Kawasaki Heavy Industries, Ltd.

KRCC providers

User's Guide

Version 1.2.0

March 1, 2023

NOTE: This document has been machine translated.

[Revision History]

Version	Date	Content
1.0.0	2021-11-24	First edition
1.1.0	2022-08-10	Change description due to CaoValiable Signal type and POSE type setting function enabled Revision of description due to change of Sample programme
1.2.0	2023-03-01	The error history acquisition function has been implemented. The operation history acquisition function has been implemented. Errors have been corrected.

[Compatible models]

Model	Version	Notes
Kawasaki Heavy Industries F series Controller		
Kawasaki Heavy Industries E series Controller		

[Operation check model]

Model	Version	Notes

Contents

1. Introduction

This manual is a user's guide for providers that use Kawasaki Heavy Industries' Robot Communication Library KRCC to connect the Robot Controller via TCP to obtain robot operation and various parameters. Figure 1-1 shows the overall configuration of this provider and the device. The providers are referred to as KRCC providers.

Fig. 1-1 Configuration Diagram

Figure 1-2 shows the correspondence between this provider and each device.
(* This is an example. It does not represent everything.)

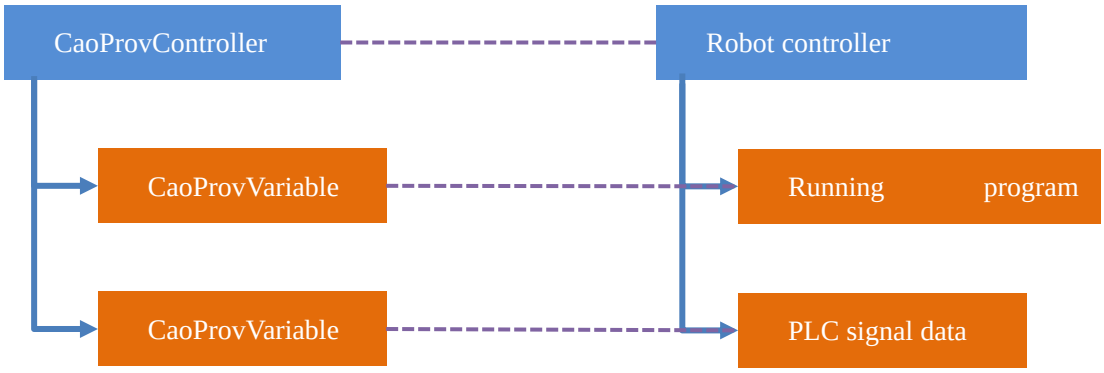

Fig. 1-2 Provider configuration and device information

2. Setting Up Your Environment for Application Development

2.1. Connection between Robot Controller and Client PC

The robot controller and client PC connect using TCP/IP protocol.

The communication library KRCC made by Kawasaki Heavy Industries is used for communication.

2.2. Setting up a PC development environment

2.2.1. Installing KRCC Providers Manually

If you install KRCC providers manually, you must register the registry as shown below. To register the registry, start the command prompt with administrator privileges and execute regsvr32 command. When executing the command, either move to the path where the file is located or specify the file path.

Table 2-1 TableKRCC Providers

File name	CaoProvKawasakiKRCC.dll
ProgID	CaoProv.Kawasaki.KRCC
Registry registration	Regsvr32 CaoProvKawasakiKRCC.dll
Deletion of registry registration	Regsvr32 /u CaoProvKawasakiKRCC.dll

3. Command Reference

3.1. Method/Property List

Table 3-1 List of methods and properties

Category	Methods/Properties[1]		Function	Reference
CaoWorkspace				
	AddController	M	Connected to controller	P.8
CaoController				
	Index	P	Obtaining the Controller Number	P.10
	Name	P	Get Controller Name	P.10
	GetVariableNames	M	Get a list of variable names that can be connected	P.10
	Variables	P	Retrieving Variable Collections Holded by the Controller	P.10
	AddVariable	M	Adding Variable Objects	P.11
	Execute	M	Execute Extended Commands	P.12
CaoVariable				
	Index	P	Obtaining Variable Numbers	P.19
	Name	P	Retrieving Variable Names	P.19
	Value	P	Get/set value	P.19

3.2. Method properties

3.2.1. CaoWorkspace classes

3.2.1.1. AddController method

In CaoWorkspace, add a controller object. KRCC controller provider refers to the parameters passed when AddController method is executed and connects to the corresponding robot controller. The following are the specifics of AddController method:

Format

```
AddController
(
"<controller name>",          // Controller name (optional)
" CaoProv.Kawasaki.KRCC",     // Provider name (fixed)
```

[1] M:Indicates methods, P: properties, and E: events, respectively.

```
"<machine name>",                    // Provider execution machine name (unused)
"<Option>"                 // Option string
)
```

Option

The following options are specified in the option string: The option string is a string consisting of the following options separated by a comma (,).

Option	Required	Description	Value Range	Default value
CONN	✓	Specify the address and port of the robot controller to connect to.		
HostName	-	Specifies the host name of the robot controller as a string.		As
Timeout	-	Specifies the timeout in milliseconds for connecting to the robot controller.	0-4294967295	1000

Sample usage (C#)

```
// Engine
ORiN2.ManagedCAO.CCaoEngine engine = new ORiN2.ManagedCAO.CCaoEngine();
// Workspace
ORiN2.ManagedCAO.CCaoWorkspace workspace = engine.AddWorkspace("NewWrks", "");
// Controller
ORiN2.ManagedCAO.CCaoController controller= workspace.AddController(
    "CONTROLLER",
    "CaoProv.***. ***",
    "",
    "CONN=TCP:192.168.0.2");
```

3.2.1.1.1. CONN Optional

The following is a Conn optional connection parameter string: Here, braces ("[]") are optional, and the underlined part in the description of each parameter indicates the default value when no options are specified.

