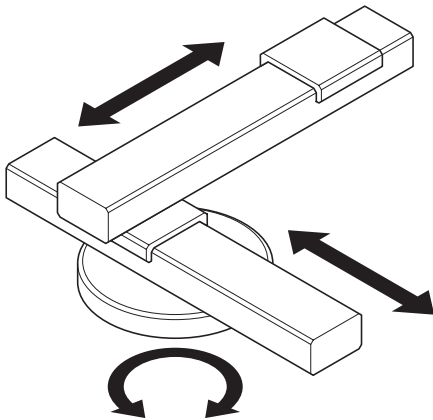


For the X-Y table (XY Theta) model, set the following parameters.

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)

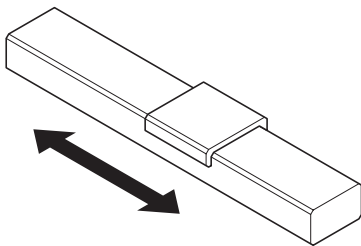
X-Y table (Theta XY) refers to a rotation component that has an X-Y table on the top.



Set the same parameters as X-Y table (XY Theta). Refer to *X-Y Table (XY Theta)* on page 4-19.

Single Axis Position Control

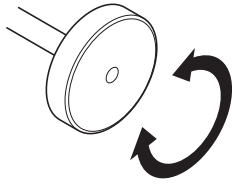
Single axis position control refers to a component that can move to any position in a straight line. For the single axis position control model, set the following parameters.



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.

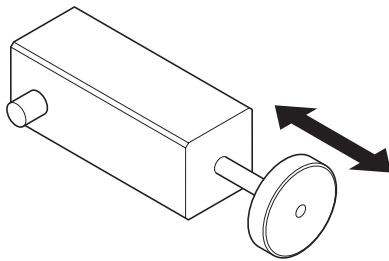


For the Motor rotation model, set the following parameters.

Item	Description	Set value
Motor: Corresponding variable	Set the corresponding axis of the motor.	Variables of <code>_sAXIS_REF</code> 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.

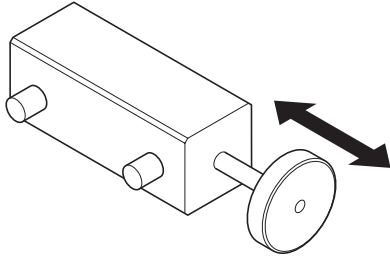


For the Air cylinder (Single solenoid type) model, set the following parameters.

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.

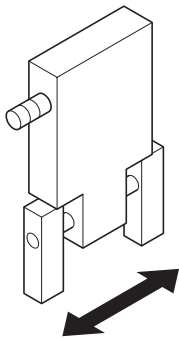


For the Air cylinder (Double solenoid type) model, set the following parameters.

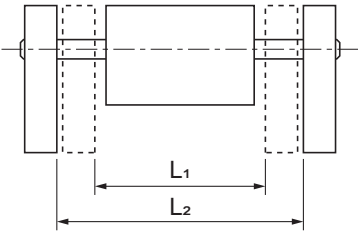
Item	Description	Set value	Initial value
Advance switch: Corresponding variable	Set a variable by which the piston is extended when the air cylinder is returned.	BOOL global variable	---
Return switch: Corresponding variable	Set a variable by which the piston is returned when the air cylinder is extended.	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.

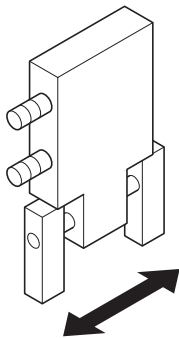


For the Robot tool (Parallel switching 2-finger type chuck/single solenoid type) model, set the following parameters.

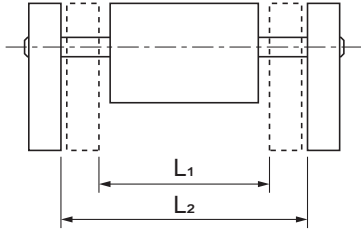
Item	Description	Set value	Initial value
Chuck: Corresponding variable	Set the I/O variable by which air injection to the chuck is started and stopped.	BOOL global variable	---
Virtual output (Open position detection): Corresponding variable	Set the I/O variable that is turned ON when the chuck is completely 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) Set the value of L2 minus L1 in the figure below. 	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) Set the value of L2 minus L1 in the figure below. 	0 to 100	10
Chuck: Initial status	Set the initial status of the chuck.	Open/Close	Open

4-3-5 Mechanical Component Settings

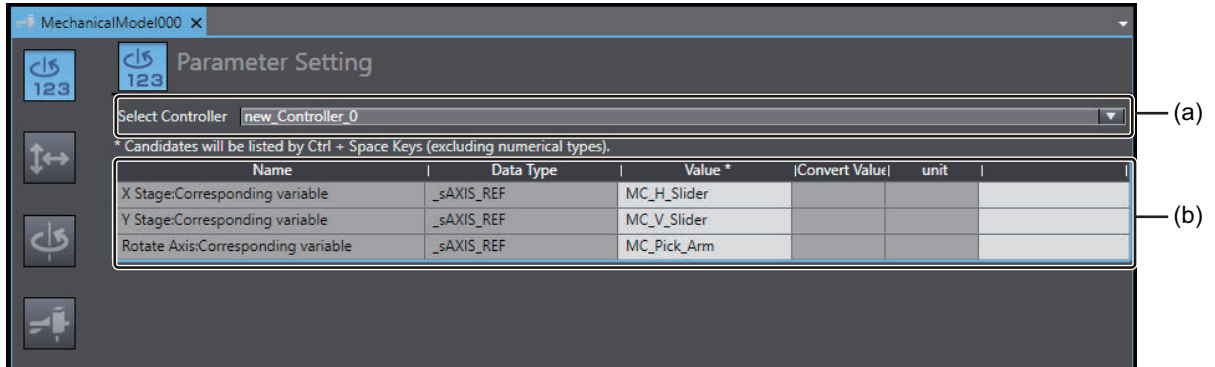
Double-click the mechanical component that you added to display the setup tab page for the mechanical component.

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.

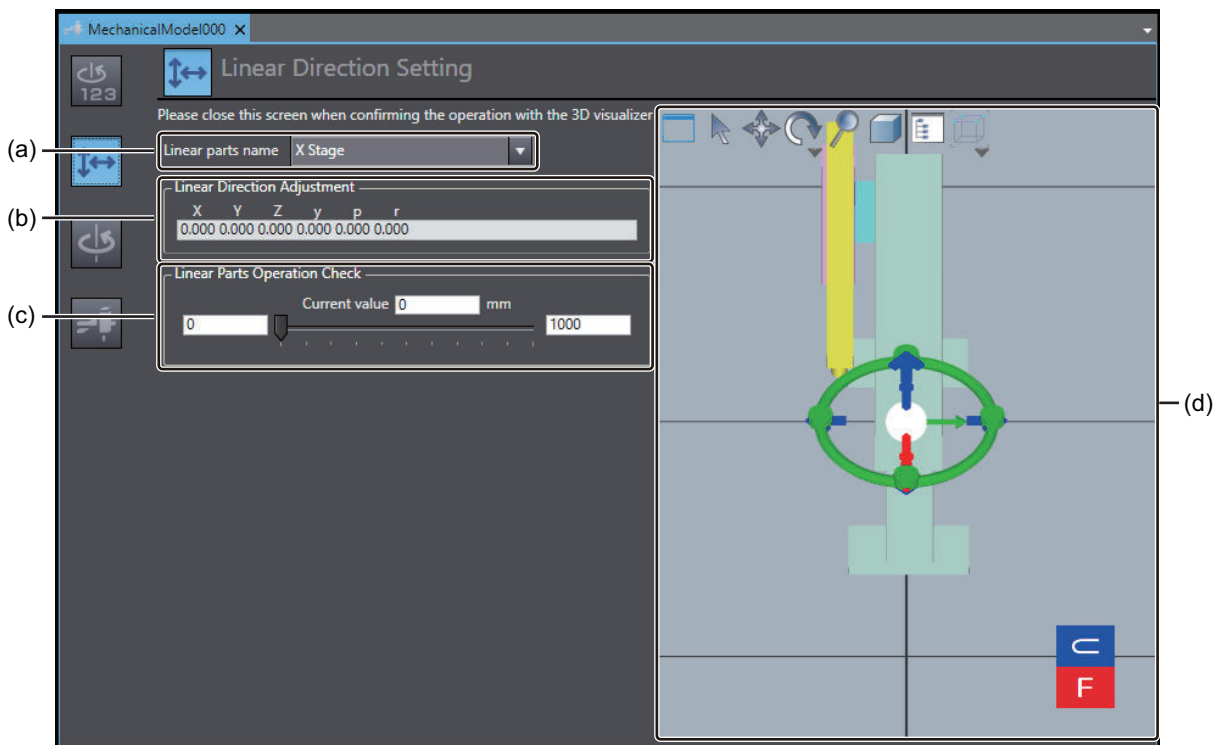


	Item	Description	Set value	Initial value ^{*1}
(a)	Select Controller	Select the Controller to control a mechanical component.	Controller name	No selection
(b)	Axis variable settings	Assign the axis variables to control the target movable parts of the mechanical component.	Axis variables for the selected 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	---

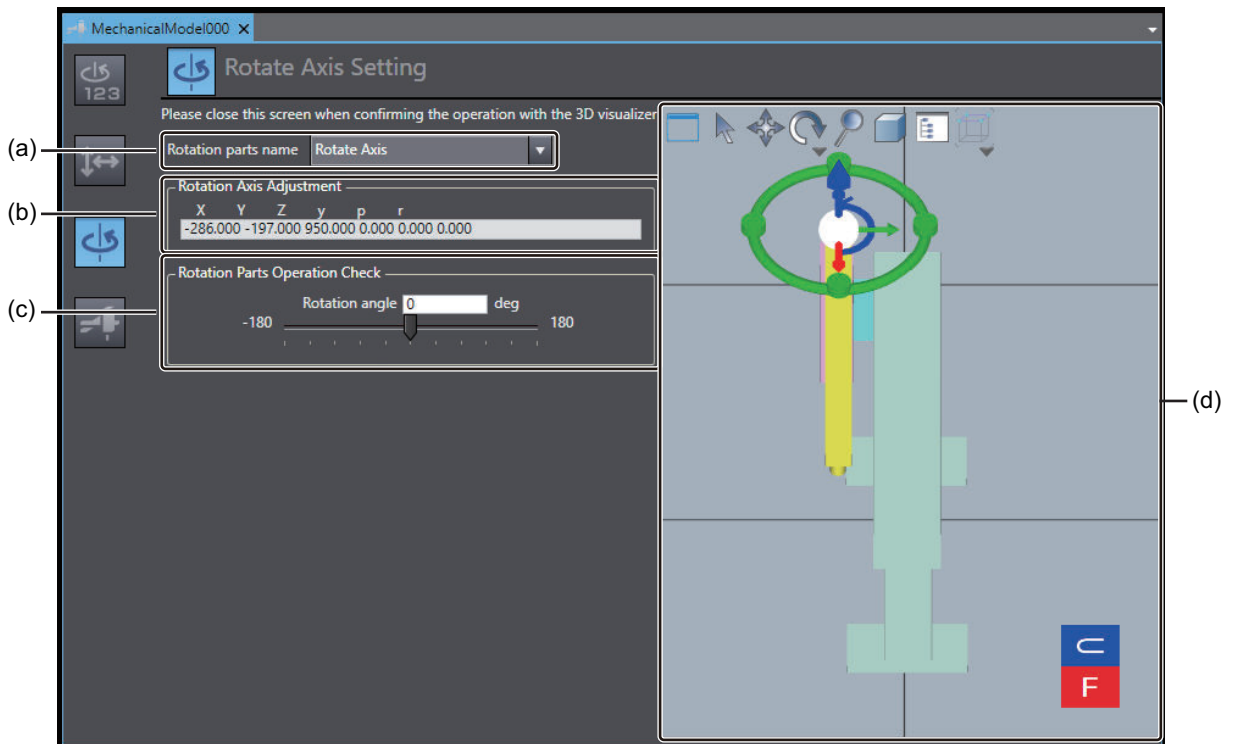
	Item*1	Description	Set value	Initial value*2
(b)	Linear Direction Adjustment	Set the start position of operation and posture of the linear parts. The location elements are X, Y, Z, y (yaw), p (pitch), and r (roll), from left to right.	For each, -1,000,000.000 to 1,000,000.000	0.000 for all
(c)	Linear 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 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

	Item*1	Description	Set value	Initial value*2
(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 .	---	---

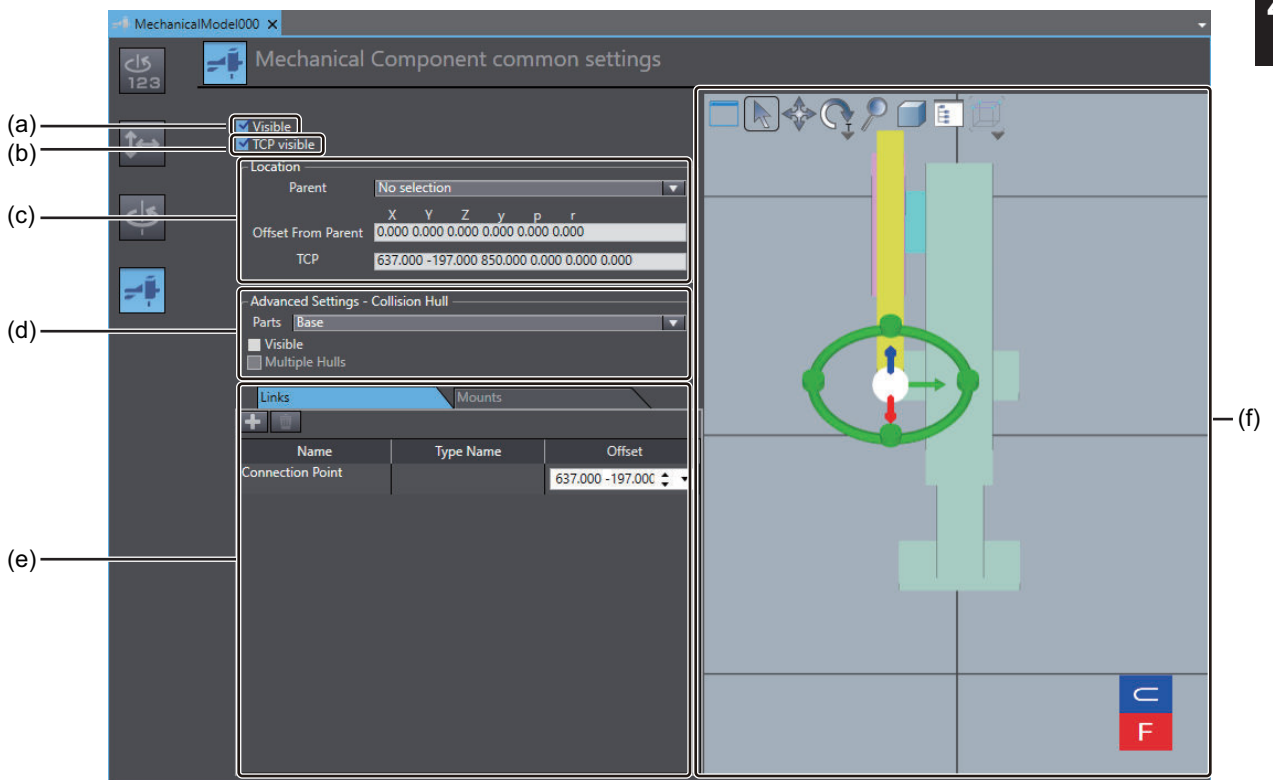
*1. For mechanical components without parts that make a rotational operation, the items display no value.

*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)	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
(b)	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

	Item		Description	Set value	Initial value
(c)	Location	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.	For each, -1,000,000.000 to 1,000,000.000	0.000 for all
		TCP	TCP is the coordinate 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 coordinate in the local coordinate system of the 3D shape data for the mechanical component.	For each, -1,000,000.000 to 1,000,000.000	0.000 for all
(d)	Advanced Settings – Collision Hull	Parts	Select the parts for which to set collision filter models.	Parts that make up the mechanical component	---
		Visible	Set whether to show collision filter models.	Checked or unchecked	Unchecked
		Multiple Hulls	Set whether to split the 3D shape data into multiple pieces to generate collision detection models. Splitting the 3D shape data increases the time required for collision detection, but improves the accuracy of the collision detection. This setting is not available if any of the following conditions is met. <ul style="list-style-type: none"> The 3D shape data consists of a single part. The 3D shape data is generated with Sysmac Studio version 1.41 or lower, which does not support collision filter model generation. You cannot change this setting when the Falling option is enabled.	Checked or unchecked	Unchecked
(e)	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 <i>5-4 Positioning with a Mount Point or a Link Point</i> on page 5-24 for information on how to set the mount points and link points.	---	---	
(f)	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 <i>Section 5 Working with the 3D Visualizer and 3D Editing Area</i> on page 5-1 for details on how to make items visible and work with them in the 3D editing area.	---	---	

4-4 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.

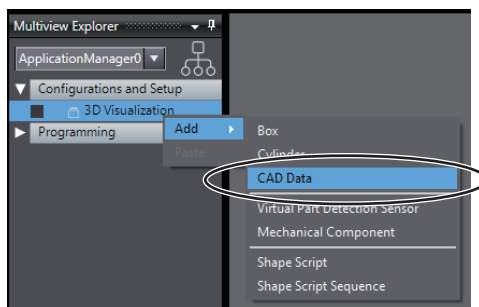
4-4-1 Importing CAD Data

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

CAD Data Import Procedure

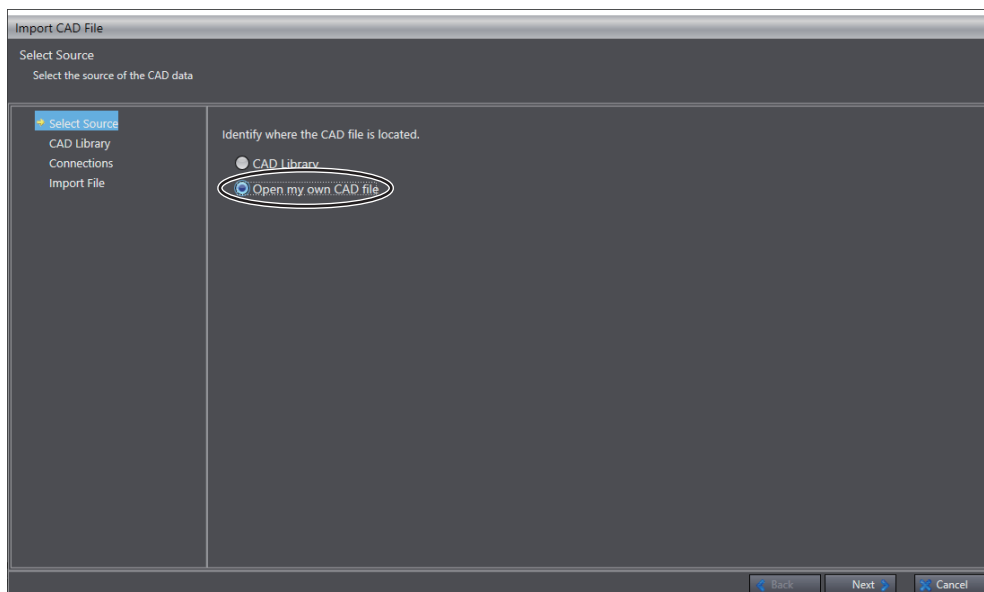
Import CAD data to Application Manager as 3D shape data for the part. Use the following import procedure.

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



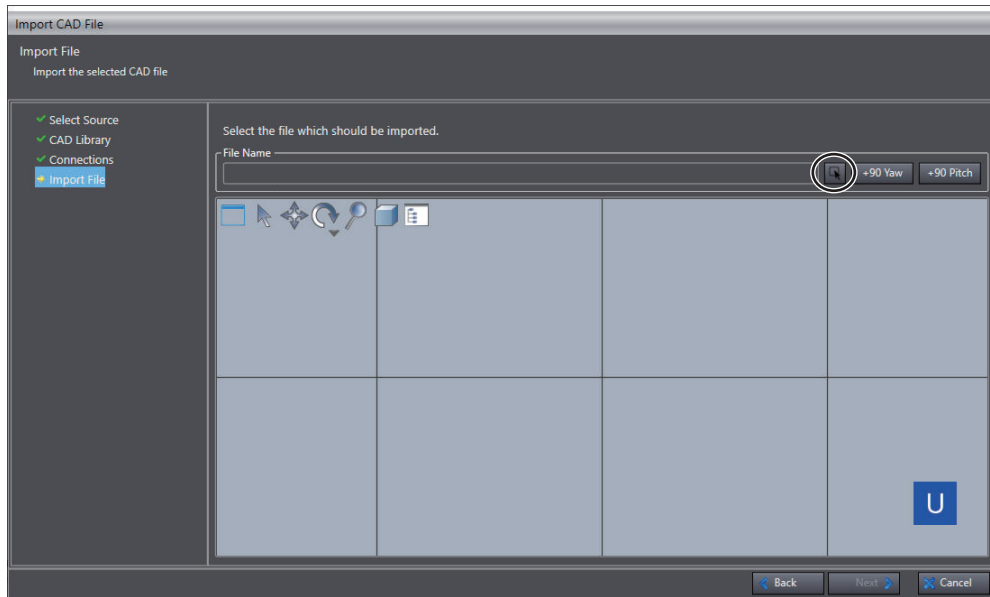
The **Import CAD File** wizard starts, and then the **Select Source** page is displayed.

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



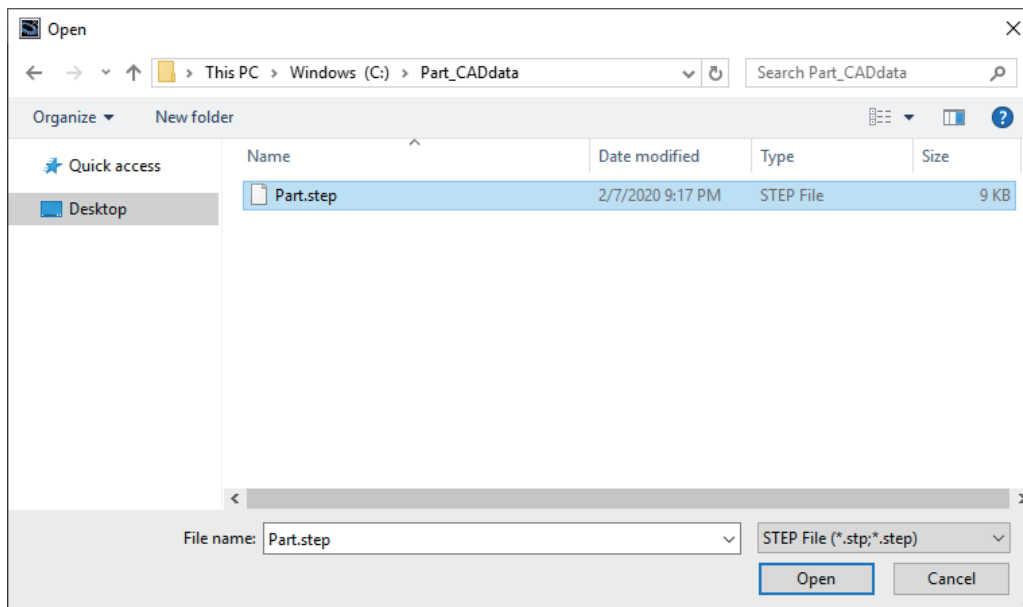
The **Import File** page is displayed.

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, or .iges file name extension, and then click the **Open** button.

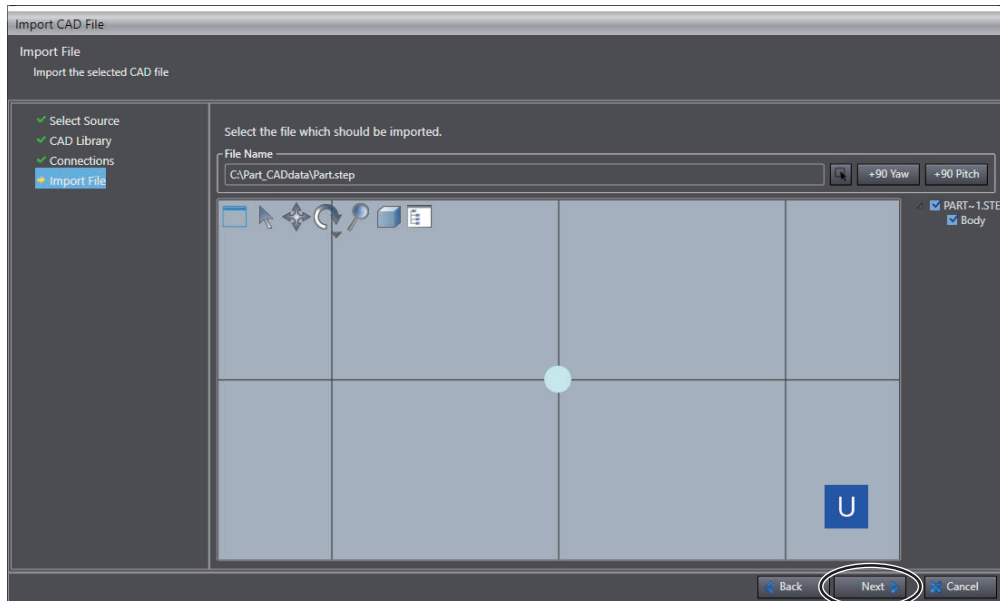


The CAD file is imported.

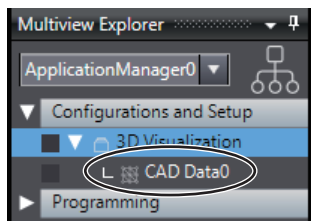
If necessary, click the **+90 Yaw** or **+90 Pitch** button to change the orientation of the CAD data to import.

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.

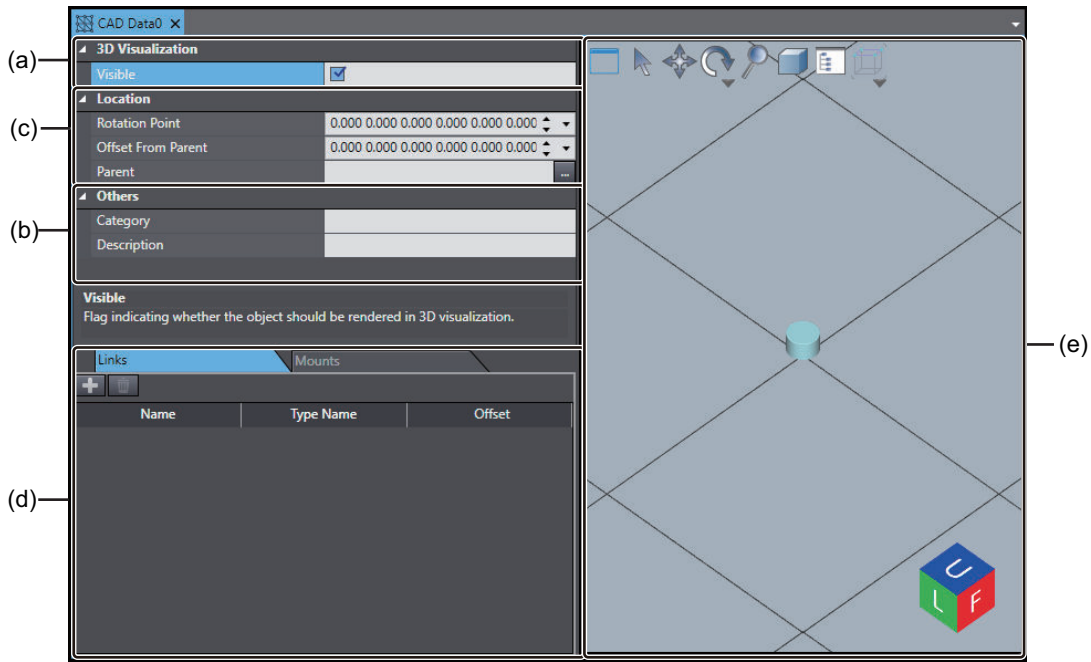


The imported CAD data is added and displayed under **3D Visualization** in the Multiview Explorer.



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
(b)	Others	Category/ Description	Describe the explanation of this 3D shape data as required.	Text Strings	None
(c)	Location	Rotation Offset	Set the rotation offset from the origin of the location 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.	For each, -1,000,000.000 to 1,000,000.000	0.000 for all
		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. 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.	For each, -1,000,000.000 to 1,000,000.000	0.000 for all
(d)	Mount point or link point setup area		Use this area to set mount points or link points for this 3D shape data. Refer to <i>5-4 Positioning with a Mount Point or a Link Point</i> on page 5-24 for information on how to set mount points and link points.	---	---

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

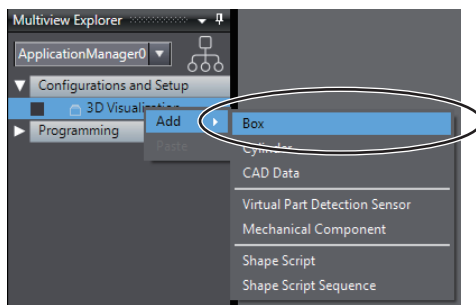
4-4-2 Adding a Box or a Cylinder

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

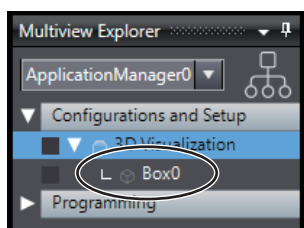
Box or Cylinder Addition Procedure

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.

