# ORiN2

# プログラミングガイド

## Version 1.0.22.0

September 2, 2024

【備考】
お買い上げ頂いたパッケージによっては，インストールされない項目もあります．

## 【改版履歴】

| 日付 | 版数 | 内容 |
|---|---|---|
| 2006-02-23 | 1.0.0.0 | 初版作成. |
| 2006-07-19 | 1.0.1.0 | RCW の GAC への登録方法を追記. Java サンプル追記. |
| 2007-04-23 | 1.0.2.0 | インストーラによる GAC への CAO と CaoSQL RCW の自動登録. |
| 2008-07-16 | 1.0.3.0 | 一部加筆. |
| 2009-06-12 | 1.0.4.0 | CaoConfig のインポート・エクスポート機能追加 |
| 2009-07-31 | 1.0.5.0 | RCW の GAC への登録方法を追記 |
| 2010-02-10 | 1.0.6.0 | LabVIEW によるクライアント作成, エラーコードを追加 |
| 2010-06-08 | 1.0.7.0 | ORiN2SDK インストール状況の確認方法を追加 |
| 2011-04-27 | 1.0.8.0 | C#によるサンプル作成追加 |
| 2011-12-22 | 1.0.9.0 | エラーコード追加 |
| 2012-07-10 | 1.0.10.0 | エラーコード追加 |
| 2012-08-24 | 1.0.11.0 | エラーコード修正 |
| 2012-09-07 | 1.0.12.0 | エラーコード修正 |
| 2014-01-20 | 1.0.13.0 | エラーコード追加<br>Visual Studio へ統合を追加 |
| 2014-09-23 | 1.0.14.0 | エラーコード追加 |
| 2015-11-04 | 1.0.15.0 | CaoConfig の SysLog 出力機能追加 |
| 2016-12-12 | 1.0.16.0 | 誤記修正 |
| 2017-03-03 | 1.0.17.0 | エラーコード追加 |
| 2017-09-21 | 1.0.18.0 | Java-COM ブリッジを使用したクライアント作成の追加 |
| 2019-06-18 | 1.0.19.0 | エラーコード追加 |
| 2022-06-24 | 1.0.20.0 | ManagedCAO を使用した C#プログラミング追記 |
| 2022-11-22 | 1.0.21.0 | 誤記修正 |
| 2024-09-02 | 1.0.22.0 | 誤記修正 |

# 目次

# 1. はじめに

ORiN (Open Robot/Resource interface for the Network)は，ロボットを始めとする各種 FA 機器やデータベースなど，さまざまなリソースの標準的なインタフェースを提供するミドルウェアです．この ORiN を利用することにより，メーカや機種に依存しないアプリケーションを開発することができるようになります．ORiN2 は図 1-1 に示すように，次の三つの基本技術から構成されています．

(1) CAO(Controller Access Object)
(2) CRD(Controller Resource Definition)
(3) CAP(Controller Access Protocol)

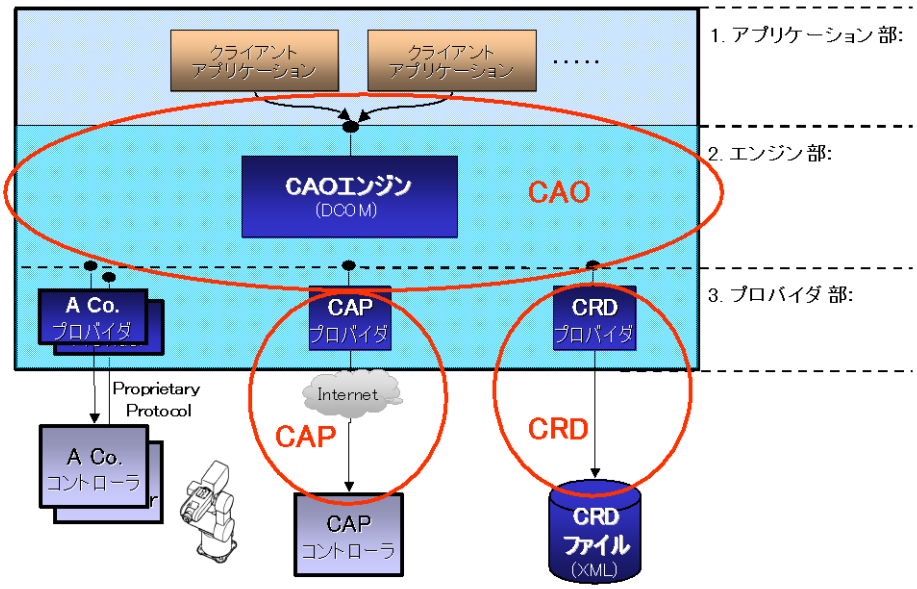



**図 1-1 ORiN2 の三つの基本技術**

CAO とは，クライアントアプリケーションに対して，ロボットコントローラにアクセスするためのインタフェースを提供する「標準プログラムインタフェース」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．

次に CRD とは，ロボットコントローラが持つリソース情報を，ロボットメーカに依存することなく XML ファイルで共有するための「標準データスキーマ」です．CAO が API の共通化であるのに対して，CRD はデータ表現の共通化を実現するものとなっています．CRD は XML 技術をベースに開発され，メーカごとに異なるさまざまなデータを 1 つの CRD スキーマで表現することができます．CAO と同様に CRD の基本データスキーマの適用範囲はロボットに限定されず，“生産情報”なども表現することができます．

CAP は，インターネット経由で CAO プロバイダにアクセスするための「インターネット向け通信プロトコル」です．CAP は SOAP(Simple Object Access Protocol)技術をベースに開発され，ORiN アプリケーション開発

者にインターネットを意識させないで遠隔のコントローラへアクセスする機能を提供します.

本書はこの三つの基本技術である CAO/CRD/CAP を対象にしたプログラミングガイドです. 第 2 章は「CAO クライアントの実装」手順を, 第 3 章では「CRD ファイルの作成」手順を, そして第 4 章では「CAP によるリモート接続」手順について解説します. それぞれの対象者を表 1-1 に示します. 各章は前半でその章に必要な基礎知識を, 後半で具体例を交えた実装方法を解説します.

**表 1-1 本書の対象者**

| 章 | 内容 | 対象 |
|---|---|---|
| 2 章(p.8) | CAO クライアントの実装 | ORiN アプリケーション開発者 |
| 3 章(p.59) | CRD ファイルの作成手引き | ロボットベンダ,<br>ORiN アプリケーション開発者 |
| 4 章(p.69) | CAP によるリモート接続 | ORiN アプリケーション開発者 |

また, 5 章では, CAO プロバイダをリモートで利用するための DCOM の設定方法を解説します. 6 章では, ORiN2 SDK 付属のツール, CaoConfig の利用方法について解説します.

# 2. CAO クライアントの実装

## 2.1. 概要

冒頭に述べたように CAO は分散オブジェクト技術をベースに開発されています. ORiN2 SDK では, この分散オブジェクト技術に, マイクロソフト社の DCOM(Distributed Compnent Object Model)を実装しています. この DCOM 版 CAO は, C++, JAVA, Visual Basic などさまざまなプログラム言語から使うことができます.

本章では, その中でも入門に最適な Visual Basic6.0(以下 VB6 と記します)を用いて解説をおこないます. 本章の構成は, 前半で VB6 の基礎知識としてオブジェクト作成の方法を解説します. 後半では, ORiN2 SDK の標準プロバイダである, Blackboard プロバイダを利用して, 変数クラスオブジェクトを扱う方法やイベント処理を使った例を交えながら解説をおこないます.

今回サンプルとして利用する Blackboard プロバイダは, 複数のクライアントアプリケーション間で変数テーブルを共有するための機能を提供しています. このプロバイダはオブジェクトとしては, 変数オブジェクトしか持っていませんが, イベントやシステム変数などの機能を備えているので, CAO プロバイダの一通りの機能を体験することができます.

以下に CAO のオブジェクトモデルを示します.

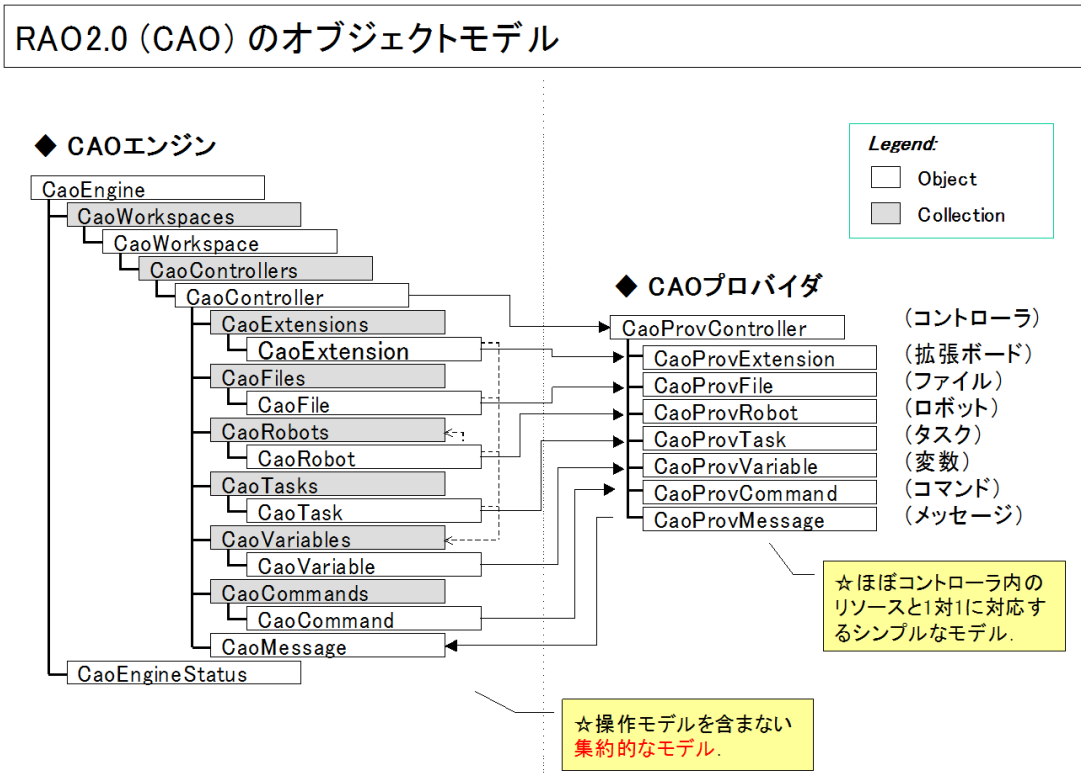



**図 2-1 CAO のオブジェクトモデル**

## 2.2. 基礎知識

### 2.2.1. アーリーバインディングとレイトバインディング

CAO クライアントにはオブジェクトの生成方法にアーリーバインディングとレイトバインディングの 2 種類の方法があります[1]. ここではこの 2 つの違いとオブジェクトの作成方法について説明します.

#### 2.2.1.1. アーリーバインディング

アーリーバインディングとは, タイプライブラリを参照することでコンパイル時にオブジェクトの型情報を取得する方法です. これにより, クライアントがオブジェクト情報を保持することでコンポーネントに型情報の問い合わせをする必要がなくなり処理速度が向上します. しかしコンパイル時にオブジェクト情報を組み込んでいるため, もしコンポーネントに変更があったときにはクライアントも再コンパイルする必要があり, 柔軟性にかけるという弱点があります.

アーリーバインディングを行うためには以下の手順をおこないます.

(1) VB6 のメニューバー内の[プロジェクト]→[参照設定]を選択します.

(2) 図 2-2 のようなダイアログが表示されますので, CAO エンジンのタイプライブラリ“CAO 1.0 タイプライブラリ”を追加します.

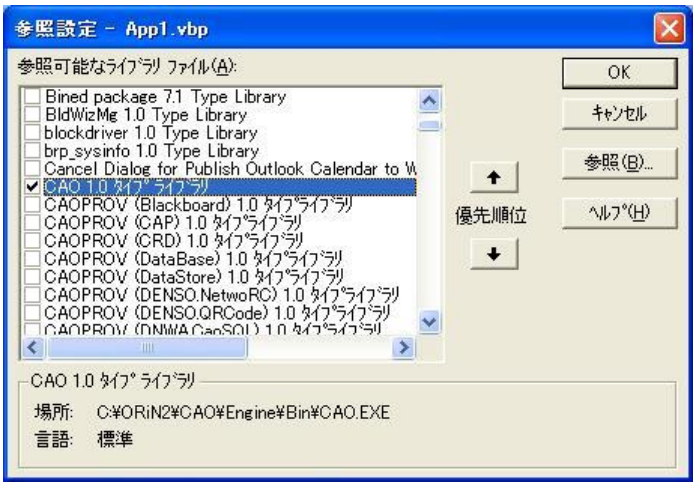



**図 2-2 VB の参照設定画面**

(3) 以下のコードを VB6 で書くことでアーリーバインディングで CaoEngine オブジェクトを生成することができます.

```
' CaoEngine 型の変数を作成します
Dim caoEng As CaoEngine
' New キーワードでインスタンスを生成して, Set ステートメントで代入します.
Set caoEng = New CaoEngine
```

ここで CaoEngine はオブジェクトなので Set ステートメントを使用しないと代入ができません[2]. また, 上記の

[1] 本書ではアーリーバインディングを使用して解説をおこないます.
[2] VB.NET では, Set ステートメントが廃止されました.

例で変数の作成時に CaoEngine 型を呼び出せるようになっているのは(2)の参照設定を行ったためです.

#### 2.2.1.2. レイトバインディング

前述のアーリーバインディングとは対照的にレイトバインディングはタイプライブラリを必要としません. よってコンパイル時にはオブジェクトの型を検査せず, 実行時に動的に検査をおこないます. このため全ての処理を実行時に行うので処理速度はアーリーバインディングには劣ります. しかしコンポーネントから完全に独立しているためにコンポーネントに変更があっても再コンパイルする必要がなく, 柔軟性に優れています.

レイトバインディングでは CreateObject を使ってオブジェクトを生成します. ここで CreateObject の定義を以下に示します.

```
CreateObject(class)
```

この関数は class で指定したオブジェクトへのポインタを返します. このときclass は“アプリケーション名. オブジェクト名”で構成されます.

レイトバインディングでオブジェクトを生成するためには以下のようなコードを書きます.

```
‘ Object 型の変数を作成
Dim caoEng As Object
‘ caoEng に CreateObject()で作成した CaoEngine オブジェクトを代入
Set caoEng = CreateObject( “CAO.CaoEngine” )
```

この例ではまず Object 型の変数を作成します. 次に CreateObject で CAO.exe の CaoEngine オブジェクトを作成しています.

### 2.2.2. オブジェクトの生成と管理

COM のオブジェクトの生成には前述の New キーワードもしくは CreateObject 関数を使用します. また作成したオブジェクトは Set ステートメントを用いて変数に代入をおこないます. そして, プログラム終了前に生成したオブジェクトを破棄する必要があります. オブジェクトを破棄する方法は変数に Nothing を代入します3.

例えば, CaoEngine オブジェクトを生成・破棄する場合は, 以下のように記述します.

```
‘ CaoEngine 型の変数を宣言
Dim caoEng As CaoEngine
‘ オブジェクトの生成
Set caoEng = New CaoEngine
‘ オブジェクトの破棄
If Not caoEng Is Nothing Then
    Set caoEng = Nothing
End If
```

ここで実際の CAO クライアントでのオブジェクト管理について説明します.

ORiN2 SDK で生成・取得することのできるオブジェクトは, A.1 CAO エンジン関数一覧 に示すものがあります. 今回のサンプルに利用する Blackboard プロバイダでは, 以下のオブジェクトを利用します.

---

[3] VB6 では, Nothing を代入することでオブジェクトを破棄することができますが, VB.NET では, 明示的にオブジェクトを開放するための関数を呼び出す必要があります. 詳細は 2.5.3 を参照してください.

- CaoEngine
- CaoWorkspace
- CaoController
- CaoVariable

これらのオブジェクトの生成は順番に行う必要があります. その順番以下に示します.

(1) CaoEngine オブジェクトを New または CreateObject()で生成します.
(2) CaoWorkspace オブジェクトを CaoEngine オブジェクトから取得します. (デフォルトワークスペース)
(3) CaoController オブジェクトを CaoWorkspace オブジェクトから生成します.
(4) CaoVariable オブジェクトを CaoController オブジェクトから生成します.

具体的な作成方法については 2.4 で解説しますが, その生成順番をまとめると図 2-3 のようになります.

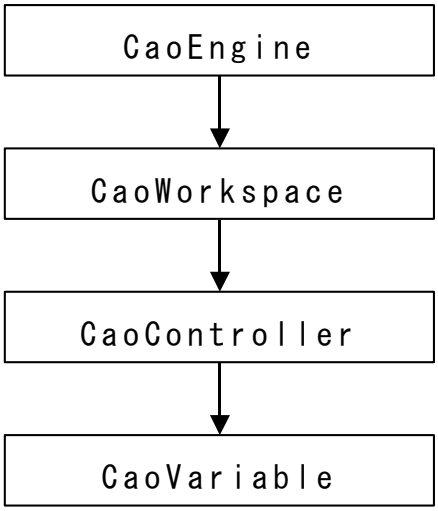



**図 2-3 オブジェクトの生成・取得順序**

ここで, 前述したとおり CAO クライアント終了時には作成したオブジェクトを全て破棄しなければなりません. もしオブジェクトの破棄が正常に行われない場合, CAO クライアント終了後も CAO.exe[4]が終了しないことがあります. CAO エンジンの破棄は, Nothing を代入するだけでよいのですが, CAO ワークスペースを含むそれ以外のオブジェクト破棄は, 生成・取得した逆の順番で, コレクションに保持されているオブジェクトを削除していきます. CAO の各オブジェクトは Add…メソッドによって作成されます. このとき, 作成されたオブジェクトは自動的にコレクションに自動的に登録されます. このため, クライアントがオブジェクトを破棄しても, コレクションに登録されていればオブジェクトは消滅しません. オブジェクトをコレクションから削除するには Remove

[4] CAO.exe とは, CAO エンジンの実行形式のファイルです. クライアントアプリケーション内で, CaoEngine オブジェクトを生成すると, 自動的に起動します.

メソッドを使用する必要があります.

以下に例を示します. この例では, CaoVariable, CaoController, CaoWorkspace, CaoEngine オブジェクトの順で破棄をおこなっています.

```
‘ caoCtrl の Variable コレクションから caoVar を削除します
If Not caoVar Is Nothing Then
    caoCtrl.Variables.remove caoVar.Index
    Set caoVar = Nothing
End If
‘ caoWS の Controller コレクションから caoCtrl を削除します
If Not caoCtrl Is Nothing Then
    caoWS.Controllers.remove caoCtrl.Index
    Set caoCtrl = Nothing
End If
‘ caoEng の Workspace コレクションから caoWS を削除します
If Not caoWS Is Nothing Then
    caoEng.Workspaces.Remove (caoWS.Index)
    Set caoWS = Nothing
End If
If Not caoEng Is Nothing Then
    Set caoEng = Nothing
End If
```

### 2.2.3. 非同期処理

CAO ではイベントを使った非同期処理をサポートしています. 例えば, 今回サンプルとして利用する Blackboard プロバイダでは, 変数が追加された場合や変数の値が変更された場合にイベントが発生し, 非同期処理を実現することができます.

非同期処理では同期処理とは異なり, プロパティの戻り値では結果を返しません. イベント発生時にオブジェクトを保持している変数のイベントプロシージャが呼ばれます. よって非同期処理のとき, クライアントは結果を受け取る処理を, イベントプロシージャに実装する必要があります.

実際に CAO の非同期処理を利用するには以下の手順でおこないます[5].

(1) CaoController オブジェクトの宣言に“WithEvents”をつけます. 以下にイベントプロシージャを持つ CaoController の宣言の例を示します.

```
Private WithEvents caoCtrl As CaoController
```

(2) イベントプロシージャに処理を追加します. イベントプロシージャ名は, コントローラ名の後ろに“_OnMessage”を追加したものとなります. 以下にイベントプロシージャの作成例を示します.

```
  Private Sub caoCtrl_OnMessage(ByVal pICaoMess As CAOLib.ICaoMessage)
‘ここに処理内容を記述してください.
  End Sub
```

ここで, イベントプロシージャの引数となっている CAOLib.ICaoMessage に実装されているメンバ変数の詳細は, 各プロバイダユーザーズガイドを参照してください. 今回利用する BlackBoard プロバイダで

[5] VB.NET では AddHandler メソッドにより, イベントプロシージャの設定をおこなう必要があります. 詳細は 2.5.4 を参照してください.

は，イベントの種類，変化した変数の名前，値などが実装されています．

### 2.2.4. COM で使用する変数型

ここでは，COM で使われる変数型の中で，特徴的でかつプロバイダの実装においてもよく使われる BSTR，VARIANT，SAFEARRAY，HRESULT の 4 つの変数型を解説します．

#### 2.2.4.1. BSTR

BSTR はワイド文字列と，その長さを DWORD 型で持つデータ型です．BSTR 型のポインタは，図 2-4 に示すように，文字列の先頭を指し示しています．

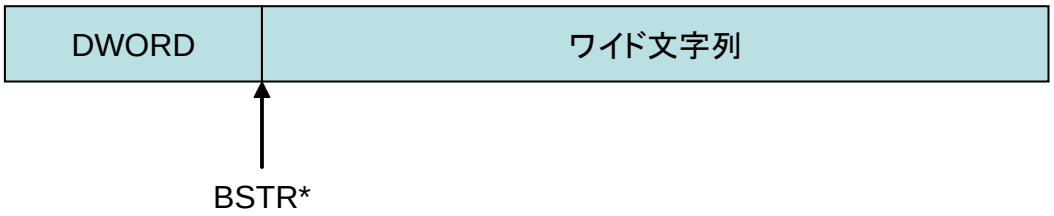



**図 2-4 BSTR の構造**

BSTR に値を代入するときは文字列の領域を確保する関数である SysAllocString()を使います．使用後は文字列の領域を SysFreeString()で解放する必要があります．また，文字列変換マクロ A2BSTR を利用することもできます．以下に BSTR に文字列“This is test.”を代入する例を示します．

```
// BSTR 型を宣言します．
BSTR bstrSample；
// 文字列 “This is test.” を代入します．
// ワイド文字列なので L“ ” で文字列を囲みます．
bstrSample = SysAllocString( L”This is test.” );
// 使い終わったら領域を解放します．
SysFreeString( bstrSample );
// 文字列変換マクロを利用して代入することもできます．
//（マクロ内でメモリが確保されています．）
bstrSample = A2BSTR( ”This is test.” );
// 使い終わったら領域を解放します．
SysFreeString( bstrSample );
```

また BSTR には CComBSTR や_bstr_t といったラッパークラスがあります．これらのクラスでは“=”を使った文字列代入などで SysAllocString()をおこない，デストラクタで SysFreeString()をおこないます．これにより，文字列の領域の確保と解放を意識せずに BSTR を使用することができます．他にもオーバーロード済みの演算子等が用意されており，例えば演算子“+=”を用いることで簡単に文字列の追加ができます．

以下に BSTR に文字列“THIS IS TEST.”を代入する例を示します.

```
// BSTR 型を宣言します.
BSTR bstrVal;
// CComBSTR を 16 文字宣言します.
CComBSTR str(16);
// “=”により文字列を代入します.
str = L"This ";
// Append メソッドにより文字列を追加します.
str.Append(L"is ");
// “+=”により文字列を追加します.
str += L"test.";
// ToUpper メソッドにより文字列を大文字に変換します.
str.ToUpper();
// CComBSTR 型から BSTR 型にコピーします.
bstrVal = str.Copy();
// CComBSTR を開放します.
delete str;
```

#### 2.2.4.2. VARIANT

VARIANT は多様なデータ型を扱うことができる構造体です. 内部はデータ型を示す VARTYPE 型の vt と, 格納される値の共用体から構成されています.

ここで基礎的なデータ型についての詳細を以下に示します.

**表 2-1 VARIANT 型の共用体(一部)**

| タイプ | 識別子 | メンバ名 | 説明 |
|---|---|---|---|
| BYTE | VT_UI1 | bVal | 文字(符号なし) |
| SHORT | VT_I2 | iVal | short(符号つき) |
| LONG | VT_I4 | lVal | long(符号つき) |
| FLOAT | VT_R4 | fltVal | float(符号つき) |
| DOUBLE | VT_R8 | dblVal | double(符号つき) |
| VARIANT_BOOL[6] | VT_BOOL | boolVal | 論理型(符号つき short として渡される) |
| SCODE | VT_ERROR | scode | ステータスコード(HRESULT) |
| DATE | VT_DATE | date | 日付(double として渡される) |
| BSTR | VT_BSTR | bstrVal | 文字列型 |
| IUnknown* | VT_UNKNOWN | punkVal | インターフェースポインタ |
| IDispatch* | VT_DISPATCH | pdispVal | IDispatch インターフェースポインタ |
| SAFEARRAY* | VT_ARRAY | parray | 配列型 |
| | VT_EMPTY | | 値が入っていない |

[6] VT_BOOL 型に値を代入する場合, 通常の BOOL 型の TRUE, FALSE を使用するのではなく, VARIANT_TRUE と VARIANT_FALSE を使用する必要があります.

この中で VT_EMPTY は値がないことを示す識別子です.

また, VARIANT は上記の値を参照渡しすることもできます. その場合, 識別子と VT_BYREF の論理和をとることにより, 参照渡しであることを識別します.

実際に VARIANT 型を使用するときには vt に使用する識別子を代入した後に, 共用体メンバに値を代入します. この代入にはマクロを利用することもできます. 以下に VARIANT に long 型の値 1000 を代入する例を示します.

```
// VRIANTDATE 型を宣言, 初期化します.
VARIANT varSample;
VariantInit( &varSample );
// データ型を long に設定します.
varSample.vt = VT_I4;
// 1000 という値を代入します.
varSample.lVal = 1000;
```

次に, マクロを利用して VARIANT に long 型の値 1000 を代入する例を示します.

```
// VRIANTDATE 型を宣言, 初期化します.
VARIANT varSample;
VariantInit( &varSample );
// データ型を long に設定します.
V_VT( &varSample ) = VT_I4;
// 1000 という値を代入します.
V_I4( &varSample ) = 1000;
```

上記の例のように新たに VARIANT を宣言したときには使用する前に必ず初期化を行う必要があります. 初期化に VariantInit()を使用すると識別子は VT_EMPTY になり, VARIANT に値が入れられていないことを示します.

また BSTR 型や後述する SAFEARRAY 型では領域の割り当てを行うため VARIANT を解放する前に関数 VariantClear()で領域の解放をする必要があります. この関数ではリソースの解放が必要かどうかを識別子によって判断しています. 以下に, VARIANT に BSTR 型の値“This is test.”を代入する例を示します.

```
// VRIANTDATE 型を宣言, 初期化します.
VARIANT varSample;
VariantClear( &varSample );
// データ型を BSTR に設定します.
varSample.vt = VT_BSTR;
// This is test. という値を代入します.
varSample.bstrVal = SysAllocString(L"This is test.");
```

VARIANT 型同士を代入するときは, 領域が割り当てられている可能性があるので, 直接代入を行わず VariantCopy()を使うべきです. この関数はコピー先メモリに新たに領域を確保してデータをそこに転送します.

VARIANT には CComVariant や_variant_t といったラッパークラスがあります. これらのクラスはコンストラクタで VariantInit()を, デストラクタで VariantClear()をおこないます. これにより開発者は VARIANT の初期化と領域の解放を意識せずに VARIANT を使用することができます. これらのラッパークラスには, 他にもオーバーロード済みの演算子等が用意されており, 例えば演算子"="を用いることで型やメモリ確保を意識せずに整数型や文字列等も直接代入することができます. 以下に, CcomVariant を利用して long 型の値 2234 を代入する例を示します.

```
// CcomVariant 型の変数を, short int 型で宣言し, 値 1234 を代入します.
CComVariant var(1234,VT_I2);
// "="を利用して, 値 2234 を代入します.
var = 2234;
// VARTYPE を VT_I2 から VT_I4 に変更します.
var.ChangeType(VT_I4);
```

#### 2.2.4.3. SAFEARRAY

SAFEARRAY とは, 主にオートメーションで使用される配列型で, 次元や要素数とデータを指すポインタのフィールドを持つ構造体です. COM では, SAFEFARRAY を使うことで, プロセス間で配列のデータの受け渡しをおこなっています. SAFEARRAY は構造体ですが, 使用するときは直接操作を行わず関数を使ってアクセスしなければなりません.

SAFEARRAY へのアクセスには以下の手順が必要となります.

(1) SAFEARRAY の準備をする.
(2) SAFEARRAY にアクセスする.
(3) SAFEARRAY とのアクセスを閉じる.

上記の手順を詳しく説明します.

(1) まず SAFEARRAYBOUND 型の構造体を宣言して生成する配列の構造を定義します. 以下に SAFEARRAYBOUND の定義を示します.

```
typedef struct tagSAFEARRAYBOUND
{
ULONG cElements;
LONG lLbound;
} SAFEARRAYBOUND;
```

cElements は配列の要素数, lLbound はインデックスの下限値をあらわしています. 例えばインデックス番号が 0 から始まる要素数 10 の配列を定義したい場合は, 以下のように記述します.

```
SAFEARRAYBOUND  bound = { 10, 0 };
```

ここで用意した SAFEARRAYBOUD は SafeArrayCreate()関数をコールする時に使用されます. SafeArrayCreate()は SAFEARRAY の実体を作る関数です. 以下に SafeArrayCreate()の定義を示します.

```
SAFEARRAY* SafeArrayCreate( VARTYPE vt, UINT cDims, SAFEARRAYBOUND rgsabound );
```

ここで vt は配列の型, cDims は配列の次元数, rgsabound は先程作成した配列の先頭アドレスです. これらを引数として SafeArrayCreate()は作成した SAFEARRAY へのポインタを返します. 以下に, SafeArrayCreate()で VT_I2 型の SAFEARRAY を作成する例を示します.

```
SAFEARRAY  pSa = SafeArrayCreate( VT_I2, 1, &bound );
```

(2) SAFEARRAY が持つデータ領域にアクセスするには SAFEARRAY の型に対応したアクセス用のポインタを用意します. すなわち VT_I4 で SAFEARRAY を作ったときは long 型のポインタを用意します. 次に SafeArrayAccessData() をコールして SAFEARRAY にアクセスします. SafeArrayAccessData()の定義を以下に示します.

```
HRESULT SafeArrayAccessData( SAFEARRAY *  psa, void HUGEP **  ppvData );
```

このとき psa はアクセスする SAFEARRAY へのポインタ, ppvData は配列データへのアクセスポインタです. これにより配列データにアクセスができるようになります. 例えば, SAFEARRAY* pSa に long* lData でアクセスする場合は以下のように記述します.

```
SafeArrayAccessData( pSa, (void**)&lData );
```

(3) SAFEARRAY への処理が終了したら, SafeArrayUnaccessData()をコールし SAFEARRAY へのアクセスを閉じる必要があります. SafeArrayUnaccessData()の定義を以下に示します.

```
HRESULT  SafeArrayUnaccessData( SAFEARRAY* psa );
```

この関数は引数に SAFEARRAY へのポインタを指定し, この引数として指定された SAFEARRAY へのアクセスを閉じます. 以下に SAFEARRAY* pSa へのアクセスを閉じる例を示します.

```
SafeArrayUnaccessData( pSa );
```

最後に SAFEARRAY が不必要になったときは SafeArratDestroy()によって SAFEARRAY に割り当てられた領域を開放します. SafeArrayDestroy()の定義は以下に示します.

```
HRESULT  SafeArrayDestroy( SAFEARRAY* psa );
```

ここで引数は解放する SAFEARRAY へのポインタとなります.

最後に, SAFEARRAY を使用した処理の全体がわかる例を示します. この例では, 作成した SAFEARRAY に short 型の配列を代入しています.

```
// short 型の配列を用意
short sample[10] = { 1, 2 ,3 ,4 ,5 ,6 ,7 ,8 ,9 ,10 };
// SAFEARRAY のポインタを宣言
SAFEARRAY* pSa;                                                    (1)
// SAFEARRAY の実体が存在しないので作成する.
SAFEARRAYBOUND  bound = { 10 ,0 };
pSa = SafeArrayCreate( VT_I2, 1, &bound );
// SAFEARRAY にアクセスする.
short* iArray;
SafeArrayAccessData( pSa, (void**)&iArray);                        (2)
for( int i = 0; i < 10; i++){
       iArray[ i ] = sample[ i ];
}
SafeArrayUnaccessData( pSa );                                      (3)
```

上記の例は short 型の配列を SAFEARRAY の配列にコピーする処理です.