"CONN=TCP:<IP>[:<Port>]"

<IP> : Destination IP address.

<Port> : Destination port . (9105)

Port defaults to the port number when connecting to the simulator.

Please note that the actual machine normally listens on port 23.

3.2.2. CaoController classes

3.2.2.1. Index Properties

Gets the controller number as a Long type (4-byte integer type). This number is used to identify the corresponding controller in CaoWorkspace.

Sample usage (C#)

```
// Get Index
Long index = controller.Index;
```

3.2.2.2. Name Properties

Retrieves the controller name specified by AddController method of CaoWorkspace class.

Sample usage (C#)

```
System.Diagnostics.Debug.WriteLine(controller.Name);
```

3.2.2.3. GetVariableNames method

Gets a list of variable names that can be connected.

Format

```
GetVariableNames
(
"<Option>"                    // Option string
)
```

Option

KRCC providers do not use optional strings.

Sample usage (C#)

```
// Get variable name list
String[] variableNames = controller.GetVariableNames("");
```

3.2.2.4. Variables Properties

Gets a collection of variables that the controller holds.

Sample usage (C#)

```
// Variable Collection Retrieval
ORiN2.ManagedCAO.CCaoVariables variables = controller.Variables;
```

```
// Variable acquisition
ORiN2.ManagedCAO.CCaoVariable variable = variables[0];
```

3.2.2.5. AddVariable method

Adds a variable object to CaoController. Only the variable names shown in 3.3.2 can be used.3.3.2 AddVariable is specifi ed as follows.

Format

```
AddVariable
(
"<variable name>",                    // Variable Name
"<Option>"              // Option string (optional)
)
```

3.2.2.6. Execute method

Execute CaoController extension. Execute is specifi ed as follows.

Format

```
Execute
(
"<extension command name>",          // Extended command name
"<Option string>"                    // Option string (optional)
)
```

The following is a list of extended commands that can be specified in Execute. The usage examples are described in detail in the extended commands.

Command	Description	Reference
Execute	Execute the program on the robot controller.	P.13
Abort	Stops the execution of a program on the robot controller.	P.13
Hold	Pauses the execution of a program on the robot controller.	P.14
Continue	Resumes program execution on the robot controller.	P.14
EReset	Clears the error of the robot controller.	P.14
SetSpeed	Sets the program execution speed on the robot controller.	P.15
Save	Save the program on the robot controller on the local PC.	P.15
Load	The program saved by Save command is transferred to the robot controller.	P.16
GetErrLog	Acquires the error history.	P.16
GetErrLogSingle	Acquires the specified error information from the error history.	P.17
SaveErrLog	The error history is saved on the local PC.	P.17
SaveOpLog	Saves the operation history on the local PC.	P.18

3.2.2.6.1. Execute Commands

Execute the program on the robot controller.

The argument specification method is VT_ARRAY | You can specify the program name and the number of executions in VT_VARIANT, or you can specify the program name in VT_BSTR.

The following are the arguments and return values:

Item	Type Description		
Argument 1	VT_ARRAY | VT_VARIANT		
	0	VT_BSTR	Specify the name of the program to be executed.
	1	VT_I4	Specify the number of executions. If a negative value is specified, execution continues indefinitely.
Argument 2	Specifies the program name to be executed by VT_BSTR. The operation is the same as when-1 is specified as the second argument in argument-1.		
Return value	None		

Sample usage (C#)

```
// Execute Execute
Object[] opt = {"test", -1};
Controller.Execute("Execute", opt);
// For continuous execution, you can also make the following calls
// controller.Execute("Execute", "test");
```

3.2.2.6.2. Abort Commands

Stops the execution of a program on the robot controller.

There are no arguments and return values.

Sample usage (C#)

```
// Execute Abort
Controller.Execute("Abort", null);
```

3.2.2.6.3. Hold Commands

Pauses the execution of a program on the robot controller.

There are no arguments and return values.

Sample usage (C#)

```
// Execute Hold
Controller.Execute("Hold", null);
```

3.2.2.6.4. Continue Commands

Resumes execution of a stopped program on the robot controller.

There are no arguments and return values.

Sample usage (C#)

```
// Execute Continue
Controller.Execute("Continue", null);
```

3.2.2.6.5. EReset Commands

Clears the error status of the robot controller.

There are no arguments and return values.

Sample usage (C#)

```
// Execute EReset
Controller.Execute("EReset", null);
```

3.2.2.6.6. SetSpeed Commands

Sets the program execution speed on the robot controller.

The following are the arguments and return values:

Item	Type Description
Argument	VT_I4 specifies the program execution speed. The range of values is 1-100.
Return value	None

Sample usage (C#)

```
// Execute SetSpeed
Controller.Execute("SetSpeed", 30);
```

3.2.2.6.7. Save Commands

Save the program on the robot controller on the local PC.

