# CAO プロバイダ作成ガイド

Version 1.2.1

September 14, 2021

【備考】

# 【改版履歴】

| 日付 | 版数 | 内容 |
|---|---|---|
| 2006-02-23 | 1.0.0.0 | 初版作成 |
| 2006-06-05 | 1.0.2.0 | Visual Studio2005 の対応情報追加 |
| 2006-10-18 | 1.0.3.0 | 通信クラスの利用方法, TCmini プロバイダ作成手順の修正 |
| 2006-12-12 | 1.0.4.0 | CaoProvControllerImpl に SetTimerInterval を追加 |
| 2007-07-03 | 1.0.5.0 | 誤記訂正 |
| 2007-11-23 | 1.0.6.0 | 通信クラスの利用方法, Tcmini プロバイダ作成手順の修正 |
| 2008-01-30 | 1.0.7.0 | エラーコード割り当て表の誤記訂正 |
| 2008-12-05 | 1.0.8.0 | インプロセスメッセージ転送機能追加 |
| 2009-08-24 | 1.0.9.0 | レジストリ情報の設定, CAO のインストール状況の確認の追加 |
| 2010-06-08 | 1.0.10.0 | インストール状況の確認修正 |
| 2011-12-22 | 1.0.11.0 | デバイスクラスのエラーコード追加 |
| 2013-09-02 | 1.0.12 | レジストリ情報に“RunAsLocal”オプション追加 |
| 2014-04-22 | 1.1.0 | プロバイダキャンセル/クリアの対応 |
| 2014-09-09 | 1.1.1 | フォルダ参照箇所の誤記訂正 |
| 2019-11-05 | 1.1.2 | プロバイダの登録方法, プロバイダの登録解除方法を追加 |
| 2020-01-29 | 1.2.0 | Visual Studio 2019 版に変更. 接続パラメータの解析, 通信クラスの利用方法修正. TCmini プロバイダ作成手順の修正 |
| 2021-09-14 | 1.2.1 | 東芝機械の社名を芝浦機械に変更 |

# 目次

1. はじめに ..... 7
1.1. プロバイダの開発言語 ..... 7
1.2. Visual C++ 2019 の設定 ..... 7
2. プロバイダの実装手順 ..... 9
2.1. 概要 ..... 9
2.2. 実装の手順 ..... 10
2.3. 新規プロバイダプロジェクトの作成 ..... 11
2.4. プロバイダの各クラス実装 ..... 16
2.4.1. 実装準備 ..... 16
2.4.1.1. クラスの実装準備 ..... 16
2.4.1.2. 各クラスの役割 ..... 19
2.4.2. CaoProvController クラス ..... 20
2.4.3. CaoProvVariable クラス ..... 20
2.4.4. 実行中の処理キャンセル ..... 23
2.4.4.1. ProviderCancel コマンドの実装 ..... 24
2.4.4.2. ProviderClear コマンドの実装 ..... 25
2.4.5. レジストリ情報の設定 ..... 25
2.5. プロバイダのデバッグとリリース ..... 27
2.5.1. プロバイダのデバッグ ..... 27
2.5.2. プロバイダのリリース ..... 30
2.5.2.1. ドキュメントの作成 ..... 30
2.6. プロバイダの配布 ..... 30
2.6.1. 依存情報の確認 ..... 30
2.6.2. プロバイダの登録方法 ..... 30
2.6.3. プロバイダの登録解除方法 ..... 31
2.6.4. ORiN2 SDK のインストール状況 ..... 31
3. プロバイダテンプレートライブラリが提供する便利な機能 ..... 32
3.1. 概要 ..... 32
3.2. オプション文字列の解析 ..... 32
3.3. 接続パラメータの解析 ..... 35
3.4. VARIANT 型の変換 ..... 37
3.5. エラー作成方法 ..... 39

3.6. メッセージイベントについて ..... 43
3.6.1. メッセージイベントによるログ出力 ..... 46
3.6.2. インプロセスメッセージ転送 ..... 47
3.6.3. メッセージイベントの一定周期発行 ..... 49
3.7. マクロプロバイダの作成方法 ..... 50
3.7.1. プロバイダオブジェクトの生成方法 ..... 50
3.7.2. OnMessage イベントの取得方法 ..... 52
3.7.2.1. EventSink の IDL ファイルへの追加 ..... 52
3.7.2.2. EventSink クラスの実装 ..... 53
3.7.2.3. EventSink の生成 ..... 54
3.8. 通信クラスの利用方法 ..... 55
3.8.1. はじめに ..... 55
3.8.2. CDevice クラス ..... 56
3.8.3. シリアル通信クラス ..... 58
3.8.3.1. 利用方法 ..... 58
3.8.3.2. エラーコード ..... 61
3.8.4. TCP ソケットクラス ..... 61
3.8.4.1. 利用方法 ..... 61
3.8.4.2. エラーコード ..... 65
3.8.5. CUDPSocket クラス ..... 65
3.8.5.1. 利用方法 ..... 65
3.8.5.2. エラーコード ..... 68

4. TCmini プロバイダの作成 ..... 69

4.1. TCmini コントローラとは ..... 69
4.1.1. 構成 ..... 69
4.1.1.1. データメモリの種類 ..... 69
4.1.1.2. データメモリの機能 ..... 70
4.1.1.3. リレーアドレス ..... 71
4.1.1.4. データレジスタアドレス ..... 71
4.1.1.5. リレー領域のバイト/ワードレジスタアドレス ..... 71
4.1.2. 接続 ..... 72
4.1.3. 通信プロトコル概要 ..... 73
4.1.3.1. I/O 読出し 1 点単位 ..... 74
4.1.3.2. データ読出し 1 語単位 ..... 75
4.1.3.3. I/O 強制セット ..... 76
4.1.3.4. I/O 強制リセット ..... 76

4.1.3.5. データ変更 1 語単位 ..... 77
4.2. TCmini プロバイダ仕様 ..... 77
4.2.1. AddController 時のオプション仕様 ..... 78
4.2.2. AddVariable の変数名とオプション仕様 ..... 79
4.2.2.1. システム変数仕様 ..... 80
4.2.2.2. ユーザ変数仕様 ..... 81
4.3. TCmini プロバイダの実装 ..... 82
4.3.1. TCmini プロバイダプロジェクトの作成 ..... 82
4.3.2. CSerial クラスの追加 ..... 82
4.3.3. CCaoProvController クラスの実装 ..... 83
4.3.3.1. 必要なメソッドのオーバーライド ..... 83
4.3.3.2. 必要なメンバの追加 ..... 84
4.3.3.3. FinalInitialize()の実装 ..... 84
4.3.3.4. ParseConnectionString()の実装 ..... 85
4.3.3.5. FinalConnect()の実装 ..... 87
4.3.3.6. FinalDisconnect()の実装 ..... 88
4.3.3.7. FinalTerminate()の実装 ..... 88
4.3.3.8. GetSerial()の実装 ..... 89
4.3.3.9. FinalGetVariableNames()の実装 ..... 89
4.3.4. CCaoProvVariable クラスの実装 ..... 91
4.3.4.1. 必要なメソッドのオーバーライド ..... 91
4.3.4.2. 必要な定数/メンバの追加 ..... 92
4.3.4.3. ParseUserVariableName()の実装 ..... 93
4.3.4.4. InitMapTable()の実装 ..... 95
4.3.4.5. FinalInitialize()の実装 ..... 96
4.3.4.6. FinalGetValue()の実装 ..... 97
4.3.4.7. FinalGetCtrlSysValue()の実装 ..... 98
4.3.4.8. FinalGetCtrlUserValue()の実装 ..... 102
4.3.4.9. FinalPutValue ()の実装 ..... 104
4.3.4.10. FinalPutCtrlUserValue()の実装 ..... 105
4.3.5. まとめ ..... 106
4.4. TCmini プロバイダのデバッグとリリース ..... 107
4.4.1. TCmini プロバイダのデバッグ ..... 107
4.4.2. TCmini プロバイダのリリース ..... 113
4.4.2.1. リリース用ドキュメントの作成 ..... 113
4.4.2.2. プロバイダ依存情報の確認 ..... 114

5. プロバイダ作成 Tips.....115
5.1. 親オブジェクトを使う変数オブジェクトの作成 .....115
6. 付録.....117
6.1. プロバイダ関数一覧 .....117
6.2. プロバイダテンプレート関数一覧 .....129
6.3. サンプルプログラム .....144

# 1. はじめに

本書は，CAO プロバイダ(以下，プロバイダ)の作成手順を具体的な例を示しながら解説する手順書です．

第 2 章では，プロバイダ作成のためにプロバイダに実装する必要のあるメソッドなどの基礎知識や，プロバイダの作成手順について解説をおこないます．

第 3 章では，プロバイダの持つ機能として，オプション文字列の解析方法や，メッセージイベントの発行方法などを解説します．さらに，ソケット通信およびシリアル通信をおこなうための通信クラスについても解説します．

第 4 章では，芝浦機械製小型プログラマブルコントローラ TCmini 用プロバイダを例に挙げ，具体的なプロバイダの実装手順を解説します．

第 5 章では，プロバイダを実装する上で知っておくと便利なテクニックを紹介します．

なお，本書では開発環境として Windows 10 OS を対象にしています．

## 1.1. プロバイダの開発言語

プロバイダの開発言語として Microsoft Visual C++6.0，2005，2008，2010，2012，2013，2015，2017，2019，Microsoft Visual C#を使用できます．本書は，Visual C++ 2019(以下，VC++)を使用した手順書になります．

## 1.2. Visual C++ 2019 の設定

Visual C++ 2019 でプロバイダを開発するためには，WindowsSDK10.0.17763.0 以上が必要になります．WindowsSDK は Visual Studio Installer を使用してインストールします．Visual Studio 2019 の詳細>変更を選択します．


図 1-1 Visual Studio Installer

ワークロードタブの「C++によるデスクトップ開発」にチェックを入れます．インストールの詳細に Windows 10 SDK(10.0.17763.0 以上)にチェックが入っていることを確認し，「閉じる」ボタンを押すとインストールが開始されます．下記の画面では Windows 10 SDK(10.0.18362.0)をインストールしていますが，そのときの最新バージョンをインストールしてください．

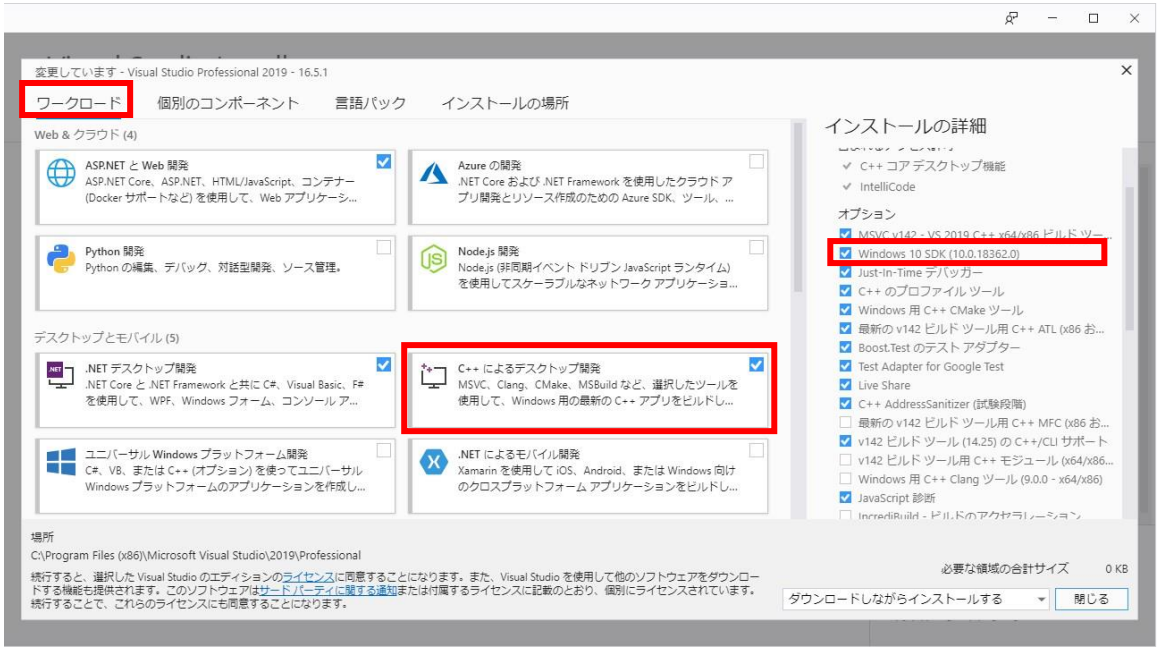



**図 1-2 Visual Studio Installer の変更画面**

# 2. プロバイダの実装手順

## 2.1. 概要

CAO(Controller Access Object)とは，ロボットを始めとする各種 FA 機器や，データベースなどさまざまなリソースに対して，共通のアクセス手段を与える API(Application Program Interface)です．CAO は分散オブジェクト技術(DCOM)をベースに開発されています．

CAO は，共通の機能を与えるエンジン部と，各メーカの違いを吸収するプロバイダ部から構成されています．CAO エンジンはクライアントアプリケーションに対して唯一のインタフェースを与え，プロバイダはエンジンに対して唯一のインタフェースを与えます．このような二層構造にすることで，プロバイダ開発者はエンジン部が提供してくれる汎用的な機能を実装する必要がなくなり，コントローラ内部の情報にアクセスするための本質的な部分だけを実装すればよいということになります．このことが，プロバイダの実装を容易にしている一つの大きな要因となっています．

さらに，プロバイダは「6.2 プロバイダテンプレート関数一覧」にあるように C++テンプレート(CAO Provider Template)と，メーカ実装部に分けられます．このテンプレートは CAO エンジンに対する共通のインタフェース実装とデフォルト実装を与えています．つまり，プロバイダを実装するユーザは，このテンプレートの関数群の中から提供したい機能に対応する関数をオーバーライドするだけで，プロバイダの実装することが可能となります．

本章では，プロバイダの実装手順を順に解説します．

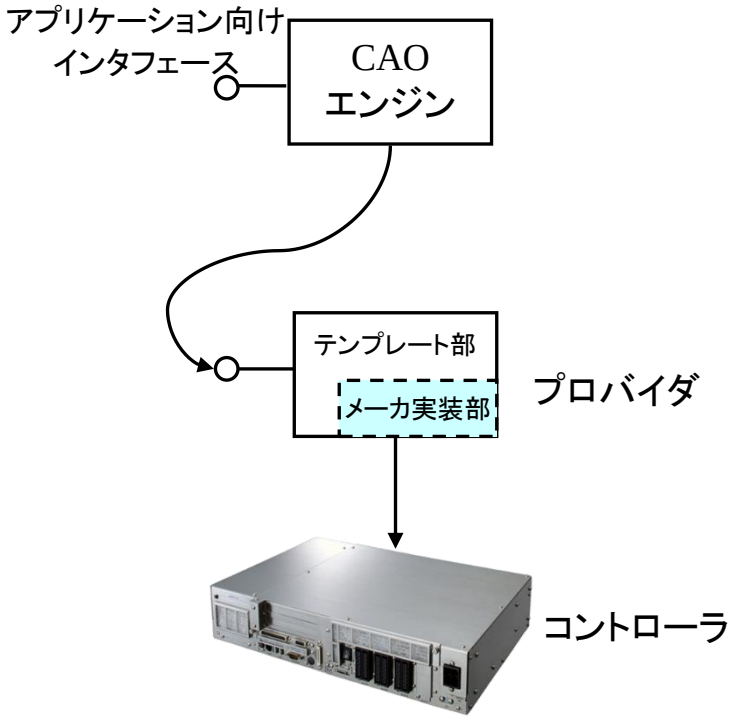



**図 2-1 CAO の構成**

## 2.2. 実装の手順

プロバイダを作成する際の大まかな流れを以下に示します.

(1) プロバイダの仕様を決定する

プロバイダを実装するためには, まず, プロバイダの仕様を決定する必要があります. アクセスするターゲットにあわせて, 以下の項目などを決定してください.

1. プロバイダの名称
2. AddController()の接続パラメータの仕様
3. 実装するクラスとメソッドの仕様

(2) プロバイダのテンプレートを作成する

プロバイダの仕様が決定したら, 次にプロバイダのテンプレートを生成します. これには, CaoProvWiz.exe を利用します.

(3) プロバイダの必要なメソッドを実装する

CaoProvWiz.exe で生成されたプロバイダのテンプレートを用いて, 必要なメソッドの実装をおこないます. また, 必要に応じてデバイス通信をおこなうためのクラスなども追加します.

(4) プロバイダのデバッグをおこなう

作成したプロバイダが意図したとおりに動作するかどうかの確認をおこないます. 正常に動作しなかった場合はデバッグ作業をおこなってください. すべてのデバッグ作業が終了すれば, プロバイダのリリースとなります.

以降では, これらの項目について詳細に解説をおこないます.

## 2.3. 新規プロバイダプロジェクトの作成

プロバイダを Visual C++で作成するためのプロジェクトは，ウィザードを用いて簡単に作成することができます．プロジェクトの作成方法を以下に説明します．

(1) スタートメニューの[ORiN2]→[CaoProvWizard]を実行する．

(2) プロバイダプロジェクトを作成する場所を選択します．

例) C:¥ORiN2¥CAO¥ProviderLib¥Sample¥Src に作成する場合


**図 2-2 CAO Provider の作成フォルダ選択画面**

(3) 作成するプロバイダの情報を入力します.

・ DLL Name ： DLL 名 (必須)
・ Vender Name ： ベンダ名 (必須)
・ Module Name ： モジュール名
・ Project Type ： 生成する VC++プロジェクトのバージョン
・ Use MFC ： MFC の使用の有無

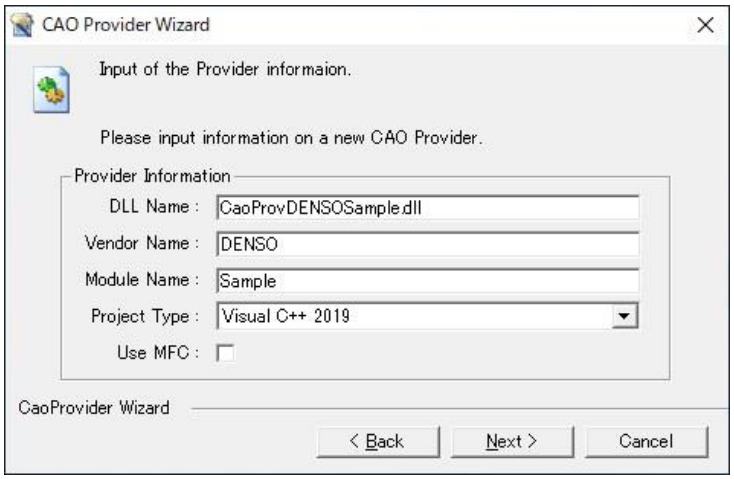



**図 2-3 プロバイダ情報入力画面**

ここでは入力した<Vender Name>,<Module Name>は COM の ProgID 名の決定に使用されます.
プロバイダの ProgID は“CaoProv.<Vender Name>.<Module Name>”となります.

(4) 確認画面が出ます. ここで“はい(Y)”ボタンを押下するとプロバイダのプロジェクトの作成が開始されます.

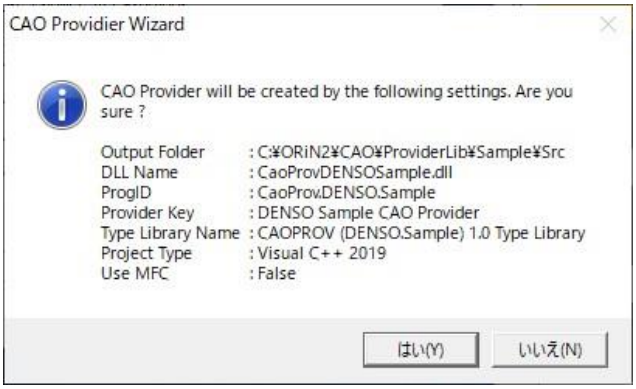



**図 2-4 作成プロバイダ情報確認画面**

(5) 以下の画面が表示されたら, プロバイダのプロジェクトは作成完了です.

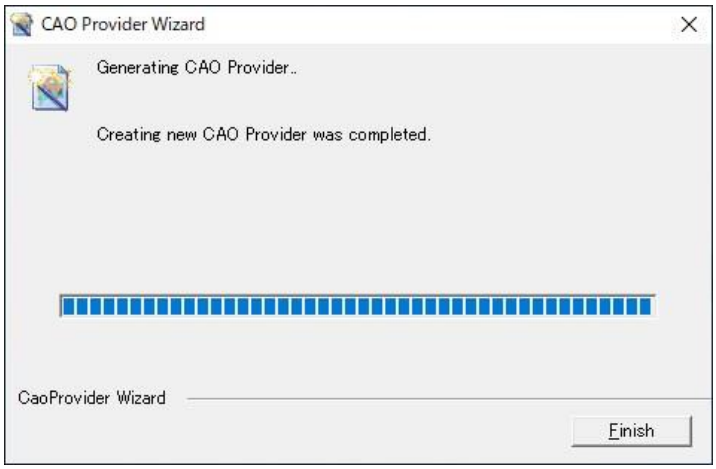



**図 2-5 プロバイダのプロジェクト作成完了画面**

プロバイダ用の VC++プロジェクトが指定したディレクトリに作成されます. 作成されたプロジェクトフォルダには CaoProv.vcxproj 以外にも複数のファイルが存在します. 下表にプロバイダに関連深い主なファイルの一覧を示します. この表に記述していないその他のファイルは Visual Studio によって管理されるため特に意識する必要はありません.

**表 2-1 プロジェクトの主なファイル一覧**

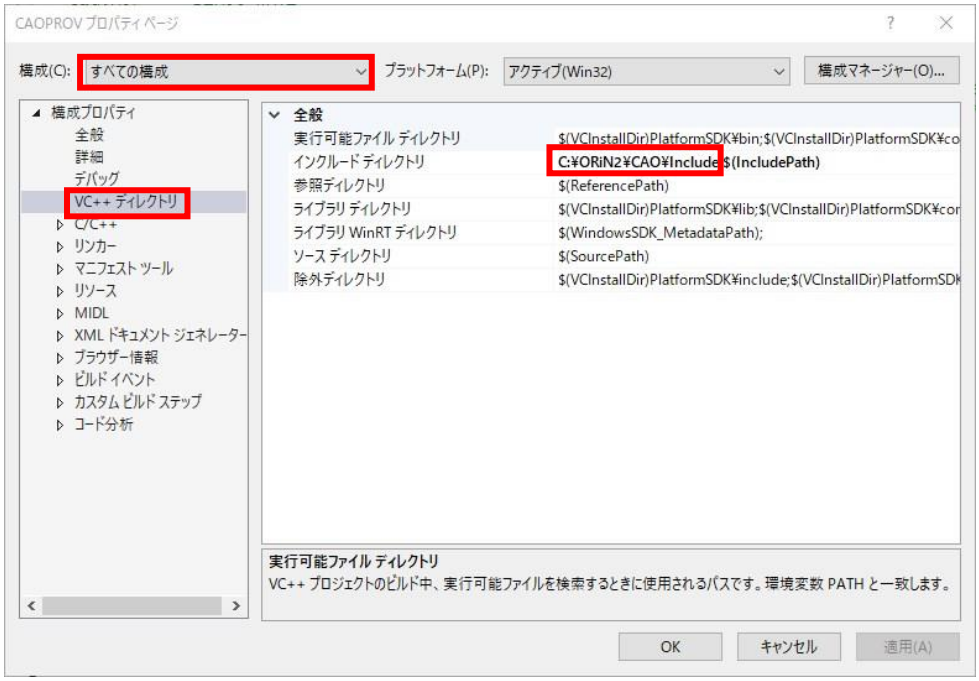
| No. | 名前 | 種類 |
|---|---|---|
| 1 | CaoProv.dsw | プロジェクトワークスペース(Visual C++ 6.0 のみ) |
| 2 | CaoProv.dsp | プロジェクトファイル(Visual C++ 6.0 のみ) |
| 3 | CaoProv.vcproj | プロジェクトファイル(Visual C++ 2005 , 2008) |
| 4 | CaoProv.vcxproj | プロジェクトファイル(Visual C++ 2010 ~ 2019) |
| 5 | CaoProv.IDL | IDL ファイル |
| 6 | Resource.h | リソースヘッダファイル |
| 7 | CaoProv.rc | リソースファイル |
| 8 | StdAfx.h | C ヘッダファイル |
| 9 | StdAfx.cpp | C++ソースファイル |
| 10 | CaoProv.h | C ヘッダファイル |
| 11 | CaoProv.cpp | C++ソースファイル |
| 12 | CaoProvController.h | C ヘッダファイル CCaoProvController クラスの定義 |
| 13 | CaoProvController.cpp | C++ソースファイル CCaoProvController クラスの実装 |
| 14 | CaoProvVariable.h | C ヘッダファイル CCaoProvVariable クラスの定義 |
| 15 | CaoProvVariable.cpp | C++ソースファイル CCaoProvVariable クラスの実装 |
| 16 | CaoProvTask.h | C ヘッダファイル CCaoProvTask クラスの定義 |
| 17 | CaoProvTask.cpp | C++ソースファイル CCaoProvTask クラスの実装 |
| 18 | CaoProvRobot.h | C ヘッダファイル CCaoProvRobot クラスの定義 |
| 19 | CaoProvRobot.cpp | C++ソースファイル CCaoProvRobot クラスの実装 |
| 20 | CaoProvFile.h | C ヘッダファイル CCaoProvFile クラスの定義 |
| 21 | CaoProvFile.cpp | C++ソースファイル CCaoProvFile クラスの実装 |
| 22 | CaoProvCommand.h | C ヘッダファイル CCaoProvCommand クラスの定義 |
| 23 | CaoProvCommand.cpp | C++ソースファイル CCaoProvCommand クラスの実装 |
| 24 | CaoProvExtension.h | C ヘッダファイル CCaoProvExtension クラスの定義 |
| 25 | CaoProvExtension.cpp | C++ソースファイル CCaoProvExtension クラスの実装 |
| 26 | CaoProvMessage.h | C ヘッダファイル CCaoProvMessage クラスの定義 |
| 27 | CaoProvMessage.cpp | C++ソースファイル CCaoProvMessage クラスの実装 |

(6) プロジェクトプロジェクトが完成したら, CaoProv.vcxproj を Visual Studio 2019（管理者権限）で開きます. 管理者権限でない場合, ビルドが通らないので注意してください.

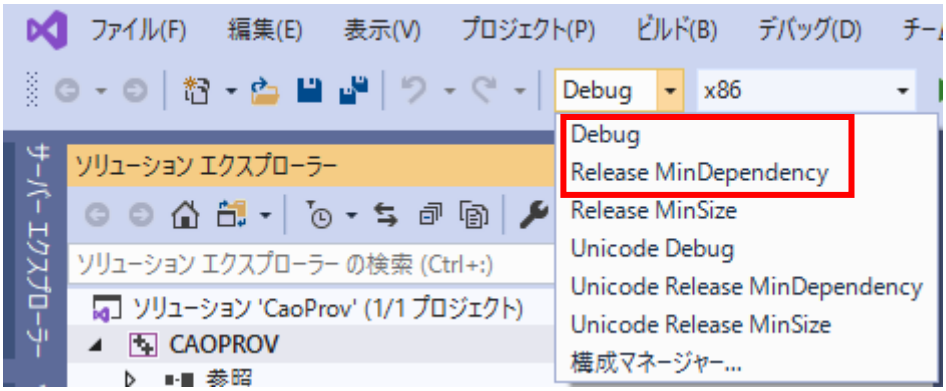
(7) インクルードディレクトリを設定します.

ソリューションエクスプローラーから CAOPROV を選択し, プロパティを開きます.


プロパティページを開いたら, [構成]ですべての構成を選択し, [構成プロパティ]→[VC++ディレクトリ]→[インクルードディレクトリ]に<インストールフォルダ>¥ORiN2¥CAO¥Include を追加してください.

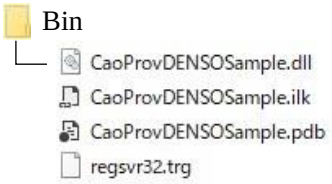


(8) 以上の操作でプロバイダを(何の機能も持ちませんが)ビルドすることができます.
アクティブソリューション構成で“Debug”(デバッグ版)か, “Release MinDependency”(リリース版)を選択し, リビルドを実行してください.




(9) ビルド成果物を確認します.
ビルドが成功すると, Bin フォルダが作成されます. Bin フォルダの中身は, “Debug”(デバッグ版)と“Release MinDependency”(リリース版)で生成されるファイルが異なります. Bin フォルダの中にある”CaoProv <Vender Name><Module Name>.dll”がプロバイダ本体になります.

<Debug の場合>




<Release MinDependency の場合>


## 2.4. プロバイダの各クラス実装

### 2.4.1. 実装準備

#### 2.4.1.1. クラスの実装準備

各クラスの定義と実装は以下のファイルにあります.

**表 2-2 クラスの定義と実装があるファイル**

| クラス名 | 定義 | 実装 |
|---|---|---|
| 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 |

プロバイダでクラスの実装をおこなうには，そのクラスの中のメソッドをオーバーライドします．メソッドのオーバーライドの準備は以下の手順でおこないます．

(1) 各クラスのヘッダファイルに記述してあるメソッドの内，使用するメソッドのコメントをはずします．
以下に，CCaoProvController クラスのメソッド FinalInitialize を使いたいときの例を示します．
(CaoProvController.h)

```
//HRESULT FinalInitialize();        //コメントをはずす前
            ↓
  HRESULT FinalInitialize();        //コメントをはずした後
```

(2) メソッドの実装がおこなわれている箇所のコメントをはずします．
以下に，CCaoProvController クラスのメソッド FinalInitialize を実装するときの例を示します．
(CaoProvController.cpp)

```
//コメントをはずす前
/*  ←  このコメント開始記号を削除
HRESULT CCaoProvVariable::FinalInitialize()
{
        : （記述内容は省略）
}
*/  ←  このコメント終了記号を削除
                ↓
//コメントをはずした後
HRESULT CCaoProvVariable::FinalInitialize()
{
   :（記述内容は省略）
}
```

(3) (2)のメソッド内の返り値を“S_OK”とします．(返り値がない場合，この作業は飛ばしてください)

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

これでメソッドのオーバーライドの準備は完了です．

この後，各メソッドに各メーカの処理を実装していきます．その際に必要であれば新しくクラスに変数とメソッドを追加することもできます．追加する変数とメソッドの定義は追加されるクラス定義の最後に追加してください．追加メソッドの実装は追加されるクラスのメソッドが実装されている cpp ファイルの最後に追加してくださ

い.

基本的なメソッドのみ記述例を後述します. また, プロバイダテンプレートで公開されているメンバ変数やオーバーライド可能な関数の一覧は,「6.2 プロバイダテンプレート関数一覧」を参照してください.

