RockwellAutomation
Logix5000 プロバイダ

Version 1.0.7

ユーザーズ ガイド

October 21, 2024

【備考】

【改版履歴】

バージョン	日付	内容
1.0.0	2016-10-31	初版.
1.0.1	2018-10-15	AddController 失敗時の処理修正
1.0.1	2018-10-29	AddController 時の Elem 上限を記載
1.0.2	2019-02-04	構造体一括取得対応 配列分割送受信対応
	2019-04-03	BOOL 型配列の注意点, リクエストパス例を記載
1.0.3	2019-09-27	内部処理(シーケンスカウント)見直し
	2019-10-23	付録 A 修正
	2019-11-28	CompactLogix の一部機種にスロット番号の指定が必要の旨を記載
1.0.4	2022-08-02	要求パケットと応答パケットの整合性チェックを追加
1.0.5	2024-01-30	ForwardOpen のシーケンス番号採番処理を変更
1.0.6	2024-10-21	断線時にロック状態になってしまうのを修正
1.0.7	2024-10-21	Vendor ID を指定するように修正 ForwardOpen 時に RedundantOwner の設定ができるように修正

【動作確認機種】

機種	注意事項	プロバイダの確認バージョン
1769-L33ER (ver.21)	1769-L33ER CompactLogix™ 5370 Controller	1.0.6
1756-L71 (ver.21)	1756-L71 ControllLogix® 5570 Controller	1.0.6
1769-L35E	1769-L35E CompactLogix SlotNo オプションの指定が必要	1.0.3
1756-L81E (ver.33.01)	1756-L81E ControlLogix® 5580 Controller (1756-EN4TR, 1756-EN2TR の EtherNetIP ユニットを使用)	1.0.7
1756-L75 (ver.33.01)	1756-L75 ControlLogix® 5570 Controller (1756-EN4TR, 1756-EN2TR の EtherNetIP ユニットを使用)	1.0.7

目次

1. はじめに

本書は，RockwellAutomation 製コントローラ(ControlLogix/CompactLogix シリーズ)に対しデータの書き込み/読み出しを行う CAO プロバイダのユーザーズガイドです．

本書で扱う CAO プロバイダ(CaoProvRockwellLogix5000.dll)を Logix5000 プロバイダと呼びます．

第 2 章に Logix5000 プロバイダの概要，変数の詳細を記載しています．

Logix5000 プロバイダで実装している通信コマンドの対応状況及びデータ列については，通信先となるコントローラ(ControlLogix/CompactLogix シリーズ)に依存します．

通信の詳細については RockwellAutomation の“1756-pm020c-en-p.pdf”並びに ODVA の CIP 仕様書“Vol2_1.4.pdf”を参照してください．

2. プロバイダの概要

2.1. 概要

Logix5000 プロバイダは，RockwellAutomation 製コントローラ(ControlLogix/CompactLogix シリーズ)に対しEthnet/IP(TCP)接続でCIPフォーマットに乗せたアクセス用コマンドを用いてデータの書き込み/読み出しを行うCAOプロバイダです．そのファイル形式はDLL(Dynamic Link Library)であり，CAOエンジンから使用時に動的にロードされます．Logix5000プロバイダを使用するにあたってはORiN2SDKをインストールするか，下表を参照して手作業でレジストリ登録を行う必要があります．

表 2-1 Logix5000 プロバイダ

ファイル名	CaoProvRockwellLogix5000.dll
ProgID	CaoProv. Rockwell. Logix5000
レジストリ登録	regsvr32 CaoProvRockwellLogix5000.dll
レジストリ登録の抹消	regsvr32 /u CaoProvRockwellLogix5000.dll

2.2. メソッド・プロパティ

2.2.1. CaoWorkspace::AddController メソッド

Logix5000 プロバイダは AddController 時に通信用の接続パラメータを参照し，通信の接続を行います．(TCP クライアントとして動作します)

