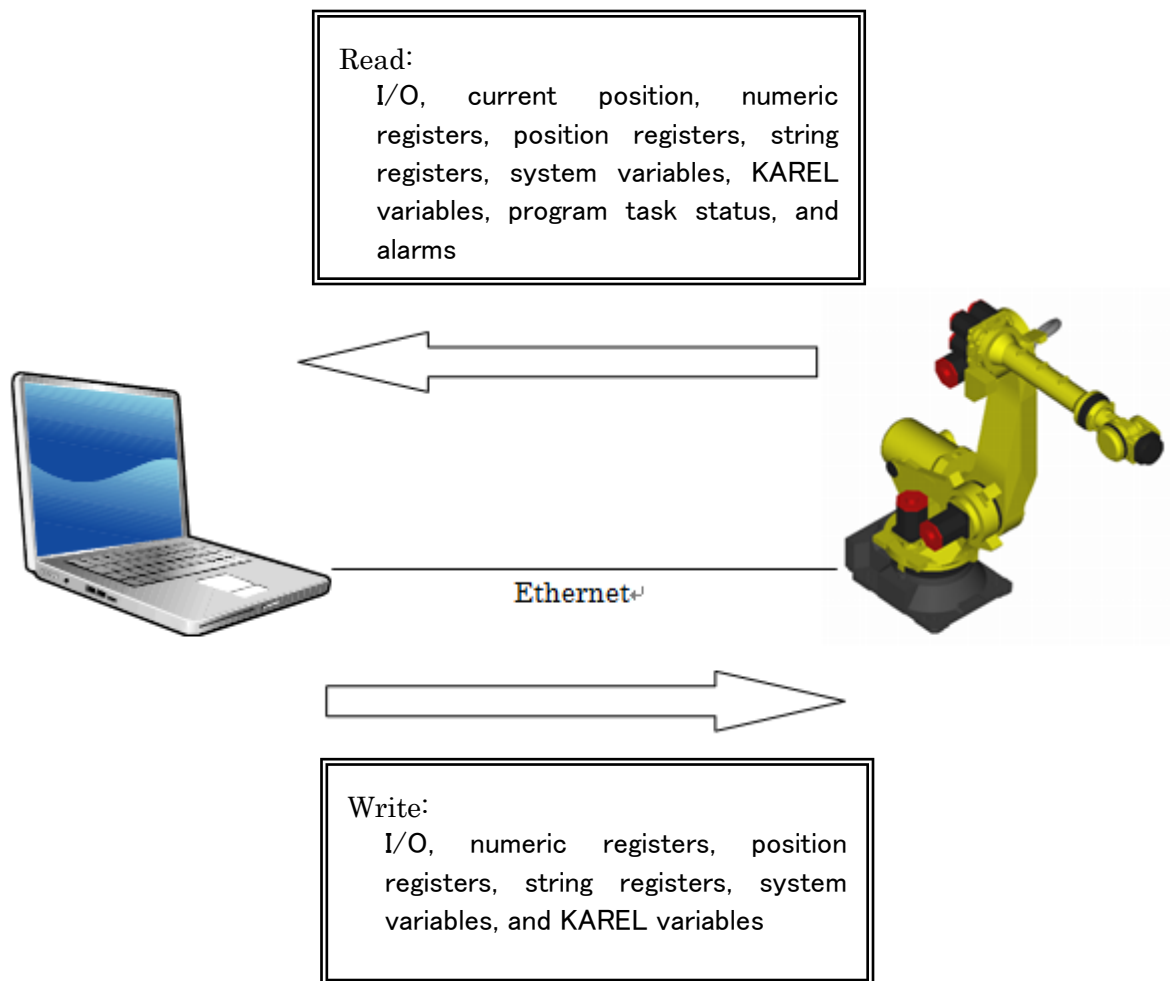


# 1 Overview

FANUC Robot Interface (FRRJIF.DLL) is Windows software module to read/write robot data with robot over Ethernet.



(\*) FANUC Robot Interface does not support file transferring. Please use FTP or HTTP for it.

## 1.Overview

---

FANUC Robot Interface is an ActiveX module. This document describes software interface of it. FRRJIF.Core object provide methods to read/write robot I/O. It is possible to read/write area of integrated PMC.

For accessing current position, position registers, string registers, comments, system variables, KAREL variables, program status and alarm history, it is necessary to add needed data to DataTable object (FRRJIF.Core.DataTable) at first. FRRJIF reads all of DataTable object at a time. It is called 'Refresh'. Read values are kept until next refresh.

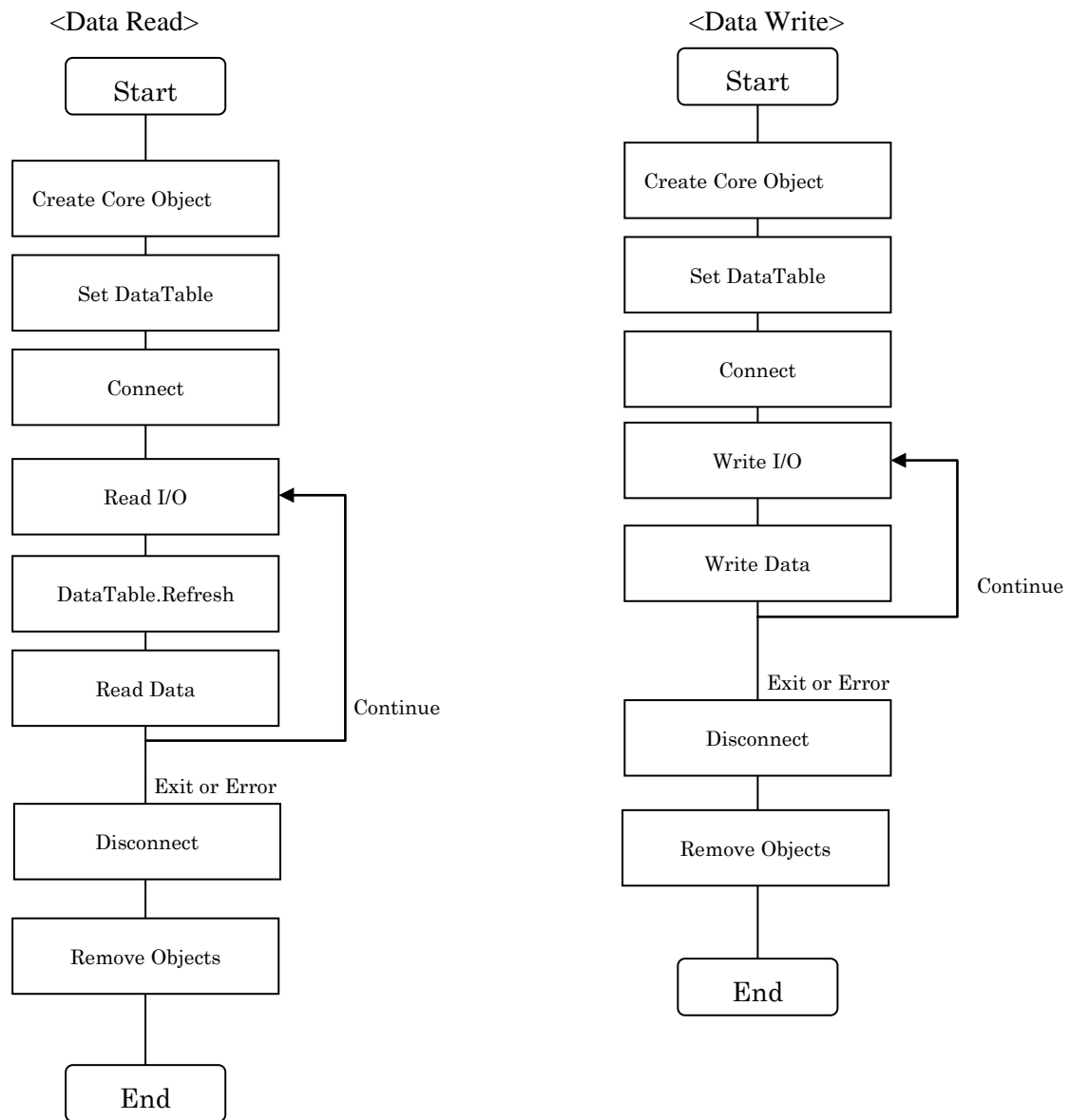
(\*) DataTable for reducing read data by accessing data at a time.

