# CORBA 版 CAO プロバイダ作成ガイド

version 1.1.0

June 30, 2014

【備考】

# 【改版履歴】

| 日付 | 版数 | 内容 |
|---|---|---|
| 2006-02-23 | 1.0.0 | 初版作成. |
| 2014-06-30 | 1.1.0 | GwCaopCorba.exe にログ出力先オプション追加(/L:n) |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

# 目次

# 1. はじめに

本書は，分散オブジェクト技術に CORBA を利用した CAO プロバイダを作成するためのユーザーズガイドです．

ORiN2 SDK の CAO プロバイダは，DCOM により分散オブジェクト技術を実現しています．そのため，CAO プロバイダを開発したい場合，そのプラットフォームは Windows に限定されてしまいます．しかし，分散オブジェクト技術に DCOM ではなく，CORBA を利用することにより，Windows や Linux などのプラットフォームに依存しない CAO プロバイダの作成が可能となります．

分散オブジェクト技術に CORBA を利用するには，図 1-1 に示すように，以下の 2 つのモジュールが必要となります．

(1) CORBA 版 CAO プロバイダ
CAO プロバイダと同等のインタフェースを持つ CORBA サーバです．CORBA メソッドの呼び出しにより，各種ロボットやリソースへのアクセスをおこないます．

(2) CORBA プロバイダ
CORBA プロバイダは，CORBA 版 CAO プロバイダのメソッド呼び出しをおこなうための CAO プロバイダです．CAO エンジンから CORBA 版 CAO プロバイダにアクセスするためのアダプタの役目をします．

この 2 つのモジュールにより，クライアントアプリケーションから，CORBA プロバイダを介して CORBA 版 CAO プロバイダにアクセスすることができます．CORBA プロバイダの内部において DCOM と CORBA の差異を吸収しているため，クライアントからは CAO プロバイダを利用する場合と同じインタフェースで CORBA 版 CAO プロバイダにアクセスすることが可能となります．

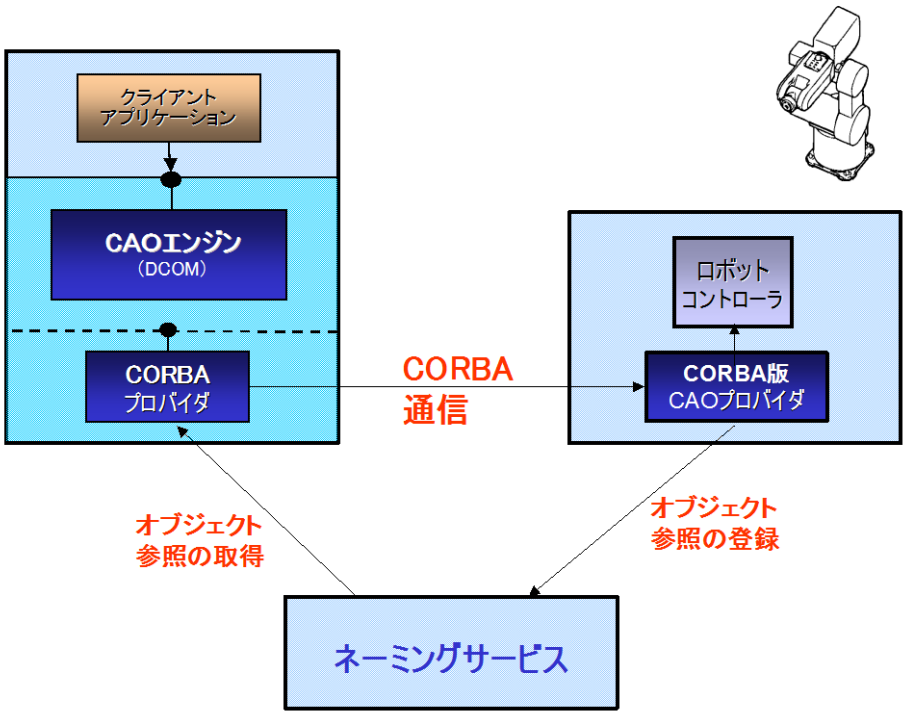


**図 1-1 CORBA 版 CAO プロバイダ概要**

ORiN2 SDK では，CORBA の実装に OmniORB を用いています．OminiORB は，Windows，Linux，Mac OS，Solaris など数多くのプラットフォームに対応しており，開発言語としては C++と Python に対応しています．

図 1-2 に CAO プロバイダと CORBA 版 CAO プロバイダのオブジェクトモデルの対応を示します．この図に示すように，CORBA 版 CAO プロバイダのインタフェースは，CAO プロバイダと同じ構成になっており，すべてのメソッドが 1 対 1 に対応します．これにより，実装者は CAO プロバイダと同様の使い方を各クラス及びメソッドに対しておこなうことができます．

また，本来 COM の仕様である VARIANT 型を擬似的に実装することにより，より CAO プロバイダの実装に近くなるように表現してあります．

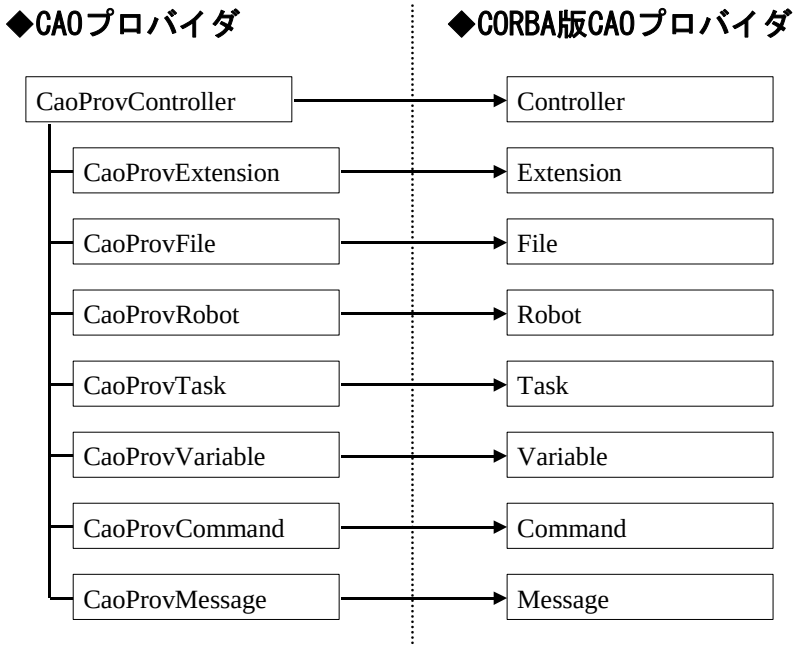


**図 1-2 CORBA 版 CAO プロバイダのオブジェクトモデル**

また，OmniORB では，OmniNames と呼ばれるネーミングサービスが提供されており，CORBA 版 CAO プロバイダでは，これを利用することでオブジェクトを名前により管理することができます．これにより，CORBA 版 CAO プロバイダが登録したオブジェクト参照を，名前をキーにして CORBA プロバイダから取得することが可能になります．

ここで，ネーミングサービスに登録する際のルールを以下に示します．この登録内容は上から順に階層構造で登録されています．

**表 1-1 CaoCorba のネーミングサービス登録ルール**

| 登録名 | 種別 |
|---|---|
| “cao” | “cao” |
| <起動マシン名> | “Server” |
| <コントローラ名> | “Controller” |

本書では，第 2 章で OmniORB のインストール方法を解説し，第 3 章では，CORBA 版 CAO プロバイダの作成手順について説明します．次に第 4 章では，CORBA 版 CAO プロバイダにアクセスするための CORBA プロバイダについて簡単に解説し，第 5 章では，CORBA 版 CAO プロバイダのサンプル実装である GwCaopCorba について説明をおこないます．

# 2. OmniORB インストール

## 2.1. 概要

ORiN2 SDK では, CORBA プロバイダおよび CORBA 版 CAO プロバイダの CORBA 実装に OmniORB を用いています.

OmniORB とは, フリーな CORBA(Common Object Request Broker Architecture)準拠の ORB(object request broker)実装の一つです. IIOP 通信プロトコルに基づいているため, CORBA に準拠していれば, 他の ORB 実装とも相互接続をおこなうことができます.

CORBA プロバイダおよび, CORBA 版 CAO プロバイダを利用・開発するためには, OmniORB をインストールする必要があります. 本章では, Windows 環境[1]における OmniORB のインストール方法および, ネーミングサービスの設定方法について解説をおこないます.

## 2.2. インストール手順

以下の手順で OmniORB のインストールをおこないます.

(1) omniORB4.0.5 をダウンロードします.
下記のサイトよりダウンロードをおこなってください.