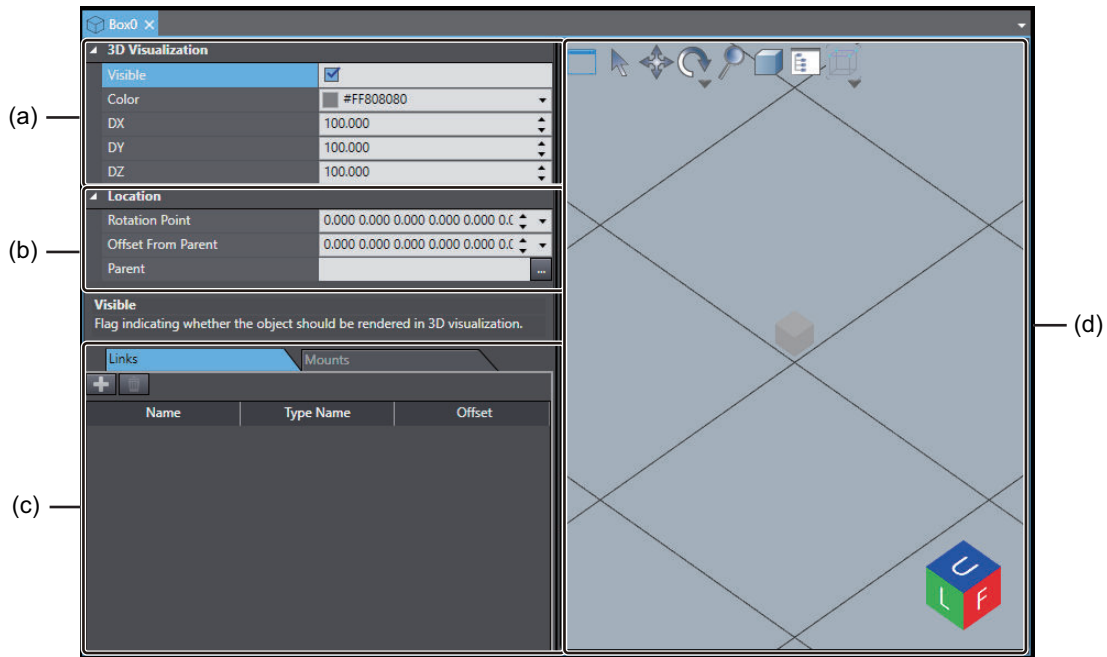


Box Settings

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 Visualization	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 unchecked	Checked
		Color	Set the color of a box.	Transparency (0 to #FF) and RGB (0 to #FF, 0 to #FF, 0 to #FF)	#FF8080
		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	Rotation Offset	Set the rotation offset from the origin of the location coordinate system for a box. 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.	For each, -1,000,000.000 to 1,000,000.000	0.00 for all
		Parent	Select the 3D shape data to be the parent of a box. 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 a box. 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.	For each, -1,000,000.000 to 1,000,000.000	0.00 for all

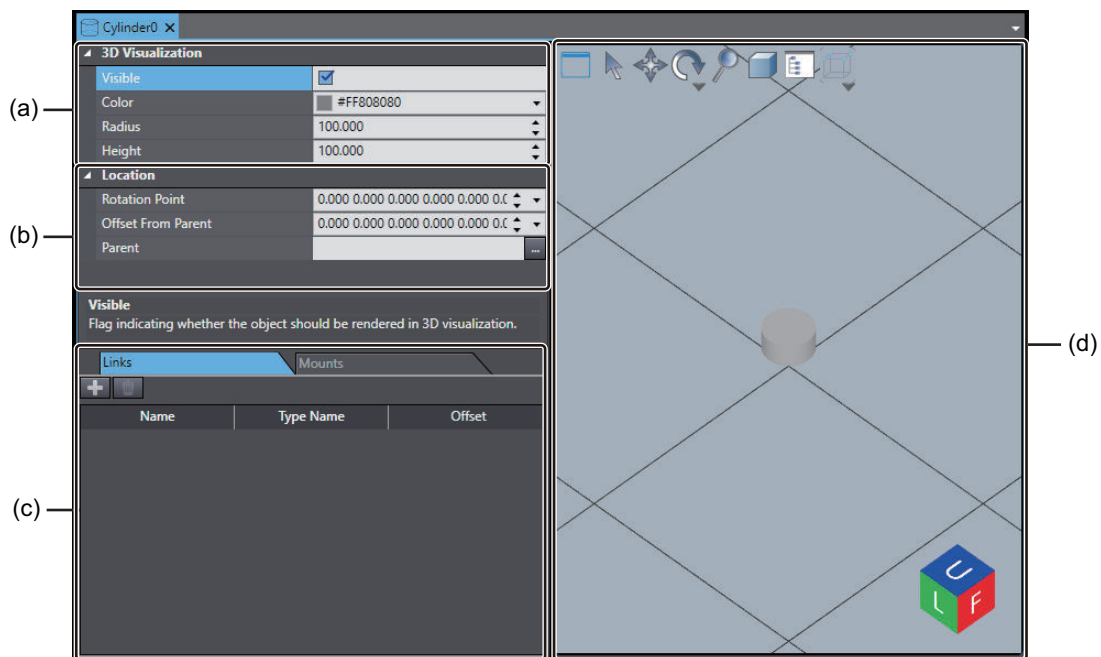
	Item	Description	Set value	Initial value
(c)	Mount point or link point setup area	Use this area to set mount points or link points for the Box. Refer to <i>5-4 Positioning with a Mount Point or a Link Point</i> on page 5-24 for information on how to set mount points and link points.	---	---
(d)	3D editing area	Use this area to display the position, pose, and mount points or link points of a box. Refer to <i>Section 5 Working with the 3D Visualizer and 3D Editing Area</i> 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

	Item		Description	Set value	Initial value
(b)	Location	Rotation Offset	Set the rotation offset from the origin of the local coordinate system for a cylinder. 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.	For each, -1,000,000.000 to 1,000,000.000	0 for all
		Parent	Select the 3D shape data to be the parent of a cylinder. Click the button at the right, and then select the parent 3D shape data from the list.	Text Strings	None
		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.	For each, -1,000,000.000 to 1,000,000.000	0.00 for all
(c)	Mount point or link point setup area		Use this area to set mount points or link points for a cylinder. Refer to <i>5-4 Positioning with a Mount Point or a Link Point</i> on page 5-24 for information on how to set mount points and link points.	---	---
(d)	3D editing area		Use this area to display the position, pose, and mount points or link points of a cylinder. Refer to <i>Section 5 Working with the 3D Visualizer and 3D Editing Area</i> on page 5-1 for details on how to make items visible and work with them in the 3D editing area.	---	---

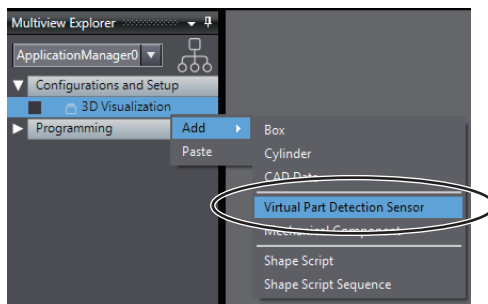
4-5 Adding the Part Detection Sensor

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.

4-5-1 Adding the Virtual Part Detection Sensor

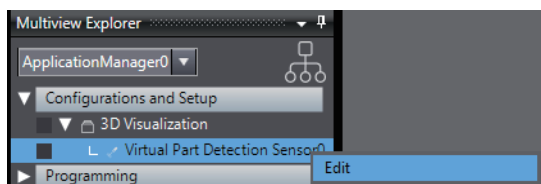
Add the Virtual Part Detection Sensor to the Application Manager.

- 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.

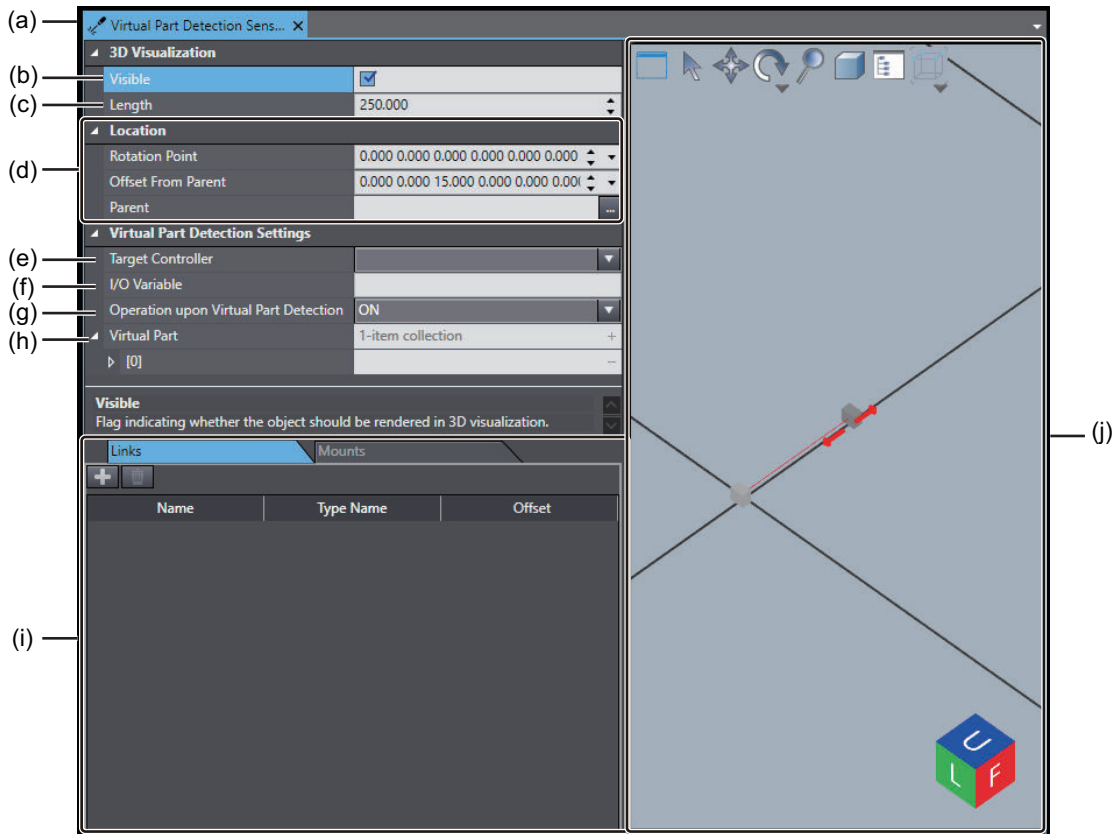


The Virtual Part Detection Sensor setup tab page is displayed.

4-5-2 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
(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.
(c)	Length	Set the length of the Virtual Part Detection Sensor. You cannot change the thickness.
(d)	Location	Configure the location settings. The settings are the same as those for CAD data, boxes, and cylinders.
(e)	Virtual Part Detection Settings	Target Controller Select the Controller to detect the part. With Sysmac Studio version 1.42 or higher, you can select robot sub-devices in addition to the Controller.
(f)		I/O Variable The I/O variable represents the status that the Virtual Part Detection Sensor has detected the part. Set a BOOL global variable for the target Controller. If the target is a robot sub-device, set the V+Digital I/O.
(g)		Operation upon Virtual Part Detection Set how the I/O variable operates when the part is detected by the Virtual Part Detection Sensor. If it changes to TRUE when the part is detected, set <i>ON upon detection</i> . If it changes to FALSE when the part is detected, set <i>OFF upon detection</i> .
(h)		Virtual Part Set the name of the target part detected by the Virtual Part Detection Sensor. You can set a part name that is included in the same Application Manager.
(i)	Mount point or link point setup area	Set mount points and link points for the Virtual Part Detection Sensor. Refer to <i>5-4 Positioning with a Mount Point or a Link Point</i> on page 5-24 for information on how to set mount points and link points.

	Item	Description
(j)	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 <i>Section 5 Working with the 3D Visualizer and 3D Editing Area</i> on page 5-1 for details on how to make items visible and work with them in the 3D editing area.

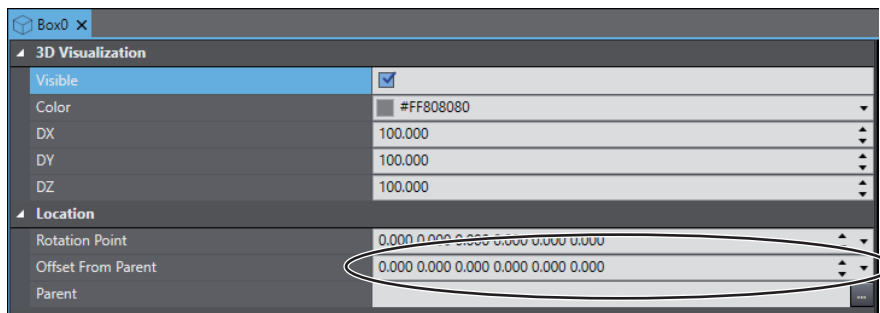
4-6 3D Shape Data Placement Methods

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

4-6-1 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.



4-6-2 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 5-19 for details.

4-6-3 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 5-24 for the setting and operating procedures.

5

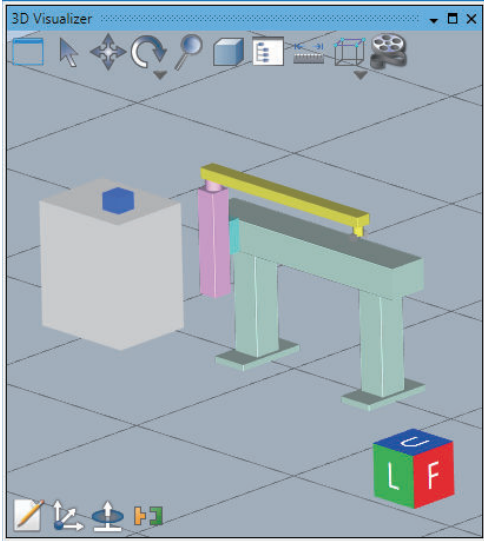
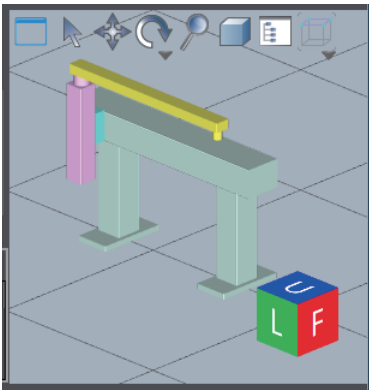
Working with the 3D Visualizer and 3D Editing Area

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	5-2
5-1-1	Displaying the 3D Visualizer	5-2
5-1-2	Displaying the 3D Editing Area	5-3
5-2	Displayed Items in the 3D Visualizer and 3D Editing Area	5-5
5-2-1	Split Window	5-6
5-2-2	Selection and Edit	5-7
5-2-3	Translate	5-7
5-2-4	Rotate	5-8
5-2-5	Zoom	5-8
5-2-6	Projection Mode	5-9
5-2-7	Scene Graph	5-10
5-2-8	Measurement Ruler	5-12
5-2-9	Snap	5-13
5-2-10	Record	5-14
5-2-11	3D View Switching Tool	5-17
5-3	Operating 3D Shape Data in the 3D Visualizer	5-19
5-3-1	Moving 3D Shape Data	5-19
5-3-2	Rotating 3D Shape Data	5-20
5-3-3	Editing 3D Shape Data Simply	5-22
5-4	Positioning with a Mount Point or a Link Point	5-24
5-4-1	Outline of a Mount Point and a Link Point	5-24
5-4-2	Setting a Mount Point	5-26
5-4-3	Setting a Link Point	5-27
5-4-4	Offset Setting Methods	5-29
5-4-5	Using a Mount Point and a Link Point to Place 3D Shape Data	5-33

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

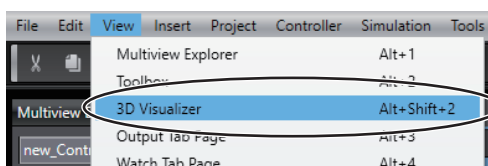
The 3D Visualizer and 3D editing area both display the 3D shape data that you added. However, the 3D Visualizer and 3D editing area are different as described below.

	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.	

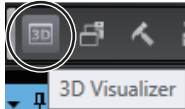
5-1-1 Displaying the 3D Visualizer

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

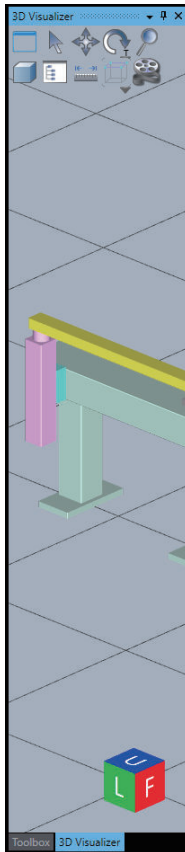
- 1 Select **3D Visualizer** from the **View** menu.



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



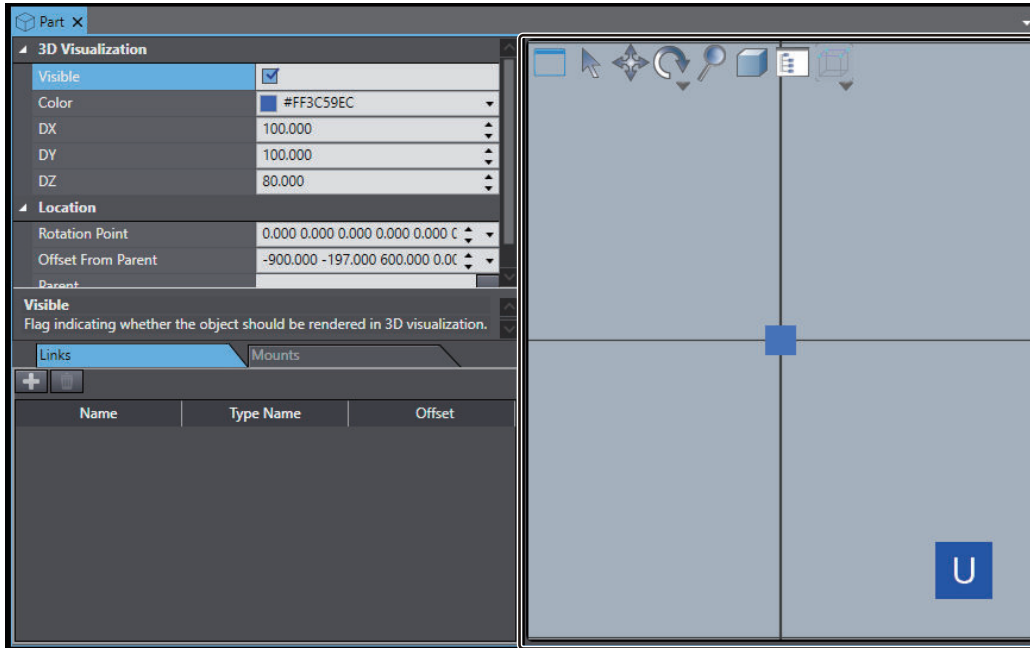
The **3D Visualizer** is displayed in the same place of the toolbox pane.



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.

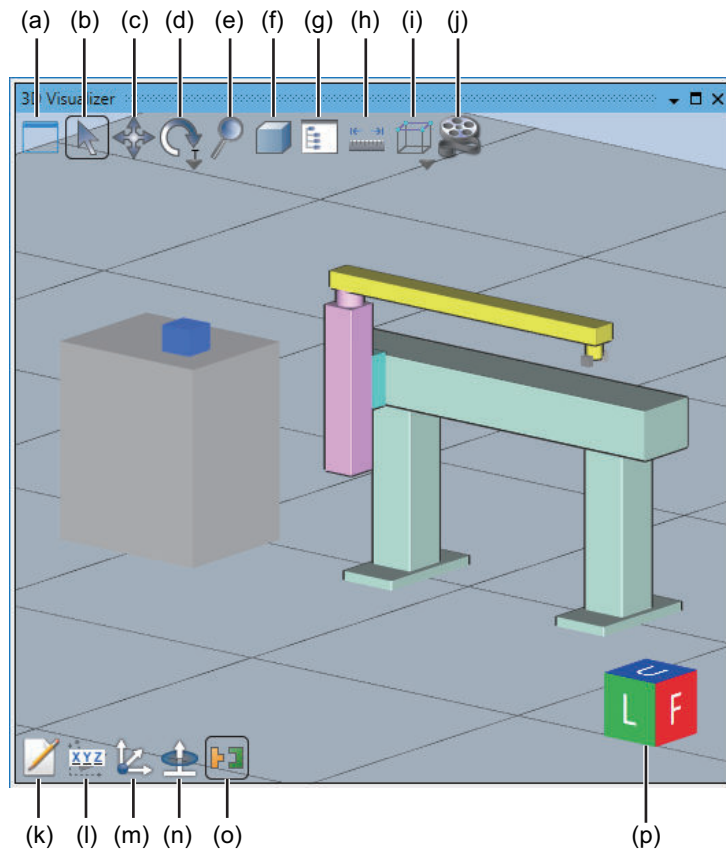
5-1-2 Displaying the 3D Editing Area

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.



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 <i>Split Window</i> on page 5-6
(b)	Selection	Selects the 3D shape data to edit.	5-2-2 <i>Selection and Edit</i> on page 5-7
(c)	Translate	Translates the point of view in the 3D Visualizer or 3D editing area.	5-2-3 <i>Translate</i> on page 5-7
(d)	Rotate	Rotates the point of view in the 3D Visualizer or 3D editing area.	5-2-4 <i>Rotate</i> on page 5-8
(e)	Zoom	Zooms in or out the 3D Visualizer or 3D editing area.	5-2-5 <i>Zoom</i> 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 <i>Projection Mode</i> 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 <i>Scene Graph</i> on page 5-10
(h)*1	Measurement Ruler	Measures the distance between 3D shape data in the 3D Visualizer.	5-2-8 <i>Measurement Ruler</i> 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 <i>Snap</i> on page 5-13
(j)*1	Record	Captures a simulation executed in the 3D Visualizer on video.	5-2-10 <i>Record</i> on page 5-14
(k)*1	Edit	Displays the settings for the selected 3D shape data.	5-2-2 <i>Selection and Edit</i> 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 <i>Editing 3D Shape Data Simply</i> on page 5-22
(m)*1	Edit Workspace Position	Moves the position of the selected 3D shape data.	5-3-1 <i>Moving 3D Shape Data</i> on page 5-19
(n)*1	Edit Workspace Orientation	Changes the orientation of the selected 3D shape data.	5-3-2 <i>Rotating 3D Shape Data</i> on page 5-20
(o)*1	Show/Hide Mount Points	Shows or hides mount points.	5-4 <i>Positioning with a Mount Point or a Link Point</i> on page 5-24
(p)	3D View Switching Tool	Switches the display direction of 3D shape data in the 3D Visualizer or 3D editing area.	5-2-11 <i>3D View Switching Tool</i> on page 5-17

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

Refer to *5-3 Operating 3D Shape Data in the 3D Visualizer* on page 5-19 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**.



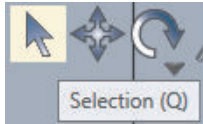
The functions of icons that you can select in the window are as follows.

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.

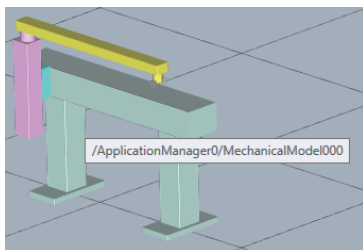
- 1 Click the **Selection** icon in the 3D Visualizer or 3D editing area. Or, press the Q key.



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

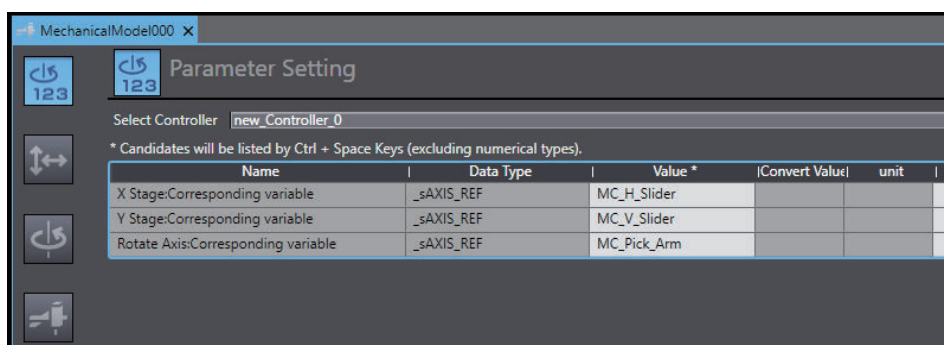


- 2 Move the cursor to the 3D shape data to select.
The name of the 3D shape data at the cursor position is displayed.



- 3 To edit the 3D shape data in the 3D Visualizer, double-click the 3D shape data. Or, click the **Edit** icon.

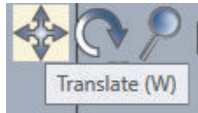
The setup tab page for the 3D shape data is displayed.



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.

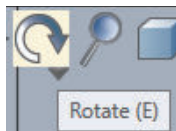


- 2 Press and hold the left mouse button, and then drag the mouse in the direction to translate.

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.

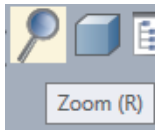
There are two rotation modes as follows.

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 $\pm 90^\circ$ 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.



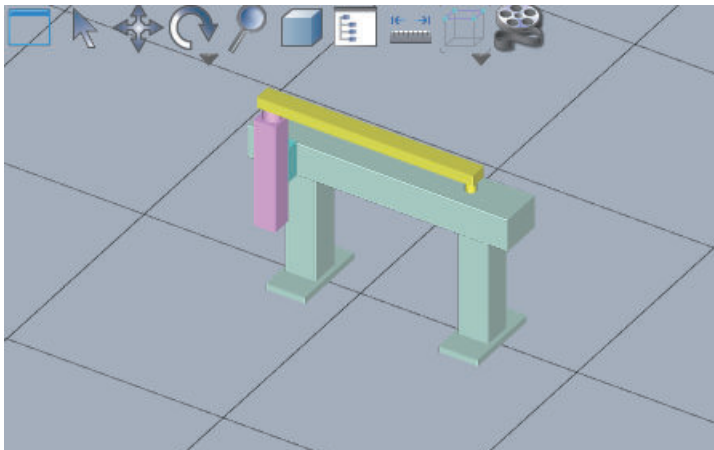
- 2 Press and hold the left mouse button, and then drag the mouse.
 Dragging up: Zoom-in
 Dragging down: Zoom-out

5-2-6 Projection Mode

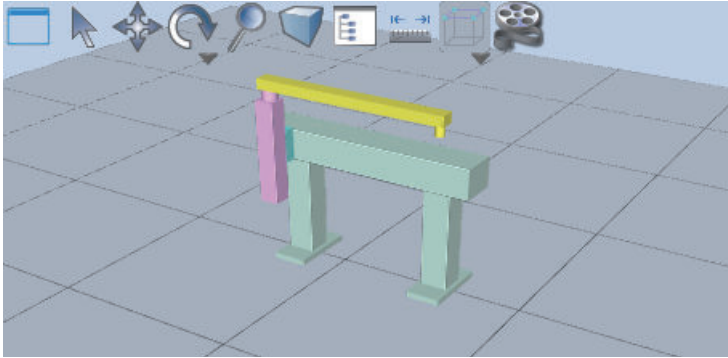
Use this icon to change the projection mode in the 3D Visualizer or 3D editing area between parallel projection and perspective projection.

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. 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. 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

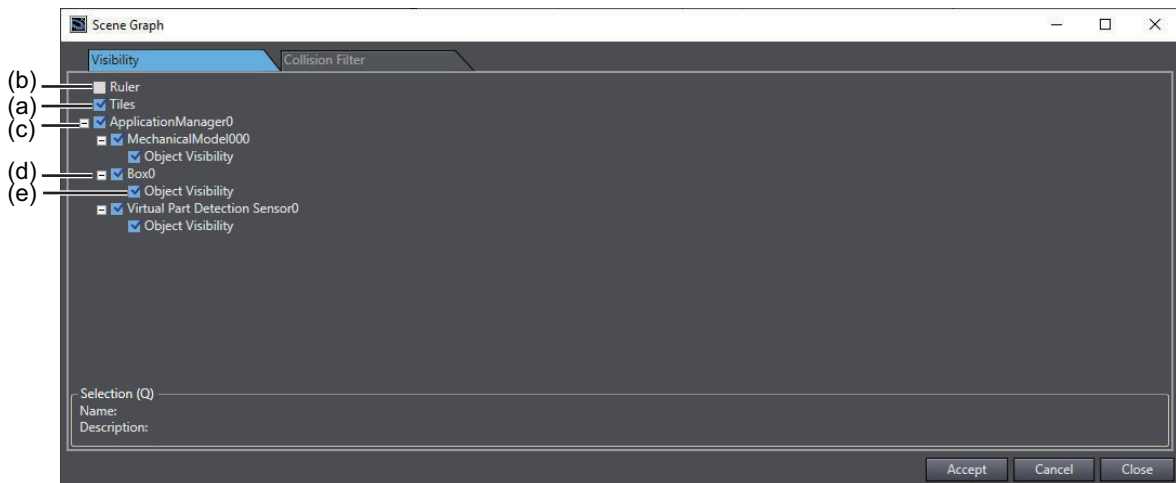
Use this icon to configure the visibility and collision detection settings for 3D shape data.

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.



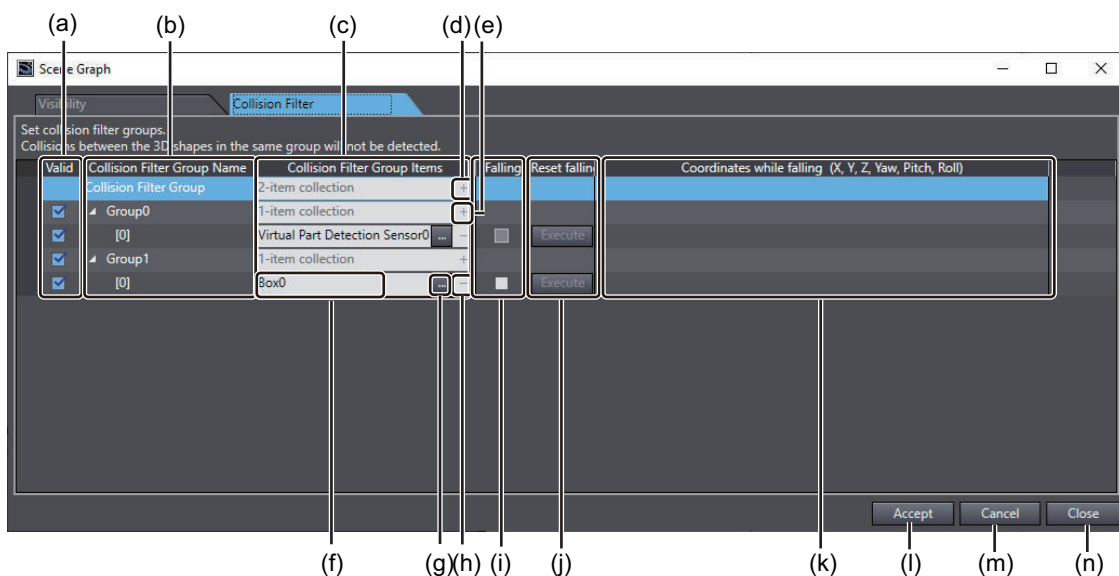
	Item	Description	Set value	Initial value
(a)	Tiles	Select whether to display the XY plane with a 1,000-mm mesh when Z=0 in the 3D Visualizer or 3D editing area.	Checked or unchecked	Checked
(b)	Ruler	Select whether to display a ruler when you measure the distance between 3D shape data in the 3D Visualizer. 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. The text box displays <i>*-itemcollection (* is the collision filter group number)</i> . Click the Add Collision Filter Group button to add a collision filter group. The group name to be added is <i>Group*</i> (<i>* is the number of collision filter groups</i>).	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. <i>*-itemcollection (* is the number of collision filter group items)</i> 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. Clicking this button adds a collision filter group to the list.	---	---
(e)	Add Collision Filter Group Item button	Adds a collision filter group item. Clicking this button adds a collision filter group item setting row to the list.	---	---

	Item	Description	Set value	Initial value
(f)	Collision Filter Group Item name	Set the target to add to the collision filter group. Refer to 7-3-1 <i>Collision Detection Target</i> on page 7-13 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. Clicking this button deletes the item.	---	---
(i)	Falling	Clicking the Accept button with this check box selected causes the target 3D shape data to fall in the negative direction of the Z axis in the 3D Visualizer or 3D editing area. This allows you to check how the falling 3D shape data collides with other 3D shape data.	Checked or unchecked	Unchecked
(j)	Reset falling	Resets the fallen target 3D shape data to the original position. When the Falling check box is selected, the target 3D shape data falls from the original position each time you click the Execute button.	---	---
(k)	Coordinates while falling (X, Y, Z, Yaw, Pitch, Roll)	The coordinate value indicating the position of a falling 3D shape data is displayed.	---	---
(l)	Accept button	Accepts the changes.	---	---
(m)	Cancel button	Cancels the changes and then, closes the Scene Graph dialog box.	---	---
(n)	Close button	Closes the Scene Graph dialog box.	---	---

Refer to 7-3 *Collision Detection Function* on page 7-13 for information for details on how to configure the collision detection settings.

