CAO プロバイダ作成ガイド

Version 1.1.1

September 9, 2014

【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

目次

1. はじめに

本書は，CAO 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．

まず第 2 章では，CAO プロバイダ作成のために CAO 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，Microsoft Visual C++ 6.0(以降，VC++と表記)による CAO プロバイダの作成手順について解説をおこないます．

次に第 3 章では，CAO 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 5 章では，東芝機械製小型プログラマブルコントローラ TCmini α　TC3-02 用 CAO プロバイダを例に挙げ，具体的な CAO プロバイダの実装手順を解説します．

最後に第 6 章では，ORiN2 SDK 付属の CaoProvExec ツールについて解説をおこないます．

2. CAO プロバイダの実装手順

2.1. 概要

CAO(Controller Access Object)とは，各メーカの産業用ロボットに対して，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をベースに開発され，当初からの対象である産業用ロボットだけでなく，PLC(Programmable Logic Controller)や NC 工作機などにも適用されその応用範囲を広げています．

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，CAO 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，CAO 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，CAO プロバイダは「6.付録 A.2CAO 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，CAO 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，CAO プロバイダの実装することが可能となります．

本章では，CAO プロバイダの実装手順を順に解説します．

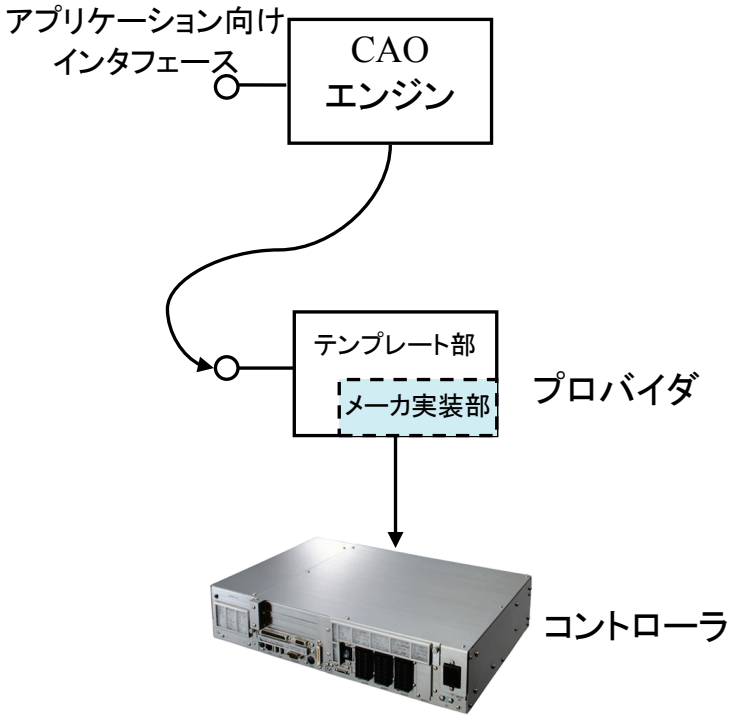


図 2-1 CAO の構成

2.2. 実装の手順

CAO 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.

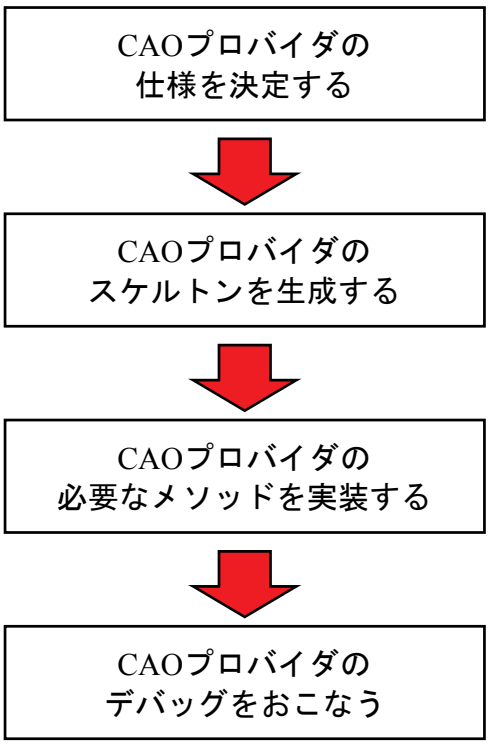


図 2-2 CAO プロバイダ実装手順

(1) CAO プロバイダの仕様を決定する

CAO 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. CAO 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) CAO プロバイダのテンプレートを作成する

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

(3) CAO プロバイダの必要なメソッドを実装する

CaoProvWiz で生成された CAO 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) CAO プロバイダのデバッグをおこなう

作成した CAO 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，CAO 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

CAO 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]→[CAO]→[Provider]→[Bin]→[CaoProvWiz]を実行する．

(2) CAO 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 2-3 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

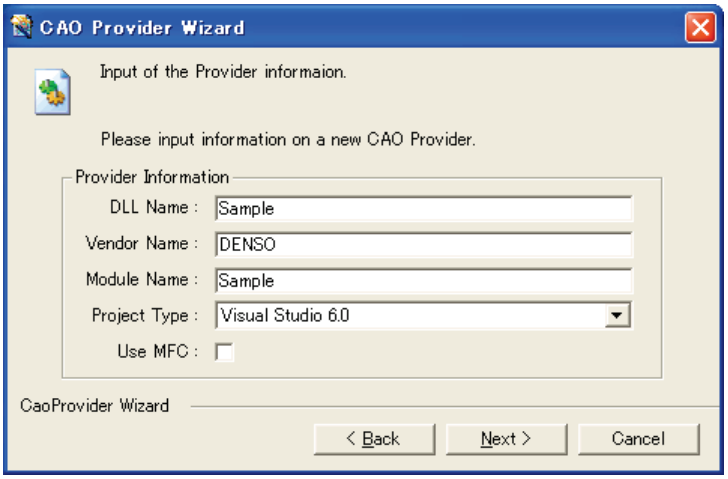


図 2-4 プロバイダ情報入力画面

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.
CAO 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スケルトンの作成が開始されます.

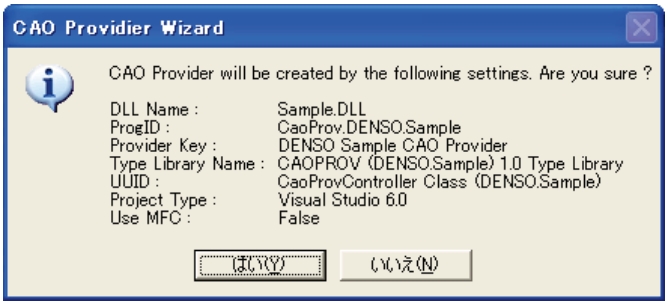


図 2-5 作成プロバイダ情報確認画面

(5) 以下の画面が表示されたら, プロバイダのスケルトンは作成完了です.

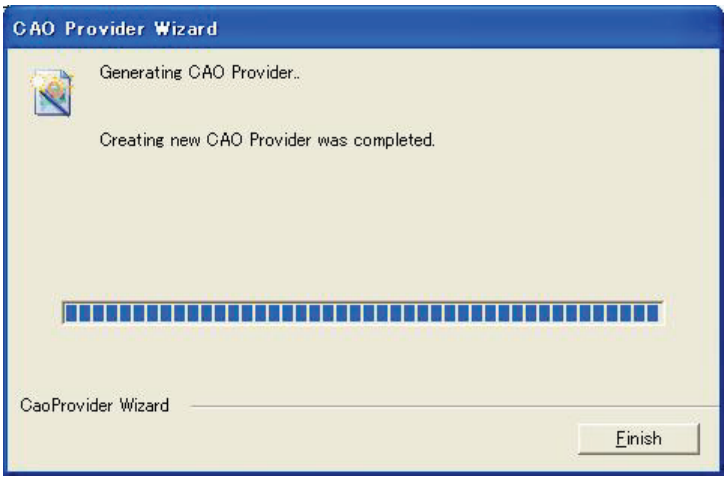


図 2-6 プロバイダのプロジェクト作成完了画面

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以下に作成されます.

この時点で CAO プロバイダの VC++プロジェクトが CAOPROV.dsw 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.dsw 以外にも複数のファイルが存在します. 下表に CAO プロバイダに関連深いおも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 VC++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 Studio 6.0 のみ)
2	CaoProv.dsp	プロジェクトファイル(Visual Studio 6.0 のみ)
3	CaoProv8.vcproj	プロジェクトファイル(Visual Studio 2005 のみ)
4	CaoProv9.vcproj	プロジェクトファイル(Visual Studio 2008 のみ)
5	CaoProv10.vcproj	プロジェクトファイル(Visual Studio 2010 のみ)
6	CaoProv.IDL	IDL ファイル
7	Resource.h	リソースヘッダファイル
8	CaoProv.rc	リソースファイル
9	StdAfx.h	C ヘッダファイル
10	StdAfx.cpp	C++ソースファイル
11	CaoProv.h	C ヘッダファイル
12	CaoProv.cpp	C++ソースファイル
13	CaoProvController.h	C ヘッダファイル CCaoProvController クラスの定義
14	CaoProvController.cpp	C++ソースファイル CCaoProvController クラスの実装
15	CaoProvVariable.h	C ヘッダファイル CCaoProvVariable クラスの定義
16	CaoProvVariable.cpp	C++ソースファイル CCaoProvVariable クラスの実装
17	CaoProvTask.h	C ヘッダファイル CCaoProvTask クラスの定義
18	CaoProvTask.cpp	C++ソースファイル CCaoProvTask クラスの実装
19	CaoProvRobot.h	C ヘッダファイル CCaoProvRobot クラスの定義
20	CaoProvRobot.cpp	C++ソースファイル CCaoProvRobot クラスの実装
21	CaoProvFile.h	C ヘッダファイル CCaoProvFile クラスの定義
22	CaoProvFile.cpp	C++ソースファイル CCaoProvFile クラスの実装
23	CaoProvCommand.h	C ヘッダファイル CCaoProvCommand クラスの定義
24	CaoProvCommand.cpp	C++ソースファイル CCaoProvCommand クラスの実装
25	CaoProvExtension.h	C ヘッダファイル CCaoProvExtension クラスの定義
26	CaoProvExtension.cpp	C++ソースファイル CCaoProvExtension クラスの実装
27	CaoProvMessage.h	C ヘッダファイル CCaoProvMessage クラスの定義
28	CaoProvMessage.cpp	C++ソースファイル CCaoProvMessage クラスの実装

(6) 以上の操作でプロバイダのスケルトンプロジェクトができているので，このまま(何の機能も持ちませ

んが)ビルドすることができます. その場合には, Visual C++の[ビルド]メニューの[アクティブな構成の設定]で, “Win32 Debug”(デバッグ版)か, “Win32 Release MinDependency”(リリース版)を選択し, [リビルド]を実行してください.

2.4. CAO 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

CAO 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 E_NOTIMPL; //実装したら返り値を S_OK にする.

}
*/  ←  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::FinalPutValue(VARIANT newVal)
{
        :（記述内容は省略）

        // 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 E_NOTIMPL; //実装したら返り値を S_OK にする.
}
```

(3) (2)のメソッド内の返り値を“S_OK”とします.

```
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.

具体的な内容の記述例は基本的なメソッドについてのみ後述します. 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.付録 A.2CAO 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 CAO プロバイダのクラス機能概要

クラス名	説明
CaoProvController	コントローラクラス. 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	変数クラス. 変数リソースに関わる機能を提供します.
CaoProvRobot	ロボットクラス. ロボットリソースに関わる機能を提供します.
CaoProvFile	ファイルクラス. 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	タスククラス. タスクリソースに関わる機能を提供します.
CaoProvCommand	コマンドクラス. コマンドリソースに関わる機能を提供します.
CaoProvExtension	拡張クラス. 拡張ボードリソースに関わる機能を提供します.
CaoProvMessage	メッセージクラス. 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
	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 CAO プロバイダの実装するときに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()，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()

このメソッドは，CaoController 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．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ます. このメソッドでは作成するプロバイダが対応するロボットコントローラとの接続処理を実装します. 接続方法は各ロボットコントローラに依存しますが, 例えばロボットコントローラへアクセスする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コールされます. このメソッドではFinalConnect()で接続したロボットコントローラとの切断処理をおこないます. 切断方法は各ロボットコントローラに依存します.

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

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. 変数は必要に応じてロボットコントローラから取得します. ロボットコントローラにアクセスする必要が無ければクラスのメンバ変数として保持することも可能です. 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.

以下にこのメソッドの, コントローラクラスのシステム変数である@VERSION の実装例を示します.

(1) StdAfx.h に必要なマクロ定義を行う.

```
// ========== 各社追加部分はこれ以下に記述する. ==========
#define CS_VERSION        0x0002              ←  ユニークな番号を割り当てる
#define CS_VERSION$       L"@VERSION"         ←  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_VERSION ),   ←  マクロ追加
        };
        :
        return S_OK;
}
```

(3) コントローラクラスシステム変数の値の取得を実装する

```
HRESULT CCaoProvVariable::FinalGetCtrlSysValue(VARIANT *pVal)
{
        switch (m_lUSysId) {
                :
        case CS_VERSION:                        ←  以下追加
                pVal->vt = VT_BSTR;
                pVal->bstrVal = SysAllocString(L"0.0.0");
                break;
                :
        }

        return hr;
}
```

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::FinalPutValue(VARIANT newVal)

変数の値を設定します.

引数 newVal にはクライアントからの値が入れられています. newVal にどのようなデータ型の値が入ってくるかは決まっていません. そのため値を設定する前に引数の型を判別して, 適宜値を VARIANT から取り出します.

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.

以下にこのメソッドの, コントローラにユーザ変数を設定する実装例を示します

List 2-1 CCaoProvVariable.cpp – FinalPutValue()

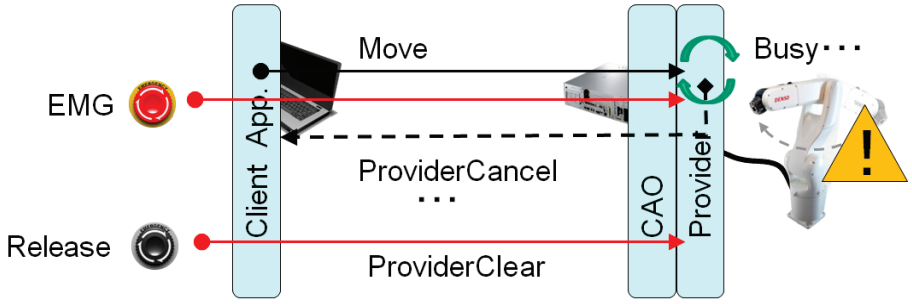
```
HRESULT CCaoProvVariable::FinalPutValue(VARIANT newVal)
{
	HRESULT hr = E_ACCESSDENIED;

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

HRESULT CCaoProvVariable:: FinalPutCtrlUserValue (VARIANT newVal)
{
    HRESULT  hr = S_OK;
    switch( newVal.vt ){
    case VT_I2:
        // m_pRobotCtrl はロボットコントローラオブジェクトへのポインタ
        // PutRCiVal()は VT_I2 型のユーザ変数の値を設定する.
        hr = m_pRobotCtrl->PutRCiVal( newVal.iVal );
        break;
    case VT_I4;
        :
        :
    default:
        hr = E_INVALIDARG;
    }
    return hr;
}
```

ここで m_pRobotCtrl と PutRCiVal()は実際には存在しません. この例では newVal のデータ型に合わせて値をコントローラに設定しています. 実際にはコントローラのアクセスメソッドに合わせて値を設定する処理を実装してください.