**表 2-1 omniORB ダウンロードサイト**

| ファイル名 | URL |
|---|---|
| omniORB-4.0.5-win32-vc6.zip | http://omniorb.sourceforge.net/ |

(2) ダウンロードしたファイルを任意のフォルダに解凍します. ここで解凍してできたフォルダを“$OMNIHOME”と呼ぶことにします.
(3) 環境変数の“Path”に“$OMNIHOME¥bin¥x86_win32”を追加します.
(4) 環境変数に“OMNINAMES_LOGDIR”を追加し, 任意のパスを指定します. ここで指定したパスにネーミングサービスのログが出力されます.

[1] Linux 環境の場合は, 同サイトより omniORB-4.0.5.tar.gz をダウンロードし, README.unix の内容に従ってください.

(5) レジストリエディタを起動してください.
スタートメニューの「ファイル名を指定して実行」に「regedit」と入力し, OK ボタンを押下してください.

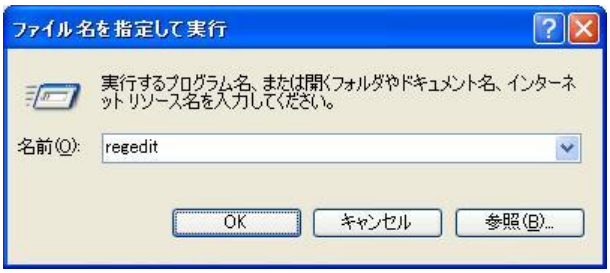


**図 2-1 レジストリエディタの起動**

(6) 以下のキーを追加します.
HKEY_LOCAL_MACHINE¥SOFTWARE¥omniORB¥InitRef

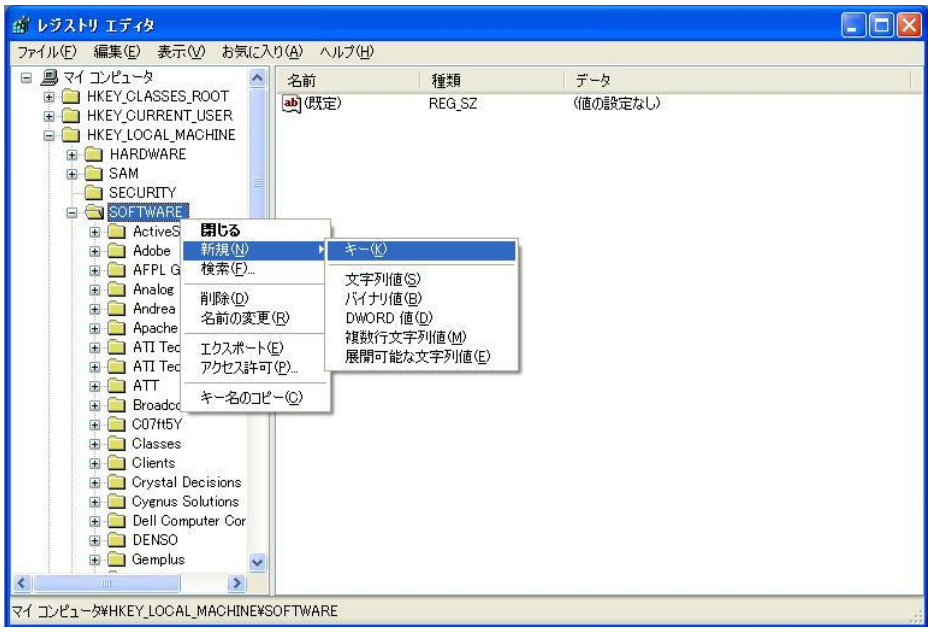


**図 2-2 キーの追加**

(7) 図 2-3 に示すように，作成したキーに文字列値を追加します．

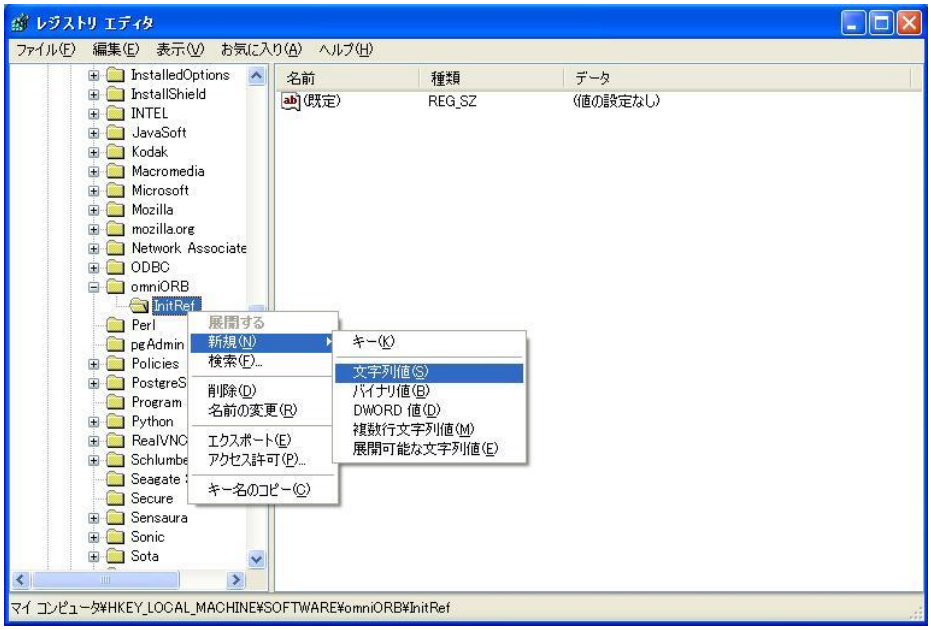


**図 2-3 文字列値の追加**

キー名とその値は，以下のように設定します．

キー名 : “1”
値 : NameService=corbaname::<IP アドレス／ホスト名>

上記の＜IP アドレス／ホスト名＞には，ネーミングサービスの起動しているコンピュータの IP アドレスまたは，ホスト名を指定してください．以下に記入例を示します．

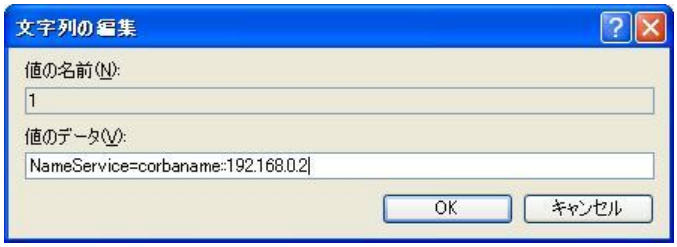


**図 2-4 IP アドレスの記入例**


**図 2-5 ホスト名の記入例**

(8) コマンドプロンプトから“omniNames -start”と入力し，ネーミングサービスを起動します[2].

[2] リモートマシンでネーミングサービスを使用する場合は，リモートマシンでのみネーミングサービスを起動し，ローカルマシンでは起動しないでください.

# 3. CORBA 版 CAO プロバイダ作成方法

## 3.1. はじめに

本章では，CORBA 版 CAO プロバイダの作成方法について解説をおこないます．

本章で作成する CORBA 版 CAO プロバイダのサンプルプログラムには，以下の機能を実装します．

(1) main 関数
- CorController オブジェクトの生成とネーミングサービスへの登録
- クライアントからの呼び出し待ち受け

(2) CorController クラス
- Connect メソッド
- GetVariable メソッド

(3) CorVariable クラス
- コンストラクタ
- initialize メソッド
- Value メソッド

CorController クラスと CorVariable クラスは，それぞれ，CAO プロバイダの CaoProvController クラスと CaoProvVariable クラスにあたります．

CORBA 版 CAO プロバイダでは，初期起動時に CorController オブジェクトを生成し，そのオブジェクトリファレンスをネーミングサービスに登録する必要があります．この処理を main 関数内に実装します．

main 関数内の処理と CorController クラスは，CORBA 版 CAO プロバイダを作成する際には，必ず実装する必要があります．

それ以外のクラス(Robot クラスや File クラスなど)は，CorVariable クラスの実装方法を参考にしながら，必要なものだけを実装するようにしてください．