書式 AddController(<bstrCtrlName:BSTR>,<bstrProvName:BSTR>,
<bstrPCName:BSTR>,<bstrOption:BSTR>))

bstrCtrlName : [in] コントローラ名
bstrProvName : [in] プロバイダ名．固定値 =" CaoProv. Rockwell. Logix5000"
bstrPcName : [in] プロバイダの実行マシン名
bstrOption : [in] オプション文字列

以下にオプション文字列に指定するリストを示します．

表 2-2 CaoWorkspace::AddController のオプション文字列

オプション ([1])	説明
Conn=<接続パラメータ>	必須． 通信形態と接続パラメータ． (参照 2.2.1.1)
MyIP[=<自 IP アドレス>]	自 IP アドレス．(複数 NIC 用途) （デフォルト：指定なし）
ConnTimeout [=<タイムアウト時間>]	TCP 接続自のタイムアウト時間．(ミリ秒) （デフォルト：3000）
TimeOut[=<タイムアウト時間>]	送受信時のタイムアウト時間．(ミリ秒) （デフォルト：3000）
PathCheck[=TRUE / FALSE]	AddVariable 時リクエストパス有効チェック． TRUE：チェックを行う FALSE：チェックを行わない （デフォルト：FALSE）
SlotNo[=<スロット番号>]	アクセス対象ユニットのスロット番号． （デフォルト：指定なし） 例1) CompactLogix ⇒ 基本的に不要[2] 例2) ComapctLogix(CPU=0) ⇒ SlotNo=0 例3) ControlLogix(CPU=0) ⇒ SlotNo=0

[1] 角括弧("[ ]")内は省略可能を示します．また，各パラメータの解説中の下線部はオプションを指定しなかったときのデフォルト値になります．
[2] CompactLogix では SlotNo の指定は基本的には不要ですが，一部機種では SlotNo の指定が必要になります．SlotNo が必要かどうかは各機種のマニュアルをご確認ください．

EnableRedundantOwner	接続時に Ethernet/IP の CIP の ForwardOpen コマンド内にある RedundantOwner の設定をします. TRUE：Redundant Owner FALSE：Non-Redundant Owner （デフォルト：TRUE）

2.2.1.1. Conn オプション

以下に Conn オプションの接続パラメータ文字列を示します.

EtherNet デバイス

"Conn=ETH:<Dest IP Address>[:<Dest Port No>[:<Src IP Address>[:<Src Port No>]]]"

"Conn=TCP:<Dest IP Address>[:<Dest Port No>[:<Src IP Address>[:<Src Port No>]]]"

< Dest IP Address >	:	TCP/IP 接続先 IP アドレス. 例："127.0.0.1", "192.168.0.1"
<Dest Port No>	:	TCP/IP 接続ポート番号. 例：44818, 5006, 5007, …任意指定可能
<Src IP Address>	:	自 IP アドレス. (複数 NIC 用途) ([3])
<Src Port No>	:	自ポート番号. (複数 NIC 用途)

[3] Conn オプションと MyIP オプションの両方で自 IP アドレスを指定するとエラーになります. 利用する場合は必ずどちらか片方で指定するようにしてください.

2.2.2. CaoController::VariableNames プロパティ

接続可能な変数名リストを取得します. 本プロパティで取得した変数名は, 後述する AddVariable メソッドの第一引数に使用することができます[4].

2.2.3. CaoController::AddVariable メソッド

CaoController クラスの AddVariable メソッドは, コントローラ(ControlLogix/CompactLogix シリーズ)内のタグ定義に対しデータの書き込み/読み出しを行うための変数オブジェクトを作成するためのメソッドです.

本メソッドでは各オプションの指定によってアクセスするデータのリクエストパス, データ型, アクセス要素数を決定します.

作成した変数への初回アクセス時(読込み, または書き込み)に型のサイズの取得を行います.

PathCheck=True を設定していた場合は, AddVariable 時に取得を行います.

書式 AddVariable(<bstrVariableName:VT_BSTR>[,<bstrOption:VT_BSTR>])

<bstrVariableName> : [in] 変数名

<bstrOption> : [in] オプション文字列

以下にオプション文字列に指定するリストを示します.

