Panasonic

FPWIN Pro7 虎の巻

2024.03 | Panasonic Industry Co., Ltd.

memo

目次

シリアル/Ethernet 通信

MEWTOCOL-COM マスタ命令 FP_WRITE_TO_SLAVE_AREA_OFFS

MEWTOCOL-COM を使用してデータ送信を行う場合、使用する命令は下記の 2 種類があります。
・FP_WRITE_TO_SLAVE
・FP_WRITE_TO_SLAVE_AREA_OFFS
ここでは、**FP_WRITE_TO_SLAVE_AREA_OFFS** を説明します。

例）10word 分のデータを、スレーブ側アドレス DT100 を先頭に転送する場合。

■POU ヘッダ（Prog）

	クラス	変数名	データ型	初期値	コメント
1	VAR	aiWriteSource	ARRAY [0..9] OF INT	[10(0)]	
2	VAR	iResult	INT	0	
3	VAR	iSlaveMemoryOffset	INT	100	
4	VAR	iStationNo	INT	1	
5	VAR	iWriteWordBit	INT	10	

■POU ボディ（Prog）

```
FP_WRITE_TO_SLAVE_AREA_OFFS(Port := SYS_ETHERNET_USER_CONNECTION_1,
                            StationNumber := iStationNo,
                            Source := aiWriteSource,
                            Words_Bits := iWriteWordBit,
                            SlaveMemoryArea := SYS_MEMORY_AREA_DT,
                            SlaveMemoryOffset := iSlaveMemoryOffset,
                            Result => iResult);
```

◎送信ポートの指定：”Port”＋”StationNumber”
Port　　・システム変数でポート番号／ユーザコネクション No を指定します。
StationNumber　　・送信先の MEWTOCOL 局番を指定します。

◎送信元の指定：”Source”＋”Words_Bits”
Source　　・送信するデータの格納先を指定します。
Words_Bits　　・データ書き込みの場合：データ数（ワード数）を指定します。
　　・ビット書き込みの場合：16#8000 を指定します。（固定）

◎送信先の指定：”SlaveMemoryArea”＋” SlaveMemoryOffset”
SlaveMemoryArea　　・システムメモリエリアで送信先のデバイス種別を指定します。
SlaveMemoryOffset　　・送信先先頭デバイスのオフセット値を指定します。

※上例の送信先が DT100 を先頭とする場合、”SlaveMemoryArea”に SYS_MEMORY_AREA_DT、
” SlaveMemoryOffset”に 100 を指定します。（上例では変数初期値に代入）

MEWTOCOL-COM マスタ命令 FP_WRITE_TO_SLAVE

MEWTOCOL-COM を使用してデータ送信を行う場合、使用する命令は下記の 2 種類があります。
・FP_WRITE_TO_SLAVE
・FP_WRITE_TO_SLAVE_AREA_OFFS
ここでは、**FP_WRITE_TO_SLAVE** を説明します。

例）10word 分のデータを、スレーブ側アドレス DT100 を先頭に転送する場合。

■グローバル変数

	クラス	変数名	FPアドレス	IECアドレス	データ型	初期値	外..	コメント
1	VAR_GLOBAL	g_aiWriteAdless	DT100	%MW5.100	ARRAY [0..9] OF INT	[10(0)]		

■POU ヘッダ（Prog）

	クラス	変数名	データ型	初期値	コメント
1	VAR_EXTERNAL	g_aiWriteAdless	ARRAY [0..9] OF INT	[10(0)]	
2	VAR	aiWriteSource	ARRAY [0..9] OF INT	[10(0)]	
3	VAR	iResult	INT	0	
4	VAR	iStationNo	INT	1	
5	VAR	iTest	INT	0	

■POU ボディ（Prog）

```
FP_WRITE_TO_SLAVE(Port := SYS_ETHERNET_USER_CONNECTION_1,
                  StationNumber := iStationNo,
                  Source := aiWriteSource,
                  SlaveAddress => g_aiWriteAdless,
                  Result => iResult);
```

◎送信ポートの指定：”Port”＋”StationNumber”
Port　　・システム変数でポート番号／ユーザコネクション No を指定します。
StationNumber　　・送信先の MEWTOCOL 局番を指定します。

◎送信元の指定：”Source”
Source　　・送信するデータの格納先を指定します。
　　※送信データサイズは、”Sours”で指定した変数のデータサイズがそのまま適用されます。

◎送信先の指定：”SlaveAddress”
SlaveAddress　　・送信先デバイス＋アドレスの一致する変数で指定します。

※上例の場合、送信先の指定用として「DT100 を先頭とした 10word 分の領域」としてグローバル変数”g_awiWriteAdless”を割り当てる必要があります。

汎用通信受信バッファ用の STRING 型の宣言（FP7 以外の機種）

汎用通信を使用する場合、FP7 以外の機種はシステムレジスタで受信バッファの設定をする必要があります。
また、FPWIN Pro7 では同時に受信バッファに設定したアドレスをグローバル変数で宣言する必要があります。
ここでは、グローバル変数に STRING 型を設定した場合の例を紹介します。

■受信バッファの設定
「プロジェクトペイン」ー「PLC」ー「システムレジスタ」ー「シリアルポート」ー「COM1」

416	ツールポート受信バッファ先頭アドレス	101
417	ツールポート 受信バッファ容量	17

上記設定では、受信文字数格納先 :1 ワード 、受信文字列格納先 :16 ワードとなり、32 文字までの受信が可能です。
設定した受信バッファには、受信時に文字列が以下の様に格納されます。

	上位バイト	下位バイト
DT101	受信文字数	
DT102	2 文字目	1 文字目
・ ・ ・	・ ・ ・	
DT116	30 文字目	29 文字目
DT117	32 文字目	31 文字目

■受信バッファ用グローバル変数の宣言

	クラス	変数名	FPアドレス	IECアドレス	データ型	初期値
1	VAR_GLOBAL	g_sReceiveBuffer	DT100	%MW5.100	STRING[32]	''

受信バッファを STRING 型で宣言する場合、グローバル変数の FP アドレスは「**受信バッファ先頭アドレスー1**」を設定します。

受信バッファ

	上位バイト	下位バイト
DT100	最大格納文字数	
DT101	受信文字数	
DT102	2 文字目	1 文字目
・ ・ ・	・ ・ ・	
DT116	30 文字目	29 文字目
DT117	32 文字目	31 文字目

STRING 型データ範囲

受信バッファ用グローバル変数を宣言する際、最大格納文字数が受信バッファの容量を上回るようにしてください。
今回の設定の様に受信バッファ容量を 17 に設定した場合、STRING 型の文字数を（17ー1）×2＝32 に設定してください。

汎用通信受信バッファ用の ARRAY 型の宣言（FP7 以外の機種）

汎用通信を使用する場合、FP7 以外の機種はシステムレジスタで受信バッファの設定をする必要があります。
また、FPWIN Pro7 では同時に受信バッファに設定したアドレスをグローバル変数で宣言する必要があります。
ここでは、グローバル変数に ARRAY 型を設定した場合の例を紹介します。

■受信バッファの設定
「プロジェクトペイン」ー「PLC」ー「システムレジスタ」ー「シリアルポート」ー「COM1」

416	ツールポート受信バッファ先頭アドレス	101
417	ツールポート 受信バッファ容量	17

上記設定では、受信文字数格納先 :1 ワード 、受信文字列格納先 :16 ワードとなり、32 文字までの受信が可能です。
設定した受信バッファには、受信時に文字列が以下の様に格納されます。

	上位バイト	下位バイト
DT101	受信文字数	
DT102	2 文字目	1 文字目
・ ・ ・	・ ・ ・	
DT116	30 文字目	29 文字目
DT117	32 文字目	31 文字目

■受信バッファ用グローバル変数の宣言

	クラス	変数名	FPアドレス	IECアドレス	データ型	初期値
1	VAR_GLOBAL	g_awReceiveBuffer	DT101	%MW5.101	ARRAY [0..16] OF WORD	[17(0)]

受信バッファを ARRAY 型で宣言する場合、グローバル変数の FP アドレスは「**受信バッファ先頭アドレス**」を設定します。

受信バッファ

	上位バイト	下位バイト
DT101	受信文字数	
DT102	2 文字目	1 文字目
・ ・ ・	・ ・ ・	
DT116	30 文字目	29 文字目
DT117	32 文字目	31 文字目

ARRAY 型データ範囲

受信バッファ用グローバル変数を宣言する際、最大格納文字数が受信バッファの容量を上回るようにしてください。
今回の設定の様に受信バッファ容量を 17 に設定した場合、ARRAY 型の配列数を 17 に設定してください。

STRING 型変数のシリアル通信(送信の場合)・・・(FP7 以外の機種の場合)

FPWIN Pro7 では STRING 型変数を使ってシリアル通信をすることが出来ます。
ここでは FP シリーズ(FP7 以外の機種)で"SendCreaters"命令を使用して、COM1 ポートから STRING 型変数をシリアルで送信する例を紹介します。

■POU ヘッダ(Prog)

	クラス	変数名	データ型	初期値	コメント
1	VAR	bStart	BOOL	FALSE	
2	VAR	sDataString	STRING[32]	'abcd'	
3	VAR	sSendBuffer	STRING[32]	''	

■POU ボディ(Prog)

```
IF (DF(bStart) AND sys_bIsComPort1TransmissionDone) THEN
  SendCharacters(Port := SYS_COM1_PORT,
                 sString := sDataString,
                 bSuppressEndCode := FALSE,
                 SendBuffer := sSendBuffer);
END_IF;
```

「sys_bIsComPort1TransmissionDone」は COM ポート 1 の送信完了信号になり、FP0H では「R9138」が割り当たっています。

「SendCharacters」命令の実行条件に、送信完了フラグのTRUEを起動条件として記述します。

STRING 型変数のシリアル通信（受信の場合）・・・（FP7 以外の機種の場合）

FPWIN Pro7 では STRING 型変数を使ってシリアル通信をすることが出来ます。
ここでは FP シリーズ（FP7 以外の機種）で”ReceiveCharacters”命令を使用して、COM1 ポートから STRING 型変数をシリアルで受信する例を紹介します。

■システムレジスタ設定

416	ツールポート受信バッファ先頭アドレス	2048		0 - 32764
417	ツールポート 受信バッファ容量	17	ワード	0 - 2048

■グローバル変数

	クラス	変数名	FPアドレス	IEC...	データ型	初期値	外..	コメント
1	VAR_GLOBAL	g_sReceiveBuffer	DT2047	%M...	STRING[32]	''		

■POU ヘッダ（Prog）

	クラス	変数名	データ型	初期値	コメント
1	VAR	sReceiveData	STRING[32]	''	

■POU ボディ（Prog）

```
IF (DF(sys_bIsComPort1ReceptionDone)) THEN
  sReceiveData := ReceiveCharacters(SYS_COM1_PORT);
  ClearReceiveBuffer(SYS_COM1_PORT);
END_IF;
```

「sys_bIsComPort1ReceptionDone」は COM1 ポートの受信完了フラグとなり、FP0H では「R913A」が割り当たっています。

終端コードを受信することで、受信完了フラグが TRUE となります。
この受信完了フラグの立ち上がりで「ReceiveCharacters」命令を実行し、受信バッファに書き込まれたデータを取り込みます。

受信完了フラグが TRUE の状態では、次のデータを受信することができません。
「ClearReceiveBuffer」命令を実行し、受信バッファをクリアすることで受信完了フラグを FALSE にします。

「SendCharacters」命令を実行することでも受信バッファのクリアは可能ですが、
受信のみを行う場合には「ClearReceiveBuffer」命令の実行が必要になります。

STRING 型変数のシリアル通信(送信の場合)・・・(FP7 の場合)

FPWIN Pro7 では STRING 型変数を使ってシリアル通信をすることが出来ます。
ここでは FP7 で"SendCreaters"命令を使用して、COM1 ポートから STRING 型変数をシリアルで送信する例を紹介します。

■POU ヘッダ(Prog)

	クラス	変数名	データ型	初期値	コメント
1	VAR	bStart	BOOL	FALSE	
2	VAR	sDataString	STRING[32]	'abcd'	
3	VAR	sSendBuffer	STRING[32]	''	

■POU ボディ(Prog)

```
IF (DF(bStart) AND sys_bIsComPort1ProgramControlled
   AND NOT(sys_bIsComPort1ProgramControlledActive))  THEN
   SendCharacters(Port := SYS_COM1_PORT,
                  sString := sDataString,
                  bSuppressEndCode := FALSE,
                  SendBuffer := sSendBuffer);
END_IF;
```

「sys_bIsComPort1ProgramControlled」は COM1 の送信可フラグ、
「sys_bIsComPort1ProgramControlledActive」は COM1 の送信中フラグとなります。
FP7 CPU ユニットの開始ワードアドレスを「0」とした場合、それぞれ「X8」「Y8」が割り当たります。

「SendCharacters」命令の実行条件に、送信可フラグの TRUE と送信中フラグの FALSE を起動条件として記述します。

STRING 型変数のシリアル通信（受信の場合）・・・（FP7 の場合）

FPWIN Pro7 では STRING 型変数を使ってシリアル通信をすることが出来ます。
ここでは FP7 で”ReceiveCharactors”命令を使用して、COM1 ポートから STRING 型変数をシリアルで受信する例を紹介します。

■POU ヘッダ（Prog）

	クラス	変数名	データ型	初期値	コメント
1	VAR	sReceiveData	STRING[32]	''	

■POU ボディ（Prog）

```
IF (DF(sys_bIsComPort1ReceptionDone)) THEN
   sReceiveData := ReceiveCharacters(SYS_COM1_PORT);
END_IF;
```

「sys_bIsComPort1ReceptionDone」は COM1 ポートの受信完了フラグとなり、
FP7 CPU ユニットの開始ワードアドレスを「0」とした場合、「X0」が割り当たります。

終端コードを受信することで、受信完了フラグが TRUE となります。
この受信完了フラグの立ち上がりで「ReceiveCharacters」命令を実行し、受信バッファに書き込まれたデータを
取り込みます。

FP7 ではシステムレジスタでの受信バッファの設定は必要ありません。
「ReceiveCharacters」命令の実行で、受信バッファをクリアすることで受信完了フラグが FALSE になるため
「ClearReceiveBuffer」命令の実行も必要ありません。

STRING 型に文字列終端コードを付加する

STRING 型変数にプログラム内で文字列終端コードを付加することが出来ます。
ここでは、CR(Hex 0D)を付加する例で紹介します。

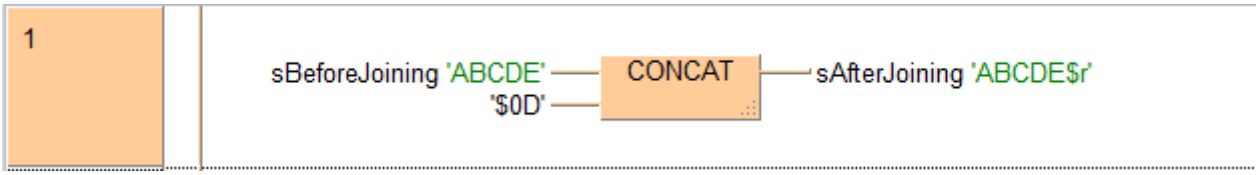
■POU ヘッダ(Prog)

	クラス	変数名	データ型	初期値	コメント
1	VAR	sBeforeJoining	STRING[32]	'ABCDE'	
2	VAR	sAfterJoining	STRING[32]	''	

■POU ボディ(Prog) ST 言語で記述した場合

```
sAfterJoining := CONCAT(sBeforeJoining,'$0D');    'ABCDE$r'  'ABCDE'
```

