CAO provider development guide

Version 1.1.2

November 5, 2019

[Remarks]

[Revision history]

Date	Number of versions	Content
2006-02-23	1.0.0.0	First edition
2006-06-05	1.0.2.0	Added Visual Studio2005-compatible information
2006-10-18	1.0.3.0	Corrected communication class usage and the TCmini provider development procedure.
2006-12-12	1.0.4.0	Added SetTimerInterval to CaoProvControllerImpl.
2007-07-03	1.0.5.0	Errors correction
2007-11-23	1.0.6.0	Corrected communication class usage and the TCmini provider development procedure.
2008-01-30	1.0.7.0	Corrected errors in the error code allocation table
2008-12-05	1.0.8.0	Added in-process message transfer function
2009-08-24	1.0.9.0	Added registry information setting and the CAO installation status confirmation.
2010-06-08	1.0.10.0	Corrected the installation status confirmation
2011-12-22	1.0.11.0	Added error codes of device class
2013-09-02	1.0.12	Added “RunAsLocal” option into the registry database
2014-04-22	1.1.0	Added Provider cancellation/clear
2014-09-09	1.1.1	Corrected errors in the reference folder
2019-11-05	1.1.2	Added how to register provider and unregister provider.

Contents

1. Introduction

This manual explains the creation procedure of CAO providers with actual examples.

Chapter 2, the top of the instruction, describes the basic knowledge for CAO provider development, such as various methods that need to be installed in CAO providers. This chapter also describes the CAO provider development procedure with Microsoft Visual C++ 6.0 (hereafter, VC++).

Chapter 3 describes the methods of the optional character string analysis and the message event issue, which are both CAO provider functions. The communication classes for the socket-communication or the serial-communication will be described in this chapter.

Chapter 5 describes the CAO provider implementation procedure with actual examples of CAO provider for TCmini α TC3-02, which is a small type programmable controller made by Toshiba Machine Co., Ltd.

Chapter 6 , the end of the instruction, describes the CaoProvExec tool that comes with ORiN2 SDK.

2. Implementation procedure of CAO provider

2.1. Overview

CAO (Controller Access Object) is API (Application Program Interface) that provides a common access method for various industrial robots made by different manufacturers. CAO was developed based on the distributed object technology and its original target was industrial robots; however it has been widely expanded its application not only to the industrial robots but also to PLCs (Programmable Logic Controller) and NC machine tools.

CAO is composed with an engine-part that provides a common function for providers and some provider-parts that absorb specification differences among manufacturers. CAO engine gives a single interface to client applications, and a CAO provider gives a single interface to CAO engine. This two-layer structure allows provider developers to save the trouble of implementation of common functions where the engine-part offers, and the developers only have to implement the substantial part to access the internal information of the controller. This facilitates the implementation of CAO provider.

In addition, CAO provider is divided into C++ template (CAO Provider Template) and the manufacturer implementation-part as shown in "Appendix A.2CAO provider template function list". This list includes templates for common interface implementation and for the default implementation to the CAO engine. In a word, any user devices where CAO provider is installed enable to implement CAO provider by simply overriding function corresponding to the function that you intend to offer from this template.

This chapter explains the implementation procedure of CAO provider.

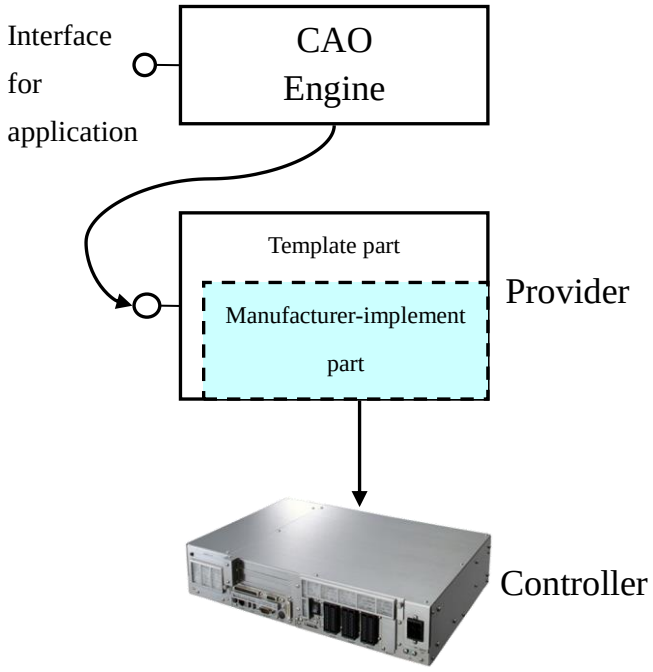


Figure2-1 Composition of CAO

2.2. Procedure of mounting

The figure below shows the flow of CAO provider development.

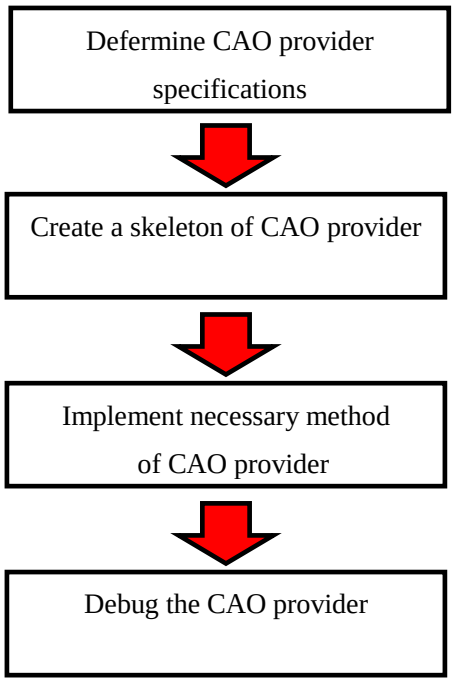


Figure2-2 CAO provider implementation procedure

(1) Determine CAO provider specification

To implement a CAO provider, you need to determine the provider specification. Set the following items for target to be accessed.

1. Name of CAO provider
2. Specification of connection parameter of AddController()
3. Class and method specifications to be implemented

(2) Create CAO provider template

When CAO provider specification has been decided, then create a template of CAO provider by using CaoProvWiz tool.

(3) Implement necessary methods of CAO provider

With a template of CAO provider created by CaoProvWiz, implement necessary methods. Add any classes for device communications if necessary.

(4) Debug CAO provider

Check if the created CAO provider operates properly as you intended. If it does not operate properly, debug the program. CAO provider will be released once all debugs are successfully completed.

The following section details about the implementation procedure.

2.3. Creating a provider project

With CAO Provider wizard, you can easily create a project creating a new CAO provider with Visual C++. The following shows how to create a project.

(1) From the Start menu, point to [ORiN2]-[CAO] -[Provider]-[Bin], and then execute [CaoProvWiz]

(2) Select a folder where you want to create a new CAO provider project.

Example) When you want to create a project in "C:¥ORiN2¥CAO¥ProviderLib¥Sample¥src"

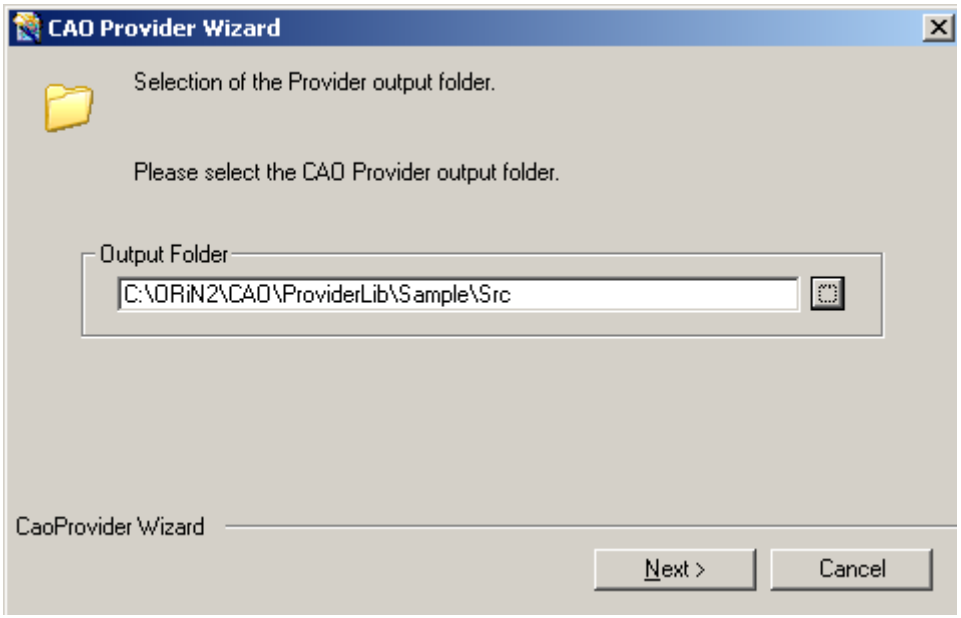


Figure2-3 Folder selection for CAO Provider output

(3) Enter provider information to be created

- DLL Name : DLL name (required)
- Vender Name : Vender name (required)
- Module Name : Module name
- Project Type : Version of VC project to be created
- Use MFC : Whether MFC is used

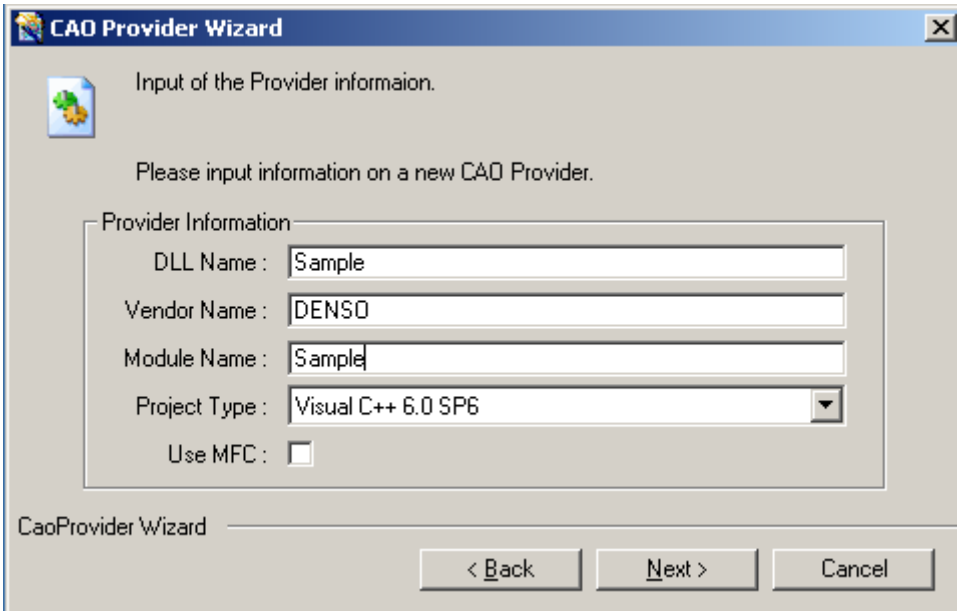


Figure2-4 Provider information input dialog

Vender Name and Module Name specified here will be used to determine a ProgID name of COM. ProgID of a CAO provider will be “CaoProv.<Vender Name>.<Module Name>".

(4) A confirmation appears. When "Yes" is clicked, a provider's skeleton will be created.

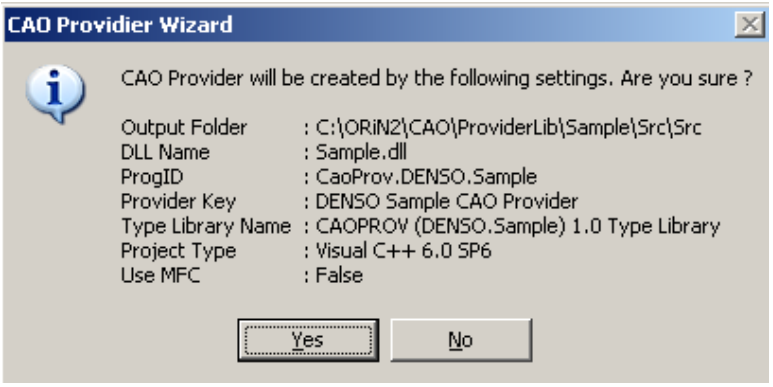


Figure2-5 Provider information setting confirmation dialog

(5) Once the following window appears, the provider skeleton has successfully completed.

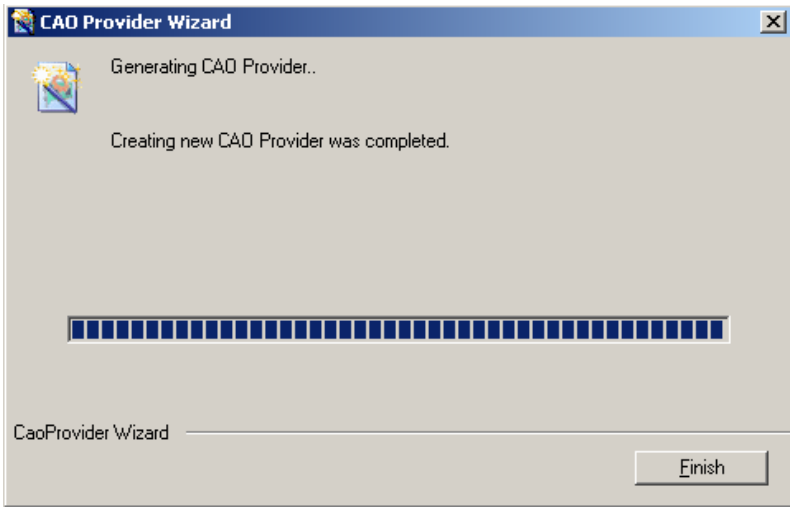


Figure2-6 Completion of creating project of provider dialog

Under the specified directory, a VC++ project for CAO provider is created as CAOPROV.dsw. Specify this VC++ project to start VC++.

The created project folder includes several files other than CaoProv.dsw. The table below lists CAO provider-related files. Other files not described in this table are managed by VC++, so you do not need to take them into consideration.

Table2-1 Main file list of project

No.	Name	Kind
1	CaoProv.dsw	Project workspace (Only Visual Studio 6.0 :).
2	CaoProv.dsp	Project file (Only Visual Studio 6.0 :).
3	CaoProv.vcproj	Project file (Only Visual Studio 2005 :).
4	CaoProv.IDL	IDL file
5	Resource.h	Resource header file
6	CaoProv.rc	Resource file
7	StdAfx.h	C header file
8	StdAfx.cpp	C++ source file
9	CaoProv.h	C header file
10	CaoProv.cpp	C++ source file
11	CaoProvController.h	Definition of C header file CCaoProvController class
12	CaoProvController.cpp	Implementation of C++ source file CCaoProvController class
13	CaoProvVariable.h	Definition of C header file CCaoProvVariable class
14	CaoProvVariable.cpp	Implementation of C++ source file CCaoProvVariable class
15	CaoProvTask.h	Definition of C header file CCaoProvTask class
16	CaoProvTask.cpp	Implementation of C++ source file CCaoProvTask class
17	CaoProvRobot.h	Definition of C header file CCaoProvRobot class
18	CaoProvRobot.cpp	Implementation of C++ source file CCaoProvRobot class
19	CaoProvFile.h	Definition of C header file CCaoProvFile class
20	CaoProvFile.cpp	Implementation of C++ source file CCaoProvFile class
21	CaoProvCommand.h	Definition of C header file CCaoProvCommand class
22	CaoProvCommand.cpp	Implementation of C++ source file CCaoProvCommand class
23	CaoProvExtension.h	Definition of C header file CCaoProvExtension class
24	CaoProvExtension.cpp	Implementation of C++ source file CCaoProvExtension class
25	CaoProvMessage.h	Definition of C header file CCaoProvMessage class
26	CaoProvMessage.cpp	Implementation of C++ source file CCaoProvMessage class

(6) Since the skeleton project of the provider has been completed by the above procedures, you can build the project as it is. (Note that it does not have any function). To do so, from the Build menu of Visual C++, select [Active Configuration...(C)], and then select “Win32 Debug" (debug version) or “Win32 Release MinDependency" (release version) to execute [Re-build].

2.4. Implementation of each class at CAO provider

2.4.1. Implementation preparation

2.4.1.1. Implementation preparation of class

Definitions and implementations of each class are stored in the following files.

Table 2-2 Files storing class definition and implementation

Class name	Definition	Implementation
CCaoProvController	CaoProvController.h	CaoProvController.cpp
CCaoProvCommand	CaoProvCommand.h	CaoProvCommand.cpp
CCaoProvExtension	CaoProvExtension.h	CaoProvExtension.cpp
CCaoProvFile	CaoProvFile.h	CaoProvFile.cpp
CCaoProvMessage	CaoProvMessage.h	CaoProvMessage.cpp
CCaoProvRobot	CaoProvRobot.h	CaoProvRobot.cpp
CCaoProvTask	CaoProvTask.h	CaoProvTask.cpp
CCaoProvVariable	CaoProvVariable.h	CaoProvVariable.cpp

To implement a class in the CAO provider, override the methods in the class. Prepare for the method override by following procedures.

(1) Among methods written in the header file of each class, remove comments from the methods that you want to use. The following example shows when “FinalInitialize” method in CCaoProvController class is used. (CaoProvController.h)

```
//      HRESULT FinalInitialize();  // Before removing the comment
  ↓
        HRESULT FinalInitialize();  // After removing the comment
```

(2) Remove comments where any methods are implemented. The following example shows when “FinalInitialize” method in CCaoProvController class is implemented. (CaoProvController.cpp)

```
// Before removing the comment
/*   Delete this comment-start sign (a combination of a backslash and an asterisk)
HRESULT CCaoProvVariable::FinalInitialize()
{
        : (omit coding example)

        //This function must be implemented at any time.
        //Execute initialization and other processing which are common to CaoProvController object
        return E_NOTIMPL; //Once implementation has completed, set the return value to “S_OK”.

}
*/   Delete this comment-end sign (a combination of an asterisk and a backslash)
↓
//After removing the comment
HRESULT CCaoProvVariable::FinalPutValue(VARIANT newVal)
{
```

```
        : (omit coding example)

        //This function must be implemented at any time.
        // Execute initialization and other processing which are common to CaoProvController
        //objects
        return E_NOTIMPL; //Once implementation has completed, set the return value to "S_OK".
}
```

(3) Set the return value of the method in (2) to "S_OK".

return E_NOTIMPL; → return S_OK;

This completes the preparation for the method override.

Once the preparation has completed, implement manufacturer-specific processing for each method. You can add variables and methods to classes, if necessary. Definitions for additional variables and methods must be added at the end of the newly added class definitions. Additional methods must be implemented at the end of a cpp file where class methods to be added are implemented.

Actual coding examples of basic methods will be described later. For about the list of member variables and overridable functions which are opened to the public by the provider template, please refer to "Appendix A.2CAO provider template function list".

2.4.1.2. Role of each class

Each class has been designated to offer different roles. The table below explains the function of each class.

Table2-3 Description of CAO provider class function

Class name	Explanation
CaoProvController	Controller class. This offers controller's resource-related total functions.
CaoProvVariable	Variable class. This offers variable resource-related functions.
CaoProvRobot	Robot class. This offers robot resource-related functions.
CaoProvFile	File class. This offers file- and folder resource-related functions.
CaoProvTask	Task class. This offers task resource-related functions.
CaoProvCommand	Command class. This offers command resource-related functions.
CaoProvExtension	Expansion class. This offers expansion board resource-related functions.
CaoProvMessage	Message class. This offers message resource-related functions.

Table2-4 shows the list of functions. Some functions mentioned here must be overridden when the said classes are implemented.

Table2-4 Indispensable method in each class implementation

Class name	Mandatory method	Explanation
CaoProvController	FinalInitialize()	Initialization
	FinalConnect()	Provider connection processing
	FinalDisconnect()	Provider cutting processing
CaoProvVariable	FinalInitialize()	Initialization
CaoProvRobot	FinalInitialize()	Initialization
CaoProvFile	FinalInitialize()	Initialization
CaoTask	FinalInitialize()	Initialization
CaoCommand	FinalInitialize()	Initialization
CaoExtension	FinalInitialize()	Initialization
CaoMessage	FinalInitialize()	Initialization

The most important class is CaoProvController. This must be implemented at any time. For the remaining classes, all classes have no need of implementation at the CAO provider implementation, and it depends on each vendor's discretion. In a word, vendors only have to implement desired classes corresponding to the function where vendors want to offer to users.

The following section describes CCaoProvController::FinalInitialize(), CCaoProvController::FinalConnect(), CCaoProvController::FinalDisconnect(), CCaoProvVariable::FinalInitialize, CCaoProvVariable::FinalPutValue(), and CCaoProvVariable::FinalGetValue() from CaoController class and CaoVariable class.

2.4.2. CaoProvController class

HRESULT CCaoProvController::FinalInitialize()

This method is called every time the CaoController object is created for object initialization. Please initialize variables used in CaoProvController class.

HRESULT CCaoProvController::FinalConnect()

This method is called at the time when CaoWorkspace::AddController processing is called. This method implements the connection processing with the robot controller where the newly created provider will accept. The connection method depends on each robot controller. For example, if any DLL accessing the robot controller has been prepared, the processing to load DLL is expected. When RS232-C communication or TCP/IP communication is performed, the connection processing will be implemented in this method.

HRESULT CCaoProvController::FinalDisconnect()

This performs the disconnection processing.

This is called at the time when the CaoControllers::Remove processing is called. This disconnects the provider from the robot controller connected by FinalConnect(). The way of disconnection depends on each robot controller.

2.4.3. CaoProvVariable class

HRESULT CCaoProvVariable::FinalInitialize()

This method is called at the time of CaoVariable object creation, and initializes the object.

pObj that is the argument of this function is a pointer for the parent object.

Because CaoVariable is composed of several objects (such as CaoController and CaoRobot), you can individually initialize them according to the parent object by using FinalIntialize(), if necessary. The parent object can be identified by member variable m_ulParentType. The table below shows m_ulParentType.

Table2-5 Parent object of CaoProvVariable

m_ulParentType	Parent object
SYS_CLS_CONTROLLER	CaoProvController
SYS_CLS_ROBOT	CaoProvRobot
SYS_CLS_FILE	CaoProvFile
SYS_CLS_TASK	CaoTask
SYS_CLS_EXTENSION	CaoExtension

HRESULT CCaoProvVariable::FinalGetValue(VARIANT *pVal)

The value of the variable is acquired.

The argument is pointer pVal of VARIANT type. The value stored in this pVal will be sent to the client. The client can get the variable from the robot controller if necessary. If there is no way to access the robot controller, the value can be hold as a member variable of Class. For the substitution method to VARIANT, refer to "ORiN2 programming guide".

The implementation example of @VERSION that is one of system variable in Controller class is as follows.

(1) Set the necessary macro definitions required for StdAfx.h.

```
// ==== Manufacturer-specific additional items should be written below. ==========
#define CS_VERSION      0x0002              <-  Assign a unique number
#define CS_VERSION$     L"@VERSION"         <-  Assign a unique name
```

(2) Add a macro for initialization of Name Map

```
HRESULT    CCaoProvVariable::InitMapTable()
{
        :
        // Initialize Variable Map
        const var_map_entry var_cs_map[] = {

                // :
                MAP_ENTRY( CS_VERSION ),   <-  Add a macro
        };
        :
        return S_OK;
}
```

(3) Implement the obtainment of the value of System variables in the Controller class.

```
HRESULT CCaoProvVariable::FinalGetCtrlSysValue(VARIANT *pVal)
{
        switch (m_lUSysId) {
                :
        case CS_VERSION:                        <-  The following program was added.
                pVal->vt = VT_BSTR;
                pVal->bstrVal = SysAllocString(L"0.0.0");
                break;
                :
```

```
        }

        return hr;
}
```

In this example, the client obtains VT_BSTR type's system variable by accessing the controller, and then substitutes the value for pVal. In the actual implementation, follow the actual access method to the controller.

HRESULT CCaoProvVariable::FinalPutValue(VARIANT newVal)

Set a value of a user variable.

Argument "newVal" contains values from the client. Because newVal can contain various data types, the type of argument must be judged before setting the value, and then extract values from VARIANT.

In case that the type of VARIANT is VT_BSTR, VT_ARRAY, or VT_BYREF, which do not have actual value, the value will be deleted once FinalPutValue method ends. If any values of these types need to be used after FinalPutValue method ends, save the values in different memory.

The following shows an example to set user variables to the controller with this method.

List 2-1 CCaoProvVariable.cpp – FinalPutValue()

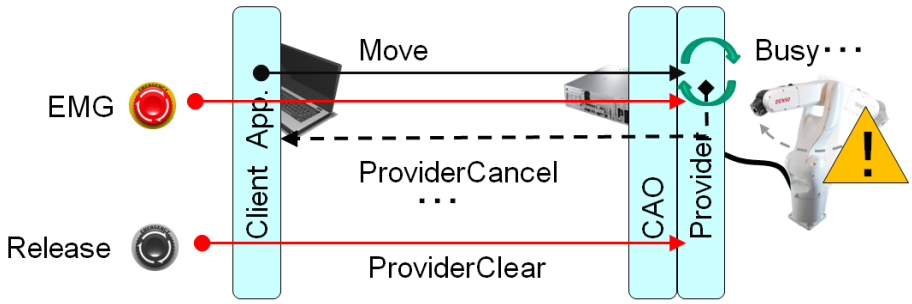
```
HRESULT CCaoProvVariable::FinalPutValue(VARIANT newVal)
{
        HRESULT hr = E_ACCESSDENIED;

        if (m_bSystem) {
                switch (m_ulParentType) {
                case SYS_CLS_CONTROLLER:
                        hr = FinalPutCtrlSysValue(newVal);
                        break;
                }
        }
        else {
                switch (m_ulParentType) {
                case SYS_CLS_CONTROLLER:
                        hr = FinalPutCtrlUserValue(newVal);
                        break;
                }
        }
        return hr;
}

HRESULT CCaoProvVariable:: FinalPutCtrlUserValue (VARIANT newVal)
{
    HRESULT  hr = S_OK;
    switch( newVal.vt ){
    case VT_I2:
        // m_pRobotCtrl is a pointer for the robot controller objects
        // For PutRCiVal(), set the value of VT_I2 type's user variable.
        hr = m_pRobotCtrl->PutRCiVal( newVal.iVal );
        break;
    case VT_I4;
        :
        :
    default:
```

```
        hr = E_INVALIDARG;
    }
    return hr;
}
```

Neither m_pRobotCtrl nor PutRCiVal() actually exist. In this example, values are set to the controller along with the data type of newVal. When in actual operation, implement an operation that sets values along with the access method of the controller.

2.4.4. Cancellation of running process

During the operation in which an emergency stop can be issued, the provider must be arranged in order that the running operation can stop immediately and the process returns to the upper client, CAO application. Doing so will allow upper applications to execute necessary operations for stop. To handle this request, the provider needs to respond to the two request commands that are specified by ORiN2, and to cancel the running processing.



These request commands will be specified as a system reserve command with a following method.

HRESULT CCaoProvController::ExecProviderCancel(VARIANT vntParam, VARIANT *pVal);
HRESULT CCaoProvController::ExecProviderClear(VARIANT vntParam, VARIANT *pVal);

Once ProverCancel command is issued, ExecProviderCancel() method and ExecProviderClear() method are asynchronously called by CAO.exe. Note that ExecProviderClear() is called against ProviderClear.

A client application will asynchronously request ProviderCancel commands to CAO.exe if provider needs to cancel running processing immediately. To resume a normal command after completing cancel processing, issue a ProviderClear command to clear the cancel request.

2.4.4.1. Implementation of ProviderCancel command

In the skeleton created by ProviderWizard, a method CCaoProvController::ExecProviderCancel() corresponding to ProviderCancel command is implemented as follows.

List 2-2 CCaoProvController.cpp – ExecProviderCancel()

```
/** Provider cancel
 *
 *  Immediately cancel the running processing
 *
 *     @param   vntPara   :        [in] Parameter
 *                                      Unused
 *     @param   pVal      :        [out] Execution result
 *                                      Unused
 *     @retval  HRESULT
 *
 */
HRESULT CCaoProvController::ExecProviderCancel(VARIANT vntParam, VARIANT *pVal)
{
        ATLASSERT(m_hProviderCancelEvent != NULL);

        // Set a provider cancel event
        ::SetEvent(m_hProviderCancelEvent);

        // TODO: Implement cancel processing for running processing
        // :

        return S_OK;
}
```

Here, you can see that m_hProviderCancelEvent is a member variable of CCaoProvController class, is defined as HANDLE type, and is implemented so that SetEvent turns it to the signal-state.

The cancel processing is achieved by implementing the operation that monitors this HANDLE during asynchronously executing loop process, and exits the loop under the signal-state, as shown below.

If any processing that should be cleared in this timing, add a necessary implementation.

```
for(int i=0; i<1000; i++)  {
        // Arbitrary processing...
        //  :
        if(::WaitForSingleObject(m_hProviderCancelEvent, 0) == WAIT_OBJECT_0) {
                //  Cancel processing
                break;
        }
}
```

2.4.4.2. Implementation of ProviderClear command

In the skeleton created by ProviderWizard, a method CCaoProvController::ExecProviderClear() corresponding to ProviderClear command is implemented as follows.

List 2-3 CCaoProvController.cpp – ExecProviderClear()

```
/** Provider clear
 *
```

```
 *  Clear the cancel processing
 *
 *     @param    vntPara   :         [in] Parameter
 *                                       Unused
 *     @param    pVal      :         [out] Execution result
 *                                       Unused
 *     @retval   HRESULT
 *
 */
HRESULT CCaoProvController::ExecProviderClear(VARIANT vntParam, VARIANT *pVal)
{
        ATLASSERT(m_hProviderCancelEvent != NULL);

        // Reset the provider cancel event
        ::ResetEvent(m_hProviderCancelEvent);

        // TODO: Must implement the processing that assures the proper operation.
        // :

        return S_OK;
}
```

To prevent the recurrence of cancel processing, reset "m_hProviderCancelEvent" (which remains as a signal-state) to the normal state by using ResetEvent. If any processing that should be cleared in this timing, add a necessary implementation.

2.4.5. Setting of registry information

An initial value of information registered in the registry is set with the CaoProvController.rgs file.

List 2-4 CCaoProvController.rgs

```
HKCR
{
    CaoProv.CaoProvController.1 = s 'CaoProvController Class'
    {
        CLSID = s '{1de03cfd-535f-42db-88cf-3a72bee12813}'
    }
    CaoProv.CaoProvController = s 'CaoProvController Class'
    {
        CLSID = s '{1de03cfd-535f-42db-88cf-3a72bee12813}'
        CurVer = s 'CaoProv.CaoProvController.1'
    }
    NoRemove CLSID
    {
        ForceRemove {1de03cfd-535f-42db-88cf-3a72bee12813} = s 'CaoProvController Class'
        {
            ProgID = s 'CaoProv.CaoProvController.1'
            VersionIndependentProgID = s 'CaoProv.CaoProvController'
            InprocServer32 = s '%MODULE%'
            {
                val ThreadingModel = s 'Free'
            }
            'TypeLib' = s '{4489ef8d-cf16-414e-9d93-cb9af157cff2}'
            'CAO Provider' = s 'Skeleton CAO Provider'
            {
                val Enabled = d '4294967295'
                val RunAsLocal = d '0'
```

```
                val Writable = d '4294967295'
                 val LocaleID = d '1024'
                 val License = s ''
                 val Parameter = s ''
                 val CRDFile = s ''
                 val BindCmds = d '4294967295'
                 val BindExec = d '0'
                 val GroupID = d '1'
            }
            val AppID = s '{1de03cfd-535f-42db-88cf-3a72bee12813}'
        }
    }
    NoRemove AppID
    {
        ForceRemove {1de03cfd-535f-42db-88cf-3a72bee12813} = s 'CaoProvController Class'
        {
            val DllSurrogate = s ''
        }
    }
}
```

Edit only the part shown in the gray in the file above. When other parts are edited, the provider might not be able to register correctly.