表 2-3 CaoController::AddVariable のオプション文字列

<table>
<tr><th>オプション</th><th>説明</th></tr>
<tr><td>Path=<リクエストパス></td><td>必須.
アクセスするデータのリクエストパスを指定.
(参照 2.2.3.1)</td></tr>
<tr><td>DataFormat[=<データ型>]</td><td>Path オプションにより指定したリクエストパスの末尾(実際にアクセスするデータ)のデータ型を指定. (10 進数)
<table>
<tr><th>値</th><th>意味</th></tr>
<tr><td>0</td><td>数値型, 数値型配列</td></tr>
<tr><td>1</td><td>文字列型, 文字列型配列</td></tr>
<tr><td>2</td><td>構造体型, 構造体型配列</td></tr>
</table>
（デフォルト:0）
(参照 2.2.3.2)</td></tr>
<tr><td>Elem[=<要素数>]</td><td>Put/Get するデータの要素数を指定. (10 進数)</td></tr>
</table>

[4] 対象の機器で定義されているタグ一覧を取得するためには, ファームウェアバージョンを 21 以降にする必要があります.

	Path オプションにより指定したリクエストパスの末尾が配列指定の場合のみ有効. （デフォルト:1） (参照 2.2.3.4)

2.2.3.1. Path オプション

アクセスするデータのリクエストパスを文字列で指定します.

構造体メンバのような階層の深いデータにアクセスする場合は, パスとパスをドット(".")で繋いで指定します. パスを配列として指定する場合はパスの末尾に"[1]"の様に角括弧で要素番号を指定したものを付与します. 具体的な指定の仕方に関しては付録 A を参照してください[5].

注意点として, リクエストパス指定の末端のデータは数値型, 文字列型, あるいはそれらの配列である必要があります. 構造体定義そのものや要素番号を指定しない配列定義をリクエストパスとして指定し, 書き込み/読み出しを行った場合は意図しない結果となる場合があります.

2.2.3.2. DataFormat オプション

Path オプションにより指定したリクエストパスの末尾(実際にアクセスするデータ)のデータ型を指定します.

プロバイダで扱う VARIANT のデータ型とコントローラ内のデータ型は以下の様に相互変換し扱います.

表 2-4 データ型対応一覧

VARIANT のデータ型([6])	コントローラ内のデータ型		サイズ	説明
VT_BOOL	BOOL		1Byte	ブール値 (TRUE / FALSE)
VT_I1	SINT		1Byte	符号つき 1 バイト整数
VT_I2	INT		2Byte	符号つき 2 バイト整数
VT_I4	DINT		4Byte	符号つき 4 バイト整数
VT_R4	REAL		4Byte	単精度実数
VT_UI4	DWORD		4Byte	符号なし 4 バイト整数
VT_I8	LINT		8Byte	符号つき 8 バイト整数
VT_BSTR	STRING	LEN	4Byte	文字列長(DINT 型)
		DATA[n]	1Byte×n	文字データの ASCII 配列 (SINT[n]型)
VT_ARRAY | VT_UI1	全ての型		1Byte×n	バイト配列

[5] BOOL 型配列へのアクセスは DWORD 単位でのアクセスになります. 要素番号が 0 の場合は 0～31 要素, 1 の場合は 32～63 要素へのアクセスになります.

[6] DataFormat=2 の場合は(VT_ARRAY | VT_UI1)の指定のみとなります. また, DataFormat=0, 1 の場合は(VT_ARRAY | VT_UI1)を指定することは出来ません.

2.2.3.3. リクエストパス有効チェック

AddController 時のオプションで PathCheck=TRUE をしていた場合に変数生成時に Path オプションで指定されたパスが有効かチェックを行います.

チェック内容としては, 内部的に get_Value 相当の処理を行い, エラーなく読出し完了出来たか否かで判断します. また, この時に型のサイズの取得を行います. PathCheck=FALSE をしていた場合は初回アクセス時(読み込み, または書き込み)に型のサイズの取得を行います.

