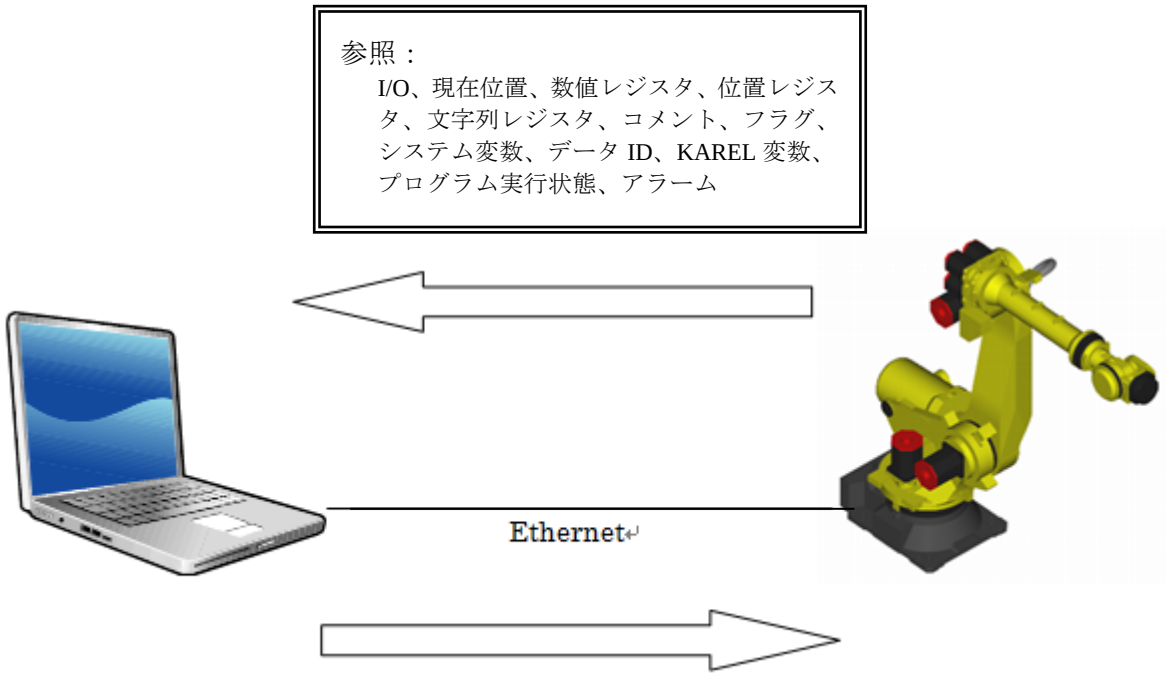
1 ロボットインタフェース概要

FANUC ロボットインタフェースは、Windows PC からロボットデータの参照・設定を行うアプリケーションを作成するための、ソフトウェアです。



(*) FANUC ロボットインタフェースは、TPプログラムなどのファイルの転送機能は含んでおりません。 ロボットとのファイルの転送には、FTPやHTTPプロトコルをお使い下さい。

1.ロボットインタフェース概要 .NET Framework

FANUC ロボットインタフェースは、イーサネットを通して以下のロボットデータにアクセスするためのランタイムモジュール（モジュール名　RobotInterfaceDotNet.dll および bridgeRobotIF.dll）から構成されます。

＜参照＞
I/O、現在位置、数値レジスタ、位置レジスタ、文字列レジスタ、コメント、システム変数、KAREL 変数、プログラム実行状態、アラーム

＜設定＞
I/O、数値レジスタ、位置レジスタ、文字列レジスタ、コメント、システム変数、KAREL 変数

RobotInterfaceDotNet.dll および bridgeRobotIF.dll は.Net Framework DLL で、Visual Basic、Visual C++、Visual C#からオブジェクトとして参照することができます。このモジュールを使用して Windows 上でロボットのデータにアクセスするユーザアプリケーションを作成することができます。bridgeRobotIF.dll は Visual C++を使用する場合のみ使用します。

I/O に関しては、FRRJIF.Core(Visual C++の場合は bridgeCore)オブジェクトから参照、設定することができます。（内蔵 PMC の領域を参照・設定することもできます。）

現在位置、数値レジスタ、位置レジスタ、文字列レジスタ、フラグ、コメント、システム変数、データ ID、KAREL 変数、プログラム実行状態、アラーム履歴に関しては、参照したいデータを DataTable オブジェクト（FRRJIF.Core.DataTable，Visual C++の場合は bridgeDataTable）にあらかじめ登録します。参照するときは、DataTable に登録されたデータを 1 度にまとめて読み込みます。これを DataTable の Refresh と呼びます。次に Refresh するまで参照したデータ値を保持します。
(*) DataTable を用いるのは、アクセス時間短縮のため参照するデータをまとめて 1 度に読み込むためです。

ロボットには同時接続数の制限があります。複数のコンピュータから 1 台のロボットにアクセスしたり、1 つのコンピュータ上の複数のアプリケーションから 1 台のロボットにアクセスするなどして、最大同時接続数を超えて通信を Connect しようとした場合、Connect に失敗します。
(*) 7DA3/06 版以降・7DA4 系列以降では、1 台のロボット全体で同時に最大 4 つの通信を接続できます。
(*) 7DA3/05 版以前では、1 台のロボット全体で最大 1 つの通信を接続できます。これを超えた場合の動作は保証されません。

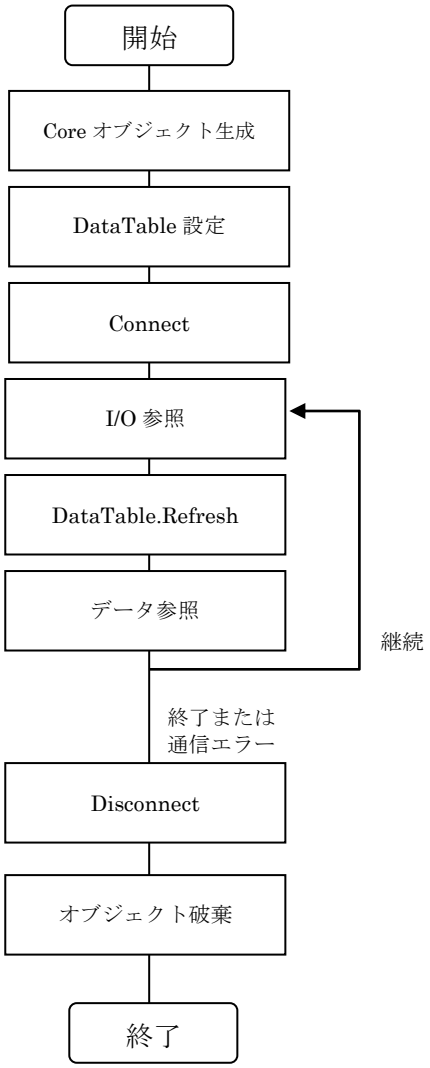
開発用 PC1 台に付き FANUC ロボットインタフェース（A08B-9410-J575）を 1 パッケージご購入いただく必要があります。開発用以外の PC での使用に関しては、追加でご購入いただく必要はありません。（V3.0.0 よりライセンス登録や USB プロテクタは必要ありません）

(*) FANUC ロボットインタフェースは、ＴＰプログラムなどのファイルの転送機能は含んでおりません。　ロボットとのファイルの転送には、ＦＴＰや HTTP プロトコルをお使い下さい。
(*) ロボット制御装置のイーサネット機能の詳細については、お使いのロボット制御装置のイーサネット機能 取扱説明書をご覧下さい。

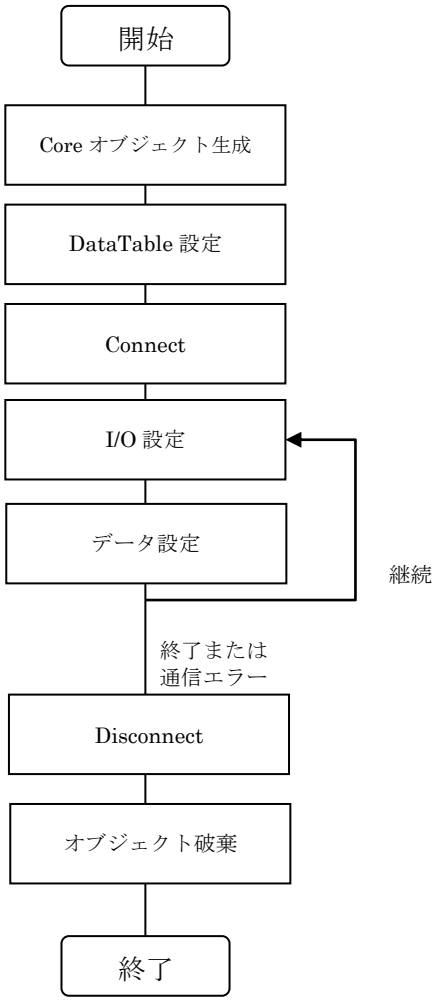
bridgeRobotIF.dll は Visual C++から RobotInterfaceDotNet.dll を円滑に利用する為のラッパークラスを提供します。
Visual C++から各機能を利用する場合はラッパークラスのオブジェクトを使用します。

基本的なフローは以下の通りです。

<データ読み込み>



<データ書き込み>



(*) 通信を終了する際および通信エラー、タイムアウトの場合は、必ずすべてのオブジェクトを破棄して下さい。再び通信する際は新たにオブジェクトを生成しなおす必要があります。

(*) データ読み込みに関しては基本的にポーリング処理となりますので、一定間隔でロボットの状態をモニタリングすることを想定しています。そのためプログラムの実行状況などを一定間隔毎に把握しますので、完全に管理することはできません。

(*) DataTable.Refresh メソッドは、データ読み込みに必要です。データ書き込みの場合は、必須ではありません。

2 動作環境

OS:

Windows 7 (32bit) 、Windows 7 (64bit)
Windows 8.1 (32bit)、Windows 8.1 (64bit)
Windows 10 (32bit) 、Windows 10 (64bit)
Windows 11

開発言語:

Microsoft Visual Basic 2013, 2015, 2017, 2019, 2022
Microsoft Visual C++ 2013, 2015, 2017, 2019, 2022
Microsoft Visual C# 2013, 2015, 2017, 2019, 2022

Microsoft .NET Framework:

.NET Framework 4.0 以降

ロボットコントローラ：

R-J3iB 7D80/45 版以降
R-J3iB 7D81/09 版以降
R-J3iB 7D82/01 版以降
R-J3iB Mate 7D91/01 版以降
R-30iA、R-30iA Mate 全バージョン (*1)
R-30iB、R-30iB Mate 全バージョン (*1)
R-30iB Plus、R-30iB Mate Plus、R-30iB Compact Plus、R-30iB Mini Plus 全バージョン (*1)
R-50iA, R-50iA Mate 全バージョン (*1)

ROBOGUIDE：(*2)

ROBOGUIDE V7 Rev.F (7N07/06)版以降
ROBOGUIDE 上のバーチャルロボット R-30iA 7DA5/15 版以降
ROBOGUIDE 上のバーチャルロボット R-30iA 7DA7/13 版以降
ROBOGUIDE 上のバーチャルロボット R-30iB 全バージョン
ROBOGUIDE 上のバーチャルロボット R-30iB Plus 全バージョン
ROBOGUIDE 上のバーチャルロボット R-50iA 全バージョン

ロボットと PC がイーサネットで通信できること。（ノイズなどの通信への影響がないこと。セキュリティの設定が正しいこと）

(*1) R650 北米専用設定を選択しているときのみ、R553 タッチパネル通信機能 オプションが必要です。R651 標準設定の場合、オプションは必要ありません。

(*2) ROBOGUIDE 上での使用についての詳細は"6. ROBOGUIDE との接続"をご参照下さい。

(注) ロボットのイーサネット機能 取扱説明書をご参照下さい。

(注) Windows、Visual Basic、Visual C++、Visual C#は米国 Microsoft Cooperation の米国及びその他の国における登録商標です。

3 制限事項

ロボットインタフェースをバージョンアップした場合、ロボットインタフェースを使用するプログラムを再コンパイルしなければならない場合があります。

ロボットインタフェースは Windows 上で動作するソフトウェアであり、通信に掛かる時間を保証することはできません。通信状況、ロボットの動作状況、PC の動作状況などにより通信に掛かる時間が長くなることがあります。タイムアウトの値（FRRJIF.Core.TimeOutValue）は、十分余裕のある値にすることを推奨します。

ノイズなどにより通信に障害が発生した場合、ロボットインタフェースは動作できません。例えばロボットが動作を開始するとノイズにより通信ができなくなる場合があります。イーサネットケーブルなどにノイズ対策を行う、ノイズ源から離すなどの対応を行って下さい。（ロボットのイーサネット機能 取扱説明書の付録 "ネットワークの設計について"、"ケーブルの接続" をご参照下さい）

通信エラーやタイムアウトした場合は、Disconnect して FRRJIF または bridgeRobotIF のオブジェクトをすべて破棄して下さい。再び Connect するときに、新たに作成しなおして下さい。（サンプルコードをご参照下さい。）