まず SAFEARRAY の実体が存在しないので作成する必要があります. そのために, リスト中の(1)に示すように, 作成した SAFEARRAY へのポインタ(pSa)を用意します. 次に SAFEARRAYBOUND 型で配列の構成を定義します. ここでは要素数 10, インデックス下限値は 0 とする配列を定義しています. そして SafeArrayCreate()により SAFEARRAY を作成します. 今回格納するデータは short 型の配列なので, SAFEARRAY の型は VT_I2 となります.

次に SAFEARRAY のデータにアクセスするためにポインタを用意します. リスト中の(2)に示すように, 今回は SafeArrayCreate()で VT_I2 の SAFEARRAY を作っているので short 型のポインタ(iArray)を用意します. そして, SafeArrayAccessData()関数を呼び出します. ここで 2 番目の引数を(void**)とし関数の型に合わせたキャストを行っています.

SAFEARRAY にアクセスしている間は SAFEARRAY のデータへのポインタ(iArray)は配列データの先頭のアドレスを保持しています. 例では short 型の配列のデータを SAFEARRAY にコピーしています. そしてコピーが終わると, リスト中の(3)に示すように, SafeArrayUnaccessData()でアクセスを閉じます.

上記例のように配列を一括で設定するのではなく，要素ごとに一つずつ設定したい場合にはSafeArrayPutElement()を用います．SafeArrayPutElement()の定義を以下に示します．

```
HRESULT SafeArrayPutElement( SAFEARRAY * psa, long * rgIndices, void * pv );
```

ここで，psa はアクセスする SAFEARRAY へのポインタ，rgIndices はアクセスする SAFEARRAY のインデックス，pv は設定する値です．

以下に，作成した SAFEARRAY に short 型の配列を代入する例を示します．

```
// short 型の配列を用意
short sample[10] = { 1, 2 ,3 ,4 ,5 ,6 ,7 ,8 ,9 ,10 };
// SAFEARRAY のポインタを宣言
SAFEARRAY* pSa;
// SAFEARRAY の実体が存在しないので作成する．
SAFEARRAYBOUND  bound = { 10 ,0 };
pSa = SafeArrayCreate( VT_I2, 1, &bound );
// SAFEARRAY にアクセスする．
for( int i = 0; i < 10; i++){
       SafeArrayPutElement( psa, &i, &sample[i] );
}
```

#### 2.2.4.4. HRESULT

HRESULT はエラーコードを格納するための構造を備えた 32 ビットの数字データです．HRESULT が持つエラーの種類は<winerr.h>によって多数用意されています．比較的重要と思われるエラーコードの一覧については表 2-4 を参照してください．

HRESULT にエラーコードを与えるには代入をおこないます．しかし関数の成功，失敗を判定するときに HRESULT を直接比較することは好ましくありません．SUCCEEDED()，FAILED()というマクロが用意されているので，これを使用してください．SUCCEEDED()は引数の HRESULT が正常終了であれば真，それ以外は偽の値を返し，FAILED()は引数の HRESULT が異常終了であれば真，それ以外は偽の値を返します．

以下に HRESULT に S_OK を代入して，FAILED()を使って判定する例を示します．

```
// HRESULT 型の hr に S_OK を代入
HRESULT  hr = S_OK;
// hr が S_OK かどうかを FAILED()で判定
if( FAILED( hr ) )
{
    // エラー処理
}
return hr;
```

### 2.2.5. データの記述方式

ORiN2 では，VARIANT 型を文字列で表現する方法を規定しています．これにより，VARIANT 型に対応していない環境においても，擬似的に Variant 型を扱えるようになります．このデータ記述方式は，RAC の送信文字列や，CaoUPnP のアイテム値の表現などに利用しています．

この書式は，以下に示すようにデータ型とデータ列をカンマ区切りで表現します．．

<データ型>,<データ列>

ここで<データ型>には VARTYPE 型で表記される整数値を示します．表 2-2 に使用できるデータ型とその

値を示します.

**表 2-2 使用できるデータ型**

| データ型 | 値 | 意味 |
|---|---|---|
| VT_I2 | 2 | 2 バイト整数型 |
| VT_I4 | 3 | 4 バイト整数型 |
| VT_R4 | 4 | 単精度浮動小数点型 |
| VT_R8 | 5 | 倍精度浮動小数点型 |
| VT_CY | 6 | 通貨型 |
| VT_DATE | 7 | 日付型 |
| VT_BSTR | 8 | 文字列型 |
| VT_BOOL | 11 | ブール型 |
| VT_VARIANT | 12 | VARIANT 型 |
| VT_UI1 | 17 | バイト型 |
| VT_ARRAY | 8192 | 配列型 |

データ型が配列のときは, VT_ARRAY とデータ型の論理和で表記します.
データ列にはデータを文字列で表記します. 配列データの表記は“, ”(カンマ)で区切って表記します.

| | | | | | |
|---|---|---|---|---|---|
| 例 1) | 2,100 | 型: | VT_I2 | 値: | 100 |
| 例 2) | 8,Sample | 型: | VT_BSTR | 値: | Sample |
| 例 3) | 8194,100,200,300 | 型: | VT_I2 \| VTARRAY | 値: | 100, 200, 300 |
| 例 4) | 8200,Sample,Test | 型: | VT_BSTR \| VTARRAY | 値: | “Sample”, “Test” |
| 例 5) | 8,Sample,Test | 型: | VT_BSTR | 値: | “Sample,Test” |

### 2.2.6. ログ出力

CAO エンジンには, CAO.exe のスタートやストップ, オブジェクトの生成・破棄などのなんらかの動作が行われた状態で, その記録をおこなうログ機能. ログの出力先には, コンソール, メッセージボックス, イベントビューア, デバッグビューア, テキストファイルの 5 種類を選択でき, この設定は CaoConfig でおこなうことができます. CaoConfig の利用方法については,「6CaoConfig」を参照してください.

CAO エンジンのログ出力タイミングには, おもに以下のようなものがあります.

- CAO エンジンの起動時, 終了エラー時.
- CaoController や CaoVariable などのオブジェクト生成時, 破棄時.
- コントローラのスレッド, 起動/終了, 稼動/停止時.

・ CAO プロバイダからのメッセージイベントによるログ出力時[7].

### 2.2.7. エラーコード

ORiN2 のエラーは，発生箇所となるモジュールが複数箇所に分かれています．以下に CAO モジュールで発生するエラーを示します．

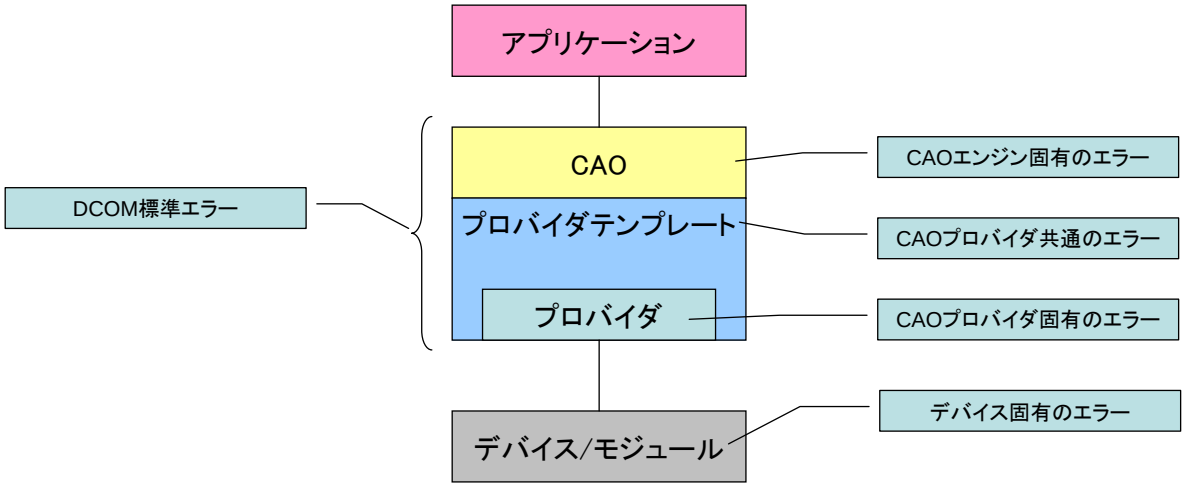



**図 2-5 ORiN2 で発生するエラー**

**表 2-3 ORiN2 で発生するエラー種別**

| エラー種別 | 発生箇所 | 説明 |
|---|---|---|
| DCOM 標準エラー | - | DCOM で発生する一般的なエラーです．<br>詳細は表 2-4 を参照してください． |
| CAO エンジン固有エラー | CAO エンジン | CAO エンジンで独自に定義されたエラーです．<br>詳細は，表 2-5 を参照してください． |
| CAO プロバイダ共通エラー | CAO プロバイダテンプレート | CAO プロバイダテンプレートで独自に定義されたエラーです．<br>詳細は，表 2-6 を参照してください． |
| CAO プロバイダ固有エラー | CAO プロバイダ | 各 CAO プロバイダで独自に定義されたエラーです．<br>詳細は，各プロバイダのユーザーズガイドを参照してください． |

[7] CAO プロバイダのメッセージイベントによりログを出力する方法は，『CAO プロバイダ作成ガイド』の「3.4.1.メッセージイベントによるログ出力」を参照してください．

| | | |
|---|---|---|
| デバイス固有エラー | デバイス | デバイスドライバ固有のエラーです.<br>詳細は, 各デバイスドライバのマニュアルを参照してください. |

**表 2-4 DCOM 標準エラー(一部)**

| エラー名 | エラー番号 | 説明 |
|---|---|---|
| S_OK | 0x00000000 | 正常終了(0x0)論理型の TRUE 値を返す. |
| S_FALSE | 0x00000001 | 正常終了(0x1)論理型の FALSE 値を返す. |
| E_UNEXPECTED | 0x8000FFFF | 破局的な障害が発生した. |
| E_NOTIMPL | 0x80004001 | 実装されていない. |
| E_OUTOFMEMORY | 0x8007000E | 必要なメモリの割り当てに失敗 |
| E_INVALIDARG | 0x80070057 | 無効な引数が 1 つ以上存在している. |
| E_POINTER | 0x80004003 | 関数に無効なポインタが渡された. |
| E_HANDLE | 0x80070006 | 関数に無効なハンドルが渡された. |
| E_ABORT | 0x80004004 | 処理が中止された. |
| E_FAIL | 0x80004005 | 予期しない障害が発生した. |
| E_ACCESSDENIED | 0x80070005 | 一般的なアクセス拒否エラーが発生した. |
| E_WINDOWS_MASK | 0x8007xxxx | Windows の標準エラーコード(16bit)を xxxx(下位 2 バイト)に格納します.<br>例)<br>エラーコード: 2 (ファイルが見つからない)<br>→ 0x80070002 |

**表 2-5 CAO エンジン固有エラー**

| エラー名 | エラー番号 | 説明 |
|---|---|---|
| E_CAO_SEM_CREATE | 0x80000200 | 同期セマフォの生成に失敗しました. |
| E_CAO_PROV_INVALID | 0x80000201 | CAO プロバイダ名が不正です. |
| E_CAO_COMPUTER_NAME | 0x80000202 | コンピュータ名の取得に失敗しました. |
| E_CAO_VARIANT_TYPE_NOSUPPORT | 0x80000203 | VARIANT 型にサポートされていない型が指定されました. |
| E_CAO_OBJECT_NOTFOUND | 0x80000204 | 対応するオブジェクトが見つかりませんでした. |
| E_CAO_COLLECTION_REGISTERED | 0x80000205 | 既にコレクションに登録されています. |

| | | |
|---|---|---|
| E_CAO_THREAD_CREATE | 0x80000207 | ワーカースレッドの生成に失敗しました. |
| E_CAO_REMOTE_ENGINE | 0x80000208 | リモート CAO へのアクセスに失敗しました. |
| E_CAO_REMOTE_PROVIDER | 0x80000209 | リモート CAO プロバイダへのアクセスに失敗しました. |
| E_CAO_NOT_WRITABLE | 0x8000020a | 書き込みできません. |
| E_CAO_CMD_EXECUTE | 0x8000020b | コマンドが実行中です. |
| E_CAO_PROV_NO_LICENSE | 0x8000020c | プロバイダに対応したライセンスがありません. |
| E_CAO_PRELOAD | 0x8000020d | CRD プレロードに失敗しました. |
| E_CAO_TEMP_REGISTERED | 0x8000020e | コレクションに一時登録されています. |
| E_CAO_NO_PARENT | 0x8000020f | 親オブジェクトが存在しません. |

**表 2-6 CAO プロバイダ共通エラー**

| エラー名 | エラー番号 | 説明 |
|---|---|---|
| E_CRDIMPL | 0x80000400 | CRD ファイルが不正です. |
| E_CAOP_SYSTEMNAME_INVALID | 0x80000401 | システム名が不正です. |
| E_CAOP_SYSTEMTYPE_INVALID | 0x80000402 | システムタイプが不正です. |
| E_CANCEL | 0x80000403 | コマンドがキャンセルされました. |
| E_CAOP_NOT_WRITABLE | 0x80000404 | 書き込みできません. |
| E_TIMEOUT | 0x80000900 | タイムアウト |
| E_NO_LICENSE | 0x80000901 | ライセンスがありません. |
| E_NOT_CONNECTED | 0x80000902 | 接続が確立していません. |
| E_NOT_USE | 0x80000903 | 使用できません. |
| E_INVALID_CMD_NAME | 0x80000904 | コマンド名が不正です. |
| E_MAX_OBJECT | 0x80000905 | 生成可能なオブジェクト数の上限を超えています. |
| E_OVERLOADING | 0x80000906 | 設定内容が重複しています |
| E_NONE_CONNECT_OPTION | 0x80000907 | 接続オプションがありません. |
| E_ALREADY_REGISTER | 0x80000908 | 既に登録されています. |
| E_TOO_MUCH_DATA | 0x80000909 | データサイズが大きすぎます. |
| E_FAILED_ALLOC | 0x8000090a | バッファの確保に失敗しました. |
| E_MAX_CONNECT | 0x8000090b | 接続数が上限を超えています. |
| E_CONVERT_CHAR_CODE | 0x8000090c | 文字コードの変換に失敗しました。 |
| ~~E_WINDOWS_MASK~~ | ~~0x8090xxxx~~ | ~~Windows の標準エラーコード(16bit)を xxxx (下位 2 バイト)に格納します.~~ |

| | | |
|---|---|---|
| | | ~~例)~~<br>~~エラーコード：2 （ファイルが見つからない）~~<br>~~→ 0x80900002~~<br>(注)<br>このエラーコードは 0x8007xxxx に修正されました. 詳細は表 2-4 を参照してください. |
| E_WINSOCK_MASK | 0x8091xxxx | Winsock のエラーコード(16bit)を xxxx（下位 2 バイト)に格納します.<br>例）<br>エラーコード： 10061 （接続拒否）<br>→ 0x8091274D |

## 2.3. 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.

## 2.4. CAO クライアントの実装

### 2.4.1. コントローラのオープン

コントローラをオープンするには以下の手順を取ります.

(1) オブジェクトを保持するための変数を用意します.
(2) CaoEngine オブジェクトを生成します.
(3) CaoWorkspace オブジェクトを取得もしくは生成します.
(4) CaoController オブジェクトを生成します.

以下に詳しい内容を説明します. ここではアーリーバインディングをおこない, オブジェクトの生成には New キーワードを用います.

(1) 最初にオブジェクトを保持するために変数を宣言します. ここでコントローラオープンに必要なオブジェクトは CaoEngine オブジェクトと CaoWorkspace オブジェクトです. また, AddController メソッドによって CaoController オブジェクトが生成されます. よってこの三つに対応する変数を用意します. 以下に, プライベート変数として各オブジェクト型の変数を宣言する例を示します.

```
Private caoEng As CaoEngine      ' Engin オブジェクト
Private caoWs As CaoWorkspace    ' CaoWorkSpace オブジェクト
Private caoCtrl As CaoController ' Controlle オブジェクト
```

(2) 次に CaoEngine オブジェクトを生成します. これは前述の 2.2.1.1 と同様に New キーワードで生成して, Set ステートメントで変数に代入します.

```
Set caoEng = New CaoEngine
```

(3) CaoEngine オブジェクトは生成時にデフォルトの CaoWorkspaces, CaoWorkspace オブジェクトを一つずつ作成しています. デフォルトの CaoWorkspace オブジェクトを取得するときは CaoWorkspaces.Item(0)を使用します. 以下にデフォルトの CaoWorkspace オブジェクトを取得する例を示します.

```
Set caoWS = caoEng.CaoWorkspaces.Item(0)
```

(4) CaoController オブジェクトは CaoWorkspace オブジェクトの AddController メソッドで生成することができます. ここで AddController メソッドの定義を示します.

```
AddController(BSTR bstrController, BSTR bstrProvider, BSTR bstrMachine, BSTR bstrPara)
```

このメソッドの引数は, 左からロボットコントローラ名, プロバイダ名, マシン名, パラメータとなっています. そして戻り値として, CaoController オブジェクトを返します. ここで登録されているプロバイダ名は CaoEngine オブジェクトの ProviderNames メソッドを用いて配列で取得することができます. また, パラメータはプロバイダごとに内容が異なるので, 詳細は各プロバイダユーザーズガイドを参照してください. 以下に Blackboard プロバイダを利用した AddController メソッドの使用例を示します.

```
‘ CaoCtrl と CaoWS はオブジェクトを保持するための変数です
Set CaoCtrl = CaoWS.AddController(“bb1”,“CaoProv.Blackboard”, “”, “”)
‘ 第 3 引数, 第 4 引数は省略することも可能です
Set CaoCtrl = CaoWS.AddController(“bb1”,“CaoProv.Blackboard”)
```

### 2.4.2. 変数の取得と設定

変数を取得・設定するためには, 先に CaoVariable オブジェクトを生成する必要があります.

(1) CaoVariable オブジェクトを生成するためには, CaoController オブジェクトの AddVaribale メソッドを用います. 以下にその定義を示します.

```
AddVariable(BSTR bstrName, BSTR bstrOption )
```

このとき, bstrName は変数名, bstrOption は変数取得時のオプションです. このメソッドは戻り値として CaoVariable オブジェクトを返します. 以下にこのメソッドの使用例を示します. ここで, bstrOption は省略することが可能です. 以下に, 変数名“Val”の CaoVariable オブジェクトを取得する例を示します.

```
‘ CaoCtrl と CaoVar はオブジェクトを保持するための変数
Set CaoVar = CaoCtrl.AddVariable(“Val”)
```

CaoVariable オブジェクトを生成した後, 変数の値を設定・取得するためには, CaoVariable オブジェクトの Value プロパティを参照します. 以下に Long 型の値“1000”を設定した後, 変数を取得する例を示します.

```
‘ CaoCtrl と CaoWS はオブジェクトを保持するための変数
Dim iData as Integer
IData = 1000
‘ 変数に iData の値を設定します.
```

```
CaoVar.Value = iData
‘ 変数の値を取得して iData に代入します.
iData = CaoVar.Value
```

### 2.4.3. システム変数の取得と設定

プロバイダによっては，プロバイダ特有のシステム変数が用意されていることがあります．今回利用する Blackboard プロバイダでは，以下のようなシステム変数があります．詳細については，各プロバイダのユーザーズガイドを参照してください．

- @COUNT　　ユーザ変数の数
- @CURRENT_TIME　　現在の時刻
- @VERSION　　バージョン

以下に，システム変数として現在の時刻を取得する例を示します．

```
‘ システム変数@CURRENT_TIME を取得する
Set caoVar = caoCtrl.AddVariable("@CURRENT_TIME")
‘ 変数オブジェクトから現在の時刻を取得する
Dim var as Date
var = caoVar.Value
```

### 2.4.4. イベント処理

先にも述べたように，Blackboard プロバイダでは，変数が追加された場合や変数の値が変更された場合にイベントが発生します．

イベントの発生を確認するために，簡単なサンプルプログラムを作成します．まずは，VB6 でテキストボックスを三つ持ったフォームを作成し，以下のようなコードを記述してください．

**List 2-1　Form1.frm**

```
Dim caoEng As CaoEngine
Dim caoWs As CaoWorkspace
Dim WithEvents caoCtrl As CaoController

’ CAO エンジンの生成と Blackboard プロバイダへの接続
Private Sub Form_Load()
    Set caoEng = New CaoEngine
    Set caoWs = caoEng.Workspaces(0)
    Set caoCtrl = caoWs.AddController("BB1", "CaoProv.Blackboard")
End Sub


‘ Blackboard コントローラのイベントプロシージャ
Private Sub caoCtrl_OnMessage(ByVal pICaoMess As CAOLib.ICaoMessage)
    Text1.Text = pICaoMess.Description
    Text2.Text = pICaoMess.Destination
    Text3.Text = pICaoMess.Value
End Sub
```

それでは，アプリケーションを起動し，イベントの発生を確認してみましょう．

まず，作成したアプリケーションを起動すると，図 2-6 に示すように，テキストボックスには，デフォルトの文字列しか表示されていません．


**図 2-6 サンプルアプリケーションの起動**

次にイベントを発生させます．イベントの発生は，ORiN2 SDK に付属している CaoTester を利用します．CaoTester の利用方法については，CaoTester ユーザーズガイドを参照してください．

まず，CaoTester を起動し，ワークスペース子ウィンドウに図 2-7 に示すように，コントローラ名(Controller Name)に“BB1”と入力し，プロバイダ名(Provider Name)に“CaoProv.Blackboard”を選択して，Add ボタンを押下してください．

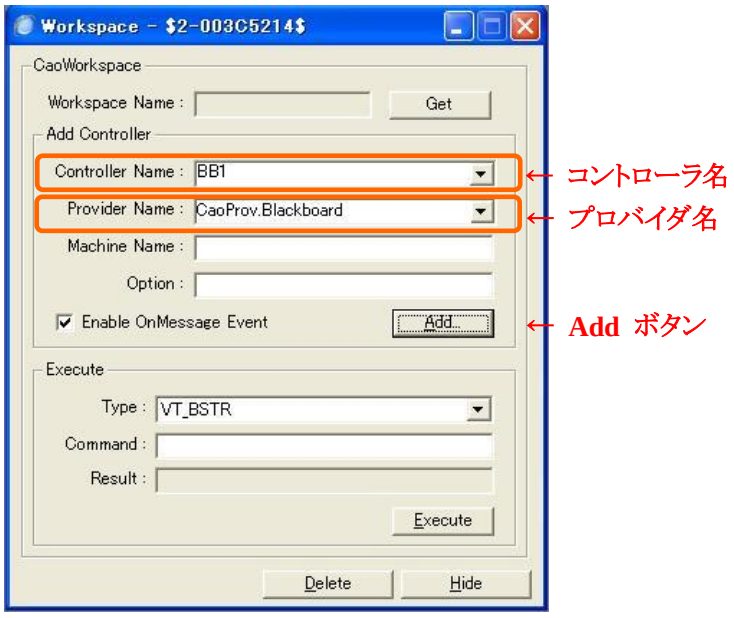



**図 2-7 Blackboard プロバイダへの接続**

Add ボタンを押下すると，図 2-8 に示すようなコントローラ子ウィンドウが作成されます．ここで，図に示すように，Variable タブを選択し，変数名(AddVariable の Name)に“test”と入力し，Add ボタンを押下してください．

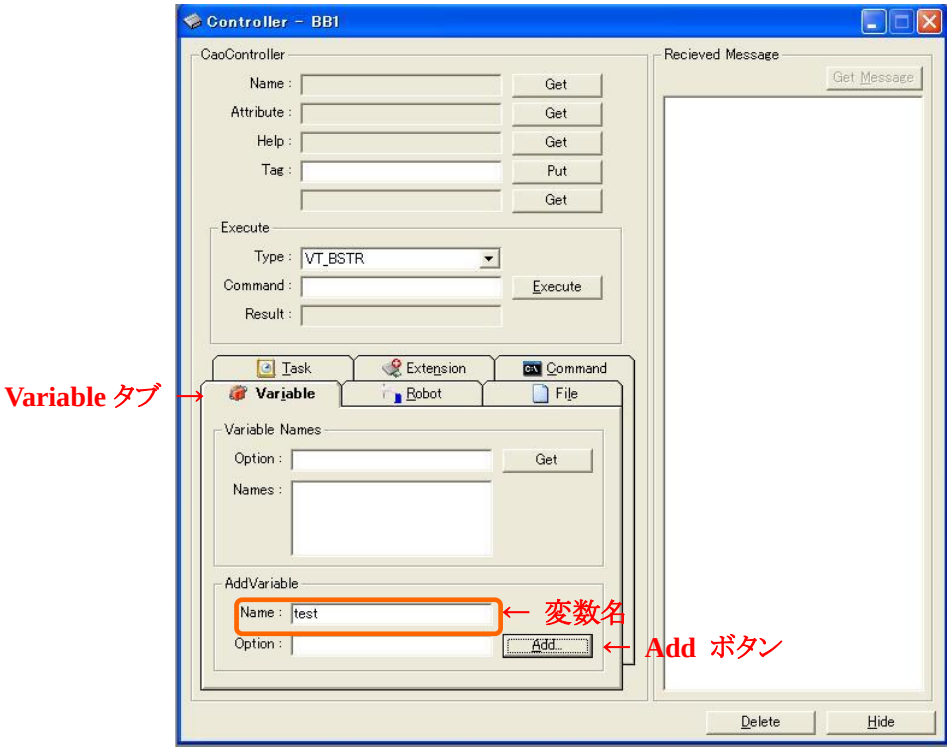



**図 2-8 変数オブジェクトの追加**

Add ボタンを押下すると，図 2-9 に示すような，変数子ウィンドウが表示されます．ここで，変数の型(Type)に“VT_I4”，変数値(Value)に“1234”と入力し，Put ボタンを押下してください．

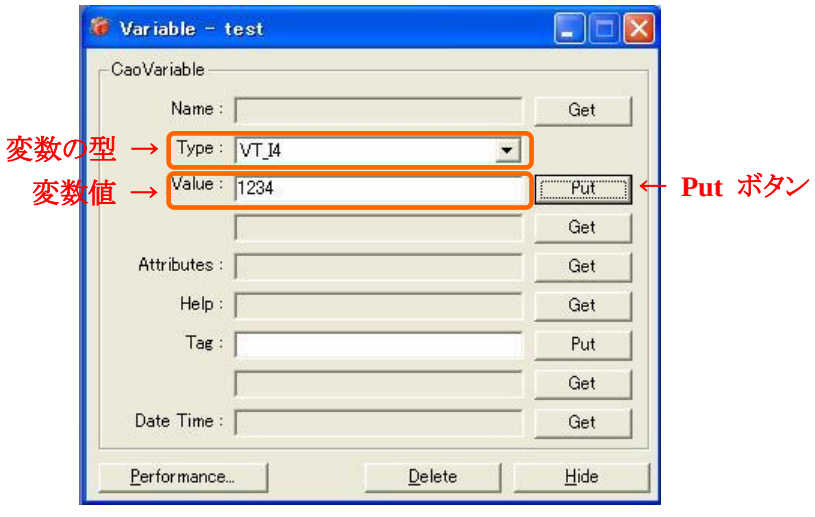



**図 2-9 変数の書込み**

ここで，最初に起動していた，サンプルプログラムのウィンドウを見てください．図 2-10 に示すように，イベ

ントが発生し，イベントプロシージャの処理により，テキストボックスに，イベントの種類，変数の名前，変数の値が表示されていることが分かります．


**図 2-10 イベントの発生**

## 2.5. VB.NET によるクライアント作成

### 2.5.1. RCW の GAC への登録

VB.NET クライアントは，Runtime Callable Wrapper (RCW) と呼ばれるモジュールを介して，COM コンポーネントにアクセスをおこないます．この RCW モジュールは，クライアントアプリケーション作成時に，自動的にタイプライブラリからローカルに生成されます．そのため，アプリケーション毎に RCW モジュールが出来てしまうため，ORiN2 SDK では，Global Assembly Cache (GAC) に登録可能な RCW モジュールの提供をおこなっています．

ORiN 用のRCWをGACに登録する方法は以下の3つの方法があります．

(1) ORiN2SDK インストーラによる自動登録

ただし，以下の条件を満たしている必要があります

- ORiN2SDK インストーラがバージョン 2.1.2 以降である．
- Microsoft .NET Framework 2.0 以降がインストールされている．

(2) Windows エクスプローラで Windows¥assembly ディレクトリに CaoRCW.dll をドロップする

(3) グローバルアセンブリキャッシュツール（Gacutil.exe）を使う[8]

登録 : ORiN2¥DotNet¥RCW> gacutil –i CaoRCW.dll

登録抹消 : ORiN2¥DotNet¥RCW> gacutil –u CaoRCW

### 2.5.2. プロジェクト作成時の設定

IDE の「参照の追加」ダイアログで CAO の RCW モジュールを追加します．RCW モジュールは以下の場所に格納してあります．

ORiN2¥DotNet¥RCW¥CaoRCW.dll

[8] gacutil.exe は，.NET Framework 2.0 SDK または Visual Studio 2005 が既にインストールしてある場合のみ使用することができます．

RCW モジュールを追加することで, CAO エンジンのクラス変数を下記のように宣言することができるようになります.

```
Dim caoEng As New ORiN2.interop.CAO.CaoEngine
```

名前空間に<ORiN2.interop.CAO>を指定することで, 上記の宣言を簡素化することが出来ます. 名前空間の指定する場合は, [プロジェクトのプロパティ]→[参照]→[インポートされた名前空間]のリストの中から<ORiN2.interop.CAO>にチェックを入れてください.

### 2.5.3. オブジェクト削除時の注意点

VB.NET では, 自動的にガーベジコレクションされるので, 変数に“Nothing”を代入してもオブジェクトの Release 処理がすぐ実行されるわけではありません. そのため, オブジェクトの削除をすぐに実行させるためには, system.Runtime.InteropServices.Marshal.ReleaseComObject 関数を使って明示的に解放処理を行う必要があります. 以下にコントローラオブジェクトを解放する例を示します.

```
caoCtrls.Remove(caoCtrl.Index)
System.Runtime.InteropServices.Marshal.ReleaseComObject(caoCtrl)
caoCtrl = Nothing
```

system.Runtime.InteropServices.Marshal.ReleaseComObject 関数はガーベジコレクタの管理下にある COM の参照カウンターをひとつ減らす役割をします. 参照の数がn回ならオブジェクトを解放するには ReleaseComObject 関数をn回呼び出す必要があります.

また, ガーベジコレクタは自動変数も管理下に置くので,

```
caoEng.Workspaces.Item(0).Controllers.Clear
```

の, [.Workspaces] も一時的な自動変数が割り当てられ管理下に置かれます. [.Item(0)] も [.Controllers] も同様です. その結果としてこれらの参照カウンターが一つ増加しますので, 解放するには ReleaseComObject 関数をこの数分だけ余分に呼び出す必要があります.

つまり, 自動変数に対する参照の数を考慮する必要があるわけですが, これをコーディングレベルでやるのは漏れもあり大変です. そこで, COM の自動変数割り当てがないように対応するローカル変数を明示的に宣言して, 以下のように管理するようにします.

```
Dim caoEng As CaoEngine
Dim caoWss As CaoWorkspaces
Dim caoWs As CaoWorkspace
Dim caoCtrls As CaoControllers

caoEng = new CaoEngine
caoWss = caoEng.Workspaces
caoWs = caoWss.Item(0)
caoCtrls = caoWs.Controllers

Dim caoCtrl As CaoController
caoCtrl = caoWs.AddController("RC1", " CaoProv.Blackboard ", "", "")

caoCtrls.Clear
```

```
System.Runtime.InteropServices.Marshal.ReleaseComObject(caoCtrl)
caoCtrl = Nothing

caoWss.Clear
System.Runtime.InteropServices.Marshal.ReleaseComObject(caoCtrls)
caoCtrls = Nothing
System.Runtime.InteropServices.Marshal.ReleaseComObject(caoWs)
caoWs = Nothing
System.Runtime.InteropServices.Marshal.ReleaseComObject(caoWss)
caoWss = Nothing
System.Runtime.InteropServices.Marshal.ReleaseComObject(caoEng)
caoEng = Nothing
```

### 2.5.4. イベント取得方法

イベントを取得する際には VB6 のように WithEvetns を用いてイベントハンドラを記述することができます. しかし, 例えばCaoControllerクラスのオブジェクトを明示的に削除したい場合にはイベントハンドラにより上手く削除できません. その場合は, AddHandler, RemoveHandler キーワードを使用して動的にイベントハンドラを設定する必要があります. イベントハンドラの設定手順は以下のようになります.