2.2.3.4. Elem オプション

Path オプションによりリクエストパスの末尾が配列指定の場合のみ有効.

変数で一度に扱うデータサイズは, 約 500Byte ですが, 送受信可能な配列数はタグ名長に依存します.

タグ名長を短くすることで送受信可能な配列数は増えます.

配列は一パケットに収まる単位に分割して送受信します.

一度に送受信可能な配列数の求め方は一度に送受信可能な配列サイズについては付録 C を参照してください.

2.2.4. CaoVariable:put_Value プロパティ

指定されたオプションに従い CIP の SendRRData に WriteTagService(0x4d)を乗せたパケットを送出し, コントローラ内のタグ定義の値に書き込みます.

配列に関しては Path オプションで書き込み開始要素番号を, Elem オプションで書き込み要素数を指定することで複数要素同時に書き込みを行うことが出来ます.

文字列型(STRING)に対するアクセスは数値型と異なり内部的に複数回書き込みパケットの送出を行います. 詳細に関しては付録 B を参照してください.

表 2-4 記載のデータ型以外を書き込む値の型に指定した場合は引数異常となります. 配列に対し書き込む際に Elem オプションで指定した書き込み要素数と書き込む値の要素数が異なる場合も引数異常として扱います.

2.2.5. CaoVariable:get_Value プロパティ

指定されたオプションに従い CIP の SendRRData に ReadTagService(0x4c)を乗せたパケットを送出し, コントローラ内のタグ定義の値を読み出します.

配列に関しては Path オプションで書き込み開始要素番号を, Elem オプションで書き込み要素数を指定することで複数要素同時に読出しを行うことが出来ます.

文字列型(STRING)に対するアクセスは数値型と異なり内部的に複数回読出しパケットの送出を行います. 詳細に関しては付録 B を参照してください.

2.3. 変数一覧

2.3.1. CaoController クラス

表 2-5 CaoController クラス システム変数一覧

変数名	データ型	説明	属性	
			get	put
@MAKER_NAME	VT_BSTR	メーカ名=”RockwellAutomation”を返す.	○	－
@VERSION	VT_BSTR	プロバイダバージョン情報.	○	－
@ERROR_STATUS	VT_I4 | VT_ARRAY	直前のエラー応答詳細情報. ([7]) [0] :Error Code [1] :Extended Error	○	－
@WROTE_STRING	VT_I4	文字列型書き込み時の書き込み完了要素数.	○	－

表 2-6 CaoController クラス ユーザ変数一覧

変数名	データ型	説明	属性	
			get	put
任意	変数型依存	コントローラ(ControlLogix/CompactLogix シリーズ)内のタグ定義にアクセスする.	○	○

[7] エラーの内容に関しては RockwellAutomation の“1756-pm020c-en-p(Data Access).pdf”の「Read Tag Service Error Codes」,「Write Tag Service Error Codes」の章を参照してください.