#### 2.4.1.2. 各クラスの役割

各クラスではそれぞれ提供する機能の種類が決まっています．以下に各クラスの機能概要を示します．

**表 2-3 プロバイダのクラス機能概要**

| クラス名 | 説明 |
|---|---|
| CaoProvController | コントローラのリソース全般に関わる機能を提供します． |
| CaoProvVariable | 変数リソースに関わる機能を提供します． |
| CaoProvRobot | ロボットリソースに関わる機能を提供します． |
| CaoProvFile | ファイル，フォルダリソースに関わる機能を提供します． |
| CaoProvTask | タスクリソースに関わる機能を提供します． |
| CaoProvCommand | コマンドリソースに関わる機能を提供します． |
| CaoProvExtension | 拡張ボードリソースに関わる機能を提供します． |
| CaoProvMessage | メッセージリソースに関わる機能を提供します． |

これらのクラスを実装する際に必ずオーバーライドしなければならない関数がいくつかあります．それを表 2-4 に示します．

**表 2-4 各クラス実装での必須メソッド**

| クラス名 | 必須メソッド | 説明 |
|---|---|---|
| CaoProvController | FinalInitialize() | 初期化処理 |
| | FinalTerminate() | 最終処理 |
| | FinalConnect() | プロバイダ接続処理 |
| | FinalDisconnect() | プロバイダ切断処理 |
| CaoProvVariable | FinalInitialize() | 初期化処理 |
| CaoProvRobot | FinalInitialize() | 初期化処理 |
| CaoProvFile | FinalInitialize() | 初期化処理 |
| CaoTask | FinalInitialize() | 初期化処理 |
| CaoCommand | FinalInitialize() | 初期化処理 |
| CaoExtension | FinalInitialize() | 初期化処理 |
| CaoMessage | FinalInitialize() | 初期化処理 |

この中でも特に重要なクラスは CaoProvController であり，必ず実装する必要があります．その他のクラスに関しては，すべてのクラスを実装する必要はなく，ベンダごとの裁量に任されます．つまり，ユーザに提供したい機能に対応するクラスだけを実装すればよいことになります．

以下では，CaoController クラスと CaoVariable クラスの中から，CCaoProvController::FinalInitialize()，CCaoProvController::FinalConnect()，CCaoProvController::FinalDisconnect()，CCaoProvController::FinalTerminate() ， CCaoProvVariable::FinalInitialize ， CCaoProvVariable::FinalPutValue() ， CCaoProvVariable::FinalGetValue()について説明をおこないます．

### 2.4.2. CaoProvController クラス

**HRESULT CCaoProvController::FinalInitialize()**

CaoWorkspace::AddController 処理が呼ばれたタイミングで呼び出され，オブジェクトの初期化をおこないます．CaoProvController クラスで利用する変数の初期化などをおこなってください．

**HRESULT CCaoProvController::FinalConnect()**

CaoWorkspace::AddController 処理が呼ばれたタイミングで呼び出され，FinalInitialize()の次に実行されます．このメソッドでは作成するプロバイダが対応する通信機器との接続処理を実装します．接続方法は各通信機器に依存しますが，例えば通信機器へアクセスする DLL などが用意されている場合は，DLL をロードする処理などが考えられます．RS-232C 通信や TCP/IP 通信をおこなっている場合は，このメソッド内にコネクション処理を実装します．

**HRESULT CCaoProvController::FinalDisconnect()**

プロバイダの切断処理をおこないます．

CaoControllers::Remove 処理が呼ばれたタイミングで呼び出されます．このメソッドでは FinalConnect()で接続した通信機器との切断処理をおこないます．切断方法は各通信機器に依存します．

**HRESULT CCaoProvController::FinalTerminate()**

オブジェクトの開放前処理をおこないます．

CaoControllers::Remove 処理が呼ばれたタイミングで呼び出され，FinalDisconnect()の次に実行されます．このメソッドでは初期化で生成したイベントなどの変数をクローズするなどの処理を行ってください．

### 2.4.3. CaoProvVariable クラス

**HRESULT CCaoProvVariable::FinalInitialize()**

このメソッドは，CaoVariable オブジェクト生成時に呼ばれ，オブジェクトの初期化をおこないます．

この関数の引数である pObj は親オブジェクトへのポインタです．CaoVariable は CaoController や CaoRobot など複数のオブジェクトから生成されるので，必要な場合それぞれ親オブジェクトに応じて FinalInitialize()で初期化処理を分けることができます．親オブジェクトはメンバ変数 m_ulParentType で判別できます．m_ulParentType の種類を以下に示します．

**表 2-5 CaoProvVariable の親オブジェクト**

| m_ulParentType | 親オブジェクト |
|---|---|
| SYS_CLS_CONTROLLER | CaoProvController |
| SYS_CLS_ROBOT | CaoProvRobot |
| SYS_CLS_FILE | CaoProvFile |
| SYS_CLS_TASK | CaoTask |
| SYS_CLS_EXTENSION | CaoExtension |

また, この関数では変数名を解析して異なる識別子を割り当ててください. 変数名にはシステム変数とユーザ変数の 2 種類が存在します. システム変数は固定文字列(“@MAKER_NAME”など), ユーザ変数は任意文字列となります. システム変数には文字列の先頭に「@」を使用するのが慣例です.

コントローラクラスのシステム変数は以下の手順で追加できます. この例では”@TEMP_DATA”を追加しています. 以下の手順を記述することで, システム変数の識別子を保持するメンバ変数 m_lUSysId に異なる識別子が割り当てられます.

＜システム変数の追加方法＞

(1) StdAfx.h に必要なマクロ定義を行う.

```
#define CS_MAKER_NAME     0x0003              ←  ユニークな番号を割り当てる
#define CS_MAKER_NAME$    L"@TEMP_DATA"       ←  ユニークな名前を割り当てる
```

(2) 名前管理マップの初期化にマクロを追加する

```
HRESULT  CCaoProvVariable::InitMapTable()
{
  :
  // 変数名マップの初期化
  const var_map_entry var_cs_map[] = {

      // :
      MAP_ENTRY( CS_TEMP_DATA ),  ←  マクロ追加
  };
  :
  return S_OK;
}
```

(3) 識別子の割り当てを行う(テンプレートからの変更なし)

```
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;
}
```

ユーザ変数の識別子の割り当て方は，”4.3.4.5FinalInitialize()の実装”を参照してください．

**HRESULT CCaoProvVariable::FinalGetValue(VARIANT *pVal)**

変数の値を取得します．引数は VARIANT のポインタ pVal です．この pVal に取得結果の値を入れることでクライアントに値を渡すことができます．テンプレートでは，親オブジェクトがコントローラクラスの場合の実装のみ記述しています．システム変数の場合 FinalGetCtrlSysValue 関数，ユーザ変数の場合 FinalGetUserValue 関数を実行します．実装する場合，FinalGetCtrlSysValue または FinalGetUserValue に記述してください．

```
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;
}
```

変数は必要に応じて通信機器から取得します．また，VARIANT への代入方法は「ORiN2 プログラミングガイド」を参照してください．

以下にコントローラクラスのシステム変数である@MAKER_NAME の実装例を示します．(デフォルトのテンプレートでは既に実装されています．) この例ではコントローラにアクセスして VT_BSTR 型のシステム変数の型を取得し，値を pVal に代入しています．実際にはコントローラへのアクセスメソッドに合わせて実装をおこなってください．

```
HRESULT CCaoProvVariable::FinalGetCtrlSysValue(VARIANT *pVal)
{
  switch (m_lUSysId) {
```

```
        :
    case CS_TEMP_DATA:
        pVal->vt = VT_BSTR;
        pVal->bstrVal = SysAllocString(L"DENSO"); ← メーカー名追加
        break;
    }
    :
    return hr;
}
```

**HRESULT CCaoProvVariable::FinalPutValue(VARIANT newVal)**

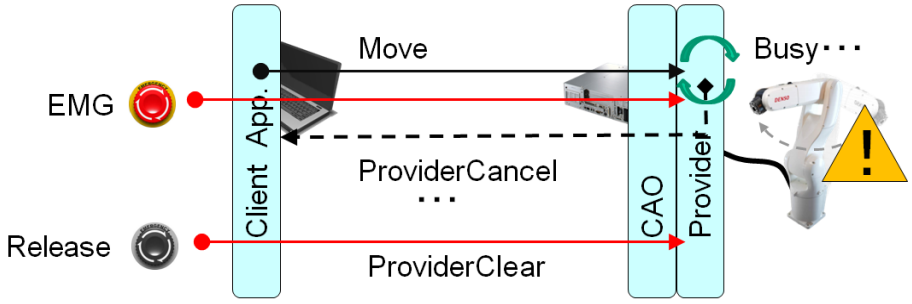
FinalPutValue 関数の引数 newVal は VARIANT 型です．クライアントが入力した値が入っています．テンプレートでは，親オブジェクトがコントローラクラスの場合の実装のみ記述しています．システム変数の場合 FinalPutCtrlSysValue 関数，ユーザ変数の場合 FinalPutUserValue 関数を実行します．実装する場合，FinalPutCtrlSysValue または FinalPutUserValue に記述してください．

ここで VARIANT の型が VT_BSTR，VT_ARRAY，VT_BYREF といった実体を持たない型の場合は，この FinalPutValue メソッドのスコープがはずれると値の参照ができなくなるので，スコープ外でも値を保持したい場合には別途メモリの確保処理をおこなってください．

```
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. 実行中の処理キャンセル

緊急停止が要求されるような場面では，プロバイダは実行中の処理を直ちにキャンセルし，上位クライアント CAO アプリケーションに処理を戻せるようにする必要があります．そうすることで上位アプリケーションは停止に必要な処理を行うことができます．この要求に対応するにはプロバイダで規定している二つの要求コマンド(ProviderCancel および ProviderClear)に応答して，実行中の処理をキャンセルする必要があります．




これらの要求コマンドは，CCaoProvController::Execute()のシステム予約コマンドとして次のように規定されています．

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

ProverCancel コマンドが発行されたら，ExecProviderCancel()メソッド，ProviderClear に対して ExecProviderClear()メソッドがそれぞれ CAO.exe から非同期で呼ばれます．
クライアントアプリケーションはプロバイダが実行中の処理を直ちにキャンセルする必要がある場合は ProviderCancel コマンドを非同期で CAO.exe に要求します．キャンセル処理が完了し，再び通常のコマンドを実行するには ProviderClear コマンドを発行し，キャンセル要求をクリア解除します．

#### 2.4.4.1. ProviderCancel コマンドの実装

ProviderWizard で作成されたプロジェクトでは ProviderCancel コマンドに対応するメソッドである CCaoProvController::ExecProviderCancel()は次の通りに実装されています．

**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;
}
```

ここで m_hProviderCancelEvent は CCaoProvController クラスのメンバー変数で HANDLE 型として定義さ

れ, SetEvent でシグナル状態になるように実装されていることがわかります. 下記例のように非同期で実行中のループの中でこの HANDLE を監視し, シグナル状態ではループを抜けるような実装を行うことでキャンセルの処理を実現できます. このタイミングでクリアすべき処理があれば続いて実装を追加します.

```
for(int i=0; i<1000; i++)  {
  // 何かの処理  ・・・
  //  :
  if(::WaitForSingleObject(m_hProviderCancelEvent, 0) == WAIT_OBJECT_0) {
       //  処理キャンセル
       break;
  }
}
```

#### 2.4.4.2. ProviderClear コマンドの実装

ProviderWizard で作成されたプロジェクトでは ProviderClear コマンドに対応するメソッドである CCaoProvController::ExecProviderClear()は次の通りに実装されています.

**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;
}
```

ここではシグナル状態になっている m_hProviderCancelEvent を ResetEvent でリセットし通常状態に戻すことでキャンセル処理を再び実行しないようにします. このタイミングでクリアすべき処理があれば続いて実装を追加します.

### 2.4.5. レジストリ情報の設定

レジストリに登録される情報の初期値を CaoProvController.rgs ファイルで設定します.

**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 ''
            }
        }
    }
```

上記のファイル内のグレーで示した箇所のみを編集してください．それ以外の箇所を編集した場合，プロバイダが正しく登録できない場合があります．

| 名前 | 値 |
|---|---|
| Enabled | 使用設定<br>0 : 使用不可<br>4294967295(0xffffffff) : 使用可 |
| RunAsLocal | CaoWorkspace::AddController 時に起動マシン名を省略した場合のプロバイダのデフォルト起動方法を指定します．<br>0 : インプロセス起動<br>4294967295(0xffffffff) : アウトプロセス起動 |
| Writable | 書き込みフラグ |

| | |
|---|---|
| | 0 : 読取り専用<br>4294967295(0xffffffff) : 読み書き可能 |
| LocaleID | ロケール |
| License | ライセンス設定 |
| ORiNlm | 予約(常に空文字列) |
| Parameter | パラメータ設定(ある程度固定化できるパラメータを設定します.) |
| CRDFile | CRD ファイルの設定.<br>各クラスのプロパティで「E_CRDIMPL」を返した場合, ここで指定した CRD ファイルの内容をクライアントに返します. |
| BindCmds | ダイナミックバインディング(Command クラス)<br>0 : ダイナミックバインディング不可<br>4294967295(0xffffffff) : ダイナミックバインディング可 |
| BindExec | ダイナミックバインディング(Execute メソッド)<br>0 : ダイナミックバインディング不可<br>4294967295(0xffffffff) : ダイナミックバインディング可 |
| GroupID | 予約(常に1) |

上記の設定内容は, CaoConfig を使用して変更することができます.

## 2.5. プロバイダのデバッグとリリース

### 2.5.1. プロバイダのデバッグ

CAO では, プロバイダの DLL モジュールは CAO エンジンである CAO.exe から必要に応じて呼ばれる構造になっています. したがって, プロバイダ DLL モジュールのデバッグを行う場合には VC++のデバッグセクションの実行可能なファイルに CAO.exe を指定する必要があります[1].

その他の手順は通常の VC++プロジェクトのデバッグと同じです. クライアントアプリケーションとしては ORiN の CAO テストツールである ORiN2¥CAO¥Tools¥CaoTester2¥Bin¥CaoTester2.exe 等を目的に合わせて使用することができます.

下記にデバッグ作業の一例を示します.

(1) ビルドターゲットを[Debug_x86]にしてください.

[1] プロバイダ DLL をアウトプロセスで起動したい場合は「デバッグセッションの実行可能ファイル」に dllhost.exe を指定します. この場合, CAO の AddController メソッドの第 2 引数に「マシン名」を指定する必要があります.


**図 2-6 ビルドターゲット指定画面**

(2) デバッグセクションの実行可能なファイルに CAO.exe を指定します.

CAOPROV プロパティページを開き, [構成]で Debug, [プラットフォーム]で Win32 を指定してから, [構成プロパティ]→[デバッグ]→[コマンド]に CAO.exe をフルパスで指定します. CAO.exe は通常 ORiN2¥CAO¥Engine¥Bin フォルダにあります.

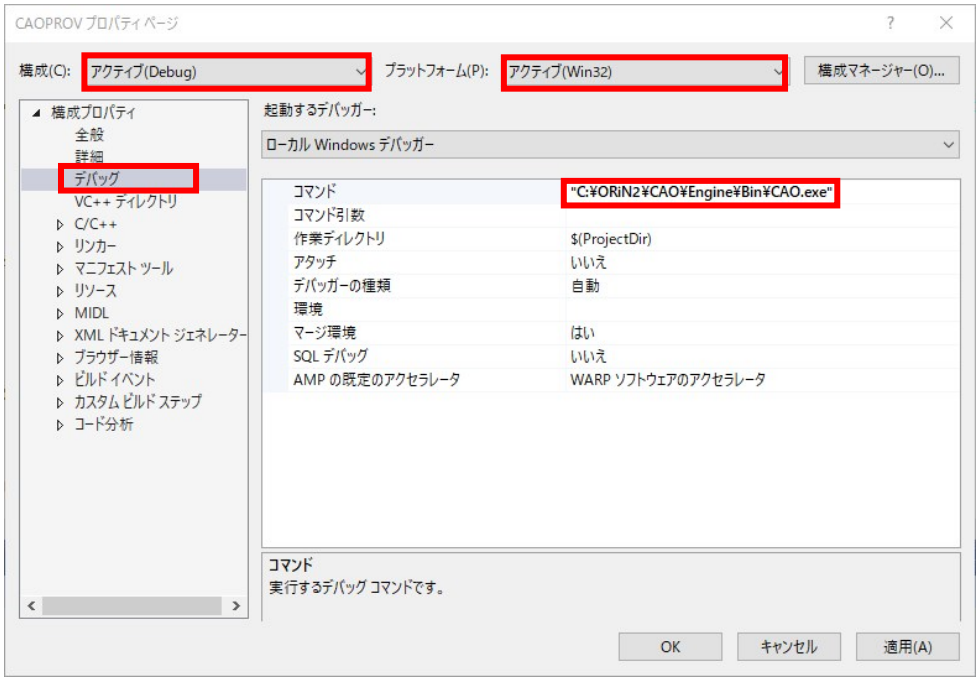



**図 2-7 CAOPROV プロパティ**

(3) 必要な位置にブレークポイントをセットしてください.

(4) デバッグの開始を実行します.

[デバッグ(D)]メニューから[デバッグの開始(S)] を実行してください.


**図 2-8 デバッグの開始画面**

(5) CaoTester2.exe を起動し，デバッグ目的のプロバイダを指定してください．

(6) [Add]ボタンを押下してコントローラと接続します．

(7) CaoTester2 で CAO のサービスを実行してプロバイダを呼び出します．

例えば，CaoVariable オブジェクトのデバッグをおこなう場合は，Variable タブを選択して AddVariable の Name に任意の名前を入力し[Add]ボタンを押下します．後は，Value の Put や Get 等を実行してください．

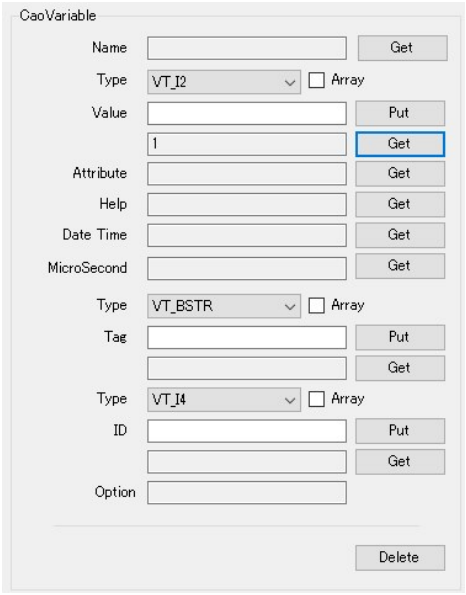



**図 2-9 システム変数の指定画面**

(8) VC++に戻り，ブレークポインタ位置で停止していれば VC++を使ってデバッグをおこないます．

### 2.5.2. プロバイダのリリース

#### 2.5.2.1. ドキュメントの作成

プロバイダの仕様が外部公開されていなければユーザからは簡単にプロバイダを使用することができません. そこでプロバイダの仕様書を作成して公開する必要があります.

プロバイダ用仕様書としては最低下記の内容がきちんと記載されている必要があります.

(1) AddController()の接続パラメータの仕様
(2) ユーザ変数の仕様
(3) システム変数の一覧とその意味
(4) その他, 重要と思われる情報(プロバイダ特有機能に関する情報, 注意事項等)

仕様書の書き方(書式)に関する規定はないため, 自由に作成してください.

## 2.6. プロバイダの配布

### 2.6.1. 依存情報の確認

CAO エンジンの本体 CAO.exe は特別な依存モジュール(DLL 等)を必要としません. 環境に関係なく安定動作するように考慮しているためです. プロバイダも同様に極力他のモジュールへの依存を無くすように作成する方が望ましいといえます. 他のモジュールへの依存を無くすには特殊なライブラリを使用しないようにする. あるいはライブラリを静的にリンクするなどの対策が必要となります.

作成したプロバイダが他のモジュールに依存している場合, どのモジュールが動作に必要なのか確認するようにしてください. 依存情報を調べるには, Dependency Walker や Process Explorer などを利用します.

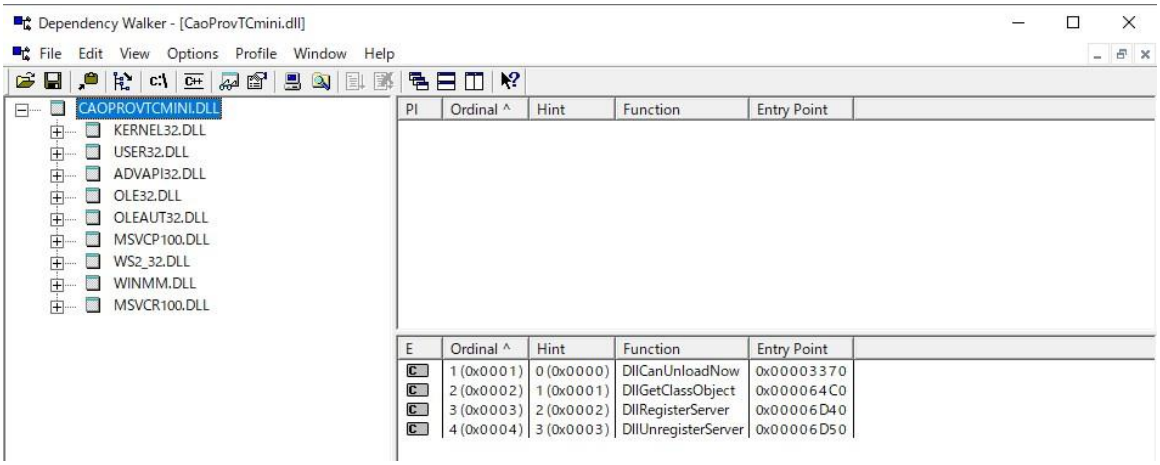



**図 2-10 TCmini.DLL の Dependency Walker 画面**

### 2.6.2. プロバイダの登録方法

プロバイダを配布した先で使用する場合, プロバイダの DLL をレジストリ内に登録する必要があります. レ

ジストリ内に登録するためには，コマンドプロンプトで regsvr32 コマンドを使用します． regvr32 コマンドのあとにプロバイダ DLL 名を入力し，実行することでレジストリ内に登録できます．なお，コマンドプロンプトを立ち上げる際，管理者モードで実行してください．

<登録例> C:¥ORiN2¥CAO¥ProviderLib¥Sample¥Bin¥CaoProvDENSOSample.dll を登録する場合

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

### 2.6.3. プロバイダの登録解除方法

regsvr32 コマンドで登録したプロバイダ DLL を登録解除する場合，regsvr32 /u コマンドを使用します．regsvr32 コマンドのあとにプロバイダ DLL 名を入力し，実行することでレジストリから登録を解除できます．

<登録解除例> C:¥ORiN2¥CAO¥ProviderLib¥Sample¥CaoProvDENSOSample.dll を登録解除する場合

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

### 2.6.4. ORiN2 SDK のインストール状況

ORiN2 SDK のインストール状況の確認方法については，「ORiN2 SDK ユーザーズガイド 3.7.ORiN2 SDK インストール状況の確認」を参照してください．

# 3. プロバイダテンプレートライブラリが提供する便利な機能

## 3.1. 概要

ORiN2 SDK では，オプション文字列の解析をおこなうメソッドや，メッセージイベントを発行するメソッドなどプロバイダに数多くの便利な機能が用意されています．また，デバイス通信をおこなうためのクラスが，いくつかのプロバイダで利用されています．

本章では，これらの機能を実装する方法について解説をおこないます．

## 3.2. オプション文字列の解析

プロバイダでは，メソッドの引数として使用するオプション文字列を GetOptionValue メソッドで取得することが出来ます．GetOptionValue()の引数仕様を以下に示します．実装の詳細に関しては<インストールフォルダ>¥ORiN2¥Cao¥Include の OptionValue.h を参照ください．

```
GetOptionValue
(
"<オプション文字列>",
"<検索文字列>",
"<要求型>",
"<結果値>"
)
```

オプション文字列は以下の書式に従います．

<オプション名 1>[=<オプション値 1>],<オプション名 2>[=<オプション値 2>]・・・

以下に，簡単な例を示します．ここでは，オプション文字列を“Opt1=test,Opt2=sample”とし，検索オプション名を“Opt1”としています．その結果，vntOptVal.bstrVal は“test”という文字列を抽出することができます．また，検索オプションを”Opt2”とすると，”sample”という文字列を抽出することができます．以下のように，GetOptionValue を用いて連続して BSTR 型の文字列を抽出する場合は，メモリリークを防ぐために VariantClear 関数を用いて開放処理を行う必要があります．

```
HRESULT hr;
VARIANT vntOptVal;
hr = GetOptionValue(CComBSTR(L"Opt1=test,Opt2=sample"), CComBSTR(L"Opt1"), VT_BSTR,
                                                                    &vntOptVal);
// vntOptVal.bstrVal == “test”
VariantClear(&vntOptVal);       // BSTR文字列の開放処理

hr = GetOptionValue(CComBSTR(L"Opt1=test,Opt2=sample"), CComBSTR(L"Opt2"), VT_BSTR,
                                                                    &vntOptVal);
// vntOptVal.bstrVal == “sample”
VariantClear(&vntOptVal);       // BSTR文字列の開放処理
```

また，オプション文字列には囲み文字を使用することができます．囲み文字として使用できるものを以下に

示します.

- 括弧(“( )”)
- 中括弧(“{ }”)
- 大括弧(“[ ]”)
- 角括弧(“< >”)

また, これらの囲み文字のうち最初に現れたものを囲み文字とし, それ以降に出てきた括弧は通常の記号として扱われます.

以下に, オプションが複数あるときの例を示します.

オプション文字列： Test1=Sample1,Sample2,Test2=Sample3

**表 3-1 オプション文字列 例1結果**

| オプション名 | オプション値 |
|---|---|
| Test1 | Sample1 |
| Test2 | Sample3 |

以下に, オプション値が括弧の中にあるときの例を示します.

オプション文字列： Test1=(Sample1,Sample2),Test2=(Sample3)

**表 3-2 オプション文字列 例 2 結果**

| オプション名 | オプション値 |
|---|---|
| Test1 | Sample1,Sample2 |
| Test2 | Sample3 |

以下に, オプション値が異なる括弧の中にあるときの例を示します.

オプション文字列： Test1=(Sample1),Test2=<Sample2>

**表 3-3 オプション文字列 例 3 結果**

| オプション名 | オプション値 |
|---|---|
| Test1 | Sample1 |
| Test2 | <Sample2> |

以下に, オプション値の括弧内に別の括弧があるときの例を示します.

オプション文字列： Test1=((Sample1)),Test2=(<Sample2>)

**表 3-4 オプション文字列 例 4 結果**

| オプション名 | オプション値 |
|---|---|
| Test1 | (Sample1) |
| Test2 | <Sample2> |

### 3.3. 接続パラメータの解析

プロバイダでは, AddController メソッドの引数として使用するオプション文字列を規定された書式で解析し, その値を取得する CConnectOption クラスがあります. このクラスを使用することで接続パラメータを他のプロバイダ同様に統一することができます. 実装の詳細に関しては<インストールフォルダ>¥ORiN2¥CAO¥Include の ConnectOption.h を参照ください.

CConnectionOption クラスが扱う接続パラメータは以下のいずれかの書式に従います. パラメータの値域ついては, 表 3-5, 表 3-6 を参照ください.

Conn = ETH:<接続先 IP>[:<接続先ポート>[:<ローカル IP>[:<ローカルポート>]]]
= TCP:<接続先 IP>[:<接続先ポート>[:<ローカル IP>[:<ローカルポート>]]]
= UDP:<接続先 IP>[:<接続先ポート>[:<ローカル IP>[:<ローカルポート>]]]
= COM:<COM 番号>[:<ボーレート>[:<パリティ>:<データビット>:<ストップビット>:<フロー制御>]]

**表 3-5 Ethernet 系(ETH, TCP, UDP)のパラメータ**

| パラメータ | 値域 | デフォルト値 |
|---|---|---|
| 接続先 IP(IPv4) | 0.0.0.0 ~ 255.255.255.255 | - |
| 接続先ポート | 0 ~ 65535 | 5001 |
| ローカル IP(IPv4) | 0.0.0.0 ~ 255.255.255.255 | 127.0.0.1 |
| ローカルポート | 0 ~ 65535 | 0 (0:空いているポートが割り当てられます) |

**表 3-6 シリアル通信(COM)のパラメータ**

| パラメータ | 値域 | デフォルト値 |
|---|---|---|
| COM 番号 | 1 ~ 256 | - |
| ボーレート[bps] | 1 ~ 4,294,967,295 | 38400 |
| パリティ | 0 ~ 4<br>(0:None , 1:Odd , 2:Even , 3:Mark, 4:Space) | 0:None |
| データビット[bit] | 4 ~ 8 | 8 |
| ストップビット[bit] | 0 ~ 2 | 1 |
| フロー制御 | 0 ~ 3<br>(0 : フロー制御なし<br>1 : -Xon/Xoff | 0 : フロー制御なし |

| | | |
|---|---|---|
| | 2：ハードウェア制御<br>(OR 演算可能)) | |

角括弧(“[ ]”)内は省略可能を示します． ETH オプションの場合，以下の 4 つの記述が可能です．省略した場合，デフォルト値が入ります．

例1) 接続先 IP：192.168.0.1
→Conn=ETH:192.168.0.1

例2) 接続先 IP：192.168.0.1，接続先ポート：8080
→Conn=ETH:192.168.0.1:8080

例3) 接続先 IP：192.168.0.1，接続先ポート：8080，ローカル IP：192.168.0.2
Conn=ETH:192.168.0.1:8080:192.168.0.2

例4) 接続先 IP：192.168.0.1，接続先ポート：8080，ローカル IP：192.168.0.2，ローカルポート：80
Conn=ETH:192.168.0.1:8080:192.168.0.2:80

また，デフォルト値を変更する場合，CConnectOption クラスの SetDefault メソッドを使用します．以下の 3 つのパターンが使用可能です．

```
/* Ethernet用のデフォルトパラメータを設定 */
HRESULT SetDefault(PARAM_CONN_ETH stEth)

/* シリアル通信のデフォルトパラメータを設定 */
HRESULT SetDefault(PARAM_CONN_COM stCom)

/* Ethernet用，シリアル通信のデフォルトパラメータを設定 */
HRESULT SetDefault(PARAM_CONN_COM stCom, PARAM_CONN_ETH stEth)


// Ethernet 用のパラメータ構造体
struct PARAM_CONN_ETH {
    DWORD dwSrcIP;          // ローカルIPアドレス inet_addr()形式
    DWORD dwSrcPort;        // ローカルポート番号
    DWORD dwDestIP;         // 送信先IPアドレス inet_addr()形式
    DWORD dwDestPort;       // 送信先ポート番号
};

// シリアル通信用のパラメータ構造体
struct PARAM_CONN_COM {
    DWORD dwPortNo;         // COMポートの番号
    DWORD dwBaudRate;       // ボーレート
    DWORD dwParity;         // パリティ
    DWORD dwDataBits;       // データビット数
    DWORD dwStopBits;       // ストップビット数
    DWORD dwFlow;           // フロー制御
};
```

オプション文字列を解析するには，GetConnectOption メソッドを使用します．

```
/*  接続オプション解析 */
HRESULT GetConnectOption(BSTR bstrSorce,        // 解析対象文字列
                         PARAM_CONN* pRet)      // 解析結果格納先


// Ethernet 用, シリアル通信用のパラメータ共用体
struct PARAM_CONN {
    WORD wType;       // パラメータタイプ
                      // ETH:TYPE_ETH, TCP:TYPE_TCP, UDP:TYPE_UDP, COM:TYPE_COM
    union {
        PARAM_CONN_ETH stEth;          // Ethernet 用のパラメータ構造体
        PARAM_CONN_COM stCom;          // シリアル通信用のパラメータ構造体
    };
};
```

以下にCConnectOptionクラスの使用例を示します. ボーレート:19200 bps, パリティ:None, データビット:8 bit, ストップビット:2 bit, フロー制御:0(制御なし) をデフォルトとして設定し, オプション文字列”Conn=COM:2:4800”を解析した場合の結果を示します.

```
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);

// 解析結果
// 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 型の変換

プロバイダではユーザから入力される値を VARIANT 型で受け付けます. VARIANT 型は VT_I1 型や VT_BSTR 型など様々な型の値を保持することができます. そのため, プロバイダのライブラリでは, VARIANT 型から任意の型へ変換する機能を持つ CDataConv クラスが提供されています. この機能を使用することで, ユーザはプロバイダから求められている型で厳密に指定しなくてもよくなり, ユーザが扱いやすいプロバイダを実装することができます. 実装の詳細に関しては<インストールフォルダ>¥ORiN2¥CAO¥Include の DataConv.cpp/h を参照ください.

VARIANT 型を変換する場合, CDataConv クラスの ChangeVarType メソッドを使用します. 変換元の

VARIANT データ vntSrc, 変換型 vt, 変換後の格納先*pRet は必須で指定し, 変換後の最大要素数 lSize およびロケール ID ulLocaleID は任意で設定します. また, 変換元の VARIANT データ vntSrc には配列(VT_ARRAY 型)も指定することができ, 各配列要素を要求する変換型 vt に変換し, 変換完了した要素数返り値として long 型で返します.