Name	Value
Enabled	Availability setting 0 : Not available. 4294967295(0xffffffff) : Available.
RunAsLocal	Set the default startup method of CAO provider when the startup machine name is omitted at CaoWorkspace::AddController execution. 0 : In-process startup 4294967295(0xffffffff) : Out-process startup
Writable	Writing flag 0 : Read only 4294967295(0xffffffff) : Read and write
LocaleID	Locale
License	License setting
ORiNlm	Reservation (always null character string)
Parameter	Parameter setting (Set a parameter that can be fixed to some extent)
CRDFile	CRD file setting. When "E_CRDIMPL" is returned in the property of each class, the content of the CRD file specified here is returned to the client.
BindCmds	Dynamic binding (Command class) 0 : Dynamic binding is impossible.

	4294967295(0xffffffff)	:	Dynamic binding is possible.
BindExec	Dynamic binding (Execute method)		
	0	:	Dynamic binding is impossible.
	4294967295(0xffffffff)	:	Dynamic binding is possible.
GroupID	Reservation (1 always)		

The settings mentioned above can be changed by using CaoConfig.

2.5. Debug and release of CAO provider

2.5.1. Debug of provider

In CAO, DLL module of CAO provider is called from CAO.exe that is CAO engine if necessary. Therefore, when CAO provider DLL module is debugged, it is necessary to specify CAO.exe for an executable file of the debugging section of VC++. [1]

To use them as a client application, use ORiN2¥CAO¥Tools¥CaoTester¥Bin¥CaoTester.exe that is CAO test tool of ORiN depending on your needs.

One of the debug processing examples is as follows.

(1) Set CAOPROV-Win32 Debug as a build target.

From the [Build(B)] menu, select [Active Configuration...(C)]. From the [Project Configuration(P)] , select CAOPROV-Win32 Debug.

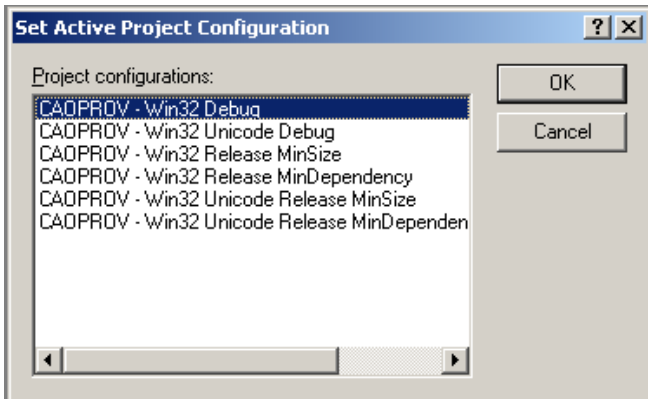


Figure2-7 Build target specification screen

(2) Set CAO.exe in the [Executable For Debug Session].

From the [Project (P)] menu, select [Settings(S) Alt+F7], and then select [Win32 Debug] from the list. Enter full path of CAO.exe in the [Executable For Debug Session (E)]. CAO.exe is usually in the

[1] To start provider DLL in the out process, dllhost.exe is specified for "Executable file at the debug session". In this case, it is necessary to specify "Machine name" for the second argument of the AddController method of CAO.

ORiN2¥CAO¥Engine¥Bin folder.

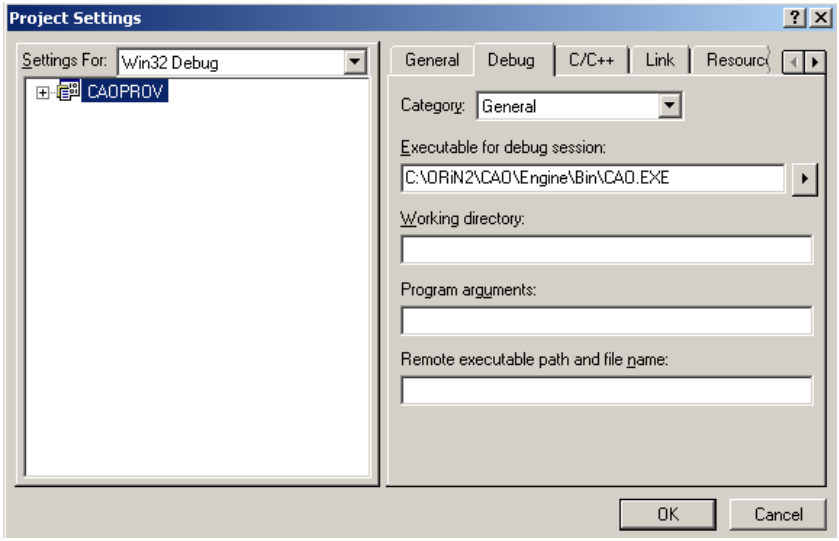


Figure2-8 Specified screen of CAO.exe

(3) Set breakpoints at necessary positions.

(4) Execute [Start Debug].

From the [Build (B)] menu, point to [Start Debug(D)], and then click [Go(G)].

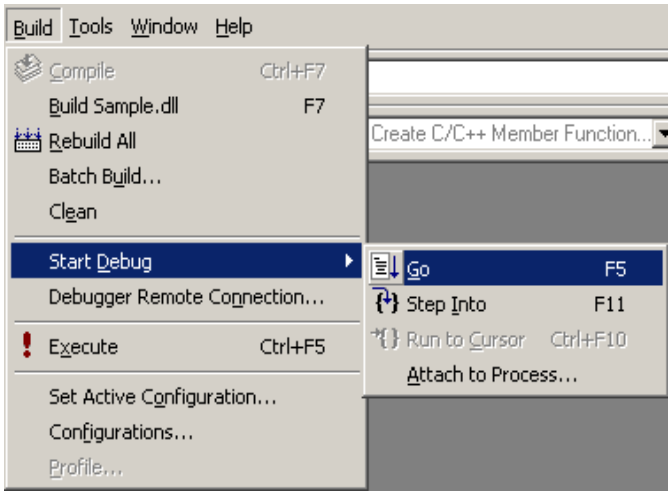


Figure2-9 Beginning screen of debugging

(5) Start CaoTester.exe, and then specify the provider to be debugged.

(6) Press the [Add] button to connect to the controller.

(7) Execute CAO service with CaoTester to call the provider.

For example, to debug a CaoVariable object, click the Variable tab and then enter desired name in Name in AddVariable, and then press [Add] button. Execute Put or Get of Value.

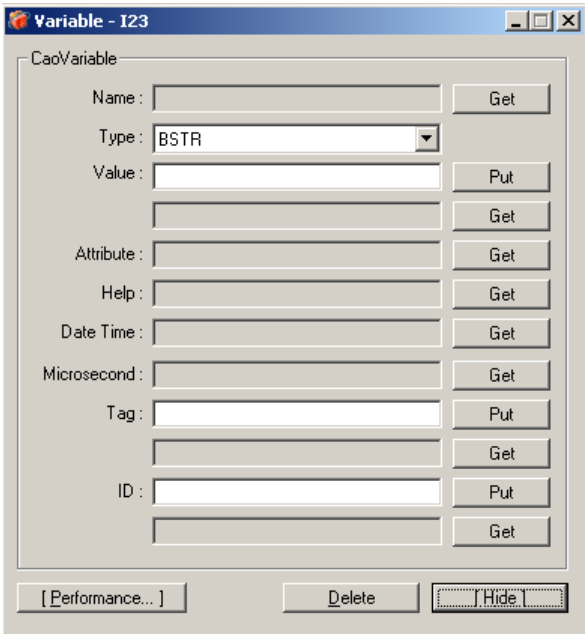


Figure2-10 Specified screen of system variable

(8) Return to VC++. If it stops at the position of break pointer, debug it with VC++.

2.5.2. Release of the provider

2.5.2.1. Creating documents

Users cannot use the provider easily unless the provider's specification is open to the public. Therefore, it is necessary to create specifications of the provider and to release it.

At least, the following contents must be described as the specifications for CAO provider.

(1) Specification of connection parameters of AddController()
(2) Specification of user variables
(3) List of system variables and its meaning
(4) Other important information (Provider specific function's information, notes, etc.)

Because there are no regulations about the writing of specifications (format), you can create it as you want. We recommend you to refer to other document, such as the "ToshibaMachine¥Doc¥CaoProvTCmini specification doc." that is stored in "ORiN2¥CAO¥ProviderLib¥" coming with ORiN SDK as a sample program.

2.6. Distribution of provider

2.6.1. Confirmation of dependence information

CAO.exe that is a main body of the CAO engine requires no specific dependent module (DLL etc.). This is because CAO engine is designed to provide stable operations in different environments. We recommend that the CAO provider is also created so as not to depend on other modules as much as possible. To do so, some countermeasures (such as not to use special libraries or to statically link with libraries) are required.

If the created CAO provider depends on other modules, confirm that which module is required for operation. To check the dependency information, use Dependency Walker for VC++, free tool Process Explorer, or others.

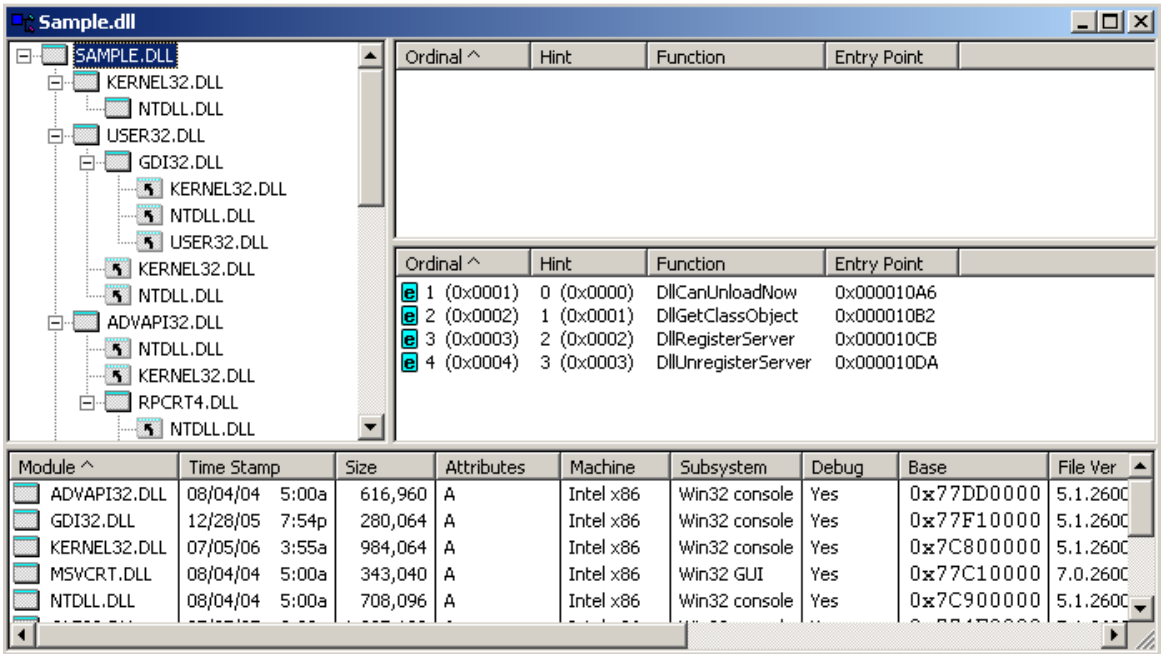


Figure2-11 Dependency Walker screen of TCmini.DLL

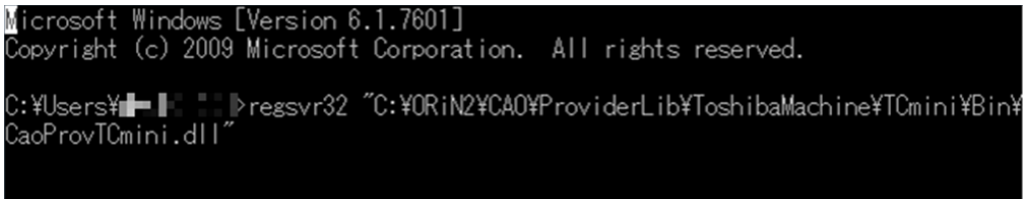
2.6.2. Installation situation of ORiN2 SDK

For the confirmation method of the installation status of ORiN2 SDK, refer to "ORiN2 SDK user's guide 3.7. the confirmation of the ORiN2 SDK installation situation".

2.6.3. How to register provider

When using the provider in the distribution destination, it is necessary to register the provider DLL in the registry. To register in the registry, use the regsvr32 command at the command prompt. You can register in the registry by entering the provider DLL name after the regvr32 command and executing it. When launching the command prompt, execute in administrator mode.

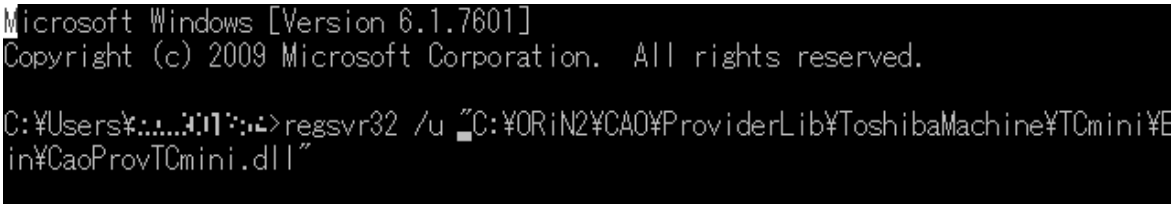
<Example of registration> CaoProvTCmini.dll



2.6.4. How to unregister provider

To unregister a provider DLL registered with the regsvr32 command, use the regsvr32 / u command. You can unregister from the registry by entering and executing the provider DLL name after the regsvr32 command.

<Example of unregistration > CaoProvTCmini.dll



2.7. Provider sample

In ORiN2SDK, the sample code for the provider is stored in the following folder.

< Install folder > ¥CAO¥ProviderLib

Many of these providers are created with Visual Studio 6.0 (henceforth VS6). To build them with Visual Studio 2005 (henceforth VS2005), some corrections are required. Open a VS6 project file (.dsw) with VS2005 first, and then update it to the project file (.vcproj) of VS2005.[2] In addition, the following problems might occur when the provider is build with VS2005. In that case, correct the source code with referring to the table below.

[2] http://msdn2.microsoft.com/en-us/7hfabkez(VS.80).aspx

Table2-6Problem and correspondence method with builds often

Problem	Correspondence method
“warning MIDL1015" is issued.	・ This warning is issued if the measure for the “warning MIDL2400 is issued” (written in the below cell) is performed. Disregard this warning.
“warning MIDL2400" is issued.	・ From the menu, select [Project]-[Property]. ・ From the Property page of CAOPROV, point to [Configurations] and then select [All Configurations]. ・ From the Property page of CAOPROV, point to [Configuration Properties] – [MIDL], and then set the [Warning level] to “0(/W0)”.
“warning C4996" is issued.	[Measure 1] ・ From the menu, select [Project]-[Property]. ・ From the Property page of CAOPROV, point to [Configurations] and then select [All Configurations] ・ From the Property page of CAOPROV, point to [Configuration Properties] - [C/C++]–[Preprocessor], and then add “_CRT_SECURE_NO_DEPRECATE[1]"” in [Preprocesssor Definitions].
	[Measure 2] ・ From the menu, select [Project]-[Property]. ・ From the Property page of CAOPROV, point to [[Configurations] and then select [All Configurations]. ・ From the Property page of CAOPROV, point to [Configuration Properties] - [C/C++]–[Preprocessor], and then add “_CRT_NON_CONFORMING_SWPRINTFS” in [Preprocesssor Definitions].
	[Measure 3] ・ Add an underbar “_” at the front of the function name that issues a warning. Example 1) wcsnicmp() →_wcsnicmp() Example 2) itoa() →_itoa()
“warning LNK4222" is issued.	・ Delete the ordinal of the export function in CaoProv.def. Example) DllCanUnloadNow @1 PRIVATE → DllCanUnloadNow PRIVATE

“error C2872" is issued in the XML-related class. [2]	[Measure 1] ・ Comment-out the place where the following two lines are described. #import "msxml4.dll" using namespace MSXML2;
	[Measure 2] ・ Set a name space for all classes where the error occurs. Example) When the Name-space name is “MSXML2" IXMLDOMNodePtr pIChildNode; → MSXML2::IXMLDOMNodePtr pIChildNode;

3. Useful functions that provider template library offers

3.1. Overview

In ORiN2 SDK, many useful functions are prepared for the CAO provider, such as the method of analyzing the option character string and the method of issuing the message event. In addition, some providers utilize the class for the device communication as well.

This chapter explains how to implement these functions.

3.2. Analysis of option character string

In CAO provider, an Option character string used as an argument of the method can be acquired by GetOptionValue method. The argument specification of GetOptionValue() is shown below.

For details about actual implementation, refer to “OptionValue.h” in “ORiN2/Cao/Include”.

```
GetOptionValue
(
"< option character string >"          // Option character string
"< character string to search >"       // Find-option name
"< request type >"                     // Data type of option value
"< result value >"                     // Option value
)
```

The option character string follows the format below.

<Option name 1>[=<Option value1>],<Option name 2>[=<Option value2>]・・・

An easy example is shown as follows. Here, the option character string is assumed to be “Opt1=test,Opt2=sample", and the search-option name is assumed to be “Opt1". As a result, vntOptVal obtains the value of “test".

The example below shows when the value of “Opt1" is acquired from the option character string.

```
HRESULT hr;
CComVariant vntOptVal;
hr = GetOptionValue(L"Opt1=test,Opt2=sample", L"Opt1", VT_BSTR, &vntOptVal);
```

Enclosure characters are available for the option character string. The following characters can be used as an enclosure character.

- Parentheses (“( )")
- Braces (“{ }")
- Brackets ("[ ]")
- Angle brackets (“<>")

Among these enclosure characters, the enclosure character that appears the first is treated as an enclosure character, and the remaining will be treated as normal signs.

The example below shows when there are two or more options.

Option character string: Test1=Sample1,Test2=Sample2

Table3-1 Result of option character string (example 1)

Option name	Option value
Test1	Sample1
Test2	Sample2

The example below shows when option values are placed between parentheses.

Option character string: Test1=(Sample1),Test2=(Sample2)

Table3-2 Result of option character string (example 2)

Option name	Option value
Test1	Sample1
Test2	Sample2

The example below shows when option values are placed in the different signs (parentheses and angle brackets).

Option character string: Test1=(Sample1),Test2=<Sample2>

Table3-3 Result of option character string (example 3)

Option name	Option value
Test1	Sample1
Test2	<Sample2>

The example below shows when one more signs (parentheses and angle brackets) are located in the parentheses of the option values.

Option character string: Test1=((Sample1)),Test2=(<Sample2>)

Table3-4 Result of option character string (example 4)

Option name	Option value
Test1	(Sample1)
Test2	<Sample2>

3.3. Analyzing connection parameters

CAO provider has CConnectOption class that analyzes the Option character string used as an argument of AddController method and obtains its value. By using this class, the connection parameters can be integrated into that of other providers. For actual implementation method, refer to ConnectOption.h in ORiN2/Cao/Include.

Connection parameters handled by CConnectionOption should follow one of the formats below.

Conn = ETH:<Connection destination IP>[:<Connection destination port>[:<Local IP>[:<Local port>]]]
= TCP:<Connection destination IP>[:<Connection destination port>[:<Local IP>[:<Local port>]]]
= UDP:< Connection destination IP>[:<Connection destination port>[:<Local IP>[:<Local port >]]]
= COM:<COM number>[:<Baud rate>[:<Parity>:<Data bit>:<Stop bit>:<Flow control>]]

The following shows simple examples.

Here, the connection parameter is assumed to be “Conn=eth:192.168.0.1:8080”. As a result of this sample, IP address and port number will be obtained.

List 3-1 CCaoProvController.cpp – FinalConnect ()

```
#include "ConnectOption.h"

HRESULT CCaoProvController:: FinalConnect ()
{
	HRESULT hr;

	// Analyzing connection option
	PARAM_CONN stRet;
	PARAM_CONN_ETH stEth = {INADDR_NONE, 0, INADDR_NONE, 9876};
	// Connection source not decided, connection source port not specified,
	// Connection destination not decided, connection destination port 9876

	CConnectOption cConnOpt(stEth);
	cConnOpt.SetTarget(TYPE_TCP | TYPE_ETH);
	hr = cConnOpt.GetConnectOption(m_bstrOption, &stRet);
	FAILED_RETURN(hr);


	// Specified connection parameters
	PVOID pConnArgs[4];
	pConnArgs[0] = &stRet.stEth.dwSrcIP;		// Local IP address
	pConnArgs[1] = &stRet.stEth.dwSrcPort;		// Local port number
	pConnArgs[2] = &stRet.stEth.dwDestIP;		// IP address
	pConnArgs[3] = &stRet.stEth.dwDestPort;		// Port number

	// Connection processing
	 :

	return hr;
}
```

3.4. Method of creating error

CAO provider can support provider's original errors and these errors can be sent to the CAO clients by adding original error codes. Sending original error information will improve debug efficiency and facilitate error processing at the CAO client implementation.

To create an error, take following procedures.

(1) Define an error code.

(2) Define an error message.

(3) Generate the error.

The details are explains as follows.

(1) The definition of the error code is defined in the header file collectively. (StdAfx.h is used here.) The implementation example of the definition (Error code E_SAMPLE is defined) is shown as follows.

```
// ERROR CODE
//
// MessageId: E_SAMPLE
//
// MessageText:
//
// This is a test.
//
#define E_SAMPLE _HRESULT_TYPEDEF_(0x80100001L)
```

For the error name, apply "E_" as a prefix. For the error code, specify numbers of 32 bits in the macro of _ HRESULT_TYPEDEF_(). Error codes prepared for the CAO provider are 0x80100000-0x8010FFFF(FACILITY CODE = 0x10). The visibility of the code will improve if the comment is applied as the example shows.

Table3-5 CAO error code allocation

Error code	Use module
0x80000200～0x800003FF	CAO
0x80000400～0x800005FF	CAO Provider template
0x80000600～0x80000FFF	Other modules
0x80100000～0x8010FFFF	CAO Provider

(2) The error message is defined with StringTable of the resource. The method of registering the error message as a resource is shown here.

1. Double-click the lower blank of StringTable.

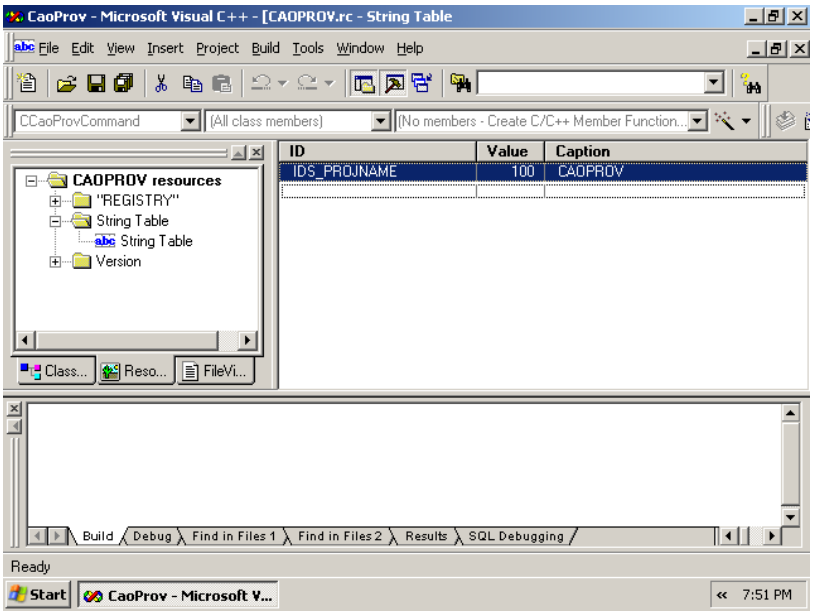


Figure3-1 String Table of CAO provider (initial state)

2. Once the String property is displayed, enter an ID and a caption.
   ID:IDS_E_SAMPLE
   Caption: “This is a test.”

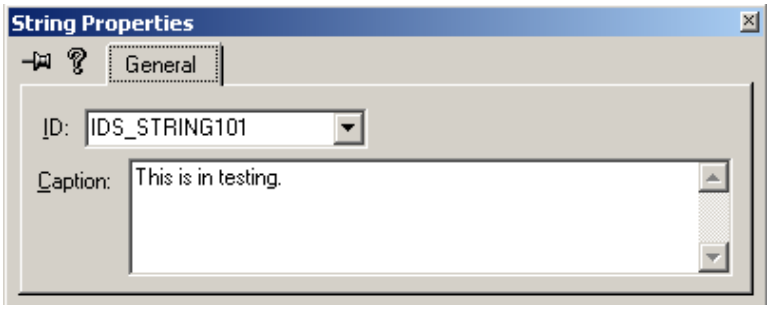


Figure3-2 String Property edit screen

ID of this dialog will be the resource ID of StringTable, and the caption will be the error message. For a resource ID, affix “IDS_” as a prefix, and add an error code defined in (1) (in this case, “E_SAMPLE) behind the “IDS”. The error message was added to the resource of Provider.

Open the Resource View (see Figure3-1) again. Check that the value field of the added resource is stored in 1024-1535. This resource value is allowed to be assigned to the CAO provider’s implementation part. If the resource value is different, it needs to be assigned again by the following procedures.

3. Compile the resource file CAOPROV.rc. A created resource will be registered in Resource.h.
4. In Resource.h, there is a part which assigns a value to the resource ID with the following definitions. Rewrite a value of this part. The example in which it tries to assign the resource assigned in 101 in 1024 is shown as follows. The following example shows when the resource assigned to 101 is re-assigned to 1024.

```
#define IDS_E_SAMPLE 101
  ↓
#define IDS_E_SAMPLE 1024
```

5. Compile CAOPROV.rc again, and check that the value of StringTable has been changed.

Do not rewrite the values other than the resource that you have changed in 2 of this procedure.

(3) Generate an error that includes an error code and resources created in (1) and (2). The error code can be handled same as the general error of HRESULT. However, to send the detailed information of the error (the resource created in (2)) to the client, “Error()” needs to be called. Error() is a method to offer the error information to the client. The definition of Error() is shown as follows.

```
static HRESULT Error( UINT nID,
                      const IID& iid = GUID_NULL,
                      HRESULT hRes = 0,
                      HINSTANCE hInst = _Module.GetResourceInstance() );
```

In order from the top, the arguments are handles for the resource ID, interface ID, the error code, and the resource. For the resource ID, enter the resource ID defined in (2). For the interface ID, enter the ID of interface that returns a created error. For the error code, enter an error name defined in (1). The handle to the last resource needs not be especially specified in this time. (This can be omitted because the default value has been assigned.)

The following example shows how to use the user-defined error "E_SAMPLE".

```
Error( IDS_E_SAMPLE, IID_ICaoProvVariable, E_SAMPLE );
return E_SAMPLE;
```

In the example above, the second argument is an interface ID of CAOVariable class. In the actual operation, enter an interface ID of the class where an error occurs.

3.5. About the message event

CAO provider can generate a message event according to arbitrary timing. The message event generates a message with CreateMessage(), and can generate the event with SendMessage(). The following shows the definitions of CreateMessage() and SendMessage().

```
STDMETHODIMP CreateMessage(
 TMess** ppMess,                    // Message to be created
 long lNumber = 0,                  // Message ID
 VARIANT *vntData = NULL,           // Data to be transmitted
 VARIANT *vntDateTime = NULL,       // Time of message event creation
 BSTR bstrReceiver = NULL,          // Reception destination
 BSTR bstrSender = NULL,            // Transmission source
 BSTR bstrDescription = NULL        // Explanation
);


STDMETHODIMP SendMessage(
 TMess* pMess,                      // Message to be transmitted
 long lOption = 0                   // Option
)
```

For CreateMessage(), the second argument is a message number, and a given value can be specified. For the third argument, enter data to be written as a log. For log output, available VARIANT is VT_BSTR[3] only. For the DateTime in the fourth argument, enter the date and time of message sending. VARIANT type is VT_DATE.[4] For the fifth argument, you do not need to specify anything, because it is disregarded at the log writing request. For the transmission source in the sixth argument, enter an object name where the message is outputted, if necessary. For the seventh argument, enter an error explanation, if necessary.

For SendMessage(), specify the message created by CreateMessage() to the first argument. For the second argument, specify a message option. The value that can be specified for the message option is indicated in Table3-6. CAO_MSG_NORMAL is a general message. A general message is transferred to the client without any processing in CAO engine.

Figure 3-3 shows the CAO message mechanism.

[3] Message event itself can accept various VT types. However, a message is not effectively output excluding the BSTR type though the message event corresponds to various VT types.
[4] If the type of VARIANT is VT_EMPTY, the date when CreateMessage is executed is automatically buried.

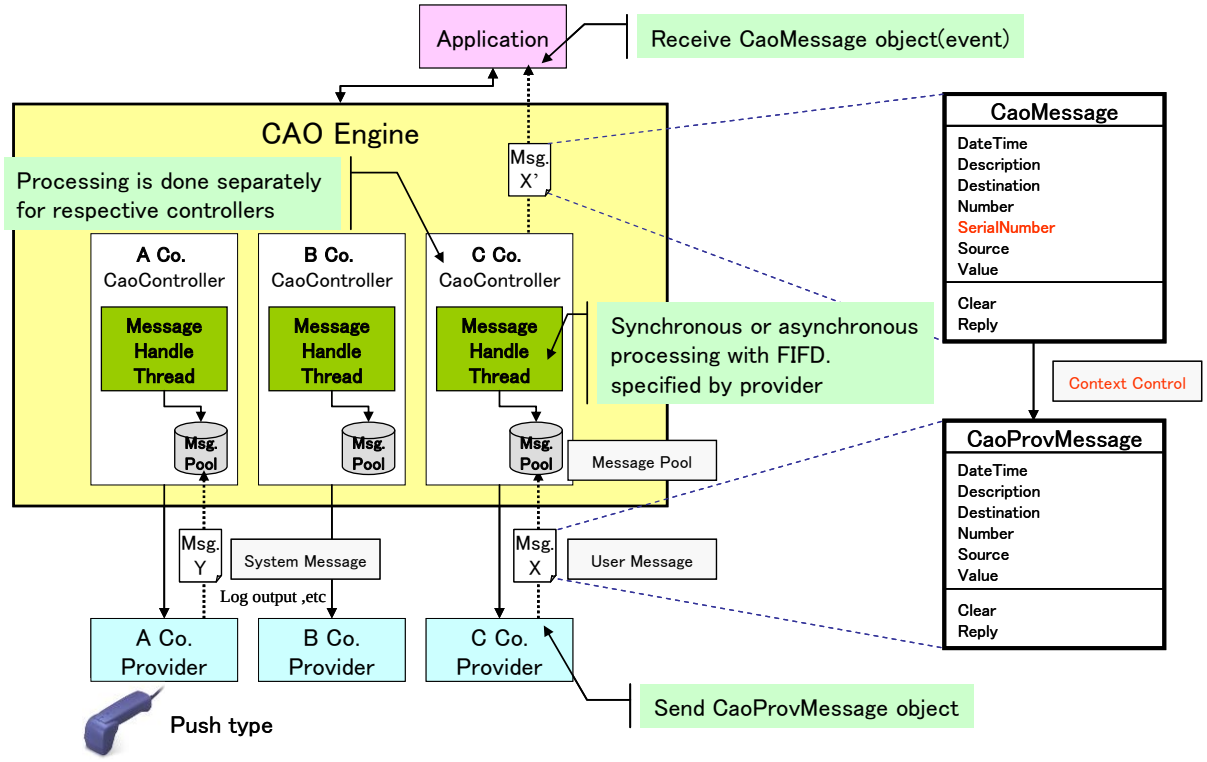


Figure3-3 CAO message mechanism

As Figure3-3 shows, messages are accumulated in the CAO engine's message pool, and then transmitted to the application. The maximum size of this message pool is 1000 pieces. When the number of outputted message exceeds the maximum, the status becomes "Wait" even if the asynchronous-type message it is. The "Wait" state continues until any message pools become empty.