■POU ボディ(Prog) LD 言語で記述した場合



「CONCAT」命令を使用して文字列を結合します。
「CR」の ASCII コード:「0D」は、文字列表記「$0D」でプログラム中に記述します。

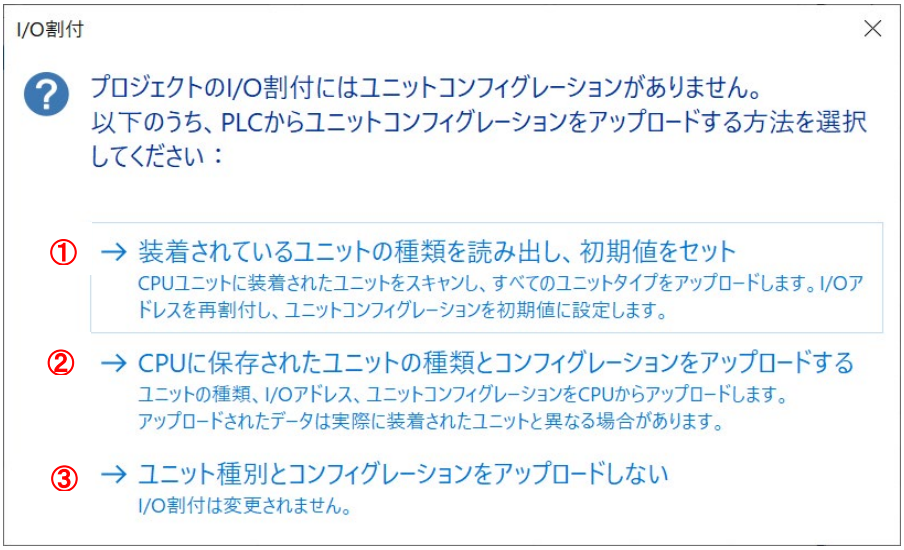
●補足
FPWIN Pro7 モニタ上の「$r」表記は終端コード「CR」を表しています。

FP7 本体の通信ポートを使用した通信プログラム作成時の必須設定項目

新プロジェクト構成時に最初に設定すべきことは、IO マップの登録です。
CPU ユニットのみであっても登録してください。
PRO の場合、IO マップが登録されていなくても、
CPU 内蔵ユニット(シリアル通信やイーサネットのこと)の制御 IO がデフォルトでワード番号 475 からの割り付けになっています。
何も登録せずに、PLC にダウンロードすると、何も IO マップに登録されていない情報が PLC にダウンロードされます。
注)オンライン切替時の下記のダイアログで③を選択した場合

PLC は、IO マップに何も登録されていない場合、CPU ユニット内蔵ユニットの制御 IO は 0 から始まるようになっているので
PLC 内のシリアル通信やイーサネット通信用の制御 IO 番号と PRO でシステムリレーを用いて作成したプログラムの制御 IO 番号がずれてしまうことにより、期待した制御ができなくなります。

IO マップに何も登録がない場合、オンライン切替時に下記のダイアログが表示されます。3つの選択肢があります。



① を選択した場合の動作
PLC に実装登録させた後、読み出して CPU ユニットの IO 開始番号を PRO のデフォルトで変更し、
装着されている全ユニットの IO 番号もスロットの IO 番号が 0 から始まるように調整し、PLC にダウンロードする。

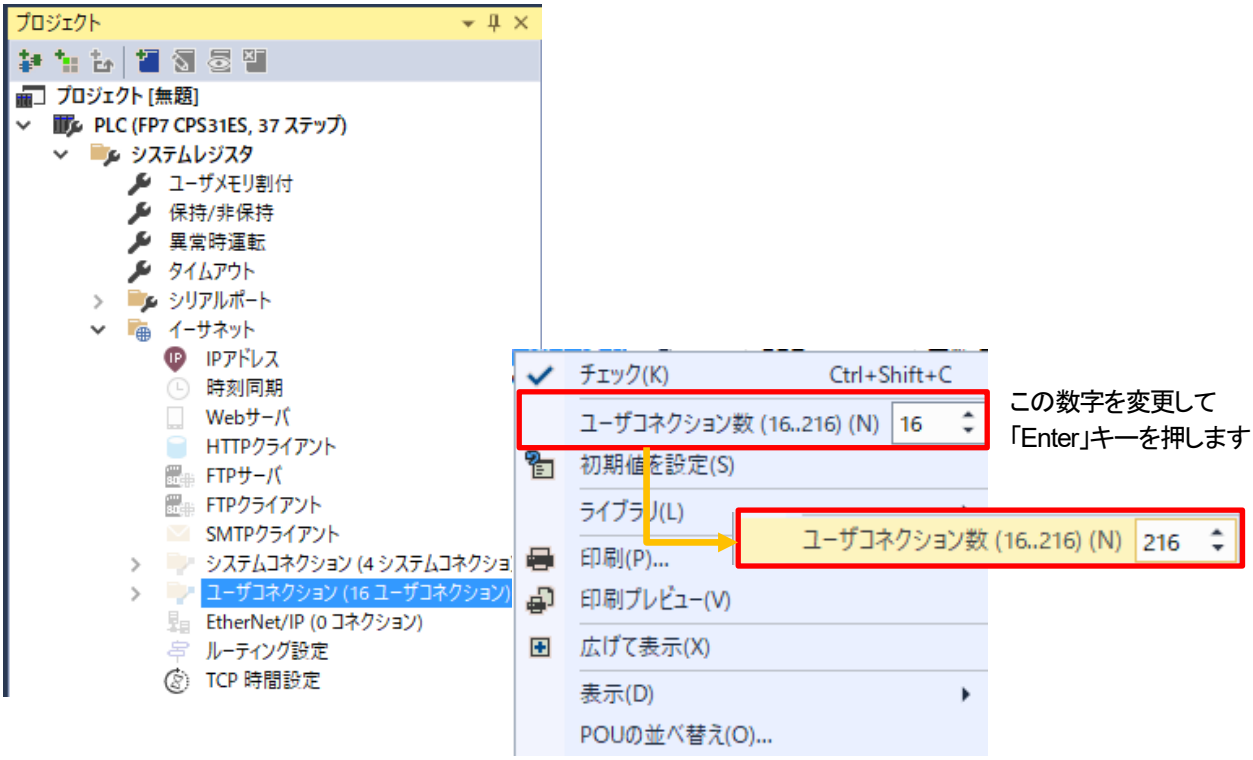
② を選択した場合の動作
PLC に登録されている IO マップを読み出して、そのままの値を採用し、ダウンロードする。PLC の設定値は不変

③ を選択した場合の動作
PLC の情報は読み出さず、PRO に登録されている値をダウンロードします。PLC の IO マップ情報は、PRO の設定値と同じになります。

FP7 ユーザコネクション数の拡張方法

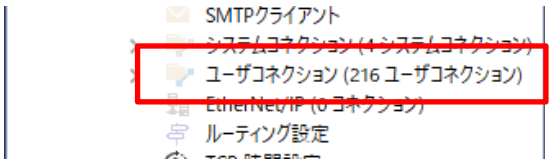
FP7 のユーザコネクションの数は初期値:16 ですが、最大 216 まで拡張が可能です。

FPWIN Pro7 にてこの設定を変更するには、
「プロジェクトペイン」ー「システムレジスタ」ー「イーサネット」ー「ユーザコネクション」を右クリックし、
設定ウインドウを開いてコネクション数を確定します。



この数字を変更して
「Enter」キーを押します

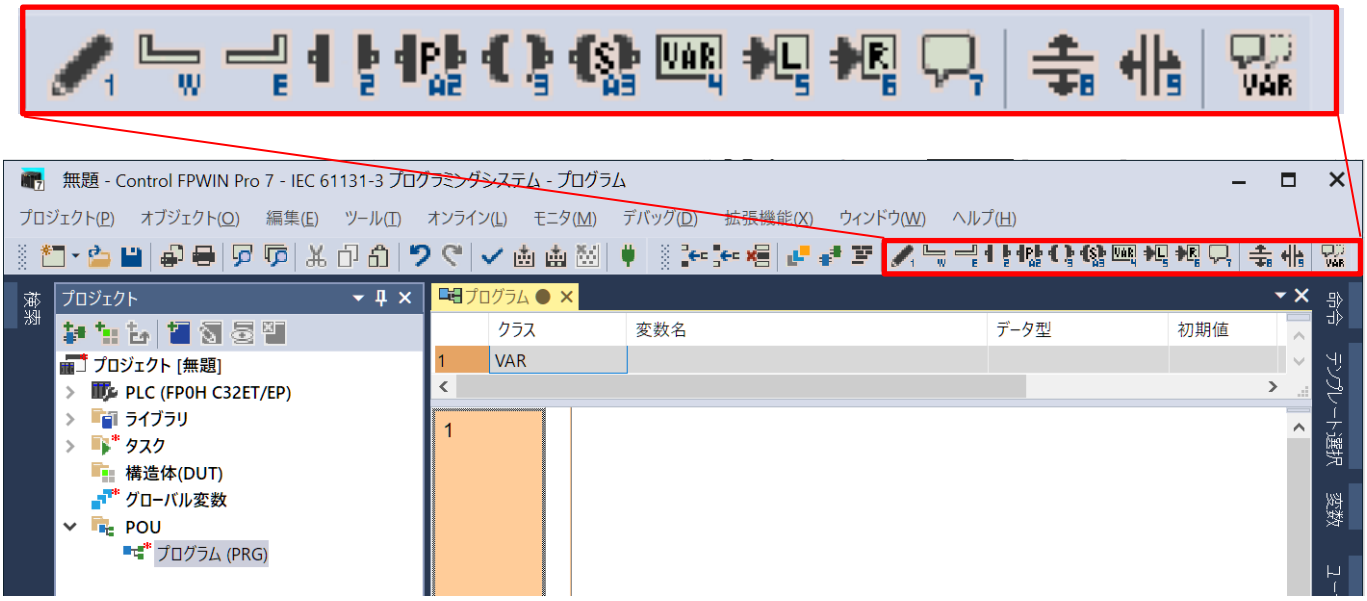
上記の手順でユーザコネクション数を「216」と設定すると、プロジェクトペイン内の表示が
「216 ユーザコネクション」と変更されます。



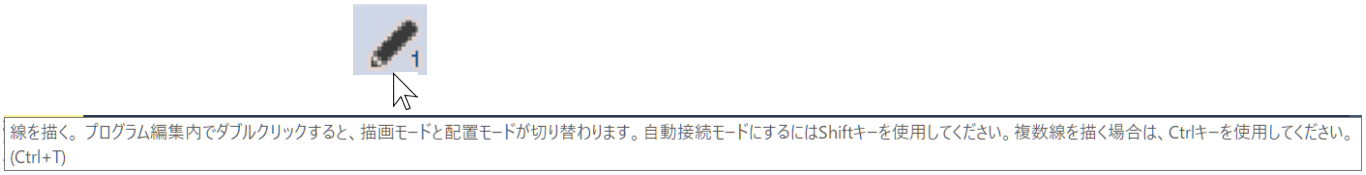
LD 言語

ショートカットキーによるプログラミング（LD）

LD（ラダーダイアグロム）でプログラムを作成する場合、a 接点などの基本的な命令のショートカットを使用することができます。



それぞれのショートカットアイコンにカーソルを合わせることで、説明を表示できます。



ショートカットアイコンの右下の数字や文字はショートカットキーを表します。
例えば、描画モード ⇔ 配置モード を変更する場合、数字の 1 キーを押すことでも変更できます。

A2 は Alt キーを押しながら数字の 2 キーを押すという意味です。

ショートカットキーを活用することで、プログラムの入力速度向上に繋がります。

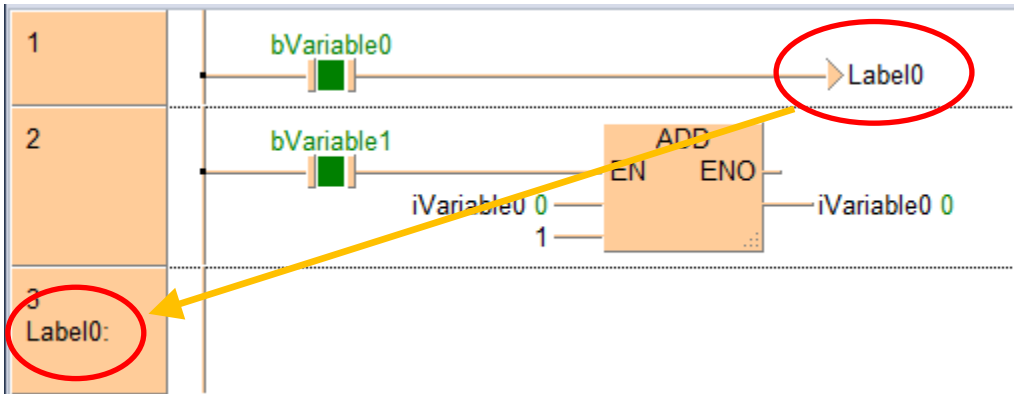
ー>Label の使用例

ー>ラベルを使用することによって、繰り返し処理やブロック間をジャンプすることができます。
ここでは、ラベルを用いてネットワークをジャンプする例で紹介します。

■POU ヘッダ（Prog）

	クラス	変数名	データ型	初期値
1	VAR	bVariable0	BOOL	FALSE
2	VAR	bVariable1	BOOL	FALSE
3	VAR	iVariable0	INT	0

■POU ボディ（Prog）



●上図プログラムの動作
“bVariable0”がON している間は、“bVariable0”がON してもネットワーク2のプログラムは実行されません。
ー>Label0 を実行することで、ネットワーク 3（Label0）にジャンプしています。

ある条件で目的のネットワーク（1つ、もしくは連続したネットワーク）に対して「動作させる」「動作させない」を
決めたい場合などに使用すると非常に便利です。
また、逆に 1 スキャン内でくり返し処理を行いたい場合などにも応用できます。

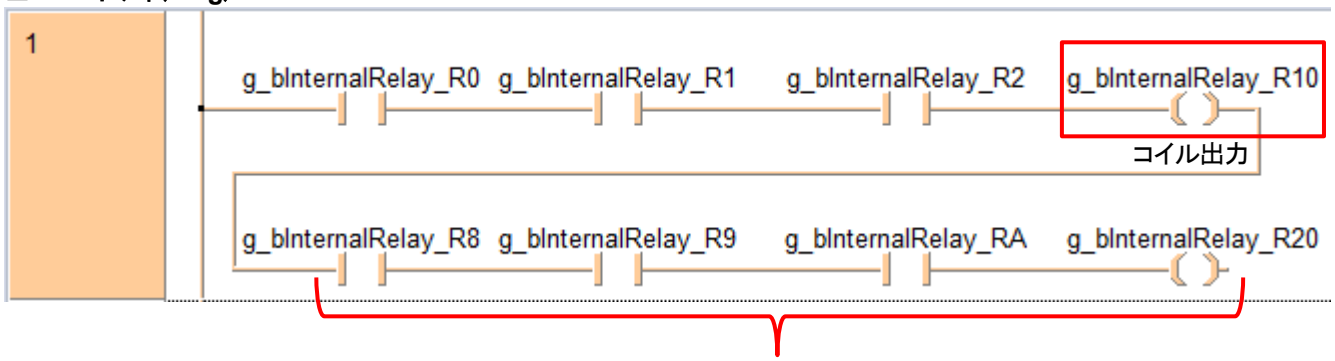
コイル（出力）の後ろにプログラム可能

FPWIN Pro7 ではコイル（出力）の後ろに続けてプログラムを記述することが可能です。
ここでは、コイルの後ろに接点とコイルを記述する例で紹介します。

■グローバル変数

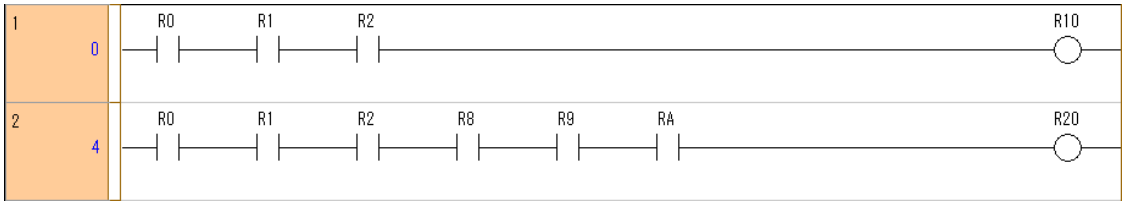
	クラス	変数名	FPアドレス	IECアドレス	データ型	初期値
1	VAR_GLOBAL	g_bInternalRelay_R0	R0	%MX0.0.0	BOOL	FALSE
2	VAR_GLOBAL	g_bInternalRelay_R1	R1	%MX0.0.1	BOOL	FALSE
3	VAR_GLOBAL	g_bInternalRelay_R2	R2	%MX0.0.2	BOOL	FALSE
4	VAR_GLOBAL	g_bInternalRelay_R8	R8	%MX0.0.8	BOOL	FALSE
5	VAR_GLOBAL	g_bInternalRelay_R9	R9	%MX0.0.9	BOOL	FALSE
6	VAR_GLOBAL	g_bInternalRelay_RA	RA	%MX0.0.10	BOOL	FALSE
7	VAR_GLOBAL	g_bInternalRelay_R10	R10	%MX0.1.0	BOOL	FALSE
8	VAR_GLOBAL	g_bInternalRelay_R20	R20	%MX0.2.0	BOOL	FALSE