```
/* VARIANT 型の変換 */
// @retval    long :  変換完了した要素数
long CDataConv::ChangeVarType(VARIANT vntSrc,                  // VARIANT データ(変換元)
                              VARTYPE vt,                      // 変換型
                              void *pRet,                      // 変換後の格納先
                              long lSize /*= NO_CHECK_SIZE*/,     // 変換後の最大要素数
                              unsigned long ulLocaleID /*= 0*/)   // ロケール ID
```

例えば, ユーザが Variable クラスの PutValue を実行した場合, CCaoProvVariable::FinalPutCtrlUserValue 関数(詳細は後述)が呼び出されます. FinalPutCtrlUserValue の引数 VARIANT 型の newVal にはユーザが設定した引数が渡ります. プロバイダが引数として VT_I2 型を要求し, DataConv クラスを使用しない場合は以下のように記述することとなり, ユーザは newVal.vt = VT_I2 とし, newVal.iVal に必ず値を入れないといけないという制限が発生します. よって, このプロバイダはユーザ視点で扱いにくいプロバイダになってしまいます.

＜CDataConv クラスを使用しない場合＞

```
HRESULT CCaoProvVariable::FinalPutCtrlUserValue(VARIANT newVal)
{
    if(newVal.vt != VT_I2){            // newVal が VT_I2 型以外はエラー
        return E_INVALIDARG;
    }
    short sParam = newVal.iVal;
    : (以降処理が続く)
}
```

CDataConv クラスを使用した例を以下に示します. CDataConv クラスでは, 引数で与えられた newVal を可能な限り VT_I2 に変換しようと試みます. 変換可能であれば, 変換完了した要素数として 1 を返します. この場合, ユーザは newVal に VT_BSTR 型や VT_UI1 型を指定できるため, より扱いやすいプロバイダを開発することができます.

＜CDataConv クラスを使用した場合＞

```
HRESULT CCaoProvVariable::FinalPutCtrlUserValue(VARIANT newVal)
```

```
{
    CDataConv dataConv;
    short sParam;
    long changedElem = dataConv.ChangeVarType(newVal,VT_I2,&sParam);
    if(changedElem != 1){
        return E_INVALIDARG;
    }
    : (以降処理が続く)
}
```

また, CDataConv で配列型の引数を変換する場合は以下のように記述します.

＜CDataConv クラスを使用した場合＞

```
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;
    }
    : (以降処理が続く)
}
```

## 3.5. エラー作成方法

プロバイダでは, プロバイダ固有のエラーに対応可能であり, 独自のエラーコード等を追加することで, これを CAO クライアントに渡すことができます. こういった独自のエラー情報を渡すことでプロバイダのデバッグ効率が向上し, CAO クライアント実装時にエラー処理が容易になります.

エラーを作成するには以下の手順をとります.

(1) エラーコードの定義する.
(2) エラーメッセージを定義する.
(3) エラーを発生させる.

以下でその詳細について説明します.

(1) エラーコードの定義は, ヘッダファイルにまとめて定義します. (ここでは StdAfx.h を使用)以下に定

義の実装例(エラーコード E_SAMPLE)を示します.

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

このときエラー名には接頭辞として“E_”をつけます. そしてエラーコードには _HRESULT_TYPEDEF_()マクロの中で 32 ビットの数字を指定します. ここでプロバイダに用意されているエラーコードは 0x80100000～0x801FFFFF(FACILITY CODE = 0x10)です.

**表 3-7 CAO のエラーコード割り当て**

| エラーコード | 使用モジュール |
|---|---|
| 0x80000200～0x800003FF | CAO |
| 0x80000400～0x800005FF | CAO Provider テンプレート |
| 0x80000600～0x80000FFF | その他のモジュール |
| 0x80100000～0x801FFFFF | CAO Provider |

(2) エラーメッセージの定義は, リソースの StringTable でおこないます. ここではエラーメッセージをリソースとして登録する方法を示します.

1. CAOPROV プロジェクト下の CAOPROV.rc をリソースビューで開き, StringTable をダブルクリックします.

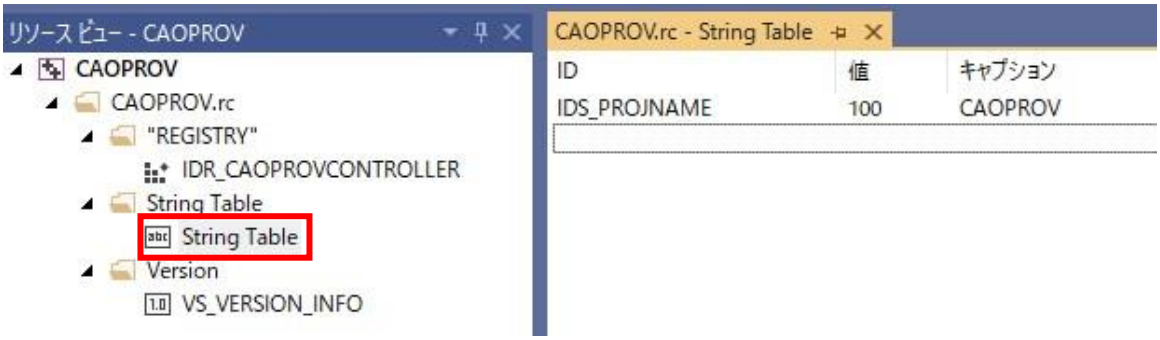



**図 3-1 プロバイダの String Table(初期状態)**

2. 以下の String のプロパティが出るので ID とキャプションを記述します. 例を示します.
   ID:IDS_E_SAMPLE
   キャプション:“This is test.”
   ここで, 自分が設定したリソース以外の値は書き換えないように注意してください.

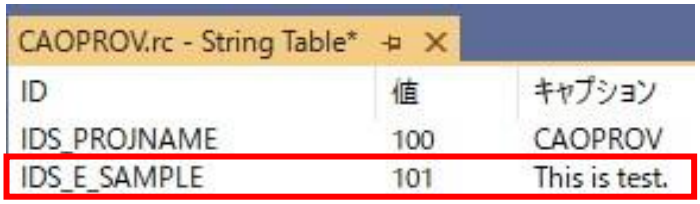



**図 3-2 String のプロパティ編集画面**

ここで ID とは StringTable のリソース ID, キャプションがエラーメッセージです. リソース ID には接頭辞として“IDS_”をつけ, その後ろに(1)で定義したエラーコード(この場合 E_SAMPLE)を付加します. これで Provider が持つリソースにエラーメッセージが追加されました.

3. リソースファイル CAOPROV.rc をコンパイルします. コンパイルに成功すると, Resource.h に作成したリソースが登録されます.

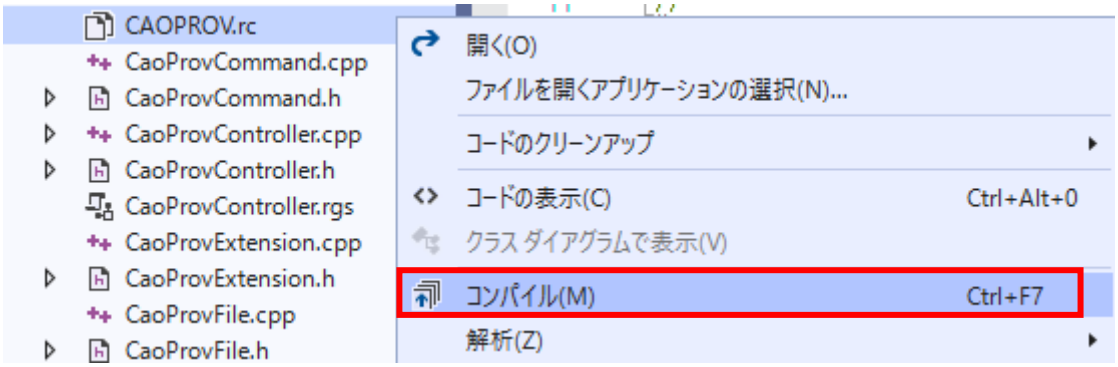



**図 3-3 CAOPROV.rc のコンパイル**

4. プロバイダの実装部に許可されているリソースの値は 1024～1535 であるため, 以下の方法で追加したリソースの値を割り当て直します. Resource.h の中で以下のような定義で, リソース ID に値を割り当てている箇所があります. ここの値を書き換えてください. 以下に, 101 に割り当てられたリソースを 1024 に割り当て直す例を示します.

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

5. 再度 CAOPROV.rc をコンパイルし, StringTable の値が変更されていることを確認してください.

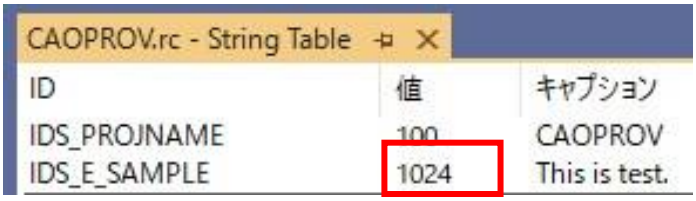



**図 3-4 再コンパイル後の String のプロパティ編集画面**

(3) (1), (2)で作成したエラーコードやリソースを使ったエラーを発生させます. エラーコードは標準の HRESULT のエラーと同様に扱うことができます. ただし, エラーの詳細情報((2)で作成したリソース)をクライアントに渡したい場合は, Error()をコールしなければなりません. Error()はクライアントにエラー情報を提供する為のメソッドです. 以下に Error()の定義を示します.

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

ここで引数は上からリソース ID, インタフェース ID, エラーコード, リソースへのハンドルです. リソース ID には, (2)で定義したリソース ID を入れます. インタフェース ID には, 作成したエラーを返すインタフェースの ID を入れます. エラーコードには(1)で定義したエラー名を入れます. 最後のリソースへのハンドルは, デフォルト値を使用するため特に指定する必要はありません.

ここでユーザ定義のエラーの使用例, ユーザ定義のエラー“E_SAMPLE”を使う場合を以下に示します.

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

上の例では Error 関数の第 2 引数は CAOVariable クラスのインタフェース ID になっています. 実際にはエラーが発生したクラスのインタフェース ID を入れてください. インタフェース ID は<インストールフォルダ>ORiN2¥CAO¥Include¥CAOPROV_i.c の中で定義された IID 型の定数から選択できます. 上記のエラーを CaoTester2.exe で発生させると, ログに以下のように表示されます.


**図 3-5 CaoTester2 のエラーログ**

## 3.6. メッセージイベントについて

プロバイダには任意のタイミングでメッセージイベントを発生させることができます. メッセージイベントはCreateMessage()でメッセージを生成し, SendMessage()でイベントを発生させることができます. ここで以下にCreateMessage()と SendMessage()の定義を示します.

```
STDMETHODIMP CreateMessage(
    TMess** ppMess,                 // メッセージオブジェクト
    long lNumber = 0,               // メッセージ番号
    VARIANT *vntData = NULL,        // メッセージ本文
    VARIANT *vntDateTime = NULL,    // 作成日時
    BSTR bstrReceiver = NULL,       // 送り先
    BSTR bstrSender = NULL,         // 送り元
    BSTR bstrDescription = NULL     // 説明
);

STDMETHODIMP SendMessage(
    TMess* pMess,               // 送信するメッセージ
    long lOption = 0            // オプション
)
```

CreateMessage()の第 2 引数は, メッセージ番号であり, 任意の値を指定することができます. 第 3 引数のデータにはログとして書き込まれるデータを入力します. ログ出力としてサポートされている VARIANT の型は VT_BSTR です[2]. 第 4 引数の発生時間にはメッセージを発信するときの日時を入れ, VARIANT の型は VT_DATE にします[3]. 第 5 引数は, ログの書き込み要求の場合, 無効になるので何も指定する必要はありません. 第 6 引数の発信元には, 必要に応じてメッセージを出力するオブジェクトの名前を入れてください. 第 7 引数のエラーの説明は, 必要であれば設定してください.

次に SendMessage()は, 第 1 引数に, CreateMessage()で生成したメッセージを指定してください. 第 2 引数には, メッセージオプションを指定します. メッセージオプションに指定できる値を表 3-8 に示します. CAO_MSG_NORMAL が一般メッセージです. 一般メッセージは, CAO エンジン内では何も処理されず, そのままクライアントへ転送されます.

以下にメッセージメカニズムの概要図を示す.

[2] メッセージイベント自体は多様な VT 型に対応するが, ログ出力機能で使用する際, BSTR 型以外はメッセージが有効に出力されません.
[3] VARIANT の型が VT_EMPTY ならば自動的に, CreateMessage を実行したときの日時が埋め込まれます.

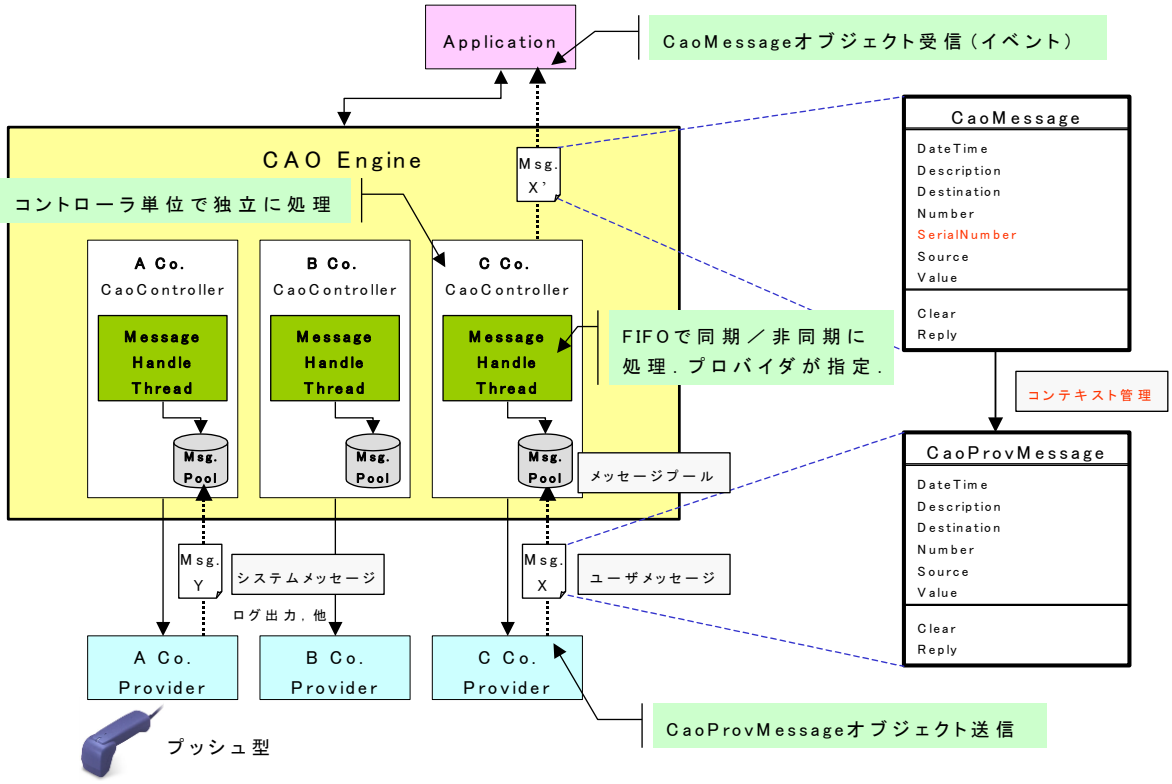



**図 3-6 CAO メッセージメカニズム**

図 3-6 のようにメッセージは，CAO エンジンのメッセージプールに蓄積された後，アプリケーションに送信されます．このメッセージプールの最大サイズは 1000 個であり，最大数を超えてメッセージを出力した場合は非同期タイプのメッセージの場合でも，メッセージプールに空きができるまで“待ち”状態になります．

**表 3-8 メッセージオプションとその動作**

| | メッセージオプション | 動作 | 備考 |
|---|---|---|---|
| 通常メッセージ | CAO_MSG_NORMAL (=0x00000000) | メッセージをメッセージプールに格納します．格納後は，メッセージの送信を確認せずにプロバイダに制御を戻します． | - |
| 同期メッセージ | CAO_MSG_SYNC (=0x00010000) | メッセージをメッセージプールに格納します．格納後は，メッセージの送信を確認した後にプロバイダに制御を戻します． | |

| | | | |
|---|---|---|---|
| ログ書き込み要求 | CAO_MSG_OUTPUT_LOG (=0x00020000) | ログとしてメッセージを出力します. | メッセージオプションの下位2バイトでログレベルを設定します.<br>Debug : 0x00020000<br>Info : 0x00020001<br>Warn : 0x00020002<br>Error : 0x00020003<br>Fatal : 0x00020004 |
| エンジン制御メッセージ | CAO_MSG_SYSTEM (=0x00040000) | エンジンに制御メッセージを送信します. | - |
| 緊急メッセージ | CAO_MSG_BYPASS (=0x00080000) | メッセージを CAO エンジンのメッセージプールに格納せずに送信を行います.<br>メッセージプールに前のメッセージがあるときは, 順番に割り込んで送信します. | - |
| インプロセスメッセージ転送 | CAO_MSG_PROVIDER (=0x00100000) | プロバイダにメッセージを転送します. | メッセージの“Detination”プロパティに送信先のプロバイダ名を設定します.<br>このとき検索するプロバイダは, 同一ワークスペース内のコントローラに限定されます.<br>複数のプロバイダに送信する場合は, カンマ(, )で区切って指定してください.<br>(例 “Test1, Test2”)<br><br>送信先が空文字列のときは, コントローラコレクションの全てのプロバイダに送信します. |

これらのオプション値は送信方法と転送先の 2 種類に大別され, この 2 種類を複合して使用することができます.

以下に組み合わせとそのオプション値の一覧を示します.

**表 3-9 メッセージオプション値の組み合わせ**

| 送信方法 / 送信先 | 通常 | 同期 | 緊急 |
|---|---|---|---|
| クライアント | CAO_MSG_NORMAL (=0x00000000) | CAO_MSG_SYNC (=0x00010000) | CAO_MSG_BYPASS (=0x00080000) |
| ログ | CAO_MSG_OUTPUT_LOG (=0x00020000) | CAO_MSG_OUTPUT_LOG ＋CAO_MSG_SYNC (=0x00030000) | CAO_MSG_OUTPUT_LOG ＋CAO_MSG_BYPASS (=0x000A0000) |
| エンジン制御 | CAO_MSG_SYSTEM (=0x00040000) | CAO_MSG_SYSTEM ＋CAO_MSG_SYNC (=0x00050000) | CAO_MSG_SYSTEM ＋CAO_MSG_BYPASS (=0x000C0000) |
| プロバイダ | CAO_MSG_PROVIDER (=0x00100000) | CAO_MSG_PROVIDER ＋CAO_MSG_SYNC (=0x00110000) | CAO_MSG_PROVIDER ＋CAO_MSG_BYPASS (=0x00180000) |

以下に，通常メッセージでログに出力するときの例を示します．

```
// メッセージの作成
CComPtr<CCaoProvMessage> pMess;
CComVariant vntData(L"This is test.");
HRESULT hr = CreateMessage(&pMess,lType, &vntData);
if (SUCCEEDED(hr)) {
  // メッセージの送信
  hr = SendMessage(pMess, CAO_MSG_OUTPUT_LOG);
}
```

### 3.6.1. メッセージイベントによるログ出力

ここでは，メッセージイベントを使用した「ログ出力」について説明します．ログ出力に関する説明は『ORiN2 プログラミングガイド』の「2.2.6.ログ出力について」を参照してください．ログ出力をおこなうには，SendMessage ()の第 2 引数メッセージ ID に CAO_MSG_OUTPUT_LOG を指定します．

ログの出力に関する設定は CAO サポートツールの CaoConfig.exe で設定します．以下に CaoConfig.exe の画面を示します．

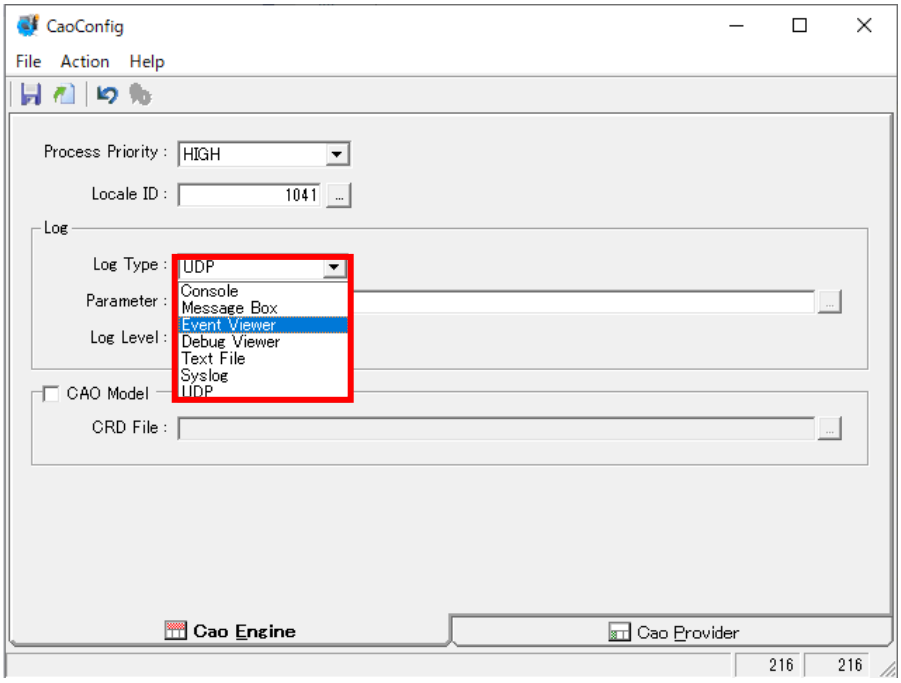



**図 3-7 CAOConfig .exe の画面**

ログの出力先はコンボボックスの Log Type から選択することができます. (図 3-7 参照)

**表 3-10 ログ出力先**

| 出力先 | 備考 |
|---|---|
| Console | コンソールに出力します |
| Message Box | メッセージボックスに出力します(サービス起動時) |
| Event Viewer | イベントビューワに出力します(サービス起動時) |
| Debug Viewer | デバッグ出力します. |
| Text File | 指定したテキストファイルに出力します. |
| Syslog | SysLog サーバに出力します. |
| UDP | UDP サーバに出力します. |

詳細は『ORiN2 プログラミングガイド』にある「CaoConfig について」を参照してください.

### 3.6.2. インプロセスメッセージ転送

ここでは, メッセージを異なるプロバイダのコントローラに転送する「インプロセスメッセージ転送」の送受信方法について説明します.

インプロセスメッセージ転送を行うときは, CaoMessage オブジェクトに転送先のコントローラ名を指定する必

要があります. 転送先のコントローラ名は, 送信するメッセージオブジェクトの get_Destination()に値を設定します. CaoMessage クラスの実装がデフォルトのとき, get_Destination()の値は CreateMessage()の第 5 引数で指定することができます. インプロセスメッセージ転送を行うことができるプロバイダは, 送信元のプロバイダが属している CaoWorkspace オブジェクト内のコントローラに限られます. また, 転送先に空文字列を指定することで, CaoWorkspace オブジェクト内の全てのコントローラにメッセージを転送することができます.

メッセージを送信するときは SendMessage()の第 2 引数に「CAO_MSG_PROVIDER」を指定します.

転送先のプロバイダでは, コントローラオブジェクトの OnMessage()が呼び出され, 第 1 引数にインプロセス転送メッセージが格納されています. このため, インプロセス転送メッセージを受け取るためには, 転送先プロバイダでコントローラオブジェクトの OnMessage()が実装されていなければなりません.

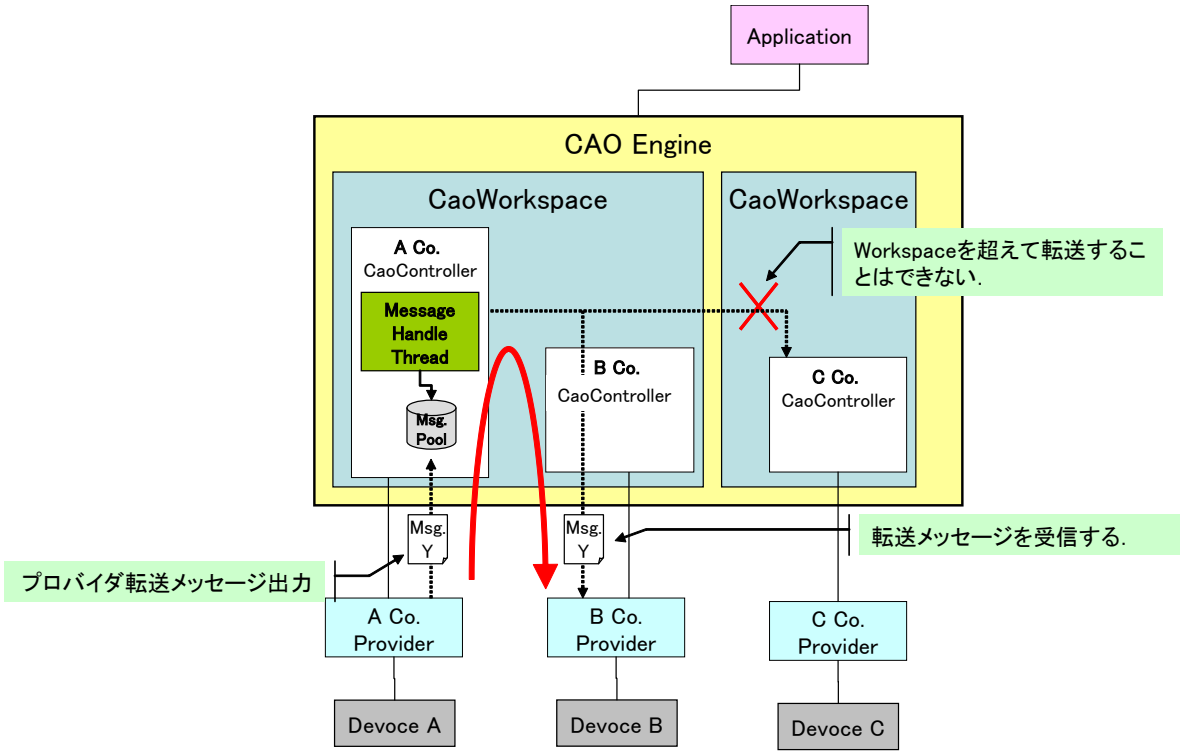



**図 3-8 プロバイダ転送メッセージ**

以下に送信側および受信側のサンプルを示します.

**List 3-1 インプロセスメッセージ転送(送信側)**

```
HRESULT CCaoProvController::CreateInprocMsg(LONG ID, VARIANT vntData, BSTR bstrDest)
{
  HRESULT hr;
  CComPtr<CCaoProvMessage> pMess;  // 生成するメッセージ

  // メッセージの作成
  hr = CreateMessage(&pMess, ID, &vntData, 0, bstrDest, 0, 0);

  // メッセージ成功時はメッセージの送信
  if (SUCCEEDED(hr)) {
      // メッセージの送信
```

```
        hr = SendMessageA(pMess, CAO_MSG_PROVIDER);
    }

    return hr;
}
```

**List 3-2 インプロセスメッセージ転送(受信側)**

```
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. メッセージイベントの一定周期発行

CAO には, CAO エンジンがプロバイダを一定周期で呼び出し, プロバイダからの依頼で CAO エンジンがクライアントに対してイベントを発行する仕組みがあります. この仕組みをプロバイダで有効にするには実装時に CAOP_TIMER_INTERVAL マクロを 1 以上に定義する必要があります.

CAOP_TIMER_INTERVAL マクロはデフォルトではプロバイダプロジェクトの StdAfx.h ファイルに以下に示すように定義されています.

```
#define CAOP_TIMER_INTERVAL    0
// 0: off, n: n(msec)毎に OnTimer イベントを発生させる. LONG_MAX 以下の正の整数.
```

CAOP_TIMER_INTERVAL は CAO エンジンがプロバイダを一定周期で呼び出す間隔をミリ秒(ms)単位で指定します. 0 の場合は呼び出しがないことを意味します.

CAOP_TIMER_INTERVAL マクロを 1 以上に定義すると, CAO エンジンはプロバイダの CCaoProvController::OnTimer()を一定周期(<CAOP_TIMER_INTERVAL >ms)で呼び出します.

以下に OnTimer()メソッドの実装例を示します.

**List 3-3 CCaoProvController.cpp – OnTimer()**

```
#if CAOP_TIMER_INTERVAL > 0
/**    タイマーイベント
 *
 * #define CAOP_TIMER_INTERVAL 0 の 定義を ms 単位指定で定義し直す.
 * これにより CAOP_TIMER_INTERVAL ms 間隔でこの関数が呼ばれる.
 *
 */
void CCaoProvController::OnTimer()
```

```
{
    HRESULT hr;

    CComPtr<CCaoProvMessage> pMess;
    CComVariant vntData(L"Log OK?");
    hr = CreateMessage(&pMess, -1, &vntData);
    if (SUCCEEDED(hr)) {
        SendMessage(pMess);
    }
    return;
}
#endif
```

CAOP_TIMER_INTERVAL の仕組みを使用すればクライアント側でポーリング処理をおこなわなくてもプロバイダが代行してくれるのでコーディングが非常にシンプルにできます.

## 3.7. マクロプロバイダの作成方法

マクロプロバイダとは, 別のプロバイダに接続するためのプロバイダです. このとき接続するプロバイダはマクロプロバイダの実装方法により, 複数のプロバイダに接続することもできます.

以下では, マクロプロバイダ作成に必要なプロバイダオブジェクトの生成方法と OnMessage イベントの取得方法について説明します.

### 3.7.1. プロバイダオブジェクトの生成方法

プロバイダオブジェクトは, コントローラのみ外部からの作成が可能になっています. コントローラオブジェクトを生成するときは, CoCreateInstance()又は CoCreateInstanceEx()を使用します.

また, コントローラ以外のオブジェクトはプロバイダのインタフェースを使用して生成します. ただし, メッセージオブジェクトは外部から作成することはできません. メッセージオブジェクトは OnMessage イベントでプロバイダが生成したオブジェクトを取得することができます. (参照 3.7.2)

以下に各オブジェクトとその生成に使用するメソッドの一覧を示します.

**表 3-11 プロバイダオブジェクトと生成メソッド**

| プロバイダオブジェクト | 生成メソッド |
|---|---|
| CaoProvController | CoCreateInstance()<br>CoCreateInstanceEx() |
| CaoProvCommand | CaoProvController::GetCommand() |
| CaoProvExtension | CaoProvController::GetExtension() |
| CaoProvFile | CaoProvController::GetFile()<br>CaoProvFile::GetFile() |
| CaoProvRobot | CaoProvController::GetRobot() |

| | |
|---|---|
| CaoProvTask | CaoProvController::GetTask() |
| CaoProvVariable | CaoProvController::GetVariable()<br>CaoProvExtension::GetVariable()<br>CaoProvFile::GetVariable()<br>CaoProvRobot::GetVariable()<br>CaoProvTask::GetVariable() |
| CaoProvMessage | OnMassage イベント |

以下にコントローラオブジェクトを生成する関数のサンプルを示します.

**List 3-4 Sample.cpp**

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

    // プロバイダの生成
    USES_CONVERSION;

    CLSID clsid;
    ICaoProvController *pICaopCtrl;

    hr = CLSIDFromProgID(bstrProvider, &clsid);
    if (SUCCEEDED(hr)) {
        // プロバイダインスタンスの作成
        if (SysStringLen(bstrMachine) == 0) {
            // インプロセス処理 (CLSCTX_INPROC_SERVER)
            hr = CoCreateInstance(clsid,
                                  NULL,
                                  CLSCTX_INPROC_SERVER,
                                  IID_ICaoProvController,
                                  (void **)&pICaopCtrl);
        } else {
            // アウトプロセス処理
            DWORD dwLen = MAX_COMPUTERNAME_LENGTH + 1;
            LPTSTR pTstr = new TCHAR[dwLen + 1];
            GetComputerName(pTstr, &dwLen);
            if (strcmpi(pTstr, W2T(bstrMachine)) == 0) {
                // 指定されたマシン名が自分のマシン名の場合
                // (CLSCTX_LOCAL_SERVER)
                hr = CoCreateInstance(clsid,
                                      NULL,
                                      CLSCTX_LOCAL_SERVER,
                                      IID_ICaoProvController,
                                      (void **)&pICaopCtrl);
            } else {
                // 指定されたマシン名が自分以外のマシン名の場合
                // (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;
        }

        // ロボットコントローラの接続
        hr = pICaopCtrl->Connect(bstrName, bstrOption);
        if (FAILED(hr)) {
            pICaopCtrl->Release();
            return hr;
        }

        *ppICaopCtrl = pICaopCtrl;

        return hr;
    }
```

### 3.7.2. OnMessage イベントの取得方法

プロバイダオブジェクトからの OnMessage イベントを取得するには, EventSink クラスをマクロプロバイダに実装する必要があります.

EventSink クラスの実装には以下の手順をおこなわなければなりません.

(1) IDL ファイルへの登録. (参照 3.7.2.1)

(2) プロバイダへの接続, 切断処理の実装. (参照 3.7.2.2)

(3) OnMessage イベント発生時の処理の実装. (参照 3.7.2.2)

(4) EventSink オブジェクト生成処理の追加. (参照 3.7.2.3)

これらの各手順については, 以下に順番に説明します.

#### 3.7.2.1. EventSink の IDL ファイルへの追加

EventSink クラスをマクロプロバイダの IDL ファイルに登録するには, 以下の手順でおこないます.

(1) EventSink クラスの IDL ファイルを作成します.

(2) マクロプロバイダの CaoProv.idl ファイルに“CAOPROV_APPEND_CLASS”マクロを使用して登録します.

ここで“CAOPROV_APPEND_CLASS”マクロの使用方法を以下に示します.