2.4. エラーコード

Logix5000 プロバイダでは，以下の固有エラーコードが定義されています．

ORiN2 共通エラーについては，「ORiN2 プログラミングガイド」のエラーコードの章を参照してください．

表 2-7 固有エラーコード

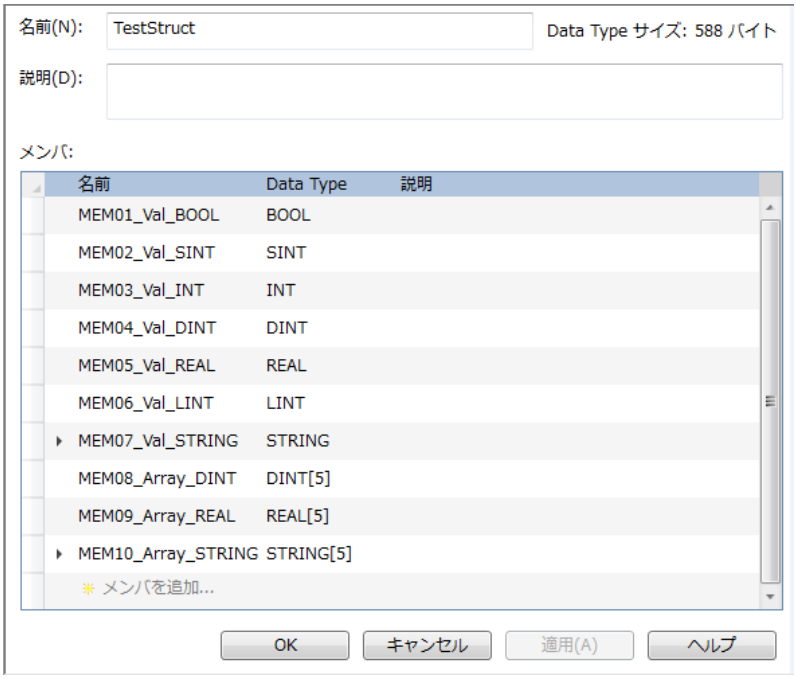
エラー名	エラー番号	説明
CIP 応答異常 (フォーマットエラー)	0x80100000	CIP 通信の応答パケットが想定外の異常なフォーマットであった場合に返ります．
CIP 応答異常 (Error Status Code)	0x80100001	CIP 通信の応答パケットがステータス異常であった場合に返ります．
CIP 応答異常 (Error General Status)	0x80100002	CIP 通信の応答パケットがエラー応答であった場合に返ります．[8] 要求時のリクエストパスやデータ型，アクセス要素数等が不正であった場合に本エラーが返ります． 本エラーを受信した場合は，システム変数の@ERROR_STATUS を読み出すことで詳細エラーコードを得ることができます． また，文字列型配列の書き込み時の場合はシステム変数の@WROTE_STRING を読み出すことで何要素目の書き込みでエラーとなったのか判断することができます．
RequestPath 異常	0x80100003	構造体情報取得時に指定したパスが見つからない場合に返ります．
構造体サイズが大きすぎる	0x80100004	構造体のサイズが大きすぎる場合に返ります． 構造体サイズを小さくしてください．
CIP 応答パケット不整合エラー	0x80100005	CIP 通信の応答パケットが想定外である状態が一定回数発生した場合に返ります．

[8] SlotNo が必要な機種(CompactLogix 含む)の場合にもこのエラーが発生します．設定が正しいと思われるのにこのエラーが発生する場合は SlotNo を指定してみてください．SlotNo が必要かどうかは各機種のマニュアルをご確認ください．

付録A. リクエストパス指定例

コントローラ上に以下に挙げる様な定義がされている場合に，各データへアクセスするためのリクエストパスの指定例を以下に示します．

[LogixDesigner で定義したユーザ定義構造体]



[LogixDesigner で定義したコントローラタグ]

名前	値	強制マスク	スタイル	Data Type
Val1	123		Decimal	DINT
Val2	'Test'	{...}		STRING
Val2.LEN	4		Decimal	DINT
Val2.DATA	{...}	{...}	ASCII	SINT[82]
Val3	{...}	{...}	Float	REAL[3]
Val3[0]	1.234		Float	REAL
Val3[1]	0.555		Float	REAL
Val3[2]	0.0		Float	REAL
Val4	0		Decimal	BOOL
Val5	{...}	{...}	Decimal	BOOL[160]
Val6	{...}	{...}		STRING[3]
Val6[0]	'Str1'	{...}		STRING
Val6[1]	'Str2'	{...}		STRING
Val6[2]	'Str3'	{...}		STRING
Val7	{...}	{...}		TestStruct
Val8	{...}	{...}		TestStruct[2]

例1) 数値型定義 (Val1)

Path=Val1, DataFormat=0

例2) 文字列型定義 (Val2)

Path=Val2, DataFormat=1

例3) 数値型配列 (Val3 の要素 0～2)

Path=(Val3[0]), DataFormat=0, Elem=3

例4) BOOL 型定義 (Val4)