(1) 生成したコントローラオブジェクトのイベントハンドラを作成する.

```
caoCtrl = caoWs.AddController("RC1", "CaoProv.Blackboard", "", "")
AddHandler caoCtrl.OnMessage, AddressOf caoCtrl_OnMessage
```

(2) 作成したイベントハンドラの追加

```
Private Sub caoCtrl_OnMessage(ByVal pICaoMess As ORiN2.interop.CAO.CaoMessage)
txtReceiveTime.Text = pICaoMess.DateTime
End Sub
```

(3) コントローラオブジェクトの削除前にイベントハンドラを破棄する.

```
RemoveHandler caoCtrl.OnMessage, AddressOf caoCtrl_OnMessage
caoCtrls.Remove(caoCtrl.Index)
System.Runtime.InteropServices.Marshal.ReleaseComObject(caoCtrl)
caoCtrl = Nothing
```

また, この OnMessage イベントはメインスレッドとは別のスレッドにより処理されますので, スレッドセーフなプログラミングに注意してください. 例えば, メインフォームに貼り付けられた TextBox1 の Text プロパティの値をOnMessageイベントハンドラ内で直接代入しようとしても「有効ではないスレッド間の操作: コントローラが作成されたスレッド以外のスレッドからコントロール'TextBox1'がアクセスされました. 」というエラーが発生します. この場合,

```
TextBox1.CheckForIllegalCrossThreadCalls = False
```

を実行することでアクセス可能にすることもできますが, そもそもスレッドセーフなプログラミングを意識してください.

## 2.6. その他言語によるクライアント作成

### 2.6.1. C++によるクライアント作成[9]

C++から CAO のオブジェクトを操作するには以下の手順をとります.

(1) DCOM の初期化

(2) CaoEngine オブジェクトの生成

(3) メソッドの実行

(4) オブジェクトの解放

(5) DCOM の終了処理

また CAO オブジェクトを扱う方法は以下の 2 種類があります.

・ #include ディレクティブ

・ #import ディレクティブ

各方法の詳細については以下に示します.

#### 2.6.1.1. #include ディレクティブの場合

C++から CAO のオブジェクトを操作するには, CAO のヘッダファイル(CAO.h)と UUID 定義ファイル(CAO_i.c)をインクルードする必要があります. これらのファイルは“<ORIN2ルート>¥CAO¥Include”にあります.

```
#include "CAO.h"
#include "CAO_i.c"
```

以下は実際に CAO オブジェクトを操作する手順になります.

(1) DCOM の初期化

CAO オブジェクトを操作する前に CoInitialize()を実行します.

(2) CaoEngine オブジェクトの生成

CoCreateInstance()を実行します. 以下に引数の例を示します.

```
ICaoEngine* pEng;
hr = CoCreateInstance(CLSID_CaoEngine,
                      NULL,
                      CLSCTX_LOCAL_SERVER,
                      IID_ICaoEngine,
                      (void **)&pEng);
```

(3) メソッドの実行

(2)で生成した CaoEngine のメソッドを実行します.

以下に AddController を行う例を示します.

```
// CaoWorkspace コレクションの取得
ICaoWorkspaces *pWss;
```

[9] DCOM の詳細な説明については以下のページを参照してください.
“http://www.microsoft.com/japan/msdn/library/default.asp?url=/japan/msdn/library/ja/jpdnguion/htm/msdn_drguion020298.asp”

```
hr = pEng->get_Workspaces(&pWss);
if (FAILED(hr)) return hr;

// CaoWorkspace の取得
ICaoWorkspace *pWs;
hr = pWss->Item(CComVariant(0L), &pWs);
if (FAILED(hr)) return hr;

// CaoController の生成
ICaoController *pCtrl;
hr = pWs->AddController(CComBSTR(L"TestCtrl"),
                        CComBSTR(L"CaoProv.DataStore"),
                        CComBSTR(L""),
                        CComBSTR(L""),
                        &pCtrl);
```

(4) オブジェクトの解放

プログラム内で生成，取得したオブジェクトは必ず解放処理をしなければなりません．解放は各オブジェクトの Release()を実行します．以下に(3)までに取得したオブジェクトの解放を行う例を示します．

```
pCtrl->Release();
pWs->Release();
pWss->Release();
pEng->Release();
```

(5) DCOM の終了処理

プログラムが終了する前に CoUnitialize()を実行します．

以下に変数の設定，取得を行うサンプルプログラムを示します．

**List 2-2 CPPSample1.cpp**

```
#include "atlbase.h"
#include "CAO.h"
#include "CAO_i.c"

HRESULT func1();

int main(int argc, char* argv[])
{
	CoInitialize(0);

	HRESULT hr = func1();

	CoUninitialize();
	return 0;
}

HRESULT func1()
{
	HRESULT hr = S_OK;
	ICaoEngine* pEng = NULL;
	ICaoWorkspaces *pWss = NULL;
	ICaoWorkspace *pWs = NULL;
	ICaoController *pCtrl = NULL;
	ICaoVariable *pVar = NULL;
	CComVariant vntVal;

	// CaoEngine の生成
	hr = CoCreateInstance(CLSID_CaoEngine,
	                      NULL,
```

```
                                        CLSCTX_LOCAL_SERVER,
                                        IID_ICaoEngine,
                                        (void **)&pEng);
        if (FAILED(hr)) {
                goto EndProc;
        }

        // CaoWorkspace コレクションの取得
        hr = pEng->get_Workspaces(&pWss);
        if (FAILED(hr)) {
                goto EndProc;
        }

        // CaoWorkspace の取得
        hr = pWss->Item(CComVariant(0L), &pWs);
        if (FAILED(hr)) {
                goto EndProc;
        }

        // CaoController の生成
        hr = pWs->AddController(CComBSTR(L"TestCtrl"),
                                        CComBSTR(L"CaoProv.DataStore"),
                                        CComBSTR(L""),
                                        CComBSTR(L""),
                                        &pCtrl);
        if (FAILED(hr)) {
                goto EndProc;
        }

        // CaoVariable の生成
        hr = pCtrl->AddVariable(CComBSTR(L"TestVal"), CComBSTR(L""), &pVar);
        if (FAILED(hr)) {
                goto EndProc;
        }

        // 値の設定, 取得
        pVar->put_Value(CComVariant(L"sample"));
        pVar->get_Value(&vntVal);

        // オブジェクトの解放
EndProc:
        if (pVar) pVar->Release();
        if (pCtrl) pCtrl->Release();
        if (pWs) pWs->Release();
        if (pWss) pWss->Release();
        if (pEng) pEng->Release();
        return hr;
}
```

#### 2.6.1.2. #import ディレクティブの場合

C++から CAO のオブジェクトを操作するもう一つの方法として#import ディレクティブを使用して, CAO.exe をインポートします.

```
#import "CAO.exe"
```

CAO オブジェクトのネームスペースは“CAOLib”になります. [10]

[10] ネームスペースの設定は#import ディレクティブのオプションで設定することができます. 詳細は MSDN を参照してください.

また#import ディレクティブを使用することで，オブジェクト名の後ろに“Ptr”をつけたスマートポインタを使用することができます．(例:ICaoEnginePtr)

以下は実際に CAO オブジェクトを操作する手順になります．

(1) DCOM の初期化

CAO オブジェクトを操作する前に CoInitialize()を実行します．

(2) CaoEngine オブジェクトの生成

CaoEngine オブジェクトのスマートポインタである ICaoEnginePtr を使用して生成します．以下に例を示します．スマートポインタを使用しない場合は，2.6.1.1 と同様の手順で生成します．

```
CAOLib::ICaoEnginePtr pEng(__uuidof(CAOLib::CaoEngine));
```

(3) メソッドの実行

(2)で生成した CaoEngine のメソッドを実行します．

以下に AddController を行う例を示します．

```
CAOLib::ICaoWorkspacesPtr pWss = pEng->GetWorkspaces();
CAOLib::ICaoWorkspacePtr pWs = pWss->Item(0L);
CAOLib::ICaoControllerPtr pCtrl = pWs->AddController(L"SampleCtrl",
                                                     CaoProv.DataStore",
                                                      L"",
                                                      L"");
```

(4) オブジェクトの解放

スマートポインタで保持しているオブジェクトは，自動的に解放処理が行われます．明示的に解放処理を行い場合，又はスマートポインタを使用していない場合は 2.6.1.1 と同様の手順で解放します．

(5) DCOM の終了処理

プログラムが終了する前に CoUnitialize()を実行します．

以下に変数の設定，取得を行うサンプルプログラムを示します．

**List 2-3 CPPSample2.cpp**

```
#import "D:¥work¥Robot¥Repos¥ORiN2¥CAO¥Engine¥Bin¥CAO.EXE"

HRESULT func1();

int main(int argc, char* argv[])
{
	CoInitialize(0);

	HRESULT hr = func1();

	CoUninitialize();
	return 0;
}
```

```
HRESULT func1()
{
	try {
		CAOLib::ICaoEnginePtr pEng(__uuidof(CAOLib::CaoEngine));
		CAOLib::ICaoWorkspacesPtr pWss = pEng->GetWorkspaces();
		CAOLib::ICaoWorkspacePtr pWs = pWss->Item(OL);
		CAOLib::ICaoControllerPtr pCtrl = pWs->AddController(L"SampleCtrl",
								L"CaoProv.DataStore",
								L"",
								L"");
		CAOLib::ICaoVariablePtr pVar = pCtrl->AddVariable(L"SampleVal", L"");

		pVar->PutValue(_variant_t(L"sample"));
		_variant_t vntTemp = pVar->GetValue();

	}
	catch (_com_error e) {
		return e.Error();
	}
	return S_OK;
}
```

### 2.6.2. C 言語によるクライアント作成 [10]

C 言語では#include ディレクティブを使用して CAO オブジェクトを操作します. インクルードするファイルは, CAO のヘッダファイル(CAO.h)と UUID 定義ファイル(CAO_i.c)になります. これらのファイルはこれらのファイルは“<ORIN2ルート>¥CAO¥Include”にあります.

```
#include "CAO.h"
#include "CAO_i.c"
```

CAO のオブジェクトを操作する手順は, 2.6.2 と同様の手順をとります.

以下に各手順の詳細を示します.

(1) DCOM の初期化

CAO オブジェクトを操作する前に CoInitialize()を実行します.

(2) CaoEngine オブジェクトの生成

CoCreateInstance()を実行します. 以下に引数の例を示します.

```
hr = CoCreateInstance(&CLSID_CaoSQLEngine,
                      NULL,
                      CLSCTX_ALL,
                      &IID_ICaoSQLEngine,
                      (void **)&pEng);
```

(3) メソッドの実行

(2)で生成した CaoEngine のメソッドを実行します. メソッドを実行は以下のように呼び出します. 以下に CaoWorkspace コレクションを取得する場合の例を示します.

```
pEng->lpVtbl->get_Workspaces(pEng, &pWSs);
```

ヘッダをインクルードする前に COBJMACROS というプリプロセッサシンボルを宣言すると, 各メソッド

に対応するマクロが得られます. 以下にマクロを使用して CaoWorkspace コレクションを取得する場合の例を示します.

```
ICaoEngine_get_Workspaces(pEng, &pWss);
```

(4) オブジェクトの解放

プログラム内で生成, 取得したオブジェクトは必ず解放処理をしなければなりません. 解放は各オブジェクトの Release()を実行します. 以下にマクロ使用した場合のオブジェクトの解放の例を示します.

```
ICaoEngine_Release(pEng)
```

(5) DCOM の終了処理

プログラムが終了する前に CoUnitialize()を実行します.

以下に変数の設定, 取得を行うサンプルプログラムを示します.

**List 2-4 ImportSample.cpp**

```
#import "D:¥work¥Robot¥Repos¥ORiN2¥CAO¥Engine¥Bin¥CAO.EXE"

HRESULT func1();

int main(int argc, char* argv[])
{
	CoInitialize(0);

	HRESULT hr = func1();

	CoUninitialize();
	return 0;
}

HRESULT func1()
{
	try {
		CAOLib::ICaoEnginePtr pEng(__uuidof(CAOLib::CaoEngine));
		CAOLib::ICaoWorkspacesPtr pWss = pEng->GetWorkspaces();
		CAOLib::ICaoWorkspacePtr pWs = pWss->Item(0L);
		CAOLib::ICaoControllerPtr pCtrl = pWs->AddController(L"SampleCtrl",
                                                             L"CaoProv.DataStore",
                                                             L"",
                                                             L"");
		CAOLib::ICaoVariablePtr pVar = pCtrl->AddVariable(L"SampleVal", L"");

		pVar->PutValue(_variant_t(L"sample"));
		_variant_t vntTemp = pVar->GetValue();

	}
	catch (_com_error e) {
		return e.Error();
	}
	return S_OK;
}
```

### 2.6.3. C#によるクライアント作成

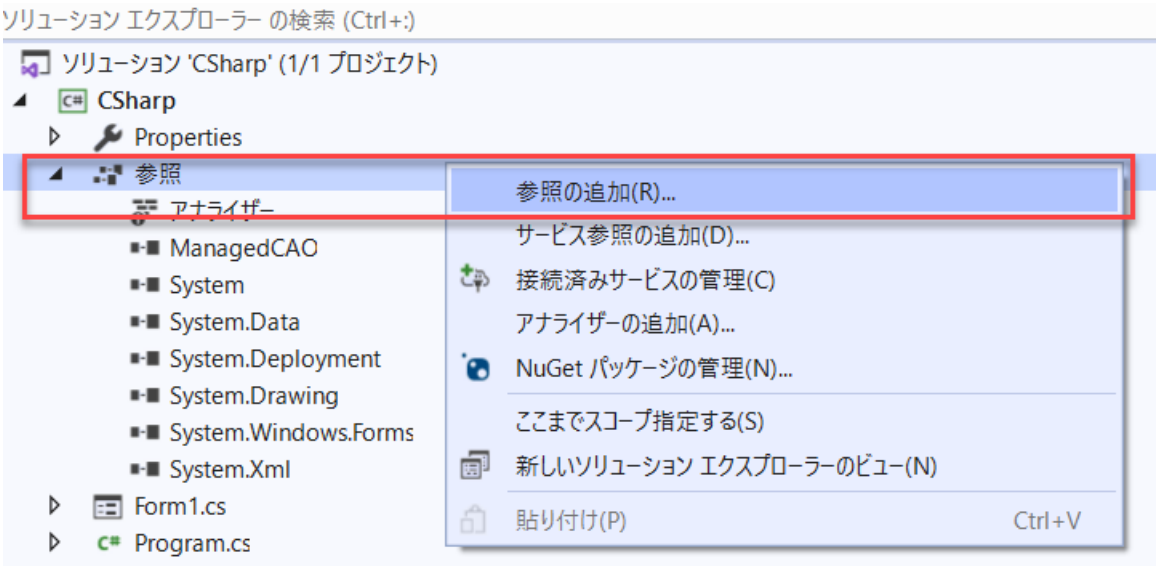
C#クライアントアプリケーション作成として，ManagedCAO 利用と RCW 利用の 2 つの方法を提供していますが，現在はより効率的な ManagedCAO 利用を推奨しています．RCW 利用の方法は従来型で互換のためにのみ存在します． RCW 利用はオブジェクトの明示的なリリース処理の記述が不可欠ですが，ManagedCAO はその部分の明示的な記述は不要であり，さらに RCW で課題であったクライアントアプリケーションの突然の終了に対してのオブジェクトのリリースを自動で実施する作りとなっている点で効率化されています．

#### 2.6.3.1. ManagedCAO 利用の場合

C#クライアントは，RCW をラップした ManagedCAO を介して COM コンポーネントにアクセスを行います．

##### 2.6.3.1.1. プロジェクト作成時の設定

C#クライアントアプリケーションのプロジェクトに対して ManagedCAO への参照を追加してください．




ManagedCAO は ORiN2 SDK のインストールフォルダ以下の下記モジュールに該当しますのでプロジェクトに対して参照指定してください．

"<ORiN2 インストールフォルダ>¥DotNet¥ManagedCAO¥Lib¥ManagedCAO.dll"

ManagedCAO モジュールの参照を追加することで，名前空間に<ORiN2.ManagedCAO>が認識されるようになりますので，「using ORiN2.ManagedCAO;」 を以下のように宣言を行ってください．

```
using System;
using System.Windows.Forms;
using ORiN2.ManagedCAO;
```

これにより, CAO エンジンのクラス変数を下記のように宣言することができるようなります.

```
public partical class Form1 : Form
{
        private CCaoEngine _engine = null;
        private CCaoController _controller = null;
        private CCaoVariable _variable = null;

        public Form1()
        :
}
```

#### 2.6.3.1.2. オブジェクト削除時の注意点

ManagedCAO では CCaoEngine の Dispose 関数が呼び出されると自動的に配下の Cao オブジェクトが破棄される仕組みになっていますので, 初期化の失敗時やアプリケーション終了時に CCaoEngine オブジェクトを開放, Dispose を実行するようにしてください.

また, メッセージを受信(イベントを取得)している場合は CCaoEngine の Dispose の呼び出し前にメッセージ受信の処理を解除し, Application.DoEvents 関数を呼び出してください. これにより, メッセージオブジェクトの解放が行われない場合に CAO が正常終了しないことを防止することができます.

以下に ReleaseCaoObject 関数として定義した記述例を示します.

```
private void ReleaseCaoObject()
{

      if (_controller != null)
      {
          _controller.OnMessage -= new OnMessageEventHandler(_controller_OnMessage);
          Application.DoEvents();
       }

       if (_engine != null)
       {
           _engine.Dispose();
       }

       _variable = null;
       _controller = null;
       _engine = null;
}
```

上記定義した ReleaseCaoObject 関数を使用することで例外時に確実に CAO が正常終了させることができます. その使用例を下記に示します.

```
private void Form1_Load(object sender, EventArgs e)
{
      try
      {
          _engine = new CCaoEngine();
          _controller = _engine.Workspaces[0].AddController("Panel", "CoProv.Dummy.Panel");
          _variable = _controller.AddVariable("Variable1", null);

          _controller.OnMessage += new OnMessageEventHandler(_controller_OnMessage);
       }

       catch (Exception ex)
       {

           ReleaseCaoObject();
           this.Close();
       }
  }
```

#### 2.6.3.1.3. スレッドセーフな呼び出し方法

WindowsForms アプリケーションでは UI スレッド以外からのコントロールへのアクセスは禁止されていいます. そのため, InvokeRequiredプロパティ(UIスレッド以外で実行されている場合はtrue)を参照し, UIスレッド以外の場合は Invoke 関数(UI スレッドに処理を渡し, 終了を待機)を通して UI スレッドで処理を行ってください. それ以外の場合は自スレッドで処理を行ってください. 下記にその使用例を示します.

```
void _controller_OnMessage(object sender, OnMessageEventArgs e)
{
    Action act = () =>
        {
          txtMessage.Text = string.Formart("Number:{0}, Source:{1}, Value:{2}",
                e.Message.Number, e.Message.Source, e.Message.Value);
      };

    if (InvokeRequired)
    {
      Invoke(act);
    }
    else
    {
      act();
    }
}
```

以下に, OnMessage イベントの値をテキストボックスに表示するサンプルを示します.

**List 2-5-1 ManagedCaoSample.cs**

```
using System;
using System.Windows.Forms;
using ORiN2.ManagedCAO;

namespace Message
{
    public partial class frmMessage : Form
    {
        private CCaoEngine caoEng;
        private CCaoWorkspaces caoWss;
        private CCaoWorkspace caoWs;
```

```
private CCaoControllers caoCtrls;
private CCaoController caoCtrl;

public frmMessage()
{
    InitializeComponent();
}

private void frmMessage_Load(object sender, EventArgs e)
{
    try
    {
        // CAO エンジン生成
        caoEng = new CCaoEngine();
        caoWss = caoEng.Workspaces;
        caoWs = caoWss[0];

        // コントローラコレクションの取得
        caoCtrls = caoWs.Controllers;

        // コントローラに接続
        caoCtrl = caoCtrls.Add("RC1", "CaoProv.Dummy", "", "");

        // OnMessage イベントハンドラの登録
        caoCtrl.OnMessage += new OnMessageEventHandler(OnMessage);

    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

private void frmMessage_FormClosed(object sender, FormClosedEventArgs e)
{
    try
    {
        // コントローラオブジェクトの解放
        if (caoCtrl != null)
        {
            caoCtrls.Remove(caoCtrl.Name);
            caoCtrl = null;
        }
        caoCtrls = null;
        caoWs = null;
        caoWss = null;
        caoEng = null;
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

private void cmdExecute_Click(object sender, EventArgs e)
{
    try
    {
        caoCtrl.Execute("", "");
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

```
            // OnMessage イベント
            private void OnMessage(object sender, OnMessageEventArgs e)
            {
                Action act = () => { textBox1.Text = e.Message.Value.ToString(); };

                if (InvokeRequired)
                {
                    Invoke(act);
                }
                else
                {
                    act();
                }
            }
        }
    }
```

#### 2.6.3.2. RCW 利用の場合

C#クライアントは，VB.net クライアントと同様に RCW を介して COM コンポーネントにアクセスを行います．設定内容は 2.5 と同様な設定を行います．

##### 2.6.3.2.1. RCW の GAC への登録

登録方法は VB.net の場合と同じです．詳細については，2.5.1 を参照してください．

##### 2.6.3.2.2. プロジェクト作成時の設定

参照の追加については，VB.net の場合と同じです．2.5.2 を参照してください．

RCW モジュールの参照を追加することで CAO エンジンのクラス変数を下記のように宣言することができるようなります．

```
private ORiN2.interop.CAO.CaoEngine caoEng;;
```

名前空間に<ORiN2.interop.CAO>を指定することで，上記の宣言を簡素化することが出来ます．名前空間の指定する場合は，以下の宣言を行います

```
using ORiN2.interop.CAO;
```

##### 2.6.3.2.3. オブジェクト削除時の注意点

C#は，VB.net と同様にガーベジコレクションがあるので，変数に“null”を代入してもオブジェクトの Release 処理がすく実行されません．オブジェクトの削除を即時実行する為には，VB.net と同様に System.Runtime.InteropServices.Marshal.ReleaseComObject 関数を使って明示的に解放処理を行う必要があります．詳細な注意点については，2.5.3 を参照してください．

##### 2.6.3.2.4. イベント取得方法

C#で，イベントを取得するときは以下の手順で行います．

(1) イベントハンドラの作成

```
private void OnMessage(CaoMessage pICaoMsg)
{
        MessageBox.Show(pICaoMsg.Value.ToString());
}
```

(2) イベントハンドラの登録

```
caoCtrl = caoCtrls.Add("RC1", "CaoProv.Dummy", "", "");
caoCtrl.OnMessage += new _ICaoControllerEvents_OnMessageEventHandler(OnMessage);
```

#### 2.6.3.2.5. スレッドセーフな呼び出し方法

OnMessage イベントは, VB. net の場合と同様にメインスレッドとは別のスレッドにより処理されるので, スレッドセーフなプログラミングに注意してください.

C#でスレッドセーフなプログラミングは以下のように行います.

(1) イベントで処理する内容を関数化する.

(2) 作成した関数を呼び出すためのデリゲートを作成する.

```
private delegate void SetTextCallback(string msg);
```

(3) OnMessage イベント内から System.Invoke 関数でイベント処理関数を呼び出す.

```
private void OnMessage(CaoMessage pICaoMsg)
{
        // イベント処理関数を設定したデリゲートを作成する
        SetTextCallback SetMsg = new SetTextCallback(SetText);

        // System.Invoke 関数でデリゲートに登録した関数を実行する.
        Invoke(SetMsg, pICaoMsg.Value.ToString());
}
```

以下に, OnMessage イベントの値をテキストボックスに表示するサンプルを示します.

**List 2-6-2 RcwSample.cs**

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using ORiN2.interop.CAO;

namespace Message
{
    public partial class frmMessage : Form
    {

        private delegate void SetTextCallback(string msg);  // OnMessage 処理用デリゲート

        private CaoEngine caoEng;
        private CaoWorkspaces caoWss;
        private CaoWorkspace caoWs;
        private CaoControllers caoCtrls;
        private CaoController caoCtrl;

        public frmMessage()
        {
            InitializeComponent();
        }
```

```
        private void frmMessage_Load(object sender, EventArgs e)
        {
            try
            {
                // CAO エンジン生成
                caoEng = new CaoEngine();
                caoWss = caoEng.Workspaces;
                caoWs = caoWss.Item(0);

                // コントローラコレクションの取得
                caoCtrls = caoWs.Controllers;

                // コントローラに接続
                caoCtrl = caoCtrls.Add("RC1", "CaoProv.Dummy", "", "");

                // OnMessage イベントハンドラの登録
                caoCtrl.OnMessage += new
_ICaoControllerEvents_OnMessageEventHandler(OnMessage);

            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

        private void frmMessage_FormClosed(object sender, FormClosedEventArgs e)
        {
            try
            {
                // コントローラオブジェクトの解放
                if (caoCtrl != null)
                {
                    caoCtrls.Remove(caoCtrl.Name);
                    System.Runtime.InteropServices.Marshal.ReleaseComObject(caoCtrl);
                    caoCtrl = null;

                }
                System.Runtime.InteropServices.Marshal.ReleaseComObject(caoCtrls);
                caoCtrls = null;
                System.Runtime.InteropServices.Marshal.ReleaseComObject(caoWs);
                caoWs = null;
                System.Runtime.InteropServices.Marshal.ReleaseComObject(caoWss);
                caoWss = null;
                System.Runtime.InteropServices.Marshal.ReleaseComObject(caoEng);
                caoEng = null;
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

        private void cmdExecute_Click(object sender, EventArgs e)
        {
            try
            {
                caoCtrl.Execute("", "");
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
```

```
            // OnMessage イベント
            private void OnMessage(CaoMessage pICaoMsg)
            {
                try
                {
                    // 別スレッドのメソッド呼び出し
                    SetTextCallback SetMsg = new SetTextCallback(SetText);
                    Invoke(SetMsg, pICaoMsg.Value.ToString());
                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message);
                }
            }

            // OnMessage 時の処理関数
            private void SetText(string msg)
            {
                textBox1.Text = msg;
            }

        }
    }
```

### 2.6.4. Delphi によるクライアント作成

#### 2.6.4.1. 作成準備

Delphi で CAO のオブジェクトを操作するには，タイプライブラリからユニットファイルを作成，使用して行います．

ユニットファイルの作成と使用方法は以下の手順になります．

(1) メニューから[プロジェクト]→[タイプライブラリの取り込み]を選択する．

(2) 表示されたダイアログで“CAO1.0 タイプライブラリ”を選択し，ユニットの作成ボタンをクリックします．（図 2-11）作成されたユニットファイルは，ユニットディレクトリ名で指定した場所にファイル名“CAOLib_TLB.pas”で作成されます．

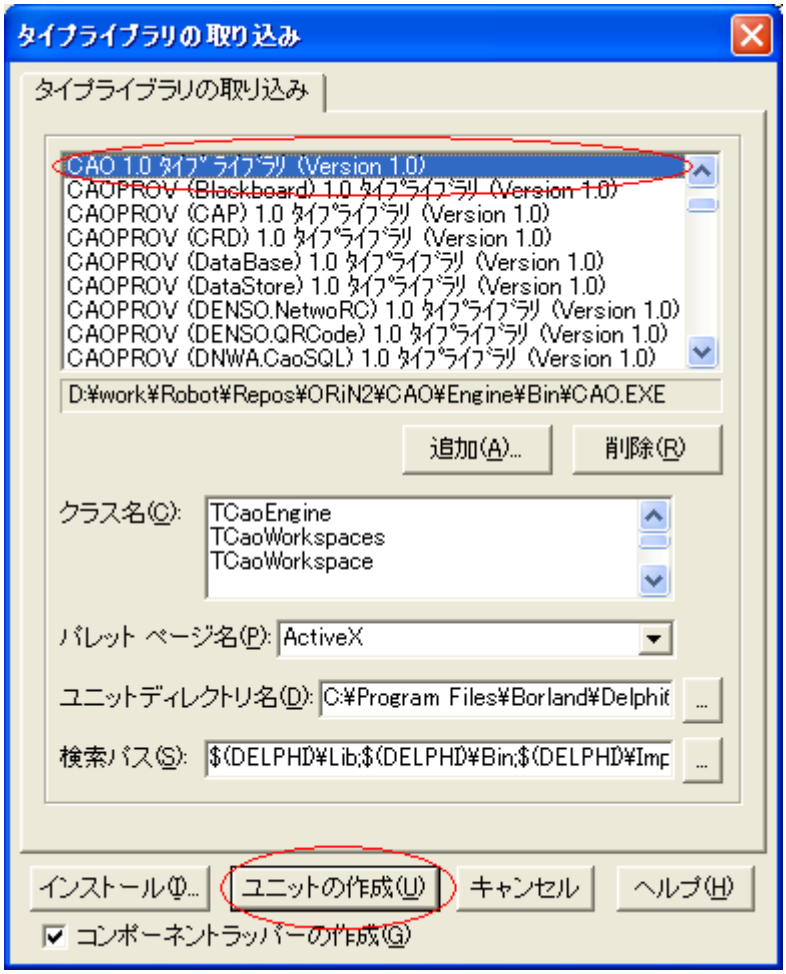



**図 2-11 タイプライブラリの取り込み画面**

(3) CAOオブジェクトを使用するソースファイルをアクティブにして，メニューから[ファイル]→[ユニットを使う]を選択します．

(4) 使用するユニットの選択画面で“CAOLib_TLB”を選択して OK ボタンをクリックする．

(5) ソースファイルの uses 節に“ComObj”を追加する．

以上の手順で CAO オブジェクトを使用することができるようになります．

#### 2.6.4.2. 操作手順

CAO オブジェクトの操作は以下の手順で行います．

(1) CaoEngine オブジェクトの生成

ProgIDToClassID()で UUID を取得した後，CreateComObject()でオブジェクトを生成します．

```
CaoObj := CreateComObject(ProgIDToClassID('CAO.CaoEngine'));
```

(2) CaoEngine インタフェースの取得

(1)で生成されたオブジェクトはIUknown型のインタフェースなのでCaoEngineインタフェースにキャストをします.

```
CaoEng := CaoObj as ICaoEngine;
```

(3) オブジェクトの操作

CaoEngineインタフェースを使ってCAOを操作します.

```
CaoWss := CaoEng.Workspaces;
```

注意する点として, CAOオブジェクトの変数が解放されるときに自動的にRelease()が実行されます. このため他の言語のように明示的に解放処理を行った場合, CAOが正しく終了しない可能性があります.

以下に変数の値を設定, 取得するサンプルを示します.

