川崎重工業
KRCC プロバイダ
ユーザーズ ガイド

Version 1.2.0

March 1, 2023

備考:

【改版履歴】

バージョン	日付	内容
1.0.0	2021-11-24	初版.
1.1.0	2022-08-10	CaoValiable Signal 型, POSE 型の設定機能有効に伴う変更記述 Sample プログラム変更に伴う内容記載の改定
1.2.0	2023-03-01	エラー履歴の取得機能を実装しました. 操作履歴の取得機能を実装しました. 誤記を修正しました.

【対応機種】

機種	バージョン	注意事項
川崎重工業 F シリーズコントローラ		
川崎重工業 E シリーズコントローラ		

【動作確認機種】

機種	バージョン	注意事項

目次

1. はじめに

本書は，川崎重工業社のロボット通信ライブラリ KRCC を用いてロボットコントローラに対して TCP で接続し，ロボットの操作や各種パラメータを取得するプロバイダのユーザーズガイドです．図 1-1 が本プロバイダとデバイスの全体構成図になります．以降本プロバイダを KRCC プロバイダと呼称します．

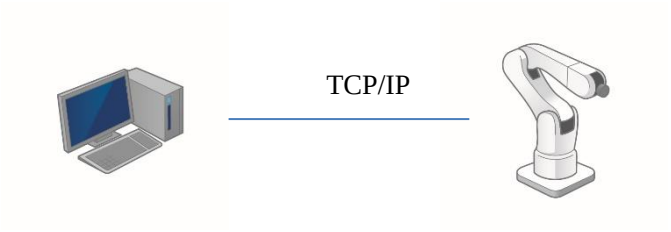

図 1-1 構成図

また，本プロバイダ及びデバイスそれぞれの対応を図 1-2 に表します．
(※一例です．全てを表しているわけではありません．)

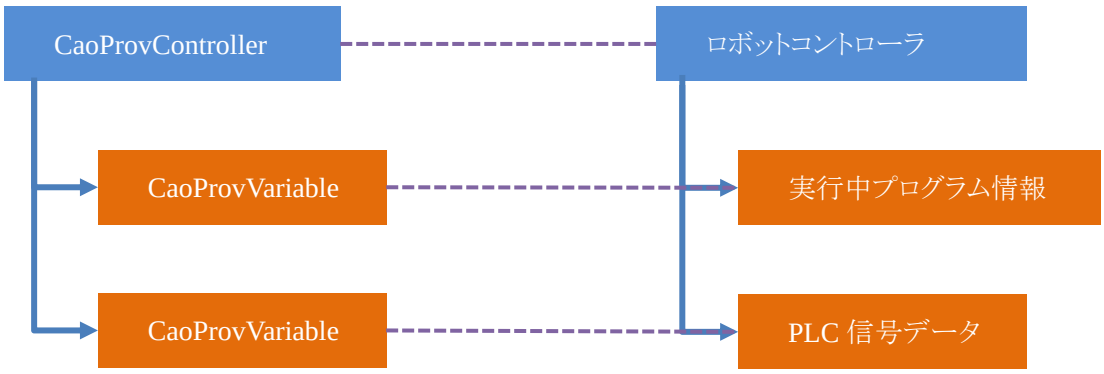

図 1-2 プロバイダの構成とデバイス情報との対応図

2. アプリケーション開発のための環境セットアップ

2.1. ロボットコントローラとクライアント PC との接続

ロボットコントローラとクライアント PC は TCP/IP プロトコルを用いて接続を行います.

通信には川崎重工業製の通信ライブラリ KRCC を使用します.

2.2. PC 開発環境のセットアップ

2.2.1. KRCC プロバイダの手動インストール

KRCC プロバイダを手動でインストールする場合は下記レジストリ登録を行う必要があります. レジストリ登録を行う場合は, 管理者権限でコマンドプロンプトを起動し, regsvr32 コマンドを実行してください. 実行する際には, ファイルのあるパスまで移動するか, ファイルパスを指定して実行してください.

表 2-1 KRCC プロバイダ

ファイル名	CaoProvKawasakiKRCC.dll
ProgID	CaoProv.Kawasaki.KRCC
レジストリ登録	regsvr32 CaoProvKawasakiKRCC.dll
レジストリ登録の抹消	regsvr32 /u CaoProvKawasakiKRCC.dll

3. コマンドリファレンス

3.1. メソッド/プロパティー覧

表 3-1 メソッド/プロパティー覧

カテゴリ	メソッド/プロパティ[1]		機能	参照
CaoWorkspace				
	AddController	M	コントローラに接続	P.8
CaoController				
	Index	P	コントローラ番号の取得	P.10
	Name	P	コントローラ名の取得	P.10
	GetVariableNames	M	接続可能な変数名リストの取得	P.10
	Variables	P	コントローラが保持する変数コレクションの取得	P.10
	AddVariable	M	変数オブジェクトの追加	P.11
	Execute	M	拡張コマンドの実行	P.12
CaoVariable				
	Index	P	変数番号の取得	P.19
	Name	P	変数名の取得	P.19
	Value	P	値の取得/設定	P.19

3.2. メソッド・プロパティ

3.2.1. CaoWorkspace クラス

3.2.1.1. AddController メソッド

CaoWorkspace に，コントローラオブジェクトを追加します．KRCC コントローラプロバイダでは，AddController メソッド実行時に渡されたパラメータを参照し，該当するロボットコントローラと接続を行います．以下に，AddController メソッドの仕様を示します．

書式

```
AddController
(
"<コントローラ名>",              // コントローラ名(任意)
" CaoProv.Kawasaki.KRCC",    // プロバイダ名（固定）
"<マシン名>",                  // プロバイダ実行マシン名(未使用)
"<オプション>"                 // オプション文字列
```

[1] M:メソッド, P:プロパティ, E:イベントをそれぞれ示します．

)

オプション

以下にオプション文字列に指定するオプションを示します. オプション文字列は下記に示す各オプションをカンマ(,)でつなげた文字列となります.