Table3-6 Message option and the operation

	Message option	Operation	Remarks
Normal Message	CAO_MSG_NORMAL (=0x00000000)	Store the message in the message pool. After it is stored, the control will be returned to the provider without confirming the message transmission.	-
Synchronous message	CAO_MSG_SYNC (=0x00010000)	Store the message in the message pool. After it is stored, the control will be returned to the	

		provider once the message transmission is confirmed.	
Log writing request	CAO_MSG_OUTPUT_LOG (=0x00020000)	Output a message as a log.	The lower two bytes of the message option are used to specify the log level. Debug : 0x00020000 Info : 0x00020001 Warn : 0x00020002 Error : 0x00020003 Fatal : 0x00020004
Engine control Message	CAO_MSG_SYSTEM (=0x00040000)	Transmit the control message to the engine.	-
Emergency message	CAO_MSG_BYPASS (=0x00080000)	Transmit a message without storing it to the pool of CAO engine. If any messages are stored in the message pool, this message takes precedence over the stored messages and is executed.	-
In-process message forwarding	CAO_MSG_PROVIDER (=0x00100000)	Transmit a message to the provider.	Set the provider name of the transmission destination to the “Destination" property of the message. A provider which can be selected is limited only to the controller in the same workspace. To send several providers, delimit provider names by comma. (Example “Test1, Test2") If the destination is null, message is transmitted to all providers of the controller collection.

These option values are divided roughly into two parts; the transmission method and the forwarding address. To use an option value, you need to select the combination of these two parts.

The table below shows the combination of the transmission method and the forwarding address.

Table3-7 Combination of message option values

Transmission method / Destination	Normal	Synchronization	Emergency
Client	CAO_MSG_NORMAL (=0x00000000)	CAO_MSG_SYNC (=0x00010000)	CAO_MSG_BYPASS (=0x00080000)
Log	CAO_MSG_OUTPUT_LOG (=0x00020000)	CAO_MSG_OUTPUT_LOG +CAO_MSG_SYNC (=0x00030000)	CAO_MSG_OUTPUT_LOG+CAO_MSG_BYPASS (=0x000A0000)
Engine control	CAO_MSG_SYSTEM (=0x00040000)	CAO_MSG_SYSTEM +CAO_MSG_SYNC (=0x00050000)	CAO_MSG_SYSTEM +CAO_MSG_BYPASS (=0x000C0000)
Provider	CAO_MSG_PROVIDER (=0x00100000)	CAO_MSG_PROVIDER +CAO_MSG_SYNC (=0x00110000)	CAO_MSG_PROVIDER +CAO_MSG_BYPASS (=0x00180000)

The example when usually outputting it to the log by the message is shown as follows.

```
// Create a message
CComPtr<CCaoProvMessage> pMess;
CComVariant vntData(L"This is test.");
HRESULT hr = CreateMessage(&pMess,lType, &vntData);
if (SUCCEEDED(hr)) {
        // Transmission of message
        hr = SendMessage(pMess, CAO_MSG_OUTPUT_LOG);
}
```

3.5.1. Log output by message event

This section explains about "Log output" using the message event. For details about the log output, refer to "2.2.6. Log output" of 'ORiN2 programming guide'. To output the log, specify CAO_MSG_OUTPUT_LOG to the second argument of SendMessage().

The setting about the log output is set with CaoConfig.exe of the CAO support tool. The window of CaoConfig.exe is shown as follows.

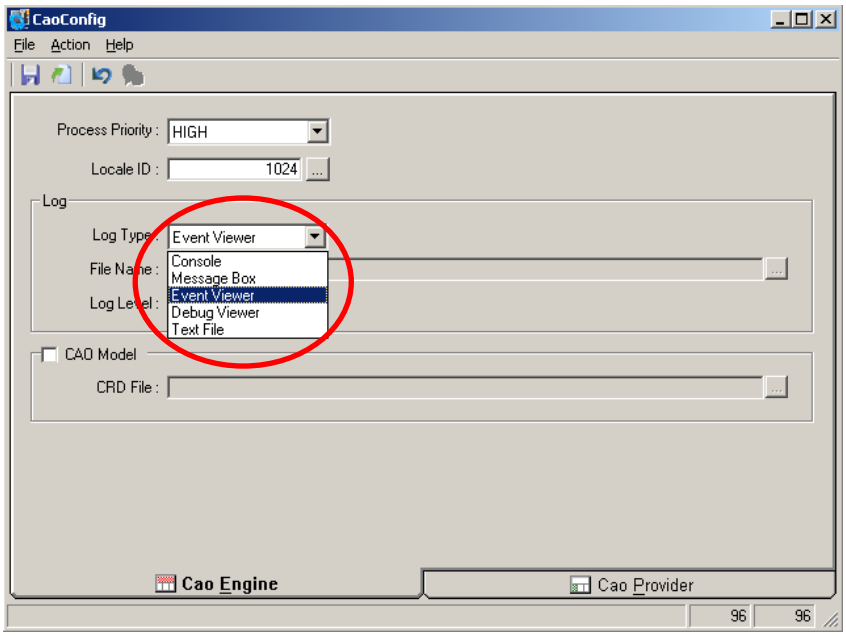


Figure3-4 CAOConfig.exe window

You can choose the output destination of the log from the Log Type combo box. (Refer to Figure3-4.)

Table3-8 Log output destination

Output destination	Remarks
Console	Output to the console.
Message Box	Output to the message box (at the service start).
Event Viewer	Output to the event viewer (at the service start).
Debug Viewer	Debug output
Text File	Output to the specified text file.

For details, refer to "CaoConfig" in the 'ORiN2 programming guide'.

3.5.2. In-process message forwarding

This section explains the way of "In-process message forwarding" that transmits a message to a controller on a different provider.

To execute In-process message forwarding, it is necessary to specify a controller name of the forwarding destination to CaoMessage object. Set a controller name of the forwarding destination to get_Destination() of the message object to be transmitted. You can use the fifth argument of CreateMessage() for the value of

get_Destination(), if CaoMessage class implementation is the default setting. The destination provider of In-process forwarding must exist in the same CaoWorkspace object where the provider of the transmission source belongs. Specifying null character strings will forward a message to all controllers in CaoWorkspace object.

When you send a message, specify CAO_MSG_PROVIDER to the second argument of SendMessage().

At the provider of the forwarding destination, OnMessage() of a controller object is called, and the In-process forwarding message is stored in the first argument. Therefore, to receive In-process forwarding message, OnMessage() of a controller object must be implemented in the forwarding destination provider.

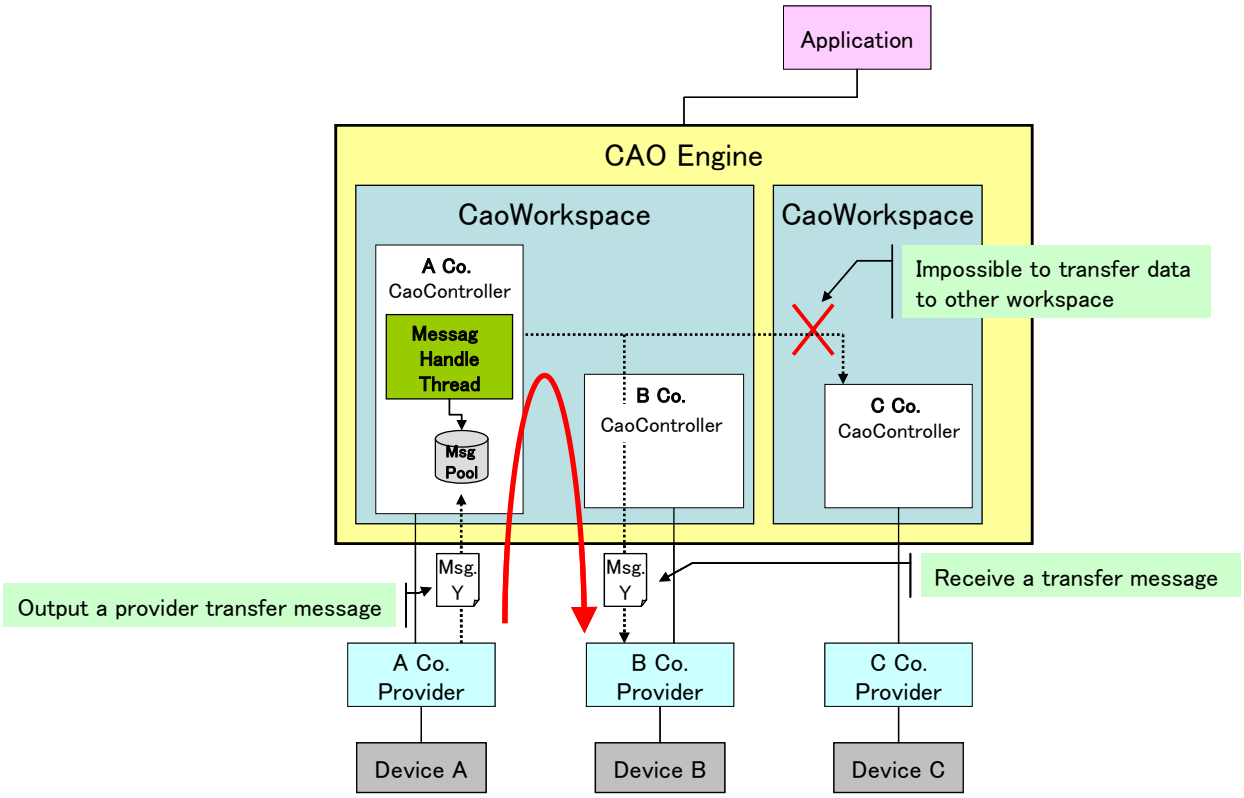


Figure3-5 Message transfer among providers

The following sample shows the transmitting side and the receiving side.

List 3-2 In-process message forwarding (sending side)

```
HRESULT CCaoProvController::CreateInprocMsg(LONG ID, VARIANT vntData, BSTR bstrDest)
{
	HRESULT hr;
	CComPtr<CCaoProvMessage> pMess;	// Message to create

	// Create a message
	hr = CreateMessage(&pMess, ID, &vntData, 0, bstrDest, 0, 0);

	// When the message succeeds, the message is transmitted.
	if (SUCCEEDED(hr)) {
		// Transmission of message
		hr = SendMessageA(pMess, CAO_MSG_PROVIDER);
	}
	return hr;
}
```

List 3-3 In-process message forwarding (receiving side)

```
HRESULT CCaoProvController::FinalOnMessage(ICaoProvMessage *pMsg)
{
	HRESULT hr = S_FALSE;

	if (m_pMsgVar) {
		hr = pMsg->get_Value(&m_vntVal);
	}
	pMsg->Release();
	return hr;
}
```

3.5.3. Issue at constant cycle of message event

CAO equips mechanisms that CAO engine calls CAO providers at a constant cycle and issues an event to the client by the request of the provider. To activate these mechanisms in the provider, set the value of CAOP_TIMER_INTERVAL macro to 1 or higher at the implementation.

In the default setting, CAOP_TIMER_INTERVAL macro is defined in StdAfx.h file of the provider project as follows.

```
#define CAOP_TIMER_INTERVAL		0
// 0: off.  n: Issue OnTimer event every “n” milliseconds. Positive integer of LONG_MAX or less
```

CAOP_TIMER_INTERVAL specifies the interval where the CAO engine calls CAO provider by millisecond (ms). Specifying 0 means there is no call.

If the value of CAOP_TIMER_INTERVAL macro is specified to 1 or higher, CAO engine will call CCaoProvController::OnTimer() of CAO provider at a certain period of time (<CAOP_TIMER_INTERVAL >ms).

The following shows an example of OnTimer() method implementation.

List 3-4 CcaprovController.cpp – OnTimer()

```
#if CAOP_TIMER_INTERVAL > 0
/**    Timer event
 *
 * Redefine "# define CAOP_TIMER_INTERVAL 0" by millisecond-unit.
 * As a result, this function is called at "CAOP_TIMER_INTERVAL"-ms intervals.
 **/
void CCaoProvController::OnTimer()
{
        HRESULT hr;
       CComPtr<CCaoProvMessage> pMess;
       CComVariant vntData(L"Log OK?");
        hr = CreateMessage(&pMess, -1, &vntData);
       if (SUCCEEDED(hr)) {
          SendMessage(pMess);
       }
       return;
}
#endif
```

CAOP_TIMER_INTERVAL mechanism simplifies the coding because CAO provider executes the polling as a proxy of the client side.

3.6. Method of creating macro provider

Macro provider is one of the CAO providers to establish the connection with another CAO provider. Depending on the way of macro provider's implementation, several CAO providers can be connected.

The following shows how to create a CAO provider object that is necessary to create a macro provider and how to obtain OnMessage event.

3.6.1. Method of creating provider object

Among provider objects, a controller object can be solely created from outside. To create a controller object, use CoCreateInstance() or CoCreateInstanceEx().

To create provider objects other than the controller object, use CAO provider's interface. Note that the message object cannot be created with CAO provider's interface. The message object obtains objects created by providers by executing OnMessage event. (Refer to 3.6.2)

The table below lists objects and methods to create these objects.

Table3-9 Provider object and creation method

Provider object	Creation zmethod
CaoProvController	CoCreateInstance() CoCreateInstanceEx()
CaoProvCommand	CaoProvController::GetCommand()
CaoProvExtension	CaoProvController::GetExtension()
CaoProvFile	CaoProvController::GetFile() CaoProvFile::GetFile()
CaoProvRobot	CaoProvController::GetRobot()
CaoProvTask	CaoProvController::GetTask()
CaoProvVariable	CaoProvController::GetVariable() CaoProvExtension::GetVariable() CaoProvFile::GetVariable() CaoProvRobot::GetVariable() CaoProvTask::GetVariable()
CaoProvMessage	OnMassage event

The following program is a sample of function to create the controller object.

List 3-5 Sample.cpp

```
HRESULT CreateCaopCtrl(
BSTR bstrName,
BSTR bstrProvider,
BSTR bstrMachine,
BSTR bstrOption,
ICaoProvController **ppICaopCtrl)
{
    HRESULT hr;

    // Create a CAO provider
    USES_CONVERSION;

    CLSID clsid;
    ICaoProvController *pICaopCtrl;

    hr = CLSIDFromProgID(bstrProvider, &clsid);
    if (SUCCEEDED(hr)) {
        // Create a provider instance
        if (SysStringLen(bstrMachine) == 0) {
            // In-process processing (CLSCTX_INPROC_SERVER)
            hr = CoCreateInstance(clsid,
                                  NULL,
                                  CLSCTX_INPROC_SERVER,
                                  IID_ICaoProvController,
                                  (void **)&pICaopCtrl);
```

```
            } else {
                // Out-process processing
                DWORD dwLen = MAX_COMPUTERNAME_LENGTH + 1;
                LPTSTR pTstr = new TCHAR[dwLen + 1];
                GetComputerName(pTstr, &dwLen);
                if (strcmpi(pTstr, W2T(bstrMachine)) == 0) {
                    // If specified machine name is own machine name
                    // (CLSCTX_LOCAL_SERVER)
                    hr = CoCreateInstance(clsid,
                                          NULL,
                                          CLSCTX_LOCAL_SERVER,
                                          IID_ICaoProvController,
                                          (void **)&pICaopCtrl);
                } else {
                    // If specified machine name is not own machine name
                    // (CLSCTX_REMOTE_SERVER)
                    COSERVERINFO csi = {0, bstrMachine, NULL, 0};
                    MULTI_QI qi = {&IID_ICaoProvController, NULL, S_OK};
                    hr = CoCreateInstanceEx(clsid,
                                          NULL,
                                          CLSCTX_REMOTE_SERVER,
                                          &csi,
                                          1,
                                          &qi);
                    if (SUCCEEDED(qi.hr)) {
                        pICaopCtrl = (ICaoProvController*)qi.pItf;
                    } else {
                        delete [] pTstr;
                        hr = qi.hr;
                        return hr;
                    }
                }
                delete [] pTstr;
            }
        } else {
            hr = E_FAIL;
        }
        if (FAILED(hr)) {
            pICaopCtrl->Release();
            return hr;
        }

        // Connect to the robot controller
        hr = pICaopCtrl->Connect(bstrName, bstrOption);
        if (FAILED(hr))      {
            pICaopCtrl->Release();
            return hr;
        }

        *ppICaopCtrl = pICaopCtrl;

        return hr;
    }
```

3.6.2. Method of acquiring OnMessage event

To obtain OnMessage event from provider objects, EventSink class must be implemented to the macro provider.

For EventSink class implementation, the following procedures must be performed.

(1) Register to the IDL file. (Reference 3.6.2.1)

(2) Implement the processing for connection/disconnection to/from the provider. (Reference 3.6.2.2)

(3) Implement the processing for OnMessage event occurrence. (Reference 3.6.2.2)

(4) Add the EventSink object creation processing. (Reference 3.6.2.3)

The following explains each procedure.

3.6.2.1. Adding EventSink to IDL file

To register the EventSink class in the IDL file of the macro provider, do the followings.

(1) Create an IDL file of the EventSink class.

(2) In the CaoProv.idl file in the macro provider, register the said IDL file by using the "CAOPROV_APPEND_CLASS" macro.

The use of the "CAOPROV_APPEND_CLASS" macro is shown below.

```
#Define CAOPROV_APPEND_CLASS <The path to the IDL file in the EventSink class >
```

The sample code below shows how to register the IDL file to the EventSink.idl file macro provider.

```
#define CAOPROV_APPEND_CLASS "EventSink.idl"
```

CAO provider will include an IDL file that exists on the path specified by "CAOPROV_APPEND_CLASS" macro.

List 3-6 shows a sample of the IDL file in the EventSink class.

List 3-6 Sample.idl

```
// IEventSink Interface
interface IEventSink;
[
    object,
    uuid(B3E98B1A-0D28-42c8-84A1-1354891EA216),
    dual,
    helpstring("IEventSink Interface"),
    pointer_default(unique)
]
interface IEventSink : IDispatch
{
     [id(1), helpstring("method OnMessage")] HRESULT OnMessage([in] ICaoProvMessage
*pICaopMsg);
};
// EventSink Class
 [
    uuid(D178B708-9330-4b87-B4FE-A540F1D4C55B),
    helpstring("EventSink Class")
]
coclass EventSink
{
     [default] interface IEventSink;
};
```

3.6.2.2. Implementation of EventSink class

EventSink class must implement the connection and disconnection processing to/from the object that creates OnMessage event. Use AdlAdvise() and AtlUnadvise() for connection and disconnection, respectively. For details about these functions, refer to MSDN.

The sample of the connection and the disconnection process is shown as follows.

List 3-7 Sample.cpp

```
// Connection processing
STDMETHODIMP CEventSink::Connect(ICaoProvController *pICaopCtrl)
{
     // Connect with the connectable object of CaoProvider.
     HRESULT hr = AtlAdvise(pICaopCtrl, GetUnknown(), DIID__ICaoProvControllerEvents,
&m_dwCookie);
     if (SUCCEEDED(hr)) {
         // Store the ICaoProvController interface
             m_pICaopCtrl = pICaopCtrl;
     }
     return hr;
}

// Disconnection processing
STDMETHODIMP CEventSink::Disconnect()
{
     // Disconnect from the connectable object of CAO provider
     // and then disregard event notification.
     if (m_dwCookie) {
         AtlUnadvise(m_pICaopCtrl, DIID__ICaoProvControllerEvents, m_dwCookie);
     }
     m_dwCookie = 0;
     return S_OK;
}
```

OnMessage event is issued for the method which DISPID of the EventSink interface registered in 3.6.2.1 is “0x01”. Therefore, the OnMessage event processing is implemented in the method which DISPID is “0x01”.

List 3-8 is the sample of the OnMessage receiving method that sends the message received from the provider to the CAO engine as an event without any modification.

List 3-8 Sample.cpp

```
STDMETHODIMP CEventSink::OnMessage(ICaoProvMessage *pICaopMsg)
{
     HRESULT hr;
     // Issue an event of the Controller class
     hr = m_pCaopCtrl->Fire_OnMessage((IUnknown*)pICaopMsg);
     return hr;
}
```

3.6.2.3. Generation of EventSink

To receive an OnMessage event, you need to execute the connection processing of the EventSink class as well as the creation of the EventSink class.

List 3-9 is the sample of the creation and destruction of the EventSink.

List 3-9 Sample.cpp

```
HRESULT CreateEventSink(ICaoProvController *pICaopCtrl, CEventSink *ppEventSink)
{
     HRESULT hr;

     // Create an instance of CEventSink
     CEventSink *pEvent;
     hr = CComObject<CEventSink>::CreateInstance(&pEvent);
     if (FAILED(hr)) {
         return hr;
     }
     pEvent->AddRef();

     // Connect with a provider
     hr = pEvent ->Connect(pICaopCtrl);
     if (FAILED(hr)) {
         pEvent->Release(m_pICaopCtrl);
         return hr;
     }

     ppEventSink = pEvent;

     return hr;
}

HRESULT DeleteEvntSink(CEventSink *pEventSink)
{
     HRESULT hr;

     // Disconnect from the provider
     hr = pEventSink->Disconnect();
     // Destroy the EventSink
     pEventSink->Release();

     return hr;
}
```

3.7. Usage of communication class

3.7.1. Introduction

In ORiN2 SDK, some communication classes are used to create CAO providers. This chapter explains how to use the following communication classes;

- CDevice class; a common class for communication,

- CSerial class; a class that inherits CDevice class and is used for RS-232C communication
- CTCPServer class; a class used for TCP/IP communication
- CTCPClient class; a class for TCP/IP communication (hereafter, TCP socket class)
- CUDPSocket class; a class for UDP communication.

For about Device class-related source files, such as Device.cpp/h, Serial.cpp/h, Socket.cpp/.h, TCPServer.cpp/h, TCPClient.cpp/h, and UDPSocket.cpp/h, you need to add desired files from ORiN2¥CAO¥Include.

3.7.2. CDevice class

CDevice class is an abstract class to communicate the device.

To implement a class for device connection, inherit this CDvice class, and then implement the connection processing and the transmission processing to the virtual functions. Also, the CDevice class offers the following functions as common functions.

- Binary mode and text mode

  Binary mode; Data is sent and received as is.

  Text mode; Data is sent and received with some modifications, such as the character-code conversion, adding a header or terminator, and so on.

- Setting of header and terminator

  At the text mode, a header and terminator are added to the data at the transmission.

- Unicode conversion mode

  At the text mode, the input data is converted to the ASCII before it is sent, or reversely, the received data is converted to Unicode before it is received.

- ISO code conversion and EIA code conversion

  At the text mode, the sending/receiving data is converted to ISO- or EIA-code before the communication.

The following shows the tree display of the CDevice class.

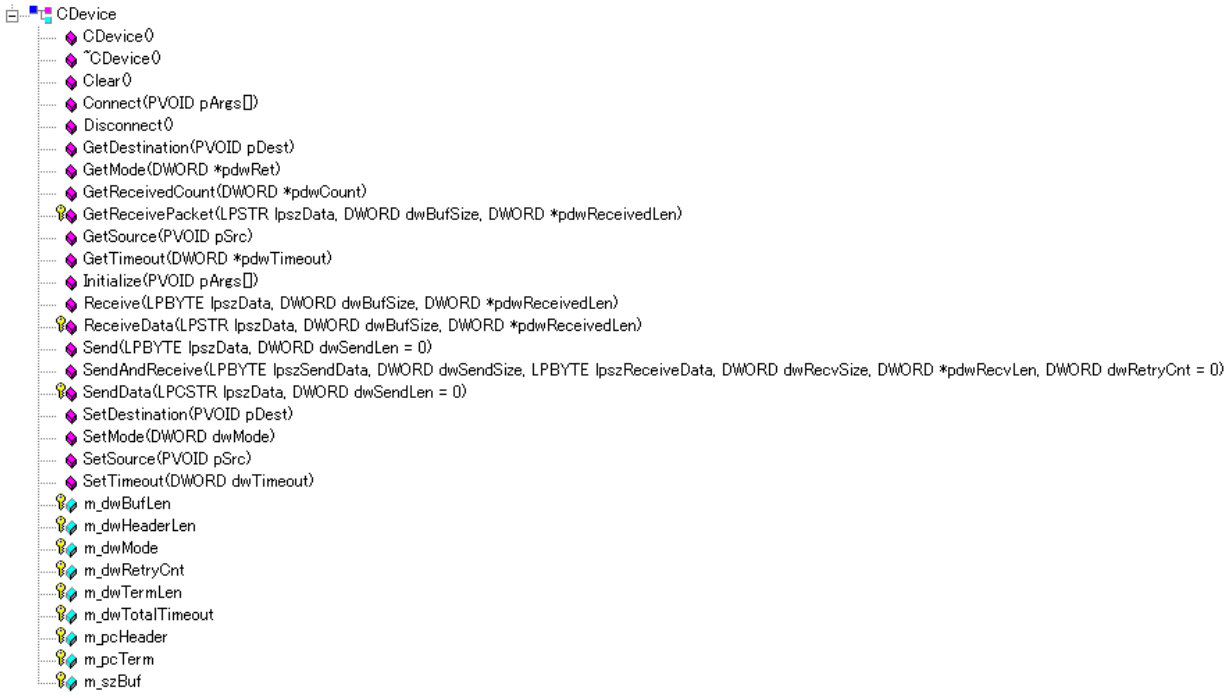


Figure3-6 Class tree of CDevice

Table3-10 shows the member list of the CDevice class. To inherit the CDevice class, override the methods written in boldface in this table, depending on your needs.

Table3-10 Member list of CDevice

	Member name	Explanation
	CDevice()	Constructor of CDevice
	~CDevice()	Destructor of CDevice
	Connect()	Buffer clear. Clear the buffer and errors.
	Disconnect()	Connection processing. Connect with the target.
	Flush()	Disconnection processing. Disconnect from the target.
	GetDestination()	Acquire the destination information.
	GetMode()	Acquire the currently selected operational mode
	GetReceivedCount()	Data reception number acquisition processing. Acquire the number of data that currently exists in the buffer.
	GetReceivePacket()	Packet acquisition processing. Acquire the data from the local reception buffer. At the text mode, the data from the header to the terminator is acquired

		as one packet. If there is no header, the data from the top of the buffer to the terminator is acquired. If there is no terminator, the data from the header to the end of the buffer is acquired. At the binary mode, data of the specified size or all data in the buffer is acquired.
	GetSource()	Acquire the transmission source information.
	GetTimeOut()	Time-out time acquisition processing. Acquire the time of the time-out.
	Initialize()	Initialization. Initialize mode settings and various conversion function settings, and so on.
	Receive()	Reception processing. Receive the character string data from the target. "ReceiveData" is called to process the data. At the text mode, the received data that have removed the header and terminator and have been converted to Unicode is returned. At the binary mode, the received data is returned as is.
	ReceiveData	Virtual function for data reception. Store the data received from the device in the buffer, and then cut one packet to return it.
	Send	Transmission processing. Send the specified character string data to the target. At the text mode, SendData is called after the addition of the header and terminator, ASCII conversion of character code, and so on. At the binary mode, SendData is called as is.
	SendAndReceive	Sending and receiving batch processing. Transmission to reception is processed as a batch processing.
	SendData	Virtual function for data transmission. Send the data received from Send to the device.
	SetDestination()	Set the destination information.
	SetMode()	Set the operational mode again.
	SetSource()	Set the transmission source information.
	SetTimeOut	Set the timeout period.
	m_bConnected	Connected flag.
	m_dwHeaderLen	Header length.
	m_dwLocalBufferLen	Local reception buffer length.
	m_dwMode	Operational mode.
	m_dwRetryCnt	Maximum number of the retry.
	m_dwTermLen	Length of the terminator. 1 to 2.
	m_dwTotalTimeOut	Maximum waiting time required for the character string transmission.

	m_pcHeader	Header code.
	m_pcTerm	Terminator code.
	m_szLocalBuffer	Local reception buffer.

: Public function : Protected function : Protected variable

3.7.3. CSerial class

3.7.3.1. Usage

This section explains how to use the CSerial class that inherits the CDevice class and is designed for the serial communication.

CSerial classes send and receive data by connecting with COM ports specified by Connect methods, and using the Receive methods and the Send methods.

The following shows how to execute RS-232C communication by using the CSerial class.

(1) Create an object

To use the CSerial class, call a constructor first for the object creation as follows.

```
// Create a CSerial object
CSerial *m_pSerial;
m_pSerial = (new CSerial);
```

(2) Initialization

Initialize the CSerial class by setting necessary parameters, such as a header and terminator added to a character string to be transmitted, the character code conversion, and so on.

```
PVOID pArgs[3];
CHAR cHeader[] = "¥0";
CHAR cTerm[] = "¥r";
DWORD dwMode = ST_MODE_TEXT;
pArgs[0] = cHeader;       // Header code
pArgs[1] = cTerm;         // Terminator
pArgs[2] = &dwMode;       // Operational mode

hr = m_pSerial->Initialize(pArgs);
```

At the initialization, specify the following four parameters for the PVOID type array.

・ Header code

Specify a header code that is added to the sending data to show the start of data.

・ Terminator code

Specify a terminator code that is added to the sending data to show the terminal of data.

・Operational mode

You can specify an operation mode at the data transmission. The operational mode can be selected from the following five types by the macro defined by Device.h. To specify two or more types, apply a logical disjunction of the following macros.

・**ST_MODE_BINARY**

Data is sent or received by the binary data type (byte array). The specified data is sent or received as it is. No addition and the removal of neither header nor the terminator are performed.

・**ST_MODE_TEXT**

Data is sent or received by the text data type (character string). Both a header and terminator are added to the character string at the data sending. When receiving data, the character string that deletes the header and the terminator from receive data is returned.

・**ST_MODE_WBSTOWCS**

Data is converted from Unicode (WCHAR) into S-JIS (CHAR), and then sent or received. This mode cannot be specified at the binary mode.

・**ST_MODE_ASCTOISO**

Data is converted from ASCII code into the ISO code, and then sent or received. When this mode is used together with the UNICODE conversion function, data is converted into the code specified by UNICODE and then sent or received.

・**ST_MODE_ASCTOEIA**

Data is converted from ASCII code into the EIA code, and then sent or received it. When this mode is used together with the UNICODE conversion function, data is converted into the code specified by UNICODE and then sent or received.

(3) Open a communication (COM) port.

Open a COM port according to the items specified at the initialization processing. To open a COM port, do as the followings.

```
PVOID pArgs[1];
DWORD dwPortNo = 1;
pArgs[0] = &dwPortNo;     // Serial port number

hr = m_pSerial->Connect(pArgs);
```

When you connect a COM port, specify the following parameter to the PVOID type array.

・Serial port number

Specify a serial port number that is used for data transmission. For example, to use COM1, specify “dwPortNo=1;” Same as this example, to use COM*n*, specify “dwPortNo=*n*;”

(4) Set communication parameters

Set various communication parameters, such as the transmission rate (baud rate), data bit, stop bit, parity, the flow control, and the time-out, etc.

To set these communication parameters, use the SetState method and the GetState method. The arguments of these two methods are the DCB structure that stores information on communication devices. The SetState function sets the device by using the DCB structure. The GetState function returns a present construction. To use these parameters, prepare an instance of the DCB structure first. With the GetState, read the default setting to this DCB structure. And then, change the values of the DCB structure’s variables that have been read, and then, in order to enable the changes, set the communication parameters by specifying this DCB structure to the SetState. Please refer to the site of MSDN and Microsoft for the DCB structure.