**List 2-7 Unit1.pas**

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ComObj;

type
  TForm1 = class(TForm)
    Button1: TButton;
    Edit1: TEdit;
    Edit2: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    procedure Button1Click(Sender: TObject);
  private
    { Private 宣言 }
  public
    { Public 宣言 }
  end;

var
  Form1: TForm1;

implementation

uses CAOLib_TLB;

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
var
  CaoObj: IUnknown;
  CaoEng: ICaoEngine;
  CaoCtrl: ICaoController;
  CaoVar: ICaoVariable;

begin
  Try
    {CaoEngineの生成}
    CaoObj := CreateComObject(ProgIDToClassID('CAO.CaoEngine'));
    CaoEng := CaoObj as ICaoEngine;
```

```
    {CaoController の生成}
    CaoCtrl := CaoEng.Workspaces.Item(0).AddController('', 'CaoProv.DataStore', '', '');

    {CaoVariable の生成}
    CaoVar := CaoCtrl.AddVariable('Test', '');

    {値の設定}
    CaoVar.Value := Edit1.text;

    {値の取得}
    Edit2.Text := CaoVar.Value;
  Except
    {エラー処理}
    ShowMessage('Error');
  End;
end;

end.
```

### 2.6.5. Java によるクライアント作成

Java からは直接 COM による呼び出しを行うことはできません. このため, Java から CAO を利用するためには, 以下の方法があります. 各接続方法については以下で説明します.

- JNI のネイティブメソッドライブラリ[11]
- Java-COM ブリッジライブラリ

#### 2.6.5.1. JNI のネイティブメソッドライブラリからの CAO 呼び出し

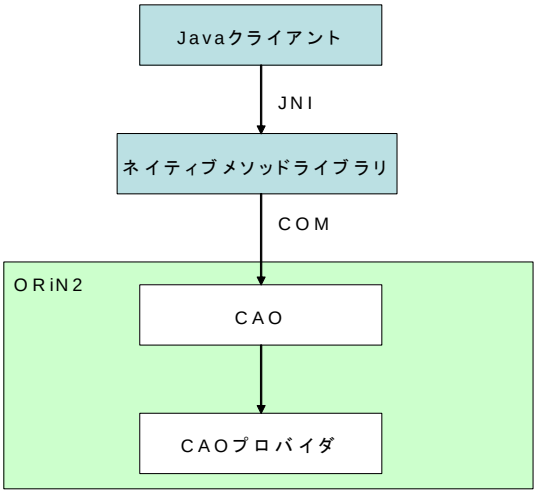



**図 2-12 Java による CAO 呼び出し**

以下にネイティブメソッドライブラリの作成手順及び Java のサンプルクライアントの紹介を行います. 各手順項目の末尾の括弧は使用言語を示しています.

(1) ネイティブメソッドの宣言(Java)

クライアントプログラムが必要とする機能を持つネイティブメソッドを宣言します. このとき宣言するネイティブメソッドは, CAO のインタフェースをラップしたメソッドである必要はありません.

ネイティブメソッドを宣言するときは, メソッド宣言に“native”キーワードをつけます.

例) `public native int AddVariable(String VarName);`

(2) ネイティブメソッドを使用するクライアントプログラムの作成(Java)

宣言したネイティブメソッドを使用したクラスを作成し, コンパイルします.

コンパイルは, コマンドプロンプトで以下のコマンドを実行します.

例) `javac CaoJNI.java`

[11] CaoSQL では Java のためにインタフェースをラップしたライブラリを提供しています.

(3) ネイティブメソッドの C++ヘッダ生成(Java)

コンパイルしたファイルからネイティブコードでインクルードするヘッダファイルを生成します.

生成は, コマンドプロンプトで以下のコマンドを実行します.

例) javah -jni CaoJNI

(4) JNI ヘッダファイルのインクルード(C++)

生成されたヘッダファイルと JNI のヘッダファイルをインクルードします.

```
例)  #include <jni.h>
     #include "CaoJNI.h"
```

(5) ネイティブメソッドライブラリの実装(C++)

2.6.1 の手順に従い, CAO オブジェクトを使用できるように設定します.

生成されたヘッダファイルで宣言されているネイティブメソッドを実装する.

(6) ネイティブメソッドライブラリのコンパイル(C++)

実装したネイティブコマンドライブラリをコンパイルします.

コンパイルは, コマンドプロンプトで以下のコマンドを実行します. ここでコンパイルのオプションについては, 環境に合わせて変更して下さい.

```
例) cl /IC:¥jdk1.3.1¥include /IC:¥jdk1.3.1¥include¥win32 /IC:¥ORiN2¥CAO¥Include
                                                    /LD CaoJNI.cpp /FeCaoJNI.dll
```

(7) サンプルクライアントの実行(Java)

サンプルクライアントを実行する. ここでネイティブメソッドライブラリのファイルが, 環境変数"CLASSPATH"で記述されたフォルダになければならない.

クライアントの実行は以下のコマンドで実行します.

```
例)  java CaoJNI
```

以下に上記手順で作成したサンプルのソースを示します.

**List 2-8 CaoJNI.java**

```
import java.io.*;

public class CaoJNI {

        /* --- Cao Native Interfases --- */
        public native int CreateCaoEngine();
        public native int AddController(String CtrlName, String ProvName, String Machine, String
Param);
        public native int AddVariable(String VarName);
        public native String GetValue();
        public native void PutValue(String strVal);
        public native void FreeMemory();

        // コンストラクタ
        public void CaoJNI() {
        }

        // CaoJNI クラスの初期化処理
```

```
        public void init() {
                System.loadLibrary("CaoJNI");
        }

        public static void main (String args[]) {

                CaoJNI cao= new CaoJNI();
                cao.init();

                int iRet;

                // CaoEngine の作成
                iRet = cao.CreateCaoEngine();
                if (iRet != 0) {
                        System.out.println("Can't make CaoEngine!");
                        return;
                }

                iRet = cao.AddController("Sample", "CaoProv.DataStore", "", "");
                if (iRet != 0) {
                        System.out.println("Error on OpenController: ErrorCode(" +
Integer.toHexString(iRet) + ")");
                        cao.FreeMemory();
                        return;
                }

                iRet = cao.AddVariable("S1");
                if (iRet != 0) {
                        System.out.println("Error on GetVariable: ErrorCode(" +
Integer.toHexString(iRet) + ")");
                        cao.FreeMemory();
                        return;
                }

                cao.PutValue("Sample Data");
                System.out.println("PutValue Succeeded");

                String strRet = cao.GetValue();
                System.out.println(strRet);

                cao.FreeMemory();
        }
}
```

**List 2-9 CaoJNI.cpp**

```
#include <jni.h>
#include "CaoJNI.h"
#include <atlbase.h>

#include "CAO.h"
#include "CAO_i.c"

BSTR jstring2BSTR(JNIEnv *env, jstring jstr);

ICaoEngine* g_pICaoEng;
ICaoController* g_pICaoCtrl;
ICaoVariable* g_pICaoVar;

// CreateCaoEngine
JNIEXPORT jint JNICALL Java_CaoJNI_CreateCaoEngine
  (JNIEnv *env, jobject me)
{
        HRESULT hr;
```

```
	hr = CoInitialize( NULL );

	/* Create CaoEngine Object */
	hr = CoCreateInstance(CLSID_CaoEngine, NULL, CLSCTX_LOCAL_SERVER, IID_ICaoEngine,
(void**)&g_pICaoEng);
	if (SUCCEEDED(hr)) {
		printf("Make CaoEngine Succeeded!¥n");
	}

	return hr;
}

// AddController
JNIEXPORT jint JNICALL Java_CaoJNI_AddController
  (JNIEnv *env, jobject me, jstring strCtrl, jstring strProv, jstring strMachine, jstring
strParam)
{
	if (g_pICaoCtrl) {
		return E_FAIL;		// 既に Controller は Open されています
	}

	HRESULT hr;

	CComBSTR bstrProv, bstrMachine, bstrCtrl, bstrParam;
	bstrProv = jstring2BSTR(env, strProv);
	bstrMachine = jstring2BSTR(env, strMachine);
	bstrCtrl = jstring2BSTR(env, strCtrl);
	bstrParam = jstring2BSTR(env, strParam);

	/* Get CaoWorkspace Object */
	CComPtr<ICaoWorkspaces> pICaoWSs;
	CComPtr<ICaoWorkspace> pICaoWS;
	hr = g_pICaoEng->get_Workspaces(&pICaoWSs);
	if (FAILED(hr)) {
		return hr;
	}
	hr = pICaoWSs->Item(CComVariant(0L), &pICaoWS);
	if (FAILED(hr)) {
		return hr;
	}

	/* Create CaoController Object */
	hr = pICaoWS->AddController(bstrCtrl, bstrProv, bstrMachine, bstrParam, &g_pICaoCtrl);
	if (SUCCEEDED(hr)) {
		printf("OpenController Succeeded!¥n");
	}

	return hr;
}

// AddVariable
JNIEXPORT jint JNICALL Java_CaoJNI_AddVariable
  (JNIEnv *env , jobject me, jstring strVar)
{
	if (g_pICaoVar) {
		g_pICaoVar->Release();
	}

	HRESULT hr;

	CComBSTR bstrVar, bstrOption;
	bstrVar = jstring2BSTR(env, strVar);
	hr = g_pICaoCtrl->AddVariable(bstrVar, bstrOption, &g_pICaoVar);
	SysFreeString(bstrVar);

	if (SUCCEEDED(hr)) {
```

```
                printf("GetVariable Succeeded!¥n");
        }

        return hr;
}

// GetValue
JNIEXPORT jstring JNICALL Java_CaoJNI_GetValue
  (JNIEnv *env, jobject me)
{
        if (!g_pICaoVar) {
                return NULL;  // Variableオブジェクトがありません
        }

        // 値の取得
        CComVariant vntValue;
        HRESULT hr = g_pICaoVar->get_Value(&vntValue);
        if (FAILED(hr)) {
                return NULL;
        }

        // 取得値を文字列変換
        hr = vntValue.ChangeType(VT_BSTR);
        if (FAILED(hr)) {
                return NULL;
        }

        return env->NewString(vntValue.bstrVal, SysStringLen(vntValue.bstrVal));
}

// PutValue
JNIEXPORT void JNICALL Java_CaoJNI_PutValue
  (JNIEnv *env, jobject me, jstring strVal)
{
        if (!g_pICaoVar) {
                return;                         // Variableオブジェクトがありません
        }

        HRESULT hr;
        CComBSTR bstrVal;
        bstrVal = jstring2BSTR(env, strVal);

        hr = g_pICaoVar->put_Value(CComVariant(bstrVal));
        return;
}

// FreeMemory
JNIEXPORT void JNICALL Java_CaoJNI_FreeMemory
  (JNIEnv *env , jobject me)
{
        if (g_pICaoVar) {
                g_pICaoVar->Release();
                printf("CaoVariable Deleted¥n");
        }
        if (g_pICaoCtrl) {
                g_pICaoCtrl->Release();
                printf("CaoController Closed¥n");
        }
        if (g_pICaoEng) {
                g_pICaoEng->Release();
                printf("CaoEngine Terminated¥n");
        }

        CoUninitialize();

        return;
```

```
}

// 文字列変換 (Java→BSTR)
BSTR jstring2BSTR(JNIEnv *env, jstring jstr)
{

    if (jstr==NULL) {
                return NULL;
      }

    const WCHAR* wc;
       wc = env->GetStringChars(jstr, NULL);
    int iLen;
       iLen = env->GetStringLength(jstr);
       BSTR bstrString;
       bstrString = SysAllocStringLen(wc, iLen);

       env->ReleaseStringChars(jstr, wc);

    return bstrString;

}
```

#### 2.6.5.2. Java-COM ブリッジを使用しての CAO 呼び出し

Java-COM ブリッジを使用することにより CAO を呼び出すことができます.

Java-COM ブリッジとは、Java のプログラムから COM の呼び出しを行うためのライブラリであり、OSS として複数種類が提供されています。(例:j-interop、COM4J 等)

CAO は COM を利用してい作成されているため、Java-COM ブリッジを使用することで Java から直接 CAO を利用することができるようになります。

具体的な使用方法は、ライブラリによって異なるため、各ライブラリの情報を参照してください。

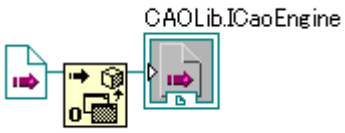
### 2.6.6. LabVIEW によるクライアント作成

LabVIEW から CAO のオブジェクトを操作するには, ActiveX クライアントとして機能する VI ファイルを作成します.

ActiveX クライアント VI の作成と使用方法は以下の手順となります.

※下記手順内の図は, LabVIEW ver8.6 の画面になります.

(1) オートメーションオープン関数を使用し, サーバへのリファレンスを開く.

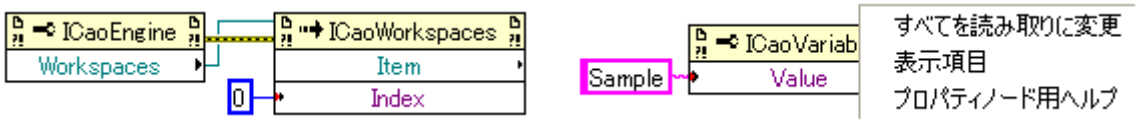



オートメーション Refnum の入力に対し, [ActiveX クラスを選択]-[参照]から
“CAO1.0 Type Library Version 1.0” を選択し, [CaoEngine]を指定する.


**図 2-13 タイプライブラリからオブジェクトを選択画面**

(2) プロパティ設定やメソッド操作を行う.

※“Read”や“Write”ができるプロパティは, プロパティノードの属性を変更して使用する.

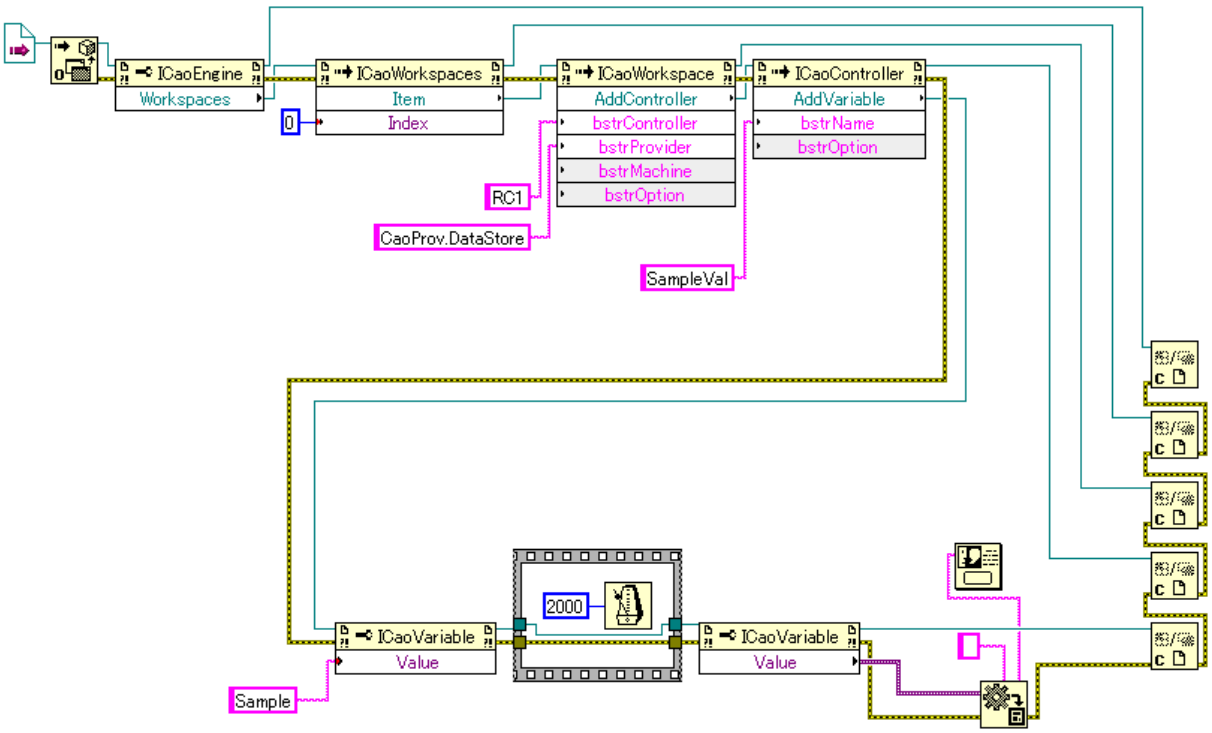



(3) リファレンスを閉じる.

オートメーション Refnum を閉じる関数より, リファレンスを閉じる.

以下に変数の値を設定, 取得するサンプルを示します.

| List 2-10 | ImportSample.vi |
|---|---|




## 2.7. CAO の特殊な機能[12]

### 2.7.1. CRD 切り替え機能

CAO エンジンは，各プロバイダのプロパティの呼び出し時に，プロバイダ毎に設定した CRD ファイルへアクセスすることができます．以下に CRD 切り替え機能を使用する際のプロバイダの設定手順を示します．

(1) CAO エンジンを実行するマシンに CRD プロバイダをインストールします．

(2) CRD 切り替え機能を使用するプロバイダのレジストリ情報“CRDFile”に，参照する CRD ファイルのパスを設定します．このパスは CaoConfig を利用して設定することができます．詳細は 6 を参照してください．

(3) プロバイダの CRD ファイルを参照するプロパティで，返り値に“E_CRDIMPL”を返します[13]．

但し，CRD ファイルから値を取得するためには，生成したコントローラ等のオブジェクト名が，CRD ファイルのものと一致している必要があります．

### 2.7.2. オブジェクト自動登録機能

CAO エンジン起動時にデフォルトワークスペースにオブジェクトを自動登録します．

[12] デンソー製 CAO エンジン特有の機能
[13] メソッドで“E_CRDIMPL”を返したときは，値をそのままクライアントに返します．

自動登録するオブジェクト構成はレジストリに登録された CRD ファイルと同じ構成を生成します．登録された CRD ファイルへのパスが間違っている場合は，登録できるオブジェクトのみを登録します．

レジストリへの登録は CaoConfig で登録することができます．

CRD ファイルを編集には XML エディタを使用すると便利です．以下にフリーソフトの XML エディタを紹介します．

Morphon XML-Editor 3.1.4： http://www.morphon.com/

以下に CRD ファイルのサンプルを示します．

**List 2-11 CRDSample.xml**

```
<?xml version="1.0" encoding="Shift_JIS"?>
<CRD xsi:schemaLocation="http://www.orin.jp/CRD/CRDSchema CRDSchema.xsd"
 xmlns="http://www.orin.jp/CRD/CRDSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

        <Help>CRD Sample Data</Help>
        <Version>1.0.0</Version>
        <Controller name="RC1" provider="CaoProv.XXX" machine="" option="Conn=eth:192.168.1.1">
                <Command name="Walk"/>
                <Variable name="Var1"/>
        </Controller>
</CRD>
```

### 2.7.3. メソッド動的追加機能

コマンドクラスをコントローラクラスのメソッドとして動的追加します．

コントローラクラスは，コマンド名をメソッド名としてコマンドの Execute メソッドを実行することができます．

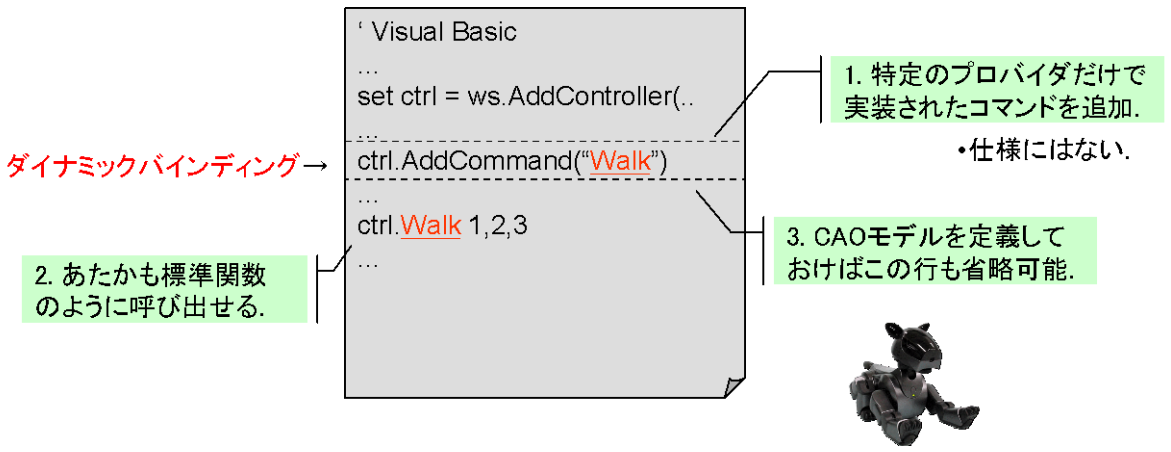



**図 2-14 ダイナミックバインディング**

以下にメソッド動的追加機能のサンプルを示します．

```
' コマンドの生成
```

```
Dim Cmd As CaoCommand
Set Cmd = Ctrl.AddCommand("Shoot")

' コマンドクラスで実行する場合
Cmd.Parameters = Array(10,10,20)
Cmd.Execute 0

' メソッド動的追加機能を使う場合
Ctrl.Shoot 10, 10, 20
```

# 3. CRD プログラミングガイド

## 3.1. 概要

CRD（Controller Resource Definition）は FA デバイスが持つ各種静的リソースを XML（Extensible Markup Language）で記述するための規格です. そのデータスキーマを CRD スキーマと呼び, この CRD スキーマに従い記述されている XML ファイルを CRD インスタンスと呼びます. CRD は FA デバイスの情報を共通のフォーマットで管理するための標準データスキーマです.

CRD は下記の 3 つの用途を想定して設計されています.

(1) デバイスが持つ静的データの定義

(2) デバイスやシステム全体のコンフィギュレーションの定義

(3) デバイスのケイパビリティ定義

本章の構成は, 前半で基礎知識として XML について簡単に解説し, 後半で具体的な CRD ファイルの作成手順を解説します.

## 3.2. 基礎知識

### 3.2.1. XML とは

XML とは, 自己拡張可能なマークアップ言語(eXtensible Markup Language)のことで, 文書構造を決定するための規則を文書の作成者が決定できます. (これが「自己拡張可能」と言われる所以です)この, 文書の構造をその作り手が自由に決定できるという機能を利用すれば, XML という標準仕様を用いながら, 特定の構造にとらわれない自由度の高いデータを広く交換できるようになります.

XML には, 以下のような特徴があります.

- 意味と内容を持ったデータ形式
- 階層的なデータ構造定義が可能
- タグを論理的に記述でき, 構造化された分かりやすい文書を作成可能
- 拡張性があり, 業界標準として普及しつつあるデータ形式

以下に簡単な XML 文書の例を示します. XML では, タグを利用してデータを表現します. このとき, 開始タグから終了タグまでのことを要素と呼びます. 下記の例では, <ORiN>や<Author>といったタグを利用して, ORiN やその作者の情報を保持しています. 要素は, 別の要素を子要素として持つことができ, これにより, データを構造的にあらわしています. さらに,「Release=“2003”」といったように, 要素に付加的な情報(属性)を持たせることもできます.

```
<ORiN>
  <ORiN1 Release= “2003”>
    <Author>ORiN 協議会</Author>
    <EngineName>RAO</EngineName>
  </ORiN 1>
  <ORiN2 Release= “2005”>
    <Author>ORiN 協議会</Author>
    <EngineName>RAO</EngineName>
  </ORiN 2>
```

```
</ROiN>
```

### 3.2.2. DOM

DOM(Document Object Model)とは，XML 文書をオブジェクトとして扱うための標準 API です．

XML 文書自体は，テキストファイルですが，これをブラウザや CAO などのアプリケーションが処理するときには，XML で記述されている個々の要素を，一つのオブジェクトとして読み出します．このとき XML をオブジェクトとして扱う方法が，アプリケーションによって異なってしまうと，開発者にとって大きな負担になります．そこで W3C により統一的な規格として定められているのが DOM です．XML の API としては，DOM の他に SAX(Simple API for XML)があり，どちらも広く使用されています．

また，DOM には，いくつかのレベルがあり，レベルが大きい方がより新しく高機能となっています．現在 W3C より勧告されている中では Level-2 が最新であり，CRD プロバイダなどでもこれを使用しています．

### 3.2.3. XML パーサ

XML 文書は通常テキストファイルとして存在しています．しかし，XML にはさまざまな表記方法が許されており，そのデータをアプリケーションで読込むためにはいくつかの手順を踏む必要があります．その手順のうち汎用的に使用されるものを取り出したものが XML パーサです．

アプリケーションから XML パーサを呼び出す場合には，標準化された API を使用します．標準 API には，先ほど説明した DOM や SAX などがあります．DOM は公開された標準であるため，現在これに準拠した XML パーサが複数公開されています．なかでも Windows 環境において Visual C++や Visual Basic で容易に使用可能なものとして MSXML があげられ，CRD プロバイダなどではこの MSXML が採用されています．また MSXML は現在複数のバージョンが存在しますが，CRD プロバイダでは MSXML4.0 を使用しています．CRD プロバイダを使用するときは，インストールされている MSXML のバージョンの確認が必要となります．

### 3.2.4. XML スキーマ

XML におけるスキーマとは，XML 文書の取り得る構造を記述したものです．つまり要素や属性の配列に正しい並び方を記述したものであり，このスキーマを記述することによって，XML パーサなどで XML 文書の正しさをある程度まで自動的にチェックすることが可能となります．

XML 用の代表的なスキーマ言語には，DTD，XML Schema，RELAX，XDR などがあげられますが，CRD では XML Schema を採用しています．

## 3.3. CRD ファイルと CRD プロバイダ

CRD ファイルを利用するクライアントアプリケーションでは，主として次の二つの方法でそのファイルにアクセスすることができます．

(1) CAO エンジンを使って間接的にアクセスする．
(2) XML パーザを使って直接的にアクセスする．

(2)の方法の説明は適当な参考書に譲りますが，(1)の方法について少し解説します．(1)の場合，ORiN2

SDK では，クライアントアプリケーションがコントローラにアクセスするのと同じ方法で CRD ファイルにアクセスすることができるように，CRD プロバイダが用意されています．このプロバイダを利用すると，実際のコントローラ内のデータも，CRD ファイルの中で表現されたデータも同じように扱うことができます．

CRD スキーマではそれぞれの要素が CAO のインタフェースにほぼ一対一に対応する形で定義されており，例えば“RC1”というコントローラ要素内に“Var1”という変数要素が定義された test.xml という CRD ファイルに対して，CRD プロバイダは，以下のような方法で CRD ファイル内の変数要素“I4”にアクセスすることができます．

```
Dim caoEng As New CaoEngine
Dim caoWS As CaoWorkspace
Dim caoCtrl As CaoController
Dim caoVar As CaoVariable

Set caoWS = CaoEng.CaoWorkspaces(0).
Set caoCtrl = caoWS.AddController("RC1", "CaoProv.CRD", "", "C:¥test.xml")
Set caoVar = caoCtrl.AddtVariable("Var1")
```

CRD ファイルを作成したら，確認や設定の変更などに CRD プロバイダを使用してみてください．

ただし，CRD プロバイダでは，6.1.3.付録 A.1CAO エンジン関数一覧の中で，R/W の項目に“W”が記述されているプロパティがだけが，データの変更をおこなうことができます．

CRD プロバイダの詳細な仕様は別途用意されている「CRD プロバイダユーザーズガイド」を参照してください．

## 3.4. CRD ファイルの作成

### 3.4.1. ヘッダとルート要素の作成

CRD ファイルは XML ファイルであるので，まずファイルの先頭には XML を使用する宣言として以下のような記述をおこないます．

```
<?xml version="1.0" encoding="Shift_JIS"?>
```

ここで encoding 属性は CRD ファイルで使用する文字コード[14]に合わせて記述する必要があります．

次に CRD ドキュメントのルート要素として以下の指定をおこないます．

```
<CRD xmlns=http://www.orin.jp/CRD/CRDSchema
    xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
    xsi:schemaLocation="http://www.orin.jp/CRD/CRDSchema CRDSchema.xsd">
```

この要素において，最後の“CRDSchema.xsd”の部分にはスキーマのパスを指定します．指定したパスにスキーマがない場合は CRD ファイルの構造のチェックがおこなわれません．

また，CRD のルート要素には，以下の子要素を持つことができます．

### 3.4.2. 静的データ定義

静的データ定義では，コントローラ以下のオブジェクト要素とプロパティ要素を XML ファイルに記述していきます．

(1) オブジェクト要素を追加します．

オブジェクト要素は CAO オブジェクトのインスタンスを示します．CRD ルート要素の下に Controller 要素を配置し，Controller 要素以下は CAO のオブジェクトツリーと同じ構造で配置します．

以下にオブジェクト要素の記述例を示します．

```
<Controller name="RC1">
<File name= “pro1.pac”>
  …
</File>
<Variable name= “I1”>
  …
</Varriable>
</Controller>
```

ここでオブジェクト要素には必ず name 属性が必要であり，name 属性はオブジェクトの名前をあらわしています．上記の例では CaoController「RC1」は，「pro1.pac」という CaoFile と，「I1」という CaoVariable を持っています．

(2) オブジェクト要素にプロパティ要素を追加します．

各オブジェクトの静的データとして記述するプロパティ要素をオブジェクト要素の下に追加します．ここで追加できるプロパティ要素はオブジェクト要素によって異なります．詳しくは「ORiN 2.1 仕様書 Part 3: CRD」を参照してください．

[14] XML では UTF-8 が推奨されています．

### 3.4.3. システムコンフィギュレーション定義

システムコンフィギュレーション定義ではコントローラ以下のオブジェクト要素を XML ファイルに追加してきます.

オブジェクト要素は静的データ定義と同様に CRD ルート要素以下にツリー構造で配置します.

以下に記述例を示します.

```
<Controller name="RC5" option="Conn=eth:10.8.109.126" provider="CaoProv.DENSO.NetwoRC">
    <Robot name="Arm1"/>
    <Variable name="I0100"/>
</Controller>
```

以下に各オブジェクト要素の属性をまとめます.

**表 3-1 オブジェクト要素と属性**

| オブジェクト要素 | 必須属性 | 任意属性 |
|---|---|---|
| Controller | Name<br>Provider | Machine<br>Option |
| Command | Name | Option |
| Extension | | |
| File | | |
| Robot | | |
| Task | | |
| Variable | | |
| Message | Number | - |

### 3.4.4. デバイスのケイパビリティ定義

デバイスのケイパビリティ定義では, オブジェクト要素とケイパビリティ定義要素を XML ファイルに追加してきます.

(1) オブジェクト要素を追加する.

オブジェクト要素は静的データ定義と同様に CRD ルート要素以下にツリー構造で配置します. 全てのオブジェクト要素は, 任意で name 属性と key 属性を付加することができます.

name 属性はオブジェクト名を示し, name 属性がないオブジェクトは名前を特に指定しないオブジェクトとして識別されます.

key 属性は同じ値を持つケイパビリティ定義要素をそのオブジェクトのケイパビリティとして識別します. key 属性がない場合は, デフォルトのケイパビリティ定義要素を使用すると識別されます.

以下にオブジェクト要素の例を示します.

```
<Controller>
  <File/>
  <Robot/>
```

```
    <Task/>
    <Variable name="@CURRENT_TIME" key="SYS_VAR_CURRENT_TIME"/>
  </Controller>
```

上記の例では以下の内容を表現しています.

- 全てのコントローラ及びその下にあるファイル, ロボット, タスクオブジェクトはデフォルトのケイパビリティ定義要素を使用します.
- 全てのコントローラの下にある“@CURRENT_TIME”変数オブジェクトは, “SYS_VAR_CURRENT_TIME”のキーを持つケイパビリティ定義要素を参照します.

(2) ケイパビリティ要素を追加する.

各オブジェクトが参照するケイパビリティ定義要素は CRD ルート要素の下に配置します. ケイパビリティ定義要素は, コントローラオブジェクトのメンバの実装状況や引数及び戻り値のデータ型や値の範囲を指定します. 各ケイパビリティ要素で定義できるメンバは異なります. 詳細はを参照してください.

ケイパビリティ要素はコントローラケイパビリティ要素の中に他のオブジェクトケイパビリティを記述します. これによりコントローラ単位でケイパビリティ定義をすることができます.

各ケイパビリティ定義要素は, ObjectKey 要素を持つことができます. これは, オブジェクト要素で指定したケイパビリティを一意に判断するためのキーです. ObjectKey 要素を持たないケイパビリティ定義要素は, デフォルトのケイパビリティを定義していると判断します.

以下にケイパビリティ定義要素の例を示します.

