CAO provider development guide

Version 1.2.1

September 14, 2021

[Remarks]

[Revision history]

Date	Version	Description
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.
2020-01-29	1.2.0	Changed to Visual Studio 2019 version. Delete Use of CaoProvExec tool.
2021-09-14	1.2.1	Changed the company name of Toshiba Machine to Shibaura Machine

Contents

1. Introduction

This guide provides specific examples of how to create CAO providers (hereinafter referred to as providers).

Chapter 2 explains the basics of what you need to implement in a provider to create a provider, and how to create a provider.

Chapter 3 explains the functions of the provider such as how to parse option strings and how to issue message events. In addition, the communication class for socket communication and serial communication is also described.

Chapter 4 describes specific provider implementation procedures, using the Shibaura Machine Small Programmable Controller TCmini provider as an example.

Chapter 5 introduces techniques that are useful to know when implementing providers.

In this document, the development environment is Windows 10 OS.

1.1. Provider's development language

Providers can use Microsoft Visual C++6.0, 2005, 2008, 2010, 2012, 2013, 2015, 2017, 2019, and Microsoft Visual C#. This manual uses Visual C++ 2019 (hereinafter referred to as "VC ++").

1.2. Visual C++ 2019 setting

To develop a provider with Visual C ++ 2019, Windows SDK 10.0.17763.0 or higher is required. Windows SDK is installed using Visual Studio Installer. Select Modify in Visual Studio 2019.

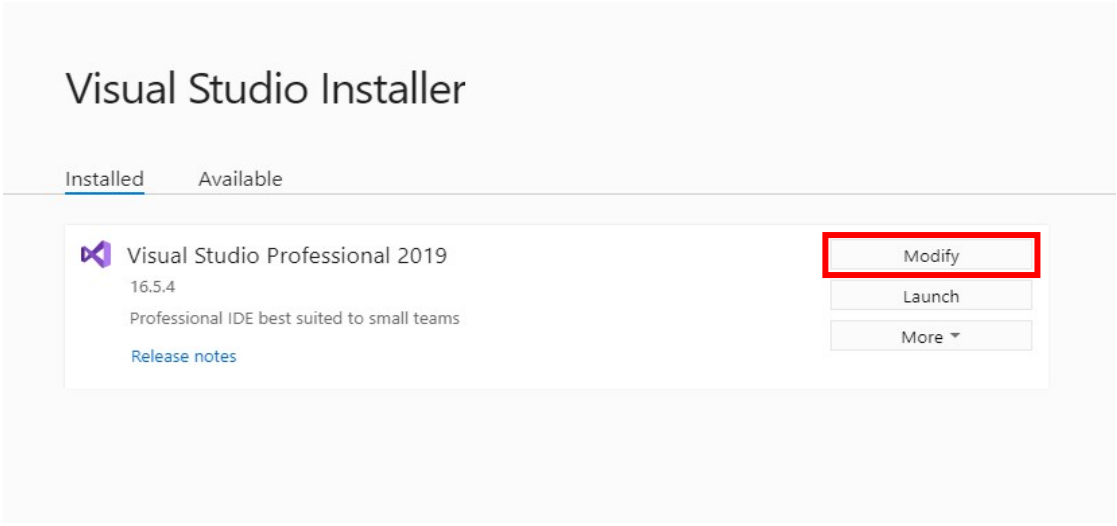


Fig. 1-1 Visual Studio Installer

Check "Desktop development with C ++" on the Workloads tab. Confirm that Windows 10 SDK (10.0.17763.0 or later) is checked in the installation details, and click the "Close" button to start the installation. Windows 10 SDK (10.0.18362.0) is installed on the screen below, but please install the latest version at that time.

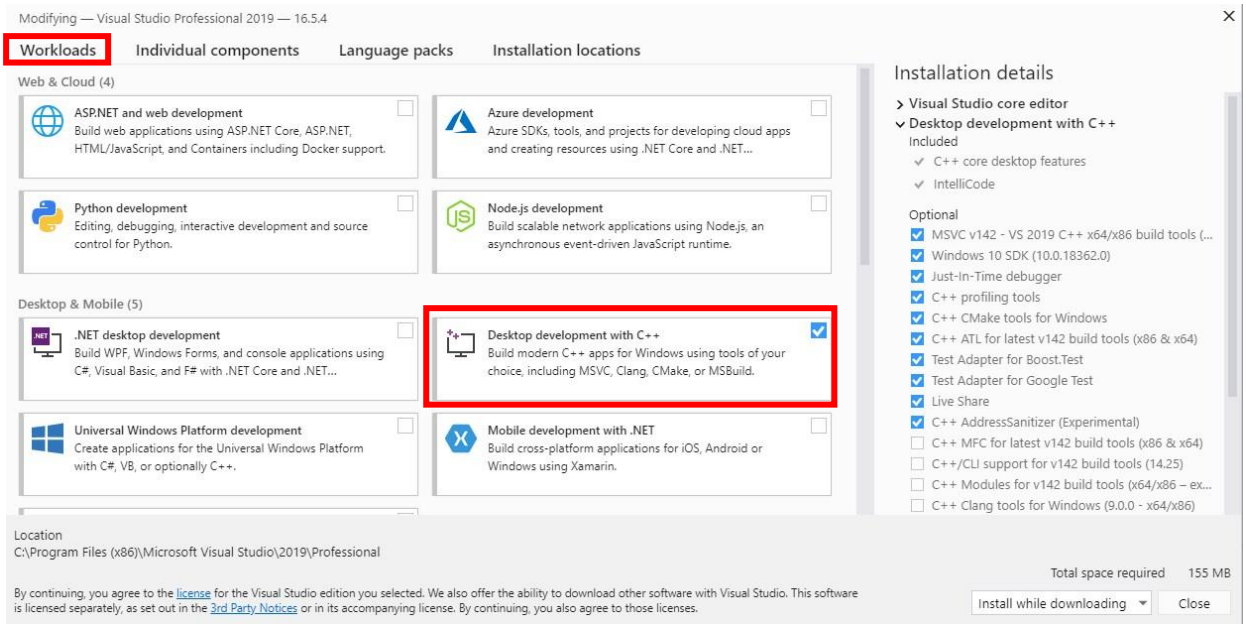


Fig. 1-2 Modifying screen of Visual Studio Installer

2. 2. Implementation procedure of CAO provider

2.1. Introduction

CAO (Controller Access Object) is an API (Application Program Interface) that provides shared access to various FA devices such as robots and databases. CAO is based on Distributed Object Technology (DCOM).

The CAO consists of an engine part that provides common functionality and a provider part that absorbs the differences between the manufacturers: the CAO engine provides the only interface to the client application, and the provider provides the only interface to the engine. This two-tiered structure eliminates the need for provider developers to implement the generic functionality provided by the Engine Department, and requires only the essential parts to access the information inside the controller. This is one of the major factors in making provider implementation easier.

In addition, providers are divided into C++ Template (CAO Provider Template) and Manufacturer Implementation Part as described in "7.2 List of Provider Template Functions". This template provides a common interface and default implementation for the CAO engine. This means that a user implementing a provider can simply override the functions in the template that correspond to the functions it wants to provide.

This chapter provides step-by-step instructions for implementing a provider.

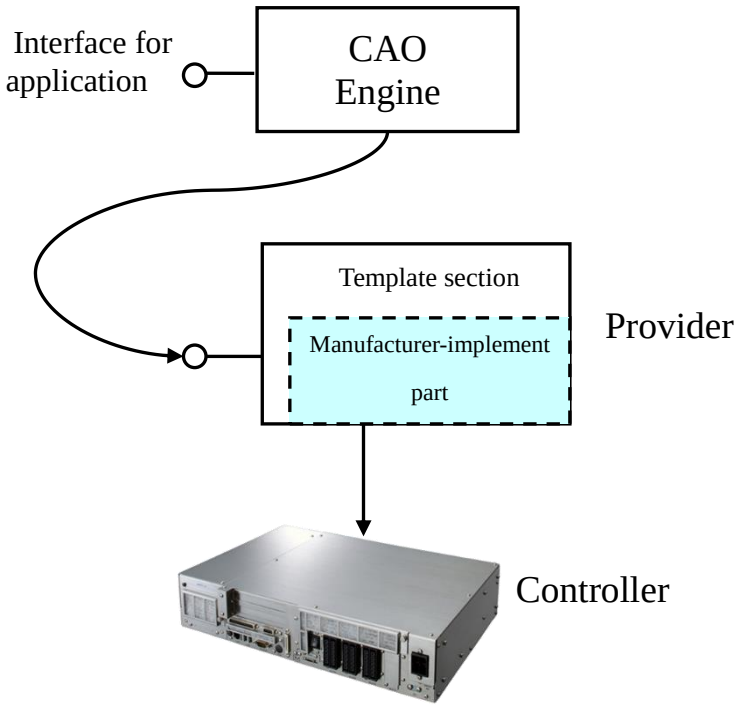


Fig. 2-1 Configuration of CAO

2.2. Implementation steps

The following is a high-level flow chart for creating a provider.

(1) Determine provider specifications

To implement a provider, you must first determine the provider's specifications. Depending on the target to be accessed, determine the following items, etc.

1. Provider's name
2. AddController() Connection Parameter Specifications
3. Class and method specifications to implement

(2) Creating a Template for a Provider

After determining the provider's specifications, the next step is to generate the provider's templates. To do this, use CaoProvWiz.exe.

(3) Implement the provider's required methods

Implement the required methods using the templates from the providers generated by CaoProvWiz.exe. In addition, classes for device communication are added as necessary.

(4) Provide a provider debug

Make sure that the provider you created works as intended. If it does not operate normally, perform debugging. Once all debugging has been done, it is the provider's release.

The following sections describe these items in detail.

2.3. Creating a New Provider Project

Projects for creating providers with Visual C ++ can be easily created using wizards. The following describes how to create a project.

(1) From the Start menu, select [ORiN2] → and then [CaoProvWizard].

(2) Select the location where you want to create the provider project.

E.g.) When creating in C:¥ORiN2¥CAO¥ProviderLib¥Sample¥Src

Fig. 2-2 Folder selection for CAO Provider output

(3) Enter the information of the provider to be created.

- DLL Name : DLL name (required)
- Vender Name : Vendor name (required)
- Module Name : Module name
- Project Type : Version of the VC++ project to generate
- Use MFC: Whether MFC is used

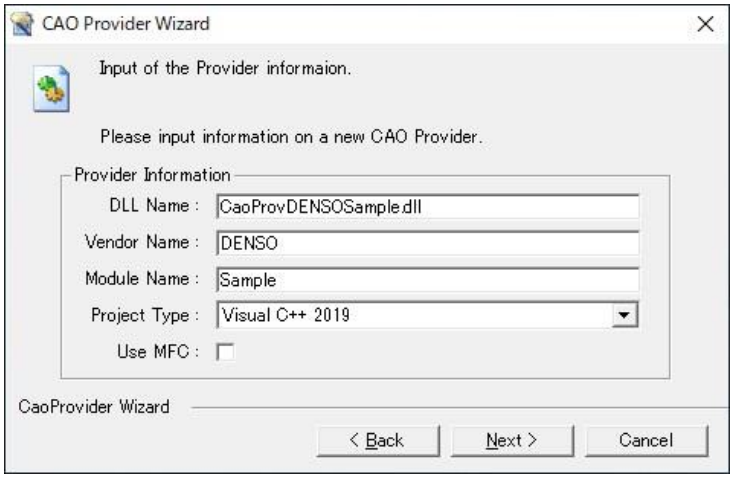


Fig. 2-3 Provider information input dialog

The <Vender Name>, <Module Name> entered here is used to determine ProgID names of COMs. ProgID of providers is "CaoProv.<Vender Name>. <Module Name>" is displayed.

(4) A confirmation screen will appear. Here, the provider's project begins to be created by pressing the "yes (Y)" button.

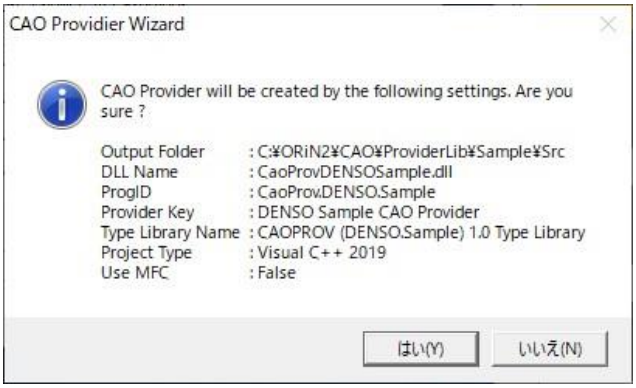


Fig. 2-4 Provider information setting confirmation dialog

(5) When the following screen appears, the project of the provider has been created.

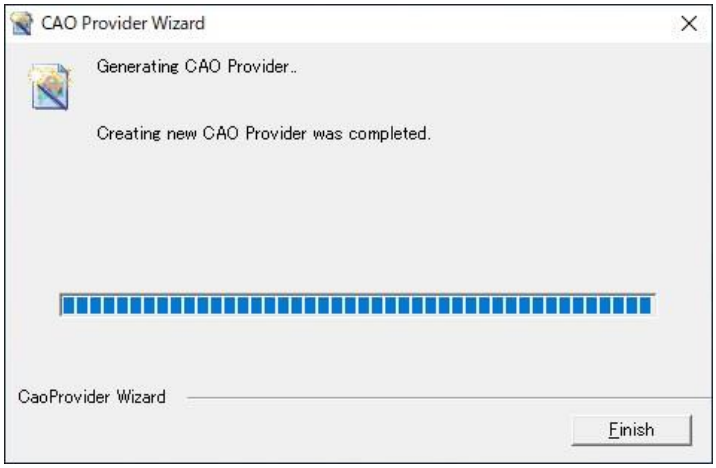


Fig. 2-5 Completion of creating project of provider dialog

A VC++ project for the provider is created in the specified directory. In addition to CaoProv.vcxproj, there are several other files in the created project folder. The table below lists the main files related to the provider. You do not need to be aware of the additional files not listed here, because they are managed by Visual Studio.

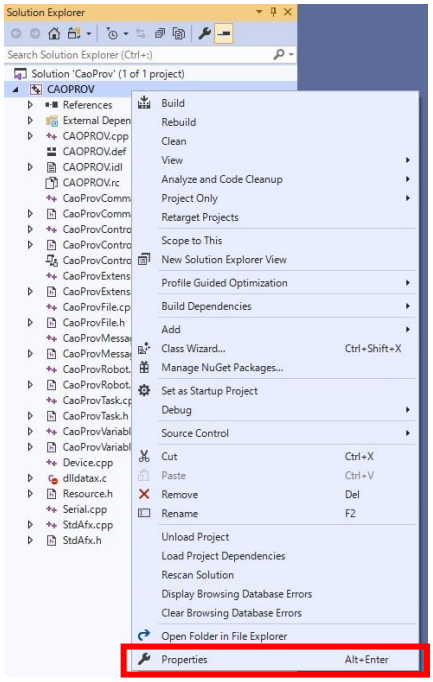
Table 2-1 Main project file list

No.	NAME	Type
1	CaoProv.dsw	Project Workspace (Visual C++ 6.0 only)
2	CaoProv.dsp	Project files (Visual C++ 6.0 only)
3	CaoProv.vcproj	Project file(Visual C++ 2005 , 2008)
4	CaoProv.vcxproj	Project files (Visual C++ 2010 to 2019)
5	CaoProv.IDL	IDL file
6	Resource.h	Resource header file
7	CaoProv.rc	Resource file
8	StdAfx.h	C header file
9	StdAfx.cpp	C++ source files
10	CaoProv.h	C header file
11	CaoProv.cpp	C++ source files
12	CaoProvController.h	Definition of C header file CCaoProvController class
13	CaoProvController.cpp	Implementation of C++ source file CCaoProvController class
14	CaoProvVariable.h	Definition of C header file CCaoProvVariable class
15	CaoProvVariable.cpp	Implementation of C++ source file CCaoProvVariable class
16	CaoProvTask.h	Definition of C header file CCaoProvTask class
17	CaoProvTask.cpp	Implementation of C++ source file CCaoProvTask class
18	CaoProvRobot.h	Definition of C header file CCaoProvRobot class
19	CaoProvRobot.cpp	Implementation of C++ source file CCaoProvRobot class
20	CaoProvFile.h	Definition of C header file CCaoProvFile class
21	CaoProvFile.cpp	Implementation of C++ source file CCaoProvFile class
22	CaoProvCommand.h	Definition of C header file CCaoProvCommand class
23	CaoProvCommand.cpp	Implementation of C++ source file CCaoProvCommand class
24	CaoProvExtension.h	Definition of C header file CCaoProvExtension class
25	CaoProvExtension.cpp	Implementation of C++ source file CCaoProvExtension class
26	CaoProvMessage.h	Definition of C header file CCaoProvMessage class
27	CaoProvMessage.cpp	Implementation of C++ source file CCaoProvMessage class

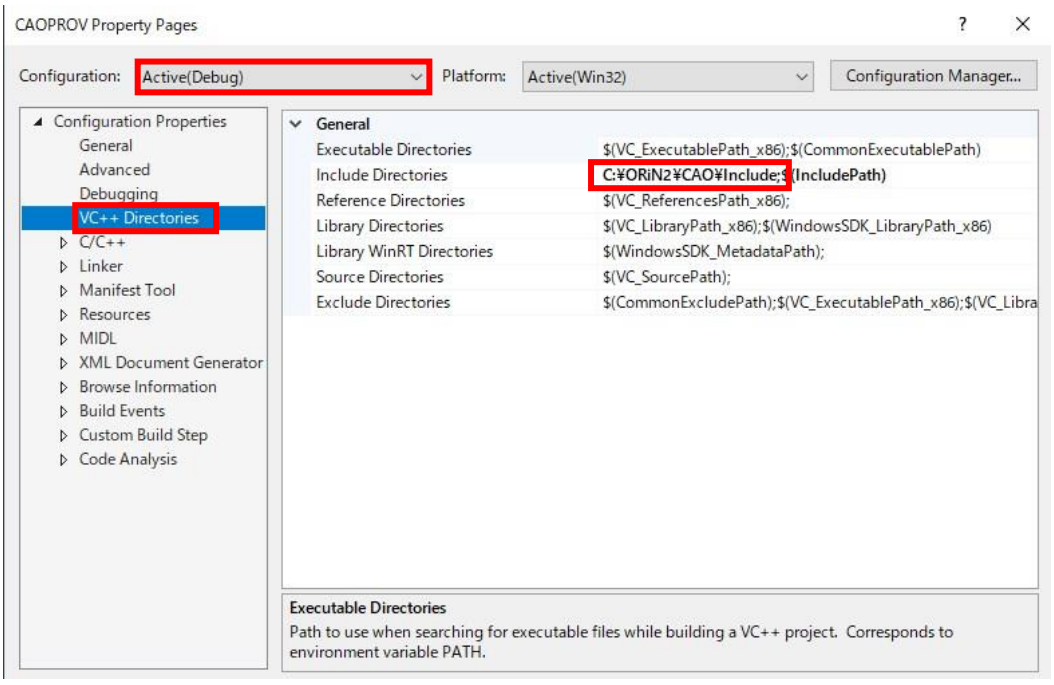
(6) When the project is complete, open CaoProv.vcxproj with Visual Studio 2019 (administrator rights). Be aware that if you do not have administrator privileges, the build will not pass.

(7) Set the include directory.

Select CAOPROV from the Solution Explorer and open Properties.

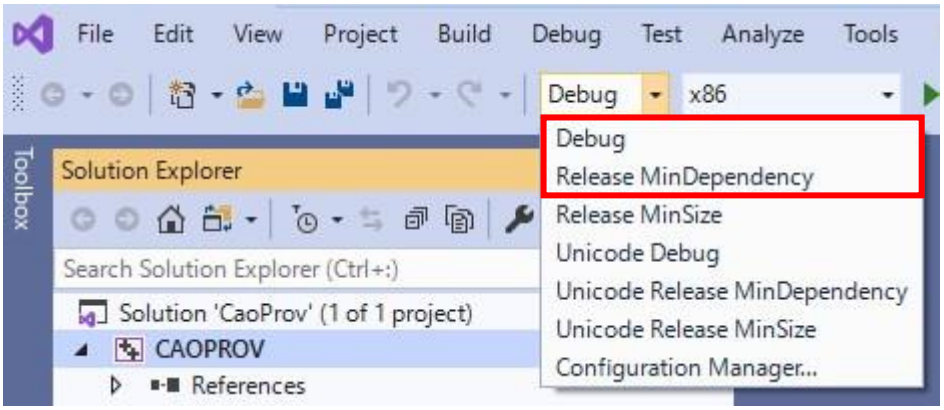


When you open the Properties page, select all configurations in Configuration, and add the <Installation folder>¥ORiN2¥CAO¥Include to the Configuration Properties →[VC++ Directories] →[Included Directories].



(8) You can build the provider (though it has no functionality) in the above operation.

Select "Debug" (debugging version) or "Release MinDependency" (release version) in the active solution configuration and perform the rebuild.



(9) Review the build artifacts.

If the build is successful, a bin folder will be created. The contents of the.Bin folder will differ between the files generated by "Debug" (debugging version) and "Release MinDependency" (release version). The "CaoProv <Vender Name><Module Name>.dll" in the.Bin folder will be the provider body.

<For Debug>

<For Release MinDependency>

2.4. Implementation of each class at CAO provider

2.4.1. Implementation preparation

2.4.1.1. Implementation preparation of classes

The definition and implementation of each class can be found in the following file:

Table 2-2 Files containing class definitions and implementations

Class name	Definitions	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 a provider, override the methods in that class. Prepare for method override by following the procedure below.

(1) Remove the comment of the method to be used from the methods described in the header file of each class.

Here is an example of how to use a method FinalInitialize for CCaoProvController classes: (CaoProvController.h)

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

(2) Remove the comment where the method is implemented.

The following are examples of how to implement a method FinalInitialize for CCaoProvController classes: (CaoProvController.cpp)

```
// Before removing comments
/* ← remove comment start symbol
HRESULT CCaoProvVariable::FinalInitialize()
{
        :
}
* / ← Remove comment end symbol
                ↓
// After removing the comment
HRESULT CCaoProvVariable::FinalInitialize()
{
        :
}
```

(3) The return value in method (2) is "S_OK". (If there is no return value, skip this task.)

```
return E_NOTIMPL; → return S_OK;
```

You are now ready to override the method.

Then, each method implements its own processing. You can also add variables and methods to the class if needed. Add variables and method definitions at the end of the added class definition. Add the implementation of

the additional method to the end of the cpp file that implements the method of the class to be added.

Only basic methods are described below. For a list of member variables and overrideable functions published in the provider template, see "7.2 List of Provider Template Functions".

2.4.1.2. Role of classes

Each class determines the type of functionality it provides. The following is an overview of the functions of each class.

Table 2-3 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.

There are some functions that must be overridden when implementing these classes. This is shown in Table 2-4.

Table 2-4 Required Methods in Each Class Implementation

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

Of these, the most significant classes are CaoProvController and must always be implemented. For other classes, there is no need to implement all the classes; it is up to the vendor's discretion. This means that you only

need to implement classes that correspond to the functionality you want to provide to your users.

Here are CaoController and CaoVariable classes: CCaoProvController::FinalInitialize(), CCaoProvController::FinalConnect(), CCaoProvController::FinalDisconnect(), and CCaoProvController: FinalTerminate(), This section describes CCaoProvVariable::FinalInitialize,CCaoProvVariable::FinalPutValue() and CCaoProvVariable::FinalGetValue().

2.4.2. CaoProvController classes

HRESULT CCaoProvController::FinalInitialize()

Called when CaoWorkspace::AddController process is called to initialize objects. Initialize variables used in CaoProvController class.

HRESULT CCaoProvController::FinalConnect()

This function is called at the timing when CaoWorkspace::AddController process is called, and is executed after FinalInitialize(). This method implements the connection processing with the communication device corresponding to the provider to be created. The connection method depends on the communication device. For example, if DLL to access the communication device is provided, DLL may be loaded. If RS-232C communication or TCP/IP communication is performed, connection processing should be implemented in this method.

HRESULT CCaoProvController::FinalDisconnect()

The provider is disconnected.

Called when CaoControllers::Remove process is called. This method disconnects from the communication device connected by FinalConnect(). The disconnection method depends on the communication device.

HRESULT CCaoProvController::FinalTerminate()

Performs pre-release processing of the object.

This function is called at the timing when CaoControllers::Remove process is called, and is executed after FinalDisconnect(). This method performs processing such as closing variables such as events generated by initialization.

2.4.3. CaoProvVariable classes

HRESULT CCaoProvVariable::FinalInitialize()

This method is called when a CaoVariable object is created, and initializes the object.

pObj argument is a pointer to a parent object. CaoVariable is derived from several objects, such as CaoController and CaoRobot, so FinalInitialize() can separate the initialization process for each parent object if needed. The parent object can be determined by the member variable m_ulParentType. The type of m_ulParentType is:

Table 2-5 Parent objects of table 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

Also, this function should parse the variable name and assign a different identifier. There are two types of variable names: System variable and User variable. System variable is fixed character strings (for example, "@MAKER_NAME"), and User variable is arbitrary character strings. The convention for System variable is to use an '@' at the beginning of the string.

You can add System variables for a controller class as follows: This example adds "@TEMP_DATA". The following procedure assigns a different identifier to the member variable m_lUSysId, which holds the identifier of the system variable.

<How to add a system variable>

(1) Define the macros required for StdAfx.h.

```
#define CS_MAKER_NAME     0x0003              ← Assign a unique number
#define CS_MAKER_NAME$    L"@TEMP_DATA"     ← Assign a unique name
```

(2) Adding Macros to Name Management Map Initialization

```
HRESULT CCaoProvVariable::InitMapTable()
{
  :
  // Initializing the variable name map
  Const var_map_entry var_cs_map[] = {

      // :
      MAP_ENTRY(CS_TEMP_DATA), ← Add macro
  };
   :
  return S_OK;
}
```

(3) Assign an identifier (no change from template)

```
HRESULT CCaoProvVariable::FinalInitialize(PVOID pObj)
{
  :
  switch (m_ulParentType) {
  case SYS_CLS_CONTROLLER:
      pCaopCtrl = (CCaoProvController*)pObj;
      if (m_bSystem) {  // System variable
                              // システム変数
          // ID search from variable name
          // 変数名からID検索
```

```
            var_map::iterator it;
            it = m_cs_map.find(m_bstrName);
            if (it != m_cs_map.end()){
                m_lUSysId = (it->second);
                hr = S_OK;
            } else {
                hr = E_INVALIDARG;
            }
        :
        }
    return hr;
}
```

For details on assigning identifiers to User variable, see "4.3.4.5 Implementation of FinalInitialize()".