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 ORiN2 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 で作成されたスケルトンでは 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 2-2 CCaoProvController.cpp – ExecProviderCancel()

```
/** プロバイダキャンセル
 *
 *  実行中の処理を直ちに中断する
 *
 *    @param   vntPara   :        [in] パラメータ
 *                                     未使用
 *    @param   pVal      :        [out] 実行結果
 *                                     未使用
 *    @retval  HRESULT
 *
 */
```

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

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

	// 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 で作成されたスケルトンでは 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 2-3 CCaoProvController.cpp – ExecProviderClear()

```
/** プロバイダクリア
 *
 *  実行中断をクリアする.
 *
 *     @param    vntPara   :        [in] パラメータ
 *                                       未使用
 *     @param    pVal      :        [out] 実行結果
 *                                       未使用
 *     @retval   HRESULT
 *
 */
HRESULT CCaoProvController::ExecProviderClear(VARIANT vntParam, VARIANT *pVal)
{
	ATLASSERT(m_hProviderCancelEvent != NULL);

	// 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: 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 2-4 CCaoProvController.rgs

```
HKCR
{
    CaoProv.CaoProvController.1 = s 'CaoProvController Class'
    {
        CLSID = s '{1de03cfd-535f-42db-88cf-3a72bee12813}'
    }
    CaoProv.CaoProvController = s 'CaoProvController Class'
    {
        CLSID = s '{1de03cfd-535f-42db-88cf-3a72bee12813}'
        CurVer = s 'CaoProv.CaoProvController.1'
    }
    NoRemove CLSID
    {
        ForceRemove {1de03cfd-535f-42db-88cf-3a72bee12813} = s 'CaoProvController Class'
        {
            ProgID = s 'CaoProv.CaoProvController.1'
            VersionIndependentProgID = s 'CaoProv.CaoProvController'
            InprocServer32 = s '%MODULE%'
            {
                val ThreadingModel = s 'Free'
            }
            'TypeLib' = s '{4489ef8d-cf16-414e-9d93-cb9af157cff2}'
            'CAO Provider' = s 'Skeleton CAO Provider'
            {
                val Enabled = d '4294967295'
                val RunAsLocal = d '0'
                val Writable = d '4294967295'
                val LocaleID = d '1024'
                val License = s ''
                val Parameter = s ''
                val CRDFile = s ''
                val BindCmds = d '4294967295'
                val BindExec = d '0'
                val GroupID = d '1'
            }
            val AppID = s '{1de03cfd-535f-42db-88cf-3a72bee12813}'
        }
    }
    NoRemove AppID
    {
        ForceRemove {1de03cfd-535f-42db-88cf-3a72bee12813} = s 'CaoProvController Class'
        {
            val DllSurrogate = s ''
```

```
        }
    }
}
```

上記のファイル内で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 0 : 使用不可 4294967295（0xffffffff） : 使用可
RunAsLocal	CaoWorkspace::AddController 時に起動マシン名を省略した場合の CAO プロバイダのデフォルト起動方法を指定します． 0 : インプロセス起動 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 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． 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） 0 : ダイナミックバインディング不可 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） 0 : ダイナミックバインディング不可 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. CAO 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，CAO 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，CAO 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¥CaoTester¥Bin¥CaoTester.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[CAOPROV-Win32 Debug]にしてください．

[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．

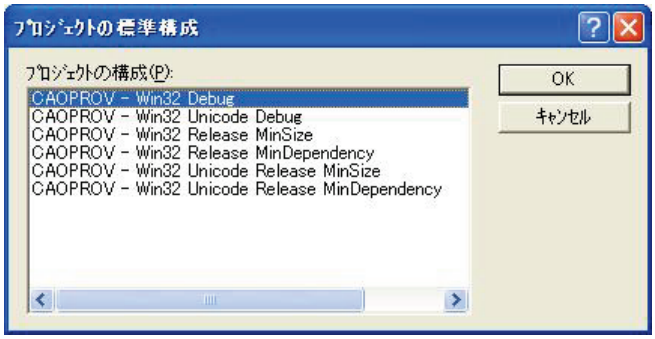


図 2-7 ビルドターゲット指定画面

(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．

[プロジェクト(P)]メニューから[設定(S) Alt+F7]を選択して，設定の対象(S)で[Win32 Debug]を指定してからデバッグセクションの実行可能なファイル(E)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．

[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．

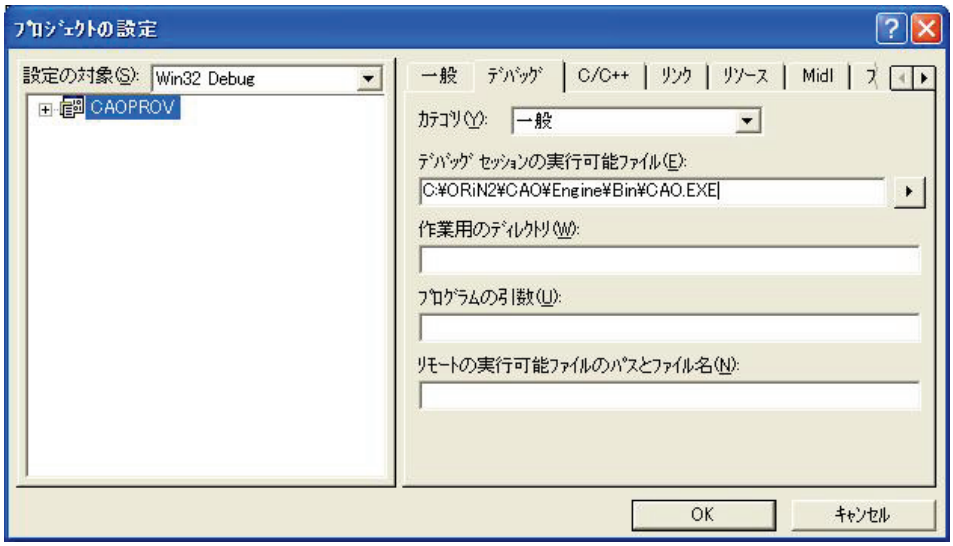


図 2-8 CAO.exe の指定画面

(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.

[ビルド(B)]メニューから[デバッグの開始(D)] – [実行(G)]してください.

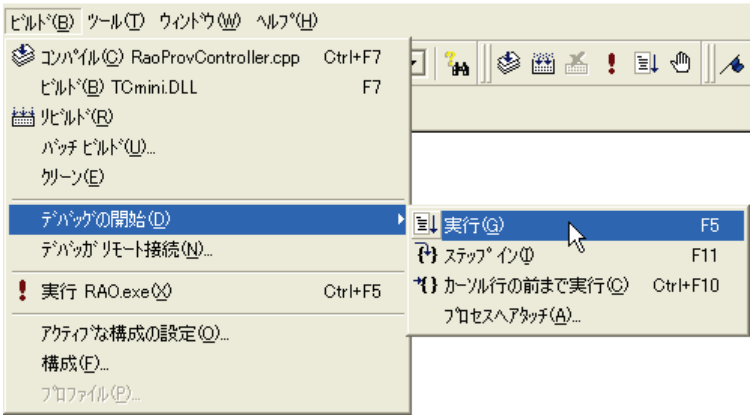


図 2-9 デバッグの開始画面

(5) CaoTester.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) CaoTester 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.

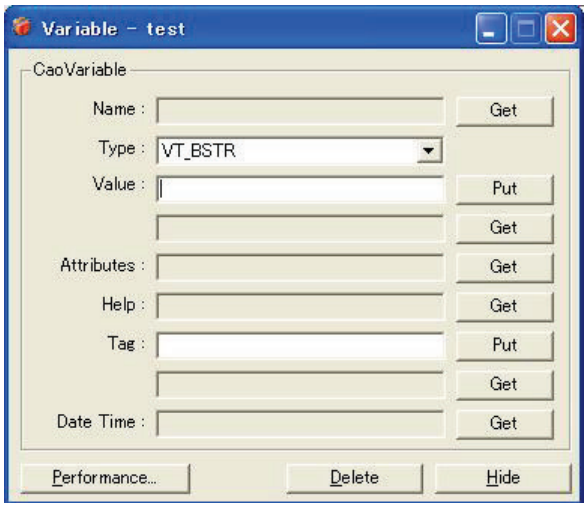


図 2-10 システム変数の指定画面

(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．

CAO 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．見本として ORiN SDK に付属する ORiN2¥CAO¥ProviderLib¥の例えば ToshibaMachine¥Doc¥CaoProvTCmini 仕様書.doc などを参考にしてください．

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．CAO 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 CAO 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，VC++の Dependency Walker やフリーツールの Process Explorer などを利用します．

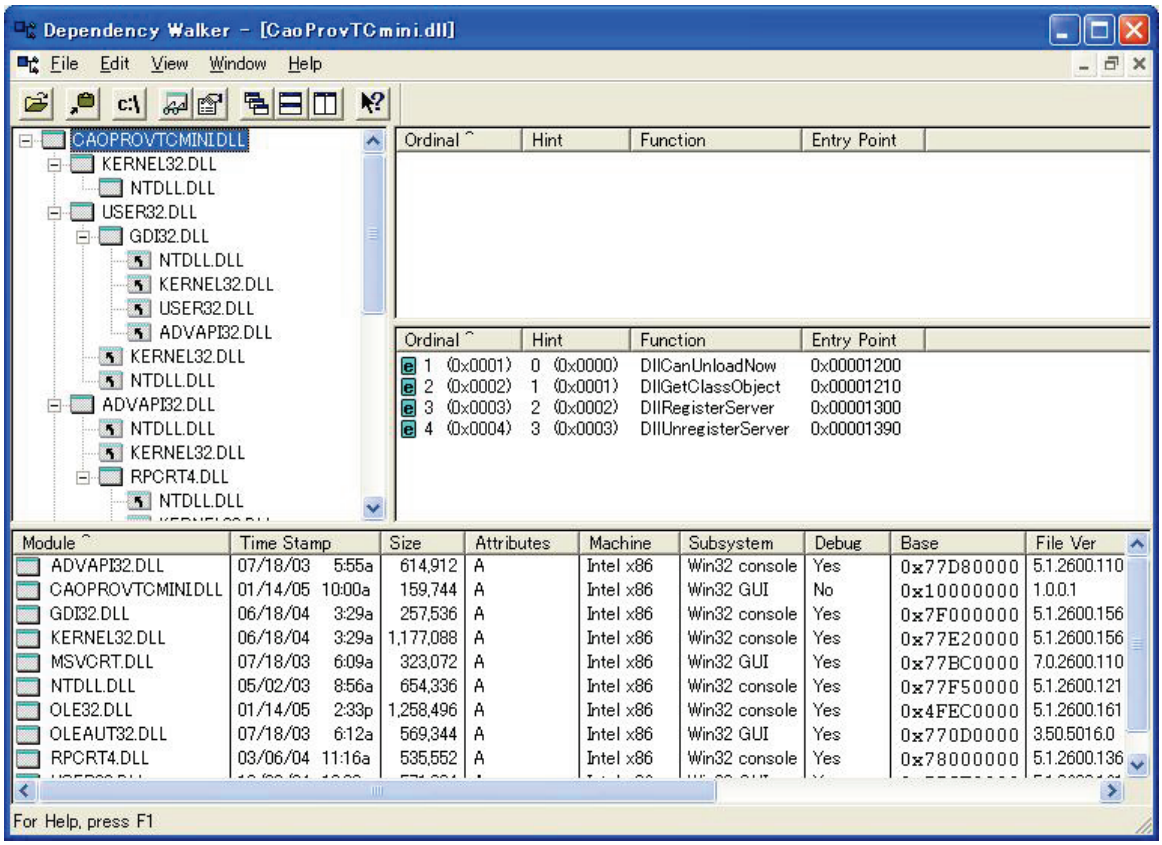


図 2-11 TCmini.DLL の Dependency Walker 画面

2.6.2. 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．

2.7. プロバイダサンプル

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[2]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.

表 2-6 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. ・ 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. ・ 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】 ・ メニューの[プロジェクト]→[プロパティ]を選択します. ・ 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. ・ 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”[3]を追加します.
	【対策方法2】 ・ メニューの[プロジェクト]→[プロパティ]を選択します. ・ 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. ・ 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】 ・ 警告の発生している関数名の先頭に“_”を追加する. 例 1) wcsnicmp() → _wcsnicmp() 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.

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

	例） DllCanUnloadNow @1 PRIVATE → DllCanUnloadNow PRIVATE
XML 関連のクラスで“error C2872”が発生する. [4]	【対策方法1】 ・ 以下の 2 行が記述してある箇所をコメントアウトする. #import "msxml4.dll" using namespace MSXML2;
	【対策方法2】 ・ 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. 例）ネームスペース名が“MSXML2”のとき IXMLDOMNodePtr pIChildNode; → MSXML2::IXMLDOMNodePtr pIChildNode;

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

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 CAO 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

CAO プロバイダでは，メソッドの引数として使用する Option 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 ORiN2/Cao/Include の OptionValue.h を参照ください．

```
GetOptionValue
(
"<オプション文字列>",          // オプション文字列
"<検索文字列>",                // 検索オプション名
"<要求型>",                    // オプション値のデータ型
"<結果値>"                     // オプション値
)
```

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 には，“test”という値を取得することができます．

以下に，オプション文字列から“Opt1”の値を取得する場合の例を示します．

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

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,Test2=Sample2

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	Sample2

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),Test2=(Sample2)

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
Test2	Sample2

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

CAO プロバイダでは，AddController メソッドの引数として使用する Option 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 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づれかの書式に従います．

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>]]

以下に，簡単な例を示します．ここでは，接続パラメータを“Conn=eth:192.168.0.1:8080”としています．その結果，IP アドレスとポート番号の値を取得する実装例です．

List 3-1 CCaoProvController.cpp – FinalConnect ()

```
#include "ConnectOption.h"

HRESULT CCaoProvController:: FinalConnect ()
{
	HRESULT hr;

	// 接続オプションの解析
	PARAM_CONN stRet;
	PARAM_CONN_ETH stEth = {INADDR_NONE, 0, INADDR_NONE, 9876};
				// 接続元未定，接続元ポート不定，接続先未定，接続先ポート 9876

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


	// 指定された接続パラメータ
	PVOID pConnArgs[4];
	pConnArgs[0] = &stRet.stEth.dwSrcIP;		// ローカル IP アドレス
	pConnArgs[1] = &stRet.stEth.dwSrcPort;		// ローカルポート番号
	pConnArgs[2] = &stRet.stEth.dwDestIP;		// IP アドレス
	pConnArgs[3] = &stRet.stEth.dwDestPort;		// ポート番号

	// 接続処理
	 :

	return hr;
}
```

3.4. エラー作成方法

CAO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 CAO 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 を定義する)を示します．

```
// ERROR CODE
//
// MessageId: E_SAMPLE
//
// MessageText:
//
//  これはテストです.
//
#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 CAO プロバイダに用意されているエラーコードは 0x80100000～0x8010FFFF（FACILITY CODE = 0x10）です．例の様にコメントをつけておくとコードの可視性が向上するでしょう．

表 3-5　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
0x80100000～0x8010FFFF	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. StringTable の一番下の空白列をダブルクリックします.

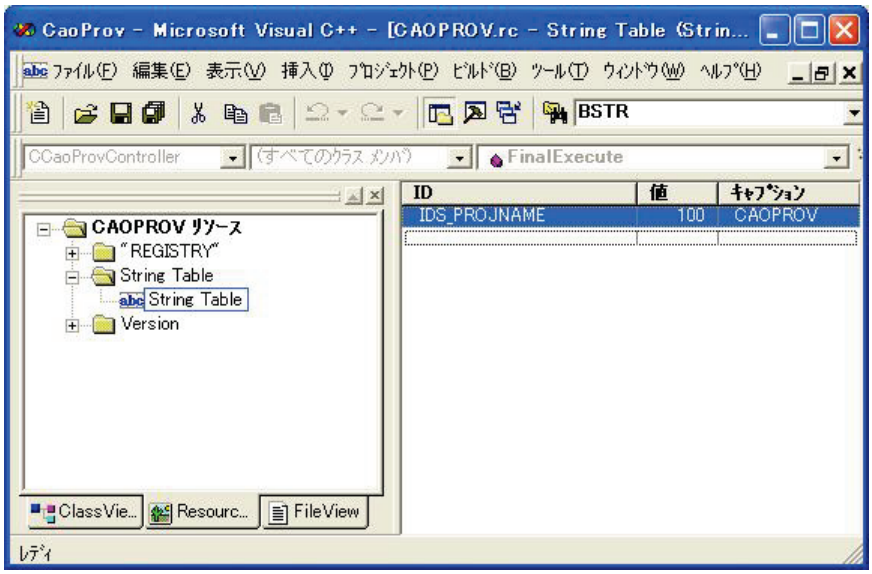


図 3-1 CAO プロバイダの String Table(初期状態)

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:“これはテストです. ”

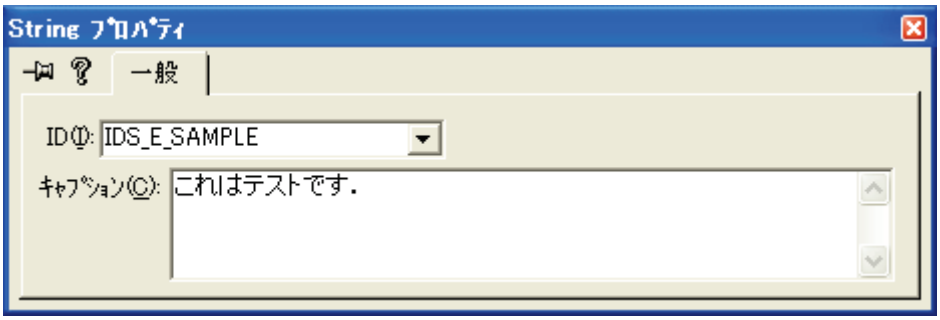


図 3-2 String のプロパティ編集画面

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.

ここでもう一度, 図 3-1 の Resource View を開きます. 追加したリソースの値フィールドが 1024～1535 に入っていることを確認してください. これは CAO プロバイダの実装部に許可されているリソースの値であり, もし異なっていれば以下の方法でリソースの値を割り当てなおす必要があります.

3. リソースファイル CAOPROV.rc をコンパイルします. すると Resource.h に作成したリソースが登録されます.
4. 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.

ここで②の手順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.

(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, 今回の場合は特に指定する必要はありません. (デフォルト値が割り当ててあるので省略することができます)

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.