オプション	必須	説明	値範囲	デフォルト値
CONN	○	接続するロボットコントローラのアドレスおよびポートを指定します.		
HostName	-	ロボットコントローラのホスト名を文字列で指定します.		as
Timeout	-	ロボットコントローラに接続する際のタイムアウトの時間をミリ秒単位で指定します.	0-4294967295	1000

使用例(C#)

```
// Engine オブジェクト
ORiN2.ManagedCAO.CCaoEngine engine = new ORiN2.ManagedCAO.CCaoEngine();
// Workspace オブジェクト
ORiN2.ManagedCAO.CCaoWorkspace workspace = engine.AddWorkspace("NewWrks", "");
// Controller オブジェクト
ORiN2.ManagedCAO.CCaoController controller= workspace.AddController(
    "CONTROLLER",
    "CaoProv.***.***",
    "",
    "CONN=TCP:192.168.0.2");
```

3.2.1.1.1. CONN オプション

以下に Conn オプションの接続パラメータ文字列を示します. ここで中括弧("[]")内は省略可能なことを, 各パラメータの解説中の下線部はオプションを指定しなかった時のデフォルト値をそれぞれ示します.

"CONN=TCP:<IP>[:<Port>]"

<IP> : 接続先 IP アドレス.

<Port> : 接続先ポート. (9105)

Port のデフォルト値はシミュレータに接続する場合のポート番号になっています.

実機は通常 23 番ポートで待ち受けを行っていますのでご注意願います.

3.2.2. CaoController クラス

3.2.2.1. Index プロパティ

コントローラ番号を Long 型(4 バイト整数型)で取得します. この番号は, CaoWorkspace において該当のコントローラを識別するための番号です.

使用例(C#)

```
// Index 取得
long index = controller.Index;
```

3.2.2.2. Name プロパティ

CaoWorkspace クラスの AddController メソッドで指定されたコントローラ名を取得します.

使用例(C#)

```
System.Diagnostics.Debug.WriteLine(controller.Name);
```

3.2.2.3. GetVariableNames メソッド

接続可能な変数名リストを取得します.

書式

```
GetVariableNames
(
"<オプション>"                    // オプション文字列
)
```

オプション

KRCC プロバイダではオプション文字列は使用しません.

使用例(C#)

```
// 変数名リスト取得
string[] variableNames = controller.GetVariableNames("");
```

3.2.2.4. Variables プロパティ

コントローラが保持する, 変数コレクションを取得します.

使用例(C#)

```
// 変数コレクション取得
ORiN2.ManagedCAO.CCaoVariables variables = controller.Variables;
// 変数取得
ORiN2.ManagedCAO.CCaoVariable variable = variables[0];
```

3.2.2.5. AddVariable メソッド

CaoController に変数オブジェクトを追加します. 変数名には 3.3.2 に示すもののみ使用できます.
以下に, AddVariable の仕様を示します.

書式

```
AddVariable
(
"<変数名>",                    // 変数名
"<オプション>"                 // オプション文字列(省略可能)
)
```

3.2.2.6. Execute メソッド

CaoController の拡張コマンドを実行します. 以下に, Execute の仕様を示します.

書式

```
Execute
(
"<拡張コマンド名>",          // 拡張コマンド名
"<オプション文字列>"         // オプション文字列(省略可能)
)
```

以下に, Execute で指定できる拡張コマンド一覧を示します. 使用例は拡張コマンドの詳細で記述しています.

コマンド	説明	参照
Execute	ロボットコントローラ上のプログラムを実行します.	P.13
Abort	ロボットコントローラ上のプログラムの実行を停止します.	P.13
Hold	ロボットコントローラ上のプログラムの実行を一時停止します.	P.14
Continue	ロボットコントローラ上のプログラムの実行を再開します.	P.14
EReset	ロボットコントローラのエラーを解除します.	P.14
SetSpeed	ロボットコントローラ上のプログラムの実行速度を設定します.	P.15
Save	ロボットコントローラ上のプログラムをローカル PC 上に保存します.	P.15
Load	Save コマンドで保存したプログラムをロボットコントローラに転送します.	P.16
GetErrLog	エラー履歴を取得します.	P.16
GetErrLogSingle	エラー履歴から指定したエラー情報を取得します.	P.17
SaveErrLog	エラー履歴をローカル PC 上に保存します.	P.17
SaveOpLog	操作履歴をローカル PC 上に保存します.	P.18

3.2.2.6.1. Execute コマンド

ロボットコントローラ上のプログラムを実行します.

引数の指定方法はVT_ARRAY | VT_VARIANTでプログラム名と実行回数を指定する方法と, VT_BSTRでプログラム名を指定する方法があります.

以下に引数と戻り値を示します.

項目	型説明		
引数 1	VT_ARRAY | VT_VARIANT		
	0	VT_BSTR	実行するプログラム名を指定します.
	1	VT_I4	実行回数を指定します. 負の値を指定した場合は無制限に連続実行します.
引数 2	VT_BSTR で実行するプログラム名を指定します. 引数 1 で第二引数に-1 を指定したときと同じ動作を行います.		
戻り値	なし		

使用例(C#)

```
// Execute 実行
object[] opt = { "test", -1 };
controller.Execute("Execute", opt);
//  連続実行を行う場合は以下のような呼び出しもできます
// controller.Execute("Execute", "test");
```

3.2.2.6.2. Abort コマンド

ロボットコントローラ上のプログラムの実行を停止します.

引数および戻り値はありません.

使用例(C#)

```
// Abort 実行
controller.Execute("Abort", null);
```

3.2.2.6.3. Hold コマンド

ロボットコントローラ上のプログラムの実行を一時停止します.

引数および戻り値はありません.

使用例(C#)

```
// Hold 実行
controller.Execute("Hold", null);
```

3.2.2.6.4. Continue コマンド

ロボットコントローラ上の停止しているプログラムの実行を再開します.

引数および戻り値はありません.