HRESULT CCaoProvVariable::FinalGetValue(VARIANT *pVal)

Gets the value of a variable. Arguments are VARIANT pointers pVal. You can pass a value to the client by populating this pVal with the value of the retrieval result. The template describes only the implementation if the parent object is a controller class. Executes FinalGetCtrlSysValue function for System variable and FinalGetUserValue function for User variable. When mounting, write it in FinalGetCtrlSysValue or FinalGetUserValue.

```
HRESULT CCaoProvVariable::FinalGetValue(VARIANT *pVal)
{
  HRESULT hr = E_ACCESSDENIED;

  if (m_bSystem) {
      switch (m_ulParentType) {
      case SYS_CLS_CONTROLLER:
          hr = FinalGetCtrlSysValue(pVal);
          break;
      }
  }
  else {
      switch (m_ulParentType) {
      case SYS_CLS_CONTROLLER:
          hr = FinalGetCtrlUserValue(pVal);
          break;
      }
  }

  return hr;
}
```

Variables are acquired from communication devices as needed. See also ORiN2 Programming Guide for information on assigning to a VARIANT.

The following example implements @MAKER_NAME, the system variable for the controller class. (This is already implemented in the default template.) This example accesses the controller to get the type of the VT_BSTR type system variable and assigns the value to pVal. Actually, please implement according to the access method to the controller.

```
HRESULT CCaoProvVariable::FinalGetCtrlSysValue(VARIANT *pVal)
{
  switch (m_lUSysId) {
       :
  case CS_TEMP_DATA:
       pVal->vt = VT_BSTR;
       pVal->bstrVal = SysAllocString(L"DENSO"); ←  Add manufacturer name
       break;
  }
   :
  return hr;
}
```

HRESULT CCaoProvVariable::FinalPutValue(VARIANT newVal)

NewVal of arguments for FinalPutValue functions is of type VARIANT. Contains the value entered by the client. The template describes only the implementation if the parent object is a controller class. Executes FinalPutCtrlSysValue function for System variable and FinalPutUserValue function for User variable. When mounting, write it in FinalPutCtrlSysValue or FinalPutUserValue.

If VARIANT type has no entity such as VT_BSTR, VT_ARRAY, or VT_BYREF, the value cannot be referenced if the scope of this FinalPutValue method is out of scope. If you want to retain the value even when it is out of scope, allocate memory separately.

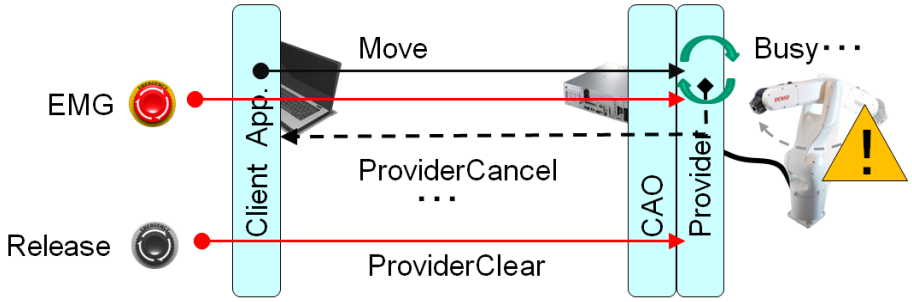
```
HRESULT CCaoProvVariable::FinalPutValue(VARIANT newVal)
{
     HRESULT hr = E_ACCESSDENIED;

     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;
}
```

2.4.4. Cancellation of running process

In situations where an emergency shutdown is required, the provider should immediately cancel any in-progress processing so that processing can be returned to the upper-level client CAO application. Doing so allows the higher-level application to take the necessary action to stop. Responding to this request requires canceling the current operation in response to the two provider-defined request commands (ProviderCancel and

ProviderClear).



These request commands are defined as CCaoProvController::Execute() system-reserved commands as follows:

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

When ProverCancel command is issued, ExecProviderCancel() method and ExecProviderClear() method for ProviderClear are called asynchronously from CAO.exe.

The client application requests ProviderCancel command asynchronously to CAO.exe if the provider needs to immediately cancel what is running. To complete the cancellation process and execute the normal command again, issue ProviderClear command to clear the cancellation request.

2.4.4.1. Implementing of ProviderCancel command

In projects created in ProviderWizard, CCaoProvController::ExecProviderCancel() method corresponding to ProviderCancel commands is implemented as follows.

List 2-1 CCaoProvController.cpp – ExecProviderCancel()

```
HRESULT CCaoProvController::ExecProviderCancel(VARIANT vntParam, VARIANT *pVal)
{
  ATLASSERT(m_hProviderCancelEvent != NULL);

  // Set provider cancel event
  // プロバイダキャンセルイベントをセットする
  ::SetEvent(m_hProviderCancelEvent);

  // TODO: Implement the process that interrupts the running process.
  // TODO: 実行中の処理を中断する処理を実装すること.
  // :
  return S_OK;
}
```

Here we can see that m_hProviderCancelEvent is defined as a HANDLE type in the member variables of CCaoProvController class, and is implemented by SetEvent to be in signal status. This HANDLE can be monitored in an asynchronous running loop as shown below, and cancellation can be accomplished in a signaled state by exiting the loop. If any processing should be cleared at this time, the implementation is added.

```
for(int i=0; i<1000; i++)  {
  // Processing something...
  //  :
  if(::WaitForSingleObject(m_hProviderCancelEvent, 0) == WAIT_OBJECT_0) {
       // Cancellation processProcessing cancel
       break;
  }
}
```

2.4.4.2. Implementing of ProviderClear command

In projects created in ProviderWizard, CCaoProvController::ExecProviderClear() method corresponding to ProviderClear commands is implemented as follows.

List 2-2 CCaoProvController.cpp – ExecProviderClear()

```
HRESULT CCaoProvController::ExecProviderClear(VARIANT vntParam, VARIANT *pVal)
{
  ATLASSERT(m_hProviderCancelEvent != NULL);

  // ReSet provider cancel event
  // プロバイダキャンセルイベントをリセットする
  ::ResetEvent(m_hProviderCancelEvent);

  // TODO: Implement processing to execute normally.
  // TODO: 正常に実行するための処理を実装すること.
  // :

  return S_OK;
}
```

In this example, the cancellation process is not executed again by resetting the m_hProviderCancelEvent which is in the signaled state with ResetEvent and returning it to the normal state. If any processing should be cleared at this time, the implementation is added.

2.4.5. Setting of registry information

You can set the default values of the data stored in the registry in CaoProvController.rgs file.

List 2-3 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 ''
        }
    }
}
```

Only edit the grayed-out part of the above file. Otherwise, the provider may 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 above settings can be changed using CaoConfig.

2.5. Debug and release of CAO provider

2.5.1. Debug of providers

In CAO, provider DLL modules are structured as needed from CAO engine CAO.exe. Therefore, when debugging a provider DLL module, you must specify CAO.exe in the executable file in the VC++ debugging section.[1]

The other steps are the same as for normal VC++ project debugging. As a client application, you can use ORiN CAO testing tool <Installation folder>¥ORiN2¥CAO¥Tools¥CaoTester2¥Bin¥CaoTester2.exe, etc. as a target.

An example of debugging is shown below.

(1) Select [Debug_x86] as the build target.

[1] If you want the provider DLLs to be launched in an out process, specify dllhost.exe in the Debug Session Executable. You must specify the machine name as the second parameter to AddController method.

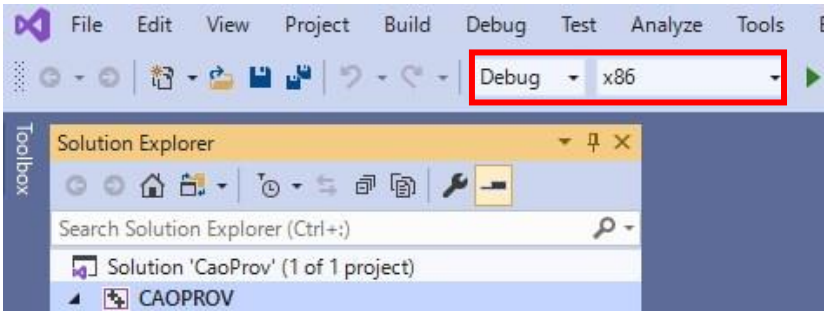


Fig. 2-6 Build target specification screen

(2) Specify CAO.exe in the executable file of the debug section.

Open CAOPROV properties page, Specify Debug in [Configuration] and Win32 in [Platform]. Then, specify CAO.exe with absolute path in [Configuration Properties]→[Debug]→[Command]. CAO.exe is usually located in <Installation folder>¥ORiN2¥CAO¥Engine¥Bin folder.

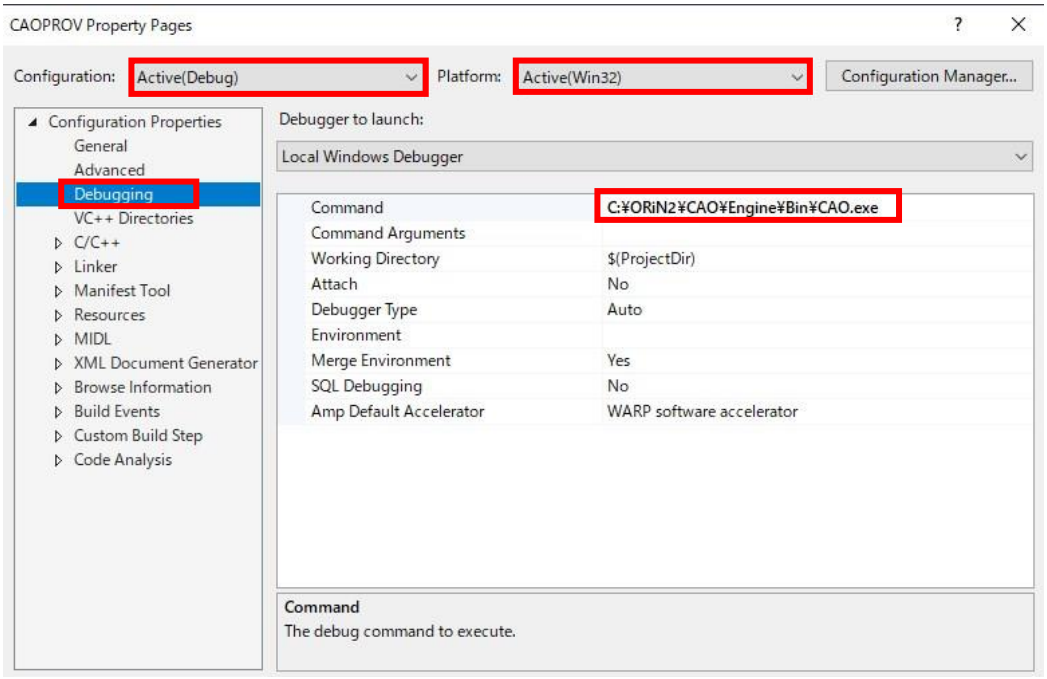


Fig. 2-7 Specified screen of CAO.exe

(3) Set a breakpoint at the desired location.

(4) Start debugging.

Execute [Start Debugging] from the [Debug] menu.

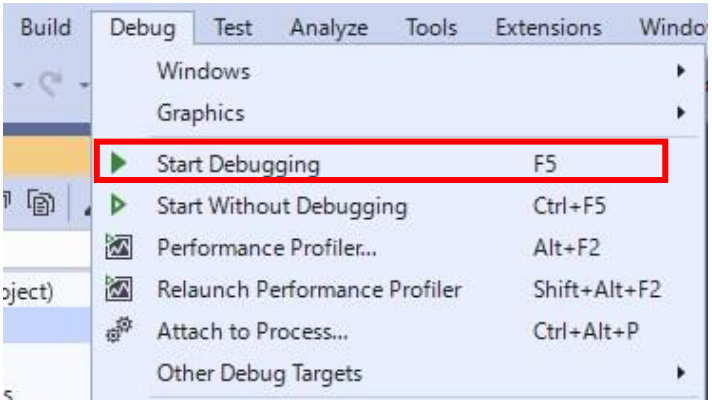


Fig. 2-8 Debug start screen

(5) Start CaoTester2.exe and specify the provider you want to debug.

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

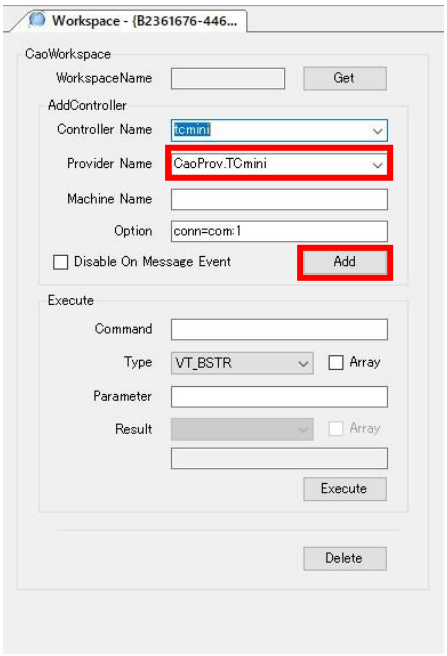


Fig. 2-9 CaoTester2.exe

(7) Call the provider by running the services of the CAO in CaoTester2.

For example, to debug CaoVariable objects, select Variable tab, enter the desired name Name AddVariable, and click Add. After that, please put or get Value.

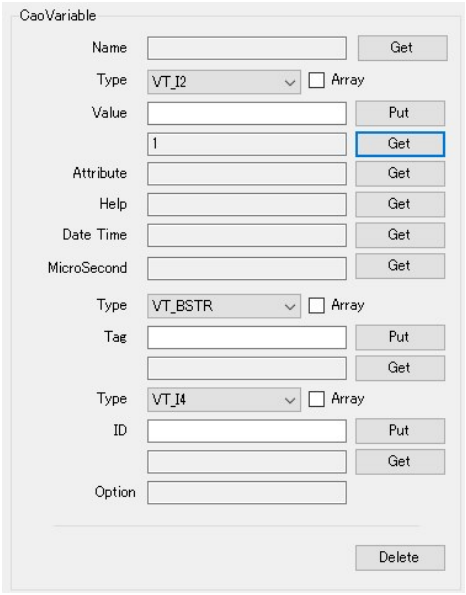


Fig. 2-10 System variable specification screen

(8) Return to VC++. If it is stopped at the break pointer position, debugging is executed using VC++.

2.5.2. Release of the providers

2.5.2.1. Creating documents

If the provider's specifications are not publicly available, the user cannot easily use the provider. Therefore, it is necessary to create and publish the specifications of the provider.

The provider's specification should include the lowest-level description.

(1) AddController() connection parameter specifications
(2) Specifications for User variable
(3) List of System variable and their meanings
(4) Other information considered important (information on provider-specific functions, precautions, etc.)

There are no rules regarding how to write (format) specifications, so please create it freely.

2.6. Distribution of providers

2.6.1. Confirmation of dependence information

The body CAO.exe of the CAO engine does not require any special dependent modules (such as DLLs). This is because stable operation is considered regardless of the environment. It is also desirable to create providers so that they do not depend on other modules as much as possible. Avoid using special libraries to avoid dependencies on other modules. Alternatively, you must take measures such as statically linking libraries.

If the provider you created depends on other modules, you should check which modules are required for operation. Use a Dependency Walker or a Process Explorer to find out what your dependencies are.

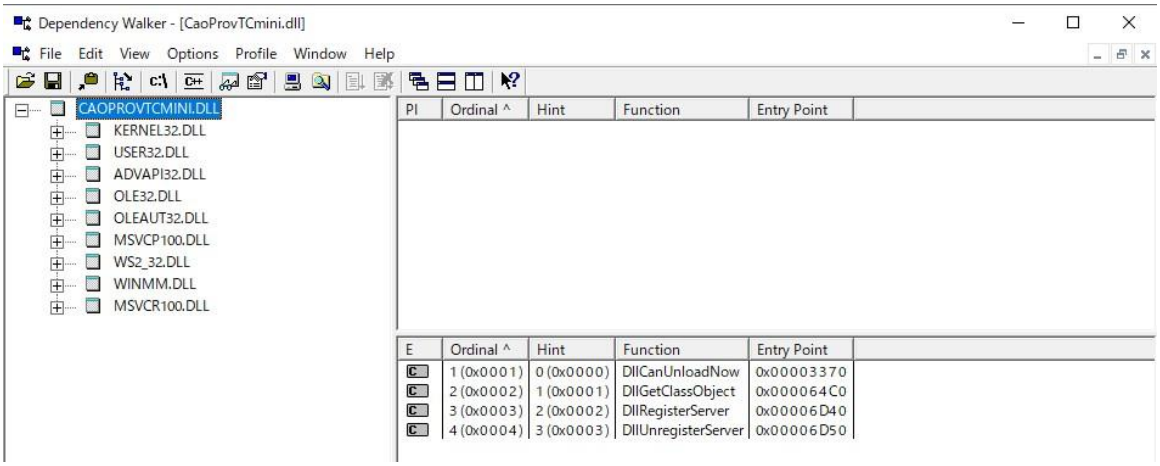


Fig. 2-11 Dependency Walker screen

2.6.2. How to register providers

If you want to use the provider in the distribution destination, you must register the provider's DLL in the registry. To register in the registry, use regsvr32 command at the command prompt. You can register in the registry by entering the provider DLL name after regvr32 command and executing the provider DLL name. Execute the command in the administrator mode when starting the command prompt.

E.g) Registering C:¥ORiN2¥CAO¥ ProviderLib¥Sample¥Bin¥CaoProvDENSOSample. dll

```
C:¥>regsvr32 "C:¥ORiN2¥CAO¥ProviderLib¥Sample¥Bin¥CaoProvDENSOSample.dll"
```

2.6.3. How to unregister providers

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

E.g.) Unregistering C:¥ORiN2¥CAO¥ ProviderLib¥Sample¥CaoProvDENSOSample. dll

```
C:¥>regsvr32 /u "C:¥ORiN2¥CAO¥ProviderLib¥Sample¥Bin¥CaoProvDENSOSample.dll"
```

2.6.4. Installation situation of ORiN2 SDK

For details about how to check ORiN2 SDK installation status, see "3.7. How to check the installation status of ORiN2 SDK " in ORiN2SDK_UsersGuide_en.pdf.

3. Useful functions that provider template library offers

3.1. Introduction

ORiN2 SDK provides a number of useful features for providers, such as methods that parse option strings and methods that publish message events. In addition, classes for device communication are used by some providers.

This chapter explains how to implement these features.

3.2. Parsing of option string

Providers can use GetOptionValue method to get option string that are used as method arguments. GetOptionValue() argument specifications are shown below. For more information, see OptionValue. h in the <Installation folder>¥ORiN2¥CAO¥Include.

```
GetOptionValue
(
   "<Option string>",
   "<Search string>",
   "<Request type>",
   "<Result value>"
)
```

Option strings follow this format:

<Option name 1> = <Option value1> , <Option name 2> = <Option value 2>...

The following is a simple example: Here, the option string is "Opt1=test,Opt2=sample", and the search option name is "Opt1". As a result, vntOptVal.bstrVal can extract the text "test". Also, you can extract the text "sample" by selecting "Opt2" as the search option. If GetOptionValue is used to extract consecutive BSTR strings, VariantClear functions must be used to release the strings in order to avoid memory leaks, as follows:

```
HRESULT hr;
VARIANT vntOptVal;
hr = GetOptionValue(CComBSTR(L"Opt1=test,Opt2=sample"), CComBSTR(L"Opt1"), VT_BSTR,
                                                                    &vntOptVal);
// vntOptVal.bstrVal == "test"
VariantClear(&vntOptVal);      // Free BSTR string

hr = GetOptionValue(CComBSTR(L"Opt1=test,Opt2=sample"), CComBSTR(L"Opt2"), VT_BSTR,
                                                                    &vntOptVal);
// vntOptVal.bstrVal == "sample"
VariantClear(&vntOptVal);      // Free BSTR string
```

In addition, you can use enclosing characters in option strings. The following can be used as an enclosing character:

- Parentheses (" ()"))

- Curly braces ("{}")
- Square brackets ("[]")
- Square brackets ("< >")

The first occurrence of any of these enclosing characters is treated as the enclosing character, and subsequent occurrences of the parentheses are treated as ordinary symbols.

Here is an example when there are multiple options:

Option string: Test1=Sample1,Sample2,Test2=Sample3

Table 3-1 Result of option string (e.g.1)

Option name	Option value
Test1	Sample1
Test2	Sample3

Here is an example of when the option value is in parentheses:

Option string: Test1=(Sample1,Sample2), Test2=(Sample3)

Table 3-2 Result of option string (e.g.2)

Option name	Option value
Test1	Sample1,Sample2
Test2	Sample3

Here is an example of when option values are in different parentheses:

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

Table 3-3 Result of option string (e.g.3)

Option name	Option value
Test1	Sample1
Test2	<Sample2>

Here is an example of an option value with additional parentheses inside the parentheses:

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

Table 3-4 Result of option string (e.g.4)

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

3.3. Parsing of connection parameters

Providers have CConnectOption classes that parse option strings for use as arguments to AddController methods in a predefined format and get their values. You can use this class to unify the connection parameters as you would any other provider. For more information, see ConnectOption.h in the <Installation folder>¥ORiN2¥CAO¥Include.

Connection parameters handled by CConnectionOption classes follow one of the following formats: Refer to Table 3-5 and Table 3-6 for the parameter value range.

Conn = ETH:<destination IP>[:<destination port>[:<local IP>[:<local port>]]
= TCP:<destination IP>[:<destination port>[:<local IP>[:<local port>]]
= UDP:<destination IP>[:<destination port>[:<local IP>[:<local port>]]
= COM:<COM number>[:<baud rate>[:<parity>:<data bit>:<stop bit>:<flow control>]]

Table 3-5 Tables Parameters of Ethernet (ETH, TCP, UDP)

Parameter	Value range	Default Value
Destination IP(IPv4)	0.0.0.0 ~ 255.255.255.255	-
Destination port	0 ~ 65535	5001
Local IP (IPv4)	0.0.0.0 ~ 255.255.255.255	127.0.0.1
Local port	0 ~ 65535	0 (0: free ports are assigned)

Table 3-6 Serial Communication (COM) Parameters

Parameter	Value range	Default Value
COM number	1 ~ 256	-
Baud rate [bps]	1 ~ 4,294,967,295	38400
Parity	0 ~ 4 (0:None, 1:Odd, 2:Even, 3:Mark, 4:Space)	0:None
Data bit [bit]	4 ~ 8	8
Stop bit [bit]	0 ~ 2	1
Flow Control	0 ~ 3 (0 : No flow control 1 : -Xon/Xoff	0 : No flow control

	2 : hardware control (OR operation possible)	

Square brackets ("[]") mean optional characters. The following four descriptions are possible for the ETH option. If omitted, the default value is used.

例1) Destination IP:192.168.0.1

→ Conn=ETH:192.168.0.1

例2) Destination IP:192.168.0.1, Destination port: 8080

→ Conn=ETH:192.168.0.1:8080

例3) Destination IP:192.168.0.1, Destination port: 8080, Local IP:192.168.0.2

Conn=ETH:192.168.0.1:8080:192.168.0.2

例4) Destination IP:192.168.0.1, Destination port: 8080, Local IP:192.168.0.2, Local port: 80

Conn=ETH:192.168.0.1:8080:192.168.0.2:80

You can also use SetDefault method of CConnectOption classes to change the defaults. The following three patterns are available:

```
/* Set defaultparameters for Ethernet */
HRESULT SetDefault(PARAM_CONN_ETH stEth)

/* Set default parameters for serial communication */
HRESULT SetDefault(PARAM_CONN_COM stCom)

/* Set defaultparameters for Ethernet, serial communication */
HRESULT SetDefault(PARAM_CONN_COM stCom, PARAM_CONN_ETH stEth)


// Parameter structures for Ethernet
struct PARAM_CONN_ETH {
    DWORD dwSrcIP;        // Local IP address inet_addr() format
    DWORD dwSrcPort;      // Local Port Number
    DWORD dwDestIP;       // Destination IP address inet_addr() format
    DWORD dwDestPort;     // Destination port number
};

// Parameter structure for serial communication
struct PARAM_CONN_COM {
    DWORD dwPortNo;       // COM port number
    DWORD dwBaudRate;     // Baud rate
    DWORD dwParity;       // Parity
    DWORD dwDataBits;     // Number of data bits
    DWORD dwStopBits;     // No. of Stop Bits
    DWORD dwFlow;         // Flow Control
};
```

Use GetConnectOption method to parse option string.

```
/* Connection Options Analysis */
HRESULT GetConnectOption(BSTR bstrSorce,          // Character string to be analyzed
                         PARAM_CONN* pRet)        // Storage location of analysis results


// Parameter union for Ethernet and serial communication
struct PARAM_CONN {
    WORD wType;       // Parameter Type
                      // ETH:TYPE_ETH, TCP:TYPE_TCP, UDP:TYPE_UDP, COM:TYPE_COM
    union {
        PARAM_CONN_ETH stEth;          // Parameter structures for Ethernet
        PARAM_CONN_COM stCom;          // Storage location of analysis results
    };
};
```

3.3.1. How to use CConnectOption class

The following is an example of using CConnectOption classes. The following shows the result of parsing the option string "Conn=COM:2:4800" with the baud rate: 19200 bps, parity: None, data bit: 8 bit, stop bit: 2 bit, flow control: 0 (no control) as the default.