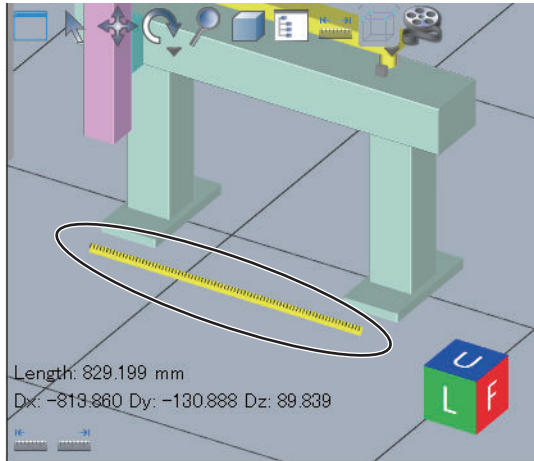
5-2-8 Measurement Ruler

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

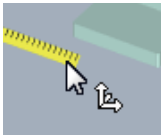
- 1 In the 3D Visualizer, click the **Measurement Ruler** icon.



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

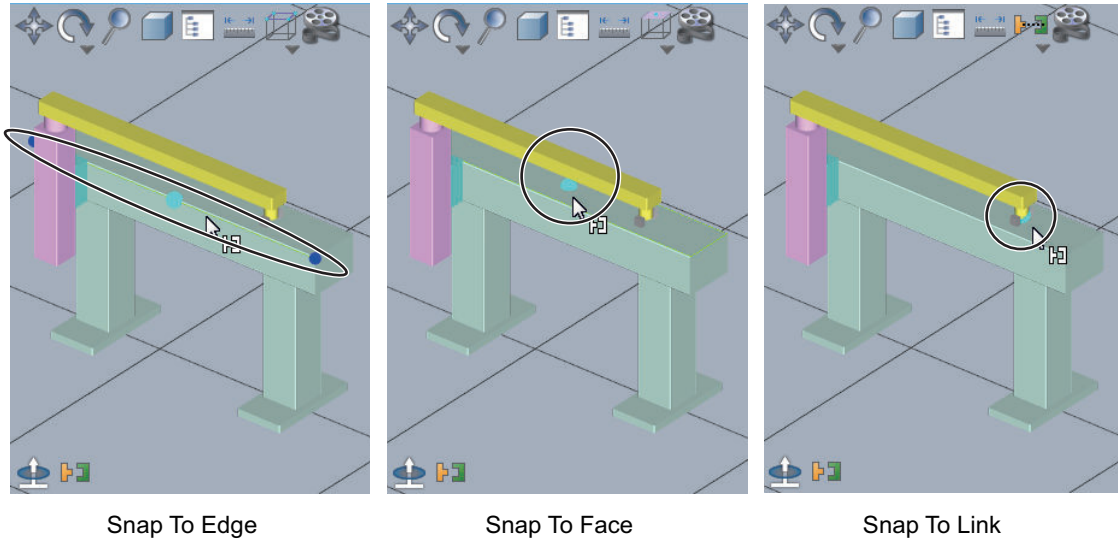
Use this icon to set an offset for a mount point or a link point of 3D shape data.

Select a mount point or a link point, and then 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
	Snap To Edge	Dragging a mount point or a link point near 3D shape data causes its edge nearest to the mouse cursor to be highlighted. 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.
	Snap To Face	Dragging a mount point or a link point near 3D shape data causes its face nearest to the mouse cursor to be highlighted. The center of gravity of the highlighted face is emphasized, and the mount point or the link point can be snapped to the position.
	Snap To Link	Dragging a mount point to near 3D shape data causes its face nearest to the mouse cursor to be highlighted. Any link point that is present on the highlighted face is displayed, and the mount point can be snapped to the position. This function is available only in the 3D Visualizer.

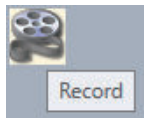


5-2-10 Record

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

Recording Video

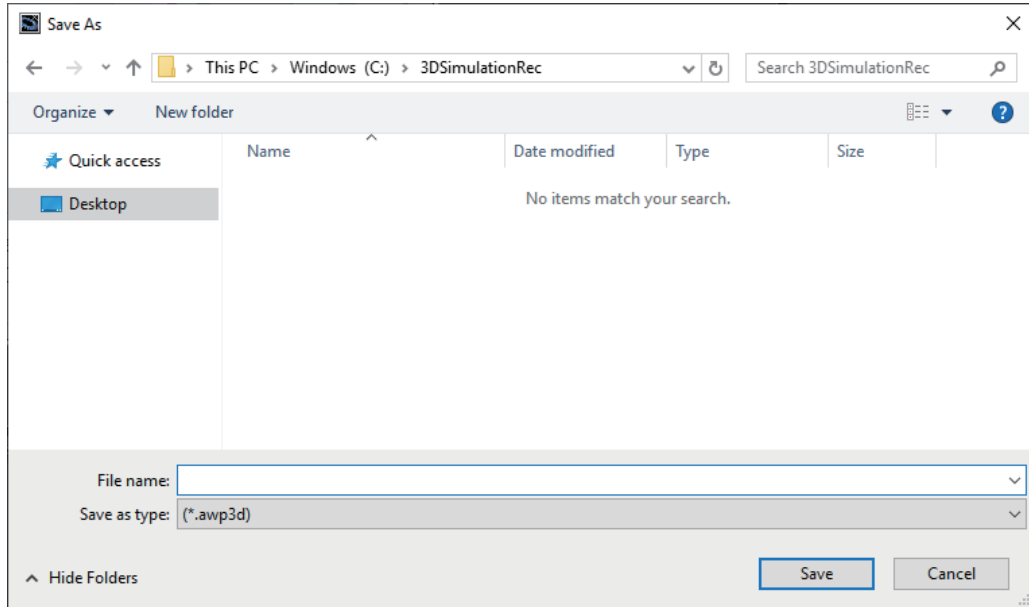
- 1 Click the **Record** icon during the execution of a 3D simulation.



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.

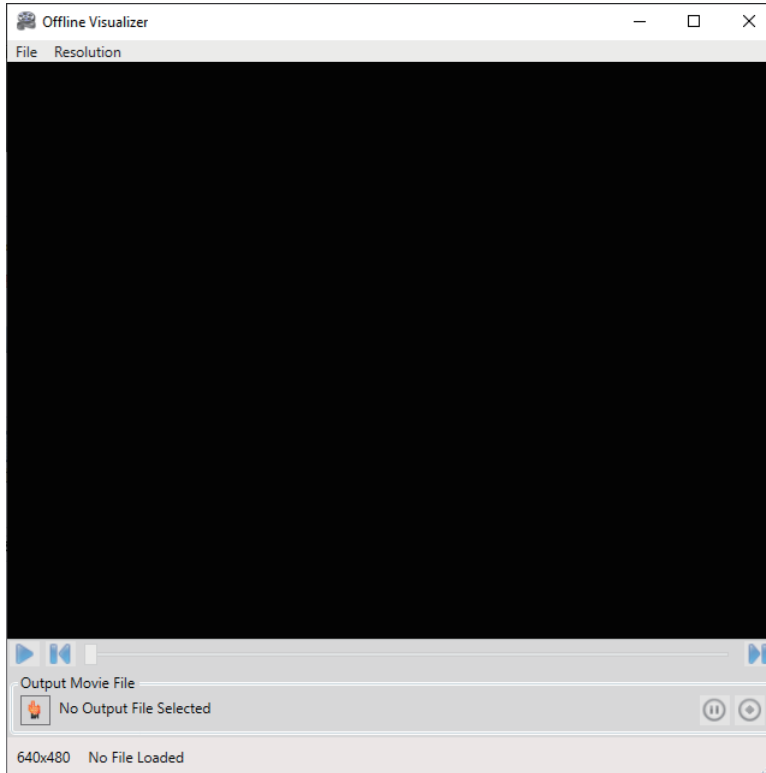


- 3 Enter the file name, and then click the **Save** button.
The video is saved to a file.

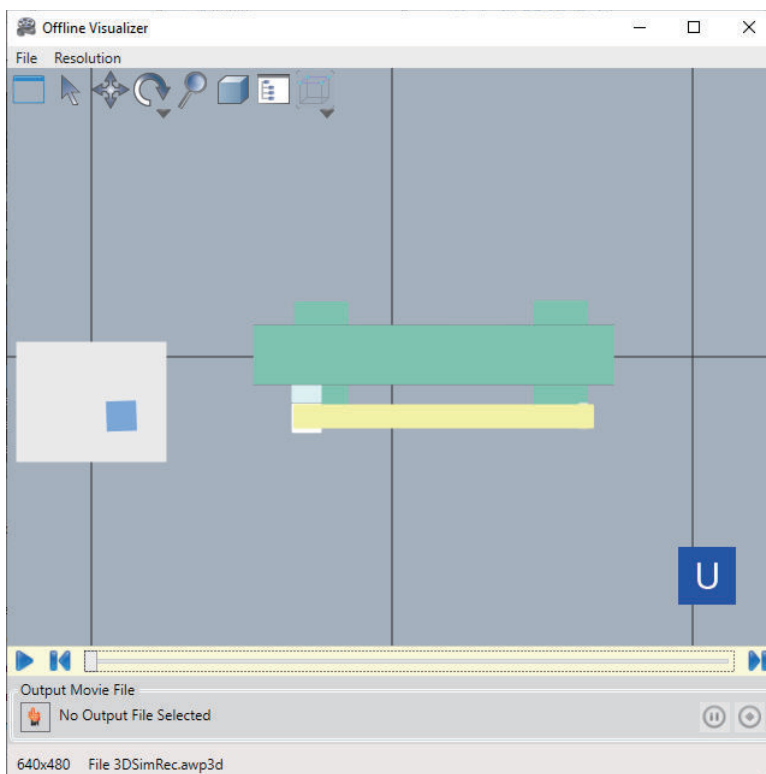
Playing Back Video

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.

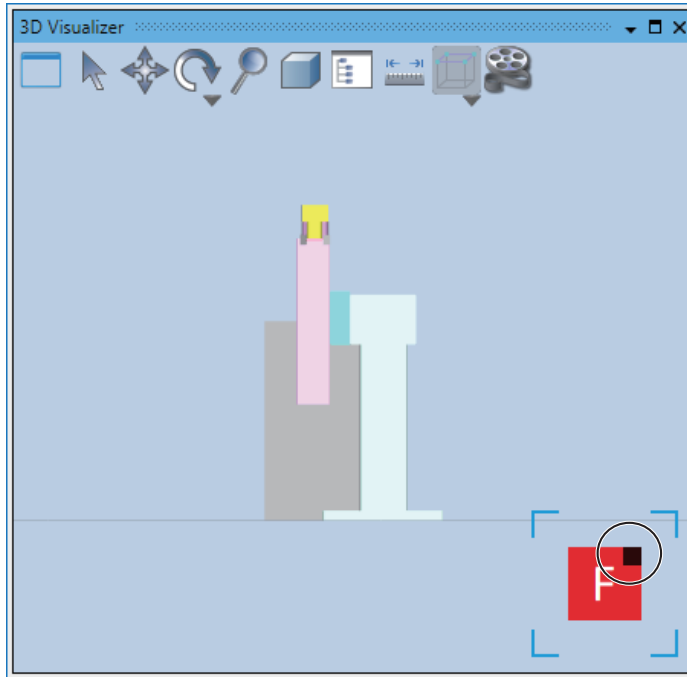
5-2-11 3D View Switching Tool

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.

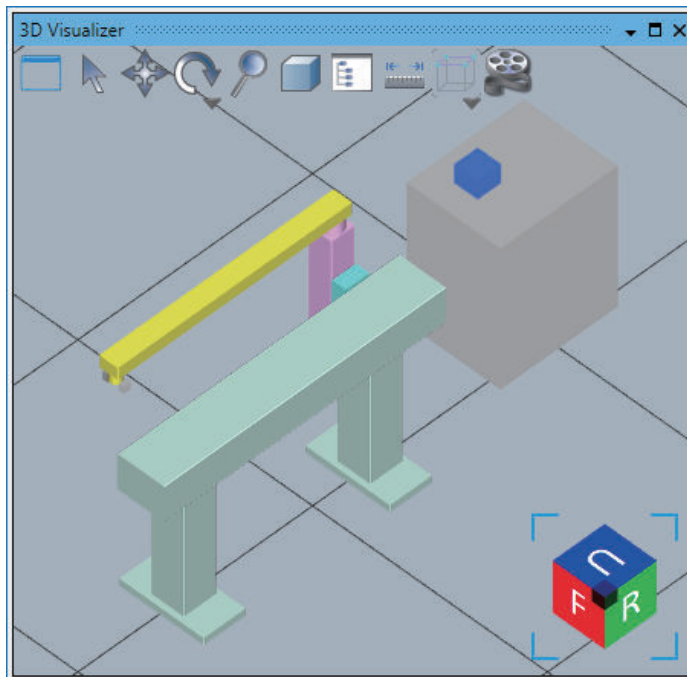
Configuration element	Name	Description
	Face	Represents a face. A face is indicated with one of the following symbols. F (Front): The front face when 3D shape data faces the zx plane R (Right): The right side face to the F face L (Left): The left side face to the F face B (Back): The face parallel to the F face U (Up): The upper orthogonal face to the F face D (Down): The lower orthogonal face to the F face
	Corner	Represents a corner.
	Edge	Represents an edge.

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.

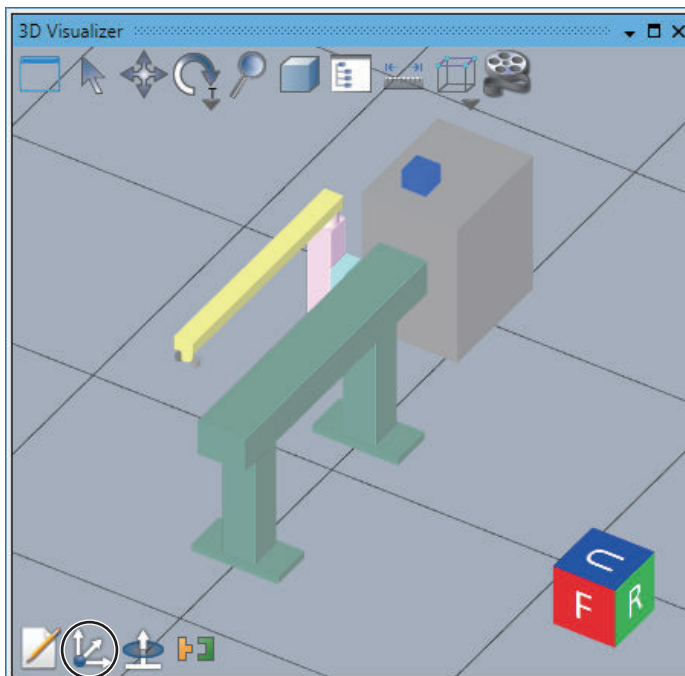
5-3 Operating 3D Shape Data in the 3D Visualizer

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

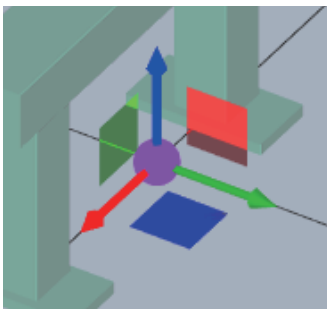
5-3-1 Moving 3D Shape Data

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.



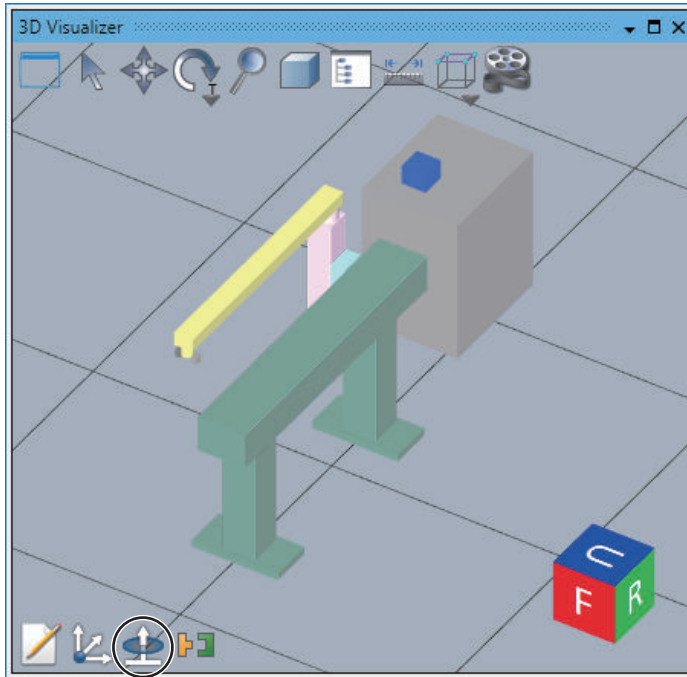
Additional Information

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

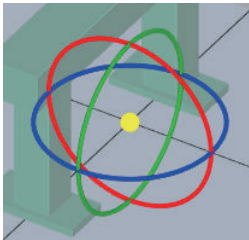
5-3-2 Rotating 3D Shape Data

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.



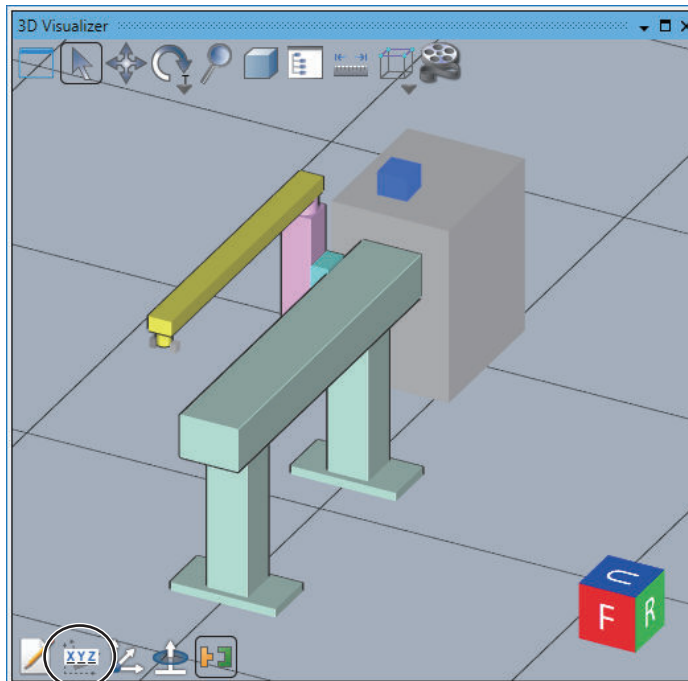
Additional Information

- 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 Work-space Position	Enter X, Y, and Z values that specify the position of the 3D shape data. <div style="text-align: center;"> <table border="1" style="margin: auto;"> <tr> <td>X</td> <td>Y</td> <td>Z</td> <td>mm</td> </tr> <tr> <td>-900.000</td> <td>-197.000</td> <td>600.000</td> <td></td> </tr> </table> </div>	X	Y	Z	mm	-900.000	-197.000	600.000	
X	Y	Z	mm							
-900.000	-197.000	600.000								

Icon	Name	Function
	Edit Work-space Orientation	Enter Yaw, Pitch, and Roll values that specify the orientation of the 3D shape data. Yaw Pitch Roll degree <input type="text" value="0.000"/> <input type="text" value="0.000"/> <input type="text" value="1.944"/>
	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. Box DX DY DZ mm <input type="text" value="100.000"/> <input type="text" value="100.000"/> <input type="text" value="80.000"/> Cylinder Radius Height mm <input type="text" value="100.000"/> <input type="text" value="100.000"/> Virtual Part Detection Sensor Length mm <input type="text" value="250.000"/> Belt Width Length mm <input type="text" value="250.000"/> <input type="text" value="3000.000"/>
	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	

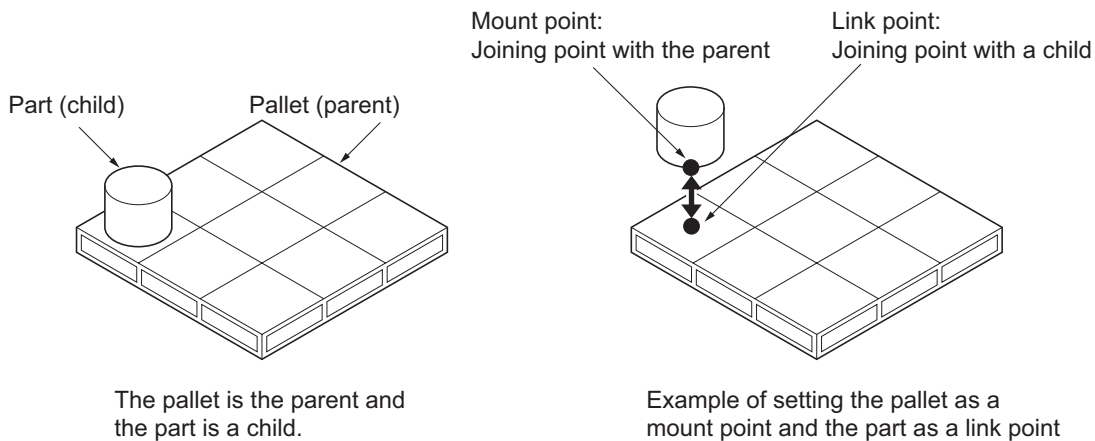
5-4 Positioning with a Mount Point or a Link Point

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.



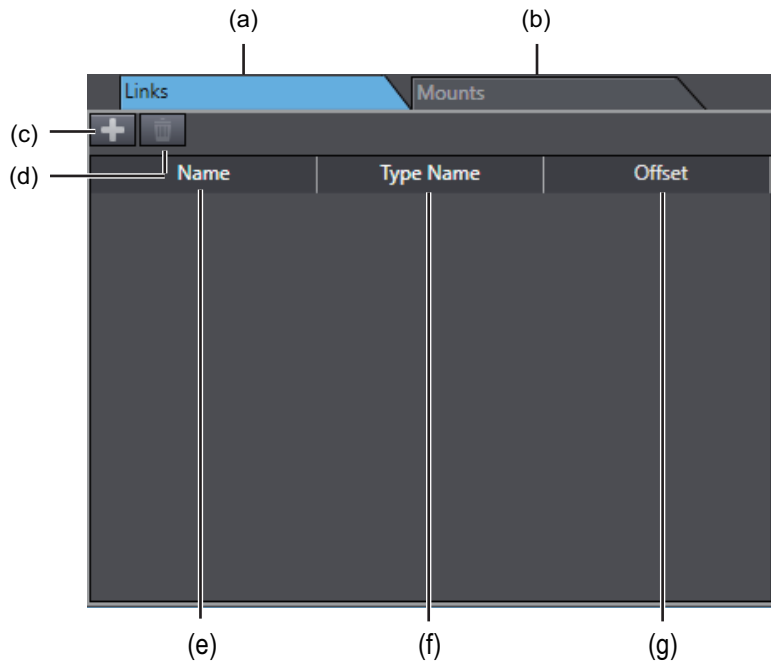
Additional Information

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.

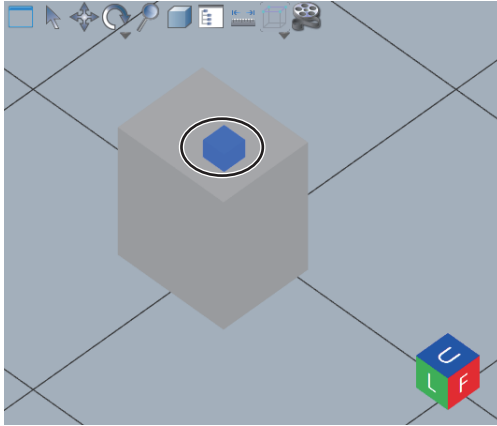


	Item	Description	Set value	Initial value
(a)	Links tab page	Use this tab page to set a link point for the 3D shape data. A link point is the point at which a child 3D shape data is joined to the parent 3D shape data.	None	Empty
(b)	Mounts tab page	Use this tab page to set a mount point for the 3D shape data. 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. You can set any name.	Text string	Connection Point
(f)	Type Name	The type of each mount point or link point is displayed. 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. 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.	For each, -1,000,000.000 to 1,000,000.000	0.000 for all

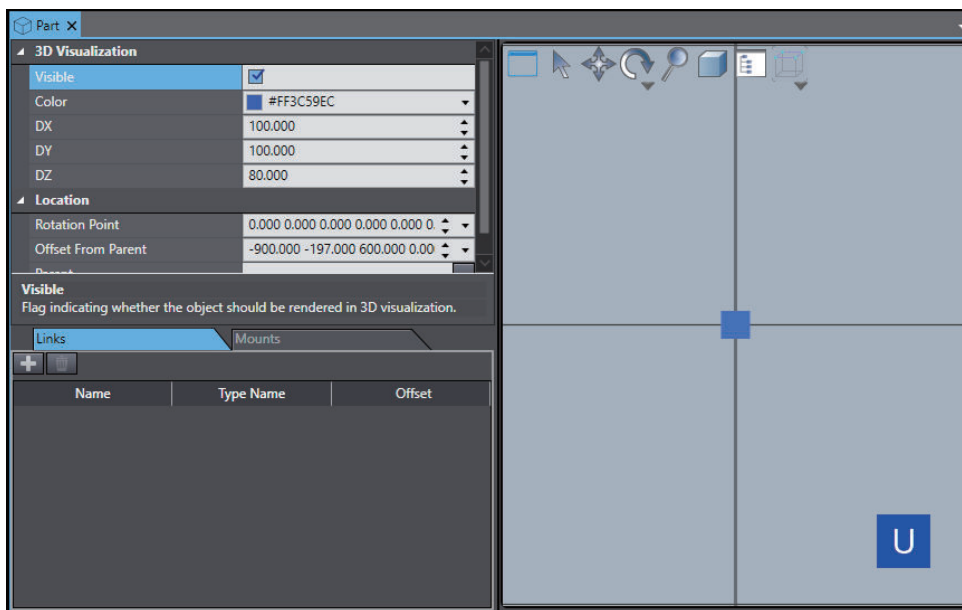
5-4-2 Setting a Mount Point

Use the following procedure to set a mount point.

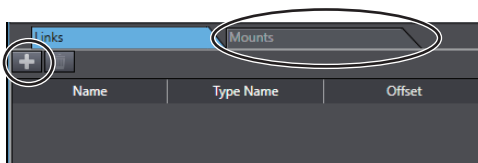
- 1 In the 3D Visualizer or Multiview Explorer, double-click the 3D shape data for which to set a mount point.



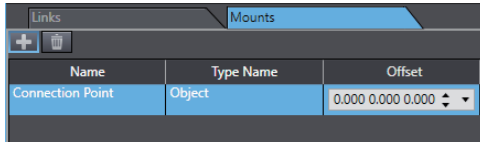
The setup tab page for the target 3D shape data is displayed.



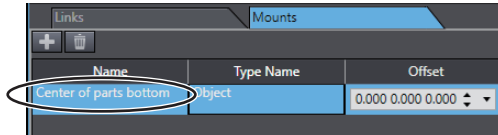
- 2 Click the **Mounts** tab, and then click the **Add** button.



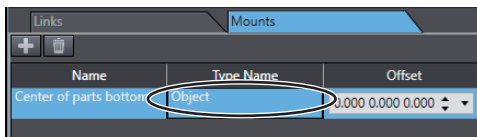
A row that contains a new mount point is added to the list.



- 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.

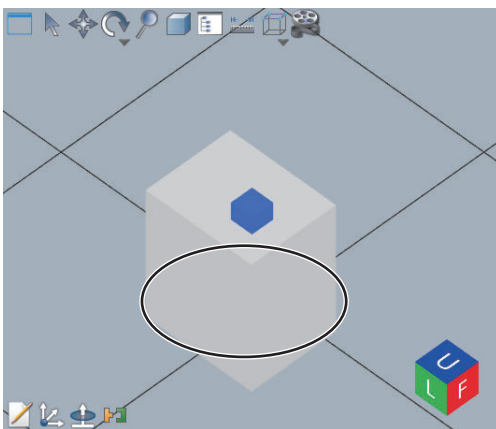


- 5** Set an offset for the mount point. Change the **Offset** value directly, or move the mount point in the 3D editing area. Refer to *5-4-4 Offset Setting Methods* on page 5-29 for the setting procedure in the 3D editing area.

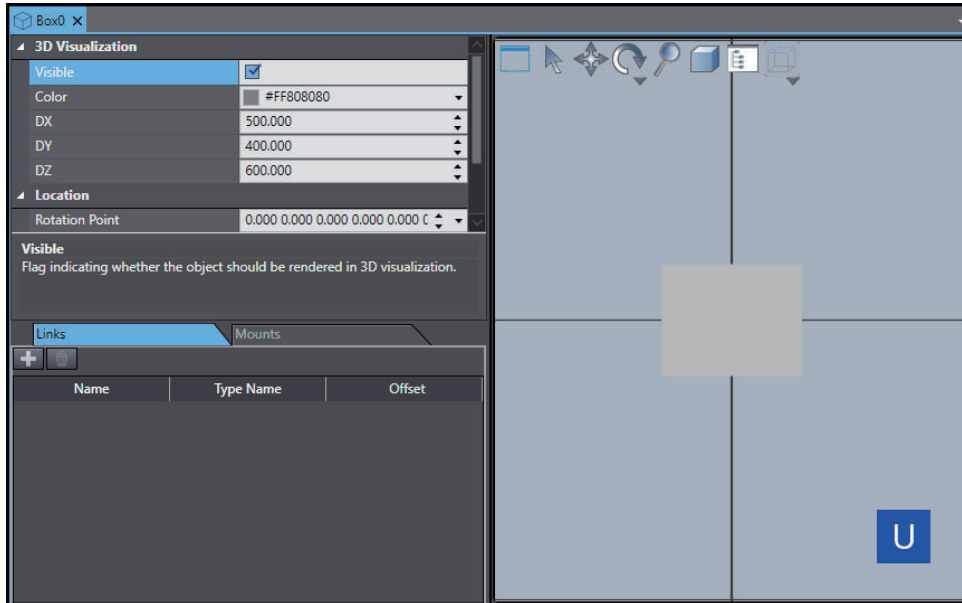
5-4-3 Setting a Link Point

Use the following procedure to set a link point.