CAO プロバイダと CORBA 版 CAO プロバイダのインタフェースは，以下のように対応しています．

**表 3-1 CAO プロバイダ-CORBA 版 CAO プロバイダ インタフェース対応表**

| CAO プロバイダ | | CORBA サーバ |
|---|---|---|
| クラス | メソッド，プロパティ名 | |
| CaoProvController | Connect | Controller::connect() |
| | Disconnect | Controller::disconnect() |
| | CommandNames | Controller::getCommandNames() |
| | ExtensionNames | Controller::getExtensionNames() |

| CAO プロバイダ | | CORBA サーバ |
|---|---|---|
| クラス | メソッド, プロパティ名 | |
| | FileNames | Controller::getFileNames() |
| | RobotNames | Controller::getRobotNames() |
| | TaskNames | Controller::getTaskNames() |
| | VariableNames | Controller::getVariableNames() |
| | GetCommand | Controller::getCommand() |
| | GetExtension | Controller::getExtension() |
| | GetFile | Controller::getFile() |
| | GetRobot | Controller::getRobot() |
| | GetTask | Controller::getTask() |
| | GetVariable | Controller::getVariable() |
| | Execute | Controller::execute() |
| | Attribute | Controller::type() |
| | Help | Controller::help() |
| CaoProvExtension | VariableNames | Extension::getVariableNames() |
| | GetVariable | Extension::getVariable() |
| | Execute | Extension::execute() |
| | Attribute | Extension::type() |
| | Help | Extension::help() |
| CaoProvFile | FileNames | File::getFileNames() |
| | VariableNames | File::getVariableNames() |
| | GetFile | File::getFile() |
| | GetVariable | File::getVariable() |
| | Execute | File::execute() |
| | Copy | File::copy() |
| | Delete | File::deleteFile() |
| | Move | File::move() |
| | Run | File::run() |
| | Attribute | File::type() |
| | Help | File::help() |
| | DateCreated | File::dataCreated() |
| | DateLastAccessed | File::dataLastAccessed() |
| | DateLastModified | File::dataLastModified() |
| | Path | File::path() |

| CAO プロバイダ | | CORBA サーバ |
|---|---|---|
| クラス | メソッド, プロパティ名 | |
| | Size | File::size() |
| | Type | File::fileType() |
| | Value | File::value() |
| CaoProvRobot | VariableNames | Robot::getVariableNames() |
| | GetVariable | Robot::getVariable() |
| | Execute | Robot::execute() |
| | Accelerate | Robot::accelerate() |
| | Cancel | Robot::cancel() |
| | Change | Robot::change() |
| | Chuck | Robot::chuck() |
| | Drive | Robot::drive() |
| | GoHome | Robot::goHome() |
| | Move | Robot::move() |
| | Rotate | Robot::rotate() |
| | Speed | Robot::speed() |
| | Unchuck | Robot::unchuck() |
| | Attribute | Robot::type() |
| | Help | Robot::help() |
| CaoProvTask | VariableNames | Task::getVariableNames() |
| | GetVariable | Task::getVariable() |
| | Execute | Task::execute() |
| | Start | Task::start() |
| | Stop | Task::stop() |
| | Delete | Task::deleteTask() |
| | Attribute | Task::type() |
| | Help | Task::help() |
| | FileName | Task::fileName() |
| CaoProvVariable | Attribute | Variable::type() |
| | Help | Variable::help() |
| | DateTime | Variable::dateTime() |
| | Value | Variable::value() |
| CaoProvCommand | Execute | Command::execute() |
| | Cancel | Command::cancel() |

| CAO プロバイダ | | CORBA サーバ |
|---|---|---|
| クラス | メソッド, プロパティ名 | |
| | Attribute | Command::type() |
| | Help | Command::help() |
| | Parameters | Command::parameters() |
| | State | Command::state() |
| | Timeout | Command::timeout() |

## 3.2. CORBA 版 CAO プロバイダの雛形作成

まずは, CORBA 版 CAO プロバイダの雛形の作成をおこないます. これは, CAO プロバイダでいうところの, CaoProvWiz.exe で自動生成される部分にあたります.

### 3.2.1. CORBA スケルトンの生成

まずは, IDL コンパイラにより, CORBA スケルトンを生成します.

(1) CaoCorba の IDL ファイルである CaoProvCorba.idl を IDL コンパイルします. CaoProvCorba.idl は, ORiN2 SDK インストールディレクトリ内の“CAO¥Include”からコピーして利用してください.

```
> omniidl.exe -bcxx -Wbh=.h -Wbs=SK.cpp CaoProvCorba.idl
```

このコマンドを実行することで, CaoProvCorba.h と CaoProvCorbaSK.cpp が生成されます[3].
CORBA では, このように IDL コンパイラを利用することで, ORB へのインタフェース部分のソースコードが自動生成されます.
なお, CaoProvCorba.h と CaoProvCorbaSK.cpp は, ORiN2 SDK インストールディレクトリ内の“CAO¥Include”にもありますので, こちらをコピーして利用することもできます.

### 3.2.2. プロジェクトの作成・設定

本章では, Visual C++6.0 を用いて CORBA 版 CAO プロバイダの作成をおこないます. Visual C++6.0 で CORBA サーバのプログラミングをおこなうためには, 以下の手順で設定をおこなう必要があります.

(1) Visual C++ 6.0 でプロジェクトを新規作成します. 「ファイル」メニューの「新規作成」から, “Win32 Console Application”を選択してください.

(2) 3.2.1 で生成された CaoProvCorbaSK.cpp と CaoProvCorba.h をプロジェクトに追加します. 追加した CaoProvCorbaSK.cpp をファイルビューから選択し, 右クリックメニューの[設定]を選択します. 表示された“プロジェクトの設定”の[C/C++]タブを選択し, カテゴリを[プリコンパイル済みヘッダ]を選択

[3] このコマンドが失敗する場合は, omniidl.exe へのパスが通っていない可能性があります. もう一度 2 章の内容を確認してください.