```
PARAM_CONN_COM stComIniValue;
stComIniValue.dwBaudRate = CBR_19200;       // #define CBR_19200  19200
stComIniValue.dwParity = NOPARITY;          // #define NOPARITY   0
stComIniValue.dwDataBits = 8;
stComIniValue.dwStopBits = TWOSTOPBITS;     // #define TWOSTOPBITS  2
stComIniValue.dwFlow = 0;

CConnectOption connectOption;
connectOption.SetDefault(stComIniValue);

PARAM_CONN connParam
HRESULT hr = connectOption.GetConnectOption(m_bstrOption, &connParam);

// Parsing results
// connParam.stCom.wType == TYPE_COM
// connParam.stCom.dwPortNo ==  2
// connParam.stCom.dwBaudRate == 4800;
// connParam.stCom.dwParity == 0;
// connParam.stCom.dwDataBits == 8;
// connParam.stCom.dwStopBits == 2;
// connParam.stCom.dwFlow == 0;
```

3.4. VARIANT type conversion

The provider accepts the value input by the user as a VARIANT type. VARIANT type can hold values of various types such as VT_I1 type and VT_BSTR type. Since it may not be possible to process with VARIANT type as it is, CDataConv class that has the function to convert from VARIANT type to any type is provided in the provider library. A provider that uses this function does not require strict type specification when the user inputs arguments, and can implement a more manageable provider. Please refer to DataConv.cpp / h in <Installation folder>¥ORiN2¥CAO¥Include for details of the implementation.

When converting VARIANT type, use ChangeVarType method of CDataConv class. It is necessary to specify the VARIANT data vntSrc of the conversion target, the conversion type vt, and the storage destination * pRet after conversion as arguments. The maximum number of elements after conversion lSize and the locale ID ulLocaleID can be specified optionally. An array (VT_ARRAY type) can also be specified in the conversion source VARIANT data vntSrc, and each array element will be converted to the required conversion type vt. As the return value, the number of elements that have been converted is returned in long type.

```
/* Convert to VARIANT type */
// @retval    long :   Number of elements that have been converted
long CDataConv::ChangeVarType(VARIANT vntSrc,         // Conversion target
                              VARTYPE vt,             // Conversion type
                              void *pRet,        // Storage destination after conversion
                              long lSize /*= NO_CHECK_SIZE*/,     //  Maximum  number  of
                              elements after conversion
                              unsigned long ulLocaleID /*= 0*/)   // Locale ID
```

3.4.1. How to use CDataConv class

Consider the case where the user requires the VT_I2 type at the time of PutValue of the Variable class. Here is an example when CDataConv is not used and when it is used.

<When CDataConv class is not used>

After executing PutValue of Variable class, CCaoProvVariable::FinalPutCtrlUserValue function (details will be described later) is called. Argument of FinalPutCtrlUserValue set by the user is passed to the newVal of VARIANT type. This time, VT_I2 type is requested as an argument, and if CDataConv class is not used, the implementation will be as follows.

```
HRESULT CCaoProvVariable::FinalPutCtrlUserValue(VARIANT newVal)
{
    if(newVal.vt != VT_I2){          // Error if newVal is not VT_I2 type
        return E_INVALIDARG;
    }
    short sParam = newVal.iVal;
     : (Processing continues thereafter)
}
```

This implementation does not accept anything other than VT_I2 type, and if you do not set a value in newVal.iVal, unexpected behavior may occur. When users uses this provider, there is a restriction that VT_I2

must be specified and newVal.iVal should have a value.

<When CDataConv class is used>

If you use CDataConv class, the implementation will be as follows.

```
HRESULT CCaoProvVariable::FinalPutCtrlUserValue(VARIANT newVal)
{
    CDataConv dataConv;
    short sParam;
    long changedElem = dataConv.ChangeVarType(newVal, VT_I2, &sParam);
    if(changedElem != 1){
        return E_INVALIDARG;
    }
    : (Processing continues thereafter)
}
```

In this implementation, the CDataConv class tries to convert newVal given as an argument to VT_I2 as much as possible. If conversion is possible, the operation will return 1 as the number of elements that have been converted. In this case, the user does not need to be aware of the type. As long as the input can be converted to VT_I2, it can be executed even when specified with VT_BSTR type or VT_UI1 type.

Also, when converting the array type argument with CDataConv, implement it as follows.

```
HRESULT CCaoProvVariable::FinalPutCtrlUserValue(VARIANT newVal)
{
    CDataConv dataConv;
    short sParam[2];
    long changedElem = dataConv.ChangeVarType(newVal, VT_I2, &sParam, 2);
    if(changedElem != 2){
        return E_INVALIDARG;
    }
    : (Processing continues thereafter)
}
```

3.5. Method of creating error

The provider can handle provider-specific errors and can pass them to the CAO client by adding its own error codes, etc. Passing such unique error information improves the debugging efficiency of the provider and facilitates error handling when implementing a CAO client.

To create an error, proceed as follows:

(1) Define error codes.

(2) Define error messages.

(3) Cause errors

The details are described below.

(1) Define error codes collectively in the header file. (In this example, StdAfx. h is used.) An implementation example of the definition (error code E_SAMPLE) is shown below.

```
#define E_SAMPLE _HRESULT_TYPEDEF_(0x80100001L)
```

In this case, prefix the error name with "E_". The error code must be a 32-bit number in the _HRESULT_TYPEDEF_() macro. The error codes provided here are 0x80100000 to 0x801FFFFF (FACILITY CODE = 0x10).

Table 3-7 CAO error code assignments

Error codes	Modules used
0x80000200 - 0x800003FF	CAO
0x80000400 - 0x800005FF	CAO Provider templates
0x80000600 - 0x80000FFF	Other Modules
0x80100000 - 0x801FFFFF	CAO Provider

(2) Error messages are defined in StringTable of resources. This section shows how to register an error message as a resource.

1. Open CAOPROV.rc under your CAOPROV project in the Resources View and double-click StringTable.

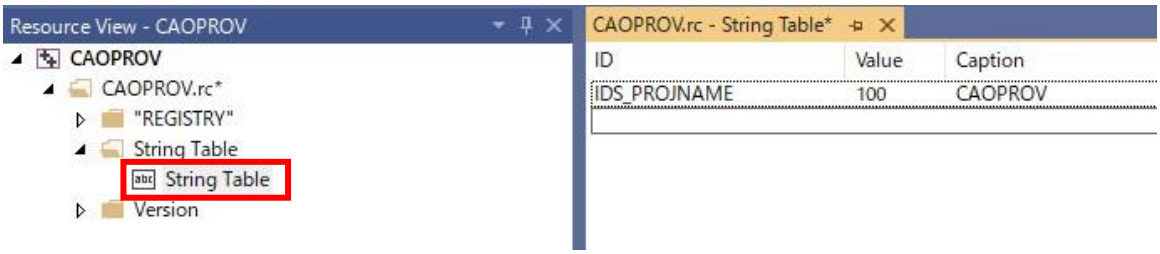


Fig. 3-1 String Table (Default)

2. The following String properties are displayed, so describe the IDs and captions. For example:
   ID:IDS_E_SAMPLE
   Captions: "This is test."
   Be careful not to overwrite any values other than those set by you.

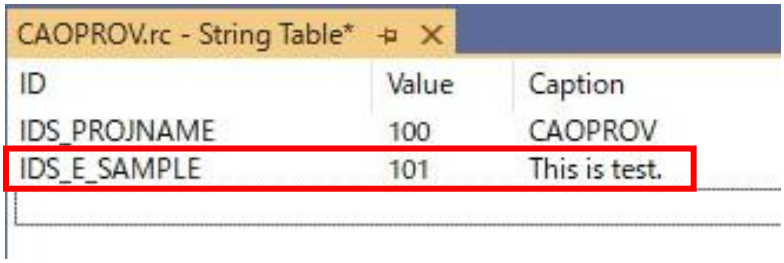


Fig. 3-2 String Properties Editor Window

Where ID is StringTable resource ID and caption is the error message. The resource ID is prefixed with "IDS_", followed by the error code defined in (1) (E_SAMPLE in this case). An error message has now been added to the resources that Provider has.

3. Compiles the resource file CAOPROV.rc. If compilation is successful, the resources created in Resource.h are registered.

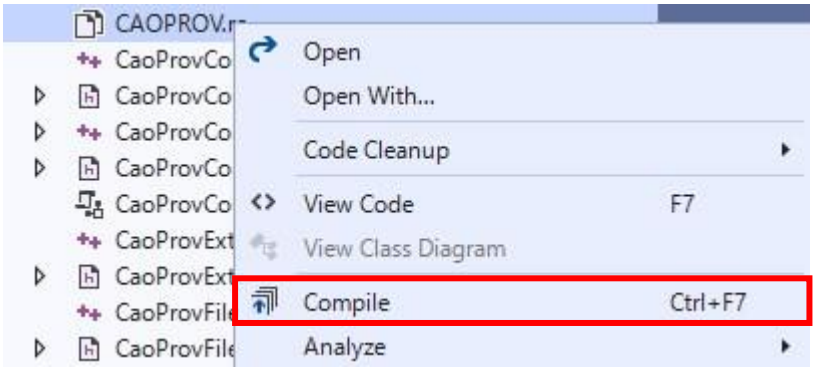


Fig. 3-3 Compiling CAOPROV.rc

4. Because the allowed resource values for provider implementations are 1024-1535, you can reassign the resource values you added in the following ways: In Resource. h, you can find the following definitions, which assign values to resource IDs: Rewrite the value here. The following example reassigns the resource allocated to 101 to 1024.

```
#define IDS_E_SAMPLE                101
```

↓

```
#define IDS_E_SAMPLE                1024
```

5. Compile CAOPROV.rc again to verify that StringTable values have changed.

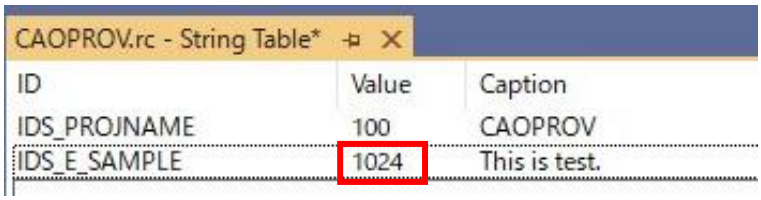


Fig. 3-4 Editing String properties after recompiling

(3) (1), Generates an error using the error code or resource created in step (2). Error codes can be treated the same as standard HRESULT errors. However, if you want to pass detailed error information (the resource created in (2)) to the client, you must call Error(). Error() is a method to provide error information to the client. Error() is defined as follows:

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

Where the arguments are the resource IDs, interface IDs, error codes, and resource handles from above. Resource ID is set to the resource ID defined in step (2). The interface ID is the ID of the interface for which the created error is to be returned. Enter the error name defined in (1) in the error code. The handle to the last resource uses the default value and does not need to be specified.

Here is an example of using a user-defined error and a user-defined error "E_SAMPLE".

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

In the example above, the second argument to Error function is the interface identifier for CAOVariable class. In fact, enter the interface ID of the class where the error occurred. Interface IDs can be selected from the IID-type constants defined in <Installation folder>¥ORiN2 ¥ CAO ¥ Include ¥ CAOPROV_i.c. If the above errors are raised in CaoTester2.exe, they are displayed in the logs as follows:

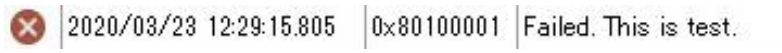


Fig. 3-5 CaoTester2 Error Log

3.6. About the message event

The provider can raise a message event at any time. Message events can be generated with CreateMessage() and events can be raised with SendMessage(). Here are the definitions for CreateMessage() and SendMessage():

```
STDMETHODIMP CreateMessage(
    TMess** ppMess,                 // Message object
    long lNumber = 0,               // Message number
    VARIANT *vntData = NULL,        // Message body
    VARIANT *vntDateTime = NULL,    // Date created
    BSTR bstrReceiver = NULL,       // Destination
    BSTR bstrSender = NULL,         // Sender
    BSTR bstrDescription = NULL     // Description
);

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

The second parameter to CreateMessage() is the message number and can be any value. The data in the third argument is the data to be written as a log. The supported VARIANT type for logging is VT_BSTR. The time when the fourth argument occurs is the date and time when the message is to be sent, and VARIANT type is VT_DATE. The fifth argument is invalid for log write requests, so you do not need to specify anything. The source of the sixth argument should be the name of the object for which the message is to be output, if necessary. Set the error description in argument 7, if necessary.[2][3]

Next, SendMessage() should specify the message generated by CreateMessage() as the first argument. The second argument specifies the message options. Table 3-8 shows the possible values for the message options. CAO_MSG_NORMAL is a general message. Generic messages are not processed within the CAO engine and are forwarded to the client as is.

An overview of the message mechanism is shown below.

[2] The message event supports various VT types, but when used in the log output function, messages other than BSTR type are not output efficiently.

[3] If the type of VARIANT is VT_EMPTY, it will automatically be padded with the date and time when CreateMessage was executed.

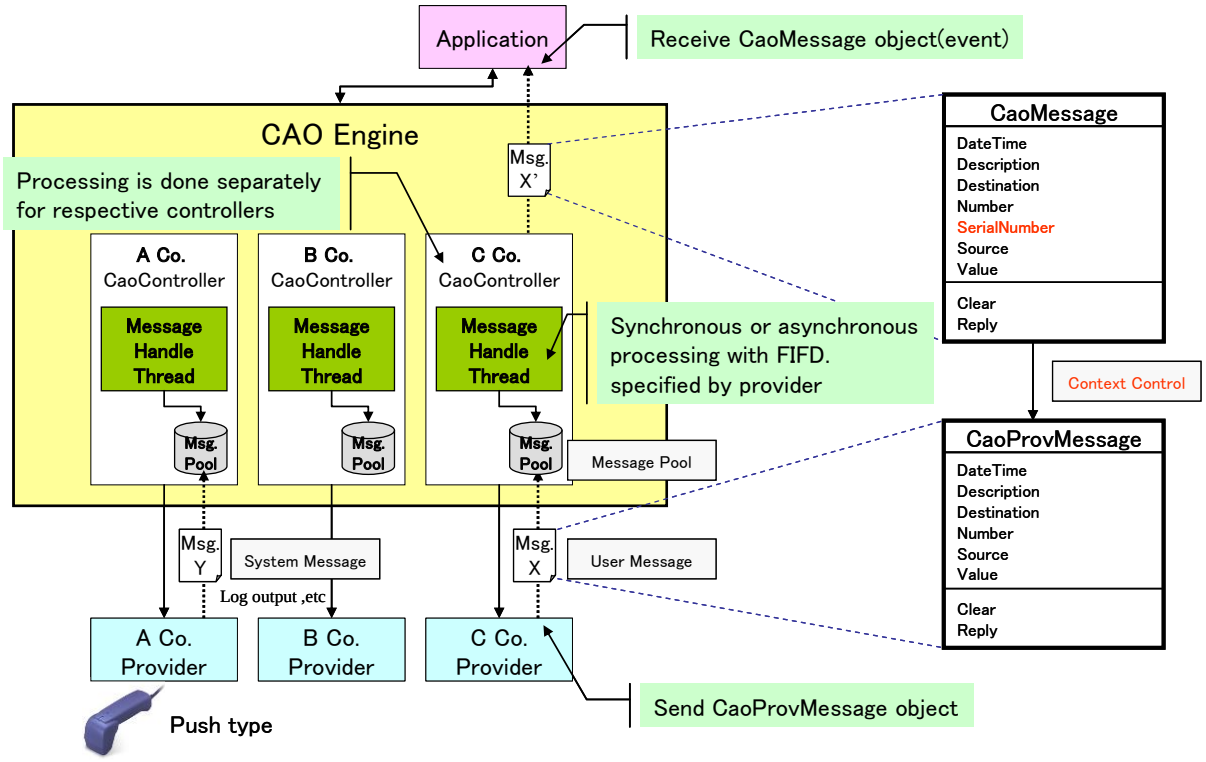


Fig. 3-6 CAO Message Mechanism

As shown in Fig. 3-6, the message is stored in the message pool of the CAO engine and then sent to the application. The maximum size of this message pool is 1000, and if the number of messages exceeds the maximum number, even if it is an asynchronous type message, it will be in "wait" status until there is space in the message pool.

Table 3-8 Message options and their behavior

	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 roughly divided into two types: transmission method and transfer destination, and these two types can be used in combination.

The following is a list of combinations and their option values.

Table 3-9 Combinations 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 following shows an example of output to the log by normal message.

```
// Message creation
CComPtr<CCaoProvMessage>pMess;
CComVariant vntData(L"This is test.");
HRESULT hr = CreateMessage(&pMess,lType, &vntData);
If (SUCCEEDED(hr)) {
    // Sending a Message
    Hr = SendMessage(pMess, CAO_MSG_OUTPUT_LOG);
}
```

3.6.1. Log output by message event

This section explains the "log output" using message events. For an explanation of log output, see "2.2.6. Log Output" in ORiN2 Programming Guide. To perform logging, specify CAO_MSG_OUTPUT_LOG as the second parameter to SendMessage().

The logging settings are configured in CaoConfig.exe of the CAO support tool. The screenshot of CaoConfig.exe is shown below.

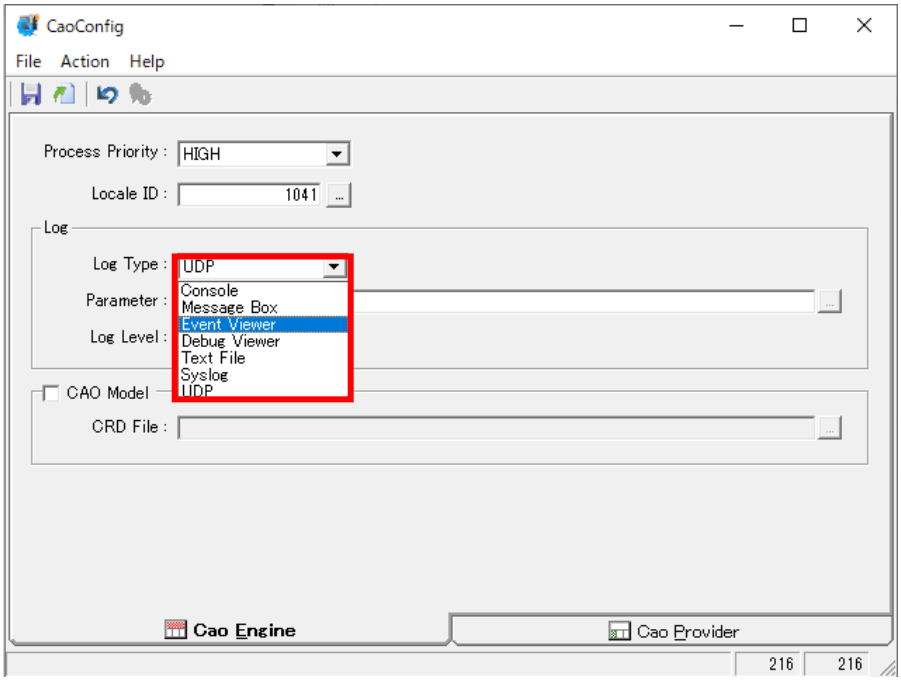


Fig. 3-7 CAOConfig.exe window

You can select the log destination from Log Type of combo boxes. (Fig. 3-7)

Table 3-10 Log output destinations

Output destination	Remarks
Console	Output to the console
Message Box	Outputs to a message box (at service startup)
Event Viewer	Outputs to the event viewer (at service startup)
Debug Viewer	Debug output.
Text File	Outputs to the specified text file.
Syslog	Prints to SysLog servers.
UDP	Output to the UDP server.

For more information, see "CaoConfig" in ORiN2 Programming guide.

3.6.2. In-process message forwarding

This section describes how to send and receive in-process message forwarding, which forwards messages to controllers of different providers.

When you perform in-process message forwarding, you must specify the destination controller names in CaoMessage objects. The destination controller name is set to the value of get_Destination() of the message object to be sent. When CaoMessage class is implemented by default, the value of get_Destination() can be specified by the fifth argument of CreateMessage(). In-process message forwarding can only be performed on controllers in CaoWorkspace objects to which the originating provider belongs. You can also forward messages to all controllers in CaoWorkspace objects by specifying an empty string as the destination.

To send messages, specify CAO_MSG_PROVIDER as the second parameter to SendMessage().

The destination provider calls OnMessage() on the controller object and stores the in-process forwarding message in the first argument. For this reason, in order to receive in-process forwarding messages, the destination providers must implement OnMessage() on the controller objects.

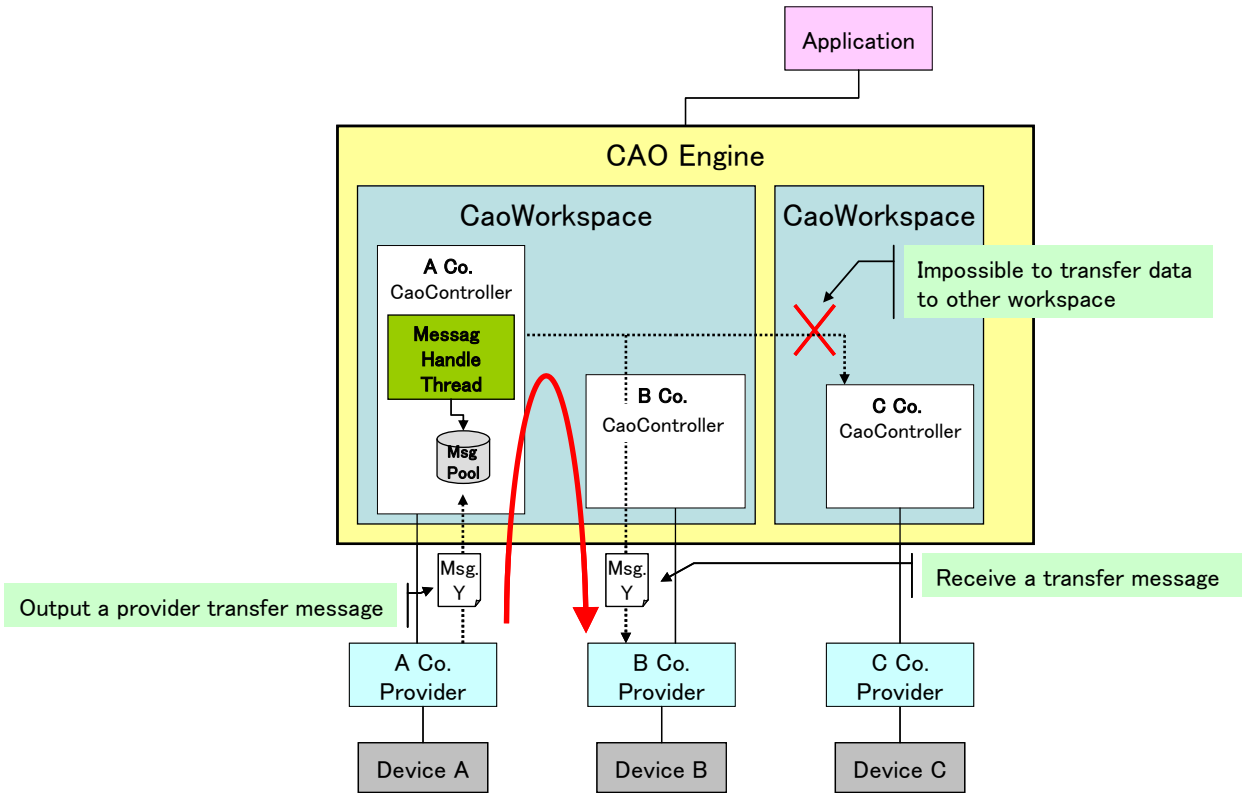


Fig. 3-8 Provider forwarding messages

The following is a sample of the sender and receiver.

List 3-1 In-process message transfer (sender)

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

  // Message creation
  hr = CreateMessage(&pMess, ID, &vntData, 0, bstrDest, 0, 0);
```

```
    // If message is created successfully
    if (SUCCEEDED(hr)) {
        // Sending a Message
        hr = SendMessageA(pMess, CAO_MSG_PROVIDER);
    }

    return hr;
}
```

List 3-2 In-Process Message Forwarding (Receiver)

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

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

    pMsg->Release();
    return hr;
}
```

3.6.3. Issue at constant cycle of message event

CAO provides a mechanism for the CAO engine to call the provider at regular intervals and for the CAO engine to issue events to the client upon request from the provider. To enable this mechanism in the provider, the CAOP_TIMER_INTERVAL macro must be defined at least 1 during implementation.

The CAOP_TIMER_INTERVAL macro defaults to StdAfx.h file of the provider project as follows:

```
#define CAOP_TIMER_INTERVAL    0
// 0: off, n: A positive integer less than or equal to.LONG_MAX that raises OnTimer events every
n (msec).
```

CAOP_TIMER_INTERVAL specifies the interval, in milliseconds, at which the CAO engine calls the provider at regular intervals. A value of 0 means no call.

If you define the CAOP_TIMER_INTERVAL macro to be greater than or equal to 1, the CAO engine invokes the provider's CCaoProvController::OnTimer() at regular intervals (<CAOP_TIMER_INTERVAL>ms).

The following are examples of how to implement OnTimer() method.

List 3-3 CCaoProvController.cpp – OnTimer()

```
#if CAOP_TIMER_INTERVAL > 0
/**    Timer Events
 *
 * Redefine the definition of #define CAOP_TIMER_INTERVAL 0 in milliseconds.
 * This calls this function 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
```

The CAOP_TIMER_INTERVAL mechanism makes coding very simple because the provider does not have to do polling on the client side.

3.7. Method of creating macro provider

A macro provider is a provider for connecting to another provider. The connecting provider can also connect to multiple providers, depending on how the macro provider is implemented.

The following sections describe how to generate the provider objects required to create macro providers and how to get OnMessage events.

3.7.1. Method of creating provider object

Provider objects can be created externally only for the controller. To create a controller object, use CoCreateInstance() or CoCreateInstanceEx().

In addition, non-controller objects are generated using the provider's interface. However, message objects cannot be created externally. Message objects can get provider-generated objects in OnMessage events. (See 3.7.2)

The following is a list of each object and the methods used to generate it.