3.5. メッセージイベントについて

CAO 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,                 // 生成するメッセージ
    long lNumber = 0,               // メッセージ ID
    VARIANT *vntData = NULL,        // 送信するデータ
    VARIANT *vntDateTime = NULL,    // 発生時間
    BSTR bstrReceiver = NULL,       // 受信先
    BSTR bstrSender = NULL,         // 送信元
    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 [5]．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 [6]．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 3-6 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．

[5] 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．
[6] 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．

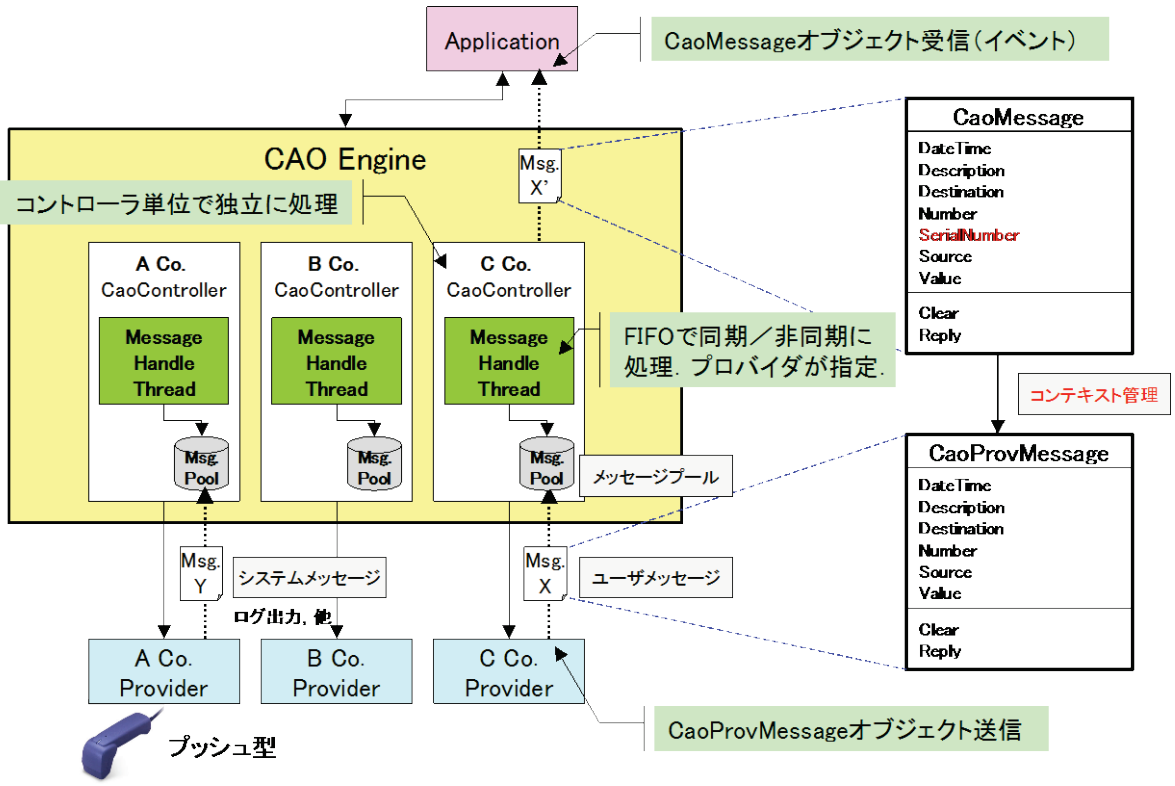


図 3-3 CAO メッセージメカニズム

図 3-3 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 3-6 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. Debug : 0x00020000 Info : 0x00020001 Warn : 0x00020002 Error : 0x00020003 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. 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. 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. 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. (例 “Test1, Test2”) 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 3-7 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);
        }
```

3.5.1. 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．

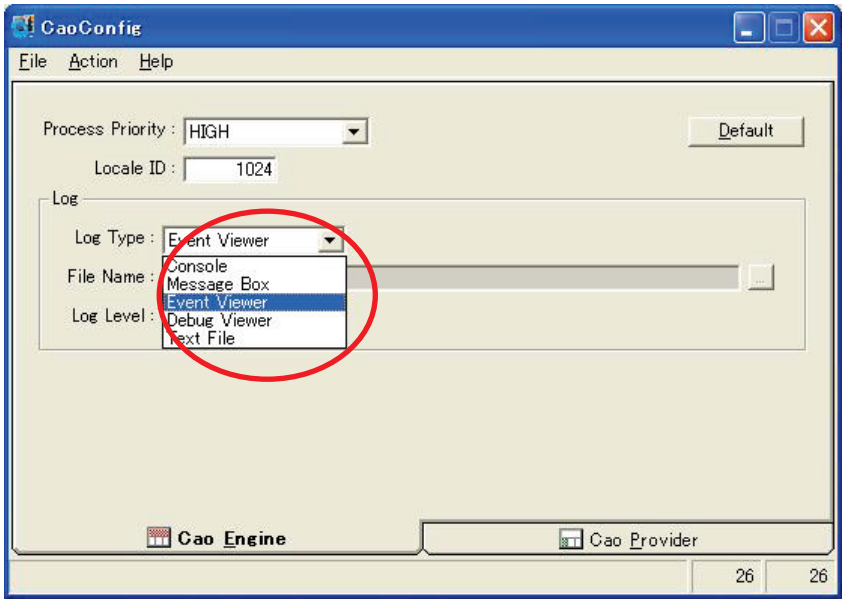


図 3-4 CAOConfig .exe の画面

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 3-4 参照)

表 3-8 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．

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.

3.5.2. 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.

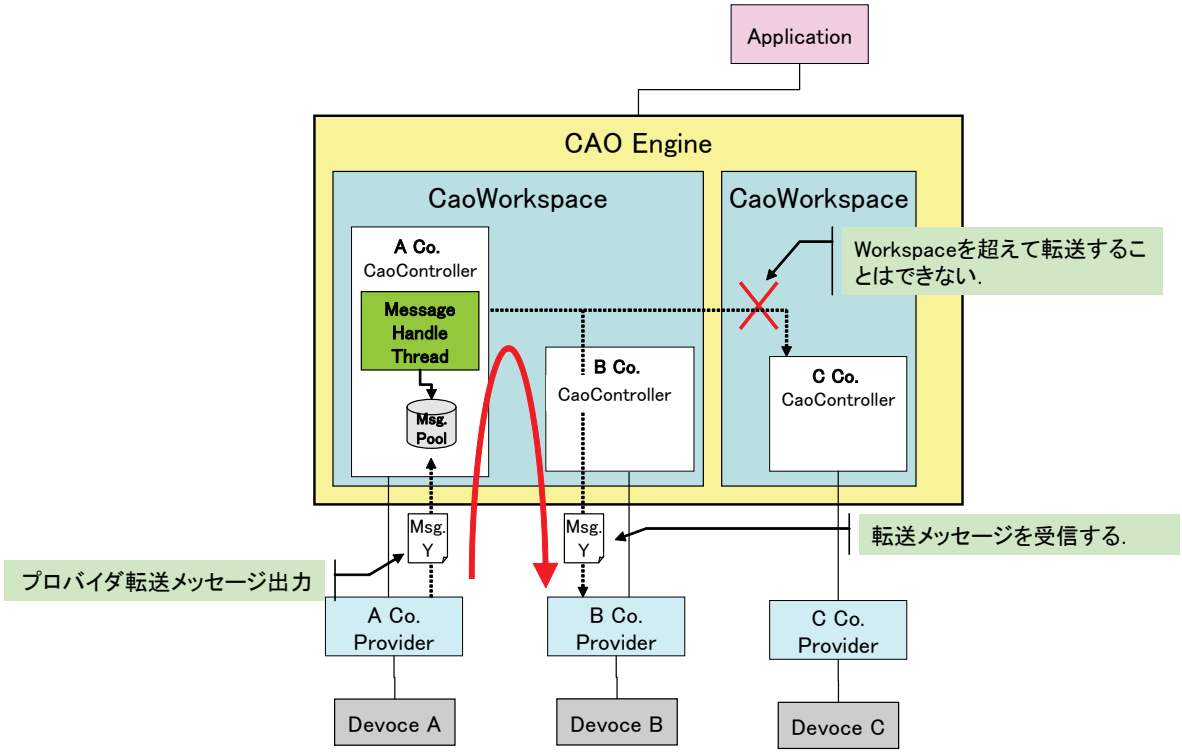


図 3-5 プロバイダ転送メッセージ

以下に送信側および受信側のサンプルを示します.

List 3-2 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 3-3　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;
}
```

3.5.3. 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 CAO 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 CAO 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 CAO 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 3-4 CcaprovController.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 CAO 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.

3.6. 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 CAO プロバイダに接続するための CAO プロバイダです. このとき接続する CAO 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のに必要な CAO 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.

3.6.1. 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 CAO 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 3.6.2)

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 3-9 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()

	CoCreateInstanceEx()
CaoProvCommand	CaoProvController::GetCommand()
CaoProvExtension	CaoProvController::GetExtension()
CaoProvFile	CaoProvController::GetFile() CaoProvFile::GetFile()
CaoProvRobot	CaoProvController::GetRobot()
CaoProvTask	CaoProvController::GetTask()
CaoProvVariable	CaoProvController::GetVariable() CaoProvExtension::GetVariable() CaoProvFile::GetVariable() CaoProvRobot::GetVariable() CaoProvTask::GetVariable()
CaoProvMessage	OnMassage 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 3-5 Sample.cpp

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

    // CAO 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;
    }
```

3.6.2. 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 3.6.2.1)

(2) プロバイダへの接続, 切断処理の実装. (参照 3.6.2.2)

(3) OnMessage イベント発生時の処理の実装. (参照 3.6.2.2)

(4) EventSink オブジェクト生成処理の追加. (参照 3.6.2.3)

これらの各手順については, 以下に順番に説明します.

3.6.2.1. 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"
```

CAO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 3-6 Sample.idl

```
// IEventSink Interface
interface IEventSink;
[
    object,
    uuid(B3E98B1A-0D28-42c8-84A1-1354891EA216),
    dual,
    helpstring("IEventSink Interface"),
    pointer_default(unique)
]
interface IEventSink : IDispatch
{
     [id(1), helpstring("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;
};
```

3.6.2.2. 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 3-7 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，3.6.2.1 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 3-8 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;
}
```

3.6.2.3. 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 3-9 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;
}
```

3.7. 通信クラスの利用方法

3.7.1. はじめに

ORiN2 SDK では, CAO 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. 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, RS-232C 通信をおこなうための CSerial クラスと, TCP/IP 通信をおこなうための CTCPServer クラス及び CTCPClient クラス(以下, TCP ソケットクラスとする), 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 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.

3.7.2. 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

テキストモードの時に，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．

以下に，CDevice クラスのツリー表示を示します．

図 3-6 CDevice のクラスツリー

また，CDevice クラスのメンバー一覧を表 3-10 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 3-10 CDevice のメンバー覧

	Member name	Explanation
	CDevice()	CDevice のコンストラクタ
	~CDevice()	CDevice のデストラクタ
	Connect()	バッファクリア．バッファクリアとエラークリアをおこないます．
	Disconnect()	接続処理．ターゲットとの接続をおこないます．
	Flush()	切断処理．ターゲットとの接続を切断します．
	GetDestination()	送信先情報を取得します．
	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． 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． 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． ReceiveData を呼び出し，データを加工します． 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．． 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． 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．
	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． 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． 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． Send から受け取ったデータをデバイスに送信します．

	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_bConnected	接続フラグ.
	m_dwHeaderLen	ヘッダ長.
	m_dwLocalBufferLen	ローカル受信バッファ長.
	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. 1～2.
	m_dwTotalTimeOut	文字列を送受信するまでの最大待ち時間.
	m_pcHeader	ヘッダコード.
	m_pcTerm	ターミネータコード.
	m_szLocalBuffer	ローカル受信バッファ.

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

3.7.3. シリアル通信クラス

3.7.3.1. 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, まず以下のようにコンストラクタを呼び出し, オブジェクトを生成します.

```
// CSerial オブジェクトの生成
CSerial *m_pSerial;
m_pSerial = (new CSerial);
```

(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.

```
PVOID pArgs[3];
CHAR cHeader[] = "¥0";
CHAR cTerm[] = "¥r";
```

```
DWORD dwMode = ST_MODE_TEXT;
pArgs[0] = cHeader;       // ヘッダコード
pArgs[1] = cTerm;         // ターミネータ
pArgs[2] = &dwMode;       // 動作モード

hr = m_pSerial->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(COM)ポートをオープンする

初期化処理で設定した項目に従って，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おこないます．

```
PVOID pArgs[1];
DWORD dwPortNo = 1;
pArgs[0] = &dwPortNo;      // シリアルポート番号

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

接続時には，PVOID 型の配列に以下のパラメータを指定します．

・ シリアルポート番号

送受信をおこなうシリアルポートの番号を指定します．例えば COM1 を使用するなら dwPortNo=1;と指定します．同様に COM2→2,COM3→3,…となります．

(4) 通信パラメータを設定する

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，タイムアウトなどの通信パラメータを設定します．

これらの通信パラメータの設定は SetState と GetState メソッドでおこないます．この二つのメソッドでは，通信デバイスに関する情報を格納する DCB 構造体を引数に持ちます．SetState 関数は，DCB 構造体を使ってデバイスの設定をおこないます．GetState 関数は，現在の構成を返します．使い方としては，まず DCB 構造体のインスタンスを用意して GetState でデフォルトの設定をこの DCB 構造体に読み込みます．続いて読み込んだ DCB 構造体のメンバ変数の値を変更し，変更した内容を有効にするために SetState にこの DCB 構造体を指定して通信パラメータを設定します．DCB 構造体については，MSDN や Microsoft のサイトを参照してください．

具体的な SetState メソッドと GetState メソッド，および DCB 構造体の使い方は次のようになります．下記の例では，ハンドシェイク用にハードウェアフロー制御を指定しています．RTS 信号は自動制御にはせずに(dcb.fDtrControl = DTR_CONTROL_ENABLE;)デバイスのオープン時に DTR 信号をアクティブにし，そのまま状態を保つようにしています．

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

// デフォルト値を代入する
dwBaudRate      = CBR_19200;       // 19200bps
dwDataBits      = 8;               // 8 ビット
dwParity        = NOPARITY;        // パリティなし
dwStopBits      = TWOSTOPBITS;     // ストップ 2 ビット

hr = m_pSerial->GetState(&dcb);    // 現在の通信パラメータ取得

if (SUCCEEDED(hr)) {
      dcb.BaudRate = dwBaudRate;                    // 速度
      dcb.ByteSize = (BYTE)dwDataBits;              // データ長
      dcb.Parity = (BYTE)dwParity;                  // パリティ
      dcb.StopBits = (BYTE)dwStopBits;              // ストップビット
```

```
        dcb.fOutX = true;                               // XON/XOFF を送信する
        dcb.fInX = true;                                // XON/XOFF を受信する
        dcb.fOutxCtsFlow = true;                        // CTS 信号を有効に
        dcb.fOutxDsrFlow = false;                       // DSR 信号を無効に
        dcb.fRtsControl = RTS_CONTROL_HANDSHAKE;        // RTS 信号を制御する
        dcb.fDtrControl = DTR_CONTROL_ENABLE;           // DTR 信号を ON する
        dcb.fDsrSensitivity = false;
        hr = m_pSerial->SetState(&dcb);                 // 通信パラメータの設定
}
```

データ送信時のタイムアウト時間(ミリ秒)を設定します.

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

(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.

```
#define LOCAL_BUFFER_MAX  16
HRESULT hr;                                     // リターンコード
BYTE lpszSendData[LOCAL_BUFFER_MAX +1];         // 送信データエリア
BYTE lpszReceiveData[LOCAL_BUFFER_MAX +1];      // 受信データエリア
DWORD dwSendLen;                                // 送信データ長
DWORD pdwRecvLen;                               // 受信データ長
```

・ **データ送信**

```
dwSendLen = 10;                                 // データ長
hr = m_pSerial->Send(       lpszSendData,       // 送信するデータのアドレス
                            dwSendLen,          // 送信データサイズ(0 は NULL 文字まで送信)
                            false);             // ヘッダとターミネータの付加しない(省略時 true)
```

・ **データ受信**

```
hr = m_pSerial->Receive(    lpszData,           // 受信データの格納先アドレス
                            10,                 // 最大受信サイズ
                            &pdwRecvedLen);     // 受信データ数
```

・ **データ送受信**

```
hr = m_pSerial->SendAndReceive(     lpszSendData,       // 送信データ文字列
                                    0,                  // 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
                                    10,                 //最大受信サイズ
                                    &pdwRecvLen,        // 受信データ数
                                    2);                 // リトライ回数 (省略可能)
```

(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 メソッドを呼び出した後に, delete でメモリの開放をおこないます. プログラムで COM ポートをオープンした場合は, アプリケーションを終了する前に必ずクローズの処理をおこなう必要があります. 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ります.

```
// COM ハンドルをクローズする
if (m_pSerial) {
	m_pSerial->Disconnect();
	delete m_pSerial;
	m_pSerial = NULL;
}
```

3.7.3.2. 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“0x8090000”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:0x80900002

3.7.4. TCP ソケットクラス

3.7.4.1. 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 CTCPClietn 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.

【サーバモードでの接続方法】

(1) オブジェクトの生成

CTCPServer クラスを利用するには, まず以下のようにコンストラクタを呼び出し, オブジェクトを生成します.

```
// CTCPServer オブジェクトの生成
CTCPServer * m_pTCPServer;
m_pTCPServer = (new CTCPServer);
```

(2) 初期化

必要なパラメータの設定をおこない, CTCPServer クラスを初期化します.

```
HRESULT hr;
PVOID pInitArgs[3];
CHAR cHeader[] = "¥0";
CHAR cTerm[] = "¥r";
DWORD dwMode = ST_MODE_TEXT;
pInitArgs [0] = cHeader;  // ヘッダコード
pInitArgs [1] = cTerm;    // ターミネータコード
pInitArgs [2] = &dwMode;  // 動作モード
```

```
hr = m_pTCPServer->Initialize(pInitArgs);

PVOID pConArgs[2];
DWORD dwMyIP = INADDR_NONE;
DWORD dwMyPort = 5006;
pConArgs[0] = &dwMyIP;    // ローカル IP アドレス
pConArgs[1] = &dwMyPort;  // ローカルポート番号
hr = m_pTCPServer->Connect(pConArgs);
```

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．

(3) 接続待ちをおこなう

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．

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

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】

(1) オブジェクトの生成

CTCPClient クラスを利用するには，まず以下のようにコンストラクタを呼び出し，オブジェクトを生成します．

```
// CTCPClient オブジェクトの生成
CTCPClient * m_pConnDevice;
m_pConnDevice = (new CTCPClient);
```

(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．

```
HRESULT hr;
PVOID pInitArgs[3];
CHAR cHeader[] = "¥0";
CHAR cTerm[] = "¥r";
DWORD dwMode = ST_MODE_TEXT;
pInitArgs [0] = cHeader;  // ヘッダコード
pInitArgs [1] = cTerm;    // ターミネータコード
pInitArgs [2] = &dwMode;  // 動作モード
hr = m_pConnDevice->Initialize(pInitArgs);
```

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.

(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, 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.