通信するデータの量や頻度が大きいと通信に時間が掛かります。高速に通信するためにはデータの量や頻度を減らして下さい。

存在しないデータ（例えば存在しないシステム変数）の読み込みを行ってもエラーとはなりません。また存在しないデータの書き込みもエラーとなりません。その場合、書き込みは無視されます。

マルチスレッドでは動作しません。

4 変更点

4.1 V3.0.5 変更点

- データ ID 対応。
  FRRJIF.DataTable.AddSysDataId, FRRJIF.DataTable.AddSysDataIdPos, FRRJIF.DataSysDataId, FRRJIF.DataSysDataIdPos をご参照下さい。

4.2 V3.0.4 変更点

- VisualStudio 2022 対応。
- 安全周辺機器入力の対応範囲を SPI[1-8]から SPI[1-128]へ拡張。
  FRRJIF.VirtualRobotSafetyIOHelper、FRRJIF.VirtualRobotSpi をご参照下さい。

4.3 V3.0.3 変更点

- VisualStudio 2019 対応。
- フラグ対応。
  FRRJIF.DataTable.AddFlag、FRRJIF.DataFlag をご参照下さい。
- バーチャルロボットの安全 I/O（SPI、CSI）対応。
  FRRJIF.VirtualRobotSafetyIOHelper、FRRJIF.VirtualRobotSpi、FRRJIF.VirtualRobotCsi をご参照下さい。

4.4 V3.0.2 変更点

- ロボットインタフェースを使ったプログラムの起動手順を追記。
  "自動運転" をご参照下さい。

4.5 V3.0.1 変更点

- 初版。

5 オブジェクトリファレンス

5.1 表記

VB Visual Basic 2013, 2015, 2017, 2019, 2022 共通
VC++ Visual C++ 2013, 2015, 2017, 2019, 2022 共通
C# Visual C# 2013, 2015, 2017, 2019, 2022 共通

5.2 FRRJIF.Core

5.2.1 メソッド一覧

- 接続メソッド
  Connect, Disconnect, TimeOutValue

- IO メソッド
  ReadSDI, ReadSDO, ReadRDI, ReadRDO, ReadSI, ReadSO, ReadUI, ReadUO, ReadGI, ReadGO, WriteSDI, WriteSDO, WriteRDO, WriteSO, WriteUO, WriteGI, WriteGO

- DataTable メソッド
  DataTable, DataTable2

- アラームメソッド
  ClearAlarm

※FRRJIF.Core のオブジェクトを生成する時、引数として「System.Text.Encoding」のオブジェクトを使って使用するエンコードを指定して下さい。指定されない場合、実行マシンのデフォルトのエンコードが使用されます。
※bridgeRobotIF.dll を使用する VC++の場合は引数としてエンコードを表す文字列(shiftjis ならば”Shift-Jis”)を bridgeCommon::setEncode の引数として指定して下さい。以後は bridgeCommon::getEncode のコールで「System.Text.Encoding」のオブジェクトを得られます。

5.2.1.1 Connect ー ロボットに接続

VB:	Function Connect(ByVal HostName As String) As Boolean
VC++	int Connect(String HostName);
C#	bool Connect(string *HostName*)

引数にホスト名（または IP アドレス）を指定します。
DataTable(後述)登録後に Connect します。

成功すると True, 失敗すると False が返ります。
VC++の場合は成功すると 1、失敗すると 0 が返ります。

5.2.1.2 Disconnectーロボットとの通信切断

VB:	Function Disconnect() As Boolean
VC++	BOOL Disconnect();

C#	bool Disconnect()

成功すると True, 失敗すると False が返ります。

(*) Disconnect した後に再 Connect する場合は、FRRJIF のオブジェクトをすべて破棄して、新たに作成しなおして下さい。

5.2.1.3 TimeOutValue－ロボットとの通信タイムアウト時間

VB:	get_TimeOutValue As Integer set_TimeOutValue(Byval newValue As Integer) or TimeOutValue property
VC++:	Int get_TimeOutValue(); void put_TimeOutValue(int newValue);
C#	int get_TimeOutValue() void set_TimeOutValue(int newValue) or TimeOutValue property

ロボットとの通信タイムアウト時間を設定・参照するプロパティです。単位は ms です。初期値は 10000（10 秒）です。最小値は 100 です。（最小値未満の値は設定できません。）小さな値を設定する場合は、タイムアウトしやすくなることを考慮して下さい。
Core オブジェクトを破棄した場合、TimeOutValue の値も破棄されます。次に Core を作成したときは初期値に戻ります。

(*) タイムアウトした場合は、Disconnect して FRRJIF または bridgeRobotIF のオブジェクトをすべて破棄して下さい。再び Connect するときに、新たに作成しなおして下さい。

5.2.1.4 ReadSDI, ReadSDO, ReadRDI, ReadRDO, ReadSI, ReadSO, ReadUI, ReadUO, ReadGI, ReadGO－I/O 参照

VB:	Function ReadXXX(ByVal Index As Integer, ByRef Buffer As System.Array, ByVal Count As Integer) As Boolean
VC++:	int bridgeLibCore.ReadXXX(short Index, array<short>^ bufArray, int Cnt);
C#	bool ReadXXX(int *Index*, ref System.Array *Buffer*, int *Count*)

（XXX は SDI, SDO, RDI, RDO, SI, SO, UI, UO のいずれか）

VB:	Function ReadXX(ByVal Index As Integer, ByRef Buffer As System.Array, ByVal Count As Integer) As Boolean
VC++	long bridgeLibCore.ReadXX(short Index, array<int>^ bufArray, int Cnt)
C#	bool ReadXX(int *Index*, ref System.Array *Buffer*, int *Count*)

（XX は GI, GO のいずれか）

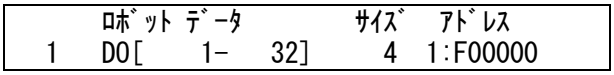
一連の I/O を 1 度に参照することができます。
引数に参照する I/O の最初のインデックス、参照する個数分の Integer(または Long)の配列、参照する個数を指定します。

R-30*i*A 以前の内蔵 PMC には以下のようにアクセスします。

- ReadSDO で内蔵 PMC に割り振られている SDO[10001]以降（K 領域）および SDO[11001]以降（R 領域）を参照することができます。ただし K 領域と R 領域をまとめて 1 度の関数呼び出しで参照することはできません。K 領域と R 領域で別々に呼び出して下さい。
- ReadGO で内蔵 PMC に割り振られている GO[10001]以降（D 領域）を参照することができます。ただし普通の GO と D 領域をまとめて 1 度の関数呼び出しで参照することはできません。普通の GO と D 領域で別々に呼び出して下さい。

R-30*i*B, R-30*i*B Plus の内蔵 PMC では、標準では、SDO[10001]以降（K 領域）、SDO[11001]以降（R 領域）、GO[10001]以降(D 領域)を参照することはできません。内蔵 PMC の互換設定機能を使用することで、従来機種と同様に、K0-K17、R0-R1499、D0-D2999 を参照することができます。ただし、互換設定機能を使用すると、内蔵 PMC 機能が制限されます。

R-30*i*B, R-30*i*B Plus で、ロボットインタフェースと内蔵 PMC の間でデータ交換する場合は、PMC 内部 I/O 割付を設定して、PMC の F 領域をロボットの DO に、または PMC の G 領域をロボットの DI に割り付けて下さい。
例えば、以下のように設定した場合は、ロボットの DO[1-32]の 32 点の信号が、第 1 系統 PMC の信号アドレス F0-F3 の 4 バイトに割り付けられます。ロボットインタフェースから DO[1-32]を参照･設定することで、F0-F3 を参照･設定することができます。



ReadSDI で WI を参照できます。ただし WI の Index に 8000 を加えます。例えば WI[1]は ReadSDI に 8001 を指定して呼び出します。
ReadSDI で WSI を参照できます。ただし WSI の Index に 8400 を加えます。例えば WSI[1]は ReadSDI に 8401 を指定して呼び出します。
ReadSDO で WO を参照できます。ただし WO の Index に 8000 を加えます。例えば WO[1]は ReadSDO に 8001 を指定して呼び出します。
ReadGI で AI を参照できます。ただし AI の Index に 1000 を加えます。例えば AI[1]は ReadGI に 1001 を指定して呼び出します。
ReadGO で AO を参照できます。ただし AO の Index に 1000 を加えます。例えば AO [1]は ReadGO に 1001 を指定して呼び出します。
Visual Basic、C#の場合、成功すると True, 失敗すると False が返ります。Visual C++の場合、成功すると 1, 失敗すると 0 が返ります。（存在しない I/O を参照した場合にも成功します。その場合の値は 0 です。）

＜例＞ SDI[1]から SDI[5]まで参照する場合

```
Dim intBuffer(0 to 4) As Integer
Dim blnResult As Boolean

blnResult = objCore.ReadSDI(1, intBuffer(), 5)
```

(*) SDI[1]から SDI[5]までをばらばらに複数回に分けて参照するより、上記のように 1 度にまとめて参照したほうが短時間で行うことができます。

(*) GI, GO は Long の配列を使用します。

5.2.1.5 WriteSDI, WriteSDO, WriteRDO, WriteSO, WriteUO, WriteGI, WriteGO－I/O 設定

VB:	Function WriteXXX(ByVal Index As Integer, ByRef Buffer As System.Array, ByVal Count As Integer) As Boolean
VC++:	long bridgeLibCore.WriteXXX(short Index, array<short>^ bufArray, int Cnt);
C#	bool WriteXXX(int *Index*, ref System.Array *Buffer*, int *Count*)

（XXX は SDI, SDO, RDO, SO, UO のいずれか）

VB:	Function WriteXX(ByVal Index As Integer, ByRef Buffer As System.Array, ByVal Count As Integer) As Boolean
VC++:	long bridgeLibCore.WriteXX(short Index, array<int>^ bufArray, int Cnt)
C#	WriteXX(int *Index*, ref System.Array *Buffer*, int *Count*)

（XX は GI, GO のいずれか）

一連の I/O を 1 度に設定することができます。
引数に設定する I/O の最初のインデックス、設定する個数分の Integer(または Long)の配列、設定する個数を指定します。

R-30*i*A 以前の内蔵 PMC には以下のようにアクセスします。

- WriteSDO で内蔵 PMC に割り振られている　SDO[10001]以降（K 領域）および SDO[11001]以降（R 領域）を設定することができます。ただし K 領域と R 領域をまとめて 1 度の関数呼び出しで設定することはできません。K 領域と R 領域で別々に呼び出して下さい。
- WriteGO で内蔵 PMC に割り振られている　GO[10001]以降（D 領域）を設定することができます。ただし普通の GO と D 領域をまとめて 1 度の関数呼び出しで設定することはできません。普通の GO と D 領域で別々に呼び出して下さい。

R-30*i*B, R-30*i*B Plus の内蔵 PMC では、標準では、SDO[10001]以降（K 領域）、SDO[11001]以降（R 領域）、GO[10001]以降(D 領域)を参照することはできません。内蔵 PMC の互換設定機能を使用することで、従来機種と同様に、K0-K17、R0-R1499、D0-D2999 を参照することができます。ただし、互換設定機能を使用すると、内蔵 PMC 機能が制限されます。
R-30*i*B, R-30*i*B Plus で、ロボットインタフェースと内蔵 PMC の間でデータ交換する場合は、PMC 内部 I/O 割付を設定して、PMC の F 領域をロボットの DO に、または PMC の G 領域をロボットの DI に割り付けて下さい。
例えば、以下のように設定した場合は、ロボットの DO[1-32]の 32 点の信号が、第 1 系統 PMC の信号アドレス F0-F3 の 4 バイトに割り付けられます。ロボットインタフェースから DO[1-32]を参照･設定することで、F0-F3 を参照･設定することができます。

	ﾛﾎﾞｯﾄ ﾃﾞｰﾀ		ｻｲｽﾞ	ｱﾄﾞﾚｽ
1	DO[	1- 32]	4	1:F00000

WriteSDO で WO を設定できます。ただし WO の Index に 8000 を加えます。例えば WO[1]は WriteSDO に 8001 を指定して呼び出します。
WriteGO で AO を設定できます。ただし AO の Index に 1000 を加えます。例えば AO [1]は WriteGO に 1001 を指定して呼び出します。

Visual Basic、C#の場合、成功すると True, 失敗すると False が返ります。Visual C++の場合、成功すると 1, 失敗すると 0 が返ります。（存在しない I/O を設定した場合にも成功します。その場合の設定は無視されます。）

＜例＞　SDO[1]から SDO[5]までオンに設定する場合

```
Dim intBuffer(0 to 4) As Integer
Dim blnResult As Boolean

intBuffer(0) = 1
intBuffer(1) = 1

intBuffer(2) = 1
intBuffer(3) = 1
intBuffer(4) = 1
blnResult = objCore.WriteSDO(1, intBuffer(), 5)
```