Table 3-11 Provider Objects and Creation Methods

Provider object	Creation method
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 is a sample of a function that generates a controller object.

List 3-4 Sample.cpp

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

    // Provider generation
    USES_CONVERSION;

    CLSID clsid;
    ICaoProvController *pICaopCtrl;

    hr = CLSIDFromProgID(bstrProvider, &clsid);
    if (SUCCEEDED(hr)) {
        // Creating 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 {
            // Process out process
            DWORD dwLen = MAX_COMPUTERNAME_LENGTH + 1;
            LPTSTR pTstr = new TCHAR[dwLen + 1];
            GetComputerName(pTstr, &dwLen);
            if (strcmpi(pTstr, W2T(bstrMachine)) == 0) {
                // If the specified machine name is your machine name
                // (CLSCTX_LOCAL_SERVER)
                hr = CoCreateInstance(clsid,
                                      NULL,
                                      CLSCTX_LOCAL_SERVER,
                                      IID_ICaoProvController,
                                      (void **)&pICaopCtrl);
            } else {
                // If the specified machine name is a machine name other than yourself
                // (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;
        }

        // Connecting the Robot Controller
        hr = pICaopCtrl->Connect(bstrName, bstrOption);
        if (FAILED(hr)) {
            pICaopCtrl->Release();
            return hr;
        }
        *ppICaopCtrl = pICaopCtrl;
        return hr;
    }
```

3.7.2. Method of acquiring OnMessage event

To get OnMessage events from provider objects, you must implement EventSink classes in the macro provider.

Implementing EventSink classes requires the following steps:

(1) Registration to IDL file. (See 3.7.2.1)

(2) Implementation of connection and disconnection processing to the provider. (See 3.7.2.2)

(3) Implement handling when OnMessage events occur. (See 3.7.2.2)

(4) Addition of EventSink object-creation process. (See 3.7.2.3)

Each of these steps is described in turn below.

3.7.2.1. Adding EventSink to IDL file

To register EventSink classes in macro-provider IDLs, follow these steps.

(1) Create IDLs for EventSink classes.

(2) Register with the "CAOPROV_APPEND_CLASS" macro in the macro provider's CaoProv.idl file.

Here is how to use the "CAOPROV_APPEND_CLASS" macro:

```
#define CAOPROV_APPEND_CLASS <path to EventSink IDL-file>
```

The following is an example of registering with EventSink.idl file macro provider.

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

The provider includes the IDL file in the path specified by the "CAOPROV_APPEND_CLASS" macro.

The following is a sample IDL file for EventSink classes.

List 3-5 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("OnMessage")] HRESULT OnMessage([in] ICaoProvMessage *pICaopMsg);
};
// EventSink Class
 [
    uuid(D178B708-9330-4b87-B4FE-A540F1D4C55B),
    helpstring("EventSink Class")
]
coclass EventSink
{
     [default] interface IEventSink;
};
```

3.7.2.2. Implementation of EventSink classe

EventSink classes must implement the objects that generate OnMessage events and the connect and disconnect operations. Use AdlAdvise() to connect and AtlUnadvise() to disconnect. For more information about these functions, see MSDN.

An example of the connection and disconnection process is shown below.

List 3-6 Sample.cpp

```
// Connection processing
STDMETHODIMP CEventSink::Connect(ICaoProvController *pICaopCtrl)
{
     // Connects to CaoProvider connectable objects.
     HRESULT hr = AtlAdvise(pICaopCtrl, GetUnknown(), DIID__ICaoProvControllerEvents,
&m_dwCookie);
```

```
    if (SUCCEEDED(hr)) {
        // Saving ICaoProvController interfaces
        m_pICaopCtrl = pICaopCtrl;
    }
    return hr;
}

// Disconnection processing
STDMETHODIMP CEventSink::Disconnect()
{
    // Disconnect from CAOProvider connectable object and disable event notification.
    if (m_dwCookie) {
        AtlUnadvise(m_pICaopCtrl, DIID__ICaoProvControllerEvents, m_dwCookie);
    }
    m_dwCookie = 0;
    return S_OK;
}
```

OnMessage events are generated for the method whose EventSink interface DISPID registered in 3.6.2.1 is "0x01". Therefore, OnMessage event handling is implemented in the method whose DISPID is "0x01".

The following is a sample OnMessage reception method that sends messages received from providers to the CAO engine as-is to the CAO engine as events.

List 3-7 Sample.cpp

```
STDMETHODIMP CEventSink::OnMessage(ICaoProvMessage *pICaopMsg)
{
    HRESULT hr;
    // Event occurrences of the controller class
    hr = m_pCaopCtrl->Fire_OnMessage((IUnknown*)pICaopMsg);
    return hr;
}
```

3.7.2.3. Genetation of EventSink

EventSink classes are generated and can receive OnMessage events by performing connectivity operations.

The following are examples of EventSink generation and disappearance.

List 3-8 Sample.cpp

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

    // Instantiate a CEventSink
    CEventSink *pEvent;
    hr = CComObject<CEventSink>::CreateInstance(&pEvent);
    if (FAILED(hr)) {
        return hr;
    }
```

```
        pEvent->AddRef();

        // Connection with the 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();
        // Remove EventSink
        pEventSink->Release();

        return hr;
}
```

3.8. Usage of communication class

3.8.1. Introduction

ORiN2 SDK uses several communication classes to create providers. The class diagram for Device class is shown in Fig. 3-9. Provider developers can easily communicate with devices by using the classes shown in red frame.

This section describes the common class CDevice class for communication, and then describes CSerial class for serial communication inherited from this CDevice class, CTCPServer class for TCP/IP communication, CTCPClient class, and CUDPSocket class for UDP communication.

For source files of Device classes (Device.cpp/h, Serial.cpp/h, Socket. cpp/.h, TCPServer.cpp/h, TCPClient.cpp/h, UDPSocket.cpp/h), add the required files from <Installation folder>¥ORiN2¥CAO¥ Include to the project.

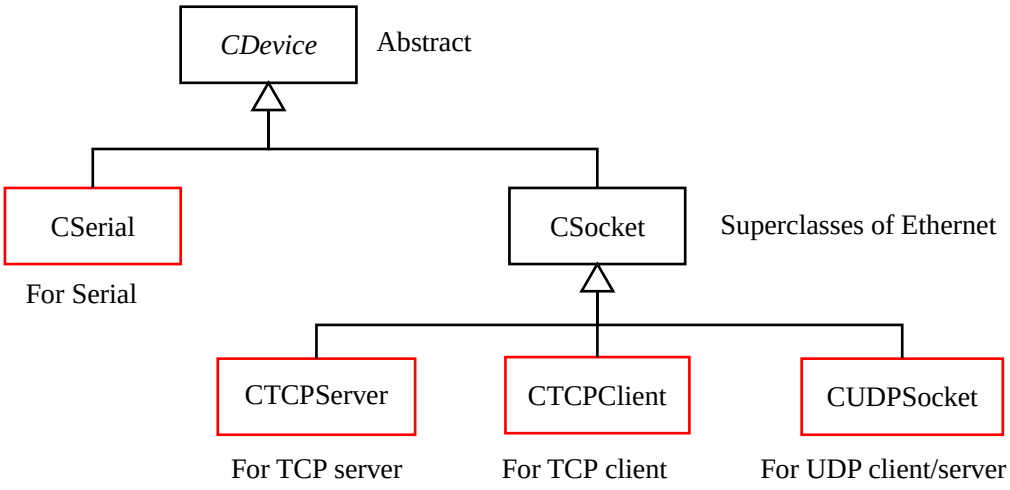


Fig. 3-9 Class diagram for Device class

3.8.2. CDevice class

CDevice class is an abstract class for device communication.

To implement a class that carries out device communication, this CDevice class can be inherited, and connection processing and send/receive processing can be implemented in virtual functions.

CDevice classes provide the following functionalities:

- Binary mode, text mode

  In binary mode, data is sent and received without processing.

  In text mode, perform character code conversion, header, terminator addition, etc. to perform transmission/reception.

- Setting of header and terminator

  Adds a header and terminator to send and receive data in text mode.

- Unicode Transformation Modes

  In text mode, the input data is converted to ASCII and sent, and the received data is converted to Unicode and returned.

- ISO code conversion, EIA code conversion

  Transmit/receive data is converted to ISO and EIA codes for communication in text mode.

Table 3-12 lists the members of CDevice class. If you want to inherit CDevice class, override the bold methods shown in this table as needed.

Table 3-12 List of members of a table CDevice

	Member functions and variables	Description
	CDevice()	CDevice Constructors
	~CDevice()	CDevice destructor
	AddRef()	The reference counter (m_lRefCnt) is incremented by 1.
	Cancel()	Receive cancellation event (m_hCancel) is turned on and data receive is canceled immediately. Data receive is disabled until CancelReset() is called.
	CancelReset()	Turn off the receive cancel event (m_hCancel) so that data receive processing can be performed.
	Clear()	Clears the buffer and clears the error. (pure virtual function)
	Connect()	Performs connection processing with the target. (pure virtual function)
	Disconnect()	Performs disconnection processing with the target. (pure virtual function)
	GetCancelMode()	Get the value of the member variable m_dwCancelMode.
	GetDestination()	Acquires the destination information.
	GetHandle()	Get a handle.
	GetMode()	Gets the currently set operation mode.
	GetReceivedCount()	Data receive count acquisition processing. Gets the number of data items currently in the buffer.
	GetReceivePacket()	Packet acquisition processing. Gets data from the local receive buffer. In text mode, the data from the header to the terminator is acquired as one packet. When there is no header, the data is acquired from the beginning of the buffer to the end of the buffer when there is no terminator. In binary mode, the command gets all data in the buffer or for the specified size.
	GetSource()	Get the sender information.
	GetTimeOut()	Timeout time acquisition processing. Get the timeout period.
	Initialize()	Initializing process. Configure settings such as mode settings and conversion functions.
	Receive()	Receive processing. Receive string data from the target. Call up ReceiveData to process data. In text mode, the data received after deleting headers and terminators, converting Unicode, etc. is returned.

		In binary mode, receive data is returned without processing.
	ReceiveData()	Virtual function for receiving data. The data received from the device is stored in a buffer, and one packet is cut and returned.
	Release()	The reference counter (m_lRefCnt) is set to-1.
	Send()	Transmission processing. Send the specified string data to the target. In text mode, SendData is called after adding headers and terminators, converting character codes to ASCII, and so on. In binary mode, SendData is called without processing.
	SendAndReceive()	Batch transmission/reception processing. Process all send → and receive in batch.
	SendData()	Virtual function for sending data. Sends the data received from Send to the device.
	SetCancelMode()	Change the mode of the cancel reception event (m_hCancel). Automatic reset if the argument is 0, manual reset mode if it is not 0.
	SetDestination()	Set the destination information.
	SetMode()	Resets the operation mode.
	SetSource()	Set the source information.
	SetTimeOut	Set the timeout period.
	m_dwBufLen	Local receive buffer length.
	m_dwCancelMode	Cancel mode.
	m_dwHeaderLen	The header length.
	m_dwMode	Operating mode.
	m_dwRetryCnt	Maximum number of retries to be performed.
	m_dwTermLen	Terminator length. 0 to 2 characters.
	m_dwTotalTimeOut	The maximum amount of time to wait before sending or receiving a string.
	m_hCancel	Cancel reception event
	m_lRefCnt	The referenced counter.
	m_pcHeader	Header code.
	m_pcTerm	Terminator code.
	m_szBuf	Local receive buffer.

:Public functions :Protected functions :Protected variables

3.8.3. Serial communication class

3.8.3.1. How to use

This section describes how to use CSerial class for serial communication inherited from CDevice class.

CSerial classes can send and receive data by connecting to the COM port specified by Connect method and using Receive method and Send method.

To perform RS-232C communication using CSerial classes, follow the steps below.

(1) Creating an Object

To use CSerial classes, you need to include Serial.h files. In this case, as shown in 2.3-(7), the include directory must be set. In addition, you must add Serial.cpp, which implements the functions of CSerial class, and Device.cpp, which implements the functions of the base class CDevice class of CSerial class, as pre-existing files for your projects.

```
#include <Serial.h>
CSerial serial;
```

(2) Initialization

Set the header/terminator to be added to the sent/received character strings, change the character codes, etc., and initialize CSerial classes.

```
CHAR cHeader[] = "";                // Header code
CHAR cTerm[] = "¥r";                // Terminator code
DWORD dwMode = ST_MODE_TEXT;        // Operation mode
PVOID pArgs[] = { cHeader,cTerm,&dwMode };

HRESULT hr = serial.Initialize(pArgs);
```

During initialization, you specify the following four parameters for an array of PVOID types:

・Header code

Specify the header code to be added to the transmitted data to indicate the start of the data.

・Terminator code

Specifies the terminator code to be appended to the transmitted data to indicate the end of the data.

・Operation mode

You can specify the operation mode when sending or receiving data. The following five operation modes can be specified by macros defined in Device.h. When specifying two or more types, take the logical sum of the following macros:

・ST_MODE_BINARY

Data is sent and received in binary format (byte array).

Send/receive the specified data as it is. Headers and terminators are not added or removed.

・**ST_MODE_TEXT**

Send and receive data in text format (character string).

When sending, a header and terminator are added to the character string.

On reception, a string with the header and terminator removed from the received data is returned.

・**ST_MODE_WBSTOWCS**

Data is converted from (Unicode(WCHAR) to (S-JIS(CHAR) and sent/received. This mode cannot be specified in binary mode.

・**ST_MODE_ASCTOISO**

Data is converted from ASCII code to ISO code and sent/received.

When used in conjunction with UNICODE conversion function, data is converted from UNICODE to the specified code and then sent/received.

・**ST_MODE_ASCTOEIA**

Data is converted from ASCII code to EIA code and sent/received.

When used in conjunction with UNICODE conversion function, data is converted from UNICODE to the specified code and then sent/received.

(3) Set up the communication parameters

Set communication parameters such as serial port number, communication speed (baud rate), data bit, stop bit, parity, flow control, etc. The parameters required for serial communication are defined in the PARAM_CONN_COM type structure.

```
ConnParam.dwDataBits = 8; // Number of data bits     : 4～8bits
ConnParam.dwStopBits = TWOSTOPBITS; // Number of stop bits    : 1(ONESTOPBIT),1.5(ONE5STOPBITS),
                                    // 2(TWOSTOPBITS)
ConnParam.dwFlow = 0; // Flow control : 0 (no control), 1 (-Xon/Xoff),
                                    // 2 (hardware control) (OR operation possible)


PARAM_CONN_COM connParam;
connParam.dwPortNo = 1;             // COM-port number    : COM1(1) ～
connParam.dwBaudRate = CBR_19200;   // Speed (bps)        : 4800bps(CBR_4800), ～
connParam.dwParity = NOPARITY;      // Parity : NOPARITY,ODDPARITY,EVENPARITY,
                                    // MARKPARITY,SPACEPARITY
connParam.dwDataBits = 8;           // Number of data bits     : 4～8bits
connParam.dwStopBits = TWOSTOPBITS; // Number of stop bits     : 1(ONESTOPBIT)
                                    //                           1.5(ONE5STOPBITS)
                                    //                           2(TWOSTOPBITS)
connParam.dwFlow = 0;               // Flow control  : 0 (no control), 1 (-Xon/Xoff),
                                    // 2 (hardware control) (OR operation possible)
```

(4) Open COM port

Open the COM port with the PARAM_CONN_COM type struct defined in (3) as arguments. Open the COM port with Connect() method as follows.

In addition, use SetTimeout() method to set the timeout time (in milliseconds) for sending and receiving data. Be sure to execute this after executing Connect method.

```
hr = serial.Connect(connParam);
hr = serial.SetTimeout(500);
```

(5) Send and receive data

Send and receive data with COM-port. When an error occurs, clear error/clear communication buffer (Flush) is executed.

Use Receive method to read data from COM ports, and Send method to write data: You can also use SendAndReceive method to send and receive data by batch.

```
BYTE lpszSendData[]{ 0x41,0x42 };     // Transmit data (character string: "AB")
BYTE lpszReceiveData[16];             // Received data
DWORD dwRecvedLen;                    // Number of receive data
DWORD dwRetryCnt = 3;                 // Number of retries
```

・Send data

```
hr = serial.Send(lpszSendData,           // Transmitted data
                 sizeof(lpszSendData), // Transmit data size (0 to send up to NULL
                                       // characters) If omitted: 0
                 true);                // Option to add header and terminator
                                       // true:append false:not append (default: true)
```

・Receive data

```
hr = serial.Receive(lpszReceiveData,         // Received data storage destination address
                    sizeof(lpszReceiveData),   // Receive data buffer size
                    &dwRecvedLen);             // Number of data actually received
```

・Send & receive data

```
hr = serial.SendAndReceive(lpszSendData,              // Transmitted data
                           sizeof(lpszSendData),     // Transmit data size (0:send up to NULL)
                           lpszReceiveData,      // Received data storage destination address
                           sizeof(lpszReceiveData),  // Buffer size of receive data
                           &dwRecvedLen,             // Number of data actually received
                           dwRetryCnt);              // Retry count (default: 1)
```

(6) Close the communication (COM) port

Close the COM port that you have finished using. Use Disconnect method to close the COM port. Be aware that if you forget this, other applications cannot use the the opened COM port.

```
serial.Disconnect();
```

3.8.3.2. Error codes

For serial communication classes, Windows standard error is masked with "0x80070000" as a unique error code.

e.g.) Windows Error: 0x02 (no files found) → CAO API Error: 0x80070002

3.8.4. TCP socket class

3.8.4.1. How to use

Next, we will explain how to use TCP socket class for Ethernet socket communication (TCP/IP), which inherits CDevice class.

Unlike CSerial classes, TCP socket uses different classes for establishing connections in server mode and client mode. Server mode uses CTCPServer class and client mode uses CTCPClietnt class.

CTCPServer class creates a CTCPClient class when Accept() is invoked by the connection request from clients. After the connection is established, data can be sent and received by calling Receive or Send method of CTCPClient class.

[Server mode]

(1) Creating an Object

To use CTCPServer classes, you need to include TCPServer.h files. In this case, as shown in 2.3-(7), the include directory must be set.2.3(7)

You must add the following three files to your project as existing files:

1. TCPServer.cpp... CTCPServer class-implementation files
2. TCPClient.cpp... CTCPClient class-implementation files
3. Implementation files for CSocket class, the base class for Socket. cpp... CTCPServer
4. Implementation files for CDevice class, the base class for Device. cpp... CSocket

```
#include <TCPServer.h>
CTCPServer tcpServer;
```

(2) Initialization

During initialization, following parameters are set; Header code, terminator code, and operation mode (the content is the same as CSerial classes).

At the time of Connect, set the IP-address and port number of socket communication. The parameters required for TCP communication are defined in a structure of type PARAM_CONN_ETH, where INADDR_NONE is your PC's IP address.

If Connect is successful, set the timeout period.

```
CHAR cHeader[] = "";          // Header code
CHAR cTerm[] = "¥r";          // Terminator code
```

```
DWORD dwMode = ST_MODE_TEXT;  // Operation mode
PVOID pArgs[] = { cHeader,cTerm,&dwMode };

HRESULT hr = tcpServer.Initialize(pArgs);


PARAM_CONN_ETH connParam;
connParam.dwSrcIP = INADDR_NONE;        // Local IP address
connParam.dwSrcPort = 5006;             // Local Port Number

hr = tcpServer.Connect(connParam);

DWORD dwTimeout = 1000;             // Timeout period [ms]
hr = tcpServer.SetTimeout(dwTimeout);
```

(3) Waiting for connection

TCPServer classes do not send or receive data on their own. The class waits for connection requests from clients and creates a new connection point each time a connection request comes.

In fact, CTCPSocket pointers to new connections are passed as arguments to Connect method, and new objects are assigned as arguments when a connection is requested from the client, as follows: CTCPSocket classes must execute their Accept methods sequentially until connect requests arrive.

```
CTCPClient* pTcpClient;
while(true)
{
  hr = tcpServer.Accept(&pTcpClient);
  if(hr == S_OK)
  {
       break;
  }
}
```

Use this newly allocated object to send and receive data.

[Client Mode]

(1) Creating an Object

To use CTCPClient classes, you need to include TCPClient.h files. In this case, as shown in 2.3-(7), the include directory must be set.

You must add the following three files to your project as existing files:

1. TCPClient.cpp... CTCPClient class implementation file
2. Socket.cpp... Implementation file of CSocket class which is the base class of CTCPClient
3. Device.cpp...Implementation file of CDevice class which is the base class of CSocket

```
#include <TCPClient.h>
CTCPClient tcpClient;
```

(2) Initialization

Set the required parameters and initialize CTCPClient class. The header code, terminator code, and

operation mode (the content is the same as CSerial class) are set as the setting parameters at Initialize.

```
CHAR cHeader[] = "";          // Hedder code
CHAR cTerm[] = "¥r";          // Terminater code
DWORD dwMode = ST_MODE_TEXT;  // Operation mode
PVOID pArgs[] = { cHeader,cTerm,&dwMode };

HRESULT hr = tcpClient.Initialize(pArgs);
```

(3) Establish a connection

Use Connect() method to establish a connection to the port which the server program is waiting for the connection. If the connection is successful, data can be sent and received using this object. Parameters required for TCP communication are defined in the PARAM_CONN_ETH type structure. Specify the local IP address and port number, and the IP address and port number of the connection destination. By setting the local port number to 0, an appropriate free port number is assigned.

If connect is successful, set the timeout with SetTimeout().

```
PARAM_CONN_ETH connParam;
connParam.dwSrcIP = INADDR_NONE;                    // Local IP address
connParam.dwSrcPort = 0;                            // Local Port Number
connParam.dwDestIP = inet_addr("127.0.0.1");        // Destination iP address
connParam.dwDestPort = 5006;                        // Number of the specific port

hr = tcpClient.Connect(connParam);
hr = tcpClient.SetTimeout(1000);                    // Timeout setting [ms]
```

[Send and receive data in the TCP socket class]

Whichever the connection mode (server or client), if you have established a connection as described above, you can send and receive data in the same way. (CTCPServer object does not implement data transfer. Use CTCPClient object generated by Accept method.)

(4) Send and receive data

To send and receive data by socket communication, use the methods shown below.

```
BYTE lpszSendData[]{ 0x41,0x42 };     // Transmit data (character string: "AB")
BYTE lpszReceiveData[16];             // Received data
DWORD dwRecvedLen;                    // Number of receive data
DWORD dwRetryCnt = 3;                 // Number of retries
```

・**Send data**

```
hr = tcpClient.Send(lpszSendData,       // Transmitted data
                    sizeof(lpszSendData), // Transmit data size (0 to send up to NULL
                                          // characters) If omitted: 0
                    true);                // Option to add header and terminator
                                          // true:append false:not append (default: true)
```

・**Receive data**

```
hr = tcpClient.Receive(lpszReceiveData,      // Received data storage destination address
                       sizeof(lpszReceiveData),   // Receive data buffer size
                       &dwRecvedLen);             // Number of data actually received
```

・**Send & receive data**

```
hr = tcpClient.SendAndReceive(lpszSendData,          // Transmitted data
                              sizeof(lpszSendData),     // Transmit data size (0:send up to NULL)
                              lpszReceiveData,      // Received data storage destination address
                              sizeof(lpszReceiveData),  // Buffer size of receive data
                              &dwRecvedLen,             // Number of data actually received
                              dwRetryCnt);              // Retry count (default: 1)
```

(5) Close the connection

When data transmission/reception is completed, it is necessary to execute the disconnection processing. The disconnection process is executed as follows using Disconnect() method.

```
// Client mode
tcpClient.Disconnect();

// Server mode
pTcpClient->Disconnect();
delete pTcpClient
pTcpClient = NULL;

tcpServer.Disconnect();
```

3.8.4.2. Error codes

For TCP socket classes, return a unique error code masking Winsock error with "0x8091000".

e.g) Winsock Error: 10061 (Reject Connections) → CAO API Error: 0x8091274D

3.8.5. CUDPSocket class

3.8.5.1. How to use

Next explanation is how to use CUDPSocket class for Ethernet socket communication (UDP) which inherits CDevice class. CUDPSocket class can send and receive data by calling Receive method and Send method in the same way as CTCPSocket class.

(1) Creating an Object

To use CUDPSocket classes, you need to include UDPSocket.h files. In this case, as shown in 2.3-(7), the include directory must be set.2.3(7)

You must add the following three files to your project as existing files:

1. UDPSocket.cpp... CUDPSocket class-implementation files
2. Implementation files for CSocket class, the base class for Socket. cpp... CUDPSocket
3. Implementation files for CDevice class, the base class for Device. cpp... CSocket

```
#include <UDPSocket.h>
CUDPSocket udpSocket;
```

[Server mode]

(2) Initialization

Set the required parameters and initialize CUDPSocket class. Specify the header code, terminator code, and operation mode (the content is the same as CSerial class) as the setting parameters at Initialize. However, to use the UDP server mode, add ST_MODE_UDP_SERVER to the operation mode with the OR operation. If you forget this description, you will not be able to receive data.

```
CHAR cHeader[] = "";            // Header code
CHAR cTerm[] = "¥r";            // Terminator code
DWORD dwMode = ST_MODE_TEXT | ST_MODE_UDP_SERVER;     // Operation mode
PVOID pArgs[] = { cHeader,cTerm,&dwMode };
HRESULT hr = udpSocket.Initialize(pArgs);
```

(3) Open a socket

Open the socket and prepare for data transmission/reception. If Connect() is successful, data can be sent/received using this object. Parameters required for UDP communication are defined in the PARAM_CONN_ETH type struct and specify the local IP address and port number.

If Connect is successful, set the timeout with SetTimeout().

```
PARAM_CONN_ETH connParam;
connParam.dwSrcIP = INADDR_NONE;    // Local IP address
connParam.dwSrcPort = 5006;         // Local Port Number

hr = updSocket.Connect(connParam);
hr = udpSocket.SetTimeout(1000);    // Timeout setting [ms]
```

[Client Mode]

(2) Initialization

Set the required parameters and initialize CUDPSocket class. Specify the header code, terminator code, and operation mode (the content is the same as CSerial class) as the setting parameters at Initialize.