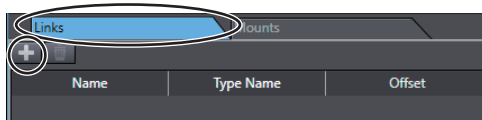
- 1** In the 3D Visualizer or Multiview Explorer, double-click the 3D shape data for which to set a link point.



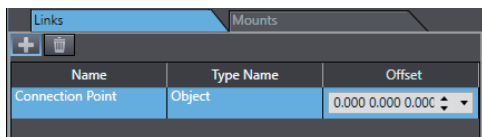
The setup tab page for the target 3D shape data is displayed.



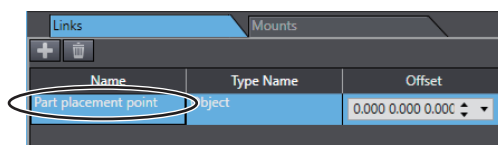
- 2 Click the Links tab, and then click the **Add** button.



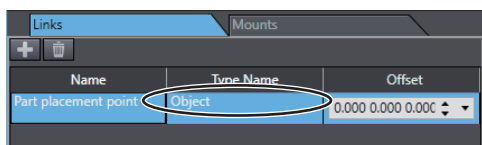
A row that contains a new link point is added to the list.



- 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 5-29 for the setting procedure in the 3D editing area.

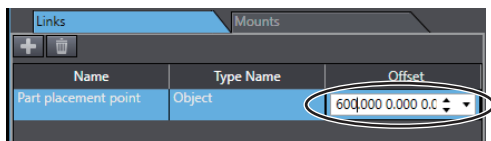
5-4-4 Offset Setting Methods

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

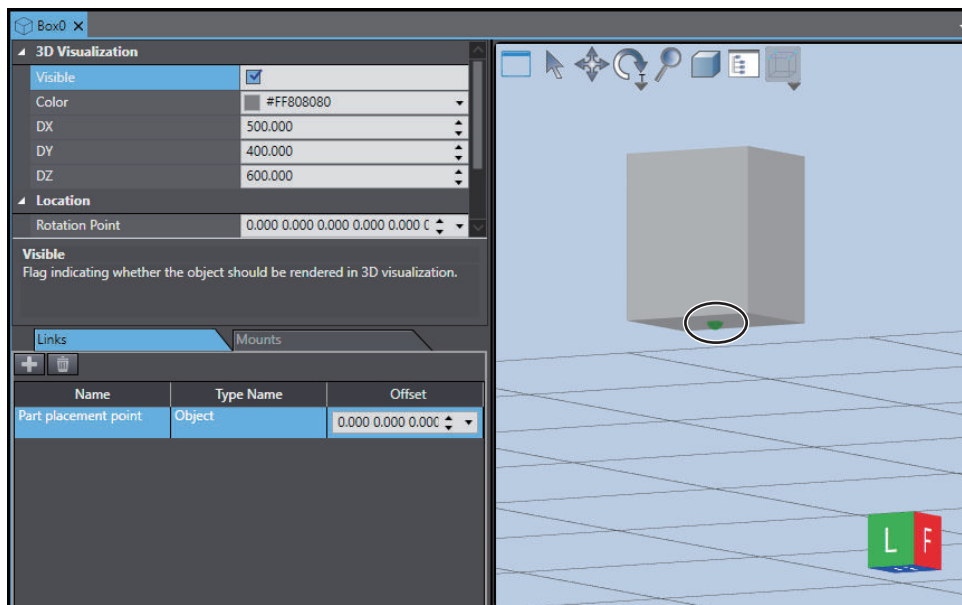
Enter the **offset** coordinate of each mount point or link point directly.



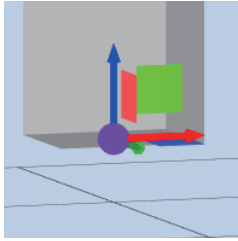
Using Icons to Set an Offset







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).

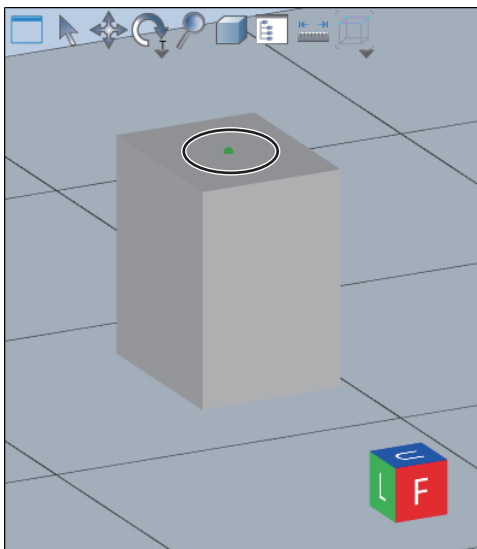


The following icon is displayed to move the link point.



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.

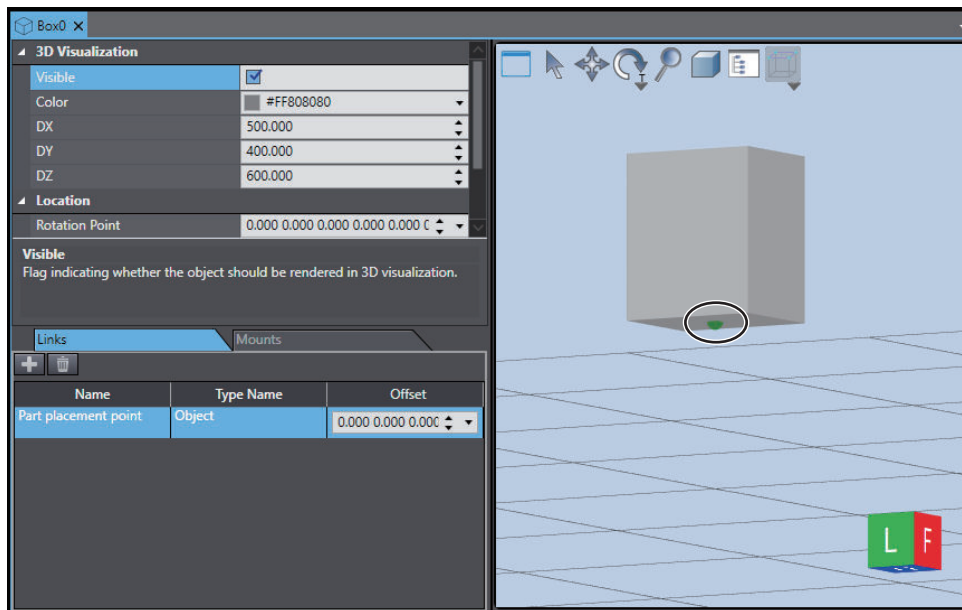
2 Drag the icon to move the link point to the target position.



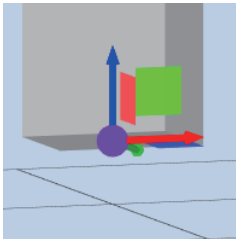
Using the Snap Function to Set an Offset

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.

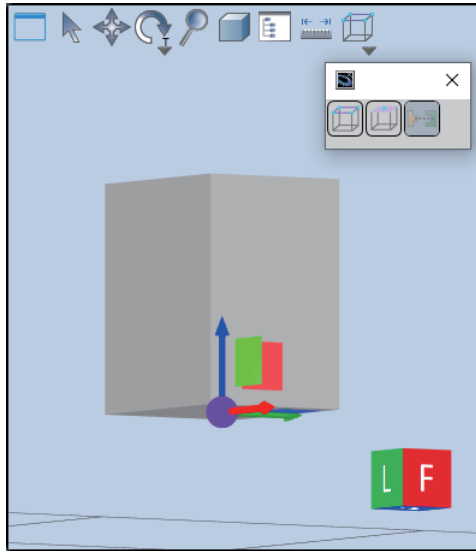
- 1 Click the green link point icon.
Normally, it is located at the origin (X=0, Y=0, Z=0).



The following icon is displayed to move the link point.



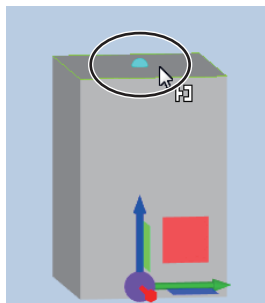
- 2 Click the Snap icon, and then click the Snap To Face button.



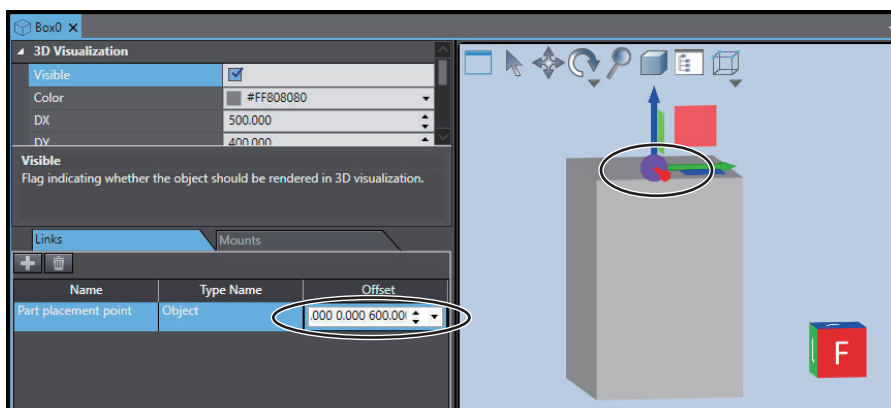
The icon changes.



- 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.



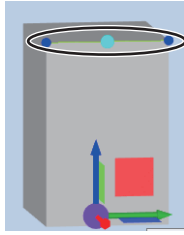
- 4 Click the light blue dot. The link point moves in the 3D editing area and an offset value is set in **Offset**.





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.

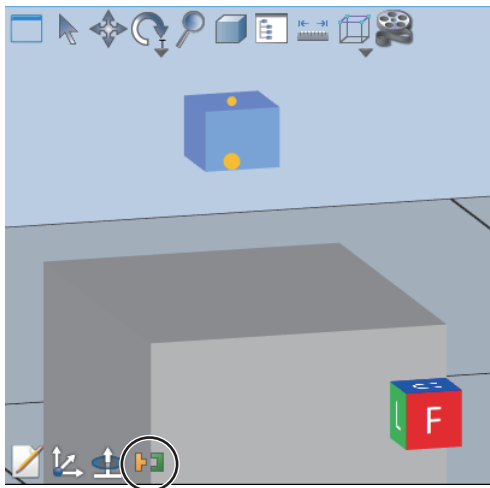


5-4-5 Using a Mount Point and a Link Point to Place 3D Shape Data

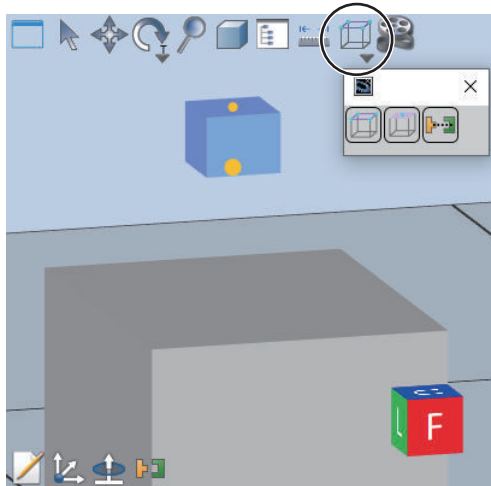
To place two sets of 3D shape data that has a mount point and a link point respectively, use the Snap function in the 3D Visualizer.

Select a mount point in the child 3D shape data, and then snap it to a link point in the parent 3D shape data.

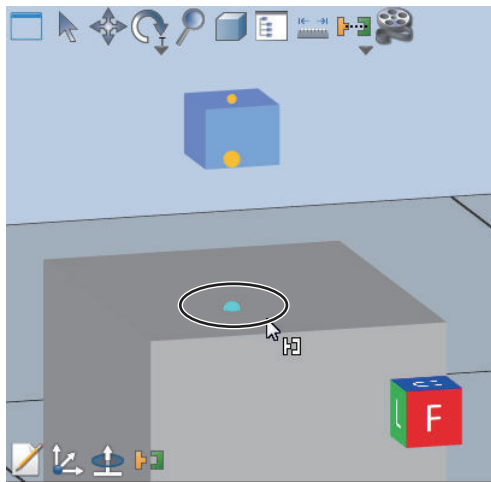
- 1 In the 3D Visualizer, select the 3D shape data to move.
- 2 Click the **Show/Hide Mount Points** icon to display mount points and select one in the 3D Visualizer.



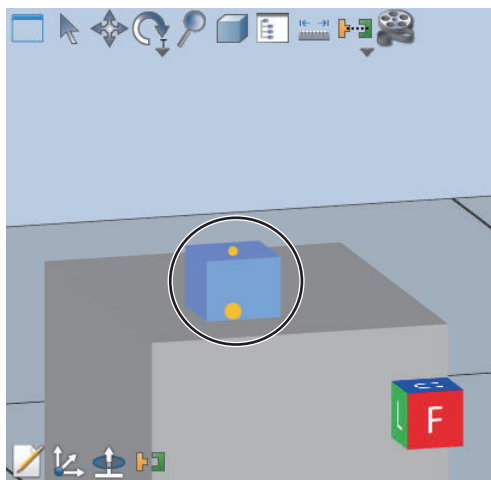
- 3 Click the **Snap** icon, and then select **Snap To Link**.



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	6-2
6-1-1	Outline of a Shape Script	6-2
6-1-2	Execution Timing and Period of Shape Scripts and Programs.....	6-3
6-2	Setting Mechanical Component	6-7
6-2-1	Generating Virtual Output Scripts of Limit Switch	6-7
6-3	Creating Operation Scripts for the Part	6-9
6-3-1	Adding Shape Scripts	6-9
6-3-2	Setting the Execution of Shape Scripts	6-9
6-4	Configuring the Operation Settings for the Virtual Part Detection Sensor	6-13
6-5	Shape Script Editor	6-15
6-5-1	Shape Script Editor Window	6-15
6-5-2	Shape Script Programming	6-18

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

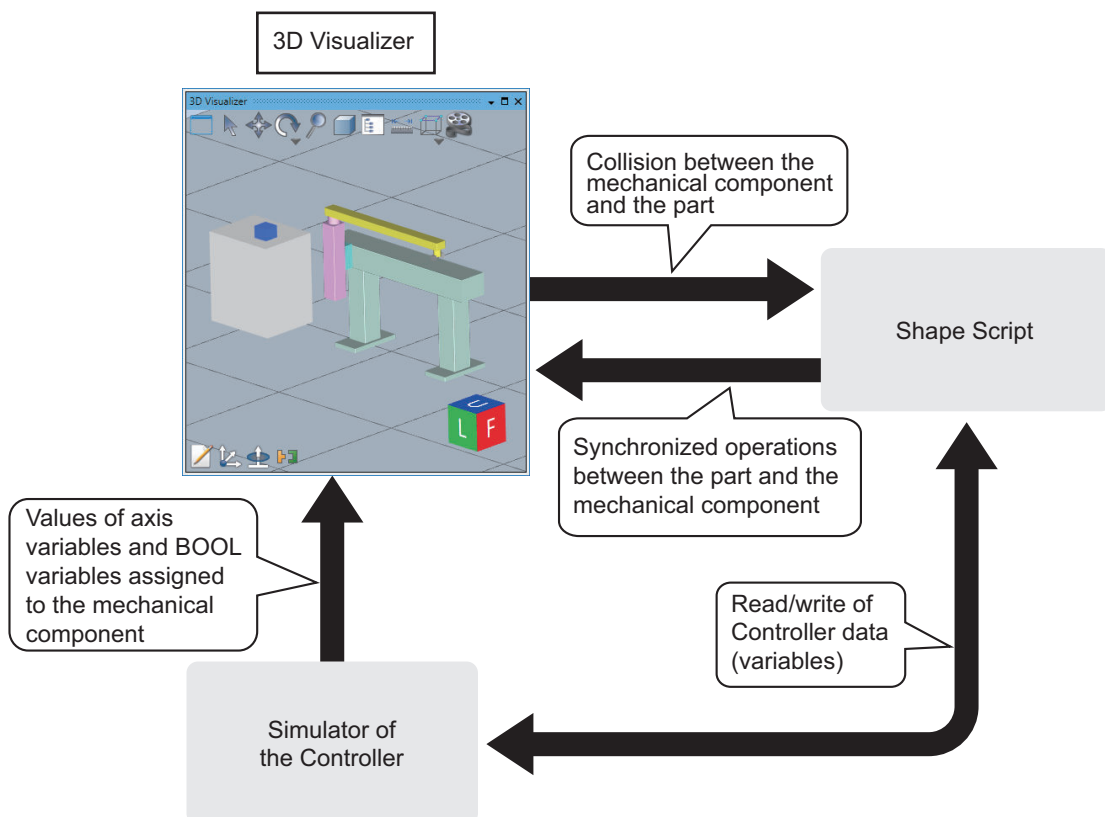
6-1-1 Outline of a Shape Script

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 6-9 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 on 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.

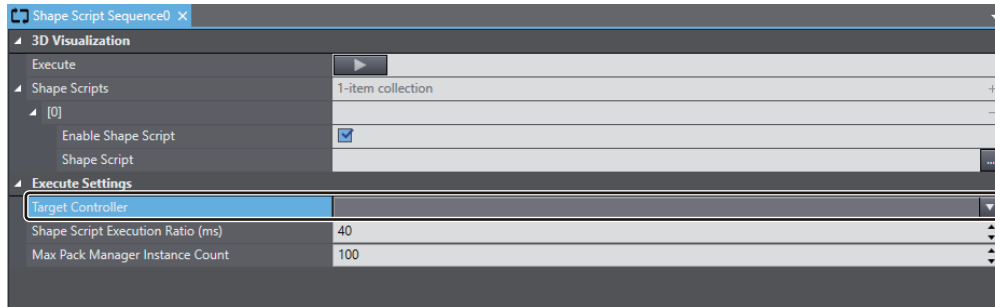
Item	Asynchronous execution	Synchronous execution
Execution speed of the Simulator of the Controller	Fast	Slow
Probability of missing collisions	Higher than synchronous execution	Lower as the execution count decreases
Timing of sending and receiving signals and variables between the Simulator of the 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 <i>Shape Script Execution Ratio</i> setting for the Shape Script Sequence. 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 <i>Execution Count</i> setting for the Shape Script Sequence. 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 an execution in which the Simulator of the Controller and a Shape Script run independently of each other without being synchronized. This is faster in the execution speed of the Simulator, but inaccurate in the timing of sending and receiving signals each other and the timing of detecting collisions between 3D shape data on the 3D Visualizer, because the Simulator of the Controller and the Shape Script do not wait for each other's completion of 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.

```
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);

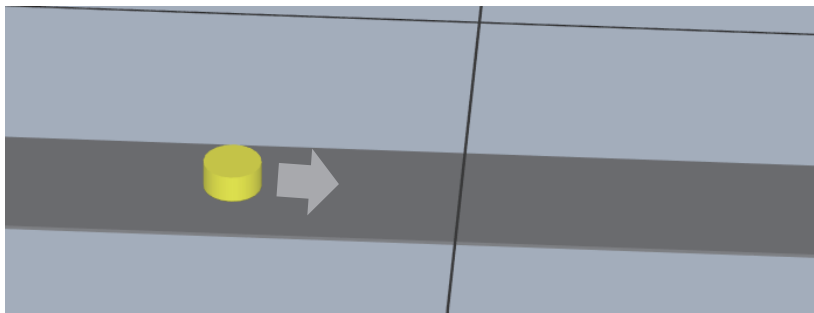
        this.previousTime = currentTime;
    }
    this.InitializeObject(renderInfoList);
}
```

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.



If you set the execution interval to 50 ms, the conveyor 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 when the execution interval is 50 ms. During this 20 mm period, no collision is detected even if the part collides with an obstacle.

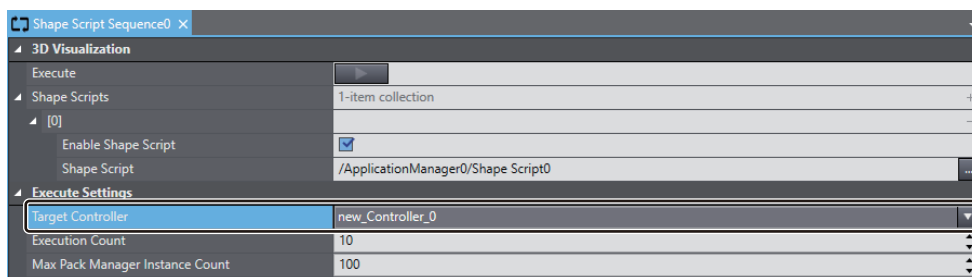
In summary, setting a shorter execution interval provides smoother drawing and is more likely to detect collisions with obstacles. However, the greater the execution count, the greater the load on the computer, and the execution of the Simulator of the Controller may become slower.

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.

Synchronous Execution and Execution Period of Shape Scripts and Programs

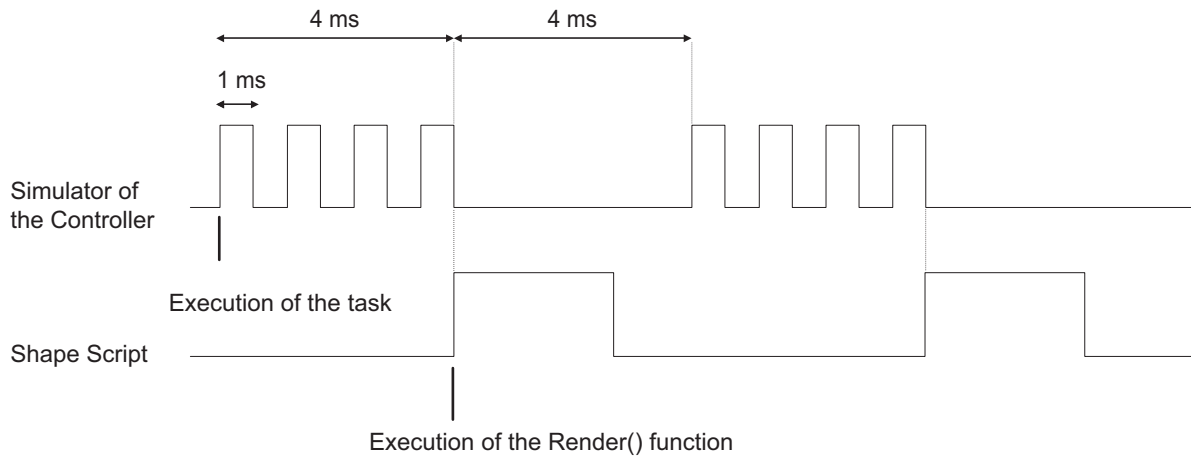
Synchronous execution refers to an execution in which the execution time of the Simulator of the Controller and that of a Shape Script are managed to achieve synchronicity. A Shape Script is executed after completion of the task that is running in the Simulator of the Controller. The Simulator of the 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 execution 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.

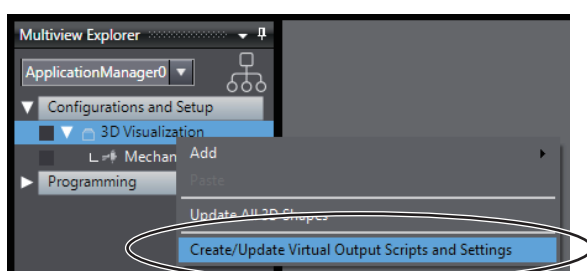
Refer to 4-3-5 *Mechanical Component Settings* on page 4-24 for details on the settings.

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

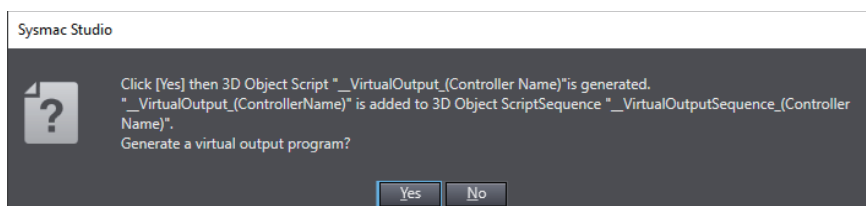
For specific mechanical components, you can automatically create operation scripts that reproduce the operations of a limit switch that the mechanical components perform in a 3D simulation.

Mechanical Component	Type of virtual output	Description
<ul style="list-style-type: none"> Air cylinder (Single solenoid type) 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.
<ul style="list-style-type: none"> Robot tool (Parallel switching 2-finger type chuck/single solenoid type) 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.

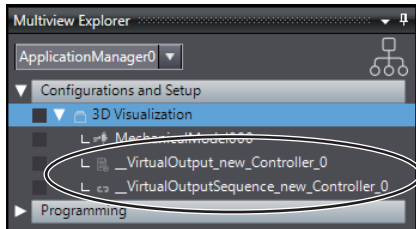


A script generation/update confirmation dialog box is displayed.



- 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 *7-1 Operating Procedures for a 3D Simulation* on page 7-2 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**

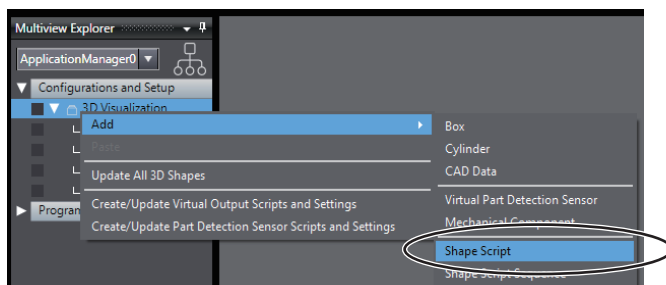
6-3 Creating Operation Scripts for the Part

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.

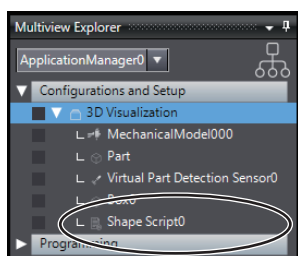
6-3-1 Adding Shape Scripts

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**.



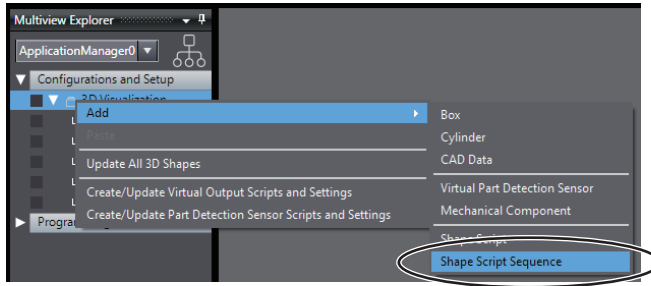
- 2 Double-click **Shape Script0**. Or, right-click and select **Edit** from the menu. The Shape Script Editor window is displayed. In the Shape Script Editor, create programs that define the operations of the part. Refer to 6-5 *Shape Script Editor* on page 6-15 for details on the Shape Script Editor window.

6-3-2 Setting the Execution of Shape Scripts

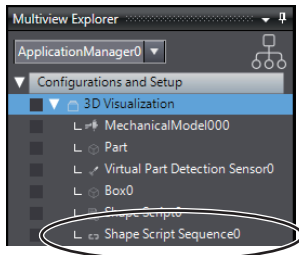
To execute the Shape Scripts that define the operations of the part, assign them to a Shape Script Sequence.

Use the following procedure to execute the Shape Scripts.

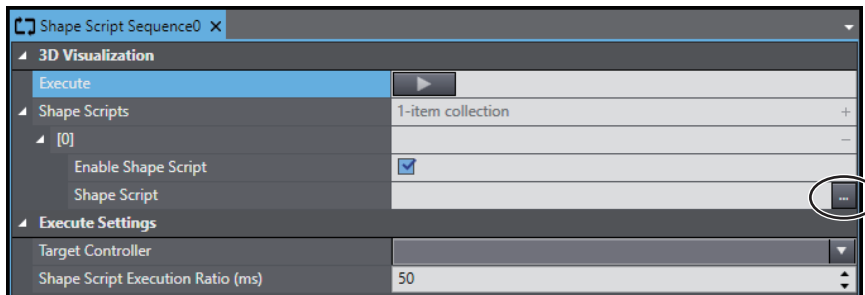
- 1 Right-click **3D Visualization** under **Configurations and Setup** and select **Add - Shape Script Sequence** from the menu.



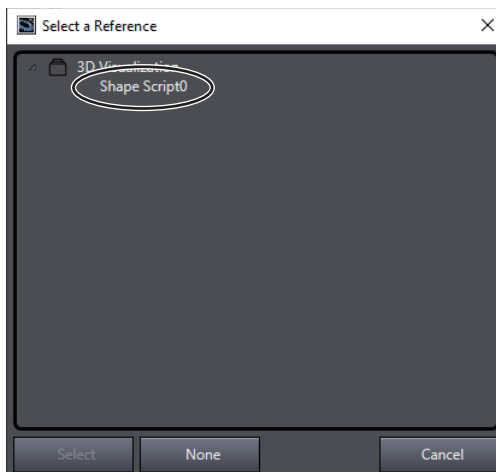
Shape Script Sequence0 is registered and displayed under 3D Visualization.



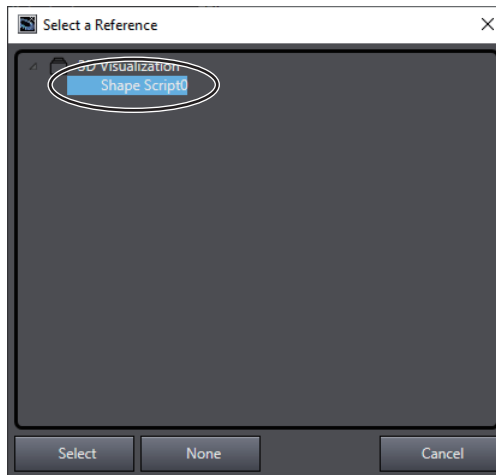
- 2** Double-click the Shape Script Sequence.
The Shape Script Sequence setup tab page is displayed.
- 3** Click the Browse button for the Shape Script.



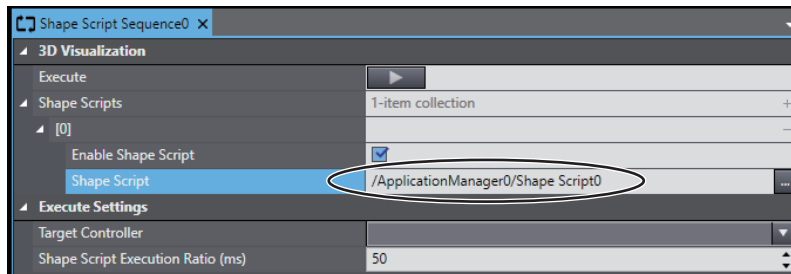
A list of registered Shape Scripts is displayed.