使用例(C#)

```
// Continue 実行
controller.Execute("Continue", null);
```

3.2.2.6.5. EReset コマンド

ロボットコントローラのエラー状態を解除します.

引数および戻り値はありません.

使用例(C#)

```
// EReset 実行
controller.Execute("EReset", null);
```

3.2.2.6.6. SetSpeed コマンド

ロボットコントローラ上のプログラムの実行速度を設定します.

以下に引数と戻り値を示します.

項目	型説明
引数	VT_I4 でプログラムの実行速度を指定します. 値の範囲は 1-100 です.
戻り値	なし

使用例(C#)

```
// SetSpeed 実行
controller.Execute("SetSpeed", 30);
```

3.2.2.6.7. Save コマンド

ロボットコントローラ上のプログラムをローカル PC 上に保存します.

以下に引数と戻り値を示します.

項目	型説明
引数	VT_BSTR で保存するファイルのパスを指定します.
戻り値	なし

使用例(C#)

```
// Save 実行
controller.Execute("Save", @"C:¥Tmp¥Save.txt");
```

3.2.2.6.8. Load コマンド

Save コマンドで保存したプログラムをロボットコントローラに転送します.

以下に引数と戻り値を示します.

項目	型説明
引数	VT_BSTR で転送するファイルのパスを指定します.
戻り値	なし

使用例(C#)

```
// Load 実行
controller.Execute("Load", @"C:¥Tmp¥Save.txt");
```

3.2.2.6.9. GetErrLog コマンド

エラー履歴を取得します.

エラー履歴は新しいものから古いものの順で番号が付与され, 直近に発生したエラーの番号が 1 となります,

以下に引数と戻り値を示します.

項目	型説明
引数	数値(VT_I4)で取得するエラー履歴の件数を指定します. 0 を指定すると保存されているすべてのエラー履歴を返します. 省略時は 0 を指定したことになります.
戻り値	VT_BSTR|VT_ARRAY で指定した件数のエラー情報が戻ります. エラー履歴の件数が指定件数未満の場合、不足した要素には空文字列が戻ります.

使用例(C#)

```
//エラー履歴を直近から 5 件取得
string[] errList = controller.Execute("GetErrLog", 5) as string[];
```

3.2.2.6.10. GetErrLogSingle コマンド

指定した番号のエラー履歴を取得します.

エラー履歴は新しいものから古いものの順で番号が付与され, 直近に発生したエラーの番号が 1 となります,

以下に引数と戻り値を示します.

項目	型説明
引数	数値で取得するエラー履歴の番号を指定します. 省略時は 1 を指定したことになります.
戻り値	VT_BSTR で指定した番号のエラー情報が戻ります. 指定した番号のエラー情報がない場合は実行エラーとなります.

使用例(C#)

```
//エラー履歴からエラー番号 1 の(最後に発生した)エラーを取得
string error = controller.Execute("GetErrLogSingle", 1) as string;
```

3.2.2.6.11. SaveErrLog コマンド

ロボットコントローラ上のエラー履歴をローカル PC 上に保存します.

以下に引数と戻り値を示します.

項目	型説明
引数	VT_BSTR で保存するファイルのパスを指定します.
戻り値	なし

使用例(C#)

```
// SaveErrLog 実行
controller.Execute("SaveErrLog", @"C:¥Tmp¥ErrLog.txt");
```

3.2.2.6.12. SaveOpLog コマンド

ロボットコントローラ上の操作履歴をローカル PC 上に保存します.

以下に引数と戻り値を示します.

項目	型説明
引数	VT_BSTR で保存するファイルのパスを指定します.
戻り値	なし

使用例(C#)

```
// SaveOpLog 実行
controller.Execute("SaveOpLog", @"C:¥Tmp¥OpLog.txt");
```

3.2.3. CaoVariable クラス

3.2.3.1. Index プロパティ

変数番号を Long 型(4 バイト整数型)で取得します. この番号は CaoController において該当の変数を識別する番号を示します.

使用例(C#)

```
// Index 取得
int index = caoVar.Index;
```

3.2.3.2. Name プロパティ

CaoContrller クラスの AddVariable メソッドで指定された変数名を取得します.

使用例(C#)

```
System.Diagnostics.Debug.WriteLine(caoVar.Name);
```

3.2.3.3. Value プロパティ

接続したロボットコントローラに対してデータを取得/設定します. 変数名によって動作が異なります. 詳細は, 3.3.変数一覧を参照してください.

3.3. 変数一覧

各クラスで使用可能な変数一覧を定義します. なお変数は, CaoVariable クラスのオブジェクトを指します.

3.3.1. システム変数とユーザー変数

プロバイダにはシステム変数とユーザー変数という 2 種類の変数が存在します.

システム変数

その変数を保持するオブジェクト内で唯一の情報にアクセスための変数です. システム変数はしばしば静的データである場合があります. システム変数は名前の先頭に"@"がついています.

例)プロバイダバージョン, デバイス製造元, シリアル番号

ユーザー変数

変数を作成する際にどのような情報にアクセスするのかを, オプション文字列を使用することでユーザーが指定できる変数です.

KRCC プロバイダにおけるユーザー変数では, ロボットコントローラで使用されているグローバル変数および信号の値を取得することができます.

3.3.2. CaoController クラス変数

変数名	説明	Value		参照
		get	put	
@MAKER_NAME	メーカー名を取得します.	○	-	P.20
@VERSION	DLL バージョンを取得します.	○	-	P.21
@CURRENT_POSITION	現在のアームの位置を変換値(X, Y, Z, O, A, T)形式で取得します.	○	-	P.21
@CURRENT_ANGLE	現在のアームの位置を各軸変位値(JT1, JT2, JT3, JT4, JT5, JT6)形式で取得します.	○	-	P.22
@NORMAL_STATUS	現在のエラー状態を取得します.	○	-	P.22
@ERROR_NUMBER	現在発生しているエラーの番号を取得します.	○	-	P.23
@ERROR_DESCRIPTION	現在発生しているエラーの説明文を取得します.	○	-	P.23
@以外で開始する任意の名前	グローバル変数および信号の値を取得します.	○	※1	P.24

※1 Type で指定した型によって動作が異なります.