(*) SDO[1]から SDO[5]までをばらばらに複数回に分けて設定するより、上記のように 1 度にまとめて設定したほうが短時間で行うことができます。

(*) GI, GO は Long の配列を使用します。

(*) SDI, GI への設定は、擬似状態で行って下さい。

5.2.1.6 DataTable－DataTable オブジェクトの取得

VB:	FRRJIf.DataTable get_DataTable()
VC++:	bridgeDataTable^ get_DataTable();
C#	FRRJIf.DataTable get_DataTable() or DataTable property

FRRJIF.DataTable オブジェクト（後述）を取得できます。

5.2.1.7 DataTable2－2 つ目の DataTable オブジェクトの取得

VB:	FRRJIf.DataTable get_DataTable2()
VC++:	bridgeDataTable^ get_DataTable2();
C#	FRRJIf.DataTable get_DataTable2()or DataTable2 property

2 つ目の FRRJIF.DataTable オブジェクト（後述）を取得できます。
FRRJIf.Core. DataTable で取得した 1 つ目の DataTable と FRRJIf.Core. DataTable2 で取得した 2 つ目の DataTable の機能は同じです。別々に Refresh することができます。Refresh する頻度が違うデータを 2 つの DataTable に分けて使うことができます。
(*) 1 つ目の DataTable の AddXXX メソッドの呼び出しが全て終了してから、2 つ目の DataTable を取得します。2 つ目の DataTable を取得した後は、1 つ目の DataTable に AddXXX の呼び出しはできません。

5.2.1.8 ClearAlarm－アラーム履歴を消去

VB:	Function ClearAlarm() As Boolean
VC++:	BOOL ClearAlarm(int vlngType);
C#	bool ClearAlarm(int *vlngType*)

Visual C++の場合、vlngType に 0 を渡します。成功すると True, 失敗すると False が返ります。

注意事項
(*) タイムアウトなどで通信が失敗した場合は、Disconnect して FRRJIF のオブジェクトをすべて破棄して下さい。再び Connect するときに、新たに作成しなおして下さい。

5.3 FRRJIF.DataTable

5.3.1 メソッド一覧

- データ登録メソッド
  Clear, AddCurPosUF, AddNumReg, AddPosReg, AddPosRegXyzwpr, AddPosRegMG, AddSysVar, AddSysVarPos, AddSysDataId, AddSysDataIdPos, AddTask, AddAlarm, AddString, AddFlag
- データ参照メソッド
  Refresh

5.3.1.1 Clear－DataTable をクリア

VB:	Function Clear() As Boolean
VC++:	BOOL Clear();
C#	bool Clear()

接続後に Clear メソッドを呼んではいけません。
成功すると True, 失敗すると False が返ります。

5.3.1.2 AddCurPosUF－DataTable に現在位置を登録

VB:	Function AddCurPosUF(ByVal DataType As FRRJIf.FRIF_DATA_TYPE, ByVal Group As Integer, ByVal UF As Integer) As FRRJIf.DataCurPos
VC++:	bridgeDataCurPos AddCurPosUF(FRRJIf.FRIF_DATA_TYPE DataType, int Group, int UF);
C#	FRRJIf.DataCurPos AddCurPosUF(FRRJIf.FRIF_DATA_TYPE *DataType*, int *Group*, int *UF*)

引数で DataType として CURPOS（Visual C++の場合、3）、動作ｸﾞﾙｰﾌﾟ番号、UF に参照に使用するユーザ座標系番号を指定します。

成功すると DataCurPos オブジェクト， 失敗すると Nothing が返ります。DataCurPos オブジェクトから UF で指定したユーザ座標系から見た現在位置を参照することができます。UF＝0 のときはワールド座標系から見た現在位置となります。次の場合、現在選択されているユーザ座標系から見た現在位置となります

7DA4（V7.40）より前のバージョン	UF=15
7DA4/P01 - 7DA4/P11	UF=63
7DA4/P12 以降	UF=-1

＜例＞　動作ｸﾞﾙｰﾌﾟ 1 を登録する場合（ユーザ座標系 1）

```
Dim objDataCurPos As DataCurPos

Set objDataCurPos = objCore.DataTable.AddCurPosUF(CURPOS, 1, 1)
```

5.3.1.3 AddNumReg－DataTable に数値レジスタを登録

VB:	Function AddNumReg(ByVal DataType As FRRJIf.FRIF_DATA_TYPE, ByVal StartIndex As Integer, ByVal EndIndex As Integer) As FRRJIf.DataNumReg
VC++:	bridgeDataNumReg AddNumReg(FRRJIf.FRIF_DATA_TYPE DataType, int StartIndex, int EndIndex);
C#	FRRJIf.DataNumReg AddNumReg(FRRJIf.FRIF_DATA_TYPE *DataType*, int *StartIndex*, int *EndIndex*)

引数で DataType として整数の場合、NUMREG_INT（Visual C++の場合、0）または実数の場合 NUMREG_REAL（Visual C++の場合、1）、と対象の最初のインデックスと最後のインデックスを指定します。
NUMREG_INT で整数を指定した場合、値が実数でも整数に変換されます。NUMREG_REAL で実数を指定した場合、値が整数でも実数に変換されます。

成功すると DataNumReg オブジェクト， 失敗すると Nothing が返ります。DataNumReg オブジェクトからレジスタを参照することができます。

＜例＞　レジ[1]からレジ[15]まで登録する場合

```
Dim objDataNumReg As DataNumReg

Set objDataNumReg = objCore.DataTable.AddNumReg(NUMREG_INT, 1, 15)
```

5.3.1.4 AddPosReg－DataTable に位置レジスタを登録

VB:	Function AddPosReg(ByVal DataType As FRRJIf.FRIF_DATA_TYPE, ByVal Group As Integer, ByVal StartIndex As Integer, ByVal EndIndex As Integer) As FRRJIf.DataPosReg
VC++:	bridgeDataPosReg AddPosReg(FRRJIf.FRIF_DATA_TYPE DataType, int Group, int StartIndex, int EndIndex);
C#	FRRJIf.DataPosReg AddPosReg(FRRJIf.FRIF_DATA_TYPE *DataType*, int *Group*, int *StartIndex*, int *EndIndex*)

引数で DataType として POSREG（Visual C++の場合、2）、動作ｸﾞﾙｰﾌﾟ番号と対象の最初のインデックスと最後のインデックスを指定します。

成功すると DataPosReg オブジェクト， 失敗すると Nothing が返ります。DataPosReg オブジェクトから位置レジスタの参照、書き込みをすることができます。

＜例＞　動作ｸﾞﾙｰﾌﾟ 1 のイチレジ[1]からイチレジ[10]まで登録する場合

```
Dim objDataPosReg As DataPosReg
```

Set objDataPosReg = objCore.DataTable.AddPosReg(POSREG, 1, 1,10)

5.3.1.5 AddPosRegXyzwpr－DataTable に位置レジスタを登録

VB:	Function AddPosRegXyzwpr(ByVal DataType As FRRJIf.FRIF_DATA_TYPE, ByVal Group As Integer, ByVal StartIndex As Integer, ByVal EndIndex As Integer) As FRRJIf.DataPosRegXyzwpr
VC++:	bridgeDataPosRegXyzwpr AddPosRegXyzwpr(FRRJIf.FRIF_DATA_TYPE DataType, int Group, int StartIndex, int EndIndex);
C#	FRRJIf.DataPosRegXyzwpr AddPosRegXyzwpr(FRRJIf.FRIF_DATA_TYPE *DataType*, int *Group*, int *StartIndex*, int *EndIndex*)

引数で DataType として POSREG_XYZWPR（Visual C++の場合、12）、動作ｸﾞﾙｰﾌﾟ番号と対象の最初のインデックスと最後のインデックスを指定します。

成功すると DataPosRegXyzwpr オブジェクト， 失敗すると Nothing が返ります。DataPosRegXyzwpr オブジェクトから位置レジスタを高速に書き込むことができます。（参照はできません）

＜例＞　動作ｸﾞﾙｰﾌﾟ 1 のイチレジ[1]からイチレジ[10]まで登録する場合
Dim objDataPosRegXyzwpr As DataPosRegXyzwpr

Set objDataPosRegXyzwpr = objCore.DataTable.AddPosRegXyzwpr (POSREG_XYZWPR, 1, 1,10)

5.3.1.6 AddPosRegMG－DataTable に位置レジスタ（マルチグループ）を登録

VB:	Function AddPosRegMG(ByVal DataType As FRRJIf.FRIF_DATA_TYPE, ByVal Group As String, ByVal StartIndex As Integer, ByVal EndIndex As Integer) As FRRJIf.DataPosRegMG
VC++:	bridgeDataPosRegMG AddPosRegMG(FRRJIf.FRIF_DATA_TYPE DataType, String Group, int StartIndex, int EndIndex);
C#	FRRJIf.DataPosReg AddPosReg(FRRJIf.FRIF_DATA_TYPE *DataType*, String *Group*, int *StartIndex*, int *EndIndex*)

引数で DataType として POSREGMG（Visual C++の場合、36）、動作ｸﾞﾙｰﾌﾟを指定する文字列と対象の最初のインデックスと最後のインデックスを指定します。

動作ｸﾞﾙｰﾌﾟを指定する文字列は、グループ 1 から順に、直交形式の場合は”C”、各軸形式の場合は“J *軸数*”を”,”で連結します。

成功すると DataPosRegMG オブジェクト， 失敗すると Nothing が返ります。DataPosRegMG オブジェクトから位置レジスタの参照、書き込みをすることができます。

＜例＞　動作ｸﾞﾙｰﾌﾟ 1 が直交形式、動作ｸﾞﾙｰﾌﾟ 2 から 5 が各軸形式で軸数が 1 のイチレジ[1]からイチレジ[10]まで登録する場合

Dim objDataPosRegMG As DataPosRegMG

Set objDataPosRegMG = objCore.DataTable.AddPosRegMG(POSREG, “C,J1,J1,J1,J1”, 1,10)

5.3.1.7 AddSysVar－DataTable にシステム変数（整数、実数、文字列型）を登録

VB:	Function AddSysVar(DataType As FRIF_DATA_TYPE, SysVarName As String) As DataSysVar
VC++:	bridgeDataSysVar AddSysVar(FRRJIf.FRIF_DATA_TYPE DataType, String SysVarName);
C#	FRRJIf.DataSysVar AddSysVar(FRRJIf.FRIF_DATA_TYPE *DataType*, string *SysVarName*)

引数で整数、BOOLEANの場合DataTypeとしてSYSVAR_INT（Visual C++の場合、5）または実数の場合SYSVAR_REAL（Visual C++の場合、6）または文字列の場合SYSVAR_STRING(Visual C++の場合、8)、システム変数名を指定します。
KAREL変数の場合、SysVarNameに”$[KARELプログラム名]変数名"と指定します。例えばHTTPKCLの変数$CMDS[1]の場合、"$[HTTPKCL]CMDS[1]"と指定します。
成功するとDataSysVarオブジェクト， 失敗するとNothingが返ります。DataSysVarオブジェクトからシステム変数, KAREL変数を参照することができます。

＜例＞　$FAST_CLOCKを登録する場合
Dim objDataSysVar As DataSysVar

Set objDataSysVar = objCore.DataTable.AddSysVar(SYSVAR_INT, “$FAST_CLOCK”)

(*)R-50iA 以降は、システム変数を参照できません。データ ID を参照するには AddSysDataId をお使い下さい。

(*)文字列型の場合、取得、設定できる最大文字数は 80 文字です。文字数が 80 文字より多い場合、文字列の 80 文字目までを取得、設定できます。

5.3.1.8 AddSysVarPos－DataTable にシステム変数（POSITION 型）を登録

VB:	Function AddSysVarPos(ByVal DataType As FRRJIf.FRIF_DATA_TYPE, ByVal SysVarName As String) As FRRJIf.DataSysVarPos
VC++:	bridgeDataSysVarPos AddSysVarPos(FRRJIf.FRIF_DATA_TYPE DataType, String SysVarName);
C#	FRRJIf.DataSysVarPos AddSysVarPos(FRRJIf.FRIF_DATA_TYPE *DataType*, string *SysVarName*)

引数で DataType として SYSVAR_POS（Visual C++の場合、7）、システム変数名を指定します。
KAREL 変数の場合、SysVarName に”$[KAREL プログラム名]変数名"と指定します。例えば HTTPKCL の変数 $CMDS[1]の場合、"$[HTTPKCL]CMDS[1]"と指定します。

成功すると DataSysVarPos オブジェクト， 失敗すると Nothing が返ります。DataSysVarPos オブジェクトから POSITION 型のシステム変数, KAREL 変数を参照することができます。

＜例＞　$MNUTOOL[1,1]を登録する場合
Dim objDataSysVarPos As DataSysVarPos

Set objDataSysVarPos = objCore.DataTable.AddSysVarPos(SYSVAR_POS, “$MNUTOOL[1,1]”)