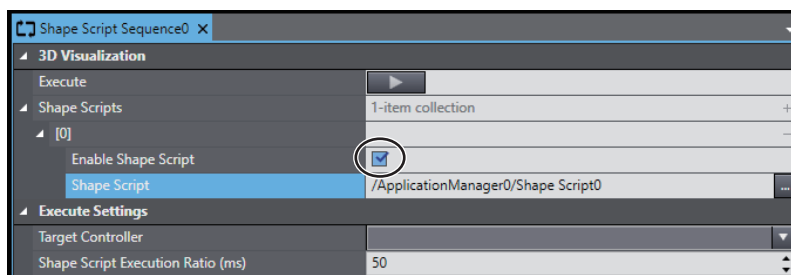
- 4** Select the Shape Scripts to execute, and then click the **Select** button.



The Shape Scripts are assigned to the Shape Script Sequence.

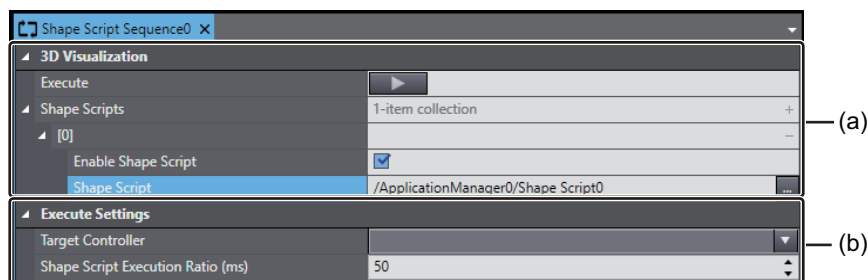


5 Select the **Enable Shape Script** check box.



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 that are 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

*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.

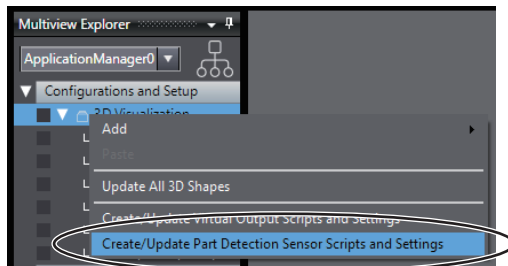
*2. **Execution Count** is displayed when the target Controller is selected, while **Shape Script Execution Ratio (ms)** is displayed when the target Controller is not selected.

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

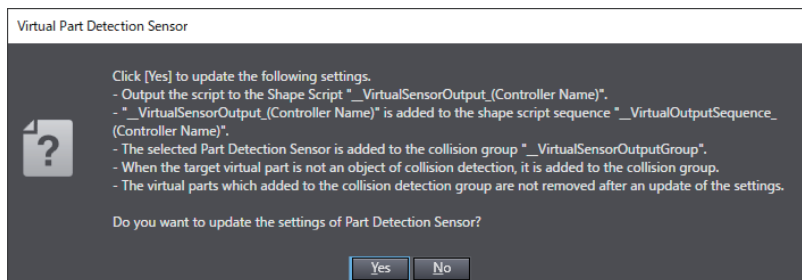
Define the operations that the Virtual Part Detection Sensor performs to detect the part in a *Virtual Part Detection Sensor script*.

Use the following procedure to add the Virtual Part Detection Sensor script.

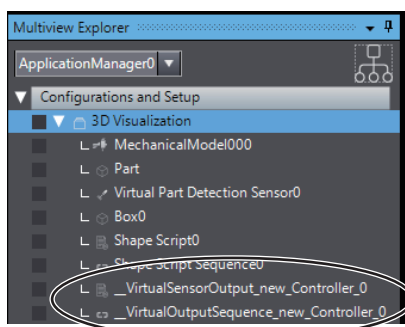
- 1 Right-click **3D Visualization** under **Configurations and Setup** in the Multiview Explorer and select **Create/Update Part Detection Sensor Scripts and Settings** from the menu.



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 *7-1 Operating Procedures for a 3D Simulation* on page 7-2 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.

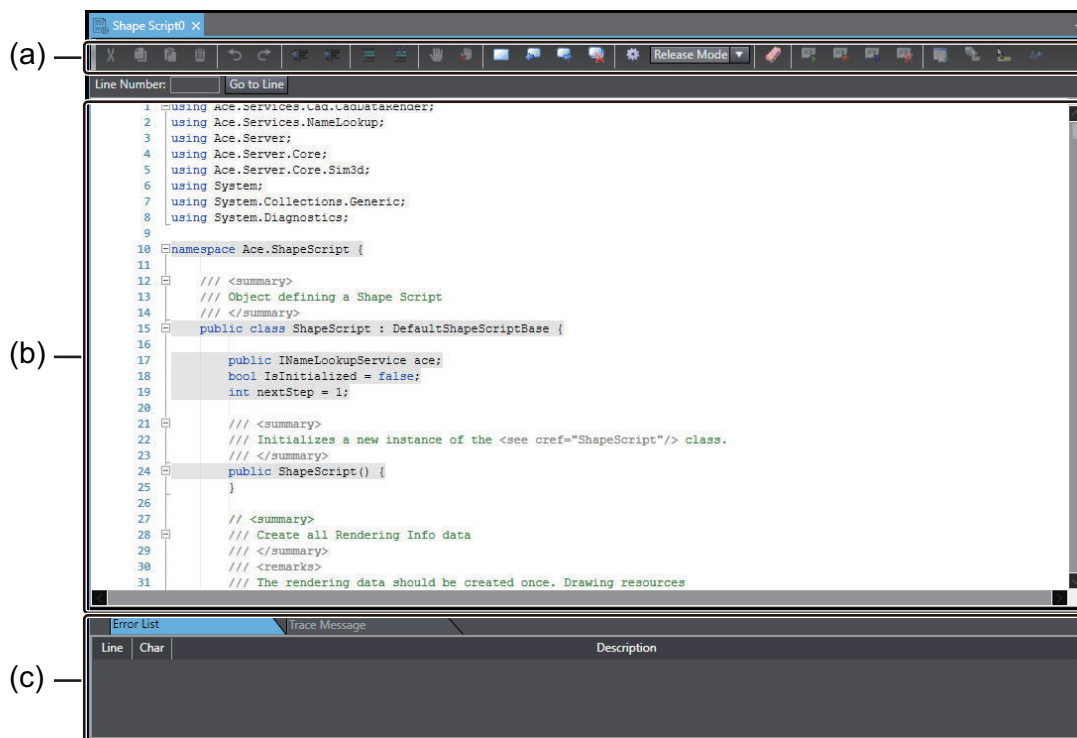
As for the language specifications of Shape Scripts, the editor uses Microsoft's C# version 6.0. Its variables, data types, constructs, functions, operators, and so on conform to the language specifications of the C# language.

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 Functions Used in Shape Scripts* 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.


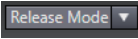















	Item	Description
(a)	Toolbar	The function buttons used for script creation are displayed. Refer to <i>Toolbar in the Shape Script Editor</i> on page 6-15 for details.
(b)	Editor	Use this editor to create scripts. Refer to <i>6-5-2 Shape Script Programming</i> on page 6-18 for details.
(c)	Tab page	Errors or trace messages are displayed. Refer to <i>Tab Pages of the Shape Script Editor</i> on page 6-17 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 Selection	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 7-2-3 <i>Trace Statement</i> on page 7-9 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 tooltip.
	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.

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.

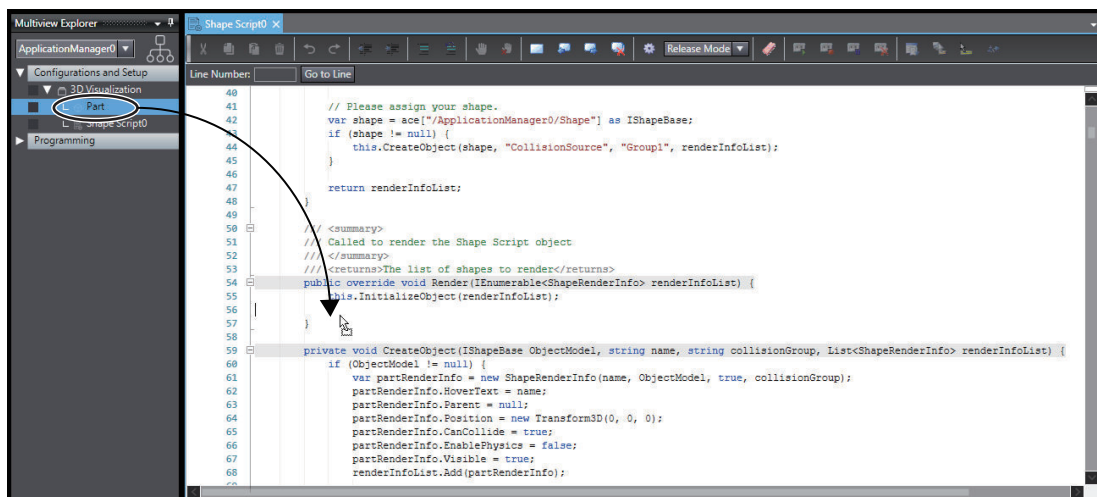
Creating a Variable Declaration for Variables that Represent 3D Shape Data in a Shape Script

To use variables that represent 3D shape data such as the part in a Shape Script, you must declare the variables.

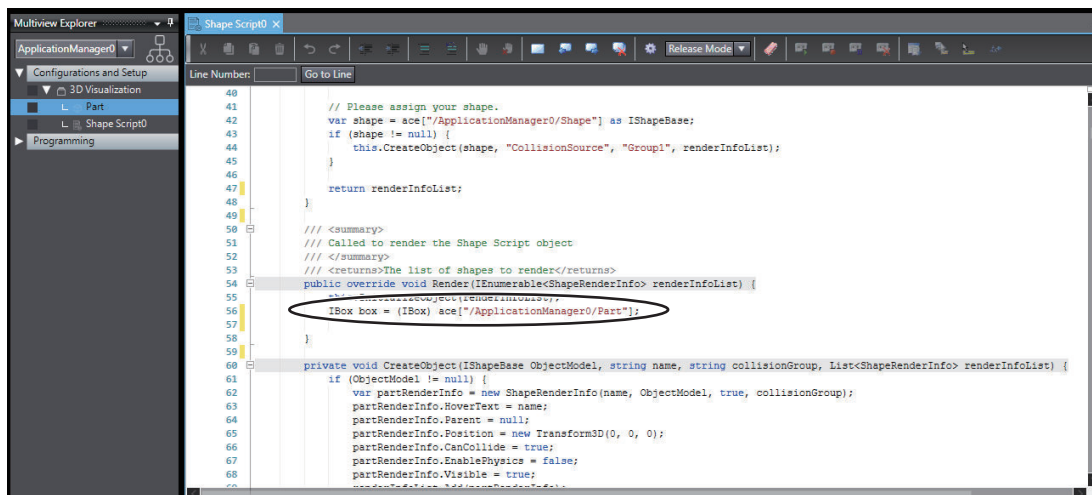
You can drag a 3D shape data item from the Multiview Explorer into the Shape Script Editor to create a variable declaration.

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.

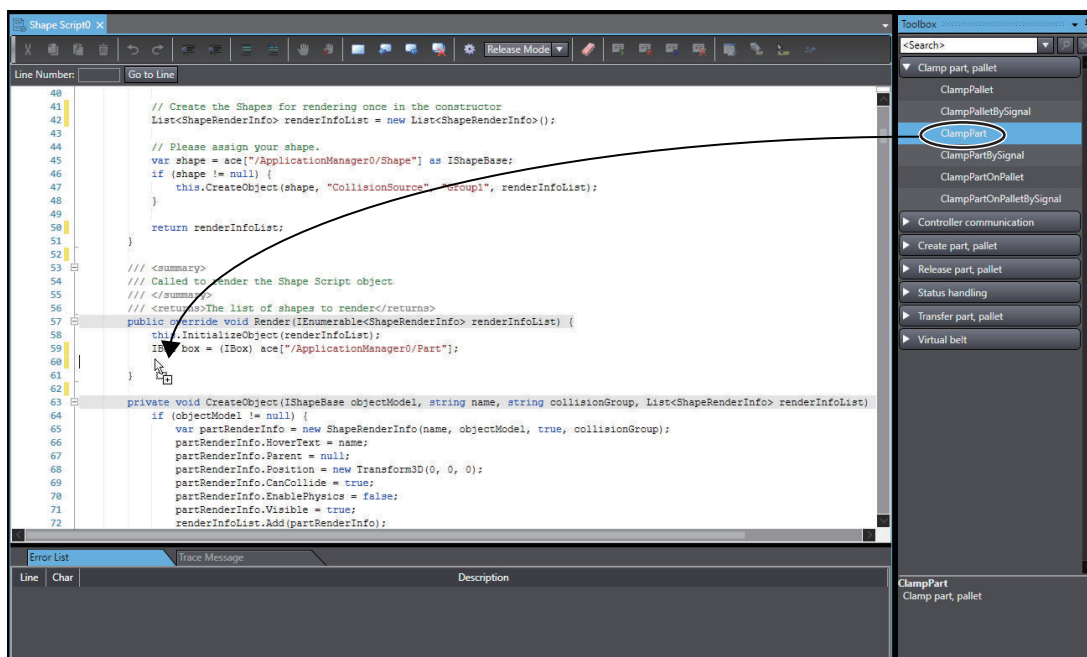


Inserting a Function from the Toolbox into a Shape Script

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.

```

37
38 // Create the Shapes for rendering once in the constructor
39 List<ShapeRenderInfo> renderInfoList = new List<ShapeRenderInfo>();
40
41 // Please assign your shape.
42 var shape = ace["/ApplicationManager0/Shape"] as IShapeBase;
43 if (shape != null) {
44     this.CreateObject(shape, "CollisionSource", "Group1", renderInfoList);
45 }
46
47 return renderInfoList;
48
49
50 /// <summary>
51 /// Called to render the Shape Script object
52 /// </summary>
53 /// <returns>The list of shapes to render</returns>
54 public override void Render(IEnumerable<ShapeRenderInfo> renderInfoList) {
55     this.InitializeObject(renderInfoList);
56     //this.ClampPart(ace["/ApplicationManager0/Part"]);
57     this.ClampPart(int stepId, IVisualizable robotTool, IShapeBase partModel, IVisualizable locationModel, renderInfoList);
58 }
59
60 private void CreateObject(IShapeBase ObjectModel, string name, string collisionGroup, List<ShapeRenderInfo> renderInfoList) {
61     if (ObjectModel != null) {
62         var partRenderInfo = new ShapeRenderInfo(name, ObjectModel, true, collisionGroup);
63         partRenderInfo.HoverText = name;
64         partRenderInfo.Parent = null;
65         partRenderInfo.Position = new Transform3D(0, 0, 0);
66         partRenderInfo.CanCollide = true;
67         partRenderInfo.EnablePhysics = false;
68         partRenderInfo.Visible = true;
69         renderInfoList.Add(partRenderInfo);

```

In the inserted function, specify appropriate variables for arguments.

This completes the insertion of the function.

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



Executing a 3D Simulation

This section describes the procedures to execute a 3D simulation and the operation check methods.

7-1	Operating Procedures for a 3D Simulation.....	7-2
7-1-1	Executing a Controller Simulation	7-2
7-1-2	Executing the Operation Script for the Virtual Part Detection Sensor and the Virtual Output Script for the Mechanical Component	7-3
7-1-3	Executing the Shape Scripts for the Part	7-3
7-1-4	Checking Operations in the 3D Visualizer	7-4
7-1-5	Executing Operations for a 3D Simulation at Once.....	7-4
7-2	Debugging a Shape Script.....	7-7
7-2-1	Breakpoint	7-7
7-2-2	Step Execution	7-8
7-2-3	Trace Statement	7-9
7-2-4	Takt Time Measurement	7-9
7-3	Collision Detection Function.....	7-13
7-3-1	Collision Detection Target.....	7-13
7-3-2	Collision Detection Setting Procedure.....	7-13
7-3-3	How to Check Detected Collisions	7-16
7-3-4	How to Detect Collisions with Shape Scripts.....	7-17
7-3-5	Collision Filter Model Settings	7-24

7-1 Operating Procedures for a 3D Simulation

This section describes the operating procedures to execute a 3D simulation.

You can execute a 3D simulation in the following workflow. Display the 3D Visualizer in advance.

1. Executing a Controller Simulation
2. Executing the Operation Script for the Virtual Part Detection Sensor and the Virtual Output Script for the Mechanical Component
3. Executing the Shape Scripts for the Part
4. Checking Operations in the 3D Visualizer

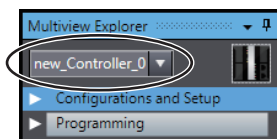


Additional Information

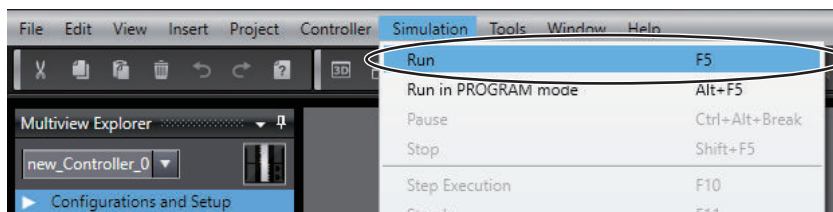
Refer to *Sysmac Studio Version 1 Operation Manual (Cat. No. W504)* for the basic simulation functions and operating procedures for the Controller.

7-1-1 Executing a Controller Simulation

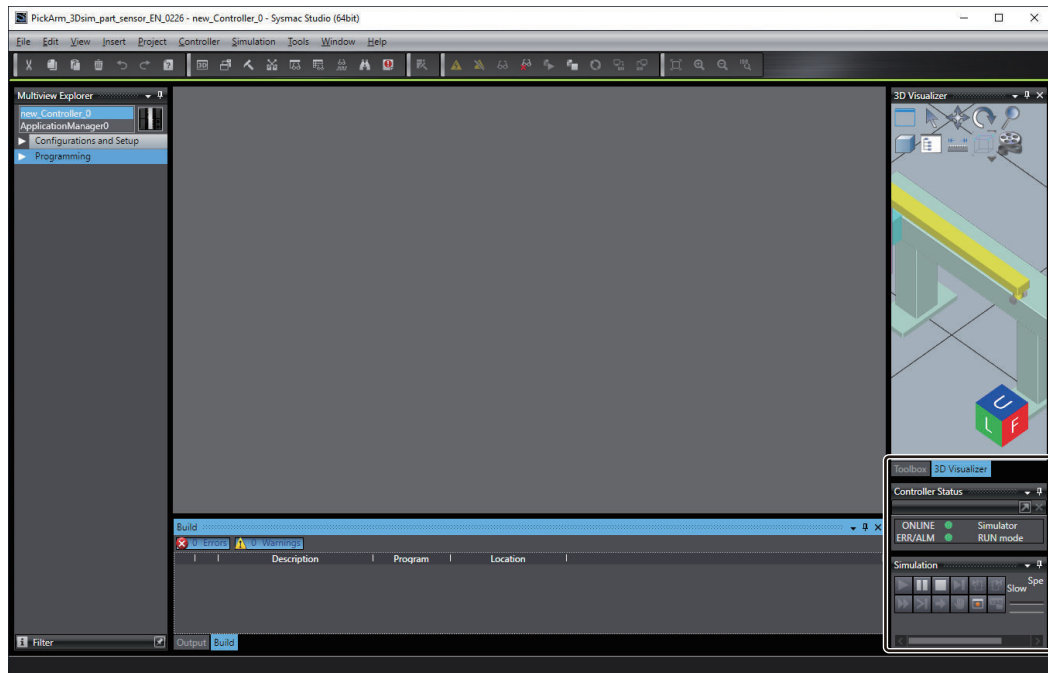
- 1 Select a Controller in the device list.
The Controller is selected as the device.



- 2 Select **Run** from the **Simulation** menu.

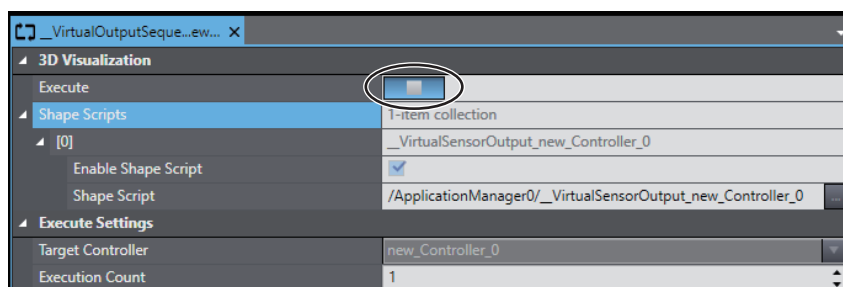


A connection is established with Simulator of the Controller.



7-1-2 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.



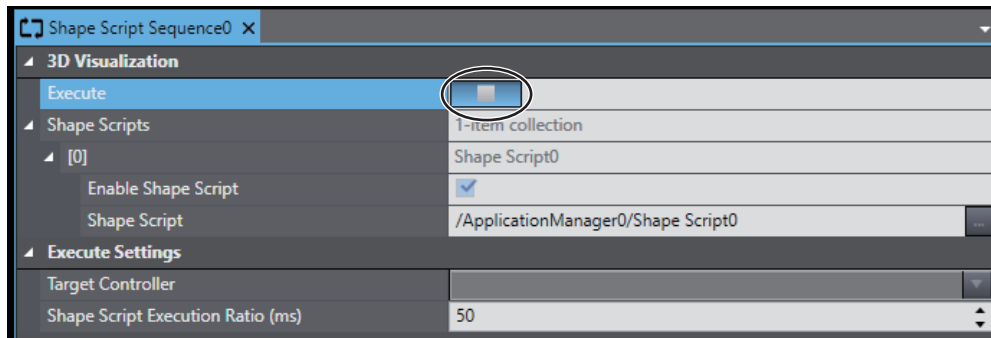
7-1-3 Executing the Shape Scripts for the Part

- 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.



7-1-4 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 *7-2 Debugging a Shape Script* on page 7-7 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.

7-1-5 Executing Operations for a 3D Simulation at Once

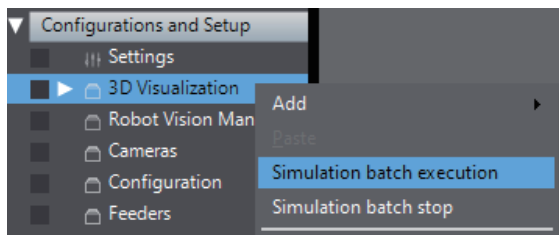
Although 3D simulation requires that you execute the following operations in order, it is also possible to have these operations executed automatically. This section describes how to execute operations required for a 3D simulation at once.

Order	Operation	Reference
1	Executing a Controller Simulation	<i>7-1-1 Executing a Controller Simulation</i> on page 7-2
2	Executing the Operation Script for the Virtual Part Detection Sensor and the Virtual Output Script for the Mechanical Component	<i>7-1-2 Executing the Operation Script for the Virtual Part Detection Sensor and the Virtual Output Script for the Mechanical Component</i> on page 7-3

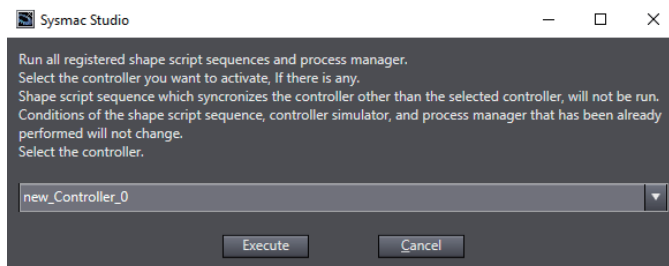
Order	Operation	Reference
3	Executing the Shape Scripts for the Part	7-1-3 Executing the Shape Scripts for the Part on page 7-3

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.



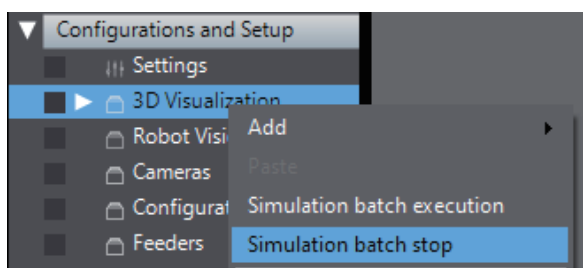
The Controller selection dialog box is displayed.



- 2 Select the Controller to execute a simulation, and then click the **Execute** button. A connection is established with the Simulator of the Controller. Then, executions of the operation script for the Virtual Part Detection Sensor, the virtual output script for a mechanical component, and the Shape Script start.

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. Then, executions of the operation script for the Virtual Part Detection Sensor, the virtual output script for a mechanical component, and the Shape Script stop.

7-2 Debugging a Shape Script

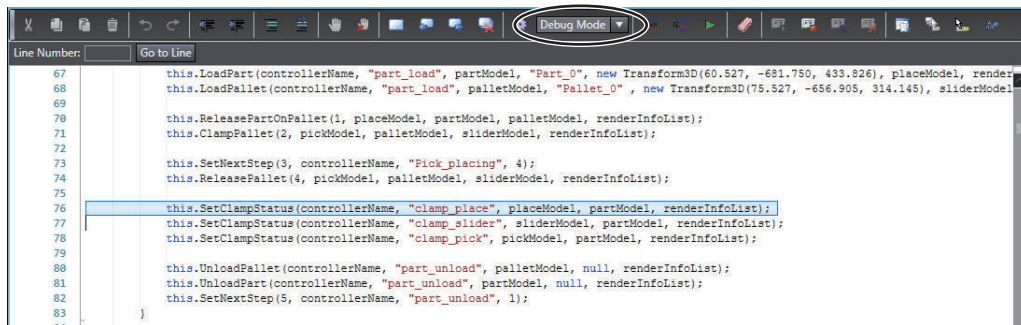
To debug a Shape Script, use the debugging functions provided in the Shape Script Editor. Double-click the target Shape Script in the Multiview Explorer to display the Shape Script Editor in advance.

7-2-1 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*.

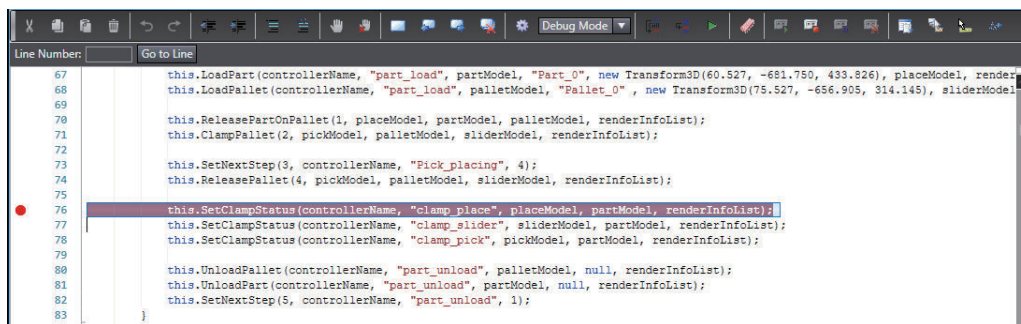


In DEBUG mode, you can click the **Toggle Breakpoint** and **Clear Breakpoints** buttons.



- 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.

```

73     this.SetNextStep(3, controllerName, "Pick_placing", 4);
74     this.ReleasePallet(4, pickModel, palletModel, sliderModel, renderInfoList);
75
76     this.SetClampStatus(controllerName, "Clamp_place", placeModel, partModel, renderInfoList);
77     this.SetClampStatus(controllerName, "Clamp_slider", sliderModel, partModel, renderInfoList);
78     this.SetClampStatus(controllerName, "Clamp_pick", pickModel, partModel, renderInfoList);

```

```

73     this.SetNextStep(3, controllerName, "Pick_placing", 4);
74     this.ReleasePallet(4, pickModel, palletModel, sliderModel, renderInfoList);
75
76     this.SetClampStatus(controllerName, "Clamp_place", placeModel, partModel, renderInfoList);
77     this.SetClampStatus(controllerName, "Clamp_slider", sliderModel, partModel, renderInfoList);
78     this.SetClampStatus(controllerName, "Clamp_pick", pickModel, partModel, renderInfoList);

```

- 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.

```

73     this.SetNextStep(3, controllerName, "Pick_placing", 4);
74     this.ReleasePallet(4, pickModel, palletModel, sliderModel, renderInfoList);
75
76     this.SetClampStatus(controllerName, "Clamp_place", placeModel, partModel, renderInfoList);
77     this.SetClampStatus(controllerName, "Clamp_slider", sliderModel, partModel, renderInfoList);
78     this.SetClampStatus(controllerName, "Clamp_pick", pickModel, partModel, renderInfoList);

```

7-2-2 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.

```

66     var controllerName = "new_Controller_0";
67     this.LoadPart(controllerName, "part_load", partModel, "Part_0", new Transform3D(60.527, -681.750, 433.826), placeModel, renderI
68     this.LoadPallet(controllerName, "part_load", palletModel, "Pallet_0", new Transform3D(75.527, -656.905, 314.145), sliderModel,
69

```

- 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.

```

66     var controllerName = "new_Controller_0";
67     this.LoadPart(controllerName, "part_load", partModel, "Part_0", new Transform3D(60.527, -681.750, 433.826), placeModel, renderI
68     this.LoadPallet(controllerName, "part_load", palletModel, "Pallet_0", new Transform3D(75.527, -656.905, 314.145), sliderModel,
69

```

- 3 Click the **Step Over** or **Step Into** button again and again. The execution position changes according to the execution result.

7-2-3 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:

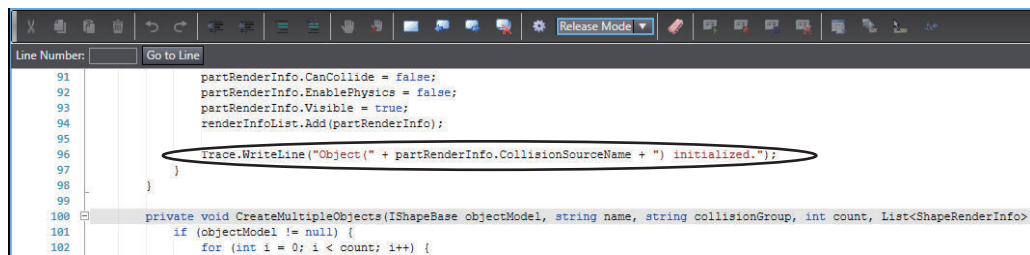
```
Trace.WriteLine (Contents);
```

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);
```



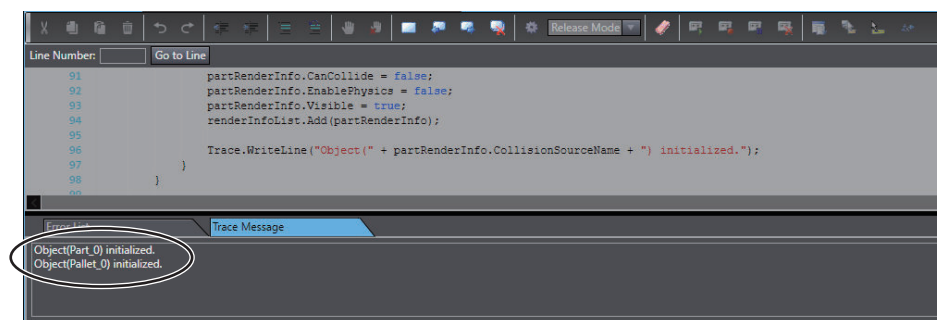
```

Line Number: 91 partRenderInfo.CanCollide = false;
92 partRenderInfo.EnablePhysics = false;
93 partRenderInfo.Visible = true;
94 renderInfoList.Add(partRenderInfo);
95
96 Trace.WriteLine("Object(" + partRenderInfo.CollisionSourceName + ") initialized.");
97
98 }
99
100 private void CreateMultipleObjects(IShapeBase objectModel, string name, string collisionGroup, int count, List<ShapeRenderInfo>
101 if (objectModel != null) {
102 for (int i = 0; i < count; i++) {

```

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.