A robot can have one connection with FANUC Robot Interface. A robot cannot have multiple connections between multiple PCs. A robot cannot have multiples connections between multiple applications on a PC. **If multiple PCs or multiple applications access to a robot with FANUC Robot Interface, it is not guaranteed.**

You need to purchase one FANUC Robot Interface (A08B-9410-J575) to one development PC. **For additional PCs that are not for development, you do not need to purchase additional FANUC Robot Interface. License registration and USB protector is not necessary from V3.0.0.**

FANUC Robot Interface does not have file transfer functions. You can use FTP to transfer files between PC and robot.

## – Basic Flow



(\*) When communication is terminated, communication error or time out occurs, please remove all robot interface objects. It is necessary to create objects in order to start communication again.

(\*) The data reading is a kind of polling procedure for monitoring. It is not possible to get all data transition since robot interface applications can read data periodically.

## 2 Environment

---

OS:

Windows 7(32bit), Windows 7(64bit) (\*1),  
Windows 8.1(32bit), Windows 8.1(64bit) (\*1)  
Windows 10(32bit), Windows 10(64bit) (\*1)

Development language:

Microsoft Visual Basic 6.0 + Service pack6 (\*2)  
Microsoft Visual Basic 2008 + Service pack1  
Microsoft Visual Basic 2010, 2012, 2013, 2015  
Microsoft Visual C++ 6.0 + Service pack6 (\*2)  
Microsoft Visual C++ 2008 + Service pack1  
Microsoft Visual C++ 2010, 2012, 2013, 2015  
Microsoft Visual C# 2008 + Service pack1  
Microsoft Visual C# 2010, 2012, 2013, 2015

Robot Controllers:

R-J3iB 7D80/45 or later  
R-J3iB 7D81/09 or later  
R-J3iB 7D82/01 or later  
R-J3iB Mate 7D91/01 or later  
R-30iA, R-30iA Mate All Versions (\*3)  
R-30iB All Versions (\*3)  
R-30iB Plus All Versions (\*3)

ROBOGUIDE: (\*4)

ROBOGUIDE V7 Rev.F(7N06) or later  
Virtual Robot R-30iA 7DA5/15 or later  
Virtual Robot R-30iA 7DA7/13 or later  
Virtual Robot R-30iB All Versions  
Virtual Robot R-30iB Plus All Versions

PC must communicate with robot over Ethernet. (Noise must not stop the communication. Correct security settings are needed. )

(\*1) Even if you use Windows x64, you need to develop 32bit application with robot interface, not 64bit application. You need to specify your project as x86. And you need to disable the Visual studio hosting process in debug setting. Windows RT is not supported. Windows Store Application is not supported. Windows Universal Application is not supported.

(\*2) You can develop robot interface applications on only Windows XP for Visual Basic 6.0 and Visual C++ 6.0.

(\*3) If R650 FRA Params is selected, R553 "HMI Device (SNPX)" is needed. If R651 FRL Params is selected, no option is needed.

(\*4) Please refer "6. Connect To ROBOGUIDE".

(\*) Please refer "Ethernet Function Operator's Manual" of robot controller.

(\*) Windows, Visual Basic, Visual C++, Visual C# is a registered trademark of Microsoft Cooperation in the United States and other countries.

# 3 Limitations

---

When you update FANUC Robot Interface, you may need to re-compile your application that uses FANUC Robot Interface.

Robot interface communication response time is not guaranteed. Communication environment, robot status and PC status may affect response time. Enough long time out value (FRRJIF.Core.TimeOutValue) is recommended.

If noise affects the communication, robot interface may not work. For example, robot interface might not work correctly during robot working. You need to shield Ethernet cables or set apart cables from noise source. (Please refer Appendix "Network design and performance", "Cable connection" of Ethernet function operator's manual.)

When communication error or time out occur, please disconnect the connection and delete all FRRJIF objects. If you need to connect again, please re-create objects.

When amount of data or the number of send data is large, it may be necessary long time. If you need fast communication, please reduce amount of data or the number of send data.

Reading nonexistent data (for example, reading nonexistent system variable) does not return error. Read data value is indefinite. Writing nonexistent data does not return error. Such writings are ignored.

**Robot interface does not work correctly on multiple thread programming.**

# **4 New Features**

---

## **4.1 V3.0.0 New features**

---

- Supported R-30iB Plus.
- License registration and USB protector is not necessary from V3.0.0.

## **4.2 V2.0.8 New features**

---

- Supported Windows 10 and Visual Studio 2015.
- Supported multiple group position register.  
Please refer FRRJIF.DataTable.AddPosRegMG and FRRJIF.DataPosRegMG.

## **4.3 V2.0.7 New features**

---

- Supported Windows 8.1 and Visual Studio 2013.

## **4.4 V2.0.6 New features**

---

- Supported logging function.  
Please refer "7. Trouble Shooting".

## **4.5 V2.0.5 New features**

---

- Supported Visual Basic 2012, Visual C++ 2012, Visual C# 2012.
- Supported Windows 8 (32bit), Windows 8 (64bit).

## **4.6 V2.0.4 New features**

---

- Supported R-30iB
- Added new data type for FRRJIF.DataTable.AddTask. Please refer FRRJIF.DataTable.AddTask.

## **4.7 V2.0.3 New features**

---

- FRRJIF.Core.DataTable2 method is added.

## **4.8 V2.0.2 New features**

---

- Supported Visual Basic 2010, Visual C++ 2010, Visual C# 2008 and Visual C# 2010.
- Supported Windows 7 (64bit).
- Supported string data (string registers and comments)  
Please refer FRRJIF.DataTable.AddString and FRRJIF.DataString.

## 4.New Features

---

- Supported weld I/O and analog I/O.  
Please refer FRRJIF.Core read I/O and write I/O.
- Supported KAREL variables.  
Please refer FRRJIF.DataTable.AddSysVar.
- FRRJIF.DataTable could have 80 items. This limitation is removed.
- FRRJIF.DataTask.GetValue supported program name that is exceeded 8 characters.
- Supported connection to ROBOGUIDE. Please refer "6. Connect To ROBOGUIDE".

## 4.9 V2.0.1 New features

---

- Supported Windows 7.
- Changed writing I/O  
On previous versions, I/O is written 8 bit at a time. For example, DO[1-8] is written at a time. From V2.0.1, I/O is written 1 bit. For example, it is possible to write DO[1] separately.

## 4.10 V2.0.0 New features

---

- Supported Visual Basic 2008, Visual C++ 6.0 and Visual C++ 2008.
- Added methods for Visual C++. (Please refer each methods.)
- Supported Windows Vista.
- Added NumReg.SetValues method that is to write a series of numeric registers at a time.
- Supported double byte alarm messages.
- Supported new license manager.

## 4.11 V1.3.0 New Features

---

- Added PosRegXyzwpr object and DataTable.AddPosRegXyzwpr method that is to write multiple position registers at a time.
- Added DataTable.AddCurPosUF method that is to read current position with a user frame.



# 5 Object Reference

## 5.1 Symbols

VB	Visual Basic 6.0, Visual Basic 2008, 2010, 2012, 2013, and 2015
VB6	Visual Basic 6.0
VB2008	Visual Basic 2008, 2010, 2012, 2013, and 2015
VC++	Visual C++ 6.0, Visual C++ 2008, 2010, 2012, 2013, and 2015
VC++6	Visual C++ 6.0
VC++2008	Visual C++ 2008, 2010, 2012, 2013, and 2015
C#	Visual C# 2008, 2010, 2012, 2013, and 2015

## 5.2 FRRJIF.Core

### 5.2.1 Methods

- Connect Methods  
Connect, Disconnect, TimeOutValue
- IO Methods  
ReadSDI, ReadSDO, ReadRDI, ReadRDO, ReadSI, ReadSO, ReadUI, ReadUO, ReadGI, ReadGO, WriteSDI, WriteSDO, WriteRDO, WriteSO, WriteUO, WriteGI, WriteGO
- DataTable Methods  
DataTable, DataTable2
- Alarm Methods  
ClearAlarm
- Protect Methods  
ProtectAvailable, ProtectTrialRemainDays, ProtectStatus

#### 5.2.1.1 Connect – Connect to a robot

VB:	Function Connect(ByVal HostName As String) As Boolean
VC++:	BOOL Connect(LPCTSTR HostName);
C#	<code>bool Connect(string HostName)</code>

Specify robot host name (or IP address) as argument.

You need to initialize DataTable before calling this method.

Success returns True, fails returns False.

#### 5.2.1.2 Disconnect – Disconnect from a robot

VB:	Function Disconnect() As Boolean
VC++:	BOOL Disconnect();
C#	<code>bool Disconnect()</code>

## 5.Object Reference

Success returns True, fails returns False.

(\*) When disconnected, please delete all FRRJIF objects. If you need to connect again, please re-create objects.