(*)R-50iA 以降は、システム変数を参照できません。データ ID を参照するには AddSysDataIdPos をお使い下さい。

5.3.1.9 AddSysDataId－DataTable にデータ ID（整数、実数、文字列型）を登録

VB:	Function AddSysDataId(DataType As FRIF_DATA_TYPE, DataIdName As String) As DataSysDataId
VC++:	bridgeDataSysDataId AddSysDataId(FRRJIf.FRIF_DATA_TYPE DataType, String DataIdName);
C#	FRRJIf.DataSysDataId AddSysDataId(FRRJIf.FRIF_DATA_TYPE *DataType*, string *DataIdName*)

引数で整数、BOOLEANの場合DataTypeとしてDATAID_INT（Visual C++の場合、38）または実数の場合DATAID_REAL（Visual C++の場合、39）または文字列の場合DATAID_STRING(Visual C++の場合、41)、データID名を指定します。
KAREL変数の場合、DataIdNameに”$[KARELプログラム名]変数名"と指定します。例えばHTTPKCLの変数$CMDS[1]の場合、"$[HTTPKCL]CMDS[1]"と指定します。
成功するとDataSysDataIdオブジェクト， 失敗するとNothingが返ります。DataSysDataIdオブジェクトからデータID, KAREL変数を参照することができます。

＜例＞　$TOOLFRAME.ACTIVE_NUM[1]を登録する場合

Dim objDataSysDataId As DataSysDataId

Set objDataSysDataId = objCore.DataTable.AddSysDataId(DATAID_INT, “$TOOLFRAME.ACTIVE_NUM[1]”)

(*)文字列型の場合、取得、設定できる最大文字数は 80 文字です。文字数が 80 文字より多い場合、文字列の 80 文字目までを取得、設定できます。

5.3.1.10 AddSysDataIdPos－DataTable にデータ ID（POSITION 型）を登録

VB:	Function AddSysDataIdPos(ByVal DataType As FRRJIf.FRIF_DATA_TYPE, ByVal DataIdName As String) As FRRJIf.DataSysDataIdPos
VC++:	bridgeDataSysDataIdPos AddSysDataIdPos(FRRJIf.FRIF_DATA_TYPE DataType, String DataIdName);
C#	FRRJIf.DataSysDataIdPos AddSysDataIdPos(FRRJIf.FRIF_DATA_TYPE *DataType*, string *DataIdName*)

引数で DataType として DATAID_POS（Visual C++の場合、40）、データ ID 名を指定します。
KAREL 変数の場合、DataIdName に”$[KAREL プログラム名]変数名"と指定します。例えば HTTPKCL の変数$CMDS[1]の場合、"$[HTTPKCL]CMDS[1]"と指定します。

成功すると DataSysDataIdPos オブジェクト， 失敗すると Nothing が返ります。DataSysDataIdPos オブジェクトから POSITION 型のシステム変数, KAREL 変数を参照することができます。

＜例＞ $TOOLFRAME.FRAME[1,1]を登録する場合
Dim objDataSysDataIdPos As DataSysDataIdPos

Set objDataSysDataIdPos = objCore.DataTable.AddSysDataIdPos(DATAID_POS, “$TOOLFRAME.FRAME[1,1]”)

5.3.1.11 AddTask－DataTable にプログラム実行状態を登録

VB:	Function AddTask(ByVal DataType As FRRJIf.FRIF_DATA_TYPE, ByVal Index As Integer) As FRRJIf.DataTask
VC++:	bridgeDataTask AddTask(FRRJIf.FRIF_DATA_TYPE DataType, int Index)
C#	FRRJIf.DataTask AddTask(FRRJIf.FRIF_DATA_TYPE *DataType*, int *Index*)

引数で DataType、タスク番号を指定します。DataType は以下の 4 つから 1 つを選択します。

DataType	意味
TASK （VisualC++の場合、4）	実行中のプログラム名、実行行が、そのまま返ります。（従来仕様）
TASK_IGNORE_MACRO （VisualC++の場合、33）	実行中のプログラムがマクロの場合、実行中のプログラム名、実行行として、呼び出し元のプログラムの値が返ります。呼び出し元が全てマクロの場合は、プログラム名""、行番号 0 が返ります。
TASK_IGNORE_KAREL （VisualC++の場合、34）	実行中のプログラムが KAREL の場合、実行中のプログラム名、実行行として、呼び出し元のプログラムの値が返ります。呼び出し元が全て KAREL の場合は、プログラム名""、行番号 0 が返ります。
TASK_IGNORE_MACRO_KAREL （VisualC++の場合、35）	実行中のプログラムがマクロまたは KAREL の場合、実行中のプログラム名、実行行として、呼び出し元のプログラムの値が返ります。呼び出し元が全てマクロまたは KAREL の場合は、プログラム名""、行番号 0 が返ります。

(*) マクロや KAREL をプログラム実行状態の取得の対象としたくないときに、TASK_IGNORE_MACRO、TASK_IGNORE_KAREL
または TASK_IGNORE_MACRO_KAREL を指定します。

(*) TASK_IGNORE_MACRO、TASK_IGNORE_KAREL および TASK_IGNORE_MACRO_KAREL は、ロボットの版数が R-30iA 7DA7/24 以降の場合に使用できます。

マルチタスクのシステムで、2 つのタスクが同時に実行されている場合は、タスク番号 1 とタスク番号 2 で、それぞれのタスクの実行状態を読み出す事が出来ます。2 つのタスクのどちらがタスク番号 1 になるかは、プログラム起動のタイミングや、通信のタイミングにより変わりますが、一度タスク番号 1 として読み出したタスクは、プログラムの実行が終了するまで、タスク番号 1 として読み出すことができます。

成功すると DataTask オブジェクト， 失敗すると Nothing が返ります。DataTask オブジェクトからプログラム実行状態を参照することができます。

＜例＞　タスク番号 1 を登録する場合
Dim objDataTask As DataTask

Set objDataTask = objCore.DataTable.AddTask(TASK, 1)

5.3.1.12 AddAlarm－DataTable にアラーム履歴を登録

VB:	Function AddAlarm(ByVal DataType As FRRJIf.FRIF_DATA_TYPE, ByVal AlarmCount As Integer, Optional ByVal AlarmMessageMode As Integer = 0) As FRRJIf.DataAlarm
VC++:	bridgeDataAlarm AddAlarm(FRRJIf.FRIF_DATA_TYPE DataType, int AlarmCount, int AlarmMessageMode);
C#	FRRJIf.DataAlarm AddAlarm(FRRJIf.FRIF_DATA_TYPE *DataType*, int *AlarmCount*, int *AlarmMessageMode*)

アラーム履歴（アラーム履歴画面で参照できる）を参照する場合、引数で DataType として ALARM_LIST（Visual C++の場合、9）、Count にアラーム履歴のうち最新のものから何個参照するかを指定します。Visual C++の場合、AlarmMessageMode に 0 を渡します。

発生アラーム（アラーム発生画面で参照できる）を参照する場合、引数で DataType として ALARM_CURRENT（Visual C++の場合、10）、Count に発生アラームのうち最新のものから何個参照するかを指定します。

成功すると DataAlarm オブジェクト， 失敗すると Nothing が返ります。DataAlarm オブジェクトからアラームを参照することができます。

＜例＞　アラーム履歴を 5 つ分登録する場合
Dim objDataAlarm As DataAlarm

Set objDataAlarm = objCore.DataTable.AddAlarm(ALARM_LIST, 5)

5.3.1.13 AddString－DataTable に文字列データを登録

VB:	Function AddString(ByVal DataType As FRRJIf.FRIF_DATA_TYPE, ByVal StartIndex As Integer, ByVal EndIndex As Integer) As FRRJIf.DataString
VC++:	bridgeDataString AddString(FRRJIf.FRIF_DATA_TYPE DataType, int StartIndex, int EndIndex);
C#	FRRJIf.DataString AddString(FRRJIf.FRIF_DATA_TYPE *DataType*, int *StartIndex*, int *EndIndex*)

引数で DataType は以下のいずれか 1 つを指定します。

文字列レジスタ	STRREG(=13)
文字列レジスタのコメント	STRREG_COMMENT(=14)
数値レジスタのコメント	NUMREG_COMMENT(=15)
位置レジスタのコメント	POSREG_COMMENT(=16)
SDI のコメント	SDI_COMMENT(=17)
SDO のコメント	SDO_COMMENT(=18)

RDI のコメント	RDI_COMMENT(=19)
RDO のコメント	RDO_COMMENT(=20)
UI のコメント	UI_COMMENT(=21)
UO のコメント	UO_COMMENT(=22)
SI のコメント	SI_COMMENT(=23)
SO のコメント	SO_COMMENT(=24)
WI のコメント	WI_COMMENT(=25)
WO のコメント	WO_COMMENT(=26)
WSI のコメント	WSI_COMMENT(=27)
GI のコメント	GI_COMMENT(=29)
GO のコメント	GO_COMMENT(=30)
AI のコメント	AI_COMMENT(=31)
AO のコメント	AO_COMMENT(=32)
フラグのコメント	FLAG_COMMENT(=37)

成功すると DataString オブジェクト， 失敗すると Nothing が返ります。DataString オブジェクトからデータを参照することができます。

＜例＞ 文字列レジスタ[1]から文字列レジスタ[5]までを登録する場合
Dim objDataString As DataString

Set objDataString = objCore.DataTable.AddString(STRREG, 1, 5)

(*)取得、設定できる最大文字数は 80 文字です。文字数が 80 文字より多い場合、文字列の 80 文字目までを取得、設定できます。

5.3.1.14 AddFlag－DataTable にフラグを登録

VB:	Function AddFlag(ByVal StartIndex As Integer, ByVal EndIndex As Integer) As FRRJIf.DataFlag
VC++:	bridgeDataFlag AddFlag(int StartIndex, int EndIndex);
C#	FRRJIf.DataFlag AddFlag(int *StartIndex*, int *EndIndex*)

引数で対象の最初のインデックスと最後のインデックスを指定します。

成功すると DataFlag オブジェクト， 失敗すると Nothing が返ります。DataFlag オブジェクトからフラグを参照することができます。

＜例＞ F[1]から F[15]まで登録する場合
Dim objDataFlag As DataFlag

Set objDataFlag = objCore.DataTable.AddFlag(1, 15)

5.3.1.15 Refresh－DataTable 上に登録された値を最新の状態に更新

VB:	Function Refresh() As Boolean
VC++:	BOOL Refresh();
C#	bool Refresh()

Refresh メソッドを呼び出すと、DataTable 上のデータが最新の状態に更新されます。次に Refresh メソッドが呼ばれるまで、その更新した値を保持します。（Refresh メソッドを呼ばないと値はずっと変わらないということです。）

成功すると True， 失敗すると False が返ります。

注意事項
(*) AddCurPos などのデータ登録は、接続前に行って下さい。接続後に行ってはいけません。 DataTable を変更するときは、Disconnect して、FRRJIF のオブジェクトをすべて破棄してから、DataTable を再作成して下さい。

5.4 FRRJIF.DataCurPos

5.4.1 メソッド一覧

Valid, GetValue

5.4.1.1 Valid－オブジェクトの有効性チェック

VB:	Property Valid As Boolean（読み取り専用）
VC++:	BOOL get_Valid();
C#	bool Valid { get; }

オブジェクトが有効なとき True, 無効なとき False が返ります。

5.4.1.2 GetValue－現在位置を参照

VB:	Function GetValue(ByRef Xyzwpr As System.Array, ByRef Config As System.Array, ByRef Joint As System.Array, ByRef UF As Short, ByRef UT As Short, ByRef ValidC As Short, ByRef ValidJ As Short) As Boolean
VC++:	BOOL GetValueXyzwpr(float* X, float* Y, float* Z, float* W, float* P, float* R, float* E1, float* E2, float* E3, short* C1, short* C2, short* C3, short* C4, short* C5, short* C6, short* C7, short* UF, short* UT, short* validc); BOOL GetValueJoint(float* J1, float* J2, float* J3, float* J4, float* J5, float* J6, float* J7, float* J8, float* J9, short* UT, short* ValidJ);
C#	bool GetValue(ref System.Array *Xyzwpr*, ref System.Array *Config*, ref System.Array *Joint*, ref short *UF*, ref short *UT*, ref short *ValidC*, ref short *ValidJ*)

Visual Basic の場合、引数 Xyzwpr()に直交値が返ります。Xyzwpr()は、9 軸（基本 6 軸、付加 3 軸）分のデータ領域が必要です。

Config()に現在位置の形態が返ります。Config(0)から Config(3)の意味は以下の表のようになります。

	0 以外	0
Config(0)	F(Flip)	N(NonFlip)
Config(1)	L(Left)	R(Right)
Config(2)	U(Up)	D(Down)
Config(3)	T(Front)	B(Back)

Config(4)から Config(6)はターンナンバーをあらわします。

引数 Joint()に各軸値が返ります。Joint()は、9 軸（基本 6 軸、付加 3 軸）分のデータ領域が必要です。

UF には、取得した現在位置のユーザ座標系番号、UT には現在位置のユーザツール番号が返ります。

ValidC は直交値が有効なとき 0 以外、無効なとき 0 が返ります。ValidJ は各軸値が有効なとき 0 以外、無効なとき 0 が返ります。例えば動作ｸﾞﾙｰﾌﾟ 2 のサーボガンのように直交値がない場合は、ValidC に 0 が返ります。