し，そこで表示された中の“プリコンパイル済みヘッダを使用しない”にチェックを入れてください．

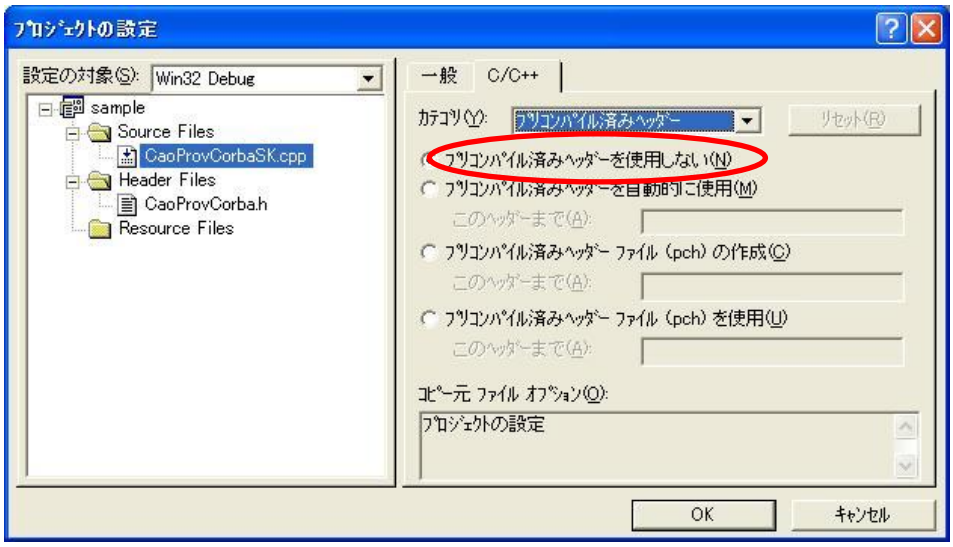


**図 3-1 プリコンパイル済みヘッダの使用**

(3) “プロジェクトの設定”の左側にあるファイルビューのルートにあるプロジェクトを選択し[C/C++]タブを選択してください．

(4) カテゴリで[C++言語]を選択し，“例外処理を有効にする”にチェックを入れます．

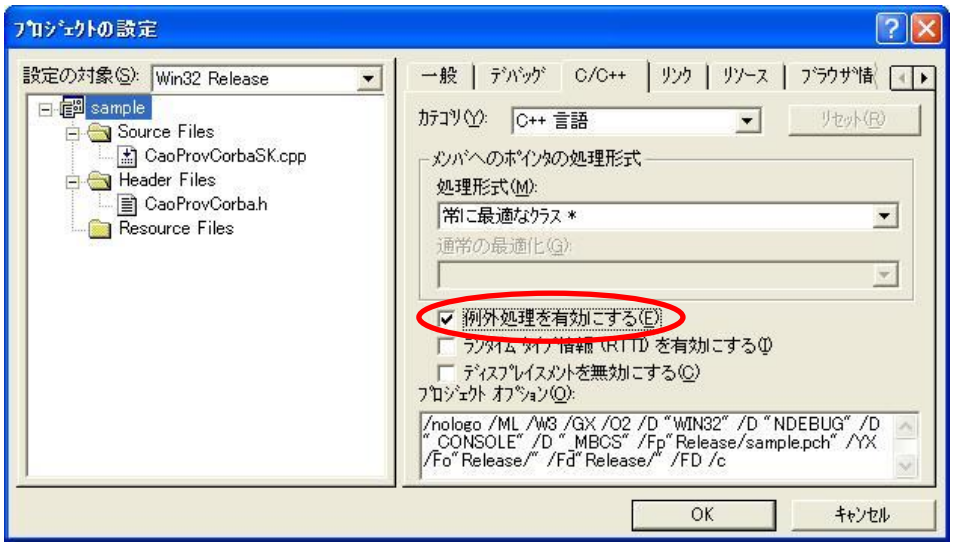


**図 3-2 例外処理を有効にする**

(5) カテゴリで[コード生成]を選択し，“使用するランタイムライブラリ”で[マルチスレッド(dll)]を選択してください．

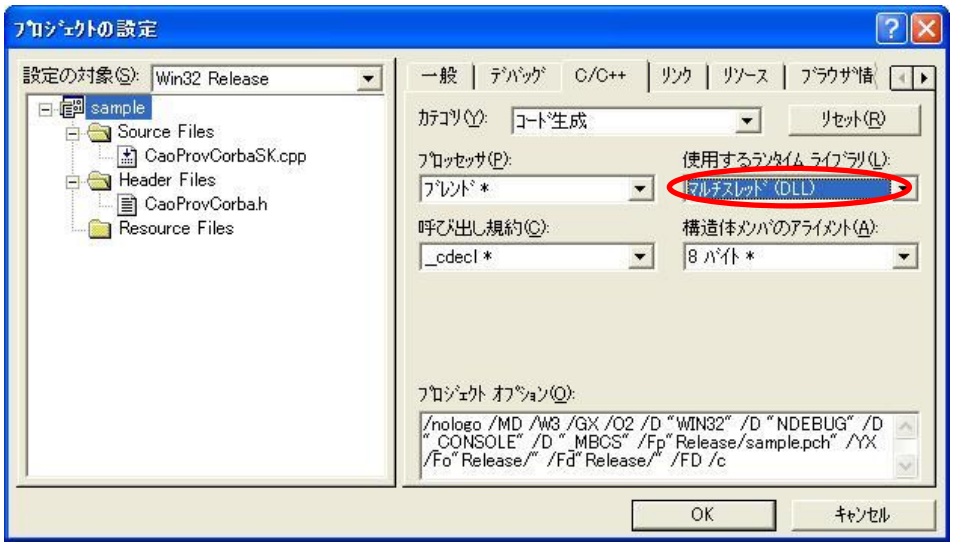


**図 3-3 使用するランタイムライブラリの選択**

(6) カテゴリで[プリプロセッサ]を選択し，“プリプロセッサの定義”に以下のものを追加します．
__WIN32__, __x86__, _WIN32_WINNT=0x0400, __NT__, __OSVERSION__=4

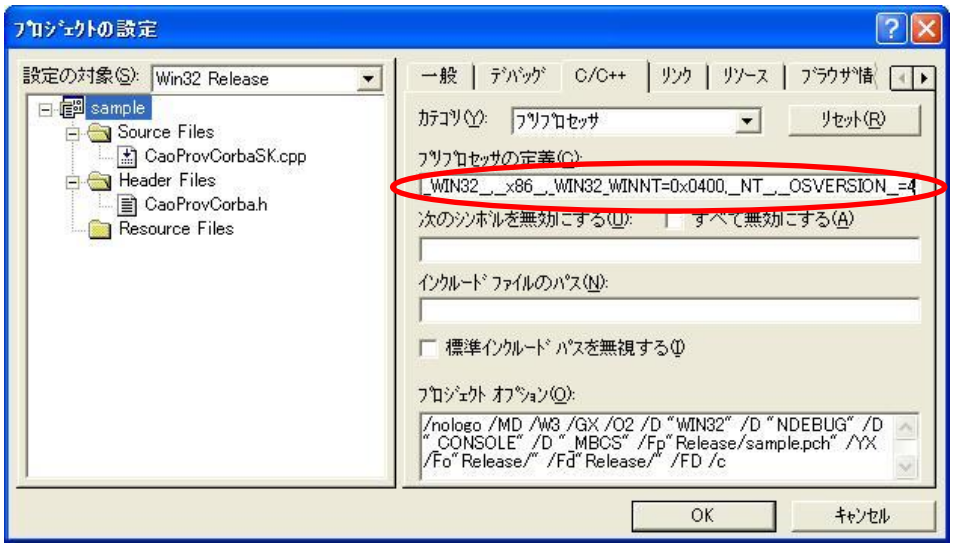


**図 3-4 プリプロセッサの定義の追加**

(7) [リンク]タブを選択し，カテゴリで[インプット]を選択する．“オブジェクト/ライブラリモジュール”に以下のものを追加します．

【Release モード】

```
ws2_32.lib mswsock.lib advapi32.lib omniORB405_rt.lib omniDynamic405_rt.lib
omnithread30_rt.lib
```

【Debug モード】

```
ws2_32.lib mswsock.lib advapi32.lib omniORB405_rtd.lib omniDynamic405_rtd.lib
omnithread30_rtd.lib
```

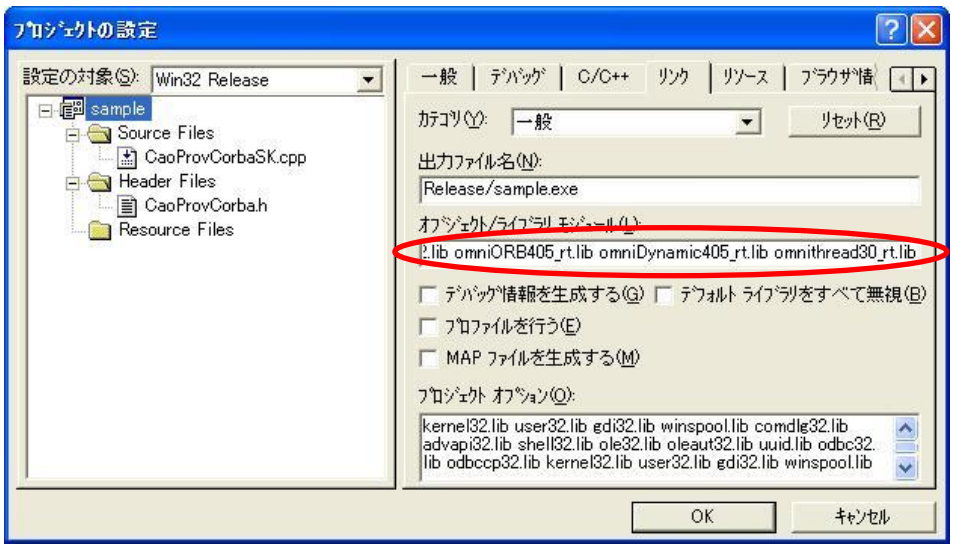


**図 3-5 オブジェクト/ライブラリモジュールの追加**

(8) メニューバーの[ツール]→[オプション]を選択し，[ディレクトリ]タブを選択します．

(9) “表示するディレクトリ”で[インクルードファイル]を選択して，omniORB の Include フォルダへのパスを追加します．

例)C:¥ Program Files¥omniORB-4.0.5¥include

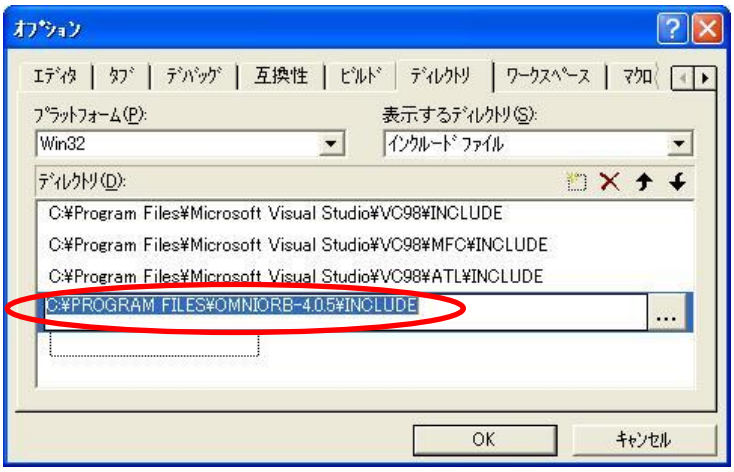


**図 3-6 インクルードファイルディレクトリの追加**

(10) “表示するディレクトリ”で[ライブラリファイル]を選択して，omniORB の“Lib¥x86_Win32”フォルダへのパスを追加します．

例)C:¥Program Files¥omniORB-4.0.5¥lib¥x86_win32

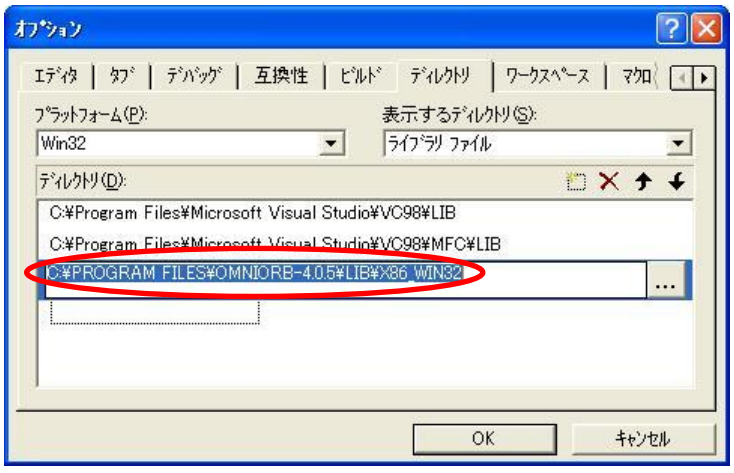


**図 3-7 ライブラリファイルディレクトリの追加**

### 3.2.3. CorController クラスの雛形作成

CAO プロバイダの CaoProvController にあたる CorController クラスの雛形を作成します. このクラスでは, 以下の 2 つのクラスを継承します.