Path=(Val4), DataFormat=0

例5) BOOL 型配列 (Val5 の要素 32～95)

Path=(Val5[1]) , DataFormat=0, Elem=2

例6) 文字列型配列 (Val6 の要素 1～2)

Path=(Val6[1]), DataFormat=1, Elem=2

例7) 構造体メンバ (Val7 の MEM03_Val_INT)

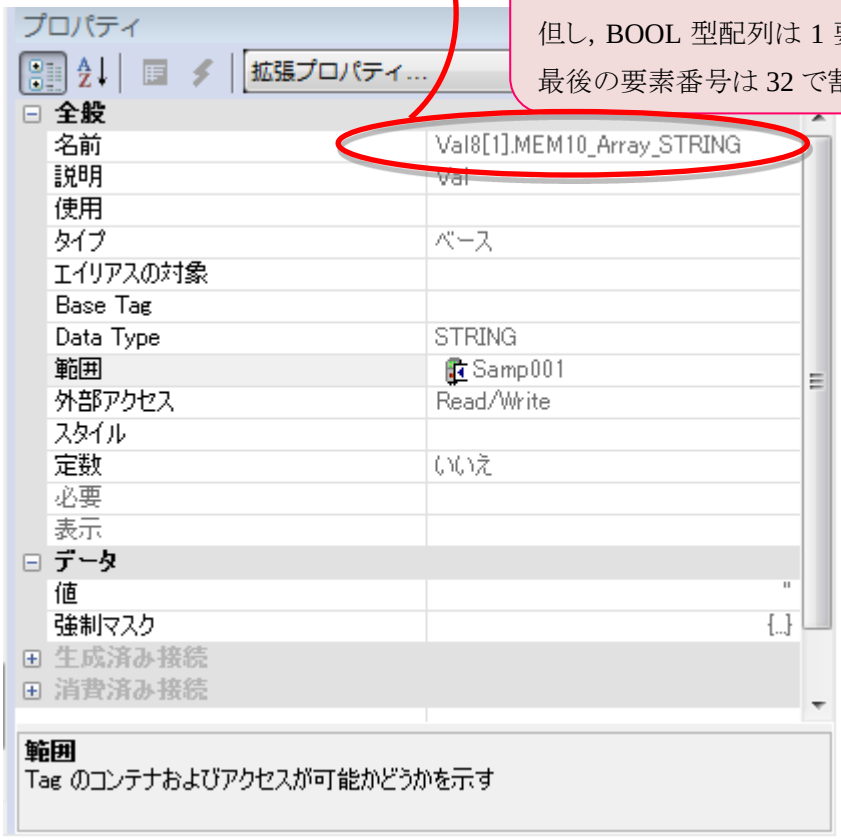
Path=Val7.MEM03_Val_INT, DataFormat=0

例8) パス途中に配列を含むパターン (Val8[1]の MEM10_Array_STRING の要素 0～2)

Path=(Val8[1].MEM10_Array_STRING[0]) , DataFormat=1, Elem=3

LogixDesigner 上の Tag 名をそのまま Path オプションに指定すれば OK

但し, BOOL 型配列は 1 要素 32 バイトでアクセスするため最後の要素番号は 32 で割った値を指定してください.



例9) 構造体 (Val7)

Path=Val7, DataFormat=2

例10) 構造体配列(Val8 の要素 0～1)

Path=(Val8[0]), DataFormat=2, Elem=2

例11) 数値型(Val1)

Path=Val1, DataFormat=2

例12) 文字列型(Val2)

Path=Val2, DataFormat=2

例13) 構造体メンバ(Val7 の MEM03_Val_INT)

Path=Val7.MEM03_Val_INT, DataFormat=2

例14) パス途中に配列を含むパターン (Val8[1]の MEM10_Array_STRING の要素 0～2)

Path=(Val8[1].MEM10_Array_STRING[0]) , DataFormat=2, Elem=3

付録B. 文字列型へのアクセスについて

文字列型(STRING)は LogixDesigner で定義すると内部に XXX.LEN と XXX.DATA[n]という二つのメンバを持った構造体のような形で作られます.