Visual C++の場合、直交値と各軸値は別の関数 GetValueXyzwpr、GetValueJoint で参照します。Visual Basic とは違い、配列は使用しません。X, Y, Z, W, P, R が Xyzwpr()、C1 から C7 が Config()、J1 から J9 が Joint()に対応します

成功すると True, 失敗すると False が返ります。

5.5 FRRJIF.DataNumReg

5.5.1 メソッド一覧

Valid, GetValue, SetValue, SetValues

5.5.1.1 Valid－オブジェクトの有効性チェック

VB:	Property Valid As Boolean（読み取り専用）
VC++:	BOOL get_Valid();
C#	bool Valid { get; }

オブジェクトが有効なとき True, 無効なとき False が返ります。

5.5.1.2 GetValue－レジスタの値を参照

VB:	Function GetValue(ByVal Index As Integer, ByRef Value As Object) As Boolean
VC++:	BOOL GetValue(long Index, object* value);
C#	bool GetValue(int *Index*, ref object *Value*)

引数 Index に参照したいレジスタのインデックスを指定します。Value に値が返ります。

成功すると True, 失敗すると False が返ります。

5.5.1.3 SetValues－レジスタの値を設定

VB:	Function SetValues(ByVal Index As Integer, ByVal Value As Object, ByVal Count As Integer) As Boolean
VC++:	BOOL SetValuesInt(long Index, array<int>^ Value, long Count); BOOL SetValuesReal(long Index, array<float>^ Value, long Count);
C#	bool SetValues(int *Index*, object *Value*, int *Count*)

一連の数値レジスタを高速に設定します。引数 Index に参照したいレジスタの最初のインデックスを指定します。Value に値を指定します。Count に設定する数値レジスタの個数を指定します。Visual C++の場合は、整数のとき SetValuesInt、実数のとき SetValuesReal を使用します。

成功すると True, 失敗すると False が返ります。

5.6 FRRJIF.DataPosReg

5.6.1 メソッド一覧

Valid, GetValue, SetValueJoint, SetValueXyzwpr

5.6.1.1 Valid－オブジェクトの有効性チェック

VB:	Property Valid As Boolean（読み取り専用）
VC++:	BOOL get_Valid();
C#	bool Valid { get; }

オブジェクトが有効なとき True, 無効なとき False が返ります。

5.6.1.2 GetValue－位置レジスタの値を参照

VB:	Function GetValue(ByVal Index As Integer, ByRef Xyzwpr As System.Array, ByRef Config As System.Array, ByRef Joint As System.Array, ByRef UF As Short, ByRef UT As Short, ByRef ValidC As Short, ByRef ValidJ As Short) As Boolean
VC++:	BOOL GetValueXyzwpr(int Index, float* X, float* Y, float* Z, float* W, float* P, float* R, float* E1, float* E2, float* E3, short* C1, short* C2, short* C3, short* C4, short* C5, short* C6, short* C7, short* UF, short* UT, short* validc); BOOL GetValueJoint(int Index, float* J1, float* J2, float* J3, float* J4, float* J5, float* J6, float* J7, float* J8, float* J9, short* UT, short* ValidJ);
C#	bool GetValue(int *Index*, ref System.Array *Xyzwpr*, ref System.Array *Config*, ref System.Array *Joint*, ref short *UF*, ref short *UT*, ref short *ValidC*, ref short *ValidJ*)

引数 Index に参照したい位置レジスタのインデックスを指定します。他の引数は現在位置参照（DataCurPos.GetValue メソッド）と同じです。

成功すると True, 失敗すると False が返ります。

5.6.1.3 SetValueJoint－各軸形式の位置レジスタの値を設定

VB:	Function SetValueJoint(ByVal Index As Integer, ByRef Joint As System.Array, ByVal UF As Short, ByVal UT As Short) As Boolean
VC++:	BOOL SetValueJoint2(int Index, float J1, float J2, float J3, float J4, float J5, float J6, float J7, float J8, float J9, short UF, short UT);
C#	bool SetValueJoint(int *Index*, ref System.Array *Joint*, short *UF*, short *UT*)

引数 Index に設定したい位置レジスタのインデックスを指定します。Joint に設定したい各軸値、UF に設定したいユーザ座標系番号、UT に設定したいユーザツール番号を指定します。
Visual C++の場合は、配列でなく、個々の値を指定します。
成功すると True, 失敗すると False が返ります。

5.6.1.4 SetValueXyzwpr－直交形式の位置レジスタの値を設定

VB:	Function SetValueXyzwpr(ByVal Index As Integer, ByRef Xyzwpr As System.Array, ByRef Config As System.Array, ByVal UF As Short, ByVal UT As Short) As Boolean
VC++:	BOOL SetValueXyzwpr2(int Index, float X, float Y, float Z, float W, float P, float R, float E1, float E2, float E3, short C1, short C2, short C3, short C4, short C5, short C6, short C7, short UF, short UT);
C#	bool SetValueXyzwpr(int *Index*, ref System.Array *Xyzwpr*, ref System.Array *Config*, short *UF*, short *UT*)

引数 Index に設定したい位置レジスタのインデックスを指定します。Xyzwpr()に設定したい直交値、Config に設定したい形態、UF に設定したいユーザ座標系番号、UT に設定したいユーザツール番号を指定します。
Visual C++の場合は、配列でなく、個々の値を指定します。
各値の詳細は、現在位置参照（DataCurPos.GetValue メソッド）をご参照下さい。

成功すると True, 失敗すると False が返ります。

5.7 FRRJIF.DataPosRegXyzwpr

5.7.1 メソッド一覧

Valid, SetValueXyzwpr, Update, Reset

5.7.1.1 Valid－オブジェクトの有効性チェック

VB:	Property Valid As Boolean（読み取り専用）
VC++:	BOOL get_Valid();
C#	bool Valid { get; }

オブジェクトが有効なとき True, 無効なとき False が返ります。

5.7.1.2 SetValueXyzwpr－直交形式の位置レジスタの値をバッファに格納

VB:	Function SetValueXyzwpr(ByVal Index As Integer, ByRef Xyzwpr As System.Array, ByRef Config As System.Array) As Boolean
VC++:	BOOL SetValueXyzwpr2(int Index, float X, float Y, float Z, float W, float P, float R, float E1, float E2, float E3, short C1, short C2, short C3, short C4, short C5, short C6, short C7);
C#	bool SetValueXyzwpr(int *Index*, ref System.Array *Xyzwpr*, ref System.Array *Config*)

引数 Index に設定したい位置レジスタのインデックスを指定します。Xyzwpr()に設定したい直交値、Config に設定したい形態を指定します。SetValueXyzwpr を呼び出しても、すぐにロボットにデータが書き込まれるわけではありません。この PosRegXyzwpr のバッファに書き込まれるだけです。Update メソッドを呼んだとき、バッファ内のデータがまとめてロボットにデータが転送されます。
Visual C++の場合は、配列でなく、個々の値を指定します。
各値の詳細は、現在位置参照（DataCurPos.GetValue メソッド）をご参照下さい。

成功すると True, 失敗すると False が返ります。

5.7.1.3 Update－バッファに格納された直交形式の位置レジスタの値をロボットへ転送

VB:	Function Update() As Boolean
VC++:	BOOL Update();
C#	bool Update()

バッファに格納されたデータをロボットへ転送します。SetValueXyzwpr で格納された最も小さいインデックスと最も大きいインデックスの間の位置レジスタをまとめてロボットに書き込みます。（SetValueXyzwpr でバッファに格納されていない位置レジスタの値は 0 になります。）書き込み後、バッファはリセットされます。

成功すると True, 失敗すると False が返ります。

5.7.1.4 Reset－バッファをリセットします

VB:	Sub Reset( )
VC++:	void Reset();
C#	void Reset()

5.8 FRRJIF.DataPosRegMG

5.8.1 メソッド一覧

Valid, GetValueJoint, GetValueXyzwpr, SetValueJoint, SetValueXyzwpr, Update, Reset

5.8.1.1 Valid－オブジェクトの有効性チェック

VB:	Property Valid As Boolean（読み取り専用）
VC++:	BOOL get_Valid();
C#	bool Valid { get; }

オブジェクトが有効なとき True, 無効なとき False が返ります。

5.8.1.2 GetValueJoint－各軸形式の位置レジスタの値を参照

VB:	Function GetValueJoint(ByVal Index As Integer, ByVal Group As Integer, ByRef float J1, ByRef float J2, ByRef float J3, ByRef float J4, ByRef float J5, ByRef float J6, ByRef float J7, ByRef float J8, ByRef float J9) As Boolean
VC++:	BOOL GetValueJoint(int Index, int Group, float* J1, float* J2, float* J3, float* J4, float* J5, float* J6, float* J7, float* J8, float* J9);
C#	bool GetValueJoint(int *Index*, int *Group*, ref float *J1*, ref float *J2*, ref float *J3*, ref float *J4*, ref float *J5*, ref float *J6*, ref float *J7*, ref float *J8*, ref float *J9*);

引数 Index に設定したい位置レジスタのインデックスを指定します。Group に設定したい動作グループを指定します。J1 から J9 で各軸値が参照できます。その動作グループが AddPosRegMG の際に各軸形式で指定されている必要があります。
成功すると True, 失敗すると False が返ります。

5.8.1.3 GetValueXyzwpr－直交形式の位置レジスタの値を参照

VB:	Function GetValueXyzwpr(int Index, int Group, ByRef float X, ByRef float Y, ByRef float Z, ByRef float W, ByRef float P, ByRef float R, ByRef float E1, ByRef float ByRef, ref float E3, ByRef short C1, ByRef short C2, ByRef short C3, ByRef short C4, ByRef short C5, ByRef short C6, ByRef short C7) As Boolean
VC++:	BOOL GetValueXyzwpr(int Index, int Group, float* X, float* Y, float* Z, float* W, float* P, float* R, float* E1, float* E2, float* E3, short* C1, short* C2, short* C3, short* C4, short* C5, short* C6, short* C7,)
C#	bool GetValueXyzwpr(int *Index*, int *Group*, ref float *X*, ref float *Y*, ref float *Z*, ref float *W*, ref float *P*, ref float *R*, ref float *E1*, ref float *E2*, ref float *E3*, ref short *C1*, ref short *C2*, ref short *C3*, ref short *C4*, ref short *C5*, ref short *C6*, ref short *C7*);

引数 Index に設定したい位置レジスタのインデックスを指定します。Group に設定したい動作グループを指定します。X,Y,Z,W,P,R で直交値が参照できます。E1 から E3 で付加軸値が参照できます。C1 から C7 で形態が参照できます。各値の詳細は、現在位置参照（DataCurPos.GetValue メソッド）をご参照下さい。その動作グループが AddPosRegMG の際に直交形式で指定されている必要があります。
成功すると True, 失敗すると False が返ります。

5.8.1.4 SetValueJoint－各軸形式の位置レジスタの値をバッファに格納

VB:	Function SetValueJoint(ByVal Index As Integer, ByVal Group As Integer, ByRef Joint As System.Array) As Boolean
VC++:	BOOL SetValueJoint2(int Index, int Group, float J1, float J2, float J3, float J4, float J5, float J6, float J7, float J8, float J9);
C#	bool SetValueJoint(int *Index*, int Group, ref System.Array *Joint*)

引数 Index に設定したい位置レジスタのインデックスを指定します。Group に設定したい動作グループを指定しま

す。Joint に設定したい各軸値を指定します。その動作グループが AddPosRegMG の際に各軸形式で指定されている必要があります。各軸値を入れる配列の大きさは AddPosRegMG の際に指定した軸数と一致している必要があります。
Visual C++の場合は、配列でなく、個々の値を指定します。
SetValueJoint を呼び出しても、すぐにロボットにデータが書き込まれるわけではありません。バッファに書き込まれるだけです。Update メソッドを呼んだとき、バッファ内のデータがまとめてロボットにデータが転送されます。

成功すると True, 失敗すると False が返ります。

5.8.1.5 SetValueXyzwpr－直交形式の位置レジスタの値をバッファに格納

VB:	Function SetValueXyzwpr(ByVal Index As Integer, ByVal Group As Integer, ByRef Xyzwpr As System.Array, ByRef Config As System.Array,) As Boolean
VC++:	BOOL SetValueXyzwpr2(int Index, int Group, float X, float Y, float Z, float W, float P, float R, float E1, float E2, float E3, short C1, short C2, short C3, short C4, short C5, short C6, short C7);
C#	bool SetValueXyzwpr(int *Index*, int Group, ref System.Array *Xyzwpr*, ref System.Array *Config*)

引数 Index に設定したい位置レジスタのインデックスを指定します。Group に設定したい動作グループを指定します。Xyzwpr()に設定したい直交値、Config に設定したい形態を指定します。その動作グループが AddPosRegMG の際に直交形式で指定されている必要があります。
Visual C++の場合は、配列でなく、個々の値を指定します。
各値の詳細は、現在位置参照（DataCurPos.GetValue メソッド）をご参照下さい。
SetValueXyzwpr を呼び出しても、すぐにロボットにデータが書き込まれるわけではありません。バッファに書き込まれるだけです。Update メソッドを呼んだとき、バッファ内のデータがまとめてロボットにデータが転送されます。