The following are the arguments and return values:

Item	Type Description
Argument	Specify the path of the file to be saved in VT_BSTR.
Return value	None

Sample usage (C#)

```
// Execute Save
Controller.Execute("Save", @"C:¥Tmp¥Save.txt");
```

3.2.2.6.8. Load Commands

The program saved by Save command is transferred to the robot controller.

The following are the arguments and return values:

Item	Type Description
Argument	Specifies the path to the file to be transferred with VT_BSTR.
Return value	None

Sample usage (C#)

```
// Execute Load
Controller.Execute("Load", @"C:¥Tmp¥Save.txt");
```

3.2.2.6.9. GetErrLog Command

Acquires the error history.

The error history is numbered from newest to oldest, and the number of the most recent error is 1.

The following are the arguments and return values:

Item	Type Description
Argument	Specify the number of error history items to be acquired as a numerical value. Specifying 0 returns the history of all stored errors. If it is omitted, 0 is specified.
Return value	VT_BSTR| Returns the error information of the number specified by VT_ARRAY. If the number of error logs is less than the specified number, an empty string is returned to the missing elements.

Sample usage (C#)

```
// Acquire the last five error logs
string[] errList = controller.Execute("GetErrLog", 5) as string[];
```

3.2.2.6.10. GetErrLogSingle Command

Acquires the error history of the specified number.

The error history is numbered from newest to oldest, and the number of the most recent error is 1.

The following are the arguments and return values:

Item	Type Description
Argument	Specifies the number of the error history to be acquired as a numeric value. If it is omitted, 1 is specified.
Return value	Returns the error information of the number specified by VT_BSTR. If there is no error information of the specified number, an execution error occurs.

Sample usage (C#)

```
// Acquire the error number 1(latest) from the error history.
string error = controller.Execute("GetErrLogSingle", 1) as string;
```

3.2.2.6.11. SaveErrLog Command

The error history on the robot controller is saved on the local PC.

The following are the arguments and return values:

Item	Type Description
Argument	Specify the path of the file to be saved with VT_BSTR.
Return value	None

Sample usage (C#)

```
// Run SaveErrLog
controller.Execute("SaveErrLog", @"C:¥Tmp¥ErrLog.txt");
```

3.2.2.6.12. SaveOpLog Command

Saves the operating history on the robot controller on the local PC.

The following are the arguments and return values:

Item	Type Description
Argument	Specify the path of the file to be saved with VT_BSTR.
Return value	None

Sample usage (C#)

```
// Run SaveOpLog
controller.Execute("SaveOpLog", @"C:¥Tmp¥OpLog.txt");
```

3.2.3. CaoVariable classes

3.2.3.1. Index Properties

Gets the variable number as a Long type (4-byte integer type). This number indicates the number that identifies the variable in CaoController.

Sample usage (C#)

```
// Get Index
Int index = caoVar.Index;
```

3.2.3.2. Name Properties

Retrieves the variable name specified by AddVariable method of CaoContrller class.

Sample usage (C#)

```
System.Diagnostics.Debug.WriteLine(caoVar.Name);
```

3.2.3.3. Value Properties

Acquires/sets data for the connected robot controller. The behavior depends on the variable name. For details, refer to section 3.3, Variable List.3.3Variable list

3.3. Variable list

Defines a list of variables that can be used in each class. Variables refer to objects of CaoVariable classes.

3.3.1. System and User Variables

There are two types of variables in the provider: system variables and user variables.

System Variables

A variable that accesses only the information in the object that holds the variable. System variables are often static data. System variables are preceded by "@".

e.g. provider version, device manufacturer, serial number

User variable

A variable that you can use to specify what information you want to access when you create a variable by using an option string.

User variables in KRCC providers allow you to get the values of global variables and signals used in robot controllers.

3.3.2. CaoController class-variable

Variable Name	Description	Value		Reference
		Get	Put	
@MAKER_NAME	Obtain the manufacturer's name.	✓	-	P.20
@VERSION	Get the DLL version.	✓	-	P.20
@CURRENT_POSITION	Gets the position of the present arm in X, Y, Z, O, A, T format.	✓	-	P.21
@CURRENT_ANGLE	Gets the position of the present arm in the form of an axis displacement (JT1, JT2, JT3, JT4, JT5, JT6).	✓	-	P.21
Any name that starts with something other than @	Gets the values of global variables and signals.	✓	※1	P.22

※1 The operation varies depending on the type specified in Type.

3.3.2.1. @MAKER_NAME

Obtain the manufacturer's name.

Data type

Type Description	
VT_BSTR	Obtain the manufacturer's name.

Sample usage (C#)

```
// Add Variable
ORiN2.ManagedCAO.CCaoVariable var = controller.AddVariable("@MAKER_NAME","");
// Acquisition of Values
String value = var.Value as string;
```

3.3.2.2. @VERSION

Gets the DLL version.

Data type

Type Description	
VT_BSTR	Get the DLL version.

Sample usage (C#)

```
// Add Variable
ORiN2.ManagedCAO.CCaoVariable var = controller.AddVariable("@VERSION","");
// Acquisition of Values
```

```
String value = var.Value as string;
```

3.3.2.3. @CURRENT_POSITION

Gets the position of the present arm in X, Y, Z, O, A, T format.

The acquired value is an array of VT_R8 elements corresponding to the number of robot axes.

Data type

Type Description	
VT_ARRAY | VT_R8	X, Y, Z, O, A, T of the position of the present arm. When the number of robot axes is larger than 6 axes, JT7 (up to the maximum JT18 for the number of robot axes) is added.