```
PVOID ConArgs[4];
DWORD dwMyIP = INADDR_NONE;
DWORD dwMyPort = 0;
DWORD dwSrvIPAdr = inet_addr("127.0.0.1");
DWORD dwSrvPortNo = 5006;
pConArgs[0] = &dwMyIP;              // ローカル IP アドレス
pConArgs[1] = &dwMyPort;            // ローカルポート番号
pConArgs[2] = &dwSrvIPAdr;          // 接続先 IP アドレス
pConArgs[3] = &dwSrvPortNo;         // 接続先ポート番号
hr = m_pConnDevice->Connect(pConArgs);
```

Connect 時には, 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.

【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)

(1) タイムアウトの設定

以下のような方法で, データ受信時のタイムアウト時間(ミリ秒)を設定することができます.

```
HRESULT hr;                                 // リターンコード
DWORD dwTimeout = 500;
hr = m_pConnDevice->SetTimeOut(dwTimeout);  //タイムアウト時間を 500msec に設定
```

(2) 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.

・ データ送信

```
HRESULT hr;                                 // リターンコード
DWORD dwDataLen = nLength;                  // データ長
hr = m_pConnDevice->Send(  lpszData,        // 送信するデータのアドレス
                           dwSendLen,       // 送信するデータの長さ（省略可能）
                           bHeadTerm);      // ヘッダとターミネータの付加（省略可能）
```

・ データ受信

```
HRESULT hr;                                 // リターンコード
#define LOCAL_BUFFER_MAX  16
DWORD dwBufSize;                            // 受信データ長
BYTE lpszData[ LOCAL_BUFFER_MAX +1 ];       // 受信データエリア
```

```
/// 受信データ処理
dwDataLen = 1L; // 1Byte 単位で受信する
hr = m_pConnDevice->Receive( lpszData,        // 受信データの格納先アドレス
                             dwBufSize,       // バッファサイズ
                             pdwRecvedLen);   // 受信データ数
```

・データ送受信

```
define LOCAL_BUFFER_MAX   16
HRESULT hr;                                              // リターンコード
DWORD dwRecvSize;                                        // 受信データ長
DWORD pdwRecvLen;                                        // 受信データ数
BYTE lpszSendData [ LOCAL_BUFFER_MAX +1 ];               // 受信データエリア
BYTE lpszReceiveData [ LOCAL_BUFFER_MAX +1 ];            // 送信データエリア
pdwRecvLen = 1L;
hr = m_pConnDevice->SendAndReceive(lpszSendData,    // 送信データ文字列
                                   0,               // 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
                                   dwRecvSize,      // lpszData のバッファサイズ
                                   pdwRecvLen,      // 受信データ数
                                   dwRetryCnt);     // リトライ回数（省略可能）
```

(3) 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．

```
// COM ハンドルをクローズする
if (pConnDevice) {
        pConnDevice->Disconnect();
        delete pConnDevice;
        pConnDevice = NULL;
}
```

3.7.4.2. 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：0x8090274D

3.7.5. CUDPSocket クラス

3.7.5.1. 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，まず以下のようにコンストラクタを呼び出し，オブジェクトを生成します．

```
// CUDPSocket オブジェクトの生成
CUDPSocket * m_pConnDevice;
m_pConnDevice = (new CUDPSocket);
```

(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．

```
HRESULT hr;
PVOID pInitArgs[3];
CHAR cHeader[] = "¥0";
CHAR cTerm[] = "¥r";
DWORD dwMode = ST_MODE_TEXT;
pInitArgs [0] = cHeader;  // ヘッダコード
pInitArgs [1] = cTerm;    // ターミネータコード
pInitArgs [2] = &dwMode;  // 動作モード
hr = m_pConnDevice ->Initialize(pInitArgs);
```

設定パラメータのヘッダコード，ターミネータコード，動作モードは CSerial クラスと同様になります．

(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．

```
PVOID ConArgs[2];
DWORD dwMyIP = INADDR_NONE;
DWORD dwMyPort = 0;
pConArgs[0] = &dwMyIP;    // ローカル IP アドレス
pConArgs[1] = &dwMyPort;  // ローカルポート番号
hr = m_pConnDevic ->Connect(pConArgs);
```

設定パラメータには，ローカルの IP アドレスとポート番号を指定します．

(4) 送信先の設定

送信先の設定を行います．

```
DWORD dwDist[2]
DWORD dwSrvIPAdr = inet_addr("127.0.0.1");
DWORD dwSrvPortNo = 5006;
dwDist [2] = &dwSrvIPAdr;          // 接続先 IP アドレス
dwDist [3] = &dwSrvPortNo;         // 接続先ポート番号
m_pConnDevice->SetDestination(dwDist);
```

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指定します．

(5) タイムアウトの設定

以下のような方法で，データ受信時のタイムアウト時間(ミリ秒)を設定することができます．

```
DWORD dwTimeout = 500;
hr = m_pConnDevice->SetTimeOut(dwTimeout);   //タイムアウト時間を 500msec に設定
```

(6) 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．

・ データ送信

```
DWORD dwDataLen = nLength;                    // データ長
HRESULT hr;                                   // リターンコード
hr = m_pConnDevice->Send(  lpszData,          // 送信するデータのアドレス
                           dwSendLen,         // 送信するデータの長さ（省略可能）
                           bHeadTerm);        // ヘッダとターミネータの付加（省略可能）
```

・ データ受信

```
#define LOCAL_BUFFER_MAX  16
HRESULT hr;                                   // リターンコード
DWORD dwBufSize;                              // 受信データ長
BYTE lpszData[ LOCAL_BUFFER_MAX +1 ];         // 受信データエリア
/// 受信データ処理
dwDataLen = 1L; // 1Byte 単位で受信する
hr = m_pConnDevice->Receive( lpszData,        // 受信データの格納先アドレス
                             dwBufSize,       // バッファサイズ
                             pdwRecvedLen);   // 受信データ数
```

・ データ送受信

```
define LOCAL_BUFFER_MAX   16
HRESULT hr;                                          // リターンコード
DWORD dwRecvSize;                                    // 受信データ長
DWORD pdwRecvLen;                                    // 受信データ数
BYTE lpszSendData [ LOCAL_BUFFER_MAX +1 ];           // 受信データエリア
BYTE lpszReceiveData [ LOCAL_BUFFER_MAX +1 ];        // 送信データエリア
pdwRecvLen = 1L;
hr = m_pConnDevice->SendAndReceive(          lpszSendData,    // 送信データ文字列
                                   0,                // 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
                                   dwRecvSize,       // lpszData のバッファサイズ
                                   pdwRecvLen,       // 受信データ数
                                   dwRetryCnt);      // リトライ回数（省略可能）
```

(7) 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．

```
// COM ハンドルをクローズする
if (m_pConnDevice) {
    m_pConnDevice->Disconnect();
    delete m_pConnDevice;
    m_pConnDevice = NULL;
}
```

3.7.5.2. 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：10061（接続拒否） → CAO API のエラー：0x8090274D

4. プロバイダ作成 Tips

ここでは CAO 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.

4.1. 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, ファイルクラスの変数(例: @VALUE)の get_Value をファイルオブジェクトの get_Value にリダイレクトしています.

(1) 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 4-1 CaoProvVariable.h

```
#ifndef __CAOPROVVARIABLE_H_
#define __CAOPROVVARIABLE_H_

#include "CaoProvVariableImpl.h"

class CCaoProvFile;                    // 親クラスの仮宣言

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

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

private:
     CCaoProvFile* m_pFile;            // 親クラスのポインタ格納先
};

#endif //__CAOPROVTASKVARIABLE_H_
```

(2) 変数クラスの初期化で親オブジェクトのポインタをメンバに格納する. 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 4-2 CaoProvVariable.cpp – FinalInitialize()

```
HRESULT CCaoProvVariable::FinalInitialize(PVOID pObj)
{
     switch (m_ulParentType) {
     case SYS_CLS_FILE:
          if (!_wcsicmp(m_bstrName, L"@Value")) {
               m_pFile = (CCaoProvFile*)pObj; // 親オブジェクトのポインタを格納
          }
          break;

     default:
          return E_NOTIMPL;
     }

     return S_OK;
}
```

(3) 親オブジェクトのメソッドを使用する.

以下の例では, CaoProvVariable::FinalGetValue()で呼び出している.

List 4-3 CaoProvVariable.cpp – FinalGetValue()

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

List 4-4 CaoProvVariable.cpp – FinalGetValue()

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

5. TCmini プロバイダの作成

本章では，CAO プロバイダの具体例の一つとして，東芝機械製小型プログラマブルコントローラ TCmini α TC3-02 を制御するための CAO プロバイダを紹介します．但し，TCmini プロバイダは ORiN Ver.2.0 の CAO プロバイダ仕様で規定されているすべてのインタフェースを実装したわけではありません．今回は TCmini α TC3-02 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)のみを第 2 章で説明した手順に従って実装します．

本書では，東芝機械製小型プログラマブルコントローラ TCmini α TC3-02 を『TCmini コントローラ』，今回作成する CAO 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 C++ 6.0 SP5 以上
- ORiN2 SDK 以上
- TCmini コントローラ本体
- C++言語および COM の基礎知識
- シリアル通信およびソケット通信の基礎知識

5.1. TCmini コントローラとは

5.1.1. 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．

5.1.1.1. データメモリの種類

表 5-1 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 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

5.1.1.2. データメモリの機能

表 5-2 データメモリの機能

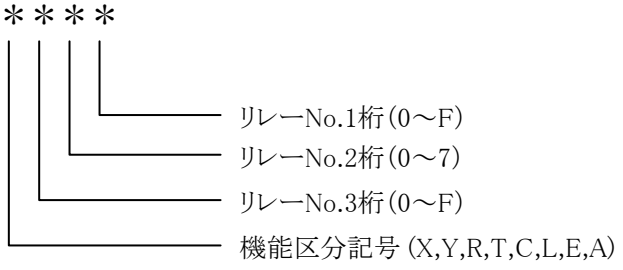
種類・使用状態		区分	機　　能
入出力リレー	入力リレー	X	入力専用リレーであり, フォトカプラ・キーボード・DISPSW 等の入力が接続されています.
	出力リレー	Y	出力専用リレーであり, リレー・パネルLED等の出力機器が接続されています.
	未認識リレー	Z	毎スキャンサイクルの入出力処理からは除外されるため, 内部リレーとして使用することができます.
内部リレー		R	外部に出力する必要のない一時記憶として使用することができます.
タイマ		T	現在値が 0 になるとタイマ接点が ON になります.
カウンタ		C	現在値が 0 になるとカウンタ接点が ON になります.
ラッチリレー		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. 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) 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 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. 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 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. 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. 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.

5.1.1.3. 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.

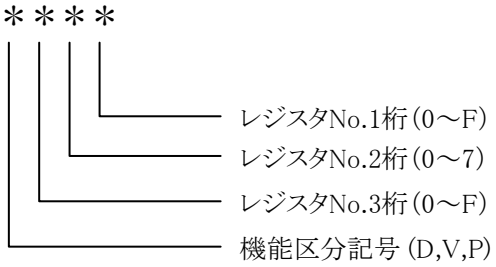
入出力リレーアドレスは実際のリレー装着と対応しますが, その他のリレーアドレスは物理的に存在しない装置(仮想装置)に対応しています. リレーアドレスは 1 点(1 ビット)ごとに付けられます.

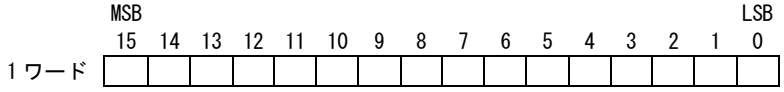


5.1.1.4. データレジスタアドレス

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.

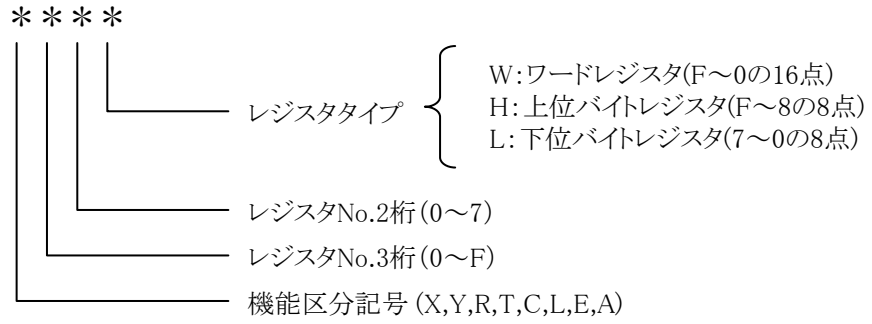
リレーアドレスが 1 点(1 ビット)単位なのに対してデータレジスタアドレスは 1 ワード(16 ビット)単位となります.



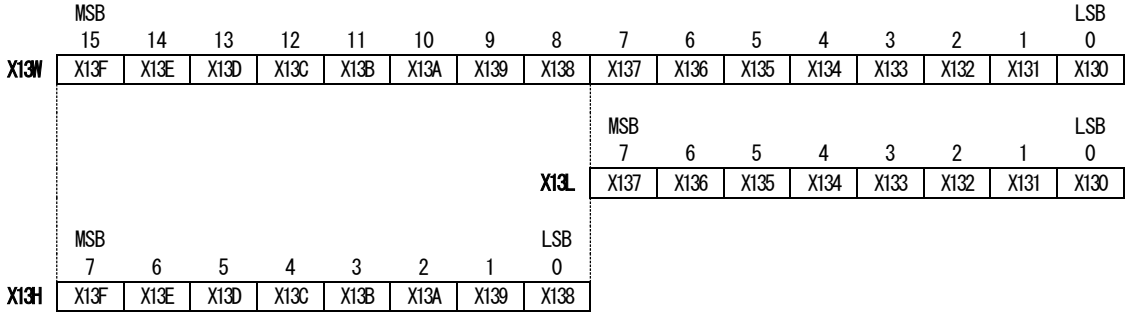


5.1.1.5. リレー領域のバイト/ワードレジスタアドレス

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



5.1.2. 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.

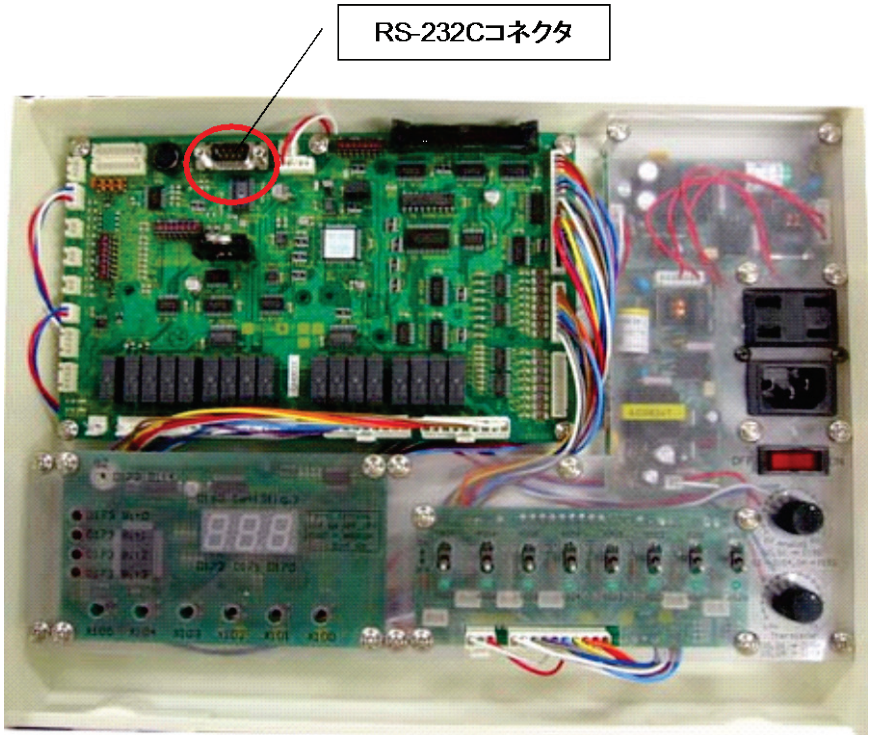


図 5-1 TCmini コントローラ

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．

通信速度(ボーレート)	9600bps/19200bps/38400bps
データビット数	8 ビット
パリティビット	なし
ストップビット数	2 ビット

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．

5.1.3. 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．

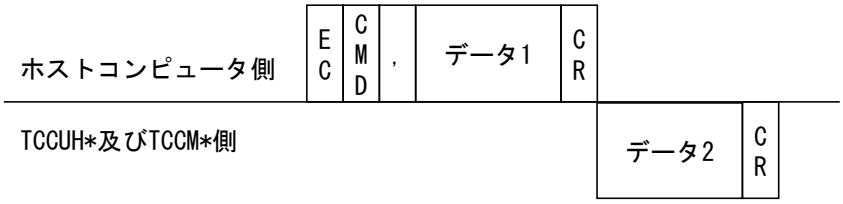


図 5-2 基本フォーマット

表 5-3 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の返信データである.

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.

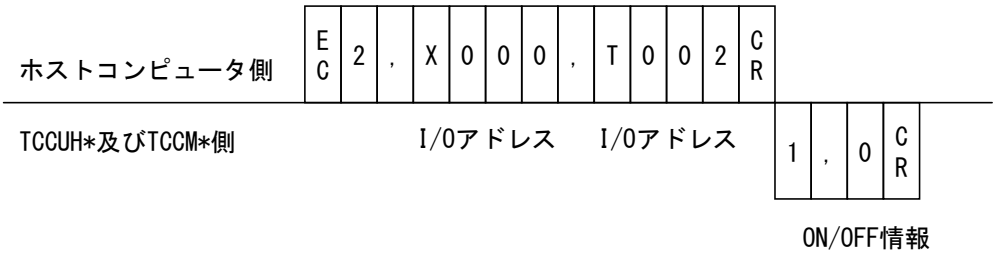
本節以降ではこの通信フォーマットの一部のコマンド仕様に関してのみ記載します. その他のコマンド仕様に関しては TCmini コントローラ付属の『RS232C ホスト通信コマンド説明書』に詳細が記載されているので必要に応じて参照してください.

5.1.3.1. 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

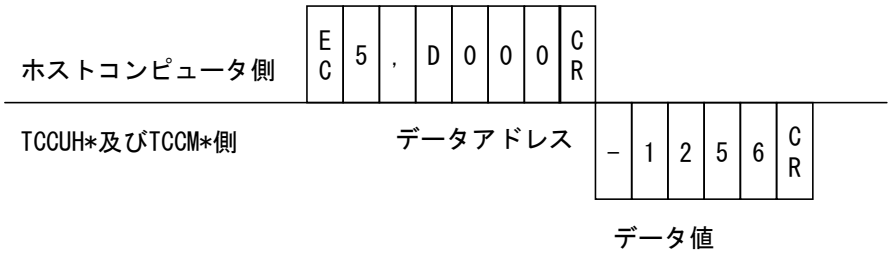
・ 区分コード(X,Y,R,L,S,T,C,E,A,G,H)は大文字で指定してください

・ 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

5.1.3.2. 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

5.1.3.3. 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

5.1.3.4. 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)

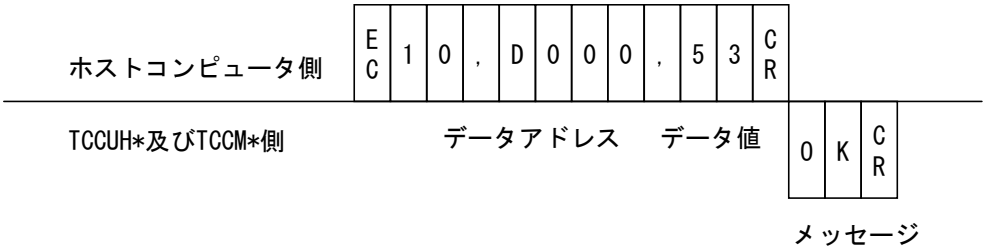
I/O アドレス(最大 42 接点)

- 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

5.1.3.5. 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

5.2. 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 CAO 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．ここでは TCmini プロバイダの仕様を決定し，次節でこの仕様に従ってプロバイダを実装します．

5.2.1. 接続パラメータ仕様

TCmini プロバイダでは CAOWorkspace::AddController 時に，通信用の接続パラメータを参照します．接続パラメータを省略した場合はデフォルトで“COM:1:19200:N:8:2”を指定した場合と同じ動作をします．以下に AddController の引数仕様を示します．