```
#define CAOPROV_APPEND_CLASS <EventSink の IDL ファイルへのパス>
```

以下に, EventSink.idl ファイルマクロプロバイダに登録する場合の例を示します.

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

プロバイダは, “CAOPROV_APPEND_CLASS”マクロで指定したパスにある IDL ファイルをインクルードします.

以下に EventSink クラスの IDL ファイルのサンプルを示します.

**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("ﾒｿｯﾄﾞ 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. EventSink クラスの実装

EventSink クラスでは, OnMessage イベントを生成するオブジェクトと接続及び切断処理を実装しなければなりません. 接続には AdlAdvise(), 切断には AtlUnadvise()を使用します. これらの関数の詳細については MSDN を参照してください.

以下に接続と切断処理のサンプルを示します.

**List 3-6 Sample.cpp**

```
// 接続処理
STDMETHODIMP CEventSink::Connect(ICaoProvController *pICaopCtrl)
```

```
{
     // CaoProvider のコネクタブルオブジェクトと接続する.
     HRESULT hr = AtlAdvise(pICaopCtrl, GetUnknown(), DIID__ICaoProvControllerEvents,
&m_dwCookie);
     if (SUCCEEDED(hr)) {
         // ICaoProvController インタフェースの保存
             m_pICaopCtrl = pICaopCtrl;
     }
     return hr;
}

// 切断処理
STDMETHODIMP CEventSink::Disconnect()
{
     // CAOProvider のコネクタブルオブジェクトとの接続を解除し，イベント通知を無効にする.
     if (m_dwCookie) {
         AtlUnadvise(m_pICaopCtrl, DIID__ICaoProvControllerEvents, m_dwCookie);
     }
     m_dwCookie = 0;
     return S_OK;
}
```

また OnMessage イベントは，3.7.2.1 で登録した EventSink インタフェースの DISPID が“0x01”のメソッドに対して発生します. よって，OnMessage イベント処理は DISPID が“0x01”のメソッドに実装します.

以下に，プロバイダから受信したメッセージを CAO エンジンにそのままイベントとして送信する，OnMessage 受信メソッドのサンプルを示します.

**List 3-7 Sample.cpp**

```
STDMETHODIMP CEventSink::OnMessage(ICaoProvMessage *pICaopMsg)
{
     HRESULT hr;
     // コントローラクラスのイベント生起
     hr = m_pCaopCtrl->Fire_OnMessage((IUnknown*)pICaopMsg);
     return hr;
}
```

#### 3.7.2.3. EventSink の生成

EventSink クラスは生成され，接続処理を実行することで OnMessage イベントを受信することができます.

以下に EventSink の生成，消滅のサンプルを示します.

**List 3-8 Sample.cpp**

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

     // CEventSink のインスタンス作成
     CEventSink *pEvent;
     hr = CComObject<CEventSink>::CreateInstance(&pEvent);
```

```
        if (FAILED(hr)) {
            return hr;
        }
        pEvent->AddRef();

        // プロバイダとの接続
        hr = pEvent ->Connect(pICaopCtrl);
        if (FAILED(hr)) {
            pEvent->Release(m_pICaopCtrl);
            return hr;
        }

        ppEventSink = pEvent;

        return hr;
    }

    HRESULT DeleteEvntSink(CEventSink *pEventSink)
    {
        HRESULT hr;

        // プロバイダとの切断
        hr = pEventSink->Disconnect();
        // EventSink の削除
        pEventSink->Release();

        return hr;
    }
```

## 3.8. 通信クラスの利用方法

### 3.8.1. はじめに

ORiN2 SDK では, プロバイダを作成するためにいくつかの通信クラスを利用しています. Device クラス系のクラス図を図 3-9 に示します. プロバイダ開発者は赤枠で示すクラスを使用することで, 機器と簡単に通信することができます.

本節では, 通信をおこなうための共通クラス CDevice クラスについて説明をおこない, この CDevice クラスを継承したシリアル通信をおこなうための CSerial クラス, TCP/IP 通信をおこなうための CTCPServer クラス, CTCPClient クラス, UDP 通信をおこなうための CUDPSocket クラスの利用方法について説明します.

Device クラス系 (Device.cpp/h , Serial.cpp/h , Socket.cpp/.h , TCPServer.cpp/h , TCPClient.cpp/h , UDPSocket.cpp/h)のソースファイルは<インストールフォルダ>¥ORiN2¥CAO¥Include から必要なファイルをプロジェクトに追加してください.

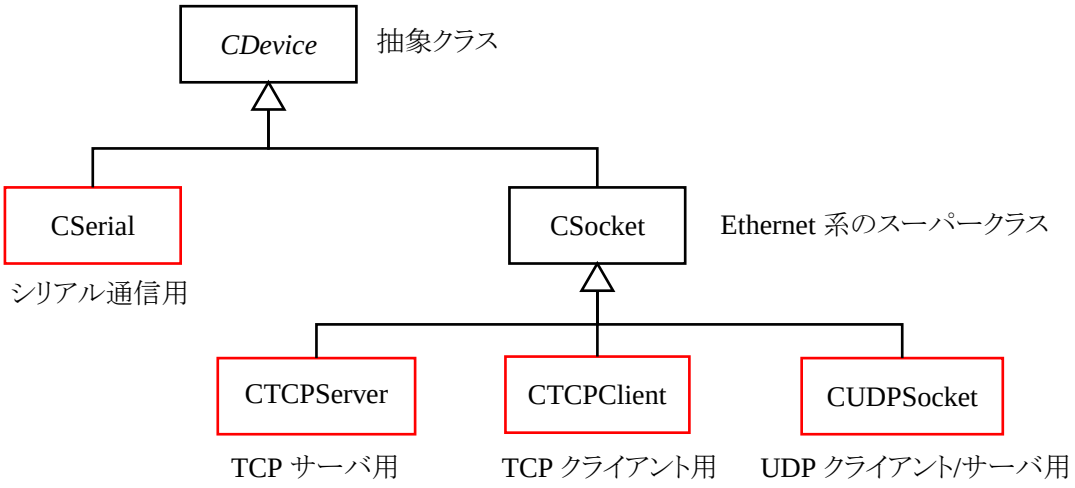



**図 3-9 Device クラス系のクラス図**

### 3.8.2. CDevice クラス

CDevice クラスは，デバイス通信をおこなうための抽象クラスです．

デバイス通信をおこなうクラスを実装するためには，この CDevice クラスを継承し，接続処理や送受信処理を仮想関数に実装すればよいということになります．

また，CDevice クラスでは共通機能として，以下の機能を提供しています．

・ バイナリモード，テキストモード

バイナリモードのときは，データを無加工で送受信します．

テキストモードのときは，文字コード変換，ヘッダ，ターミネータの付加などを行って送受信を行います．

・ ヘッダ，ターミネータの設定

テキストモードのときに，ヘッダ，ターミネータを付加して送受信を行います．

・ Unicode 変換モード

テキストモードのときに，入力データを ASCII に変換して送信し，受信データを Unicode 変換して返します．

・ ISO コード変換，EIA コード変換

テキストモードのときに，送受信データを ISO，EIA コードに変換して通信します．

また，CDevice クラスのメンバ一覧を表 3-12 に示します．CDevice クラスを継承する場合は，この表の太字で示したメソッドを必要に応じてオーバーライドしてください．

**表 3-12 CDevice のメンバー覧**

| | メンバ関数・変数 | 説明 |
|---|---|---|
| | **CDevice()** | CDevice のコンストラクタ |
| | **~CDevice()** | CDevice のデストラクタ |
| | **AddRef()** | 参照カウンタ(m_lRefCnt)を+1 します. |
| | **Cancel()** | 受信のキャンセルイベント(m_hCancel)を ON にし，受信を直ちにキャンセルします. CancelReset()が呼び出されるまで受信できなくなります. |
| | **CancelReset()** | 受信のキャンセルイベント(m_hCancel)を OFF にし，受信処理をできるようにします. |
| | **Clear()** | バッファクリアとエラークリアを行います.（純粋仮想関数） |
| | **Connect()** | ターゲットとの接続処理を行います.（純粋仮想関数） |
| | **Disconnect()** | ターゲットとの切断処理を行います.（純粋仮想関数） |
| | **GetCancelMode()** | メンバ変数 m_dwCancelMode の値を取得します. |
| | **GetDestination()** | 送信先情報を取得します. |
| | **GetHandle()** | ハンドルを取得します. |
| | **GetMode()** | 現在設定されている動作モードを取得します. |
| | **GetReceivedCount()** | データ受信数取得処理. 現在バッファにあるデータ数を取得します. |
| | GetReceivePacket() | パケット取得処理. ローカル受信バッファからデータを取得します.<br>テキストモードのときは，ヘッダからターミネータまでを1パケットとして取得します. ヘッダがないときはバッファの先頭から，ターミネータがないときはバッファの最後まで取得します.<br>バイナリモードのときは，指定されたサイズ分又はバッファ内のすべてのデータを取得します. |
| | **GetSource()** | 送信元情報を取得します. |
| | **GetTimeOut()** | タイムアウト時間取得処理. タイムアウトの時間を取得します. |
| | **Initialize()** | 初期化処理. モードの設定や各種変換機能などの設定を行います. |
| | Receive() | 受信処理. ターゲットから文字列データを受信します.<br>ReceiveData を呼び出し，データを加工します.<br>テキストモードの時は，ヘッダとターミネータの削除，Unicode 変換などを行った受信データを返します. .<br>バイナリモードのときは，無加工で受信データを返します. . |
| | **ReceiveData()** | データ受信のための仮想関数.<br>デバイスから受信したデータをバッファに格納し，1パケットを切り出して返します. |
| | **Release()** | 参照カウンタ(m_lRefCnt)を-1 します. |

| | | |
|---|---|---|
| | Send() | 送信処理. 指定文字列データをターゲットに送信します.<br>テキストモードの時は, ヘッダとターミネータの付加, 文字コードの ASCII 変換などを行った後, SendData を呼び出します.<br>バイナリモードのときは, 無加工で SendData を呼び出します. |
| | SendAndReceive() | 送受信一括処理. 送信→受信を一括して処理します. |
| | **SendData()** | データ送信のための仮想関数.<br>Send から受け取ったデータをデバイスに送信します. |
| | **SetCancelMode()** | 受信のキャンセルイベント(m_hCancel)のモードを変更します.<br>引数が 0 の場合は自動リセット, 0 以外の場合は手動リセットモードになります. |
| | **SetDestination()** | 送信先情報を設定します. |
| | **SetMode()** | 動作モードを再設定します. |
| | **SetSource()** | 送信元情報を設定します. |
| | **SetTimeOut** | タイムアウト時間を設定します. |
| | m_dwBufLen | ローカルの受信バッファ長. |
| | m_dwCancelMode | キャンセルモード. |
| | m_dwHeaderLen | ヘッダ長. |
| | m_dwMode | 動作モード. |
| | m_dwRetryCnt | リトライをおこなう最大回数. |
| | m_dwTermLen | ターミネータ長. 0～2 文字. |
| | m_dwTotalTimeOut | 文字列を送受信するまでの最大待ち時間. |
| | m_hCancel | 受信のキャンセルイベント |
| | m_lRefCnt | 参照されるカウンタ. |
| | m_pcHeader | ヘッダコード. |
| | m_pcTerm | ターミネータコード. |
| | m_szBuf | ローカルの受信バッファ. |

:Public な関数 :Protected な関数 :Protected な変数

### 3.8.3. シリアル通信クラス

#### 3.8.3.1. 利用方法

CDevice クラスを継承した, シリアル通信をおこなうための CSerial クラスの利用方法について説明します.

CSerial クラスは, Connect メソッドで指定した COM ポートに接続をおこない, Receive メソッドや Send メソッドを利用することで, データの送受信をおこなうことができます.

CSerial クラスを利用して, RS-232C 通信をおこなうには, 以下の手順のようになります.

(1) オブジェクトの生成

CSerialクラスを利用するには，Serial.hファイルをインクルードする必要があります．このとき，2.3-(7)で示したように，インクルードディレクトリを設定している必要があります．また，プロジェクトにはCSerial クラスの関数を実装している Serial.cpp と，CSerial クラスの基底クラス CDevice クラスの関数を実装している Device.cpp を既存のファイルとして追加する必要があります．

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

(2) 初期化処理

送受信文字列に付加するヘッダ・ターミネータの設定や文字コード変換の設定などをおこない，CSerial クラスを初期化します．

```
CHAR cHeader[] = "";            // ヘッダコード
CHAR cTerm[] = "¥r";            // ターミネータコード
DWORD dwMode = ST_MODE_TEXT;    // 動作モード
PVOID pArgs[] = { cHeader,cTerm,&dwMode };

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

初期化時には，PVOID 型の配列に以下の 4 つのパラメータを指定します．

・ ヘッダコード

データの始まりをあらわすために，送信データに付加するヘッダコードを指定します．

・ ターミネータコード

データの終端をあらわすために，送信データに付加するターミネータコードを指定します．

・ 動作モード

データの送受信時の動作モードを指定することができます．動作モードは Device.h で定義されているマクロにより，以下の 5 種類を指定することができます．2 種類以上を指定する場合は，以下のマクロの論理和をとってください．

・ **ST_MODE_BINARY**

データをバイナリ形式(バイト配列)で送受信します．

指定したデータをそのまま送受信します．ヘッダとターミネータの付加および除去は行いません．

・ **ST_MODE_TEXT**

データをテキスト形式（文字列）で送受信します．

送信時には，文字列にヘッダとターミネータを付加して送信します．

受信時には，受信データからヘッダとターミネータを削除した文字列を返します．

・ **ST_MODE_WBSTOWCS**

データを Unicode(WCHAR)から S-JIS(CHAR)に変換して送受信します. バイナリモードのときは, このモードを指定することはできません.

・ **ST_MODE_ASCTOISO**

データを ASCII コードから ISO コードに変換して送受信します.
UNICODE 変換機能と併用した時, データは UNICODE から指定したコードに変換して送受信をおこないます.

・ **ST_MODE_ASCTOEIA**

データを ASCII コードから EIA コードに変換して送受信します.
UNICODE 変換機能と併用した時, データは UNICODE から指定したコードに変換して送受信をおこないます.

(3) 通信パラメータを設定する

シリアルポート番号, 通信速度(ボーレート), データビット, ストップビット, パリティ, フロー制御などの通信パラメータを設定します. シリアル通信で必要となるパラメータは PARAM_CONN_COM 型の構造体に定義されています.

```
PARAM_CONN_COM connParam;
connParam.dwPortNo = 1;                 // COM ポートの番号    : COM1(1) ～
connParam.dwBaudRate = CBR_19200;       // スピード(bps)       : 4800bps(CBR_4800), ～
connParam.dwParity = NOPARITY;          // パリティ            : NOPARITY,ODDPARITY,EVENPARITY,
                                        //                       MARKPARITY,SPACEPARITY
connParam.dwDataBits = 8;               // データビット数      : 4～8bits
connParam.dwStopBits = TWOSTOPBITS;     // ストップビット数    : 1(ONESTOPBIT),1.5(ONE5STOPBITS),
                                        //                       2(TWOSTOPBITS)
connParam.dwFlow = 0;                   // フロー制御          : 0(制御なし),1(-Xon/Xoff),
                                        //                       2(ハードウェア制御)(OR 演算可能)
```

(4) 通信(COM)ポートをオープンする

(3)で定義した PARAM_CONN_COM 型の構造体を引数にして COM ポートをオープンします. COM ポートのオープンは以下のように Connect()でおこないます.

また, SetTimeout()でデータ送受信時のタイムアウト時間(ミリ秒)を設定します. 必ず Connect メソッドを実行した後に実施してください.

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

(5) データの読み書きをおこなう

COM ポートからデータを読み込んだり(Receive), 書き込んだり(Send)します. エラーが発生した場合はエラークリア・通信バッファクリア(Flush)をおこないます.

COM ポートからデータを読み込む場合は Receive メソッドを，書き込む場合は Send メソッドを次のように使用します．また SendAndReceive メソッドにより，データの送受信を一括でおこなうことができます．

```
BYTE lpszSendData[]{ 0x41,0x42 };     // 送信データ(文字列:"AB")
BYTE lpszReceiveData[16];             // 受信データ
DWORD dwRecvedLen;                 // 受信データ数
DWORD dwRetryCnt = 3;              // リトライ回数
```

・ データ送信

```
hr = serial.Send(lpszSendData,           // 送信データ
                 sizeof(lpszSendData),   // 送信データサイズ(0はNULL文字まで送信)
                                         // 省略した場合：0
                 true);                  // ヘッダとターミネータを付加オプション
                                         // true:付加 false:付加しない(省略時：true)
```

・ データ受信

```
hr = serial.Receive(lpszReceiveData,             // 受信データの格納先アドレス
                    sizeof(lpszReceiveData),     // 受信データのバッファサイズ
                    &dwRecvedLen);               // 実際に受信したデータ数
```

・ データ送信&受信

```
hr = serial.SendAndReceive(lpszSendData,             // 送信データ
                           sizeof(lpszSendData),     // 送信データサイズ(0はNULL文字まで送信)
                           lpszReceiveData,          // 受信データの格納先アドレス
                           sizeof(lpszReceiveData),  // 受信データのバッファサイズ
                           &dwRecvedLen,             // 実際に受信したデータ数
                           dwRetryCnt);              // リトライ回数（省略時：1回）
```

(6) 通信(COM)ポートをクローズする

使い終わったCOMポートを閉じます．COMポートをクローズするには，Disconnectメソッドを使用します．これを忘れるとオープンした COM ポートを他のアプリケーションで使用できなくなるので注意が必要です．

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

#### 3.8.3.2. エラーコード

シリアル通信クラスでは，固有のエラーコードとしてWindowsの標準エラーを“0x80070000”でマスクした値を返します．

例)Windows エラー:0x02(ファイルが見つからない) → CAO API のエラー:0x80070002

### 3.8.4. TCP ソケットクラス

#### 3.8.4.1. 利用方法

次に CDevice クラスを継承した，Ethernet のソケット通信(TCP/IP)をおこなう TCP ソケットクラスの利用方法について説明します．

TCP ソケットクラスは，CSerial クラスとは異なり，サーバモードとクライアントモードで接続を確立する際に利用するクラスが違います．それぞれサーバモードでは CTCPServer クラスを，クライアントモードでは CTCPClietnt クラスを利用します．

CTCPServer クラスは，クライアントからの接続を Accept()することで，CTCPClient クラスを生成します．接続の確立後は，CTCPClient クラスの Receive メソッドや Send メソッドを呼び出すことで，データの送受信をおこなうことができます．

**【サーバモードの場合】**

(1) オブジェクトの生成

CTCPServer クラスを利用するには，TCPServer.h ファイルをインクルードする必要があります．このとき，2.3-(7)で示したように，インクルードディレクトリを設定している必要があります．

プロジェクトには以下の 3 つのファイルを既存のファイルとして追加する必要があります．

1. TCPServer.cpp…CTCPServer クラスの実装ファイル
2. TCPClient.cpp…CTCPClient クラスの実装ファイル
3. Socket.cpp…CTCPServer の基底クラスである CSocket クラスの実装ファイル
4. Device.cpp…CSocket の基底クラスである CDevice クラスの実装ファイル

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

(2) 初期化

Initialize 時の設定パラメータとしてヘッダコード，ターミネータコード，動作モード（内容は CSerial クラスと同じ）になります．

Connect 時には，ソケット通信の接続を待ち受ける IP アドレスとポート番号を設定します． TCP 通信で必要となるパラメータは PARAM_CONN_ETH 型の構造体に定義されています．INADDR_NONE は自身の IP アドレスを意味します．

Connect が成功したら，タイムアウト時間を設定します．

```
CHAR cHeader[] = "";          // ヘッダコード
CHAR cTerm[] = "¥r";          // ターミネータコード
DWORD dwMode = ST_MODE_TEXT;  // 動作モード
PVOID pArgs[] = { cHeader,cTerm,&dwMode };

HRESULT hr = tcpServer.Initialize(pArgs);


PARAM_CONN_ETH connParam;
connParam.dwSrcIP = INADDR_NONE;        // ローカルIPアドレス
connParam.dwSrcPort = 5006;             // ローカルポート番号

hr = tcpServer.Connect(connParam);

DWORD dwTimeout = 1000;             // タイムアウト時間[ms]
hr = tcpServer.SetTimeout(dwTimeout);
```

(3) 接続待ち

TCPServer クラスは，そのオブジェクト自身がデータの送受信をおこなうものではありません．クライアントからの接続要求を待ち受け，接続要求が来るたびに，新しい接続ポイントを生成します．

実際には以下のように，新しく接続をおこなうための CTCPSocket のポインタを Connect メソッドの引数で渡し，クライアント側から接続要求があった場合は，新しいオブジェクトを引数に割り当てます．接続要求が来るまで，CTCPSocket クラスは Accept メソッドを連続的に実行する必要があります．

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

データの送受信をおこなうには，この新しく割り当てられたオブジェクトを利用します．

【クライアントモードの場合】

(1) オブジェクトの生成

CTCPClient クラスを利用するには，TCPClient.h ファイルをインクルードする必要があります．このとき，2.3-(7)で示したように，インクルードディレクトリを設定している必要があります．

プロジェクトには以下の 3 つのファイルを既存のファイルとして追加する必要があります．

1. TCPClient.cpp…CTCPClient クラスの実装ファイル
2. Socket.cpp…CTCPClient の基底クラスである CSocket クラスの実装ファイル
3. Device.cpp…CSocket の基底クラスである CDevice クラスの実装ファイル

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

(2) 初期化

必要なパラメータの設定をおこない，CTCPClient クラスを初期化します．Initialize 時の設定パラメータとしてヘッダコード，ターミネータコード，動作モード（内容は CSerial クラスと同じ）になります．

```
CHAR cHeader[] = "";          // ヘッダコード
CHAR cTerm[] = "¥r";          // ターミネータコード
DWORD dwMode = ST_MODE_TEXT;  // 動作モード
PVOID pArgs[] = { cHeader,cTerm,&dwMode };

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

(3) 接続を確立する

サーバプログラムが待ち受けているポートに対して，Connect()メソッドで接続要求をおこないます．

接続が成功すれば，このオブジェクトを利用してデータの送受信をおこなうことができます．TCP 通信で必要となるパラメータは PARAM_CONN_ETH 型の構造体に定義されています．ローカルの IP アドレスとポート番号，接続先の IP アドレスとポート番号を指定します．ローカルポート番号を 0 に設定することで，空いている適当なポート番号が割り振られます．

Connect が成功したら，SetTimeout()でタイムアウト時間を設定します．

```
PARAM_CONN_ETH connParam;
connParam.dwSrcIP = INADDR_NONE;                    // ローカルIPアドレス
connParam.dwSrcPort = 0;                            // ローカルポート番号
connParam.dwDestIP = inet_addr("127.0.0.1");        // 接続先の IP アドレス
connParam.dwDestPort = 5006;                        // 接続先のポート番号

hr = tcpClient.Connect(connParam);

hr = tcpClient.SetTimeout(1000);                    // タイムアウトの設定[ms]
```

**【TCP ソケットクラスでのデータの送受信】**

サーバモード，クライアントモードのどちらであれ，上記の方法で接続を確立すると，同じようにデータの送受信をおこなうことができます．(CTCPServer オブジェクトは，データの送受信処理は実装されていません．Accept メソッドで生成された CTCPClient オブジェクトを利用してください)

(4) データの送受信をおこなう

ソケット通信でデータの送受信をおこなうには，以下に示すメソッドを利用します．

```
BYTE lpszSendData[]{ 0x41,0x42 };  // 送信データ(文字列："A,B")
BYTE lpszReceiveData[16];          // 受信データ
DWORD dwRecvedLen;                 // 受信データ数
DWORD dwRetryCnt = 3;              // リトライ回数
```

・ **データ送信**

```
hr = tcpClient.Send(lpszSendData,          // 送信データ
                    sizeof(lpszSendData),  // 送信データサイズ(0はNULL文字まで送信)
                                           // 省略した場合：0
                    true);                 // ヘッダとターミネータを付加オプション
                                           // true:付加 false:付加しない(省略時：true)
```

・ **データ受信**

```
hr = tcpClient.Receive(lpszReceiveData,    // 受信データの格納先アドレス
                       sizeof(lpszReceiveData),  // 受信データのバッファサイズ
                       &dwRecvedLen);      // 実際に受信したデータ数
```

・ **データ送信&受信**

```
hr = tcpClient.SendAndReceive(lpszSendData,          // 送信データ
                              sizeof(lpszSendData),  // 送信データサイズ(0はNULL文字まで送信)
                              lpszReceiveData,       // 受信データの格納先アドレス
                              sizeof(lpszReceiveData),  // 受信データのバッファサイズ
                              &dwRecvedLen,          // 実際に受信したデータ数
                              dwRetryCnt);           // リトライ回数（省略時：1回）
```

(5) 接続をクローズする

データの送受信が終了したら，切断処理をおこなう必要があります．切断処理は Disconnect()メソッドを利用して，以下のようにおこないます．

```
// クライアントモードの場合
tcpClient.Disconnect();

// サーバモードの場合
pTcpClient->Disconnect();
delete pTcpClient
pTcpClient = NULL;

tcpServer.Disconnect();
```

#### 3.8.4.2. エラーコード

TCP ソケットクラスでは，固有のエラーコードとして Winsock のエラーを“0x8091000”でマスクした値を返します．

例）Winsock エラー：10061（接続拒否） → CAO API のエラー：0x8091274D

### 3.8.5. CUDPSocket クラス

#### 3.8.5.1. 利用方法

次に CDevice クラスを継承した，Ethernet のソケット通信(UDP)をおこなう CUDPSocket クラスの利用方法について説明します．CUDPSocket クラスは，CTCPSocket クラスと同じように Receive メソッドや Send メソッドを呼び出すことで，データの送受信をおこなうことができます．

(1) オブジェクトの生成

CUDPSocket クラスを利用するには，UDPSocket.h ファイルをインクルードする必要があります．このとき，2.3-(7)で示したように，インクルードディレクトリを設定している必要があります．

プロジェクトには以下の 3 つのファイルを既存のファイルとして追加する必要があります．

1. UDPSocket.cpp…CUDPSocket クラスの実装ファイル
2. Socket.cpp…CUDPSocket の基底クラスである CSocket クラスの実装ファイル
3. Device.cpp…CSocket の基底クラスである CDevice クラスの実装ファイル

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

【サーバモードの場合】

(2) 初期化

必要なパラメータの設定をおこない，CUDPSocket クラスを初期化します．Initialize 時の設定パラメータとしてヘッダコード，ターミネータコード，動作モード（内容は CSerial クラスと同じ）を指定します．

ただし，UDP のサーバモードで使用する場合，動作モードに ST_MODE_UDP_SERVER を OR 演算で追加してください．この記述を忘れると，データ受信ができません．

```
CHAR cHeader[] = "";            // ヘッダコード
CHAR cTerm[] = "¥r";            // ターミネータコード
DWORD dwMode = ST_MODE_TEXT | ST_MODE_UDP_SERVER;     // 動作モード
PVOID pArgs[] = { cHeader,cTerm,&dwMode };
HRESULT hr = udpSocket.Initialize(pArgs);
```

(3) ソケットをオープンする

ソケットのオープンをおこない，データ送受信の準備をおこないます．Connect()が成功すれば，このオブジェクトを利用してデータの送受信をおこなうことができます．UDP 通信で必要となるパラメータは PARAM_CONN_ETH 型の構造体に定義されており，ローカルの IP アドレスとポート番号を指定します．

Connect が成功したら，SetTimeout()でタイムアウト時間を設定します．

```
PARAM_CONN_ETH connParam;
connParam.dwSrcIP = INADDR_NONE;    // ローカルIPアドレス
connParam.dwSrcPort = 5006;         // ローカルポート番号

hr = updSocket.Connect(connParam);

hr = udpSocket.SetTimeout(1000);    // タイムアウトの設定[ms]
```

【クライアントモード】

(2) 初期化

必要なパラメータの設定をおこない，CUDPSocket クラスを初期化します．Initialize 時の設定パラメータとしてヘッダコード，ターミネータコード，動作モード（内容は CSerial クラスと同じ）を指定します．

```
CHAR cHeader[] = "";            // ヘッダコード
CHAR cTerm[] = "¥r";            // ターミネータコード
DWORD dwMode = ST_MODE_TEXT;    // 動作モード
PVOID pArgs[] = { cHeader,cTerm,&dwMode };
HRESULT hr = udpSocket.Initialize(pArgs);
```

(3) ソケットをオープンする

ソケットのオープンをおこない，データ送受信の準備をおこないます．Connect()が成功すれば，このオブジェクトを利用してデータの送受信をおこなうことができます．UDP 通信で必要となるパラメータは PARAM_CONN_ETH 型の構造体に定義されており，ローカルの IP アドレスとポート番号を指定します．ローカルポート番号を 0 に設定することで，空いている適当なポート番号が割り振られます．

Connect が成功したら，SetTimeout()でタイムアウト時間を設定します．

```
PARAM_CONN_ETH connParam;
connParam.dwSrcIP = INADDR_NONE;    // ローカルIPアドレス
```

```
connParam.dwSrcPort = 0;            // ローカルポート番号

hr = udpSocket.Connect(connParam);

hr = udpSocket.SetTimeout(1000);    // タイムアウトの設定[ms]
```

**【UDP ソケットクラスでのデータの送受信】**

(4) 送信先の設定(データ受信のみを使用する場合，この設定は不要です．)

SetDestination()で送信先の設定を行います．設定パラメータには，送信先の IP アドレスとポート番号を DWORD 型で指定します．

```
DWORD dwDist[2];
dwDist[0] = inet_addr("127.0.0.1");   // 接続先IPアドレス
dwDist[1] = 5006;                     // 接続先ポート番号
udpSocket.SetDestination((PVOID)dwDist);
```

(5) データの送受信をおこなう

ソケット通信でデータの送受信をおこなうには，以下に示すメソッドを利用します．

```
BYTE lpszSendData[]{ 0x41,0x42 };  // 送信データ(文字列："A,B")
BYTE lpszReceiveData[16];          // 受信データ
DWORD dwRecvedLen;                 // 受信データ数
DWORD dwRetryCnt = 3;              // リトライ回数
```

・ **データ送信**

```
hr = udpSocket.Send(lpszSendData,          // 送信データ
                    sizeof(lpszSendData),  // 送信データサイズ(0はNULL文字まで送信)
                                           // 省略した場合：0
                    true);                 // ヘッダとターミネータを付加オプション
                                           // true:付加 false:付加しない(省略時：true)
```

・ **データ受信**

```
hr = udpSocket.Receive(lpszReceiveData,        // 受信データの格納先アドレス
                       sizeof(lpszReceiveData),// 受信データのバッファサイズ
                       &dwRecvedLen);          // 実際に受信したデータ数
```

・ **データ送信&受信**

```
hr = udpSocket.SendAndReceive(lpszSendData,          // 送信データ
                              sizeof(lpszSendData),  // 送信データサイズ(0はNULL文字まで送信)
                              lpszReceiveData,       // 受信データの格納先アドレス
                              sizeof(lpszReceiveData),// 受信データのバッファサイズ
                              &dwRecvedLen,          // 実際に受信したデータ数
                              dwRetryCnt);           // リトライ回数（省略時：1回）
```

(6) 接続をクローズする

データの送受信が終了したら，切断処理をおこなう必要があります．切断処理は Disconnect メソッドを利用して，以下のようにおこないます．

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

#### 3.8.5.2. エラーコード

UDP ソケットクラスでは，固有のエラーコードとして Winsock のエラーを“0x8091000”でマスクした値を返します.