```

Line Number: 91 partRenderInfo.CanCollide = false;
92 partRenderInfo.EnablePhysics = false;
93 partRenderInfo.Visible = true;
94 renderInfoList.Add(partRenderInfo);
95
96 Trace.WriteLine("Object(" + partRenderInfo.CollisionSourceName + ") initialized.");
97
98 }
99

```

Trace Message

```

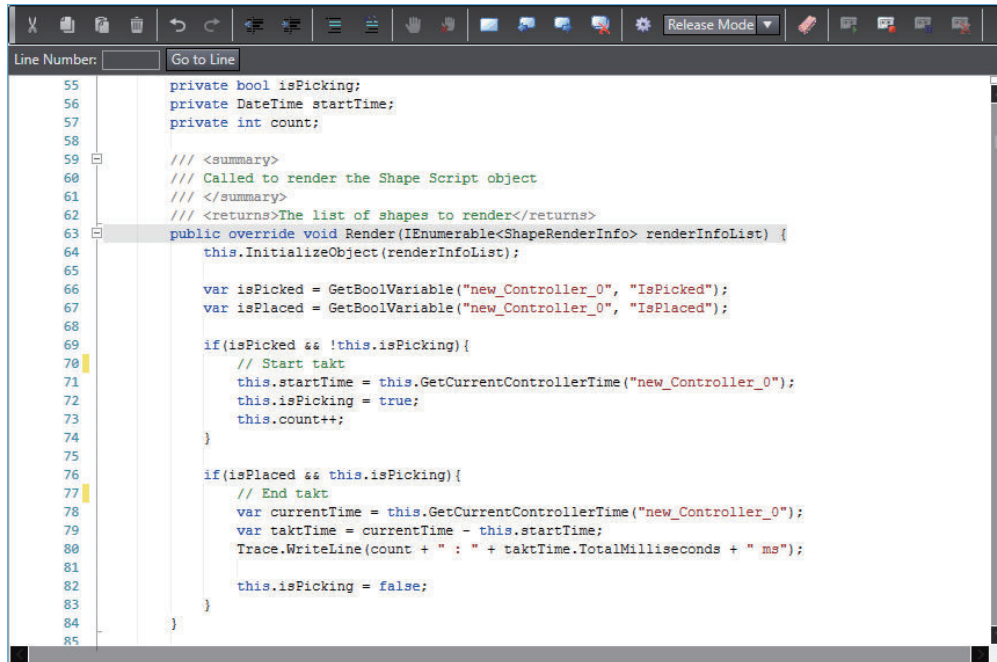
Object(Part_0) initialized.
Object(Pallet_0) initialized.

```

7-2-4 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.



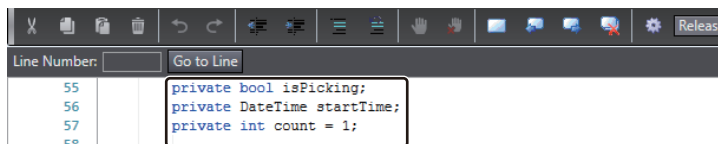
```

55     private bool isPicking;
56     private DateTime startTime;
57     private int count;
58
59     /// <summary>
60     /// Called to render the Shape Script object
61     /// </summary>
62     /// <returns>The list of shapes to render</returns>
63     public override void Render(IEnumerable<ShapeRenderInfo> renderInfoList) {
64         this.InitializeObject(renderInfoList);
65
66         var isPicked = GetBoolVariable("new_Controller_0", "IsPicked");
67         var isPlaced = GetBoolVariable("new_Controller_0", "IsPlaced");
68
69         if(isPicked && !this.isPicking){
70             // Start takt
71             this.startTime = this.GetCurrentControllerTime("new_Controller_0");
72             this.isPicking = true;
73             this.count++;
74         }
75
76         if(isPlaced && this.isPicking){
77             // End takt
78             var currentTime = this.GetCurrentControllerTime("new_Controller_0");
79             var taktTime = currentTime - this.startTime;
80             Trace.WriteLine(count + " : " + taktTime.TotalMilliseconds + " ms");
81
82             this.isPicking = false;
83         }
84     }
85

```

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.



```

55     private bool isPicking;
56     private DateTime startTime;
57     private int count = 1;
58

```

2 Get the Controller's variables in the Render function of the Shape Script.

Assume that the Controller has the following variables.

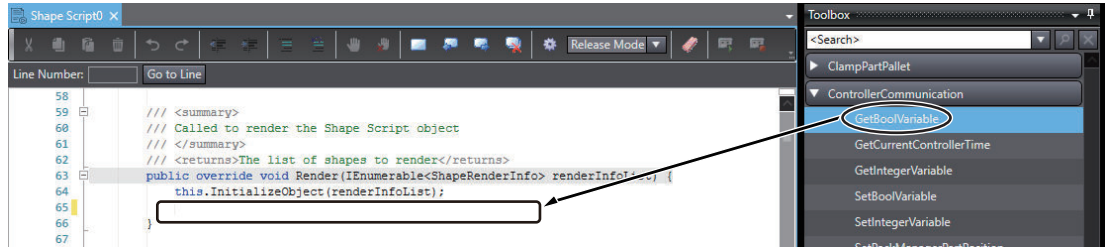
- IsPicked (BOOL): A BOOL variable that is TRUE while the part is picked up.
- IsPlaced (BOOL): A BOOL variable that is TRUE while the part is placed, until the picking of the next part starts.

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*.

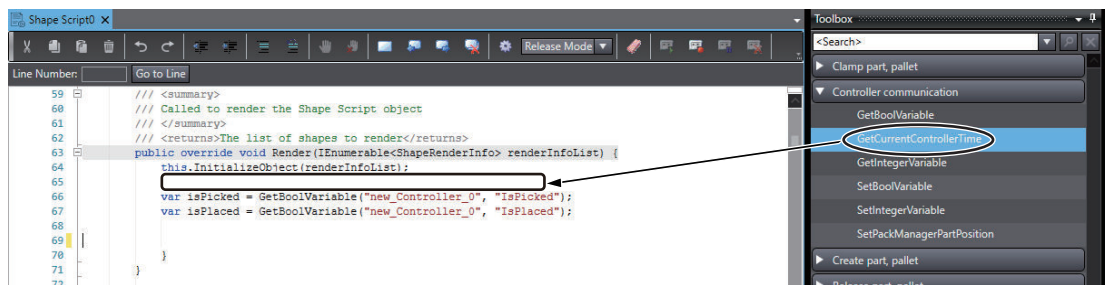
```
63 public override void Render(IEnumerable<ShapeRenderInfo> renderInfoList) {
64     this.InitializeObject(renderInfoList);
65
66     var isPicked = GetBoolVariable("new_Controller_0", "IsPicked");
67     var isPlaced = GetBoolVariable("new_Controller_0", "IsPlaced");
68 }
```

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 **GetCurrentControllerTime** 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.

```
65
66     var isPicked = GetBoolVariable("new_Controller_0", "IsPicked");
67     var isPlaced = GetBoolVariable("new_Controller_0", "IsPlaced");
68
69     if(isPicked && !this.isPicking){
70         // Start task
71         this.startTime = this.GetCurrentControllerTime("new_Controller_0");
72         this.isPicking = true;
73         this.count++;
74     }
75 }
```

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.

```

73         this.count++;
74     }
75
76     if(isPlaced && this.isPicking){
77         // End takt
78         var currentTime = this.GetCurrentControllerTime("new_Controller_0");
79         var taktTime = currentTime - this.startTime;
80         Trace.WriteLine(count + " : " + taktTime.TotalMilliseconds + " ms");
81
82         this.isPicking = false;
83     }
84
85

```

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.

```

65
66     var isPicked = GetBoolVariable("new_Controller_0", "IsPicked");
67     var isPlaced = GetBoolVariable("new_Controller_0", "IsPlaced");
68
69     if(isPicked && !this.isPicking){
70         // Start task
71         this.startTime = this.GetCurrentControllerTime("new_Controller_0");
72         this.isPicking = true;
73         this.count++;
74     }
75
76     if(isPlaced && this.isPicking){
77         // End takt
78         var currentTime = this.GetCurrentControllerTime("new_Controller_0");
79         var taktTime = currentTime - this.startTime;
80         Trace.WriteLine(count + " : " + taktTime.TotalMilliseconds + " ms");
81
82         this.isPicking = false;
83     }
84
85

```



The screenshot shows the Trace Message window with the following output:

```

1 : 4720 ms
2 : 1440 ms
3 : 1480 ms
4 : 1280 ms

```

7-3 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.

7-3-1 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

7-3-2 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.

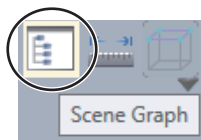
This section describes the setting procedure for collision detection between the mechanical component and the part, using an example case where a part of the mechanical component comes in contact with the part.

Register 3D shape data for the following mechanical component and part in advance.

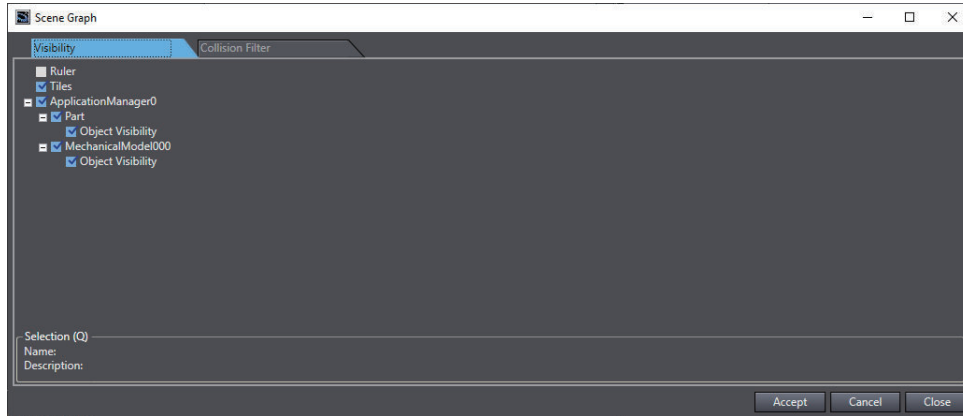
Name of mechanical component: MechanicalModel000

Name of part: Part

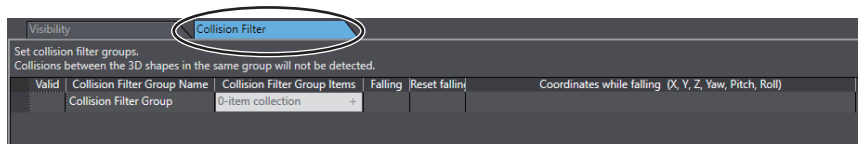
- 1 In the 3D Visualizer, click the **Scene Graph** icon.



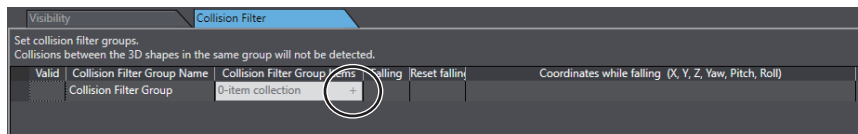
The **Scene Graph** dialog box is displayed.



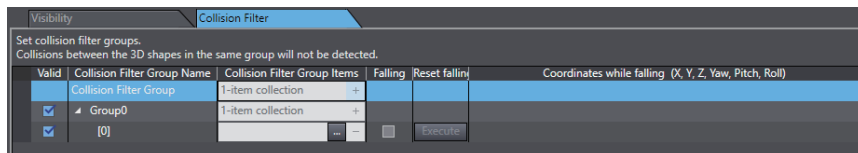
- 2 Click the **Collision Filter** tab.
The display changes to the **Collision Filter** tab page.



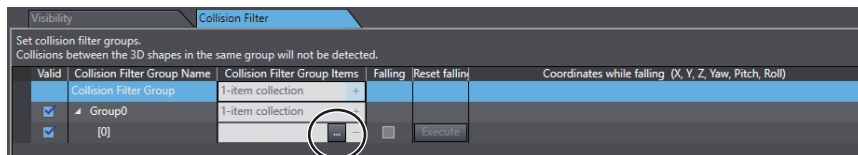
- 3 Click the **+** button for Collision Filter Group.



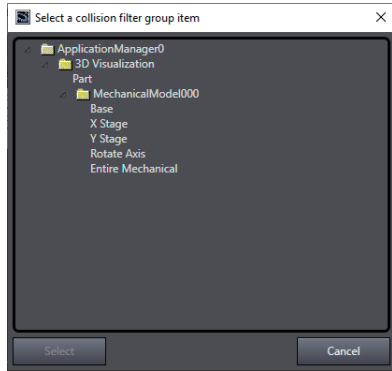
A Collision Filter Group is added.



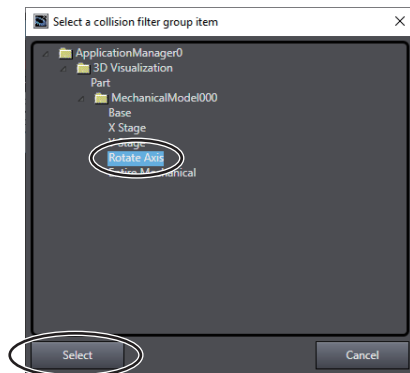
- 4 Click the Item Selection button for the added Collision Filer Group.



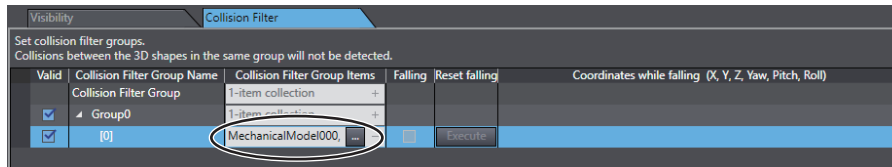
A dialog box is displayed for you to register the target 3D shape data in the Collision Filer Group.



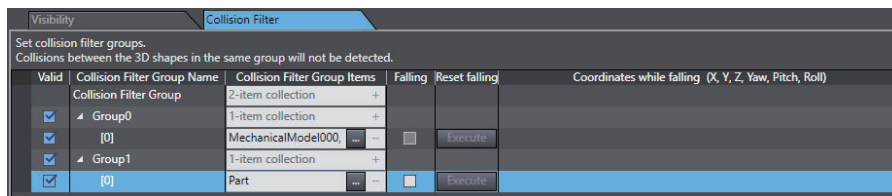
- 5 Select the 3D shape data for the target Mechanical Component to register in the Collision Filter Group, and then click the **Select** button.



The selected 3D shape data is registered in the Collision Filter 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 Filter 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.

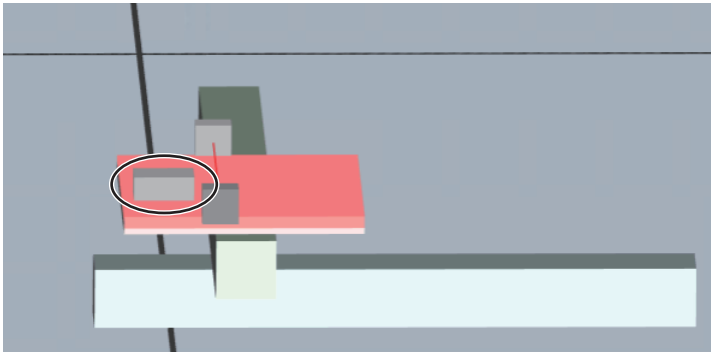
7-3-3 How to Check Detected Collisions

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.

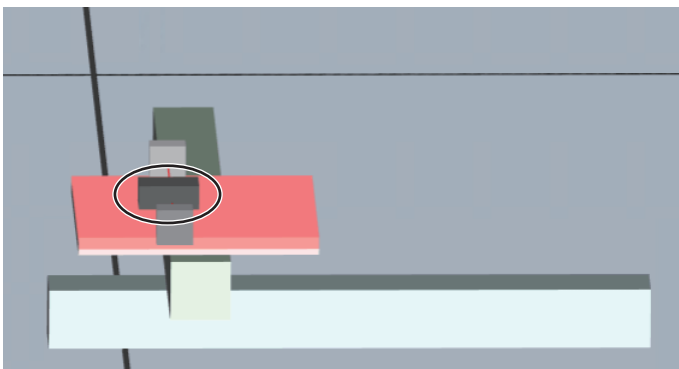
3D Shape Data That Did Not Cause a Collision

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

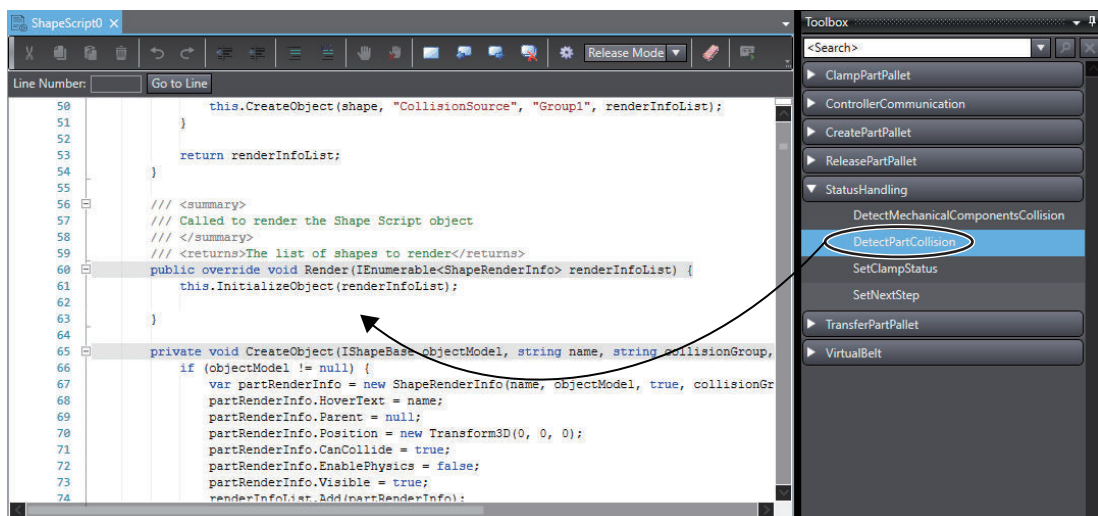
7-3-4 How to Detect Collisions with Shape Scripts

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.

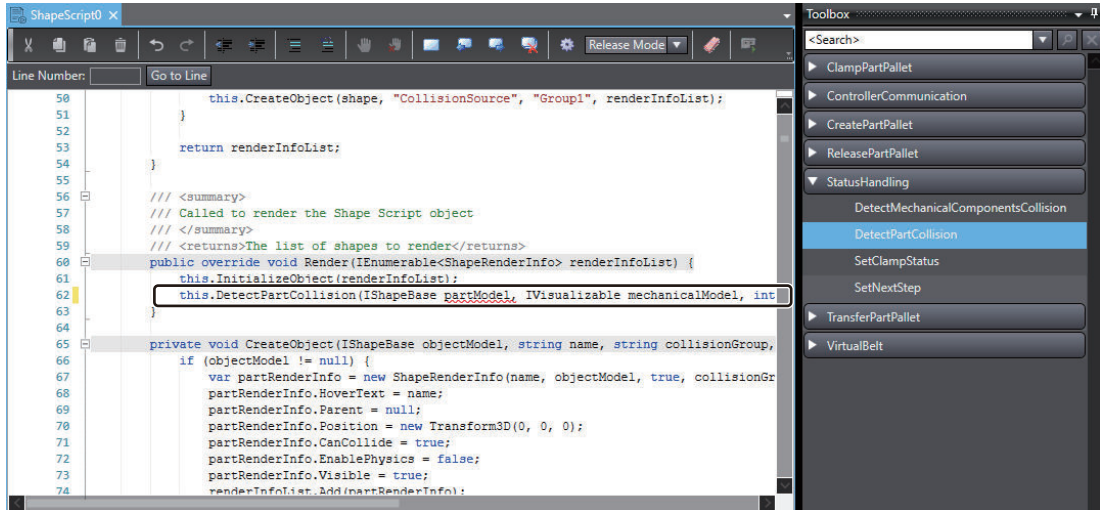
Detecting a Collision between a Part or Pallet and a Mechanical Component

The following describes the procedure to detect a collision between a part or pallet and a mechanical component.

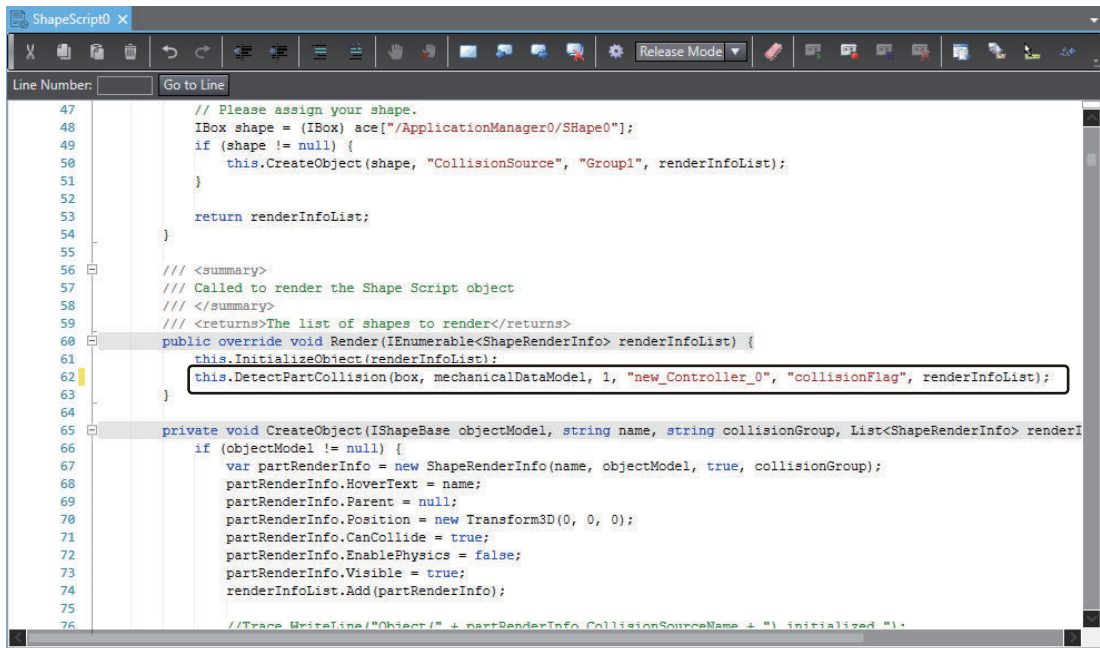
- 1 Add the mechanical component for which to detect a collision to the Collision Filter Group. Refer to *7-3-2 Collision Detection Setting Procedure* on page 7-13 for information on how to add a mechanical component to the Collision Filter.
- 2 Add the DetectPartCollision() function to the Render() function in the Shape Script. Select **DetectPartCollision** under **StatusHandling** in the Toolbox and drag it to the Render() function.



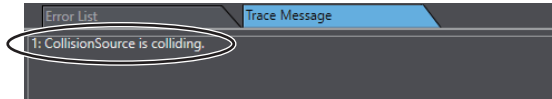
The DetectPartCollision() function is inserted into the point at which you drop it.



- 3 Specify appropriate variables as arguments to the DetectPartCollision() function. Refer to *DetectPartCollision* on page A-28 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 6-9 for information on how to add a Shape Script. Refer to 7-1-3 *Executing the Shape Scripts for the Part* on page 7-3 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.

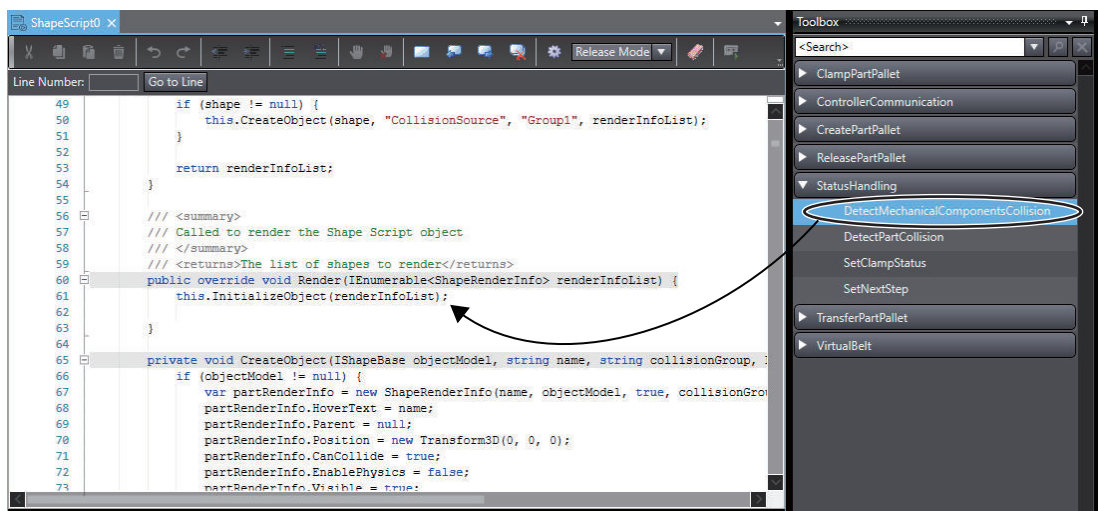


In the example in step 3, the BOOL global variable *collisionFlag* in the Controller *new_Controller_0* changes to TRUE.

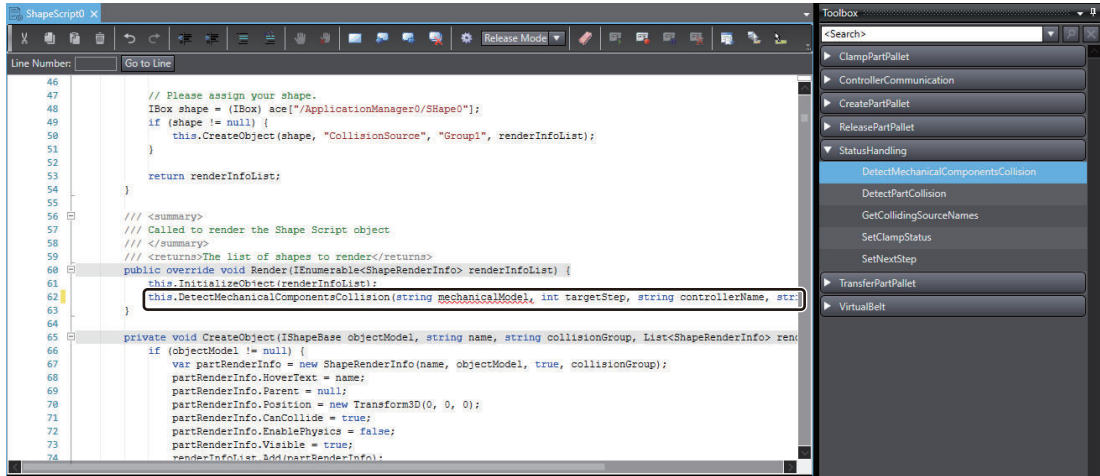
Detecting a Collision between Mechanical Components

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 7-3-2 *Collision Detection Setting Procedure* on page 7-13 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.



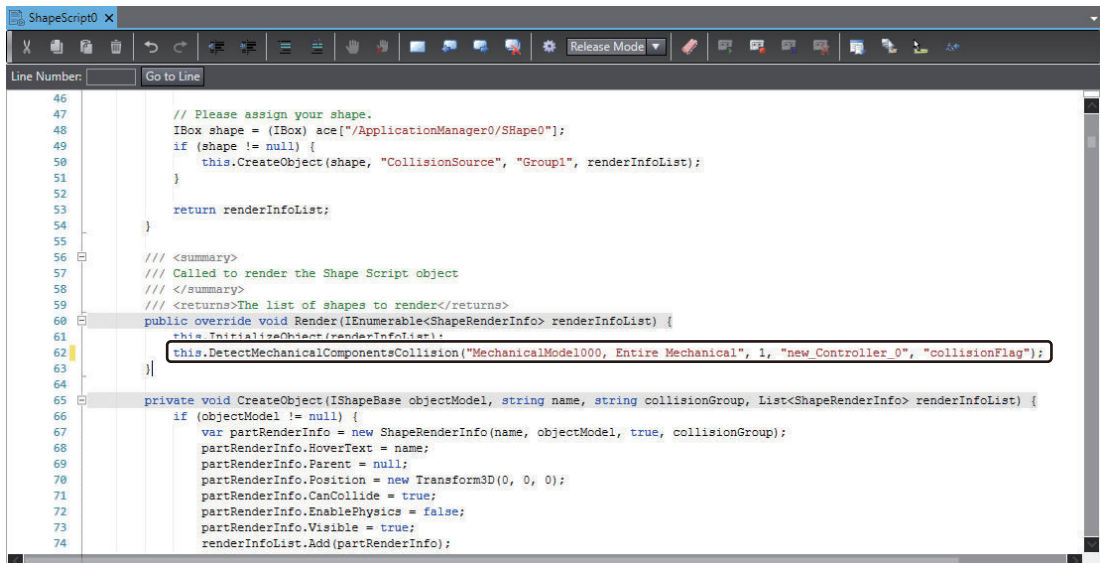
The DetectMechanicalComponentsCollision() function is inserted into the point at which you drop it.



3 Specify appropriate variables as arguments to the DetectMechanicalComponentsCollision() function.

Refer to *DetectMechanicalComponentsCollision* on page A-29 for details on the DetectMechanicalComponentsCollision() function.

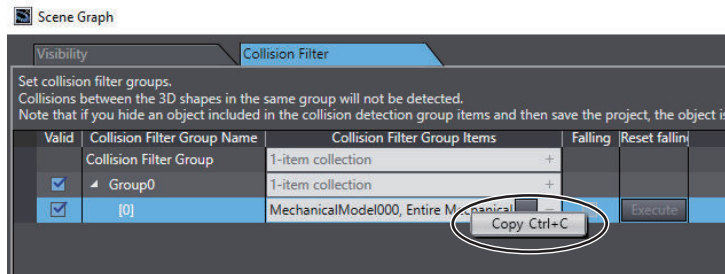
The following is an example where the BOOL global variable *collisionFlag* in the Controller *new_Controller_0* changes to TRUE when the mechanical component *MechanicalModel000* collides with another mechanical component.



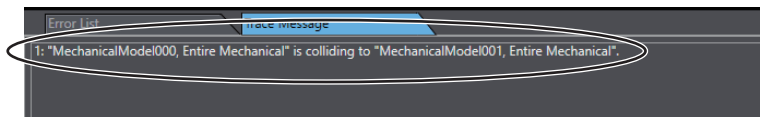


Additional Information

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 6-9 for information on how to add a Shape Script. Refer to 7-1-3 *Executing the Shape Scripts for the Part* on page 7-3 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.



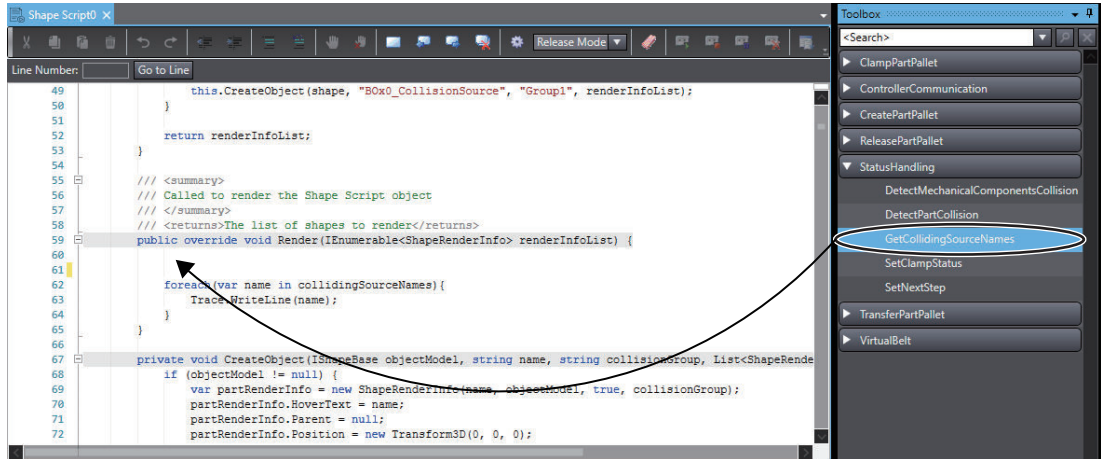
Additional Information

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).

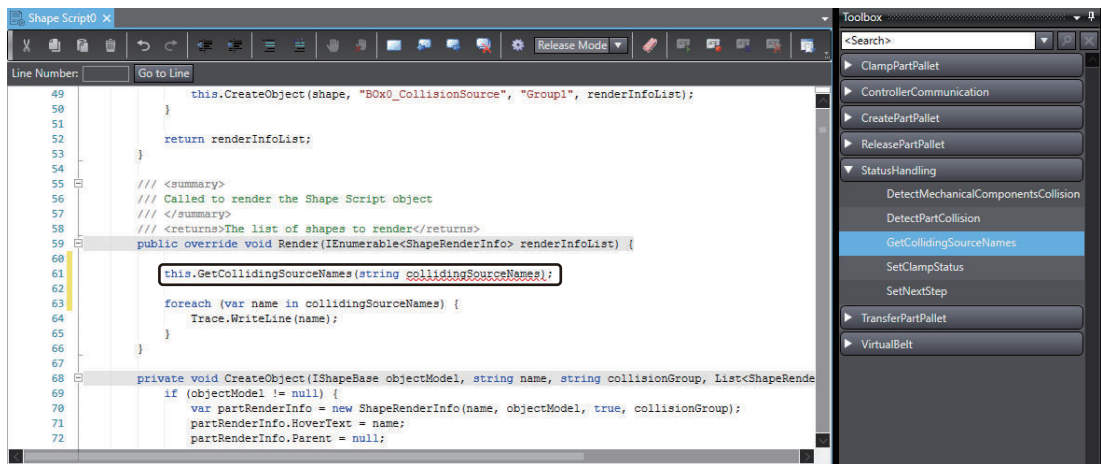
Getting the Name of 3D Shape Data That Caused a Collision

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-30 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*.