```
AddController
(
    "CaoProv.SIT.TCmini ",          // プロバイダ名．固定．
    "<マシン名>",                   // プロバイダの実行マシン名
    "<コントローラ名>",             // 任意の文字列
    "<接続パラメータ >"             // Tcmini コントローラ接続用パラメータ
)
```

接続パラメータの詳細については「TCmini プロバイダユーザーズガイド」を参照してください．

5.2.2. ユーザ変数仕様

CAOController クラスの AddVariable メソッドは，CAO の一般的意味では，変数にアクセスするためのメソッドです．TCmini プロバイダでは，このメソッドで作成した CAOVariable オブジェクトの Value プロパティの参照に TCmini α　TC3-02 の通信コマンドをそのままラップします．つまり，CAO のシステム変数にない情報でも，このユーザ変数を使ってアクセスすることができます．以下に，AddVariable の引数仕様を示します．

```
AddVariable
(
    "<区分コード><アドレス>"
)
```

<区分コード>　：＝　リレー(1bit)X,Y,Z,R,E,L,T,C,A またはレジスタ(16bits)D,P,V

<アドレス>　：＝　区分コードで指す変数のアドレス

アドレスは 3 文字固定の 16 進数値で指定します．

リレー変数は，語尾が‘W’,‘L’,‘H’の範囲指定が付加使用可能です．[7]

(例 1)　“X100”,“Y020”,“D150”,“T12A”

(例 2)　“X10W”・・・X100 から X10F までの範囲

(例 3)　“x10L”　・・・X100 から X107 までの範囲 [8]

(例 4)　“X10h”　・・・X108 から X10F までの範囲

例えば VB で X007 の現在値を取得するには次のようになります．

```
Dim iRes As Integer
Set CAOVar = CAOCtrl.AddVariable(“X007”)
iRes = CAOVar.Value ‘
```

接続パラメータの詳細については「TCmini プロバイダユーザーズガイド」を参照してください．

5.2.3. システム変数仕様

システム変数は CAOController クラスの AddVariable メソッドで指定する BSTR 型文字列です．

TCmini プロバイダでは下表に示す仕様でこのシステム変数を実装します．

表 5-4　TCmini プロバイダで実装するシステム変数一覧

変数名	データ型	説明	属性	
			get	put
@CURRENT_TIME	VT_DATE	コントローラ保有の現在時刻 [D125-D120]	○	×

[7] ワードサイズである D,P,V レジスタには’W’,’L’,’H’の範囲指定はできません．

[8] ユーザ変数名には大文字，小文字の区別はない．すべて大文字として処理されます．

@ERROR_DESCRIPTION	VT_BSTR	アラームメッセージ [通信コマンド: 1]	○	×
@MAKER_NAME	VT_BSTR	“Toshiba Machine”	○	×
@TYPE	VT_BSTR	“TCmini”	○	×
@VERSION	VT_BSTR	プロバイダのバージョン	○	×

5.3. TCmini プロバイダの実装

5.3.1. TCmini プロバイダプロジェクトの作成

TCminiプロバイダ用のスケルトン(雛形)プロジェクトはCAOプロバイダのプロジェクトウィザード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.dsw 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.dsw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 5-5 編集をおこなうファイル一覧

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

次項からはいよいよ具体的なプロバイダの実装をおこないます.

5.3.2. 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, “Devich.cpp”, “Serial.cpp”の 2 つのファイルをプロジェクトフォルダにコピーします [9].

次に, 図 5-3 に示すように,「File View」の「Source Files」を右クリックし,「ファイルをフォルダへ追加(F)」を選択し, “Devich.cpp”, “Serial.cpp”を追加してください.

[9] これらのファイルは ORiN2¥CAO¥Include¥のものをコピーしてください.

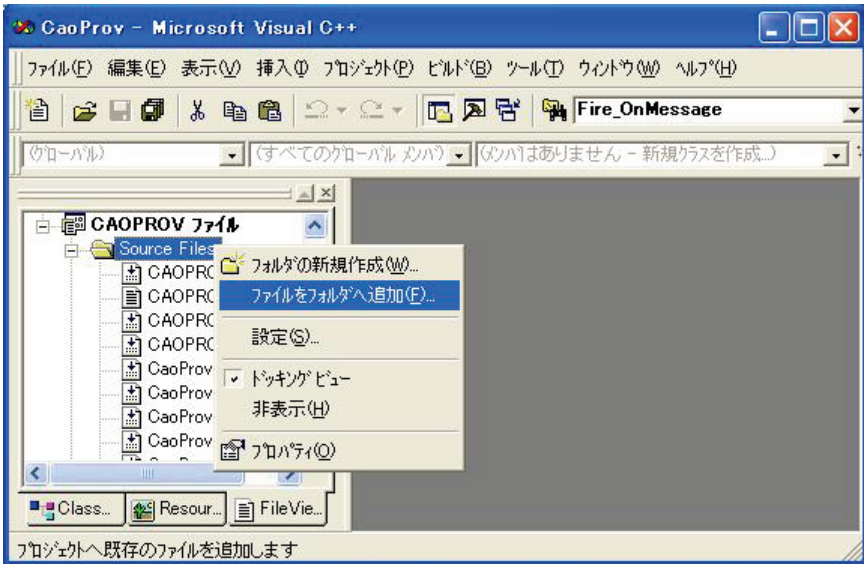


図 5-3 ファイルをフォルダへ追加画面

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．

List 5-1 StdAfx.h

```
private:
// ========== 各社追加部分はこれ以下に記述する. ==========
//↓[1]--------------------------------------------------------------------↓
#include "Serial.h"
//↑[1]--------------------------------------------------------------------↑
```

5.3.3. CCaoProvController クラスの実装

5.3.3.1. 必要なメソッドのオーバーライドをおこなう

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. HRESULT FinalGetVariableN**ames**(/*[in]*/ BSTR bstrOption,
   /*[out, retval]*/ VARIANT *pVal)

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) 上記に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()
{
        // CaoProvController オブジェクト共通の初期化処理などを行う．
        return E_NOTIMPL; // このクラスを実装するならば 返り値を S_OK にする．
}
*/
    ↓
HRESULT CCaoProvController::FinalInitialize()       //コメントをはずした後
{
        /// CaoProvController オブジェクト共通の初期化処理などを行う．
        return E_NOTIMPL;
}
```

5.3.3.2. 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．

追加するメンバ変数は，以下の 3 つとなります．

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
m_bConnected	:	接続フラグ
		TRUE　接続状態
		FALSE　未接続状態

List 5-2　CaoProvController.h

```
private:
// ========== 各社追加関数はこれ以下に記述する．==========
//↓[2]-------------------------------------------------------------------------↓
    CDevice*    m_pSerial;          // デバイスオブジェクトへのポインタ
    BOOL        m_bConnected;       // 接続フラグ
//↑[2]-------------------------------------------------------------------------↑
```

5.3.3.3. 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，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．

List 5-3 CaoProvController.cpp – FinalInitialize()

```
/**     初期化処理
 *
 *  CaoProvController オブジェクトが作成されて最初に呼ばれる関数である.
 *
 *      @retval   HRESULT
 *
 */

HRESULT CCaoProvController::FinalInitialize()
{
     // CaoProvController オブジェクト共通の初期化処理などを行う.
//↓[3]-------------------------------------------------------------------------↓
     m_pSerial = NULL;
     return S_OK;         // このクラスを実装するならば 返り値を S_OK にする.
//↑[3]-------------------------------------------------------------------------↑
}
```

5.3.3.4. ParseParameter メソッドを追加する

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の private な関数を追加します. 仕様としては下記の通りとします.

```
HRESULT ParseSerialParameter(
    BSTR bstrParam,                          // 接続パラメータ文字列
    PARAM_SERIAL *pParam            // 解析内容格納先
);
```

ParseParameter()関数は接続パラメータ(bstrParam)を BSTR 型で受取, COM ポートをオープン, 設定するために必要な値を構造体に代入します. まずは, 解析内容を格納するための構造体を追加します.

```
/** Connect()引数のパラメータ構造体 */
struct PARAM_SERIAL {
       DWORD dwPortNo;    /* COM ポートの番号 */
       DWORD dwBaudRate;  /* スピード(bps) */
       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 */
};
```

この構造体定義を StdAfx.h ヘッダファイルに追加します.

List 5-4 StdAfx.h – PARAM_SERIAL

```
// ========== 各社追加部分はこれ以下に記述する. ==========
//↓[1]-------------------------------------------------------------------------↓
#include "Serial.h"
//↑[1]-------------------------------------------------------------------------↑
#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.
```

```
//↓[4]-------------------------------------------------------------------↓
/** Connect()引数のパラメータ構造体 */
struct PARAM_SERIAL {
	/** COM ポートの番号		: COM1(1) ～ */
	DWORD dwPortNo;
	/**
	 * スピード(bps)	: 4800bps(4800), ～
	 * マクロ			: CBR_4800,CBR_9600,...
	 */
	DWORD dwBaudRate;
	/**
	 * パリティ		: 0 ～4 = None, Odd, Even, Mark, Space
	 * マクロ		: NOPARITY,ODDPARITY,EVENPARITY,MARKPARITY,SPACEPARITY
	 */
	DWORD dwParity;
	/** データビット数 : 4bits(4) ～ 8bits(8) */
	DWORD dwDataBits;
	/**
	 * ストップビット数		: 0 ～2 = 1,1.5,2 bit(s)
	 * マクロ : ONESTOPBIT,ONE5STOPBITS,TWOSTOPBITS
	 */
	DWORD dwStopBits;
	/**
	 * フロー制御 : 0～3
	 */
	DWORD dwFlow;
};

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

次に，ParseParameter 関数のプロトタイプ宣言を，CCaoProvController クラスのヘッダファイル(CaoProvController.h)に追加します. ここまでの CCaoProvController.h ファイルの変更を示します.

List 5-5 CaoProvController.h

```
/** @file CaoProvController.h
 *
 *  @brief CCaoProvController の宣言
 *
 *  @version    1.0
 *     @date            2003/8/8
 *     @author          DENSO WAVE
 *
 */

#ifndef __CAOPROVCONTROLLER_H_
#define __CAOPROVCONTROLLER_H_

#include "resource.h"       // メイン シンボル
#include "CAOPROVCP.h"
#include "CaoProvCommand.h"
#include "CaoProvVariable.h"
#include "CaoProvRobot.h"
#include "CaoProvFile.h"
#include "CaoProvTask.h"
#include "CaoProvExtension.h"
#include "CaoProvControllerImpl.h"
```

```
/**    CAO プロバイダの Controller クラス
 *
 *     デュアルインターフェースを持つ COM のクラス. 詳細は ICaoProvController を参照. <br>
 *     CAO プロバイダにおいて COM クライアントが唯一インスタンス作成可能なクラス. <br>
 *  インターフェースは ICaoProvControllerImpl クラスで実装されている.
 *
 */
class ATL_NO_VTABLE CCaoProvController :
     public ICaoProvControllerImpl<CCaoProvController, CCaoProvExtension, CCaoProvFile,
CCaoProvRobot, CCaoProvTask, CCaoProvVariable, CCaoProvCommand, CCaoProvMessage>
{
protected:
   HRESULT FinalInitialize();
// void FinalTerminate();

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

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

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

// ========== 各社追加関数はこれ以下に記述する. ==========

private:
// ========== 各社追加関数はこれ以下に記述する. ==========
//↓[5]--------------------------------------------------------------------↓
     HRESULT ParseSerialParameter(
         BSTR bstrParam,            // 接続パラメータ文字列
         PARAM_SERIAL *pParam       // 解析内容格納先
    );
//↑[5]--------------------------------------------------------------------↑
//↓[2]--------------------------------------------------------------------↓
     CSerial *m_pSerial;            // デバイスオブジェクトへのポインタ
     DWORD       m_dwMode;          // 動作モード
     BOOL        m_bConnected;      // 接続フラグ
//↑[2]--------------------------------------------------------------------↑
};

#endif //__CAOPROVCONTROLLER_H_
```

ParseParameter()関数の処理は以下のようになります.

List 5-6 CaoProvController.cpp – ParseParameter()

```
// ========== 各社追加関数はこれ以下に記述する. ==========
//↓[6]--------------------------------------------------------------------↓
/**    接続パラメータ解析
```

```
*
*  接続パラメータを解析しコントローラとの接続準備をします.
*
*  [com:<COM Port>[:<BaudRate>[:<Parity>:<DataBits>:<StopBits>]]]
*
*     [" ～ "]" は省略可能の意味                                    デフォルト
*     <COM Port>          := 1 ～                                  : 1
*     <BaudRate>          := 4800,9600,19200, ...                  : 19200
*     <Parity>            := N,O,E,M,S                             : N
*     <DataBits>          := 4 ～ 8                                : 8
*     <DataBits>          := 1, 1.5, 2                             : 2
*
*     @param   bstrParam:         [in] 接続パラメータ文字列
*     @param   pParam    :        [out] 解析内容格納先
*     @retval  HRESULT
*
*/
HRESULT CCaoProvController::ParseSerialParameter(BSTR bstrParam, PARAM_SERIAL *pParam )
{
    DWORD *pdwPortNo;     // COM ポートの番号: COM1(1) ～
     DWORD *pdwBaudRate;  // スピード(bps)   : 4800bps(4800), ～
                          //  マクロ : CBR_4800,CBR_9600,...
     DWORD *pdwParity;    // パリティ: 0 ～4 = None, Odd, Even, Mark, Space
                          //  マクロ : NOPARITY,ODDPARITY,EVENPARITY,MARKPARITY,SPACEPARITY
     DWORD *pdwDataBits;  // データビット数  : 4bits(4) ～ 8bits(8)
     DWORD *pdwStopBits;  // ストップビット数          : 0 ～2 = 1,1.5,2 bit(s)
                          //  マクロ : ONESTOPBIT,ONE5STOPBITS,TWOSTOPBITS
     DWORD *pdwFlow;      // フロー制御                : 0 ～3