■POU ボディ（Prog）



コイル（出力）の後ろにプログラムが可能

■コンパイル後のプログラム（FPWIN GR7 ラダー表記）



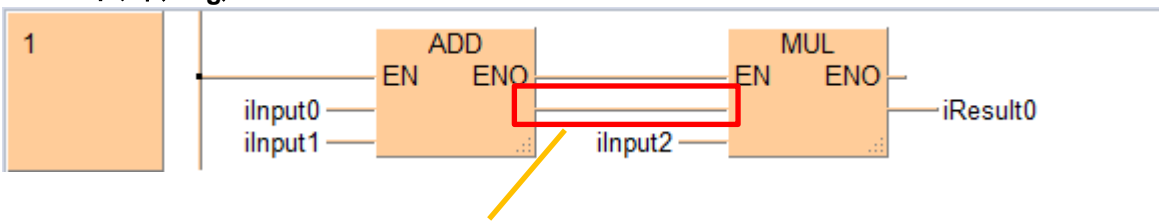
3 項以上の四則演算①

3項以上の四則演算は、プログラムでよく使用されます。
FPWIN Pro7 では計算用のレジスタを使用することなく計算結果を得ることができます。
ここでは(A＋B)×C＝D の計算を行う例で紹介します。

■POU ヘッダ(Prog)

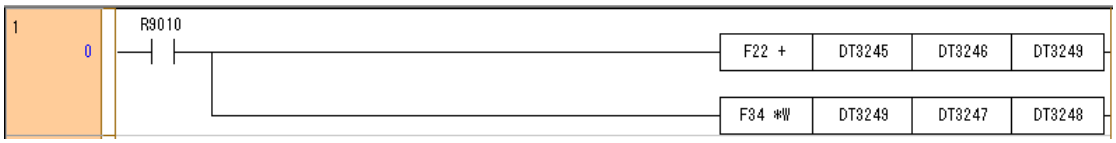
	クラス	変数名	データ型	初期値
1	VAR	iInput0	INT	0
2	VAR	iInput1	INT	0
3	VAR	iInput2	INT	0
4	VAR	iResult0	INT	0

■POU ボディ(Prog)



ADD 命令の出力と MUL 命令の入力をつなぐことで、
途中の計算結果用の変数(レジスタ)を宣言することなく結果(iResult_0)を得られます。

■コンパイル後のプログラム(FPWIN GR7 ラダー表記)



ADD 命令、MUL 命令共に入力変数に INT 型を使用した場合、コンパイルで F22 命令(16 ビット加算)と
F34 命令(16 ビット乗算)が割り当たります。

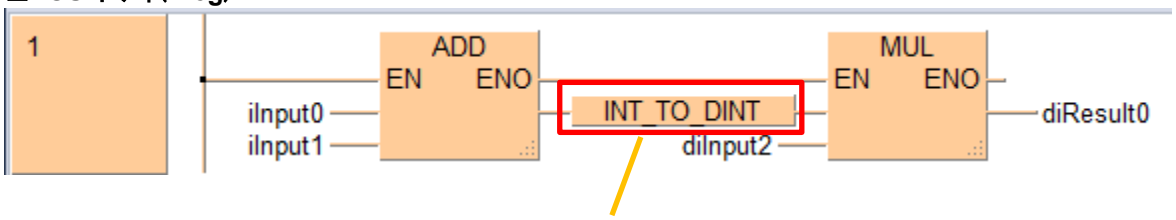
3 項以上の四則演算②

3項以上の四則演算で、複数のデータ型変数が混在する場合の例です。

■POU ヘッダ(Prog)

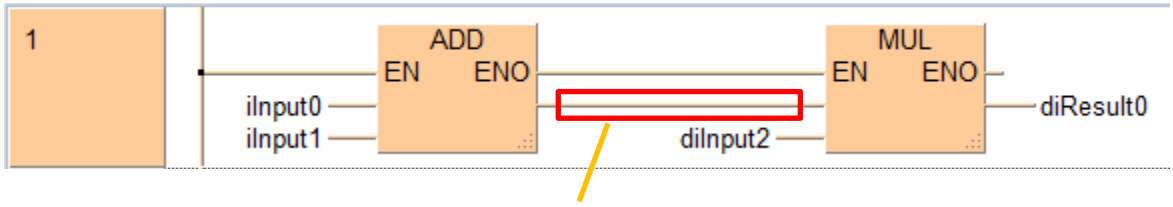
	クラス	変数名	データ型	初期値
1	VAR	iInput0	INT	0
2	VAR	iInput1	INT	0
3	VAR	diInput2	DINT	0
4	VAR	diResult0	DINT	0

■POU ボディ(Prog)

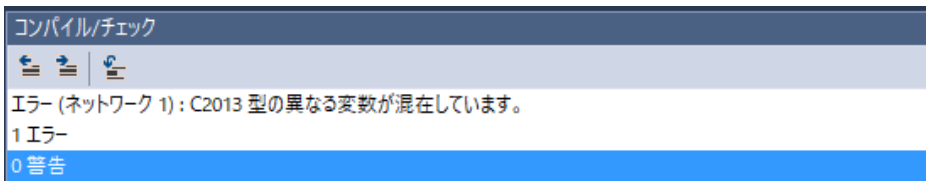


“INT_TO_DINT”を使用することでADD出力(INT型) ⇒ MUL入力(DINT型)の型変換を行います。

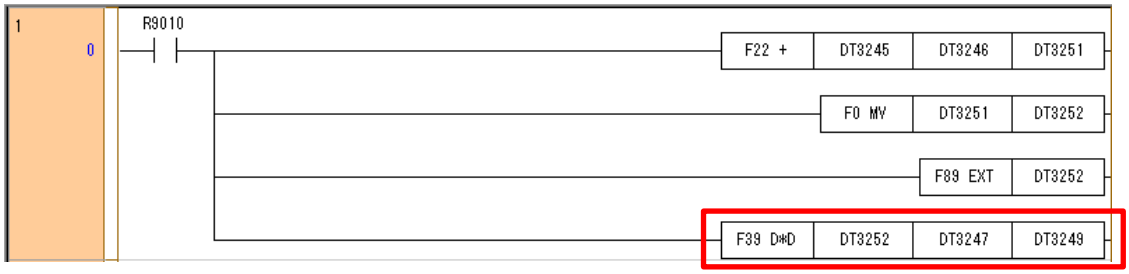
●補足



“INT_TO_DINT”を使用しない場合、コンパイルエラーが発生します。



■コンパイル後のプログラム(FPWIN GR7 ラダー表記)



MUL 命令は、入力に DINT 型が使用されているとコンパイルで F39 命令(32 ビット乗算)が割り当てられます。
この命令は入力に 32 ビットデータを使用しなければならないため、INT 型が計算結果で出力される演算子を割り当てるとコンパイルエラーが発生します。

比較演算子を使用する場合の注意点

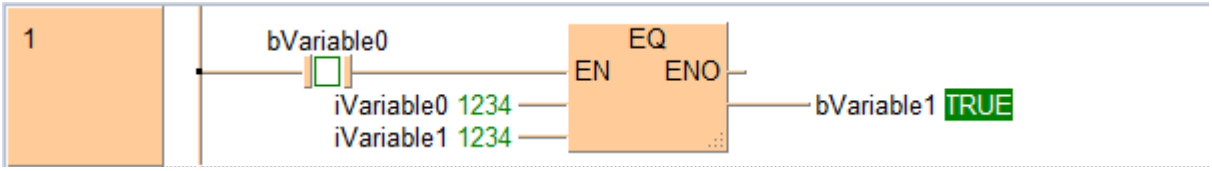
FPWIN Pro7では、LD、FBD言語において“＝”(EQ)，”≠”(NE)，“＞”(GT)，”≧”(GE)，“＜”(LT) ，”≦”(LE)といった比較演算子が用意されていますが、条件の成立時に保持するようになっています。

ここでは”＝”(EQ[イコール])の場合についての例で紹介します。

■POU ヘッダ(Prog)

	クラス	変数名	データ型	初期値
1	VAR	bVariable0	BOOL	FALSE
2	VAR	bVariable1	BOOL	FALSE
3	VAR	iVariable0	INT	0
4	VAR	iVariable1	INT	0

■POU ボディ(Prog)

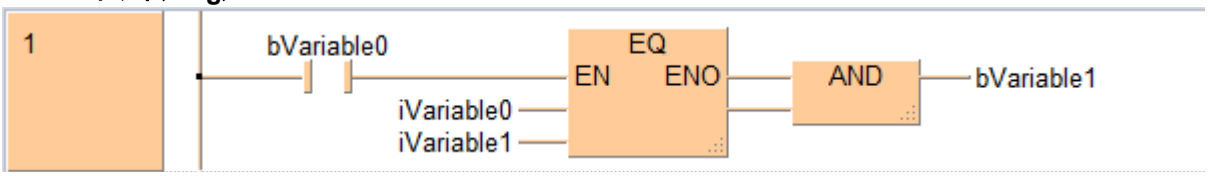


“bVariable0”がONして、比較データが”＝”(イコール)であった場合、”bVariable1”はONしますが、“bVariable0”が OFF しても”bVariable1”の ON 状態は保持したままとなります。

●条件に連動して比較出力を行う

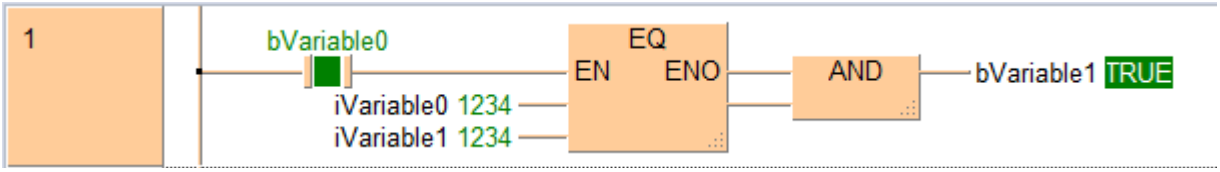
”bVariable0”の状態に応じて、比較出力を行いたい場合の例を示します。

■POU ボディ(Prog)

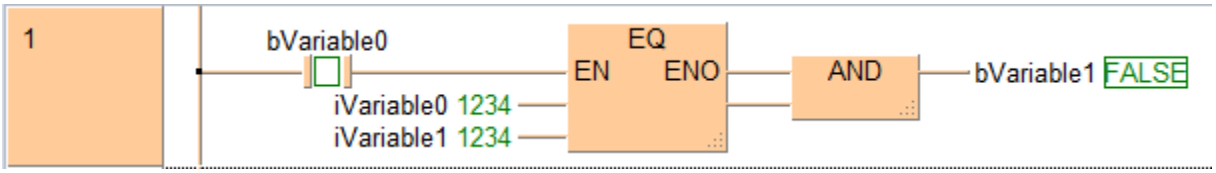


比較演算子の出力に”AND”を付けることで実現できます。

動作確認



実行条件”bVariable0”がONした時、比較結果がイコールであれば”bVariable1”はONします。



実行条件”bVariable0”が OFF した時、”bVariable1”は OFF します。

比較演算子で入力点数が 3 個以上の動作

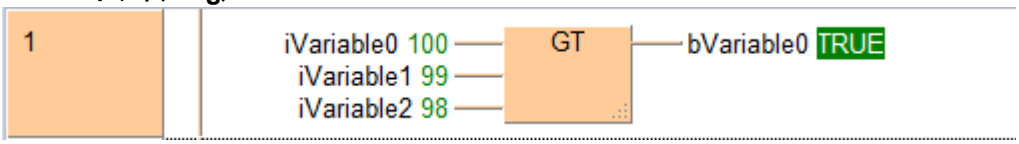
LD、FBD 言語における比較演算子の入力点数は 2～100 個まで指定することができます。

ここでは、入力点数を 3 個指定した場合を例に紹介します。

■POU ヘッダ（Prog）

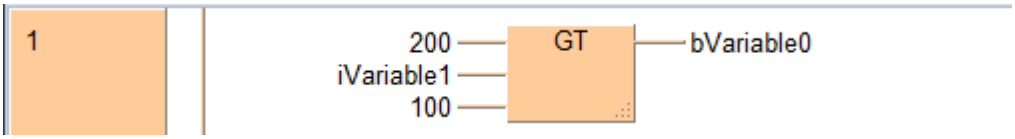
	クラス	変数名	データ型	初期値
1	VAR	iVariable0	INT	0
2	VAR	iVariable1	INT	0
3	VAR	iVariable2	INT	0
4	VAR	bVariable0	BOOL	FALSE

■POU ボディ（Prog）



比較演算子GTは”＞”の比較を行いますが、上図に示すように入力点数が3個あれば、
iVariable0＞iVariable1＞iVariable2が真であればONという動作になります。

帯域比較命令としても使用できます。



上図例は、200＞iVariable1＞100 であれば、bVariable0 が ON します。

また他の比較演算子も同様に、
GEの場合　:iVariable0≧iVariable1≧iVariable2　・・・・・
LTの場合　:iVariable0＜iVariable1＜iVariable2　・・・・・
LEの場合　:iVariable0≦iVariable1≦iVariable2　・・・・・
となります。

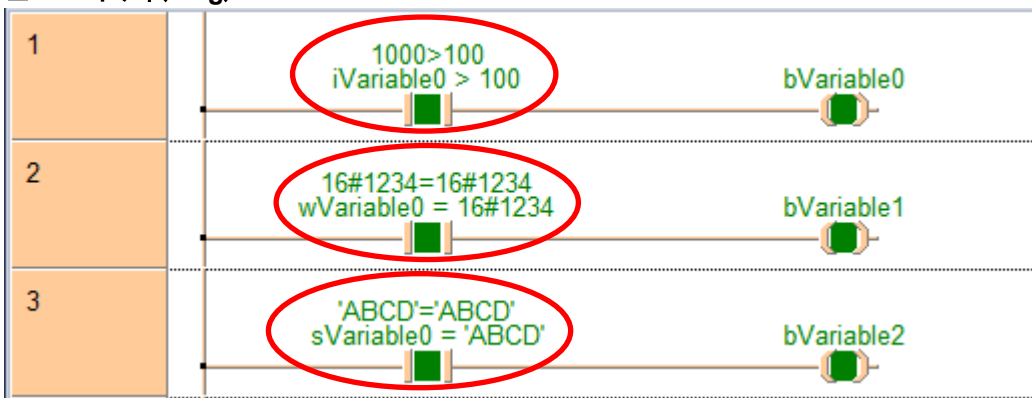
LD 言語の比較接点について

LD 言語では接点シンボル名に直接比較条件を入力する事で比較接点として使用することができます。
ここでは、比較接点として使用できるいくつかの例を紹介します。

■POU ヘッダ（Prog）

	クラス	変数名	データ型	初期値
1	VAR	bVariable0	BOOL	FALSE
2	VAR	bVariable1	BOOL	FALSE
3	VAR	bVariable2	BOOL	FALSE
4	VAR	iVariable0	INT	0
5	VAR	wVariable0	WORD	0
6	VAR	sVariable0	STRING[32]	''

■POU ボディ（Prog）



ST 言語

複数行の Tab 挿入(ST)

ストラクチャードテキスト(ST)のプログラムに複数行にまたがって、行始めに Tab を挿入することができます。
ここでは IF 文内の式を複数行に Tab を挿入する例で説明します。