Sample usage (C#)

```
// Add Variable
ORiN2.ManagedCAO.CCaoVariable var = controller.AddVariable("@CURRENT_POSITION","");
// Acquisition of Values
Double[] value = var.Value as double[];
Double x = value[0];
```

3.3.2.4. @CURRENT_ANGLE

Gets the position of the present arm in the form of an axis displacement (JT1, JT2, JT3, JT4, JT5, JT6).

The acquired value is an array of VT_R8 elements corresponding to the number of robot axes.

Data type

Type Description	
VT_ARRAY | VT_R8	The respective axis displacement (JT1, JT2, JT3, JT4, JT5, JT6) of the position of the present arm. When the number of robot axes is larger than 6 axes, JT7 (up to the maximum JT18 for the number of robot axes) is added.

Sample usage (C#)

```
// Add Variable
ORiN2.ManagedCAO.CCaoVariable var = controller.AddVariable("@CURRENT_ANGLE","");
// Acquisition of Values
Double[] value = var.Value as double[];
Double jt1 = value[0];
```

3.3.2.5. @NORMAL_STATUS

Get the current error status.

Data type

Type Description	
VT_BOOL	True if no error has occurred, false if an error has occurred.

Sample usage (C#)

```
// Add Variable
ORiN2.ManagedCAO.CCaoVariable var = controller.AddVariable("@NORMAL_STATUS","");
// Get Value
bool value = (bool)var.Value;
```

3.3.2.6. @ERROR_NUMBER

Get the number of the error that is currently occurring.

Data type

Type Description	
VT_I4	VT_I4 Number of the error that is currently occurring. If no error occurs, the number is 0.

Sample usage (C#)

```
// Add Variable
ORiN2.ManagedCAO.CCaoVariable var = controller.AddVariable("@ERROR_NUMBER","");
// Get Value
int value = (int)var.Value;
```

3.3.2.7. @ERROR_DESCRIPTION

Get a description of the error that is currently occurring.

Data type

Type Description	
VT_BSTR	A description of the error that is currently occurring. If no error occurs, it is an empty string.

Sample usage (C#)

```
// Add Variable
ORiN2.ManagedCAO.CCaoVariable var = controller.AddVariable("@ERROR_DESCRIPTION","");
// Get Value
```

```
string value = var.Value as string;
```

3.3.2.8. User variable

Variables whose names do not begin with "@" are treated as user variables.

User variables get the values of global variables and signals used in the robot controller.

Specify the name of the variable or signal in Name option of the option string and the variable or signal type in Type property.

Option

Option	Required	Description	Value Range	Default value
Name	✓	Specifies the name of a variable or signal. The value of the entire array variable cannot be obtained. To obtain the value of the array variable, specify the value up to the index. e.g., Name = (a[0]), Type=REAL		
Type	○	Specifies the variable or signal type. Refer to Table 3-2 Type Options for the specifiable values.		

Data type

Refer to Table 3-2 Type Options List for the type of 2 Type to be retrieved.

Sample usage (C#)

```
// Add Variable
ORiN2.ManagedCAO.CCaoVariable var =
    Controller.AddVariable("VAR1","Name=a,Type=REAL");
// Retrieving Values
Double value = System.Convert.ToDouble(var.Value);
// Setting Values
Var.Value = 12.34;
```

Table 3-2 Type Options

Type Optional Values	Variable or signal type to be acquired/set	Value Type	Put_Value
SIGNAL	Signal	VT_BOOL	✓
STRING	String variable	VT_BSTR	✓
REAL	Real variable	VT_R8	✓
INT	Integer variable	VT_I4	✓
POSE	Position variable	VT_ARRAY | VT_R8	✓

4. Programming by KRCC providers

With KRCC providers, you can connect the client PC to the robot controller as follows:

- Creating a CaoEngine
- Creating a CaoWorkspace
- Creating a CaoController

After connecting to the robot controller, you can access the robot controller information using Execute method of CaoController or by generating a CaoVariabe object.

4.1. Sample Programming

This section describes, as an example, a program that reads fixed data from the robot controller and executes finger-low-frequency processing.

Table 4-1 and Fig. 4-1 describe the requirements and flow of the sample program, respectively.

Table 4-1 Sample program requirements

Requirements	Description
Host	Connect with a TCP/IP
	The destination IP address is 127.0.0.1.
	The destination port number is 9105.
Process Description	Obtaining and Displaying Data for the System Variables below @MAKER_NAME @VERSION @CURRENT_POSITION @CURRENT_ANGLE
	Repeat the specified number of operations below. − Repeat connection/disconnection with the controller - Repeat retrieval of setting variable values - Repeat program execution and interruption - Repeat program pause and resume - Changing in operation speed - Save / Load
	Exit when key is pressed