```
<Controller_Info>
  <Args>
    <Arg index="1">
      <HelpString>コントローラ名</HelpString>
      <VarInfo>
        <DataInfo>
          <Min>0</Min>
          <Max>10</Max>
        </DataInfo>
      </VarInfo>
    </Arg>
    <Arg index="2"/>
  </Args>
  <Variable_Info>
    <ObjectKey>SYS_VAR_CURRENT_TIME</ObjectKey>
    <Put_Value_Info>
      <Args>
        <Arg index="1">
          <VarInfo type="VT_DATE"/>
        </Arg>
      </Args>
    </Put_Value_Info>
    <Get_Value_Info>
      <Result>
        <VarInfo type="VT_DATE"/>
      </Result>
    </Get_Value_Info>
  </Variable_Info>
</Controller_Info>
```

上記の例では以下の内容を表現しています.

・ デフォルトを指定したコントローラの AddController の引数の数は2個である.
・ デフォルトを指定したコントローラの AddController の第 1 引数はコントローラ名である.
・ デフォルトを指定したコントローラの AddController の第 1 引数の値は 1～10 の間の値を取る.
・ デフォルトを指定したコントローラ以下で Key = “SYS_VAR_CURRENT_TIME”を指定した変数は, Put_Value(), Get_Value()が実装されている.
・ 上記変数の Put_Value()は VT_DATE 型を設定する.
・ 上記変数の Get_Value()は VT_DATE 型を取得する.

## 3.5. CRD サンプル

ここでは分かりやすいように CRD の三つ用途に沿ってサンプルを紹介します. もちろん一つにまとめることもできますが, 実際の運用においてもこのような用途毎に分割した使われ方が一般的だと考えられます.

### 3.5.1. 静的データの定義サンプル

**List 3-1 CRDSchema2.xsd**

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!--
    ********** [ORiN2] CRD Sample **********
-->
<CRD xsi:schemaLocation="http://www.orin.jp/CRD/CRDSchema ../../Schema/CRDSchema2.xsd"
     xmlns=http://www.orin.jp/CRD/CRDSchema
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <Help>CRD Test Data</Help>
  <Version>1.0.0</Version>
  <Controller name="RC1">
    <Attribute>0</Attribute>
    <Help>Sample Ctrl</Help>

    <!-- VT_I2 型の場合 -->
    <Variable name="Var1">
      <Value type="VT_I2">
        <iVal>10</iVal>
      </Value>
    </Variable>

    <!-- VT_I2 型の 1 次元配列の場合 -->
    <Variable name="Var2">
      <Value type="VT_ARRAY">
        <array type="VT_I2">
          <!-- 配列の情報 -->
          <dimension>1</dimension>
          <arrayBound>
            <lBound>0</lBound>
            <elements>3</elements>
          </arrayBound>
          <!-- データ -->
          <arrayData>
            <iVal>0</iVal>
            <iVal>1</iVal>
            <iVal>2</iVal>
          </arrayData>
        </array>
      </Value>
    </Variable>
  </Controller>
```

```
</CRD>
```

### 3.5.2. システムコンフィギュレーション定義サンプル

**List 3-2 CRDSchema2.xsd**

```
<?xml version="1.0" encoding="UTF-8"?>
<CRD xmlns=http://www.orin.jp/CRD/CRDSchema
     xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
     xsi:schemaLocation=" http://www.orin.jp/CRD/CRDSchema  ../../Schema/CRDSchema2.xsd">

  <Controller name="RC5" option="Conn=eth:10.8.109.126" provider="CaoProv.DENSO.NetwoRC">
    <Robot name="Arm1"/>
    <Variable name="IO100"/>
  </Controller>
</CRD>
```

### 3.5.3. デバイスのケイパビリティ定義サンプル

**List 3-3 CRDSample_DC.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<CRD xsi:schemaLocation=http://www.orin.jp/CRD/CRDSchema ../../Schema/CRDSchema2.xsd
     xmlns=http://www.orin.jp/CRD/CRDSchema
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <Controller_Info>
    <Args>
      <Arg index="1">
        <HelpString>コントローラ名</HelpString>
        <VarInfo>
          <DataInfo>
            <Min>0</Min>
            <Max>10</Max>
          </DataInfo>
        </VarInfo>
      </Arg>
      <Arg index="2"/>
    </Args>
    <Get_VariableNames_Info>
      <Result>
        <VarInfo type="VT_ARRAY|VT_VARIANT"/>
      </Result>
    </Get_VariableNames_Info>
    <Execute_Info>
      <Args>
        <Arg index="1">
          <VarInfo>
            <DataInfo>
              <List>getAutoMode</List>
              <List>putAutoMode</List>
            </DataInfo>
          </VarInfo>
        </Arg>
        <Arg index="2">
          <VarInfo type="VT_EMPTY"/>
          <VarInfo type="VT_I2">
            <DataInfo>
              <List>10</List>
              <List>20</List>
            </DataInfo>
```

```
      </VarInfo>
      <VarInfo type="VT_I4">
        <DataInfo>
          <Min>-1</Min>
          <Max>100</Max>
        </DataInfo>
      </VarInfo>
    </Arg>
  </Args>
  <Result>
    <VarInfo type="VT_EMPTY"/>
    <VarInfo type="VT_I2"/>
    <VarInfo type="VT_I4"/>
  </Result>
</Execute_Info>

<File_Info>
  <Get_Value_Info>
    <Result>
      <VarInfo type="VT_BSTR"/>
      <VarInfo type="VT_ARRAY|VT_UI1"/>
    </Result>
  </Get_Value_Info>
  <Get_VariableNames_Info>
    <Result>
      <VarInfo type="VT_ARRAY|VT_VARIANT"/>
    </Result>
  </Get_VariableNames_Info>
  <Copy_Info/>
  <Delete_Info/>
  <Move_Info/>
</File_Info>

<Robot_Info>
  <Halt_Info/>
  <Move_Info/>
  <Speed_Info/>
</Robot_Info>

<Task_Info>
  <Start_Info/>
  <Stop_Info/>
</Task_Info>

<Variable_Info/>

<Message_Info>
  <Clear_Info/>
</Message_Info>

<Variable_Info>
  <ObjectKey>SYS_VAR_CURRENT_TIME</ObjectKey>
  <Put_Value_Info>
    <Args>
      <Arg index="1">
        <VarInfo type="VT_DATE"/>
      </Arg>
    </Args>
  </Put_Value_Info>
  <Get_Value_Info>
    <Result>
      <VarInfo type="VT_DATE"/>
    </Result>
  </Get_Value_Info>
</Variable_Info>
```

```
  </Controller_Info>

  <Controller>
    <File/>
    <Robot/>
    <Task/>
    <Variable name="@CURRENT_TIME" key="SYS_VAR_CURRENT_TIME"/>
  </Controller>
</CRD>
```

# 4. CAP プログラミングガイド

## 4.1. 概要

CAP はインターネット経由で CAO プロバイダにアクセスするための通信プロトコルです. ORiN2 SDK では, SOAP を利用してインターネット経由での分散オブジェクト技術を実現しています.

本章では, 実際にWebサーバを構築し, CAPを利用したプロバイダ呼び出し方法の解説をおこないます. 本章の構成は, 前半で基礎知識として SOAP を解説し, CAP の動作概要を説明します. 後半では主に CAP を利用するための環境構築方法について説明し, 最後にサンプルプログラムの作成をおこないます.

## 4.2. 基礎知識

### 4.2.1. SOAP

SOAP とは, XML フォーマットで記述されたメッセージを HTTP で送信することにより, リモートの PC 上にあるオブジェクトを呼び出すためのプロトコルです.

リモートの PC 上のオブジェクトを呼び出す技術としては, CORBA や DCOM などがあります. しかし, これらの技術は, 独自のポート番号を利用して通信をおこなうために, ファイアウォールを介して通信をおこなう場合, 特定のポートに対して通信を許可する設定をおこなう必要があります. 近年では, DCOM のセキュリティホールを利用したワームなどが存在しますので, 一般的には, DCOM のポートに通信を許可する設定はおこないません.

そこで, これらの問題を解決するために SOAP という技術が開発されました. SOAP では, オブジェクト間でやり取りされるメッセージの構造を規定しているだけなので, 通信用プロトコルには既存の HTTP や SMTP を利用します. これにより, HTTP や SMTP が通信可能であれば, SOAP を利用することにより, ファイアウォールを介しての分散オブジェクト技術が利用可能になります.

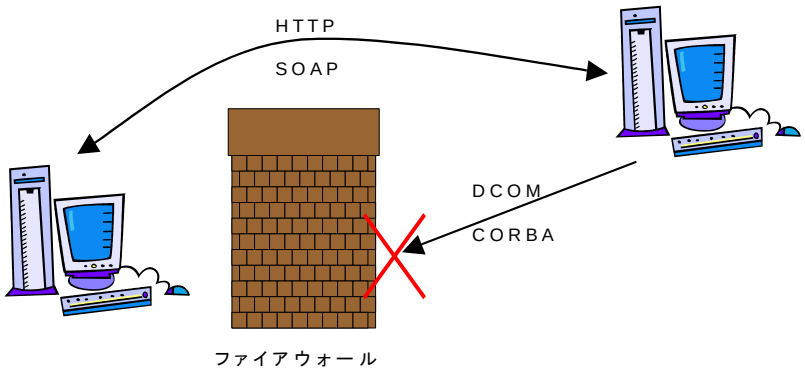



**図 4-1 SOAP のイメージ**

## 4.3. CAP プロバイダと CAP リスナ

CAO プロバイダにインターネット経由でアクセスするために, ORiN2 SDK では, CAP という仕様を規定し

ています. クライアントとサーバ間で, この CAP の仕様に準じたメッセージの送受信と解析をおこなうことで CAO プロバイダのリモートアクセスをおこなうことができます.

しかし, CAP に準じたメッセージの送受信と解析処理を作成するためには手間がかかるため, ORiN2 SDK では, CAP プロバイダと CAP リスナを提供しています.

CAP プロバイダと CAP リスナの動作概要を図 4-2 に示します. 図に示すように CAP プロバイダは, リモートプロバイダへとアクセスするためのメッセージを生成・送信するためのプロバイダです. このメッセージは Web サーバへが公開している CAP リスナへと送信されます. CAP リスナとは, CAP に準したメッセージを受信・解析し, COM コンポーネント(CAO プロバイダ)を呼び出すためのインタフェースです.

このように, CAP プロバイダ・リスナを経由することにより, 他のローカルなプロバイダと同じようにリモートのプロバイダを利用することが可能となります.

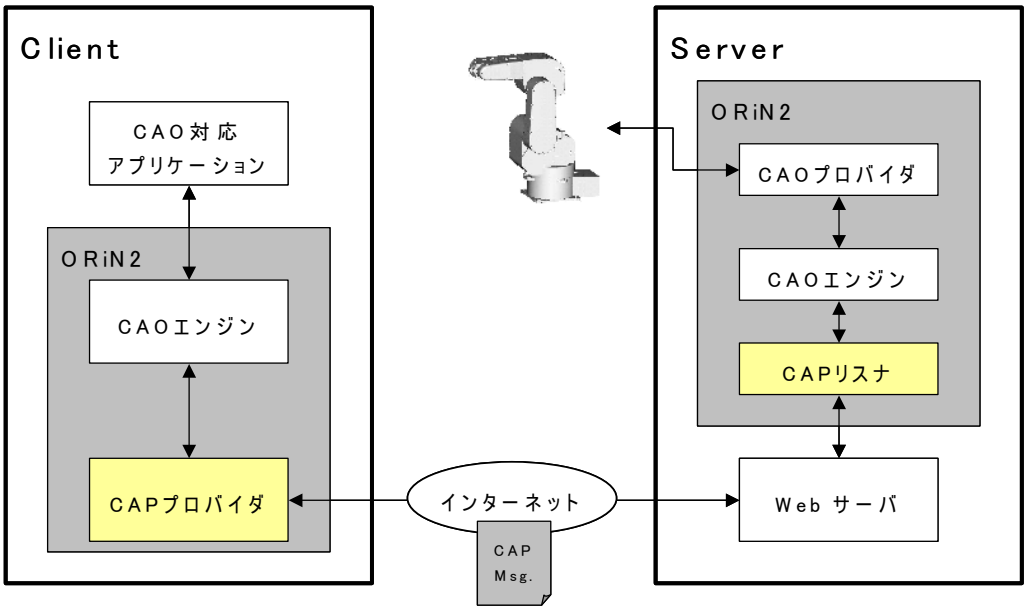



**図 4-2 CAP の概要**

## 4.4. 環境構築

本節では，CAP を利用するための環境構築の方法について解説します．

以降では，CAP プロバイダを利用して，リモートプロバイダを呼び出す側を“クライアント”，呼び出される側を“サーバ”と表現します．

なお，サーバ PC およびクライアント PC 共に ORiN2 SDK がインストールされているものとします．

### 4.4.1. サーバ側の設定

(1) Web サーバのインストール

上述したように，SOAP で送信されたメッセージは，Web サーバにより CAP リスナを起動し，メッセージを解析しています．そのため，まずは Web サーバを構築する必要があります．

Windows 2000 Server / Windows 2000 Professional / Windows XP Professional には，Web サーバ IIS(Internet Information Service)が標準で添付されています[15]．しかし，デフォルトインストールでは，インストールされていないため，追加インストールをおこなう必要があります．(Windows 2000 Server の場合は，標準でインストールされています．)

本項では，Windows XP Professional を例に挙げて説明をおこないます．

スタートメニューより，「設定」→「コントロールパネル」→「プログラムの追加と削除」を起動し，「Windows コンポーネントの追加と削除」を選択します．

図 4-3 に示すような画面が表示されますので，その中からインターネットインフォメーションサービス(IIS)のチェックボックスにチェックを入れます．


**図 4-3 Windows コンポーネントの追加**

[15] Windows 2000 Professional / Windows XP Professional 付属の IIS は Windows 2000 Server のものと比較すると機能が制限されています．本格的なサービスを提供したい場合は Windows 2000 Server を利用してください．また，Windows XP Home Edition には，IIS は付属していませんのでご注意ください．

図 4-4 に示すような画面が表示されますので，「WWW(World Wide Web)サービス」をダブルクリックし，すべてのコンポーネントのチェックボックスにチェックを入れてください．「インターネットインフォメーションサービススナップイン」と「共通コンポーネント」には，自動的にチェックが入ります．

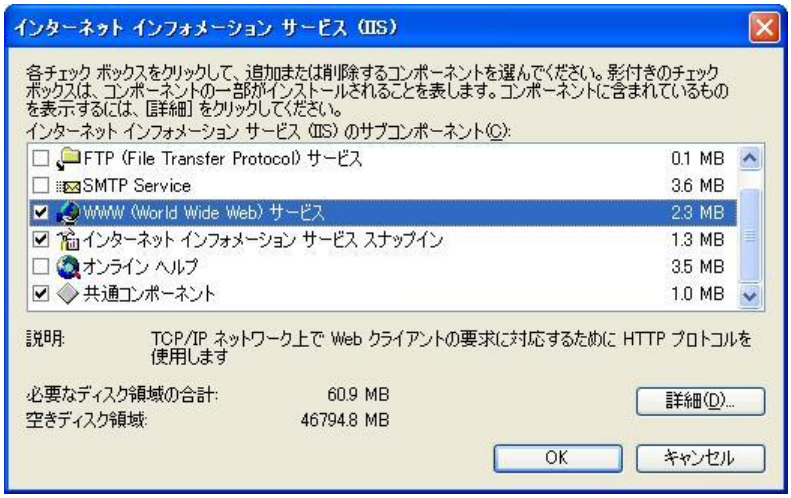



**図 4-4 インターネットインフォメーションサービスの追加**

インストールするコンポーネントを選択して，「次へ」ボタンを押下すると，Windows コンポーネントのインストールが開始されます．

インストール中に，「Widows XP Professional インストールディスク」が要求されます．

(2) SOAP Toolkit のインストール

次に，SOAP で送信されてきたメッセージを解析するために，Microsoft SOAP Toolkit 3.0 をインストールします．

ダウンロードした soapsdk.exe をダブルクリックし，指示に従ってボタンを押下するとインストールは完了します．設定はデフォルトのままでかまいません．

**表 4-1 Soap Toolkit 3.0 ダウンロードサイト**

| ファイル名 | URL |
|---|---|
| soapsdk.exe | http://www.microsoft.com/downloads/details.aspx?FamilyID=c943c0dd-ceec-4088-9753-86f052ec8450&DisplayLang=en |

(3) 仮想ディレクトリの作成

Microsoft SOAP Toolkit 3.0 の SOAPVDIR.CMD を使用して仮想ディレクトリを作成します. ここでの仮想ディレクトリとは, HTTP でアクセスする際のアドレスを, 物理パスに割り当てたものです.

SOAPVDIR.CMD は以下のディレクトリにあります.

<SOAP インストールディレクトリ>¥Binaries

作成にはコマンドプロンプトで以下のコマンドを入力します.

SOAPVDIR.CMD CREATE Cap <CapListener の Bin ディレクトリ>

(4) IIS の設定

次に, IIS から CAP リスナを起動するための設定をおこないます.

スタートメニューより,「設定」→「コントロールパネル」→「管理ツール」を起動し,「インターネットインフォメーションサービス」を起動します.

インターネットインフォメーションサービスから,「ローカルコンピュータ」→「Web サイト」→「既定の Web サイト」を選択すると, 先ほど作成した仮想ディレクトリ「Cap」が存在することを確認できます.

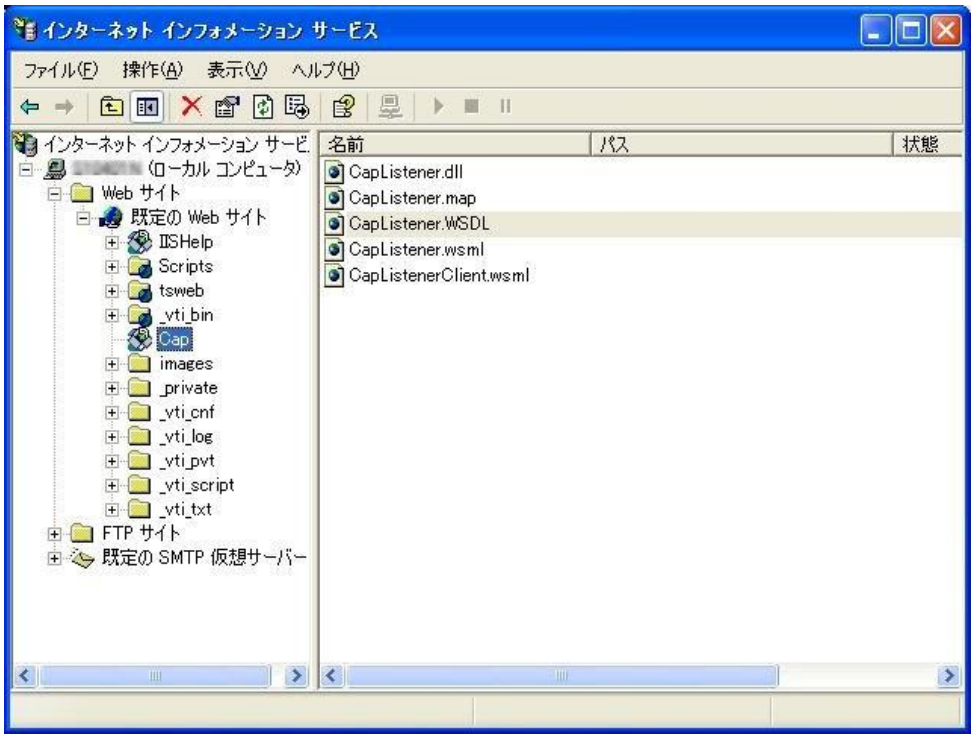



**図 4-5 IIS の設定画面**

ここで, Cap ディレクトリ内にある以下の二つのファイルを右クリックし, プロパティを表示させ,「読み取り」のチェックボックスにチェックを入れてください.

・ CapLister.WSML

・ CapListerClient.WSML

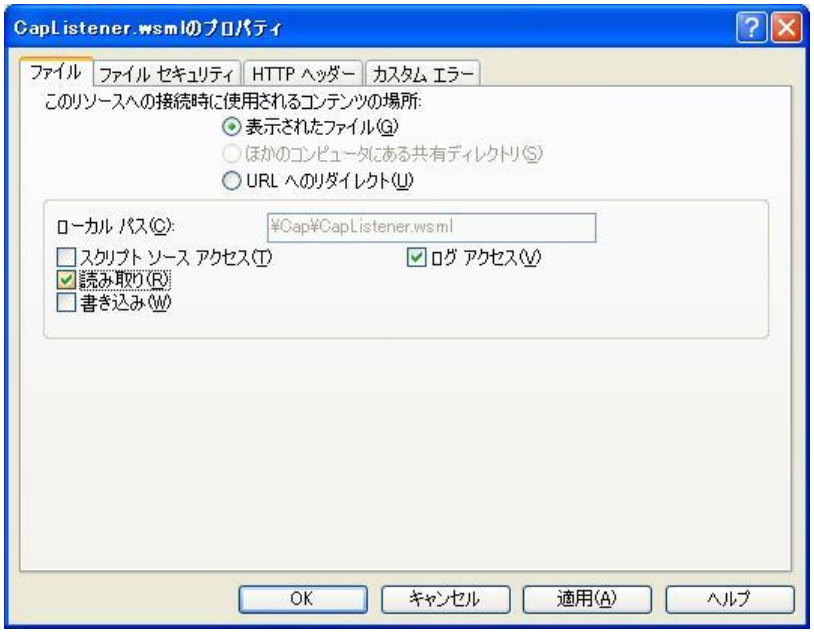



**図 4-6 WSML ファイルの設定画面**

(5) CAO エンジンの起動権限設定

次は，インターネット経由でアクセスしてきたユーザが，CAO エンジンを起動できるように権限の設定をおこないます．

<ORiN インストールディレクトリ>¥CAO¥Engine¥Bin¥CAO.EXE を右クリックし，「プロパティ」→「セキュリティ」→「追加」選択してください．

図 4-7 のようなダイアログが表示されますので，インターネットゲストアカウントを追加し，起動権限を追加します．

インターネットゲストアカウント名は，スタートメニューから「設定」→「コントロールパネル」→「ユーザーアカウント」で確認することができます．

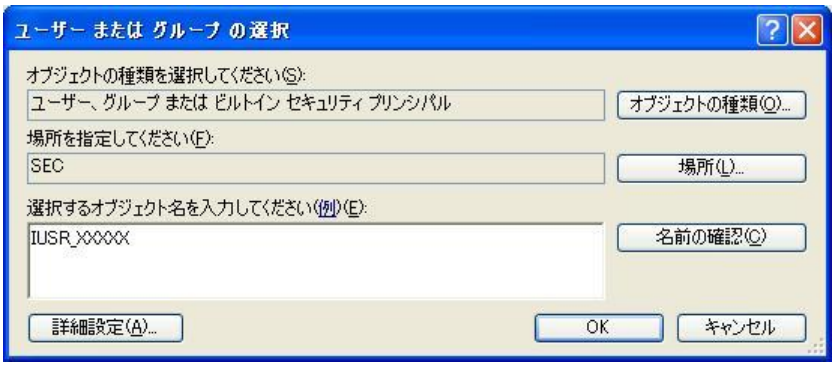


**図 4-7 インターネットゲストアカウントの追加**

アカウントの追加に成功すると，図 4-8 のようにアカウント名が表示されます．

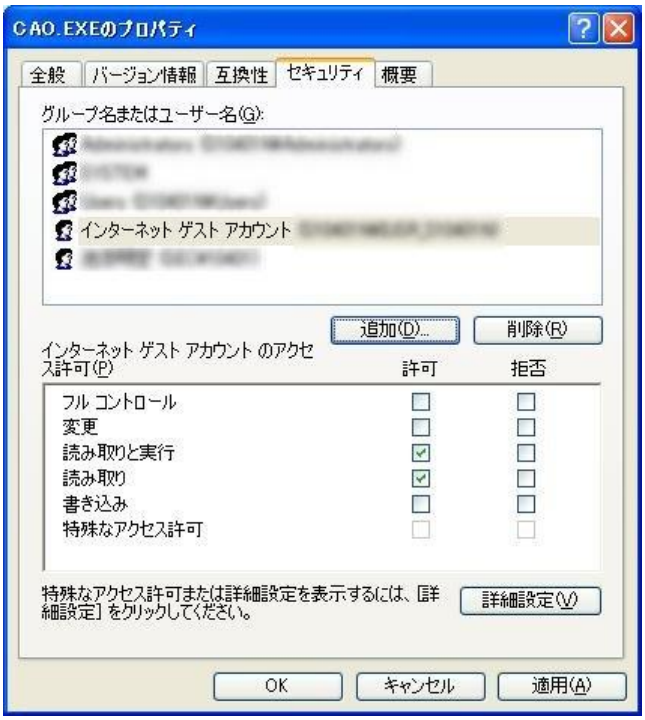



**図 4-8 インターネットゲストアカウント追加結果**

(6) CapListner.WSDL ファイルの設定

CapListener ディレクトリ内にある CapListner.WSDL ファイルの 3459 行目のサーバ名だけを実際の Web サーバの名前に変更します．

(例)soap:address location='http://**HOSTNAME**/Cap/CapListener.WSDL' />

(7) Windows XP SP2 の場合

Windows XP Service Pack 2(SP2)を適用している場合，デフォルトでは，ファイアウォールの設定によりIIS が利用できないため，CAP を利用することはできません．この問題を解決するには，以下の 2 種類の方法があります．

- Windows ファイアウォールの設定を“無効にする”
- IIS の接続を例外として許可する設定をおこなう

既にファイアウォールなどで PC が十分に保護されている場合には，Windows ファイアウォールの設定を無効にすることができます．ファイアウォールの設定を無効にする場合は，以下に記述する設定を行わなくても CAP プロバイダを利用することが可能です．その場合は，図 4-9 に示すように，スタートメニューより，「コントロールパネル」→「Windows ファイアウォール」を起動し，“無効”にチェックを入れてください．

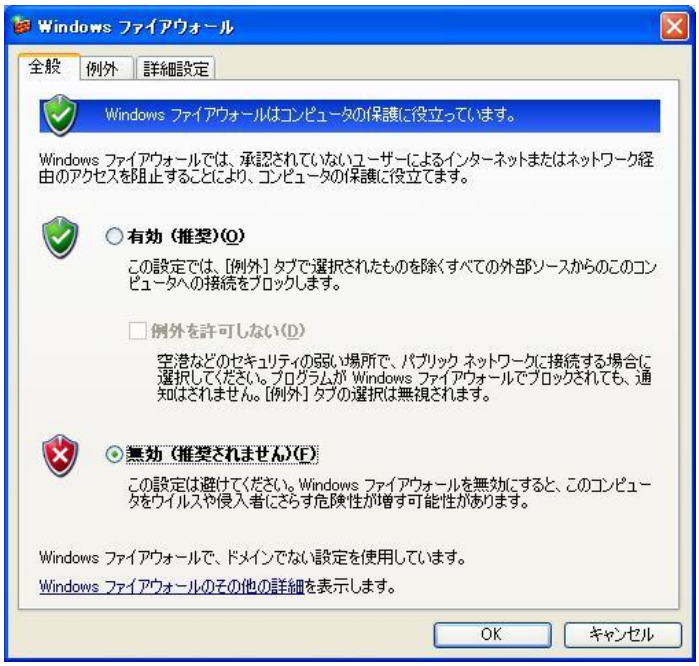



**図 4-9 Windows ファイアウォールの無効化**

ファイアウォールで IIS の接続を許可する場合は，以下の設定をおこないます．

スタートメニューから「コントロールパネル」→「Windows ファイアウォール」を起動し，「詳細設定」タブを選択してください．

図 4-10 に示すように，「ネットワーク接続の設定」の中から，現在の環境に応じた接続方法を選択し，「設定」ボタンを押下してください．(今回の例では，“ローカルエリア接続”を選択しました)

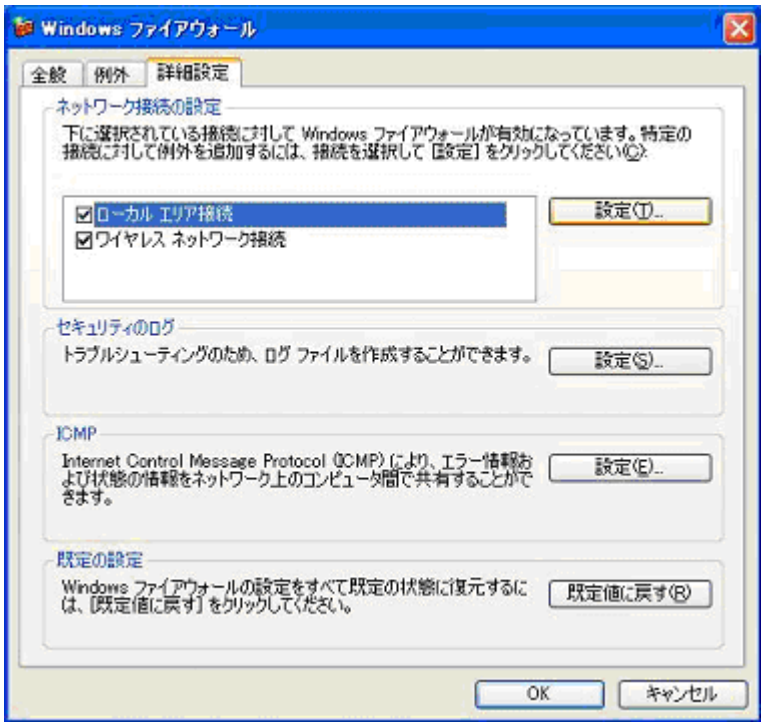



**図 4-10 接続の例外設定**

図 4-11 のようなダイアログが表示されますので，“Web サーバー(HTTP)”にチェックを入れてください．


**図 4-11 例外に追加するサービスの選択**

### 4.4.2. クライアント側

(1) SOAP Toolkit のインストール

Microsoft SOAP Toolkit 3.0 をインストールします. (表 4-1 参照)

## 4.5. サンプルプログラム

CAP の動作確認をおこなうために, サンプルアプリケーションを作成します. VB6 を起動して, 二つのボタンと一つのテキストボックスを持つフォームを作成し, 以下のようなコードを記述してください.

**List 4-1 Sample.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( _                                          (1)
        "RC1", "CaoProv.CAP", "", "Provider=CaoProv.DataStore,Server=XXXXX")
    ' 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()
    Text1.Text = var
End Sub
```

ソースコード中の(1)を見てください. この AddController メソッドにより, インターネット経由でのプロバイダ接続をおこなっています. このメソッドの第 1～第 3 引数は, 他のプロバイダと同様のものです. 第 4 引数の内容を以下に示します.

Provider : インターネット経由でアクセスするプロバイダ名

Server : Web サーバのホスト名

Option : リモートアクセスするプロバイダのオプション文字列

このように，CAP プロバイダを利用して，AddController メソッドで取得した CaoController オブジェクトは，指定したプロバイダ名のコントローラオブジェクトとして利用することができます．

上記例の場合では，ctrl オブジェクトは，DataStore プロバイダのコントローラオブジェクトとして扱うことができます．

# 5. 環境設定

## 5.1. DCOM 設定手順

ORiN2 SDK において，CAO または CAO プロバイダをリモート起動させたい場合，DCOM のコンフィギュレーションツールを使って設定を行う必要があります．DCOM クライアントおよびサーバアプリケーションにセキュリティを正しく設定していないと，エラーが発生しリモート起動またはアクセスに失敗します．

ここではそのコンフィギュレーションツール DCOMCNFG.exe を使った DCOM の設定手順を解説します．詳しくは下記の参考資料を参照してください．

・ COM / COM+ セキュリティ
http://www.microsoft.com/japan/com/compapers.asp - comseq

・ [VB5] DCOMCNFG.EXE で VB5 アプリに DCOM 設定をする方法
http://support.microsoft.com/default.aspx?scid=kb;ja;JP183607

・ DCOM デプロイメントガイド，翔泳社，Frank E. Redmond III 著，トップスタジオ 訳，金子 良治 監修

ここで紹介する内容は，CAO，またはCAOプロバイダをDCOMでリモート起動するための設定で，本書の例ではセキュリティ設定がローレベル(セキュリティがかかっていない状態)になっています．この設定では外部から悪意のあるプログラムに攻撃される可能性があるので十分に注意してください．可能であれば，個々の環境に合わせて設定の見直しをおこなってください．

### 5.1.1. 準備

(2) DCOM サーバとなるマシンと DCOM クライアントとなるマシンを用意してください．
ここでは以下の OS を対象とします．

**表 5-1 OS と DCOM の対応**

| OS | DCOM サーバ | DCOM クライアント |
|---|---|---|
| Windows95 | ○ | ○ |
| Windows98 | ○ | ○ |
| WindowsNT 4.0 WorkStation | ○ | ○ |
| WindowsNT 4.0 Server | ○ | ○ |
| Windows2000 Professional | ○ | ○ |
| Windows2000 Server | ○ | ○ |
| Windows XP Professional | ○ | ○ |

(3) DCOM クライアントとサーバを同一ドメインに参加させてください．

### 5.1.2. WindowsNT / 2000 での設定項目

(1) DCOMCNFG.EXE を起動する

スタートメニューの“ファイル名を指定して実行”から“dcomcnfg”と入力し, DCOMCNFG.EXE を起動します.