■POU ボディ(Prog)

```
IF (DF(bExecute)) THEN
iSum := iData1 + iData2;
iAverage := iSum / 2;
END_IF;
```

Tab を挿入する行を含むように文字を選択し、キーボード「Tab」を入力します。

```
IF (DF(bExecute)) THEN
iSum := iData1 + iData2;
iAverage := iSum / 2;
END_IF;
```

```
IF (DF(bExecute)) THEN
    iSum := iData1 + iData2;
    iAverage := iSum / 2;
END_IF;
```

複数行の行始めに Tab が挿入されました。

●備考
Shift＋Tab を入力することで、複数行の行始めの Tab を削除できます。

Tab を削除する行を含むように文字を選択し、キーボード「Shift」を押しながら「Tab」を入力します。

```
IF (DF(bExecute)) THEN
    iSum := iData1 + iData2;
    iAverage := iSum / 2;
END_IF;
```

```
IF (DF(bExecute)) THEN
iSum := iData1 + iData2;
iAverage := iSum / 2;
END_IF;
```

記述済みの変数を入れ替える(ST)

POU ボディに記述済みの変数をダブルクリックで選択でき、簡単に別の変数に入れ替えることができます。
ここでは、「bStop」を「bEnable」に入れ替える例で説明します。

■POU ヘッダ(Prog)

	クラス	変数名	データ型	初期値
1	VAR	bEnable	BOOL	FALSE
2	VAR	bStop	BOOL	FALSE
3	VAR	iData1	INT	0
4	VAR	iData2	INT	0
5	VAR	iSum	INT	0

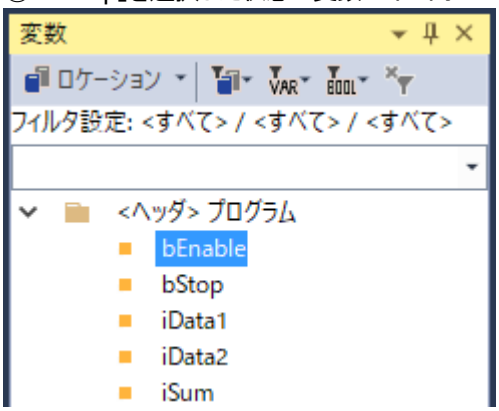
■POU ボディ(Prog)

```
IF (bStop) THEN
    iSum := iData1 + iData2;
END_IF;
```

① 「bStop」をダブルクリックすると、変数が選択されます。

```
IF (bStop) THEN
    iSum := iData1 + iData2;
END_IF;
```

② 「bStop」を選択した状態で変数ペイン内の「bEnable」をダブルクリックすると、「bStop」の位置に「bEnable」が挿入されます。




```
IF (bEnable) THEN
    iSum := iData1 + iData2;
END_IF;
```

構文ペア検索機能(ST)

FPWIN Pro7 では構文のペアまでカーソルをジャンプする機能があります。
ここでは、IF のペアである END_IF にジャンプする例で説明します。

■POU ヘッダ(Prog)

	クラス	変数名	データ型	初期値
1	VAR	bEnable	BOOL	FALSE
2	VAR	iAverage	INT	0
3	VAR	iData1	INT	0
4	VAR	iData2	INT	0
5	VAR	iSum	INT	0

■POU ボディ(Prog)

```
IF (bEnable) THEN
   iSum := iData1 + iData2;
   iAverage := iSum;
END_IF;
```

① 文字入力のカーソルを「IF」の行に合わせます。

```
IF (bEnable) THEN
   iSum := iData1 + iData2;
   iAverage := iSum;
END_IF;
```

② Ctrl キーを押しながら E キーを押します

```
IF (bEnable) THEN
   iSum := iData1 + iData2;
   iAverage := iSum;
END_IF;
```

文字入力のカーソルがペアの「END_IF」の位置に移動します。

IF 文がネスト(入れ子)で記述されている場合などに、
この機能を使用することで、プログラムが正しく記述されているかを簡単に確認できます。

ボディから変数新規宣言(ST)

ストラクチャードテキスト(ST)はボディから新規宣言ダイアログを使用して、変数宣言することができます。
ここでは、「bExecute」を新規宣言する例で説明します。

■POU ヘッダ(Prog)

	クラス	変数名	データ型	初期値	コメント
1	VAR	iData1	INT	0	
2	VAR	iData2	INT	0	
3	VAR	iSum	INT	0	

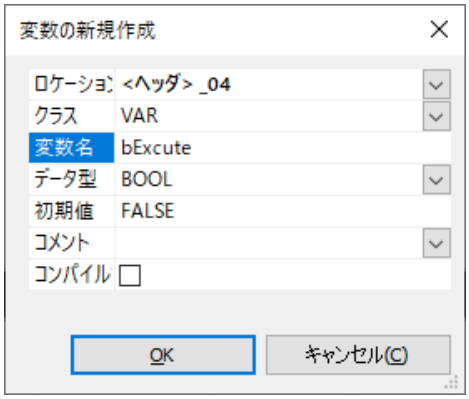
■POU ボディ(Prog)

```
IF (DF(bExcute)) THEN
   iSum := iData1 + iData2;
END_IF;
```

① 「bExecute」の先頭文字の左 から 最終文字の右 までのいずれかに文字入力のカーソルを合わせます。

```
IF (DF(bExcute)) THEN
   iSum := iData1 + iData2;
END_IF;
```

② キーボード「Alt」を押しながら「N」を入力します。新規の変数宣言ダイアログが表示されます。



③ 「OK」を押すと、POU ヘッダ(Prog)に「bExecute」が宣言されています。

	クラス	変数名	データ型	初期値	コメント
1	VAR	iData1	INT	0	
2	VAR	iData2	INT	0	
3	VAR	iSum	INT	0	
4	VAR	bExcute	BOOL	FALSE	

ST 言語では F 命令のオペランドに定数の演算が可能

ST 言語では、F 命令で定数を指定できるオペランド部に定数の演算値を直接入力することができます。
ここでは、F151(WRT)と F22(＋)の応用命令を例に説明します。

■グローバル変数

	クラス	変数名	FPアドレス	IECアドレス	データ型	初期値
1	VAR_GLOBAL	g_iStartAddress_DT100	DT100	%MW5.100	INT	0
2	VAR_GLOBAL	g_iStorage_DT200	DT200	%MW5.200	INT	0

■POU ボディ(Prog)

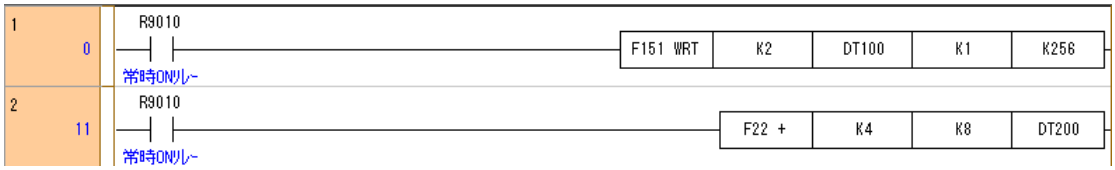
```
F151_WRT(s1_BankSlot  := 1+1,
         s2_Start     := g_iStartAddress_DT100,
         n_Number     := 1,
         d_Start      := 16#0100);

F22_ADD2(1+3, 2*4, g_iStorage_DT200);
```

1＋1＝2(スロット No.が"2"ということになります)

2×4＝8

1＋3＝4

■コンパイル後のプログラム(FPWIN GR7 ラダー表記)



演算結果が直接オペランド部に定数として指定されていることが分かります！

ST 言語における比較演算子の使い方

FPWIN Pro7 では、ST 言語において”=”, ”>”, ”<”の意味を持つ比較演算子があります。
ここでは、”=”(イコール)の場合についての例で説明します。

■POU ヘッダ(Prog)

	クラス	変数名	データ型	初期値
1	VAR	bVariable0	BOOL	FALSE
2	VAR	iVariable0	INT	0
3	VAR	iVariable1	INT	0

■POU ボディ(Prog)

●IF 文による記述例

```
IF (iVariable0 = iVariable1) THEN    100      100
   bVariable0 := TRUE;
END_IF;
```

比較結果が TRUE の時

結果が TRUE(ON)となります。

```
IF (iVariable0 = iVariable1) THEN    100      200
   bVariable0 := TRUE;
END_IF;
```

比較結果が FALSE になっても

結果は TRUE(ON)を保持します。

●IF 文を使用しない記述例

```
bVariable0 := iVariable0 = iVariable1;    100      100
```

結果が TRUE(ON)となります。

比較結果が TRUE の時

```
bVariable0 := iVariable0 = iVariable1;    100      200
```

結果は FALSE(OFF)となります。

比較結果が FALSE の時

■備考

```
bVariable0 := (iVariable0 = iVariable1) AND (iVariable2 = iVariable3);
```

比較条件が複数ある場合は、上記のように記述することができます。

ST 言語において出力の記述が必須でないオペランド

ST 言語において FB 命令に「＊＊ =>？＊＊?」の記述がある場合は必須ではありません。削除することができます。
ここでは、TON 命令を用いた例で紹介します。

■POU ヘッダ（Prog）

	クラス	変数名	データ型	初期値
1	VAR	TM_0	TON	
2	VAR	bVariable0	BOOL	FALSE

■POU ボディ（Prog）

```
TM_0(IN := bVariable0,
     PT := T#10s,
     Q => ?BOOL?,
     ET => ?TIME?);
```

削除できます。

```
TM_0(IN := bVariable0,
     PT := T#10s);
```

TON（FB）の他に、下記のデータ型でも同様に
「＊＊ =>？＊＊?」を削除することができます。
・FP Library
CT_FB、IsReceptionDoneByTimeOut
・IEC Standard Library
CTD / CTU / CTUD / F_TRIG / R_TRIG / RS / SR / TON / TOF / TP
・自作の FB / FUN
VAR_OUTPUT が存在するもの

```
<ROM_Download プログラムコード (33 ステップ)>
0 エラー
0 警告
```

チェック／コンパイル後、エラーが発生していないことが分かります。
ただし、プログラム内に「?＊＊?」を残している状態ではコンパイルエラーとなります。

消したオペランドは「変数名.オペランド記号」として使用することができます。
上記で宣言した、「TM_0」の出力信号:Q を使用する場合は、「TM_0.Q」と記述します。

```
IF DF(TM_0.Q) THEN
 bLamp := TRUE;
END_IF;
```

ST 言語において記述が必須でないオペランド

ST 言語の命令記述の際、”??? :=”の記述は必須ではありません。削除することができます。
ここでは、LIMIT 命令を用いた例で紹介します。

■POU ヘッダ(Prog)

	クラス	変数名	データ型	初期値
1	VAR	iVariable0	INT	0
2	VAR	iVariable1	INT	0

■POU ボディ(Prog)

```
iVariable1 := LIMIT(MN := 0, IN := iVariable0, MX := 100);
```

削除できます。

```
iVariable1 := LIMIT(0, iVariable0, 100);
```

```
<ROM_Download プログラムコード (33 ステップ)>
0 エラー
0 警告
```

チェック／コンパイル後、エラーが発生していないことが分かります。

■注意

上記例の LIMIT 命令と同様に他の命令でも”:=”が含まれている場合は削除することができますが、
削除する場合は、すべてのオペランドに対して”:=”を削除してください。
すべてのオペランドに対して削除しない場合は、コンパイルエラーになりますので注意が必要です。

```
iVariable1 := LIMIT(MN := 0, iVariable0, MX := 100);
```

1 つだけ削除

■備考

F 命令や FP 命令等でも同様に削除が可能です。

```
F6_DGT(s := wVariable0, n := 16#0012, d => wVariable1);
```

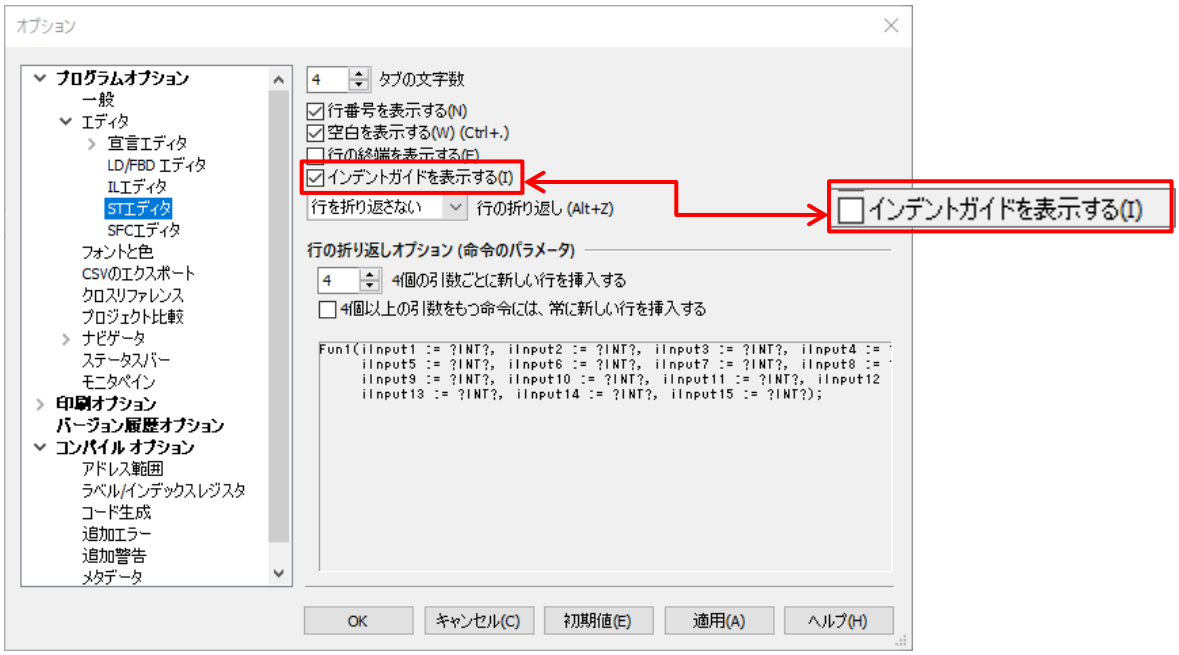
```
F6_DGT(wVariable0, 16#0012, wVariable1);
```

ST エディタ表示設定:インデントガイドの表示

Control FPWIN Pro7 の ST エディタでは、記述した ST のインデントガイドを表示させるかの有無を切り替えることができます。

ツールバーの「拡張機能(X)」ー「オプション(O)」をクリックして「オプション」のポップアップウインドウを表示させます。

オプションのポップアップウインドウから、「プログラムオプション」ー「エディタ」ー「ST エディタ」を選択します。
「インデントガイドを表示する」のチェックボックスのチェックの有無で、ST エディタ内でのインデントガイド表示を切り替えることができます。デフォルトでは、「インデントガイドを表示する」は有効になっています。




```
1
2   IF (DF(bSet01)) THEN
3       bStart := TRUE;
4       IF (0 < iData) AND (iData < 100) THEN
5           iResult := 1;
6       ELSIF (101 < iData) AND (iData < 200) THEN
7           iResult := 2;
8       ELSE
9           iResult := 3;
10      END_IF;
11  END_IF;
```