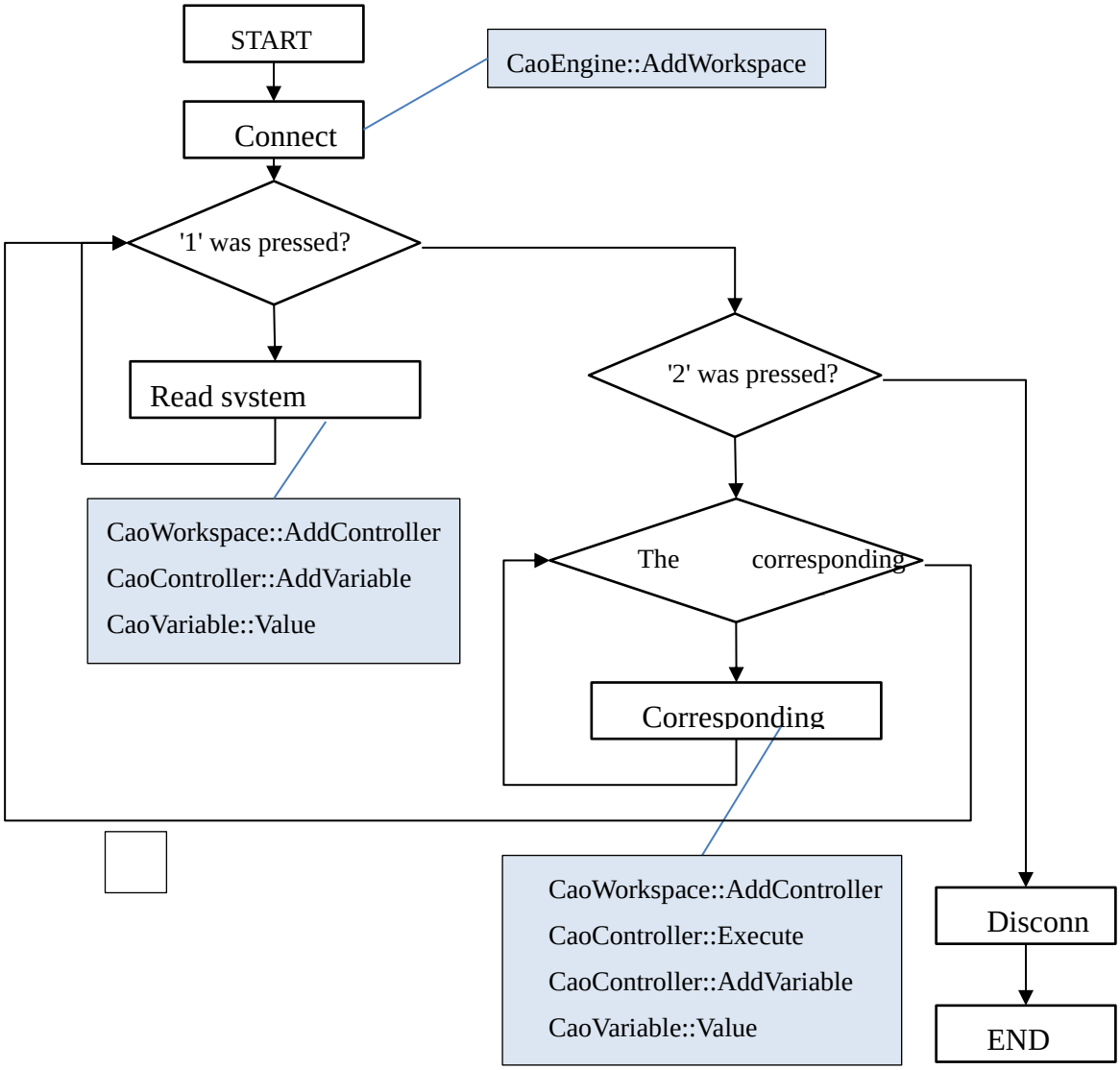

Fig. 4-1 Flow of program for displaying current position

Specific codes are given in the following sections.

4.1.1. Sample program

The entire sample program is shown below.