成功すると True, 失敗すると False が返ります。

5.8.1.6 Update－バッファに格納された全ての位置レジスタの値をロボットへ転送

VB:	Function Update() As Boolean
VC++:	BOOL Update();
C#	bool Update()

バッファに格納されたデータをロボットへ転送します。この DataPosRegMG に指定された開始インデックスから終了インデックスまでの**すべての位置レジスタをまとめてロボットに書き込みます。**（SetValueJoint、SetValueXyzwpr でバッファに格納されていない位置レジスタの値は 0 になります。）書き込み後、バッファはリセットされます。

成功すると True, 失敗すると False が返ります。

5.8.1.7 Reset－バッファをリセットします

VB:	Sub Reset( )
VC++:	void Reset();
C#	void Reset()

5.9 FRRJIF.DataSysVar

(*)R-50iA 以降は、システム変数を参照できません。データ ID を参照するには DataSysDataId をお使い下さい。

(*)文字列型の場合、取得、設定できる最大文字数は 80 文字です。文字数が 80 文字より多い場合、文字列の 80 文字目までを取得、設定できます。

5.9.1 メソッド一覧

Valid, GetValue

5.9.1.1 Valid－オブジェクトの有効性チェック

VB:	Property Valid As Boolean（読み取り専用）
VC++:	BOOL get_Valid();
C#	bool Valid { get; }

オブジェクトが有効なとき True, 無効なとき False が返ります。

5.9.1.2 GetValue－システム変数の値を参照

VB:	Function GetValue(ByRef Value As Object) As Boolean
VC++:	BOOL GetValue(object* value, System.Text.Encoding encode);
C#	bool GetValue(ref object *Value*)

Value に値が返ります。

成功すると True, 失敗すると False が返ります。

5.9.1.3 SetValue－システム変数の値を設定

VB:	Function SetValue(ByVal Value As Object) As Boolean
VC++:	BOOL SetValue(const object value);
C#	bool SetValue(object *Value*)

Value に値を指定します。

成功すると True, 失敗すると False が返ります。

5.10 FRRJIF.DataSysVarPos

(*)R-50iA 以降は、システム変数を参照できません。データ ID を参照するには DataSysDataIdPos をお使い下さい。

5.10.1 メソッド一覧

Valid, GetValue

5.10.1.1 Valid－オブジェクトの有効性チェック

VB:	Property Valid As Boolean（読み取り専用）
VC++:	BOOL get_Valid();
C#	bool Valid { get; }

オブジェクトが有効なとき True, 無効なとき False が返ります。

5.10.1.2 GetValue－POSITION 型システム変数の値を参照

VB:	Function GetValue(ByRef Xyzwpr As System.Array, ByRef Config As System.Array, ByRef Joint As System.Array, ByRef UF As Short, ByRef UT As Short, ByRef ValidC As Short, ByRef ValidJ As Short) As Boolean
VC++:	BOOL GetValueXyzwpr(float* X, float* Y, float* Z, float* W, float* P, float* R, float* E1, float* E2, float* E3, short* C1, short* C2, short* C3, short* C4, short* C5, short* C6, short* C7, short* UF, short* UT, short* validc); BOOL GetValueJoint(float* J1, float* J2, float* J3, float* J4, float* J5, float* J6, float* J7, float* J8, float* J9, short* UT, short* ValidJ);
C#	bool GetValue(ref System.Array *Xyzwpr*, ref System.Array *Config*, ref System.Array *Joint*, ref short *UF*, ref short *UT*, ref short *ValidC*, ref short *ValidJ*)

引数は現在位置参照（DataCurPos.GetValue メソッド）と同じです。

成功すると True, 失敗すると False が返ります。

5.10.1.3 SetValueJoint－各軸形式のシステム変数の値を設定

VB:	Function SetValueJoint(ByRef Joint As System.Array, ByVal UF As Short, ByVal UT As Short) As Boolean
VC++:	BOOL SetValueJoint2(float J1, float J2, float J3, float J4, float J5, float J6, float J7, float J8, float J9, short UF, short UT);
C#	bool SetValueJoint(ref System.Array *Joint*, short *UF*, short *UT*)

Joint()に設定したい各軸値、UF に設定したいユーザ座標系番号、UT に設定したいユーザツール番号を指定します。Visual C++の場合は、配列でなく、個々の値を指定します。

成功すると True, 失敗すると False が返ります。

5.10.1.4 SetValueXyzwpr－直交形式のシステム変数の値を設定

VB:	Function SetValueXyzwpr(ByRef Xyzwpr As System.Array, ByRef Config As System.Array, ByVal UF As Short, ByVal UT As Short) As Boolean
VC++:	BOOL SetValueXyzwpr2(float X, float Y, float Z, float W, float P, float R, float E1, float E2, float E3, short C1, short C2, short C3, short C4, short C5, short C6, short C7, short UF, short UT);
C#	bool SetValueXyzwpr(ref System.Array *Xyzwpr*, ref System.Array *Config*, short *UF*, short *UT*)

Xyzwpr()に設定したい直交値、Config に設定したい形態、UF に設定したいユーザ座標系番号、UT に設定したいユーザツール番号を指定します。Visual C++の場合は、配列でなく、個々の値を指定します。
各値の詳細は、現在位置参照（DataCurPos.GetValue メソッド）をご参照下さい。

成功すると True, 失敗すると False が返ります。

5.11 FRRJIF.DataSysDataSysId

ロボット R-50iA 7DH1/11 版以降でのみ使用可能です。

(*)文字列型の場合、取得、設定できる最大文字数は 80 文字です。文字数が 80 文字より多い場合、文字列の 80 文字目までを取得、設定できます。

5.11.1 メソッド一覧

Valid, GetValue

5.11.1.1 Valid－オブジェクトの有効性チェック

VB:	Property Valid As Boolean（読み取り専用）
VC++:	BOOL get_Valid();
C#	bool Valid { get; }

オブジェクトが有効なとき True, 無効なとき False が返ります。

5.11.1.2 GetValue－データ ID の値を参照

VB:	Function GetValue(ByRef Value As Object) As Boolean
VC++:	BOOL GetValue(object* value, System.Text.Encoding encode);
C#	bool GetValue(ref object *Value*)

Value に値が返ります。

成功すると True, 失敗すると False が返ります。

5.11.1.3 SetValue－データ ID の値を設定

VB:	Function SetValue(ByVal Value As Object) As Boolean
VC++:	BOOL SetValue(const object value);
C#	bool SetValue(object *Value*)

Value に値を指定します。

成功すると True, 失敗すると False が返ります。

5.12 FRRJIF.DataSysDataIdPos

ロボット R-50iA 7DH1/11 版以降でのみ使用可能です。

5.12.1 メソッド一覧

Valid, GetValue

5.12.1.1 Valid－オブジェクトの有効性チェック

VB:	Property Valid As Boolean（読み取り専用）
VC++:	BOOL get_Valid();
C#	bool Valid { get; }

オブジェクトが有効なとき True, 無効なとき False が返ります。

5.12.1.2 GetValue－POSITION 型データ ID の値を参照

VB:	Function GetValue(ByRef Xyzwpr As System.Array, ByRef Config As System.Array, ByRef Joint As System.Array, ByRef UF As Short, ByRef UT As Short, ByRef ValidC As Short, ByRef ValidJ As Short) As Boolean
VC++:	BOOL GetValueXyzwpr(float* X, float* Y, float* Z, float* W, float* P, float* R, float* E1, float* E2, float* E3, short* C1, short* C2, short* C3, short* C4, short* C5, short* C6, short* C7, short* UF, short* UT, short* validc); BOOL GetValueJoint(float* J1, float* J2, float* J3, float* J4, float* J5, float* J6, float* J7, float* J8, float* J9, short* UT, short* ValidJ);
C#	bool GetValue(ref System.Array *Xyzwpr*, ref System.Array *Config*, ref System.Array *Joint*, ref short *UF*, ref short *UT*, ref short *ValidC*, ref short *ValidJ*)

引数は現在位置参照（DataCurPos.GetValue メソッド）と同じです。

成功すると True, 失敗すると False が返ります。

5.12.1.3 SetValueJoint－各軸形式のデータ ID の値を設定

VB:	Function SetValueJoint(ByRef Joint As System.Array, ByVal UF As Short, ByVal UT As Short) As Boolean
VC++:	BOOL SetValueJoint2(float J1, float J2, float J3, float J4, float J5, float J6, float J7, float J8, float J9, short UF, short UT);
C#	bool SetValueJoint(ref System.Array *Joint*, short *UF*, short *UT*)

Joint()に設定したい各軸値、UF に設定したいユーザ座標系番号、UT に設定したいユーザツール番号を指定します。Visual C++の場合は、配列でなく、個々の値を指定します。

成功すると True, 失敗すると False が返ります。

5.12.1.4 SetValueXyzwpr－直交形式のデータ ID の値を設定

VB:	Function SetValueXyzwpr(ByRef Xyzwpr As System.Array, ByRef Config As System.Array, ByVal UF As Short, ByVal UT As Short) As Boolean
VC++:	BOOL SetValueXyzwpr2(float X, float Y, float Z, float W, float P, float R, float E1, float E2, float E3, short C1, short C2, short C3, short C4, short C5, short C6, short C7, short UF, short UT);
C#	bool SetValueXyzwpr(ref System.Array *Xyzwpr*, ref System.Array *Config*, short *UF*, short *UT*)

Xyzwpr()に設定したい直交値、Config に設定したい形態、UF に設定したいユーザ座標系番号、UT に設定したいユーザツール番号を指定します。Visual C++の場合は、配列でなく、個々の値を指定します。
各値の詳細は、現在位置参照（DataCurPos.GetValue メソッド）をご参照下さい。

成功すると True, 失敗すると False が返ります。

5.13 FRRJIF.DataTask

5.13.1 メソッド一覧

Valid, GetValue

5.13.1.1 Valid－オブジェクトの有効性チェック

VB:	Property Valid As Boolean（読み取り専用）
VC++:	BOOL get_Valid();
C#	bool Valid { get; }

オブジェクトが有効なとき True,無効なとき False が返ります。

5.13.1.2 GetValue－プログラム実行状態を参照

VB:	Function GetValue(ByRef ProgName As String, ByRef LineNumber As Short, ByRef State As Short, ByRef ParentProgName As String) As Boolean
VC++:	BOOL GetValue(String* ProgName, short* LineNumber, short* State, String* ParentProgName, System.Text.Encoding encode);
C#	bool GetValue(ref string *ProgName*, ref short *LineNumber*, ref short *State*, ref string *ParentProgName*, System.Text.Encoding *Encode*)

引数 ProgName に現在実行中のプログラム名が返ります。
引数 LineNumber に現在実行中のプログラム行数が返ります。
引数 State に実行状態が返ります。State の値は終了状態のとき 0、一時停止状態のとき 1、実行中は 2 となります。
引数 ParentProgName に最初に起動されたプログラムの名前が返ります。サブプログラムを呼び出していない時は、ProgName と同じになります。

(*) 現在実行中のプログラム名、プログラム行数の対象は FRRJIF.DataTable.AddTask を呼んだときの DataType によって変わります。詳細は、FRRJIF.DataTable.AddTask をご参照下さい。

(*) 取得できるプログラム名の最大文字数は 16 文字です。文字数が 16 文字より多い場合、文字列の 16 文字目までを取得できます。

成功すると True, 失敗すると False が返ります。

5.14 FRRJIF.DataAlarm

5.14.1 メソッド一覧

Valid, GetValue

5.14.1.1 Valid－オブジェクトの有効性チェック

VB:	Property Valid As Boolean（読み取り専用）
VC++:	BOOL get_Valid();
C#	bool Valid { get; }

オブジェクトが有効なとき True,無効なとき False が返ります。

5.14.1.2 GetValue－アラームを参照

VB:	Function GetValue(ByVal Count As Integer, ByRef AlarmID As Short, ByRef AlarmNumber As Short, ByRef CauseAlarmID As Short, ByRef CauseAlarmNumber As Short, ByRef Severity As Short, ByRef Year As Short, ByRef Month As Short, ByRef Day As Short, ByRef Hour As Short, ByRef Minute As Short, ByRef Second As Short, ByRef AlarmMessage As String, ByRef CauseAlarmMessage As String, ByRef SeverityMessage As String) As Boolean
VC++:	BOOL GetValue(int Count, short* AlarmID, short* AlarmNumber, short* CauseAlarmID, short*

	CauseAlarmNumber, short* Severity, short* Year, short* Month, short* Day, short* Hour, short* Minute, short* Second, String* AlarmMessage, String* CauseAlarmMessage, String* SeverityMessage, System.Text.Encoding encoding);
C#	bool GetValue(int *Count*, ref short *AlarmID*, ref short *AlarmNumber*, ref short *CauseAlarmID*, ref short *CauseAlarmNumber*, ref short *Severity*, ref short *Year*, ref short *Month*, ref short *Day*, ref short *Hour*, ref short *Minute*, ref short *Second*, ref string *AlarmMessage*, ref string *CauseAlarmMessage*, ref string *SeverityMessage*)