### 5.2.1.3 TimeOutValue - Time out value

VB6	Property TimeOutValue As Long
VB2008	get_TimeOutValue As Integer set_TimeOutValue(ByVal newValue As Integer) or TimeOutValue property
VC++6	long GetTimeOutValue(); void SetTimeOutValue(long nNewValue);
VC++2008	long get_TimeOutValue(); void put_TimeOutValue(long newValue);
C#	int get_TimeOutValue() void set_TimeOutValue(int newValue) or TimeOutValue property

This is time out value for communication error. Default value is 10000 (msec). Minimum value is 100. If you apply small time out value, then you may encounter time out often. The time out value is discard at deleting Core object. Please specify time out again when Core object is created.

(\*) When timeout, please disconnect and delete all FRRJIF objects. If you need to connect again, please re-create objects.

### 5.2.1.4 ReadSDI, ReadSDO, ReadRDI, ReadRDO, ReadSI, ReadSO, ReadUI, ReadUO, ReadGI, ReadGO – Read I/O

VB6:	Function ReadXXX(Index As Long, Buffer() As Integer, Count As Long) As Boolean
VB2008:	Function ReadXXX(ByVal Index As Integer, ByRef Buffer As System.Array, ByVal Count As Integer) As Boolean
VC++6:	long ILibCore.ReadXXX(short Index, short* Buf, short Cnt);
VC++2008	long CLibCore.ReadXXX(short Index, short * Buf, short Cnt);
C#	bool ReadXXX(int Index, ref System.Array Buffer, int Count)

(XXX is one of SDI, SDO, RDI, RDO, SI, SO, UI or UO)

VB6:	Function ReadXX(Index As Long, Buffer() As Long, Count As Long) As Boolean
VB2008:	Function ReadXX(ByVal Index As Integer, ByRef Buffer As System.Array, ByVal Count As Integer) As Boolean
VC++6:	long ILibCore.ReadXX(short Index, long* Buf, short Cnt);
VC++2008	long CLibCore.ReadXX(short Index, long * Buf, short Cnt);
C#	bool ReadXX(int Index, ref System.Array Buffer, int Count)

(XX is one of GI or GO)

This method is to read a series of I/O.

Specify the first index of referred I/O, integer (or Long) array to store values and the number of referred I/O as arguments.

In integrated PMC R-30iA or before:

- You can read integrated PMC K area (SDO[10001] or later) and R area (SDO[11001] or later) with ReadSDO method. However, you cannot read both K area and R area at a time. You need to read K area and R area separately.
- You can read integrated PMC D area (GO[10001] or later) with ReadGO method. However, you cannot read both normal GO and D area at a time. You need to read normal GO and D area separately.

In R-30iB and R-30iB Plus, the PMC data, SDO[10001-] (K area), SDO[11001-] (R area) and GO[10001-] (D area), cannot be accessed by default. By using the compatibility setup function, the PMC data, K0-K17, R0-R1499 and D0-D2999, can be accessed as the same as the conventional model. However, a part of PMC function is restricted by the compatibility setup function.

To exchange the data between Robot Interface and PMC in R-30iB and R-30iB Plus, please setup PMC internal I/O assignment to assign F of PMC to DO of Robot, or G of PMC to DI of Robot. For example, when the following setting is done, the 32 points signal of DO[1-32] of Robot are assigned to the 4 byte PMC signal address of F0-F3 in 1st path PMC. Robot Interface can access F0-F3 by accessing DO[1-32].

Robot data			Size	Address
1	DO[	1- 32]	4	1:F00000

You can read WI with ReadSDI method. You need to add 8000 to the index. For example, you need to specify 8001 in order to access WI[1].

You can read WSI with ReadSDI method. You need to add 8400 to the index. For example, you need to specify 8401 in order to access WSI[1].

You can read WO with ReadSDO method. You need to add 8000 to the index. For example, you need to specify 8001 in order to access WO[1].

You can read AI with ReadGI method. You need to add 1000 to the index. For example, you need to specify 1001 in order to access AI[1].

You can read AO with ReadGO method. You need to add 1000 to the index. For example, you need to specify 1001 in order to access AO[1].

Success returns True, fails returns False in Visual Basic, C# cases. Success returns 0, fails returns non 0 in Visual C++ cases. Return value of Visual Basic, C# cases is different from Visual C++ cases. (When accessed non-exist I/O, this method returns True. In the case, the values are unknown values.)

<Example> Read SDI[1] to SDI[5]

Dim intBuffer(0 to 4) As Integer

Dim blnResult As Boolean

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

(\*) You had better read a series of I/O at a time to reduce access time.

(\*) Long array is needed to GI, GO.

### 5.2.1.5 WriteSDI, WriteSDO, WriteRDO, WriteSO, WriteUO, WriteGI, WriteGO – Write I/O

VB6:	Function WriteXXX(Index As Long, Buffer() As Integer, Count As Long) As Boolean
VB2008:	Function WriteXXX(ByVal Index As Integer, ByRef Buffer As System.Array, ByVal Count As Integer) As Boolean
VC++6:	long ILibCore.WriteXXX(short Index, short* Buf, short Cnt);
VC++2008	long ClibCore.WriteXXX(short Index, short * Buf, short Cnt);
C#	bool WriteXXX(int Index, ref System.Array Buffer, int Count)

(XXX is one of SDI, SDO, RDO, SO or UO)

VB6:	Function WriteXX(Index As Long, Buffer() As Long, Count As Long) As Boolean
------	---

## 5.Object Reference

VB2008:	Function WriteXX(ByVal Index As Integer, ByRef Buffer As System.Array, ByVal Count As Integer) As Boolean
VC++6:	long ILibCore.WriteXX(short Index, long* Buf, short Cnt);
VC++2008	long CLibCore.WriteXX(short Index, long * Buf, short Cnt)
C#	WriteXX(int Index, ref System.Array Buffer, int Count)

(XX is one of GI or GO)

This method is to write a series of I/O at a time.

Specify the first index of written I/O, integer (or Long) array to store values and the number of referred I/O as arguments.

In integrated PMC R-30iA or before:

- You can write integrated PMC K area (SDO[10001] or later) and R area (SDO[11001] or later) with WriteSDO method. However, you cannot write both K area and R area at a time. You need to write K area and R area separately.

- You can write integrated PMC D area (GO[10001] or later) with ReadGO method. However, you cannot write both normal GO and D area at a time. You need to write normal GO and D area separately.

You can write WO with WriteSDO method. You need to add 8000 to the index. For example, you need to specify 8001 in order to access WO[1].

In R-30iB and R-30iB Plus, the PMC data, SDO[10001-] (K area), SDO[11001-] (R area) and GO[10001-] (D area), cannot be accessed by default. By using the compatibility setup function, the PMC data, K0-K17, R0-R1499 and D0-D2999, can be accessed as the same as the conventional model. However, a part of PMC function is restricted by the compatibility setup function.

To exchange the data between Robot Interface and PMC in R-30iB and R-30iB Plus, please setup PMC internal I/O assignment to assign F of PMC to DO of Robot, or G of PMC to DI of Robot. For example, when the following setting is done, the 32 points signal of DO[1-32] of Robot are assigned to the 4 byte PMC signal address of F0-F3 in 1st path PMC. Robot Interface can access F0-F3 by accessing DO[1-32].

Robot data			Size	Address
1	DO[	1-	32]	4 1:F00000

You can Write AO with WriteGO method. You need to add 1000 to the index. For example, you need to specify 1001 in order to access AO[1].

Success returns True, fails returns False in Visual Basic, C# cases. Success returns 0 or positive number, fails returns negative number in Visual C++ cases. Return value of Visual Basic, C# cases is different from Visual C++ cases. (When accessed non-exist I/O, this method returns True. In the case, the values are unknown values.)

<Example> Set SDO[1] to SDO[5] to On

```
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)
```

(\*) You had better write a series of I/O at a time to reduce access time.

(\*) Long array is needed to GI, GO.

(\*) SDI and GI should be simulated for writing.

### 5.2.1.6 DataTable – Get DataTable object

VB6:	Property DataTable As DataTable
VB2008	get_DataTable() As DataTable or DataTable property
VC++6:	LPDISPATCH GetDataTable();
VC++2008	LPDISPATCH get_DataTable();
C#	<a href="#">FRRJif.DataTable</a> get_DataTable() or DataTable property

This method returns FRRJIF.DataTable object.

### 5.2.1.7 DataTable2 – Get Second DataTable object

VB6:	Property DataTable2 As DataTable
VB2008	get_DataTable2() As DataTable or DataTable2 property
VC++6:	LPDISPATCH GetDataTable2();
VC++2008	LPDISPATCH get_DataTable2();
C#	<a href="#">FRRJif.DataTable</a> get_DataTable2() or DataTable2 property

This method returns FRRJIF.DataTable object.