```
CHAR cHeader[] = "";            // Hedder code
CHAR cTerm[] = "¥r";            // Terminator code
DWORD dwMode = ST_MODE_TEXT;    // Operation mode
PVOID pArgs[] = { cHeader,cTerm,&dwMode };
HRESULT hr = udpSocket.Initialize(pArgs);
```

(3) Open a socket

Open the socket and prepare for data transmission/reception. If Connect() is successful, data can be sent/received using this object. Parameters required for UDP communication are defined in the PARAM_CONN_ETH type struct and specify the local IP address and port number. By setting the local port number to 0, an appropriate free port number is assigned.

If Connect is successful, set the timeout with SetTimeout().

```
PARAM_CONN_ETH connParam;
connParam.dwSrcIP = INADDR_NONE;    // Local IP address
connParam.dwSrcPort = 0;            // Local Port Number

hr = udpSocket.Connect(connParam);
hr = udpSocket.SetTimeout(1000);    // Timeout setting [ms]
```

[Send and receive data in the UDP socket class]

(4) Destination setting (This setting is not necessary when only data reception is used.)

Use SetDestination() to specify the destination. In the configuration parameters, specify the destination IP address and port number as DWORD type.

```
DWORD dwDist[2];
dwDist[0] = inet_addr("127.0.0.1");   // Destination IP address
dwDist[1] = 5006;                     // Number of the specific port
udpSocket.SetDestination((PVOID)dwDist);
```

(5) Send and receive data

To send/receive data via socket communication, use the following method.

```
BYTE lpszSendData[]{ 0x41,0x42 };     // Transmit data (character string: "AB")
BYTE lpszReceiveData[16];             // Received data
DWORD dwRecvedLen;                    // Number of receive data
DWORD dwRetryCnt = 3;                 // Number of retries
```

・Send data

```
hr = udpSocket.Send(lpszSendData,         // Transmitted data
                    sizeof(lpszSendData), // Transmit data size (0 to send up to NULL
                                          // characters) If omitted: 0
                    true);                // Option to add header and terminator
                                          // true:append false:not append (default: true)
```

・Receive data

```
hr = udpSocket.Receive(lpszReceiveData,      // Received data storage destination address
                       sizeof(lpszReceiveData),   // Receive data buffer size
                       &dwRecvedLen);             // Number of data actually received
```

・Send & receive data

```
hr = udpSocket.SendAndReceive(lpszSendData,          // Transmitted data
                              sizeof(lpszSendData),     // Transmit data size (0:send up to NULL)
                              lpszReceiveData,      // Received data storage destination address
```

```
                                    sizeof(lpszReceiveData),   // Buffer size of receive data
                                    &dwRecvedLen,              // Number of data actually received
                                    dwRetryCnt);               // Retry count (default: 1)
```

(6) Close the connection

When data transmission/reception is completed, it is necessary to execute the disconnection processing. The disconnection process is executed as follows using Disconnect method.

```
udpSocket.Disconnect();
```

3.8.5.2. Error codes

In the UDP socket class Winsock error is masked with "0x8091000" as a unique error code.

e.g.) Winsock error:10048 (Specified address in use) → CAO API error:0x80912740

4. Creating a TCmini provider

This chapter describes how to create providers for the Small Programmable Controller TCmini made by Shibaura Machine. This time, the functions (methods) required for TCmini relays and register reads/writes are implemented according to the procedures described in Section 2.2

In this manual, Shibaura Machine's Small Programmable Controller TCmini is called "TCmini Controller", and the provider created this time is called "TCmini Provider".

To create TCmini providers, the following items are required.

- Microsoft Visual Studio 2019 Professional
- ORiN2 SDK Provider Development
- TCmini controllers
- Basic knowledge of the C++ language and COM
- Basic knowledge of serial communication

4.1. What is the TCmini controllers

4.1.1. Configuration

TCmini controller is a compact programmable controller with 16 photocoupler inputs, 16 relay outputs, 8 dip switch inputs, 4 temperature inputs (thermistors), 2 analog inputs (0-5V), 2 analog outputs (0-5V), calendar function, RS-232C communication function, RS-485 communication function, and extended function.

TCmini controller relays consist of 1 point = 1 bit and 16 points = 16 bits. Relays and registers are basically represented by "<functional symbol>+<address>". <functional symbol> stands for a type and specifies one of the following alphabetic characters: 'X', 'Y', 'R', 'T', 'C', 'L', 'E', 'A', 'D', 'V', or 'P'. <address> is a three-character number represented in hexadecimal. However, in the case of relays, data can be handled as byte registers for 8 points at a time, or as word registers for 16 points at a time. In this case, the last character (the third character) of the <address> is the register type. The register type can contain one letter 'L', 'H', or 'W'.

4.1.1.1. Data memory types

Table 4-1 Data memory types

Type	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 preset	96 words			V000 - V05F
T.C Recent Value	96 words			P000 - P05F

4.1.1.2. Function of data memory

Table 4-2 Data memory functions

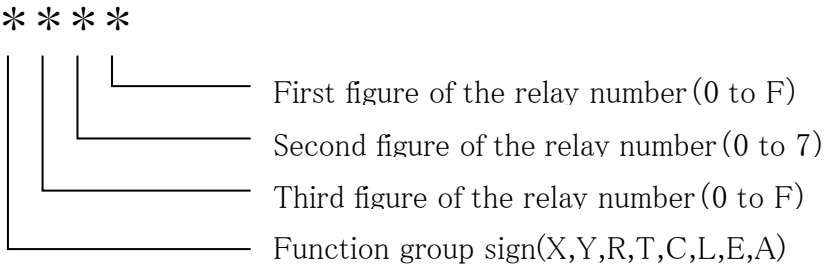
Name		Group	Function
I/O relay	Input relay	X	A relay only for inputs where various inputs (such as photo couplers, keyboard, and DISPSW) are connected.
	Output relay	Y	A relay only for outputs where various output devices (such as relays, panel LED) are connected.
	Unrecognition relay	Z	This performs 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.

4.1.1.3. Relay address

Relay addresses are expressed with a relay number and function division function.

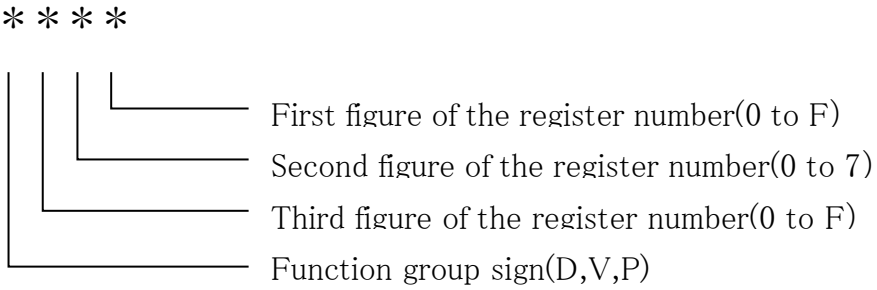
I/O relay addresses correspond to actual relay installations, while other relay addresses correspond to devices (virtual devices) that do not physically exist. Relay addresses are assigned one point (1 bit) at a time.

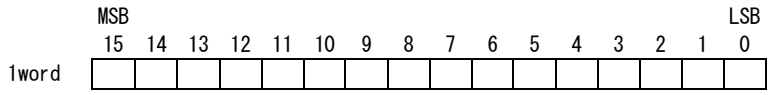


4.1.1.4. Data register address

The data register address has the same representation as the relay address.

Relay addresses are in units of 1 point (1 bit), while data register addresses are in units of 1 word (16 bit).

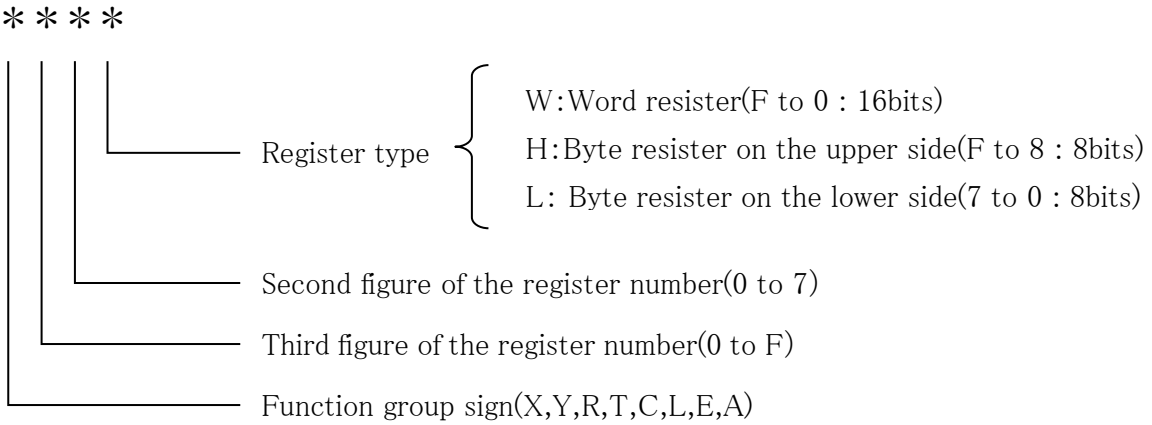




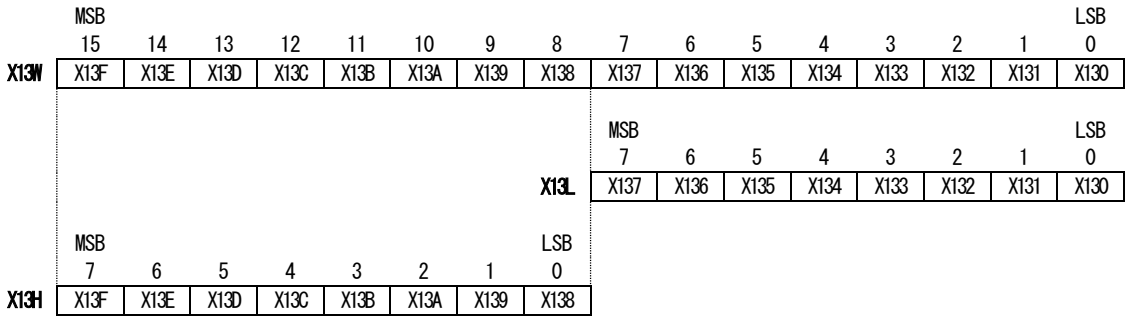
4.1.1.5. Byte/word register address in relay area

The relay area can be handled as a byte register for every 8 points and data as a word register for every 16

points. The "Relay No.1 digit" part of the relay address is the register type.



E.g.) Correspondence when X130 to X13F is specified by word register and byte register



4.1.2. Connection

Since various services from TCmini controllers are provided via RS-232C, connect RS-232C connector on the front of the module to the serial (COM) port on the PC (host computer) using a straight cable. Please note that the cable used is not a cross cable.

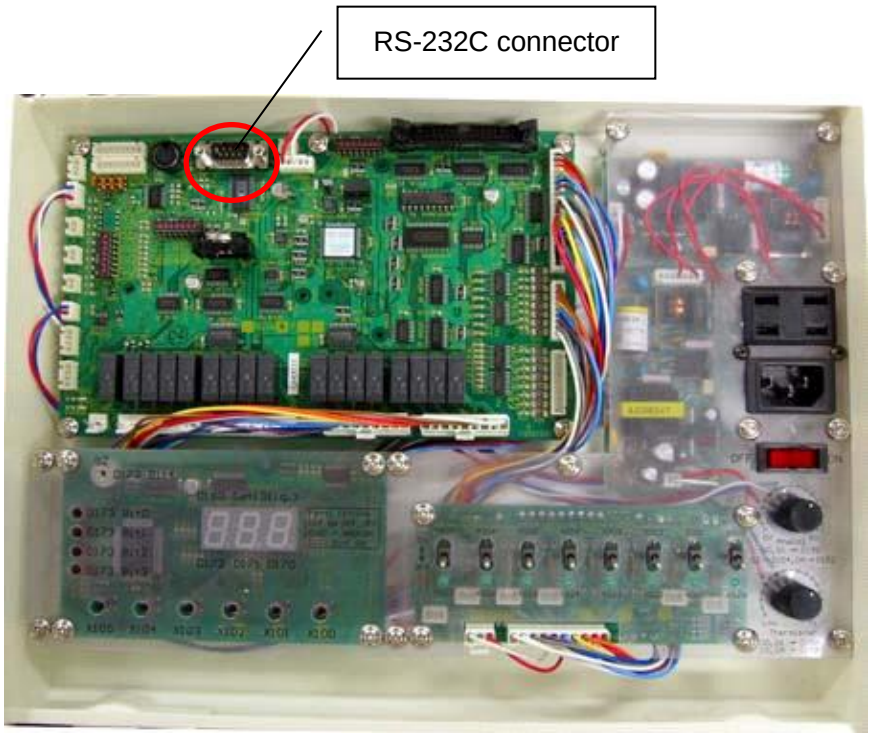


Fig. 4-1 TCmini controller

In order to communicate with TCmini controller, the COM port must be set as follows.

Communication Parameter	Setting Value
Baudrate [bps]	9600/19200/38400
Data bit [bits]	8
Parity bit	None
Stop bit[bits]	2
Fllow Control	None

TCmini setting is basically unnecessary. TCmini controller automatically judges the baud rate if the baud rate is within the 9600/19200/38400bps range. The default setting is 19200bps.

4.1.3. Overview of the communication protocol

When the PC (host computer side) and TCmini controller (TCCUH* and TCCM* side) communicate, data must be accessed from the PC side. The figure is shown below.

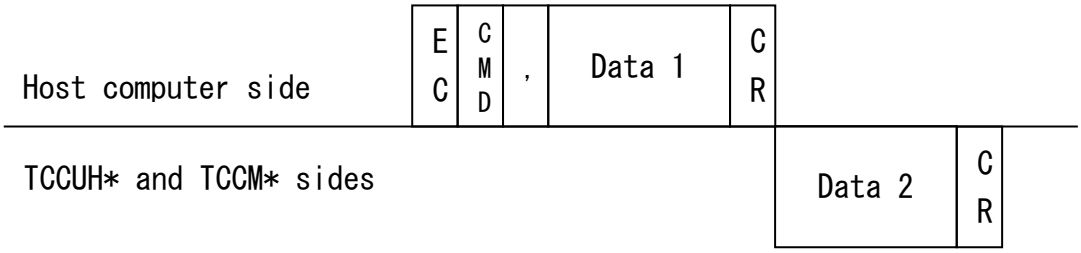


Fig. 4-2 Basic Format

Table 4-3 Fixed data of communication protocol

Symbol	Description
EC	Means escaping, ASCII codes are 0x1B.
CMD	Use 0x30 (='0') to 0x39 (='9') of ASCII codes with the numbers "0" to "32" for the command number.
,	A symbol (comma) used to distinguish the command number and data. ASCII codes are 0x2C.
CR	Signifies a carriage return, ASCII Code 0x0D.
Data 1	Supplementary data for the command.
Data 2	The reply data corresponds to the command.

When commands (CMDs) and contents (data 1) are sent from the PC-side to TCmini controller in the format described above, TCmini controller interprets the sent data, executes the requested process, and returns the reply data (data 2). If the command sent to the controller is not correct, an error message such as "Parameter error" is returned as response data.

All communications are performed using ASCII codes.

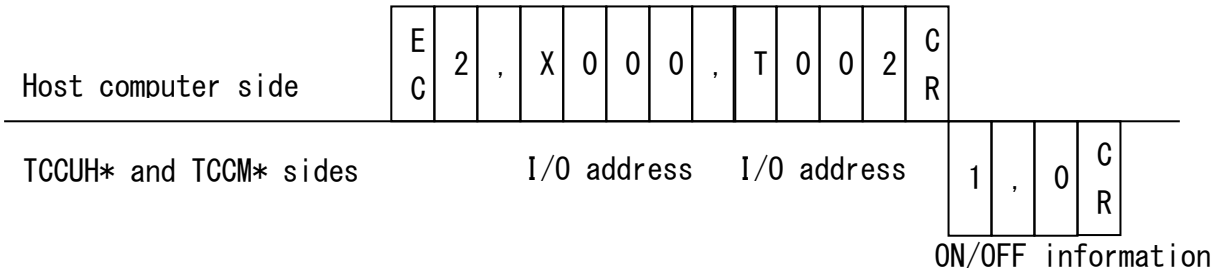
The following sections describe only some command specifications for this communication format. For other command specifications, refer to the "Instructions for RS232C Host Communication Commands" supplied with TCmini controller.

4.1.3.1. I/O reading out per one point

Reads ON/OFF data of up to 42 contacts.

<CMD>"2" (0x32)

The examples below show X000 (ON) and T002 (OFF).



ON/OFF info

0 : OFF

1 : ON

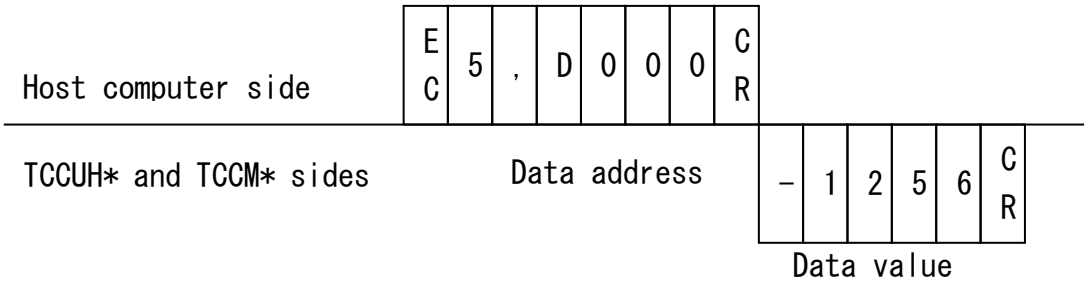
I/O address (Max. 42 contacts)

- Be sure to specify four digits.
- Category code (X, Y, R, L, S, T, C, E, A, G, H) must be specified in upper case.
- Specifying a nonexistent address results in a "Parameter Error" message.

4.1.3.2. Data reading out per one word

Up to 32 words of data can be read at the same time.

<CMD>"5" (0x35)



Data address

- Be sure to specify four digits.
- Classification code (X, Y, R, L, S, T, C, E, A, G, H, D, P, V, B) word specification W, byte specification H, L should be specified in upper case.
- Specifying a nonexistent address results in a "Parameter Error" message.
- Do not mix words and bytes in the address specification

Data value

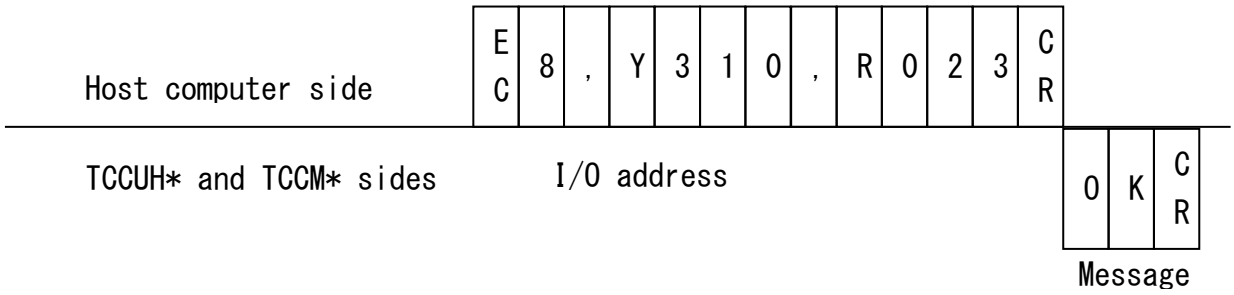
- Data returns BIN data as a decimal number
- Word data ranges from-32768 to 32767

- Byte data can range from 0 to 255.

4.1.3.3. I/O forced set

Up to 42 relay contacts can be ON (forcibly set) at the same time.

<CMD>"8" (0x38)



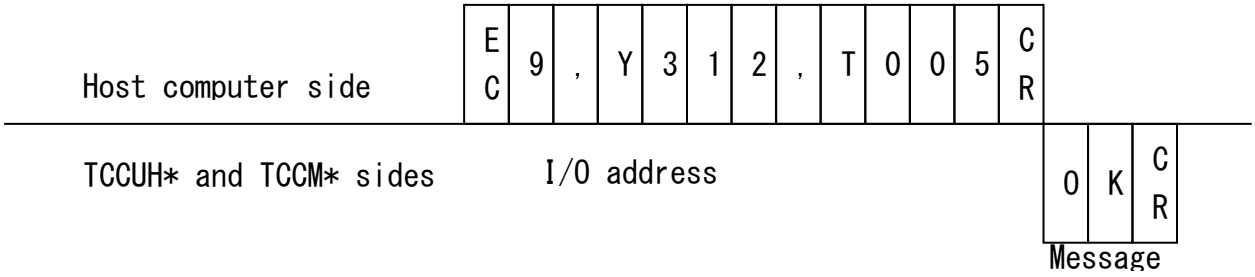
I/O address (Max. 42 contacts)

- Be sure to specify four digits.
- Category code (X, Y, R, L, S, T, C, E, A, G, H) must be specified in upper case.
- Specifying a nonexistent address results in a "Parameter Error" message.
- It is not possible to set the area being reset by the sequence program.
- When the timer/counter area is set, the current value is set to 0 and the contact is turned ON.

4.1.3.4. I/O forced reset

Up to 42 relay contacts are turned OFF (forced reset) at the same time.

<CMD>"9" (0x39)



I/O address (Max. 42 contacts)

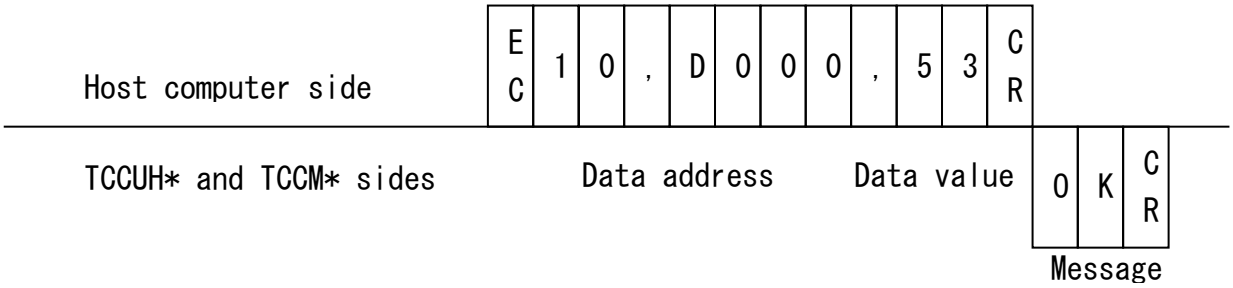
- Be sure to specify four digits.
- Category code (X, Y, R, L, S, T, C, E, A, G, H) must be specified in upper case.
- Specifying a nonexistent address results in a "Parameter Error" message.
- The area set by the sequence program cannot be reset.

・ When the timer/counter area is reset, the current value is equal to the set value and the contact is turned OFF.

4.1.3.5. Data change per one word

Up to 32 data can be changed at the same time.

<CMD>"10" (0x31,0x30)



Data address

・ Be sure to specify four digits.

・ Classification code (X, Y, R, L, S, T, C, E, A, G, H, D, P, V, B) word specification W, byte specification H, L should be specified in upper case.

・ Specifying a nonexistent address results in a "Parameter Error" message.

・ Do not mix words and bytes in the address specification

Data value

・ Data returns BIN data as a decimal number

・ Word data ranges from-32768 to 32767

・ Byte data can range from 0 to 255.

4.2. TCmini provider specification

Before you can create a TCmini provider, you must first determine its behaviour, that is, its specification. Fig. 4-3 shows the correspondence between TCmini and classes of providers. CaoProvController is TCmini main unit, and CaoProvVariable is TCmini relay address and data register address. Implement the provider according to this specification.

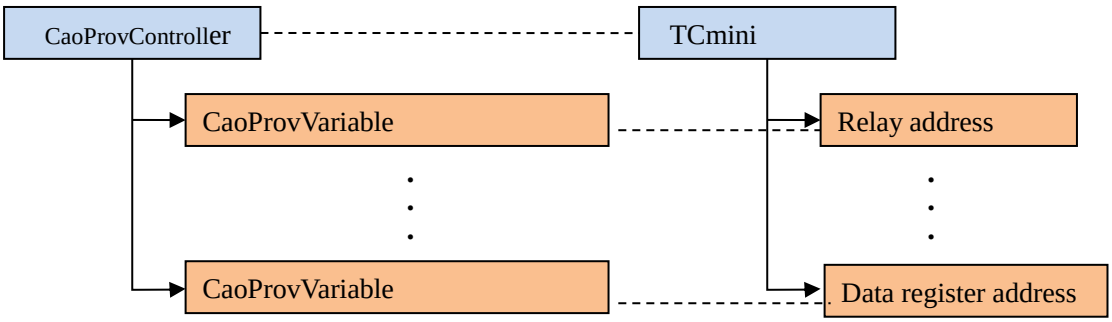


Fig. 4-3 Correspondence between providers classes and TCmini

4.2.1. Optional specifications for AddController

Four arguments are required at CaoWorkspace::AddController. This section determines the specifications of option string to pass to the fourth argument when using TCmini providers.

Format