引数 Count でアラーム履歴の何番目を参照するか指定します。（最新のアラームを参照するときは 1 を指定します。）
引数 AlarmID にアラーム ID が返ります。「ｻｰﾎﾞ-001」の場合、「ｻｰﾎﾞ」を表すアラーム ID の 11 が読めます。アラーム ID の番号は、R-J3 の取扱説明書のアラームコード表に記載されています。発生アラームがないとき、AlarmID は 0 と読めます。
引数 AlarmNumber にアラーム番号が返ります。「ｻｰﾎﾞ-001」の場合、「001」の 1 が読めます。発生アラームがないとき、AlarmNumber は 0 と読めます。
引数 CauseAlarmID にコーズコードアラーム ID が返ります。通常アラームが発生すると、教示操作盤の一番上の行にアラームが表示されますが、アラームによっては、教示操作盤の上から 2 行目にもアラームが表示される事があります。この時 2 行目に表示されているのがコーズコードです。
ここでは、コーズコードのアラーム ID を読むことが出来ます。コーズコードの無いアラームの場合は 0 が読めます。
引数 CauseAlarmNumber にアラーム番号が返ります。コーズコードのアラーム番号を読むことが出来ます。コーズコードの無いアラームの場合は 0 が読めます。
引数 Severity にアラーム重度が返ります。下の表のアラーム重度を表す数値を読むことが出来ます。

NONE	128
WARN	0, 4
PAUSE.L	2
PAUSE.G	34
STOP.L	6
STOP.G	38
SERVO	54
ABORT.L	11
ABORT.G	43
SERVO2	58
SYSTEM	123

引数 Year, Month, Day, Hour, Minute, Second にアラームが発生した日時（24 時制）が返ります。

引数 AlarmMessage にアラームメッセージが返ります。「ｻｰﾎﾞ-001」のようなアラームコードの部分の文字を含む、教示操作盤の一番上の行に表示されている内容がそのまま読めます。（全角のアラームメッセージに対応しています。）

引数 CauseAlarmMessage にコーズコードアラームメッセージが返ります。（全角のアラームメッセージに対応しています。）
引数 SeverityMessage にアラーム重度を示す WARN 等の文字が返ります。

成功すると True, 失敗すると False が返ります。

5.15 FRRJIF.DataString

ロボット R-30iA 7DA7/13 版以降でのみ使用可能です。

(*)取得、設定できる最大文字数は 80 文字です。文字数が 80 文字より多い場合、文字列の 80 文字目までを取得、設定できます。

5.15.1 メソッド一覧

Valid, GetValue, SetValue, Update, Reset

5.15.1.1 Valid－オブジェクトの有効性チェック

VB:	Property Valid As Boolean（読み取り専用）
VC++:	BOOL get_Valid();
C#	bool Valid { get; }

オブジェクトが有効なとき True, 無効なとき False が返ります。

5.15.1.2 GetValue－文字列データを参照

VB:	Function GetValue(ByVal Index As Integer, ByRef Value As String) As Boolean
VC++:	BOOL GetValue(long Index, String* Value, System.Text.Encoding encode);
C#	bool GetValue(int *Index*, ref string *Value*)

引数 Index に参照したい文字列データ（文字列レジスタまたはコメント）のインデックスを指定します。Value に値が返ります。

成功すると True, 失敗すると False が返ります。

5.15.1.3 SetValue－文字列データの値をバッファに格納

VB:	Function SetValue(ByVal Index As Integer, ByVal Value As String) As Boolean
VC++:	BOOL SetValue(long Index, String Value);
C#	bool SetValue(int *Index*, string *Value*)

引数 Index に設定したい文字列データ（文字列レジスタまたはコメント）のインデックスを指定します。Value に設定したい値を指定します。SetValue を呼び出しても、すぐにロボットにデータが書き込まれるわけではありません。バッファに書き込まれるだけです。SetValue を複数回読んで一連の文字列データを設定した後、Update メソッドを呼んだときにバッファ内のデータがまとめてロボットにデータが転送されます。

成功すると True, 失敗すると False が返ります。

5.15.1.4 Update－バッファに格納された文字列データの値をロボットへ転送

VB:	Function Update() As Boolean
VC++:	BOOL Update();
C#	bool Update()

バッファに格納されたデータをロボットへ転送します。SetValue で格納された最も小さいインデックスと最も大きいインデックスの間のデータをまとめてロボットに書き込みます。書き込み後、バッファはリセットされます。

成功すると True, 失敗すると False が返ります。

5.15.1.5 Reset－バッファをリセットします

VB:	Sub Reset( )
VC++:	void Reset();
C#	void Reset()

5.16 FRRJIF.DataFlag

5.16.1 メソッド一覧

Valid, GetValue, SetValues

5.16.1.1 Valid－オブジェクトの有効性チェック

VB:	Property Valid As Boolean（読み取り専用）
VC++:	BOOL get_Valid();
C#	bool Valid { get; }

オブジェクトが有効なとき True, 無効なとき False が返ります。

5.16.1.2 GetValue－フラグの値を参照

VB:	Function GetValue(ByVal Index As Integer, ByRef Value As Short) As Boolean
VC++:	BOOL GetValue(long Index, short* value);
C#	bool GetValue(int *Index*, ref short *Value*)

引数 Index に参照したいフラグのインデックスを指定します。Value に値が返ります。

成功すると True, 失敗すると False が返ります。

5.16.1.3 SetValues－フラグの値を設定

VB:	Function SetValues(ByVal Index As Integer, ByRef Value As System.Array, ByVal Count As Integer) As Boolean
VC++:	BOOL SetValues(long Index, array<short>^ Value, long Count);
C#	bool SetValues(int *Index*, object *Value*, int *Count*)

一連のフラグを設定します。引数 Index に参照したいフラグの最初のインデックスを指定します。Value に値を指定します。Count に設定するフラグの個数を指定します。

成功すると True, 失敗すると False が返ります。

5.17 FRRJIF.VirtualRobotSafetyIOHelper

バーチャルロボット R-30iB Plus 7DF1 版以降でのみ使用可能です。R-50iA では使用できません。

(*)ロボット実機との接続時は動作しないため、実機では使用しないでください。バーチャルロボットとの接続時のみ使用可能です。

(*)安全 I/O を使用するには、適切なソフトウェアオプションがオーダされたバーチャルロボットを使用する必要があります。

5.17.1 メソッド一覧

AddSpiTo, AddCsiTo

5.17.1.1 AddSpiTo－DataTable に SPI（安全周辺機器入力）を登録

VB:	Function AddSpiTo(ByVal DataTable As FRRJIf.DataTable, ByVal StartIndex As Integer, ByVal EndIndex As Integer) As FRRJIf.VirtualRobotSpi
VC++:	bridgeVirtualRobotSpi^ AddSpiTo(FRRJIf.DataTable DataTable, int StartIndex, int EndIndex);
C#	FRRJIf.VirtualRobotSpi AddSpiTo(FRRJIf.DataTable *DataTable*, int *StartIndex*, int *EndIndex*)

成功すると VirtualRobotSpi オブジェクト， 失敗すると Nothing が返ります。VirtualRobotSpi オブジェクトから SPI を参照することができます。

＜例＞ SPI[1]から SPI[5]まで登録する場合
Dim objVirtualRobotSpi As VirtualRobotSpi

Set objVirtualRobotSpi = VirtualRobotSafetyIOHelper.AddSpiTo(objCore.DataTable, 1, 5)

(*) 接続されたバーチャルロボットが対応している範囲で、最大 SPI[128]まで使用できます。対応範囲はソフト版数やオーダされているソフトウェアオプション、各種設定などに依存します。

5.17.1.2 AddCsiTo－DataTable に CSI（CIP セイフティ入力）を登録

VB:	Function AddCsiTo(ByVal DataTable As FRRJIf.DataTable, ByVal StartIndex As Integer, ByVal EndIndex As Integer) As FRRJIf.VirtualRobotCsi
VC++:	bridgeVirtualRobotCsi^ AddCsiTo(FRRJIf.DataTable DataTable, int StartIndex, int EndIndex);
C#	FRRJIf.VirtualRobotCsi AddCsiTo(FRRJIf.DataTable *DataTable*, int *StartIndex*, int *EndIndex*)

成功すると VirtualRobotCsi オブジェクト， 失敗すると Nothing が返ります。VirtualRobotCsi オブジェクトから CSI を参照することができます。

＜例＞ CSI[1]から CSI[5]まで登録する場合
Dim objVirtualRobotCsi As VirtualRobotCsi

Set objVirtualRobotCsi = VirtualRobotSafetyIOHelper.AddCsiTo(objCore.DataTable, 1, 5)

(*) 接続されたバーチャルロボットが対応している範囲で、最大 CSI[64]まで使用できます。対応範囲はソフト版数やオーダされているソフトウェアオプション、各種設定などに依存します。

5.18 FRRJIF.VirtualRobotSpi

バーチャルロボット R-30iB Plus 7DF1 版以降でのみ使用可能です。R-50iA では使用できません。

(*)ロボット実機との接続時は動作しないため、実機では使用しないでください。バーチャルロボットとの接続時のみ使用可能です。

(*)接続されたバーチャルロボットが対応している範囲で、最大 SPI[128]まで参照、設定できます。対応範囲はソフト版数やオーダされているソフトウェアオプションの構成、増設安全 I/O ボード、各種設定などに依存します。

5.18.1 メソッド一覧

Valid, Read, Write

5.18.1.1 Valid－オブジェクトの有効性チェック

VB:	Property Valid As Boolean（読み取り専用）
VC++:	BOOL get_Valid();
C#	bool Valid { get; }

オブジェクトが有効なとき True, 無効なとき False が返ります。

5.18.1.2 Read－バーチャルロボットの SPI（安全周辺機器入力）を参照

VB:	Function Read(ByVal Index As Integer, ByRef Buffer As System.Array, ByVal Count As Integer) As Boolean
VC++:	long Read(short Index, array<short>^ Buffer, int Cnt);
C#	bool Read(int *Index*, System.Array *Buffer*, int *Count*)

一連の SPI を 1 度に参照することができます。
引数に参照する SPI の最初のインデックス、参照する個数分の Integer(または Long)の配列、参照する個数を指定します。

Visual Basic、C#の場合、成功すると True, 失敗すると False が返ります。Visual C++の場合、成功すると 1, 失敗すると 0 が返ります。

5.18.1.3 Write－バーチャルロボットの SPI（安全周辺機器入力）を設定

VB:	Function Write(ByVal Index As Integer, ByRef Buffer As System.Array, ByVal Count As Integer) As Boolean
VC++:	long Write(short Index, array<short>^ Buffer, int Cnt);
C#	bool Write(int *Index*, System.Array *Buffer*, int *Count*)

一連の SPI を 1 度に設定することができます。
引数に参照する SPI の最初のインデックス、参照する個数分の Integer(または Long)の配列、設定する個数を指定します。

Visual Basic、C#の場合、成功すると True, 失敗すると False が返ります。Visual C++の場合、成功すると 1, 失敗すると 0 が返ります。

(*)最後に DataTable を Refresh してから SPI の値が変更されている場合は、設定する前に必ず DataTable の Refresh を呼んでください。

(*)ばらばらに複数回に分けて設定するより、1 度にまとめて設定したほうが短時間で行うことができます。

5.19 FRRJIF.VirtualRobotCsi

バーチャルロボット R-30iB Plus 7DF1 版以降でのみ使用可能です。

(*)ロボット実機との接続時は動作しないため、実機では使用しないでください。バーチャルロボットとの接続時のみ使用可能です。

(*)バーチャルロボットが対応している範囲で、最大 CSI[64]まで参照、設定できます。対応範囲はソフト版数やオーダされているソフトウェアオプション、各種設定などに依存します。

5.19.1 メソッド一覧

Valid, Read, Write

5.19.1.1 Valid－オブジェクトの有効性チェック

VB:	Property Valid As Boolean（読み取り専用）
VC++:	BOOL get_Valid();
C#	bool Valid { get; }

オブジェクトが有効なとき True, 無効なとき False が返ります。

5.19.1.2 Read－バーチャルロボットの CSI（CIP セイフティ入力）を参照

VB:	Function Read(ByVal Index As Integer, ByRef Buffer As System.Array, ByVal Count As Integer) As Boolean
VC++:	long Read(short Index, array<short>^ Buffer, int Cnt);
C#	bool Read(int *Index*, System.Array *Buffer*, int *Count*)

一連の CSI を 1 度に参照することができます。
引数に参照する CSI の最初のインデックス、参照する個数分の Integer(または Long)の配列、参照する個数を指定します。

Visual Basic、C#の場合、成功すると True, 失敗すると False が返ります。Visual C++の場合、成功すると 1, 失敗すると 0 が返ります。