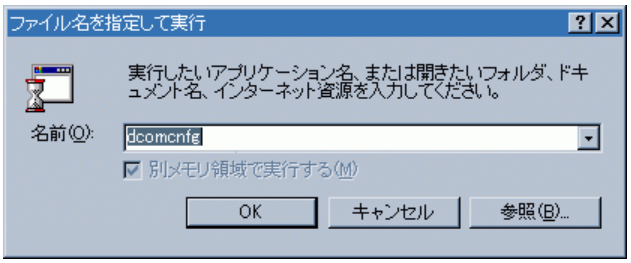



**図 5-1 DCOMCNFG の起動**

(2) 既定値の設定

既定の項目設定をすることによって, ローカル PC にインストールされた全ての DCOM アプリケーションの既定となる値を編集することが可能です. よって, 慎重に設定する必要があります. また, セキュリティに問題がある場合はこの設定は変更しないでください. (但しその場合は CAO の DCOM 経由使用ができない場合があります)

「既定のプロパティ」タブを選択してください.

図 5-2 に示すように, “このコンピュータ上で分散 COM を使う”にチェックを入れてください. 既定の分散 COM 通信プロパティの「認証レベル」を“なし”に, 「偽装レベル」を“識別する”に設定してください.


**図 5-2 既定のプロパティ**

(3) アプリケーション毎の設定

開いた DCOMCNFG.EXE のタブから「アプリケーション」タブを選択し, DCOM 起動をさせたいアプリケーションをアクティブにし, プロパティボタンを押下してください.

この操作はリモート起動するアプリケーション全てに対しておこなう必要があります. つまり, プロバイダを DCOM 起動する場合は, 起動するプロバイダ全てに対してこの処理をおこないます. またリモート CAO を使用するのであれば, CAO にも同様の処理をする必要があります.

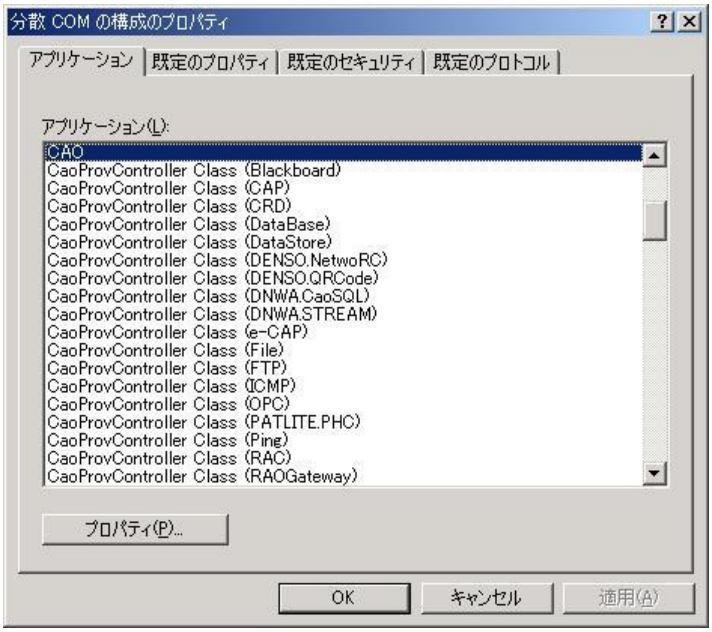



**図 5-3 アプリケーションごとの設定**

まずは,「全般」タブを選択してください. ここでは既定の認証レベルで設定した値を使用するので,「認証レベル」を“既定”にしてください.

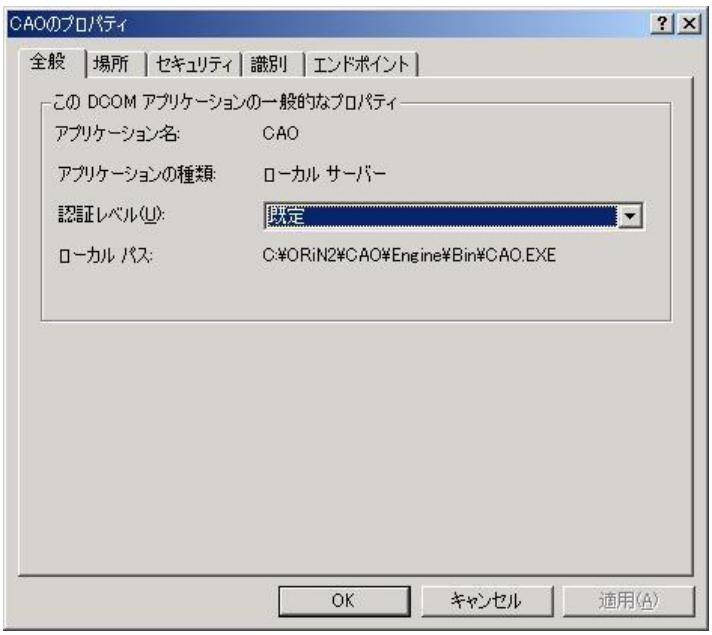



**図 5-4 全般設定**

次に「場所」タブを選択してください．ここではアプリケーションを実行する場所を指定することができます．“このコンピュータ上でアプリケーションを実行する”を選択してください．他のチェックは外しておいてください．


**図 5-5 場所設定**

次は「セキュリティ」タブを選択してください．

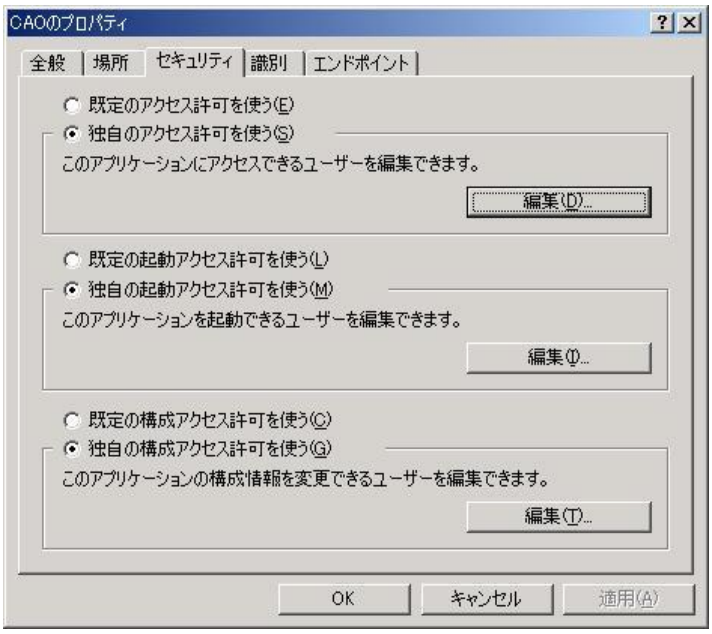



**図 5-6 セキュリティ設定**

「独自のアクセス権を使う」を選択し，「編集」ボタンを押下して以下のアクセス権を追加してください．

- Everyone - アクセスの許可
- Interactive - アクセスの許可
- System - アクセスの許可

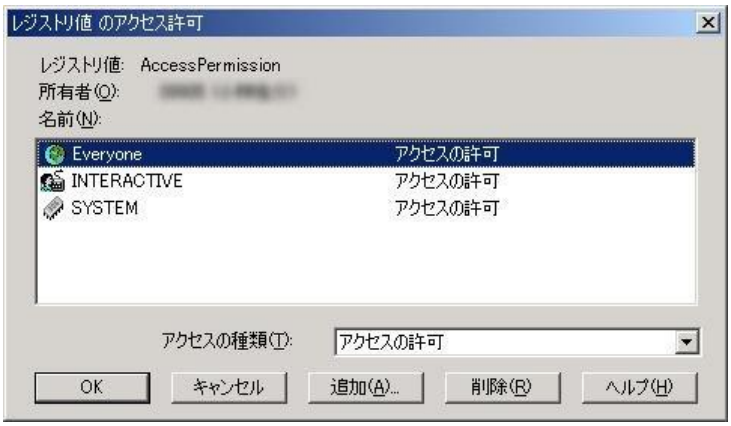



**図 5-7 アクセス権の追加**

同様に「独自の起動アクセス権を使う」を選択し，「編集」ボタンを押下して以下の起動アクセス権を追加してください．

- Everyone - アクセスの許可
- Interactive - アクセスの許可
- System - アクセスの許可

最後に「識別」タブを選択してください．そして，“対話ユーザ”を選択します．WindowsNT で DCOM サーバを起動する場合は“起動したユーザ”でも可能です．また“次のユーザ”で適宜ユーザを入力してもかまいません．但しサービス登録時には“起動ユーザ”と“対話ユーザ”を選択することはできません．(代わりにシステムアカウントが使えます)

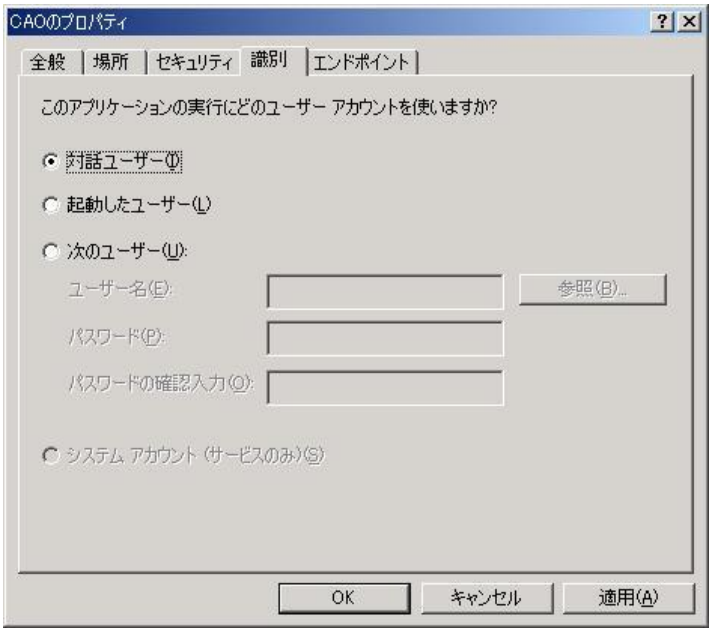



**図 5-8 識別設定**

### 5.1.3. Windows9x での設定項目

Windows98, Windows95 を使用する場合は以下の URL から OS に合わせて DCOM98 又は 95 と DCOMCNFG98&95 をダウンロードし, PC にインストールしてください.

< Microsoft COM テクノロジ 関連ファイル ダウンロード >

http://www.microsoft.com/com/default.mspx

(1) DCOMCNFG.EXE を起動する

スタートメニューの「ファイル名を指定して実行」から“DCOMCNFG”と入力し, DCOMCNFG.EXE を起動します.

(2) 既定のプロパティの設定

以下の様に設定してください.

- 認証レベル: なし
- 偽装レベル: 識別する

(3) 既定のアクセス権の設定

以下のアクセス権を追加してください.

- 世界 - 許可する

Windows 9x の DCOM98 ではリモート起動の機能がサポートされていませんので, 9x 系の PC をサーバにするためには予めそのプログラムを起動しておかなくてなりません. CAO エンジンではなく, プロバイダ DLL を予め起動しておきたい場合には dllhost.exe を利用して, コマンドプロンプトで下記のように起動してください.

```
> dllhost.exe {<AppID>}
```

例えば TCmini プロバイダの場合は, 下記のように実行します.

```
> dllhost.exe {2c140adc-c109-43df-a059-c3b116257658}
```

<AppID>は各プロバイダソースプロジェクトの CaoProvController.RGS ファイルに記載されています. また ORiN2 SDK 付属の CaoProvExec ツールを利用すると GUI を使って上記と同じ設定ができます. CaoProvExec の利用方法については,「CAO プロバイダ作成ガイド」を参照してください.

### 5.1.4. Windows XP での設定項目

(1) DCOMCNFG の起動

スタートメニューの“ファイル名を指定して実行”から“dcomcnfg”と入力し，DCOMCNFG.EXE を起動します．

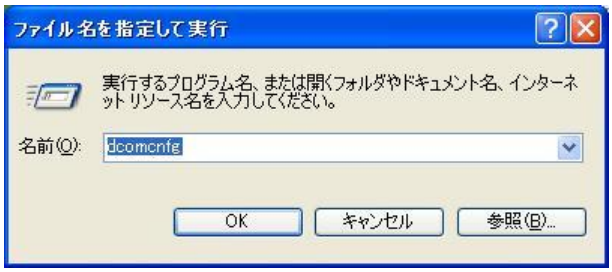



**図 5-9 DCOMCNFG の起動方法**

(2) マイコンピュータの設定

図 5-10 に示すように，「コンポーネントサービス」→「コンピュータ」→「マイコンピュータ」を選択し，右クリックから「プロパティ」を起動します．

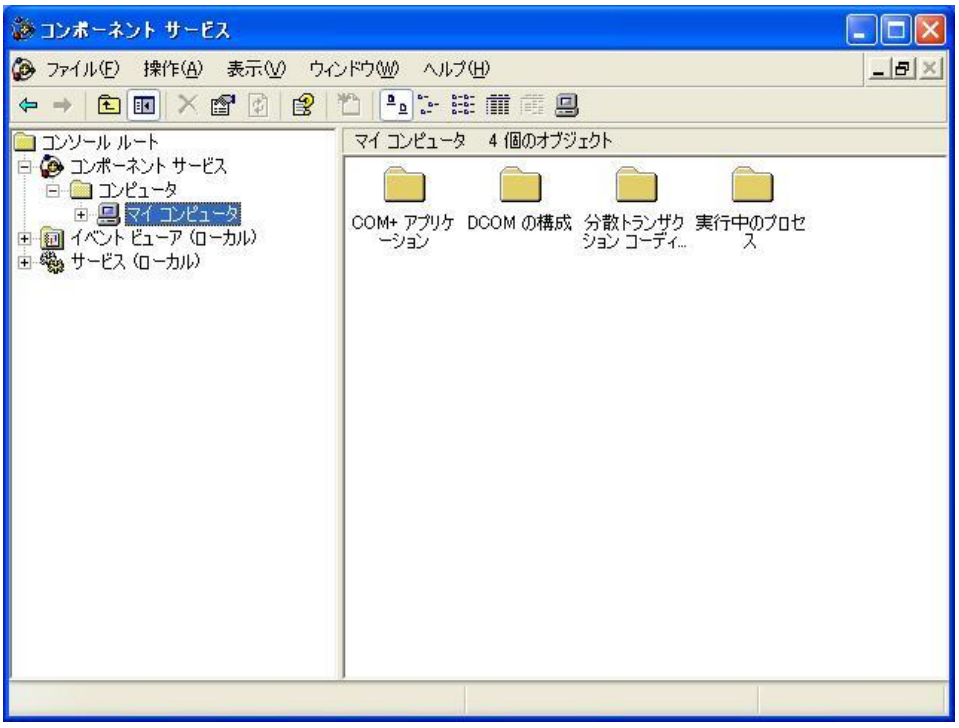



**図 5-10 DCOMCNFG の設定画面**

図 5-11 に示すように,「規定のプロパティ」タブを選択します.
「このコンピュータ上で分散 COM を有効にする」にチェックを入れます.
「規定の認証レベル」を“なし”,「規定の偽装レベル」を“識別する”に設定します.

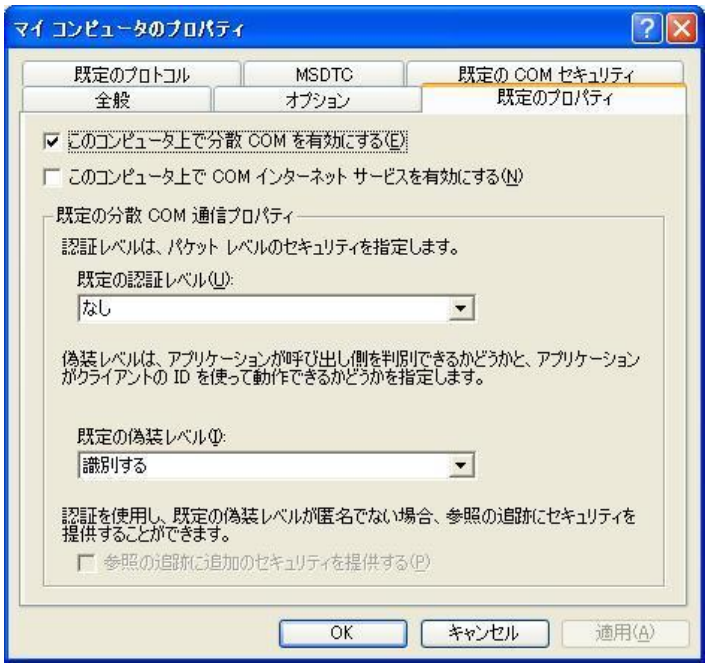



**図 5-11 マイコンピュータのプロパティ**

(3) アプリケーションごとの設定

「コンポーネントサービス」→「コンピュータ」→「マイコンピュータ」→「DCOM の構成」の中から，「CAO」を選択し，右クリックのプロパティを起動します．

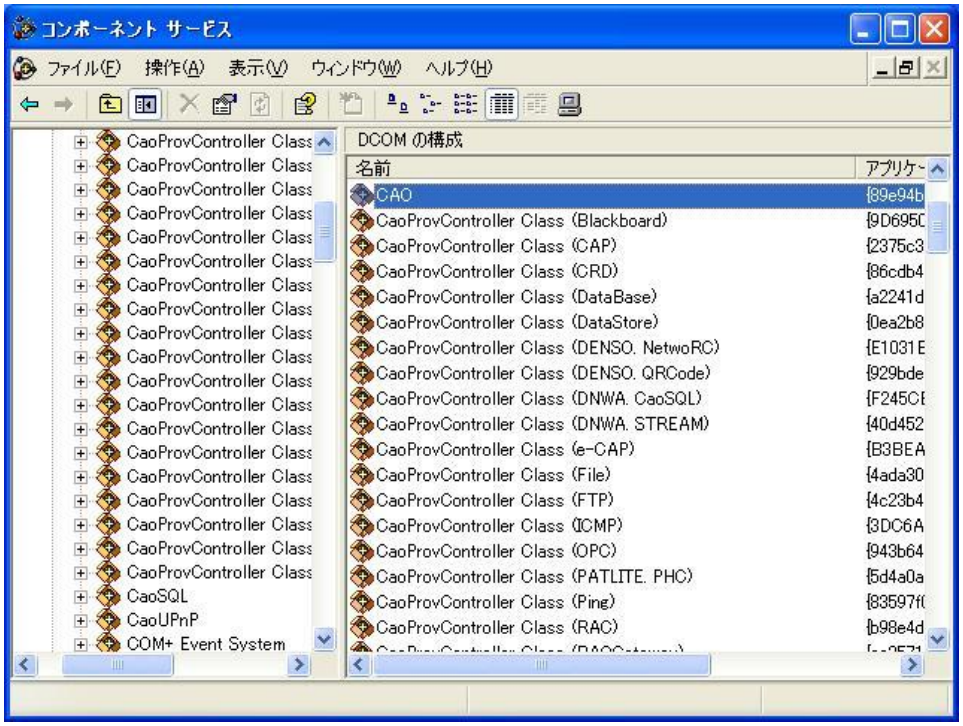



**図 5-12 DCOMCNFG DCOM の構成設定画面**

「全般」タブを選択し，「認証レベル」を“既定”に設定します．

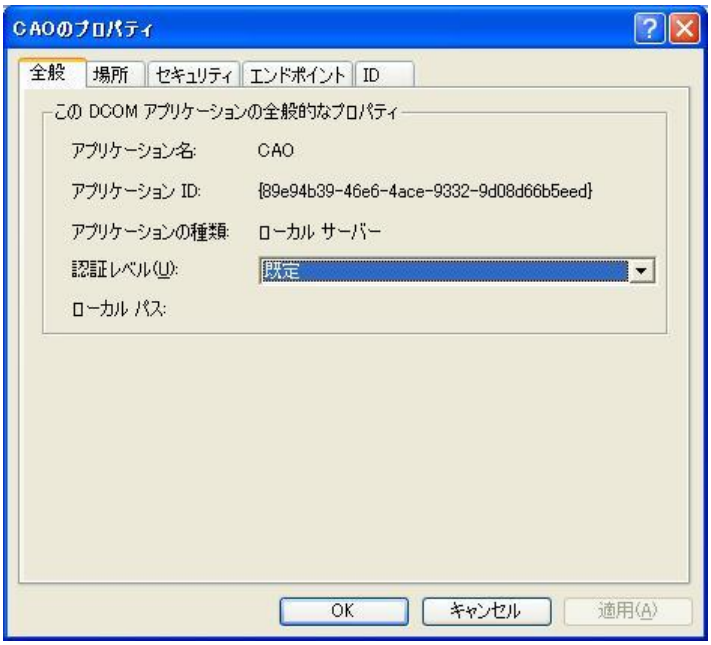



**図 5-13 アプリケーションごとの設定［全般］**

「場所」タブを選択し，“このコンピュータでアプリケーションを実行する”にチェックを入れます．

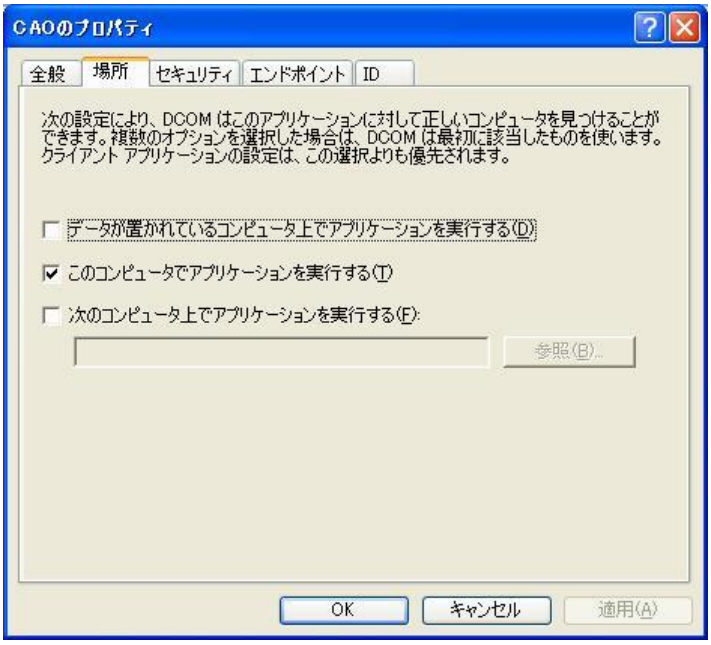



**図 5-14 アプリケーションごとの設定［場所］**

「セキュリティ」タブを選択し，「起動許可」および「アクセス許可」の“カスタマイズ”にチェックを入れ，

「編集」ボタンを押下します.

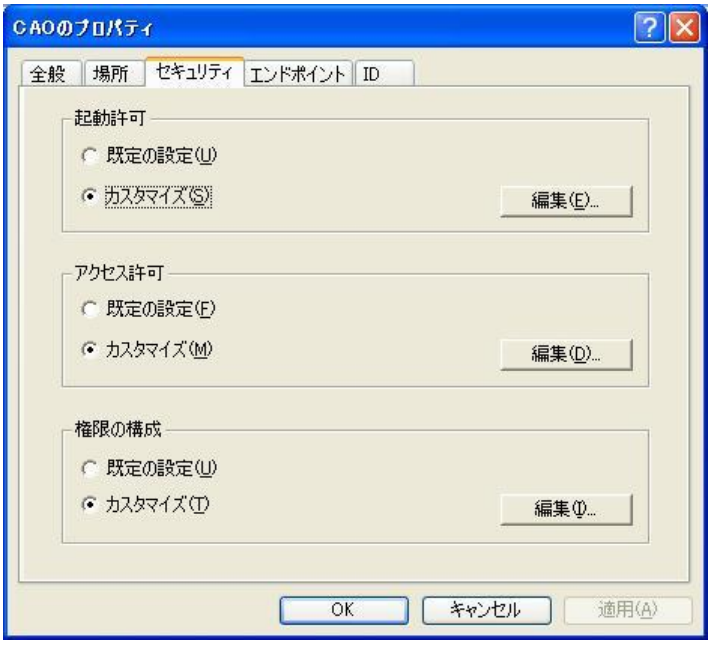



**図 5-15 アプリケーションごとの設定［セキュリティ］**

「追加」ボタンを押下し，“Everyone”と“Interactive”と“SYSTEM”というユーザを追加し，“アクセス許可”にチェックを入れます.

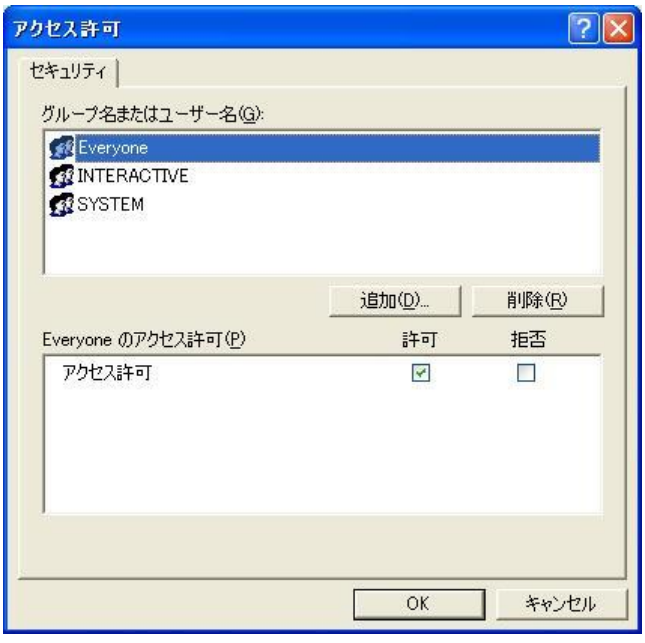



**図 5-16 アプリケーションごとの設定 ［セキュリティ アクセス許可］**

「ID」タブを選択し，“対話ユーザー”にチェックを入れます.

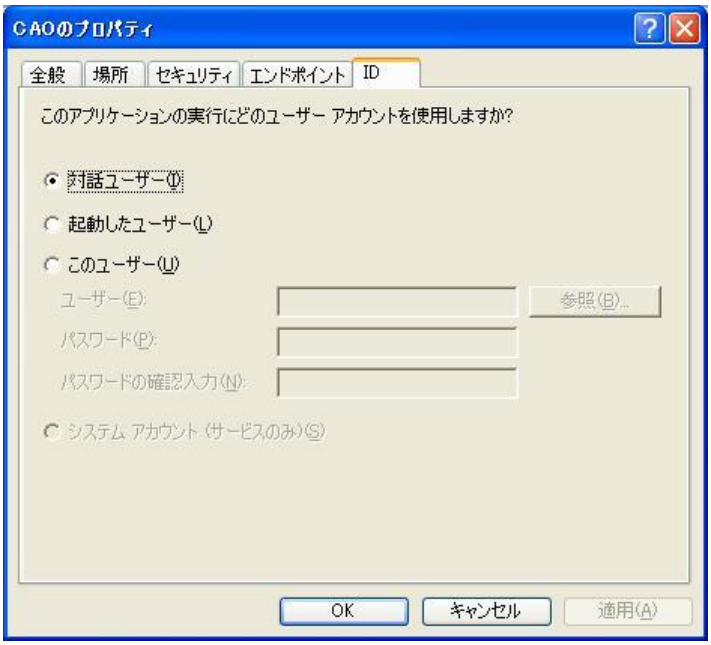



**図 5-17 アプリケーションごとの設定［ID］**

利用する CAO プロバイダの設定も同様におこないます.

### 5.1.5. Windows XP SP2 での設定項目

Windows XP Service Pack 2(SP2)は, より強固なセキュリティを確立するために, DCOM などのセキュリティ項目が強化され, さらにファイアウォールの機能がデフォルトで有効になっています.

そのために, リモートの CAO プロバイダにアクセスすることができません.

#### 5.1.5.1. Windows ファイアウォールの設定

既にファイアウォールなどで PC が十分に保護されている場合には, Windows ファイアウォールの設定を無効にすることができます. ファイアウォールの設定を無効にする場合は, 以下に記述する設定を行わなくてもリモートで CAO プロバイダを利用することが可能です.

スタートメニューの「設定」→「コントロールパネル」から「Windows ファイアウォール」を起動します.