```
AddController
(
    "<Controller>",          // Controller name (optional)
    "<ProvID>"               // Provider name
    "<machine name>",             // Name of the provider's running machine
    "<options>"              // Option string
)
```

To connect to TCmini, you need to determine the connection parameters, which port the user will communicate on, and the baud rate, in bits per second. It is also useful to allow the user to set a timeout period for sending and receiving.

Based on the above, Table 4-4 shows the specifications of TCmini optional character strings. We will use "Conn", which means the connection parameters for communication, and "Timeout", which means the timeout time for transmission and reception, as option strings. Each option string (Conn or Timeout) is followed by a "=" to specify the connection parameters and timeouts based on how they are described. Also, Conn string is required, but Timeout string is optional. If two options are specified at the same time, they can be described by adding "," between each option string.

Table 4-4 Option string specifications

Item	Option string	Required	Description method

Connection parameters	Conn	○	"Conn=com:<COMPort>[:<BaudRate>[:<Parity>:<DataBits>:<StopBits>[:<Flow>]]]" <COM Port>: COM-port number. '1'-COM1,'2'-COM2, ... <BaudRate>: Baud rate. 4800,9600,19200,38400,57600,115200. <Parity> : Parity 'N'-NONE,'E'-EVEN,'O'-ODD <DataBits>: Number of data bits. '7'-7bit,'8'-8bit. <StopBits>: Number of stop bits. '1'-1bit,'2'-2bit. <Flow> : Flow control. '0'-no flow-control, '1'-Xon/Xoff,'2'-hardware-control. (OR operation specifiable) * The underlined part is the default value when no options are specified.
Timeout period	Timeout	-	"Timeout=<Time>" <Time>: Timeout period. Specify within the range of 0 to 4294967295 ms. (Default: 500 ms)

E.g.1） COM port number: 1, Communication speed: 9600 bps, Parity: E, Data bits: 7 bit, Stop bits: 1 bit, When specifying flow control: Xon/Xoff, timeout: 1000 ms.
(Description) → "Conn = com:1:9600:E:7:1:1,Timeout=1000"

E.g.2） COM port number: 1, Communication speed: 19200 bps, Parity: None, Data bits: 8 bit, Stop bits: 1 bit, When flow control: None, timeout: 500 ms is specified.
(Description) → "Conn=com:1" (Defaults are optional)

Usage example (C#)

```
using ORiN2.ManagedCAO;
CCaoEngine eng = new CCaoEngine();
CCaoWorkspace ws = eng.AddWorkspace("SampleWorkspace", "");
CCaoController ctrl = ws.AddController("controller1","CaoProv.TCmini", string.Empty,
                                                    "Conn=com:1,Timeout=1000");
```

4.2.2. AddVariable variable name and optional specifications

Two arguments are required at CaoController::AddVariable; the variable name and the option string. There are two types of variable names: System variable and User variable. System variable are fixed character strings, and User variable are arbitrary character strings. The System variable must use "@" at the beginning of the string, and you can get the list of System variable by executing the get_VariableNames property. This section determines the specifications of System variable and User variable when using TCmini providers.

Format

```
AddVariable
(
    "<variable name>"          // Variable name
    "<options>"                // Option string
)
```

4.2.2.1. System variable specification

Table 4-5 shows the specifications of the System variable. All System variables should not use option strings. Only the get_Value property is allowed, and the put_Value property is not implemented. In addition, the get_Value properties of @CURRENT_TIME and @ERROR_DESCRIPTION communicate with TCmini to get data, but other System variable do not communicate with TCmini.

Table 4-5 System Variable Specifications Implemented by TCmini Providers

Variable name	Option string	Data Type	Description	Attribute	
				Get	Put
@CURRENT_TIME	None	VT_DATE	Present time of TCmini retention	○	×
@ERROR_DESCRIPTION	None	VT_BSTR	TCmini retained alarm messages	○	×
@MAKER_NAME	None	VT_BSTR	Manufacturer name. "Shibaura Machine"	○	×
@TYPE	None	VT_BSTR	Device name. "TCmini"	○	×
@VERSION	None	VT_BSTR	Provider version	○	×

Usage example (C#)

```
string[] variableNmaes = ctrl.GetVariableNames(string.Empty);
```

```
Debug.WriteLine(variableNmaes[0]);   // @CURRENT_TIME
Debug.WriteLine(variableNmaes[1]);   // @ERROR_DESCRIPTION
CCaoVariable varCurrentTime = ctrl.AddVariable("@CURRENT_TIME", "");
CCaoVariable varErrorDescription = ctrl.AddVariable(variableNmaes[1], "");
Debug.WriteLine(varCurrentTime.Value);              // 2020/01/01 12:00:00
Debug.WriteLine(varErrorDescription.Value);     // END
```

4.2.2.2. User variable specification

When accessing arbitrary data memories, the user variable is used to access them. Table 4-6 shows the specifications of User variable.

In this specification, arbitrary data memories (relay address, data register address, and byte/word register address of relay area) can be specified as variable names. The option string is not used, and the data type is VT_I2. Reading to the data memory specified by the variable name (get_Value property) and writing (put_Value property) are enabled. However, if you execute get/put with a data memory that does not exist in the variable name (for example, "X999"), the VT_BSTR displays the string "parameter error" returned by TCmini.

Any communication command can also be specified as a variable name. Do not use option strings; assume the data is of type VT_BSTR. When get_Value is executed, the message <ESC>+<variablename>+<CR> is sent according to TCmini protocols. put_Value is not supported. By enabling an arbitrary communication command in this way, the user can set commands such as "2" and "X100,X200"(X100,X200 acquiring command at the same time).

Table 4-6 User Variable Specifications Implemented by TCmini Providers

Variable name	Option string	Data Type	Description	Attribute	
				Get	Put
Data memory Ex) "X100"	None	VT_I2	Access any data memory. * If you run get/put with a nonexistent data memory, VT_BSTR returns the string "Parameter error". e.g., "X999"	○	○
COMMUNICATION COMMANDS Ex) "2,X100,X200"	None	VT_BSTR	Send arbitrary communication commands. * Executing get with an invalid communication command returns the string "Parameter error".	○	-

Usage example (C#)

```
CCaoVariable varX100 = ctrl.AddVariable("X100", "");
CCaoVariable varX10W = ctrl.AddVariable("X10W", "");
CCaoVariable varD150 = ctrl.AddVariable("D150", "");
CCaoVariable varX100X200 = ctrl.AddVariable("2,X100,X200", "");
Debug.WriteLine(varX100.Value);          // When X100 is OFF: 0
                                         // When X100 is on: 1
Debug.WriteLine(varX10W.Value);          // When X100~X10F is all OFF:-1
                                         // When X100~X10F is all ON: 0
Debug.WriteLine(varD150.Value);          // -32768~32767
Debug.WriteLine(varX100X200.Value);      // When X100,X200 is off: 0,0
```

4.3. Implementation of TCmini provider

4.3.1. Creating TCmini provider projects

Projects for TCmini providers can be easily created using CaoProvWiz.exe. Follow the instructions in section 2.3”Creating a New Provider Project” to create a TCmini provider project

Now that the VC++ project for TCmini providers is created as CaoProv.vcxproj, start VC++ with this project file.

In addition to CaoProv.vcxproj, there are several other files in the created project folder. The following table lists the files to be implemented this time.

Table 4-7 File List to Edit

No.	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
5	StdAfx.h	Precompiled header

The example code already implemented is stored in the following folder, so please read it as necessary.

<Installation folder>¥CAO¥ProviderLib¥ToshibaMachine¥TCmini¥Src

4.3.2. Adding CSerial classes

TCmini providers communicate with TCmini controllers through serial ports. Therefore, we use CSerial class, which is the communication class described in the previous chapter.

First, add the two files "Device. cpp", "Serial. cpp" to the project folder. Right-click CAOPROV project, select the [Add] → [Exisiting Item…] and add the "Device.cpp" and "Serial.cpp" files in the <Installation folder>¥ORiN2¥CAO¥ Include, as shown in Fig. 4-4.

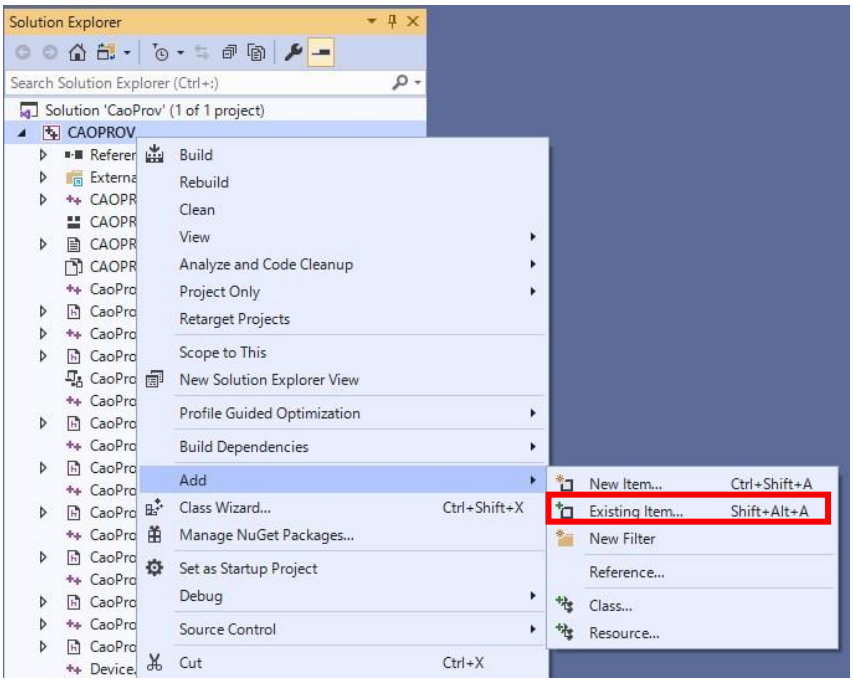


Fig. 4-4 File Add to Project Screen

Next, include the header file so that TCmini providers can use CSerial classes. Add the following to StdAfx.h file. Black shaded areas indicate the added code.

List 4-1 StdAfx.h

```
// ========== Additional functions are writed below. ==========
// ========== 各社追加部分はこれ以下に記述する. ==========

// Common include file
// 共通のインクルードファイル
#include "Serial.h"
```

4.3.3. Implementation of CCaoProvController class

4.3.3.1. Override necessary methods

To prepare your CCaoProvController classes for implementation, first override the required methods as follows:

(1) In the method described in the header file (CaoProvController. h) of CCaoProvController class, enable (uncomment) the following method.

1. HRESULT FinalInitialize()
2. HRESULT FinalConnect()
3. HRESULT FinalDisconnect()
4. void FinalTerminate()
5. HRESULT FinalGetVariableNames()

Examples of CCaoProvController::FinalInitialize() are shown below.

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

(2) Remove the comments from CaoProvController.cpp where the above method is implemented.

Examples of CCaoProvController::FinalInitialize() are shown below.

```
/*
HRESULT CCaoProvController::FinalInitialize()    // Before removing comment
{
  // Omitted
}
*/
    ↓
HRESULT CCaoProvController::FinalInitialize()    //After removing the comment
{
  // Omitted
}
```

4.3.3.2. Addtion of the necessary members

Then add the required member variables for CCaoProvController classes.

Add a variable as a member variable that holds a pointer to the object that communicates with the COM port, as follows:

m_pSerial : Pointer to the object that communicates with the COM port

List 4-2 CaoProvController.h

```
// ========== Additional functions are writed below. ==========
// ========== 各社追加関数はこれ以下に記述する. ==========
private:
CSerial *m_pSerial;
```

4.3.3.3. Implementation of FinalInitialize()

Then we implement FinalInitialize method of CaoProvController classes. This method is called when CaoWorkspace::AddController is executed. Be sure to implement this function.

This method is called when CaoProvController objects are generated. In this method, a pointer to a device object is initialized. If you have implemented the method, change the return value from E_NOTIMPL to S_OK.

List 4-3 CaoProvController.cpp – FinalInitialize()

```
HRESULT CCaoProvController::FinalInitialize()
{
    _Module.LogEvent(LOG_LEVEL_DEBUG, "CCaoProvController::FinalInitialize()¥n");

#ifdef CAOP_CANCEL_SUPPORT
    m_hProviderCancelEvent = ::CreateEvent(NULL, TRUE, FALSE, NULL);

    // Register Execute command and function pointer
    // Execute用コマンドと関数ポインタの登録
    AddFunction(L"ProviderCancel", &CCaoProvController::ExecProviderCancel);
    AddFunction(L"ProviderClear", &CCaoProvController::ExecProviderClear);
    // :
    // Sample of provider cancellation
    // プロバイダキャンセルサンプル
    AddFunction(L"ProviderCancelSampleMethod",&CCaoProvController::ExecProviderCancelSampleMe
thod);
#endif

  // Initialization of member variables
  // メンバ変数の初期化処理
    m_pSerial = NULL;

    // This function must be implemented.Perform initialization processing common to
CaoProvController objects.
    // この関数は必ず実装するCaoProvControllerオブジェクト共通の初期化処理などを行う.
    return S_OK; // After implementation, set the return value to S_OK.
                 // 実装したら返り値を S_OK にする.
}
```

4.3.3.4. Implementation of ParseConnectionString()

ParseConnectionString functions in CaoProvController.h/cpp are then edited to analyze the communication connection parameters passed during CaoWorkspace::AddController.

This time, ParseConnectionString function is implemented as a function to analyze the connection parameters and timeout [ms] of COM communication. PARAM_CONN type comParam for storing connection parameters of COM communication and DWORD type dwTimeout for storing timeout time are arguments. This function is used within FinalConnect function.

Define ParseConnectionString prototyping declarations in CaoProvController.h.

List 4-4 CaoProvController.h

```
HRESULT ParseConnectionString();          // Before change
↓
HRESULT ParseConnectionString(PARAM_CONN* comParam, DWORD* dwTimeout);      // After change
```

Place your ParseConnectionString implementations in CaoProvController. cpp. The fourth argument of AddController (option string) holds the string as m_bstrOption in CaoProvController. cpp. This function parses m_bstrOption for connection parameters and a timeout period.

GetConnectOption function of CConnectOption class is used to analyze the connection parameters, and GetOptionValue function of COptionValue class is used to analyze the timeout period.

Format

```
AddController
(
    "<Controller>",         // Controller name (optional)
    "<ProvID>",       // Provider name
    "<machine name>",     // Name of the provider's running machine
    "<options>"             // Option string
)
```

List 4-5 CaoProvController.cpp – ParseConnectionString()

```
HRESULT CCaoProvController::ParseConnectionString(PARAM_CONN* comParam, DWORD* dwTimeout)
{
    // Set initial value of communication option(Conn=)
    // 通信オプション(Conn=)の初期値設定
    PARAM_CONN_COM stComIniValue;
    stComIniValue.dwPortNo = 1;
    stComIniValue.dwBaudRate = CBR_19200;
    stComIniValue.dwParity = NOPARITY;
    stComIniValue.dwDataBits = 8;
    stComIniValue.dwStopBits = TWOSTOPBITS;
    stComIniValue.dwFlow = 0;

    CConnectOption connectOption;
    connectOption.SetDefault(stComIniValue);

    // Analysis of communication options (Conn =)
    // 通信オプション(Conn=)の解析
    HRESULT hr = connectOption.GetConnectOption(m_bstrOption, comParam);
```

```
    if (FAILED(hr)) {
        return E_NONE_OPTION;       // Option setting error
                                    // オプション設定エラー
    }

    // Set initial value of timeout option(Timeout=)
    // Timeoutオプション(Timeout=)の初期値設定
    *dwTimeout = 500;       //デフォルトタイムアウト：500[ms]

    // Analysis of timeout option(Timeout=)
    // Timeoutオプション(Timeout=)の解析
    CComVariant vntOptVal;
    hr = GetOptionValue(m_bstrOption, CComBSTR("Timeout"), VT_UI4, &vntOptVal);
    if (vntOptVal.vt == VT_UI4) {
        *dwTimeout = vntOptVal.ulVal;       // Assign set value
                                            // 設定値の代入
    }

    return hr;
}
```

4.3.3.5. Implementation of FinalConnect()

The following implements FinalConnect() method: FinalConnect handles connections. This method is called after FinalInitialize() has finished executing CaoWorkspace::AddController. Be sure to implement this function.

This function analyzes optional character strings (the fourth argument of AddController method) using ParseConnectionString function described in the previous section. When the analysis of the optional character string is completed successfully, the member variable m_pSerial is generated using the new operator, and the initialization processing (Initialize function), connection processing (Connect function), and timeout are set in that order. If you implement this function, change the return value from E_NOTIMPL to S_OK.

List 4-6 CaoProvController.cpp – FinalConnect()

```
HRESULT CCaoProvController::FinalConnect()
{
    _Module.LogEvent(LOG_LEVEL_DEBUG, "CCaoProvController::FinalConnect()¥n");

    // This function must be implemented.Write the connection process of the provider here.
    // この関数は必ず実装する．プロバイダの接続処理をここで行う．

    // Get connection parameters
    // 接続パラメータの取得
    PARAM_CONN connParam;
    DWORD dwTimeout;
    HRESULT hr = ParseConnectionString(&connParam, &dwTimeout);
    FAILED_RETURN(hr);

    // Initialization parameters(Hedder:No, Terminator:CR, Mode:Text)
    // 初期化パラメータ(ヘッダー：なし，ターミネータ：CR，動作モード：テキスト)
    CHAR cHeader[] = "";
    CHAR cTerm[] = "¥r";
```

```
    DWORD dwMode = ST_MODE_TEXT;
    PVOID pArgs[] = { cHeader,cTerm,&dwMode };

    // Create CSerial object and initialization
    // CSerialオブジェクトの生成と初期化処理
    this->m_pSerial = new CSerial();
    m_pSerial->Initialize(pArgs);

    // Connection processing
    // 接続処理
    hr = m_pSerial->Connect(connParam.stCom, dwTimeout);
    FAILED_DISCONNECT_RETURN(hr);

    // Set timeout
    // タイムアウト設定
    hr = m_pSerial->SetTimeout(dwTimeout);
    FAILED_DISCONNECT_RETURN(hr);

    return S_OK; // After implementation, set the return value to S_OK
                 // 実装したら返り値を S_OK にする.
}
```

4.3.3.6. Implementation of FinalDisconnect()

Next, FinalDisconnect() method is implemented. FinalDisconnect performs the disconnection process. This method is called first when CaoControllers::Remove is executed. Be sure to implement this function.

The following operations are performed in FinalDisconnect to disconnect the serial connection and to release the object. If you implement this function, change the return value from E_NOTIMPL to S_OK.

List 4-7 CaoProvController.cpp – FinalDisconnect()

```
HRESULT CCaoProvController::FinalDisconnect()
{
    _Module.LogEvent(LOG_LEVEL_DEBUG, "CCaoProvController::FinalDisconnect()¥n");

    // Disconnect processing
    // 切断処理
    if (m_pSerial != NULL) {
        m_pSerial->Disconnect();
        SAFE_DELETE(m_pSerial);
    }

    // This function must be implemented.Write the disconnection process of the provider here.
    // この関数は必ず実装する. プロバイダの切断処理をここで行う.
    return S_OK; // After implementation, set the return value to S_OK
                 //実装したら返り値を S_OK にする.
}
```

4.3.3.7. Implementation of FinalTerminate()

Implement FinalTerminate () method. This function is called after FinalDisconnect () is finished when CaoControllers::Remove is executed, and performs the pre-release processing of the object. Be sure to implement this function.

In this method, the variables (events, etc.) generated by initialization are closed, but since TCmini provider does not use variables such as events, there is no special implementation. Please only overload the function this time.

List 4-8 CaoProvController.cpp

```
void CCaoProvController::FinalTerminate()
{
    _Module.LogEvent(LOG_LEVEL_DEBUG, "CCaoProvController::FinalTerminate()¥n");

#ifdef CAOP_CANCEL_SUPPORT
    if (m_hProviderCancelEvent != NULL) {
        ::CloseHandle(m_hProviderCancelEvent);
        m_hProviderCancelEvent = NULL;
    }
#endif

    // This function must be implemented.Perform pre-release processing for CaoProvController
object.
    // この関数は必ず実装する CaoProvController オブジェクトの解放前処理をする.
}
```

4.3.3.8. Implementation of GetSerial()

Because COM ports are connected within CCaoProvController classes, pointers to serial connection objects (m_pSerial) are required to communicate with TCmini from CaoProvVariable objects. Therefore, we add a method GetSerial function to get the serial connection object so that COM communication can be performed from CaoVariable class.

GetSerial function is a public member function of CCaoProvController, called CaoProvController . Define to h.

List 4-9 CaoProvController.h

```
// ========== Additional functions are writed below. ==========
// ========== 各社追加関数はこれ以下に記述する. ==========
private:
CSerial *m_pSerial;

public:
HRESULT GetSerial(CSerial** ppSerial);
```

The implementation of GetSerial functions is described in CaoProvController.cpp.

List 4-10 CaoProvController.cpp – GetSerial()

```
HRESULT CCaoProvController::GetSerial(CSerial** ppSerial)
{
    *ppSerial = m_pSerial;
    return S_OK;
}
```

4.3.3.9. Implementation of FinalGetVariableNames()

TCmini providers allow you to get the five System variables shown in Table 4-5. "@MAKER_NAME" and "@VERSION" are already created in the default templates, so implement the other three System variables.

The 'L' before the string is a VC++ proprietary extension to indicate that the next string is to be treated as a wide string (UNICODE).

List 4-11 StdAfx.h-macro definition

```
// Variable name of Controller class
// Controllerクラスの変数名
#define CS_MAKER_NAME            0x0001
#define CS_MAKER_NAME$           L"@MAKER_NAME"
#define CS_VERSION               0x0002
#define CS_VERSION$              L"@VERSION"

// ========== Additional functions are writed below. ==========
// ========== 各社追加部分はこれ以下に記述する. ==========

// Common include file
// 共通のインクルードファイル
#include "Serial.h"

// Variable name of Controller class
// Controllerクラスの変数名
#define CS_CURRENT_TIME          0x0003
#define CS_CURRENT_TIME$         L"@CURRENT_TIME"
#define CS_ERROR_DESCRIPTION     0x0004
#define CS_ERROR_DESCRIPTION$    L"@ERROR_DESCRIPTION"
#define CS_TYPE                  0x0005
#define CS_TYPE$                 L"@TYPE"

// Response result of system variable
// システム変数の返答結果
#define MAKER_NAME          L"Shibaura Machine"
#define PRODUCT_TYPE        L"TCmini"
```

Implement the method FinalGetVaribaleNames() to get a list of defined system variable names: Because there are five System variables, change nDefines value to nDefines=5.

List 4-12 CaoProvController.cpp – FinalGetVariableNames()

```
HRESULT CCaoProvController::FinalGetVariableNames(BSTR bstrOption, VARIANT* pVal)
{
  // Declaring the number of variable names as nDefines. * Correct nDefines if the number of variable names is changed.
  // 変数名の数をnDefinesとして宣言 *変数名の数を変更した場合は修正が必要
  const int    nDefines = 5;

  pVal->vt = VT_ARRAY | VT_VARIANT;
  SAFEARRAY* psa;
  psa = SafeArrayCreateVector(VT_VARIANT, 0, nDefines);

  VARIANT* vntArray;

  SafeArrayAccessData(psa, (void**)&vntArray);

  int i = 0;

    // @CURRENT_TIME
  vntArray[i].vt = VT_BSTR;
  vntArray[i].bstrVal = SysAllocString(CS_CURRENT_TIME$);
  i++;

    // @ERROR_DESCRIPTION
  vntArray[i].vt = VT_BSTR;
  vntArray[i].bstrVal = SysAllocString(CS_ERROR_DESCRIPTION$);
  i++;

    // @MAKER_NAME
  vntArray[i].vt = VT_BSTR;
  vntArray[i].bstrVal = SysAllocString(CS_MAKER_NAME$);
  i++;

    // @TYPE
  vntArray[i].vt = VT_BSTR;
  vntArray[i].bstrVal = SysAllocString(CS_TYPE$);
  i++;

    // @VERSION
  vntArray[i].vt = VT_BSTR;
  vntArray[i].bstrVal = SysAllocString(CS_VERSION$);
  i++;

  ATLTRACE("nDefines = %d¥n", i);
  ATLASSERT(i == nDefines);

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

  return S_OK;
}
```

4.3.4. Implementation of CCaoProvVariable class

4.3.4.1. Override necessary methods

To prepare CCaoProvVariable classes for implementation, first override the required FinalInitialize, FinalGetValue, and FinalPutValue methods as follows:

(1) Uncomment the following method in the header file (CaoProvVariable. h) of CCaoProvVariable class.

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

(2) Remove the comments from the point (CaoProvVariable.cpp) where the above method is implemented.

4.3.4.2. Addtion of necessary constants/members

The constants used in CaoProvVariable.cpp are defined in CaoProvVariable.h.

Constant name	Value	Explanation
CMD_EC	0x1B	Command start code
CMD_CR	0x0D	Command end code
USER_VARIABLE_LENGTH	4	User variable name's length
VAR_COMMAND_MAX	256	Maximum number of characters in command

When CaoProvVariable objects are created, the names of User variable passed by clients can be pre-analyzed and quantified to simplify further processing. Define the following member functions as functions for parsing user variable names:

```
HRESULT ParseUserVariableName()
```

Also, to keep the results of parsing user variable names in the class, add the following enumeration and member variables:

・ Data type information (enumerator) for relays (BITs), registers (WORD), or others

```
Typedef enum {VAR_TYPE_UNKNOWN, VAR_TYPE_BIT, VAR_TYPE_WORD} VAR_DATA_TYPE;
```

・ Member variables that hold data type information

```
VAR_DATA_TYPE m_usrDataType
```

・ Copying a Variable Name Converted to Uppercase

Because the original user variable name (m_bstrName) can be mixed case, define a variable name that has been converted to all uppercase letters.

```
CComBSTR m_bstrUpperName
```

It also adds pointers to serial communication objects as member variables of CCaoProvVariable class so that serial communication can be performed from CaoProvVariable class.