Sample **Program.cs**

```
Using System;
Using System.Collections.Generic;
Using System.Threading;
Using ORiN2.ManagedCAO;

Namespace Sample
{
    /// <summary>
    /// Kawasaki Heavy Industries F series Controller Provider Sample Program
    /// </summary>
    /// <remarks>
    /// </remarks>
    Class Program
    {
        Static string CTL_NAME = "SampleController";
        Static string CTL_PROV = "CaoProv.Kawasaki.KRCC";
        // The access point is set to PCAS of your personal computer. Please change it if necessary.
        Static string CTL_OPTION = "CONN=TCP:127.0.0.1";
        Static string SV_PATH = "C:¥¥temp¥¥orin¥¥FCON_DATA.pg";
        Static string LD_PATH = "C:¥¥temp¥¥orin¥¥orin.pg";

        Static void Main(string[] args)
        {
            // Create a CCaoEngine
            // using clause automatically invokes and releases Dispose when the scope expires
            Using (CCaoEngine engine = new CCaoEngine())
            {
                Bool run_fg = true;
                While (run_fg)
                {
                    Console.WriteLine("----------------------------------------------------------------");
                    Console.WriteLine("Controler Check ");
                    Console.WriteLine("");
                    Console.WriteLine("1:Variables check 2:Loop Running Check other: end");
```

```
                Console.WriteLine("-----------------------------------------------------------------");

                String key =Console.ReadLine();
                If (int.TryParse(key,out int result))
                {
                   Switch(result)
                   {
                        Case 1:
                            ValiableCheck(engine);
                            Break;

                        Case 2:
                            LoopCheck(engine);
                            Break;

                        Default:
                            Run_fg = false;
                            Break;
                    }
                }
                Else
                {
                     Run_fg = false;
                }
            }

        }
}

Static void valiableCheck(CCaoEngine engine)
{
    CCaoController controller = engine.Workspaces[0].AddController(
                                                    CTL_NAME, CTL_PROV, null, CTL_OPTION);

    // System Valiable Set & Read

    CCaoVariable val_sName = controller.AddVariable("@MAKER_NAME", "");
```

```
String s_val = val_sName.Value.ToString();
Console.WriteLine($"@MAKER_NAME : [{s_val}] OK");

CCaoVariable val_sVer = controller.AddVariable("@VERSION", "");
s_val = val_sVer.Value.ToString();
Console.WriteLine($"@VERSION : [{s_val}] OK");

CCaoVariable val_sPosit = controller.AddVariable("@CURRENT_POSITION", "");
Double[] p_val = val_sPosit.Value as double[];
s_val = "";
For (int i = 0; i < p_val.Length; i++) s_val += $" {p_val[i]},";
Console.WriteLine($"@CURRENT_POSITION : [{s_val}] OK");

CCaoVariable val_sAngle = controller.AddVariable("@CURRENT_ANGLE", "");
Double[] a_val = val_sAngle.Value as double[];
s_val = "";
For (int i = 0; i < a_val.Length; i++) s_val += $" {a_val[i]},";
Console.WriteLine($"@CURRENT_ANGLE : [{s_val}] OK");

// User Valiable
// Below is an example of defining user-specified variables and setting Get/Put.
// Please refer if necessary

//<Int type variable>
//CCaoVariable val_uInt=controller.AddVariable(
                                        "U_IntData","Name=@IntData, Type = INT");
//val_uInt.Value = 123;
//s_val = val_uInt.Value.ToString();
//Console.WriteLine($"U_IntData : [{s_val}] OK");



//<REAL type>
//CCaoVariable val_uReal = controller.AddVariable(
                                          "U_RealData", "Name=RealData,Type=REAL");
//val_uReal.Value = 123.4567;
//s_val = val_uReal.Value.ToString();
//Console.WriteLine($"U_RealData : [{s_val}] OK");
```

```
        //<STRING type>
        //CCaoVariable val_uStr = controller.AddVariable(
                                                    "U_StrData", "Name=$StrData,Type=STRING");
        //val_uStr.Value = "TEST_Data";
        //s_val = val_uStr.Value.ToString();
        //Console.WriteLine($"U_StrData : [{s_val}] OK");



        //<POSE type>
        //CCaoVariable val_uPose = controller.AddVariable(
                                                    "U_PosData", "Name=PosData,Type=POSE");
        //List<double> inp_val= new List<double>() {1.1, 2.2, 3.3, 4.4, 5.5, 6.6};
        //val_uPose.Value = (object)inp_val;
        //double[] put_val = val_uPose.Value as double[];
        //s_val = "";
        //for (int i = 0; i < put_val.Length; i++) s_val += $" {put_val[i]},";
        //Console.WriteLine($"@CURRENT_ANGLE : [{s_val}] OK");

    }

    Static void loopCheck(CCaoEngine engine)
    {
        CCaoController controller;
        CCaoVariable variable;
        Bool run_fg = true;
        Int idx;
        System.Diagnostics.Process proc =
                                                System.Diagnostics.Process.GetCurrentProcess();
        While (run_fg)
        {
            Console.WriteLine("< 2.Loop Check >");
            Console.WriteLine(" 1 - Controller add and remove.");
            Console.WriteLine(" 2 - Variable add and remove.");
            Console.WriteLine(" 3 - Variable get value.");
            Console.WriteLine(" 4 - Program execute and abort.");
```

```
                Console.WriteLine(" 5 - Program hold and continue.");
                Console.WriteLine(" 6 - Set speed.");
                Console.WriteLine(" 7 - Save.");
                Console.WriteLine(" 8 - Load.");
                Console.WriteLine(" other - back to main");
                Console.WriteLine("-------------------------------------");
                Console.WriteLine("input {No} {count}");

                String keys = Console.ReadLine();
                String[] key = keys.Split(' ');

                If (key.Length < 2) return;
                If (!int.TryParse(key[0], out int ikey)) return;
                If (!int.TryParse(key[1], out int icnt)) return;
                Proc.Refresh();

Console.WriteLine($"Memory(.WorkingSet:{proc.WorkingSet64:N0} .VirtualMemory:{proc.VirtualMemoryS
ize64:N0})");
                Switch (ikey)
                {
                    Case 1:

                        Console.WriteLine(" Controller add and remove start.");
                        // Amount of virtual memory ex) 290,201,600
                        For (int i=0;i<icnt;i++)
                        {
                            Console.WriteLine(
  $"CController Add Name:[{CTL_NAME}] ProvName:[{CTL_PROV}],Option :[{CTL_OPTION}]");