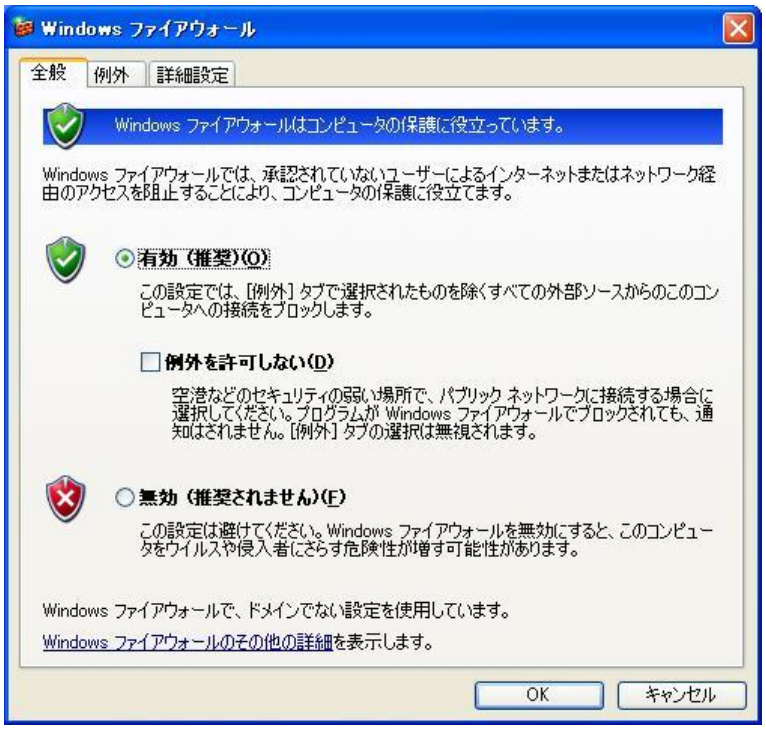



**図 5-18 Windows ファイアウォール設定**

#### 5.1.5.2. ファイアウォールの例外設定

スタートメニューの「設定」→「コントロールパネル」から「Windows ファイアウォール」を起動し，「例外」タブを選択します．

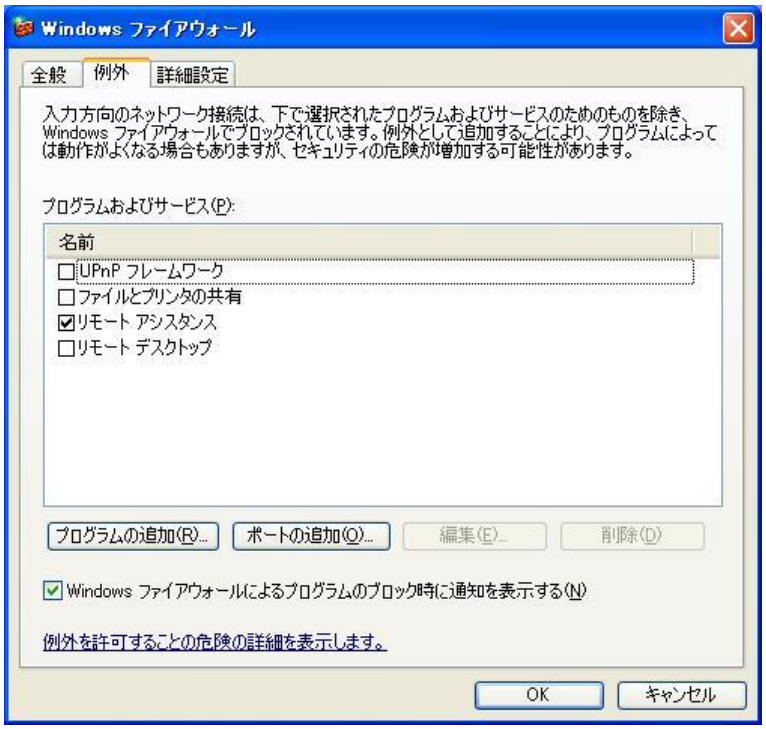



**図 5-19 例外処理の追加**

例外タブの「プログラムの追加」ボタンを押下すると図 5-20 に示すようなダイアログが表示されますので，以下のプログラムを追加してください．プログラムがリストに存在しない場合は，「参照」ボタンを押下してください．

<Windows インストールディレクトリ>¥WINDOWS¥system32¥dllhost.exe

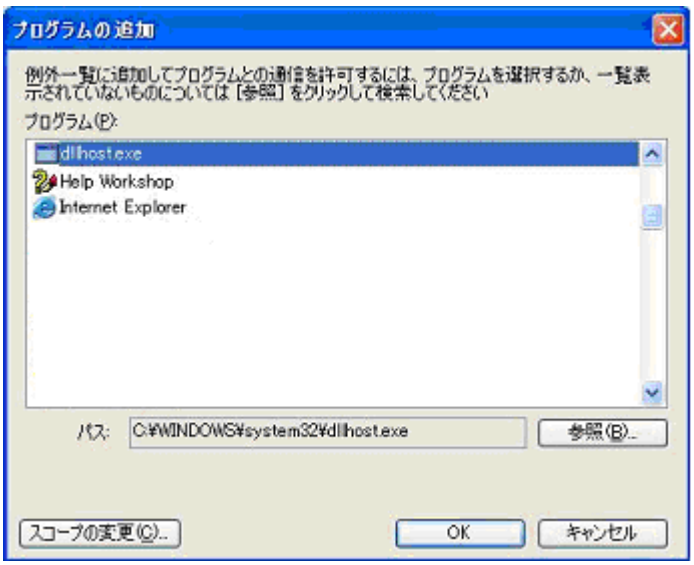



**図 5-20 例外プログラムの追加**

次に「Windows ファイアウォール」の「例外」タブで，ポートの追加を選択します．
ポートの追加ダイアログで，以下のポートを追加します．

- 名前：DCOM
- ポート番号：135

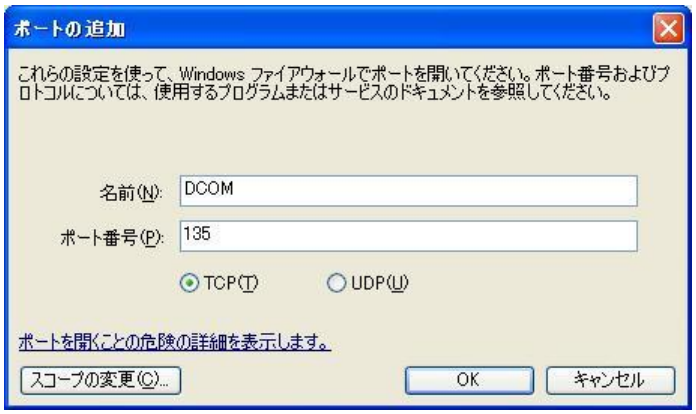



**図 5-21 例外ポート番号の追加**

図 5-22 に示すように，“DCOM”と“dllhost.exe”にチェックが入っていることを確認し，「OK」ボタンを押下します．以上でファイアウォールのセキュリティ設定は完了です．

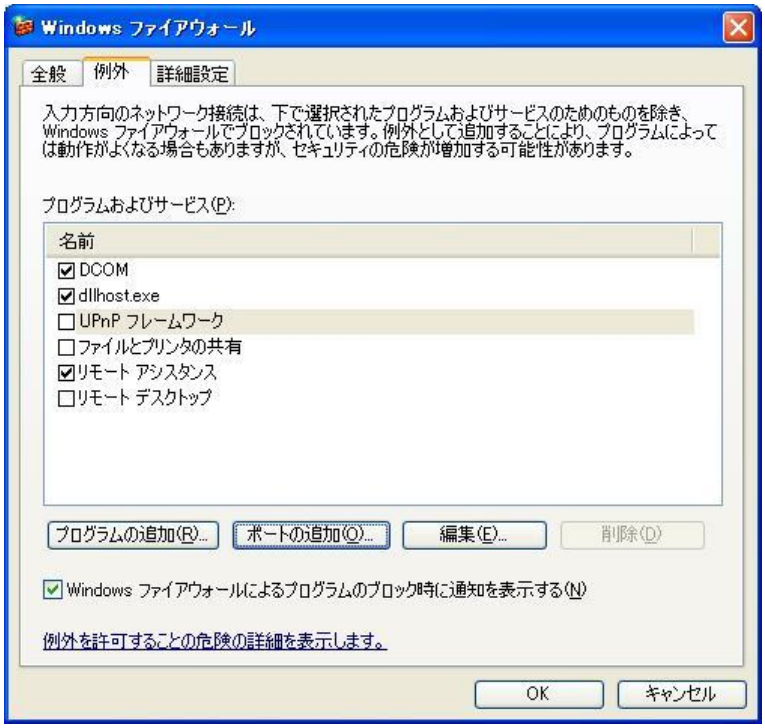



**図 5-22 例外の追加結果**

#### 5.1.5.3. DCOM の設定

5.1.4Windows XP での設定項目と同様の設定をおこないます．

### 5.1.6. DCOM の設定確認

DCOM の設定を確認するために，ORiN2 SDK に付属のテストツール“CaoTester”を利用して，リモートでのプロバイダ接続テストをおこないます．

CaoTester の詳細な利用方法は，CaoTester ユーザーズガイドを参照してください．

今回は，リモート接続をおこなうプロバイダとして，“DataStore プロバイダ”を利用します．DataStore プロバイダに対して，前述した「アプリケーションごとの設定」をおこなってください．

クライアント側の PC で CaoTester を起動し, 図 5-23 に示すように, ワークスペース子ウィンドウに以下の項目を入力してください.

Controller Name : 任意の文字列を入力してください.
Provider Name : 今回は CaoProv.DataStore を選択してください.
Machine Name : DCOM サーバのホスト名を入力してください.

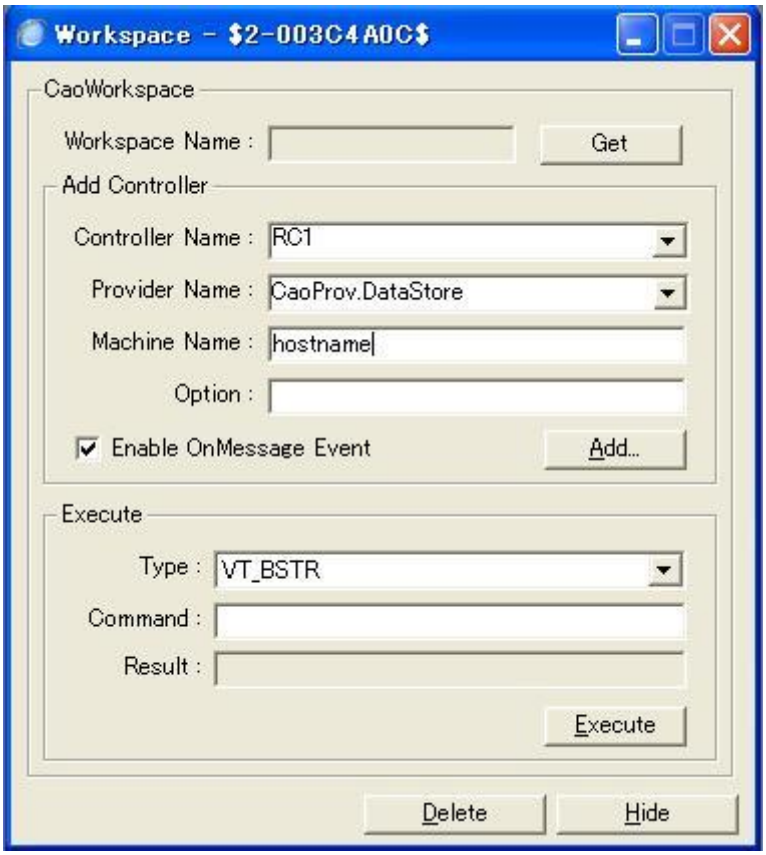



**図 5-23 DCOM によるリモートプロバイダ接続**

ワークスペース子ウィンドウの“Add”ボタンを押下して下さい. リモート接続に成功すると, CaoTester 左下のログ表示ウィンドウに「Succeeded.」(図 5-24 参照)と表示されます. リモート接続に失敗すると,「Failed. ～」(図 5-25 参照)と表示されます.

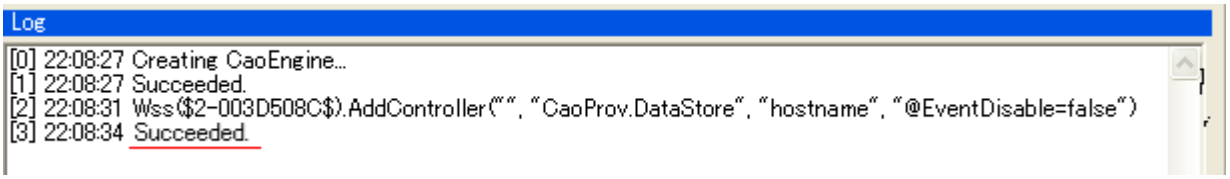



**図 5-24 DCOM 接続結果(成功)**

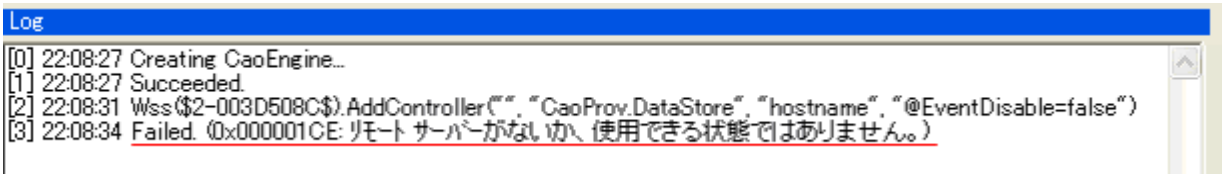



**図 5-25 DCOM 接続結果(失敗)**

## 5.2. Visual Studio へ統合

ORiN アプリケーション開発をより便利にするために，ORiN2 SDK を Visual Studio へ統合することができます．

Visual Studio へ統合するには，<ORiN インストールディレクトリ>¥Tools¥VSAddins¥Bin¥ORiN2AddIns.exe をダブルクリックしてください．

統合したい Visual Studio のバージョンを選択し Install ボタン（図 5-26 参照）を押すと，ORiN2 SDK が Visual Studio へ統合されます．統合を解除する場合は，Uninstall ボタン（図 5-27 参照）を押してください．

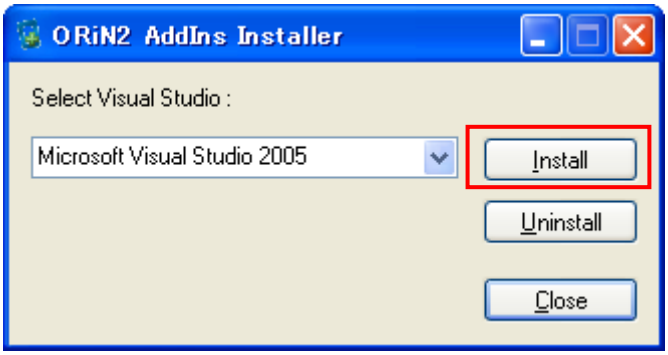



**図 5-26 Visual Studio へ統合**

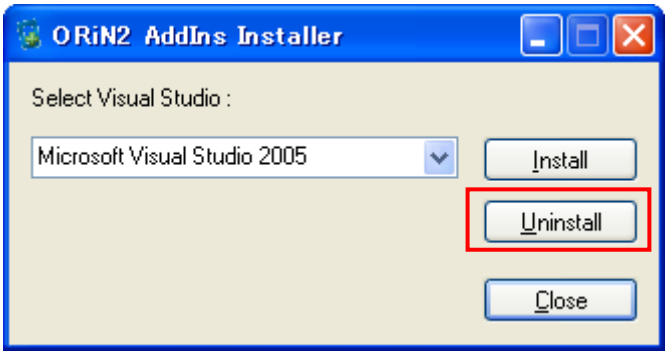



**図 5-27 Visual Studio から統合解除**

### 5.2.1. Visual Studio メニューバー

ORiN2 SDK を Visual Studio へ統合すると，Visual Studio のメニューバーに ORiN2 の項目が追加されます（図 5-28 参照）．ORiN2 メニューには，ORiN2 SDK が提供するツールおよびドキュメントが登録されています．これらをクリックすることで必要なツールやドキュメントを簡単に呼び出すことができます．

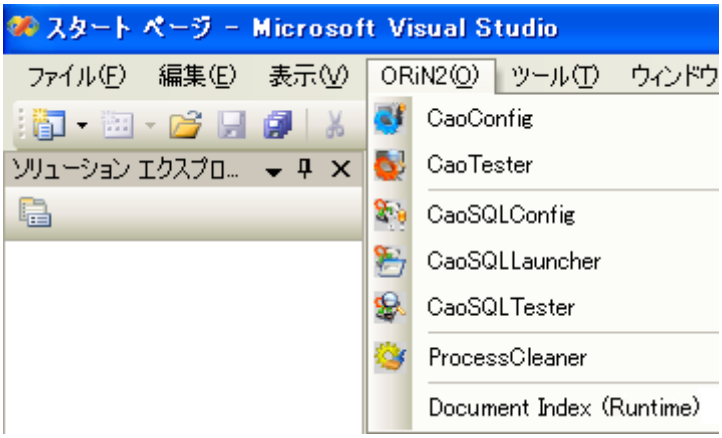



**図 5-28 ORiN2 メニュー一覧**

### 5.2.2. CAO プロバイダウィザード

お買い上げ頂いたパッケージが「ORiN2 SDK プロバイダ開発用」であれば, Visual Studio の新しいプロジェクトに ORiN2 の項目が追加されます(図 5-29 参照). 「ORiN2 CAO Provider Wizard」を選択して OK ボタンを押すと, プロバイダ開発用のプロジェクトが作成されます. プロバイダ開発の詳細は, <ORiN インストールディレクトリ>¥CAO¥Provider¥Doc¥ProviderDevGuide_ja.pdf を参照してください.

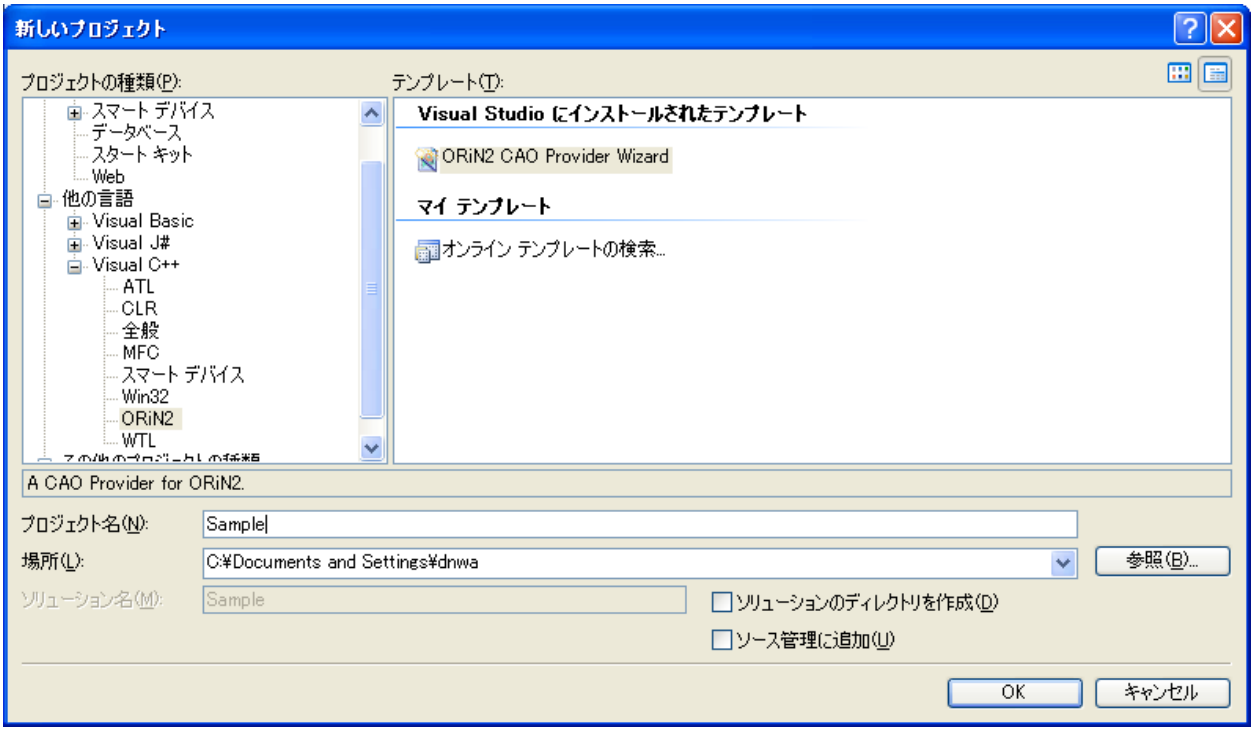



**図 5-29 CAO プロバイダウィザード**

# 6. CaoConfig

CaoConfig は，CAO エンジンおよび CAO プロバイダの動作設定を変更するためのツールです．CAO エンジンの優先度や，ログの出力レベル，CAO プロバイダごとの有効／無効切り替えや，CRD 切り替え機能などを設定することができます．

CaoConfig で設定した項目は，レジストリに反映されます．

## 6.1. 画面構成

ここでは CaoConfig の画面構成について説明します．

### 6.1.1. メニュー

#### 6.1.1.1. File メニュー

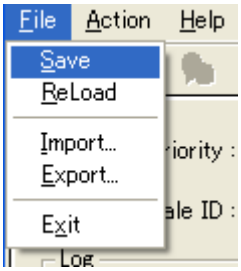



**図 6-1 CaoConfig File メニュー**

**[保存]** – Save

レジストリに CaoConfig の情報を設定します．

**[再読み込み]** – Reload

レジストリから CaoConfig の情報を取得します．

**[インポート]** – Import

XML ファイルから CaoConfig の情報を読み込みます．

**[エクスポート]** – Export

CaoConfig の情報を XML ファイルに保存します．

**[終了]** – Exit

CaoConfig を終了します．

#### 6.1.1.2. Action メニュー


**図 6-2 CaoConfig Action メニュー**

**[デフォルト設定]** – Default Settings

選択中の項目をデフォルトの設定に戻します.

Cao Engine タブを選択していた場合, CaoEngine タブ内の全ての項目をデフォルト設定に戻します.

Cao Provider タブを選択していた場合, Provider List で選択中のプロバイダをデフォルト設定に戻します.

**[サービス開始]** – Service Start

CAO サービスを開始します. CAO をサービスとして登録してある場合のみ, 使用できます.

**[サービス停止]** – Service Stop

CAO サービスを停止します. CAO をサービスとして登録してある場合のみ, 使用できます.

**[サービス再起動]** – Service Restart

CAO サービスを再起動します. CAO をサービスとして登録してある場合のみ, 使用できます.

#### 6.1.1.3. Help メニュー

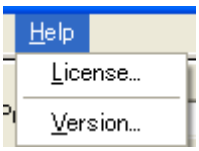



**図 6-3 CaoConfig Help メニュー**

**[ライセンス] –License**

CAO のライセンス管理画面を表示します.

**[バージョン] –Version**

CaoConfig のバージョン情報を表示します.

### 6.1.2. Cao Engine タブ

Cao Engine タブでは，CAO エンジンの動作設定を行うことができます．

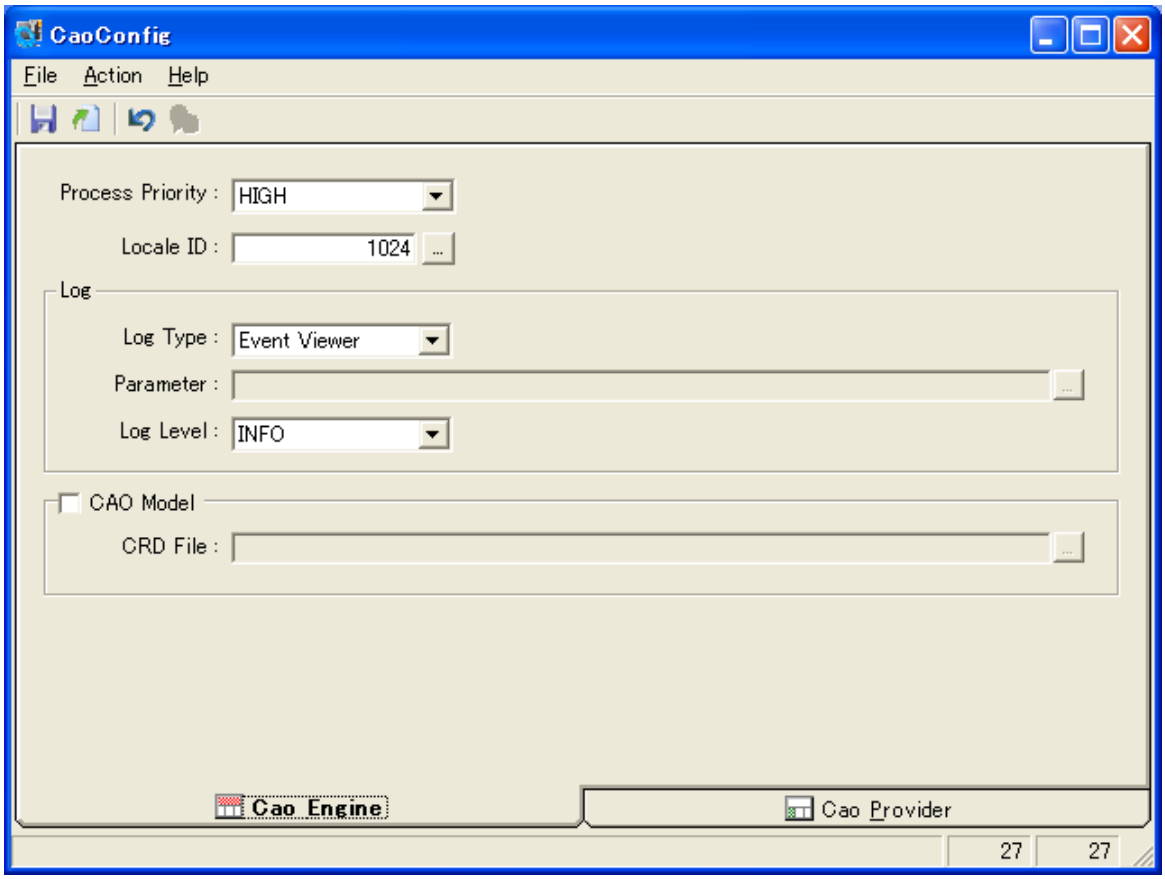



**図 6-4 Cao Engine タブ**

設定できる項目を以下に説明します．

**[プロセス優先度]** – Process Priority

CAO エンジンのプロセス優先度を設定します．優先度に対する調整は以下の通りです．

REAL TIME > HIGH > **NORMAL** > IDEL

**[ローカル ID]** – Local ID

使用する言語 ID を設定します．

**[ログタイプ]** - Log Type

CAO.exe のログの出力タイプを選択します．ログの出力タイプは以下のものを選択することができます．

表 6-1 CaoEngine ログタイプ

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

**[パラメータ]** – Parameter

ログ出力のパラメータを設定します.

設定内容はログタイプによって以下の用に異なります.

表 6-2 CaoEngine パラメータ

| ログタイプ | パラメータ |
|---|---|
| Console | (未使用) |
| Message Box | (未使用) |
| Event Viewer | (未使用) |
| Debug Viewer | (未使用) |
| Text File | 出力先テキストファイルパス<br>例) C:¥CaoLog.txt |
| SysLog | 出力先 SysLog サーバ<br>Conn=UDP:<SysLog サーバの IP アドレス><br>例) Conn=UDP:192.168.0.1 |

**[ログレベル]** – Log Level

ログの出力レベルを設定します. (設定したログレベル以上のログを出力します)ログレベルの設定は, 以下の 5 つのレベルから選択することができます. “FATAL”がもっとも深刻度が高く, “DEBUG”に近づくほど深刻度は低くなります. 標準では“INFO”に設定されています.

FATAL > ERROR > WARN > **INFO** > DEBUG

**[オブジェクトの自動登録]** – CAO Model

エンジンのオブジェクト自動登録機能の有無を設定します.

オブジェクト自動登録機能の詳細については 2.7.2 を参照して下さい.

**[CRD ファイル]** – CRD File

エンジンのオブジェクト自動登録で使用する CRD ファイルのパスを指定します.

### 6.1.3. Cao Provider タブ

Cao Provider タブでは, プロバイダごとの動作設定を行うことができます.

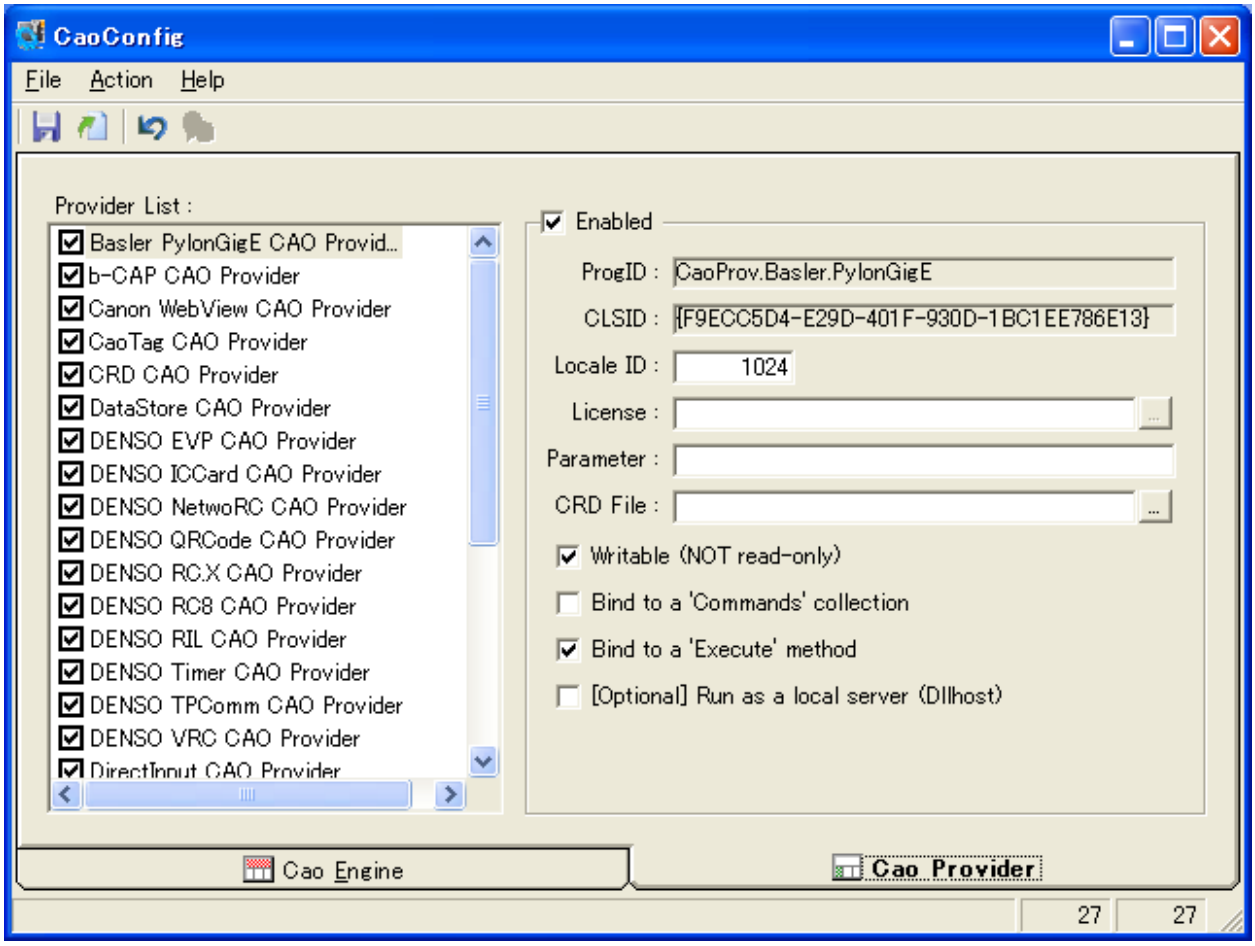



**図 6-5 Cao Provider タブ**

設定できる項目を以下に説明します.

**[プロバイダリスト]** – Provider List

インストールされているプロバイダの一覧を表示します. チェックボックスにより, Enable の切り替えを行うことができます.

**[有効/無効設定]** – Enable

プロバイダの使用可／不可の設定を行います.

**[プログラム ID]** – ProgID

プロバイダのプログラム ID を表示します.

**[クラス ID]** – CLSID

プロバイダのクラス ID を表示します.

**[ローカル ID]** – Locale ID

プロバイダのロケール ID を設定します.

**[ライセンス]** – Lisence

プロバイダのライセンスを設定します.

**[パラメータ]** – Parameter

プロバイダのパラメータを設定します.

**[CRD ファイル]** – CRD File

プロバイダごとの CRD ファイル切り替え機能時に使用する CRD ファイルへのパスを設定します. 詳細については, 2.7.1 を参照して下さい.

**[書込み禁止設定]** – Writable (NOT read-only)

プロバイダの書込み禁止設定を行います.

**[Commands コレクションメソッド動的追加機能]** – Bind to a ‘Commands’ collection

CaoCommands のメソッド動的追加機能の有無を設定します. 詳しくは 2.7.3 を参照して下さい.

**[Execute メソッド動的追加機能]** – Bind to a ‘Execute’ method

CaoController::Execute のメソッド動的追加機能機能の有無を設定します. 詳しくは 2.7.3 を参照して下さい.

**[ローカルサーバ実行]** – Run as a local server

プロバイダを CAO とは別のプロセス（dllhost.exe）で実行します.

# 付録A. 付録

## 付録A.1. CAO エンジン関数一覧

◆ CaoEngine Object ― エンジン

| 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 | | 備考 |
|---|---|---|---|---|---|---|---|
| | | | | | IN | OUT RETVAL | |
| CaoEngine<br><br>(クラス番号:0) | EngineStatus | P | エンジンステータスオブジェクトの取得 | R | | オブジェクト:<br>IcaoEngineStatus | |
| | Workspaces | P | ワークスペースコレクションの取得 | R | | コレクション:<br>ICaoWorkspaces | |
| | AddWorkspace | M | ワークスペースオブジェクトの追加 | S | ワークスペース名:BSTR,<br>[オプション:BSTR] | オブジェクト:<br>ICaoWorkspace | CaoWorkspaces::Add()と同じ機能. |
| | Execute | M | 拡張コマンドの実行 | S | コマンド:BSTR<br>[パラメータ:VARIANT] | 結果:VARIANT | 機能拡張用(予約) |
| 記号の意味 | M:メソッド<br>P:プロパティ<br>E:イベント | | | (注1) | ・[]内は省略可能引数.<br>・BSTR 型の省略可能引数のデフォルト値は NULL.<br>・数値型の省略可能引数のデフォルト値はゼロ.<br>・VARIANT 型の省略可能引数のデフォルト値は<br>VT_ERROR. | | |

(注1)記号の意味は次の通り. この中で, CAO エンジンのアクセス制限機能(書込み制限機能)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.