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

     // デフォルト値を代入する
     *pdwPortNo = 1;
     *pdwDataBits = 8;
     *pdwBaudRate = CBR_19200;
     *pdwParity = NOPARITY;
     *pdwStopBits = TWOSTOPBITS;
     *pdwFlow = 0;

     int nlen;
     if (!bstrParam || (nlen = wcslen( bstrParam )) <= 0) {
       return S_OK; // すべて省略
     }

     // 先頭 4 文字が"com:"かどうか
     if (_wcsnicmp(L"com:", bstrParam, 4) != 0) {
          return E_INVALIDARG; // 不正な引数です！
     }

     WCHAR *pwcParam = bstrParam;

     // L':'の数を数えて指定されたパラメータの書式で設定値を取得する
     // ========================================================================
     int nTokens = 0;
     for (int i=0; i<nlen; i++) {
            if (pwcParam[i] == L':') {
                 nTokens++;
            }
       }

     int n;
```

```
int iPortNo = 1, iBaudRate = CBR_19200, iDataBits = 8, iFlow = 0;  // デフォルト値
float fStopBits = 2.0;
WCHAR wcParity = L'N';
WCHAR wcDumy;

pwcParam = &bstrParam[4]; // "com:"の次から位置
switch (nTokens) {
case 6: // com:<COM Port>:<BaudRate>:<Parity>:<DataBits>:<StopBits>:<Flow>
     n = swscanf(pwcParam, L"%d:%d:%c:%d:%f:%d%c", &iPortNo, &iBaudRate, &wcParity,
               &iDataBits, &fStopBits, &iFlow, &wcDumy);
     break;
case 5: // com:<COM Port>:<BaudRate>:<Parity>:<DataBits>:<StopBits>
     n = swscanf(pwcParam, L"%d:%d:%c:%d:%f%c", &iPortNo, &iBaudRate, &wcParity,
               &iDataBits, &fStopBits, &wcDumy);
          break;
case 2: // com:<COM Port>:<BaudRate>
n = swscanf(pwcParam, L"%d:%d%c", &iPortNo, &iBaudRate, &wcDumy);
          break;
  case 1: // com:<COM Port>
          n = swscanf(pwcParam, L"%d%c", &iPortNo, &wcDumy);
          break;
  default:
          return E_INVALIDARG; // 不正な引数です！
          break;
  }
  // 変換が正しかった個数がトークンの数分たけあるかをチェック
  if (n != nTokens) {
          return E_INVALIDARG; // 不正な引数です！
  }

  // 変換が必要なものは変換を行う
  // ========================================================================
  switch (wcParity) { // パリティ
  case L'N':
  case L'n':
          *pdwParity = NOPARITY;
          break;
  case L'O':
  case L'o':
          *pdwParity = ODDPARITY;
          break;
  case L'E':
  case L'e':
          *pdwParity = EVENPARITY;
          break;
  case L'M':
  case L'm':
          *pdwParity = MARKPARITY;
          break;
  case L'S':
  case L's':
          *pdwParity = SPACEPARITY;
          break;
  default:
          return E_INVALIDARG; // 不正な引数です！
          break;
  }

  int iStopBitsx10 =  (int)(fStopBits * 10.0);
  switch (iStopBitsx10) {// ストップビット
  case 10:
          *pdwStopBits = ONESTOPBIT;
          break;
  case 15:
          *pdwStopBits = ONE5STOPBITS;
          break;
```

```
        case 20:
                *pdwStopBits = TWOSTOPBITS;
                break;
        default:
                return E_INVALIDARG; // 不正な引数です！
                break;
        }

        // ==========================================================================
        // その他のチェックは行わない
        //                -> 不正なら実際に接続する際にエラーとなるため
        *pdwPortNo              = (DWORD)iPortNo;
        *pdwDataBits    = (DWORD)iDataBits;
        *pdwBaudRate    = (DWORD)iBaudRate;
        *pdwFlow        = (DWORD)iFlow;

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

5.3.3.5. FinalConnect()の実装

次は, FinalConnect()メソッドの実装をおこないます. このメソッドは, CaoWorkspace::AddController 実行時に呼び出されます.

本処理では, まず, AddController メソッドの第 4 引数(接続パラメータ m_bstrPara)で指定された“Conn”オプションの値を GetOptionValue 関数で取得します. この取得した接続パラメータを前項で定義した ParseParameter()関数に渡し, 通信パラメータ(pParam)を取得ます. 続いて COM ポートをオープン(Connect)してデフォルトの DCB 構造体(dcb)の値を取得(GetState)します. DCB 構造体(dcb)の必要な値を変更して, 通信デバイスの設定(SetState)をおこなっています.

List 5-7 CaoProvController.cpp – FinalConnect()

```
/**     プロバイダ接続開始処理
 *
 *  プロバイダ内で最初に呼ばれる関数である.
 *
 *      @retval   HRESULT
 *
 */

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

    // 接続パラメータの取得
    CComBSTR bstrPara;
    CComVariant vntOptVal;
    hr = GetOptionValue(m_bstrOption, L"Conn", VT_BSTR, &vntOptVal);
    if (FAILED(hr) || vntOptVal.vt != VT_BSTR) {
        return E_INVALIDARG;
    }
    bstrPara = vntOptVal.bstrVal;

    // 接続パラメータを解析して設定情報を取得する
```

```
    PARAM_SERIAL SerialPara;
    m_pSerial = new CSerial;
    hr = ParseSerialParameter(bstrPara, &SerialPara);
    if (FAILED(hr)) {
         return E_INVALIDARG;
    }

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

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

              if (SUCCEEDED(hr)) {
                   dcb.BaudRate = SerialPara.dwBaudRate;          // 速度
                   dcb.ByteSize = (BYTE)SerialPara.dwDataBits;    // データ長
                   dcb.Parity = (BYTE)SerialPara.dwParity;        // パリティ
                   dcb.StopBits = (BYTE)SerialPara.dwStopBits;    // ストップビット
                   if (SerialPara.dwFlow && FLOW_XINOUT) {
                        dcb.fOutX = true;                         // XON/XOFF を送信する
                        dcb.fInX = true;                          // XON/XOFF を受信する
                   }
                   else {
                        dcb.fOutX = false;                        // XON/XOFF を送信しない
                        dcb.fInX = false;                         // XON/XOFF を受信しない
                   }

                   if (SerialPara.dwFlow && FLOW_HARDWARE) {
                        dcb.fOutxCtsFlow = true;                  // CTS 信号を有効に
                        dcb.fOutxDsrFlow = false;                 // DSR 信号を無効に
                        dcb.fRtsControl = RTS_CONTROL_HANDSHAKE; // RTS 信号を制御する
                        dcb.fDtrControl = DTR_CONTROL_ENABLE;     // DTR 信号を ON する
                   }
                   else {
                        dcb.fOutxCtsFlow = false;                 // CTS 信号を無効に
                        dcb.fOutxDsrFlow = false;                 // DSR 信号を無効に
                        dcb.fRtsControl = RTS_CONTROL_ENABLE;     // RTS 信号を ON する
                        dcb.fDtrControl = DTR_CONTROL_ENABLE;     // DTR 信号を ON する
                   }
                   dcb.fDsrSensitivity = false;
                   hr = ((CSerial*)m_pSerial)->SetState(&dcb);
              }
         }
    }
    if (FAILED(hr)) {
         FinalDisconnect();
         return hr;
    }

    // タイムアウト時間の設定
    hr = GetOptionValue(m_bstrOption, L"TimeOut", VT_I4, &vntOptVal);
    if (FAILED(hr)) {
         return E_INVALIDARG;
    }
    if (vntOptVal.vt == VT_I4) {
         hr = m_pSerial->SetTimeOut((DWORD)vntOptVal.lVal);
    }

    return hr;
```

```
//↑[7]--------------------------------------------------------------------↑
}
```

5.3.3.6. FinalDisconnect()の実装

続いて，FinalDisconnect()メソッドの実装をおこないます．このメソッドは，CaoControllers::Remove 時に呼び出されます．このメソッド内では，シリアル接続の切断処理とオブジェクトの開放処理をおこないます．

List 5-8 CaoProvController.cpp – FinalDisconnect()

```
/**    プロバイダ切断処理
 *
 *    @retval  HRESULT
 *
 */
HRESULT CCaoProvController::FinalDisconnect()
{
      // プロバイダの切断処理をここで行う．
//↓[8]--------------------------------------------------------------------↓
    // 切断
    m_bConnected = FALSE;
    if (m_pSerial) {
        m_pSerial->Disconnect();
        delete m_pSerial;
        m_pSerial = NULL;
    }
//↑[8]--------------------------------------------------------------------↑
    return S_OK; // このクラスを実装するならば 返り値を S_OK にする．
}
```

5.3.3.7. 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デバイスオブジェクトへののポインタが必要になります．そこで，CaoVariable クラスから COM 通信がおこなえるように，シリアル接続オブジェクトを取得するメソッドを追加します．

List 5-9 CaoProvController.cpp – GetSerial()

```
//↓[9]--------------------------------------------------------------------↓
/**    シリアル接続オブジェクトの取得
 *
 *    @param   ppSerial :        [out] シリアル接続オブジェクト
 *    @retval  HRESULT
 *
 */
HRESULT CCaoProvController::GetSerial(CSerial **ppSerial)
{
    *ppSerial = m_pSerial;
    return S_OK;
}
//↑[9]--------------------------------------------------------------------↑
```

5.3.3.8. FinalGetVariableNames()の実装

表 5-4 に示したように, TCmini プロバイダでは, 5 つのシステム変数を取得することができるようにします. そこでまず, システム変数名を以下のように定義します.

```
// システム変数
#define CS_CURRENT_TIME           L"@CURRENT_TIME"
#define CS_ERROR_DESCRIPTION      L"@ERROR_DESCRIPTION"
#define CI_MAKER_NAME             L"@MAKER_NAME"
#define CI_TYPE                   L"@TYPE"
#define CI_VERSION                L"@VERSION"
```

ここで, “CI_MAKER_NAME”, “CI_TYPE”, “CI_VERSION”の 3 つの変数は, 常に定数を返すシステム変数であるので, その値を以下のように定義します.

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

ここで返す BSTR 型の文字列の前の‘**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.

以上の定義を StdAfx.h に追加します.

List 5-10 StdAfx.h – マクロ定義

```
// ========== 各社追加部分はこれ以下に記述する. ==========
//↓[1]------------------------------------------------------------------------↓
#include "Serial.h"
//↑[1]------------------------------------------------------------------------↑

#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.

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

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

この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.

List 5-11 CaoProvController.cpp – FinalGetVariableNames()

```
/**    変数名リストの取得
 *
 *     @param   bstrOption        :        [in] オプション
 *     @param   pVal              :        [out] 変数名リスト
 *     @retval  HRESULT
 *
 */

HRESULT CCaoProvController::FinalGetVariableNames(BSTR bstrOption, VARIANT *pVal)
{
     // 変数名リストを文字列の SAFEARRAY として作成.
     // そのポインタを VARIANT(pVal->vt = VT_BSTR|VT_ARRAY, pVal->parray = xxx)に格納する.
//↓[11]-------------------------------------------------------------------↓
     pVal->vt = VT_ARRAY | VT_BSTR;
     SAFEARRAY* psa;
     SAFEARRAYBOUND bounds = {5, 0};
     psa = SafeArrayCreate(VT_BSTR, 1, &bounds);

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

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

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

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

     return S_OK;
}
```

5.3.4. CCaoProvVariable クラスの実装

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.

5.3.4.1. FinalInitialize()の実装

まずは, FinalInitialize()メソッドの実装をおこないます.

このメソッドは, CCaoProvController::AddVariable()メソッド実行時に呼び出され, CCaoVariable クラスの初期化をおこないます.

ここで, CaoProvVariable クラスからシリアル通信がおこなえるように, デバイスオブジェクトへのポインタ m_pSerial を CCaoProvVariable クラスのメンバに追加します. m_pSerial 変数の初期化は CaoProvVariable オブジェクトが生成されたときに呼び出される CCaoProvVariable::FinalInitialize()メソッドでおこないます.

```
private:
Cdevice* m_pSerial;        // デバイスオブジェクトへのポインタ
```

CCaoProvController::AddVariable()では, 引数として, 変数名を指定します. そこで, 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. この情報をクラス内で保持します.

ユーザ変数名に関しては, 以下の情報が必要となります.

・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

これを扱うための型を下記の通りに定義します.

```
// 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;
```

・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.

・システム変数の識別情報

List 5-12 CaoProvVariable.h

```
/** @file CaoProvVariable.h
 *
 *  @brief CCaoProvVariable の宣言
 *
 *  @version    1.0
 *     @date            2003/8/8
 *     @author          DENSO WAVE
 *
 */

#ifndef __CAOPROVVARIABLE_H_
#define __CAOPROVVARIABLE_H_
```

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

class CCaoProvController; // 前方宣言
                          // RaoProvController.h ->#include "RaoProvVariable.h"のため
//↓[12]-------------------------------------------------------------------------↓
class CCaoProvController;           // 前方宣言
#define CMD_EC 0x1B                 // コマンドの先頭コード
#define CMD_CR 0x0D                 // コマンドの終了コード
                                    // TCmini のコマンドは下記の通り
                                    //       <EC> + <CMD> + ',' + <データ> + <CR>
                                    //  CMD = '0'～'31'までの ASCII 数字
#define CMD_ADD_LEN 4               // アドレスの長さ
#define VAR_COMMAND_MAX 256         // コマンドの最大文字数

// 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;
//↑[12]-------------------------------------------------------------------------↑

/**  TCmini プロバイダの Variable クラス
 *
 *  デュアルインターフェースを持つ COM のクラス. 詳細は ICaoProvVariable を参照. <br>
 *  CCaoProvController クラス,CCaoProvExtension クラス,CCaoProvFile クラス,<br>
*  CCaoProvRobot クラス及び CCaoProvTask クラスから呼び出されるクラス. <br>
 *  外部からインスタンスの作成は不可能. <br>
 *  インターフェースは ICaoProvVariableImpl クラスで実装されている. <br>
 *  通信コマンドの作成し, 送受信を行う.
 *
 */
class ATL_NO_VTABLE CCaoProvVariable :
     public ICaoProvVariableImpl<CCaoProvVariable>
{
protected:
       HRESULT FinalInitialize(PVOID pObj);
//     void FinalTerminate();

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

// ========== 各社追加関数はこれ以下に記述する. ==========

private:
// ========== 各社追加関数はこれ以下に記述する. ==========
//↓[13]-------------------------------------------------------------------------↓
       CDevice *m_pSerial;                  // デバイスオブジェクトへのポインタ
       VAR_DATA_TYPE m_usrDataType;         // データ型を格納する変数
       BSTR m_bstrUpperName;                // 大文字のみの変数名
       DWORD m_dwSysID;                     // システム変数 ID
//↑[13]-------------------------------------------------------------------------↑
};

#endif //__CAOPROVTASKVARIABLE_H_
```

続いて, ユーザ変数名の解析処理に関して説明します. この処理は最初に一度だけ行えばよいので

CCaoProvVariable::FinalInitialize() でおこないます.

このメソッドでは, まず引数として与えられた変数が, ユーザ変数かシステム変数かを判別します.

システム変数名であった場合は, 適切な変数名であるかをチェックします. 適切で合った場合には, m_dwSysID にシステム ID を割り当てます.

ユーザ変数であった場合は, 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 5-13 CaoProvVariable.cpp – FinalInitialize()

```
/**    初期化処理
 *
 *  CaoProvVariable オブジェクトが作成されて最初に呼ばれる関数である.
 *  変数名は, "<区分コード><アドレス>"で指定しなければならない.
 *
 *     @param   pObj     :        [in] 親オブジェクト
 *     @retval  HRESULT
 *
 */
HRESULT CCaoProvVariable::FinalInitialize(PVOID pObj)
{
//↓[14]--------------------------------------------------------------------↓
    CCaoProvController* pCaopCtrl = NULL;

    // 親オブジェクトの判定
    switch (m_ulParentType) {
    case SYS_CLS_CONTROLLER:
        pCaopCtrl = (CCaoProvController*)pObj;
        break;
    default:
    return E_NOTIMPL;
    }

    // シリアルデバイスオブジェクトの取得
    HRESULT hr = pCaopCtrl->GetSerial(&m_pSerial);
    if (FAILED(hr)) {
        return hr;
    }

    // 変数名のチェック
    if (m_bSystem) {              // システム変数の場合
        // 実装していないシステム変数はエラーを返す.
        if (!_wcsicmp(m_bstrName, VAR_CURRENT_TIME$)) {
            m_dwSysID = VAR_CURRENT_TIME;
        }
        else if (!_wcsicmp(m_bstrName, VAR_ERROR_DESCRIPTION$)) {
            m_dwSysID = VAR_ERROR_DESCRIPTION;
        }
        else if (!_wcsicmp(m_bstrName, VAR_MAKER_NAME$)) {
            m_dwSysID = VAR_MAKER_NAME;
        }
        else if (!_wcsicmp(m_bstrName, VAR_TYPE$)) {
            m_dwSysID = VAR_TYPE;
```

```
        }
        else if (!_wcsicmp(m_bstrName, VAR_VERSION$)) {
            m_dwSysID = VAR_VERSION;
        }
        else {
            return E_INVALIDARG;
        }
    }
    else {                          // ユーザ変数の場合

        // 大文字に変換
        CComBSTR bstrTemp(m_bstrName);
        hr = bstrTemp.ToUpper();
        if (FAILED(hr)) {
            return hr;
        }
        int ilen = bstrTemp.Length();
        m_bstrUpperName = bstrTemp.Detach();

        // 先頭の文字でデータタイプを判定する.
        //ビット型で語尾に'W','L','H'が付く場合はワード単位になる

        // 先頭一文字目を判定
        switch (m_bstrUpperName[0]) {
            case L'X':
            case L'Y':
            case L'Z':
            case L'R':
            case L'E':
            case L'L':
            case L'T':
            case L'C':
            case L'A':
                // アドレス文字列長の検査
                if (ilen != CMD_ADD_LEN) {
                    return E_INVALIDARG;
                }
                // 最後の文字を判定
                switch (m_bstrUpperName [ilen-1]) {
                case L'W':
                case L'H':
                case L'L':
                    m_usrDataType = VAR_TYPE_WORD;
                    break;
                default:
                    m_usrDataType = VAR_TYPE_BIT;
                    break;
                }
                break;
            case L'D':
            case L'P':
            case L'V':
                // アドレス文字列長の検査
                if (ilen != CMD_ADD_LEN) {
                    return E_INVALIDARG;
                }
                m_usrDataType = VAR_TYPE_WORD;
                break;
            // それ以外
            default:
                // 汎用コマンドとしてそのままターゲットへ指定文字列を送信する.
                // ex. "2,X000,Y007,T002"
                //   0x1B+"2,X000,Y007,T002"+0x0D としてターゲットへ送信する.
                //   応答が "1,0"+0x0D の場合, 文字列"1,0"を返す.
                m_usrDataType = VAR_TYPE_UNKNOWN;     // 不明な型
                break;
```

```
        }
    }
    return S_OK;
//↑[14]----------------------------------------------------------------↑
}
```

5.3.4.2. FinalGetValue()の実装

次に FinalGetValue()メソッドの実装をおこないます.