3.3.2.1. @MAKER_NAME

メーカー名の取得をします.

データ型

型説明	
VT_BSTR	メーカー名を取得します.

使用例(C#)

```
// 変数追加
ORiN2.ManagedCAO.CCaoVariable var = controller.AddVariable("@MAKER_NAME","");
// 値取得
string value = var.Value as string;
```

3.3.2.2. @VERSION

DLL のバージョンの取得をします.

データ型

型説明	
VT_BSTR	DLL のバージョンを取得します.

使用例(C#)

```
// 変数追加
ORiN2.ManagedCAO.CCaoVariable var = controller.AddVariable("@VERSION","");
// 値取得
string value = var.Value as string;
```

3.3.2.3. @CURRENT_POSITION

現在のアームの位置を変換値(X, Y, Z, O, A, T)形式で取得します.

取得した値はロボット軸数分の要素の VT_R8 の配列となります.

データ型

型説明	
VT_ARRAY | VT_R8	現在のアームの位置の変換値(X, Y, Z, O, A, T). ロボット軸数が 6 軸より多い場合は JT7・・・(ロボット軸数分, 最大 JT18 まで)が追加されます.

使用例(C#)

```
// 変数追加
ORiN2.ManagedCAO.CCaoVariable var = controller.AddVariable("@CURRENT_POSITION","");
// 値取得
double[] value = var.Value as double[];
double x = value[0];
```

3.3.2.4. @CURRENT_ANGLE

現在のアームの位置を各軸変位値(JT1, JT2, JT3, JT4, JT5, JT6)形式で取得します.
取得した値はロボット軸数分の要素の VT_R8 の配列となります.

データ型

型説明	
VT_ARRAY | VT_R8	現在のアームの位置の各軸変位値(JT1, JT2, JT3, JT4, JT5, JT6). ロボット軸数が 6 軸より多い場合は JT7・・・(ロボット軸数分, 最大 JT18 まで)が追加されます.

使用例(C#)

```
// 変数追加
ORiN2.ManagedCAO.CCaoVariable var = controller.AddVariable("@CURRENT_ANGLE","");
// 値取得
double[] value = var.Value as double[];
double jt1 = value[0];
```

3.3.2.5. @NORMAL_STATUS

現在のエラー状態を取得します.

データ型

型説明	
VT_BOOL	エラーが発生していない場合は True, エラーが発生している場合は False.

使用例(C#)

```
// 変数追加
ORiN2.ManagedCAO.CCaoVariable var = controller.AddVariable("@NORMAL_STATUS","");
// 値取得
bool value = (bool)var.Value;
```

3.3.2.6. @ERROR_NUMBER

現在発生しているエラーの番号を取得します.

データ型

型説明	
VT_I4	現在発生しているエラーの番号. エラーが発生していない場合は 0.

使用例(C#)

```
// 変数追加
ORiN2.ManagedCAO.CCaoVariable var = controller.AddVariable("@ERROR_NUMBER","");
// 値取得
int value = (int)var.Value;
```

3.3.2.7. @ERROR_DESCRIPTION

現在発生しているエラーの説明文を取得します.

データ型

型説明	
VT_BSTR	現在発生しているエラーの説明文. エラーが発生していない場合は空文字列.

使用例(C#)

```
// 変数追加
ORiN2.ManagedCAO.CCaoVariable var =
    controller.AddVariable("@ERROR_DESCRIPTION","");
// 値取得
string value = var.Value as string;
```

3.3.2.8. ユーザー変数

変数名が"@"以外で始まる変数はユーザー変数として扱われます.

ユーザー変数はロボットコントローラで使用されているグローバル変数及び信号の値を取得します.

オプション文字列の Name オプションで変数または信号の名前を, Type プロパティで変数または信号の種類を指定してください.

オプション

オプション	必須	説明	値範囲	デフォルト値
Name	○	変数または信号の名前を指定します. 配列変数全体の値を取得することはできませんので, 配列変数の値を取得する場合は添字まで指定してください. ※名前が[ ]やカンマを含む場合, 全体は( )で囲んで明示的に名前の範囲を指定してください. 例） Name=(a[0]),Type=REAL		
Type	○	変数または信号の種類を指定します. 指定できる値は表 3-2 Type オプション一覧を参照してください.		

データ型

取得される値の型は表 3-2 Type オプション一覧を参照してください.

使用例(C#)

```
// 変数追加
ORiN2.ManagedCAO.CCaoVariable var =
    controller.AddVariable("VAR1","Name=a,Type=REAL");
// 値の取得
double value = System.Convert.ToDouble(var.Value);
// 値の設定
var.Value = 12.34;
```

表 3-2 Type オプション一覧

Type オプションの値	取得/設定する変数または信号の種類	値の型	put_Value
SIGNAL	信号	VT_BOOL	○
STRING	文字列変数	VT_BSTR	○
REAL	実変数	VT_R8	○
INT	整数変数	VT_I4	○
POSE	位置変数	VT_ARRAY | VT_R8	○

4. KRCC プロバイダによるプログラミング

KRCC プロバイダでは，以下の手順でクライアント PC とロボットコントローラを接続することができます．

- CaoEngine の作成
- CaoWorkspace の作成
- CaoController の作成

ロボットコントローラに接続した後は，CaoController の Execute メソッドを使用する，もしくは，CaoVariabe オブジェクトを生成することで， ロボットコントローラの情報にアクセスすることができます．