例）Winsock エラー：10048（指定アドレスが使用中） → CAO API のエラー：0x80912740

# 4. TCmini プロバイダの作成

本章では，芝浦機械製小型プログラマブルコントローラ TCmini を対象としたプロバイダ作成方法を紹介します．今回はTCminiのリレー，レジスタの読み書きで必要とされる関数(メソッド)を，第2章で説明した手順に従って実装します．

本書では，芝浦機械製小型プログラマブルコントローラ TCmini を『TCmini コントローラ』，今回作成するプロバイダを『TCmini プロバイダ』と呼びます．

なお，TCmini プロバイダを作成するにあたり，下記のものが必要となります．

- Microsoft Visual Studio 2019 Professional
- ORiN2 SDK Provider Development
- TCmini コントローラ
- C++言語および COM の基礎知識
- シリアル通信の基礎知識

## 4.1. TCmini コントローラとは

### 4.1.1. 構成

TCminiコントローラは，フォトカプラ入力 16 点，リレー出力 16 点，ディップスイッチ入力 8 点，温度入力(サーミスタ)4 点，アナログ入力(0-5V)2 点，アナログ出力(0-5V)2 点，カレンダー機能，RS-232C 通信機能，RS-485 通信機能，拡張機能を持つ小型プログラマブルコントローラです．

TCmini コントローラのリレーは 1 点＝1 ビット，レジスタは 16 点=16 ビットで構成されています．リレーおよびレジスタは基本的に『<機能区分記号>+<アドレス>』であらわします．<機能区分記号>は種別を意味し，‘X’,‘Y’,‘R’,‘T’,‘C’,‘L’,‘E’,‘A’,‘D’,‘V’,‘P’のアルファベット1文字を指定します．<アドレス>は 16 進数で表現される 3 文字の数値です．ただし，リレーの場合は 8 点ごとにまとめてバイトレジスタとして，あるいは 16 点毎にまとめてワードレジスタとしてデータを取り扱うことができます．その場合，<アドレス>の最後の文字(3文字目)の部分がレジスタタイプとなります．レジスタタイプには‘L’,‘H’,‘W’のアルファベット1文字が指定できます．

#### 4.1.1.1. データメモリの種類

**表 4-1 データメモリの種類**

| 種類 | 容量 | リレーアドレス | バイトレジスタ | ワードレジスタ |
|---|---|---|---|---|
| 入出力リレー | 256 点 | X/Y000～X/Y17F | X/Y00H/L～X/Y17H/L | X/Y00W～X/Y17W |
| 内部リレー | 256 点 | R000～R17F | R00H/L～R17H/L | R00W～R17W |
| タイマカウンタ | 96 点 | T/C000～T/C05F | T/C00H/L～T/C05H/L | T/C00W～T/C05W |
| ラッチリレー | 32 点 | L000～L01F | L00H/L～L01H/L | L00W～L01W |
| エッジリレー | 64 点 | E000～E03F | E00H/L～E03H/L | E00W～E03W |

| | | | | |
|---|---|---|---|---|
| 特殊補助リレー | 240 点 | A000～A16F | A00H/L～A16H/L | A00W～A16W |
| レジスタ | 208 ワード | | | D000～D05F<br>D120～D14F |
| レジスタ | 64 ワード | | | D060～D11F |
| 特殊レジスタ | 48 ワード | | | D150～D17F |
| タイマカウンタ設定値 | 96 ワード | | | V000～V05F |
| タイマカウンタ現在値 | 96 ワード | | | P000～P05F |

#### 4.1.1.2. データメモリの機能

**表 4-2 データメモリの機能**

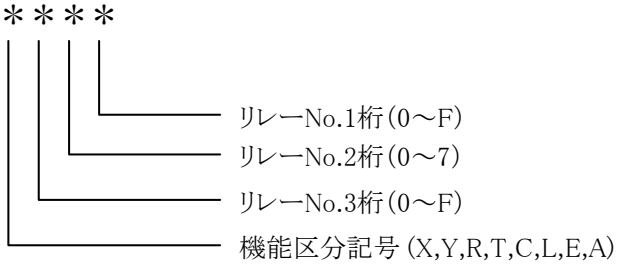
| 種類・使用状態 | | 区分 | 機　　能 |
|---|---|---|---|
| 入出力リレー | 入力リレー | X | 入力専用リレーであり，フォトカプラ・キーボード・DISPSW 等の入力が接続されています. |
| | 出力リレー | Y | 出力専用リレーであり，リレー・パネルLED等の出力機器が接続されています. |
| | 未認識リレー | Z | 毎スキャンサイクルの入出力処理からは除外されるため，内部リレーとして使用することができます. |
| 内部リレー | | R | 外部に出力する必要のない一時記憶として使用することができます. |
| タイマ | | T | 現在値が 0 になるとタイマ接点が ON になります. |
| カウンタ | | C | 現在値が 0 になるとカウンタ接点が ON になります. |
| ラッチリレー | | L | セット，リセット型のラッチリレーのためセットが OFF してもリセットが ON するまでラッチ接点は ON 状態を保持します. |
| エッジリレー | | E | エッジ接点として使用できます. |
| レジスタ | | D | ワード長(16 ビット)レジスタでバイトレジスタとしての指定はできません.<br>D060～D11F のデータに変更があると EEPROM に書き込みます. (保持可能)<br>D150～D17F は特殊データレジスタです. 特殊データレジスタのうち，使用していないレジスタは汎用データレジスタとして使用することができます. |
| タイマカウンタ<br>設定値 | | V | ワード長(16 ビット)レジスタでバイトレジスタとしての指定はできません.<br>タイマカウンタとして使用していない領域はデータレジスタとして使用することができます. |
| タイマカウンタ<br>現在値 | | P | ワード長(16 ビット)レジスタでバイトレジスタとしての指定はできません.<br>タイマカウンタの実行，現在値を読み出すことができます. (減算タイマ・減算カウンタ) |

| | | |
|---|---|---|
| 特殊補助リレー | A | 演算フラグ, スキャンタイム, クロックなどの CPU が書き込む領域です. したがってユーザプログラムでコイル, データレジスタのディスティネーションとしては使用できません.<br>接点, データレジスタのソースとしてユーザプログラムで使用できます. |

#### 4.1.1.3. リレーアドレス

リレーアドレスはリレーNo., 機能区分機能を付けて表現します.

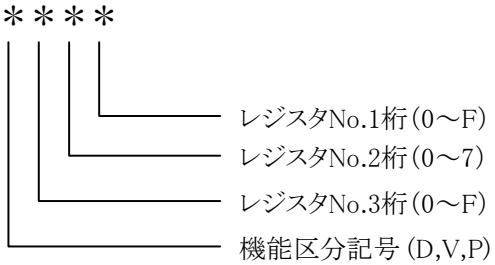
入出力リレーアドレスは実際のリレー装着と対応しますが, その他のリレーアドレスは物理的に存在しない装置(仮想装置)に対応しています. リレーアドレスは 1 点(1 ビット)ごとに付けられます.

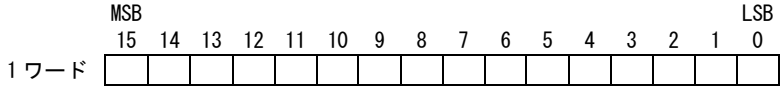



#### 4.1.1.4. データレジスタアドレス

データレジスタアドレスもリレーアドレスと同様の表現になります.

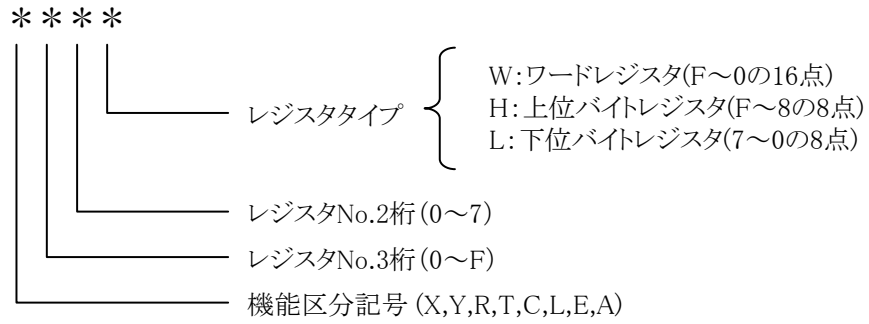
リレーアドレスが 1 点(1 ビット)単位なのに対してデータレジスタアドレスは 1 ワード(16 ビット)単位となります.



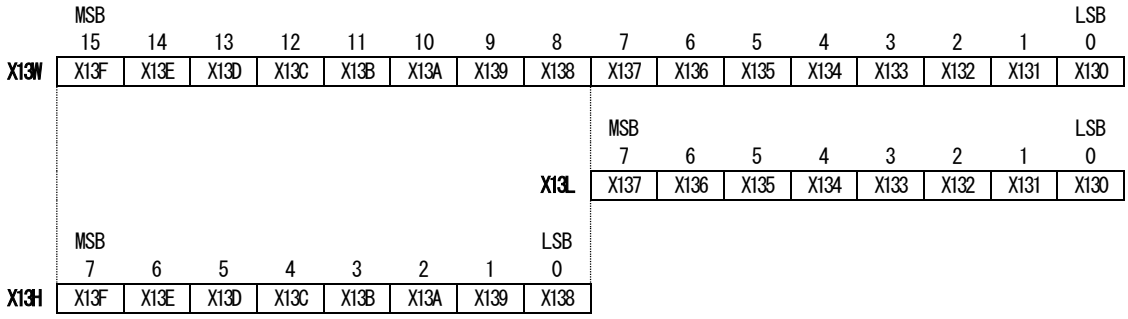




#### 4.1.1.5. リレー領域のバイト/ワードレジスタアドレス

リレー領域は 8 点ごとにまとめてバイトレジスタとして, 16 点毎にまとめてワードレジスタとしてデータを取り扱うことができます. アドレスはリレーアドレスの「リレーNo.1 桁」の部分がレジスタタイプとなります.




例） X130～X13F をワードレジスタ, バイトレジスタで指定した場合の対応




### 4.1.2. 接続

TCmini コントローラからの各種サービスは RS-232C 経由で行われるため, 本体モジュール表面の RS-232C コネクタと PC(ホストコンピュータ)のシリアル(COM)ポートをストレートケーブルで接続してください. 使用するケーブルはクロスケーブルではありませんのでご注意ください.

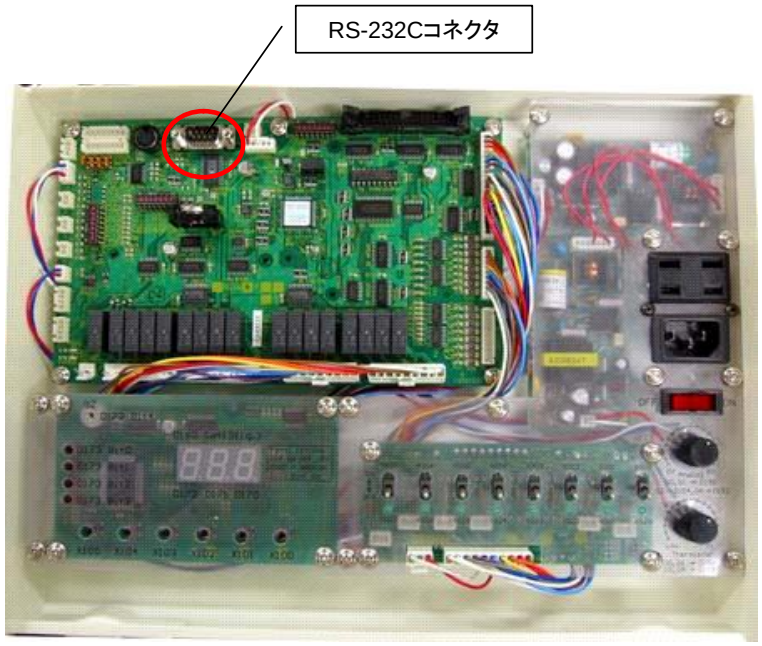



**図 4-1 TCmini コントローラ**

PC 側は TCmini コントローラと交信するために，COM ポートを下記の仕様に設定する必要があります．

| 通信パラメータ | 設定値 |
|---|---|
| ボーレート[bps] | 9600/19200/38400 |
| データビット[bits] | 8 |
| パリティビット | なし |
| ストップビット[bits] | 2 |
| フロー制御 | なし |

TCmini 側の設定は基本的に不要です．TCmini コントローラは 9600/19200/38400bps の範囲であれば自動的にボーレートを判定します．デフォルトでは 19200bps の設定になっています．

### 4.1.3. 通信プロトコル概要

PC(ホストコンピュータ側)と TCmini コントローラ(TCCUH*及び TCCM*側)が交信する場合，必ず PC 側からのデータアクセスにより開始します．下記にその図を示します．

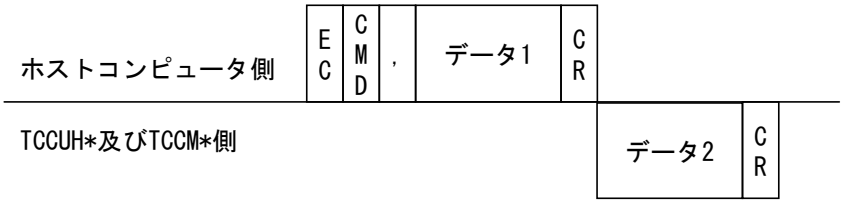



**図 4-2 基本フォーマット**

**表 4-3 通信プロトコル固定データ**

| 記号 | 内容 |
|---|---|
| EC | エスケープを意味し，ASCII コードは 0x1B である. |
| CMD | コマンド番号で，“0”～“32”の数字で ASCII コードの 0x30(=‘0’)から 0x39(=‘9’)を使用する. |
| , | コマンド番号とデータを判別するための記号(カンマ)である. ASCII コードは 0x2C である. |
| CR | キャリッジリターンを意味し，ASCII コードは 0x0D である. |
| データ 1 | コマンドの補足データである. |
| データ 2 | コマンドに対応する返信データである. |

PC 側から TCmini コントローラに対して上記のフォーマットでコマンド(CMD)とデータ(データ 1)が送信されると TCmini コントローラは送信された内容を解釈し，要求された処理を実行してその応答データ(データ 2)を返します. コントローラに送信されたコマンドが正しくない場合は応答データとして“ﾊﾟﾗﾒｰﾀｴﾗｰ”等のエラーメッセージが返ります.

交信はすべて ASCII コードで行われます.

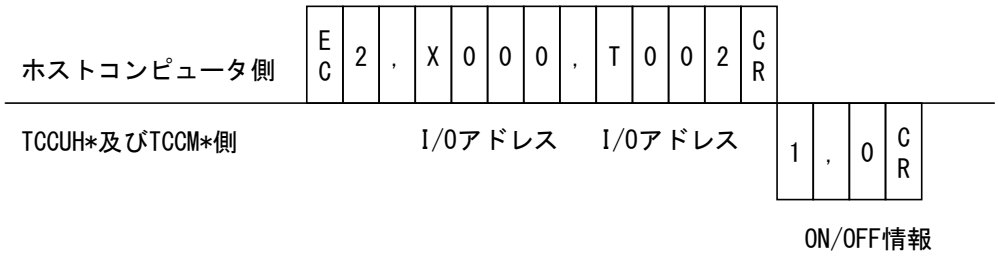
本節以降ではこの通信フォーマットの一部のコマンド仕様に関してのみ記載します. その他のコマンド仕様に関しては TCmini コントローラ付属の『RS232C ホスト通信コマンド説明書』に詳細が記載されているので必要に応じて参照してください.

#### 4.1.3.1. I/O 読出し 1 点単位

最大 42 点までの接点の ON/OFF 情報を読み出します.

<CMD> “2” (0x32)

下記例では X000(ON)，T002(OFF)を示します.




ON/OFF 情報

0 ： OFF

1 ： ON

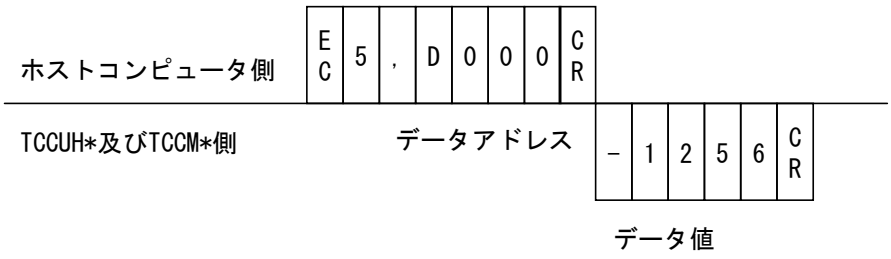
I/O アドレス(最大 42 接点)

- 必ず 4 桁で指定してください
- 区分コード(X,Y,R,L,S,T,C,E,A,G,H)は大文字で指定してください
- 存在しないアドレスを指定すると，“ﾊﾟﾗﾒｰﾀｴﾗｰ”のメッセージが返ります

#### 4.1.3.2. データ読出し 1 語単位

最大 32 ワードのデータを同時に読み出します.

<CMD>“5” (0x35)




データアドレス

- 必ず 4 桁で指定してください
- 区分コード(X,Y,R,L,S,T,C,E,A,G,H,D,P,V,B)とワード指定 W, バイト指定 H,L は大文字で指定してください
- 存在しないアドレスを指定すると，“ﾊﾟﾗﾒｰﾀｴﾗｰ”のメッセージが返ります
- アドレスの指定でワードと，バイトを混在させないでください

データ値

- データは BIN データを 10 進数として返します
- ワードデータは-32768～32767 の範囲となります
- バイトデータは 0～255 の範囲となります

#### 4.1.3.3. I/O 強制セット

最大 42 個のリレー接点を同時に ON(強制セット)します.

<CMD>“8” (0x38)


I/O アドレス(最大 42 接点)

- 必ず 4 桁で指定してください
- 区分コード(X,Y,R,L,S,T,C,E,A,G,H)は大文字で指定してください
- 存在しないアドレスを指定すると，“ﾊﾟﾗﾒｰﾀｴﾗｰ”のメッセージが返ります
- シーケンスプログラムでリセットしているエリアをセットすることはできません
- タイマ・カウンタエリアをセットすると現在値を 0 として，接点を ON します

#### 4.1.3.4. I/O 強制リセット

最大 42 個のリレー接点を同時に OFF(強制リセット)します.

<CMD>“9” (0x39)

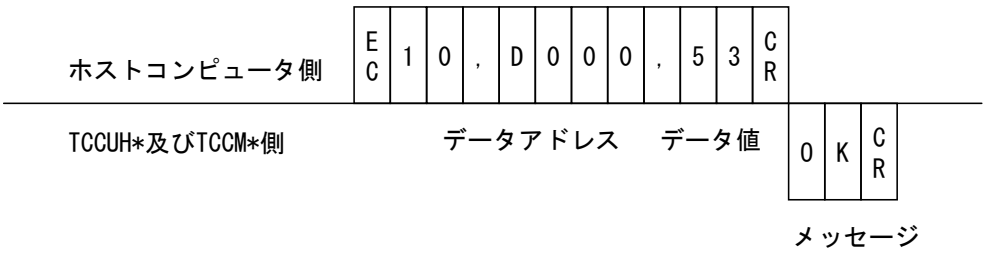

I/O アドレス(最大 42 接点)

- 必ず 4 桁で指定してください
- 区分コード(X,Y,R,L,S,T,C,E,A,G,H)は大文字で指定してください
- 存在しないアドレスを指定すると，“ﾊﾟﾗﾒｰﾀｴﾗｰ”のメッセージが返ります
- シーケンスプログラムでセットしているエリアをリセットすることはできません
- タイマ・カウンタエリアをリセットすると現在値は設定値と等しくなり接点を OFF します

#### 4.1.3.5. データ変更 1 語単位

最大 32 個のデータを同時に変更できます.

<CMD>“10” (0x31,0x30)




データアドレス

- 必ず 4 桁で指定してください
- 区分コード(X,Y,R,L,S,T,C,E,A,G,H,D,P,V,B)とワード指定 W, バイト指定 H,L は大文字で指定してください
- 存在しないアドレスを指定すると, “ﾊﾟﾗﾒｰﾀｴﾗｰ”のメッセージが返ります
- アドレスの指定でワードと, バイトを混在させないでください

データ値

- データは BIN データを 10 進数として返します
- ワードデータは-32768～32767 の範囲となります
- バイトデータは 0～255 の範囲となります

### 4.2. TCmini プロバイダ仕様

TCmini プロバイダを作成するにあたり最初にプロバイダとしての振る舞い, つまり動作仕様を決める必要があります. 図 4-3 にプロバイダの各クラスと TCmini の対応を示します. CaoProvController は TCmini 本体に, CaoProvVariable は TCmini のリレーアドレスおよびデータレジスタアドレスに対応する仕様とします. この仕様に従ってプロバイダを実装します.

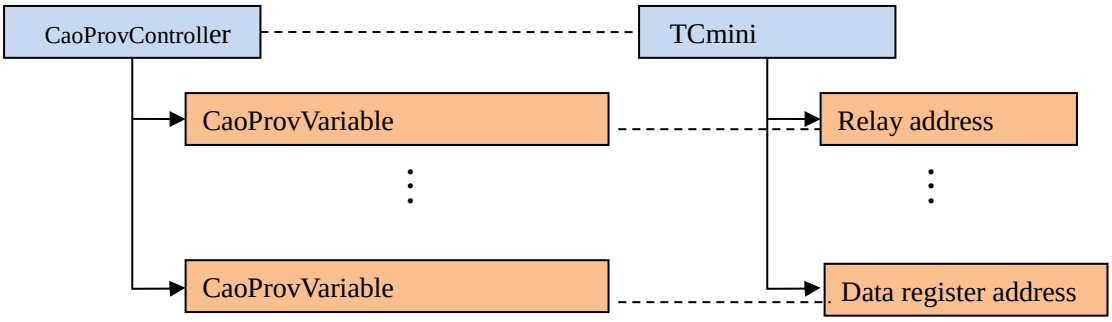


**図 4-3 プロバイダ各クラスと TCmini の対応**

### 4.2.1. AddController 時のオプション仕様

CAOWorkspace::AddController 時に 4 つの引数が必要です. ここでは, TCmini プロバイダを使用するときに, 第 4 引数に渡すオプション文字列の仕様を決定します.

書式

```
AddController
(
    “<Controller 名>",    // コントローラ名(任意)
    "<ProvID>",           // プロバイダ名
    "<マシン名>",         // プロバイダの実行マシン名
    "<オプション>"        // オプション文字列
)
```

TCmini と接続するためには, ユーザは何番ポートで通信するのか, ボーレートは何 bps にするのか, という接続パラメータを決定する必要があります. また, 送受信時のタイムアウト時間をユーザが設定できるようにすると便利です.

以上を踏まえて, TCmini のオプション文字列の仕様を表 4-4 に示します. 通信用の接続パラメータを意味する”Conn”と, 送受信時のタイムアウト時間を意味する”Timeout”をオプション文字列として使用することにします. 各オプション文字列(Conn または Timeout)のあとに「=」を付与し, それぞれの記述方法に基づいて接続パラメータとタイムアウト時間を指定します. また, オプション文字列として Conn は必須としますが, Timeout は任意のオプション文字列とします. 2 つを同時に指定する場合, 各オプション文字列との間に「, 」を付与することで記述できます.

**表 4-4 オプション文字列の仕様**

| 項目 | オプション文字列 | 必須 | 記述方法 |
|---|---|---|---|
| 接続パラメータ | Conn | ○ | "Conn=com:<COMPort>[:<BaudRate>[:<Parity>:<DataBits>:<StopBits>[:<Flow>]]]"<br><br><COM Port>: COM ポート番号.<br>‘1’-COM1, ‘2’-COM2, …<br><BaudRate> : 通信速度. |

| | | | |
|---|---|---|---|
| | | | 4800, 9600, 19200, 38400, 57600, 115200.<br><Parity> : パリティ.<br>'N'-NONE, 'E'-EVEN, 'O'-ODD<br><DataBits> : データビット数.<br>'7'-7bit, '8'-8bit.<br><StopBits> : ストップビット数.<br>'1'-1bit, '2'-2bit.<br><Flow> : フロー制御.<br>'0'-フロー制御なし, '1'-Xon/Xoff, '2'-ハードウェア制御.<br>(OR 演算指定可能)<br>*下線部はオプションを指定しなかったときのデフォルト値. |
| タイムアウト時間 | Timeout | - | "Timeout=<Time>"<br><br><Time>: タイムアウト時間.<br>0~4294967295 ms で指定.(デフォルト:500 ms) |

例1) COM ポート番号:1, 通信速度:9600 bps, パリティ:E, データビット数:7 bit, ストップビット:1 bit, フロー制御:Xon/Xoff, タイムアウト:1000 ms を指定する場合.
(記述方法) → "Conn=com:1:9600:E:7:1:1,Timeout=1000"

例2) COM ポート番号:1, 通信速度:19200 bps, パリティ:なし, データビット数:8 bit, ストップビット:1 bit, フロー制御:なし, タイムアウト:500 ms を指定する場合.
(記述方法) → "Conn=com:1" (デフォルト値は省略可能)

使用例(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 の変数名とオプション仕様

CaoController::AddVariable 時には, 変数名とオプション文字列の 2 つの引数が必要です. 変数名には, システム変数とユーザ変数の 2 種類が存在します. システム変数は固定文字列, ユーザ変数は任意文字列となります. システム変数には「@」を使用するのが必須で, get_VariableNames プロパティを実行することでシ

ステム変数一覧を取得できます. ここでは, TCmini プロバイダを使用するときのシステム変数とユーザ変数の仕様を決定します.

書式

```
AddVariable
(
    “<変数名>“          // 変数名
    “<オプション>“      // オプション文字列
)
```

#### 4.2.2.1. システム変数仕様

システム変数の仕様を表 4-5 に示します. 全てのシステム変数でオプション文字列は使わない仕様とします. get_Value プロパティのみ可能とし, put_Value プロパティは実装しません. また, @CURRENT_TIME と@ERROR_DESCRIPTION の get_Value プロパティは TCmini と通信してデータを取得しますが, それ以外のシステム変数は TCmini とは通信しません.

**表 4-5 TCmini プロバイダで実装するシステム変数仕様**

| 変数名 | オプション文字列 | データ型 | 説明 | 属性 | |
|---|---|---|---|---|---|
| | | | | get | put |
| @CURRENT_TIME | なし | VT_DATE | TCmini 保有の現在時刻 | ○ | × |
| @ERROR_DESCRIPTION | なし | VT_BSTR | TCmini 保有のアラームメッセージ | ○ | × |
| @MAKER_NAME | なし | VT_BSTR | メーカ名. “Shibaura Machine” | ○ | × |
| @TYPE | なし | VT_BSTR | 機器名. “TCmini” | ○ | × |
| @VERSION | なし | VT_BSTR | プロバイダのバージョン | ○ | × |

使用例(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. ユーザ変数仕様

任意のデータメモリにアクセスする際は，ユーザ変数を使用してアクセスする仕様とします．ユーザ変数の仕様を表 4-6 に示します．

今回の仕様では，任意のデータメモリ(リレーアドレス，データレジスタアドレス，リレー領域のバイト/ワードレジスタアドレス)を変数名として指定可能とします．オプション文字列は使用せず，データ型は VT_I2 型とし，変数名で指定したデータメモリへの読み込み(get_Value プロパティ)，書き込み(put_Value プロパティ)動作を可能とします．ただし，変数名に存在しないデータメモリ(“X999”など)を指定して get/put を実行すると，VT_BSTR で TCmini が返す”ﾊﾟﾗﾒｰﾀｴﾗｰ”の文字列を表示します．

また，変数名として任意の通信コマンドも指定可能とします．オプション文字列は使用せず，データは VT_BSTR 型とします．また，get_Value 実行時に TCmini のプロトコルに従い，<ESC>+<変数名>+<CR>の電文を送信します．put_Value には対応していません．このように，任意の通信コマンドを使用可能にすることで，”2,X100,X200”(X100,X200 を同時に取得するコマンド)などをユーザが設定できるようになります．

**表 4-6 TCmini プロバイダで実装するユーザ変数仕様**

| 変数名 | オプション文字列 | データ型 | 説明 | 属性 | |
|---|---|---|---|---|---|
| | | | | get | put |
| データメモリ<br>ex) “X100” | なし | VT_I2 | 任意のデータメモリにアクセス．<br>*存在しないデータメモリを指定して get/put を実行すると， VT_BSTR で”ﾊﾟﾗﾒｰﾀｴﾗｰ”の文字列を返します．<br>例)”X999”など | ○ | ○ |
| 通信コマンド<br>ex) “2,X100,X200” | なし | VT_BSTR | 任意の通信コマンドを送信．<br>*不正な通信コマンドを指定して get を実行すると，”ﾊﾟﾗﾒｰﾀｴﾗｰ”の文字列を返します． | ○ | - |

使用例(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);          // X100がOFFの場合:0
                                         // X100がONの場合:1
Debug.WriteLine(varX10W.Value);          // X100~X10Fが全てOFFの場合:-1
                                         // X100~X10Fが全てONの場合:0
Debug.WriteLine(varD150.Value);          // -32768~32767
Debug.WriteLine(varX100X200.Value);      // X100,X200 が OFF の場合:  0,0
```

## 4.3. TCmini プロバイダの実装

### 4.3.1. TCmini プロバイダプロジェクトの作成

TCmini プロバイダ用のプロジェクトは，CaoProvWiz.exe を用いて簡単に作成することができます．「2.3 新規プロバイダプロジェクトの作成」の手順でまず TCmini プロバイダプロジェクトを作成してください．

この時点で TCmini プロバイダ用の VC++プロジェクトが CaoProv.vcxproj として作成されるのでこれを指定して VC++を起動します．

作成されたプロジェクトフォルダには CaoProv.vcxproj 以外にも複数のファイルが存在します．下表に，今回実装するファイルの一覧を示します．

**表 4-7 編集するファイル一覧**

| No. | 名前 | 種類 |
|---|---|---|
| 1 | CaoProvController.h | C ヘッダファイル CCaoProvController クラスの定義 |
| 2 | CaoProvController.cpp | C++ソースファイル CCaoProvController クラスの実装 |
| 3 | CaoProvVariable.h | C ヘッダファイル CCaoProvVariable クラスの定義 |
| 4 | CaoProvVariable.cpp | C++ソースファイル CCaoProvVariable クラスの実装 |
| 5 | StdAfx.h | C ヘッダファイル |

すでに実装済みのサンプルコードは以下のフォルダに格納していますので，必要に応じてご覧ください．

<インストールフォルダ>¥CAO¥ProviderLib¥ToshibaMachine¥TCmini¥Src

### 4.3.2. CSerial クラスの追加

TCmini プロバイダから TCmini コントローラへの交信はシリアルポート経由でおこなわれます．そこで，前章で説明した通信クラスである CSerial クラスを利用します．

まずは，“Device.cpp”，“Serial.cpp”の 2 つのファイルをプロジェクトフォルダに追加します．図 4-4 に示すように，CAOPROV プロジェクトを右クリックし，[追加(D)]→[既存の項目(G)]を選択し，<インストールフォルダ>¥ORiN2¥CAO¥Include 内の“Device.cpp”と“Serial.cpp”を追加してください．

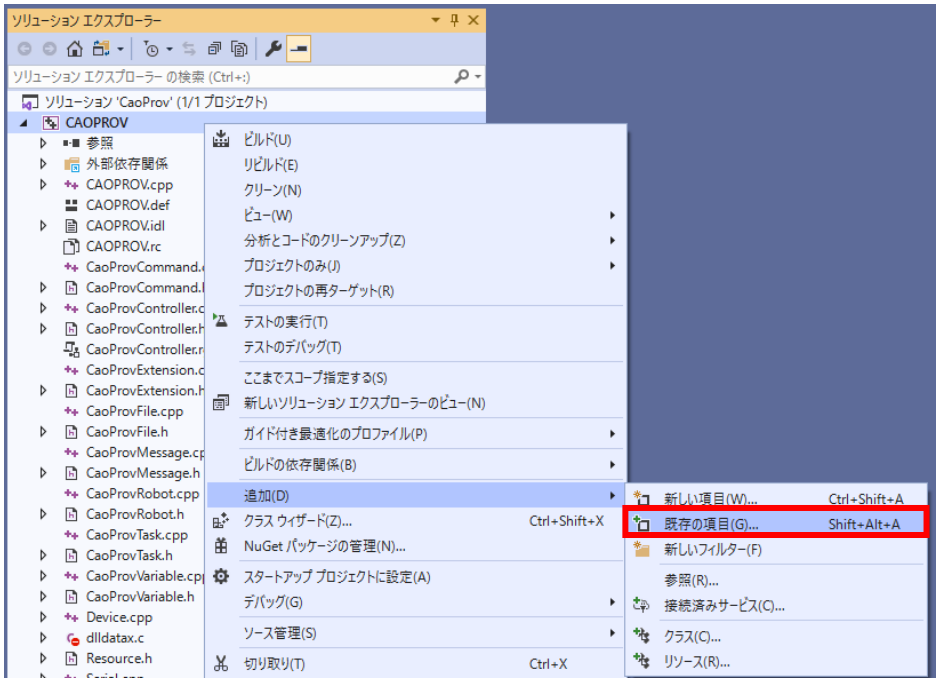



**図 4-4 ファイルをプロジェクトへ追加画面**

次に，TCmini プロバイダが CSerial クラスを利用できるように，ヘッダファイルをインクルードします．StdAfx.h ファイルに以下の記述を追加してください．黒の網掛け部分が追加したコードを示します．

**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. CCaoProvController クラスの実装

#### 4.3.3.1. 必要なメソッドのオーバーライド

CCaoProvController クラスの実装準備としてまず必要なメソッドのオーバーライドを以下の手順でおこないます．

(1) CCaoProvController クラスのヘッダファイル(CaoProvController.h)に記述してあるメソッドの内，以下のメソッドを有効に(コメントアウトを解除)します．

1. HRESULT FinalInitialize()
2. HRESULT **FinalConnect**()
3. HRESULT FinalDisconnect()

4. void FinalTerminate()
5. HRESULT FinalGetVariableNames()

以下に，CCaoProvController::**FinalInitialize**()の場合の例を示します．