LEN は DINT 型で文字列長(バイト数)を, DATA は SINT 型の配列で文字データを ASCII にしたものをそれぞれ格納しています. (DATA の文字が格納されていない領域は NULL=0x00 で初期化されています)

String	'Test'	{...}		STRING
String.LEN	4		Decimal	DINT
String.DATA	{...}	{...}	ASCII	SINT[82]
String.DATA[0]	'T'		ASCII	SINT
String.DATA[1]	'e'		ASCII	SINT
String.DATA[2]	's'		ASCII	SINT
String.DATA[3]	't'		ASCII	SINT
String.DATA[4]	'$00'		ASCII	SINT
String.DATA[5]	'$00'		ASCII	SINT
String.DATA[6]	'$00'		ASCII	SINT

put_Value による書き込み時は“Path=(String.DATA[0]), DataFormat=0, Elem=書き込み文字数”指定相当の WriteTagService パケットを生成し, VT_BSTR で指定された文字列を ASCII としてセットし送出します. DATA 部書き込み成功後,“Path=String.LEN, DataFormat=0”指定相当の WriteTagService パケットを生成し, 文字列長をセットし送出します. (DATA→LEN と 2 回に分けて WriteTagService パケットの送出を行います)

get_Value による読み出し時は“Path=String.LEN, DataFormat=0”指定相当の ReadTagService パケットを生成・送出し, 文字列長を取得します. LEN 読み出し成功後,“Path=(String.DATA[0]), DataFormat=0, Elem=文字列長”指定相当の ReadTagService パケットを生成・送出し, 読み出したデータを連結して VT_BSTR で返します. (LEN→DATA と 2 回に分けて ReadTagService パケットの送出を行います)

書き込み/読み込み共に文字列型の配列へアクセスする場合は, 上記の処理をアクセス要素数分繰り返すことになります. (このため配列要素に複数同時アクセスを試みてもデータの同時性は保証されません)

付録C. 一度に送受信可能な配列サイズについて

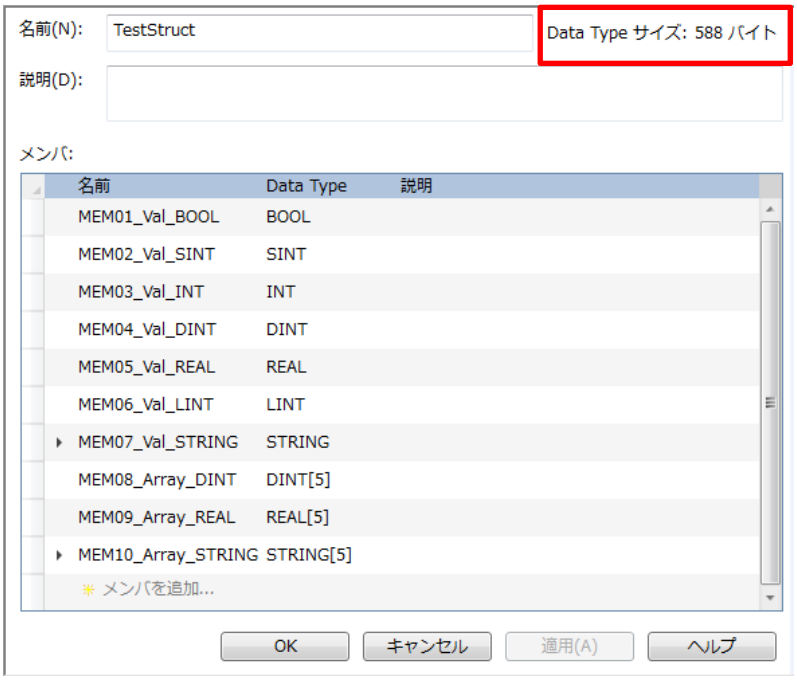
変数で一度に送受信可能な要素数は読み込み時と書込み時に可能な要素数の小さい方に合わせて送受信が行われます.