```
CSerial *m_pSerial
```

Describe these definitions in CaoProvVariable. h.

List 4-13 CaoProvVariable.h

```
#ifndef __CAOPROVVARIABLE_H_
#define __CAOPROVVARIABLE_H_

#include "CaoProvVariableImpl.h"

// TCmini's command : <EC> + <CMD> + ',' + <DATA> + <CR>
// CMD = ASCII numbers from '0' to '31'
// CMD = '0'～'31'までのASCII数字
#define CMD_EC  0x1B                    // Command start code
                                        // コマンドの先頭コード
#define CMD_CR  0x0D                    // Command end code
                                        // コマンドの終了コード

#define USER_VARIABLE_LENGTH    4       // User Variable name length
                                        // ユーザ変数名の長さ
#define VAR_COMMAND_MAX 256             // Maximum number of characters in command
                                        // コマンドの最大文字数

// TCmini data type (Relay: BIT, Register: WORD)
// TCminiのデータ型(リレー：BIT, レジスタ：WORD)
typedef enum { VAR_TYPE_UNKNOWN, VAR_TYPE_BIT, VAR_TYPE_WORD } VAR_DATA_TYPE;

:
:

// ========== Additional functions are writed below. ==========
// ========== 各社追加関数はこれ以下に記述する. ==========
private:
    HRESULT ParseUserVariableName();            // User variable name parsing function
                                                // ユーザー変数名の解析関数

    CSerial *m_pSerial;                         // Pointer to serial communication object
                                                // シリアル通信オブジェクトへのポインタ

    VAR_DATA_TYPE m_usrDataType;                // Variable that stores the data type of the
variable name
                                                // 変数名のデータ型を格納する変数
    CComBSTR m_bstrUpperName;                   // Variable name with only uppercase letters
                                                // 大文字のみの変数名

};
```

4.3.4.3. Implementation of ParseUserVariableName()

Describe the implementation of the defined member function in CaoProvVariable.cpp. The first argument (variable name) of CCaoProvController::AddVariable () is stored in the member variable m_bstrName, and when m_bstrName is the user variable name, the ParseUserVariableName function analyzes the string. The option string is stored in m_bstrOption, but it is not used in the TCmini provider implemented this time.

Format

```
AddVariable
(
    "<variable name>"     // Variable name
    "<options>"           // Option string (not used)
)
```

First, copy the variable name converted to upper case to m_bstrUpperName.It then refers to the first character of the resulting variable name and branches processing according to whether it is a relay type (X,Y,Z,R,E,L,T,C,A) or register type (D,P,V) or some other type. For the register type, m_usrDataType = VAR_TYPE_WORD;. For an iterator type, refer to the last character (the fourth character) to specify 'L', 'H', and if 'W', specify the byte/word addressing of the relay area, so VAR_TYPE_WORD; and otherwise specify VAR_TYPE_BIT. The other types are assumed to be VAR_TYPE_UNKNOWN.

List 4-14 CaoProvVariable.cpp – ParseUserVariableName()

```
HRESULT CCaoProvVariable::ParseUserVariableName()
{
// Add variable name to m_bstrUpperName
// m_bstrUpperNameに変数名を追加
HRESULT hr = m_bstrUpperName.AppendBSTR(m_bstrName);
FAILED_RETURN(hr);

// Convert to uppercase
// 大文字に変換
hr = m_bstrUpperName.ToUpper();
FAILED_RETURN(hr);

// Get string length
// 文字列の長さを取得
int iVarLength = m_bstrUpperName.Length();

// Determine data type
// データタイプを判定
switch (m_bstrUpperName[0]) {
// When the first character is X, Y, Z, R, E, L, T, C, A
// 先頭1文字目がX, Y, Z, R, E, L, T, C, Aの場合
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':
        // Check address string length (argument error except 4 characters)
        // アドレス文字列長の検査(4文字以外は引数エラー)
        if (iVarLength != USER_VARIABLE_LENGTH) {
            return E_INVALIDARG;
        }
        // Judge the last character. If the suffix is 'W', 'L', 'H', the data type is word type.
Otherwise, bit type
        // 最後の文字を判定. 語尾に'W','L','H'が付く場合, データタイプはワード型. それ以外はビ
ット型
        switch (m_bstrUpperName[iVarLength - 1]) {
        case L'W':
        case L'H':
        case L'L':
            m_usrDataType = VAR_TYPE_WORD;
            break;
        default:
            m_usrDataType = VAR_TYPE_BIT;
            break;
        }
        break;

    // If the first character is D, P, V, word type.
    // 先頭1文字目がD, P, Vの場合, ワード型
    case L'D':
    case L'P':
    case L'V':
        // Check address string length (argument error except 4 characters)
        // アドレス文字列長の検査(4文字以外は引数エラー)
        if (iVarLength != USER_VARIABLE_LENGTH) {
            return E_INVALIDARG;
        }
        m_usrDataType = VAR_TYPE_WORD;
        break;

    // Other than the above are identified as communication commands
    // 上記以外は通信コマンドとして判別
    default:
        // ex. "2,XOOO,YOO7,TOO2"
        m_usrDataType = VAR_TYPE_UNKNOWN;
        break;
    }
    return S_OK;
}
```

4.3.4.4. Implementation of InitMapTable()

Next, implement the default-defined member function InitMapTable. InitMapTable registers the system variable name in the member variable m_cs_map, where m_cs_map is a variable name list that contains the system variable and is used to check that the variable name passed at AddVariable run time matches the registered system variable.

By implementing the following, AddVariable will fail when the system variable name is a character string other than @MAKER_NAME, @VERSION, @CURRENT_TIME, @ERROR_DESCRIPTION, @TYPE.

List 4-15 CaoProvVariable.cpp – InitMapTable()

```
HRESULT  CCaoProvVariable::InitMapTable()
{
if (m_bInitializedMap) return S_FALSE;

    // Initialize variable name map
// 変数名マップの初期化
const var_map_entry var_cs_map[] = {
    MAP_ENTRY( CS_MAKER_NAME  ),
    MAP_ENTRY( CS_VERSION ),
    MAP_ENTRY( CS_CURRENT_TIME  ),
    MAP_ENTRY( CS_ERROR_DESCRIPTION  ),
    MAP_ENTRY( CS_TYPE  ),
    // :

    };
m_cs_map.insert( var_cs_map, var_cs_map +sizeof(var_cs_map)/sizeof(var_cs_map[0]) );

m_bInitializedMap = true;

return S_OK;
}
```

4.3.4.5. Implementation of FinalInitialize()

Implement the FinalInitialize () method. This method is called when CCaoProvController::AddVariable() method is executed to initialize CCaoVariable classes.FinalInitialize () initializes the member variable defined in CCaoProvVariable class. Also, the parent object m_ulParentType is discriminated and the variable name is analyzed. In the default template, the program that analyzes the variable name when the parent object is CCaoProvController class is described.

If the variable name is a system variable (the first string is "@"), m_bSystem becomes true. Compare with the variable name registered in.m_cs_map and assign the identification number of the system variable to m_lUSysId if it matches. The identification number of the.m_lUSysId is used to GET/PUT the system variable.

TCmini providers use the default templates as they create Variable classes only in Controller classes. Use defined ParseUserVariableName functions to analyze User variable. If the variable name is a valid system or user variable, hr = S_OK. The address of the connected serial communication object is assigned to the member variable m_pSerial so that communication commands can be executed from CCaoVariable class. You can then communicate with TCmini by executing the m_pSerial method.

List 4-16 CaoProvVariable.cpp – FinalInitialize()

```
HRESULT CCaoProvVariable::FinalInitialize(PVOID pObj)
{
  HRESULT hr = E_INVALIDARG;

    // MapTable initialization
  // MapTableの初期化
  InitMapTable();

  CCaoProvController* pCaopCtrl = NULL;
  CCaoProvExtension* pCaopExp = NULL;
  CCaoProvFile* pCaopFile = NULL;
  CCaoProvRobot* pCaopRobot = NULL;
  CCaoProvTask* pCaopTask = NULL;

    // Initialize data member
  // データメンバーの初期化
  m_lUSysId = 0;
  m_usrDataType = VAR_TYPE_UNKNOWN;
  m_bstrUpperName.Empty();
  m_pSerial = NULL;

    // Determine parent object
  // 親オブジェクトの判定
  switch (m_ulParentType) {
  case SYS_CLS_CONTROLLER:
      pCaopCtrl = (CCaoProvController*)pObj;
      if (m_bSystem) {  // System variable
                              // システム変数
           // ID search from variable name
          // 変数名からID検索
          var_map::iterator it;
          it = m_cs_map.find(m_bstrName);
          if (it != m_cs_map.end()){
              m_lUSysId = (it->second);
              hr = S_OK;
          } else {
              hr = E_INVALIDARG;
          }
      }
      else {   // User variable
               // ユーザー変数
          hr = ParseUserVariableName();       // m_bstrNameの解析
      }

      // シリアル通信オブジェクトの取得
      if (hr == S_OK) {
          pCaopCtrl->GetSerial(&m_pSerial);
      }

      break;
  case SYS_CLS_EXTENSION:
      pCaopExp = (CCaoProvExtension*)pObj;
      break;
  case SYS_CLS_FILE:
      pCaopFile = (CCaoProvFile*)pObj;
      break;
  case SYS_CLS_ROBOT:
      pCaopRobot = (CCaoProvRobot*)pObj;
      break;
  case SYS_CLS_TASK:
```

```
        pCaopTask = (CCaoProvTask*)pObj;
        break;
    }

    return hr;
}
```

4.3.4.6. Implementation of FinalGetValue()

The next step is to implement FinalGetValue() method: CaoVariable::get_Value executes this function. The default templates implement the case where the parent objects are CCaoController classes. System variable execute FinalGetCtrlSysValue functions, and User variable execute FinalGetCtrlUserValue functions.

Now that TCmini providers assume that their parent objects are CCaoController classes only, we will use them without changing the default templates and only modify FinalGetCtrlSysValue and FinalGetCtrlUserValue functions.

List 4-17 CaoProvVariable.cpp – FinalGetValue()

```
HRESULT CCaoProvVariable::FinalGetValue(VARIANT *pVal)
{
    HRESULT hr = E_ACCESSDENIED;

    if (m_bSystem) {
        switch (m_ulParentType) {
        case SYS_CLS_CONTROLLER:
            hr = FinalGetCtrlSysValue(pVal);
            break;
        }
    }
    else {
        switch (m_ulParentType) {
        case SYS_CLS_CONTROLLER:
            hr = FinalGetCtrlUserValue(pVal);
            break;
        }
    }

    return hr;
}
```

4.3.4.7. Implementation of FinalGetCtrlSysValue()

Describe the macro definitions to be used by FinalGetCtrlSysValue functions in StdAfx.h. First, describe the macro definitions for the System variable in StdAfx.h: Since the two System variables @MAKER_NAME and @TYPE always return constants, the result is macro-defined in StdAfx. h.

List 4-18 StdAfx.h - Macro definition

```
// ========== Additional functions are writed below. ==========
// ========== 各社追加部分はこれ以下に記述する. ==========

// Common include file
// 共通のインクルードファイル
#include "Serial.h"

// Variable name of Controller class
// Controllerクラスの変数名
#define CS_CURRENT_TIME          0x0003
#define CS_CURRENT_TIME$         L"@CURRENT_TIME"
#define CS_ERROR_DESCRIPTION     0x0004
#define CS_ERROR_DESCRIPTION$    L"@ERROR_DESCRIPTION"
#define CS_TYPE                  0x0005
#define CS_TYPE$                 L"@TYPE"

// Response result of system variable
// システム変数の返答結果
#define MAKER_NAME           L"Shibaura Machine"
#define PRODUCT_TYPE         L"TCmini"
```

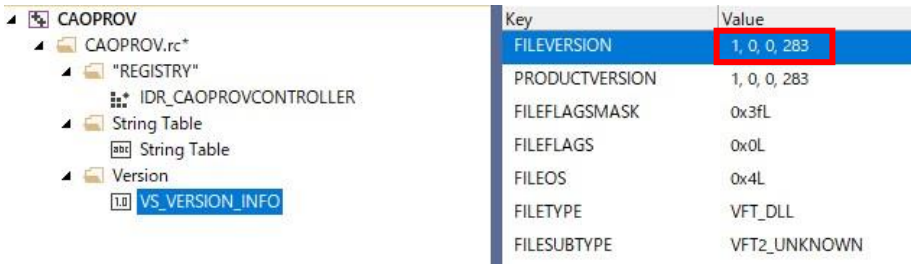
TCmini providers enable you to get five types of System variables. Therefore, FinalGetCtrlSysValue function adds processing to each system variable, as shown below. By entering a value in the return value pVal, the result of executing GetValue is notified to the user.

1. Support for "@MAKER_NAME"

   Since it corresponds to CS_MAKER_NAME of switch-case syntax, add processing to return "Shibaura Machine" as BSTR type as specified.

2. Support for "@VERSION"

   It corresponds to switch-case form of CS_VERSION and executes GetDllVersion function. This is a three-digit number (the major number) as described in FILEVERSION to CAOPROV. rc. Minor number. Assign the revision number to pVal.



3. Support for "@CURRENT_TIME"

Adds an action to refer to the calendars in TCmini controllers and return the current time, corresponding to the CS_CURRENT_TIME switch-case construct.

In TCmini controllers, the present time can be obtained by referring to D120-D126 register. However, if A009 = 1 is not set, the time data cannot be acquired correctly, so debugging requires attention.

Use the <CMD>"5" (0x35) command to read all D120-D126 register data. Returns the current time returned by this command as a VT_DATA type.

4. Support for "@ERROR_DESCRIPTION"

Add handling to respond to switch-case construct CS_ERROR_DESCRIPTION to return informational errors in TCmini controllers.

The controller error information can be acquired by using the <CMD> "1" (0x31) command. Returns BSTR type of the string returned by this function.

5. Support for "@TYPE"

Since it corresponds to CS_TYPE of switch-case syntax, add processing to return "TCmini" as BSTR type as specified.

To use the string type variable szCmdBuf, it is necessary to describe "using namespace std;" at the beginning of the file.

In addition, because the initialization setting is made so that "¥ r" (<CR>) is added to the send / receive data when the CSerial class is initialized, it is not necessary to add <CR> to the end of szCmdBuf. .

List 4-19 CaoProvVariable.cpp – FinalGetCtrlSysValue()

```
#include "stdafx.h"
#include "CAOPROV.h"
#include "CaoProvController.h"
#include "CaoProvVariable.h"

using namespace std;
:
:

HRESULT CCaoProvVariable::FinalGetCtrlSysValue(VARIANT *pVal)
{
  HRESULT hr = E_FAIL;
  string szCmdBuf;                    // Send buffer
                                      // 送信バッファ

  szCmdBuf = CMD_EC;                  // Command start code
                                      // コマンド開始コード

  char szDataBuf[VAR_COMMAND_MAX];    // Receive buffer
                                      // 受信バッファ
```

```
    DWORD dwDataLen;                    // Receive size
                                            // 受信サイズ

    switch (m_lUSysId) {
    // @MAKER_NAME
    case CS_MAKER_NAME:
        pVal->vt = VT_BSTR;
        pVal->bstrVal = SysAllocString(MAKER_NAME);
        hr = S_OK;
        break;

    // @VERSION
    case CS_VERSION:
        hr = GetDllVersion(pVal);
        break;

    // @CURRENT_TIME
    case CS_CURRENT_TIME:
        // Creating a send command string
        // 送信コマンド文字列の作成
        // ----------
        // Get time information(D126~120) using the command ([5,]) that receives data collectively
in word units.(Note) Time information cannot be got unless A009 = 1 is set.
        // 時刻情報(D126～D120)をデーター括受信コマンド([5,])を使って取得.(注意) A009 = 1 がセ
ットされていないと時刻情報は取得できない
        //
        //   <EC>+"5,D126,D125,D124,D123,D122,D121,D120"+<CR>
        szCmdBuf += "5,D126,D125,D124,D123,D122,D121,D120";

        // Clear receive buffer
        // 受信バッファのクリア
        m_pSerial->Clear();

        // Send command and receive response
        // コマンド送信と応答受信
        hr = m_pSerial->SendAndReceive((LPBYTE)szCmdBuf.c_str(), 0, (LPBYTE)szDataBuf,
sizeof(szDataBuf), &dwDataLen);
        FAILED_RETURN(hr);

        // Analysis of received data
        // 受信データの解析
        // ----------
        // D126:Day of week(00-06 = Sunday-Saturday)
        // D125:Year
        // D124:Month
        // D123:Day
        // D122:Hour
        // D121:Minute
        // D120:Second
        WORD wNum, wYear, wMonth, wDay, wHour, wMinute, wSecond, wDayOfWeek;
        wNum = sscanf(szDataBuf, "%hd,%hd,%hd,%hd,%hd,%hd,%hd", &wDayOfWeek, &wYear, &wMonth,
&wDay, &wHour, &wMinute, &wSecond);
        if (wNum != 7) {
            return E_UNEXPECTED;  // When it wasn't got correctly
                                        // 正しく取得できなかった場合
        }

        // Year : wYear=02 means 2002
        // 年 : wYear=02 -> 2002年 なので
        wYear += 2000;
```

```
            SYSTEMTIME tm;
            tm.wYear = wYear;
            tm.wMonth = wMonth;
            tm.wDay = wDay;
            tm.wHour = wHour;
            tm.wMinute = wMinute;
            tm.wSecond = wSecond;
            tm.wDayOfWeek = wDayOfWeek;
            tm.wMilliseconds = 0;

            // Convert to VT_DATE and set result
            // DATE型に変換と結果の格納
            if (!SystemTimeToVariantTime(&tm, &(pVal->date))) {
                return E_UNEXPECTED;
            }
            pVal->vt = VT_DATE;

            break;

        // @ERROR_DESCRIPTION
        case CS_ERROR_DESCRIPTION:
            // Creating a send command string
            // 送信コマンド文字列の作成
            // ----------
            // Get alarm code
            // アラームコードを取得する
            //
            //  <EC>+"1"+<CR>
            szCmdBuf += "1";  // <CMD> = '1'

            // Clear receive buffer
            // 受信バッファのクリア
            m_pSerial->Clear();

            // Send command and receive response
            // コマンド送信と応答受信
            hr = m_pSerial->SendAndReceive((LPBYTE)szCmdBuf.c_str(), 0, (LPBYTE)szDataBuf,
    sizeof(szDataBuf), &dwDataLen);
            FAILED_RETURN(hr);

            // Set results
            // 結果の格納
            pVal->vt = VT_BSTR;
            pVal->bstrVal = CComBSTR(dwDataLen, szDataBuf).Detach();
            break;

        // @TYPE
        case CS_TYPE:
            // Set "TCmini"
            // "TCmini"を格納
            pVal->vt = VT_BSTR;
            pVal->bstrVal = SysAllocString(PRODUCT_TYPE);
            break;

        default:
            hr = E_ACCESSDENIED;
            break;
        }
```

```
    return hr;
}
```

4.3.4.8. Implementation of FinalGetCtrlUserValue()

FinalGetCtrlUserValue function is implemented as follows. Define szCmdBuf as a send buffer and prepend <EC> codes to the command string according to TCmini protocols.

If the data type is VAR_TYPE_BIT, store "2," + "variable name" to execute one point unit of I/O read, and if the data type is VAR_TYPE_WORD, store "5," + "variable name" in the send buffer to execute one word unit of data read. If the data type is VAR_TYPE_UNKNOWN, the variable name is copied to the send buffer as is.

Call the m_pSerial->Clear() function to clear the receive buffer before actually communicating with TCmini, because the serial communication receive buffer may contain the previous data. After the buffer is cleared, send the string of the send buffer szCmdBuf created using the m_pSerial->SendAndReceive function and receive the response from TCmini. At this time, the terminator <CR> codes are automatically appended by CSerial classes, and when they are received, they are deleted and a reply is returned.

For VAR_TYPE_BIT and VAR_TYPE_WORD, the received string is converted to a numeric value. For VAR_TYPE_UNKNOWN, the received string is returned. By putting a value in the return value pVal in this way, the result of executing GetValue is notified to the user.

List 4-20 CaoProvVariable.cpp – FinalGetCtrlUserValue()

```
HRESULT CCaoProvVariable::FinalGetCtrlUserValue(VARIANT *pVal)
{
  HRESULT hr = E_FAIL;
  string szCmdBuf;                // Send buffer
                                  // 送信バッファ

  szCmdBuf = CMD_EC;              // Command start code
                                  // コマンド開始コード

  char szDataBuf[VAR_COMMAND_MAX];  // Receive buffer
                                    // 受信バッファ

  DWORD dwDataLen;                // Receive size
                                  // 受信サイズ

  // Perform processing by data type
  // データタイプ別に処理を実行
  switch (m_usrDataType) {
  case VAR_TYPE_BIT:
       // BIT read command([2,])
       // BIT単位の読み出しコマンド([2,])
       szCmdBuf += "2,"; // <CMD> = '2'
       break;

  case VAR_TYPE_WORD:
       // WORD read command([5,])
```

```
            // WORD単位の読み出しコマンド([5,])  →  応答は 10進数
            szCmdBuf += "5,"; // <CMD> = '5'
            break;

        case VAR_TYPE_UNKNOWN:
            // Send as a communication command.
            // 通信コマンドとしてそのまま送信.
            break;
        }

        // Add variable name to send command buffer
        // 変数名を送信コマンドバッファに追加
        szCmdBuf += ConvertBSTRToString(m_bstrUpperName);

        // Clear receive buffer
        // 受信バッファのクリア
        m_pSerial->Clear();

        // Send command and receive response
        // コマンド送信と応答受信
        hr = m_pSerial->SendAndReceive((LPBYTE)szCmdBuf.c_str(), 0, (LPBYTE)szDataBuf,
    sizeof(szDataBuf), &dwDataLen);
        FAILED_RETURN(hr);

        // Convert response string to data
        // 応答文字列からデータへ変換
        BOOL bError = FALSE;
        switch (m_usrDataType) {
        case VAR_TYPE_BIT:
        case VAR_TYPE_WORD:
            pVal->vt = VT_I2;
            pVal->iVal = (short)atoi(szDataBuf);

            // When conversion fails(atoi function returns 0)
            // 値変換失敗の時(atoi関数は0を返却)
            if (pVal->iVal == 0 && szDataBuf[0] != '0') {
                bError = TRUE;
            }
            break;

        case VAR_TYPE_UNKNOWN:
            bError = TRUE;
            break;
        }

        // When conversion fails or VAR_TYPE_UNKNOWN type
        // 変換失敗時またはVAR_TYPE_UNKNOWN型の場合
        if (bError) {
            pVal->vt = VT_BSTR;
            pVal->bstrVal = CComBSTR(dwDataLen, szDataBuf).Detach();
        }

        return hr;
    }
```

4.3.4.9. Implementation of FinalPutValue ()

Finally, we implement FinalPutValue() method. When CaoVariable::put_Value is executed, it is executed and the argument newVal contains the value set by the user.

Format

```
Put_Value
(
    "Setting Value"
)
```

The default templates implement the case where the parent objects are CCaoController classes. System variable execute FinalPutCtrlSysValue functions, and User variable execute FinalPutCtrlUserValue functions.

Now, TCmini providers assume that the parent objects are CCaoController classes only, so they should be used without changing the default templates. Also, TCmini providers do not implement FinalPutCtrlSysValue function because the system variable put_Value does not correspond to the system variable, so they implement only FinalPutCtrlUserValue function.

List 4-21 CaoProvVariable.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;
}
```

4.3.4.10. Implementation of FinalPutCtrlUserValue()

In the case of VAR_TYPE_UNKNOWN (communication command), Put_Value does not work, so it returns an error.

Define szCmdBuf as a send buffer, and prepend <EC> codes to the command string according to TCmini protocol. The value of argument newVal is converted to VT_I2 and stored in sSetValue. If the conversion fails, an error is returned at that point.

When VAR_TYPE_BIT and sSetValue is not 0, it is determined that "8," + variable name" is an I/O forced reset when sSetValue is 0, and "9," + variable name" is put in the send buffer. When VAR_TYPE_WORD, it is executed by one word of data change, so put "10," + variable name + "," and "+ sSetValue" in the send buffer.

Call the m_pSerial->Clear() function to clear the receive buffer before actually communicating with TCmini, because the serial communication receive buffer may contain the previous data. After the buffer is cleared, send the string of the send buffer szCmdBuf created using the m_pSerial->SendAndReceive function and receive the response from TCmini. If the response string is "OK", it is judged that the setting of the value was completed successfully.

List 4-22 CaoProvVariable.cpp – FinalPutValue()

```
HRESULT CCaoProvVariable::FinalPutCtrlUserValue(VARIANT newVal)
{
  // VAR_TYPE_UNKNOWN type PUT is not implemented
  // VAR_TYPE_UNKNOWN型のPUTは未実装
  if (m_usrDataType == VAR_TYPE_UNKNOWN)
  {
      return E_NOTIMPL;
  }

  HRESULT hr = E_FAIL;
  string szCmdBuf;                // Send buffer
                                       // 送信バッファ

  szCmdBuf = CMD_EC;                   // Command start code
                                       // コマンド開始コード

  char szDataBuf[VAR_COMMAND_MAX];     // Receive buffer
                                           // 受信バッファ

  DWORD dwDataLen;                     // Receive size
                                           // 受信サイズ

  // Convert set value to VT_I2 type
  // 設定値をVT_I2型に変換
  CComVariant vntVal;
  hr = vntVal.ChangeType(VT_I2, &newVal);
  FAILED_RETURN(hr);
  short sSetValue = vntVal.iVal;

  // Creating a send command string
  // 送信コマンド文字列の作成
  switch (m_usrDataType) {
  case VAR_TYPE_BIT:
       // Set of BIT units…[8,], Reset of BIT units… [9,]
       // BIT単位のセット… [8,], BIT単位のリセット… [9,]
       szCmdBuf += (sSetValue ? "8," : "9,");       // <CMD> = '8'-ON , '9'-OFF
       szCmdBuf += ConvertBSTRToString(m_bstrUpperName);
       break;
```

```
    case VAR_TYPE_WORD:
        // Set of WORD units…[10,]
        // WORD単位の書き込み…[10,]
        szCmdBuf += "10,";                          // <CMD> = '10'
        szCmdBuf += ConvertBSTRToString(m_bstrUpperName);
        szCmdBuf += (',' + to_string((LONGLONG)sSetValue));
        break;
    }

    // Clear receive buffer
    // 受信バッファのクリア
    m_pSerial->Clear();

    // Send command and receive response
    // コマンド送信と応答受信
    hr = m_pSerial->SendAndReceive((LPBYTE)szCmdBuf.c_str(), 0, (LPBYTE)szDataBuf,
sizeof(szDataBuf), &dwDataLen);
    FAILED_RETURN(hr);

    // Normal if the response string is "OK"
    // 応答文字列が"OK"なら正常
    if (strcmp(szDataBuf, "OK")) {
        hr = E_FAIL;
    }

    return hr;
}
```

4.3.5. Create Dll

Coding of TCmini providers is now complete. Finally, from the menus, run [Build Solution] to create CaoProvTCmini.dll.