```
//HRESULT FinalInitialize();        //コメントをはずす前
                ↓
  HRESULT FinalInitialize();        //コメントをはずした後
```

(2) 上記のメソッドの実装が行われている箇所(CaoProvController.cpp)のコメントをはずします．
以下に，CCaoProvController::**FinalInitialize**()の場合の例を示します．

```
/*                          //コメントをはずす前
HRESULT CCaoProvController::FinalInitialize()
{
  // 省略
}
*/
    ↓
HRESULT CCaoProvController::FinalInitialize()    //コメントをはずした後
{
  // 省略
}
```

#### 4.3.3.2. 必要なメンバの追加

次に CCaoProvController クラスに必要なメンバ変数の追加をおこないます．

以下に示す，COM ポートと通信するオブジェクトへのポインタを保持する変数をメンバ変数として追加します．

m_pSerial : COM ポートと通信をおこなうオブジェクトへのポインタ

**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. FinalInitialize()の実装

次に CaoProvController クラスの FinalInitialize メソッドを実装します．このメソッドは，CaoWorkspace::AddController 実行時に呼び出されます．この関数は必ず実装してください．

このメソッドは，CaoProvController オブジェクトが生成されたときに呼び出されます．このメソッド内では，デバイスオブジェクトへのポインタを初期化します．また，メソッドを実装した場合，返り値を E_NOTIMPL から S_OK に変更してください．

**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. ParseConnectionString()の実装

続いて CaoWorkspace::AddController 時に渡される通信用の接続パラメータを解析するため，CaoProvController.h/cpp 内の ParseConnectionString 関数を編集します.

今回，ParseConnectionString 関数は COM 通信の接続パラメータとタイムアウト[ms]を解析する関数として実装します. COM 通信の接続パラメータを格納するための PARAM_CONN 型 comParam と，タイムアウト時間を格納するための DWORD 型 dwTimeout を引数にします. この関数は FinalConnect 関数内で使用します.

ParseConnectionString のプロトタイプ宣言を CaoProvController.h に定義します.

**List 4-4 CaoProvController.h**

```
HRESULT ParseConnectionString();        // 変更前
↓
HRESULT ParseConnectionString(PARAM_CONN* comParam, DWORD* dwTimeout);      // 変更後
```

ParseConnectionString の実装を CaoProvController.cpp に記述します. AddController の第 4 引数(オプション文字列)は，CaoProvController.cpp 内で m_bstrOption として文字列を保持しています. この関数では

m_bstrOption に接続パラメータとタイムアウト時間が入っているかを解析します.

接続パラメータの解析にはCConnectOptionクラスのGetConnectOption関数, タイムアウト時間の解析にはCOptionValueクラスのGetOptionValue関数を使用しています.

書式

```
AddController
(
    “<Controller 名>",          // コントローラ名(任意)
    "<ProvID>",           // プロバイダ名
    "<マシン名>",          // プロバイダの実行マシン名
    "<オプション>"         // オプション文字列
)
```

**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. FinalConnect()の実装

次は，FinalConnect()メソッドの実装を行います． FinalConnect では接続処理を行います．このメソッドは，CaoWorkspace::AddController 実行時に FinalInitialize()が終了したあとに呼び出されます．この関数は必ず実装してください．

本関数では，前節の ParseConnectionString 関数を用いてオプション文字列(AddController メソッドの第 4 引数)を解析しています．オプション文字列の解析が正常に終了した場合，メンバ変数 m_pSerial を new 演算子で生成し，初期化処理(Initialize 関数)，接続処理(Connect 関数)，タイムアウト時間の設定を順に行っています．この関数を実装した場合，返り値を E_NOTIMPL から S_OK に変更してください．

**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. FinalDisconnect()の実装

続いて，FinalDisconnect()メソッドの実装をおこないます．FinalDisconnect では切断処理を行います．このメソッドは，CaoControllers::Remove 実行時に最初に呼び出されます．この関数は必ず実装してください．

以下のように，FinalDisconnect 内でシリアル接続の切断処理とオブジェクトの開放処理を行っています．この関数を実装した場合，返り値を E_NOTIMPL から S_OK に変更してください．

**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. FinalTerminate()の実装

FinalTerminate()メソッドの実装をおこないます．この関数は CaoControllers::Remove 実行時に FinalDisconnect ()が終了したあとに呼び出され，オブジェクトの開放前処理を行います．この関数は必ず実装してください．

このメソッドでは初期化で生成した変数(イベントなど)をクローズする処理を行いますが，TCmini プロバイダではイベントなどの変数を使用していないため，特別に追加する実装はありません． 今回は関数のオーバーロードのみ実施してください．

**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. GetSerial()の実装

COM ポートへの接続は CCaoProvController クラス内でおこなっているので，CaoProvVariable オブジェクトから TCmini と通信するためにはシリアル接続オブジェクト(m_pSerial)へのポインタが必要になります．そこで，CaoVariable クラスから COM 通信が行えるように，シリアル接続オブジェクトを取得するメソッド GetSerial 関数を追加します．

GetSerial 関数は CCaoProvController のパブリックなメンバ関数として，CaoProvController.h に定義します．

**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);
```

GetSerial 関数の実装を CaoProvController.cpp に記述します．

**List 4-10 CaoProvController.cpp – GetSerial()**

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

#### 4.3.3.9. FinalGetVariableNames()の実装

TCmini プロバイダでは，表 4-5 に示した 5 つのシステム変数を取得できるようにします．システム変数の「@MAKER_NAME」と「@VERSION」はデフォルトのテンプレートで既に作成済みであるため，それ以外の 3 つのシステム変数の実装をします．

文字列の前の‘**L**’は VC++独自拡張で，次に続く文字列がワイド文字列(UNICODE)で扱われるように指

示するためのものです.

**List 4-11 StdAfx.h – マクロ定義**

```
// 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"
```

定義したシステム変数名の一覧を取得するためのメソッド FinalGetVaribaleNames()を以下のように実装します. システム変数の数が 5 つのため, nDefines の値を「nDefines=5」に変更してください.

**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. CCaoProvVariable クラスの実装

#### 4.3.4.1. 必要なメソッドのオーバーライド

CCaoProvVariable クラスの実装準備として，まず必要な FinalInitialize メソッド，FinalGetValue メソッド，FinalPutValue メソッドのオーバーライドを以下の手順でおこないます．

(1) CCaoProvVariable クラスのヘッダファイル(CaoProvVariable.h)に記述してある以下のメソッドのコメントアウトを解除します．

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

(2) 上記メソッドの実装がおこなわれている箇所(CaoProvVariable.cpp)のコメントをはずします．

#### 4.3.4.2. 必要な定数/メンバの追加

CaoProvVariable.cpp で使用する定数を CaoProvVariable.h で定義しておきます.

| 定数名 | 値 | 説明 |
|---|---|---|
| CMD_EC | 0x1B | コマンドの先頭コード |
| CMD_CR | 0x0D | コマンドの終了コード |
| USER_VARIABLE_LENGTH | 4 | ユーザ変数名の長さ |
| VAR_COMMAND_MAX | 256 | コマンドの最大文字数 |

CaoProvVariable オブジェクトが作成されたときに, クライアントから渡されるユーザ変数名を予め解析して数値化しておくとその後の処理が簡略化できます. ユーザ変数名を解析するための関数として以下のメンバ関数を定義します.

```
HRESULT ParseUserVariableName()
```

また, ユーザ変数名を解析した結果をクラス内で保持するため, 以下の列挙体およびメンバ変数を追加します.

・リレー(BIT), レジスタ(WORD), あるいはその他であるかのデータタイプ情報(列挙体)

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

・データタイプ情報を保持するメンバ変数

```
VAR_DATA_TYPE m_usrDataType
```

・大文字に変換した変数名のコピー

オリジナルのユーザ変数名(m_bstrName)は大小文字が混在している可能性があるため, すべて大文字に変換した変数名を定義します.

```
CComBSTR m_bstrUpperName
```

また, CaoProvVariable クラスからシリアル通信が行えるように, シリアル通信オブジェクトへのポインタを CCaoProvVariable クラスのメンバ変数として追加します.

```
CSerial *m_pSerial
```

以上の定義を CaoProvVariable.h に記述します.

**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. ParseUserVariableName()の実装

定義したメンバ関数の実装をCaoProvVariable.cppに記述します. CCaoProvController::AddVariable()の第一引数(変数名)はメンバ変数 m_bstrName に格納されており, m_bstrName がユーザ変数名の場合, ParseUserVariableName 関数で文字列を解析します.

書式

```
AddVariable
(
    “<変数名>“          // 変数名
    “<オプション>“      // オプション文字列(未使用)
)
```

まず, 変数名を大文字に変換したものを m_bstrUpperName にコピーします. 続いて得た変数名の先頭 1 文字目を参照して, それがリレータイプ(X,Y,Z,R,E,L,T,C,A)かレジスタタイプ(D,P,V)かあるいはその他のタイ

プかで処理を分岐します. レジスタタイプならm_usrDataType = VAR_TYPE_WORDとしています. リレータイプならさらに最後の文字(4文字目)を参照して‘L’,‘H’,‘W’ならリレー領域のバイト/ワードアドレス指定であるので VAR_TYPE_WORD として, それ以外は VAR_TYPE_BIT としています. その他のタイプは VAR_TYPE_UNKNOWN としています.

**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,X000,Y007,T002"
            m_usrDataType = VAR_TYPE_UNKNOWN;
            break;
        }
        return S_OK;
    }
```

#### 4.3.4.4. InitMapTable()の実装

デフォルトで定義されているメンバ関数 InitMapTable を実装します. InitMapTable はメンバ変数 m_cs_map にシステム変数名を登録します. m_cs_map はシステム変数を格納する変数名リストであり, AddVariable 実行時に渡される変数名が登録したシステム変数と一致しているかチェックするために使用します.

下記の記述をすることで, システム変数名が@MAKER_NAME, @VERSION, @CURRENT_TIME, @ERROR_DESCRIPTION, @TYPE 以外の文字列の場合, AddVariable が失敗します.

**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. FinalInitialize()の実装

FinalInitialize()メソッドの実装を行います. このメソッドは CCaoProvController::AddVariable()が実行されたときに最初に呼び出されます. FinalInitialize()では CCaoProvVariable クラスで定義したメンバ変数の初期化します. また, 親オブジェクト m_ulParentType を判別し, 変数名の解析を行います. デフォルトのテンプレートでは, 親オブジェクトが CCaoProvController クラスの場合に変数名の解析を行うプログラムを記述しています.

変数名がシステム変数(最初の文字列が「@」)の場合, m_bSystem は true となります. m_cs_map で登録した変数名と比較し, 一致していれば m_lUSysId にシステム変数の識別番号を代入します. m_lUSysId の識別番号は, システム変数を GET/PUT する際に使用します.

TCmini プロバイダでは, Controller クラスのみで Variable クラスを作成するため, デフォルトのテンプレートをそのまま使用します. ユーザ変数の解析には定義した ParseUserVariableName 関数を用います. 変数名が適正なシステム変数またはユーザ変数の場合, hr = S_OK となります. その場合, CCaoVariable クラスからも通信コマンドを実行できるように, 接続済みのシリアル通信オブジェクトのアドレスをメンバ変数 m_pSerial に代入しています. 以後, m_pSerial のメソッドを実行することで, TCmini と通信することができます.

**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. FinalGetValue()の実装

次に FinalGetValue()メソッドの実装をおこないます. CaoVariable::get_Value を実行した場合, この関数が実行されます. デフォルトのテンプレートでは, 親オブジェクトが CCaoController クラスの場合の実装をしています. システム変数は FinalGetCtrlSysValue 関数, ユーザ変数は FinalGetCtrlUserValue 関数を実行します.

今回, TCmini プロバイダでは, 親オブジェクトが CCaoController クラスの場合のみを想定しているので, デフォルトのテンプレートを変更せずに使用し, FinalGetCtrlSysValue 関数, FinalGetCtrlUserValue 関数のみを変更していきます.

**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. FinalGetCtrlSysValue()の実装

FinalGetCtrlSysValue 関数で使用するマクロ定義を StdAfx.h に記述します. まず, システム変数のマクロ定義を以下のように StdAfx.h に記述します.「@MAKER_NAME」,「@TYPE」の 2 つのシステム変数は, 常に定数を返すので, 返答結果を StdAfx.h にマクロ定義しています.

**List 4-18 StdAfx.h – マクロ定義**

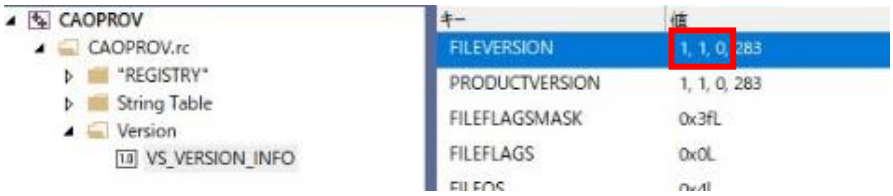
```
// ========== Additional functions are writed below. ==========
// ========== 各社追加部分はこれ以下に記述する. ==========

// 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 プロバイダでは, 5 種類のシステム変数を取得できるようにします. そこで, FinalGetCtrlSysValue 関数では以下に示すように, システム変数ごとに処理を追加します. 戻り値 pVal に値を入れることで, GetValue

の実行結果がユーザに通知されます.

1. "@MAKER_NAME"への対応
   switch-case 構文の CS_MAKER_NAME に対応しているので, 仕様の通り, "Shibaura Machine"を BSTR 型で返すように処理を追加します.

2. "@VERSION"への対応
   switch-case 構文の CS_VERSION に対応しており, GetDllVersion 関数を実行します. この関数は CAOPROV.rc の FILEVERSION に記載された 3 桁の値(メジャー番号.マイナー番号.リビジョン番号)を pVal に代入します.




3. "@CURRENT_TIME"への対応
   switch-case 構文の CS_CURRENT_TIME に対応して, TCmini コントローラ内のカレンダーを参照し, 現在時刻を返すように処理を追加します.
   TCmini コントローラでは D120～D126 レジスタを参照すれば現在時刻が取得できるようになっています. ただし, A009 = 1 がセットされていないと時刻情報は正しく取得できないのでデバッグ時に注意が必要となります.
   D120～D126 レジスタの情報は＜CMD＞“5”(0x35)コマンドを使用して一括読み出しをおこないます. このコマンドで帰ってきた現在時刻を VT_DATA 型で返します.

4. "@ERROR_DESCRIPTION"への対応
   switch-case 構文の CS_ERROR_DESCRIPTION に対応して, TCmini コントローラ内のエラー情報を返すように処理を追加します.
   コントローラのエラー情報は＜CMD＞“1”(0x31)コマンドを使用して取得できます. このコマンドで返って来た文字列を BSTR 型で返します.

5. "@TYPE"への対応
   switch-case 構文の CS_TYPE に対応しているので, 仕様の通り, "TCmini"を BSTR 型で返すように処理を追加します.

string 型の変数 szCmdBuf を使用するために, ”using namespace std;”をファイルの先頭に記述する必要があります.

また, CSerial クラスの初期化時に送受信データに”¥r”(<CR>)を付加して送受信するように初期化設定を

行っているため, szCmdBuf の最後尾に<CR>の追加は不要です.

**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. FinalGetCtrlUserValue()の実装

FinalGetCtrlUserValue 関数の実装を以下に示します. szCmdBuf を送信バッファとして定義し, TCmini のプロトコルに従い, コマンド用文字列の先頭に<EC>コードを付加します.

データタイプが VAR_TYPE_BIT なら I/O 読出し 1 点単位を実行するため「"2," + "変数名"」を, VAR_TYPE_WORD ならデータ読出し 1 語単位を実行するため「"5," + "変数名"」を送信バッファに格納します. データタイプが VAR_TYPE_UNKNOWN の場合, そのまま変数名を送信バッファにコピーします.

実際に TCmini と通信する前に, シリアル通信の受信バッファに前回のデータが入っている可能性があるため, m_pSerial->Clear()関数を呼び出し, 受信バッファをクリアしておきます. バッファクリアが終了したら, m_pSerial->SendAndReceive 関数を使用して作成した送信バッファ szCmdBuf の文字列を送信し, TCmini からの応答を受信します. このときターミネータの<CR>コードは, CSerial クラスによって自動付加されて送信し, 受信時には削除して応答を返します.

VAR_TYPE_BIT,VAR_TYPE_WORD の場合, 受信した文字列を数値に変換して返します. VAR_TYPE_UNKNOWN の場合は受信した文字列を返します. このように戻り値 pVal に値を入れることで, GetValue の実行結果がユーザに通知されます.

**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. FinalPutValue ()の実装

最後に FinalPutValue()メソッドを実装します. CaoVariable::put_Value を実行した場合, この関数が実行され, 引数 newVal にユーザが設定した値が格納されています.

書式

```
put_Value
(
    “設定値“
)
```

デフォルトのテンプレートでは, 親オブジェクトが CCaoController クラスの場合の実装をしています. システム変数は FinalPutCtrlSysValue 関数, ユーザ変数は FinalPutCtrlUserValue 関数を実行します.

今回, TCmini プロバイダでは, 親オブジェクトが CCaoController クラスの場合のみを想定しているので, デフォルトのテンプレートを変更せずに使用します. また, TCmini プロバイダでは, システム変数の put_Value は対応しないため, FinalPutCtrlSysValue 関数は実装しません FinalPutCtrlUserValue 関数のみを実装していきます.

**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. FinalPutCtrlUserValue()の実装

FinalPutCtrlUserValue 関数の実装を以下に示します. VAR_TYPE_UNKNOWN(通信コマンド)の場合, Put_Value の動作はしないため, エラーを返します.

szCmdBuf を送信バッファとして定義し, TCmini のプロトコルに従い, コマンド用文字列の先頭に<EC>コードを付加します. 引数 newVal の値を VT_I2 に変換し, sSetValue に格納します. 変換に失敗すれば, その時点でエラーを返します.

VAR_TYPE_BIT で, sSetValue が 0 以外の場合 I/O 強制セットと判断して「"8," + 変数名」を, sSetValue が 0 の場合 I/O 強制リセットと判断して「"9," + 変数名」を送信バッファに入れます. VAR_TYPE_WORD の場合, データ変更 1 語単位を実行するため, 「"10," + 変数名 + "," + sSetValue」を送信バッファに入れます.

実際に TCmini と通信する前に, シリアル通信の受信バッファに前回のデータが入っている可能性があるため, m_pSerial->Clear()関数を呼び出し, 受信バッファをクリアしておきます. バッファクリアが終了したら, m_pSerial->SendAndReceive 関数を使用して作成した送信バッファ szCmdBuf の文字列を送信し, TCmini からの応答を受信します. 応答文字列が"OK"であれば, 値の設定が正常に完了したと判断します.

**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. まとめ

以上で TCmini プロバイダのコーティングが終了しました.

ここでおこなった作業を以下にまとめます.

(1) プロバイダのプロジェクトウィザードを使ってプロジェクトを作成しました.
(2) RS-232C 制御する CSerial クラスを追加しました.
(3) CCaoProvController::FinalInitialize()で初期化処理を実装しました.
(4) CCaoProvController::ParseConnectionString()で接続パラメータ解析処理を実装しました.
(5) CCaoProvController::FinaleConnect()で接続処理を実装しました.
(6) CCaoProvController::FinalDisconnect()で切断処理を実装しました.
(7) CCaoProvController::GetSerial()でシリアル通信オブジェクトのポインタ取得処理を実装しました.
(8) CCaoProvController::FinalGetVariableNames()で変数名一覧取得処理を実装しました.
(9) CCaoProvVariable::FinalInitialize()で初期化処理を実装しました.
(10) CCaoProvVariable::FinalGetValue()でユーザ変数取得処理を実装しました.
(11) CCaoProvVariable::FinalPutValue()でユーザ変数の設定処理を実装しました.

最後に VC++のメニューから[ソリューションのビルド(B)]を実行して CaoProvTCmini.dll を作成します.

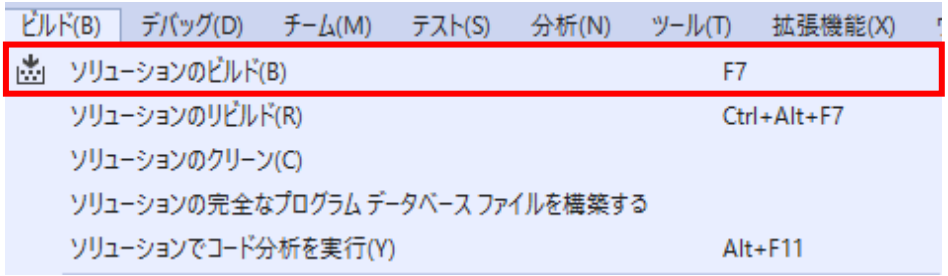



**図 4-5 TCmini.DLL のビルド**

## 4.4. TCmini プロバイダのデバッグとリリース

### 4.4.1. TCmini プロバイダのデバッグ

プロバイダの DLL モジュールは CAO エンジンである CAO.exe から必要に応じて呼ばれる構造になっています. したがって, プロバイダ DLL モジュールのデバッグをおこなう場合には VC++のデバッグセクションの実行可能なファイルに CAO.exe を指定する必要があります.

TCmini プロバイダのデバッグ手順は次のようになります. クライアントアプリケーションとしては ORiN の CAO 標準ツールである ORiN2¥CAO¥Tools¥CaoTester2.exe が適しているのでこれを使用します.

PC と TCmini コントローラはデバッグを開始する前にシリアルケーブルで接続し, コントローラの電源を入れておく必要があります.

(1) ビルドターゲットを[CAOPROV-Win32 Debug]にします.
[ビルド(B)]メニューから[アクティブな構成の設定(C)…]を選択して, プロジェクトの構成(P)から[CAOPROV-Win32 Debug]を指定します.


**図 4-6 ビルドターゲット指定画面**

(2) デバッグセクションの実行可能なファイルに CAO.exe を指定します.

CAOPROV プロパティページを開き, [構成]で Debug, [プラットフォーム]で Win32 を指定してから, [構成プロパティ]→[デバッグ]→[コマンド]に CAO.exe をフルパスで指定します. CAO.exe は通常 ORiN2¥CAO¥Engine¥Bin フォルダにあります.

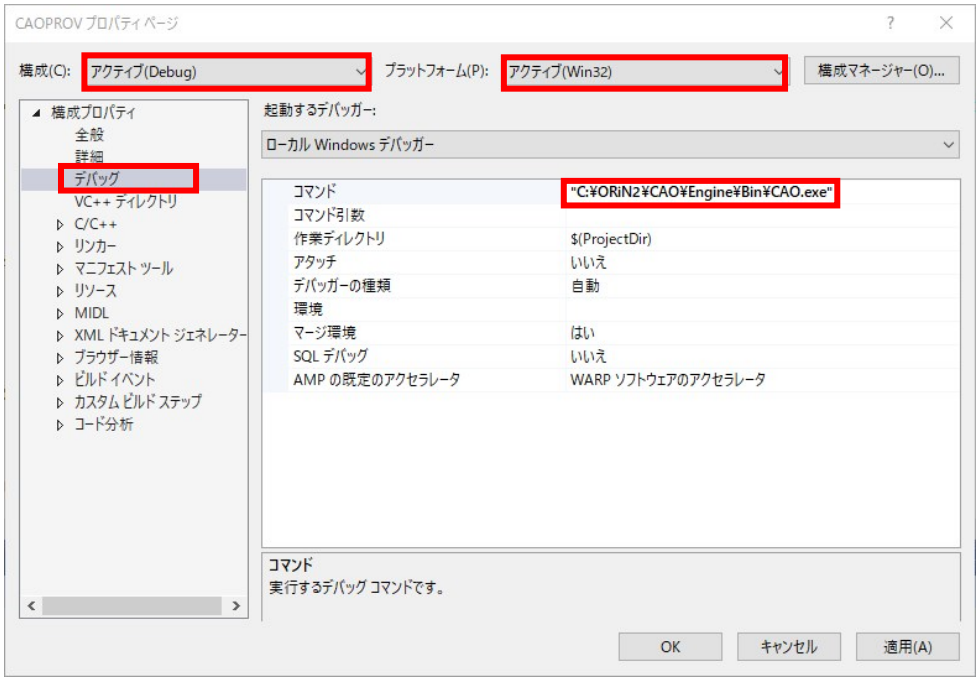



**図 4-7 CAOPROV プロパティの指定画面**

(3) デバッグしたい位置にブレークポイントをセットします.

デバッグしたい行を選択し, F9 でブレークポイントをセットすることができます. 以降は CCaoVariable クラスの FinalGetCtrlUserValue にブレークポイントをセットした例で説明します.

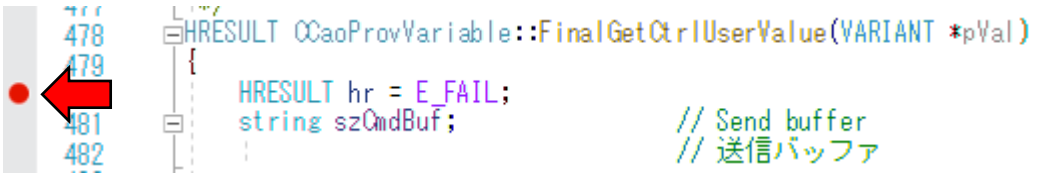


**図 4-8 ブレークポイント**

(4) デバッグの開始を実行します.

[ビルド(B)]メニューから[デバッグの開始(D)] – [実行(G)]を選択します.


**図 4-9 デバッグの開始画面**

(5) CaoTester2.exe を起動し, 以下の設定をします.

Controller Name: Sample(任意の文字列)

Provider Name: “CaoProv.TCmini”

Option:”Conn=com:1”(COM:1 で接続する場合)

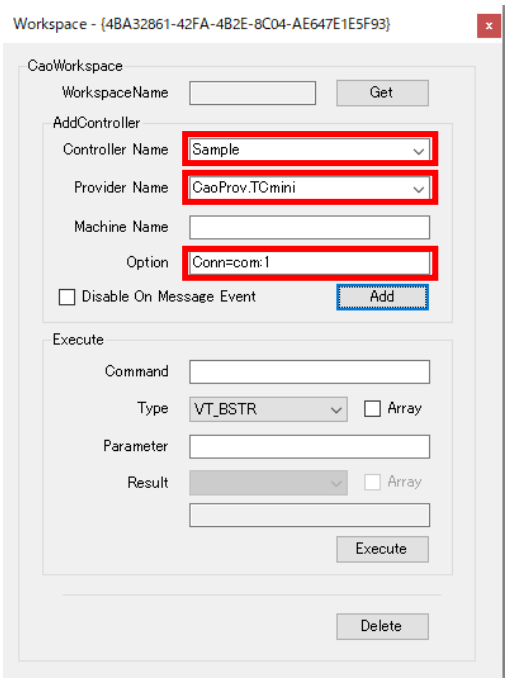



**図 4-10 CaoTester2 の CaoWorkspace 画面**

(6) [Add]ボタンを押下して COM:1 と接続を開始します.

このとき CaoWorkspace::AddController メソッドが実行され, CCaoProvController クラスで実装した FinalInitialize()メソッド→FinalConnect()メソッドの順番で呼び出されます. これらの関数の実行が成功すると, 以下の画面に切り替わります.

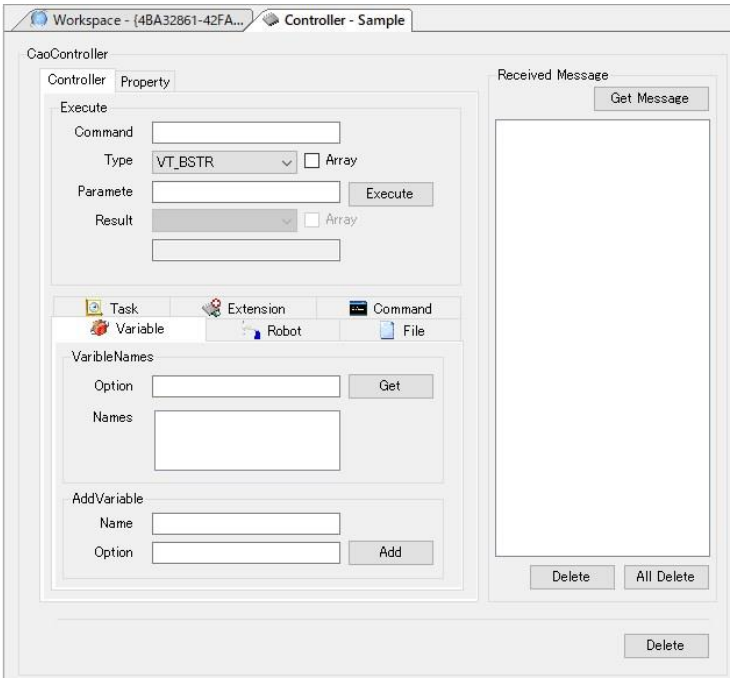



**図 4-11 CaoController クラスの画面**

(7) [Add]ボタンを押下して Variable クラスのオブジェクトを生成します.

<システム変数の場合>

Variable タブを選択して VariableNames の[Get]ボタンを押下します. このとき CaoController::get_VariableNames が実行され, CCaoProvController クラスで実装した FinalGetVariableNames()メソッドが呼び出されます. 関数の実行が終了すると, Names の部分にシステム変数の一覧が取得できます.

システム変数一覧からシステム変数を選択すると, 自動的に AddVariable の Name にシステム変数が入力されます. その状態で[Add]ボタンを押下すると, CaoController::AddVariable メソッドが実行されます. AddVariable が実行されると, CCaoProvVariable の FinalInitialize()が呼び出されます. 正常に終了すると, Variable クラスのオブジェクトが生成され, 画面が切り替わります.

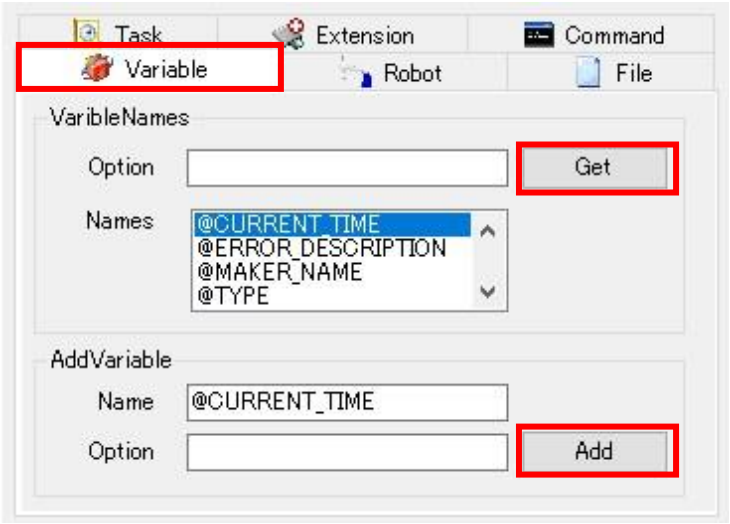



**図 4-12 システム変数の指定画面**

<ユーザ変数の場合>

ユーザ変数の場合，ユーザ自身で変数名を手動入力する必要があります．例として，X000 のデータメモリにアクセスする場合，AddVariable の Name に「X000」を入力し， [Add]ボタンを押下します．システム変数のときと同じように，CCaoProvVariable の FinalInitialize()が呼び出されます．正常に終了すると，Variable クラスのオブジェクトが生成され，画面が切り替わります．

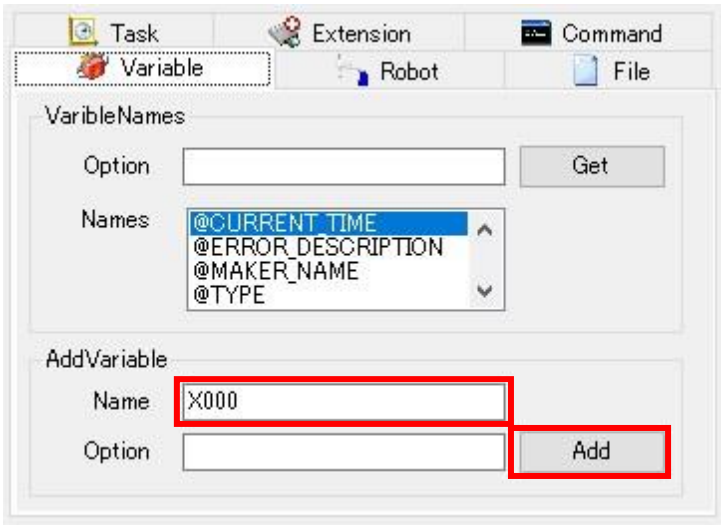



**図 4-13 ユーザ変数の指定画面**

(8) Variable クラスの GET，PUT

ユーザ変数”X000”の GET，PUT 実行について説明します．システム変数でも同様の動作をします．

<GET >

CaoVariable 画面の[GET]ボタンを押下すると, CaoVariable::get_Value が実行されます. プロバイダでは CCaoProvVariable クラスの FinalGetValue()が呼び出されます. 正常に終了すると, 以下の画面のように Type と Value が取得できます.