◆ CaoWorkspace(s) Object ― ワークスペース

| 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 | | 備考 |
|---|---|---|---|---|---|---|---|
| | | | | | IN | OUT RETVAL | |
| CaoWorkspaces (クラス番号：1) | Count | P | コレクション数の取得 | R | | コレクション数：long | |
| | Add | M | ワークスペースオブジェクトの追加 | S | ワークスペース名：BSTR,<br>[オプション：BSTR] | オブジェクト：<br>ICaoWorkspace | ワークスペース名は任意の文字列．ワークスペース名を省略(NULL 指定)した場合は自動的にユニークな名前が付けられます． |
| | Clear | M | 全てのワークスペースオブジェクトの解放 | S | | | |
| | IsMember | M | ワークスペースオブジェクトの登録確認 | R | ワークスペース名/番号：<br>VARIANT | チェック結果：<br>VARIANT_BOOL | |
| | Item | M | ワークスペースオブジェクトの取得 | R | ワークスペース名/番号：<br>VARIANT | オブジェクト：<br>ICaoWorkspace | デフォルトメンバー |
| | Remove | M | ワークスペースオブジェクトの開放 | S | ワークスペース名/番号：<br>VARIANT | | |
| CaoWorkspace (クラス番号：2) | Controllers | P | コントローラコレクションの取得 | R | | コレクション：<br>ICaoControllers | |
| | Index | P | ワークスペース番号の取得 | R | | ワークスペース番号：long | |
| | Name | P | ワークスペース名の取得 | R | | ワークスペース名：BSTR | |
| | ProviderNames | P | プロバイダ名リストの取得 | R | [オプション：BSTR] | プロバイダ名リスト：<br>VARIANT<br>(VT_VARIANT\|VT_ARRAY) | オプションはフィルター条件等． |
| | AddController | M | コントローラオブジェクトの追加 | S | コントローラ名：BSTR,<br>プロバイダ名：BSTR,<br>[マシン名：BSTR,<br>[オプション：BSTR]] | オブジェクト：ICaoController | CaoControllers::Add()と同じ機能． |
| | Execute | M | 拡張コマンドの実行 | S | コマンド：BSTR<br>[パラメータ：VARIANT] | 結果：VARIANT | 機能拡張用(予約) |
| 記号の意味 | M：メソッド<br>P：プロパティ<br>E：イベント | | | | ・[ ]内は省略可能引数．<br>・BSTR 型の省略可能引数のデフォルト値は NULL．<br>・数値型の省略可能引数のデフォルト値はゼロ． | | |

| 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 | | 備考 |
|---|---|---|---|---|---|---|
| | | | | IN | OUT RETVAL | |
| | | | | ・VARIANT 型の省略可能引数のデフォルト値は VT_ERROR. | | |

◆ CaoController(s) 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 | | 備考 |
|---|---|---|---|---|---|---|---|
| | | | | | IN | OUT RETVAL | |
| CaoControllers (クラス番号:3) | Count | P | コレクション数の取得 | R | | コレクション数:long | |
| | Add | M | コントローラオブジェクトの追加 | S | コントローラ名:BSTR,<br>プロバイダ名:BSTR,<br>[マシン名:BSTR],<br>[オプション:BSTR] | オブジェクト:<br>ICaoController | コントローラ名は任意の文字列.<br>オプションはタイムアウト値, イベント許可など.<br>イベント許可: @EventEnable[=True/False]<br>デフォルト値は"True" |
| | Clear | M | 全てのコントローラオブジェクトの解放 | S | | | |
| | IsMember | M | コントローラオブジェクトの登録確認 | R | コントローラ名/番号:<br>VARIANT | チェック結果:<br>VARIANT_BOOL | |
| | Item | M | コントローラオブジェクトの取得 | R | コントローラ名/番号:<br>VARIANT | オブジェクト:ICaoController | デフォルトメンバー |
| | Remove | M | コントローラオブジェクトの開放 | S | コントローラ名/番号:<br>VARIANT | | |
| CaoController (クラス番号:4) | Attribute | P | 属性の取得 | R | | 属性:long | |
| | CommandNames | P | 拡張ボード名リストの取得 | R | [オプション:BSTR] | コマンド名リスト:VARIANT<br>(VT_VARIANT\|VT_ARRAY) | オプションはフィルター条件等. |
| | Commands | P | コマンドコレクションの取得 | R | | コレクション:<br>ICaoCommands | |
| | ExtensionNames | P | コマンド名リストの取得 | R | [オプション:BSTR] | 拡張ボード名リスト:<br>VARIANT<br>(VT_VARIANT\|VT_ARRAY) | オプションはフィルター条件等. |
| | Extensions | P | 拡張ボードコレクションの取得 | R | | コレクション:<br>ICaoExtensions | |
| | 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:<br>VARIANT<br>(VT_VARIANT\|VT_ARRAY) | オプションはフィルター条件等. |
| | Files | P | ファイルコレクションの取得 | R | | コレクション:ICaoFiles | ルートディレクトリのファイルコレクション. |
| | Help | P | ヘルプ | R | | ヘルプ文字列:BSTR | |

| 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 | | 備考 |
|---|---|---|---|---|---|---|---|
| | | | | | IN | OUT RETVAL | |
| | ID | P | ID | R/W | ID：VARIANT | ID：VARIANT | |
| | Index | P | コントローラ番号の取得 | R | | コントローラ番号：long | |
| | Name | P | コントローラ名の取得 | R | | コントローラ名：BSTR | |
| | 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：<br>VARIANT<br>(VT_VARIANT\|VT_ARRAY) | オプションはフィルター条件等. |
| | Robots | P | ロボットコレクションの取得 | R | | コレクション：ICaoRobots | |
| | Tag | P | タグ | R/S | タグデータ：VARIANT | タグデータ：VARIANT | |
| | 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：<br>VARIANT<br>(VT_VARIANT\|VT_ARRAY) | オプションはフィルター条件等. |
| | Tasks | P | タスクコレクションの取得 | R | | コレクション：ICaoTasks | |
| | VariableNames | P | 変数名リストの取得 | R | [オプション：BSTR] | 変数名リスト：<br>VARIANT<br>(VT_VARIANT\|VT_ARRAY) | オプションはフィルター条件等. |
| | Variables | P | 変数コレクションの取得 | R | | コレクション：ICaoVariables | |
| | AddCommand | M | コマンドオブジェクトの追加 | S | コマンド名：BSTR,<br>[オプション：BSTR] | オブジェクト：ICaoCommand | CaoCommands::Add()と同じ機能. |
| | AddExtension | M | 拡張ボードオブジェクトの追加 | S | 拡張ボード名：BSTR,<br>[オプション：BSTR] | オブジェクト：ICaoExtension | CaoExtensions::Add()と同じ機能. |
| | AddFile | M | ファイルオブジェクトの追加 | S | ファイル名：BSTR,<br>[オプション：BSTR] | オブジェクト：ICaoFile | CaoFiles::Add()と同じ機能. |
| | AddRobot | M | ロボットオブジェクトの追加 | S | ロボット名：BSTR,<br>[オプション：BSTR] | オブジェクト：ICaoRobot | CaoRobots::Add()と同じ機能. |
| | AddTask | M | タスクオブジェクトの追加 | S | タスク名：BSTR,<br>[オプション：BSTR] | オブジェクト：ICaoTask | CaoTasks::Add()と同じ機能. |
| | AddVariable | M | 変数オブジェクトの追加 | S | 変数名：BSTR,<br>[オプション：BSTR] | オブジェクト：ICaoVariable | CaoVariables::Add()と同じ機能. |
| | Execute | M | 拡張コマンドの実行 | S | コマンド：BSTR<br>[パラメータ：VARIANT] | 結果：VARIANT | 機能拡張用 |
| | GetMessage | M | メッセージの取得 | R | | メッセージ：ICaoMessage | エンジン内のメッセージキューからメッセージ |

| 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 | | 備考 |
|---|---|---|---|---|---|---|
| | | | | IN | OUT RETVAL | |
| | | | | | | を取得．OnMessage イベント機能が無効の場合にのみ取得できる． |
| | OnMessage E | メッセージ受信イベント | R | メッセージ：ICaoMessage | | ・プロバイダ(コントローラ)→エンジン→クライアントの呼び出しを実現する．<br>・メッセージ番号の負の範囲は ORiN で予約されている．<br>・イベントの無効化は AddController()のオプションで指定する． |
| 記号の意味 | M：メソッド<br>P：プロパティ<br>E：イベント | | | ・[]内は省略可能引数．<br>・BSTR 型の省略可能引数のデフォルト値は NULL．<br>・数値型の省略可能引数のデフォルト値はゼロ．<br>・VARIANT 型の省略可能引数のデフォルト値は VT_ERROR． | | |

◆ CaoExtension(s) 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 | | 備考 |
|---|---|---|---|---|---|---|---|
| | | | | | IN | OUT RETVAL | |
| CaoExtensions (クラス番号:5) | Count | P | コレクション数の取得 | R | | コレクション数:long | |
| | Add | M | 拡張ボードオブジェクトの追加 | S | 拡張ボード名:BSTR, [オプション:BSTR] | オブジェクト:ICaoExtension | |
| | Clear | M | 全ての拡張ボードオブジェクトの解放 | S | | | |
| | IsMember | M | 拡張ボードオブジェクトの登録確認 | R | 拡張ボード名/番号: VARIANT | チェック結果: VARIANT_BOOL | |
| | Item | M | 拡張ボードオブジェクトの取得 | R | 拡張ボード名/番号: VARIANT | オブジェクト:ICaoExtension | デフォルトメンバー |
| | Remove | M | 拡張ボードオブジェクトの開放 | S | 拡張ボード名/番号: VARIANT | | |
| CaoExtension (クラス番号:6) | Attribute | P | 属性の取得 | R | | 属性:long | |
| | Help | P | ヘルプ | R | | ヘルプ文字列:BSTR | |
| | ID | P | ID | R/W | ID:VARIANT | ID:VARIANT | |
| | Index | P | 拡張ボード番号の取得 | R | | 拡張ボード番号:long | |
| | Name | P | 拡張ボード名の取得 | R | | 拡張ボード名:BSTR | |
| | Tag | P | タグ | R/S | タグデータ:VARIANT | タグデータ:VARIANT | |
| | 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) | オプションはフィルター条件等. |
| | Variables | P | 変数コレクションの取得 | R | | コレクション:ICaoVariables | |
| | AddVariable | M | 変数オブジェクトの追加 | S | 変数名:BSTR, [オプション:BSTR] | オブジェクト:ICaoVariable | CaoVariables::Add()と同じ機能. |
| | 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 | 機能拡張用 |
| 記号の意味 | M:メソッド<br>P:プロパティ<br>E:イベント | | | | ・[]内は省略可能引数.<br>・BSTR 型の省略可能引数のデフォルト値は NULL.<br>・数値型の省略可能引数のデフォルト値はゼロ.<br>・VARIANT 型の省略可能引数のデフォルト値は | | |

| 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 | | 備考 |
|---|---|---|---|---|---|---|
| | | | | IN | OUT RETVAL | |
| | | | | VT_ERROR. | | |

◆ CaoFile(s) 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 | | 備考 |
|---|---|---|---|---|---|---|---|
| | | | | | IN | OUT RETVAL | |
| CaoFiles (クラス番号:7) | Count | P | コレクション数の取得 | R | | コレクション数:long | |
| | Add | M | ファイルオブジェクトの追加 | S | ファイル名:BSTR, [オプション:BSTR] | オブジェクト:ICaoFile | ファイル作成: @Create=[数値]デフォルト値は 0<br>0: 作成しない<br>0 以外:作成する<br>プロバイダが書込み禁止のとき, @Create オプションが 0 以外のときは書き込みエラーになる. |
| | Clear | M | 全てのファイルオブジェクトの解放 | S | | | |
| | IsMember | M | ファイルオブジェクトの登録確認 | R | ファイル名/番号:VARIANT | チェック結果: VARIANT_BOOL | |
| | Item | M | ファイルオブジェクトの取得 | R | ファイル名/番号:VARIANT | オブジェクト:ICaoFile | デフォルトメンバー |
| | Remove | M | ファイルオブジェクトの開放 | S | ファイル名/番号:VARIANT | | |
| CaoFile (クラス番号:8) | Attribute | P | 属性の取得 | R | | 属性:long | |
| | DateCreated | P | 作成日時 | R | | 作成日時:VARIANT | |
| | DateLastAccessed | P | 最終アクセス日時 | R | | 最終アクセス日時: VARIANT | |
| | DateLastModified | P | 最終変更日時 | R | | 最終変更日時:VARIANT | |
| | 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) | 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. |
| | Files | P | ファイルコレクションの取得 | R | | コレクション:ICaoFiles | |
| | Help | P | ヘルプ | R | | ヘルプ文字列:BSTR | |
| | ID | P | ID | R/W | ID:VARIANT | ID:VARIANT | |
| | Index | P | ファイル番号の取得 | R | | ファイル番号:long | |
| | Name | P | ファイル名の取得 | R | | ファイル名:BSTR | |

| 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 | | 備考 |
|---|---|---|---|---|---|---|---|
| | | | | | IN | OUT RETVAL | |
| | Path | P | パスの取得 | R | | パス名:BSTR | 絶対パス. ファイル名を含めない. 最後の区切り文字は含める. |
| | Size | P | ファイルサイズの取得 | R | | ファイルサイズ:long | |
| | Tag | P | タグ | R/S | タグデータ:VARIANT | タグデータ:VARIANT | |
| | Type | P | ファイルのタイプの取得 | R | | ファイルタイプ:BSTR | |
| | Value | P | ファイルの内容 | R/W | データ:VARIANT | データ:VARIANT | デフォルトメンバー |
| | VariableNames | P | 変数名リストの取得 | R | [オプション:BSTR] | 変数名リスト:<br>VARIANT<br>(VT_VARIANT\|VT_ARRAY) | オプションはフィルター条件等. |
| | Variables | P | 変数コレクションの取得 | R | | コレクション:ICaoVariables | |
| | AddFile | M | ファイルオブジェクトの追加 | S | ファイル名:BSTR,<br>[オプション:BSTR] | オブジェクト:ICaoFile | CaoFiles::Add()と同じ機能. |
| | AddVariable | M | 変数オブジェクトの追加 | S | 変数名:BSTR,<br>[オプション:BSTR] | オブジェクト:ICaoVariable | CaoVariables::Add()と同じ機能. |
| | Copy | M | 複写 | W | 複写先ファイル名:BSTR<br>[オプション:BSTR] | | |
| | Delete | M | 削除 | W | [オプション:BSTR] | | |
| | Execute | M | 拡張コマンドの実行 | S | コマンド:BSTR<br>[パラメータ:VARIANT] | 結果:VARIANT | 機能拡張用 |
| | Move | M | 移動 | W | 移動先ファイル名:BSTR<br>[オプション:BSTR] | | |
| | Run | M | タスクの生成 | W | [オプション:BSTR] | タスク名:BSTR | |
| 記号の意味 | M:メソッド<br>P:プロパティ<br>E:イベント | | | | ・[]内は省略可能引数.<br>・BSTR 型の省略可能引数のデフォルト値は NULL.<br>・数値型の省略可能引数のデフォルト値はゼロ.<br>・VARIANT 型の省略可能引数のデフォルト値は VT_ERROR. | | |

◆ CaoRobot(s) 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 | | 備考 |
|---|---|---|---|---|---|---|---|
| | | | | | IN | OUT RETVAL | |
| CaoRobots (クラス番号：9) | Count | P | コレクション数の取得 | R | | コレクション数：long | |
| | Add | M | ロボットオブジェクトの追加 | S | ロボット名：BSTR,<br>[オプション：BSTR] | オブジェクト：ICaoRobot | |
| | Clear | M | 全てのロボットオブジェクトの解放 | S | | | |
| | IsMember | M | ロボットオブジェクトの登録確認 | R | ロボット名/番号：VARIANT | チェック結果：<br>VARIANT_BOOL | |
| | Item | M | ロボットオブジェクトの取得 | R | ロボット名/番号：VARIANT | オブジェクト：ICaoRobot | デフォルトメンバー |
| | Remove | M | ロボットオブジェクトの開放 | S | ロボット名/番号：VARIANT | | |
| CaoRobot (クラス番号：10) | Attribute | P | 属性の取得 | R | | 属性：long | |
| | Help | P | ヘルプ | R | | ヘルプ文字列：BSTR | |
| | ID | P | ID | R/W | ID：VARIANT | ID：VARIANT | |
| | Index | P | ロボット番号の取得 | R | | ロボット番号：long | |
| | Name | P | CaoRobot 名の取得 | R | | ロボット名：BSTR | |
| | Tag | P | タグ | R/S | タグデータ：VARIANT | タグデータ：VARIANT | |
| | VariableNames | P | 変数名リストの取得 | R | [オプション：BSTR] | 変数名リスト：<br>VARIANT<br>(VT_VARIANT\|VT_ARRAY) | オプションはフィルター条件等. |
| | Variables | P | 変数コレクションの取得 | R | | コレクション：ICaoVariables | |
| | Accelerate | M | SLIM の ACCEL 文の仕様参照 | W | 軸番号：long,<br>加速度：float,<br>[減速度：float] | | 軸番号は，-1：手先加(減)速度，0：全軸加(減)速度，その他は指定軸の加(減)速度. |
| | AddVariable | M | 変数オブジェクトの追加 | S | 変数名：BSTR,<br>[オプション：BSTR] | オブジェクト：ICaoVariable | CaoVariables::Add()と同じ機能. |
| | Change | M | SLIM の CHANGE 文の仕様参照 | W | ハンド名：BSTR | | |
| | Chuck | M | SLIM の GRASP 文の仕様参照 | W | [オプション：BSTR] | | |

| 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 | | 備考 |
|---|---|---|---|---|---|---|---|
| | | | | | IN | OUT RETVAL | |
| | Drive | M | SLIM の DRIVE 文の仕様参照 | W | 軸番号：long,<br>移動量：float,<br>[動作オプション：BSTR] | | SLIM のように同時に複数の軸を動かすことはできない．その場合は，MOVE を使うことを推奨． |
| | Execute | M | 拡張コマンドの実行 | S | コマンド：BSTR<br>[パラメータ：VARIANT] | 結果：VARIANT | 機能拡張用 |
| | GoHome | M | SLIM の GOHOME 文の仕様参照 | W | | | |
| | Hold | M | SLIM の HOLD 文の仕様参照 | W | [オプション：BSTR] | | 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] | | SLIM ではプログラムの強制停止の意味だか，CAO ではロボット動作の強制停止を意味している． |
| | Move | M | SLIM の MOVE 文の仕様参照 | W | 補間指定：long,<br>ポーズ列：VARIANT,<br>[動作オプション：BSTR] | | ・補間指定は，1：PTP，2：直線，3：円弧．<br>・ポーズの指定は各メーカ依存．円弧補間の場合などは，配列になる．<br>・動作オプションはのデフォルト値は""． |
| | Rotate | M | SLIM の ROTATE 文の仕様参照 | W | 回転面：VARIANT,<br>角度：float,<br>回転中心：VARIANT,<br>[動作オプション：BSTR] | | 回転面の指定は各メーカ依存． |
| | Speed | M | SLIM の SPEED/JSPEED 文の仕様参照 | W | 軸番号：long,<br>速度：float | | 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] | | 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] | | SLIM では HOLD 文はプログラムの一時停止の意味であるため，再開のコマンドが規定されていないが，CAO ではロボット動作の一時停止を意味しているので，その再開に使う． |
| 記号の意味 | M：メソッド<br>P：プロパティ | | | | ・[]内は省略可能引数．<br>・BSTR 型の省略可能引数のデフォルト値は NULL． | | |

| 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 | | 備考 |
|---|---|---|---|---|---|---|
| | | | | IN | OUT RETVAL | |
| | E:イベント | | | ・数値型の省略可能引数のデフォルト値はゼロ.<br>・VARIANT 型の省略可能引数のデフォルト値は VT_ERROR. | | |

◆ CaoTask(s) 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 | | 備考 |
|---|---|---|---|---|---|---|---|
| | | | | | IN | OUT RETVAL | |
| CaoTasks (クラス番号：11) | Count | P | コレクション数の取得 | R | | コレクション数：long | |
| | Add | M | タスクオブジェクトの追加 | S | タスク名：BSTR,<br>[オプション：BSTR] | オブジェクト：ICaoTask | |
| | Clear | M | 全てのタスクオブジェクトの解放 | S | | | |
| | IsMember | M | タスクオブジェクトの登録確認 | R | タスク名/番号：VARIANT | チェック結果：<br>VARIANT_BOOL | |
| | Item | M | タスクオブジェクトの取得 | R | タスク名/番号：VARIANT | オブジェクト：ICaoTask | デフォルトメンバー |
| | Remove | M | タスクオブジェクトの開放 | S | タスク名/番号：VARIANT | | |
| CaoTask (クラス番号：12) | Attribute | P | 属性の取得 | R | | 属性：long | |
| | FileName | P | 対応ファイル名の取得 | R | | ファイル名：BSTR | |
| | Help | P | ヘルプ | R | | ヘルプ文字列：BSTR | |
| | ID | P | ID | R/W | ID：VARIANT | ID：VARIANT | |
| | Index | P | タスク番号の取得 | R | | タスク番号：long | |
| | Name | P | タスク名の取得 | R | | タスク名：BSTR | |
| | Tag | P | タグ | R/S | タグデータ：VARIANT | タグデータ：VARIANT | |
| | VariableNames | P | 変数名リストの取得 | R | [オプション：BSTR] | 変数名リスト：<br>VARIANT<br>(VT_VARIANT\|VT_ARRAY) | オプションはフィルター条件等. |
| | Variables | P | 変数コレクションの取得 | R | | コレクション：ICaoVariables | |
| | AddVariable | M | 変数オブジェクトの追加 | S | 変数名：BSTR,<br>[オプション：BSTR] | オブジェクト：ICaoVariable | CaoVariables::Add()と同じ機能. |
| | Delete | M | タスクの削除 | W | [オプション：BSTR] | | |
| | Execute | M | 拡張コマンドの実行 | S | コマンド：BSTR<br>[パラメータ：VARIANT] | 結果：VARIANT | 機能拡張用 |
| | Start | M | タスクの開始 | W | モード：long,<br>[オプション：BSTR] | | モードは1：1サイクル実行，2：連続実行，<br>3：1ステップ 送り，4：1ステップ 戻し<br>オプションは開始位置など. |

| 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 | | 備考 |
|---|---|---|---|---|---|---|
| | | | | IN | OUT RETVAL | |
| | Stop M | タスクの停止 | W | モード：long,<br>[オプション：BSTR] | | モードは0：デフォルト停止，1：瞬時停止，2：ステップ停止，3：サイクル停止．4：初期化停止<br>(注)デフォルト停止とは，実際には何れかの停止方法． |
| 記号の意味 | M：メソッド<br>P：プロパティ<br>E：イベント | | | ・[]内は省略可能引数．<br>・BSTR 型の省略可能引数のデフォルト値は NULL．<br>・数値型の省略可能引数のデフォルト値はゼロ．<br>・VARIANT 型の省略可能引数のデフォルト値は VT_ERROR． | | |

◆ CaoVariable(s) 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 | | 備考 |
|---|---|---|---|---|---|---|---|
| | | | | | IN | OUT RETVAL | |
| CaoVariables (クラス番号：13) | Count | P | コレクション数の取得 | R | | コレクション数：long | |
| | Add | M | 変数オブジェクトの追加 | S | 変数名：BSTR,<br>[オプション：BSTR] | オブジェクト：ICaoVariable | |
| | Clear | M | 全ての変数オブジェクトの解放 | S | | | |
| | IsMember | M | 変数オブジェクトの登録確認 | R | 変数名/番号：VARIANT | チェック結果：<br>VARIANT_BOOL | |
| | Item | M | 変数オブジェクトの取得 | R | 変数名/番号：VARIANT | オブジェクト：ICaoVariable | デフォルトメンバー |
| | Remove | M | 変数オブジェクトの開放 | S | 変数名/番号：VARIANT | | |
| CaoVariable (クラス番号：14) | Attribute | P | 属性の取得 | R | | 属性：long | |
| | DateTime | P | タイムスタンプ(日時)の取得 | R | | タイムスタンプ：VARIANT | |
| | Help | P | ヘルプ | R | | ヘルプ文字列：BSTR | |
| | ID | P | ID | R/W | ID：VARIANT | ID：VARIANT | |
| | Index | P | 変数番号の取得 | R | | 変数番号：Long | |
| | Microsecond | P | タイムスタンプ(マイクロ秒)の取得 | R | | タイムスタンプ：Long | |
| | Name | P | 変数名の取得 | R | | 変数名：BSTR | システム変数名は@で始まる. |
| | Tag | P | タグ | R/S | タグデータ：VARIANT | タグデータ：VARIANT | |
| | Value | P | 値の取得 | R/W | 値：VARIANT | 値：VARIANT | デフォルトメンバー |
| 記号の意味 | M：メソッド<br>P：プロパティ<br>E：イベント | | | | ・[]内は省略可能引数.<br>・BSTR 型の省略可能引数のデフォルト値は NULL.<br>・数値型の省略可能引数のデフォルト値はゼロ. | | |

◆ CaoCommand(s) 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 | | 備考 |
|---|---|---|---|---|---|---|---|
| | | | | | IN | OUT RETVAL | |
| CaoCommands (クラス番号：15) | Count | P | コレクション数の取得 | R | | コレクション数：long | |
| | Add | M | コマンドオブジェクトの追加 | S | コマンド名：BSTR,<br>[オプション：BSTR] | オブジェクト：ICaoCommand | |
| | Clear | M | 全てのコマンドオブジェクトの解放 | S | | | |
| | IsMember | M | コマンドオブジェクトの登録確認 | R | コマンド名/番号：VARIANT | チェック結果：<br>VARIANT_BOOL | |
| | Item | M | コマンドオブジェクトの取得 | R | コマンド名/番号：VARIANT | オブジェクト：ICaoCommand | デフォルトメンバー |
| | Remove | M | コマンドオブジェクトの開放 | S | コマンド名/番号：VARIANT | | |
| | ExecuteComplete | E | コマンド実行完了イベント | | | コマンド番号：long | 結果は Result プロパティで取得する. |
| CaoCommand (クラス番号：16) | Attribute | P | 属性の取得 | R | | 属性：long | |
| | Help | P | ヘルプ | R | | ヘルプ文字列：BSTR | |
| | ID | P | ID | R/W | ID：VARIANT | ID：VARIANT | |
| | Index | P | コマンド番号の取得 | R | | コマンド番号：long | |
| | Name | P | コマンド名の取得 | R | | コマンド名：BSTR | |
| | Parameters | P | コマンドパラメータ | R/W | コマンドパラメータ：<br>VARIANT | コマンドパラメータ：<br>VARIANT | |
| | Result | P | 直前の Execute()の実行結果 | R | | 実行結果：VARIANT | |
| | State | P | 状態の取得 | R | | 状態：long | 状態の1ビット目は，0：待機中，1：実行中.<br>その他のビットはプロバイダ依存. |
| | Tag | P | タグ | R/S | タグデータ：VARIANT | タグデータ：VARIANT | |
| | Timeout | P | タイムアウト | R/W | タイムアウト：long | タイムアウト：long | |
| | Cancel | M | 実行中コマンドのキャンセル | | | | |
| | Execute | M | コマンドの実行 | | モード：long | | モードは 0：同期実行，1：非同期実行. 非同期実行のときは S_FALSE を返す. |
| 記号の意味 | M：メソッド<br>P：プロパティ<br>E：イベント | | | | ・[]内は省略可能引数.<br>・BSTR 型の省略可能引数のデフォルト値は NULL.<br>・数値型の省略可能引数のデフォルト値はゼロ. | | |

◆ CaoMessage 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 | | 備考 |
|---|---|---|---|---|---|---|---|
| | | | | | IN | OUT RETVAL | |
| CaoMessage<br>(クラス番号：17) | DateTime | P | 作成日時 | R | | 作成日時：VARIANT | |
| | Description | P | 説明 | R | | 説明：BSTR | |
| | Destination | P | 送り先 | R | | 送り先：BSTR | |
| | Number | P | メッセージ番号 | R | | メッセージ番号：long | |
| | SerialNumber | P | メッセージ連番 | R | | メッセージ連番：long | エンジンで自動的に付加される0～LONG_MAX までの連番．LONG_MAX に達したら0にクリアされる． |
| | Source | P | 送り元 | R | | 送り元：BSTR | |
| | Value | P | メッセージ本文 | R | | メッセージ本文：VARIANT | |
| | Clear | M | メッセージのクリア | W | | | |
| | Reply | M | メッセージの返信 | W | | | |
| 記号の意味 | M：メソッド<br>P：プロパティ<br>E：イベント | | | （注1） | ・[]内は省略可能引数．<br>・BSTR 型の省略可能引数のデフォルト値は NULL．<br>・数値型の省略可能引数のデフォルト値はゼロ． | | |

◆ CaoEngineStatus Object ― エンジンステータス

| 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 | | 備考 |
|---|---|---|---|---|---|---|---|
| | | | | | IN | OUT RETVAL | |
| CaoEngineStatus<br>(クラス番号：18) | CurrentDateTime | P | 現在の日時 | R | | 現在日時：VARIANT | |
| | ComputerName | P | コンピュータ名 | R | | コンピュータ名：BSTR | |
| | ObjectCounts | P | 各クラスごとのオブジェクト数 | R | クラス名／番号：<br>VARIANT | オブジェクト数：long | |
| | StartDateTime | P | 開始日時 | R | | 開始日時：VARIANT | |
| | Values | P | (将来用の予約) | R | ステータス名／番号：<br>VARIANT | データ：VARIANT | |
| | Version | P | エンジンのバージョンの取得 | R | | バージョン：BSTR | <Major Ver.>.<Minor Ver.>.<Revision> |
| 記号の意味 | M：メソッド<br>P：プロパティ<br>E：イベント | | | (注1) | ・[]内は省略可能引数.<br>・BSTR 型の省略可能引数のデフォルト値は NULL.<br>・数値型の省略可能引数のデフォルト値はゼロ. | | |

## 付録A.2. 用語集

■ CAO (Controller Access Object)

CAO とは，クライアントアプリケーションおよびロボットコントローラに対して，共通のインタフェースと機能を提供する「標準プログラムインタフェース」です．

■ CAO インタフェース (CAO Interface) (API)

クライアントアプリケーションに対して CAO が提供するロボットコントローラを操作するためのインタフェースです．“CAO API”とは，このインタフェースを指します．

■ CAO エンジン (CAO Engine)

CAO エンジンとは，CAO のインタフェースを実装したミドルウェア(EXE)で，クライアントに対して CAO プロバイダに対する共通の機能と CAO インタフェースを提供します．CAO プロバイダに対する共通の機能として，コレクション機能，メッセージ出力，CRD 切り替え機能等があります．

■ CAO プロバイダ (CAO Provider)

CAO プロバイダとは，CAO の下位モジュールで，ロボットメーカ各社に依存した部分を実装した DLL です．

■ CAO プロバイダテンプレート (CAO Provider template)

CAO Provider の実装をサポートする C++テンプレートです．このテンプレートは ProviderWizard で自動生成されます．

■ CAP(Controller Access Protocol)

インターネット経由で CAO プロバイダにアクセスするための「インターネット向け通信プロトコル」です．CAP では，SOAP を使用した CAO プロバイダにアクセスするためのメッセージを定義しています．

■ CAP プロバイダ(CAP Provider)

CAP メッセージの生成，送受信をおこなうための CAO プロバイダです．

■ CAP リスナ(CAP Listener)

CAP メッセージを受信して，リモートマシン上で CAO を動作させるためのサーバプログラムです．

■ CAP メッセージ(CAP Message)
CAP により定義される SOAP メッセージです.

■ CRD (Controller Resource Definition)
メーカや機種によって異なるロボットのリソース情報を表現するための汎用のデータフォーマットを定義するデータスキーマです.

■ CRD リソース (CRD Resource)
CRD によって表現されるロボットのリソースです.

■ CRD データ (CRD Data)
CRD に従って記述されたロボットのリソースデータで, XML によって記述されます.

■ CRD データスキーマ (CRD Data Schema)
CRD データのフォーマットを定義するスキーマです.

■ CRD ファイル (CRD File)
CRD データの記述された XML ファイルです.

■ CRD プロバイダ (CRD Provider)
CRD データにアクセスするためのプロバイダです.

■ RAC
クライアントアプリケーションとロボット間で, コマンドの送受信をおこなうための規格です.