The first data table from DataTable method and the second data table from DataTable2 method have the same function. It is possible to refresh the tables separately. It allows you to handle two data group that refresh rate is different

(\*) It is necessary to get DataTable2 after all AddXXX method for the first data table finish. When you get second data table, you cannot call AddXXX for the first data table.

### 5.2.1.8 ClearAlarm – Clear alarms

VB:	Function ClearAlarm() As Boolean
VC++:	BOOL ClearAlarm(long vlngType);
C#	<a href="#">bool</a> ClearAlarm( <a href="#">int</a> vlngType)

Clear alarm history.

For Visual C++, pass 0 as vlngType. Success returns True, fails returns False.

### 5.2.1.9 ProtectAvailable – Get protect availability

VB6:	Property ProtectAvailable As Boolean
VB2008	get_ProtectAvailable() As Boolean or ProtectAvaialable property
VC++6:	BOOL GetProtectAvailable();
VC++2008	BOOL get_ProtectAvailable();
C#	<a href="#">bool</a> get_ProtectAvailable() or ProtectAvaialable property

This property is always True.

## 5.Object Reference

---

### 5.2.1.10 ProtectTrialRemainDays – Get remain days

---

VB6:	Property ProtectTrialRemainDays As Long
VB2008:	get_ProtectTrialRemainDays() As Integer or ProtectTrialRemainDays property
VC++6:	long GetProtectTrialRemainDays();
VC++2008	long get_ProtectTrialRemainDays();
8	
C#	int get_ProtectTrialRemainDays() or ProtectTrialRemainDays property

This property returns 0.

### 5.2.1.11 ProtectStatus – Get protect status string

---

VB6:	Property ProtectStatus As String
VB2008:	get_ProtectStatus() As String or ProtectStatus property
VC++6:	CString GetProtectStatus();
VC++2008	CString get_ProtectStatus();
C#	string get_ProtectStatus() or ProtectStatus property

This property returns “REGISTERED”.

Remark

(\*) When communication fails like timeout, please disconnect and delete all FRRJIF objects. If you need to connect again, please re-create objects.

## 5.3 FRRJIF.DataTable

---

### 5.3.1 Methods

---

- Data register methods  
Clear, AddCurPos, AddCurPosUF, AddNumReg, AddPosReg, AddPosRegXyzwpr, AddPosRegMG, AddSysVar, AddSysVarPos, AddTask, AddAlarm, AddString
- Data read method  
Refresh

#### 5.3.1.1 Clear – Clear DataTable

---

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

You must not call Clear method after connecting.  
Success returns True, fails returns False.

#### 5.3.1.2 AddCurPos - Register ‘Current position’ data to DataTable

---

VB6:	Function AddCurPos(DataType As FRIF_DATA_TYPE, Group As Long) As DataCurPos
------	---

VB2008:	Function AddCurPos(ByVal DataType As FRRJIf.FRIF_DATA_TYPE, ByVal Group As Integer) As FRRJIf.DataCurPos
VC++:	LPDISPATCH AddCurPos(long DataType, long Group);
C#	<a href="#">FRRJIf.DataCurPos</a> AddCurPos( <a href="#">FRRJIf.FRIF_DATA_TYPE</a> DataType, <a href="#">int</a> Group)

Specify DataType(CURPOS=3) and motion group number as arguments.

When it succeeds, this method returns DataCurPos object. When it fails, this method returns Nothing. DataCurPos object is to read current world position.

<Example> Register current position of motion group 1  
Dim objDataCurPos As DataCurPos

Set objDataCurPos = objCore.DataTable.AddCurPos(CURPOS, 1)

### 5.3.1.3 AddCurPosUF - Register 'Current position with user frame' data to DataTable

VB6:	Function AddCurPosUF(DataType As FRIF_DATA_TYPE, Group As Long, UF As Long) As DataCurPos
VB2008:	Function AddCurPosUF(ByVal DataType As FRRJIf.FRIF_DATA_TYPE, ByVal Group As Integer, ByVal UF As Integer) As FRRJIf.DataCurPos
VC++:	LPDISPATCH AddCurPosUF(long DataType, long Group, long UF);
C#	<a href="#">FRRJIf.DataCurPos</a> AddCurPosUF( <a href="#">FRRJIf.FRIF_DATA_TYPE</a> DataType, <a href="#">int</a> Group, <a href="#">int</a> UF)

Specify DataType(CURPOS=3), motion group number and user frame number as arguments.

When it succeeds, this method returns DataCurPos object. When it fails, this method returns Nothing. DataCurPos object is to read current position with specified user frame number. The user frame number zero means world position. The following user frame number means current frame number.

7DA4 (V7.40) before	UF=15
7DA4/P01 – 7DA4/P11	UF=63
7DA4/P12 or later	UF=-1

<Example> Register current position of motion group 1 with user frame 1.  
Dim objDataCurPos As DataCurPos

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

### 5.3.1.4 AddNumReg - Register numeric registers to DataTable

VB6:	Function AddNumReg(DataType As FRIF_DATA_TYPE, StartIndex As Long, EndIndex As Long) As DataNumReg
VB2008:	Function AddNumReg(ByVal DataType As FRRJIf.FRIF_DATA_TYPE, ByVal StartIndex As Integer, ByVal EndIndex As Integer) As FRRJIf.DataNumReg
VC++:	LPDISPATCH AddNumReg(long DataType, long StartIndex, long EndIndex);
C#	<a href="#">FRRJIf.DataNumReg</a> AddNumReg( <a href="#">FRRJIf.FRIF_DATA_TYPE</a> DataType, <a href="#">int</a> StartIndex, <a href="#">int</a> EndIndex)

Specify DataType(NUMREG\_INT=0 for integer case, NUMREG\_REAL=1 for float case) , the first and end index of numeric registers.



## 5.Object Reference

When NUMREG\_INT is specified, float values are translated to integer. When NUMREG\_REAL is specified, integer values are translated to float.

When it succeeds, this method returns DataNumReg object. When it fails, this method returns Nothing. DataNumReg object is to read/write numeric register values.

<Example> Register R[1] to R[15]  
Dim objDataNumReg As DataNumReg

Set objDataNumReg = objCore.DataTable.AddNumReg(NUMREG\_INT, 1, 15)

### 5.3.1.5 AddPosReg - Register position registers to DataTable

VB6:	Function AddPosReg(DataType As FRIF_DATA_TYPE, Group As Long, StartIndex As Long, EndIndex As Long) As DataPosReg
VB2008:	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++:	LPDISPATCH AddPosReg(long DataType, long Group, long StartIndex, long EndIndex);
C#	<a href="#">FRRJIf.DataPosReg</a> AddPosReg( <a href="#">FRRJIf.FRIF_DATA_TYPE</a> DataType, <a href="#">int</a> Group, <a href="#">int</a> StartIndex, <a href="#">int</a> EndIndex)

Specify DataType(POSREG=2) , motion group number, the first and end index of position registers.

When it succeeds, this method returns DataPosReg object. When it fails, this method returns Nothing. DataPosReg object is to read/write position register values.

<Example> Register PR[1] to PR[10] of motion group 1.  
Dim objDataPosReg As DataPosReg

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

### 5.3.1.6 AddPosRegXyzwpr - Register position registers to DataTable

VB6:	Function AddPosRegXyzwpr(DataType As FRIF_DATA_TYPE, Group As Long, StartIndex As Long, EndIndex As Long) As DataPosRegXyzwpr
VB2008:	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++:	LPDISPATCH AddPosRegXyzwpr(long DataType, long Group, long StartIndex, long EndIndex);
C#	<a href="#">FRRJIf.DataPosRegXyzwpr</a> AddPosRegXyzwpr( <a href="#">FRRJIf.FRIF_DATA_TYPE</a> DataType, <a href="#">int</a> Group, <a href="#">int</a> StartIndex, <a href="#">int</a> EndIndex)

Specify DataType(POSREG\_XYZWPR=12) , motion group number, the first and end index of position registers.

When it succeeds, this method returns DataPosRegXyzwpr object. When it fails, this method returns Nothing. DataPosRegXyzwpr object is to write position register values fast.