- POA_CaoProv::Controller
- PortableServer::RefCountServantBase

POA_CaoProv::Controller は, IDL ファイルに記述した, Controller クラスの抽象クラスです. このクラスを継承し, CORBA 版 CAO プロバイダに必要なメソッドの実装をおこないます.

PortableServer::RefCountServantBase は, このサーバントを指しているリファレンスの数をカウントするためのクラスです. このクラスを継承することで, サーバントのためのメモリ領域が自動的に管理されます.

プロジェクトに以下のヘッダファイルとソースファイルを追加してください. ここでは, POA_CaoProv::Controller クラスから継承したメンバ以外に, コンストラクタとコントローラ名を保持する“m_szCtrlName”を追加しています.

**List 3-1 CorController.h**

```
#ifndef __CorController_hh__
#define __CorController_hh__

#include "CaoProvCorba.h"

using namespace CaoProv;

class CCorController : public POA_CaoProv::Controller,
                       public PortableServer::RefCountServantBase
{
public :
    CCorController();
    virtual ~CCorController();
    virtual Variant* getExtensionNames(const CORBA::WChar* wszOption);
    virtual Variant* getFileNames(const CORBA::WChar* wszOption);
    virtual Variant* getRobotNames(const CORBA::WChar* wszOption);
    virtual Variant* getTaskNames(const CORBA::WChar* wszOption);
    virtual Variant* getVariableNames(const CORBA::WChar* wszOption);
    virtual Variant* getCommandNames(const CORBA::WChar* wszOption);
    virtual void connect(const CORBA::WChar* wszName,
                         const CORBA::WChar* wszOption, ControllerEvents_ptr pCaoEvent);
    virtual void disconnect();
    virtual Extension_ptr getExtension(const CORBA::WChar* wszName,
                                       const CORBA::WChar* wszOption);
    virtual File_ptr getFile(const CORBA::WChar* wszName, const CORBA::WChar* wszOption);
    virtual Robot_ptr getRobot(const CORBA::WChar* wszName, const CORBA::WChar* wszOption);
    virtual Task_ptr getTask(const CORBA::WChar* wszName, const CORBA::WChar* wszOption);
    virtual Variable_ptr getVariable(const CORBA::WChar* wszName,
                                     const CORBA::WChar* wszOption);
    virtual Command_ptr getCommand(const CORBA::WChar* wszName, const CORBA::WChar* wszOption);
    virtual Variant* execute(const Variant& Command);
    virtual CORBA::WChar* name();
    virtual CORBA::Long type();
    virtual CORBA::WChar* help();
    virtual Variant* id();
    virtual void id(const Variant& _v);

private:
```

```
    char m_szCtrlName[MAX_STR_LEN];          // コントローラ名

};

#endif
```

**List 3-2 CorController.cpp**

```
#include "CaoProvCorbaImpl.h"

/**    コンストラクタ */
CCorController::CCorController()
{
    return;
}

/**    デストラクタ */
CCorController::~CCorController()
{
    return;
}

/**    拡張ボード名取得
 *
 *    @param    wszOption          :        [in]     オプション
 *    @retval   Variant            :        取得した拡張ボード名
 *
 */
Variant* CCorController::getExtensionNames(const CORBA::WChar* wszOption)
{
    return NULL;
}

/**    ファイル名取得
 *
 *    @param    wszOption          :        [in]     オプション
 *    @retval   Variant            :        取得したファイル名
 *
 */
Variant* CCorController::getFileNames(const CORBA::WChar* wszOption)
{
    return NULL;
}

/**    ロボット名取得
 *
 *    @param    wszOption          :        [in]     オプション
 *    @retval   Variant            :        取得したロボット名
 *
 */
Variant* CCorController::getRobotNames(const CORBA::WChar* wszOption)
{
    return NULL;
}

/**    タスク名取得
 *
 *    @param    wszOption          :        [in]     オプション
 *    @retval   Variant            :        取得したタスク名
 *
 */
Variant* CCorController::getTaskNames(const CORBA::WChar* wszOption)
```

```
{
    return NULL;
}

/**    変数名取得
 *
 *    @param    wszOption          :        [in]     オプション
 *    @retval   Variant            :        取得した変数名
 *
 */
Variant* CCorController::getVariableNames(const CORBA::WChar* wszOption)
{
    return NULL;
}

/**    コマンド名取得
 *
 *    @param    wszOption          :        [in]     オプション
 *    @retval   Variant            :        取得したコマンド名
 *
 */
Variant* CCorController::getCommandNames(const CORBA::WChar* wszOption)
{
    return NULL;
}

/**    接続処理
 *
 *    @param    wszName            :        [in]     コントローラ名
 *    @param    wszOption          :        [in]     オプション
 *
 */
void CCorController::connect(const CORBA::WChar* wszName, const CORBA::WChar* wszOption,
ControllerEvents_ptr pCaoEvent)
{
    return;
}

/**    切断処理 */
void CCorController::disconnect()
{
    return;
}

/**    CorExtension 取得処理
 *
 *    @param    wszName            :        [in]     拡張ボード名
 *    @param    wszOption          :        [in]     オプション
 *    @retval   Extension_ptr      :        CorExtension のオブジェクトリファレンス
 *
 */
Extension_ptr CCorController::getExtension(const CORBA::WChar* wszName, const CORBA::WChar*
wszOption)
{
    return NULL;
}

/**    CorFile 取得処理
 *
 *    @param    wszName            :        [in]     ファイル名
 *    @param    wszOption          :        [in]     オプション
 *    @retval   File_ptr           :        CorFile のオブジェクトリファレンス
 *
 */
File_ptr CCorController::getFile(const CORBA::WChar* wszName, const CORBA::WChar* wszOption)
{
```

```
    return NULL;
}

/**    CorRobot 取得処理
 *
 *    @param    wszName          :        [in]     ロボット名
 *    @param    wszOption        :        [in]     オプション
 *    @retval   Robot_ptr        :        CorRobot のオブジェクトリファレンス
 *
 */
Robot_ptr CCorController::getRobot(const CORBA::WChar* wszName, const CORBA::WChar* wszOption)
{
    return NULL;
}

/**    CorTask 取得処理
 *
 *    @param    wszName          :        [in]     タスク名
 *    @param    wszOption        :        [in]     オプション
 *    @retval   Task_ptr         :        CorTask のオブジェクトリファレンス
 *
 */
Task_ptr CCorController::getTask(const CORBA::WChar* wszName, const CORBA::WChar* wszOption)
{
    return NULL;
}

/**    CorVariable 取得処理
 *
 *    @param    wszName          :        [in]     変数名
 *    @param    wszOption        :        [in]     オプション
 *    @retval   Variable_ptr     :        CorVariable のオブジェクトリファレンス
 *
 */
Variable_ptr CCorController::getVariable(const CORBA::WChar* wszName, const CORBA::WChar*
wszOption)
{
    return NULL;
}

/**    CorCommand 取得処理
 *
 *    @param    wszName          :        [in]     コマンド名
 *    @param    wszOption        :        [in]     オプション
 *    @retval   Command_ptr      :        CorCommand のオブジェクトリファレンス
 *
 */
Command_ptr CCorController::getCommand(const CORBA::WChar* wszName, const CORBA::WChar*
wszOption)
{
    return NULL;
}

/**    拡張コマンド実行処理
 *
 *    @param    wszName          :        [in]     実行コマンド名
 *    @retval   Variant*         :        拡張コマンドの実行結果
 *
 */
Variant* CCorController::execute(const Variant& Command)
{
    return NULL;
}

/**    コントローラ名取得処理
 *
```

```
 *    @retval    WChar*              :        取得したコントローラ名
 *
 */
CORBA::WChar* CCorController::name()
{
    return NULL;
}

/**    属性取得処理
 *
 *    @retval    Long                :        取得した属性
 *
 */
CORBA::Long CCorController::type()
{
    return NULL;
}

/**    ヘルプ取得処理
 *
 *    @retval    WChar*              :        取得したヘルプ
 *
 */
CORBA::WChar* CCorController::help()
{
    return NULL;
}

/**    ID 取得処理
 *
 *    @retval    Variant*            :        取得した ID
 *
 */
Variant* CCorController::id()
{
    return NULL;
}

/**    ID 設定処理
 *
 *    @param     _v                  :        設定する ID
 *
 */
void CCorController::id(const Variant& _v)
{
    return;
}
```

### 3.2.4. CorVariable クラスの雛形作成

次に CAO プロバイダでの CaoProvVariable にあたる CorVariable クラスの雛形を作成します. このクラスでは, 以下の 2 つのクラスを継承します.