The following sample shows a concrete usage of the SetState method, the GetState method, and the DCB structure. In the following example, the hardware flow control is specified for handshaking. The RTS signal is set to the controlled-state (dcb.fDtrControl = DTR_CONTROL_ENABLE;), and the DTR signal is activated when the device opens, and the status of DTR signal will be kept.

```
DWORD dwBaudRate, dwDataBits, dwParity, dwStopBits;
DCB dcb;

// Substitute the default value.
dwBaudRate      = CBR_19200;       // 19200bps
dwDataBits      = 8;               // 8 bits
dwParity        = NOPARITY;        // Without parity
dwStopBits      = TWOSTOPBITS;     // Two stop bits

hr = m_pSerial->GetState(&dcb);    // Obtain the currently selected communication parameters.

if (SUCCEEDED(hr)) {
       dcb.BaudRate = dwBaudRate;                      // Speed
       dcb.ByteSize = (BYTE)dwDataBits;                // Data length
       dcb.Parity = (BYTE)dwParity;                    // Parity
       dcb.StopBits = (BYTE)dwStopBits;                // Stop bit
       dcb.fOutX = true;                               // XON/XOFF is transmitted.
       dcb.fInX = true;                                // XON/XOFF is received.
       dcb.fOutxCtsFlow = true;                        // Enable the CTS signal
       dcb.fOutxDsrFlow = false;                       // Set the DSR signal to invalid
       dcb.fRtsControl = RTS_CONTROL_HANDSHAKE;        // Set the RTS signal to controlled-state
       dcb.fDtrControl = DTR_CONTROL_ENABLE;           // Turn ON the DTR signal
       dcb.fDsrSensitivity = false;
       hr = m_pSerial->SetState(&dcb);                 // Setting of the communication parameter
}
```

Set the time-out period (millisecond) at the data transmission.

```
DWORD dwTimeout;
dwTimeout = 500;
hr = m_pSerial->SetTimeOut( dwTimeout );
```

(5) Read /Write data

Read (receive) or write (send) data from/to the COM port. If any errors occur, execute an Error clear and Communication buffer clear (Flush).

To read and write data from/to COM port, use the Receive method and the Send method, respectively. The SendAndReceive method is capable of data sending and receiving simultaneously.

```
define LOCAL_BUFFER_MAX   16
HRESULT hr;                                  // Return code
BYTE lpszSendData[ LOCAL_BUFFER_MAX +1 ];    // Sending data area
BYTE lpszReceiveData[ LOCAL_BUFFER_MAX +1 ];// Receiving data area
DWORD dwSendLen;                             // Sending data length
DWORD pdwRecvLen;                            // Receiving data length
```

- **Data sending**

```
dwDataLen = 10;                              // Data length
hr = m_pSerial->Send(      lpszSendData,     // Address of data to be sent.
                           dwSendLen,        // Data length to be sent.
                                    // (Specifying 0 will send the data until the null terminator)
                           false);           // Header and terminator are not added.
                                             // (This should be true when it is omitted.)
```

- **Data reception**

```
hr = m_pSerial->Receive(   lpszData,         // Storage destination address of receiving data
                           10,               // Maximum receivable size
                           &pdwRecvedLen);   // Number of receiving data
```

- **Data sending and receiving**

```
hr = m_pSerial->SendAndReceive(    lpszSendData,    // Character strings of sending data
                                   0,               // Data length to be sent.
                                  // (Specifying 0 will send the data until the null terminator)
                           lpszReceiveData,// Storage destination address of receiving data
                           10,             // Maximum receiving size
                           &pdwRecvLen,    // Number of receiving data
                           2);             // Number of retry count (omittable)
```

(6) Close the communication (COM) port

Close the COM port. To close the COM port, call the Disconnect method, and then release the memory with “delete”. If the COM port has been opened from a program, be sure to close the COM port before exiting the application. It becomes impossible to use the opened COM port by other applications when this is forgotten.

```
// Close the COM handle.
if (m_pSerial) {
       m_pSerial->Disconnect();
       delete m_pSerial;
       m_pSerial = NULL;
}
```

3.7.3.2. Error code

In the serial communication class, a value that is a Windows standard error masked by 0x80900000 is returned as an original error code.

Ex) Windows error : 0x02 (File not found) >>> CAO API error : 0x80900002

3.7.4. TCP socket class

3.7.4.1. Usage

This section explains how to use the TCP socket class that inherits the CDevice class and is designed for the socket communication (TCP/IP) with Ethernet.

Unlike the CSerial class, the TCP socket class uses different classes for the server mode connection and the client mode connection. The server mode uses the CTCPServer class, whereas the client mode uses a CTCPClient class.

The CTCPSever class creates the CTCPClient class by accepting the connection from the client. Once the connection has been established, the data can be sent or received by calling a Send method or a Receive method in the CTCPClient class.

[Connection method in the server mode]

(1) Create an object

To use the CTCPServer class, call a constructor first to create an object as follows.

```
// Create a CTCPServe
CTCPServer *  m_pTCPServer;
m_pTCPServer = (new CTCPServer);
```

(2) Initialization

Initialize the CTCPServer class by setting necessary parameters.

```
HRESULT hr;
PVOID pInitArgs[3];
CHAR cHeader[] = "¥0";
CHAR cTerm[] = "¥r";
DWORD dwMode = ST_MODE_TEXT;
pInitArgs [0] = cHeader;  // Header code
pInitArgs [1] = cTerm;    // Terminator code
pInitArgs [2] = &dwMode;  // Operation mode
hr = m_pTCPServer->Initialize(pInitArgs);

PVOID pConArgs[2];
DWORD dwMyIP = INADDR_NONE;
DWORD dwMyPort = 5006;
pConArgs[0] = &dwMyIP;    // Local IP address
pConArgs[1] = &dwMyPort;  // Local port number
hr = m_pTCPServer->Connect(pConArgs);
```

Parameters required for the initialization are the header code, terminator code, and the operation mode. (The contents are the same as the CSerial class.)

To execute Connect, specify IP address and the port number that wait the connection from the socket communication.

(3) Wait the connection

Objects in the TCPServer class do not send and receive data spontaneously. They wait a request from the client and will create a new connection point every time a connection is requested.

In the actual operation, a CTCPSocket pointer for establishing a new connection is passed by the argument of the Connect method, and then a new object is assigned to the argument when a connection is requested from the client side.

```
CTCPClient* pDev;
hr = m_pTCPServer->Accept(&pDev);
```

This object newly assigned is used to send and receive data.

[Client mode]

(1) Create an object

To use the CTCPClient class, call a constructor first to create an object as follows.

```
// Create a CTCPClient object
CTCPClient  *  m_pConnDevice;
m_pConnDevice = (new CTCPClient);
```

(2) Initialization

Initialize the CTCPClient class by setting necessary parameters.

```
HRESULT hr;
PVOID pInitArgs[3];
CHAR cHeader[] = "¥0";
CHAR cTerm[] = "¥r";
DWORD dwMode = ST_MODE_TEXT;
pInitArgs [0] = cHeader;  // Header code
pInitArgs [1] = cTerm;    // Terminator code
pInitArgs [2] = &dwMode;  // Operation mode
hr = m_pConnDevice->Initialize(pInitArgs);
```

Parameters required for the initialization are the header code, terminator code, and the operation mode. (The contents are the same as the CSerial class.)

(3) Establish a connection

Request a connection for the port where the server program is waiting. If the connection is successfully completed, the data will be sent and received with this object.

```
PVOID ConArgs[4];
DWORD dwMyIP = INADDR_NONE;
DWORD dwMyPort = 0;
DWORD dwSrvIPAdr = inet_addr("127.0.0.1");
DWORD dwSrvPortNo = 5006;
pConArgs[0] = &dwMyIP;              // Local IP address
pConArgs[1] = &dwMyPort;            // Local port number
```

```
pConArgs[2] = &dwSrvIPAdr;          // Connection destination IP address
pConArgs[3] = &dwSrvPortNo;         // Connection destination port number
hr = m_pConnDevice->Connect(pConArgs);
```

To execute Connect, specify the local IP address and port number, the connection destination's IP address and port number.

[Data sending and receiving in TCP socket class]

Once the connection is established by the method mentioned above, the data sending and receiving will be available for both the server mode and the client mode with the same way. It can be either the server mode or the client mode, and data can be similarly sent and received by establishing the connection by the above-mentioned method. (Note that the CTCPServer is not implemented the data sending and receiving process. Please use the CTCPClient object created by the Accept method.)

(1) Set the time-out period

Set the timeout period (millisecond) at the data sending and receiving.

```
HRESULT hr;                                 // Return code
DWORD dwTimeout = 500;
hr = m_pConnDevice->SetTimeOut(dwTimeout);  // Set the timeout period to 500msec.
```

(2) Send and receive the data

To execute the data sending and receiving with the socket communication, use following ways.

・ **Data sending**

```
HRESULT hr;                                 // Return code
DWORD dwDataLen = nLength;                  // Data length
hr = m_pConnDevice->Send(  lpszData,        // Address ot the data to be sent
                           dwSendLen,       // Data length to be sent (omittable).
                           bHeadTerm);      // Addition of header and terminator (omittable).
```

・ **Data receiving**

```
HRESULT hr;                                 // Return code
#define LOCAL_BUFFER_MAX  16
DWORD dwBufSize;                            // Receiving data length
BYTE lpszData[ LOCAL_BUFFER_MAX +1 ];       // Receiving data area
/// Data receiving processing
dwDataLen = 1L; // Receive the data per one byte.
hr = m_pConnDevice->Receive( lpszData,      // Storage destination address of receiving data
                           dwBufSize,       // Buffer size
                           pdwRecvedLen);   // Number of receiving data
```

・ **Data sending and receiving**

```
define LOCAL_BUFFER_MAX   16
HRESULT hr;                                         // Return code
DWORD dwRecvSize;                                   // Receiving data length
DWORD pdwRecvLen;                                   // Number of receiving data
BYTE lpszSendData [ LOCAL_BUFFER_MAX +1 ];          // Receiving data area
BYTE lpszReceiveData [ LOCAL_BUFFER_MAX +1 ];       // Sending data area
```

```
pdwRecvLen = 1L;
hr =  m_pConnDevice->SendAndReceive (lpszSendData // Sending data character string
                                0,                      // Size of the sending data
                                       // (Specifying 0 will send the data until the null terminator)
                                lpszReceiveData, // Storage destination address of receiving data
                                dwRecvSize,      // Buffer size of lpszData
                                pdwRecvLen,      // Number of receiving data
                                dwRetryCnt);     // Number of retry count (omittable)
```

(3) Close the connection.

Once the data sending/receiving is finished, the disconnection processing must be performed. Use the Disconnect method for disconnection, as follows.

```
// Close the COM handle
if (pConnDevice) {
       pConnDevice->Disconnect();
       delete pConnDevice;
       pConnDevice = NULL;
}
```

3.7.4.2. Error code

In the TCP socket class, a value that is a Winsock error masked by “0x80910000” is returned as an original error.

Ex) Winsock error : 10061 (Connection denied) >> CAO API error : 0x8091274D

3.7.5. CUDPSocket class

3.7.5.1. Usage

This section explains how to use the CUDPSocket class that inherits the CDevice class and is designed for the socket communication (UDP) with Ethernet.

Same as the CTCPSocket class, the CUDPSocket class sends and receives data by calling the Receive method and the Send method.

(1) Create an object

To use the CUDPSocket class, call a constructor first to create an object as follows.

```
// Create a CUDPSocket object
CUDPSocket *  m_pConnDevice;
m_pConnDevice = (new CUDPSocket);
```

(2) Initialization

Initialize the CUDPSocket class by setting necessary parameters.

```
HRESULT hr;
PVOID pInitArgs[3];
CHAR cHeader[] = "¥0";
CHAR cTerm[] = "¥r";
```

```
DWORD dwMode = ST_MODE_TEXT;
pInitArgs [0] = cHeader;  // Header code
pInitArgs [1] = cTerm;    // Terminator code
pInitArgs [2] = &dwMode;  // Operation mode
hr = m_pConnDevice ->Initialize(pInitArgs);
```

Parameters required for the initialization are the header code, terminator code, and the operation mode. (The contents are the same as the CSerial class.)

(3) Open the socket

Open the socket to prepare for the data sending and receiving.

```
PVOID ConArgs[2];
DWORD dwMyIP = INADDR_NONE;
DWORD dwMyPort = 0;
pConArgs[0] = &dwMyIP;    // Local IP address
pConArgs[1] = &dwMyPort;  // Local port number
hr = m_pConnDevic ->Connect(pConArgs);
```

For the setting parameters, specify the local IP address and the local port number.

(4) Setting of destination

Set the destination parameters.

```
DWORD dwDist[2]
DWORD dwSrvIPAdr = inet_addr("127.0.0.1");
DWORD dwSrvPortNo = 5006;
dwDist [2] = &dwSrvIPAdr;   // Connection destination IP address
dwDist [3] = &dwSrvPortNo;  // Connection destination port number
m_pConnDevice->SetDestination(dwDist);
```

For the setting parameters, specify the destination's IP address and the destination's port number.

(5) Set the time-out period

Set the timeout period (millisecond) at the data receiving. by the following way.

```
DWORD dwTimeout = 500;
hr = m_pConnDevice->SetTimeOut(dwTimeout); // Set the timeout period to 500msec.
```

(6) Send and receive the data

To execute the data sending and receiving with the socket communication, use following ways.

・ **Data sending**

```
DWORD dwDataLen = nLength;                  // Data length
HRESULT hr;                                 // Return code
hr = m_pConnDevice->Send(  lpszData,        // Address of data to be sent
                           dwSendLen,       // Length of data to be sent (omittable).
```

```
                                bHeadTerm);       // Addition of header and terminator (omittable).
```

・ **Data receiving**

```
#define LOCAL_BUFFER_MAX  16
HRESULT hr;                                     // Return code
DWORD dwBufSize;                                // Receiving data length
BYTE lpszData[ LOCAL_BUFFER_MAX +1 ];           // Receiving data area
/// Data receiving processing
dwDataLen = 1L; // Receive the data per one byte.
hr = m_pConnDevice->Receive( lpszData,          // Storage location address of receivinge data
                             dwBufSize,         // Buffer size
                             pdwRecvedLen);     // Number of receiving data
```

・ **Data sending and receiving**

```
define LOCAL_BUFFER_MAX   16
HRESULT hr;                                              // Return code
DWORD dwRecvSize;                                        // Receiving data length
DWORD pdwRecvLen;                                        // Numberof receiving data
BYTE lpszSendData [ LOCAL_BUFFER_MAX +1 ];               // Receiving data area
BYTE lpszReceiveData [ LOCAL_BUFFER_MAX +1 ];            // Sending data area
pdwRecvLen = 1L;
hr = m_pConnDevice->SendAndReceive(LpszSendData          // Sending data character string
                                0,     // Transmission data size
                                       //(Specifying 0 will send the data until the null terminator)
                                lpszReceiveData, // Storage destination address of receiving data
                                dwRecvSize,      // Buffer size of lpszData
                                pdwRecvLen,      // Number of receiving data
                                dwRetryCnt);     //  Number of retry count (omittable)
```

(7) Close the connection

Once the data sending/receiving is finished, the disconnection processing must be performed. Use the Disconnect method for disconnection, as follows.

```
// Close the COM handle
if (m_pConnDevice) {
 m_pConnDevice->Disconnect();
 delete m_pConnDevice;
 m_pConnDevice = NULL;
}
```

3.7.5.2. Error code

In the UDP socket class, a value that is a Winsock error masked by “0x80910000” is returned as an original error.

Ex） Winsock error : 10061 (Connection denied) >> CAO API error : 0x8091274D

4. Tips for Provider development

This chapter shows useful tips for the impleientation of the CAO provider.

4.1. Creating a variable object that uses parent object

By storing a pointer to the parent object as a member when the variable object is created, the variable object will be capable of calling methods and priperties of the parent object.

Adding these functions will give the following advantage.

- Methods and properties implemented by the parent object will be open for the client that can access only the variable classes (such as CaoSQL).

The following shows concrete implementation examples. In this example, get_Value, which is the variable in the File class (example: @VALUE), is redirected to get_Value, which is the file object.

(1) Add a storage destianation of the pointer to the parent object to the member of the variable object.

List 4-1 CaoProvVariable.h

```
#ifndef __CAOPROVVARIABLE_H_
#define __CAOPROVVARIABLE_H_

#include "CaoProvVariableImpl.h"

class CCaoProvFile;                // Temporary declaration of parent class

class ATL_NO_VTABLE CCaoProvVariable :
	public ICaoProvVariableImpl<CCaoProvVariable>
{
protected:
	HRESULT FinalInitialize(PVOID pObj);
	void FinalTerminate();

//	HRESULT FinalGetAttribute( /*[out, retval ]*/ long *pVal );
//	HRESULT FinalGetHelp(/*[out, retval]*/ BSTR *pVal);
//	HRESULT FinalGetDateTime(/*[out, retval]*/ VARIANT *pVal);
	HRESULT FinalGetValue(/*[out, retval]*/ VARIANT *pVal);
	HRESULT FinalPutValue(/*[in]*/ VARIANT newVal);
//	HRESULT FinalGetID(/*[out, retval]*/ VARIANT *pVal);
//	HRESULT FinalPutID(/*[in]*/ VARIANT newpVal);
//	HRESULT FinalGetMicrosecond(/*[out, retval]*/ long *pVal);

private:
	CCaoProvFile* m_pFile;      // Parent class's pointer storage destination
};

#endif //__CAOPROVTASKVARIABLE_H_
```

(2) Store the pointer to the parent object in the member by initializing the variable class. If the parent object is discarded by the CAO engine, the reference couting (AddRef) is not required because functions of the variable class will not be called.

List 4-2 CaoProvVariable.cpp - FinalInitialize()

```
HRESULT CCaoProvVariable::FinalInitialize(PVOID pObj)
{
	switch (m_ulParentType) {
	case SYS_CLS_FILE:
		if (!_wcsicmp(m_bstrName, L"@Value")) {
		   m_pFile = (CCaoProvFile*)pObj; //Store the pointer for the parent object
		}
		break;

	default:
		return E_NOTIMPL;
	}

	return S_OK;
}
```

(3) Use the method of the parent object. In the following examples, the results are called with CaoProvVariable::FinalGetValue().

List 4-3 CaoProvVariable.cpp - FinalGetValue()

```
HRESULT CCaoProvVariable::FinalGetValue(VARIANT *pVal)
{
	return m_pFile->get_Value(pVal);
}
```

List 4-4 CaoProvVariable.cpp - FinalGetValue()

```
HRESULT CCaoProvVariable::FinalPutValue(VARIANT newVal)
{
	return m_pFile->put_Value(newVal);;
}
```

5. Creating a TCmini provider

This chapter introduces a CAO provider that controls TCminiα TC3-02, which is a programmable small controller from Toshiba Machine, as a concrete example. However, note that TCmini provider does not equip all the interfaces specified by the CAO provider of ORiN Ver 2.0. In this example, only the methods required for the reading/writing of Tcminiα TC3-02's relays and registers are implemented, based on the procedures written in Chapter 2.

In this document, a programmable small controller TCminiα TC3-02 from Toshiba Machine is called 'TCmini controller', and a CAO provider that is created in this chapter for the TCmini controller is called 'TCmini provider'.

Before creating the TCmini provider, the following items shoule be prepared.

- Microsoft Visual C++ 6.0 SP5 or more
- ORiN2 SDK or more
- TCmini controller main unit
- C++ language and basic knowledge of COM
- Basic knowledge of serial communications and socket communication

5.1. What is the TCmini controller?

5.1.1. Composition

The TCmini controller is a programmable small controller that consists of 16 photo coupler inputs, 16 relay outputs, eight DIP switch inputs, four temperature inputs (thermisters), two analog inputs (0-5V), two analog outputs (0-5V), calender function, RS-232C communication function, RS-485 communication function, and expansion function.

In TCmini controller, a relay uses one bit, a register uses 16 bits. Relays and registeres are basically expressed as "<Function group sign> + <Address>". "Function group sign" stands for the type of the function, and is specified by an aplhabet character ('X', 'Y', 'R', 'T', 'C', 'L', 'E', 'A', 'D', 'V', or 'P'). "Address" is a value of three characters expressed by the hexadecimal number.

However, about relays, data can be handled in every eight points as a byte register, or in every 16 points as a word register. In that case, the last character of < Address > (the third character) stands for the register type. For a register type, an alphabet from 'L', 'H', and 'W' can be specified.

5.1.1.1. Data memory types

Table5-1 Data memory types

Name	Capacity	Relay address	Byte register	Word register
I/O relay	256 points	X/Y000 - X/Y17F	X/Y00H/L - X/Y17H/L	X/Y00W - X/Y17W
Internal relay	256 points	R000 - R17F	R00H/L - R17H/L	R00W - R17W
Timer counter	96 points	T/C000 - T/C05F	T/C00H/L - T/C05H/L	T/C00W - T/C05W
Latch relay	32 points	L000 - L01F	L00H/L - L01H/L	L00W - L01W
Edge relay	64 points	E000 - E03F	E00H/L - E03H/L	E00W - E03W
Special auxiliary relay	240 points	A000 - A16F	A00H/L - A16H/L	A00W - A16W
Register	208 words			D000 - D05F D120 - D14F
Register	64 words			D060 - D11F
Special register	48 words			D150 - D17F
Timer counter set value	96 words			V000 - V05F
Timer counter present value	96 words			P000 - P05F

5.1.1.2. Function of data memory

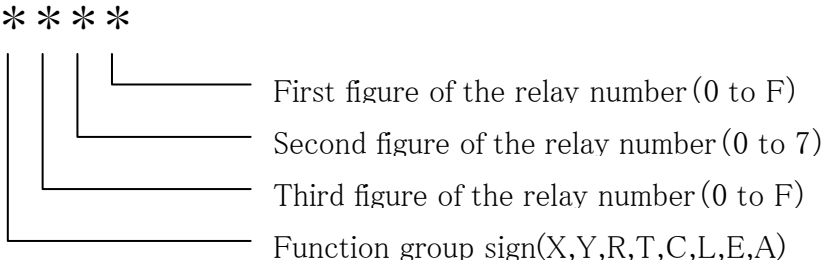
Table5-2 Function of data memory

Name		Group	Function
I/O relay	Input relay	X	A relay only for inputs where various inputs (such as photo couplers, keyboard, and DISPSW) are connected.
	Output relay	Y	A relay only for outputs where various output devices (such as relays, panel LED) are connected.
	Unrecognition relay	Z	This performs as an internal relay because being excluded from I/O processings of every scanning cycle. is possible to use it as an internal relay because it is excluded from the I/O processing at every scanning cycle.
Internal relay		R	This performs as a temporary data storage that has no need of output outside.
Timer		T	When the present value turns 0, the timer contact point is turned on.
Counter		C	When the present value turns 0, the counter contact point is turned on.
Latch relay		L	A latching relay of set/reset type. Once the “set” is turned ON, even if the “set” is turned OFF, the latching contact point keeps ON-state until the “reset” is turned ON.
Edge relay		E	This can be used as an edge contact point.
Register		D	Because this is the word length register (16-bit), it is impossible to specify for the byte register. If the data in between D060 and D11F has been changed, the change is written in the EEPROM. (this is retainable). Data between D150 and D17F are the special data register. Among special data registers, registers not used can be used as user data registers.
Timer counter Set value		V	Because this is the word length register (16-bit), it is impossible to specify for the byte register The area not used as a timer counter can be used as a data register.
Timer counter present value		P	Because this is the word length register (16-bit), it is impossible to specify for the byte register The timer counter can be executed, and the present value can be read (subtraction timer and subtraction counter)
Special auxiliary relay		A	This area is for the CPU writing, such as operation flag, scanning time, clock an others. Therefore, it is impossible to use it by the user program as destinations of the coil and the data register. This can be used as a source of the contact point or data register in the user program.

5.1.1.3. Relay address

The relay address is expressed by the combination of the relay numbers and a function group sign.

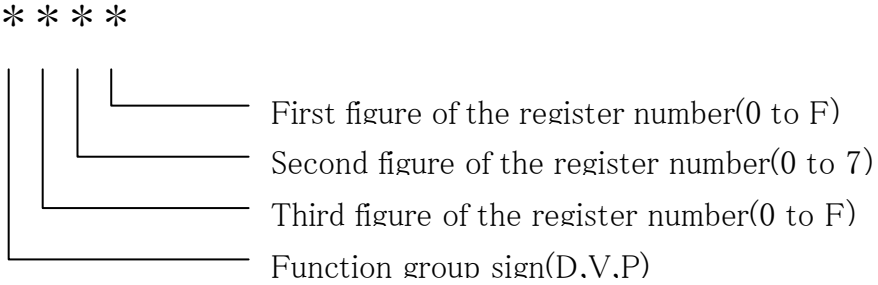
Whereas I/O relay addresses correspond to the actual relays, other relay addresses correspond to the virtual devices that do not exist physically. The relay address is assigned to one point (one bit) each.

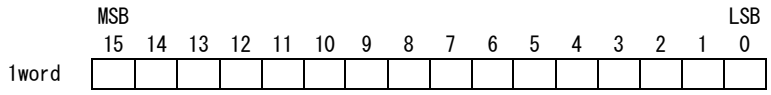


5.1.1.4. Data register address

The expression of the data register address is the same as that of the relay address.

The data register address is a unit of one word (16 bits), whereas the relay address is a unit of one point (one bit).

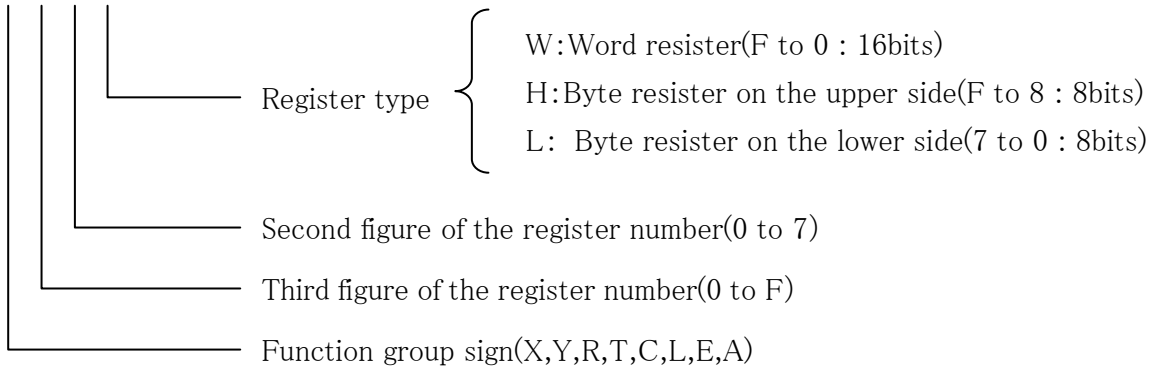




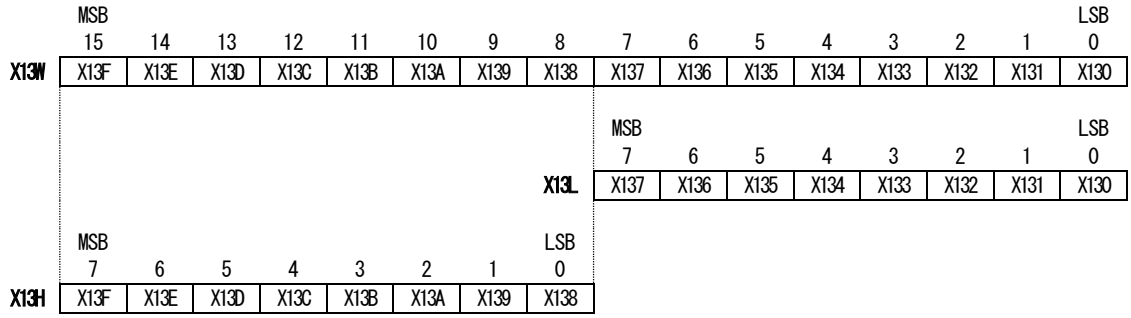
5.1.1.5. Byte/word register address in relay area

The relay area can handle data as a word register collectively in every eight points collectively in every 16 points as byte register. The part of "Relay No.1 digit" in the relay address becomes a register type in the address. The relay area can be treated as the byte register by handling every eight points, and is treated as the word register by handling every 16 points. The leftmost of the relay address expresses the register type.

＊＊＊＊



Example) When X130 to X13F are specified by the word register and byte register



5.1.2. Connection

To provide various services of the TCmini controller through RS-232C, connect the RS-232C connector on the top of the main module with the serial port (COM) of PC (host computer) through the straight cable. Note that the cable used is not a cross cable.

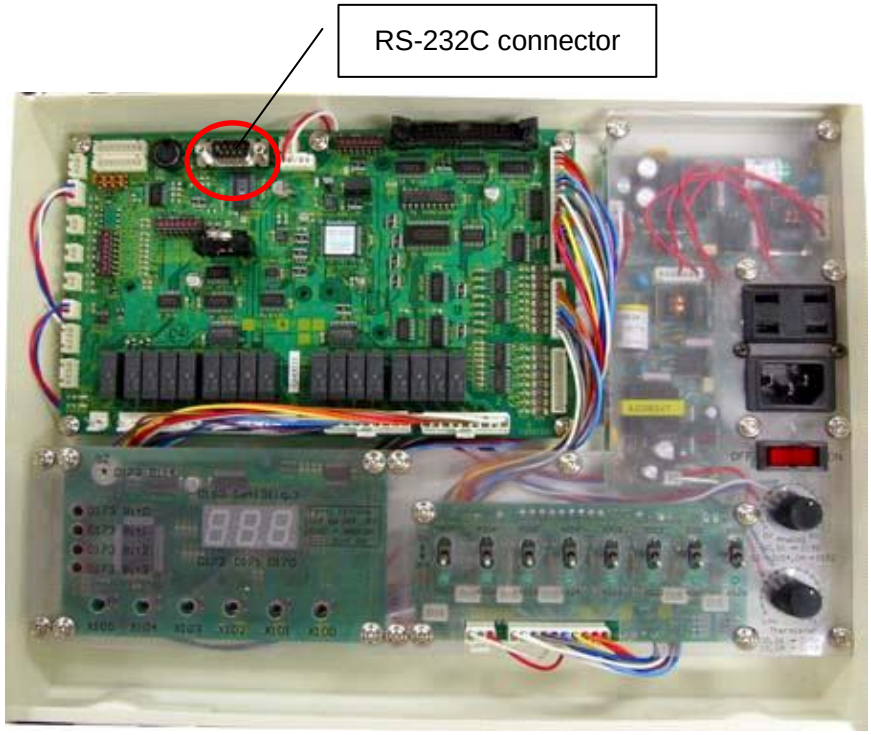


Figure5-1 TCmini controller

To communicate with the TCmini controller, a COM port on the PC side should set to the following specification.

Transmission rate (baud rate)	9600 bps/19200 bps/38400 bps
Number of data bits	Eight bits
Parity bit	None
Number of stop bits	Two bits

The setting on the TCmini side is basically unnecessary. The TCmini controller automatically determines the baud rate if it is in the range of 9600/19200/38400bps. The default setting is 19200 bps.

5.1.3. Overview of the communication protocol

The communication between PC (host computer side) and the TCmini controller (TCCUH * and TCCM * sides) must start with the data access from the PC side always, as the following figure shows.