インデントガイドは　　で表示されます。

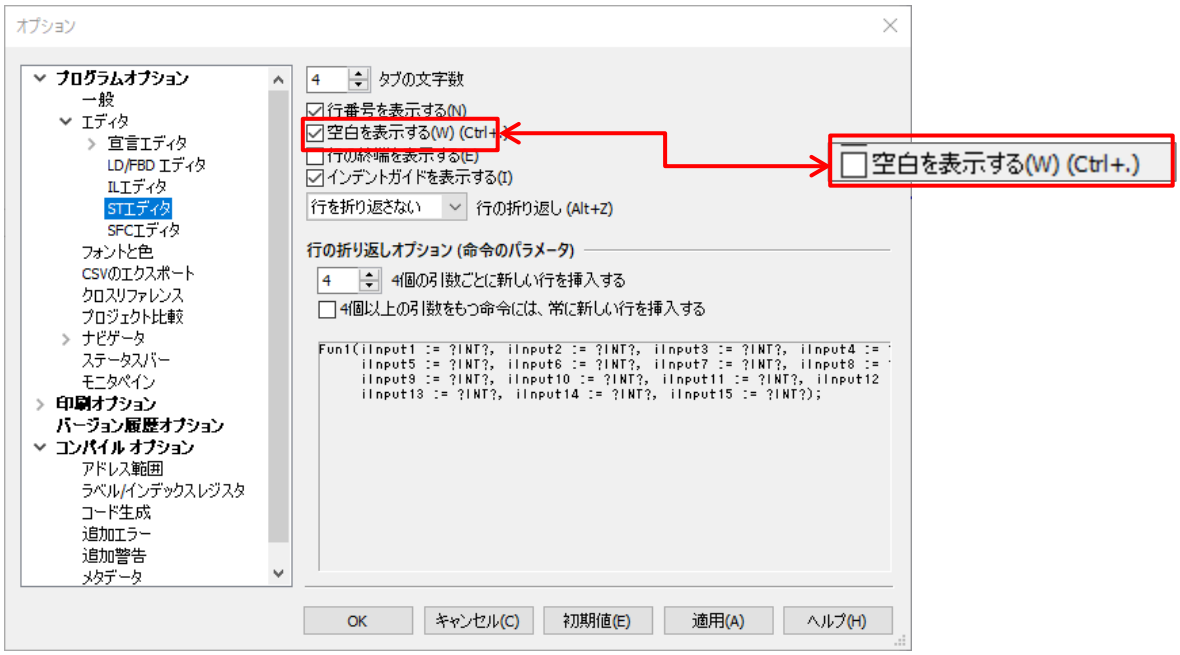
```
1
2   IF (DF(bSet01)) THEN
3       bStart := TRUE;
4       IF (0 < iData) AND (iData < 100) THEN
5           iResult := 1;
6       ELSIF (101 < iData) AND (iData < 200) THEN
7           iResult := 2;
8       ELSE
9           iResult := 3;
10      END_IF;
11  END_IF;
```

ST エディタ表示設定：空白の表示

Control FPWIN Pro7 の ST エディタでは、記述した ST の空白部分に記号表示させるかの有無を切り替えることができます。

ツールバーの「拡張機能(X)」ー「オプション(O)」をクリックして「オプション」のポップアップウインドウを表示させます。

オプションのポップアップウインドウから、「プログラムオプション」ー「エディタ」ー「ST エディタ」を選択します。
「空白を表示する」のチェックボックスのチェックの有無で、ST エディタ内での空白に記号表記させるかを切り替えることができます。
デフォルトでは、「空白を表示する」は有効になっており、記号表示されている状態になります。




```
1
2   IF·(DF(bSet01))·THEN
3   ⟶bStart·:=·TRUE;
4   ⟶IF·(0·<·iData)·AND·(iData·<·100)·THEN
5   ⟶⟶iResult·:=·1;
6   ⟶ELSIF·(101·<·iData)·AND·(iData·<·200)·THEN
7   ⟶⟶iResult·:=·2;
8   ⟶ELSE·
9   ⟶⟶iResult·:=·3;
10  ⟶END_IF;
11  END_IF;
```

空白のうち、スペースは ·
タブは ⟶
で表示が区別されています。

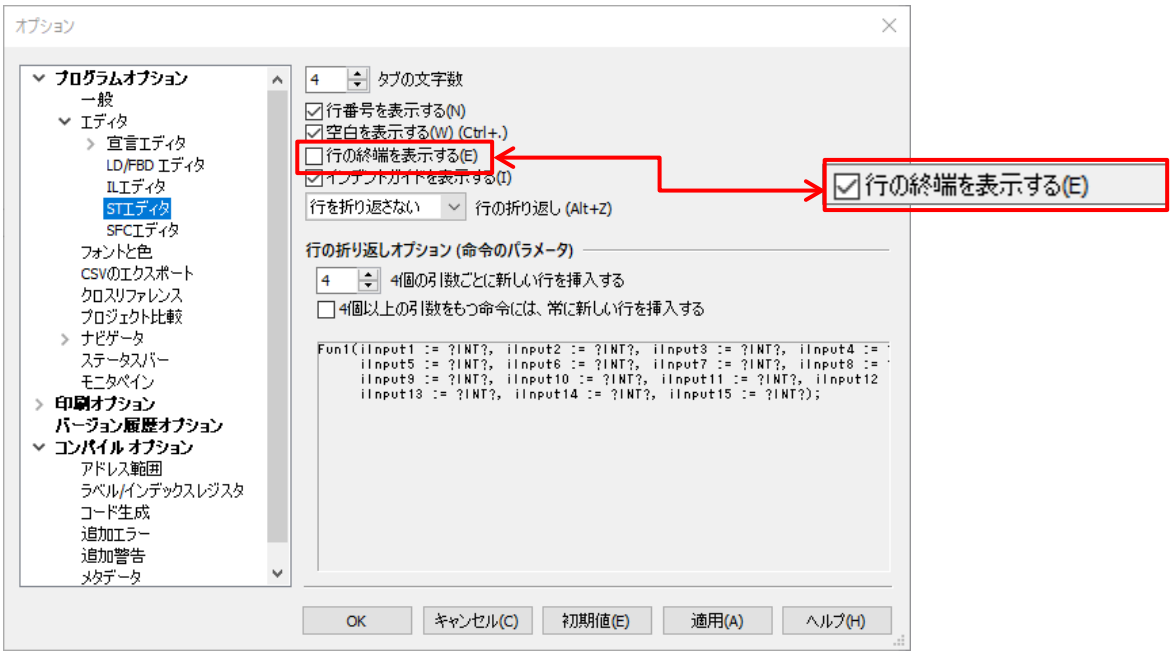
```
1
2   IF (DF(bSet01)) THEN
3       bStart := TRUE;
4       IF (0 < iData) AND (iData < 100) THEN
5           iResult := 1;
6       ELSIF (101 < iData) AND (iData < 200) THEN
7           iResult := 2;
8       ELSE
9           iResult := 3;
10      END_IF;
11  END_IF;
```

ST エディタ表示設定:行の終端の表示

Control FPWIN Pro7 の ST エディタでは、記述した ST の行終端を表示させるかの有無を切り替えることができます。

ツールバーの「拡張機能(X)」ー「オプション(O)」をクリックして「オプション」のポップアップウインドウを表示させます。

オプションのポップアップウインドウから、「プログラムオプション」ー「エディタ」ー「ST エディタ」を選択します。
「行の終端を表示する」のチェックボックスのチェックの有無で、ST エディタ内での行終端表示を切り替えることができます。
デフォルトでは、「行の終端を表示する」は無効になっています。




```
1
2   IF (DF(bSet01)) THEN
3       bStart := TRUE;
4       IF (0 < iData) AND (iData < 100) THEN
5           iResult := 1;
6       ELSIF (101 < iData) AND (iData < 200) THEN
7           iResult := 2;
8       ELSE
9           iResult := 3;
10      END_IF;
11  END_IF;
```

行の終端は CR LF で表示されます。

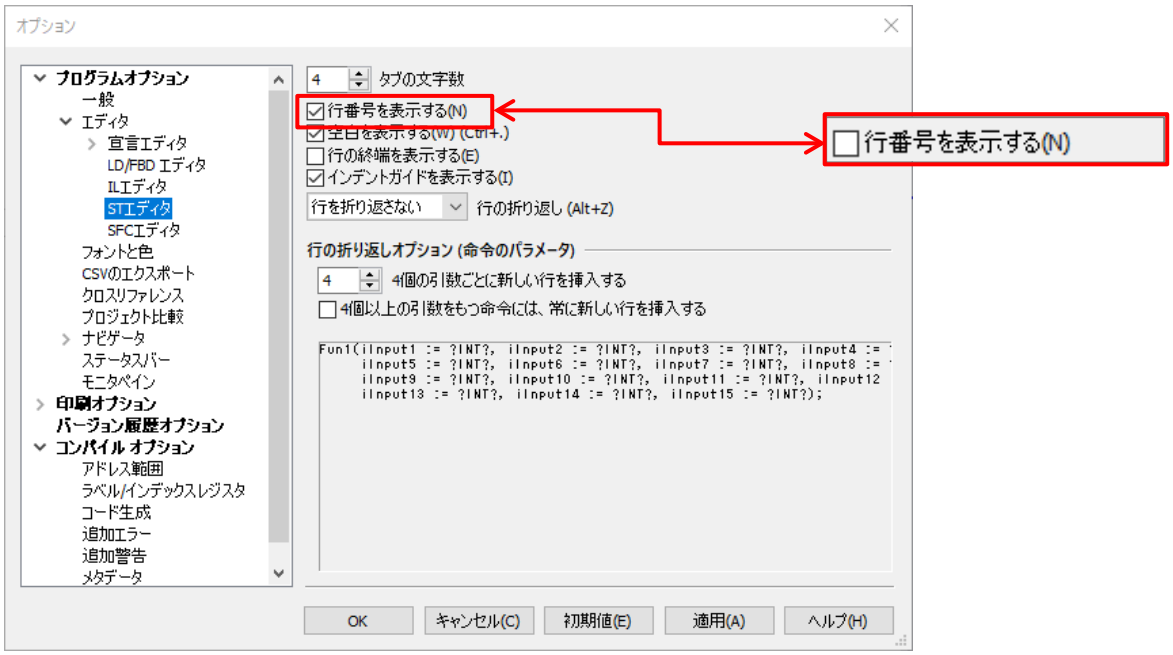
```
1
2   IF (DF(bSet01)) THEN CR LF
3       bStart := TRUE; CR LF
4       IF (0 < iData) AND (iData < 100) THEN CR LF
5           iResult := 1; CR LF
6       ELSIF (101 < iData) AND (iData < 200) THEN CR LF
7           iResult := 2; CR LF
8       ELSE CR LF
9           iResult := 3; CR LF
10      END_IF; CR LF
11  END_IF; CR LF
```

ST エディタ表示設定:行番号の表示

Control FPWIN Pro7 の ST エディタでは、記述した ST の行番号表示の有無を切り替えることができます。

ツールバーの「拡張機能(X)」ー「オプション(O)」をクリックして「オプション」のポップアップウインドウを表示させます。

オプションのポップアップウインドウから、「プログラムオプション」ー「エディタ」ー「ST エディタ」を選択します。
「行番号を表示する」のチェックボックスのチェックの有無で、ST エディタ内での行番号表示を切り替えることができます。
デフォルトでは、「行番号を表示する」は有効になっています。



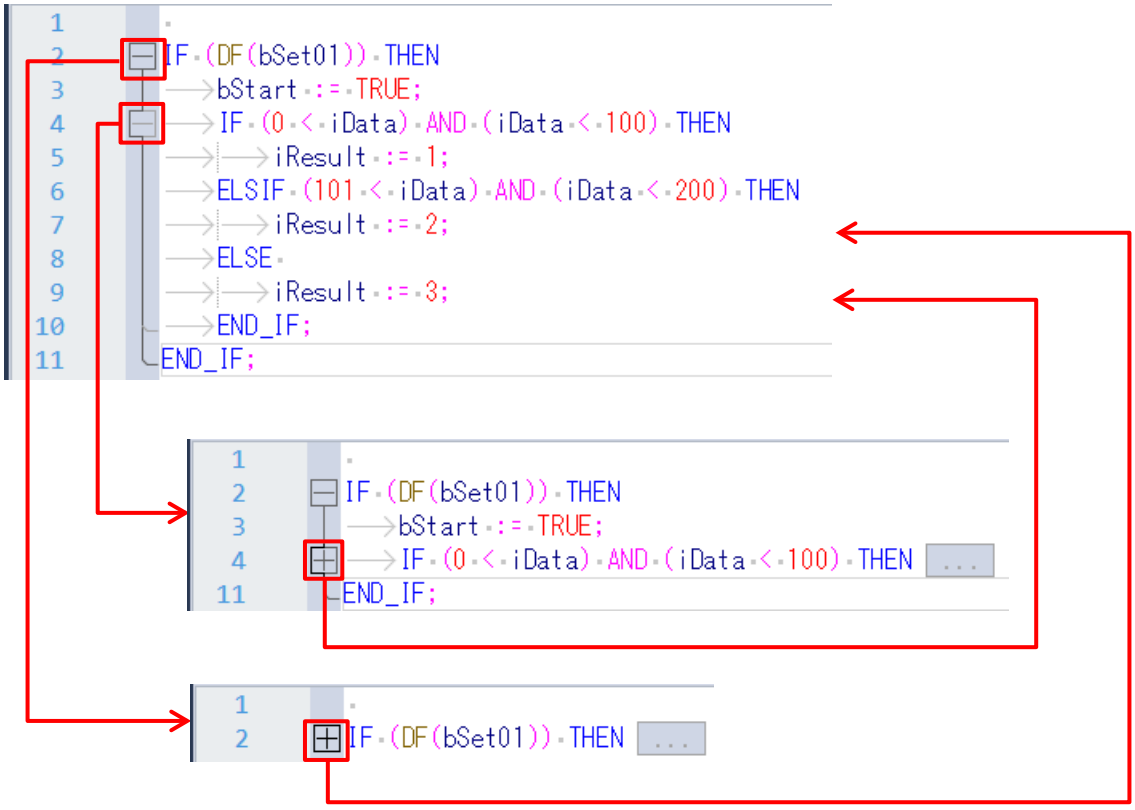

```
1
2   IF (DF(bSet01)) THEN
3       bStart := TRUE;
4       IF (0 < iData) AND (iData < 100) THEN
5           iResult := 1;
6       ELSIF (101 < iData) AND (iData < 200) THEN
7           iResult := 2;
8       ELSE
9           iResult := 3;
10      END_IF;
11  END_IF;
```

```
    IF (DF(bSet01)) THEN
        bStart := TRUE;
        IF (0 < iData) AND (iData < 100) THEN
            iResult := 1;
        ELSIF (101 < iData) AND (iData < 200) THEN
            iResult := 2;
        ELSE
            iResult := 3;
        END_IF;
    END_IF;
```

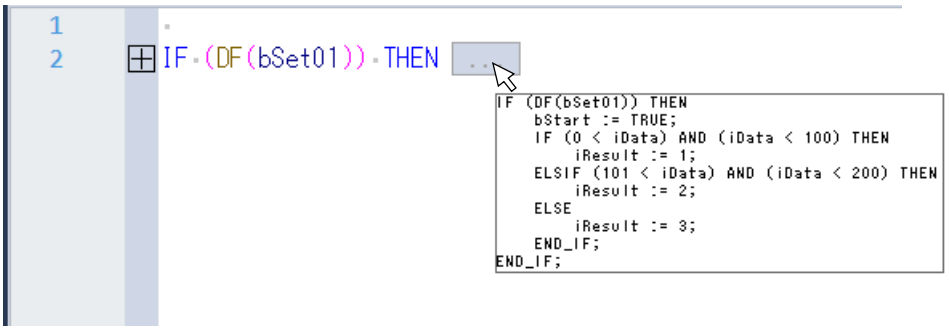
ST コードの折り畳みと展開

ST コードは、インデント単位で折り畳み⇔展開が可能です。

ST コードはデフォルト状態では展開されているので、ST エディタ内の ⊟ をクリックすることで折り畳むことができます。
また、折り畳んだ ST コードは、同じく ST エディタ内の ⊞ をクリックすることで展開することができます。



※折り畳まれた内容は、折り畳まれた状態に表示される ... にマウスカーソルを合わせることで確認することができます。



ST プログラム中の1行コメントと複数行コメント

ST プログラム中にプログラムのコメントを記述することで、プログラムの処理や流れを理解しやすくすることができます。
コメントには、3 種類のタイプのコメントがあります。

1. (* で始まり、*)で終了するコメント: 複数行に渡ってコメントとすることができます。
2. /* で始まり、*/で終了するコメント: 複数行に渡ってコメントとすることができます。
3. // で始まる 1 行単位のコメント: 同一行内の // 以降はすべてコメントとみなされます。

1 のタイプは、複数行に渡る文章をコメント化するのに便利です。
3 のタイプは、始端に // を入力するだけで末尾のコメント終了記号は入力不要なので、簡単に 1 行単位のコメントを入力できます。