                            Controller = engine.Workspaces[0].AddController(
                                                CTL_NAME, CTL_PROV, null, CTL_OPTION);
                            Idx = controller.Index;
                            Thread.Sleep(100);
                            Console.WriteLine($"CController Renove:Name:[{CTL_NAME}]");
                            If (!engine.Workspaces[0].Controllers.Remove(CTL_NAME))
                            {
                                Console.WriteLine(" Remove error!");
                                Run_fg = false;
```

```
                Break;
            }
            Console.WriteLine($"{i}: OK");
            Thread.Sleep(100);

        }
        Console.WriteLine(" Controller add and remove fin.");
        Break;

    Case 2:
        Controller = engine.Workspaces[0].AddController(
                                        CTL_NAME, CTL_PROV, null, CTL_OPTION);
        Idx = controller.Index;

        Console.WriteLine(" Variable add and remove Start.");
        For (int i = 0; i < icnt; i++)
        {
            Console.WriteLine("AddVariable Name:[tst_str], option:[Name =$tst_str, Type =
STRING]");
            Variable =controller.AddVariable(
                                            "tst_str", "Name=$tst_str,Type=STRING");
            Thread.Sleep(100);
            Console.WriteLine("RemoveVariable Name:[tst_str]");
            Controller.Variables.Remove("tst_str");
            Thread.Sleep(100);
            Console.WriteLine($"{i}: OK");
        }
        Console.WriteLine(" Variable add and remove fin.");
        If (!engine.Workspaces[0].Controllers.Remove(idx))
        {
            Console.WriteLine(" Remove error!");
            Run_fg = false;
        }
        Break;

    Case 3:
        Controller = engine.Workspaces[0].AddController(
```

```
                                        CTL_NAME, CTL_PROV, null, CTL_OPTION);
    Idx = controller.Index;
    Variable=controller.AddVariable(
                                            "tst_str", "Name=$tst_str,Type=STRING");

    Console.WriteLine(" Variable get value. Start.");
    For (int i = 0; i < icnt; i++)
    {

        Object str= variable.Value;
        Console.WriteLine($"{i}:getValiable[tst_str] {str.ToString()}] OK");
        Thread.Sleep(100);
    }
    Console.WriteLine(" Variable get value. fin.");
    If (!controller.Variables.Remove("tst_str"))
    {
        Console.WriteLine(" Variable Remove error!");
        Return;
    }
    If (!engine.Workspaces[0].Controllers.Remove(idx))
    {
        Console.WriteLine(" Controller Remove error!");
        Run_fg = false;
    }
    Break;

Case 4:
    Controller = engine.Workspaces[0].AddController(
                                        CTL_NAME, CTL_PROV, null, CTL_OPTION);

    Console.WriteLine(" Program execute and abort. Start.");
    For (int i = 0; i < icnt; i++)
    {
        Console.WriteLine("Execute[orin]");
        Controller.Execute("Execute","orin");
        Thread.Sleep(1000);
        Console.WriteLine("Abort");
```

```
            Controller.Execute("Abort", "");
            Thread.Sleep(1000);
            Console.WriteLine($"{i}:OK");
        }
        Console.WriteLine(" Program execute and abort.. fin.");
        If (!engine.Workspaces[0].Controllers.Remove(CTL_NAME))
        {
            Console.WriteLine(" Controller Remove error!");
            Run_fg = false;
        }
        Break;

    Case 5:
        Controller = engine.Workspaces[0].AddController(
                                    CTL_NAME, CTL_PROV, null, CTL_OPTION);
        Object[] opt = {"orin", 5};
        Controller.Execute("Execute", opt);
        Console.WriteLine(" Program hold and continue. Start.");
        For (int i = 0; i < icnt; i++)
        {
            Console.WriteLine("Hold");
            Controller.Execute("Hold", "");
            Thread.Sleep(500);
            Console.WriteLine("Continue");
            Controller.Execute("Continue", "");
            Thread.Sleep(1000);
            Console.WriteLine($"{i}:OK");
        }
        Console.WriteLine(" Program hold and continue. fin.");
        If (!engine.Workspaces[0].Controllers.Remove(CTL_NAME))
        {
            Console.WriteLine(" Controller Remove error!");
            Run_fg = false;
        }
        Break;

    Case 6:
```

```
        Controller = engine.Workspaces[0].AddController(
                                        CTL_NAME, CTL_PROV, null, CTL_OPTION);
        Console.WriteLine(" Set speed. Start.");
        For (int i = 0; i < icnt; i++)
        {
            Controller.Execute("SetSpeed", 10);
            Thread.Sleep(300);
            Console.WriteLine($"{i}:SetSpeed OK");
        }
        Console.WriteLine(" Set speed. fin.");
        If (!engine.Workspaces[0].Controllers.Remove(CTL_NAME))
        {
            Console.WriteLine(" Controller Remove error!");
            Run_fg = false;
        }
        Break;

    Case 7:
        Controller = engine.Workspaces[0].AddController(
                                        CTL_NAME, CTL_PROV, null, CTL_OPTION);
        Console.WriteLine(" Save. Start.");
        For (int i = 0; i < icnt; i++)
        {
            Controller.Execute("Save", SV_PATH);
            Thread.Sleep(500);
            Console.WriteLine($"{i}:Save:OK");
        }
        Console.WriteLine(" Save. fin.");
        If (!engine.Workspaces[0].Controllers.Remove(CTL_NAME))
        {
            Console.WriteLine(" Controller Remove error!");
            Run_fg = false;
        }
        Break;

    Case 8:
        Controller = engine.Workspaces[0].AddController(
```

```
                                                    CTL_NAME, CTL_PROV, null, CTL_OPTION);
                    Console.WriteLine(" Load. Start.");
                    For (int i = 0; i < icnt; i++)
                    {
                        Controller.Execute("Save", LD_PATH);
                        Thread.Sleep(500); Console.WriteLine($"{i}:Load: OK");
                    }
                    Console.WriteLine(" Load. fin.");
                    If (!engine.Workspaces[0].Controllers.Remove(CTL_NAME))
                    {
                        Console.WriteLine(" Controller Remove error!");
                        Run_fg = false;
                    }
                    Break;

                Default:
                    Run_fg = false;
                    Break;
            }
            Proc.Refresh();

Console.WriteLine($"Memory(.WorkingSet:{proc.WorkingSet64:N0}  .VirtualMemory:{proc.VirtualMemoryS
ize64:N0})");
                }
            }
        }
    }
```