```

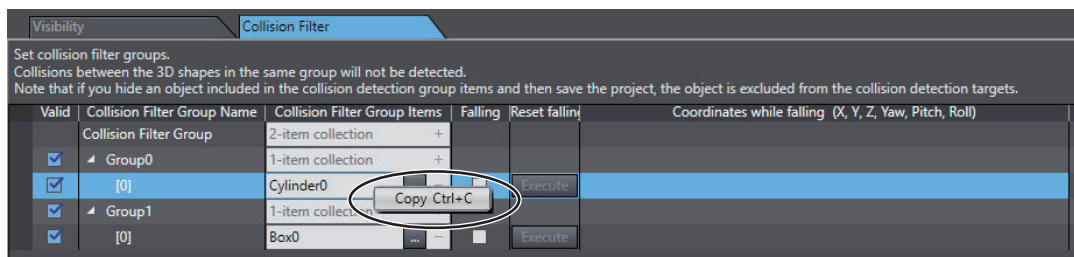
43
44 // Create the Shapes for rendering once in the constructor
45 List<ShapeRenderInfo> renderInfoList = new List<ShapeRenderInfo>();
46
47 // Please assign your shape.
48 var shape = ace["/ApplicationManager0/Box0"] as IShapeBase;
49 if (shape != null) {
50     this.CreateObject(shape, "Box0_CollisionSource", "Group1", renderInfoList);
51 }
52
53 return renderInfoList;
54 }
55
56 /// <summary>
57 /// Called to render the Shape Script object
58 /// </summary>
59 /// <returns>The list of shapes to render</returns>
60 public override void Render(IEnumerable<ShapeRenderInfo> renderInfoList) {
61     var collidingSourceNames = this.GetCollidingSourceNames("Cylinder0");
62
63     foreach (var name in collidingSourceNames) {
64         Trace.WriteLine(name);
65     }
66 }

```

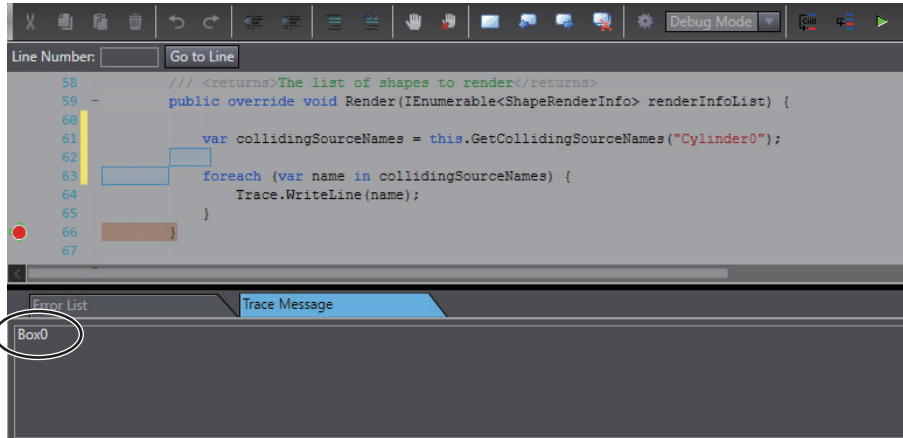


Additional Information

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.

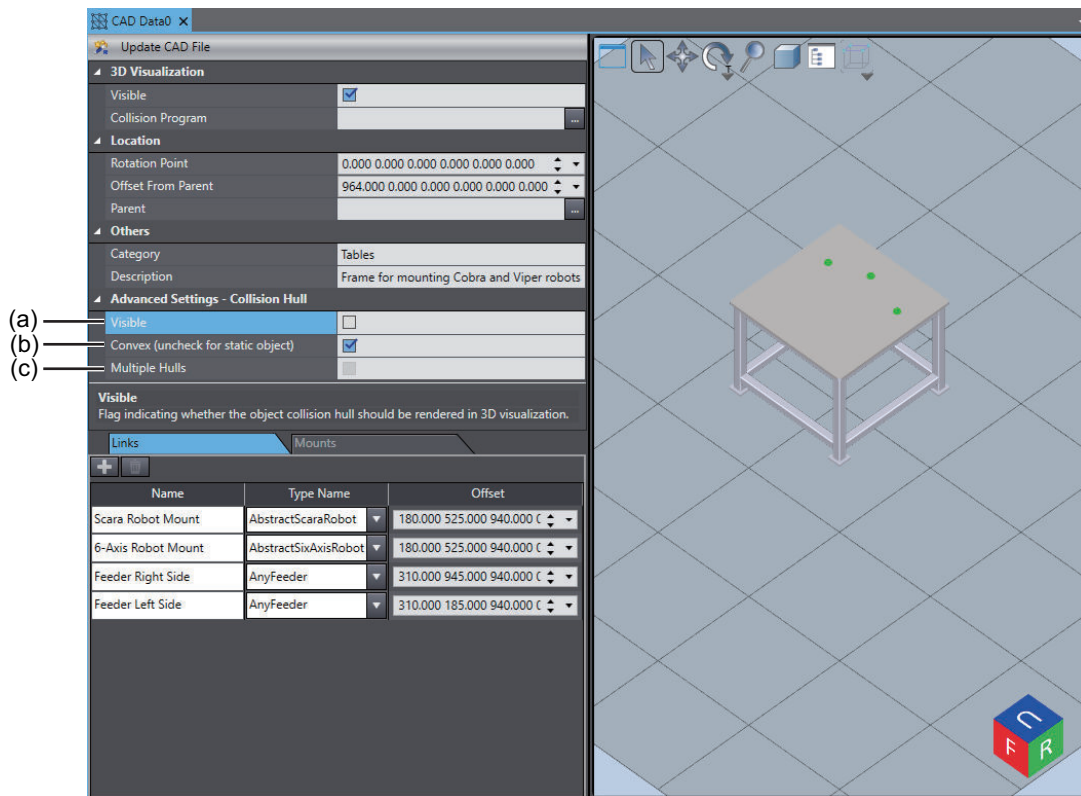


- 3 Start the Simulator of the Controller.
- 4 Add a Shape Script that executes the `GetCollidingSourceNames()` function to the Shape Script Sequence and execute the Shape Script.
Refer to 6-3-2 *Setting the Execution of Shape Scripts* on page 6-9 for information on how to add a Shape Script. Refer to 7-1-3 *Executing the Shape Scripts for the Part* on page 7-3 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*.



7-3-5 Collision Filter Model Settings

A collision filter model 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 filter models 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 filter models.	Checked or unchecked	Unchecked

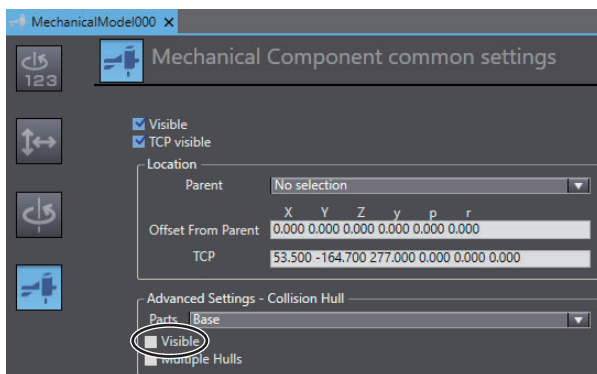
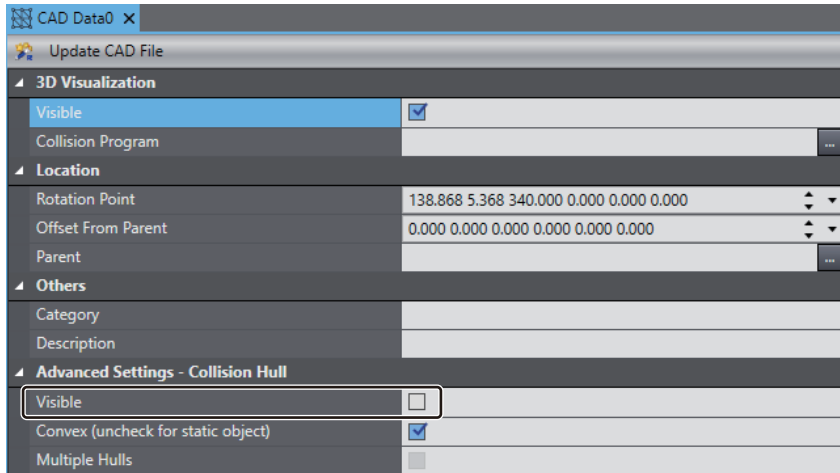
	Item	Description	Set value	Initial value
(b)	Convex*1	<p>Set whether to simplify collision filter models.</p> <p>Simplifying the shapes of collision filter models reduces the accuracy of collision detection, but improves the detection speed.</p> <p>However, simplifying static 3D shape data such as belts and tables does not improve the accuracy of collision detection very much.</p> <p>If you clear this check box to disable the function in order to simplify the shapes of collision filter models, the collision detection operates as follows.</p> <ul style="list-style-type: none"> • Collisions between objects are not detected. • Only surface collisions are detected. Internal collisions are not detected. • The Falling option cannot be set. <p>You cannot change this setting when the Falling option is enabled.</p>	Checked or unchecked	Checked
(c)	Multiple Hulls	<p>Set whether to split the 3D shape data into multiple pieces to generate collision detection models.</p> <p>Splitting the 3D shape data increases the time required for collision detection, but improves the accuracy of the collision detection.</p> <p>This setting is not available if any of the following conditions is met.</p> <ul style="list-style-type: none"> • The 3D shape data consists of a single part. • The 3D shape data is generated with Sysmac Studio, which does not support collision filter model generation. <p>You cannot change this setting when the Falling option is enabled.</p>	Checked or unchecked	Unchecked

*1. For static objects, clear this check box.

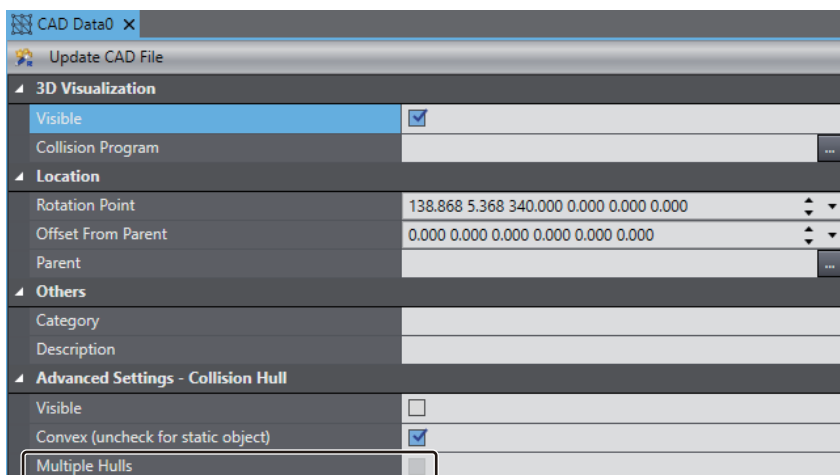


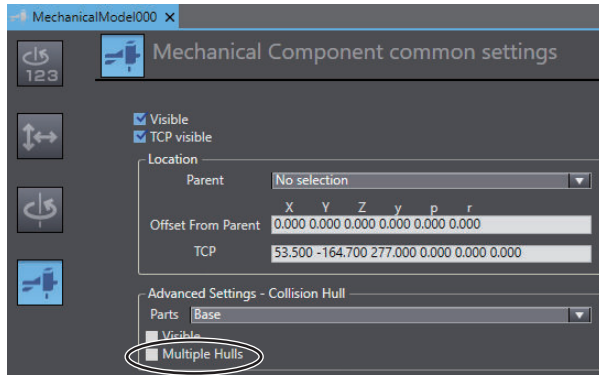
Precautions for Correct Use

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 filter model to be displayed on the 3D Visualizer.

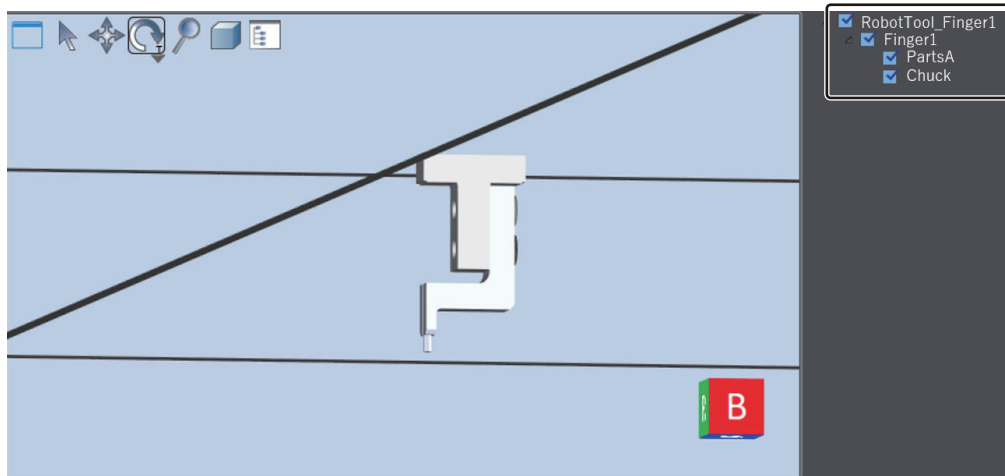


To use a collision filter model that looks closer to the displayed 3D shape data for collision detection, select the **Multiple Halls** check box in **Advanced Settings – Collision Hull**. Selecting this check box causes the 3D shape data to be analyzed and internally split into multiple parts. A collision filter model is then generated for each of the split parts.





However, when the 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. For more accurate collision detection, use 3D CAD software to split the 3D shape data into parts and import them again.



8

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.

8-1	Outline of a 3D Simulation of Robot Systems with an Application Controller	8-2
8-1-1	Types of 3D Simulation	8-2
8-1-2	3D Simulation of Mechanical Components Controlled by IEC 61131-3 Languages.....	8-2
8-1-3	3D Simulation of Robot Applications	8-2
8-2	3D Simulation of Robot Systems.....	8-4
8-2-1	Using a Peripheral Device to Manipulate Parts That Were Manipulated by a Robot.....	8-4
8-2-2	Using a Robot to Manipulate Parts That Were Manipulated by a Peripheral Device	8-4
8-2-3	Detecting Collisions of Parts That Are Manipulated Only by a Robot	8-6

8-1 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).

8-1-1 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 palletes, 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.

8-1-2 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.

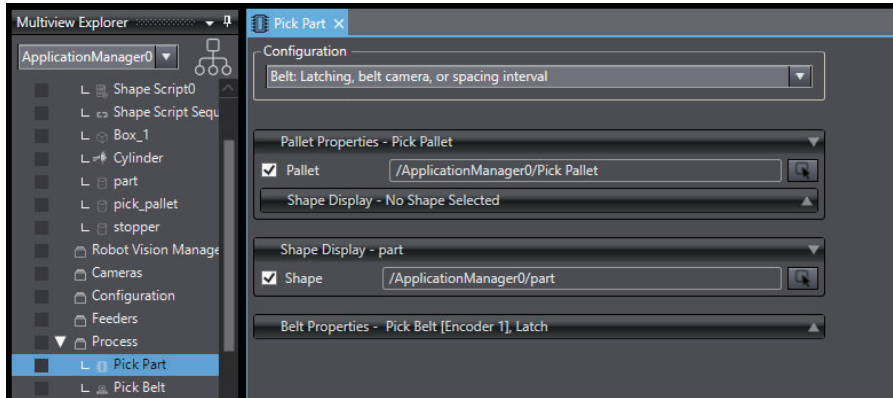
In the Application Manager, right-click **3D Visualization** and select **Add – Shape Script** from the context menu to add a Shape Script. In the same way, select **Add – Shape Script Sequence** to add a Shape Script Sequence. Write a program with the Shape Script that you added, register it to the 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.

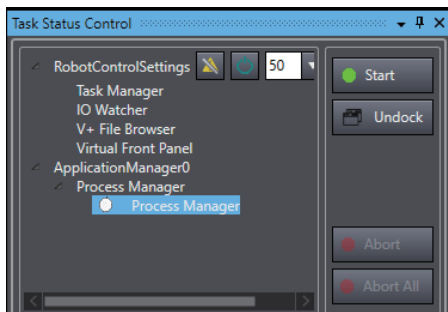
8-1-3 3D Simulation of Robot Applications

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.



8-2 3D Simulation of Robot Systems

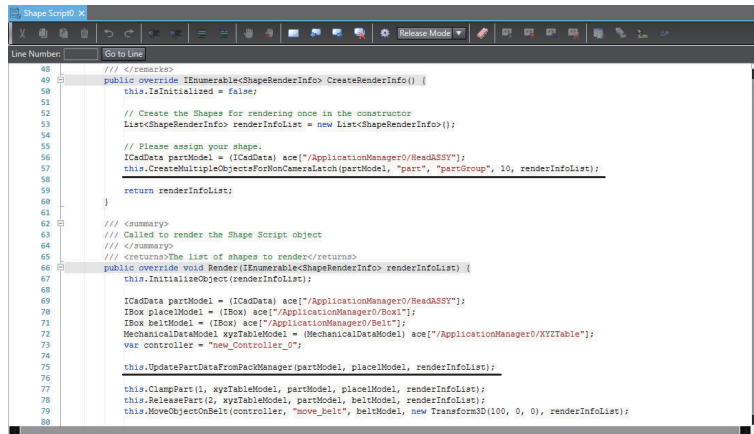
Using a Shape Script, a Shape Script Sequence, and the emulation function of the Process Manager enables you to execute a 3D simulation of a robot and peripheral devices (mechanical components, etc.) including the part.

8-2-1 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.



```

48 // // // </remarks>
49 public override IEnumerable<ShapeRenderInfo> CreateRenderInfo() {
50     this.IsInitialized = false;
51
52     // Create the Shapes for rendering once in the constructor
53     List<ShapeRenderInfo> renderInfoList = new List<ShapeRenderInfo>();
54
55     // Please assign your shape.
56     ICadData partModel = (ICadData) ace["/ApplicationManager/HeadASSY"];
57     this.CreateMultipleObjectsForNonCameraLatch(partModel, "part", "partGroup", 10, renderInfoList);
58
59     return renderInfoList;
60 }
61
62 // // // <summary>
63 // // // Called to render the Shape Script object
64 // // // </summary>
65 // // // returns The list of shapes to render/returns
66 public override void Render(IEnumerable<ShapeRenderInfo> renderInfoList) {
67     this.InitializeObject(renderInfoList);
68
69     ICadData partModel = (ICadData) ace["/ApplicationManager/HeadASSY"];
70     IBox placeModel = (IBox) ace["/ApplicationManager/Box1"];
71     IBox beltModel = (IBox) ace["/ApplicationManager/Belt"];
72     MechanicalDataModel xyTableModel = (MechanicalDataModel) ace["/ApplicationManager/XYTable"];
73     var controller = "New_Controller_0";
74
75     this.UpdatePartDataFromPackManager(partModel, placeModel, renderInfoList);
76
77     this.ClampPart(1, xyTableModel, partModel, placeModel, renderInfoList);
78     this.ReleasePart(2, xyTableModel, partModel, beltModel, renderInfoList);
79     this.MoveObjectOnBelt(controller, "move_belt", beltModel, new Transform3D(100, 0, 0), renderInfoList);
80 }

```

Refer to *A-1 Functions Used in Shape Scripts* on page A-2 for details on the methods.

8-2-2 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 MovePartOnSensorLatchBelt 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.

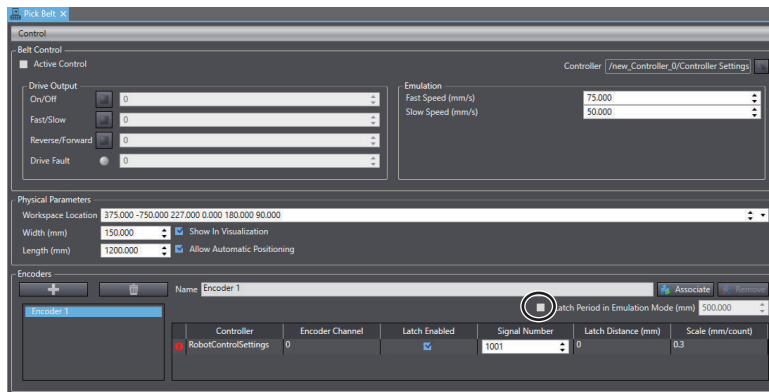
```

Shape Script X
Line Number:
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     // Please assign your shape.
47     var shape = ace["ApplicationManager/HeadASSY"] as IShapeBase;
48     this.CreateMultipleObjects(shape, "part", "partGroup", 5, renderInfoList);
49
50     return renderInfoList;
51 }
52
53 // <summary>
54 // Called to render the Shape Script object
55 // </summary>
56 // <returns>The list of shapes to render</returns>
57 public override void Render(IEnumerable<ShapeRenderInfo> renderInfoList) {
58     this.InitializeObject(renderInfoList);
59
60     ICoaData partModel = (ICoaData) ace["ApplicationManager/HeadASSY"];
61     IBelt beltModel = (IBelt) ace["ApplicationManager/Pick Belt"];
62     IProcessManager processManager = (IProcessManager) ace["ApplicationManager/Process Manager"];
63     IBox placeModel = (IBox) ace["ApplicationManager/Box1"];
64     IMechanicalDataModel xyzTableModel = (IMechanicalDataModel) ace["ApplicationManager/XYZTable"];
65     var controller = "new_Controller_0";
66
67     this.LoadPart(controller, "part_load", partModel, null, new Transform3D(375, -700.0, 227.0), placeModel, renderInfoList);
68
69     this.ClampPart(1, xyzTableModel, partModel, placeModel, renderInfoList);
70     this.ReleasePart(2, xyzTableModel, partModel, beltModel, renderInfoList);
71
72     this.MovePartOnSensorLatchBelt(processManager, beltModel, renderInfoList);
73
74     this.UnloadPart(controller, "part_unload", partModel, null, renderInfoList);
75 }
76
77
78
79
80
81
82
83
84
85

```

Refer to *A-1 Functions Used in Shape Scripts* on page A-2 for details on the methods.

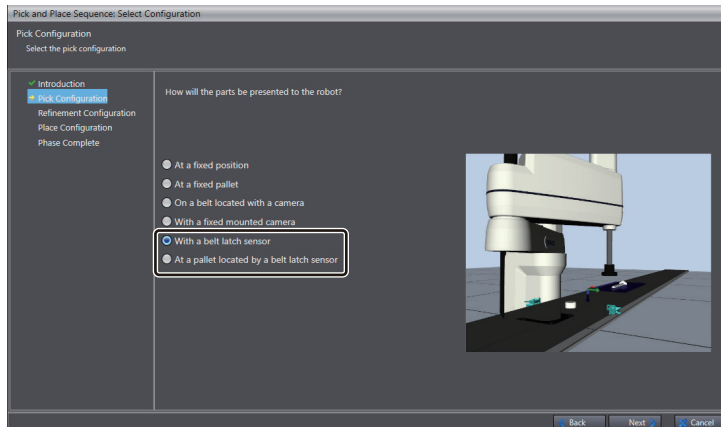
In addition, clear the **Latch Period in Emulation Mode (mm)** check box in the Process Manager's belt settings to stop the Process Manager from generating parts.





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.



8-2-3 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 CreateMultipleObjectsForNonCameraLatch method.

Program the Shape Script as follows.