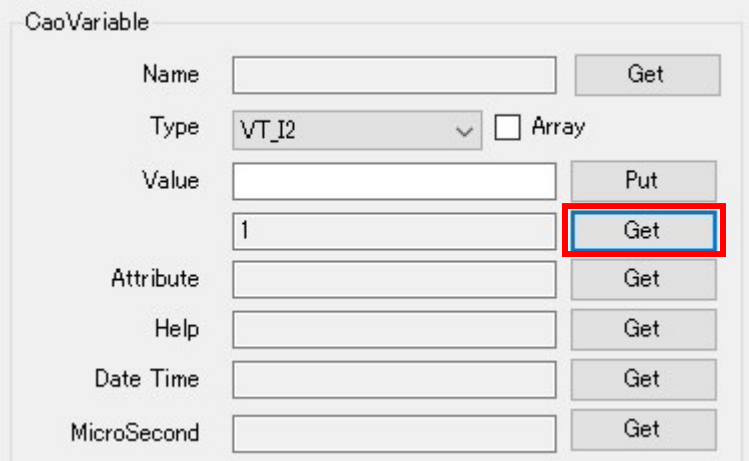



**図 4-14 Value の取得画面**

<PUT >

データ型と値を以下のように入力し, [PUT]ボタンを押下すると, CaoVariable::get_Value が実行されます. プロバイダでは CCaoProvVariable クラスの FinalGetValue()が呼び出されます. 正常に終了すると, 以下の画面のように Type と Value が取得できます.

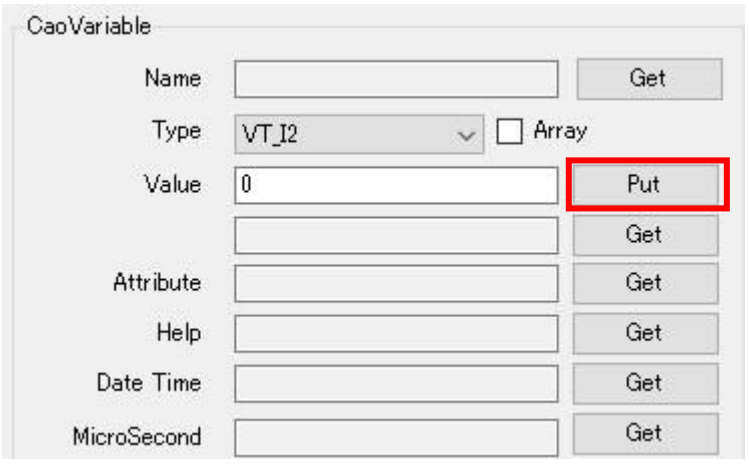



**図 4-15 Value の設定画面**

(9) Visual Studio に戻り, ブレークポイント位置で停止していれば成功です.

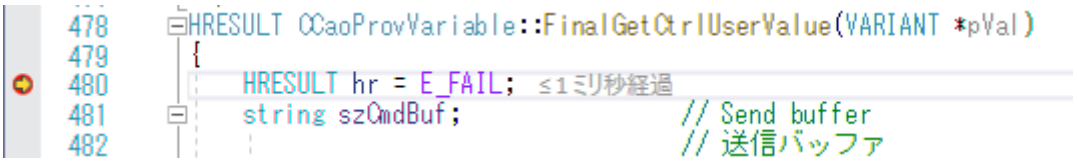



**図 4-16 ブレークポイント位置での停止**

### 4.4.2. TCmini プロバイダのリリース

ビルドターゲットを[Release MinDependency]に変更してビルドを行い，最終的なCaoProvTCmini.dllモジュールを作成します．Release MinDependency と Release MinSize の違いは以下の通りです．

Release MinDependency…DLL のサイズは大きくなりますが，DLL を使用する PC にモジュール(ATL100.dll)のインストールが不要です．

Release MinSize … DLL のサイズは最小になりますが，DLL を使用する PC にモジュール(ATL100.dll)がインストールされている必要があります．

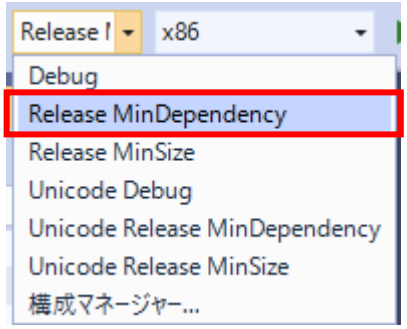



**図 4-17 ビルドターゲット指定画面**

#### 4.4.2.1. リリース用ドキュメントの作成

作成したプロバイダの仕様が外部公開されていなければユーザからは簡単にプロバイダを使用することができません．そこでプロバイダの仕様書を作成して公開する必要があります．

プロバイダ用仕様書としては最低下記の内容が記載されている必要があります．

(1) AddController()の接続パラメータの仕様
(2) ユーザ変数の仕様
(3) システム変数の仕様
(4) その他，重要と思われる情報(プロバイダ特有機能に関する情報，注意事項等)

Microsoft Word のテンプレートファイル(<インストールフォルダ>¥ORiN2¥CAO¥Provider¥Doc¥xxx_ProvGuide_jp(en).dot)を活用してください．具体的なサンプルとしては『TCmini 用プロバイダユーザーズガイド』(<インストールフォルダ>ORiN2¥CAO¥ProviderLib¥ToshibaMachine¥TCmini¥Doc¥TCmini_ProvGuide_jp(en).pdf)等を参考にしてください．

#### 4.4.2.2. プロバイダ依存情報の確認

VC++付属の Dependency Walker ツールを使って作成したモジュールの依存関係を調べることができます.

Dependency Walker を起動して TCmini.dll を指定すると下記の画面が表示されます.

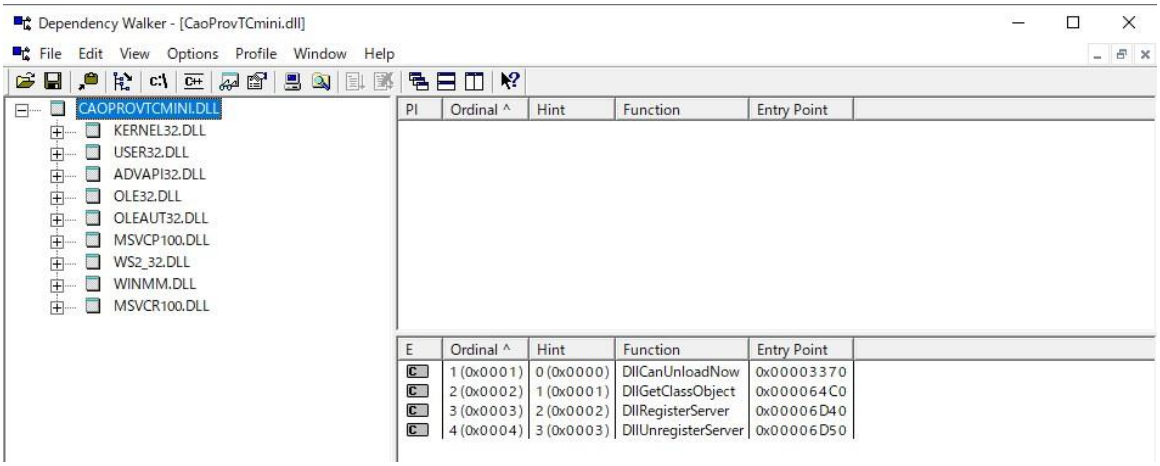



**図 4-18 TCmini.DLL の Dependency Walker 画面**

この画面から TCmini.dll は標準的なモジュールのみを必要としていることが判ります[4]. よって CAO エンジンを TCmini プロバイダに対応させるには単に TCmini.dll をコピーして Regsvr32 で登録すればよいということになります.

[4] モジュールによっては動的に他のモジュールをロードするものもあるのでその場合は Dependency Walker では検出されません. これを調べるには実際に調べたいモジュールを実行させてそのときメモリにロードされているモジュールを調査する必要があります.

# 5. プロバイダ作成 Tips

ここではプロバイダを実装する上で知っておくと便利なテクニックを紹介します.

## 5.1. 親オブジェクトを使う変数オブジェクトの作成

変数オブジェクトの作成時に親オブジェクトのポインタをメンバとして格納しておくことで, 親オブジェクトのメソッド及びプロパティを変数オブジェクトから呼び出すことができるようになります.

このような機能を追加することで以下のような利点があります.

- CaoSQL のように変数クラスにしかアクセスできないクライアントに対し, 親オブジェクトで実装したメソッド及びプロパティを公開できる.

以下に具体的な実装例を示します. ここでは, コントローラクラスのシステム変数(例:@PROVIDER_CANCEL)の put_Value を実行したときに, コントローラオブジェクトの Execute メソッド"ProviderCancelSampleMethod"を呼び出しています.

(1) メンバ変数の宣言

変数オブジェクトのメンバに親オブジェクトのポインタの格納先を追加する.

**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) 親オブジェクトのポインタを格納

変数クラスの初期化時に(1)で宣言したメンバ変数に親オブジェクトのポインタを格納する. 具体的には CCaoProvVariable の FinalInitialize()メソッドに以下のコードを追記します.

CAO エンジンによって親オブジェクトが破棄された場合は変数クラスの関数はコールされないので参照カウンタをカウントアップする(AddRef)必要はありません.

**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) 親オブジェクトのメソッドの呼び出し

システム変数"@PROVIDER_CANCEL"の put_Value が実行されたときに, コントローラオブジェクトの Execute メソッド"ProviderCancelSampleMethod"を呼び出すコードを CCaoProvVariable::FinalPutCtrlSysValue()に追記します.

**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. 付録

## 6.1. プロバイダ関数一覧

◆ CaoProvController Object ― コントローラ

| クラス | プロパティ, メソッド, イベント | | 説明 | R/W | 関数の引数 | | 呼び出されるタイミング | 備考 |
|---|---|---|---|---|---|---|---|---|
| | | | | | IN | OUT RETVAL | | |
| CaoProvController | Attribute | P | 属性の取得 | R | | 属性:Long | CaoController::Attribute()がコールされたとき. | |
| | CommandNames | P | コマンド名リストの取得 | R | [オプション:BSTR] | コマンド名リスト:VARIANT (VT_VARIANT\|VT_ARRAY) | CaoController::CommandNames()がコールされたとき. | オプションはフィルター条件等. |
| | ExtensionNames | P | 拡張ボード名リストの取得 | R | [オプション:BSTR] | 拡張ボード名リスト:VARIANT (VT_VARIANT\|VT_ARRAY) | CaoController::ExtensionNames()がコールされたとき. | オプションはフィルター条件等. |
| | FileNames | P | ファイル名リストの取得 | R | [オプション:BSTR] | ファイル名リスト:VARIANT (VT_VARIANT\|VT_ARRAY) | CaoController::FileNames()がコールされたとき. | オプションはフィルター条件等. ルートディレクトリのファイルー覧を返す. |
| | Help | P | ヘルプ | R | | ヘルプ文字列:BSTR | CaoController::Help() がコールされたとき. | |
| | RobotNames | P | ロボット名リストの取得 | R | [オプション:BSTR] | ロボット名リスト:VARIANT (VT_VARIANT\|VT_ARRAY) | CaoController::RobotNames()がコールされたとき. | オプションはフィルター条件等. |
| | TaskNames | P | タスク名リストの取得 | R | [オプション:BSTR] | タスク名リスト:VARIANT (VT_VARIANT\|VT_ARRAY) | CaoController::TaskNames()がコールされたとき. | オプションはフィルター条件等. |

| クラス | プロパティ，メソッド，イベント | | 説明 | R/W | 関数の引数 | | 呼び出されるタイミング | 備考 |
|---|---|---|---|---|---|---|---|---|
| | | | | | IN | OUT RETVAL | | |
| | VariableNames | P | 変数名リストの取得 | R | [オプション：BSTR] | 変数名リスト：VARIANT (VT_VARIANT\|VT_ARRAY) | CaoController::VariableNames()がコールされたとき. | オプションはフィルター条件等. |
| | Connect | M | 接続 | S | コントローラ名：BSTR, [オプション：BSTR] | | CaoWorkspace::AddController()がコールされたとき. | |
| | Disconnect | M | 切断 | S | | | CaoController オブジェクトが消滅するとき. | |
| | GetCommand | M | コマンドオブジェクトの取得 | S | コマンド名：BSTR, [オプション：BSTR] | オブジェクト：ICaoProvCommand | CaoController::AddCommand()がコールされたとき. | |
| | GetExtension | M | 拡張ボードオブジェクトの取得 | S | 拡張ボード名：BSTR, [オプション：BSTR] | オブジェクト：ICaoProvExtension | CaoController::AddExtension()がコールされたとき. | |
| | GetFile | M | ルートファイルオブジェクトの取得 | S | ファイル名：BSTR, [オプション：BSTR] | オブジェクト：ICaoProvFile | CaoController::AddFile()がコールされたとき. | |
| | GetRobot | M | ロボットオブジェクトの取得 | S | ロボット名：BSTR, [オプション：BSTR] | オブジェクト：ICaoProvRobot | CaoController::AddRobot()がコールされたとき. | |
| | GetTask | M | タスクオブジェクトの取得 | S | タスク名：BSTR, [オプション：BSTR] | オブジェクト：ICaoProvTask | CaoController::AddTask()がコールされたとき. | |
| | GetVariable | M | 変数オブジェクトの取得 | S | 変数名：BSTR, [オプション：BSTR] | オブジェクト：ICaoProvVariable | CaoController::AddVariable()がコールされたとき. | |
| | Execute | M | 拡張コマンドの実行 | S | コマンド：BSTR [パラメータ：VARIANT] | 結果：VARIANT | CaoController::Execute()がコールされたとき. | 機能拡張用 |
| | OnMessage | E | メッセージ受信イベント | R | メッセージ：ICaoProvMessage | | n/a | ・プロバイダ(コントローラ)→エンジン→クライアントの呼び出しを実現する.<br>・システムメッセージオプションを設定した場合は，クライアントへは届かない.<br>・メッセージ番号の負の範囲 |

| クラス | プロパティ，メソッド，イベント | 説明 | R/W | 関数の引数 | | 呼び出されるタイミング | 備考 |
|---|---|---|---|---|---|---|---|
| | | | | IN | OUT RETVAL | | |
| | | | | | | | は ORiN で予約されている. |
| 記号の意味 | M：メソッド<br>P：プロパティ<br>E：イベント | | （注 1） | ・[]内は省略可能引数.<br>・BSTR 型の省略可能引数のデフォルト値は NULL.<br>・数値型の省略可能引数のデフォルト値はゼロ.<br>・VARIANT 型の省略可能引数のデフォルト値は VT_ERROR. | | | |

(注1)記号の意味は次の通り. この中で，DAO エンジンのアクセス制限機能(書き込み制限機能)の ON/OFF に影響を受けるのは，W 属性のメソッドおよびプロパティのみ.
R －Read ： コントローラ，またはプロバイダ，またはエンジンのステータスやコンフィギュレーションを取得する.
W －Write ： コントローラのステータスや，コンフィギュレーションを変化させる. ただし，Execute メソッドはコマンドの内容に依存するので，S 属性とする. 必要があれば，プロバイダで書き込み制限を実装する.
S －Setup ： プロバイダ，またはエンジンのステータスやコンフィギュレーションを変化させる.

◆ CaoProvExtension Object ― 拡張ボード

| クラス | プロパティ，メソッド，イベント | | 説明 | R/W | 関数の引数 | | 呼び出されるタイミング | 備考 |
|---|---|---|---|---|---|---|---|---|
| | | | | | IN | OUT RETVAL | | |
| CaoProvExtension | Attribute | P | 属性の取得 | R | | 属性：Long | CaoExtension::Attribute()がコールされたとき. | |
| | Help | P | ヘルプ | R | | ヘルプ文字列：BSTR | CaoExtension::Help() がコールされたとき. | |
| | VariableNames | P | 変数名リストの取得 | R | [オプション：BSTR] | 変数名リスト：VARIANT (VT_VARIANT\|VT_ARRAY) | CaoExtension::VariableNames()がコールされたとき. | オプションはフィルター条件等. |
| | GetVariable | M | 変数オブジェクトの取得 | S | 変数名：BSTR,<br>[オプション：BSTR] | オブジェクト：ICaoProvVariable | CaoExtension::AddVariable()がコールされたとき. | |
| | Execute | M | 拡張コマンドの実行 | S | コマンド：BSTR<br>[パラメータ：VARIANT] | 結果：VARIANT | CaoExtension::Execute()がコールされたとき. | 機能拡張用 |
| 記号の意味 | M：メソッド<br>P：プロパティ<br>E：イベント | | | （注1） | ・[]内は省略可能引数.<br>・BSTR 型の省略可能引数のデフォルト値は NULL.<br>・数値型の省略可能引数のデフォルト値はゼロ.<br>・VARIANT 型の省略可能引数のデフォルト値は VT_ERROR. | | | |

◆ CaoProvFile Object ― ファイル

| クラス | プロパティ，メソッド，イベント | | 説明 | R/W | 関数の引数 | | 呼び出されるタイミング | 備考 |
|---|---|---|---|---|---|---|---|---|
| | | | | | IN | OUT RETVAL | | |
| CaoProvFile | Attribute | P | 属性の取得 | R | | 属性：Long | CaoFile::Attribute() がコールされたとき. | |
| | DateCreated | P | 作成日時 | R | | 作成日時：VARIANT | CaoFile::DateCreated() がコールされたとき. | |
| | DateLastAccessed | P | 最終アクセス日時 | R | | 最終アクセス日時：VARIANT | CaoFile::DateLastAccessed()がコールされたとき. | |
| | DateLastModified | P | 最終変更日時 | R | | 最終変更日時：VARIANT | CaoFile::DateLastModified()がコールされたとき. | |
| | FileNames | P | ファイル名リストの取得 | R | [オプション：BSTR] | ファイル名リスト：VARIANT (VT_VARIANT\|VT_ARRAY) | CaoFile::FileNames()がコールされたとき. | オプションはフィルター条件等. 属性がディレクトリの場合に子ファイル名一覧を返す. |
| | Help | P | ヘルプ | R | | ヘルプ文字列：BSTR | CaoFile::Help()がコールされたとき. | |
| | Path | P | パスの取得 | R | | パス名：BSTR | CaoFile::Path()がコールされたとき. | |
| | Size | P | ファイルサイズの取得 | R | | ファイルサイズ：Long | CaoFile::Size()がコールされたとき. | |
| | Type | P | ファイルのタイプの取得 | R | | ファイルタイプ：BSTR | CaoFile::Type()がコールされたとき. | |
| | Value | P | ファイルの内容 | R/W | データ：VARIANT | データ：VARIANT | CaoFile::Value()がコールされたとき. | |
| | VariableNames | P | 変数名リストの取得 | R | [オプション：BSTR] | 変数名リスト：VARIANT (VT_VARIANT\|VT_ARRAY) | CaoFile::VariableNames()がコールされたとき. | オプションはフィルター条件等. 型は(VT_VARIANT\|VT_ARRAY) |
| | GetFile | M | ファイルオブジェクトの取得 | S | ファイル名：BSTR, [オプション：BSTR] | オブジェクト：ICaoProvFile | CaoFile::AddFile()がコールされたとき. | |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | Copy | M | 複写 | W | 複写先ファイル名：BSTR<br>[オプション：BSTR] | | CaoFile::Copy()がコールされたとき. | |
| | Delete | M | 削除 | W | [オプション：BSTR] | | CaoFile::Delete() がコールされたとき. | |
| | Execute | M | 拡張コマンドの実行 | S | コマンド：BSTR<br>[パラメータ：VARIANT] | 結果：VARIANT | CaoFile::Execute()がコールされたとき. | |
| | GetVariable | M | 変数オブジェクトの取得 | S | 変数名：BSTR,<br>[オプション：BSTR] | オブジェクト：ICaoProvVariable | CaoFile::AddVariable() がコールされたとき. | |
| | Move | M | 移動 | W | 移動先ファイル名：BSTR<br>[オプション：BSTR] | | CaoFile::Move()がコールされたとき. | |
| | Run | M | タスクの生成 | W | [オプション：BSTR] | タスク名：BSTR | CaoFile::Run()がコールされたとき. | |
| 記号の意味 | M：メソッド<br>P：プロパティ<br>E：イベント | | | （注1） | ・[]内は省略可能引数.<br>・BSTR 型の省略可能引数のデフォルト値は NULL.<br>・数値型の省略可能引数のデフォルト値はゼロ.<br>・VARIANT 型の省略可能引数のデフォルト値は VT_ERROR. | | | |

◆ CaoProvRobot Object ― ロボット

| クラス | プロパティ, メソッド, イベント | | 説明 | R/W | 関数の引数 | | 呼び出されるタイミング | 備考 |
|---|---|---|---|---|---|---|---|---|
| | | | | | IN | OUT RETVAL | | |
| CaoProvRobot | Attribute | P | 属性の取得 | R | | 属性:Long | CaoRobot::Attribute() がコールされたとき. | |
| | Help | P | ヘルプ | R | | ヘルプ文字列:BSTR | CaoRobot::Help() がコールされたとき. | |
| | VariableNames | P | 変数名リストの取得 | R | [オプション:BSTR] | 変数名リスト:VARIANT (VT_VARIANT\|VT_ARRAY) | CaoRobot::VariableNames()がコールされたとき. | オプションはフィルター条件等. |
| | Accelerate | M | SLIM の ACCEL 文の仕様参照 | W | 軸番号:Long,<br>加速度:Float,<br>[減速度:Float] | | CaoRobot::Accelerate() がコールされたとき. | |
| | Change | M | SLIMの CHANGE 文の仕様参照 | W | ハンド名:BSTR | | CaoRobot::Change() がコールされたとき. | |
| | Chuck | M | SLIM の GRASP 文の仕様参照 | W | [オプション:BSTR] | | CaoRobot::Chuck() がコールされたとき. | |
| | Drive | M | SLIMの DRIVE 文の仕様参照 | W | 軸番号:Long,<br>移動量:float,<br>[動作オプション:BSTR] | | CaoRobot::Drive()がコールされたとき. | |
| | Execute | M | 拡張コマンドの実行 | S | コマンド:BSTR<br>[パラメータ:VARIANT] | 結果:VARIANT | CaoRobot::Execute()がコールされたとき. | |
| | GetVariable | M | CaoProvVariable の取得 | S | 変数名:BSTR,<br>[オプション:BSTR] | オブジェクト:ICaoVariable | CaoRobot::AddVariable() がコールされたとき. | |
| | GoHome | M | SLIM の GOHOME 文の仕様参照 | W | | | CaoRobot::GoHome() がコールされたとき. | |
| | Hold | M | SLIMの HOLD 文の仕様参照 | W | [オプション:BSTR] | | CaoRobot::Hold() がコールされたとき. | SLIM ではプログラムの一時停止の意味だか, CAO ではロボット動作の一時停止を意味している. |
| | Halt | M | SLIM の HALT 文の仕様参照 | W | [オプション:BSTR] | | CaoRobot::Halt() がコールされたとき. | SLIM ではプログラムの強制停止の意味だか, CAO ではロボット動作の強制停止を意味している. |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | Move | M | SLIM の MOVE 文の仕様参照 | W | 補間指定：Long,<br>ポーズ列：VARIANT,<br>[動作オプション：BSTR] | | CaoRobot::Move()がコールされたとき. | |
| | Rotate | M | SLIM の ROTATE 文の仕様参照 | W | 回転面：VARIANT,<br>角度：float,<br>回転中心：VARIANT,<br>[動作オプション：BSTR] | | CaoRobot::Rotate() がコールされたとき. | |
| | Speed | M | SLIM の SPEED/JSPEED 文の仕様参照 | W | 軸番号：Long,<br>速度：Float | | CaoRobot::Speed() がコールされたとき. | 軸番号は，-1：手先速度，0：全軸速度，その他は指定軸の軸速度. |
| | Unchuck | M | SLIM の RELEASE 文の仕様参照 | W | [オプション：BSTR] | | CaoRobot::Chuck() がコールされたとき. | SLIM の Release は予約語で使えないため Chuck/Unchuck に変更. |
| | Unhold | M | SLIM の HOLD 文の解除 | W | [オプション：BSTR] | | CaoRobot::Unhold() がコールされたとき. | SLIM では HOLD 文はプログラムの一時停止の意味であるため，再開のコマンドが規定されていないが，CAO ではロボット動作の一時停止を意味しているので，その再開に使う. |
| 記号の意味 | M：メソッド<br>P：プロパティ<br>E：イベント | | | （注1） | ・[]内は省略可能引数.<br>・BSTR 型の省略可能引数のデフォルト値は NULL.<br>・数値型の省略可能引数のデフォルト値はゼロ.<br>・VARIANT 型の省略可能引数のデフォルト値は VT_ERROR. | | | |

◆ CaoProvTask Object ― タスク

| クラス | プロパティ，メソッド，イベント | | 説明 | R/W | 関数の引数 | | 呼び出されるタイミング | 備考 |
|---|---|---|---|---|---|---|---|---|
| | | | | | IN | OUT RETVAL | | |
| CaoProvTask | Attribute | P | 属性の取得 | R | | 属性：Long | CaoTask::Attribute()がコールされたとき. | |
| | FileName | P | 対応ファイル名の取得 | R | | ファイル名：BSTR | CaoTask::FileName がコールされたとき. | |
| | Help | P | ヘルプ | R | | ヘルプ文字列：BSTR | CaoTask::Help()がコールされたとき. | |
| | VariableNames | P | 変数名リストの取得 | R | [オプション：BSTR] | 変数名リスト：VARIANT (VT_VARIANT\|VT_ARRAY) | CaoTask::VariableNames()がコールされたとき. | オプションはフィルター条件等. |
| | GetVariable | M | 変数オブジェクトの取得 | S | 変数名：BSTR,<br>[オプション：BSTR] | オブジェクト：ICaoProvVariable | CaoTask::AddVariable()がコールされたとき. | |
| | Delete | M | タスクの削除 | W | [オプション：BSTR] | | CaoTask::Delete()がコールされたとき. | |
| | Execute | M | 拡張コマンドの実行 | S | コマンド：BSTR<br>[パラメータ：VARIANT] | 結果：VARIANT | CaoTask::Execute() がコールされたとき. | |
| | Start | M | タスクの開始 | W | モード：Long,<br>[オプション：BSTR] | | CaoTask::Start() がコールされたとき. | モードは1：1サイクル実行，2：連続実行，3：1ステップ送り，4：1ステップ 戻し<br>オプションは開始位置など. |
| | Stop | M | タスクの停止 | W | モード：Long,<br>[オプション：BSTR] | | CaoTask::Stop() がコールされたとき. | モードは0：デフォルト停止，1：瞬時停止，2：ステップ停止，3：サイクル停止．4：初期化停止<br>(注)デフォルト停止とは，実際には何れかの停止方法. |
| 記号の意味 | M：メソッド<br>P：プロパティ<br>E：イベント | | | （注1） | ・[]内は省略可能引数.<br>・BSTR 型の省略可能引数のデフォルト値は NULL.<br>・数値型の省略可能引数のデフォルト値はゼロ.<br>・VARIANT 型の省略可能引数のデフォルト値は VT_ERROR. | | | |

◆ CaoProvVariable Object ― 変数

| クラス | プロパティ，メソッド，イベント | | 説明 | R/W | 関数の引数 | | 呼び出されるタイミング | 備考 |
|---|---|---|---|---|---|---|---|---|
| | | | | | IN | OUT RETVAL | | |
| CaoProvVariable | Attribute | P | 属性の取得 | R | | 属性：Long | CaoVariable::Attribute() がコールされたとき. | |
| | DateTime | P | タイムスタンプの取得 | R | | タイムスタンプ：VARIANT | CaoVariable::DateTime() がコールされたとき. | |
| | Help | P | ヘルプ | R | | ヘルプ文字列：BSTR | CaoVariable::Help() がコールされたとき. | |
| | Value | P | 値の取得 | R/W | 値：VARIANT | 値：VARIANT | CaoVariable::Value() がコールされたとき. | |
| 記号の意味 | M：メソッド<br>P：プロパティ<br>E：イベント | | | （注1） | ・[]内は省略可能引数.<br>・BSTR 型の省略可能引数のデフォルト値は NULL.<br>・数値型の省略可能引数のデフォルト値はゼロ.<br>・VARIANT 型の省略可能引数のデフォルト値は VT_ERROR. | | | |

◆ CaoProvCommand Object ― コマンド

| クラス | プロパティ，メソッド，イベント | | 説明 | R/W | 関数の引数 | | 呼び出されるタイミング | 備考 |
|---|---|---|---|---|---|---|---|---|
| | | | | | IN | OUT RETVAL | | |
| CaoProvCommand | Attribute | P | 属性の取得 | R | | 属性：Long | CaoCommand::Attribute()がコールされたとき. | |
| | Help | P | ヘルプ | R | | ヘルプ文字列：BSTR | CaoCommand::Help() がコールされたとき. | |
| | Parameters | P | コマンド，パラメータ | R/W | コマンドパラメータ：VARIANT | コマンドパラメータ：VARIANT | CaoCommand::Parameter()がコールされたとき. | |
| | State | P | 状態の取得 | R | | 状態：Long | CaoCommand::State() がコールされたとき. | |
| | Timeout | P | タイムアウト | R/W | タイムアウト：Timeout | タイムアウト：Timeout | CaoCommand::State() がコールされたとき. | |
| | Cancel | M | 実行中コマンドのキャンセル | S | | | CaoCommand::Cancel()がコールされたとき. | |
| | Execute | M | コマンドの実行 | S | オプション：Long | 実行結果：VARIANT | CaoCommand::Execute()がコールされたとき. | |
| 記号の意味 | M：メソッド<br>P：プロパティ<br>E：イベント | | | （注1） | ・[]内は省略可能引数.<br>・BSTR 型の省略可能引数のデフォルト値は NULL.<br>・数値型の省略可能引数のデフォルト値はゼロ.<br>・VARIANT 型の省略可能引数のデフォルト値は VT_ERROR. | | | |

◆ CaoProvMessage Object ― メッセージ

| クラス | プロパティ, メソッド, イベント | | 説明 | R/W | 関数の引数 | | 呼び出されるタイミング | 備考 |
|---|---|---|---|---|---|---|---|---|
| | | | | | IN | OUT RETVAL | | |
| CaoProMessage | DateTime | P | 作成日時 | R | | 作成日時:VARIANT | CaoMessage::DateTime()がコールされたとき. | |
| | Description | P | 説明 | R | | 説明:BSTR | CaoMessage::Description()がコールされたとき. | |
| | Destination | P | 送り先 | R | | 送り先:BSTR | CaoMessage::Destination()がコールされたとき. | |
| | Number | P | メッセージ番号 | R | | メッセージ番号:Long | CaoMessage::Number()がコールされたとき. | |
| | Option | P | オプション | R | | オプション:Long | DAO エンジンでメッセージを処理するとき. | 1 - 同期型メッセージ. (デフォルトは非同期)<br>2 - ログ出力.<br>4 - エンジン制御メッセージ. (予約) |
| | Source | P | 送り元 | R | | 送り元:BSTR | CaoMessage::Source()がコールされたとき. | |
| | Value | P | メッセージ本文 | R | | メッセージ本文:VARIANT | CaoMessage::Value()がコールされたとき. | |
| | Clear | M | メッセージのクリア | W | | | CaoMessage::Clear()がコールされたとき. | |
| | Reply | M | メッセージの返信 | W | 返信メッセージ:VARIANT | | CaoMessage::Reply()がコールされたとき. | |
| 記号の意味 | M:メソッド<br>P:プロパティ<br>E:イベント | | | (注1) | ・[]内は省略可能引数.<br>・BSTR 型の省略可能引数のデフォルト値は NULL.<br>・数値型の省略可能引数のデフォルト値はゼロ.<br>・VARIANT 型の省略可能引数のデフォルト値は VT_ERROR. | | | |