このメソッドでは, まず, 現在扱っている変数がシステム変数かユーザ変数かを判別します. システム変数であった場合は, GetSystemValue 関数を呼び出します. この関数の説明については, 後述します.

ユーザ変数であった場合は, ユーザ変数が VAR_TYPE_BIT なら＜CMD＞“2”(0x32)コマンドの“2,”を, VAR_TYPE_WORD なら＜CMD＞“5”(0x35)コマンドの“5,”をそれぞれ先頭に挿入したコマンド用文字列を作します. ユーザ変数が VAR_TYPE_UNKNOWN ならそのままコピーしたコマンド用文字列を作成します.

次に作成したコマンド用文字列の先頭に<EC>コードを付加して送信コマンドを作成します.

次に送信コマンドを TCmini コントローラに送信してその応答を受信します. (SendAndReceive)

このときターミネータの<CR>コードは, Serial 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 なら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.

List 5-14 CaoProvVariable.cpp – FinalGetValue()

```
/**    変数の値の取得
 *
 *  ICaoVariable::get_Value()の実体となる.
 *
 *    @param   pVal     :        [in]  変数値
 *    @retval  HRESULT
 *
 */
HRESULT CCaoProvVariable::FinalGetValue(VARIANT *pVal)
{
//↓[15]----------------------------------------------------------------↓
    HRESULT hr = S_OK;
    int iCmdBufLen = 0;

    // システム変数の場合
    if (m_bSystem) {
        return GetSystemValue(pVal);
    }

    // 送信コマンド文字列の作成
    // ---------
    // ビット単位の場合
        // BIT 単位の読み出しには [2,]  →  応答は '0'/'1'
    // ワード単位の場合
        // WORD 単位の読み出しには [5,]  →  応答は 10 進数
    // それ以外
        // 汎用コマンドとしてそのままターゲットへ指定文字列を送信する.
        // ex. "2,X000,Y007,T002"
```

```
        //   0x1B+"2,X000,Y007,T002"+0x0D としてターゲットへ送信する.
        //   応答が "1,0"+0x0D の場合, 文字列"1,0"を返す.

    WCHAR szCmdBuf[VAR_COMMAND_MAX];     // 送信バッファ
    szCmdBuf[iCmdBufLen++] = CMD_EC;     // コマンド開始コード
    switch (m_usrDataType) {
    case VAR_TYPE_BIT:
        szCmdBuf[iCmdBufLen++] = L'2';     // <CMD> = '2'
        szCmdBuf[iCmdBufLen++] = L',';
        break;
    case VAR_TYPE_WORD:
        szCmdBuf[iCmdBufLen++] = L'5';     // <CMD> = '5'
        szCmdBuf[iCmdBufLen++] = L',';
        break;
    case VAR_TYPE_UNKNOWN:
        break;
    default:
        return E_NOTIMPL;     // データタイプエラー
    }

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

    // コマンド送信と応答受信
    WCHAR szDataBuf[VAR_COMMAND_MAX];
    DWORD dwDataLen;
    hr = m_pSerial->SendAndReceive((LPBYTE)szCmdBuf,
                                   0,
                                   (LPBYTE)szDataBuf,
                                   VAR_COMMAND_MAX * sizeof(WCHAR),
                                   &dwDataLen);
    if (FAILED(hr)) {
        return hr;
    }

    // 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 の vt へ VT_I2 を代入
        pVal->iVal = _wtoi(szDataBuf); //int 型に変換し, pVal の vt へ代入

        // 値変換失敗の時
        if (pVal->iVal == 0 && _wcsicmp(szDataBuf, L"0")) {
            bError = TRUE;
        }
        break;
    case VAR_TYPE_UNKNOWN:
        bError = TRUE;
        break;
    }

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

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

5.3.4.3. GetSystemValue()の実装

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，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．

1. "@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．

2. "@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．

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

4. "I_TYPE"への対応
   switch-case 構文の CI_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．

5. "I_VERSION"への対応
   switch-case 構文の CI_TYPE に対応しているので，仕様の通り，"1.0.0"を BSTR 型で返すように処理を追加します．

以上の処理を行ったコードを下記に示します．

List 5-15 CaoProvVariable.cpp – GetSystemValue()

```
// ========== 各社追加関数はこれ以下に記述する．==========
//↓[16]----------------------------------------------------------------↓
/** システム変数の値の取得
 *
```

```
 *     @param   pVal     :        [out]  変数値
 *     @retval  HRESULT
 *
 */
HRESULT CCaoProvVariable::GetSystemValue(VARIANT *pVal)
{
     HRESULT hr = S_OK;
     WCHAR szCmdBuf[VAR_COMMAND_MAX];
     WCHAR szDataBuf[VAR_COMMAND_MAX];
     DWORD dwDataLen;

     // "@CURRENT_TIME":コントローラ保有の現在時刻
     switch (m_dwSysID) {
     case VAR_CURRENT_TIME:

          // 送信コマンド文字列の作成
          // ----------
          // ワード単位でデーター括受信[5,]コマンド使って時刻情報を取得する
          //     (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,D125,D124,D123,..."+<CR>
          //               D120:     秒
          //               D121:     分
          //               D122:     時
          //               D123:     日
          //               D124:     月
          //               D125:     年
          //               D126:     曜日     00-06 = 日曜-土曜

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

          // コマンド送信と応答受信
          hr = m_pSerial->SendAndReceive((LPBYTE)szCmdBuf,
                                         0,
                                         (LPBYTE)szDataBuf,
                                         VAR_COMMAND_MAX * sizeof(WCHAR),
                                         &dwDataLen);
          if (FAILED(hr)) {
               return hr;
          }

          int n, iYear, iMonth, iDay, iHour, iMinute, iSecond, iDayOfWeek;
          n = swscanf(szDataBuf, L"%d,%d,%d,%d,%d,%d,%d",
                      &iDayOfWeek, &iYear, &iMonth, &iDay,
                      &iHour, &iMinute, &iSecond);
          if (n != 7) { // 正しく取得できなかった
               return E_UNEXPECTED;
          }

          // 年は 02 -> 2002 なので
          iYear += 2000;

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

          // DATE 変換したものを返す
          if (!SystemTimeToVariantTime(&tm, &(pVal->date))) {
               return E_UNEXPECTED;
          }
```

```
            pVal->vt = VT_DATE;

            hr = S_OK;
            break;

        // "@ERROR_DESCRIPTION":発生中のエラーの説明.
        case VAR_ERROR_DESCRIPTION:

            // 送信コマンド文字列の作成
            // ----------
            // アラームコードを取得する
            //     <EC>+"1"+<CR>

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

            // コマンド送信と応答受信
            hr = m_pSerial->SendAndReceive((LPBYTE)szCmdBuf,
                                           0
                                           (LPBYTE)szDataBuf,
                                           VAR_COMMAND_MAX * sizeof(WCHAR),
                                           &dwDataLen);
            if (FAILED(hr)) {
                return hr;
            }

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

        // "I_MAKER_NAME" : コントローラのメーカ名(任意の文字列)
        case VAR_MAKER_NAME:
            pVal->vt = VT_BSTR;
            pVal->bstrVal = SysAllocString(TC_PROV_MAKER);
            hr = S_OK;
            break;

        // "I_TYPE" : コントローラの型式(任意の文字列)
        case VAR_TYPE:
            pVal->vt = VT_BSTR;
            pVal->bstrVal = SysAllocString(TC_PROV_TYPE);
            hr = S_OK;
            break;

        // "I_VERSION" : コントローラのバージョン(任意の文字列)
        case VAR_VERSION:
            pVal->vt = VT_BSTR;
            pVal->bstrVal = SysAllocString(TC_PROV_VER);
            hr = S_OK;
            break;

        default:
            hr = E_INVALIDARG;
        }

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

5.3.4.4. FinalPutValue ()の実装

最後に FinalPutValue()メソッドを実装します.

このメソッドでの処理は基本的に GetVarValue()の処理と同じです. 違いは, ユーザ変数が VAR_TYPE_BIT の場合, セットなら＜CMD＞“8”(0x38)コマンドを, リセットなら＜CMD＞“9”(0x39)コマンドを使用, VAR_TYPE_WORD の場合, ＜CMD＞“10”(0x31,0x30)コマンドをそれぞれ使用しているという点と応答の“OK”をチェックしている点です.