この要素数単位以下になるようにパケットが分割されて送信されます.

構造体の場合, 構造体サイズ含めて一パケット以内に収まらない場合はエラーとなります. 文字列型も構造体と同様の扱いとなります.

構造体のデータサイズは LogixDesigner から確認することができます.

[LogixDesigner で定義したユーザ定義構造体]



送受信可能なサイズは読み込み時はタグ名を使用しませんので固定サイズになり，書込み時はタグ名を使用しますので可変サイズとなります．

以下の表のようになります．

型(Data Type)	サイズ	最大読み込み要素数[9]	最大書き込み要素数 [9]
BOOL	4 Byte	123	(504 - リクエストパスサイズ) / 4
SINT	1 Byte	492	(504 - リクエストパスサイズ) / 1
INT	2 Byte	246	(504 - リクエストパスサイズ) / 2
DINT	4 Byte	123	(504 - リクエストパスサイズ) / 4
REAL	4 Byte	123	(504 - リクエストパスサイズ) / 4
DWORD	4 Byte	123	(504 - リクエストパスサイズ) / 4
LINT	8 Byte	61	(504 - リクエストパスサイズ) / 8
STRING[10]	1 Byte[11]	492	(504 - リクエストパスサイズ[12]) / 1
構造体 [10]	x Byte[13]	492 / 構造体サイズ	(502 - リクエストパスサイズ) / 構造体サイズ

[9] 表は Compact Logix です．Control Logix は Compact Logix より送受信可能サイズが 10 バイト小さくなります．

[10] DataFormat=2 を指定した場合，STRING は構造体(LEN(4 バイト)，DATA(n バイト))の扱いになります．
DataFormat=1 と異なり，有効文字列長(LEN)の如何に関わらず全データ取得しますので注意してください．

[11] STRING 型は要素数ではなく，定義文字数がサイズになるので注意してください．
最大文字数を 100 に設定した場合は，サイズが 100Byte になります．

[12] STRING 型はタグ名.Data[n]にアクセスするため他の型と違い，リクエストパスサイズが 10 バイト分長くなります．

[13] 構造体は各メンバ変数で 4 バイトになるようにパディングされます．
INT 型要素と DINT 型の 2 つのメンバを持つ構造体の場合，構造体サイズは 8 バイト(INT(2 バイト)+Padding(2 バイト)+DINT(4 バイト))になります．

[読み込み可能サイズの算出方法]

・受信可能データサイズ

非構造体:492 バイト

構造体,, STRING 型:492 バイト

・受信可能要素数

受信可能データサイズを型サイズで割った商

例1) DINT 型

受信可能要素数 ＝ 受信可能データサイズ(492) / 型サイズ(4) ＝ 123

[書き込み可能サイズの算出方法]

・送信可能データサイズ

非構造体:504 バイト

構造体, STRING:502 バイト

・リクエストパスサイズ

リクエストパスをドット(“.”)単位で分解します(分解した要素をリクエストパス要素とする).

リクエストパス要素:

[非配列部]

リクエストヘッダ(2) + 要素文字列長 + 偶数パディング(1)

[配列部]

リクエストヘッダ(2) + 要素文字列長 + 偶数パディング(1) ＋ 配列指定サイズ(4)

リクエストパスサイズ:

リクエストパスサイズ = リクエストパス要素[0] + リクエストパス要素[1] + … + リクエストパス要素[n]

例1) DINT 型

Path=Val5.MEM03_Val_DIN[1], DataFormat=0, Elem=2

データ要素[0]:6 バイト

リクエストヘッダ(2) + 要素文字列長(4) + 偶数パディング(0)

データ要素[1]:20 バイト

リクエストヘッダ(2) + 要素文字列長(13) + 偶数パディング(1) + 配列指定サイズ(4)

送信可能要素数 = 送信可能データサイズ(502) – リクエストパスサイズ(26) / 型サイズ(4) = 119