2 と 3 のコメントは、Control FPWIN Pro7 バージョン 7.6.0.0 以降でサポートされています。
それ以前のバージョンでは、エラーとなります。

記述を整列して見やすくする記述方法

変数名、システム命令、ファンクション、ファンクションブロックの名前、それら命令の引数の名前は、いずれも意味を分かりやすくするために　長い名前になりがちです。
引数の数が多いと命令の長さが長くなりすぎてページをはみ出してしまう場合があり、その場合、画面縮小して表示するか、
スクロールバーでスクロールする必要があります。

また、モニタ実施時には、横並びの変数がつながって表示されるので、どの変数の内容なのか識別しにくいことがあります。
これらの状態を回避して、プログラムや引数のモニタを見やすくするには、１行に表示する引数を1つにして、
表示場所をそろえることが効果的です。

■具体例
FP_ETHERNET_CONNECTION_SET イーサネットコネクション設定命令について説明します。
命令入力直後

```
//コネクション1を設定
FP_ETHERNET_CONNECTION_SET(sMode := ?STRING?, sAddress := ?STRING?, nStartPort := ?ANY16?, n
                           bError => ?BOOL?);
```

画面からはみ出して自動的に折り返して表示されています。見えない引数部分があります。
複数の引数が同じ行にあるとモニタ時に識別が困難になります。
下図は、各引数の後ろのコンマ「，」の後で改行し、各引数の表示位置を「タブ」でそろえたものです。

```
FP_ETHERNET_CONNECTION_SET(sMode := ?STRING?,
                           sAddress := ?STRING?,
                           nStartPort := ?ANY16?,
                           nEndPort := ?ANY16?,
                           bError => ?BOOL?);
```

この記述方法によれば、ページをスクロールしたり縮小することなく命令全体を見ることができ、
モニタ時にも、１つの引数の横に１つのモニタ値が表示されるので、見やすく間違いの少ないプログラムにすることができます。

条件分岐の動作詳細説明(case 文)

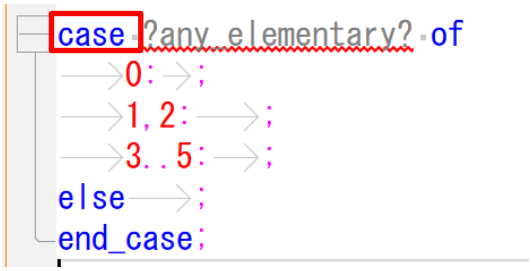
ここでは、ST 言語でプログラム作成する際に必須の case 文による条件分岐の動作説明をします。
本説明ではプログラミングツールによってコンパイルされた結果(PLC のオリジナル命令)から、動作を確認します。

【case 文の記述例】

下記は、case 文の記述例です。
変数 A(下図赤字)の内容によって、処理を分けることができます。

```
case 変数 A of
        0:
                処理 1;     // 変数 A が 0 の時に実行されます。
        1,2:
                処理 2;     // 変数 A が 1 または 2 の時に実行されます。
        3..5:
                処理 3;     // 変数 A が 3～5 の時に実行されます。
else
        処理 4;             // 変数 A が上記以外の時に実行されます。
end_case;
```

Control FPWIN Pro7 の ST 編集画面で、case に続いてスペースキーを入力すると(下図赤枠)
case 文の典型的なパターンが自動入力されますので、必要に応じて変形して使用します。



【補足】

Case 文での各分岐内の処理は、条件が成立する場合のみ実行されます。

条件分岐の動作詳細説明(if 文)

ここでは、ST 言語でプログラム作成する際に必須の if 文による条件分岐の動作説明をします。
本説明ではプログラミングツールによってコンパイルされた結果(PLC のオリジナル命令)から、動作を確認します。

【if 文の記述例】

下記は、良くある if 文の記述例です。

(1)

```
if 条件式 A then
        処理        //条件式 A が true の時に実行されます
end if;
```

(2)

```
if 条件式 A then
        処理 1 ;    //条件式 A が true の時に実行されます
else
        処理 2 ;    //条件式 A が false の時に実行されます
endif;
```

(3)

```
if 条件式 A then
        処理 1 ;    //条件式 A が true の時に実行されます
        elsif 条件式 B then
                処理 2 ;    //条件式 A が true、かつ条件式 B が true の時に実行されます。
        else
                処理 3 ;    //条件式 A が true、かつ条件式 B が false の時に実行されます。
        end if;
else
```

SFC 言語

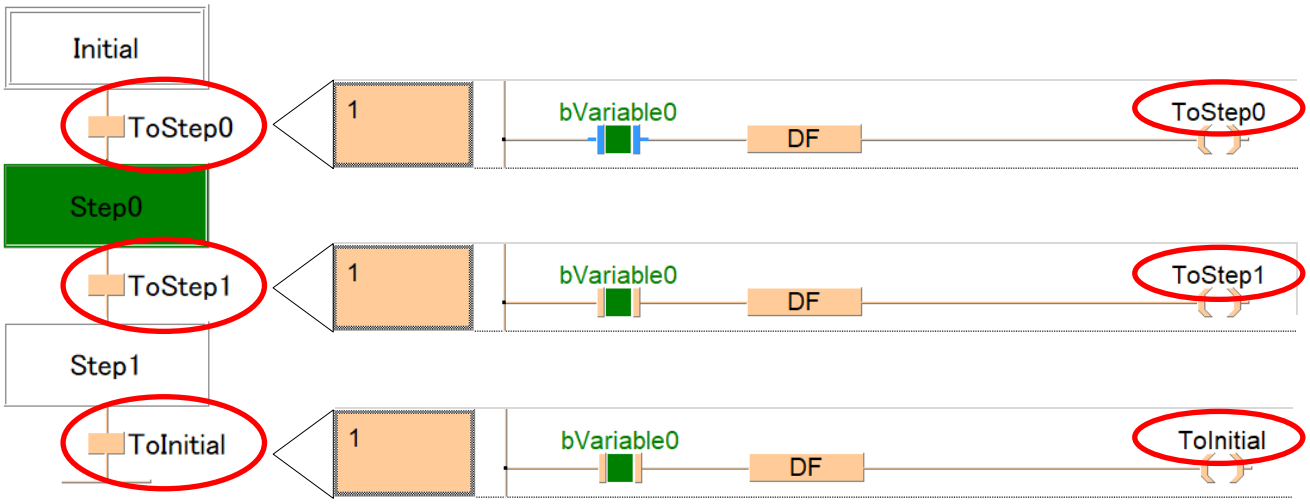
同一信号の連続入力による SFC 工程移行

SFC で工程を移行する際に同じ信号の ON／OFF 繰り返しでも移行することができます。
ここでは、”bVariable0”が繰り返し入力され、ON した時の立ち上がりで移行する例で紹介します。

■POU ヘッダ（Prog）

	クラス	変数名	データ型	初期値	コメント
1	VAR	bInput_0	BOOL	FALSE	

■POU ボディ（Prog）



トラディション内にトラディション名の出力を 1 スキャン ON させることで、
1 つの信号の立ち上がりで工程を移行することができます。

シリアル通信の受信完了信号などに使用すると便利です。

FUN/FB

“EN／ENO 付き、なし”での動作の違い

ファンクションブロック（FB）、ファンクション（FUN）では“EN／ENO 付き、なし”を選択して作成できます。
ここでは、簡単なファンクションブロックを例に“EN”“ENO”が「あり」と「なし」で動作の違いを紹介します。

■POU ヘッダ（FB:EN_ENO_Test）

	クラス	変数名	データ型	初期値
1	VAR_OUTPUT	bOut	BOOL	FALSE

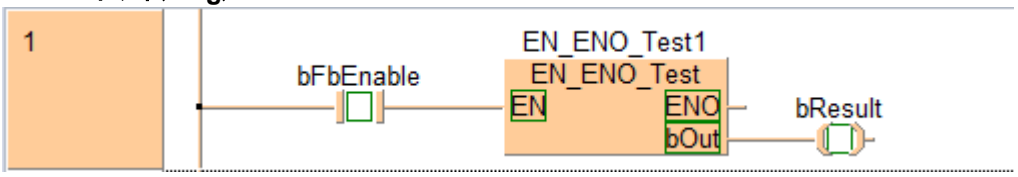
■POU ボディ（FB:EN_ENO_Test）

■POU ヘッダ（Prog）

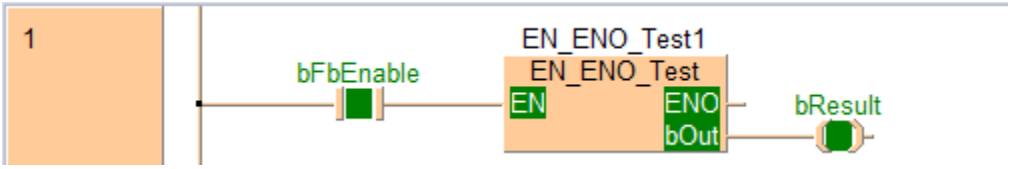
	クラス	変数名	データ型	初期値
1	VAR	bFbEnable	BOOL	FALSE
2	VAR	bResult	BOOL	FALSE
3	VAR	EN_ENO_Test1	EN_ENO_Test	

●”EN／ENO 付き”の場合

■POU ボディ（Prog）



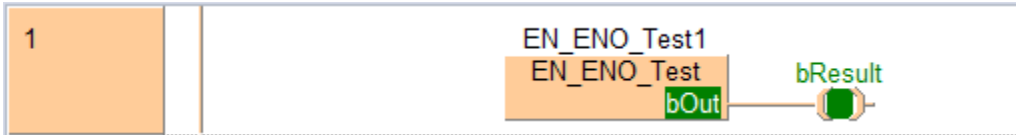
“EN”が OFF（FALSE）の間、出力（bOut）は動作（ON）しません。



“EN”が ON（TRUE）で、出力（bOut）は動作（ON）します。

●”EN／ENO なし”の場合

■POU ボディ（Prog）



ファンクションブロックをボディに張り付けた段階で、常時 FB を実行することになります。

“EN／ENO あり”の場合のファンクションブロック（FB）、ファンクション（FUN）出力状態

“EN／ENO あり”で作成したファンクションブロック、ファンクションの出力は、“EN”が OFF した時の状態で保持されます。
ここでは、簡単なファンクションブロックで出力が保持される例を紹介します。

■POU ヘッダ（FB:EN_ENO_Test）

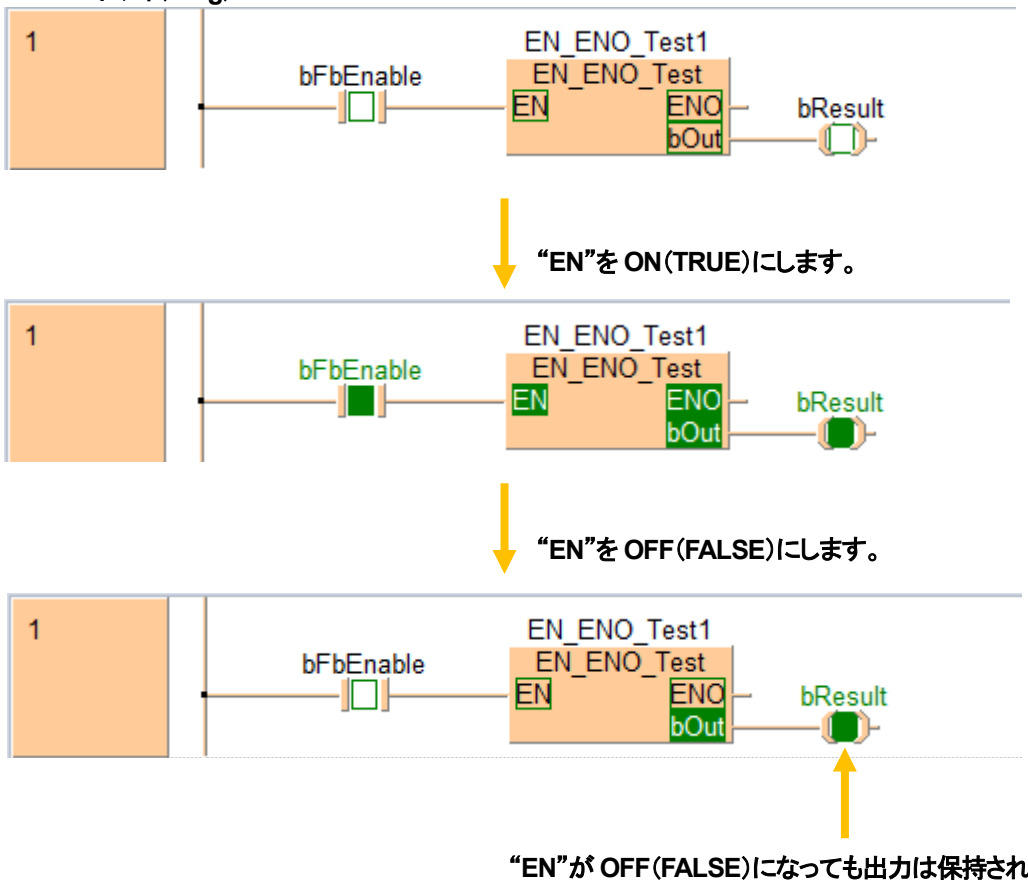
	クラス	変数名	データ型	初期値
1	VAR_OUTPUT	bOut	BOOL	FALSE

■POU ボディ（FB:EN_ENO_Test）

■POU ヘッダ（Prog）

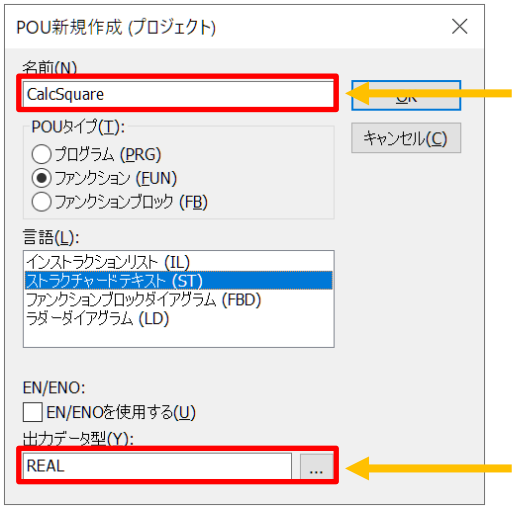
	クラス	変数名	データ型	初期値
1	VAR	bFbEnable	BOOL	FALSE
2	VAR	bResult	BOOL	FALSE
3	VAR	EN_ENO_Test1	EN_ENO_Test	

■POU ボディ（Prog）



ファンクション（FUN）の出力

ファンクションではファンクション名に型宣言を行い、出力できるようになっています。



ファンクション名
ここでは”CalcSquare”とします。

出力のデータ型宣言
ここでは“REAL 型”を宣言します。

■POU ヘッダ（FUN：CalcSquare）

	クラス	変数名	データ型	初期値
1	VAR_INPUT	bStart	BOOL	FALSE
2	VAR_INPUT	iInputData	INT	0

■POU ボディ（FUN：CalcSquare）

INT 型のデータに平方根計算を行い、REAL 型の結果を得るためのファンクションです。

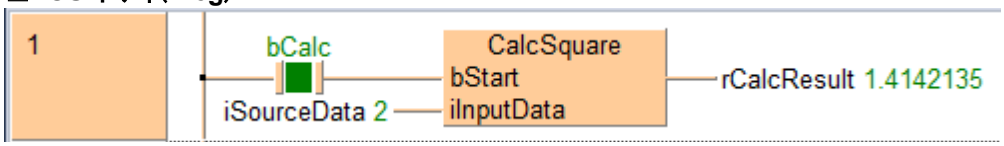
```
IF (bStart) THEN
 CalcSquare := SQRT(INT_TO_REAL(iInputData));
   END_IF;
```

上手のように、最終結果に“REAL 型”で宣言した“CalcSquare”を指定することができます。

■POU ヘッダ（Prog）

	クラス	変数名	データ型	初期値
1	VAR	bCalc	BOOL	FALSE
2	VAR	iSourceData	INT	0
3	VAR	rCalcResult	REAL	0