4.1. サンプルプログラミング

ここでは例としてロボットコントローラから固定のデータの読み込み，指低回数の処理の実行を行うプログラムについて記載します

表 4-1 にサンプルプログラムの要件を，図 4-1 にサンプルプログラムの流れをそれぞれ記述しています．

表 4-1 サンプルプログラムの要件

要件	説明
接続先	TCP/IP で接続する
	接続先 IP アドレスは 127.0.0.1
	接続先ポート番号は 9105
処理内容	以下システム変数のデータの取得と表示 @MAKER_NAME @VERSION @CURRENT_POSITION @CURRENT_ANGLE
	以下処理の指定回数の繰り返し ・コントローラとの接続・解除の繰り返し ・設定変数の値の取得の繰り返し ・プログラムの実行と中断の繰り返し ・プログラムの一時停止と再開の繰り返し ・動作速度の変更 ・Save・Load
	キーが押されたら終了

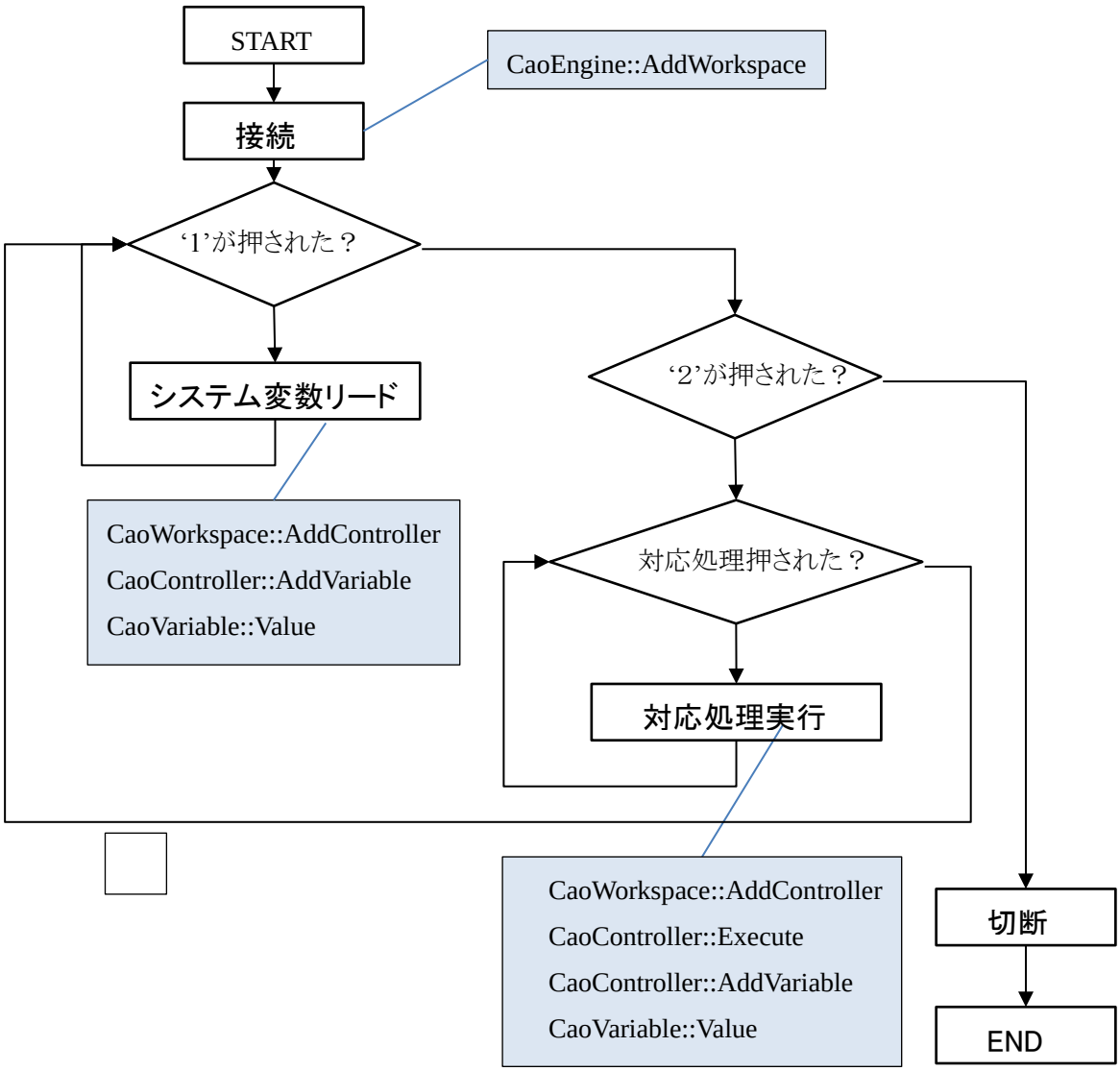

図 4-1 現在位置を表示するプログラムの流れ

以降の節から具体的なコードを示します.

4.1.1. サンプルプログラム

以下にサンプルプログラムの全体を示します.