4.1.1.1. Connection

To connect to the robot controller, proceed as follows:

(1) Generate CCaoEngine required to use ORiN.

CaoEngine is generated using the new keyword.

You can use using clause during generation to avoid forgetting to release.

Sample usage (C#)

```
Using (CCaoEngine engine = new CCaoEngine())
```

(2) Creates a CaoController for connecting to the robot controller.

To generate a CCaoController object, specify various parameters for AddController method of CCaoWorkspace: controller name, provider name, machine name, and optional string.

For KRCC providers, specify the address to connect to, the port, the timeout, and so on, in an optional string.

The following is a code example:

Sample usage (C#)

```
Static string CTL_NAME = "SampleController";
Static string CTL_PROV = "CaoProv.Kawasaki.KRCC";
Static string CTL_OPTION = "CONN=TCP:127.0.0.1";
    :
    :
CCaoController controller =
    Engine.Workspaces[0].AddController(CTL_NAME, CTL_PROV, null, CTL_OPTION)
```

4.1.1.2. Retrieving Variable Values

Creates a CCaoVariable for retrieving.

To create a CCaoVariable object, specify a variable name and an optional string in AddVariable method of CCaoController. (The variable used this time does not specify an option string.)

For system variables that start with @, the option string can be "".

In the option string for other user variables

Name = {Variable name on robot controller}, Type = {Value type (see Table 3.2)}.

In the case of a character string type, the variable name must be prefixed with '$', '@' for integers, and '#' for positional information. (Decimal point type, not required for (REAL) or axis-value)

To get the value of a variable, use Value property of CCaoVariable object.

When actually using a value in the program, process the value obtained from Value property of CCaoVariable object programmatically.

You can also change the value of a variable by setting the value of Value property. (with the exception of some variables)

The code is shown below.

Sample usage (C#)

```
// System Valiable Set & Read
CCaoVariable val_sName = controller.AddVariable("@MAKER_NAME", "");
String s_val = val_sName.Value.ToString();

CCaoVariable val_sVer = controller.AddVariable("@VERSION", "");
s_val = val_sVer.Value.ToString();

CCaoVariable val_sPosit = controller.AddVariable("@CURRENT_POSITION", "");
Double[] p_val = val_sPosit.Value as double[];
s_val = "";
For (int i = 0; i < p_val.Length; i++) s_val += $" {p_val[i]},";

CCaoVariable val_sAngle = controller.AddVariable("@CURRENT_ANGLE", "");
Double[] a_val = val_sAngle.Value as double[];
s_val = "";
For (int i = 0; i < a_val.Length; i++) s_val += $" {a_val[i]},";

// User Valiable
// Below is an example of defining user-specified variables and setting Get/Put.
```

```
// Please refer if necessary

//<Int type variable>
CCaoVariable val_uInt=controller.AddVariable(
                                                "U_IntData","Name=@IntData, Type = INT");
Val_uInt.Value = 123;
s_val = val_uInt.Value.ToString();

//<REAL type>
 CCaoVariable val_uInt=controller.AddVariable(
                                              "U_RealData", "Name=RealData,Type=REAL");
Val_uReal.Value = 123.4567;
s_val = val_uReal.Value.ToString();

//<STRING type>
CCaoVariable val_uStr = controller.AddVariable(
                                                "U_StrData", "Name=$StrData,Type=STRING");
Val_uStr.Value = "TEST_Data";
 s_val = val_uStr.Value.ToString();


//<POSE type>
CCaoVariable val_uPose = controller.AddVariable(
                                                "U_PosData", "Name=PosData,Type=POSE");
List<double> inp_val= new List<double>() {1.1, 2.2, 3.3, 4.4, 5.5, 6.6};
Val_uPose.Value = (object)inp_val;
Double[] put_val = val_uPose.Value as double[];
s_val = "";
For (int i = 0; i < put_val.Length; i++) s_val += $" {put_val[i]},";
```

4.1.1.3. Program Execution

CCaoController::Execute method is used for control such as executing or stopping the programming of the robot.

Set CCaoController::Execute method to a command for controlling and execute it.

The main commands to be executed are as follows.

Process	First argument (command)	Second argument (command)
Execution	Execute	Program name
Pause	Hold	""
Resume	Continue	""
Stop	Abort	""

Sample usage (C#)

```
Controller.Execute("Execute","orin");
Controller.Execute("Hold", "");
Controller.Execute("Continue", "");
Controller.Execute("Abort", "");
```

4.1.1.4. Disconnect

When disconnecting from a controller, you release the generated objects and delete the objects that you create from the collection class that manages the objects. However, if you use ORiN.ManagedCAO, you do not need to explicitly release objects other than CCaoEngine because calling Dispose method of CCaoEngine automatically releases all objects belonging to CCaoEngine and removes them from the collection class that manages the object.

You can also specify scopes in using clause when generating CCaoEngine to avoid forgetting to release because Dispose method is automatically called when you leave scope.

Remember to call Dispose method prior to exiting the application if you need to generate and release CCaoEngine in another method, for example, in a form application.

5. KRCC Provider Error Codes

This provider has its own error code represented by 0x80100001.

If an error with error code 0x80100001 occurs, the error message notified from the robot controller is set in the error message. For the details of the error, refer to the error message.