<Example> Register PR[1] to PR[10] of motion group 1.



```
Dim objDataPosRegXyzwpr As DataPosRegXyzwpr
```

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

### 5.3.1.7 AddPosRegMG - Register position registers(multiple group) to DataTable

VB6:	Function AddPosRegMG(DataType As FRIF_DATA_TYPE, Group As String, StartIndex As Long, EndIndex As Long) As DataPosRegMG
VB2008:	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++:	LPDISPATCH AddPosRegMG(long DataType, LPCTSTR Group, long StartIndex, long EndIndex);
C#	<a href="#">FRRJIf.DataPosReg</a> AddPosReg( <a href="#">FRRJIf.FRIF_DATA_TYPE</a> DataType, <a href="#">String</a> Group, <a href="#">int</a> StartIndex, <a href="#">int</a> EndIndex)

Specify DataType(POSREGMG=36) , a string to specify motion group, the first and end index of position registers.

The string to specify motion group is each motion group strings (“C” for Cartesian, “J#” for joint) that are connected with comma.

When it succeeds, this method returns DataPosRegMG object. When it fails, this method returns Nothing. DataPosRegMG object is to read/write position register values fast.

<Example> Register PR[1] to PR[10]. Motion group 1 is Cartesian. From motion group 2 to 5 are joint 1 axis

```
Dim objDataPosRegMG As DataPosRegMG
```

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

### 5.3.1.8 AddSysVar - Register system variables (integer, real and string type) to DataTable

VB:	Function AddSysVar(DataType As FRIF_DATA_TYPE, SysVarName As String) As DataSysVar
VC++:	LPDISPATCH AddSysVar(long DataType, LPCTSTR SysVarName);
C#	<a href="#">FRRJIf.DataSysVar</a> AddSysVar( <a href="#">FRRJIf.FRIF_DATA_TYPE</a> DataType, <a href="#">string</a> SysVarName)

Specify DataType(SYSVAR\_INT=5 for integer case, SYSVAR\_REAL=6 for float case, SYSVAR\_STRING=8 for string case) and system variable name.

You need to specify "\$[KAREL\_PROGRAM]KAREL\_VARIABLE" for KAREL variable cases. For example, you need to specify "\$[HTTPKCL]CMDS[1]" for \$CMDS[1] of HTTPKCL.

When it succeeds, this method returns DataSysVar object. When it fails, this method returns Nothing. DataSysVar object is to access system variable and KAREL variable.

<Example> Register \$FAST\_CLOCK

## 5.Object Reference

```
Dim objDataSysVar As DataSysVar
```

```
Set objDataSysVar = objCore.DataTable.AddSysVar(SYSVAR_INT, "$FAST_CLOCK")
```

### 5.3.1.9 AddSysVarPos - Register system variables (position type) to DataTable

VB6:	Function AddSysVarPos(DataType As FRIF_DATA_TYPE, SysVarName As String) As DataSysVarPos
VB2008:	Function AddSysVarPos(ByVal DataType As FRRJIf.FRIF_DATA_TYPE, ByVal SysVarName As String) As FRRJIf.DataSysVarPos
VC++:	LPDISPATCH AddSysVarPos(long DataType, LPCTSTR SysVarName);
C#	<a href="#">FRRJIf.DataSysVarPos</a> AddSysVarPos( <a href="#">FRRJIf.FRIF_DATA_TYPE</a> DataType, <a href="#">string</a> SysVarName)

Specify DataType(SYSVAR\_POS=7) and system variable name.

You need to specify "\$[KAREL\_PROGRAM]KAREL\_VARIABLE" for KAREL variable cases. For example, you need to specify "\$[HTTPKCL]CMDS[1]" for \$CMDS[1] of HTTPKCL.

When succeeded, this method returns DataSysVarPos object. When failed, this method returns Nothing. DataSysVarPos object is to access position type system variable and KAREL variable.

<Example> Register \$MNUTOOL[1,1]

```
Dim objDataSysVarPos As DataSysVarPos
```

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

### 5.3.1.10 AddTask - Register program execution status to DataTable

VB6:	Function AddTask(DataType As FRIF_DATA_TYPE, Index As Long) As DataTask
VB2008:	Function AddTask(ByVal DataType As FRRJIf.FRIF_DATA_TYPE, ByVal Index As Integer) As FRRJIf.DataTask
VC++:	LPDISPATCH AddTask(long DataType, long Index);
C#	<a href="#">FRRJIf.DataTask</a> AddTask( <a href="#">FRRJIf.FRIF_DATA_TYPE</a> DataType, <a href="#">int</a> Index)

Specify DataType and task number. DataType should be one of the followings.

DataType	Meaning
TASK(=4)	Return executing program name and line number. (Original)
TASK_IGNORE_MACRO (=33)	If executing program is a macro program, return caller program name and line number instead of the macro program. If all caller programs are macro, program name is "" and line number is zero.
TASK_IGNORE_KAREL (=34)	If executing program is a KAREL program, return caller program name and line number instead of the KAREL program. If all caller programs are KAREL, program name is "" and line number is zero.
TASK_IGNORE_MACRO_KAREL	If executing program is a macro or KAREL program,

(=35)	return caller program name and line number instead of the macro or KAREL program. If all caller programs are macro or KAREL, program name is "" and line number is zero.
-------	--

(\*) If you would like to ignore macro program or KAREL program, you specify TASK\_IGNORE\_MACRO, TASK\_IGNORE\_KAREL or TASK\_IGNORE\_MACRO\_KAREL.

(\*) You can use TASK\_IGNORE\_MACRO, TASK\_IGNORE\_KAREL and TASK\_IGNORE\_MACRO\_KAREL for robot controller 7DA7/24 or later.

When two tasks run in the multiple task system, task number 1 is to access the first one, task number 2 is to access second one. A task keeps the task number when executing.

When it succeeds, this method returns DataTask object. When it fails, this method returns Nothing. DataTask object is to read program execution status.

<Example> Register task number 1  
Dim objDataTask As DataTask

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

### 5.3.1.11 AddAlarm - Register alarm history to DataTable

VB6:	Function AddAlarm(DataType As FRIF_DATA_TYPE, Count As Long) As DataAlarm
VB2008:	Function AddAlarm(ByVal DataType As FRRJIf.FRIF_DATA_TYPE, ByVal AlarmCount As Integer, Optional ByVal AlarmMessageMode As Integer = 0) As FRRJIf.DataAlarm
VC++:	LPDISPATCH AddAlarm(long DataType, long AlarmCount, long AlarmMessageMode);
C#	<a href="#">FRRJIf.DataAlarm</a> AddAlarm( <a href="#">FRRJIf.FRIF_DATA_TYPE</a> DataType, <a href="#">int</a> AlarmCount, <a href="#">int</a> AlarmMessageMode)

Specify DataType is ALARM\_CURRENT(=10) for active alarm reference, DataType is ALARM\_LIST(=9) for alarm history reference and the number of alarm items from top. Specify AlarmMessageMode is 0 in Visual C++.

When it succeeds, this method returns DataAlarm object. When it fails, this method returns Nothing. DataAlarm object is to read alarm history.

<Example> register five alarm items  
Dim objDataAlarm As DataAlarm

Set objDataAlarm = objCore.DataTable.AddAlarm(ALARM\_LIST, 5)

### 5.3.1.12 AddString - Register string data to DataTable

VB6:	Function AddString(DataType As FRIF_DATA_TYPE, StartIndex As Long, EndIndex As Long) As DataNumReg
VB2008:	Function AddString(ByVal DataType As FRRJIf.FRIF_DATA_TYPE, ByVal StartIndex As Integer, ByVal EndIndex As Integer) As FRRJIf.DataNumReg

## 5.Object Reference

VC++:	LPDISPATCH AddString(long DataType, long StartIndex, long EndIndex);
C#	<a href="#">FRRJIf.DataString</a> AddString( <a href="#">FRRJIf.FRIF_DATA_TYPE</a> <i>DataType</i> , <a href="#">int</a> <i>StartIndex</i> , <a href="#">int</a> <i>EndIndex</i> )

Specify one of the followings as DataType.

String register	STRREG(=13)
String register comment	STRREG_COMMENT(=14)
Numeric register comment	NUMREG_COMMENT(=15)
Position register comment	POSREG_COMMENT(=16)
SDI comment	SDI_COMMENT(=17)
SDO comment	SDO_COMMENT(=18)
RDI comment	RDI_COMMENT(=19)
RDO comment	RDO_COMMENT(=20)
UI comment	UI_COMMENT(=21)
UO comment	UO_COMMENT(=22)
SI comment	SI_COMMENT(=23)
SO comment	SO_COMMENT(=24)
WI comment	WI_COMMENT(=25)
WO comment	WO_COMMENT(=26)
WSI comment	WSI_COMMENT(=27)
GI comment	GI_COMMENT(=29)
GO comment	GO_COMMENT(=30)
AI comment	AI_COMMENT(=31)
AO comment	AO_COMMENT(=32)

When it succeeds, this method returns DataString object. When it fails, this method returns Nothing. DataString object is to access string data.