Sample　Program.cs

```
using System;
using System.Collections.Generic;
using System.Threading;
using ORiN2.ManagedCAO;

namespace Sample
{
    /// <summary>
    /// 川崎重工 F シリーズコントローラプロバイダ サンプルプログラム
    /// </summary>
    /// <remarks>
    /// </remarks>
    class Program
    {
        static string CTL_NAME = "SampleController";
        static string CTL_PROV = "CaoProv.Kawasaki.KRCC";
        // 接続先を自 PC の PCAS にしています. 必要に応じて変更してください
        static string CTL_OPTION = "CONN=TCP:127.0.0.1";
        static string SV_PATH = "C:¥¥temp¥¥orin¥¥FCON_DATA.pg";
        static string LD_PATH = "C:¥¥temp¥¥orin¥¥orin.pg";

        static void Main(string[] args)
        {
            // CCaoEngine の作成
            // using 節を使うとスコープが切れたときに自動的に Dispose が呼び出されて解放される
            using (CCaoEngine engine = new CCaoEngine())
            {
                bool run_fg = true;
                while (run_fg)
                {
                    Console.WriteLine("----------------------------------------------------------------");
                    Console.WriteLine("Controler Check ");
                    Console.WriteLine("");
                    Console.WriteLine("1:Variables check    2:Loop Running Check    other:
```

```
end");
                        Console.WriteLine("-----------------------------------------------------------------");

                        string key =Console.ReadLine();
                        if (int.TryParse(key,out int result))
                        {
                            switch(result)
                            {
                                    case 1:
                                        valiableCheck(engine);
                                        break;

                                    case 2:
                                        loopCheck(engine);
                                        break;

                                    default:
                                        run_fg = false;
                                        break;
                                }
                        }
                        else
                        {
                              run_fg = false;
                        }
                    }

                }
            }

            static void valiableCheck(CCaoEngine engine)
            {
                CCaoController controller = engine.Workspaces[0].AddController(
                                                        CTL_NAME, CTL_PROV, null, CTL_OPTION);

                //   System Valiable    Set & Read
```

```
CCaoVariable val_sName = controller.AddVariable("@MAKER_NAME", "");
string s_val = val_sName.Value.ToString();
Console.WriteLine($"@MAKER_NAME : [{s_val}] OK");

CCaoVariable val_sVer = controller.AddVariable("@VERSION", "");
s_val = val_sVer.Value.ToString();
Console.WriteLine($"@VERSION : [{s_val}] OK");

CCaoVariable val_sPosit = controller.AddVariable("@CURRENT_POSITION", "");
double[] p_val = val_sPosit.Value as double[];
s_val = "";
for (int i = 0; i < p_val.Length; i++) s_val += $" {p_val[i]},";
Console.WriteLine($"@CURRENT_POSITION : [{s_val}] OK");

CCaoVariable val_sAngle = controller.AddVariable("@CURRENT_ANGLE", "");
double[] a_val = val_sAngle.Value as double[];
s_val = "";
for (int i = 0; i < a_val.Length; i++) s_val += $" {a_val[i]},";
Console.WriteLine($"@CURRENT_ANGLE : [{s_val}] OK");

// User Valiable
//  下記はユーザ指定の変数の定義と  Get/Put の設定例です.
//  必要に応じて参照してください

//<Int 型変数>
//CCaoVariable val_uInt=controller.AddVariable(
                                    "U_IntData","Name=@IntData,  Type  =  INT"  );
//val_uInt.Value = 123;
//s_val = val_uInt.Value.ToString();
//Console.WriteLine($"U_IntData : [{s_val}] OK");


//<REAL 型変数>
//CCaoVariable val_uReal = controller.AddVariable(
                                      "U_RealData", "Name=RealData,Type=REAL");
//val_uReal.Value = 123.4567;
//s_val = val_uReal.Value.ToString();
```

```
        //Console.WriteLine($"U_RealData : [{s_val}] OK");


        //<STRING 型変数>
        //CCaoVariable val_uStr = controller.AddVariable(
                                                "U_StrData", "Name=$StrData,Type=STRING");
        //val_uStr.Value = "TEST_Data";
        //s_val = val_uStr.Value.ToString();
        //Console.WriteLine($"U_StrData : [{s_val}] OK");


        //<POSE 型変数>
        //CCaoVariable val_uPose = controller.AddVariable(
                                                "U_PosData", "Name=PosData,Type=POSE");
        //List<double> inp_val= new List<double>() { 1.1, 2.2, 3.3, 4.4, 5.5, 6.6 };
        //val_uPose.Value = (object)inp_val;
        //double[] put_val = val_uPose.Value as double[];
        //s_val = "";
        //for (int i = 0; i < put_val.Length; i++) s_val += $" {put_val[i]},";
        //Console.WriteLine($"@CURRENT_ANGLE : [{s_val}] OK");

    }

    static void loopCheck(CCaoEngine engine)
    {
        CCaoController controller;
        CCaoVariable variable;
        bool run_fg = true;
        int idx;
        System.Diagnostics.Process proc =
                                            System.Diagnostics.Process.GetCurrentProcess();
        while (run_fg)
        {
            Console.WriteLine("< 2.Loop Check >");
            Console.WriteLine("    1 - Controller add and remove.");
            Console.WriteLine("    2 - Variable add and remove.");
            Console.WriteLine("    3 - Variable get value.");
```

```
                    Console.WriteLine("    4 - Program execute and abort.");
                    Console.WriteLine("    5 - Program hold and continue.");
                    Console.WriteLine("    6 - Set speed.");
                    Console.WriteLine("    7 - Save.");
                    Console.WriteLine("    8 - Load.");
                    Console.WriteLine("    other - back to main");
                    Console.WriteLine("---------------------------------------");
                    Console.WriteLine("input {No} {count}");

                    string keys = Console.ReadLine();
                    string[] key = keys.Split(' ');

                    if (key.Length < 2) return;
                    if (!int.TryParse(key[0], out int ikey)) return;
                    if (!int.TryParse(key[1], out int icnt)) return;
                    proc.Refresh();

Console.WriteLine($"Memory(.WorkingSet:{proc.WorkingSet64:N0}    .VirtualMemory:{proc.VirtualMe
morySize64:N0})");
                    switch (ikey)
                    {
                        case 1:

                            Console.WriteLine(" Controller add and remove start.");
                            // 仮想メモリ量  ex) 290,201,600
                            for (int i=0;i<icnt;i++)
                            {
                                Console.WriteLine(