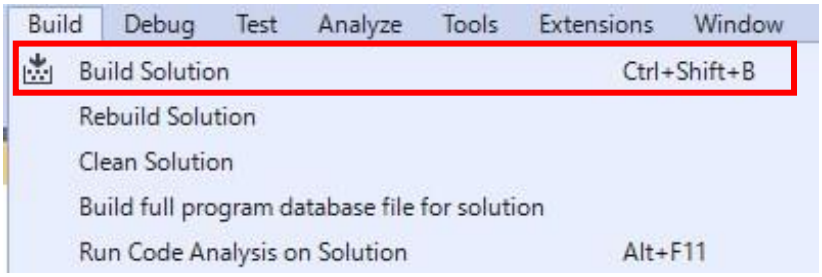


Fig. 4-5 Building TCmini.DLL

4.4. Debugging and releasing TCmini Provider

4.4.1. Debugging TCmini Provider

The provider's DLL modules are structured as needed from CAO engine “CAO.exe”. Therefore, when debugging a provider DLL module, you must specify CAO.exe in the executable file in the debug section of VC++.

Debugging TCmini providers involves the following steps: Use ORiN2 CAO-standard tool, ORiN2¥CAO¥Tools¥CaoTester2.exe, as a client application.

The PC and TCmini controller must be connected via a serial cable and the controller must be powered on prior to starting debugging.

(1) Switch the build target to [Debug_x86].

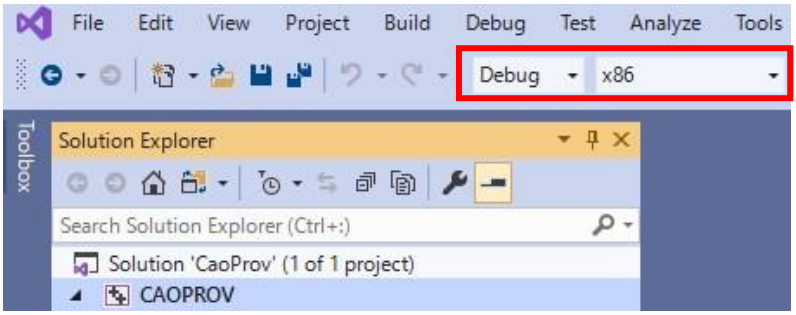


Fig. 4-6 Build Target Specification Screen

(2) Specify CAO.exe in the executable file of the debug section.

Open CAOPROV properties page, select [Configuration]:Debug, [Platform]:Win32, and then in the [Configuration Properties] → [Debugging] → [Command], specify CAO.exe in its Absolute path. CAO.exe is usually located in <Installation folder>¥ORiN2¥CAO¥Engine¥Bin folder.

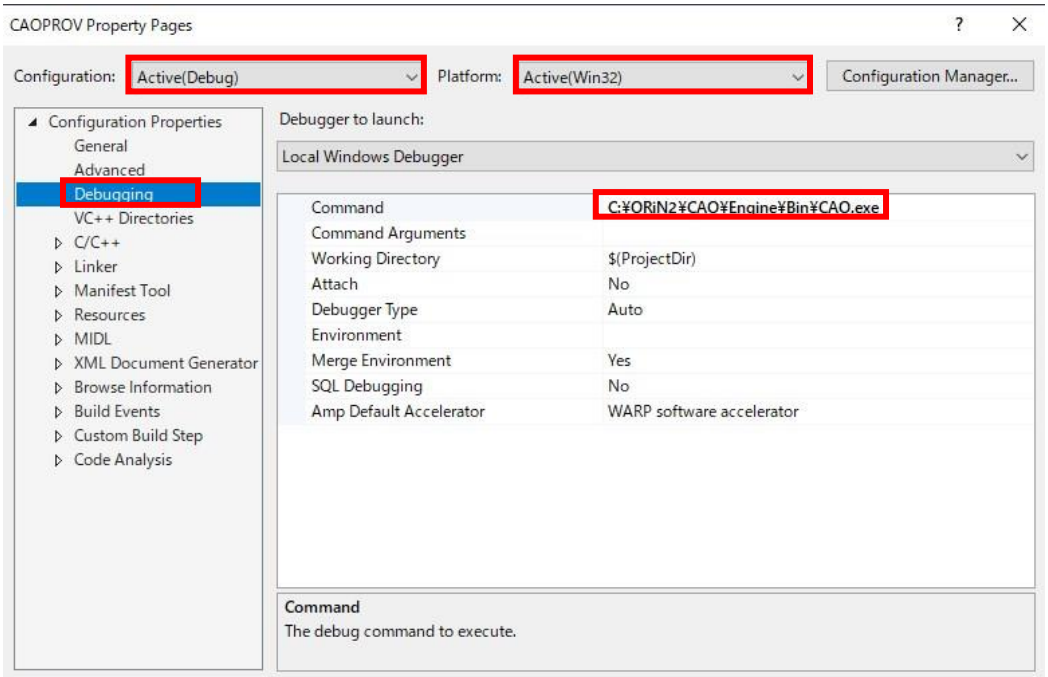


Fig. 4-7 Specifying CAOPROV Properties

(3) Set a breakpoint at the line you want to debug.

You can select the lines you want to debug and set a breakpoint with F9. The following are examples of setting breakpoints in FinalGetCtrlUserValue of CCaoVariable classes.

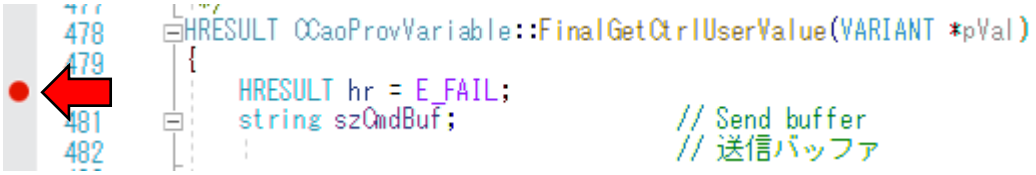


Fig. 4-8 Breakpoints

(4) Start debugging.

From the [Debug] menu, select [Start Debugging].

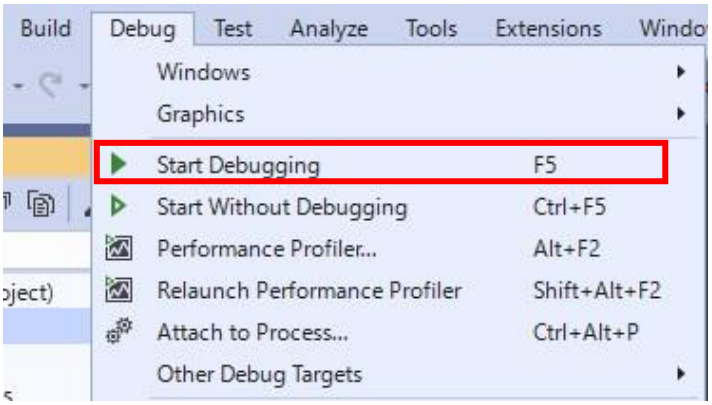


Fig. 4-9 Debug start screen

(5) Start CaoTester2.exe and make the following settings.

Controller Name: “Sample” (any text)

Provider Name: "CaoProv.TCmini"

Option:"Conn=com:1 (to connect with COM:1)

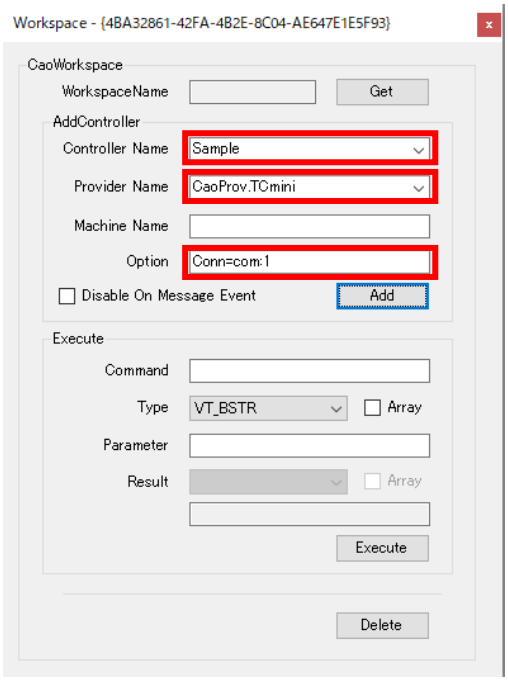


Fig. 4-10 CaoWorkspace Display of CaoTester2

(6) [Add] Press the button to start the connection with COM:1.

At this time, CaoWorkspace::AddController method is executed and is called in the order of FinalInitialize() method →FinalConnect() method, which is implemented in CCaoProvController class. Successful execution of these functions switches to the following screen.

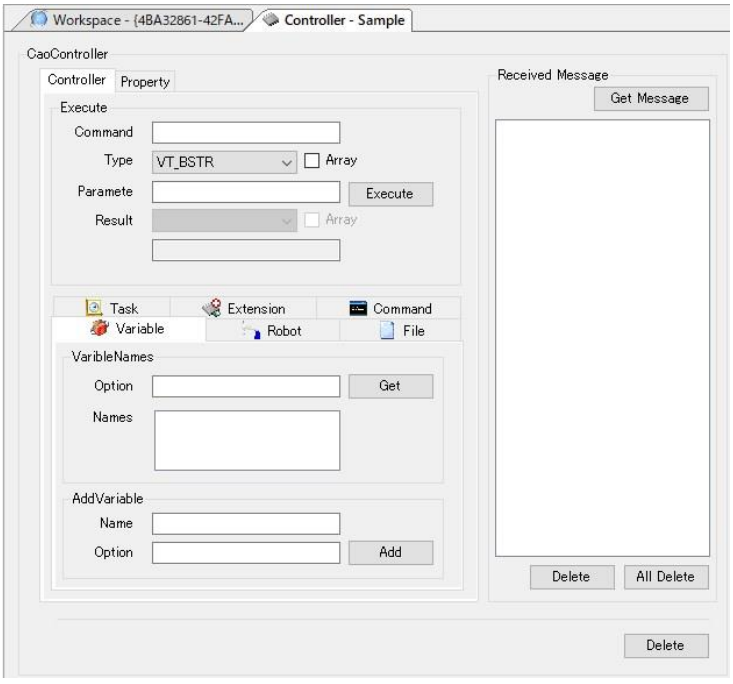


Fig. 4-11 CaoController Classes Window

(7) [Add] Click to create objects of Variable classes.

<System variable>

Select Variable tab and click Get on VariableNames. At this time, CaoController::get_VariableNames is executed, and FinalGetVariableNames() method implemented in CCaoProvController class is called. When the function finishes executing, you can list the System variables in Names part.

Selecting a system variable from the system variable list automatically populates Name of AddVariable with the system variable. If you press [Add], CaoController::AddVariable method will be executed. When AddVariable is executed, FinalInitialize() of CCaoProvVariable will be called. Upon successful completion, objects of Variable classes are generated and the screens are switched.

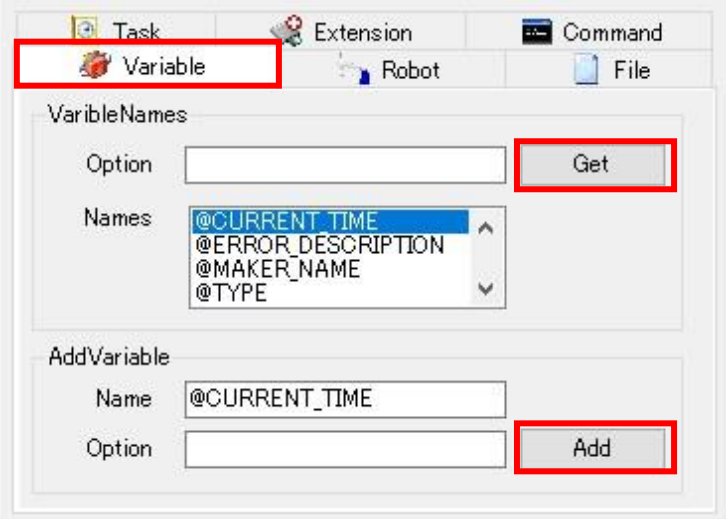


Fig. 4-12 System Variable Specification Screen

<User variable>

For User variable, you must manually enter the variable name yourself. As an example, when accessing X000 data memories, enter "X000" in Name on AddVariable and press [Add]. CCaoProvVariable's FinalInitialize() function is called as with the System variable. Upon successful completion, objects of Variable classes are generated and the screens are switched.

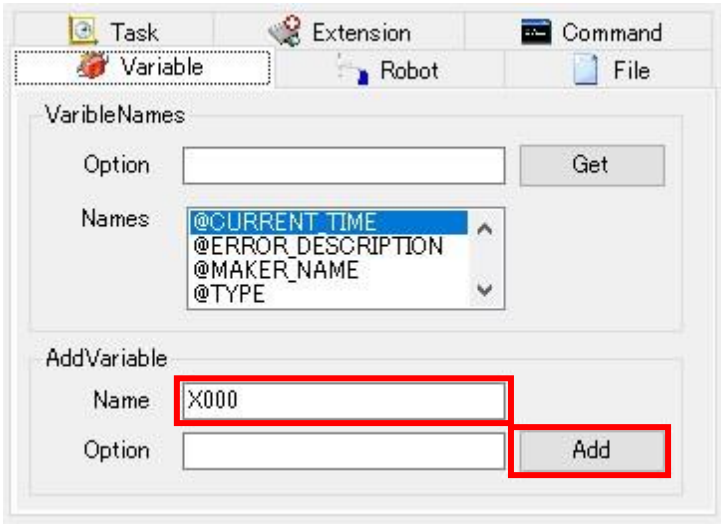


Fig. 4-13 User Variable Setting Screen

(8) GET/PUT of Variable classes

This section explains how to execute GET/PUT of the User variable "X000". System variable works similarly.

<GET >

Click [GET] in CaoVariable window to execute CaoVariable::get_Value. The providers call FinalGetValue() for CCaoProvVariable classes. Upon successful completion, Type and Value can be gotten as shown in the following screens.

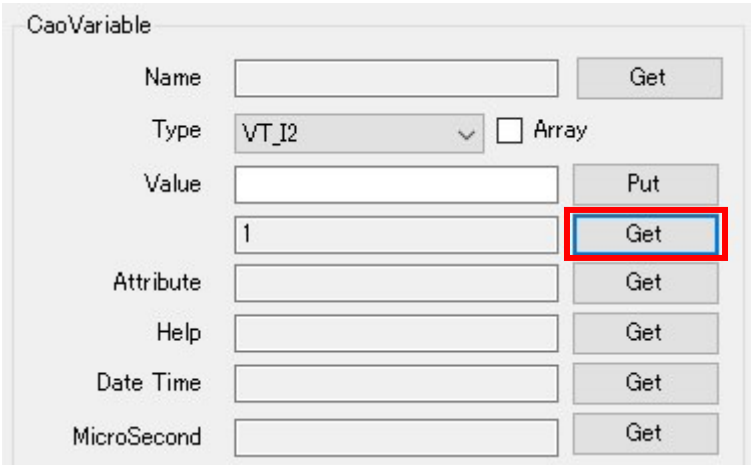


Fig. 4-14 Value Acquisition Window

<PUT >

Enter the data type and value as shown below and click [Put] to execute CaoVariable::get_Value.

The providers call FinalGetValue() for CCaoProvVariable classes. Upon successful completion, Type and Value can be gotten as shown in the following screens.

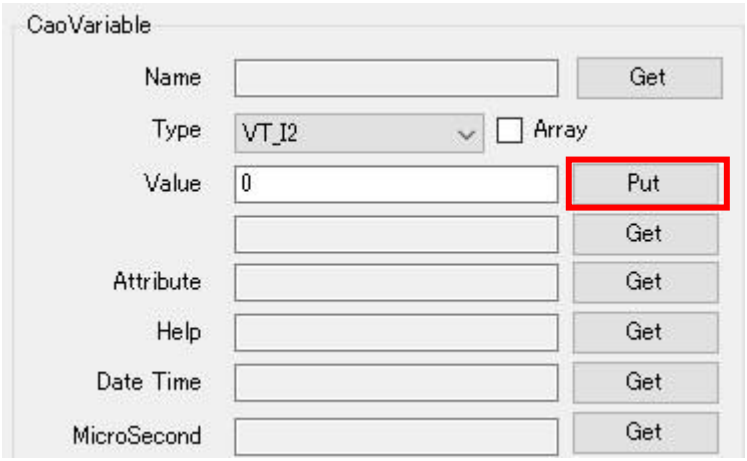


Fig. 4-15 Value Settings Window

(9) Return to Visual Studio and succeed if it is stopped at the breakpoint position.

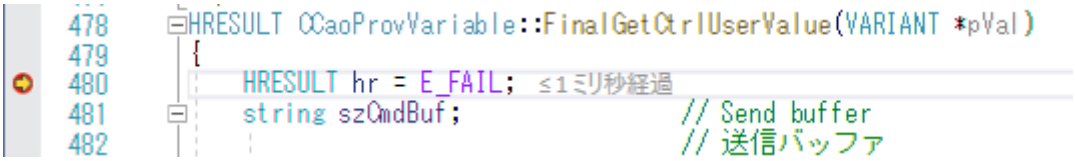


Fig. 4-16 4-16 Stopping at a breakpoint

4.4.2. Release of TCmini provider

Change the build target to Release MinDependency and build the final CaoProvTCmini.dll module. The difference between Release MinDependency and Release MinSize is as follows:

Release MinDependency...DLL will be larger, but it is not necessary to install the modules (ATL100. dll) on the PCs that use the DLL.

Release MinSize ... The DLL should be the smallest size, but the module (ATL100. dll) must be installed on the computer where the DLL will be used.

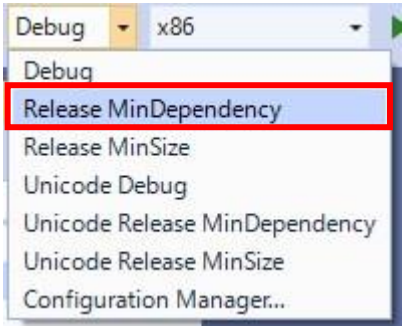


Fig. 4-17 Build target specification screen

4.4.2.1. Creating a document for release

If the specifications of the created provider are not publicly available, the user cannot easily use the provider. Therefore, it is necessary to create and publish the specifications of the provider.

The specifications for the provider must include the lowest-level description.

(1) AddController() Connection Parameter Specifications

(2) Specifications for User variable

(3) System Variable Specifications

(4) Other information considered important (information on provider-specific functions, precautions, etc.)

Microsoft Word templates files (<Installation folder>¥ORiN2¥CAO¥Provider¥Doc¥ Use xxx_ProvGuide_jp(en).dot. For specific examples, see the Providers User's Guide for TCmini (<Installation folder>ORiN2¥CAO¥ProviderLib¥ToshibaMachine¥TCmini¥Doc¥ TCmini_ProvGuide_en(jp).pdf)

4.4.2.2. Confirmation of provider dependence information

You can use Dependency Walker tools provided with VC++ to examine the dependencies of modules that you have created.

When you start Dependency Walker and specify TCmini.dll, the following window is displayed.

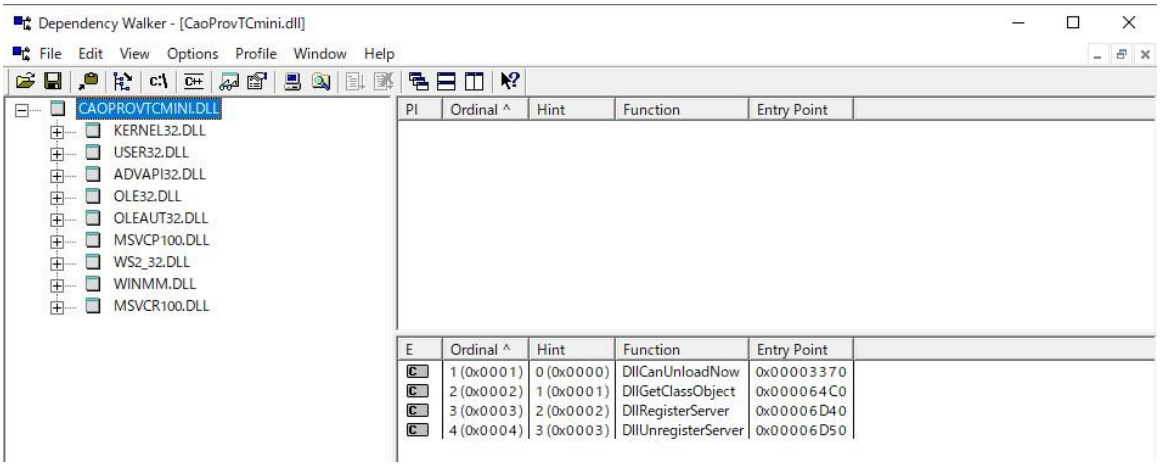


Fig. 4-18 Dependency Walker screen

From this window you can see that TCmini.dll only needs generic modules. Therefore, to make the CAO engines work with TCmini providers, you simply need to copy TCmini.dll file and register it in Regsvr32.[4]

[4] Some modules dynamically load other modules, so Dependency Walker does not detect this. To see this, you need to run the module you actually want to examine and examine the modules loaded into memory at that time.

5. Tips for Provider development

Here are some techniques that may be useful to know when implementing a provider.

5.1. Creating variable objects using parent objects

By storing the pointer of the parent object as a member when the variable object is created, the methods and properties of the parent object can be called from the variable object.

Adding this kind of functionality has the following advantages:

- It is possible to expose the methods and properties implemented in the parent object to clients that only have access to the variable classes, such as CaoSQL.

A specific implementation example is shown below. Here, Execute method "ProviderCancelSampleMethod" of the controller object is called when put_Value of the system variable of the controller class (e.g., @PROVIDER_CANCEL) is executed.

(1) Declaring Member Variables

Adds a pointer to the parent object to a member of the variable object.

List 5-1 CaoProvVariable.h

```
// ========== Additional functions are writed below. ==========
// ========== 各社追加関数はこれ以下に記述する. ==========
private:
  CCaoProvController* m_pCaoCtrl;
```

(2) Stores the parent object pointer

The parent object pointer is stored in the member variable declared in (1) when the variable class is initialized. Specifically, add the following code to FinalInitialize() method of CCaoProvVariable.

If the CAO engine destroys the parent object, the function of the variable class is not called, so there is no need to AddRef the reference counter.

List 5-2 CaoProvVariable.cpp – FinalInitialize()

```
HRESULT CCaoProvVariable::FinalInitialize(PVOID pObj)
{
  HRESULT hr = E_INVALIDARG;

  // MapTable initialization
  // MapTableの初期化
  InitMapTable();

  m_pCaoCtrl = NULL;
```

```
    CCaoProvExtension* pCaopExp = NULL;
    CCaoProvFile* pCaopFile = NULL;
    CCaoProvRobot* pCaopRobot = NULL;
    CCaoProvTask* pCaopTask = NULL;

    // Initialize data member
    // データメンバーの初期化
    m_lUSysId = 0;

     // Determine parent object
    // 親オブジェクトの判定
    switch (m_ulParentType) {
    case SYS_CLS_CONTROLLER:
            m_pCaoCtrl = (CCaoProvController*)pObj;
            :
}
```

(3) Invoking Methods of Parent Objects

CCaoProvVariable::FinalPutCtrlSys code that calls Execute method "ProviderCancelSampleMethod" of the controller object when put_Value of the system variable "@PROVIDER_CANCEL" is executed. Append to Value().

List 5-3 CaoProvVariable.cpp – FinalPutCtrlSysValue()

```
HRESULT CCaoProvVariable::FinalPutCtrlSysValue(VARIANT *pVal)
{
    HRESULT hr = E_FAIL;
    CComVariant vntParam;

    switch (m_lUSysId) {
    case CS_PROVIDER_CANCEL:        // “@PROVIDER_CANCEL”
        hr = m_pCaoCtrl->Execute(L"ProviderCancelSampleMethod", vntParam, pVal);
        break;
     }
     return hr;
}
```

6. Appendix

6.1. CAO provider functions 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 client when the system message

Class	Property, method, and event	Explanation	R/W	Argument of function		Called timing	Remarks
				IN	OUT RETVAL		
							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 SLIM.	W			When CaoRobot::GoHome() is called.	
	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.			

6.2. 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 string with specified type.	Option: BSTR Search parameter: BSTR Request type: VARTYPE	Value:VARIANT	n/a	･Format of option characetr string is “<parameter 1>[=<value1>] [,<parameter2>[=<value2>]]”. - 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 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)
	FinalGetVariableN	E	Acquisition of variable	[Option: BSTR]	Variable name list:	When	

	ames		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 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 SLIM.			When CaoProvRobot::GoHome() is called.	
	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 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 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 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 string with specified type.	Option: BSTR, Search parameter: BSTR, Request type:	Value:VARIANT	n/a	Same as CaoProvControllerImpl::GetOptionValue().

				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.			

6.3. Sample program

ORiN2SDK contains sample providers in the following folders:

<Installation folder>¥CAO¥ ProviderLib

Many of these providers are written in Visual Studio 6. 0 (VS6 below). To build with Visual Studio 2005 (VS2005 below), you need to make some modifications.

First, open VS6 project file (.dsw) in VS2005 and update it to VS2005 project file (.vcproj). Also, if you build with VS2005, the following problems may occur. Correct the source code by referring to the table below.[5]

Problem 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

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

	“_CRT_NON_CONFORMING_SWPRINTFS” in [Preprocesssor Definitions].
	[Measure 3] ・ Add an underbar “_” at the front of the function name that issues a warning. E.g. 1) wcsnicmp() →_wcsnicmp() E.g. 2) itoa() →_itoa()
“warning LNK4222" is issued.	・ Delete the ordinal of the export function in CaoProv.def. E.g.)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. E.g.) When the Name-space name is “MSXML2" IXMLDOMNodePtr pIChildNode; → MSXML2::IXMLDOMNodePtr pIChildNode;