<Example> register five string registers  
Dim objDataString As DataString

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

### 5.3.1.13 Refresh - Read data values from robot to refresh data

VB:	Function Refresh() As Boolean
VC++:	BOOL Refresh();
C#	<a href="#">bool</a> Refresh()

When Refresh method is called, data of DataTable object are refreshed. DataTable object keeps the value until next Refresh method calling. If you want to get the latest robot data, you need to call this method. If you do not call this method, data on data table are not changed.

Success returns True, fails returns False.

Remark

(\*) You must call data register methods like AddCurPos before connecting. You must not call data register methods after connecting. If you need to change data table, then you need to disconnect, delete all FRRJIF objects and create data table again.

## 5.4 FRRJIF.DataCurPos

### 5.4.1 Methods

Valid, GetValue

#### 5.4.1.1 Valid - Check object validity

VB:	Property Valid As Boolean (read only)
VC++6:	BOOL GetValid();
VC++2008	BOOL get_Valid();
C#	<a href="#">bool</a> Valid { get; }

When this object is valid, returns True. When this object is invalid, returns False.

#### 5.4.1.2 GetValue - Read current position

VB6:	Function GetValue(Xyzwpr() As Single, Config() As Integer, Joint() As Single, UF As Integer, UT As Integer, validc As Integer, validj As Integer) As Boolean
VB2008:	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#	<a href="#">bool</a> GetValue(ref <a href="#">System.Array</a> Xyzwpr, ref <a href="#">System.Array</a> Config, ref <a href="#">System.Array</a> Joint, ref <a href="#">short</a> UF, ref <a href="#">short</a> UT, ref <a href="#">short</a> ValidC, ref <a href="#">short</a> ValidJ)

Argument Xyzwpr() will have returned Cartesian values in Visual Basic. Xyzwpr() should have 9 elements (robot 6 axes + 3 extended axes).

Argument Config() will have returned configuration of current position.  
Config(0) to Config(3) mean as follows:

	Non 0	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) to Config(6) mean turn numbers.

Argument Joint() will have returned joint values. Joint() should have 9 elements (robot 6 axes + 3 extended axes)

Argument UF will have returned user frame number. Argument UT will have returned user tool number.

When current position has valid Cartesian values, argument ValidC will have non 0. When current position do not have valid Cartesian values, argument ValidC will have 0. When current position has valid joint values, argument ValidJ will have non 0. When current position do not have valid joint values,

## 5.Object Reference

---

argument ValidJ will have 0. (For example, servo gun of motion group2 will return ValidC as 0)

In Visual C++, you use GetValueXyzwpr to get Cartesian values and use GetValueJoint to get joint values. Do not use array as arguments.

Success returns True, fails returns False.

## 5.5 FRRJIF.DataNumReg

---

### 5.5.1 Methods

---

Valid, GetValue, SetValue, SetValues

#### 5.5.1.1 Valid - Check object validity

---

VB:	Property Valid As Boolean (read only)
VC++6:	BOOL GetValid();
VC++2008	BOOL get_Valid();
C#	bool Valid { get; }

When this object is valid, returns True. When this object is invalid, returns False.

#### 5.5.1.2 GetValue - Read numeric register value

---

VB6:	Function GetValue(Index As Long, Value As Variant) As Boolean
VB2008:	Function GetValue(ByVal Index As Integer, ByRef Value As Object) As Boolean
VC++:	BOOL GetValue(long Index, VARIANT* Value);
C#	bool GetValue(int Index, ref object Value)

Specify the index of target numeric register. Argument Value will have returned value.

Success returns True, fails returns False.

#### 5.5.1.3 SetValue - Set numeric register value

---

VB6:	Function SetValue(Index As Long, Value) As Boolean
------	--

This method is for compatibility. Please use SetValues instead of SetValue.

Success returns True, fails returns False.

#### 5.5.1.4 SetValues – Set a series of numeric register values

---

VB6:	Function SetValues(Index As Long, Value As Variant, Count As Long) As Boolean
VB2008:	Function SetValues(ByVal Index As Integer, ByVal Value As Object, ByVal Count As Integer) As Boolean
VC++:	BOOL SetValuesInt(long Index, long* Value, long Count); BOOL SetValuesReal(long Index, float* Value, long Count);
C#	bool SetValues(int Index, object Value, int Count)

This method is to set a series of numeric register values at a time. It is faster than SetValue method.

Specify the index of the first target numeric register. Argument Value is array of set values. Specify the number of registers to argument Count.

In Visual C++, you use SetValueInt to set integer values, use SetValueReal to set real values.

Success returns True, fails returns False.

## 5.6 FRRJIF.DataPosReg

### 5.6.1 Methods

Valid, GetValue, SetValueJoint, SetValueXyzwpr

#### 5.6.1.1 Valid - Check object validity

VB:	Property Valid As Boolean (read only)
VC++6:	BOOL GetValid();
VC++2008	BOOL get_Valid();
C#	bool Valid { get; }

When this object is valid, returns True. When this object is invalid, returns False.

#### 5.6.1.2 GetValue - Read position register value

VB6:	Function GetValue(Index As Long, Xyzwpr() As Single, Config() As Integer, Joint() As Single, UF As Integer, UT As Integer, validc As Integer, validj As Integer) As Boolean
VB2008:	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(long 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(long 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)

Specify the index of target position register. Other arguments are the same as arguments of DataCurPos.GetValue method.

Success returns True, fails returns False.

#### 5.6.1.3 SetValueJoint - Set joint value to position register

VB6:	Function SetValueJoint(Index As Long, Joint() As Single, UF As Integer, UT As Integer) As Boolean
VB2008:	Function SetValueJoint(ByVal Index As Integer, ByRef Joint As System.Array,

## 5.Object Reference

	ByVal UF As Short, ByVal UT As Short) As Boolean
VC++:	BOOL SetValueJoint2(long 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)

Specify the index of target position register. Specify joint values, user frame number and user tool number to Joint(), UF and UT.

Success returns True, fails returns False.

### 5.6.1.4 SetValueXyzwpr - Set Cartesian value to position register

VB6:	Function SetValueXyzwpr(Index As Long, Xyzwpr() As Single, Config() As Integer, UF As Integer, UT As Integer) As Boolean
VB2008:	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(long 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)

Specify the index of target position register. Specify Cartesian values, configuration, user frame number and user tool number to Joint(), Config(), UF and UT.

In Visual C++, use each value, not array of values.

Success returns True, fails returns False.

## 5.7 FRRJIF.DataPosRegXyzwpr

### 5.7.1 Methods

Valid, SetValueXyzwpr, Update, Reset

#### 5.7.1.1 Valid - Check object validity

VB:	Property Valid As Boolean (read only)
VC++6:	BOOL GetValid();
VC++2008	BOOL get_Valid();
C#	bool Valid { get; }

When this object is valid, returns True. When this object is invalid, returns False.

#### 5.7.1.2 SetValueXyzwpr - Store Cartesian value to buffer

VB6:	Function SetValueXyzwpr(Index As Long, Xyzwpr() As Single, Config() As Integer) As Boolean
VB2008:	Function SetValueXyzwpr(ByVal Index As Integer, ByRef Xyzwpr As System.Array, ByRef Config As System.Array) As Boolean



VC++:	BOOL SetValueXyzwpr2(long 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)

Specify the index of target position register. Specify Cartesian values and configuration. The data is store in buffer. The data is not transfer immediately. You call SetValueXyzwpr multiple times to store a series of position registers, then you need to call Update method to transfer stored data.

In Visual C++, use each value, not array of values.

Success returns True, fails returns False.

### 5.7.1.3 Update - Transfer buffered data to robot position registers

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

This method is to transfer data in the buffer to robot. The transferred data is from minimum index that you call SetValueXyzwpr to maximum index that you call SetValueXyzwpr. (Transfer data that you do not call SetValueXyzwpr is zero value.) The buffer is reset after the transferring.

Success returns True, fails returns False.

### 5.7.1.4 Reset - Reset the buffer

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

## 5.8 FRRJIF.DataPosRegMG

### 5.8.1 Methods

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

#### 5.8.1.1 Valid - Check object validity

VB:	Property Valid As Boolean (read only)
VC++6:	BOOL GetValid();
VC++2008	BOOL get_Valid();
C#	bool Valid { get; }

When this object is valid, returns True. When this object is invalid, returns False.

#### 5.8.1.2 GetValueJoint - Get joint values from a position register

VB6:	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,
------	--

## 5.Object Reference

	ByRef float J7, ByRef float J8, ByRef float J9) As Boolean
VB2008:	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(long Index, long 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);

Specify the index of target position register. Specify the group number. The motion group should be specified as joint in AddPosRegMG calling.