  $"CController Add Name:[{CTL_NAME}] ProvName:[{CTL_PROV}],Option :[{CTL_OPTION}]");
                                controller = engine.Workspaces[0].AddController(
                                                CTL_NAME, CTL_PROV, null, CTL_OPTION);
                                idx = controller.Index;
                                Thread.Sleep(100);
                                Console.WriteLine($"CController Renove:Name:[{CTL_NAME}]");
                                if (!engine.Workspaces[0].Controllers.Remove(CTL_NAME))
                                {
                                    Console.WriteLine(" Remove error!");
```

```
                    run_fg = false;
                    break;
                }
                Console.WriteLine($"{i}: OK");
                Thread.Sleep(100);

            }
            Console.WriteLine(" Controller add and remove fin.");
            break;

        case 2:
            controller = engine.Workspaces[0].AddController(
                                    CTL_NAME, CTL_PROV, null, CTL_OPTION);
            idx = controller.Index;

            Console.WriteLine(" Variable add and remove Start.");
            for (int i = 0; i < icnt; i++)
            {
                Console.WriteLine("AddVariable Name:[tst_str], option:[Name
=$tst_str, Type = STRING]");
                variable =controller.AddVariable(
                                        "tst_str", "Name=$tst_str,Type=STRING");
                Thread.Sleep(100);
                Console.WriteLine("RemoveVariable Name:[tst_str]");
                controller.Variables.Remove("tst_str");
                Thread.Sleep(100);
                Console.WriteLine($"{i}: OK");
            }
            Console.WriteLine(" Variable add and remove fin.");
            if (!engine.Workspaces[0].Controllers.Remove(idx))
            {
                Console.WriteLine(" Remove error!");
                run_fg = false;
            }
            break;

        case 3:
```

```
        controller = engine.Workspaces[0].AddController(
                                    CTL_NAME, CTL_PROV, null, CTL_OPTION);
        idx = controller.Index;
        variable=controller.AddVariable(
                                        "tst_str", "Name=$tst_str,Type=STRING");

        Console.WriteLine(" Variable get value. Start.");
        for (int i = 0; i < icnt; i++)
        {

            object str= variable.Value;
            Console.WriteLine($"{i}:getValiable[tst_str] {str.ToString()}] OK");
            Thread.Sleep(100);
        }
        Console.WriteLine(" Variable get value. fin.");
        if (!controller.Variables.Remove("tst_str"))
        {
            Console.WriteLine(" Variable Remove error!");
            return;
        }
        if (!engine.Workspaces[0].Controllers.Remove(idx))
        {
            Console.WriteLine(" Controller Remove error!");
            run_fg = false;
        }
        break;

    case 4:
        controller = engine.Workspaces[0].AddController(
                                    CTL_NAME, CTL_PROV, null, CTL_OPTION);

        Console.WriteLine(" Program execute and abort. Start.");
        for (int i = 0; i < icnt; i++)
        {
            Console.WriteLine("Execute[orin]");
            controller.Execute("Execute","orin");
            Thread.Sleep(1000);
```

```
            Console.WriteLine("Abort");
            controller.Execute("Abort", "");
            Thread.Sleep(1000);
            Console.WriteLine($"{i}:OK");
        }
        Console.WriteLine(" Program execute and abort.. fin.");
        if (!engine.Workspaces[0].Controllers.Remove(CTL_NAME))
        {
            Console.WriteLine(" Controller Remove error!");
            run_fg = false;
        }
        break;

    case 5:
        controller = engine.Workspaces[0].AddController(
                                CTL_NAME, CTL_PROV, null, CTL_OPTION);
        object[] opt = { "orin", 5 };
        controller.Execute("Execute", opt);
        Console.WriteLine(" Program hold and continue. Start.");
        for (int i = 0; i < icnt; i++)
        {
            Console.WriteLine("Hold");
            controller.Execute("Hold", "");
            Thread.Sleep(500);
            Console.WriteLine("Continue");
            controller.Execute("Continue", "");
            Thread.Sleep(1000);
            Console.WriteLine($"{i}:OK");
        }
        Console.WriteLine(" Program hold and continue. fin.");
        if (!engine.Workspaces[0].Controllers.Remove(CTL_NAME))
        {
            Console.WriteLine(" Controller Remove error!");
            run_fg = false;
        }
        break;
```

```
case 6:
    controller = engine.Workspaces[0].AddController(
                                CTL_NAME, CTL_PROV, null, CTL_OPTION);
    Console.WriteLine(" Set speed. Start.");
    for (int i = 0; i < icnt; i++)
    {
        controller.Execute("SetSpeed", 10);
        Thread.Sleep(300);
        Console.WriteLine($"{i}:SetSpeed OK");
    }
    Console.WriteLine(" Set speed. fin.");
    if (!engine.Workspaces[0].Controllers.Remove(CTL_NAME))
    {
        Console.WriteLine(" Controller Remove error!");
        run_fg = false;
    }
    break;

case 7:
    controller = engine.Workspaces[0].AddController(
                                CTL_NAME, CTL_PROV, null, CTL_OPTION);
    Console.WriteLine(" Save. Start.");
    for (int i = 0; i < icnt; i++)
    {
        controller.Execute("Save", SV_PATH);
        Thread.Sleep(500);
        Console.WriteLine($"{i}:Save:OK");
    }
    Console.WriteLine(" Save. fin.");
    if (!engine.Workspaces[0].Controllers.Remove(CTL_NAME))
    {
        Console.WriteLine(" Controller Remove error!");
        run_fg = false;
    }
    break;

case 8:
```

```
                    controller = engine.Workspaces[0].AddController(
                                        CTL_NAME, CTL_PROV, null, CTL_OPTION);
                    Console.WriteLine(" Load. Start.");
                    for (int i = 0; i < icnt; i++)
                    {
                        controller.Execute("Save", LD_PATH);
                        Thread.Sleep(500);
                        Console.WriteLine($"{i}:Load: OK");
                    }
                    Console.WriteLine(" Load. fin.");
                    if (!engine.Workspaces[0].Controllers.Remove(CTL_NAME))
                    {
                        Console.WriteLine(" Controller Remove error!");
                        run_fg = false;
                    }
                    break;

                default:
                    run_fg = false;
                    break;
            }
            proc.Refresh();

Console.WriteLine($"Memory(.WorkingSet:{proc.WorkingSet64:N0}    .VirtualMemory:{proc.VirtualMe
morySize64:N0})");
            }
        }
    }
}
```