List 5-16 CaoProvVariable.cpp – FinalPutValue()

```
/**     変数の値の変更
 *
 *  ICaoVariable::put_Value()の実体となる.
 *
 *     @param   newVal    :        [in]  変数値
 *     @retval  HRESULT
 *
 */
HRESULT CCaoProvVariable::FinalPutValue(VARIANT newVal)
{
//↓[17]-----------------------------------------------------------------------↓
    HRESULT hr = S_OK;
    WCHAR szCmdBuf[VAR_COMMAND_MAX];
    int iCmdBufLen = 0;

    // 設定値を取得
    CComVariant vntVal;
    hr = vntVal.ChangeType(VT_I2, &newVal);
    if (FAILED(hr)) {
        return hr;
    }
    short nVal = vntVal.iVal;

    // 送信コマンド文字列の作成
    // ----------
    // ビット単位の場合
        // BIT 単位のセットには [8,] , リセットには [9,]をそれぞれ用いる
    // ワード単位の場合
        // WORD 単位の書き込みには [10,]
    // それ以外
        // 未実装

    szCmdBuf[iCmdBufLen++] = CMD_EC;     // コマンド開始コード
    switch (m_usrDataType) {
    case VAR_TYPE_BIT:
        szCmdBuf[iCmdBufLen++] = (nVal ? '8' : '9'); // <CMD> = '8'-ON, '9'-OFF
        szCmdBuf[iCmdBufLen++] = ',';
        break;
    case VAR_TYPE_WORD:
        szCmdBuf[iCmdBufLen++] = '1';                // <CMD> = '10'
        szCmdBuf[iCmdBufLen++] = '0';
        szCmdBuf[iCmdBufLen++] = ',';
        break;
    case VAR_TYPE_UNKNOWN:
        return E_NOTIMPL;     // 未実装
        break;
    default:
        return E_NOTIMPL;     // データタイプエラー
    }

    wcscpy(&szCmdBuf[iCmdBufLen], m_bstrUpperName);
```

```
    iCmdBufLen += SysStringLen(m_bstrUpperName);

    // ワード単位の場合設定値を指定
    if (m_usrDataType == VAR_TYPE_WORD) {
        swprintf( &(szCmdBuf[iCmdBufLen]), L",%d", nVal );
        iCmdBufLen += wcslen(szCmdBuf);
    }
    szCmdBuf[iCmdBufLen] = NULL;

    // コマンド送信と応答受信
    WCHAR szDataBuf[VAR_COMMAND_MAX];
    DWORD dwDataLen;
    hr = m_pSerial->SendAndReceive((LPBYTE)szCmdBuf,
                                   0
                                   (LPBYTE)szDataBuf,
                                   VAR_COMMAND_MAX * sizeof(WCHAR),
                                   &dwDataLen);
    if (FAILED(hr)) {
        return hr;
    }

    // 応答文字列をチェックする
    switch (m_usrDataType) {
    case VAR_TYPE_BIT:
    case VAR_TYPE_WORD:
        // "OK"+<CR>なら正常
        if (wcscmp(szDataBuf, L"OK¥r")) {
            hr = E_FAIL;
        }
        break;
    }

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

5.3.5. 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) CAO プロバイダのプロジェクトウィザードを使ってスケルトン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::ParseParameter()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()でデバイス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::GetSystemValue()でシステム変数取得処理を実装しました.

(12) 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[ビルド(B)]を実行して TCmini.DLL を作成します.

図 5-4 TCmini.DLL のビルド

5.4. TCmini プロバイダのデバッグとリリース

5.4.1. TCmini プロバイダのデバッグ

CAO 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, CAO 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¥CaoTester.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.

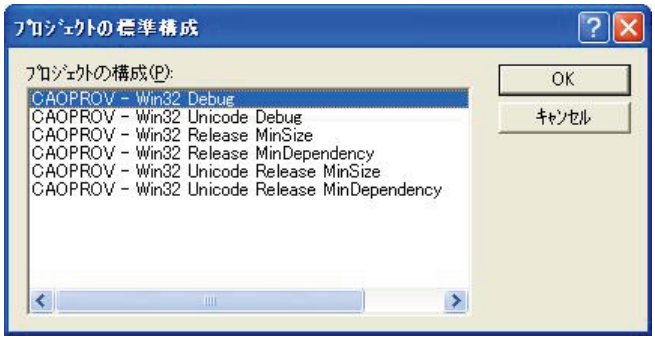


図 5-5 ビルドターゲット指定画面

(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.

[プロジェクト(P)]メニューから[設定(S) Alt+F7]を選択して, 設定の対象(S)で[Win32 Debug]を指定してからデバッグセクションの実行可能なファイル(E)に CAO.exe をフルパスで指定します. CAO.exe は通常 ORiN2¥CAO¥Engine¥Bin フォルダの下にあります.

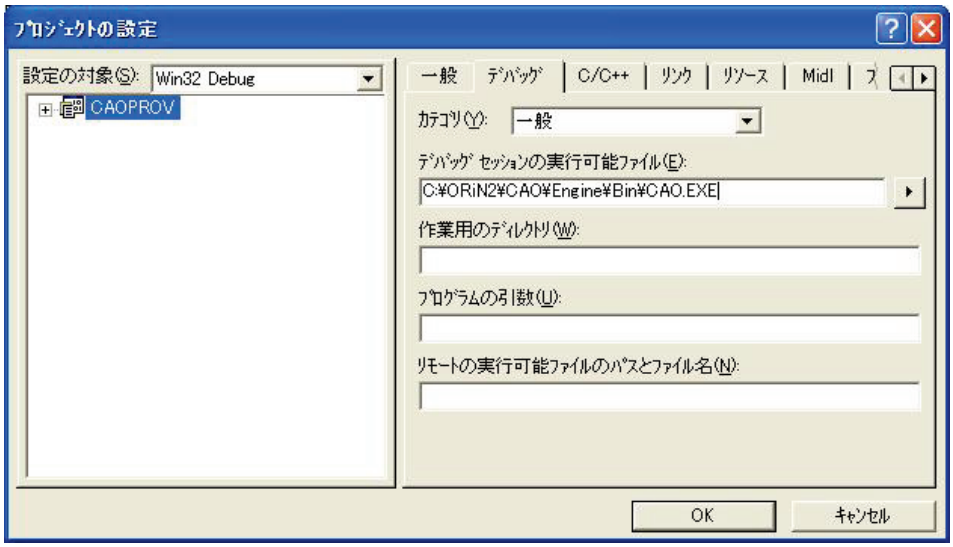


図 5-6 CAO.exe の指定画面

(3) 必要な位置にブレークポイントをセットします.

(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.

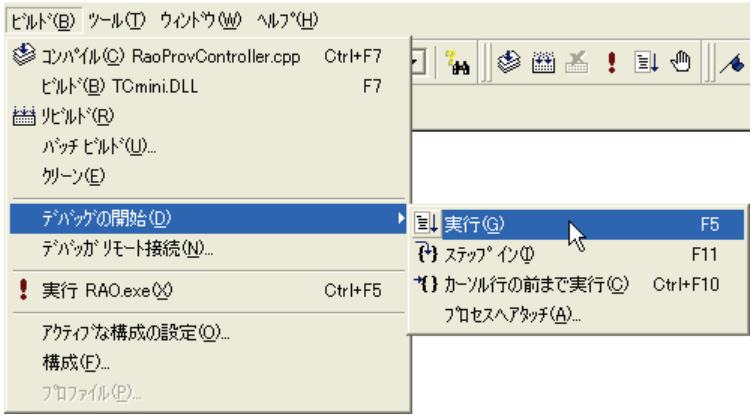


図 5-7 デバッグの開始画面

(5) CaoTester.exe を起動し，TCmini プロバイダを指定します．

CaoWorkspace の ProviderName に[CaoProv.TCmini]を選択し，Option に“Conn=com:1”を入力します．

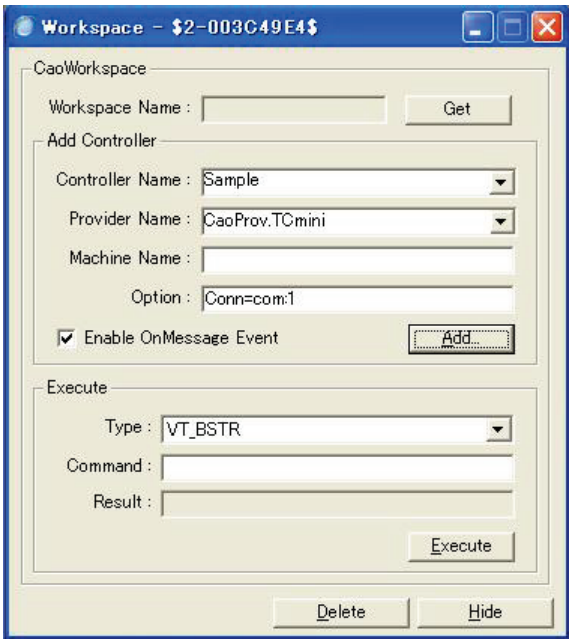


図 5-8 ImplChk – TCmini プロバイダ指定画面

(6) [Add]ボタンを押下してコントローラと接続します．

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

例えば，ユーザ変数の“X007”を作成してその値を取得するには次の手順でおこないます．

1. Controller ウィンドウの Variable タブ- Name に X007 と入力します．
2. [Add]ボタンを実行します．
3. Variable ウィンドウの Value:の[get]ボタンを押下して X007 の現在の状態を取得します．

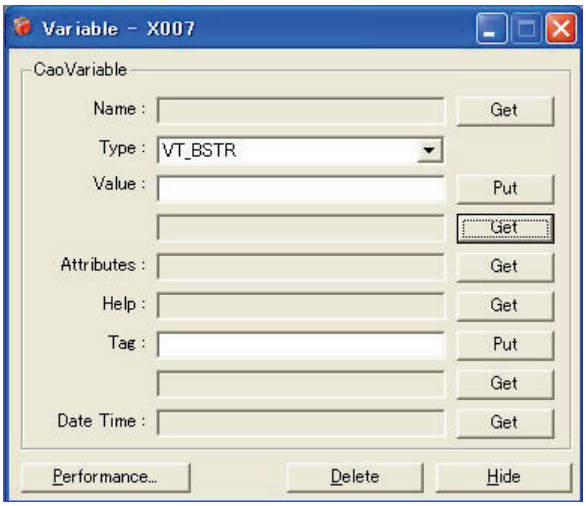


図 5-9 Value の取得画面

値の設定は，例えば，ユーザ変数の“Y027”を作成してその値を入力，Value の[put]でおこないます．

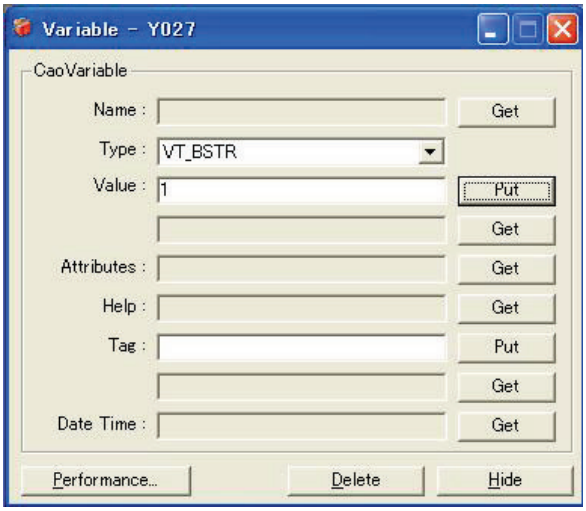


図 5-10 Value の設定画面

システム変数を扱う場合は，Variable タブを選択して VariableNames の[Get]ボタンを押下します．

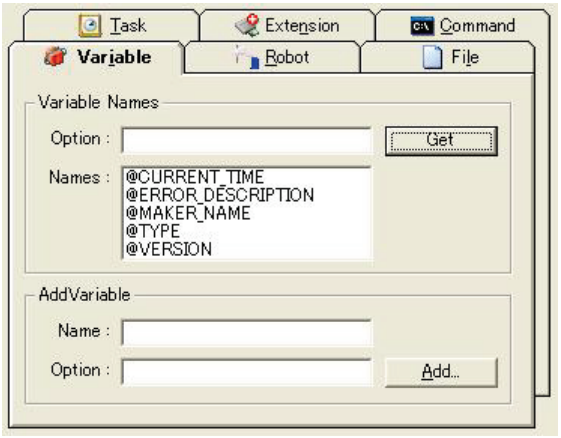


図 5-11 システム変数の指定画面

(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．

5.4.2. TCmini プロバイダのリリース

デバッグが完了したら，ビルドターゲットを[CAOPROV – Win32 Release MinDependency]に変更し，ビルドを行って最終的な TCmini.dll モジュールを作成します．

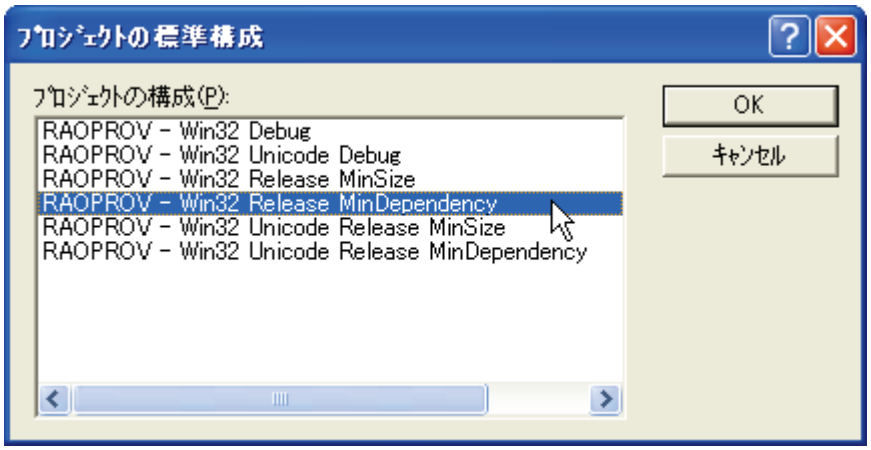


図 5-12 ビルドターゲット指定画面

5.4.2.1. 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.

CAO 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) 対応している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)

今回作成した TCmini プロバイダには特別に考慮しなくてはならない点はないので①から③までの内容を網羅したプロバイダ仕様書を作成します. Microsoft Word のテンプレートファイル(<ORiN2>¥CAO¥Provider¥Doc¥xxx_ProvGuide_jp.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』等を参考にしてください.

5.4.2.2. 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.

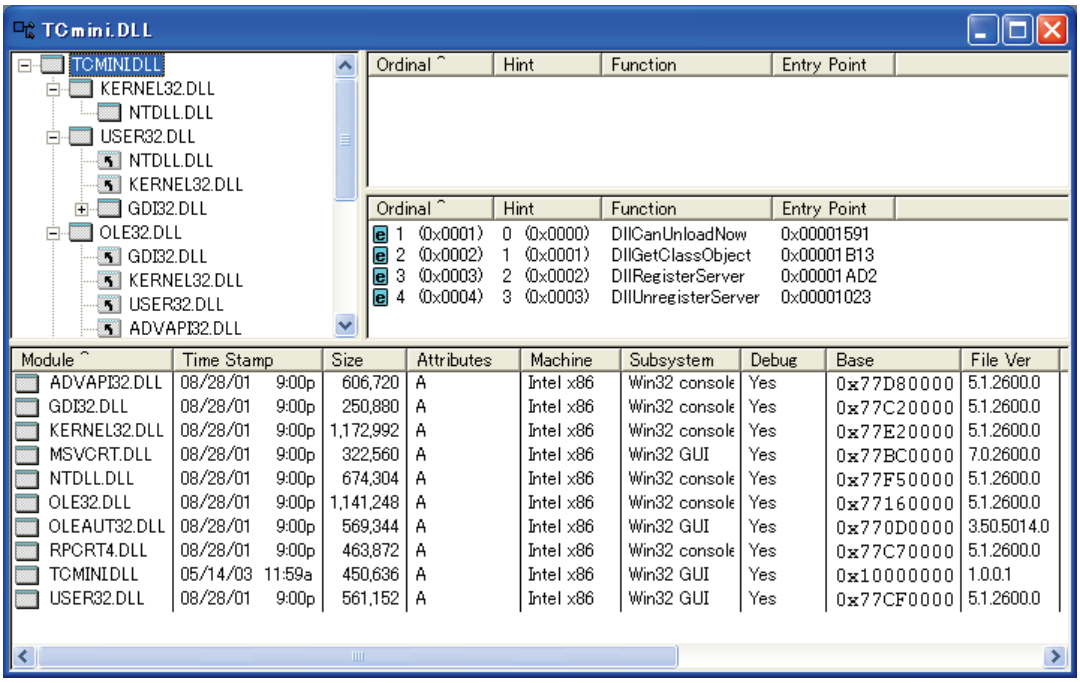


図 5-13 TCmini.DLL の Dependency Walker 画面

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 [10]. 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.

[10] 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.

6. CaoProvExec ツールの利用方法

CaoProvExec は，プロバイダを代理プロセス(dllhost.exe)で起動するためのツールです．

Windows95 / 98 において COM を使用するためには，モジュールを起動しておく必要がありますが，dll 単体では起動することができません．そこで，代理プロセスを使用して CAO プロバイダの起動をおこないます．

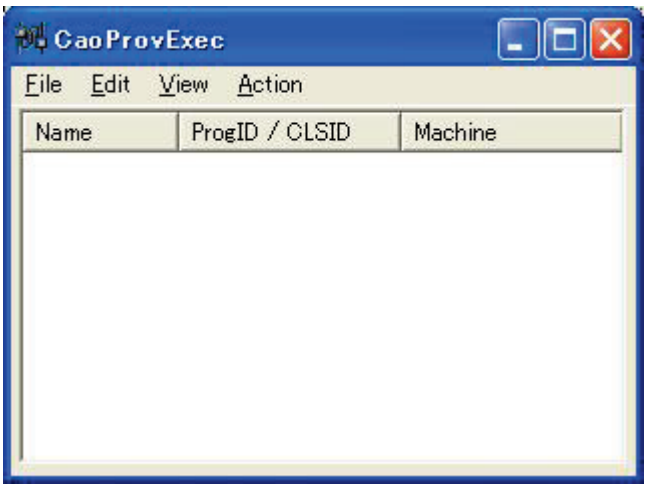


図 6-1 CaoProvExec ツール

メニューの「Edit」から「Add」を選選択すると，以下のようなダイアログが表示されますので，プロバイダの名前，プログラム ID またはクラス ID，起動マシン名を入力し，OK ボタンを押下してください．

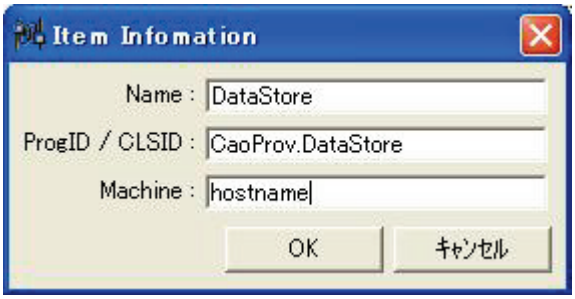


図 6-2 プロバイダの追加

メニューの「Action」→「CreateInstance」から，プロバイダのインスタンスを生成することができます．

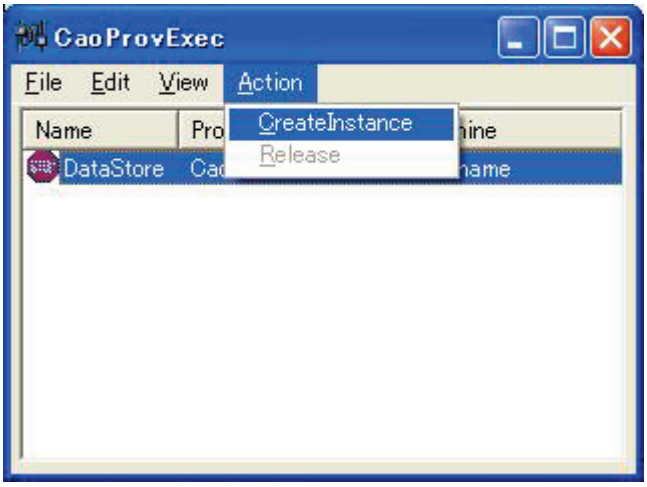


図 6-3 CreateInstance

付録A. 付録

付録A.1. CAO 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. ・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. ・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 P：プロパティ E：イベント		（注 1）	・[]内は省略可能引数. ・BSTR 型の省略可能引数のデフォルト値は NULL. ・数値型の省略可能引数のデフォルト値はゼロ. ・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, [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 [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 P：プロパティ E：イベント			（注1）	・[]内は省略可能引数. ・BSTR 型の省略可能引数のデフォルト値は NULL. ・数値型の省略可能引数のデフォルト値はゼロ. ・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 [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 [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, [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 [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 P：プロパティ E：イベント			（注1）	・[]内は省略可能引数. ・BSTR 型の省略可能引数のデフォルト値は NULL. ・数値型の省略可能引数のデフォルト値はゼロ. ・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, 加速度:Float, [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, 移動量:float, [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 [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, [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, ポーズ列：VARIANT, [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, 角度：float, 回転中心：VARIANT, [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, 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 P：プロパティ E：イベント			（注1）	・[]内は省略可能引数. ・BSTR 型の省略可能引数のデフォルト値は NULL. ・数値型の省略可能引数のデフォルト値はゼロ. ・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, [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 [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, [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 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, [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 (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 P：プロパティ E：イベント			（注1）	・[]内は省略可能引数. ・BSTR 型の省略可能引数のデフォルト値は NULL. ・数値型の省略可能引数のデフォルト値はゼロ. ・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 P：プロパティ E：イベント			（注1）	・[]内は省略可能引数. ・BSTR 型の省略可能引数のデフォルト値は NULL. ・数値型の省略可能引数のデフォルト値はゼロ. ・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 P：プロパティ E：イベント			（注1）	・[]内は省略可能引数. ・BSTR 型の省略可能引数のデフォルト値は NULL. ・数値型の省略可能引数のデフォルト値はゼロ. ・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) 2 - ログ出力. 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 P:プロパティ E:イベント			(注1)	・[]内は省略可能引数. ・BSTR 型の省略可能引数のデフォルト値は NULL. ・数値型の省略可能引数のデフォルト値はゼロ. ・VARIANT 型の省略可能引数のデフォルト値は VT_ERROR.			

付録A.2. CAO 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, [メッセージ本文:VARIANT], [作成日時:VARIANT], [送り先:BSTR], [送り元:BSTR], [説明:BSTR], [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. メッセージ番号 : 0 メッセージ本文 : VT_EMPTY 作成日時 : CreateMessage の実行日時 送り先 : 空文字 送り元 : コントローラ名 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. ・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. ・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, 検索パラメータ:BSTR, 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． ・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． ・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) M：メソッド(HRESULT) D：データ			・[]内は省略可能引数． ・BSTR 型の省略可能引数のデフォルト値は NULL． ・数値型の省略可能引数のデフォルト値はゼロ． ・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 [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, 検索パラメータ：BSTR, 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) M：メソッド(HRESULT) D：データ			・[]内は省略可能引数. ・BSTR 型の省略可能引数のデフォルト値は NULL. ・数値型の省略可能引数のデフォルト値はゼロ. ・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 [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 [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 [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, 検索パラメータ:BSTR, 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) M:メソッド(HRESULT) D:データ			・[]内は省略可能引数. ・BSTR 型の省略可能引数のデフォルト値は NULL. ・数値型の省略可能引数のデフォルト値はゼロ. ・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, 加速度：Float, [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, 移動量：float, [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 [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, [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, 角度：float, 回転中心：VARIANT, [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, 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, 検索パラメータ：BSTR, 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) M：メソッド(HRESULT) D：データ			・[]内は省略可能引数. ・BSTR 型の省略可能引数のデフォルト値は NULL. ・数値型の省略可能引数のデフォルト値はゼロ. ・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 [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, [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 オプションは開始位置など.
	FinalStop	E	タスクの停止	モード：Long, [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 (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, 検索パラメータ：BSTR, 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) M：メソッド(HRESULT) D：データ			・[]内は省略可能引数. ・BSTR 型の省略可能引数のデフォルト値は NULL. ・数値型の省略可能引数のデフォルト値はゼロ. ・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, 検索パラメータ:BSTR, 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) M:メソッド(HRESULT) D:データ			・[]内は省略可能引数. ・BSTR 型の省略可能引数のデフォルト値は NULL. ・数値型の省略可能引数のデフォルト値はゼロ. ・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, 検索パラメータ：BSTR, 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) M:メソッド(HRESULT) D:データ		・[]内は省略可能引数. ・BSTR 型の省略可能引数のデフォルト値は NULL. ・数値型の省略可能引数のデフォルト値はゼロ. ・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) M:メソッド(HRESULT) D:データ			・[]内は省略可能引数. ・BSTR 型の省略可能引数のデフォルト値は NULL. ・数値型の省略可能引数のデフォルト値はゼロ. ・VARIANT 型の省略可能引数のデフォルト値は VT_ERROR.			