```

48  // </remarks>
49  public override IEnumerable<ShapeRenderInfo> CreateRenderInfo() {
50      this.Initialized = false;
51
52      // Create the shapes for rendering once in the constructor
53      List<ShapeRenderInfo> renderInfoList = new List<ShapeRenderInfo>();
54
55      // Please assign your shape.
56      ICadData partModel = (ICadData) ace!["ApplicationManager0/HeadASSY"];
57      this.CreateMultipleObjectsForCameraMatch(partModel, "part", "partGroup", 10, renderInfoList);
58      return renderInfoList;
59  }
60
61  // <summary>
62  // Called to render the Shape Scripts object
63  // </summary>
64  // <returns>The list of shapes to render</returns>
65  public override void Render(IEnumerable<ShapeRenderInfo> renderInfoList) {
66      this.InitializeObject(renderInfoList);
67
68      ICadData partModel = (ICadData) ace!["ApplicationManager0/HeadASSY"];
69      IBox box1Model = (IBox) ace!["ApplicationManager0/Box1"];
70      IBox box2Model = (IBox) ace!["ApplicationManager0/Box2"];
71      var controller = "new_Controller_0";
72
73      this.DetectPartCollision(partModel, box1Model, 1, controller, "check1");
74      this.DetectPartCollision(partModel, box2Model, 1, controller, "check2");
75  }
76
77

```

Refer to *A-1 Functions Used in Shape Scripts* on page A-2 for details on the methods.



Useful Functions

This section describes useful 3D simulation functions.

9-1	Updating All CAD Data.....	9-2
9-1-1	Procedure to Update All CAD Data	9-2

9-1 Updating All CAD Data

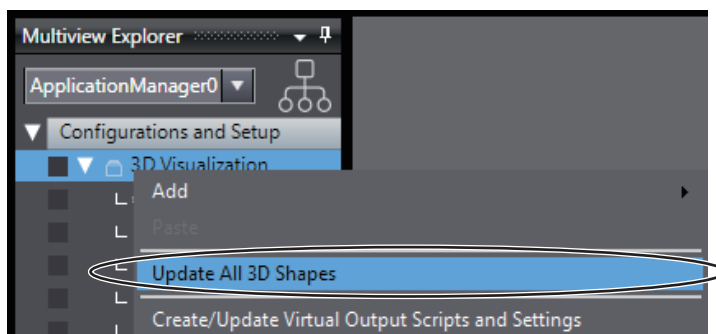
You can update one set of CAD data that contains 3D shape data with another set of CAD data at once.

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.

9-1-1 Procedure to Update All CAD Data

Use the following procedure to update all CAD data.

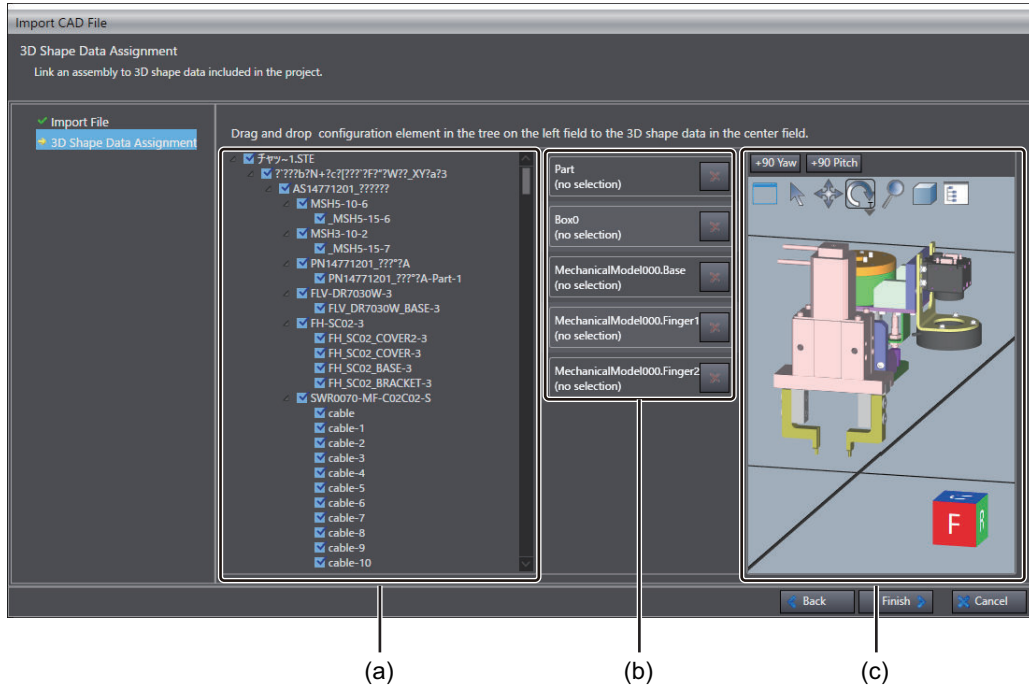
- 1 Right-click **3D Visualization** in the Multiview Explorer and select **Update All 3D Shapes** from the menu.



The Import CAD File wizard starts and the Select file page is displayed.

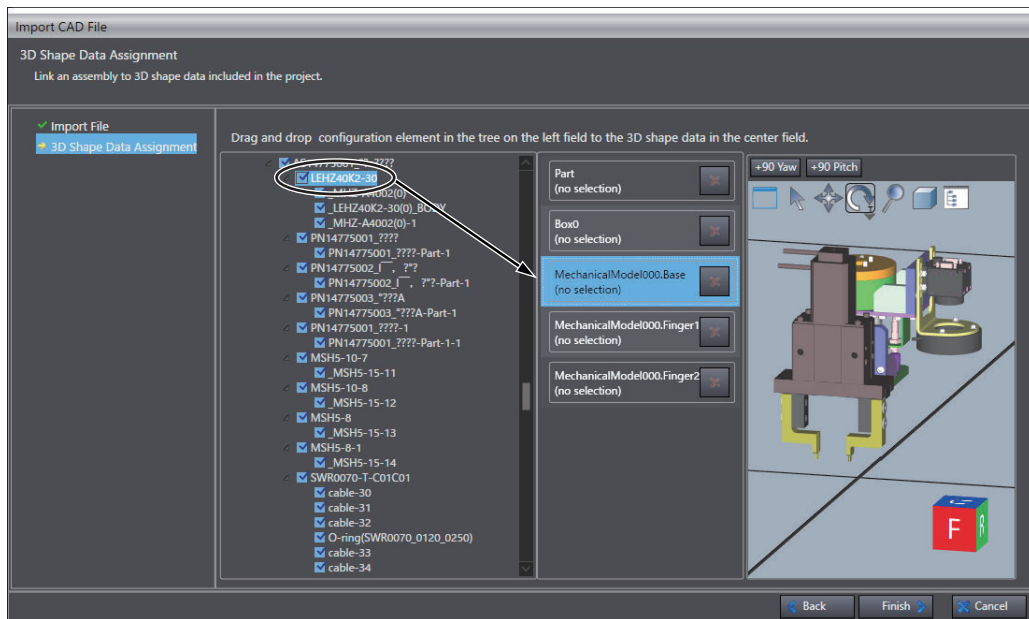
In the Select file page, you can select a single CAD file. Refer to *4-3-1 Types of Supported CAD Data Files* on page 4-7 for the types of supported CAD files.

- 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.

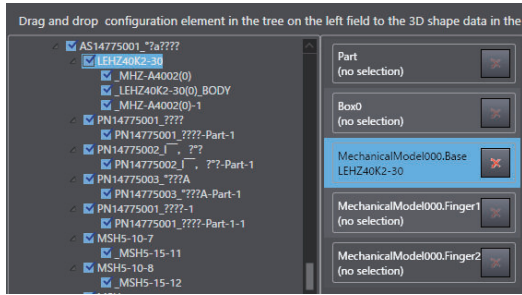


	Item	Description
(a)	Source CAD data	Data in the selected CAD file is displayed. This represents the structure of parts that make up the CAD data.
(b)	Target 3D shape data	3D shape data in the project is listed.
(c)	Source CAD data in the 3D Visualizer	A 3D view of the source CAD data is displayed.

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 X 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.



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	A-2
A-1-1	Creating and Displaying Parts and Pallets	A-5
A-1-2	Conveying Parts and Pallets	A-13
A-1-3	Virtual Conveyor	A-15
A-1-4	Clamping Parts and Pallets	A-16
A-1-5	Placing Parts and Pallets	A-22
A-1-6	Changing and Detecting Status.....	A-27
A-1-7	Sharing Variables with the Controller	A-30
A-1-8	Cooperation with the Process Manager	A-39
A-2	Differences between the Simulator and the Physical Controller.....	A-43
A-2-1	Operation of Functions	A-43

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.

Functions available in Shape Script are listed below.

Refer to 6-5-2 *Shape Script Programming* on page 6-18 for how to input functions.

Category	Variable name	Description	Reference
Creating and displaying parts and pallets	CreateObject	Create a part or a pallet.	<i>CreateObject</i> on page A-5
	CreateMultipleObjects	Create multiple parts or pallets.	<i>CreateMultipleObjects</i> on page A-6
	CreateMultipleObjects-ForNonCameraLatch	Create multiple parts or pallets that are used by the emulation function of the Application Manager.	<i>CreateMultipleObjectsForNonCameraLatch</i> on page A-7
	LoadPart	Display parts.	<i>LoadPart</i> on page A-8
	LoadPallet	Display pallets.	<i>LoadPallet</i> on page A-9
	LoadPartOnPallet	Display parts on a pallet.	<i>LoadPartOnPallet</i> on page A-10
	UnloadPart	Hide the part by entering the BOOL variable of the Controller.	<i>UnloadPart</i> on page A-11
	UnloadPallet	Hide the pallet by entering the BOOL variable of the Controller.	<i>UnloadPallet</i> on page A-12
	UnloadCollidingPart	Hide the part after colliding with the designated 3D shape data.	<i>UnloadCollidingPart</i> on page A-12
UnloadCollidingPallet	Hide the pallet after colliding with the designated 3D shape data.	<i>UnloadCollidingPallet</i> on page A-13	
Conveying parts and pallets	PushPart	Push the part by the cylinder.	<i>PushPart</i> on page A-13
	PushPallet	Push the pallet by the cylinder.	<i>PushPallet</i> on page A-14
Virtual conveyor	MoveObjectOnBelt	Move the part or pallet on the conveyor.	<i>MoveObjectOnBelt</i> on page A-15
Clamping parts and pallets	ClampPart	Pick up the part.	<i>ClampPart</i> on page A-16
	ClampPallet	Pick up the pallet.	<i>ClampPallet</i> on page A-17
	ClampPartOnPallet	Pick up the part on the pallet.	<i>ClampPartOnPallet</i> on page A-18
	ClampPartBySignal	Pick up the part on a grasping signal.	<i>ClampPartBySignal</i> on page A-18
	ClampPalletBySignal	Pick up the pallet on a grasping signal.	<i>ClampPalletBySignal</i> on page A-19
	ClampPartOnPalletBySignal	Pick up the part on the pallet on a grasping signal.	<i>ClampPartOnPalletBySignal</i> on page A-20

Category	Variable name	Description	Reference
Placing parts and pallets	ReleasePart	Place the part picked up.	<i>ReleasePart</i> on page A-22
	ReleasePallet	Place the pallet picked up.	<i>ReleasePallet</i> on page A-22
	ReleasePartOnPallet	Place the part which picked up to the pallet.	<i>ReleasePartOnPallet</i> on page A-23
	ReleasePartBySignal	Release (place) the part according to a grasping signal of the robot hand.	<i>ReleasePartBySignal</i> on page A-24
	ReleasePalletBySignal	Release (place) the pallet according to a grasping signal of the robot hand.	<i>ReleasePalletBySignal</i> on page A-25
	ReleasePartOnPalletBySignal	Release (place) the part to the pallet according to a grasping signal of the robot hand.	<i>ReleasePartOnPalletBySignal</i> on page A-26
Changing and detecting status	SetNextStep	Change the value of step.	<i>SetNextStep</i> on page A-27
	DetectPartCollision	Detect that the part or pallet collides with the mechanical component.	<i>DetectPartCollision</i> on page A-28
	DetectMechanicalComponentsCollision	Detect that the mechanical component collides with another mechanical component.	<i>DetectMechanicalComponentsCollision</i> on page A-29
	SetClampStatus	Write the grasping (colliding) status of the robot hand to the variable of the Controller.	<i>SetClampStatus</i> on page A-29
	GetCollidingSourceNames	Get the names of all 3D shape data that collided with the specified 3D shape data.	<i>GetCollidingSourceNames</i> on page A-30
Sharing variables with the Controller	GetBoolVariable	Read values of BOOL variables of the specified Controller.	<i>GetBoolVariable</i> on page A-30
	SetBoolVariable	Write values of BOOL variables of the specified Controller.	<i>SetBoolVariable</i> on page A-31
	GetIntegerVariable	Read values of DINT variables of the specified Controller.	<i>GetIntegerVariable</i> on page A-31
	SetIntegerVariable	Write values to DINT variables of the specified Controller.	<i>SetIntegerVariable</i> on page A-31
	GetCurrentControllerTime	Read the internal simulation time of the specified Controller.	<i>GetCurrentControllerTime</i> on page A-32

Category	Variable name	Description	Reference
	Get**Variable	Read values of integer variables of the specified Controller. For "*" of the function name, specify the data type of the Controller variable.	<i>Get**Variable</i> on page A-32
	Set**Variable	Write values to integer variables of the specified Controller. For "*" of the function name, specify the data type of the Controller variable.	<i>Set**Variable</i> on page A-33
	CreateGetVariableList	Create a new read variable list, which is used to read multiple variables from the Controller.	<i>CreateGetVariableList</i> on page A-34
	AddToGetVariableList	Add the variable to read from the Controller to the read variable list.	<i>AddToGetVariableList</i> on page A-34
	GetVariableValues	Read the values of multiple variables from the Controller at a time.	<i>GetVariableValues</i> on page A-35
	GetValueFromVariableValueList	Get the value of a variable that you read with the <i>GetVariableValues()</i> function.	<i>GetValueFromVariableValueList</i> on page A-35
	CreateSetVariableList	Create a new write variable list, which is used to write multiple variables from the Controller.	<i>CreateSetVariableList</i> on page A-37
	AddToSetVariableList	Add the variable and value to write to the Controller to the write variable list.	<i>AddToSetVariableList</i> on page A-37
	SetVariableValues	Write values to multiple Controller variables at a time.	<i>SetVariableValues</i> on page A-38

Category	Variable name	Description	Reference
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.	<i>MovePartOnPackManagerSensorLatchBelt</i> on page A-39
	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.	<i>MovePalletOnPackManagerSensorLatchBelt</i> on page A-39
	SetPackManagerPartPosition	Write the position of the part that is manipulated in the Process Manager to a Controller variable.	<i>SetPackManagerPartPosition</i> on page A-40
	UpdatePartDataFromPackManager	Allow the Shape Script to continue to manipulate the part that was picked up and placed in the Process Manager.	<i>UpdatePartDataFromPackManager</i> on page A-41
	UpdatePartDataOnPalletFromPackManager	Allow the Shape Script to continue to manipulate the part that was picked up and placed on the pallet in the Process Manager.	<i>UpdatePartDataOnPalletFromPackManager</i> on page A-41

A-1-1 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 object-Model	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.	---

Argument	Description	Notation example
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 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"

CreateMultipleObjectsForNonCameraLatch

Create multiple parts or pallets that are used by the emulation function of the Application 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(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.	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.	---

● **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(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
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 <i>renderInfoList</i>	"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 the part when the global variable *variableName* changes from OFF to ON. The change of the specified *variableName* from OFF to ON is detected only when the LoadPart() function that specifies the *variableName* in the Render() function is called the first time.

The following is an example in which the LoadPart() function is not executed correctly.

When line 62 is executed, the program checks whether the variable *var_LoadPart* changed from OFF to ON. If the variable changed from OFF to ON, among parts that are registered in the *renderInfoList* the part that corresponds to the *partModel* is displayed.

After this, the part that corresponds to *partModel1* is not displayed even if line 63 is executed because the program determines that the variable *var_LoadPart* has already changed from OFF to ON.

```

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      this.InitializeObject(renderInfoList);
61
62      this.LoadPart("new_controller_0", "var_LoadPart", partModel, "Part", new Transform3D(0, 0, 0), placeModel, renderInfoList);
63      this.LoadPart("new_controller_0", "var_LoadPart", partModel1, "Part", new Transform3D(0, 0, 0), placeModel1, renderInfoList);
64  }
    
```

To display the part when line 63 is executed, first add a BOOL global variable other than *var_LoadPart* on the Controller. Then create a program that also changes the newly added global variable from OFF to ON as soon as *var_LoadPart* changes from OFF to ON. The newly added global variable must be set as an argument in line 63.

LoadPallet

Display pallets.

● **Function Call**

Function	LoadPallet(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
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 <i>renderInfoList</i>	"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

Argument	Description	Notation example
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-8 for the restrictions.

LoadPartOnPallet

Display parts on a pallet.

● **Function Call**

Function	LoadPartOnPallet(string controllerName, string variableName, IShapeBase partModel, string partName, Transform3D worldcoordinate, IShapeBase palletModel, string palletName, IEnumerable<ShapeRenderInfo> renderInfoList, IDictionary<string, object> variableValues = null)
Used in	Render()

● **Arguments**

Argument	Description	Notation example
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 <i>renderInfoList</i>	"partModel"
string partName	Specify the instance name of the part 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)
IShapeBase palletModel	Specify a variable that designates the 3D shape data for a pallet registered in <i>renderInfoList</i>	palletModel
string palletName	Specify the instance name of the pallet 3D shape data.	"Pallet"

Argument	Description	Notation example
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 *partName*, and the pallet's 3D shape data name *palletModel* and instance name *palletName* registered 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(string controllerName, string variableName, IShapeBase partModel, string name, IEnumerable<ShapeRenderInfo> renderInfoList, IDictionary<string, object> variableValues = null)
Used in	Render()

● **Arguments**

Argument	Description	Notation example
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 <i>renderInfoList</i>	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(string controllerName, string variableName, IShapeBase palletModel, string name, IEnumerable<ShapeRenderInfo> renderInfoList, IDictionary<string, object> variableValues = null)
Used in	Render()

● Arguments

Argument	Description	Notation example
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 <i>renderInfoList</i>	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(IShapeBase palletModel, IVisualizable targetModel, 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 <i>renderInfoList</i>	partModel
IVisualizable targetModel	Specify the variable corresponding to the 3D shape data that collides with the part.	targetModel

Argument	Description	Notation example
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(IShapeBase palletModel, IVisualizable targetModel, IEnumerable<ShapeRenderInfo> renderInfoList)
Used in	Render()

● **Arguments**

Argument	Description	Notation example
IShapeBase palletModel	Specify a variable that designates the 3D shape data for a pallet registered in <i>renderInfoList</i>	palletModel
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-1-2 Conveying Parts and Pallets

PushPart

Push the part by the cylinder.

● **Function Call**

Function	PushPart(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
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 <i>renderInfoList</i>	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(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
int stepId	Specify part/pallet state number. *1	1
string controllerName	Specify the Controller name by string.	"new_controller_0"

Argument	Description	Notation example
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 <i>renderInfoList</i>	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 is 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-1-3 Virtual Conveyor

MoveObjectOnBelt

Move the part or pallet on the conveyor.

● Function Call

Function	MoveObjectOnBelt(string controllerName, string variableName, IVisualizable beltModel, Transform3D distancePerSecond, IEnumerable<ShapeRenderInfo> renderInfoList, IDictionary<string, object> variableValues = null)
Used in	Render()

● Arguments

Argument	Description	Notation example
string controllerName	Specify the Controller name by string.	"new_controller_0"

Argument	Description	Notation example
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 distance-PerSecond	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 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.

A-1-4 Clamping Parts and Pallets

ClampPart

Pick up the part.

● **Function Call**

Function	ClampPart(int stepId, IVisualizable robotTool , IShapeBase partModel, IVisualizable locationModel, IEnumerable<ShapeRenderInfo> renderInfoList)
Used in	Render()

● **Arguments**

Argument	Description	Notation example
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 <i>renderInfoList</i>	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.	---

*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 *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(int stepId, IVisualizable robotTool , IShapeBase palletModel, IVisualizable locationModel, IEnumerable<ShapeRenderInfo> renderInfoList)
Used in	Render()

● **Arguments**

Argument	Description	Notation example
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 <i>renderInfoList</i>	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 <i>CreateRenderInfo</i> .	---

*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(int stepId, IVisualizable robotTool , IShapeBase partModel, IShapeBase palletModel, IEnumerable<ShapeRenderInfo> renderInfoList)
Used in	Render()

● Arguments

Argument	Description	Notation example
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 <i>renderInfoList</i>	partModel
IShapeBase palletModel	Specify a variable that designates the 3D shape data that is registered in <i>renderInfoList</i> 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 is 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(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
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 <i>renderInfoList</i>	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 is 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(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
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 <i>renderInfoList</i>	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(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
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 <i>renderInfoList</i>	partModel
IShapeBase palletModel	Specify a variable that designates the 3D shape data for a pallet registered in <i>renderInfoList</i>	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-1-5 Placing Parts and Pallets

ReleasePart

Place the part picked up.

● Function Call

Function	ReleasePart(int stepId, IVisualizable robotTool, IShapeBase partModel, IVisualizable locationModel, IEnumerable<ShapeRenderInfo> renderInfoList)
Used in	Render()

● Arguments

Argument	Description	Notation example
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 <i>renderInfoList</i> .	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(int stepId, IVisualizable robotTool, IShapeBase palletModel, IVisualizable locationModel, IEnumerable<ShapeRenderInfo> renderInfoList)
-----------------	--

Used in	Render()
----------------	----------

● **Arguments**

Argument	Description	Notation example
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 <i>renderInfoList</i> .	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 is 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(int stepId, IVisualizable robotTool, IShapeBase partModel, IShapeBase palletModel, IEnumerable<ShapeRenderInfo> renderInfoList)
Used in	Render()

● **Arguments**

Argument	Description	Notation example
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 <i>renderInfoList</i> .	partModel

Argument	Description	Notation example
IShapeBase palletModel	Specify a variable that indicates the pallet 3D shape data registered in <i>renderInfoList</i> .	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(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
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 <i>renderInfoList</i> .	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.	---

Argument	Description	Notation example
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 is 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(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
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 <i>renderInfoList</i> .	palletModel
IVisualizable locationModel	Specify a variable equivalent to the 3D shape data placing the pallet.	placeModel

Argument	Description	Notation example
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(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
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 <i>renderInfoList</i> .	partModel
IShapeBase palletModel	Specify a variable that indicates the pallet 3D shape data registered in <i>renderInfoList</i> .	palletModel

Argument	Description	Notation example
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-1-6 Changing and Detecting Status

SetNextStep

Change the value of *stepId*.

● **Function Call**

Function	SetNextStep(int stepId, string controllerName, string variableName, int value, IDictionary<string, object> variableValues = null)
Used in	Render()

● **Arguments**

Argument	Description	Notation example
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

Argument	Description	Notation example
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 is 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(IShapeBase partModel, IVisualizable mechanicalModel, int targetStep , string controllerName, string variableName, IEnumerable<ShapeRenderInfo> renderInfoList)
Used in	Render()

● **Arguments**

Argument	Description	Notation example
IShapeBase partModel	Specify a variable that indicates the Part/Pallet 3D shape data registered in <i>renderInfoList</i> .	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.	---

● **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.

DetectMechanicalComponetsCollision

Detect that the mechanical component collides with another mechanical component.

● Function Call

Function	DetectMechanicalComponetsCollision(string csName, int targetStep, string controllerName, string variableName)
Used in	Render()

● Arguments

Argument	Description	Notation example
string csName	Set the name(CollisionSourceName) , which is registered in Collision Filter, to <i>csName</i> .	"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"

● 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(string controllerName, string variableName, IVisualizable robotTool, IShapeBase partModel, IEnumerable<ShapeRenderInfo> renderInfoList)
Used in	Render()

● Arguments

Argument	Description	Notation example
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 <i>renderInfoList</i> .	partModel
IEnumerable<ShapeRenderInfo> renderInfoList	Specify the name of the Part/Pallet Instance List specified in CreateRenderInfo.	---

● **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()

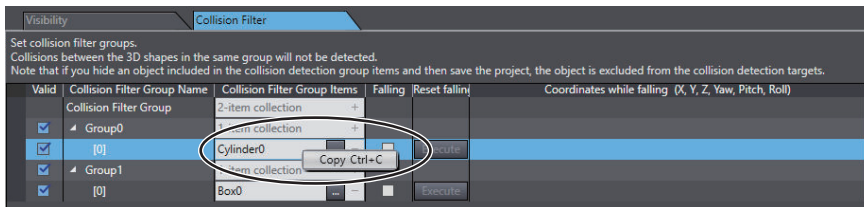
● **Arguments**

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-1-7 Sharing Variables with the Controller

GetBoolVariable

Read values of BOOL variables of the specified Controller.

● **Function Call**

Function	Bool GetBoolVariable(string controllerName, string variableName)
Used in	Render()

● **Arguments**

Argument	Description	Notation example
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(string controllerName, string variableName, bool value)
Used in	Render()

- **Arguments**

Argument	Description	Notation example
string controllerName	Specify the Controller name by string.	"new_controller_0"
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 values of DINT variables of the specified Controller.

- **Function Call**

Function	int GetIntegerVariable(string controllerName, string variableName)
Used in	Render()

- **Arguments**

Argument	Description	Notation example
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(string controllerName, string variableName, int value)
Used in	Render()

● **Arguments**

Argument	Description	Notation example
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.	---
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*.

GetCurrentControllerTime

Read the internal simulation time of the specified Controller.

● **Function Call**

Function	DateTime GetCurrentControllerTime(string controllerName)
Used in	Render()

● **Arguments**

Item	Description	Notation example
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.

Get**Variable

Read values of 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-32 below for the data types of variables and specific function names that you can specify.

● **Function Call**

Function	sbyte GetSintVariable(string controllerName, string variableName) short GetIntVariable(string controllerName, string variableName) int GetDintVariable(string controllerName, string variableName) long GetLintVariable(string controllerName, string variableName) byte GetUsintVariable(string controllerName, string variableName) ushort GetUintVariable(string controllerName, string variableName) uint GetUdintVariable(string controllerName, string variableName) ulong GetUlintVariable(string controllerName, string variableName)
-----------------	--

Used in	Render()
----------------	----------

● **Arguments**

Item	Description	Notation example
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-33 below for the data types of variables and specific function names that you can specify.

● **Function Call**

Function	SetSintVariable(string controllerName, string variableName, object value) SetIntVariable(string controllerName, string variableName, object value) SetDintVariable(string controllerName, string variableName, object value) SetLintVariable(string controllerName, string variableName, object value) SetUsintVariable(string controllerName, string variableName, object value) SetUintVariable(string controllerName, string variableName, object value) SetUdintVariable(string controllerName, string variableName, object value) SetUlintVariable(string controllerName, string variableName, object value)
Used in	Render()

● **Arguments**

Item	Description	Notation example
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. SetSintVariable: sbyte SetIntVariable: short SetDintVariable: int SetLintVariable: long SetUsintVariable: byte SetUintVariable: ushort SetUdintVariable: uint 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.

CreateGetVariableList

Create a new read variable list, which is used to read multiple variables from the Controller.

● Function Call

Function	<code>IList<Tuple<string, Type>> CreateGetVariableList()</code>
Used in	<code>Render()</code>

● 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".

```

65 public override void Render(IEnumerable<ShapeRenderInfo> renderInfoList) {
66     this.InitializeObject(renderInfoList);
67
68     var getVariableList = this.CreateGetVariableList();
69     this.AddToGetVariableList("boolVar1", "BOOL", getVariableList);
70     this.AddToGetVariableList("boolVar2", "BOOL", getVariableList);
71     this.AddToGetVariableList("boolVar3", "BOOL", getVariableList);
72
73     var variableValueList = this.GetVariableValues("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 variable 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.

AddToGetVariableList

Add the variable to read from the Controller to the read variable list.

● Function Call

Function	<code>bool AddToGetVariableList(string variableName, string iecDataType, IList<Tuple<string, Type>> getVariableList, bool isShowMessage = true)</code>
Used in	<code>Render()</code>

● 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.	---

Item	Description	Notation example
bool <code>isShowErrorMessage</code>	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(string controllerName, IList<Tuple<string, Type>> getVariableList)
Used in	Render()

● **Arguments**

Item	Description	Notation example
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.

```

65 public override void Render(IEnumerable<ShapeRenderInfo> renderInfoList) {
66     this.InitializeObject(renderInfoList);
67
68     var getVariableList = this.CreateGetVariableList();
69     this.AddToGetVariableList("boolVar1", "BOOL", getVariableList);
70     this.AddToGetVariableList("boolVar2", "BOOL", getVariableList);
71     this.AddToGetVariableList("boolVar3", "BOOL", getVariableList);
72
73     var variableValueList = this.GetVariableValues("new_Controller_0", getVariableList);
74     var result = (bool)this.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.

```

61  /// <summary>
62  /// Called to render the Shape Script object
63  /// </summary>
64  /// <returns>The list of shapes to render</returns>
65  public override void Render(IEnumerable<ShapeRenderInfo> renderInfoList) {
66      this.InitializeObject(renderInfoList);
67
68      var setVariableList = this.CreateSetVariableList();
69      this.AddToSetVariableList("intVar1", "INT", 1, setVariableList);
70      this.AddToSetVariableList("intVar2", "INT", 2, setVariableList);
71      this.AddToSetVariableList("intVar3", "INT", 3, setVariableList);
72
73      this.SetVariableValues("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 variable 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 <i>variableName</i>	Specify the Controller variable name by string.	---
string <i>iecDataType</i>	Specify the data type of the variable by string.	"BOOL"
object <i>setValue</i>	Specify the value to write to the Controller variable.	---
ICollection<Tuple<string, object>> <i>setVariableList</i>	Specify a reference to the write variable list.	---
bool <i>isShowErrorMessage</i>	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(string controllerName, ICollection<Tuple<string, object>> setVariableList)
Used in	Render()

● Arguments

Item	Description	Notation example
string <i>controllerName</i>	Specify the Controller name by string.	"new_controller_0"
ICollection<Tuple<string, object>> <i>setVariableList</i>	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.

A-1-8 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(IProcessManager processManager, IBelt beltModel, IEnumerable<ShapeRenderInfo> renderInfoList)
Used in	Render()

● Arguments

Argument	Description	Notation example
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/ProcessManager"];
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(IProcessManager processManager, IBelt beltModel, IEnumerable<ShapeRenderInfo> renderInfoList)
Used in	Render()

● Arguments

Argument	Description	Notation example
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(string controllerName, string variableNameDX, string variableNameDY, string variableNameDZ, IShapeBase partModel, IEnumerable<ShapeRenderInfo> renderInfoList)
Used in	Render()

● Arguments

Argument	Description	Notation example
string ControllerName	Specify the Controller name by string.	"new_Controller_0"
string variableNameDX	Specify the LREAL global array variable name of the Controller that stores the X position of the part by string.	"PositionX"
string variableNameDY	Specify the LREAL global array variable name of the Controller that stores the Y position of the part by string.	"PositionY"
string variableNameDZ	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 <i>renderInfoList</i> .	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 <i>renderInfoList</i> .	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	UpdatePartDataFromPackManager(IShapeBase partModel, IShapeBase palletModel, 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 <i>renderInfoList</i> .	partModel
IShapeBase palletModel	Specify a variable that indicates the pallet 3D shape data registered in <i>renderInfoList</i> .	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-2 Differences between the Simulator and the Physical Controller

A

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	<p>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.</p> <ul style="list-style-type: none"> • Thickness adjustment for the optical axis • Simulation of beam reflection • Simulation of hysteresis



Index



Index

A	
AddToGetVariableList.....	A-34
AddToSetVariableList.....	A-37
C	
ClampPallet.....	A-17
ClampPalletBySignal.....	A-19
ClampPart.....	A-16
ClampPartBySignal.....	A-18
ClampPartOnPallet.....	A-18
ClampPartOnPalletBySignal.....	A-20
CreateGetVariableList.....	A-34
CreateMultipleObjects.....	A-6
CreateMultipleObjectsForNonCameraLatch.....	A-7
CreateObject.....	A-5
CreateSetVariableList.....	A-37
D	
DetectMechanicalComponetsCollision.....	A-29
DetectPartCollision.....	A-28
G	
Get**Variable.....	A-32
GetBoolVariable.....	A-30
GetCollidingSourceNames.....	A-30
GetCurrentControllerTime.....	A-32
GetIntegerVariable.....	A-31
GetValueFromVariableValueList.....	A-35
GetVariableValues.....	A-35
L	
LoadPallet.....	A-9
LoadPart.....	A-8
LoadPartOnPallet.....	A-10
M	
MoveObjectOnBelt.....	A-15
MovePalletOnPackManagerSensorLatchBelt.....	A-39
MovePartOnPackManagerSensorLatchBelt.....	A-39
P	
PushPallet.....	A-14
PushPart.....	A-13
R	
ReleasePallet.....	A-22
ReleasePalletBySignal.....	A-25
ReleasePart.....	A-22
ReleasePartBySignal.....	A-24
ReleasePartOnPalletBySignal.....	A-26
ReleasePartOnPallet.....	A-23
S	
Set**Variable.....	A-33
SetBoolVariable.....	A-31
SetClampStatus.....	A-29
SetIntegerVariable.....	A-31
SetNextStep.....	A-27
SetPackManagerPartPosition.....	A-40
SetVariableValues.....	A-38
Sysmac Studio Option License.....	1-3
Sysmac Studio Options.....	2-3
U	
UnloadCollidingPallet.....	A-13
UnloadCollidingPart.....	A-12
UnloadPallet.....	A-12
UnloadPart.....	A-11
UpdatePartDataFromPackManager.....	A-41
UpdatePartDataOnPalletFromPackManager.....	A-41

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.

No. 438A Alexandra Road # 05-05/08 (Lobby 2),
Alexandra Technopark,
Singapore 119967
Tel: (65) 6835-3011/Fax: (65) 6835-2711

OMRON (CHINA) CO., LTD.

Room 2211, Bank of China Tower,
200 Yin Cheng Zhong Road,
PuDong New Area, Shanghai, 200120, China
Tel: (86) 21-5037-2222/Fax: (86) 21-5037-2200

Authorized Distributor:

© OMRON Corporation 2020 All Rights Reserved.
In the interest of product improvement,
specifications are subject to change without notice.

Cat. No. W618-E1-03

1020