- POA_CaoProv::Variable
- PortableServer::RefCountServantBase

CorController クラスと同様に, POA_CaoProv::Variable は, IDL ファイルに記述した, Variable クラスの抽象クラスです. このクラスを継承し, CORBA 版 CAO プロバイダに必要なメソッドの実装をおこないます.

プロジェクトに以下のヘッダファイルとソースファイルを追加してください. ここでは, POA_CaoProv::Variable クラスから継承したメンバ以外に, コンストラクタと“initialize”メソッド, さらに, 変数名を保持する“m_szVarName”と変数値を保持する“data”を追加しています.

**List 3-3 CorVariable.h**

```
#ifndef __CorVariable_hh__
#define __CorVariable_hh__

#include "CaoProvCorba.h"

using namespace CaoProv;

class CCorVariable : public POA_CaoProv::Variable,
                     public PortableServer::RefCountServantBase
{
public:
    CCorVariable();
    virtual ~CCorVariable();
    CORBA::Boolean initialize(char *variableName)
    virtual CORBA::WChar* name();
    virtual CORBA::Long type();
    virtual CORBA::WChar* help();
    virtual Variant* dateTime();
    virtual CORBA::Long microsecond();
    virtual Variant* value();
    virtual void value(const Variant& _v);
    virtual Variant* id();
    virtual void id(const Variant& _v);

private:
    char m_szVarName[MAX_STR_LEN]; // 変数名
    Variant *data;                 // 変数値
};

#endif
```

**List 3-4 CorVariable.cpp**

```
#include "CaoProvCorbaImpl.h"

/**    コンストラクタ */
CCorVariable::CCorVariable()
```

```
{
    return;
}

/**    デストラクタ */
CCorVariable::~CCorVariable()
{
    return;
}

/**    初期化処理
 *
 *    @param    pArgs[]            :        [in]pArgs[0] - 変数名
 *    @retval   Boolean            :        実行結果
 *
 */
CORBA::Boolean CCorVariable::initialize(char *variableName)
{
    return false;
}

/**    変数名取得
 *
 *    @retval   WChar*             :        取得した変数名
 *
 */
CORBA::WChar* CCorVariable::name()
{
    return NULL;
}

/**    属性取得
 *
 *    @retval   Long               :        取得した属性
 *
 */
CORBA::Long CCorVariable::type()
{
    return 0;
}

/**    ヘルプ取得
 *
 *    @retval   WChar*             :        取得したヘルプ
 *
 */
CORBA::WChar* CCorVariable::help()
{
    return NULL;
}

/**    日時取得
 *
 *    @retval   Variant*           :        取得した日時
 *
 */
Variant* CCorVariable::dateTime()
{
    return NULL;
}

/**    秒数取得
 *
 *    @retval   Long               :        取得した時間[msec]
 *
 */
```

```
CORBA::Long CCorVariable::microsecond()
{
    return 0;
}

/**    変数値取得処理
 *
 *    @retval    Variant*          :        取得した変数
 *
 */
Variant* CCorVariable::value()
{
    return NULL;
}

/**    変数値設定処理
 *
 *    @param    _v                 :        設定する変数
 *
 */
void CCorVariable::value(const Variant& _v)
{
    return;
}

/**    ID 取得処理
 *
 *    @retval    Variant*          :        取得した ID
 *
 */
Variant* CCorVariable::id()
{
    return NULL;
}

/**    ID 設定処理
 *
 *    @param    _v                 :        設定する ID
 *
 */
void CCorVariable::id(const Variant& _v)
{
    return;
}
```

上記以外のクラスの雛形を作成する場合も，CorController や CorVariable と同様の方法でおこなってください．

### 3.3. CorController のネーミングサービスへの登録処理

CORBA プロバイダでは，AddController メソッドが実行されると，ネーミングサービスに登録されている CorController のオブジェクトリファレンスの取得をおこないます．その後の処理は，すべて CorController のオブジェクトリファレンスを利用して，CORBA 版 CAO プロバイダへのアクセスを実現しています．

CORBA 版 CAO プロバイダでは，初期起動時に CorController オブジェクトを生成し，に示すルールに従って，オブジェクトリファレンスをネーミングサービスに登録する必要があります．今回作成するプログラムでは，起動マシン名を GetComputerName 関数[4]により取得し，コントローラ名をプログラム起動時の引数として指定します．

プロジェクトに以下のヘッダファイルとソースファイルを追加してください．

**List 3-5 CaoProvCorbaImpl.h**

```
#include <iostream.h>

#define MAX_STR_LEN       256

// VARTYPE の定義
#define VT_EMPTY      0
#define VT_NULL       1
#define VT_I2         2
#define VT_I4         3
#define VT_R4         4
#define VT_R8         5
#define VT_CY         6
#define VT_DATE       7
#define VT_BSTR       8
#define VT_DISPATCH   9
#define VT_ERROR      10
#define VT_BOOL       11
#define VT_VARIANT    12
#define VT_UNKNOWN    13
#define VT_UI1        17
#define VT_RESERVED   0x8000
#define VT_BYREF      0x4000
#define VT_ARRAY      0x2000

#include "CaoProvCorba.h"
#include "CorController.h"
#include "CorVariable.h"

static CORBA::Boolean
bindObjectToName(CORBA::ORB_ptr orb, CORBA::Object_ptr objref, const char * controllerName);

using namespace CaoProv;
```

**List 3-6 CaoProvCorbaImpl.cpp**

```
#include "CaoProvCorbaImpl.h"

int main(int argc, char **argv)
```

[4] GetComputerName は Win32 API なので，Linux などの環境の場合は，gethostname 関数などを利用してください．

```
{
    char controllerName[MAX_STR_LEN];

    // プログラムの引数をコントローラ名に設定
    strcpy(controllerName,(argc > 1) ? argv[1] : "sample" );

    try {
    // ORB の初期化
        CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);
    // RootPOA の取得
        CORBA::Object_var obj = orb->resolve_initial_references("RootPOA");
        PortableServer::POA_var poa = PortableServer::POA::_narrow(obj);

    //CorController を生成
        CCorController* prov = new CCorController();
    //CorController を活性化
        PortableServer::ObjectId_var myCorID = poa->activate_object(prov);

    // CorController のオブジェクトリファレンスを取得
        obj = prov->_this();

    // CorController をネーミングサービスに登録
        if( !bindObjectToName(orb, obj, controllerName) )
            return 1;

        prov->_remove_ref();

    // POA を活性化
        PortableServer::POAManager_var pman = poa->the_POAManager();
        pman->activate();

    // ORB の起動
        orb->run();         // このメソッドで待ち受ける

    // ORB の破棄
        orb->destroy();
    }
    catch(CORBA::SystemException&) {
        cerr << "Caught CORBA::SystemException." << endl;
    }
    catch(CORBA::Exception&) {
        cerr << "Caught CORBA::Exception." << endl;
    }
    catch(omniORB::fatalException& fe) {
        cerr << "Caught omniORB::fatalException:" << endl;
        cerr << "  file: " << fe.file() << endl;
        cerr << "  line: " << fe.line() << endl;
        cerr << "  mesg: " << fe.errmsg() << endl;
    }
    catch(...) {
        cerr << "Caught unknown exception." << endl;
    }

    return 0;
}

/**    ネーミングサービス登録
 *
 * 引数で渡されたオブジェクトをネーミングサービスに登録する
 *
 *   @param    orb              :    [in] オブジェクトリクエストブローカー
 *   @param    objref           :    [in] 登録するオブジェクトのリファレンス
 *   @param    controllerName   :    [in] オブジェクトの登録名
 *   @retval   Boolean          :    実行結果
 *
 */
```

```
static CORBA::Boolean
bindObjectToName(CORBA::ORB_ptr orb, CORBA::Object_ptr objref, const char * controllerName)
{
    CosNaming::NamingContext_var rootContext;
    CosNaming::NamingContext_var CurContext;
    CosNaming::Name CtrlName;

    try {
    // ネーミングサービスを検索
        CORBA::Object_var obj;
        obj = orb->resolve_initial_references("NameService");

    // ネーミングサービスの参照を取り出す
        rootContext = CosNaming::NamingContext::_narrow(obj);
        if( CORBA::is_nil(rootContext) ) {
            cerr << "Failed to narrow the root naming context." << endl;
            return 0;
        }
    }
    catch(CORBA::ORB::InvalidName&) {
        cerr << "Service required is invalid [does not exist]." << endl;
        return 0;
    }

    try {
    // コンテキストの設定
        CosNaming::Name contextName;
        contextName.length(1);
        contextName[0].id   = (const char*) "cao";
        contextName[0].kind = (const char*) "cao";
    // 新しいコンテキストを指定した名前で登録
        try {
            CurContext = rootContext->bind_new_context(contextName);
        }
        catch(CosNaming::NamingContext::AlreadyBound&) {
            CORBA::Object_var obj;
            obj = rootContext->resolve(contextName);
            CurContext = CosNaming::NamingContext::_narrow(obj);
            if( CORBA::is_nil(CurContext) ) {
                cerr << "Failed to narrow naming context." << endl;
                return 0;
            }
        }

    // コンピュータ名を格納するバファを確保します.
        TCHAR computerName[MAX_STR_LEN];
        DWORD length = MAX_STR_LEN;

    // GetComputerName API を呼び出してコンピュータ名を取得します.
        GetComputerName(computerName, &length);

    // コンピュータ名を設定
        CosNaming::Name objectName;
        objectName.length(1);
        objectName[0].id   = (const char*) computerName;
        objectName[0].kind = (const char*) "Server";
    // 新しいコンテキストを指定した名前で登録
        try {
            CurContext = CurContext->bind_new_context(objectName);
        }
        catch(CosNaming::NamingContext::AlreadyBound&) {
            CORBA::Object_var serverObj;
            serverObj = CurContext->resolve(objectName);
            CurContext = CosNaming::NamingContext::_narrow(serverObj);
            if( CORBA::is_nil(CurContext) ) {
                cerr << "Failed to narrow naming context."<< endl;
```

```
                throw;
            }
        }

    // コントローラ名を設定
        CtrlName.length(1);
        CtrlName[0].id    = (const char*)controllerName;
        CtrlName[0].kind     = (const char*)"Controller";

    // CorController をネーミングサービスに登録
        try {
            CurContext->bind(CtrlName, objref);
        }
        catch(CosNaming::NamingContext::AlreadyBound&) {
            CurContext->rebind(CtrlName, objref);
        }

    }
    catch(CORBA::COMM_FAILURE&) {
        cerr << "Caught system exception COMM_FAILURE -- unable to contact the "
                << "naming service." << endl;
        return 0;
    }
    catch(CORBA::SystemException&) {
        cerr << "Caught a CORBA::SystemException while using the naming service."
     << endl;
        return 0;
    }

    return 1;
}
```