Success returns True, fails returns False.

### 5.8.1.3 GetValueXyzwpr - Get Cartesian values from a position register

VB6:	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
VB2008:	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(long Index, long 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);

Specify the index of target position register. Specify the group number. The motion group should be specified as Cartesian in AddPosRegMG calling.

Success returns True, fails returns False.

### 5.8.1.4 SetValueJoint - Store joint values to buffer

VB6:	Function SetValueJoint(Index As Long, Group As Long, Joint() As Single) As Boolean
VB2008:	Function SetValueJoint(ByVal Index As Integer, ByVal Group As Integer, ByRef Joint As System.Array) As Boolean
VC++:	BOOL SetValueJoint2(long Index, long Group, float J1, float J2, float J3, float J4, float J5, float J6, float J7, float J8, float J9);
C#	<a href="#">bool</a> SetValueJoint( <a href="#">int</a> Index, int Group, ref <a href="#">System.Array</a> Joint)

Specify the index of target position register. Specify the motion group number. The motion group should be specified as joint in AddPosRegMG calling. The number of elements of the joint array must equal joint number in AddPosRegMG calling.

The data is store in buffer. The data is not transfer immediately. You call SetValueJoint or SetValueXyzwpr multiple times to store a series of position registers, and then you need to call Update

method to transfer stored data.

Success returns True, fails returns False.

### 5.8.1.5 SetValueXyzwpr - Store Cartesian value to buffer

VB6:	Function SetValueXyzwpr(Index As Long, Group As Long, Xyzwpr() As Single, Config() As Integer) As Boolean
VB2008:	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(long Index, long 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)

Specify the index of target position register. Specify the group number. The motion group should be specified as Cartesian in AddPosRegMG calling.

The data is store in buffer. The data is not transfer immediately. You call SetValueJoint or SetValueXyzwpr multiple times to store a series of position registers, and then you need to call Update method to transfer stored data.

Success returns True, fails returns False.

### 5.8.1.6 Update - Transfer buffered data to robot position registers

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

This method is to transfer data in the buffer to robot. The transferred data is from minimum index that is specified in AddPosRegMG calling to maximum index that is specified in AddPosRegMG calling. (Transfer data that you do not call SetValueJoint or SetValueXyzwpr is zero value.) The buffer is reset after the transferring.

Success returns True, fails returns False.

### 5.8.1.7 Reset - Reset the buffer

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

## 5.9 FRRJIF.DataSysVar

### 5.9.1 Methods

Valid, GetValue

## 5.Object Reference

---

### 5.9.1.1 Valid - Check object validity

---

VB:	Property Valid As Boolean (read only)
VC++6:	BOOL GetValid();
VC++2008	BOOL get_Valid();
C#	bool Valid { get; }

When this object is valid, returns True. When this object is invalid, returns False.

### 5.9.1.2 GetValue - Read system variable value

---

VB6:	Function GetValue(Value As Variant) As Boolean
VB2008:	Function GetValue(ByRef Value As Object) As Boolean
VC++:	BOOL GetValue(VARIANT* Value);
C#	bool GetValue(ref object Value)

Argument Value will have returned system variable value.

Success returns True, fails returns False.

### 5.9.1.3 SetValue - Set system variable value

---

VB6:	Function SetValue(Value As Variant) As Boolean
VB2008:	Function SetValue(ByVal Value As Object) As Boolean
VC++:	BOOL SetValue(const VARIANT& Value);
C#	bool SetValue(object Value)

Specify value to be set.

Success returns True, fails returns False.

## 5.10 FRRJIF.DataSysVarPos

---

### 5.10.1 Methods

---

Valid, GetValue, SetValueJoint, SetValueXyzwpr

#### 5.10.1.1 Valid - Check object validity

---

VB:	Property Valid As Boolean (read only)
VC++6:	BOOL GetValid();
VC++2008	BOOL get_Valid();
C#	bool Valid { get; }

When this object is valid, returns True. When this object is invalid, returns False.

#### 5.10.1.2 GetValue - Read POSITION type system variable value

---

VB6:	Function GetValue(Xyzwpr() As Single, Config() As Integer, Joint() As Single,
------	---

	UF As Integer, UT As Integer, validc As Integer, validj As Integer) As Boolean
VB2008:	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#	<a href="#">bool</a> GetValue(ref <a href="#">System.Array</a> Xyzwpr, ref <a href="#">System.Array</a> Config, ref <a href="#">System.Array</a> Joint, ref <a href="#">short</a> UF, ref <a href="#">short</a> UT, ref <a href="#">short</a> ValidC, ref <a href="#">short</a> ValidJ)

Arguments are the same as arguments of DataCurPos.GetValue method.

Success returns True, fails returns False.

### 5.10.1.3 SetValueJoint - Set joint value to system variable

VB6:	Function SetValueJoint(Joint() As Single, UF As Integer, UT As Integer) As Boolean
VB2008:	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#	<a href="#">bool</a> SetValueJoint(ref <a href="#">System.Array</a> Joint, <a href="#">short</a> UF, <a href="#">short</a> UT)

Specify joint values, user frame number and user tool number to Joint(), UF and UT. In Visual C++, use each value, not array of values.

Success returns True, fails returns False.

### 5.10.1.4 SetValueXyzwpr - Set Cartesian value to system variable

VB6:	Function SetValueXyzwpr(Xyzwpr() As Single, Config() As Integer, UF As Integer, UT As Integer) As Boolean
VB2008:	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#	<a href="#">bool</a> SetValueXyzwpr(ref <a href="#">System.Array</a> Xyzwpr, ref <a href="#">System.Array</a> Config, <a href="#">short</a> UF, <a href="#">short</a> UT)

Specify Cartesian values, configuration, user frame number and user tool number to Joint(), Config(), UF and UT. In Visual C++, use each value, not array of values.

Success returns True, fails returns False.

## 5.11 FRRJIF.DataTask

---

### 5.11.1 Methods

---

Valid, GetValue

#### 5.11.1.1 Valid - Check object validity

---

VB:	Property Valid As Boolean (read only)
VC++6:	BOOL GetValid();
VC++2008	BOOL get_Valid();
C#	bool Valid { get; }

When this object is valid, returns True. When this object is invalid, returns False.

#### 5.11.1.2 GetValue - Read program execution status

---

VB6:	Function GetValue(ProgName As String, LineNumber As Integer, State As Integer, ParentProgName As String) As Boolean
VB2008:	Function GetValue(ByRef ProgName As String, ByRef LineNumber As Short, ByRef State As Short, ByRef ParentProgName As String) As Boolean
VC++:	BOOL GetValue(BSTR* ProgName, short* LineNumber, short* State, BSTR* ParentProgName);
C#	bool GetValue(ref string ProgName, ref short LineNumber, ref short State, ref string ParentProgName)

Argument ProgName will have returned current executing program name.

Argument LineNumber will have returned current executing program line number.

Argument State will have returned execution status. When program stopped, State is 0. When program paused, State is 1. When program executing, State is 2.

Argument ParentProgName will have returned program name that is executed first. If no child program is called, value of ParentProgName is the same as value of ProgName.

(\*) DataType for FRRJIF.DataTable.AddTask affects target current executing program. Please refer FRRJIF.DataTable.AddTask.

Success returns True, fails returns False.

## 5.12 FRRJIF.DataAlarm

---

### 5.12.1 Methods

---

Valid, GetValue

#### 5.12.1.1 Valid - Check object validity

---

VB:	Property Valid As Boolean (read only)
VC++6:	BOOL GetValid();

VC++2008	BOOL get_Valid();
C#	bool Valid { get; }

When this object is valid, returns True. When this object is invalid, returns False.

### 5.12.1.2 GetValue - Read alarm history

VB6:	Function GetValue(Count As Long Count As Long, AlarmID As Integer, AlarmNumber As Integer, CauseAlarmID As Integer, CauseAlarmNumber As Integer, Severity As Integer, Year As Integer, Month As Integer, Day As Integer, Hour As Integer, Minute As Integer, Second As Integer, AlarmMessage As String, CauseAlarmMessage As String, SeverityMessage As String) As Boolean
VB2008:	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(long Count, short* AlarmID, short* AlarmNumber, short* CauseAlarmID, short* CauseAlarmNumber, short* Severity, short* Year, short* Month, short* Day, short* Hour, short* Minute, short* Second, BSTR* AlarmMessage, BSTR* CauseAlarmMessage, BSTR* SeverityMessage);
C#	<i>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)</i>

Specify argument Count as index of target alarm history item. (Specify 1 for the first item.) Argument AlarmID will have returned alarm ID. In case of 'SRVO-001', AlarmID is 11 that represents 'SRVO'. Please see alarm code table in R-J3 reference. If there is no active alarm, AlarmID is zero for active alarm reference.