5.19.1.3 Write－バーチャルロボットの CSI（CIP セイフティ入力）を設定

VB:	Function Write(ByVal Index As Integer, ByRef Buffer As System.Array, ByVal Count As Integer) As Boolean
VC++:	long Write(short Index, array<short>^ Buffer, int Cnt);
C#	bool Write(int *Index*, System.Array *Buffer*, int *Count*)

一連の CSI を 1 度に設定することができます。
引数に参照する CSI の最初のインデックス、参照する個数分の Integer(または Long)の配列、設定する個数を指定します。

Visual Basic、C#の場合、成功すると True, 失敗すると False が返ります。Visual C++の場合、成功すると 1, 失敗すると 0 が返ります。

(*)最後に DataTable を Refresh してから CSI の値が変更されている場合は、設定する前に必ず DataTable の Refresh を呼んでください。

(*)ばらばらに複数回に分けて設定するより、1 度にまとめて設定したほうが短時間で行うことができます。

6 ROBOGUIDE との接続

FANUC のロボットアニメーションツール ROBOGUIDE を使用すると PC で仮想的なロボット（バーチャルロボット）が使用できます。ロボットインタフェースは、ロボット実機だけでなく、バーチャルロボットに対しても使用できます。

ロボットの実機が使えない場合でも、バーチャルロボットを使えば、ロボットインタフェースを使用したアプリケーションの開発、テストができます。

6.1 動作環境

ROBOGUIDE V7 Rev.F (7N07/06)版以降が必要です。また ROBOGUIDE 上のバーチャルロボット R-30iA 7DA5/15, R-30iA 7DA7/13 版以降に対して動作します。

6.2 接続先バーチャルロボットの IP アドレス

接続先のバーチャルロボットの IP アドレスを確認するには、そのロボットを選択して ROBOGUIDE のメニュー ロボット/規定のブラウザ でブラウザを表示して下さい。通常、インタネットエクスプローラのアドレスに"http://127.0.0.1/" や"http://127.0.0.1:9001/" と表示されます。これはバーチャルロボットの IP アドレスが"127.0.0.1"であることを示します。" 127.0.0.2"など別のアドレスの場合もあります。
"127.0.0.#"で接続できない場合は、その PC の IP アドレスや"localhost"をお試し下さい。

6.3 バーチャルロボットの指定

1 つの ROBOGUIDE のワークセル上に複数のバーチャルロボットを配置することができます。また 1 つの PC 上で複数の ROBOGUIDE を使用することができます。そのため 1 つの PC に複数のバーチャルロボットが存在することになります。
ロボットインタフェースで作成したアプリケーションの接続先ロボットは 1 つです。1 つの PC 上の複数のバーチャルロボット中から 1 つのバーチャルロボットを指定する必要があります。バーチャルロボットをループバックアドレスで起動している場合は、各バーチャルロボットに異なるループバックアドレスが割り振られるため、IP アドレスでバーチャルロボットを指定します。そうでない場合は、各バーチャルロボットに異なるポート番号が割り振られるため、ポート番号でバーチャルロボットを指定します。

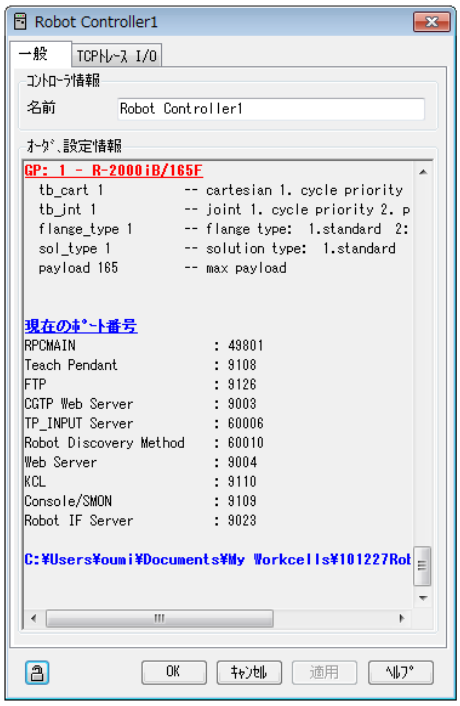
(*) バーチャルロボットをループバックアドレスで起動するには ROBOGUIDE V9 Rev.T（7N09/22）版以降が必要です。ループバックアドレスは 127.0.0.#（127.0.0.2 など）として表される IP アドレスです。

(*) ポート番号を指定するのはバーチャルロボットに対してだけです。実ロボットに対しては既定のポート番号を使用するため、指定する必要はありません。

6.3.1 接続先バーチャルロボットのポート番号の取得

接続先のバーチャルロボットのポート番号はバーチャルロボットが起動するたびに変更になる可能性があります。そのため接続前に毎回ポート番号を確認する必要があります。

ROBOGUIDE で接続先のロボットコントローラのプロパティページを開きます。オーダ、設定情報の中に"現在のポート番号"の情報があります。ここで"Robot IF Server"のポート番号がロボットインタフェースでの接続先ロボットのポート番号です。次の例では 9023 が接続先バーチャルロボットのポート番号です。



(*) ROBOGUIDE またはバーチャルロボットのバージョンが古い場合、上記のポート番号は表示されません。ROBOGUIDE のアップデートが必要です。
(*) バーチャルロボットが R650 北米専用設定を選択していて、R553 タッチパネル通信機能 オプションがない場合、上記のポート番号は表示されません。バーチャルロボットに R553 タッチパネル通信機能 オプションを追加して下さい。

もう 1 つのポート番号取得方法はワークセルフォルダ内のファイルから読み取る方法です。バーチャルロボットは生成時に ID が割り振られます。セルブラウザのロボットコントローラノードは"C: ID - 名前”のフォーマットで表示されるので、ここから ID がわかります。例えばセルブラウザのロボットコントローラノードの表示が "C: 1 - Left Robot" なら ID は 1 です。
ROBOGUIDE でバーチャルロボットを起動するとワークセルフォルダの下に Robot_ID¥services.txt というファイルが生成されます。（Robot_ID は ID が 1 なら Robot_1 です）services.txt はポート番号の情報を保持しています。
services.txt を読み込み"Robot IF Server"で始まる行からポート番号を取得できます。

6.3.2 ポート番号の設定

取得したポート番号を接続前に FRRJIf.Core オブジェクトの PortNumber プロパティに設定します。

＜サンプルコード - "ホスト名:ポート番号"の形式で指定＞

```
'get host name
If HostName = "" Then
    strHost = GetSetting(cnstApp, cnstSection, "HostName")
    strHost = InputBox("Please input robot host name", , strHost)
    If strHost = "" Then
        End
    End If
    SaveSetting cnstApp, cnstSection, "HostName", strHost
    HostName = strHost
Else
    strHost = HostName
End If

'check port number
```

```
Dim strArray() As String
If InStr(strHost, ":") > 0 Then
    strArray = Split(strHost, ":")
    strHost = strArray(0)
    If IsNumeric(strArray(1)) Then
        mobjCore.PortNumber = CInt(strArray(1))
    End If
End If

'connect
blnRes = mobjCore.Connect(strHost)
If blnRes = False Then
    msubDisconnected
Else
    msubConnected
End If
```

7 自動運転

ロボットインタフェースにはロボットのプログラムを起動する専用機能はありません。しかし、ロボットインタフェースを使用してフラグ(F[*])を設定することで、ネットワーク経由でプログラムを起動できます。また、プログラム番号選択（PNS）機能やロボット起動要求（RSR）機能を使うことで、起動するプログラムを選択できます。

ロボットインタフェースを使用したプログラム起動の一例として、RSR を使用する場合の設定手順を説明します。なお、ロボットの各設定の詳細は、お使いのロボット制御装置の取扱説明書をご参照下さい。

7.1 ロボットの設定

先ず、周辺機器 I/O（UOP）の割り付けを設定します。システム設定画面の「UOP 自動割り付け」を「全て」に設定します。そして、I/O 画面から UI をフラグ（ラック:34、スロット:1）に割り付けます。割り付けたフラグの値を変更すると、対応する UI の値が連動することをご確認下さい。

次に、起動するプログラムを作成し、RSR の設定を行います。RSR の設定については、ロボット制御装置の取扱説明書を参照し、設定して下さい。ロボット起動要求信号（RSR1~8）に対応するフラグの値を操作して、任意のプログラムが起動することをご確認下さい。

(*) フラグからプログラムを起動できるよう設定した場合、意図せずフラグが設定されてプログラムが起動する場合があります。安全には十分お気をつけ下さい。

(*) ご使用環境によって、I/O の割り付け等の設定は異なります。上記手順は一例なので、ご使用環境に合わせた設定を行って下さい。

(*) プログラムを起動するには、リモート条件や動作可能条件が成立している必要があります。詳細はロボット制御装置の取扱説明書をご参照下さい。

(*) RSR 以外の機能を使った場合であっても、フラグからプログラムを起動できるように設定することで、ロボットインタフェースを使用してプログラムを起動できます。

7.2 PC 用プログラムの作成

割り付けたフラグを設定する PC 用プログラムを作成します。ロボット起動要求信号（RSR1~8）に対応するフラグの立ち上がりでプログラムが起動するため、一度オフに設定してからオンに設定する必要があります。
ロボットインタフェースでフラグを設定するには、DataFlag.SetValues を使用します。

(*) PC 用プログラムの内容によっては、意図せずプログラムが起動する場合があります。安全には十分お気をつけ下さい。

(*) ご使用環境やロボットの設定によって、プログラム起動に必要なフラグの設定は異なります。ご使用環境に合わせてフラグを設定して下さい。

8 通信エラー時の対処について

- ロボットインタフェース以外でロボットと PC が通信できることをご確認下さい。例えば Internet Explorer でロボットのアドレスを指定してロボットのホームページを開きます。ロボットのホームページが開けない場合、イーサネット通信自体に問題があります。IP アドレスの設定、ケーブル、ハブなどをご確認下さい。

- ロボットのオプションとして R650 北米専用設定を選択されているときは、R553 タッチパネル通信機能 オプションが必要です。北米のロボットの場合は、この点に注意が必要です。（R651 標準設定の場合、オプションは必要ありません）

- ロボットインタフェースのサンプルプログラムで動作を確認して下さい。

- タイムアウトの設定値はできるだけ余裕のある値に設定して下さい。タイムアウトの設定が小さすぎるとエラーになることがあります。常に一定の時間で通信できるとは限らないからです。

- 可能ならば通信エラーの際に、いったん通信を切ってロボットインタフェースのオブジェクトをすべて破棄から再接続して通信をリトライするようにするとより堅牢になります。

- ロボットインタフェースは通信エラー発生時にOutputDebugString APIで以下のようなエラーメッセージを出力します。

  208: [RobotIF.acc_socket] Time Out at line 369, 0, 501, 10000
  208: [RobotIF.snpx_rw] Error at line 290

  メッセージ中に"Time Out"が含まれている場合は、タイムアウトの設定値を大きくすることで通信エラーを回避できる可能性があります。

  (*) メッセージを補足するツールdbmon.exeがインストールCD-ROMのToolsフォルダにあります。PCのローカルディスクにコピーしてお使い下さい。

- ロボット側でアラーム「PRIO-096 SNPX接続に空きがありません」が発生し、接続に失敗する場合は、ロボットの最大同時接続数を超えて通信をConnectしようとしている可能性があります。ロボットと各アプリケーションとの接続状況をご確認下さい。また、ロボットに不要な接続が存在する可能性があります。通信が不要になった場合は適切に切断し、切断が成功したことをご確認下さい。ロボットと再接続する際は、FRRJIFのオブジェクトをすべて破棄する必要があります。

  (*) Zero Down Time、FIELD system、MT-LINKiにロボットを接続している場合、ロボットの同時接続数が別途制限される可能性があります。その他の弊社製アプリケーションをロボットに接続している場合も同様です。

- 通信エラー発生時には一般に配布されている Windows 用のネットワーク･プロトコル･アナライザ ソフトウェアにより状況が把握できることがあります。

- ロボットの動作中にだけエラーが発生する場合は、ロボットの動作によるノイズの影響が考えられます。ノイズ対策を行って下さい。

- ロボットインタフェースでは、ロボットのイーサネットポート CD38C による通信ができません。CD38A、もしくは CD38B をお使い下さい。

- 次の文字列値のレジストリを設定することでログを取ることができます。

  [HKEY_CURRENT_USER¥Software¥VB and VBA Program Settings¥FRRJIF¥Setting]
  "DebugOut"="True"

  ログの出力先はレジストリ
  [HKEY_CURRENT_USER¥Software¥VB and VBA Program Settings¥FRRJIF¥Setting]
  "DebugOutPath"

  で設定できます。（設定しない場合はインストール先（通常 C:¥Program Files¥FANUC¥FRRJIf））
  ログは log*yymmddhhmmss*.txt というファイルで書き込まれます。（ログの書き込み先によっては書き込むために管理者として実行する必要がある場合があります）

  FRRJIF.Core オブジェクトの DebugLog プロパティに True を設定するとさらに詳しい情報がログされます。

  (*) ログは必要な場合のみ有効にして下さい。パフォーマンスが低下する可能性があります。