■POU ボディ（Prog）



クラス VAR_IN_OUT の使い方

ファンクションブロック(FB)、ファンクション(FUN)の作成時、クラスに VAR_IN_OUT を登録すると、
入力と出力の両方の変数として使用できます。
ここでは、インクリメントプログラム(FP_INC)の例で紹介します。

■POU ヘッダ(FUN:DataIncrement)

	クラス	変数名	データ型	初期値
1	VAR_INPUT	bEnable	BOOL	FALSE
2	VAR_IN_OUT	iData	INT	

クラスに"VAR_IN_OUT"を登録

■POU ボディ(FUN:DataIncrement)

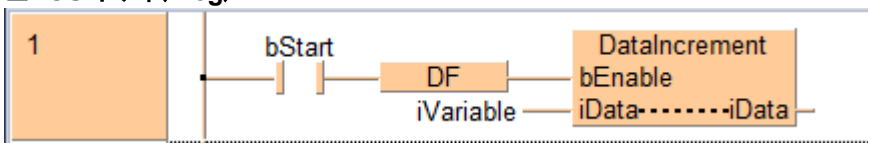
```
IF (bEnable) THEN
   FP_INC(d := iData);
END_IF;
```

iData の値を 1 加算して、iData に格納するプログラム

■POU ヘッダ(Prog)

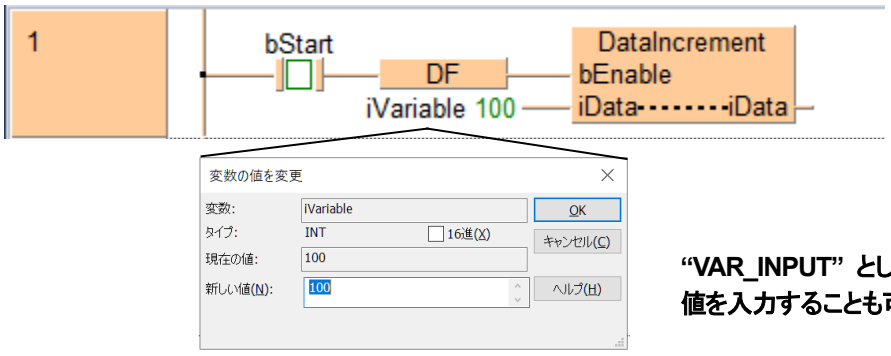
	クラス	変数名	データ型	初期値
1	VAR	bStart	BOOL	FALSE
2	VAR	iVariable	INT	0

■POU ボディ(Prog)



注)"VAR_IN_OUT"に定数を指定することは出来ません。

"bStart"が ON する度に iVariable の値に 1 を加算して iVariable に格納するファンクションブロックです。
クラスに"VAR_IN_OUT"を使用することで、ファンクションブロックの入力変数と出力変数を同じ変数に設定できます。



"VAR_INPUT" として、
値を入力することも可能です。

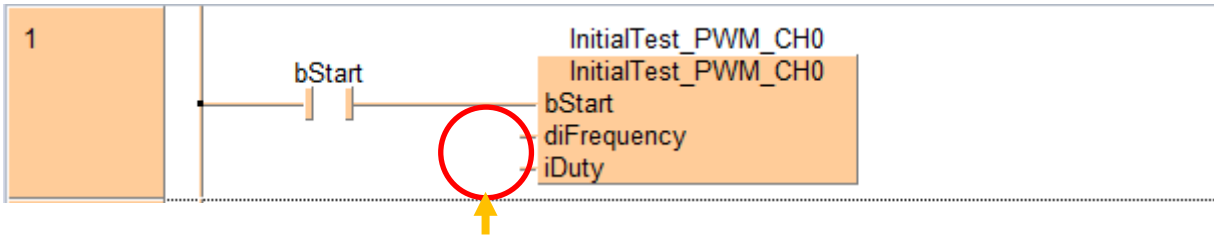
ファンクションブロック(FB)の初期値

ファンクションブロックの作成時に初期値を設定しておくことで、プログラムを簡素化することができます。
ここでは、FP0H の F173PulseOut_PWM_Hz 命令のデータテーブルを書き込む例を用いて紹介します。

■POU ヘッダ(Prog)

	クラス	変数名	データ型	初期値
1	VAR	InitialTest_PWM_CH0	InitialTest_PWM_CH0	
2	VAR	bStart	BOOL	FALSE

■POU ボディ(Prog)

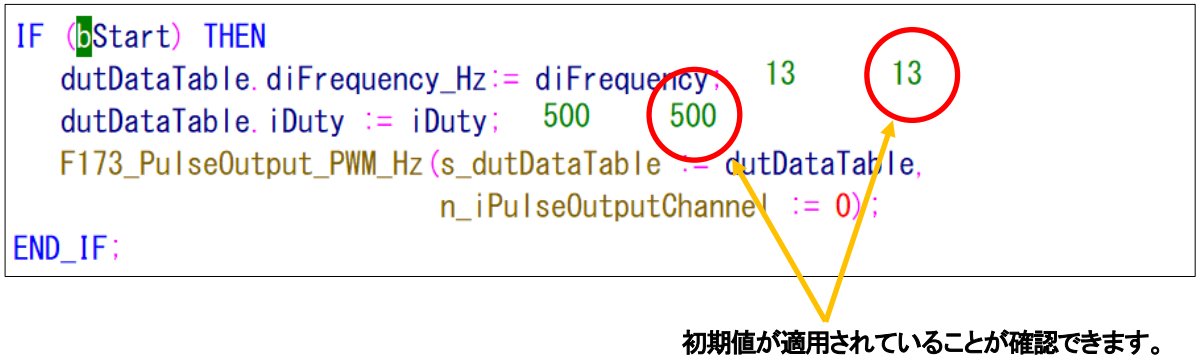


FB には入力変数がなくてもエラーにはなりません。
入力変数がない場合は、FB が持つ VAR_INPUT の初期値を適用します。

■POU ヘッダ(FB:InitialTest_PWM_CH0)

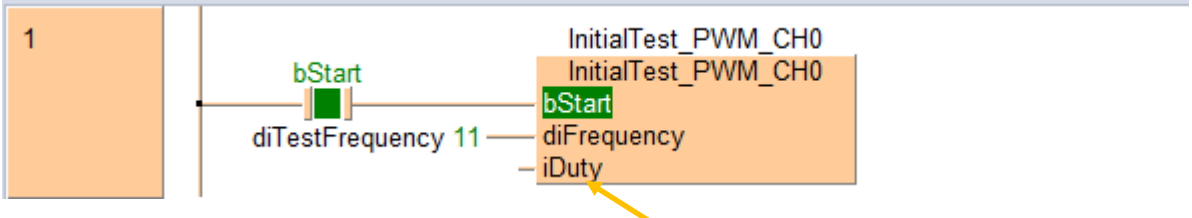
	クラス	変数名	データ型	初期値
1	VAR_INPUT	bStart	BOOL	FALSE
2	VAR_INPUT	diFrequency	DINT	13
3	VAR_INPUT	iDuty	INT	500
4	VAR	dutDataTable	F173_PulseOutput_PWM_Hz_DUT	

■POU ボディ(FB:InitialTest_PWM_CH0)



初期値が適用されていることが確認できます。

●備考



初期値“500”(50.0%)が設定されています。

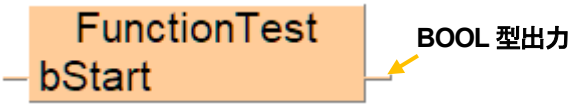
※すなわち、初期値以外で動作させたい入力変数に対してのみ値を入力することでプログラムを簡素化できます。

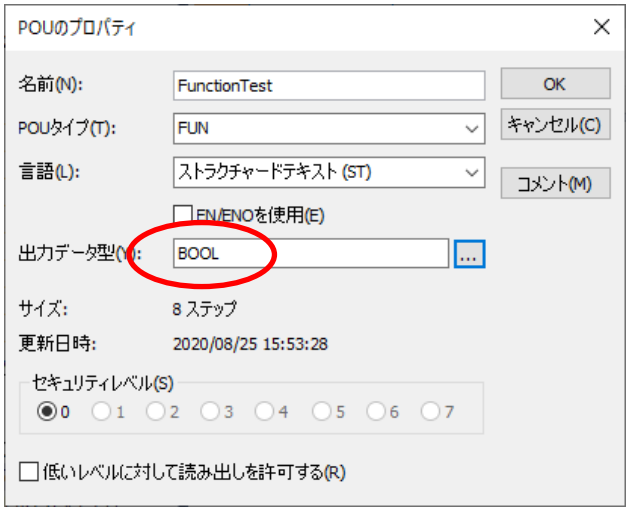
出力結果を持たないファンクション（FUN）

出力のデータ型に“VOID 型”を指定することで、出力結果を持たないファンクションを作成することが出来ます。
結果は VAR_OUTPUT を用いて出力するので、ファンクションの出力は必要ないという場合に使用します。

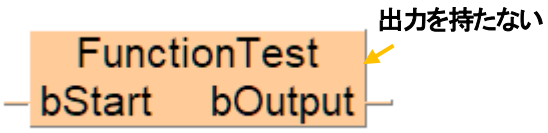
LD 言語

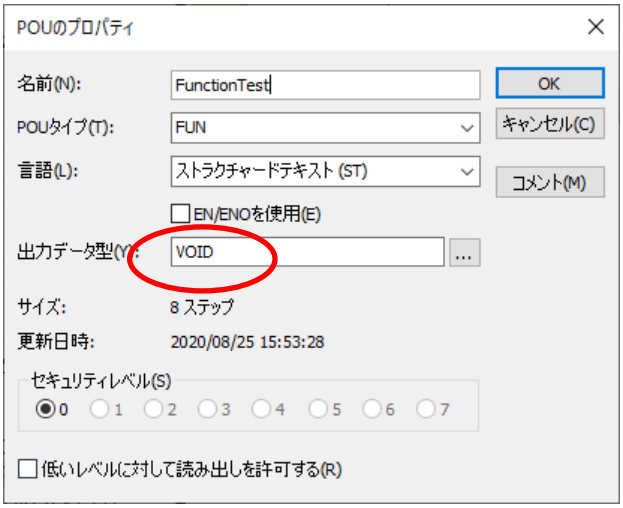
●ファンクション出力型に BOOL 型を指定時





●ファンクション出力型に VOID 型を指定時





ST 言語

■POU ヘッダ（Prog）

	クラス	変数名	データ型	初期値
1	VAR	bFunExe	BOOL	FALSE
2	VAR	bResult	BOOL	FALSE

■POU ボディ（Prog）

●ファンクション出力型に BOOL 型を指定時

```
bResult := FunctionTest(bStart := bFunExe);
```

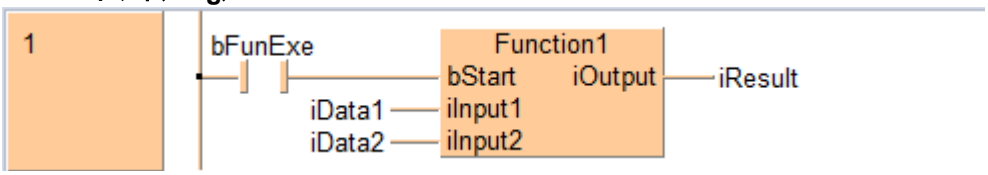
●ファンクション出力型に VOID 型を指定時

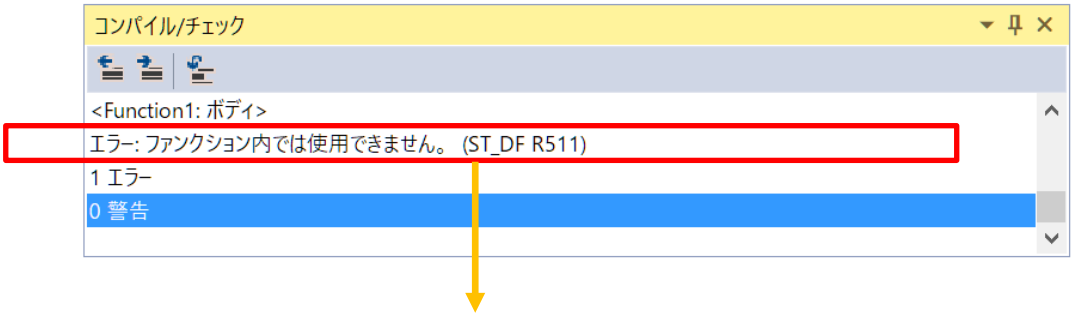
```
FunctionTest(bStart := bFunExe, bOutput => bResult);
```

ファンクション(FUN)の制限

ファンクション内で DF 命令を使用することはできません。
ここでは、故意に FUN に DF 命令を使用し、エラーを発生させる例で紹介します。

■POU ボディ(Prog)



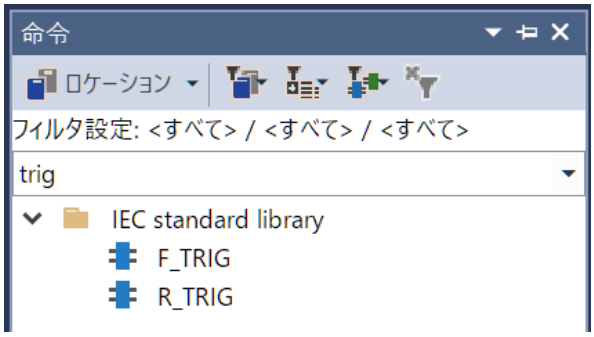


●Function1 内 POU ボディ(FUN)

```
IF (DF(bStart)) THEN
        iOutput := (iInput1 + iInput2) / 2;
   END_IF;
```

FUN 内で(DF)命令を使用しているため、コンパイルエラーが発生します。

●備考
ファンクション内では、IEC_Standard_Library の“R_TRIG(立ち上がり微分)”、”F_TRIG(立下り微分)“も、
選択することはできません。



左図はプログラム内で選択できる”R_TRIG”と“F_TRIG”
ファンクション内では両命令とも表示されません。

また、”R_TRIG”と“F_TRIG”は ST 言語では使用できません。

＜Return＞の使用例

＜Return＞を使用することによって、ファンクションブロック(FB)の動作を停止することができます。

■POU ヘッダ(FB)

	クラス	変数名	データ型	初期値
1	VAR_INPUT	bStart	BOOL	FALSE
2	VAR_INPUT	iInput	INT	0
3	VAR_OUTPUT	iOutput	INT	0

■POU ボディ(FB)

```
IF ((iInput < 0) OR (3 < iInput)) THEN
   RETURN;
END_IF;

IF (bStart = TRUE) THEN
   iOutput := iOutput + 1;
END_IF;
```

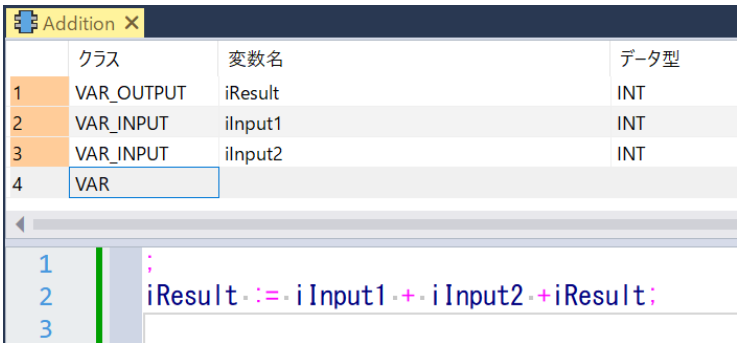
1 つ目の IF 文の iInput が 0~3 以外の時、＜Return＞が動作します。
＜Return＞が実行した時点で、この FB を停止して読み出し元の PRG(プログラム)に戻りますので、
それ以下のプログラム(上図では 2 つ目の IF 文)は動作を行いません。