以上で, CORBA 版 CAO プロバイダの雛形が完成しました. この状態でプロジェクトをビルドすると, CORBA 版 CAO プロバイダが生成されます. このプログラムを実行すると, 何の機能も持たない CORBA 版 CAO プロバイダが起動します.

## 3.4. CorController クラスの実装

### 3.4.1. Connect メソッド

クライアントアプリケーションにおいて, CaoWorkspace::AddController メソッドを実行すると, CORBA プロバイダは CORBA 版 CAO プロバイダの CorController::Connect メソッドを実行します.

このメソッド内には, ロボットコントローラとの接続処理などを実装します.

今回作成するサンプルプログラムでは, ロボットコントローラを扱わないので, 接続時のコントローラ名をクラスのメンバ変数に設定する処理のみを追加します.

**List 3-7 CorController.cpp – Connect**

```
/**    接続処理
 *
 *    @param   wszName          :       [in]    コントローラ名
 *    @param   wszOption        :       [in]    オプション
 *    @param   pCaoEvent        :       [in]    イベントクラスのオブジェクト参照
 *
```

```
 */
void CCorController::connect(const CORBA::WChar* wszName, const CORBA::WChar* wszOption,
ControllerEvents_ptr pCaoEvent)
{
    char controllerName[MAX_STR_LEN];
    int size = wcslen( wszName ) + 1;
    // ワイド文字をマルチバイト文字に変換
    wcstombs( controllerName, wszName, size );
    cout << (char *)controllerName << endl;

    strcpy(m_szCtrlName, controllerName);

    return;
}
```

### 3.4.2. GetVariable メソッド

次に GetVariable メソッドの実装をおこないます.

クライアントアプリケーションにおいて， CaoController::AddVariable メソッドを実行すると, CORBA プロバイダは, CORBA 版 CAO プロバイダの CorController::GetVariable メソッドを実行します.

このメソッドの中では, CorVariable クラスのオブジェクトを生成し, そのオブジェクトリファレンスを戻り値として返す処理を実装する必要があります.

**List 3-8 CorController.cpp – GetVariable**

```
/**    CorVariable 取得処理
 *
 *    @param    wszName          :        [in]     変数名
 *    @param    wszOption        :        [in]     オプション
 *    @retval   Variable_ptr     :        CorVariable のオブジェクトリファレンス
 *
 */
Variable_ptr CCorController::getVariable(const CORBA::WChar* wszName, const CORBA::WChar*
wszOption)
{
    char variableName[MAX_STR_LEN];
    int size = wcslen( wszName ) + 1;
    wcstombs( variableName, wszName, size );

    // Variable オブジェクトを生成
     CCorVariable* var = new CCorVariable;
    var->initialize( variableName );

    // 生成したオブジェクトのリファレンスを返す
    return Variable::_narrow(var->_this());
}
```

## 3.5. CorVariable クラスの実装

### 3.5.1. コンストラクタ

まずは，CorVariable クラスのコンストラクタを実装します．コンストラクタは，前述した CorController::GetVariable メソッドの中で呼び出されます．

ここでは，CorVariable クラスが保持する変数値の初期化をおこなっています．

**List 3-9 CorVariable.cpp –コンストラクタ**

```
/**    コンストラクタ */
CCorVariable::CCorVariable()
{
    data = new Variant;
    return;
}
```

### 3.5.2. initialize メソッド

次に，initialize メソッドの実装をおこないます．このメソッドは必ずしも作成する必要はありません．今回のサンプルプログラムでは，CAO プロバイダとの対応を分かりやすくするために，このメソッドを実装しました．

以下に initilize メソッドの記述を示します．ここでは，CorVariable クラスのメンバ変数に，AddVariable 呼び出し時の変数名を設定しています．

**List 3-10 CorVariable.cpp – initialize**

```
/**    初期化処理
 *
 *    @param    pArgs[]          :        [in]pArgs[0] - 変数名
 *    @retval   Boolean          :        実行結果
 *
 */
CORBA::Boolean CCorVariable::initialize(char *variableName)
{
    strcpy(m_szVarName, variableName);
    return false;
}
```

### 3.5.3. value メソッド(Get)

次に value メソッドの実装をおこないます．value メソッドはオーバーロードされており，引数の指定により，変数を取得するための処理と変数を設定するための処理に分かれます．

まずは，変数を取得する側の value メソッドについて実装をおこないます．

このメソッドは CORBA プロバイダの CaoVariable::FinalGetValue メソッド内から呼び出されます．

**List 3-11 CorVariable.cpp – Value**

```
/**    変数値取得処理
 *
 *    @retval    Variant*          :        取得した変数
 *
 */
Variant* CCorVariable::value()
{

    Variant *vnt = new Variant;
    vnt->vt = data->vt;
    vnt->val = data->val;
    vnt->sa = data->sa;
    return vnt;
}
```

ここで, CORBA 版 CAO プロバイダで利用されている擬似 Variant 型について簡単に説明します.

擬似 Variant 型の IDL 定義を以下に示します.

```
typedef sequence <Variant> SafeArray;
struct Variant {
       unsigned short     vt;
       any                val;
       SafeArray          sa;
};
```

このように, Variant の vt(VARTYPE)は unsigned shrot 型で, val は CORBA の Any 型で表現し, さらに, SafeArray は Variant 型の可変長 1 次元配列で表現しています.

CORBA の Any 型では, “<<=”や“>>=”といった演算子がオーバーロードされており, 右辺の型を自動的に判断し, 値の代入をおこなうことができます.

例えば, long 型の“1234”を代入する場合は, 以下のように記述します.

```
long test = 1234;
Variant *vnt = new Variant;
vnt->vt = VT_I4;
vnt->val <<= test;
```

### 3.5.4. value メソッド(Put)

次に変数を設定する側の value メソッドを実装します.

このメソッドは CORBA プロバイダの CaoVariable::FinalPutValue メソッド内から呼び出されます.

**List 3-12 CorVariable.cpp – Value**

```
/**    変数値設定処理
 *
 *    @param     _v              :        設定する変数
 *
 */
```

```
void CCorVariable::value(const Variant& _v)
{
    data->val = _v.val;
    data->vt = _v.vt;
    data->sa = _v.sa;
    return;
}
```

## 3.6. イベント機能

CORBA 版 CAO プロバイダにおいて, イベント機能を実装する場合は, CorController::Connect メソッドの引数である pCaoEvent を用います. この変数は CORBA プロバイダの CaoControllerEvent クラスのオブジェクトリファレンスであり, onMessege メソッドをコールすることでイベントを発生させることができます. なお, onMessage メソッドの引数には Message クラスのオブジェクトリファレンスが必要となるので, Message クラスも実装する必要があります.

## 3.7. まとめ

以上で CORBA 版 CAO プロバイダの作成は終了です.

今回作成したサンプルプロバイダは, Controller クラスと Variable クラスのみを実装しており, 変数の取得・設定の機能しか持っていませんが, 他のクラスを実装する場合も, このサンプルプログラムを参考にしながら作成することができます.