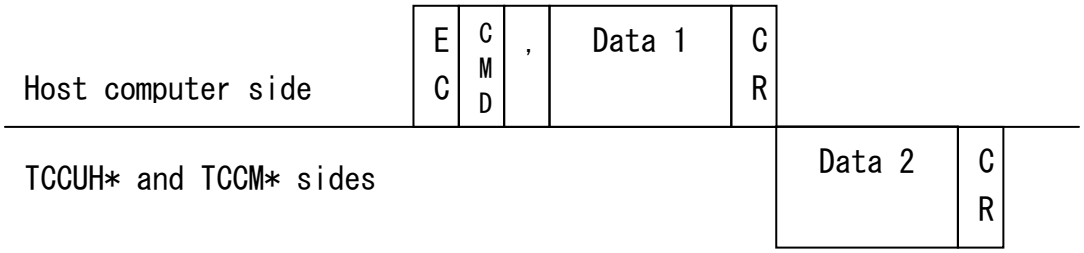


Figure5-2 Basic format

Table5-3 Communication protocol fixation data

Sign	Description
EC	Escape. ASCII code is 0x1B.
CMD	Specify a command number (“0” to ”32”) by using ASCII code from 0x30 (=”0”) to 39(=‘9') .
,	A comma that distinguishes the command number and data. ASCII code is 0x2C.
CR	Carriage return. ASCII code is 0x0D.
Data 1	Supplementation data of the command.
Data 2	Response data corresponing to the command.

When receiving command (CMD) and data (Data 1) from the PC with the said format, the TCmini controller interpretes the contents of received data, executes the processing required, and then sends the response data (Data 2). If the command transmitted to the controller is incorrect, an error message (such as “Parameter error”) will be returned.

All communications are executed by ASCII code.

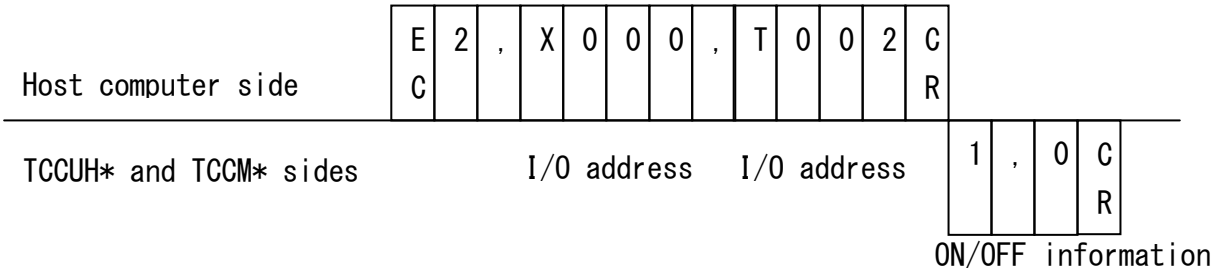
The following chapter will explains only a part of the command specifications in this communication format. For other commands, refer to the 'RS232C Host communication command manual' which comes with the TCmini controller, depending on your needs.

5.1.3.1. I/O reading out per one point

Read the ON/OFF information on the contact point up to 42 points.

<CMD> “2" (0x32)

The following example shows X000(ON) and T002(OFF).



ON/OFF information

0 : OFF

1 : ON

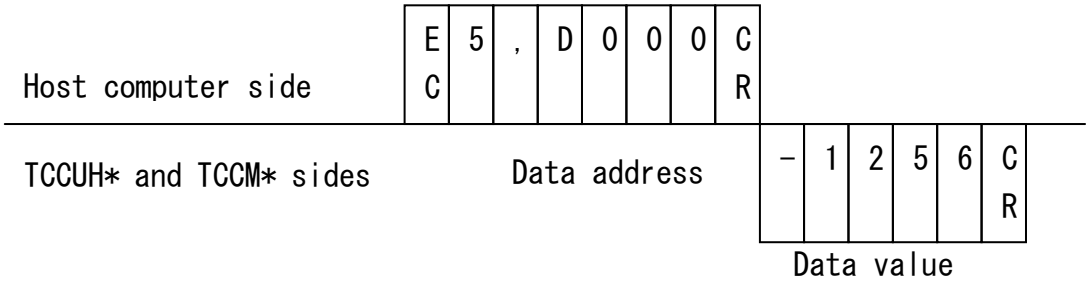
I/O address (up to 42 contact points)

- It must be specified by four digits.
- The group code (X,Y,R,L,S,T,C,E,A,G,H) must be wirtten in a capital letter.
- If any addresss that does not exist is specified, an error message “Parameter error" will be returned.

5.1.3.2. Data reading out per one word

Read the data (up to 32 words) at the same time.

<CMD> “5" (0x35)



Data address

- It must be specified by four digits.
- The group code (X,Y,R,L,S,T,C,E,A,G,H,D,P,V,B) and word specification W, byte specification H, and L must be written in a capital letter.
- If any addresss that does not exist is specified, an error message “Parameter error" will be returned.
- Do not mix the word data with the byte data at the address designation.

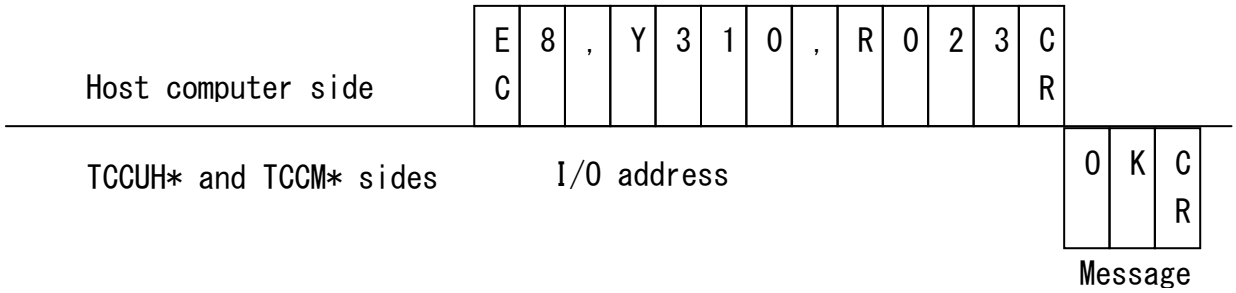
Data value

- Data returns the BIN data as a decimal number.
- Word data ranges from -32768 to 32767.
- Byte data ranges from 0 to 255.

5.1.3.3. I/O compulsion set

Compulsory set the relay contact points (up to 42 relay points) at the same time.

<CMD> “8" (0x38)



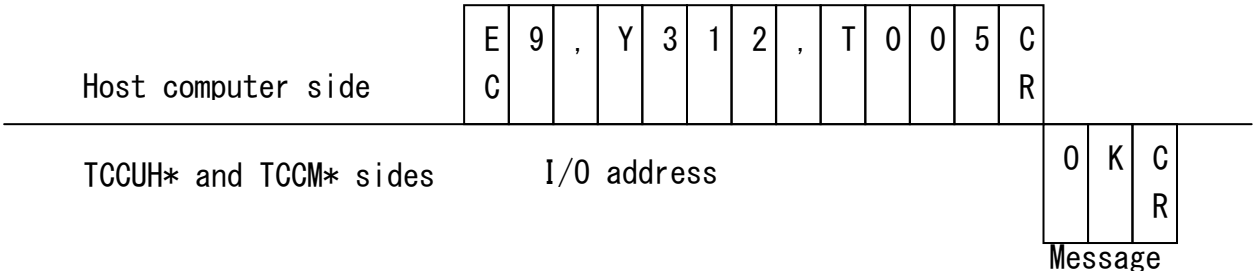
I/O address (up to 42 contact points)

- It must be specified by four digits.
- The group code (X,Y,R,L,S,T,C,E,A,G,H) must be written in a capital letter.
- If any addresss that does not exist is specified, an error message “Parameter error" will be returned.
- The area reset by the sequence program cannot be set.
- If the timer counter area is set, the present value is assumed to be 0 and the contact point is turned ON.

5.1.3.4. I/O compulsion reset

Compulsory reset the rely contact point (up to 42 relay points) at the same time.

<CMD> “9" (0x39)



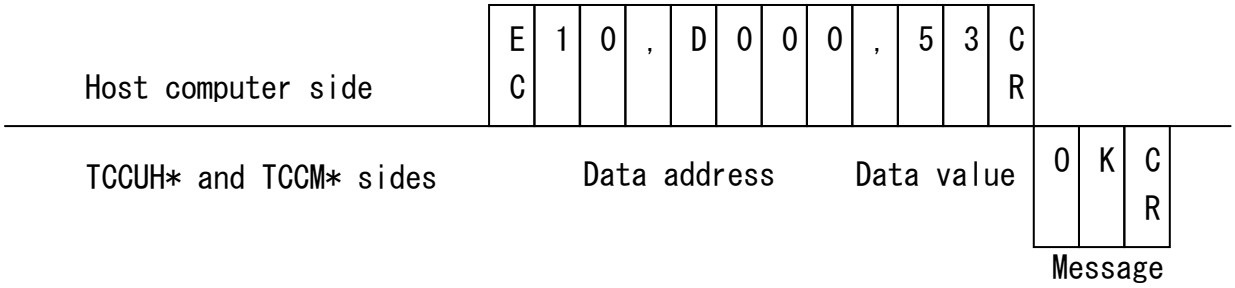
I/O address (up to 42 contact points)

- It must be specified by four digits.
- The group code (X,Y,R,L,S,T,C,E,A,G,H) must be written in a capital letter.
- If any addresss that does not exist is specified, an error message “Parameter error" will be returned.
- The area set by the sequence program cannot be reset.
- If the timer counter area is reset, the present value becomes equal to the set value, and the contact point is turned OFF.

5.1.3.5. Data change per one word

Change the data (up to 32 data) at the same time.

<CMD> “10" (0x31,0x30)



Data address

- It must be specified by four digits.
- The group code (X,Y,R,L,S,T,C,E,A,G,H,D,P,V,B) and word specification W, byte specification H, and L must be written in a capital letter.
- If any addresss that does not exist is specified, an error message “Parameter error" will be returned.
- Do not mix the word data with the byte data at the address designatin..

Data value

- Data returns the BIN data as a decimal number.
- Word data ranges from -32768 to 32767
- Byte data ranges from 0 to 255.

5.2. TCmini provider specification

In the event that the TCmini provider is created, the behavior as a CAO provider, which means the operation specification, need to be decided. This section explains the way to decide the TCmini provider’s specification, and the next section explains the way of provider implementation according to this specification.

5.2.1. Connection parameter specification

TCmini provider refers to the connection parameters for communication at CAOWorkspace::AddController. If the communication parameters are omitted, it will behave the same as when “COM:1:19200:N:8:2" is specified in the default setting. The following shows the argument specification of AddController.

```
AddController
(
 "CaoProv.SIT.TCmini ",   // Provider name.Fixed.
"< machine name >"        // Execution machine name of provider
"< controller name >"     // Arbitral character string
"< connection parameter >"// Parameter for TCmini controller connection
)
```

For details about the connection parameters, refer to "TCmini provider user's guide" .

5.2.2. User variable specification

In a general meaning of CAO, the AddVariable method of the CAOController class is a method to provide the access to variables. The TCmini provider wraps a communication command of TCminiα TC3-02 to the reference of the CAOVariable object' Value property created by this AddVariable method, without any modification. In other words, using this user variable allows access to any information that is not in the CAO's system variable. The following shows the specifications for arguments of the AddVariable

```
AddVariable
(
"< group code >< address >"
)
```

< group code >	:=	Relay (1 bit) X, Y, Z, R, E, L, T, C, A or Register (16 bits) D, P, and V
< address >	:=	Address of variable indicated by group code The address is specified by a hexadecimal three-digit number fixed. The relay variable can include an alphabet to indicate the register type (L, H, or W) in the last digit.[5]
(example 1)		"X100","Y020","D150","T12A"
(example 2)		"X10W" ・・・Range from X100 to X10F
(example 3)		"x10L" ・・・Range from X100 to X107[6]
(example 4)		"X10h" ・・・Range from X108 to X10F

For instance, the following shows the way to acquire the present value of X007 with VB.

```
Dim iRes As Integer
Set CAOVar = CAOCtrl.AddVariable("X007")
iRes = CAOVar.Value '
```

For details about connection parameters, refer to "TCmini provider user's guide".

5.2.3. System variable specification

The system variable is BSTR type character string specified by the AddVariable method of the CAOController class.

The TCmini provider implements tyis system variable based on the follwing specification.

[5] The range specification of 'W', 'L', and 'H' cannot be done in D, P, and V register that is the word size.
[6] There is no distinction between the capital letter and the small letter in the user variable name. Everything is processed as a capital letter.

Table5-4 System variable list mounted in TCmini provider

Variable	Data type	Description	Attribute	
			get	put
@CURRENT_TIME	VT_DATE	Present time of the controller [D125-D120]	✓	
@ERROR_DESCRIPTION	VT_BSTR	Alarm message Communication command: 1	✓	
@MAKER_NAME	VT_BSTR	“Toshiba Machine"	✓	
@TYPE	VT_BSTR	“TCmini"	✓	
@VERSION	VT_BSTR	Version of provider	✓	

5.3. Implementation of TCmini provider

5.3.1. Creating TCmini provider projects

Skeleton (model) project for the TCmini provider can be easily created by using the project wizard of the CAO provider. First of all, create a TCmini provider project according to the procedure of "2.3Creating a provider project".

As a new VC++ project for the TCmini provider is created as CAOPROV.dsw, specify this CAOPROV.dsw to execute VC++ .

In the project folder newly created, several files other than the CAOPROV.dsw exist. The table below shows the list of the file implemented this time.

Table5-5 File list to edit

No.	File name	Type
1	CaoProvController.h	C header file for the CCaoProvController class definition
2	CaoProvController.cpp	C++ source file for the CcaoProvController class implementation
3	CaoProvVariable.h	C header file for the CCaoProvVariable class definition
4	CaoProvVariable.cpp	C++ source file for the CCaoProvVariable class implementation

The following section will explains the way of provider implementation with concrete examples.

5.3.2. Addition of CSerial class

The communication from the TCmini provider to the TCmini controller is done through the serial port. This

example uses the CSerial class that have been explained in the preceding chapter for communication.

First of all, copy the two files of “Devich.cpp" and “Serial.cpp" to the project folder. [7]

Next, right-click in "Source Files" of "File View" as shown in Figure5-3, select "Adde the file to the folder (F)", and then add “Devich.cpp" and “Serial.cpp".

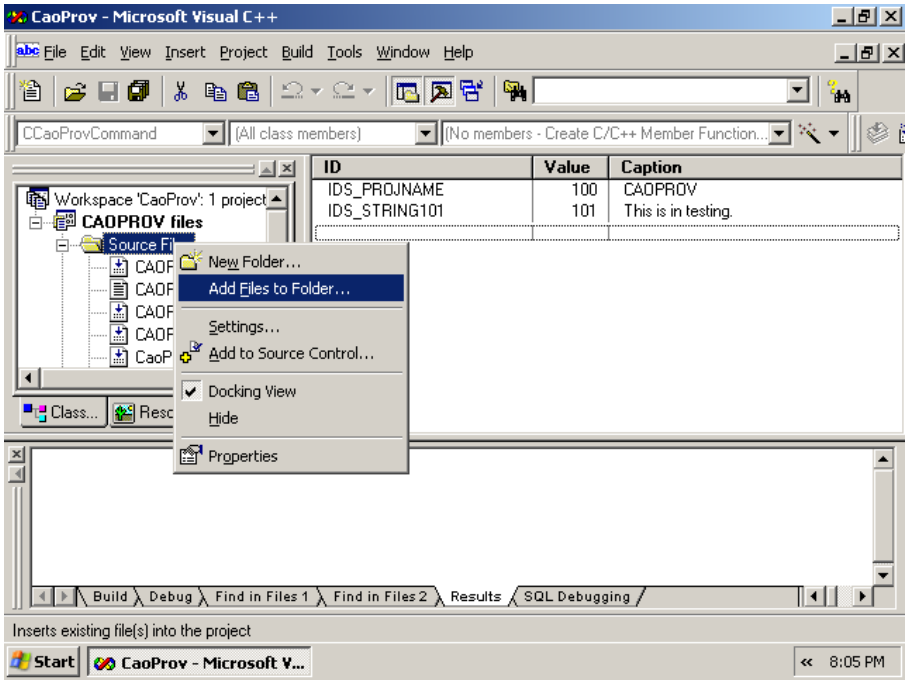


Figure5-3 Add a file to the the folder

Next, include a header file, so that the TCmini provider could use the CSerial class. Add the following descriptions to the StdAfx.h file.

List 5-1 StdAfx.h

```
private:
// ========== Describe company-specific additional part in the following. ==========
//↓[1]------------------------------------------------------------------------↓
#include "Serial.h"
//↑[1]------------------------------------------------------------------------↑
```

5.3.3. Implementation of CCaoProvController class

5.3.3.1. Override necessary methods

As a preparation of the CcaoProvController class implementation, at first, override the necessary methods

[7] Please copy the one of ORiN2¥CAO¥Include¥ onto these files.

according to the following procedures.

(1) In the methods written in the header files of the CCaoProvController class, enable the following methods by deleting the "//" (disable the comment-out).

1. HRESULT FinalInitialize()
2. HRESULT **FinalConnect**()
3. HRESULT FinalDisconnect()
4. HRESULT FinalGetVariableNames(/*[in]*/ BSTR bstrOption, /*[out, retval]*/ VARIANT *pVal)

The example for CCaoProvController::FinalInitialize() is shown as follows.

```
//     HRESULT FinalInitialize();          // Before removing the comment
                  ↓
       HRESULT FinalInitialize();          // After removing the comment
```

(2) Comment-out the areas where the said methods are implemented (CaoProvController.cpp).

The example for CCaoProvController::FinalInitialize() is shown as follows.

```
/*                                                    // Before removing the comment
HRESULT CCaoProvController::FinalInitialize()
{
     // Execute intialization and other processings which are common to CaoProvController object.
       return E_NOTIMPL; // Set a return value to "S_OK" if you want to implement this class.
 }
*/
       ↓
HRESULT CCaoProvController::FinalInitialize()       // After removing the comment
{
      //Execute intialization and other processings which are common to CaoProvController object.
       return E_NOTIMPL;
}
```

5.3.3.2. Addition of the necessary member

Next, add the member variables necessary for the CCaoProvController class.

The following three member variables are added.

m_pSerial	:	Pointer to object that communicates with COM port
m_bConnected	:	Connection flag
		TRUE Connected state
		FALSE Unconnected state

List 5-2 CaoProvController.h

```
private:
// ===== Manufacturer-specific additional functions should be written below. ======
//↓[2]-------------------------------------------------------------------↓
 CDevice* m_pSerial;      // Pointer to device object
 BOOL     m_bConnected;   // Connection flag
```

```
//↑[2]----------------------------------------------------------------------↑
```

5.3.3.3. Implementation of FinalInitialize()

Next, implement the FinalInitialize method of the CaoProvController class.

This method is called when the CaoProvController object is created. The pointer to the device object is initialized in this method.

List 5-3 CaoProvController.cpp - FinalInitialize()

```
/**    Initialization
 *
 *  This function is called first after the CaoProvController object creation.
 *
 *     @retval   HRESULT
 *
 */

HRESULT CCaoProvController::FinalInitialize()
{
// Execute intialization and other processings which are common to CaoProvController object.
//↓[3]----------------------------------------------------------------------↓
 m_pSerial = NULL;
 return S_OK; // Set a return value to "S_OK" if you want to implement this class.
//↑[3]----------------------------------------------------------------------↑
}
```

5.3.3.4. Adding the ParseParameter method

Next, add a private parameter that analyzes communication parameters passed at CaoWorkspace::AddController. The specification is shown as below.

```
HRESULT ParseSerialParameter(
        BSTR bstrParam,                     // Connection parameter character string
        PARAM_SERIAL *pParam                // Storage destination of the analysis.
);
```

The ParseParameter() function receives the connection parameters (bstrParam) by BSTR-type, opens the COM ports, and substitutes a value necessary to open for the structure. First of all, add the structure to store the analysis.

```
/** Parameter structure of Connect() argument */
struct PARAM_SERIAL {
       DWORD dwPortNo;    /* Number of COM port */
       DWORD dwBaudRate;  /* speed (bps) */
       DWORD dwParity;    /* Parity */
       DWORD dwDataBits;  /* Number of data bits */
       DWORD dwStopBits;  /* Number of stop bits */
       DWORD dwFlow;      /* Flow control */
};
```

Add this structure definition to the StdAfx.h header file.

List 5-4 StdAfx.h – PARAM_SERIAL

```
// ====== Manufacturer-specific additional items should be written below. ==========
//↓[1]------------------------------------------------------------------------↓
#include "Serial.h"
//↑[1]------------------------------------------------------------------------↑
#define CAOP_TIMER_INTERVAL 0  //  0: Off
          //n: Issue an OnTimer event every n-msec. Positive integer of LONG_MAX or less.

//↓[4]------------------------------------------------------------------------↓
/** Parameter structure of Connect() argument */
struct PARAM_SERIAL {
       /** Number of COM port      : COM1(1) to  */
       DWORD dwPortNo;
       /**
        * Speed (bps)     : 4800bps(4800),  -
        * Macro                   : CBR_4800,CBR_9600,...
        */
       DWORD dwBaudRate;
       /**
        * Parity: 0 to 4 = None, Odd, Even, Mark, Space
        * Macro          : NOPARITY,ODDPARITY,EVENPARITY,MARKPARITY,SPACEPARITY
        */
       DWORD dwParity;
       /** Number of data bits     : 4bits(4) to 8bits(8) */
       DWORD dwDataBits;
       /**
        * Number of stop bits      : 0 to 2 = 1,1.5,2 bit(s)
        * Macro: ONESTOPBIT, ONE5STOPBITS, and TWOSTOPBITS
        */
       DWORD dwStopBits;
       /**
        * Flow control: 0 to 3
        */
       DWORD dwFlow;
};

#define FLOW_XINOUT               1
#define FLOW_HARDWARE             2
//↑[4]------------------------------------------------------------------------↑
```

Next, add the prototype declaration of the ParseParameter function to header file (CaoProvController.h) of the CCaoProvController class. Changes which have been done for the CCaoProvController.h file so far is shown below.

List 5-5 CaoProvController.h

```
/** @file CaoProvController.h
 *
 *  Declaration of @brief CCaoProvController
 *
 *  @version    1.0
```

```
 *     @date             2003/8/8
 *     @author           DENSO WAVE
 *
 */

#ifndef __CAOPROVCONTROLLER_H_
#define __CAOPROVCONTROLLER_H_

#Include "Resource.h" // main symbol
#include "CAOPROVCP.h"
#include "CaoProvCommand.h"
#include "CaoProvVariable.h"
#include "CaoProvRobot.h"
#include "CaoProvFile.h"
#include "CaoProvTask.h"
#include "CaoProvExtension.h"
#include "CaoProvControllerImpl.h"

/**    Controller class of CAO provider
 *
 *     COM class with dual interfaces. For details, refer to ICaoProvController. <br>
 *     In CAO provider, COM client is allowed to create an instance only in this class.<br>
 *     The interface is implemented in the ICaoProvControllerImpl class.
 *
 */
class ATL_NO_VTABLE CCaoProvController :
 public ICaoProvControllerImpl<CCaoProvController, CCaoProvExtension, CCaoProvFile,
CCaoProvRobot, CCaoProvTask, CCaoProvVariable, CCaoProvCommand, CCaoProvMessage>
{
protected:
 HRESULT FinalInitialize();
// void FinalTerminate();

 HRESULT FinalConnect();
 HRESULT FinalDisconnect();
// HRESULT FinalExecute(/*[in]*/ VARIANT Command, /*[out, retval]*/ VARIANT *pVal);

// HRESULT FinalGetAttribute(/*[out, retval]*/ long *pVal);
// HRESULT FinalGetHelp(/*[out, retval]*/ BSTR *pVal);
// HRESULT FinalGetCommandNames(/*[in]*/ BSTR bstrOption, /*[out, retval]*/ VARIANT *pVal);
// HRESULT FinalGetExtensionNames(/*[in]*/ BSTR bstrOption, /*[out, retval]*/ VARIANT *pVal);
// HRESULT FinalGetFileNames(/*[in]*/ BSTR bstrOption, /*[out, retval]*/ VARIANT *pVal);
// HRESULT FinalGetRobotNames(/*[in]*/ BSTR bstrOption, /*[out, retval]*/ VARIANT *pVal);
// HRESULT FinalGetTaskNames(/*[in]*/ BSTR bstrOption, /*[out, retval]*/ VARIANT *pVal);
 HRESULT FinalGetVariableNames(/*[in]*/ BSTR bstrOption, /*[out, retval]*/ VARIANT *pVal);

public:
#if CAOP_TIMER_INTERVAL > 0
 void OnTimer();
#endif

// ========== Manufacturer-specific additional functions should be written below. ==========

private:
// ========== Manufacturer-specific additional functions should be written below.==========
//↓[5]-------------------------------------------------------------------------↓
 HRESULT ParseSerialParameter(
 BSTR bstrParam,          // Connection parameter character string
 PARAM_SERIAL *pParam     // Sstorage destination of the analysis.
 );
//↑[5]-------------------------------------------------------------------------↑
//↓[2]-------------------------------------------------------------------------↓
 CSerial *m_pSerial;              // Pointer to device object
 DWORD m_dwMode;                  // Operational mode
 BOOL  m_bConnected;              // Connection flag
//↑[2]-------------------------------------------------------------------------↑
```

```
};

#endif //__CAOPROVCONTROLLER_H_
```

The processing of the ParseParameter() function is as follows.

List 5-6 CaoProvController.cpp - ParseParameter()

```
// ======== Manufacturer-specific additional functions should be written below. ==========
//↓[6]--------------------------------------------------------------------------↓
/**    Connection parameter analysis
 *
 * Analyze the connection parameter to prepare for the connection with the controller.
 *
 *  [com:<COM Port>[:<BaudRate>[:<Parity>:<DataBits>:<StopBits>]]]
 *
 *     ["-"]"means omittable.                                  Default
 *     <COM Port>         := 1  -                              : 1
 *     <BaudRate>         := 4800,9600,19200, ...              : 19200
 *     <Parity>           := N,O,E,M,S                         : N
 *     <DataBits>         := 4  -  8                           : 8
 *     <DataBits>         := 1, 1.5, 2                         : 2
 *
 *     @param   bstrParam:        [in] Connection parameter character string
 *     @param   pParam    :        [out]Storage destination of the analysis
 *     @retval  HRESULT
 *
 */
HRESULT CCaoProvController::ParseSerialParameter(BSTR bstrParam, PARAM_SERIAL *pParam )
{
       DWORD *pdwPortNo;  // Number of COM port       : COM1(1)  -
       DWORD *pdwBaudRate;          // Speed (bps)     : 4800bps(4800),  -
                          // Macro: CBR_4800 and CBR_9600.
       DWORD *pdwParity;  // Parity:  0 -4 =  None, Odd, Even, Mark, and Space
                     // Macro: NOPARITY, ODDPARITY, EVENPARITY, MARKPARITY, and SPACEPARITY
       DWORD *pdwDataBits;          // Number of data bits      : 4bits(4)  -  8bits(8)
       DWORD *pdwStopBits;          // Number of stop bits      : 0  - 2 = 1,1.5,2 bit(s)
                          // Macro: ONESTOPBIT, ONE5STOPBITS, and TWOSTOPBITS
       DWORD *pdwFlow;              // Flow control             : 0  - 3

       pdwPortNo = &(pParam->dwPortNo);
       pdwBaudRate = &(pParam->dwBaudRate);
       pdwParity = &(pParam->dwParity);
       pdwDataBits = &(pParam->dwDataBits);
       pdwStopBits = &(pParam->dwStopBits);
       pdwFlow = &(pParam->dwFlow);

       // Substitute the default value.
       *pdwPortNo        = 1;
       *pdwDataBits = 8;
       *pdwBaudRate = CBR_19200;
       *pdwParity = NOPARITY;
       *pdwStopBits = TWOSTOPBITS;
       *pdwFlow = 0;

       int nlen;
       if (!bstrParam || (nlen = wcslen( bstrParam )) <= 0) {
                return S_OK; // Omit all
       }
```

```
	// Whether the first four character is “com.” or not
	if (_wcsnicmp(L"com:", bstrParam, 4) != 0) {
		return E_INVALIDARG; // Invalid argument.
	}

	WCHAR *pwcParam = bstrParam;

	//Set a setting value by counting the number of L':' with a specified parameter format.
	// ========================================================================
	int nTokens = 0;
	for (int i=0; i<nlen; i++) {
		if (pwcParam[i] == L':') {
		    nTokens++;
		}
	}

int n;
int iPortNo = 1, iBaudRate = CBR_19200, iDataBits = 8, iFlow = 0;	// Default value
float fStopBits = 2.0;
WCHAR wcParity = L'N';
WCHAR wcDumy;

pwcParam = &bstrParam[4]; // Position from the next of "com:"
switch (nTokens) {
	case 6: // com:<COM Port>:<BaudRate>:<Parity>:<DataBits>:<StopBits>:<Flow>
	    n = swscanf(pwcParam, L"%d:%d:%c:%d:%f:%d%c", &iPortNo, &iBaudRate, &wcParity,
		  &iDataBits, &fStopBits, &iFlow, &wcDumy);
	    break;
	case 5: // com:<COM Port>:<BaudRate>:<Parity>:<DataBits>:<StopBits>
	    n = swscanf(pwcParam, L"%d:%d:%c:%d:%f%c", &iPortNo, &iBaudRate, &wcParity,
		  &iDataBits, &fStopBits, &wcDumy);
	    break;
	case 2: // com:<COM Port>:<BaudRate>
	n = swscanf(pwcParam, L"%d:%d%c", &iPortNo, &iBaudRate, &wcDumy);
	    break;
	case 1: // com:<COM Port>
		n = swscanf(pwcParam, L"%d%c", &iPortNo, &wcDumy);
		break;
	default:
		return E_INVALIDARG; // Invalid argument.
		break;
	}
	//Check if the number of correctly converted item is equal or more than the number of token.
	if (n != nTokens) {
		return E_INVALIDARG; // Invalid argument.
	}

	// Convert items which need to be converted.
	// ========================================================================
	switch(wcParity){ //parity
	case L'N':
	case L'n':
		*pdwParity = NOPARITY;
		break;
	case L'O':
	case L'o':
		*pdwParity = ODDPARITY;
		break;
	case L'E':
	case L'e':
		*pdwParity = EVENPARITY;
		break;
	case L'M':
	case L'm':
		*pdwParity = MARKPARITY;
		break;
```

```
        case L'S':
        case L's':
                *pdwParity = SPACEPARITY;
                break;
        default:
                return E_INVALIDARG; // Invalid argument.
                break;
        }

        int iStopBitsx10 = (int)(fStopBits * 10.0);
        switch (iStopBitsx10) { // stop bit
        case 10:
                *pdwStopBits = ONESTOPBIT;
                break;
        case 15:
                *pdwStopBits = ONE5STOPBITS;
                break;
        case 20:
                *pdwStopBits = TWOSTOPBITS;
                break;
        default:
                return E_INVALIDARG; // Invalid argument..
                break;
        }

        // ========================================================================
        // Other checks are not done.
        //-> Because it becomes an error at actual connection it if it is illegal.
        *pdwPortNo                  = (DWORD)iPortNo;
        *pdwDataBits        = (DWORD)iDataBits;
        *pdwBaudRate        = (DWORD)iBaudRate;
        *pdwFlow            = (DWORD)iFlow;

        return S_OK;
}
//↑[6]-------------------------------------------------------------------------↑
```

5.3.3.5. Implementation of FinalConnect()

The next step is implementation of the FinalConnect() method. This method is called when CaoWorkspace::AddController is executed.

In this processing, obtain the value of “Conn" option with the GetOptionValue function first. “Conn” option has been specified by the fourth argument (connection parameter of m_bstrPara) of AddController method. Then, pass the obtained connection parameter to the ParseParameter() function that have been defined in the previous section to obtain the communication parameters. Next, open the COM port (connect) and obtain the default value of the DCB structure (dcb) (GetState). Change the necessary value in the DCB structure (dcb) to set the communication device (SetState).