Argument AlarmNumber will have returned alarm number. In case of 'SRVO-001', AlarmNumber is 1. If there is no active alarm, AlarmNumber is zero for active alarm reference.

Argument CauseAlarmID will have returned cause alarm ID. Some alarm have two alarm messages. The second alarm is cause code. This argument is to read cause code. If there is no cause code, CauseAlarmID is 0.

Argument CauseAlarmNumber will have returned cause alarm Number. This argument is to read cause code alarm number. If there is no cause code, CauseAlarmNumber is 0.

Argument Severity will have return alarm severity. Severity value means as follows:

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

## 5.Object Reference

---

Argument Year, Month, Day, Hour, Minute, Second will have returned alarm occurred date and time (24 hours format).

Argument AlarmMessage will have returned alarm message. The message is the top line to teach pendant screen includes alarm code like 'SRVO-001'. (Kanji message not supported.)

Argument CauseAlarmMessage will have returned cause code alarm message. (Kanji message not supported)

Argument SeverityMessage will have returned alarm severity string like 'WARN'.

Success returns True, fails returns False.

## 5.13 FRRJIF.DataString

---

Only R-30iA 7DA7/13 or later supports this service.

### 5.13.1 Methods

---

Valid, GetValue, SetValue, Update, Reset

#### 5.13.1.1 Valid - Check object validity

---

VB:	Property Valid As Boolean (read only)
VC++6:	BOOL GetValid();
VC++2008	BOOL get_Valid();
C#	bool Valid { get; }

When this object is valid, returns True. When this object is invalid, returns False.

#### 5.13.1.2 GetValue - Get string value

---

VB6:	Function GetValue(Index As Long, Value As String) As Boolean
VB2008:	Function GetValue(ByVal Index As Integer, ByRef Value As String) As Boolean
VC++:	BOOL GetValue(long Index, BSTR* Value);
C#	bool GetValue(int Index, ref string Value)

Specify the index of target string data (string register or comment). Argument Value will have returned string data value.

Success returns True, fails returns False.

#### 5.13.1.3 SetValue - Store string value to buffer

---

VB6:	Function SetValue(Index As Long, Value As String) As Boolean
VB2008:	Function SetValue(ByVal Index As Integer, ByVal Value As String) As Boolean
VC++:	BOOL SetValue(long Index, LPCTSTR Value);
C#	bool SetValue(int Index, string Value)

Specify the index of target string data (string register or comment). Specify a string data. The data is store in buffer. The data is not transfer immediately. You call SetValue multiple times to store a series of string data, then you need to call Update method to transfer stored data.



Success returns True, fails returns False.

#### 5.13.1.4 Update - Transfer buffered data to robot

---

VB:	Function Update() As Boolean
VC++:	BOOL Update();
C#	<a href="#">bool</a> Update()

This method is to transfer data in the buffer to robot. The transferred data is from most minimum index that you call SetValue to maximum index that you call SetValue. The buffer is reset after the transferring.

Success returns True, fails returns False.

#### 5.13.1.5 Reset - Reset the buffer

---

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

# 6 Connect To ROBOGUIDE

---

You can connect your robot interface application to virtual robots on ROBOGUIDE. You can use ROBOGUIDE for developing/testing your application with virtual robots.

## 6.1 Environment

---

ROBOGUIDE V7 Rev.F(7N06) or later is needed. Virtual robot should be R-30iA 7DA5/15, R-30iA 7DA7/13 or later.

## 6.2 About IP Address Of The Virtual Robot

---

To confirm IP address of the virtual robot, please click ROBOGUIDE menu Robot/Default Browser to open the robot home page. Internet Explorer address bar indicates the robot IP address like "http://127.0.0.1/" or "http://127.0.0.1:9001/". The IP address is "127.0.0.1" in this example. Your virtual robot may have another IP address like "127.0.0.2."

If "127.0.0.#" is not accepted, please try to use IP address of the PC or "localhost".

## 6.3 About Port Number

---

A ROBOGUIDE workcell can have multiple virtual robots. And a PC can have multiple ROBOGUIDE. Thus, the PC may have multiple virtual robots at a time. Your robot interface application will connect to a virtual robot. It is necessary to specify the target virtual robot. Port number is used for the purpose. Each robot on a PC has different port number for robot interface. You get the port number of the target virtual robot, then connect to the robot with the port number.

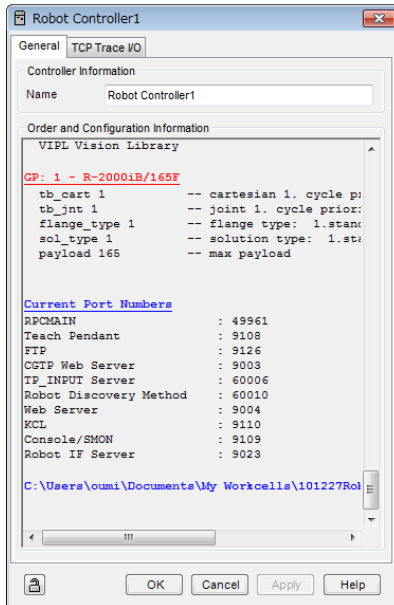
(\*) Port number is used for virtual robot connection. Port number is not used for actual robot connection.

### 6.3.1 How To Get Port Number

---

A virtual robot may have different port number every start up. Thus, you need to get port number of target virtual robot before connection every time.

You can confirm port number in the robot controller property page of ROBOGUIDE. The property page has "Order and Configuration Information". It includes "Current Port Numbers". "Robot IF Server" is the port number. In the below example, the port number is 9023.



(\*) If you cannot see port number information in the property page, ROBOGUIDE and/or virtual robot is old. You need to update ROBOGUIDE.

(\*) If R650 FRA Params is selected and R553 "HMI Device (SNPX)" is not selected on the virtual robot, you cannot see the port number. Please add R553 "HMI Device (SNPX)" to the virtual robot.

Another way to get port number is reading from a file in the workcell directory. ROBOGUIDE assigns ID to every virtual robot at creating. You can confirm ID of target virtual robot in cell browser. Robot controller node has text like "C: ID - Name". For example, the robot controller node has "C : 1 - Left Robot", the ID is 1. After starting up virtual robots, you can access workcell\_folder/robot\_ID/services.txt. (If ID is 1. Robot\_ID is Robot\_1.) Services.txt has port number information. You can get the port number in the line that begins with "Robot IF Server".

## 6.3.2 How To Use Port Number

It is need to set the port number to FRRJIf.Core.PortNumber property before the connection.

<Sample Code - "HostName:PortNumber" format>

```
'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, ":")
```

## 6.Connect To ROBOGUIDE

---

```
    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 Trouble Shooting

- Please confirm communication between PC and robot without robot interface. For example, open robot home page with Internet Explorer. If you cannot open robot home page, your network may have trouble. Please confirm IP configuration, cable, hub and so on.
- If the robot has R650 FRA Params option, R553 "HMI Device (SNPX)" is needed. (If R651 FRL Params is selected, no option is needed.)
- Please use robot interface sample program in order to test communication between robot and PC.
- Please set enough time out value. If time out value is too short, it may cause communication errors since response time is not constant.
- Please consider to retry connection for communication errors. It will be more robust software.
- When communication error occur, robot interface write error messages with OutputDebugString API. You will get like the following messages.

208: [RobotIF.acc\_socket] Time Out at line 369, 0, 501, 10000

208: [RobotIF.snpx\_rw] Error at line 290

If the message includes "Time Out", enlarge time out value may fix the communication error.

(\*) FANUC provides dbmon.exe that is to capture the messages. Please find dbmon.exe in install CD-ROM tools folder. Please use in on PC local disk.

- Network protocol analyzer for Windows may help you to grasp the situation.
- If communication errors occur with robot moving, the noise may cause the errors. Please suppress noise such cases.
- To enable logging function, please specify the following registry.

[HKEY\_CURRENT\_USER¥Software¥VB and VBA Program Settings¥FRRJIF¥Setting]

"DebugOut"="True"

The following registry values defines log output directory.

[HKEY\_CURRENT\_USER¥Software¥VB and VBA Program Settings¥FRRJIF¥Setting]

"DebugOutPath"

If log output directory is not defined. Log file is written in install directory (usually C:¥Program Files¥FANUC¥FRRJIf).

Log file name is logyymmddhmmss.txt. (It may be necessary to execute your application as administrator in order to write the log file.)

Please FRRJIF.Core object DebugLog to True to get more detail information.

(\*) Please enable logging function only when you need it. It may make performance down.