# 4. CORBA プロバイダ

## 4.1. 概要

CORBA プロバイダは，CORBA 版 CAO プロバイダに接続，通信するための CAO プロバイダです．

CORBA プロバイダは，ネーミングサービスより CORBA 版 CAO プロバイダの Controller クラスのオブジェクトリファレンスを検索して接続します．CORBA 版 CAO プロバイダのインタフェース構造は，CAO プロバイダと同様な構造を持つため，接続後は CAO プロバイダと同様に CORBA 版 CAO プロバイダにアクセスすることができます．

CORBA プロバイダの詳細については，「CORBA プロバイダユーザーズガイド」を参照してください．

## 4.2. メソッド・プロパティ

### 4.2.1. CaoWorkspace::AddController メソッド

CORBA 版 CAO プロバイダの Controller クラスのオブジェクトリファレンスをネーミングサービスより取得し，Controller クラスの Connect メソッドを実行します．

このメソッドのコントローラ名に，CORBA サーバで起動しているコントローラの名前を指定します．

**表 4-1 CaoWorkspace::AddController のオプション文字列**

| オプション | 意味 |
|---|---|
| Server=<CORBA サーバ名> | CORBA サーバの起動しているマシン名を指定します．(必須) |
| Option[=<オプション文字列>] | CORBA サーバの Connect()実行時に使用するオプション文字列．(デフォルト値：空文字列) |

以下に CAP プロバイダを使用して，DataStore プロバイダを起動するための，AddController メソッドの実行例を示します．

```
AddController
(
    “sample”,                   // コントローラ名 ＝ RC1
    “CaoProv.DNWA.CORBA”        // 固定
    “”,                         // CORBA プロバイダの起動マシン名
    “Server=SampleServer”       // CORBA サーバ “SampleServer” に接続
);
```

### 4.2.2. AddController 以外のメソッド・プロパティ

CORBA プロバイダは，コントローラ，ロボット，ファイル，タスク，変数，拡張ボードクラスのすべてのメソッド，プロパティが実装されています．前述の AddController 以外のメソッド，プロパティは，CORBA 版 CAO プロバイダで実装されているすべてのメソッド，プロパティを実行します．

## 4.3. サンプルプログラム

本節では，第 3 章で作成した CORBA 版 CAO プロバイダに接続し，変数の設定・取得をおこなうサンプルプログラムを作成します．ここで，CORBA 版 CAO プロバイダの動作しているコンピュータ名を“SampleServer”とします．

まずは，Visual Basic にて，以下のようなフォームを作成してください．

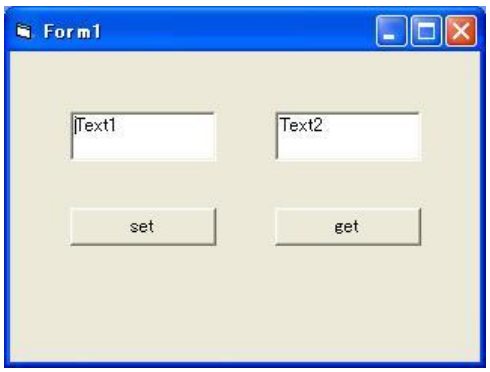


**図 4-1 サンプルプログラムのフォーム**

次に以下のようなソースコードを記述してください．

**List 4-1 Form1.frm**

```
Private eng As CaoEngine
Private ctrl As CaoController
Private var As CaoVariable

Private Sub Form_Load()
    Dim ws As CaoWorkspace
    Set eng = New CaoEngine
    Set ws = eng.Workspaces(0)
    ' サーバと接続
    Set ctrl = ws.AddController("sample", _
                                "CaoProv.DNWA.CORBA", _
                                "", _
                                "Server=SampleServer")
    ' 変数の取得
    Set var = ctrl.AddVariable("Var1")
End Sub

' 変数の設定
Private Sub Command1_Click()
    var = Text1.Text
End Sub

' 変数の取得
Private Sub Command2_Click()
    Text2.Text = var
End Sub
```

次にサンプルプログラムを実行します.

(1) ネーミングサービスの起動

ネーミングサービスを起動させるコンピュータにおいて, コマンドプロンプトより, 以下のように入力してください.

```
> omniNames -start
```

(2) CORBA 版 CAO プロバイダの起動

3 章で作成した CORBA 版 CAO プロバイダを起動してください.

(3) クライアントアプリケーションの起動

本章で作成したクライアントアプリケーションを起動してください.

クライアントアプリケーションのテキストボックス 1 に文字列を入力し“set”ボタンを押下します. 次に“get”ボタンを押下すると, テキストボックス 1 に入力した文字列が, テキストボックス 2 に表示されます.

# 5. GwCaopCorba.exe

## 5.1. 概要

GwCaopCorba は，CORBA プロバイダからの要求に応じて CAO プロバイダのメソッドを実行する CORBA プロバイダ－CAO プロバイダ間のゲートウェイプログラムです．このプログラムを使用することで，CORBA 通信を用いて，CAO プロバイダにアクセスすることができます．

また，GwCaopCorba は，CORBA 版 CAO プロバイダの実装をおこなったサンプルプログラムという位置づけでもあります．CORBA 版 CAO プロバイダのすべてのクラスとメソッドを実装しているので，CORBA 版 CAO プロバイダを作成するときの参考にしてください．

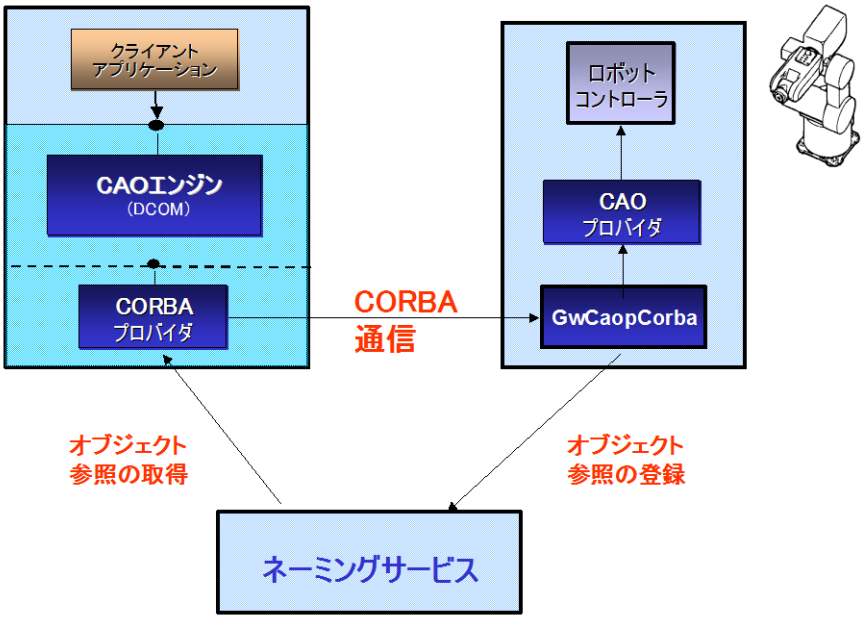


図 5-1 GwCaopCorba 動作概要

## 5.2. 使用方法

GwCaopCorba.exe は，起動する時に，CORBA のネーミングサービスに登録する内容を引数で指定する必要があります．ネームサービスに登録されるコントローラ名は大文字と小文字を区別して登録されます．GwCaopCorba.exe は，ネームサービスに登録するマシン名（サーバ名）はすべて大文字で登録します．

引数の仕様は以下の通りです．<コントローラ名>と<プロバイダ名>は必ず最後の引数にしてください．

```
GwCaopCorba.exe [/L:n | /RegServer | /UnregServer | /Service | /UnregName <コントローラ名> | /Force <コントローラ名> <プロバイダ名>]
```

**表 5-1 GwCaopCorba の引数仕様**

| 引数リスト | 実行内容 |
|---|---|
| <コントローラ名> | ネーミングサービスへの登録，および接続する CAO コントローラ名を指定します． |
| <プロバイダ名> | 起動する CAO プロバイダ名を指定します． |
| /Force | 指定したコントローラ名がネームサーバに登録済みの場合は上書きします． |
| /UnregName | ネーミングサービスから指定したコントローラを削除します[5]． |
| /RegServer | マシンへ GwCaopCorba をインストールします． |
| /UnregServer | マシンから GwCaopCorba をアンインストールします． |
| /Service | GwCaopCorba をサービス登録します． |
| /L:n | ログ出力先指定，1: MessageBox (default), 2:EventLog, 3:DebugViewer |

コントローラ名“Sample”，CAO プロバイダ“CaoProv.DataStore”を起動する場合の例を以下に示します．

```
> GwCaopCorba.exe Sample CaoProv.DataStore
```

次に，コントローラ名“Sample”をネーミングサービスから削除する場合の例を以下に示します．

```
> GwCaopCorba.exe /UnregName Sample
```

[5] ここで GwCaopCorba.exe はサービスとして終了するとき以外は，タスクマネージャーでプロセスを終了させる以外に終了することができません．また，そのように終了した時は，ネーミングサービスから登録した内容は削除されませんので注意してください．