List 5-7 CaoProvController.cpp – FinalConnect()

```
/**    Processing for starting provider connection
 *
 * This function is called first in the provider.
 *
 *     @retval  HRESULT
```

```
 *
 */

HRESULT CCaoProvController::FinalConnect()
{
//↓[7]-------------------------------------------------------------------------------↓
	HRESULT hr;

	// Obtain the connection parameter
	CComBSTR bstrPara;
	CComVariant vntOptVal;
	hr = GetOptionValue(m_bstrOption, L"Conn", VT_BSTR, &vntOptVal);
	if (FAILED(hr) || vntOptVal.vt != VT_BSTR) {
		return E_INVALIDARG;
	}
	bstrPara = vntOptVal.bstrVal;

	// Obtain the setting information by analyzing the connection parameters.
	PARAM_SERIAL SerialPara;
	m_pSerial = new CSerial;
	hr = ParseSerialParameter(bstrPara, &SerialPara);
	if (FAILED(hr)) {
		return E_INVALIDARG;
	}

	CHAR cHeader[] = "¥0";
	CHAR cTerm[] = "¥r";
	DWORD dwMode = ST_MODE_TEXT | ST_MODE_WBSTOWCS;
	PVOID pInitArgs[] = {cHeader, cTerm, &dwMode};
	hr = m_pSerial->Initialize(pInitArgs);
	if (SUCCEEDED(hr)) {
		PVOID pConArgs[] = {&SerialPara.dwPortNo};
		hr = m_pSerial->Connect(pConArgs);

		if (SUCCEEDED(hr)) {
		    DCB dcb;
		    hr = ((CSerial*)m_pSerial)->GetState(&dcb);

		    if (SUCCEEDED(hr)) {
			dcb.BaudRate = SerialPara.dwBaudRate;		// Speed
			dcb.ByteSize = (BYTE)SerialPara.dwDataBits;	// Data length
			dcb.Parity = (BYTE)SerialPara.dwParity;		// Parity
			dcb.StopBits = (BYTE)SerialPara.dwStopBits;	// Stop bit
			if (SerialPara.dwFlow && FLOW_XINOUT) {
				dcb.fOutX = true;		// Send XON/XOFF
				dcb.fInX = true;		// Receive XON/XOFF
			}
			else {
				dcb.fOutX = false;		// not send XON/XOFF
				dcb.fInX = false;		// not receive XON/XOFF
			}

			if (SerialPara.dwFlow && FLOW_HARDWARE) {
				dcb.fOutxCtsFlow = true;	// Enable the CTS signal
				dcb.fOutxDsrFlow = false;	// Disable the DSR signal
				dcb.fRtsControl= RTS_CONTROL_HANDSHAKE;//Ctrl RTS signal
			dcb.fDtrControl = DTR_CONTROL_ENABLE;// Turn ON the DTR signal
			}
			else {
				dcb.fOutxCtsFlow = false;	// Disable the CTS signal.
				dcb.fOutxDsrFlow = false;	// Disable the DSR signal.
				dcb.fRtsControl = RTS_CONTROL_ENABLE;//TurnON RTS signal
				dcb.fDtrControl = DTR_CONTROL_ENABLE;//TurnON DTR signal
			}
			dcb.fDsrSensitivity = false;
			hr = ((CSerial*)m_pSerial)->SetState(&dcb);
```

```
                }
        }
 }
 if (FAILED(hr)) {
        FinalDisconnect();
        return hr;
 }

// Setting of timeout period
 hr = GetOptionValue(m_bstrOption, L"TimeOut", VT_I4, &vntOptVal);
 if (FAILED(hr)) {
        return E_INVALIDARG;
 }
 if (vntOptVal.vt == VT_I4) {
        hr = m_pSerial->SetTimeOut((DWORD)vntOptVal.lVal);
 }

 return hr;
//↑[7]-------------------------------------------------------------------↑
}
```

5.3.3.6. Implementation of FinalDisconnect()

The next step is impmenentation of the FinalDisconnect(). This method is called at CaoControllers::Remove. This method executes the disconnection of serial connection and the release of objects.

List 5-8 CaoProvController.cpp – FinalDisconnect()

```
/**    Provider disconnection processing
 *
 *     @retval  HRESULT
 *
 */
HRESULT CCaoProvController::FinalDisconnect()
{
        // The disconnection of the provider is performed here.
//↓[8]-------------------------------------------------------------------↓
 //Disconnection
  m_bConnected = FALSE;
  if (m_pSerial) {
        m_pSerial->Disconnect();
        delete m_pSerial;
        m_pSerial = NULL;
  }
//↑[8]-------------------------------------------------------------------↑
 return S_OK; // Set a return value to “S_OK” if you want to implement this class.
}
```

5.3.3.7. Implementation of GetSerial()

To communicate with TCmini from the CaoProvVariable object, a pointer to the device object is required. This is because the connection to the COM port is executed in the CCaoProvController class. In this section, add a method to obtain a serial connection object so as to perform the COM communication from CaoVariable

class.

List 5-9 CaoProvController.cpp - GetSerial()

```
//↓[9]-------------------------------------------------------------------↓
/**    Obtain a serial connection object
 *
 *     @param   ppSerial :        [out] sereal connection object
 *     @retval  HRESULT
 *
 */
HRESULT CCaoProvController::GetSerial(CSerial **ppSerial)
{
       *ppSerial = m_pSerial;
       return S_OK;
}
//↑[9]-------------------------------------------------------------------↑
```

5.3.3.8. Implementation of FinalGetVariableNames()

As shown in Table5-4, TCmini is set so that the following five system variables are available. First of all, system variable names are defined as follows.

```
// System variable
#define CS_CURRENT_TIME           L"@CURRENT_TIME"
#define CS_ERROR_DESCRIPTION      L"@ERROR_DESCRIPTION"
#define CI_MAKER_NAME             L"@MAKER_NAME"
#define CI_TYPE                   L"@TYPE"
#define CI_VERSION                L"@VERSION"
```

Amoung system variables, the three variables of “CI_MAKER_NAME", “CI_TYPE", and “CI_VERSION" always return the constants. The value is defined as follows.

```
#define TC_PROV_MAKER             L"Toshiba Machine"
#define TC_PROV_TYPE              L"TCmini"
#define TC_PROV_VER               L"1.0.0"
```

The ‘L' ahead of the returned BSTR-type character string is a VC++ specific extension. This indicates that the succeeding character strings is handled as wide character strings (UNICODE).

Add the said definition to StdAfx.h.

List 5-10 StdAfx.h - Macro definition

```
// ====== Manufacturer-specific additional items should be written below.==========
//↓[1]-------------------------------------------------------------------↑
#include "Serial.h"
//↑[1]-------------------------------------------------------------------↑

#define CAOP_TIMER_INTERVAL 0         //  0: Off, n: Issue OnTimer event every “n” msec.
```

```
                                        // Positive integer of LONG_MAX orless.

//↓[10]-------------------------------------------------------------------------↑
#define TC_PROV_MAKER             L"Toshiba Machine"
#define TC_PROV_TYPE              L"TCmini"
#define TC_PROV_VER               L"1.0.0"

// System variable
#define VAR_CURRENT_TIME          0
#define VAR_CURRENT_TIME$         L"@CURRENT_TIME"
#define VAR_ERROR_DESCRIPTION     1
#define VAR_ERROR_DESCRIPTION$    L"@ERROR_DESCRIPTION"
#define VAR_MAKER_NAME            2
#define VAR_MAKER_NAME$           L"@MAKER_NAME"
#define VAR_TYPE                  3
#define VAR_TYPE$                 L"@TYPE"
#define VAR_VERSION               4
#define VAR_VERSION$              L"@VERSION"
//↑[10]-------------------------------------------------------------------------↑
```

FinalGetVaribaleNames() method that obtains the list of the defined system variable names is implemented as follows.

List 5-11 CaoProvController.cpp - FinalGetVariableNames()

```
/**    Obtain the variable name list
 *
 *     @param   bstrOption        :        [in] option
 *     @param   pVal              :        [out] variable name list
 *     @retval  HRESULT
 *
 */

HRESULT CCaoProvController::FinalGetVariableNames(BSTR bstrOption, VARIANT *pVal)
{
       // Create a variable name list as SAFEARRAY of the character string.
       // Store the pointer in VARIANT(pVal->vt = VT_BSTR|VT_ARRAY,pVal->parray = xxx).
//↓[11]-------------------------------------------------------------------------↓
       pVal->vt = VT_ARRAY | VT_BSTR;
       SAFEARRAY* psa;
       SAFEARRAYBOUND bounds = {5, 0};
       psa = SafeArrayCreate(VT_BSTR, 1, &bounds);

       CComBSTR* bstrArray;
       SafeArrayAccessData(psa, (void**)&bstrArray);

       bstrArray[0] = VAR_CURRENT_TIME$;
       bstrArray[1] = VAR_ERROR_DESCRIPTION$;
       bstrArray[2] = VAR_MAKER_NAME$;
       bstrArray[3] = VAR_TYPE$;
       bstrArray[4] = VAR_VERSION$;

       SafeArrayUnaccessData(psa);
       pVal->parray = psa;

       return S_OK;
//↑[11]-------------------------------------------------------------------------↑
```

```
        return S_OK;
}
```

5.3.4. Implementation of CCaoProvVariable class

As a preparation of the CcaoProvVariable class implementation, at first, override the necessary methods (FinalInitialize method, FinalGetValue method, and FinalPutValue method), according to the following procedures.

(1) In the methods written in the header file (CaoProvVariable.h) of the CCaoProvVariable class, enable the following methods (disable the comment-out).

1. HRESULT **FinalInitialize**(PVOID pObj);
2. HRESULT **FinalGetValue**(/*[out, retval]*/ VARIANT *pVal);
3. HRESULT **FinalPutValue**(/*[in]*/ VARIANT newVal);

(2) Comment-out the area where the said methods are implemented (CaoProvVariable.cpp).

5.3.4.1. Implementation of FinalInitialize()

First of all, implement the FinalInitialize() method.

This method is called when the CCaoProvController::AddVariable() method is executed, and initializes the CCaoVariable class.

Add a pointer to the device object (m_pSerial) into the CCaoProvVariable class member so as to establish the serial communication from the CaoProvVariable class. To initialize m_pSerial variable, use CcaoProvVariable::FinalInitialize() method that is called when the CaoProvVariable object is created.

```
private:
Cdevice* m_pSerial;       // Pointer to device object
```

In CCaoProvController::AddVariable(), variable names are specified for arguments. Analyzing user variable names passed from clients and converting the variables to numbers at the CaoProvVariable object creation will simplify the following processings. This information is maintained in the class.

The following information about user variable names are required.

・ Relay (BIT), register (WORD) or other data type information

The following expression defines these data types.

```
// Data type of TCmini (Relay: BIT, Register: WORD)
typedef enum { VAR_TYPE_UNKNOWN, VAR_TYPE_BIT, VAR_TYPE_WORD } VAR_DATA_TYPE;
```

・ Copy of variable name converted into Capital letter

Because original user variable name (m_bstrName) could be the combination of capital and small

letters, all variable names converted into the capital letter are defined.

・ The identification data of system variable

List 5-12 CaoProvVariable.h

```
/** @file CaoProvVariable.h
 *
 * Declare @brief CcaoProvVariable
 *
 *  @version    1.0
 *     @date              2003/8/8
 *     @author            DENSO WAVE
 *
 */

#ifndef __CAOPROVVARIABLE_H_
#define __CAOPROVVARIABLE_H_

#include "CaoProvVariableImpl.h"
#include "Serial.h"

class CCaoProvController; // Forward declaration
                         // For RaoProvController.h ->#include "RaoProvVariable.h".
//↓[12] ------------------------------------------------------------------------↓
class CCaoProvController;          // Forward declaration
#define CMD_EC 0x1B                // The first code of command
#define CMD_CR 0x0D                // The last code of command
                                   // The command of TCmini is as follows.
                                   //       <EC> + <CMD> + ',' + <data> + <CR>
                                   // CMD = ASCII character from 0 to 31
#define CMD_ADD_LEN 4              // Length of address
#define VAR_COMMAND_MAX 256        // The maximum character count of the command

// Data type of TCmini (Relay: BIT, Register: WORD)
typedef enum { VAR_TYPE_UNKNOWN, VAR_TYPE_BIT, VAR_TYPE_WORD } VAR_DATA_TYPE;
//↑[12]------------------------------------------------------------------------↑

/** Variable class of TCmini provider
 *
 * COM class with dual interfaces. For details, refer toICaoProvVariable. <br>
 *  CCaoProvController class, CCaoProvExtension class, CCaoProvFile class < br >
*  Class called from CCaoProvRobot class or CCaoProvTask class. <br>
 * Impossible to create instance from the outside. <br>
 * The interface is implemented in the ICaoProvVariableImpl class. <br>
 * Create a communication command, and sent and received.
 *
 */
class ATL_NO_VTABLE CCaoProvVariable :
        public ICaoProvVariableImpl<CCaoProvVariable>
{
protected:
       HRESULT FinalInitialize(PVOID pObj);
//     void FinalTerminate();

//     HRESULT FinalGetAttribute( /*[out, retval ]*/ long *pVal );
//     HRESULT FinalGetDateTime(/*[out, retval]*/ VARIANT *pVal);
//     HRESULT FinalGetHelp(/*[out, retval]*/ BSTR *pVal);
       HRESULT FinalGetValue(/*[out, retval]*/ VARIANT *pVal);
       HRESULT FinalPutValue(/*[in]*/ VARIANT newVal);
//     HRESULT FinalGetID(VARIANT *pVal);
//     HRESULT FinalPutID(VARIANT newpVal);
//     HRESULT FinalGetMicrosecond(long *pVal);
```

```
// ======== Manufacturer-specific additional functions should be written below.==========

private:
// ======== Manufacturer-specific additional functions should be written below.==========
//↓[13]-------------------------------------------------------------------------↓
      CDevice *m_pSerial;                  // Pointer to device object
      VAR_DATA_TYPE m_usrDataType;         // Variable that stores data type
      BSTR m_bstrUpperName;                // Variable name made of capital letter only
      DWORD m_dwSysID;                     // System variable ID
//↑[13]-------------------------------------------------------------------------↑
};

#endif //__CAOPROVTASKVARIABLE_H_
```

The following explains the analytical processing of user variable names. This processing is done with the CcaoProvVariable::FinalInitialize() since it needs to be executed only one time at the beginning.

This method firstly distinguishes whether the variable provided as an argument is a system variable or a user variable.

If it is a system variable, check if the variable name is an appropreate name. If the name is confirmed as appropriate, assign a system ID to m_dwSysID.

If it is a user variable, copy the variable name that is converted to the chapital letters into m_bstrUpperName. And then, by referring the first letter of obtained variable, classify the data into three types; relay type (X,Y,Z,R,E,L,T,C,A), register type (D,P,V), or other types. The succeeding processing will be branched depending on the data type. If it is judged as a register type, it will be m_usrDataType = VAR_TYPE_WORD

If it is judged as a relay type, it will be further refferred to the last letter (the fourth letter). If the last letter is L, H, or W, the user data type is byte/word addressing, so it is written as usrDataType = VAR_TYPE_WORD. If the letter is other than L, H or W, the user data type is written as userDataType = VAR_TYPE_BIT.

Other types are written as usrDataType =VAR_TYPE_UNKNOWN.

List 5-13 CaoProvVariable.cpp – FinalInitialize()

```
/**    Initialization
 *
 *   This function is called first after the CaoProvVariable object creation
 *   Variable names must be specified by "< group code >< address >".
 *
 *     @param   pObj     :        [in] parent object
 *     @retval  HRESULT
 *
 */
HRESULT CCaoProvVariable::FinalInitialize(PVOID pObj)
{
//↓[14]-------------------------------------------------------------------------↓
 CCaoProvController* pCaopCtrl = NULL;

// Judgment of parent object
 switch (m_ulParentType) {
 case SYS_CLS_CONTROLLER:
       pCaopCtrl = (CCaoProvController*)pObj;
       break;
```

```
default:
return E_NOTIMPL;
}

// Obtain a serial device object
HRESULT hr = pCaopCtrl->GetSerial(&m_pSerial);
if (FAILED(hr)) {
      return hr;
}

// Check the variable name
if (m_bSystem) {                  // For the system variable
      // Return an error for the system variable unimpmeneted.
      if (!_wcsicmp(m_bstrName, VAR_CURRENT_TIME$)) {
               m_dwSysID = VAR_CURRENT_TIME;
      }
      else if (!_wcsicmp(m_bstrName, VAR_ERROR_DESCRIPTION$)) {
               m_dwSysID = VAR_ERROR_DESCRIPTION;
      }
      else if (!_wcsicmp(m_bstrName, VAR_MAKER_NAME$)) {
               m_dwSysID = VAR_MAKER_NAME;
      }
      else if (!_wcsicmp(m_bstrName, VAR_TYPE$)) {
               m_dwSysID = VAR_TYPE;
      }
      else if (!_wcsicmp(m_bstrName, VAR_VERSION$)) {
               m_dwSysID = VAR_VERSION;
      }
      else {
               return E_INVALIDARG;
      }
}
else {                            // For the user variable

      // Convert it into the capital letter.
      CComBSTR bstrTemp(m_bstrName);
      hr = bstrTemp.ToUpper();
      if (FAILED(hr)) {
               return hr;
      }
      int ilen = bstrTemp.Length();
      m_bstrUpperName = bstrTemp.Detach();

      // Judte the data type by the first character.
      // It will be word-unit if it is a bit-type with the last character of W, L, or H.

      // Judge the  first character.
      switch (m_bstrUpperName[0]) {
               case L'X':
               case L'Y':
               case L'Z':
               case L'R':
               case L'E':
               case L'L':
               case L'T':
               case L'C':
               case L'A':
                        // Inspect the address character string length
                        if (ilen != CMD_ADD_LEN) {
                                 return E_INVALIDARG;
                        }
               // Judge the last character.
               switch (m_bstrUpperName [ilen-1]) {
               case L'W':
               case L'H':
               case L'L':
```

```
                        m_usrDataType = VAR_TYPE_WORD;
                        break;
                default:
                        m_usrDataType = VAR_TYPE_BIT;
                        break;
                }
                break;
        case L'D':
        case L'P':
        case L'V':
                // Inspect the address character string length
                if (ilen != CMD_ADD_LEN) {
                        return E_INVALIDARG;
                }
                m_usrDataType = VAR_TYPE_WORD;
                break;
        // Others
        default:
           // Send a specified character string as a general command
           // to the target without any modification.
           // ex. "2,X000,Y007,T002"
           //   Send it to the target as 0x1B + "2, X000,Y007,T002"+ 0x0D.
           //   If the response is "1,0"+0x0D, character string of "1,0" is returned.
           m_usrDataType = VAR_TYPE_UNKNOWN;     // Unknown type
           break;
           }
}
 return S_OK;
//↑[14]-------------------------------------------------------------------↑
}
```

5.3.4.2. Implementation of FinalGetValue()

The next step is implementation of the FinalGetValue() method.

This method firstly distinguishes whether the variable that is currently handled is a system variable or a user variable. If it is a system variable, the GetSystemValue function is called. The explanation of this function is described later.

If it is a user variable, create a character strings for command which has “2” or “5” on the top. If the user variable is VAR_TYPE_BIT, use “2”. This deribes from <CMD>”2”(0x32) command. If the user variable is VAR_TYPE_WORD, use “5”. This deribes from <CMD>”5”(0x35) command. If the user variable is VAR_TYPE_UNKNOWN, create a character string for command by copying as it is.

Next, create a transmission command by adding < EC > code on the top of the character string.

And then, send a commands to the TCmini controller to receive a response. (SendAndReceive)

At this time, < CR > code is automatically added as a terminator by the Serial class, and it is deleted before the response is returned.

If it is VAR_TYPE_BIT, and VAR_TYPE_WORD, the received character string is converted into the numerical value and is returned. If it is VAR_TYPE_UNKNOWN, the received character string is returned.

List 5-14 CaoProvVariable.cpp – FinalGetValue()

```
/**    Obtain a variable value
 *
 *  This will be the substance of ICaoVariable::get_Value().
 *
 *     @param   pVal     :        [in] variable value
 *     @retval  HRESULT
 *
 */
HRESULT CCaoProvVariable::FinalGetValue(VARIANT *pVal)
{
//↓[15]--------------------------------------------------------------------↓
       HRESULT hr = S_OK;
       int iCmdBufLen = 0;

       // For the system variable
       if (m_bSystem) {
                return GetSystemValue(pVal);
       }

       // Create a sending command character string
       // ----------
       // For bit-unit
                // For reading by bit-unit [2,] >> Response '0'/'1'
       // For word-unit
                // For reading by word-unit [5,] >> Response is decimal number
       // Others
            // Send a specified character string as a general command
            // to the target without any modification.
            // ex. "2,X000,Y007,T002"
            // Sent it to the target as 0x1B+"2, X000,Y007,T002"+0x0D
            // If the response is "1,0"+0x0D, character string "1,0" is returned

       WCHAR szCmdBuf[VAR_COMMAND_MAX];     // Sending buffer
       szCmdBuf[iCmdBufLen++] = CMD_EC;     // Command start code
       switch (m_usrDataType) {
       case VAR_TYPE_BIT:
            szCmdBuf[iCmdBufLen++] = L'2';     // <CMD> = '2'
            szCmdBuf[iCmdBufLen++] = L',';
            break;
       case VAR_TYPE_WORD:
            szCmdBuf[iCmdBufLen++] = L'5';     // <CMD> = '5'
            szCmdBuf[iCmdBufLen++] = L',';
            break;
       case VAR_TYPE_UNKNOWN:
            break;
       default:
            return E_NOTIMPL;     // Data type error
       }

       wcscpy(&szCmdBuf[iCmdBufLen], m_bstrUpperName);
       iCmdBufLen += SysStringLen(m_bstrUpperName);
       szCmdBuf[iCmdBufLen] = NULL;

       // Command sending and response receiving
       WCHAR szDataBuf[VAR_COMMAND_MAX];
       DWORD dwDataLen;
       hr = m_pSerial->SendAndReceive((LPBYTE)szCmdBuf,
                                      0,
                                     (LPBYTE)szDataBuf,
                                     VAR_COMMAND_MAX * sizeof(WCHAR),
                                     &dwDataLen);
       if (FAILED(hr)) {
            return hr;
       }

       // Convert the receved character string to data.
```

```
        BOOL bError = FALSE;
        switch (m_usrDataType) {
        case VAR_TYPE_BIT:
        case VAR_TYPE_WORD:
              pVal->vt = VT_I2;              //Substitute VT_I2 for vt of pVal.
              pVal->iVal = _wtoi(szDataBuf); // Convert into int-type
                                            //and then substitutes it for vt of pVal
              // If the value conversion is failed
              if (pVal->iVal == 0 && _wcsicmp(szDataBuf, L"0")) {
                 bError = TRUE;
              }
              break;
        case VAR_TYPE_UNKNOWN:
              bError = TRUE;
              break;
        }

        if (bError) {
              pVal->vt = VT_BSTR;
              pVal->bstrVal = SysAllocStringByteLen((char*)szDataBuf, dwDataLen);
        }

 return hr;
//↑[15]--------------------------------------------------------------------↑
}
```

5.3.4.3. Implementation of GetSystemValue()

TCmini-provider is set so that five types of system variables can be obtained. The processing of each system variable is added as follows.

1. Correspondence to "@CURRENT_TIME"

   Add a processing so as to return the present time by referring to the TCmini controller built-in calendar corresponding to the CS_CURRENT_TIME which is switch-case syntax.

   In TCmini-controller, the present time is obtained by referring to a register between D120 to D126. Note that A009=1 must be set to get the present time correctly. Be sure to check this when debugging.

   Information on the D120-D126 register is read at a time by using < CMD>"5" (0x35) command. The present time returned by this command will be returned by VT_DATA type.

2. Correspondence to "@ERROR_DESCRIPTION"

   Add processing so as to return error information in the TCmini controller, corresponding to CS_ERROR_DESCRIPTION of the switch-case syntax.

   Controller's error information can be acquired by using < CMD>"1" (0x31) command. The character string returned by this command will be returned by BSTR type.

3. Correspondence to "I_MAKER_NAME"

   Add processing so as to return "Toshiba Machine" by BSTR type as shown in the specification, since

it corresponds to CI_MAKER_NAME of the switch-case syntax.

4. Correspondence to "I_TYPE"

   Add processing so as to return "TCmini" by BSTR type as shown in the specification, since it corresponds to CI_TYPE of the switch-case syntax.

5. Correspondence to "I_VERSION"

   Add processing so as to return "1.0.0" by BSTR type as shown in the specification, since it corresponds to CI_TYPE of the switch-case syntax.

Following sample code contains the processing described above.

List 5-15 CaoProvVariable.cpp – GetSystemValue()

```
// ======= Manufacturer-specific additional functions should be written below.==========
//↓[16]-----------------------------------------------------------------------↓
/** Acquisition of value of system variable
 *
 *     @param   pVal     :        [Out] Variable value
 *     @retval  HRESULT
 *
 */
HRESULT CCaoProvVariable::GetSystemValue(VARIANT *pVal)
{
       HRESULT hr = S_OK;
       WCHAR szCmdBuf[VAR_COMMAND_MAX];
       WCHAR szDataBuf[VAR_COMMAND_MAX];
       DWORD dwDataLen;

       // "@CURRENT_TIME": Current time where the controller possesses.
       switch (m_dwSysID) {
       case VAR_CURRENT_TIME:

                // Create a command character string to be sent
                // ----------
                // Receive data as a batch process per word-unit[5,] and
                // obtain time information with a command.
                //(Attention) Time information can't be got unless "A009=1" is set
                // <EC>+"5,D125,D124,D123,..."+<CR>
                //       D120: Second
                //       D121: Minute
                //       D122: Hour
                //       D123: Day
                //       D124: Month
                //       D125: Year
                //       D126: Day of the week  00-06 = Sunday - Saturday

                swprintf(szCmdBuf,L"%c5,D126,D125,D124,D123,D122,D121,D120%c",
                         CMD_EC, CMD_CR);

                // Command sending and response receiving
                hr = m_pSerial->SendAndReceive((LPBYTE)szCmdBuf,
                                   0,
                                   (LPBYTE)szDataBuf,
                                   VAR_COMMAND_MAX * sizeof(WCHAR),
                                   &dwDataLen);
```

```
                if (FAILED(hr)) {
                        return hr;
                }

                int n, iYear, iMonth, iDay, iHour, iMinute, iSecond, iDayOfWeek;
                n = swscanf(szDataBuf, L"%d,%d,%d,%d,%d,%d,%d",
                        &iDayOfWeek, &iYear, &iMonth, &iDay,
                        &iHour, &iMinute, &iSecond);
                if (n!=7) {// failed to obtain correctly
                        return E_UNEXPECTED;
                }

                // Since “Year” is 02 -> 2002
                iYear += 2000;

                SYSTEMTIME tm;
                tm.wYear = (WORD)iYear;
                tm.wMonth = (WORD)iMonth;
                tm.wDay = (WORD)iDay;
                tm.wHour = (WORD)iHour;
                tm.wMinute = (WORD)iMinute;
                tm.wSecond = (WORD)iSecond;
                tm.wDayOfWeek = (WORD)iDayOfWeek;
                tm.wMilliseconds = 0;

                // Return a value converted to DATE.
                if (!SystemTimeToVariantTime(&tm, &(pVal->date))) {
                        return E_UNEXPECTED;
                }
                pVal->vt = VT_DATE;

                hr = S_OK;
                break;

        //"@ERROR_DESCRIPTION": Explanation of currently generating error.
        case VAR_ERROR_DESCRIPTION:

                // Create a sending command character string
                // ----------
                // Obtain an alarm code.
                // <EC>+"1"+<CR>

                swprintf(szCmdBuf, L"%c1%c", CMD_EC, CMD_CR);

                //Command sending and response receiving
                hr = m_pSerial->SendAndReceive((LPBYTE)szCmdBuf,
                                0
                                (LPBYTE)szDataBuf,
                                VAR_COMMAND_MAX * sizeof(WCHAR),
                                &dwDataLen);
                if (FAILED(hr)) {
                        return hr;
                }

                pVal->vt = VT_BSTR;
                pVal->bstrVal = SysAllocStringByteLen((char*)szDataBuf, dwDataLen);
                hr = S_OK;
                break;

         //"I_MAKER_NAME": Controller's manufacturer name (given character string)
        case VAR_MAKER_NAME:
                pVal->vt = VT_BSTR;
                pVal->bstrVal = SysAllocString(TC_PROV_MAKER);
                hr = S_OK;
                break;
```

```
	//"I_TYPE": Controller's model (given character string)
	case VAR_TYPE:
		pVal->vt = VT_BSTR;
		pVal->bstrVal = SysAllocString(TC_PROV_TYPE);
		hr = S_OK;
		break;

	//"I_VERSION": Controller's version (given character string)
	case VAR_VERSION:
		pVal->vt = VT_BSTR;
		pVal->bstrVal = SysAllocString(TC_PROV_VER);
		hr = S_OK;
		break;

	default:
		hr = E_INVALIDARG;
	}

	return hr;
}
//↑[16]----------------------------------------------------------------↑
```

5.3.4.4. Implementation of FinalPutValue ()

As a last step of the preparation for the CCaoProvVariable class implementation, install the FinalPutValue() method.

The difference between FinalGetValue and FinalPutValue are commands used. In this FinalPutValue(), if user variable is VAR_TYPE_BIT, <CMD>"8"(0x38) and <CMD>"9"(0x39) are used for set and reset, respectively. If user variable is VAR_TYPE_WORD, <CMD>"10"(0x30. 0x31) is used. Also, the response character string is set to "OK" in FinalPutValue, while this is not allowed in FinalGetValue.

List 5-16 CaoProvVariable.cpp – FinalPutValue()

```
/** Change of value of variable
 *
 *  It becomes the substance of ICaoVariable::put_Value().
 *
 *     @param   newVal    :        [in]Variable value
 *     @retval  HRESULT
 *
 */
HRESULT CCaoProvVariable::FinalPutValue(VARIANT newVal)
{
//↓[17]----------------------------------------------------------------↓
 HRESULT hr = S_OK;
 WCHAR szCmdBuf[VAR_COMMAND_MAX];
 int iCmdBufLen = 0;

// A set value is acquired.
 CComVariant vntVal;
 hr = vntVal.ChangeType(VT_I2, &newVal);
 if (FAILED(hr)) {
	return hr;
 }
 short nVal = vntVal.iVal;
```

```
// Create a sending command character string
// ----------
// For bit-unit
      // To set with bit-unit, use [8,]. To reset with-bit unit, use [9,].
// For word-unit
      // To write with word-unit, use [10,].
// For others
      //Unimplemented

szCmdBuf[iCmdBufLen++] = CMD_EC;     // Command starting code
switch (m_usrDataType) {
case VAR_TYPE_BIT:
      szCmdBuf[iCmdBufLen++] = (nVal ? '8' : '9'); // <CMD> = '8'-ON, '9'-OFF
      szCmdBuf[iCmdBufLen++] = ',';
      break;
case VAR_TYPE_WORD:
      szCmdBuf[iCmdBufLen++] = '1';                // <CMD> = '10'
      szCmdBuf[iCmdBufLen++] = '0';
      szCmdBuf[iCmdBufLen++] = ',';
      break;
case VAR_TYPE_UNKNOWN:
      return E_NOTIMPL;     // Unimplemented
      break;
default:
      return E_NOTIMPL;     // Data type error
}

wcscpy(&szCmdBuf[iCmdBufLen], m_bstrUpperName);
iCmdBufLen += SysStringLen(m_bstrUpperName);

// If word-unit, specify a set value.
if (m_usrDataType == VAR_TYPE_WORD) {
      swprintf( &(szCmdBuf[iCmdBufLen]), L",%d", nVal );
      iCmdBufLen += wcslen(szCmdBuf);
}
szCmdBuf[iCmdBufLen] = NULL;

// Command sending and response receiving
WCHAR szDataBuf[VAR_COMMAND_MAX];
DWORD dwDataLen;
hr = m_pSerial->SendAndReceive((LPBYTE)szCmdBuf,
                                    0
                                    (LPBYTE)szDataBuf,
                                    VAR_COMMAND_MAX * sizeof(WCHAR),
                                    &dwDataLen);
if (FAILED(hr)) {
      return hr;
}

// Check the response character string.
switch (m_usrDataType) {
case VAR_TYPE_BIT:
case VAR_TYPE_WORD:
      // "OK"+<CR> is normal.
      if (wcscmp(szDataBuf, L"OK¥r")) {
               hr = E_FAIL;
      }
      break;
}

return hr;
//↑[17]----------------------------------------------------------------↑
}
```

5.3.5. Summary

The coding of the TCmini provider has completed.