4.1.1.1. 接続

ロボットコントローラと接続するためには，以下の手順を取ります．

(1) ORiN を使用するために必要な CCaoEngine オブジェクトを生成します．
CaoEngine オブジェクトは new キーワードを使って生成します．
生成時に using 節を使用すると解放忘れを防ぐことができます．

使用例(C#)

```
using (CCaoEngine engine = new CCaoEngine())
```

(2) ロボットコントローラに接続するための CaoController オブジェクトを生成します．
CCaoController オブジェクトを生成するには，CCaoWorkspace の AddController メソッドにコントローラ名，プロバイダ名，マシン名，オプション文字列の各種パラメータを指定します．
KRCC プロバイダでは，接続先のアドレス，ポートやタイムアウトなどをオプション文字列で指定します．
以下にコード例を示します．

使用例(C#)

```
static string CTL_NAME = "SampleController";
static string CTL_PROV = "CaoProv.Kawasaki.KRCC";
static string CTL_OPTION = "CONN=TCP:127.0.0.1";
    :
    :
CCaoController controller =
    engine.Workspaces[0].AddController(CTL_NAME, CTL_PROV, null, CTL_OPTION)
```

4.1.1.2. 変数の値の取得

値を取得するための CCaoVariable オブジェクトを生成します.
CCaoVariable オブジェクトを生成するには CCaoController の AddVariable メソッドに変数名とオプション文字列を指定します.（今回使用する変数ではオプション文字列の指定はありません.）
@で始まるシステム変数に関してはオプション文字列は“”でかまいません.
その他のユーザ変数に関しては, オプション文字列に
Name={ロボットコントローラ上の変数名}, Type={値の型（表3. 2参照）}を入れます.
また, 文字列型の場合変数名の先頭に'$'を, 整数の場合は'@'を, 位置情報の場合は'＃'をつける必要があります.（小数点型,（REAL）や各軸値の場合は不要です）
変数の値を取得するには CCaoVariable オブジェクトの Value プロパティを使用します.
実際にプログラムで値を使用する際には CCaoVariable オブジェクトの Value プロパティから取得した値をプログラムで加工してください.
また, Value プロパティに値を設定することで, 変数の値の変更ができます.（一部変数を除く）
以下にコードを示します.

使用例(C#)

```
// System Valiable Set & Read
CCaoVariable val_sName = controller.AddVariable("@MAKER_NAME", "");
string s_val = val_sName.Value.ToString();

CCaoVariable val_sVer = controller.AddVariable("@VERSION", "");
s_val = val_sVer.Value.ToString();

CCaoVariable val_sPosit = controller.AddVariable("@CURRENT_POSITION", "");
double[] p_val = val_sPosit.Value as double[];
s_val = "";
for (int i = 0; i < p_val.Length; i++) s_val += $" {p_val[i]},";

CCaoVariable val_sAngle = controller.AddVariable("@CURRENT_ANGLE", "");
double[] a_val = val_sAngle.Value as double[];
s_val = "";
for (int i = 0; i < a_val.Length; i++) s_val += $" {a_val[i]},";

// User Valiable
// 下記はユーザ指定の変数の定義と Get/Put の設定例です.
// 必要に応じて参照してください
```

```
//<Int 型変数>
CCaoVariable val_uInt=controller.AddVariable(
                                        "U_IntData","Name=@IntData, Type = INT" );
val_uInt.Value = 123;
s_val = val_uInt.Value.ToString();

//<REAL 型変数>
CCaoVariable val_uInt=controller.AddVariable(
                                    "U_RealData", "Name=RealData,Type=REAL");
val_uReal.Value = 123.4567;
s_val = val_uReal.Value.ToString();

//<STRING 型変数>
CCaoVariable val_uStr = controller.AddVariable(
                                     "U_StrData", "Name=$StrData,Type=STRING");
val_uStr.Value = "TEST_Data";
s_val = val_uStr.Value.ToString();


//<POSE 型変数>
CCaoVariable val_uPose = controller.AddVariable(
                                        "U_PosData", "Name=PosData,Type=POSE");
List<double> inp_val= new List<double>() { 1.1, 2.2, 3.3, 4.4, 5.5, 6.6 };
val_uPose.Value = (object)inp_val;
double[] put_val = val_uPose.Value as double[];
s_val = "";
for (int i = 0; i < put_val.Length; i++) s_val += $" {put_val[i]},";
```

4.1.1.3. プログラムの実行

ロボットに対するプログラムの実行，停止等の制御は CCaoController::Execute メソッドを使用します．

CCaoController::Execute メソッドにコントロール用のコマンドを設定し実行します

実行する主なコマンドは以下の通りです．

処理	第一引数(コマンド)	第二引数(コマンド)
実行	Execute	プログラム名
一時停止	Hold	“”
再開	Continue	“”
停止	Abort	“”

使用例(C#)

```
controller.Execute("Execute","orin");
controller.Execute("Hold", "");
controller.Execute("Continue", "");
controller.Execute("Abort", "");
```

4.1.1.4. 切断

コントローラと切断する場合には，生成したオブジェクトを解放すると共に，オブジェクトを管理するコレクションクラスから生成したオブジェクトを削除します．ただし，ORiN.ManagedCAO を使用した場合は CCaoEngine の Dispose メソッドを呼び出すことで CCaoEngine に所属する全オブジェクトが自動的に解放され，オブジェクトを管理するコレクションクラスから削除されるため，CCaoEngine 以外のオブジェクトを明示的に解放する必要はありません．

また，CCaoEngine を生成する際に using 節でスコープを指定すると，スコープを離れたときに自動的に Dispose メソッドが呼び出されるため，解放忘れを防ぐことができます．

フォームアプリケーションなどで CCaoEngine の生成と解放を別のメソッドで行う必要がある場合はアプリケーションの終了前に忘れずに Dispose メソッドを呼び出すようにしてください．

5. KRCC プロバイダエラーコード

本プロバイダには，0x80100001 で表される独自エラーコードが存在します．

エラーコード 0x80100001 のエラーが発生した場合，エラーメッセージにロボットコントローラから通知されたメッセージが設定されますので，エラーの内容はエラーメッセージを参照してください．