## 6.2. プロバイダテンプレート関数一覧

◆ CaoProvControllerImpl Template ― コントローラ

| クラス | プロパティ, メソッド, イベント | | 説明 | 関数の引数 | | 呼び出されるタイミング | 備考 |
|---|---|---|---|---|---|---|---|
| | | | | IN | OUT RETVAL | | |
| CaoProvControllerImpl | FinalGetAttribute | E | 属性の取得 | | 属性:Long | CaoProvController::get_Attribute()がコールされたとき. | |
| | FinalGetCommandNames | E | コマンド名リストの取得 | [オプション:BSTR] | コマンド名リスト:VARIANT (VT_VARIANT\|VT_ARRAY) | CaoProvController::get_CommandNames()がコールされたとき. | オプションはフィルター条件等. |
| | FinalGetExtensionNames | E | 拡張ボード名リストの取得 | [オプション:BSTR] | 拡張ボード名リスト:VARIANT (VT_VARIANT\|VT_ARRAY) | CaoProvController::get_ExtensionNames()がコールされたとき. | オプションはフィルター条件等. |
| | FinalGetFileNames | E | ファイル名リストの取得 | [オプション:BSTR] | ファイル名リスト:VARIANT (VT_VARIANT\|VT_ARRAY) | CaoProvController::get_FileNames()がコールされたとき. | オプションはフィルター条件等. ルートディレクトリのファイル一覧を返す. |
| | FinalGetHelp | E | ヘルプ | | ヘルプ文字列:BSTR | CaoProvController::get_Help()がコールされたとき. | |
| | FinalGetRobotNames | E | ロボット名リストの取得 | [オプション:BSTR] | ロボット名リスト:VARIANT (VT_VARIANT\|VT_ARRAY) | CaoProvController::get_RobotNames()がコールされたとき. | オプションはフィルター条件等. |
| | FinalGetTaskNames | E | タスク名リストの取得 | [オプション:BSTR] | タスク名リスト:VARIANT (VT_VARIANT\|VT_ARRAY) | CaoProvController::get_TaskNames()がコールされたとき. | オプションはフィルター条件等. |
| | FinalGetVariableNames | E | 変数名リストの取得 | [オプション:BSTR] | 変数名リスト:VARIANT (VT_VARIANT\|VT_ARRAY) | CaoProvController::get_VariableNames()がコールされたとき. | オプションはフィルター条件等. |
| | FinalExecute | E | 拡張コマンドの実行 | コマンド:BSTR [パラメータ:VARIANT] | 結果:VARIANT | CaoProvController::Execute()がコールされたとき. | 機能拡張用 |

| | | | | | | |
|---|---|---|---|---|---|---|
| FinalConnect | E | 接続 | | | CaoProvController::Connect()がコールされたとき. | |
| FinalDisconnect | E | 切断 | | | CaoProvController::Disonnect()がコールされたとき. | |
| FinalInitialize | E | 前処理 | | | オブジェクトが生成されるとき. | |
| FinalTerminate | E | 後処理 | | | オブジェクトが消滅するとき. | |
| OnTimer | E | タイマ割り込み | 自オブジェクト:thisポインタ | | 設定時間が経過したとき. | 周期は DAOP_TIMER_INTERVAL マクロで定義する. |
| CreateMessage | M | メッセージの作成 | メッセージ番号:Long,<br>[メッセージ本文:VARIANT],<br>[作成日時:VARIANT],<br>[送り先:BSTR],<br>[送り元:BSTR],<br>[説明:BSTR],<br>[オプション:Long] | メッセージ:CCaoProvMessage | n/a | ・引数省略時のメッセージのメンバは以下の値に設定する.<br>メッセージ番号 : 0<br>メッセージ本文 : VT_EMPTY<br>作成日時 : CreateMessageの実行日時<br>送り先 : 空文字<br>送り元 : コントローラ名<br>説明 : 空文字 |
| FireOnMessage | M | メッセージの送信 | メッセージ:CCaoProvMessage | | n/a | ・プロバイダ(コントローラ)→エンジン→クライアントの呼び出しを実現する.<br>・エンジン制御メッセージ(4)を設定した場合は, クライアントへは届かない.<br>・メッセージ番号の負の範囲はORiN で予約されている. |
| GetOptionValue | M | オプション文字列から指定したオプションの値を指定した型で取得する. | オプション:BSTR,<br>検索パラメータ:BSTR,<br>要求型:VARTYPE | 値:VARIANT | n/a | ・オプション文字列のフォーマットは, "<パラメータ1>[=<値1>] [, <パラメータ2>[=<値2>]]". <値>が省略されている場合は, 値はセットされないが関数自体は S_OK を返す. このときの返り値の型は VT_EMPTY である. |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | | | ・検索パラメータが見つからないときは，値はセットされないが関数自体は S_OK を返す．このときの返り値の型は VT_EMPTY である．<br>・要求型で指定できるのは，VT_I2，VT_I4，VT_R4，VT_R8，VT_BSTR, VT_BOOL である． |
| | SetTimerInterval | M | OnTimer()イベントのインターバル設定 | インターバル：DWORD | | n/a | ・単位はミリ秒です．<br>・0（ゼロ）を設定するとOnTimerイベントは無効になります．逆に 0 以外を設定するとイベントが有効になります． |
| | m_bstrName | D | コントローラ名 | コントローラ名：BSTR | n/a | | |
| | m_bstrOption | D | オプション | オプション：BSTR | n/a | | |
| | m_bTimer | D | OnTimer() イベントの有効・無効 | フラグ：BOOL | n/a | FALSE にするとOnTimer()イベントは発生しません． | |
| | m_dwInterval | D | OnTimer()イベントのインターバル | インターバル：DWORD | n/a | 単位はミリ秒． | |
| | m_dwLocaleID | D | ロケール ID[レジストリ] | ロケール ID：DWORD | n/a | CaoConfig 等でレジストリに保存された値． | |
| | m_szLicense | D | ライセンス[レジストリ] | ライセンス：TCHAR | n/a | CaoConfig 等でレジストリに保存された値． | |
| | m_szParameter | D | パラメータ[レジストリ] | パラメータ：TCHAR | n/a | CaoConfig 等でレジストリに保存された値． | |
| 記号の意味 | E：イベント(HRESULT)<br>M：メソッド(HRESULT)<br>D：データ | | | ・[]内は省略可能引数．<br>・BSTR 型の省略可能引数のデフォルト値は NULL．<br>・数値型の省略可能引数のデフォルト値はゼロ．<br>・VARIANT 型の省略可能引数のデフォルト値は VT_ERROR． | | | |

◆ CaoProvExtensionImpl Template ― 拡張ボード

| クラス | プロパティ，メソッド，イベント | | 説明 | 関数の引数 | | 呼び出されるタイミング | 備考 |
|---|---|---|---|---|---|---|---|
| | | | | IN | OUT RETVAL | | |
| CaoProvExtensionImpl | FinalGetAttribute | E | 属性の取得 | | 属性：Long | CaoProvExtension::get_Attribute()がコールされたとき. | |
| | FinalGetHelp | E | ヘルプ | | ヘルプ文字列：BSTR | CaoProvExtension::get_Help()がコールされたとき. | |
| | FinalGetVariableNames | E | 変数名リストの取得 | [オプション：BSTR] | 変数名リスト：VARIANT(VT_VARIANT\|VT_ARRAY) | CaoProvExtension::get_VariableNames()がコールされたとき. | オプションはフィルター条件等. |
| | FinalExecute | E | 拡張コマンドの実行 | コマンド：BSTR<br>[パラメータ：VARIANT] | 結果：VARIANT | CaoProvExtension::Execute()がコールされたとき. | 機能拡張用 |
| | FinalInitialize | E | 前処理 | 親オブジェクト：void | | CaoProvController::GetExtension()がコールされたとき. | |
| | FinalTerminate | E | 後処理 | | | オブジェクトが消滅するとき. | |
| | GetOptionValue | M | オプション文字列から指定したオプションの値を指定した型で取得する. | オプション：BSTR,<br>検索パラメータ：BSTR,<br>要求型：VARTYPE | 値：VARIANT | n/a | CaoProvControllerImpl::GetOptionValue()と同じ. |
| | m_bstrName | D | 拡張ボード名 | 拡張ボード名：BSTR | n/a | | |
| | m_bstrOption | D | オプション | オプション：BSTR | n/a | | |
| | m_bstrParent | D | 親オブジェクト名 | 親オブジェクト名：BSTR | n/a | | |
| 記号の意味 | E：イベント(HRESULT)<br>M：メソッド(HRESULT)<br>D：データ | | | ・[ ]内は省略可能引数.<br>・BSTR 型の省略可能引数のデフォルト値は NULL.<br>・数値型の省略可能引数のデフォルト値はゼロ.<br>・VARIANT 型の省略可能引数のデフォルト値は VT_ERROR. | | | |

◆ CaoProvFileImpl Template ― ファイル

| クラス | プロパティ，メソッド，イベント | 説明 | 関数の引数 | | 呼び出されるタイミング | 備考 |
|---|---|---|---|---|---|---|
| | | | IN | OUT RETVAL | | |
| CaoProvFileImpl | FinalGetAttribute E | 属性の取得 | | 属性：Long | CaoProvFile::get_Attribute()がコールされたとき. | |
| | FinalGetDateCreated E | 作成日時 | | 作成日時：VARIANT | CaoProvFile::get_DateCreated()がコールされたとき. | |
| | FinalGetDateLastAccessed E | 最終アクセス日時 | | 最終アクセス日時：VARIANT | CaoProvFile::get_DateLastAccessed()がコールされたとき. | |
| | FinalGetDateLastModified E | 最終変更日時 | | 最終変更日時：VARIANT | CaoProvFile::get_DateLastModified()がコールされたとき. | |
| | FinalGetFileNames E | ファイル名リストの取得 | [オプション：BSTR] | ファイル名リスト：VARIANT (VT_VARIANT\|VT_ARRAY) | CaoProvFile::get_FileNames()がコールされたとき. | オプションはフィルター条件等. 属性がディレクトリの場合に子ファイル名一覧を返す. |
| | FinalGetHelp E | ヘルプ | | ヘルプ文字列：BSTR | CaoProvFile::get_Help()がコールされたとき. | |
| | FinalGetPath E | パスの取得 | | パス名：BSTR | CaoProvFile::get_Path()がコールされたとき. | |
| | FinalGetSize E | ファイルサイズの取得 | | ファイルサイズ：Long | CaoProvFile::get_Size()がコールされたとき. | |
| | FinalGetType E | ファイルのタイプの取得 | | ファイルタイプ：BSTR | CaoProvFile::get_Type()がコールされたとき. | |
| | FinalGetValue E | ファイルの内容の取得 | | データ：VARIANT | CaoProvFile::get_Value()がコールされたとき. | |
| | FinalPutValue E | ファイルの内容の設定 | データ：VARIANT | | CaoProvFile::put_Value()がコールされたとき. | オプションはフィルター条件等. 型は(VT_VARIANT\|VT_ARRAY) |
| | FinalGetVariableNames E | 変数名リストの取得 | [オプション：BSTR] | 変数名リスト：VARIANT (VT_VARIANT\|VT_ARRAY) | CaoProvFile::get_VariableNames()がコールされたとき. | |
| | FinalCopy E | 複写 | 複写先ファイル名： | | CaoProvFile::Copy()がコー | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | BSTR<br>[オプション:BSTR] | | ルされたとき. | |
| | FinalDelete | E | 削除 | [オプション:BSTR] | | CaoProvFile::Delete()がコールされたとき. | |
| | FinalExecute | E | 拡張コマンドの実行 | コマンド:BSTR<br>[パラメータ:VARIANT] | 結果:VARIANT | CaoProvFile::Execute() がコールされたとき. | |
| | FinalInitialize | E | 前処理 | 親オブジェクト:void | | CaoProvController::GetFile()がコールされたとき. | |
| | FinalMove | E | 移動 | 移動先ファイル名:BSTR<br>[オプション:BSTR] | | CaoProvFile::Move() がコールされたとき. | |
| | FinalRun | E | タスクの生成 | [オプション:BSTR] | タスク名:BSTR | CaoProvFile::Run()がコールされたとき. | |
| | FinalTerminate | E | 後処理 | | | オブジェクトが消滅するとき. | |
| | GetOptionValue | M | オプション文字列から指定したオプションの値を指定した型で取得する. | オプション:BSTR,<br>検索パラメータ:BSTR,<br>要求型:VARTYPE | 値:VARIANT | n/a | CaoProvControllerImpl::GetOptionValue()と同じ. |
| | m_bstrName | D | ファイル名 | ファイル名:BSTR | n/a | | |
| | m_bstrOption | D | オプション | オプション:BSTR | n/a | | |
| | m_bstrParent | D | 親オブジェクト名 | 親オブジェクト名:BSTR | n/a | | |
| 記号の意味 | E:イベント(HRESULT)<br>M:メソッド(HRESULT)<br>D:データ | | | ・[]内は省略可能引数.<br>・BSTR 型の省略可能引数のデフォルト値は NULL.<br>・数値型の省略可能引数のデフォルト値はゼロ.<br>・VARIANT 型の省略可能引数のデフォルト値は VT_ERROR. | | | |

◆ CaoProvRobotImpl Template ― ロボット

| クラス | プロパティ，メソッド，イベント | | 説明 | 関数の引数 | | 呼び出されるタイミング | 備考 |
|---|---|---|---|---|---|---|---|
| | | | | IN | OUT RETVAL | | |
| CaoProvRobotImpl | FinalGetAttribute | E | 属性の取得 | | 属性：Long | CaoProvRobot::get_Attribute()がコールされたとき. | |
| | FinalGetHelp | E | ヘルプの取得 | | ヘルプ文字列：BSTR | CaoProvRobot::get_Help() がコールされたとき. | |
| | FinalGetVariableNames | E | 変数名リストの取得 | [オプション：BSTR] | 変数名リスト：VARIANT (VT_VARIANT\|VT_ARRAY) | CaoProvRobot::get_VariableNames() がコールされたとき. | オプションはフィルター条件等. |
| | FinalAccelerate | E | SLIM の ACCEL 文の仕様参照 | 軸番号：Long,<br>加速度：Float,<br>[減速度：Float] | | CaoProvRobot::Accelerate() がコールされたとき. | |
| | FinalChange | E | SLIM の CHANGE 文の仕様参照 | ハンド名：BSTR | | CaoProvRobot::Change() がコールされたとき. | |
| | FinalChuck | E | SLIM の GRASP 文の仕様参照 | [オプション：BSTR] | | CaoProvRobot::Chuck() がコールされたとき. | |
| | FinalDrive | E | SLIM の DRIVE 文の仕様参照 | 軸番号：Long,<br>移動量：float,<br>[動作オプション：BSTR] | | CaoProvRobot::Drive() がコールされたとき. | |
| | FinalExecute | E | 拡張コマンドの実行 | コマンド：BSTR<br>[パラメータ：VARIANT] | 結果：VARIANT | CaoProvRobot::Execute() がコールされたとき. | |
| | FinalInitialize | E | 前処理 | 親オブジェクト：void | | CaoProvController::GetRobot()がコールされたとき. | |
| | FinalGoHome | E | SLIM の GOHOME 文の仕様参照 | | | CaoProvRobot::GoHome() がコールされたとき. | |
| | FinalHold | E | SLIM の HOLD 文の仕様参照 | [オプション：BSTR] | | CaoProvRobot::Hold()がコールされたとき. | SLIM ではプログラムの一時停止の意味だか，CAO ではロボット動作の一時停止を意味している. |
| | FinalHalt | E | SLIM の HALT 文の仕様参照 | [オプション：BSTR] | | CaoProvRobot::Halt()がコールされたとき. | SLIM ではプログラムの強制停止の意味だか，CAO ではロボット動作の強制停止を意味している. |
| | FinalMove | E | SLIM の MOVE 文の仕様 | 補間指定：Long, | | CaoProvRobot::Move() がコ | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | 参照 | ポーズ列：VARIANT,<br>[動作オプション：BSTR] | | ールされたとき. | |
| | FinalRotate | E | SLIM の ROTATE 文の仕様参照 | 回転面：VARIANT,<br>角度：float,<br>回転中心：VARIANT,<br>[動作オプション：BSTR] | | CaoProvRobot::Rotate()がコールされたとき. | |
| | FinalSpeed | E | SLIM の SPEED/JSPEED 文の仕様参照 | 軸番号：Long,<br>速度：Float | | CaoProvRobot::Speed()がコールされたとき. | 軸番号は，-1：手先速度，0：全軸速度，その他は指定軸の軸速度. |
| | FinalUnchuck | E | SLIM の RELEASE 文の仕様参照 | [オプション：BSTR] | | CaoProvRobot::Unchuck()がコールされたとき. | SLIM の Release は予約語で使えないため Chuck/Unchuck に変更. |
| | FinalUnhold | E | SLIM の HOLD 文の解除 | [オプション：BSTR] | | CaoProvRobot::Unhold()がコールされたとき. | SLIM では HOLD 文はプログラムの一時停止の意味であるため，再開のコマンドが規定されていないが，CAO ではロボット動作の一時停止を意味しているので，その再開に使う. |
| | FinalTerminate | E | 後処理 | | | オブジェクトが消滅するとき. | |
| | GetOptionValue | M | オプション文字列から指定したオプションの値を指定した型で取得する. | オプション：BSTR,<br>検索パラメータ：BSTR,<br>要求型：VARTYPE | 値：VARIANT | n/a | CaoProvControllerImpl::GetOptionValue()と同じ. |
| | m_bstrName | D | ロボット名 | ロボット名：BSTR | n/a | | |
| | m_bstrOption | D | オプション | オプション：BSTR | n/a | | |
| | m_bstrParent | D | 親オブジェクト名 | 親オブジェクト名：BSTR | n/a | | |
| 記号の意味 | E：イベント(HRESULT)<br>M：メソッド(HRESULT)<br>D：データ | | | ・[]内は省略可能引数.<br>・BSTR 型の省略可能引数のデフォルト値は NULL.<br>・数値型の省略可能引数のデフォルト値はゼロ.<br>・VARIANT 型の省略可能引数のデフォルト値は VT_ERROR. | | | |

◆ CaoProvTaskImpl Template ― タスク

| クラス | プロパティ，メソッド，イベント | | 説明 | 関数の引数 | | 呼び出されるタイミング | 備考 |
|---|---|---|---|---|---|---|---|
| | | | | IN | OUT RETVAL | | |
| CaoProvTaskImpl | FinalGetAttribute | E | 属性の取得 | | 属性：Long | CaoProvTask::get_Attribute()がコールされたとき. | |
| | FinalGetFileName | E | 対応ファイル名の取得 | | ファイル名：BSTR | CaoProvTask::get_FileNameがコールされたとき. | |
| | FinalGetHelp | E | ヘルプ | | ヘルプ文字列：BSTR | CaoProvTask::get_Help() がコールされたとき. | |
| | FinalGetVariableNames | E | 変数名リストの取得 | [オプション：BSTR] | 変数名リスト：VARIANT(VT_VARIANT\|VT_ARRAY) | CaoProvTask::get_VariableNames()がコールされたとき. | オプションはフィルター条件等. |
| | FinalDelete | E | タスクの削除 | [オプション：BSTR] | | CaoProvTask::Delete() がコールされたとき. | |
| | FinalExecute | E | 拡張コマンドの実行 | コマンド：BSTR<br>[パラメータ：VARIANT] | 結果：VARIANT | CaoProvTask::Execute()がコールされたとき. | |
| | FinalInitialize | E | 前処理 | 親オブジェクト：void | | CaoProvController::GetTask()がコールされたとき. | |
| | FinalStart | E | タスクの開始 | モード：Long,<br>[オプション：BSTR] | | CaoProvTask::Start()がコールされたとき. | モードは1：1サイクル実行，2：連続実行，3：1ステップ 送り，4：1ステップ 戻し<br>オプションは開始位置など. |
| | FinalStop | E | タスクの停止 | モード：Long,<br>[オプション：BSTR] | | CaoProvTask::Stop()がコールされたとき. | モードは0：デフォルト停止，1：瞬時停止，2：ステップ停止，3：サイクル停止．4：初期化停止<br>(注)デフォルト停止とは，実際には何れかの停止方法. |
| | FinalTerminate | E | 後処理 | | | オブジェクトが消滅するとき. | |
| | GetOptionValue | M | オプション文字列から指定したオプションの値を指定した型で取得する. | オプション：BSTR,<br>検索パラメータ：BSTR,<br>要求型：VARTYPE | 値：VARIANT | n/a | CaoProvControllerImpl::GetOptionValue()と同じ. |
| | m_bstrName | D | タスク名 | タスク名：BSTR | n/a | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | m_bstrOption | D | オプション | オプション：BSTR | n/a | | |
| | m_bstrParent | D | 親オブジェクト名 | 親オブジェクト名：BSTR | n/a | | |
| 記号の意味 | E：イベント(HRESULT)<br>M：メソッド(HRESULT)<br>D：データ | | | ・[]内は省略可能引数.<br>・BSTR 型の省略可能引数のデフォルト値は NULL.<br>・数値型の省略可能引数のデフォルト値はゼロ.<br>・VARIANT 型の省略可能引数のデフォルト値は VT_ERROR. | | | |

◆ CaoProvVariableImpl Template ― 変数

| クラス | プロパティ, メソッド, イベント | | 説明 | 関数の引数 | | 呼び出されるタイミング | 備考 |
|---|---|---|---|---|---|---|---|
| | | | | IN | OUT RETVAL | | |
| CaoProvVariableImpl | FinalGetAttribute | E | 属性の取得 | | 属性:Long | CaoProvVariable::get_Attribute()がコールされたとき. | |
| | FinalGetDateTime | E | タイムスタンプの取得 | | タイムスタンプ:VARIANT | CaoProvVariable::get_DateTime()がコールされたとき. | |
| | FinalGetHelp | E | ヘルプ | | ヘルプ文字列:BSTR | CaoProvVariable::get_Help()がコールされたとき. | オプションはフィルター条件等. |
| | FinalGetValue | E | 値の取得 | | 値:VARIANT | CaoProvVariable::get_Value()がコールされたとき. | |
| | FinalPutValue | E | 値の設定 | 値:VARIANT | | CaoProvVariable::put_Value()がコールされたとき. | |
| | FinalInitialize | E | 前処理 | 親オブジェクト:void | | CaoProvController::GetVariable()がコールされたとき. | |
| | FinalTerminate | E | 後処理 | | | オブジェクトが消滅するとき. | |
| | GetOptionValue | M | オプション文字列から指定したオプションの値を指定した型で取得する. | オプション:BSTR,<br>検索パラメータ:BSTR,<br>要求型:VARTYPE | 値:VARIANT | n/a | CaoProvControllerImpl::GetOptionValue()と同じ. |
| | m_bstrName | D | 変数名 | 変数名:BSTR | n/a | | |
| | m_bstrOption | D | オプション | オプション:BSTR | n/a | | |
| | m_bstrParent | D | 親オブジェクト名 | 親オブジェクト名:BSTR | n/a | | |
| 記号の意味 | E:イベント(HRESULT)<br>M:メソッド(HRESULT)<br>D:データ | | | ・[]内は省略可能引数.<br>・BSTR 型の省略可能引数のデフォルト値は NULL.<br>・数値型の省略可能引数のデフォルト値はゼロ.<br>・VARIANT 型の省略可能引数のデフォルト値は VT_ERROR. | | | |

◆ CaoProvCommandImpl Template ― コマンド

| クラス | プロパティ，メソッド，イベント | | 説明 | 関数の引数 | | 呼び出されるタイミング | 備考 |
|---|---|---|---|---|---|---|---|
| | | | | IN | OUT RETVAL | | |
| CaoProvCommandImpl | FinalGetAttribute | E | 属性の取得 | | 属性：Long | CaoProvCommand::get_Attribute()がコールされたとき. | |
| | FinalGetHelp | E | ヘルプ | | ヘルプ文字列：BSTR | CaoProvCommand::get_Help()がコールされたとき. | |
| | FinalGetParameters | E | コマンド，パラメータの取得 | | コマンド，パラメータ：VARIANT | CaoProvCommand::get_Parameters()がコールされたとき. | |
| | FinalPutParameters | E | コマンド，パラメータの設定 | コマンド，パラメータ：VARIANT | | CaoProvCommand::put_Parameters()がコールされたとき. | |
| | FinalGetState | E | 状態の取得 | | 状態：Long | CaoProvCommand::get_State()がコールされたとき. | |
| | FinalGetTimeout | E | タイムアウトの取得 | | タイムアウト：Long | CaoProvCommand::get_Timeout()がコールされたとき. | |
| | FinalPutTimeout | E | タイムアウトの設定 | タイムアウト：Long | | CaoProvCommand::put_Timeout()がコールされたとき. | |
| | FinalCancel | E | 実行中コマンドのキャンセル | コマンド：Long | 結果：VARIANT | CaoProvCommand::Cancel()がコールされたとき. | |
| | FinalExecute | E | コマンドの実行 | コマンド：Long | 結果：VARIANT | CaoProvCommand::Execute()がコールされたとき. | |
| | FinalInitialize | E | 前処理 | 親オブジェクト：void | | CaoProvController::GetCommand()がコールされたとき. | |
| | FinalTerminate | E | 後処理 | | | オブジェクトが消滅するとき. | |
| | GetOptionValue | M | オプション文字列から指定したオプションの値を指定した型で取得する. | オプション：BSTR,<br>検索パラメータ：BSTR,<br>要求型：VARTYPE | 値：VARIANT | n/a | CaoProvControllerImpl::GetOptionValue()と同じ. |
| | m_bstrName | D | コマンド名 | コマンド名：BSTR | n/a | | |
| | m_bstrOption | D | オプション | オプション：BSTR | n/a | | |

| | | | | | | |
|---|---|---|---|---|---|---|
| | m_bstrParent D | 親オブジェクト名 | 親オブジェクト名 : BSTR | n/a | | |
| | m_vntParameters D | パラメータ | パラメータ : VARIANT | n/a | | |
| 記号の意味 | E : イベント(HRESULT)<br>M : メソッド(HRESULT)<br>D : データ | | ・[ ]内は省略可能引数.<br>・BSTR 型の省略可能引数のデフォルト値は NULL.<br>・数値型の省略可能引数のデフォルト値はゼロ.<br>・VARIANT 型の省略可能引数のデフォルト値は VT_ERROR. | | | |

◆ CaoProvMessageImpl Template ― メッセージ

| クラス | プロパティ，メソッド，イベント | | 説明 | 関数の引数 | | 呼び出されるタイミング | 備考 |
|---|---|---|---|---|---|---|---|
| | | | | IN | OUT RETVAL | | |
| CaoProvMessageImpl | FinalGetDateTime | E | 説明の取得 | | 説明：BSTR | CaoMessage::get_Description()がコールされたとき. | この関数をオーバーライドしなかった場合は Template 内で自動的に m_vntDateTime が返される. |
| | FinalGetDescription | E | 説明の取得 | | 説明：BSTR | CaoMessage::get_Description()がコールされたとき. | この関数をオーバーライドしなかった場合は Template 内で自動的に m_bstrDescription が返される. |
| | FinalGetDestination | E | 送り先の取得 | | 送り先：BSTR | CaoMessage::get_Destination()がコールされたとき. | この関数をオーバーライドしなかった場合は Template 内で自動的に m_bstrDestination が返される. |
| | FinalGetNumber | E | メッセージ番号の取得 | | メッセージ番号：Long | CaoMessage::get_Number()がコールされたとき. | この関数をオーバーライドしなかった場合は Template 内で自動的に m_lNumber が返される. |
| | FinalGetSource | E | 送り元の取得 | | 送り元：BSTR | CaoMessage::get_Source()がコールされたとき. | この関数をオーバーライドしなかった場合は Template 内で自動的に m_bstrSource が返される. |
| | FinalGetValue | E | メッセージ本文の取得 | | メッセージ本文：VARIANT | CaoMessage::get_Value() がコールされたとき. | この関数をオーバーライドしなかった場合は Template 内で自動的に m_vntValue が返される. |
| | FinalClear | E | メッセージのクリア | | | CaoMessage::Clear() がコールされたとき. | |
| | FinalReply | E | メッセージの返信 | 返信メッセージ：VARIANT | | CaoMessage::Reply() がコールされたとき. | |
| | FinalInitialize | E | 前処理 | 親オブジェクト：void | | CaoProvControllerImpl::FireOnMessage() がコールされたとき. | |
| | FinalTerminate | E | 後処理 | | | オブジェクトが消滅するとき. | |
| | m_vntDateTime | D | 作成日時 | 作成日時：VARIANT | n/a | | |
| | m_bstrDescription | D | 説明 | 説明：BSTR | n/a | | |
| | m_bstrDestination | D | 送り先 | 送り先：BSTR | n/a | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | m_lNumber | D | メッセージ番号 | メッセージ番号：Long | n/a | | |
| | m_bstrParent | D | 親オブジェクト名 | 親オブジェクト名：BSTR | n/a | | |
| | m_bstrSource | D | 送り元 | 送り元：BSTR | n/a | | |
| | m_vntValue | D | メッセージ本文 | メッセージ本文：VARIANT | n/a | | |
| 記号の意味 | E：イベント(HRESULT)<br>M：メソッド(HRESULT)<br>D：データ | | | ・[]内は省略可能引数.<br>・BSTR 型の省略可能引数のデフォルト値は NULL.<br>・数値型の省略可能引数のデフォルト値はゼロ.<br>・VARIANT 型の省略可能引数のデフォルト値は VT_ERROR. | | | |

## 6.3. サンプルプログラム

ORiN2SDK には，プロバイダのサンプルコードが以下のフォルダに格納されています．

**<インストールフォルダ>¥CAO¥ProviderLib**

これらのプロバイダの多くは Visual Studio 6.0（以下 VS6）で作成されています．Visual Studio 2005（以下 VS2005）でビルドするためには，いくつかの修正が必要になります．

まずは，VS6 のプロジェクトファイル（.dsw）を VS2005 で開き，VS2005 のプロジェクトファイル（.vcproj）にアップデート[5]してください．また，VS2005 でビルドを行った場合，以下ようなの問題が発生する場合がありますので，下表を参考にしてソースコードを修正してください．

**ビルド時のよくある問題点と対応方法**

| 問題点 | 対応方法 |
|---|---|
| “warning MIDL1015”が発生する. | ・ 下記の「warning MIDL2400 が発生する. 」の対処方法を行うと発生します. この警告は無視してください. |
| “warning MIDL2400”が発生する. | ・ メニューの[プロジェクト]→[プロパティ]を選択します.<br>・ CAOPROV のプロパティページの[構成]で「全ての構成」を選択します.<br>・ CAOPROV のプロパティページで[構成プロパティ]→[MIDL]から, [警告レベル]を“0 (/W0)”を選択します. |
| “warning C4996”が発生する. | 【対策方法1】<br>・ メニューの[プロジェクト]→[プロパティ]を選択します.<br>・ CAOPROV のプロパティページの[構成]で“全ての構成”を選択します.<br>・ CAOPROV のプロパティページで[構成プロパティ]→[C/C++]→[プリプロセッサ]から，[プリプロセッサの定義]に“_CRT_SECURE_NO_DEPRECATE”[6]を追加します. |
| | 【対策方法2】<br>・ メニューの[プロジェクト]→[プロパティ]を選択します.<br>・ CAOPROV のプロパティページの[構成]で「全ての構成」を選択します.<br>・ CAOPROV のプロパティページで[構成プロパティ]→[C/C++]→[プリプロセッサ]から，[プリプロセッサの定義]に“_CRT_NON_CONFORMING_SWPRINTFS”を追加します. |
| | 【対策方法3】 |

[5] http://msdn2.microsoft.com/en-us/7hfabkez(VS.80).aspx
[6] http://msdn2.microsoft.com/en-us/8ef0s5kh(VS.80).aspx

| | |
|---|---|
| | ・ 警告の発生している関数名の先頭に“_”を追加する.<br>例 1) wcsnicmp() → _wcsnicmp()<br>例 2) itoa() → _itoa() |
| “warning LNK4222”が発生する. | ・ CaoProv.def 内のエクスポート関数の序数を削除します.<br>例） DllCanUnloadNow @1 PRIVATE<br>→ DllCanUnloadNow PRIVATE |
| XML 関連のクラスで“error C2872”が発生する. [7] | 【対策方法1】<br>・ 以下の 2 行が記述してある箇所をコメントアウトする.<br>#import "msxml4.dll"<br>using namespace MSXML2; |
| | 【対策方法2】<br>・ エラーが発生している全クラスにネームスペースを指定する.<br>例）ネームスペース名が“MSXML2”のとき<br>IXMLDOMNodePtr pIChildNode;<br>→ MSXML2::IXMLDOMNodePtr pIChildNode; |

[7] http://support.microsoft.com/default.aspx?scid=kb;ja;316317