All the process we have done so far is as follows.

(1) A skeleton project was created by the project wizard of the CAO provider.
(2) CSerial class that controls RS-232C communication was added.
(3) Initialization was implemented with CCaoProvController::FinalInitialize().
(4) Connection parameter analysis process was implemented with CCaoProvController::ParseParameter().
(5) Connection process was implemented with CCaoProvController::FinaleConnect().
(6) The disconnection process was mounted with CCaoProvController::FinalDisconnect().
(7) The device object pointer acquisition process was implemented with CCaoProvController::GetSerial().
(8) The variable name list acquisition process was implemented with CCaoProvController::FinalGetVariableNames().
(9) Initialization was implemented with CCaoProvVariable::FinalInitialize().
(10) The user variable acquisition process was implemented with CCaoProvVariable::FinalGetValue().
(11) The system variable acquisition process was implemented with CCaoProvVariable::GetSystemValue().
(12) The user variable setting process was implemented with CCaoProvVariable::FinalPutValue().

From the menu tab of VC++, execute [Build (B)] to create TCmini.DLL.

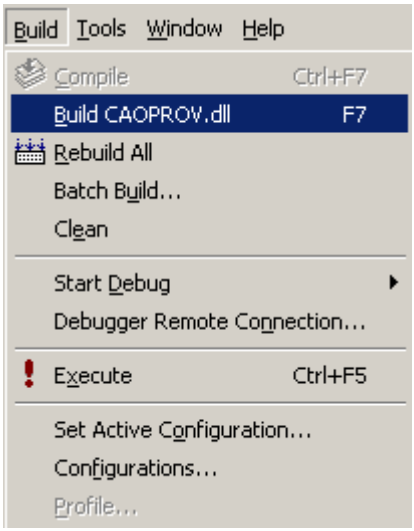


Figure5-4 Build of TCmini.DLL

5.4. Debugging and release of TCmini provider

5.4.1. Debugging of TCmini provider

The DLL module of the CAO provider is called by CAO.exe which is a part of CAO engine when needed. Therefore, to debug the CAO provider DLL module, you need to set in Executable for Debug session in VC++

as CAO.exe.

The debugging procedure of the TCmini provider is as follows. Use ORiN2¥CAO¥Tools¥CaoTester.exe, which is CAO standard tool of ORiN, as a client application.

Connect a PC and TCmini controller with a serial cable and turn ON the controller before you start debug.

(1) Set the build-target to [CAOPROV-Win32 Debug].

From the [Build(B)] menu, select [Active Configuration...(C)]. And then, from the [Project Configuration(P)], select [CAOPROV-Win32 Debug].

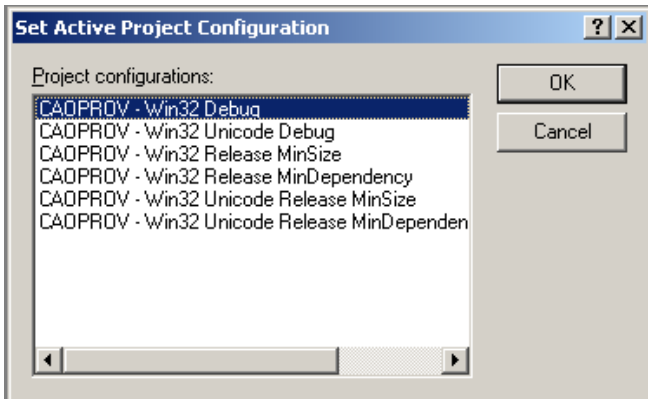


Figure5-5 Build-target specification dialog

(2) Set CAO.exe in Executable For Debug Session..

From the [Project(P)] menu, click [Settings(S) Alt+F7]. From the drop-down list box on the left side, select [Win32 Debug], and then enter Cao.exe with full path in the [Executable For Debug Session(E)] textbox. CAO.exe is usually under the ORiN2¥CAO¥Engine¥Bin folder.

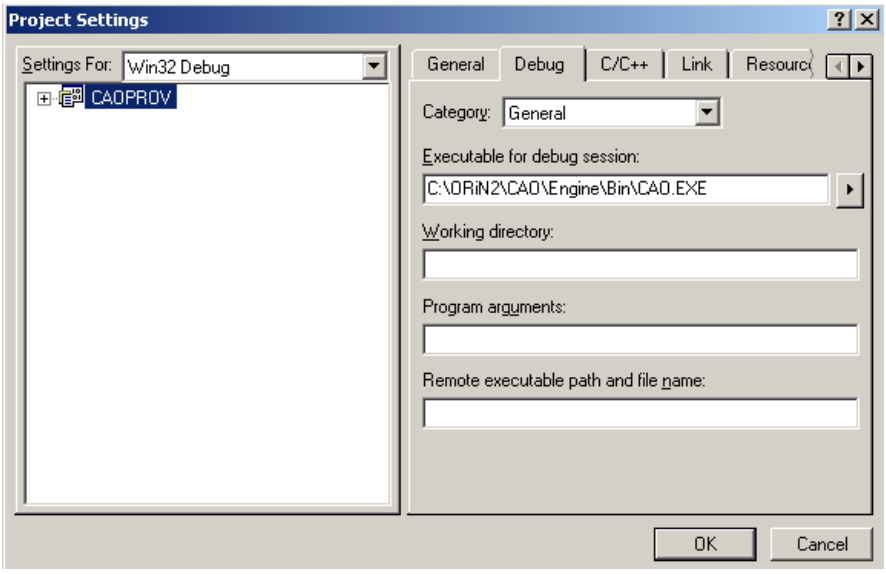


Figure5-6 CAO.exe setting dialog

(3) Set breakpoints at necessary positions.

(4) Start debugging

From the [Build(B)] menu, select [Start Debug (D)] – [Go(G)].

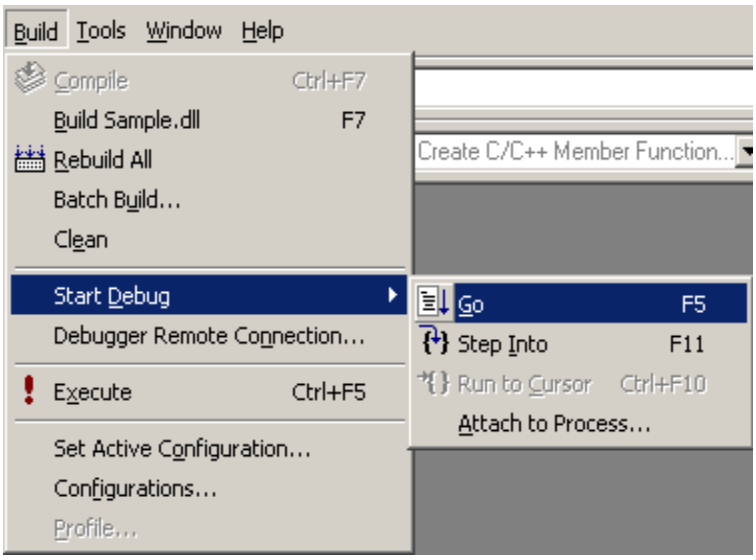


Figure5-7 Start debug

(5) Start CaoTester.exe, and then specify TCmini provider.

From the pull-down menu of Provider Name, select [CaoProv.TCmini]. Enter “Conn=com:1” in the Option text box.

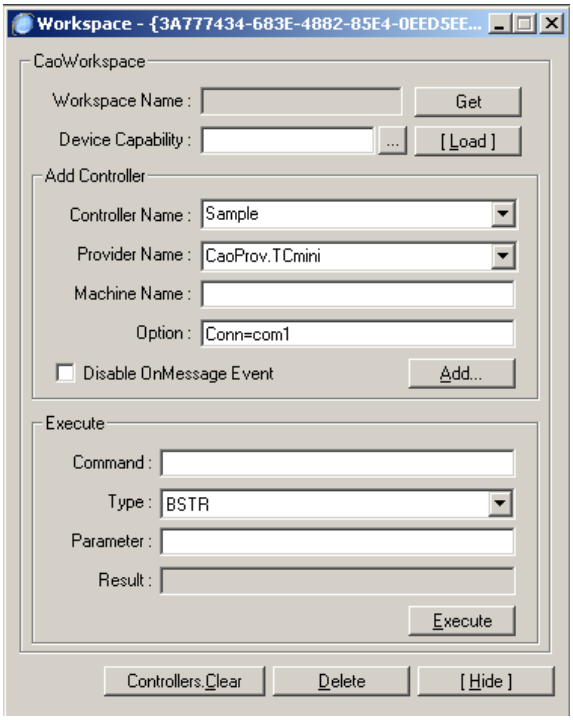


Figure5-8 ImplChk - TCmini provider specification dialog

(6) Connect with the controller by pressing [Add].

(7) Execute CAO service with CaoTester to call TCmini provider.

For example, to create a user variable “X007" and to acquire its value, do the following steps.

1. From the Controller window, select Variable tab. Enter “X007” in the Name textbox.
2. Click [Add].
3. From the Variable window, in the Value area, click [Get] to obtain the current state of X007.

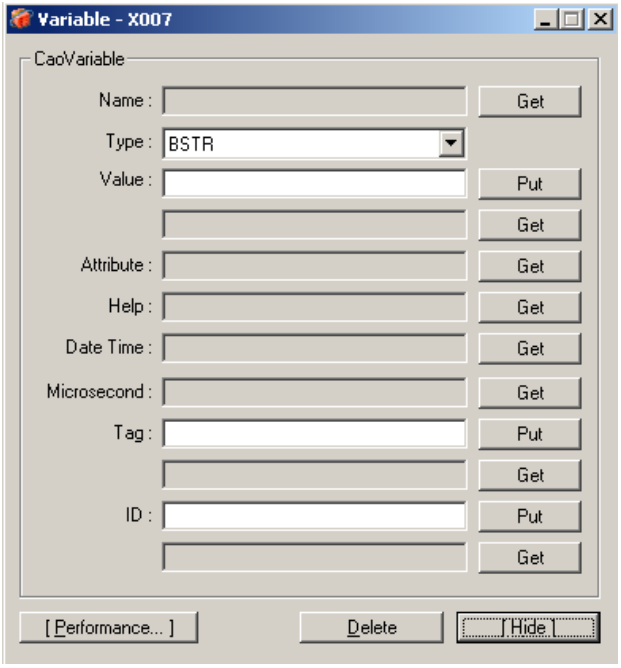


Figure5-9 Value acquisition dialog

To set a value, create a variable name first, enter a value, and then click [Put].

For example, to set “1” in a user variable”Y027”, create “Y027”, enter ”1” in the Value text box, and then click [Put] in the Value area.

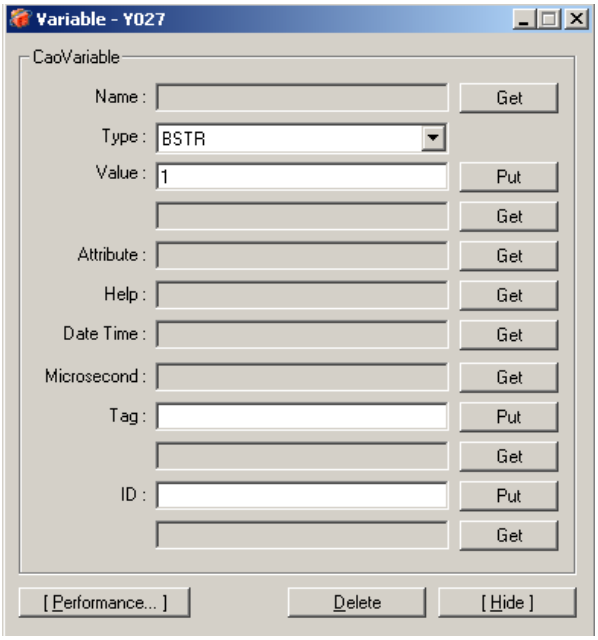


Figure5-10 Value setting dialog

To handle a System variable, from the Variable tab, in Variable Names pane, click [Get] button.

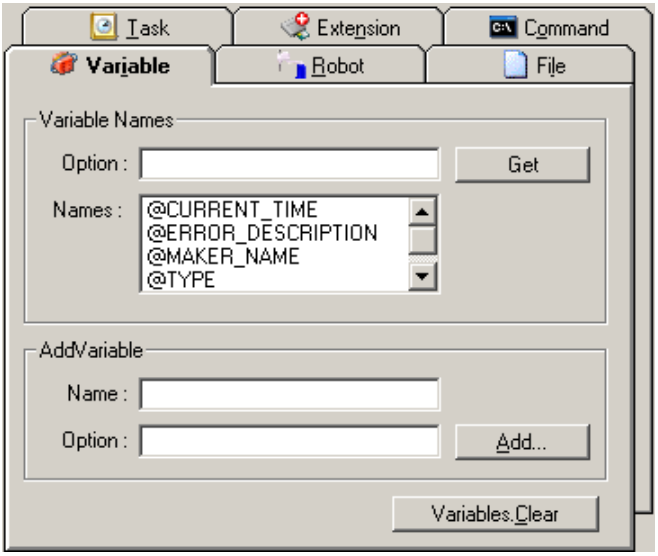


Figure5-11 System variable selection dialog

(8) Once return to VC++, if the program stops at a break pointer position, debug with VC++.

5.4.2. Release of TCmini provider

Once debug has completed, change the build-target to [CAOPROV-Win32 Release MinDependency], and then create a final version of TCmini.dll module.

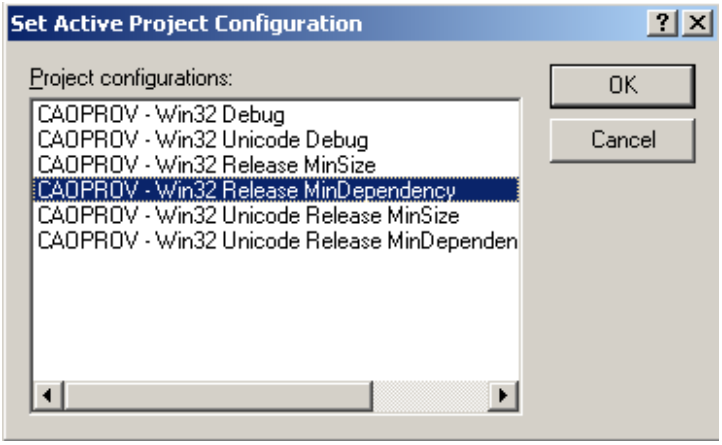


Figure5-12 Build-target setting dialog

5.4.2.1. Creating a document for release

The provider cannot be easily used from the user unless the created provider's specification is open to the public. Then, it is necessary to create a provider's specification document to open the provider to the public.

At least, the following items should be written in the specification for the CAO provider

(1) Specification of connection parameter of AddController()

(2) Specification of user variable

(3) Corresponding list of system variable and the meaning

(4) Other information that seems to be important (information and notes concerning provider original function, etc.)

Since there are no contents that you have to take into special consideration in TC-mini provider we have created this time, create a provider specification document that covers the contents from 1 to 3 of above. Use a template file (<ORiN2>¥CAO¥Provider¥Doc¥xxx_ProvGuide_jp.dot) of Microsoft Word. Refer to "User's Guide for TC-mini Provider" as a sample.

5.4.2.2. Confirmation of provider dependence information

You can check the module dependence relationship by using Dependency Walker tool that is accompanied with VC++.

When TCmini.dll is specified by starting Dependency Walker, the following window is displayed.

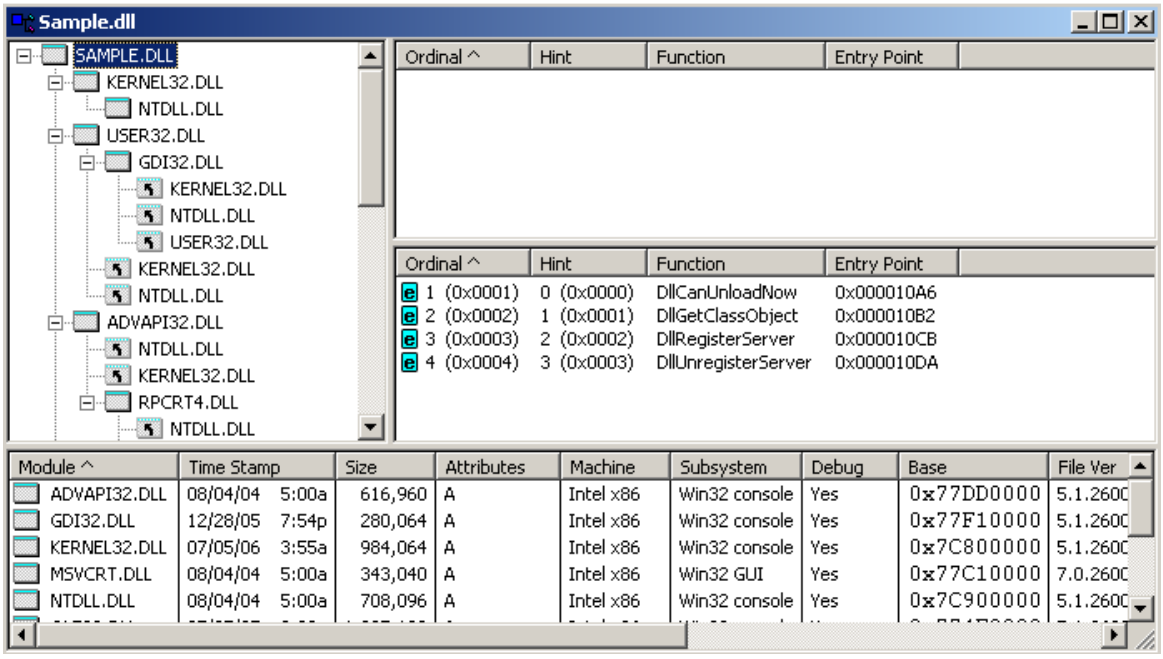


Figure5-13 Dependency Walker window in TCmini.DLL

From this window, we can see that the TCmini.dll requires standard modules only.[8] Therefore, to link CAO engine to TCmini provider, you simply copy TCmini.dll and to register it with Regsvr32.

[8] Some modules load other module dynamically. In that case, it is not detected in Dependency Walker. To check that, execute a module you want to examine, and investigate modules loaded into the memory at that time..

6. Use of CaoProvExec tool

CaoProvExec is a tool to start a provider with a substitute process (dllhost.exe).

To use COM in Windows 95/98, you need to start modules beforehand, but dll alone cannot be executed. So, to start CAO provider, use a substitute process instead.

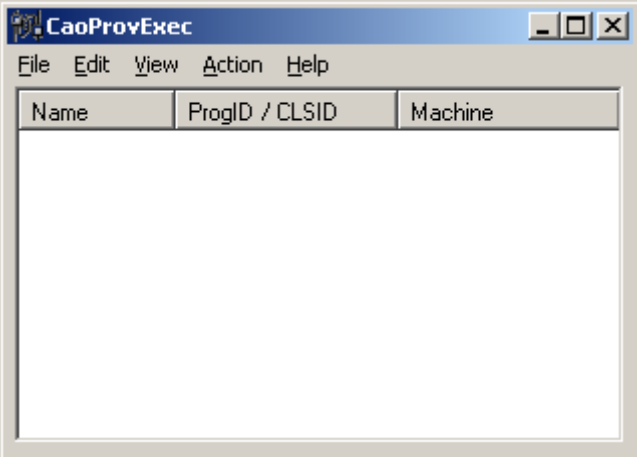


Figure6-1 CaoProvExec tool

From the [Edit] menu, select [Add] to display the following dialog. Enter a Name, Program ID or class ID and the Machine name to start, and then click [OK].

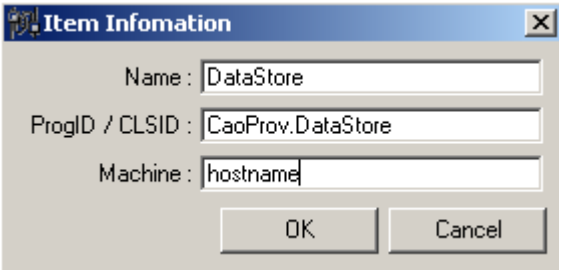


Figure6-2 Addition of provider

You can create a provider instance from [Action] – [CreateInstance].

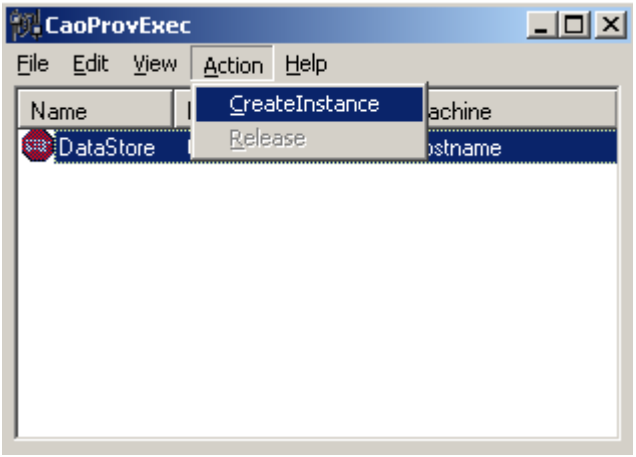


Figure6-3 Create Instance

Appendix A.

Appendix A.1. CAO provider function list

- CaoProvController Object-Controller

Class	Property, method, and event		Explanation	R/W	Argument of function		Called timing	Remarks
					IN	OUT RETVAL		
CaoProvController	Attribute	P	Acquisition of attribute	R		Attribute: Long	When CaoController::Attribute() is called.	
	CommandNames	P	Acquisition of command name list	R	[Option: BSTR]	Command name list: VARIANT (VT_VARIANT|VT_ARRAY)	When CaoController::CommandNames() is called.	Available options are the filter condition and others..
	ExtensionNames	P	Acquisition of extension board name list	R	[Option: BSTR]	Extension board name list: VARIANT (VT_VARIANT|VT_ARRAY)	When CaoController::ExtensionNames() is called.	Available options are the filter condition and others..
	FileNames	P	Acquisition of file name list	R	[Option: BSTR]	File name list: VARIANT (VT_VARIANT|VT_ARRAY)	When CaoController::FileNames() is called.	Available options are the filter condition and others.. The file list of the root directory is returned.
	Help	P	Help	R		Help character string: BSTR	When CaoController::Help() is called.	
	RobotNames	P	Acquisition of robot name list	R	[Option: BSTR]	Robot name list: VARIANT (VT_VARIANT|VT_ARRAY)	When CaoController::RobotNames() is called.	Available options are the filter condition and others..
	TaskNames	P	Acquisition of task name list	R	[Option: BSTR]	Task name list: VARIANT (VT_VARIANT|VT_ARRAY)	When CaoController::TaskNames() is called.	Available options are the filter condition and others..

Class	Property, method, and event		Explanation	R/W	Argument of function		Called timing	Remarks
					IN	OUT RETVAL		
	VariableNames	P	Acquisition of variable name list	R	[Option: BSTR]	Variable name list: VARIANT (VT_VARIANT|VT_ARRAY)	When CaoController::VariableNames() is called.	Available options are the filter condition and others..
	Connect	M	Connection	S	Controller name: BSTR [Option:BSTR]		When CaoWorkspace::AddController() is called.	
	Disconnect	M	Disconnection	S			When the CaoController object disappears.	
	GetCommand	M	Acquisition of command object	S	Command name: BSTR [Option:BSTR]	Object: ICaoProvCommand	When CaoController::AddCommand() is called.	
	GetExtension	M	Acquisition of extension board object	S	Extension board name: BSTR [Option:BSTR]	Object: ICaoProvExtension	When CaoController::AddExtension() is called.	
	GetFile	M	Acquisition of route file object	S	File name: BSTR [Option:BSTR]	Object: ICaoProvFile	When CaoController::AddFile() is called.	
	GetRobot	M	Acquisition of robot object	S	Robot name: BSTR [Option:BSTR]	Object: ICaoProvRobot	When CaoController::AddRobot() is called.	
	GetTask	M	Acquisition of task object	S	Task name: BSTR [Option:BSTR]	Object: ICaoProvTask	When CaoController::AddTask() is called.	
	GetVariable	M	Acquisition of variable object	S	Variable name: BSTR [Option:BSTR]	Object: ICaoProvVariable	When CaoController::AddVariable() is called.	
	Execute	M	Execution of expansion command	S	Command: BSTR [Parameter: VARIANT]	Result: VARIANT	When CaoController::Execute() is called.	For functional expansion
	OnMessage	E	Message reception event	R	Message: ICaoProvMessage		n/a	The call of Provider (controller) > Engine > Client is achieved. - This doesn't reach the

Class	Property, method, and event	Explanation	R/W	Argument of function		Called timing	Remarks
				IN	OUT RETVAL		
							client when the system message option is set. - A negative range of the message number has been reserved with ORiN.
Meaning of sign	M:Method P:Property E:Event		(Note 1)	・Arguments enclosed by square brackets can be omitted. ・The default value of the BSTR type's omittable argument is NULL. ・The default value of the numerical type's omittable argument is 0. ・The default value of the VARIANT type's omittable argument is VT_ERROR.			

(Note 1)The meaning of each sign is as follows. Only the method and the property of W attribute will be influenced by ON/OFF of the DAO engine access limitation function (writing limitation function) .
R -Read: Status and the configuration of the controller or the provider or the engine are acquired.
W -Write: Controller's status and configuration are changed. However, because the Execute method depends on the command's argument, it is assumed S attribute. The writing limitation can be mounted in the provider, if necessary.
S -Setup: Status and the configuration of the provider or the engine are changed.

◆ CaoProvExtension Object-Extension board

Class	Property, method, and event		Explanation	R/W	Argument of function		Called timing	Remarks
					IN	OUT RETVAL		
CaoProvExtension	Attribute	P	Acquisition of attribute	R		Attribute: Long	When CaoExtension::Attribute() is called.	
	Help	P	Help	R		Help character string: BSTR	When CaoExtension::Help() is called.	
	VariableNames	P	Acquisition of variable name list	R	[Option: BSTR]	Variable name list: VARIANT (VT_VARIANT|VT_ARRAY)	When CaoExtension::VariableNames() is called.	Available options are the filter condition and others..
	GetVariable	M	Acquisition of variable object	S	Variable name: BSTR [Option: BSTR]	Object: ICaoProvVariable	When CaoExtension::AddVariable() is called.	
	Execute	M	Execution of expansion command	S	Command: BSTR [Parameter: VARIANT]	Result: VARIANT	When CaoExtension::Execute() is called.	For function expansion
Meaning of sign	M:Method P:Property E:Event			(Note 1)	・Arguments enclosed by square brackets can be omitted. ・The default value of the BSTR type's omittable argument is NULL. ・The default value of the numerical type's omittable argument is 0. ・The default value of the VARIANT type's omittable argument is VT_ERROR.			

◆ CaoProvFile Object-File

Class	Property, method, and event		Explanation	R/W	Argument of function		Called timing	Remarks
					IN	OUT RETVAL		
CaoProvFile	Attribute	P	Acquisition of attribute	R		Attribute: Long	When CaoFile::Attribute() is called.	
	DateCreated	P	Date and time of creation	R		Date and time of creation: VARIANT	When CaoFile::DateCreated() is called.	
	DateLastAccessed	P	Last access date and time	R		Last access date and time: VARIANT	When CaoFile::DateLastAccessed() is called.	
	DateLastModified	P	The final change date	R		The final change date: VARIANT	When CaoFile::DateLastModified() is called.	
	FileNames	P	Acquisition of file name list	R	[Option: BSTR]	File name list: VARIANT (VT_VARIANT|VT_ARRAY)	When CaoFile::FileNames() is called.	Available options are the filter condition and others.. When the attribute is a directory, the child file name list is returned.
	Help	P	Help	R		Help character string: BSTR	When CaoFile::Help() is called.	
	Path	P	Acquisition of path	R		Path name: BSTR	When CaoFile::Path() is called.	
	Size	P	Acquisition of size of file	R		Size of file: Long	When CaoFile::Size() is called.	
	Type	P	Acquisition of type of file	R		File type: BSTR	When CaoFile::Type() is called.	
	Value	P	Content of file	R/W	Data: VARIANT	Data: VARIANT	When CaoFile::Value() is called.	
	VariableNames	P	Acquisition of variable name list	R	[Option: BSTR]	Variable name list: VARIANT (VT_VARIANT|VT_ARRAY)	When CaoFile::VariableNames() is called.	Available options are the filter condition and others. Data type is (VT_VARIANT|VT_ARRAY)

	GetFile	M	Acquisition of file object	S	File name: BSTR [Option: BSTR]	Object: ICaoProvFile	When CaoFile::AddFile() is called.	
	Copy	M	Copy	W	Copy destination file name: BSTR [Option: BSTR]		When CaoFile::Copy() is called.	
	Delete	M	Deletion	W	[Option: BSTR]		When CaoFile::Delete() is called.	
	Execute	M	Execution of expansion command	S	Command: BSTR [Parameter: VARIANT]	Result: VARIANT	When CaoFile::Execute() is called.	
	GetVariable	M	Acquisition of variable object	S	Variable name:BSTR [Option:BSTR]	Object: ICaoProvVariable	When CaoFile::AddVariable() is called.	
	Move	M	Move	W	Moving destination file name: BSTR [Option: BSTR]		When CaoFile::Move() is called.	
	Run	M	Task creation	W	[Option: BSTR]	Task name: BSTR	When CaoFile::Run() is called.	
Meaning of sign	M:Method P:Property E:Event			(Note 1)	・Arguments enclosed by square brackets can be omitted. ・The default value of the BSTR type's omittable argument is NULL. ・The default value of the numerical type's omittable argument is 0. ・The default value of the VARIANT type's omittable argument is VT_ERROR.			

◆ CaoProvRobot Object-Robot

Class	Property, method, and event		Explanation	R/W	Argument of function		Called timing	Remarks
					IN	OUT RETVAL		
CaoProvRobot	Attribute	P	Acquisition of attribute	R		Attribute: Long	When CaoRobot::Attribute() is called.	
	Help	P	Help	R		Help character string: BSTR	When CaoRobot::Help() is called.	
	VariableNames	P	Acquisition of variable name list	R	[Option: BSTR]	Variable name list: VARIANT (VT_VARIANT|VT_ARRAY)	When CaoRobot::VariableNames() is called.	Available options are the filter condition and others.
	Accelerate	M	Refer to the specification of the ACCEL sentence of SLIM.	W	Axis number: Long Acceleration: Float [Deceleration:Float]		When CaoRobot::Accelerate() is called.	
	Change	M	Refer to the specification of the CHANGE sentence of SLIM.	W	Hand name: BSTR		When CaoRobot::Change() is called.	
	Chuck	M	Refer to the specification of the GRASP sentence of SLIM.	W	[Option: BSTR]		When CaoRobot::Chuck() is called.	
	Drive	M	Refer to the specification of the DRIVE sentence of SLIM.	W	Axis number: Long Distance: Float [Motion option:BSTR]		When CaoRobot::Drive() is called.	
	Execute	M	Execution of expansion command	S	Command: BSTR [Parameter: VARIANT]	Result: VARIANT	When CaoRobot::Execute() is called.	
	GetVariable	M	Acquisition of CaoProvVariable	S	Variable name:BSTR [Option:BSTR]	Object: ICaoVariable	When CaoRobot::AddVariable() is called.	
	GoHome	M	Refer to the specification of the GOHOME sentence of	W			When CaoRobot::GoHome() is called.	

		SLIM.					
Hold	M	Refer to the specification of the HOLD sentence of SLIM.	W	[Option: BSTR]		When CaoRobot::Hold() is called.	In SLIM program, it means a temporary stop of the program. However, in CAO program, it means a temporary stop of the robot operation.
Halt	M	Refer to the specification of the HALT sentence of SLIM.	W	[Option: BSTR]		When CaoRobot::Halt() is called.	In SLIM program, it means a mandatory stop of the program. However, in CAO program, it means a mandatory stop of the robot operation.
Move	M	Refer to the specification of the MOVE sentence of SLIM.	W	Interpolation:Long Pose : Variant [Motion option:BSTR]		When CaoRobot::Move() is called.	
Rotate	M	Refer to the specification of the ROTATE sentence of SLIM.	W	Rotation surface: VARIANT Angle: Float Center of rotation: VARIANT [Motion option:BSTR]		When CaoRobot::Rotate() is called.	
Speed	M	Refer to the specification of the SPEED/JSPEED sentence of SLIM.	W	Axis number: Long Speed: Float		When CaoRobot::Speed() is called.	The axis numbers are; -1: Speed of the tool end, 0: Speed of all axes, Others; Speed of the specified axis.
Unchuck	M	Refer to the specification of the RELEASE sentence of SLIM.	W	[Option: BSTR]		When CaoRobot::Chuck() is called.	Changed to Chuck and Unchuck. In SLIM, “Release” is not available because it is a reserved word.
Unhold	M	Release of HOLD sentence of SLIM	W	[Option: BSTR]		When CaoRobot::Unhold() is called.	In SLIM, since “Hold” is used to suspend a program, no resume command is prepared. In CAO, since “Hold” is used to suspend robot

							operation, any resume command is required. “Unhold” is used for that.
Meaning of sign	M:Method P:Property E:Event		(Note 1)	・Arguments enclosed by square brackets can be omitted. ・The default value of the BSTR type’s omittable argument is NULL. ・The default value of the numerical type’s omittable argument is 0. ・The default value of the VARIANT type’s omittable argument is VT_ERROR.			

◆ CaoProvTask Object-Task

Class	Property, method, and event		Explanation	R/W	Argument of function		Called timing	Remarks
					IN	OUT RETVAL		
CaoProvTask	Attribute	P	Acquisition of attribute	R		Attribute: Long	When CaoTask::Attribute() is called.	
	FileName	P	Acquisition of correspondence file name	R		File name: BSTR	When CaoTask::FileName is called.	
	Help	P	Help	R		Help character string: BSTR	When CaoTask::Help() is called.	
	VariableNames	P	Acquisition of variable name list	R	[Option: BSTR]	Variable name list: VARIANT (VT_VARIANT|VT_ARRAY)	When CaoTask::VariableNames() is called.	Available options are the filter condition and others.
	GetVariable	M	Acquisition of variable object	S	Variable name:BSTR [Option:BSTR]	Object: ICaoProvVariable	When CaoTask::AddVariable() is called.	
	Delete	M	Deletion of task	W	[Option: BSTR]		When CaoTask::Delete() is called.	
	Execute	M	Execution of expansion command	S	Command: BSTR [Parameter: VARIANT]	Result: VARIANT	When CaoTask::Execute() is called.	
	Start	M	Start of task	W	Mode:Long [Option:BSTR]		When CaoTask::Start() is called.	Available modes are; 1: Run one cycle, 2: Continuous operation, 3: One step forward, 4: One step back. Available options are the start position, etc.
	Stop	M	Stop of task	W	Mode:Long [Option:BSTR]		When CaoTask::Stop() is called.	Available modes are; 0: Default stop 1: Instantaneous stop 2: Step stop 3: Cycle stop 4: Initialization stop Note that the “Default

							stop" means one of the stop methods. (Step stop, Cycle stop, Initialization stop)
Meaning of sign	M:Method P:Property E:Event		(Note 1)	・Arguments enclosed by square brackets can be omitted. ・The default value of the BSTR type's omittable argument is NULL. ・The default value of the numerical type's omittable argument is 0. ・The default value of the VARIANT type's omittable argument is VT_ERROR.			

◆ CaoProvVariable Object-Variable

Class	Property, method, and event		Explanation	R/W	Argument of function		Called timing	Remarks
					IN	OUT RETVAL		
CaoProvVariable	Attribute	P	Acquisition of attribute	R		Attribute: Long	When CaoVariable::Attribute() is called.	
	DateTime	P	Acquisition of time stamp	R		Time stamp: VARIANT	When CaoVariable::DateTime() is called.	
	Help	P	Help	R		Help character string: BSTR	When CaoVariable::Help() is called.	
	Value	P	Acquisition of value	R/W	Value:VARIANT	Value:VARIANT	When CaoVariable::Value() is called.	
Meaning of sign	M:Method P:Property E:Event			(Note 1)	・Arguments enclosed by square brackets can be omitted. ・The default value of the BSTR type's omittable argument is NULL. ・The default value of the numerical type's omittable argument is 0. ・The default value of the VARIANT type's omittable argument is VT_ERROR.			

◆ CaoProvCommand Object-Command

Class	Property, method, and event		Explanation	R/W	Argument of function		Called timing	Remarks
					IN	OUT RETVAL		
CaoProvCommand	Attribute	P	Acquisition of attribute	R		Attribute: Long	When CaoCommand::Attribute() is called.	
	Help	P	Help	R		Help character string: BSTR	When CaoCommand::Help() is called.	
	Parameters	P	Command and parameter	R/W	Command parameter: VARIANT	Command parameter: VARIANT	When CaoCommand::Parameter() is called.	
	State	P	Acquisition of state	R		State: Long	When CaoCommand::State() is called.	
	Timeout	P	Time-out	R/W	Time-out: Timeout	Time-out: Timeout	When CaoCommand::State() is called.	
	Cancel	M	Cancellation of command when being executing it	S			When CaoCommand::Cancel() is called.	
	Execute	M	Execution of command	S	Option: Long	Execution result: VARIANT	When CaoCommand::Execute() is called.	
Meaning of sign	M:Method P:Property E:Event			(Note 1)	・Arguments enclosed by square brackets can be omitted. ・The default value of the BSTR type's omittable argument is NULL. ・The default value of the numerical type's omittable argument is 0. ・The default value of the VARIANT type's omittable argument is VT_ERROR.			

◆ CaoProvMessage Object-Message

Class	Property, method, and event		Explanation	R/W	Argument of function		Called timing	Remarks
					IN	OUT RETVAL		
CaoProMessage	DateTime	P	Date and time of creation	R		Date and time of creation: VARIANT	When CaoMessage::DateTime() is called.	
	Description	P	Explanation	R		Explanation: BSTR	When CaoMessage::Description() is called.	
	Destination	P	Destination	R		Destination: BSTR	When CaoMessage::Destination() is called.	
	Number	P	Message number	R		Message number: Long	When CaoMessage::Number() is called.	
	Option	P	Option	R		Option: Long	When you process the message with the DAO engine.	1 - Synchronous message. (Default is “Asynchronized”.) 2 - Log output. 4 - Engine control message. (reserved)
	Source	P	Source	R		Source: BSTR	When CaoMessage::Source() is called.	
	Value	P	Message text	R		Message text: VARIANT	When CaoMessage::Value() is called.	
	Clear	M	Clearness of message	W			When CaoMessage::Clear() is called.	
	Reply	M	Reply of message	W	Reply message: VARIANT		When CaoMessage::Reply() is called.	
Meaning of sign	M:Method P:Property E:Event			(Note 1)	・Arguments enclosed by square brackets can be omitted. ・The default value of the BSTR type’s omittable argument is NULL. ・The default value of the numerical type’s omittable argument is 0. ・The default value of the VARIANT type’s omittable argument is VT_ERROR.			

Appendix A.2. CAO provider template function list

- CaoProvControllerImpl Template-Controller

Class	Property, method, and event		Explanation	Argument of function		Called timing	Remarks
				IN	OUT RETVAL		
CaoProvControllerImpl	FinalGetAttribute	E	Acquisition of attribute		Attribute: Long	When CaoProvController::get_Attribute() is called.	
	FinalGetCommandNames	E	Acquisition of command name list	[Option: BSTR]	Command name list: VARIANT (VT_VARIANT|VT_ARRAY)	When CaoProvController::get_CommandNames() is called.	Available options are the filter condition and others..
	FinalGetExtensionNames	E	Acquisition of extension board name list	[Option: BSTR]	Extension board name list: VARIANT (VT_VARIANT|VT_ARRAY)	When CaoProvController::get_ExtensionNames() is called.	Available options are the filter condition and others..
	FinalGetFileNames	E	Acquisition of file name list	[Option: BSTR]	File name list: VARIANT (VT_VARIANT|VT_ARRAY)	When CaoProvController::get_FileNames() is called.	Available options are the filter condition and others..The file list of the root directory is returned.
	FinalGetHelp	E	Help		Help character string: BSTR	When CaoProvController::get_Help() is called.	
	FinalGetRobotNames	E	Acquisition of robot name list	[Option: BSTR]	Robot name list: VARIANT (VT_VARIANT|VT_ARRAY)	When CaoProvController::get_RobotNames() is called.	Available options are the filter condition and others..
	FinalGetTaskNames	E	Acquisition of task name list	[Option: BSTR]	Task name list: VARIANT (VT_VARIANT|VT_ARRAY)	When CaoProvController::get_TaskNames() is called.	Available options are the filter condition and others..
	FinalGetVariableNames	E	Acquisition of variable name list	[Option: BSTR]	Variable name list: VARIANT (VT_VARIANT|VT_ARRAY)	When CaoProvController::get_VariableNames() is called.	Available options are the filter condition and others..

FinalExecute	E	Execution of expansion command	Command: BSTR [Parameter: VARIANT]	Result: VARIANT	When CaoProvController::Execute() is called.	For function expansion
FinalConnect	E	Connection			When CaoProvController::Connect() is called.	
FinalDisconnect	E	Cutting			When CaoProvController::Disonnect() is called.	
FinalInitialize	E	Preprocessing			When the object is generated.	
FinalTerminate	E	Postprocessing			When the object disappears.	
OnTimer	E	Timer interrupt	Object: This pointer		When set time passes.	The cycle is defined by the DAOP_TIMER_INTERVAL macro.
CreateMessage	M	Making of message	Message number: Long, [Message text: VARIANT], [Date of creation: VARIANT], [Destination: BSTR], [Source: BSTR] [Description: BSTR] [Option: Long]	Message: CCaoProvMessage	n/a	·When arguments are omitted, the following values are set to the members of the message. Message number: 0 Message text: VT_EMPTY Date of creation: Execution date of CreateMessage Destination: Null character Source: Controller name Explanation: Null character
FireOnMessage	M	Transmission of message	Message: CCaoProvMessage		n/a	The call of Provider (controller) > Engine > Client is achieved. - This doesn't reach the client when engine control message (4) is set. - A negative range of the message number has been reserved with ORiN.
GetOptionValue	M	Acquire an option value that is specified by the option character string with specified type.	Option: BSTR Search parameter: BSTR Request type:	Value:VARIANT	n/a	·Format of option characetr string is “<parameter 1>[=<value1>][,<parameter2>[=<value2>]]”.

				VARTYPE			- When <value> is omitted, the function returns S_OK, but the value is not set. In this case, the type of return value is VT_EMPTY. - When the search parameter is not found, the function returns S_OK though the value is not set. The type of return value at this time is VT_EMPTY. - For request type, available data types are VT_I2, VT_I4, VT_R4, VT_R8, VT_BSTR, and VT_BOOL.
	SetTimerInterval	M	Setting of OnTimer() event interval	Interval: DWORD		n/a	･The unit is a millisecond. ･If 0 is set, the OnTimer event becomes invalid. The event becomes effective if any number other than 0 is set.
	m_bstrName	D	Controller name	Controller name: BSTR	n/a		
	m_bstrOption	D	Option	Option: BSTR	n/a		
	m_bTimer	D	Enable/Disable the OnTimer() event	Flag: BOOL	n/a	The OnTimer() event is not generated when making it to FALSE.	
	m_dwInterval	D	Interval of OnTimer() event	Interval: DWORD	n/a	The unit is a millisecond.	
	m_dwLocaleID	D	Locale ID [Registry]	Locale ID:DWORD	n/a	Value preserved in registry with CaoConfig etc.	
	m_szLicense	D	License [Registry]	License: TCHAR	n/a	Value preserved in registry with CaoConfig etc.	
	m_szParameter	D	Parameter [Registry]	Parameter: TCHAR	n/a	Value preserved in registry with CaoConfig etc.	
Meaning of sign	E:Event (HRESULT) M:Method (HRESULT) D:Data			･Arguments enclosed by square brackets can be omitted. ･The default value of the BSTR type's omittable argument is NULL. ･The default value of the numerical type's omittable argument is 0. ･The default value of the VARIANT type's omittable argument is VT_ERROR.			

◆ CaoProvExtensionImpl Template-Extension board

Class	Property, method, and event		Explanation	Argument of function		Called timing	Remarks
				IN	OUT RETVAL		
CaoProvExtensionImpl	FinalGetAttribute	E	Acquisition of attribute		Attribute: Long	When CaoProvExtension::get_Attribute() is called.	
	FinalGetHelp	E	Help		Help character string: BSTR	When CaoProvExtension::get_Help() is called.	
	FinalGetVariableNames	E	Acquisition of variable name list	[Option: BSTR]	Variable name list: VARIANT (VT_VARIANT|VT_ARRAY)	When CaoProvExtension::get_VariableNames() is called.	Available options are the filter condition and others..
	FinalExecute	E	Execution of expansion command	Command: BSTR [Parameter: VARIANT]	Result: VARIANT	When CaoProvExtension::Execute() is called.	For function expansion
	FinalInitialize	E	Preprocessing	Parent object: void		When CaoProvController::GetExtension() is called.	
	FinalTerminate	E	Postprocessing			When the object disappears.	
	GetOptionValue	M	Acquire an option value that is specified by the option character string with specified type.	Option: BSTR, Search parameter: BSTR Request type: VARTYPE	Value:VARIANT	n/a	Same as CaoProvControllerImpl::GetOptionValue().
	m_bstrName	D	Extension board name	Extension board name: BSTR	n/a		
	m_bstrOption	D	Option	Option: BSTR	n/a		
	m_bstrParent	D	Parent object name	Parent object name: BSTR	n/a		
Meaning of sign	E:Event (HRESULT) M:Method (HRESULT) D:Data			・Arguments enclosed by square brackets can be omitted. ・The default value of the BSTR type's omittable argument is NULL. ・The default value of the numerical type's omittable argument is 0. ・The default value of the VARIANT type's omittable argument is VT_ERROR.			

◆ CaoProvFileImpl Template-File

Class	Property, method, and event	Explanation	Argument of function		Called timing	Remarks
			IN	OUT RETVAL		
CaoProvFileImpl	FinalGetAttribute E	Acquisition of attribute		Attribute: Long	When CaoProvFile::get_Attribute() is called.	
	FinalGetDateCreated E	Date and time of creation		Date and time of creation: VARIANT	When CaoProvFile::get_DateCreated() is called.	
	FinalGetDateLastAccessed E	Last access date and time		Last access date and time: VARIANT	When CaoProvFile::get_DateLastAccessed() is called.	
	FinalGetDateLastModified E	The final change date		The final change date: VARIANT	When CaoProvFile::get_DateLastModified() is called.	
	FinalGetFileNames E	Acquisition of file name list	[Option: BSTR]	File name list: VARIANT (VT_VARIANT|VT_ARRAY)	When CaoProvFile::get_FileNames() is called.	Available options are the filter condition and others.When the attribute is a directory, the child file name list is returned.
	FinalGetHelp E	Help		Help character string: BSTR	When CaoProvFile::get_Help() is called.	
	FinalGetPath E	Acquisition of path		Path name: BSTR	When CaoProvFile::get_Path() is called.	
	FinalGetSize E	Acquisition of size of file		Size of file: Long	When CaoProvFile::get_Size() is called.	
	FinalGetType E	Acquisition of type of file		File type: BSTR	When CaoProvFile::get_Type() is called.	
	FinalGetValue E	Acquisition of content of file		Data: VARIANT	When CaoProvFile::get_Value() is called.	
	FinalPutValue E	Setting of content of file	Data: VARIANT		When CaoProvFile::put_Value() is called.	Available options are the filter condition and others.The type (VT_VARIANT|VT_ARRAY)
	FinalGetVariable E	Acquisition of variable	[Option: BSTR]	Variable name list:	When	

	Names		name list		VARIANT (VT_VARIANT|VT_ARRAY)	CaoProvFile::get_VariableNames() is called.	
	FinalCopy	E	Copy	Copy destination file name: BSTR [Option:BSTR]		When CaoProvFile::Copy() is called.	
	FinalDelete	E	Deletion	[Option: BSTR]		When CaoProvFile::Delete() is called.	
	FinalExecute	E	Execution of expansion command	Command: BSTR [Parameter: VARIANT]	Result: VARIANT	When CaoProvFile::Execute() is called.	
	FinalInitialize	E	Preprocessing	Parent object: void		When CaoProvController::GetFile() is called.	
	FinalMove	E	Move	Moving destination file name: BSTR [Option:BSTR]		When CaoProvFile::Move() is called.	
	FinalRun	E	Task creation	[Option: BSTR]	Task name: BSTR	When CaoProvFile::Run() is called.	
	FinalTerminate	E	Postprocessing			When the object disappears.	
	GetOptionValue	M	Acquire an option value that is specified by the option character string with specified type.	Option: BSTR Search parameter: BSTR Request type: VARTYPE	Value:VARIANT	n/a	Same as CaoProvControllerImpl::GetOptionValue().
	m_bstrName	D	File name	File name: BSTR	n/a		
	m_bstrOption	D	Option	Option: BSTR	n/a		
	m_bstrParent	D	Parent object name	Parent object name: BSTR	n/a		
Meaning of sign	E:Event (HRESULT) M:Method (HRESULT) D:Data			・Arguments enclosed by square brackets can be omitted. ・The default value of the BSTR type's omittable argument is NULL. ・The default value of the numerical type's omittable argument is 0. ・The default value of the VARIANT type's omittable argument is VT_ERROR.			

◆ CaoProvRobotImpl Template-Robot

Class	Property, method, and event		Explanation	Argument of function		Called timing	Remarks
				IN	OUT RETVAL		
CaoProvRobotImpl	FinalGetAttribute	E	Acquisition of attribute		Attribute: Long	When CaoProvRobot::get_Attribute() is called.	
	FinalGetHelp	E	Acquisition of help		Help character string: BSTR	When CaoProvRobot::get_Help() is called.	
	FinalGetVariableNames	E	Acquisition of variable name list	[Option: BSTR]	Variable name list: VARIANT (VT_VARIANT|VT_ARRAY)	When CaoProvRobot::get_VariableNames() is called.	Available options are the filter condition and others..
	FinalAccelerate	E	Refer to the specification of the ACCEL sentence of SLIM.	Axis number: Long Acceleration: Float [Deceleration:Float]		When CaoProvRobot::Accelerate() is called.	
	FinalChange	E	Refer to the specification of the CHANGE sentence of SLIM.	Hand name: BSTR		When CaoProvRobot::Change() is called.	
	FinalChuck	E	Refer to the specification of the GRASP sentence of SLIM.	[Option: BSTR]		When CaoProvRobot::Chuck() is called.	
	FinalDrive	E	Refer to the specification of the DRIVE sentence of SLIM.	Axis number: Long Distance: Float [Motion option:BSTR]		When CaoProvRobot::Drive() is called.	
	FinalExecute	E	Execution of expansion command	Command: BSTR [Parameter: VARIANT]	Result: VARIANT	When CaoProvRobot::Execute() is called.	
	FinalInitialize	E	Preprocessing	Parent object: void		When CaoProvController::GetRobot() is called.	
	FinalGoHome	E	Refer to the specification of the GOHOME sentence of			When CaoProvRobot::GoHome() is called.	

		SLIM.				
FinalHold	E	Refer to the specification of the HOLD sentence of SLIM.	[Option: BSTR]		When CaoProvRobot::Hold() is called.	In SLIM program, it means a temporary stop of the program. However, in CAO program, it means a temporary stop of the robot operation.
FinalHalt	E	Refer to the specification of the HALT sentence of SLIM.	[Option: BSTR]		When CaoProvRobot::Halt() is called.	In SLIM program, it means a mandatory stop of the program. However, in CAO program, it means a mandatory stop of the robot operation.
FinalMove	E	Refer to the specification of the MOVE sentence of SLIM.	Interpolation: Long Pose: VARIANT [Motion option: BSTR]		When CaoProvRobot::Move() is called.	
FinalRotate	E	Refer to the specification of the ROTATE sentence of SLIM.	Rotation surface: VARIANT Angle:Float Center of rotation; VARIANT [Motion option: BSTR]		When CaoProvRobot::Rotate() is called.	
FinalSpeed	E	Refer to the specification of the SPEED/JSPEED sentence of SLIM.	Axis number: Long Speed: Float		When CaoProvRobot::Speed() is called.	The axis numbers are; -1: Speed of the tool end, 0: Speed of all axes, Others; Speed of the specified axis.
FinalUnchuck	E	Refer to the specification of the RELEASE sentence of SLIM.	[Option: BSTR]		When CaoProvRobot::Unchuck() is called.	Changed to Chuck and Unchuck. In SLIM, “Release” is not available because it is a reserved word.
FinalUnhold	E	Release of HOLD sentence of SLIM	[Option: BSTR]		When CaoProvRobot::Unhold() is called.	In SLIM, since “Hold” is used to suspend a program, no resume command is prepared. In CAO, since “Hold” is used to suspend robot operations, any resume command is required. “Unhold” is used for that.
FinalTerminate	E	Postprocessing			When an object	

					dissapears.	
	GetOptionValue M	Acquire an option value that is specified by the option character string with specified type.	Option: BSTR Search parameter: BSTR Request type: VARTYPE	Value:VARIANT	n/a	Same as CaoProvControllerImpl::GetOptionValue().
	m_bstrName D	Robot name	Robot name: BSTR	n/a		
	m_bstrOption D	Option	Option: BSTR	n/a		
	m_bstrParent D	Parent object name	Parent object name: BSTR	n/a		
Meaning of sign	E:Event (HRESULT) M:Method (HRESULT) D:Data		・Arguments enclosed by square brackets can be omitted. ・The default value of the BSTR type's omittable argument is NULL. ・The default value of the numerical type's omittable argument is 0. ・The default value of the VARIANT type's omittable argument is VT_ERROR.			

◆ CaoProvTaskImpl Template-Task

Class	Property, method, and event		Explanation	Argument of function		Called timing	Remarks
				IN	OUT RETVAL		
CaoProvTaskImpl	FinalGetAttribute	E	Acquisition of attribute		Attribute: Long	When CaoProvTask::get_Attribute() is called.	
	FinalGetFileName	E	Acquisition of correspondence file name		File name: BSTR	When CaoProvTask::get_FileName is called.	
	FinalGetHelp	E	Help		Help character string: BSTR	When CaoProvTask::get_Help() is called.	
	FinalGetVariableNames	E	Acquisition of variable name list	[Option: BSTR]	Variable name list: VARIANT (VT_VARIANT|VT_ARRAY)	When CaoProvTask::get_VariableNames() is called.	Available options are the filter condition and others.
	FinalDelete	E	Deletion of task	[Option: BSTR]		When CaoProvTask::Delete() is called.	
	FinalExecute	E	Execution of expansion command	Command: BSTR [Parameter: VARIANT]	Result: VARIANT	When CaoProvTask::Execute() is called.	
	FinalInitialize	E	Preprocessing	Parent object: void		When CaoProvController::GetTask() is called.	
	FinalStart	E	Start of task	Mode: Long [Option: BSTR]		When CaoProvTask::Start() is called.	Available modes are; 1: Run one cycle, 2: Continuous operation, 3: One step forward, 4: One step back. Available options are the start position, etc.
	FinalStop	E	Stop of task	Mode: Long [Option: BSTR]		When CaoProvTask::Stop() is called.	Available modes are; 0: Default stop 1: Instantaneous stop 2: Step stop 3: Cycle stop 4: Initialization stop

						Note that the “Default stop” means one of the stop methods. (Step stop, Cycle stop, Initialization stop)
	FinalTerminate E	Postprocessing			When the object disappears.	
	GetOptionValue M	Acquire an option value that is specified by the option character string with specified type.	Option: BSTR Search parameter: BSTR Request type: VARTYPE	Value:VARIANT	n/a	Same as CaoProvControllerImpl::GetOptionValue().
	m_bstrName D	Task name	Task name: BSTR	n/a		
	m_bstrOption D	Option	Option: BSTR	n/a		
	m_bstrParent D	Parent object name	Parent object name: BSTR	n/a		
Meaning of sign	E:Event (HRESULT) M:Method (HRESULT) D:Data		・Arguments enclosed by square brackets can be omitted. ・The default value of the BSTR type’s omittable argument is NULL. ・The default value of the numerical type’s omittable argument is 0. ・The default value of the VARIANT type’s omittable argument is VT_ERROR.			

- CaoProvVariableImpl Template-Variable

Class	Property, method, and event		Explanation	Argument of function		Called timing	Remarks
				IN	OUT RETVAL		
CaoProvVariableImpl	FinalGetAttribute	E	Acquisition of attribute		Attribute: Long	When CaoProvVariable::get_Attribute() is called.	
	FinalGetDateTime	E	Acquisition of time stamp		Time stamp: VARIANT	When CaoProvVariable::get_DateTime() is called.	
	FinalGetHelp	E	Help		Help character string: BSTR	When CaoProvVariable::get_Help() is called.	Available options are the filter condition and others..
	FinalGetValue	E	Acquisition of value		Value:VARIANT	When CaoProvVariable::get_Value() is called.	
	FinalPutValue	E	Setting of value	Value:VARIANT		When CaoProvVariable::put_Value() is called.	
	FinalInitialize	E	Preprocessing	Parent object: void		When CaoProvController::GetVariable() is called.	
	FinalTerminate	E	Postprocessing			When the object disappears.	
	GetOptionValue	M	Acquire an option value that is specified by the option character string with specified type.	Option: BSTR Search parameter: BSTR Request type: VARTYPE	Value:VARIANT	n/a	Same as CaoProvControllerImpl::GetOptionValue().
	m_bstrName	D	Variable name	Variable name: BSTR	n/a		
	m_bstrOption	D	Option	Option: BSTR	n/a		
	m_bstrParent	D	Parent object name	Parent object name: BSTR	n/a		
Meaning of sign	E:Event (HRESULT) M:Method (HRESULT) D:Data			・Arguments enclosed by square brackets can be omitted. ・The default value of the BSTR type's omittable argument is NULL. ・The default value of the numerical type's omittable argument is 0. ・The default value of the VARIANT type's omittable argument is VT_ERROR.			

- CaoProvCommandImpl Template-Command

Class	Property, method, and event		Explanation	Argument of function		Called timing	Remarks
				IN	OUT RETVAL		
CaoProvCommandImpl	FinalGetAttribute	E	Acquisition of attribute		Attribute: Long	When CaoProvCommand::get_Attribute() is called.	
	FinalGetHelp	E	Help		Help character string: BSTR	When CaoProvCommand::get_Help() is called.	
	FinalGetParameters	E	Acquisition of command and parameter		Command and parameter: VARIANT	When CaoProvCommand::get_Parameters() is called.	
	FinalPutParameters	E	Setting of command and parameter	Command and parameter: VARIANT		When CaoProvCommand::put_Parameters() is called.	
	FinalGetState	E	Acquisition of state		State: Long	When CaoProvCommand::get_State() is called.	
	FinalGetTimeout	E	Acquisition of time-out		Time-out: Long	When CaoProvCommand::get_Timeout() is called.	
	FinalPutTimeout	E	Setting of time-out	Time-out: Long		When CaoProvCommand::put_Timeout() is called.	
	FinalCancel	E	Cancellation of command when being executing it	Command: Long	Result: VARIANT	When CaoProvCommand::Cancel() is called.	
	FinalExecute	E	Execution of command	Command: Long	Result: VARIANT	When CaoProvCommand::Execute() is called.	
	FinalInitialize	E	Preprocessing	Parent object: void		When CaoProvController::GetCommand() is called.	
	FinalTerminate	E	Postprocessing			When the object disappears.	
	GetOptionValue	M	Acquire an option value that is specified by the option character string	Option: BSTR, Search parameter: BSTR,	Value:VARIANT	n/a	Same as CaoProvControllerImpl::GetOptionValue().

			with specified type.	Request type: VARTYPE			
	m_bstrName	D	Command name	Command name: BSTR	n/a		
	m_bstrOption	D	Option	Option: BSTR	n/a		
	m_bstrParent	D	Parent object name	Parent object name: BSTR	n/a		
	m_vntParameters	D	Parameter	Parameter: VARIANT	n/a		
Meaning of sign	E:Event (HRESULT) M:Method (HRESULT) D:Data			・Arguments enclosed by square brackets can be omitted. ・The default value of the BSTR type's omittable argument is NULL. ・The default value of the numerical type's omittable argument is 0. ・The default value of the VARIANT type's omittable argument is VT_ERROR.			

- CaoProvMessageImpl Template-Message

Class	Property, method, and event		Explanation	Argument of function		Called timing	Remarks
				IN	OUT RETVAL		
CaoProvMessageImpl	FinalGetDateTime	E	Acquisition of explanation		Explanation: BSTR	When CaoMessage::get_Description() is called.	When this function is not overwritten, m_vntDateTime is returned automatically in Template.
	FinalGetDescription	E	Acquisition of explanation		Explanation: BSTR	When CaoMessage::get_Description() is called.	When this function is not overwritten, m_bstrDescription is returned automatically in Template.
	FinalGetDestination	E	Acquisition of destination address		Destination: BSTR	When CaoMessage::get_Destination() is called.	When this function is not overwritten, m_bstrDestination is returned automatically in Template.
	FinalGetNumber	E	Acquisition of message number		Message number: Long	When CaoMessage::get_Number() is called.	When this function is not overwritten, m_lNumber is returned automatically in Template.
	FinalGetSource	E	Acquisition of source		Source: BSTR	When CaoMessage::get_Source() is called.	When this function is not overwritten, m_bstrSource is returned automatically in Template.
	FinalGetValue	E	Acquisition of message text		Message text: VARIANT	When CaoMessage::get_Value() is called.	When this function is not overwritten, m_vntValue is returned automatically in Template.
	FinalClear	E	Clearness of message			When CaoMessage::Clear() is called.	
	FinalReply	E	Reply of message	Reply message: VARIANT		When CaoMessage::Reply() is called.	
	FinalInitialize	E	Preprocessing	Parent object: void		When CaoProvControllerImpl::FireOnMessage() is called.	
	FinalTerminate	E	Postprocessing			When the object disappears.	

	m_vntDateTime	D	Date and time of creation	Date and time of creation: VARIANT	n/a		
	m_bstrDescription	D	Explanation	Explanation: BSTR	n/a		
	m_bstrDestination	D	Destination address	Destination: BSTR	n/a		
	m_lNumber	D	Message number	Message number: Long	n/a		
	m_bstrParent	D	Parent object name	Parent object name: BSTR	n/a		
	m_bstrSource	D	Source	Source: BSTR	n/a		
	m_vntValue	D	Message text	Message text: VARIANT	n/a		
Meaning of sign	E:Event (HRESULT) M:Method (HRESULT) D:Data			・Arguments enclosed by square brackets can be omitted. ・The default value of the BSTR type's omittable argument is NULL. ・The default value of the numerical type's omittable argument is 0. ・The default value of the VARIANT type's omittable argument is VT_ERROR.			