ある条件で FB を＜動作させる＞＜動作させない＞を決めたい場合に、使用すると便利です。

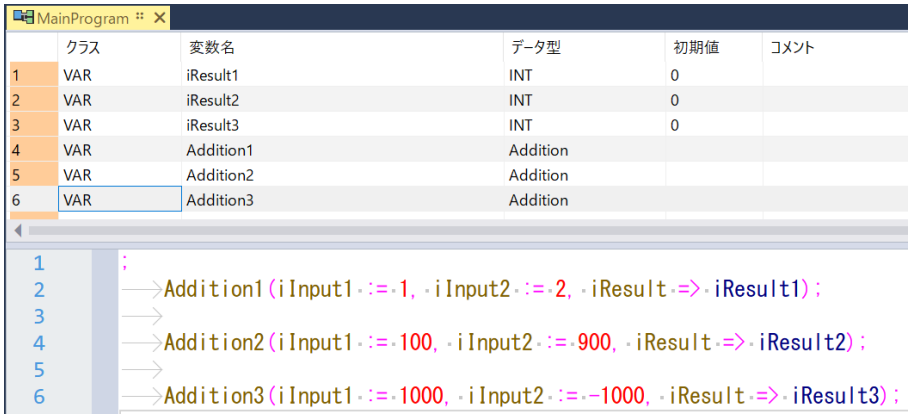
FB（ファンクションブロック）内をモニターする方法

内部の変数を＋1する FB を想定
複数の FB 呼び出しを行った場合、FB はそれぞれインスタンス変数で定義されます。
どの FB をモニターするのか、右クリック→インスタンスの指定で、指定する必要があります

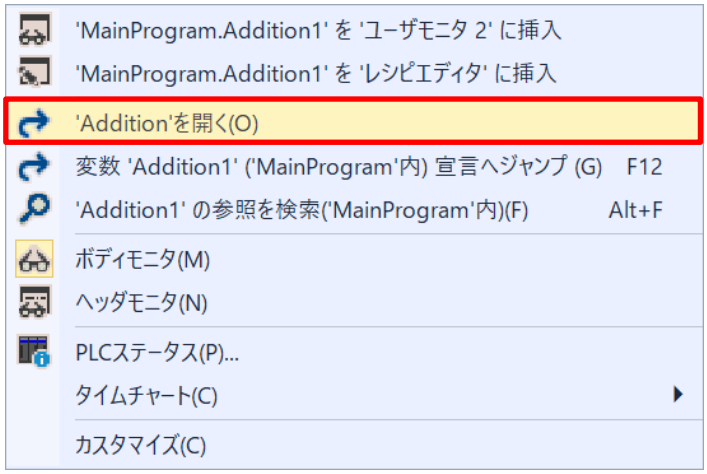
ここでは、ファンクションブロック Addition を定義してメインプログラムで呼び出す場合に
Addition の中のプログラムを確認しながら変数をモニターする方法について説明します。
ファンクションブロック Addition の定義は以下の内容とします。



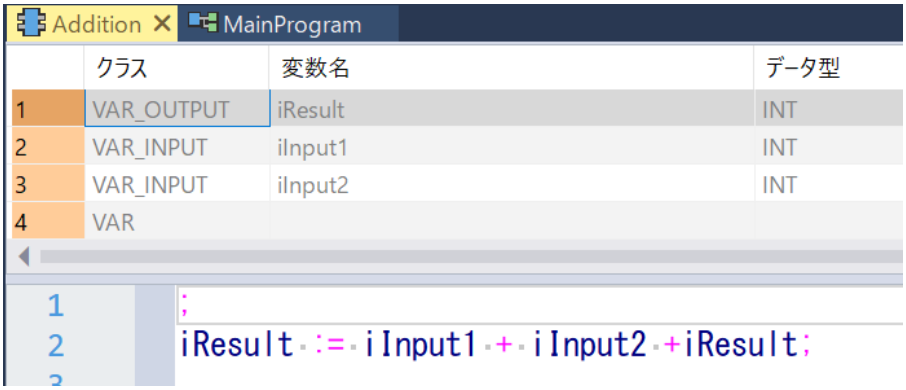
メインプログラムでの呼び出し例
ファンクションブロック Addition を 3 回呼び出しています。それぞれのインスタンスは Addtion1, Addtion2, Additon3 です。



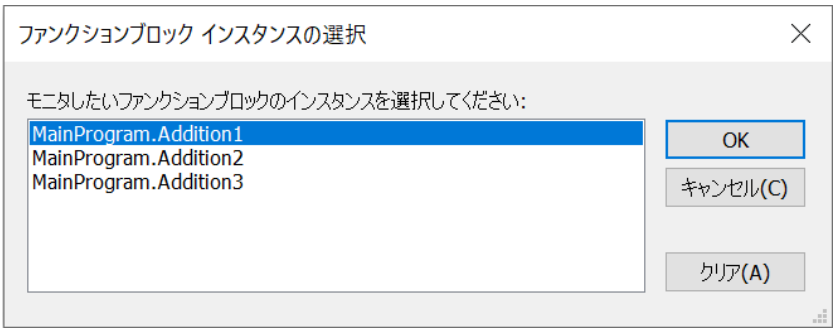
ここで、各インスタンスの中をモニターするには、各インスタンス変数上で右クリックし、下図のコンテキストメニューの Addtion を開くを選択します。



ファンクションブロック Addition を開いてプログラムが表示されます。

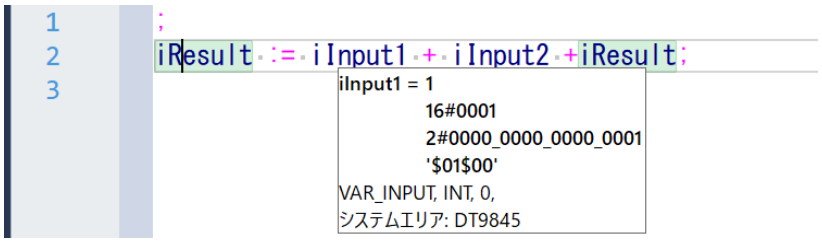


編集画面の任意の場所で右クリック→コンテキストメニュー→インスタンスの選択とクリックすると(左下図)
インスタンスの選択ダイアログが表示されるので、モニターしたいインスタンスを選択して[OK]をクリックします。

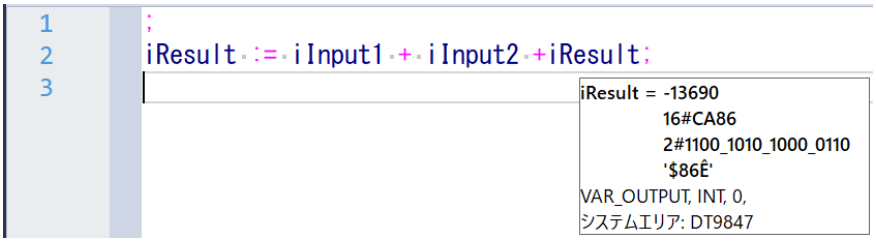


その後、モニターする変数の上にマウスカーソルを合わせると下図のように内容がモニターされます。

iInput1 にマウス ON した場合



右端の iResult にマウス ON した場合



あるインスタンスをモニター中でも、同様の方法で別のインスタンスを指定してモニターすることができます。

ファンクション（FUN）、ファンクションブロック（FB）内の使用命令確認方法

プロジェクトペインの POU タブ内で各ファンクション／ファンクションブロック内で使用されている命令を確認することができます。
演算エラーの原因を探るときなどに活用することができます。

■POU ヘッダ（FB:Test）

	クラス	変数名	データ型	初期値
1	VAR_INPUT	sInput	STRING[20]	''
2	VAR_OUTPUT	sOutput1	STRING[4]	''
3	VAR_OUTPUT	sOutput2	STRING[4]	''
4	VAR	iInput	INT	0

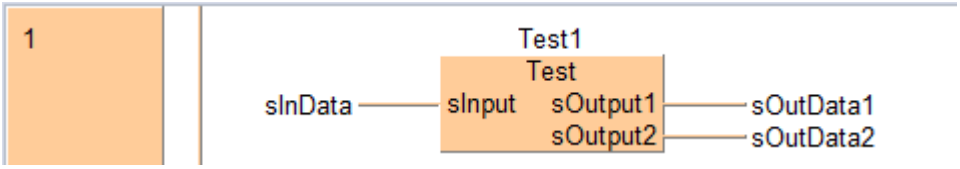
■POU ボディ（FB:Test）

```
iInput := LEN(sInput);
IF (iInput = 8) THEN
   sOutput1 := MID(IN := sInput,L := 4,P := 1);
   sOutput2 := MID(IN := sInput,L := 4,P := 5);
END_IF;
```

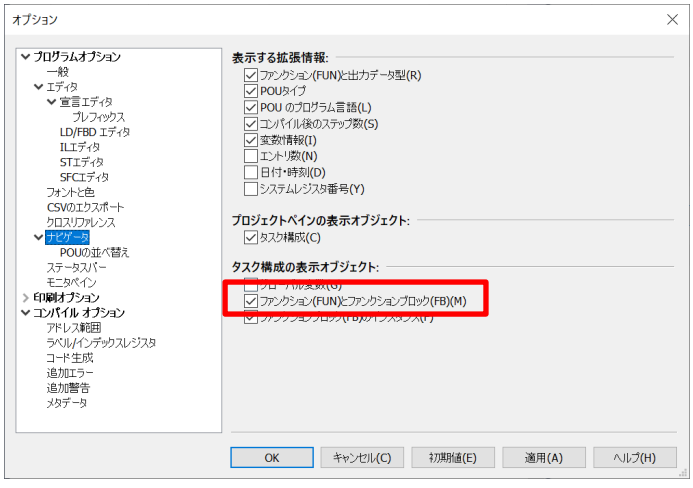
■POU ヘッダ（Prog）

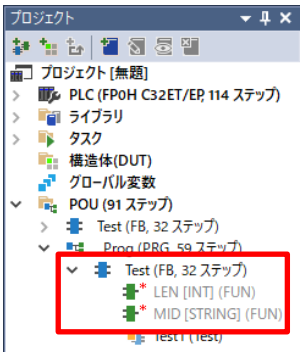
	クラス	変数名	データ型	初期値
1	VAR	Test1	Test	
2	VAR	sInData	STRING[20]	''
3	VAR	sOutData1	STRING[4]	''
4	VAR	sOutData2	STRING[4]	''

■POU ヘッダ（Prog）



メニューバーの「拡張機能」ー「オプション」ー「ナビゲータ」ー「タスク構成の表示オブジェクト」から、
「ファンクション（FUN）とファンクションブロック（FB）」を有効にする。





FB:Test 内で使用している命令（LEN,MID）を
確認することが出来ます。

コンパイル

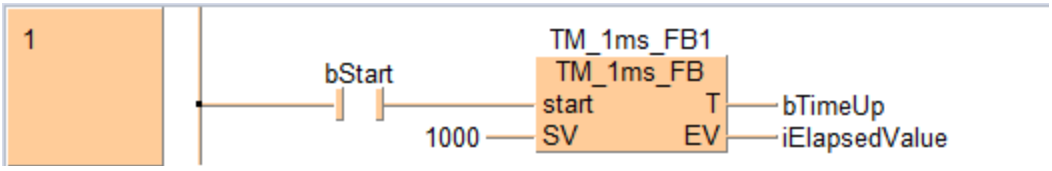
タイマ命令(ファンクションブロック(FB)使用による)コンパイル後の接点番号

ファンクションブロックを使用したタイマ命令は、その CPU が所有するタイマ領域の最後から使用します。
ここでは、簡単なタイマ命令を用いて紹介します。

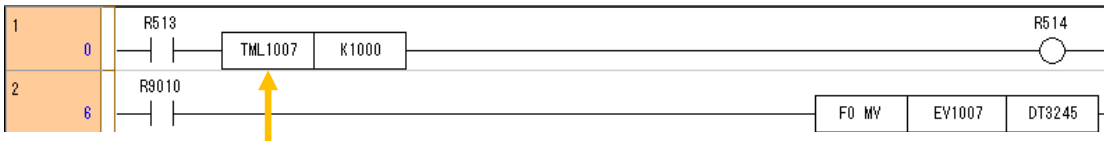
■POU ヘッダ(Prog)

	クラス	変数名	データ型	初期値
1	VAR	bStart	BOOL	FALSE
2	VAR	TM_1ms_FB1	TM_1ms_FB	
3	VAR	bTimeUp	BOOL	FALSE
4	VAR	iElapsedValue	INT	0

■POU ボディ(Prog)



■コンパイル後のプログラム(FPWIN GR7 ラダー表記)



タイマ領域の最後尾アドレスから使用

保持/非保持

No	名称	データ	単位	範囲
5	カウンタ開始アドレス	1008		0 - 1024
6	タイマ/カウンタ 保持エリア開始アドレス	1008		0 - 1024
7	内部リレー保持型エリアの開始アドレス(ワード...	504		0 - 512
8	データレジスタ保持型エリアの開始アドレス	32450		0 - 32765

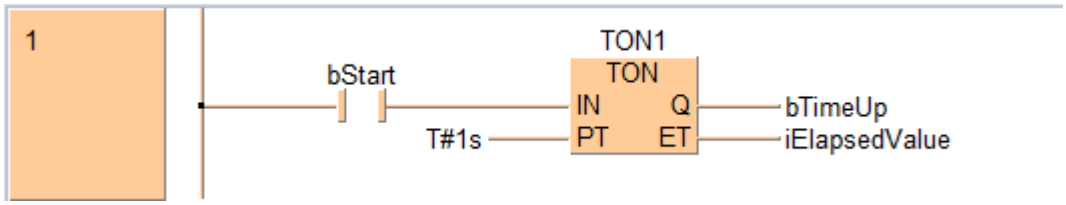
TON（オンディレイタイマ）命令と TM 命令の違い

TM 命令が PLC のタイマ命令を使用するのに対して、
TON 命令は補助タイマ命令（FP0H:F183 命令、FP7:SPTM 命令）を使用しています。

■POU ヘッダ（Prog）

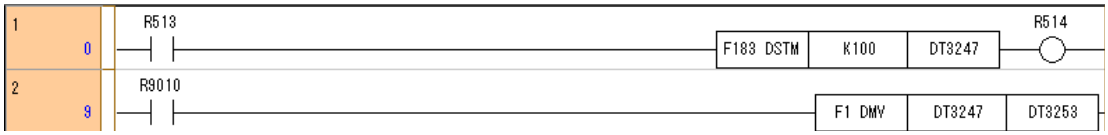
	クラス	変数名	データ型	初期値
1	VAR	TON1	TON	
2	VAR	bStart	BOOL	FALSE
3	VAR	bTimeUp	BOOL	FALSE
4	VAR	iElapsedValue	TIME	T#0s

■POU ボディ（Prog）

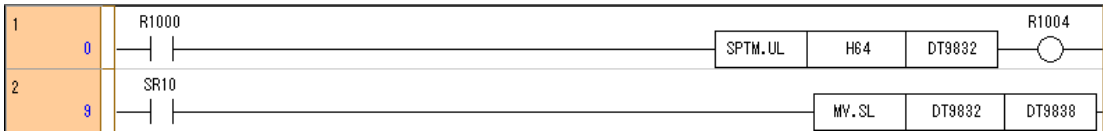


■コンパイル後のプログラム（FPWIN GR7 ラダー表記）

FP0H



FP7



設定時間（PT：TIME 型）は 32 ビットの領域を使用するため、0~21,474,836.47s の広い範囲で設定が可能です。
ただし、F183 命令、SPTM 命令の仕様からタイマ単位は 10ms という制限があります。

保持型変数(VAR_RETAIN)のデータレジスタ割り付け先

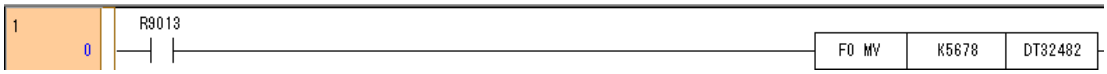
保持型変数(VAR_RETAIN)で宣言した変数はデータレジスタの保持エリアに割り付けられます。
つまり、PLC のシステムレジスタにより割り付け先が変わります。

ここでは、1 つの変数を VAR_RETAIN で宣言して、DT の保持型エリアに割付けられる例を用いて紹介します。

■POU ヘッダ(Prog)

	クラス	変数名	データ型	初期値
1	VAR_RETAIN	iVariable	INT	5678

■コンパイル後のプログラム(FPWIN GR7 ラダー表記)

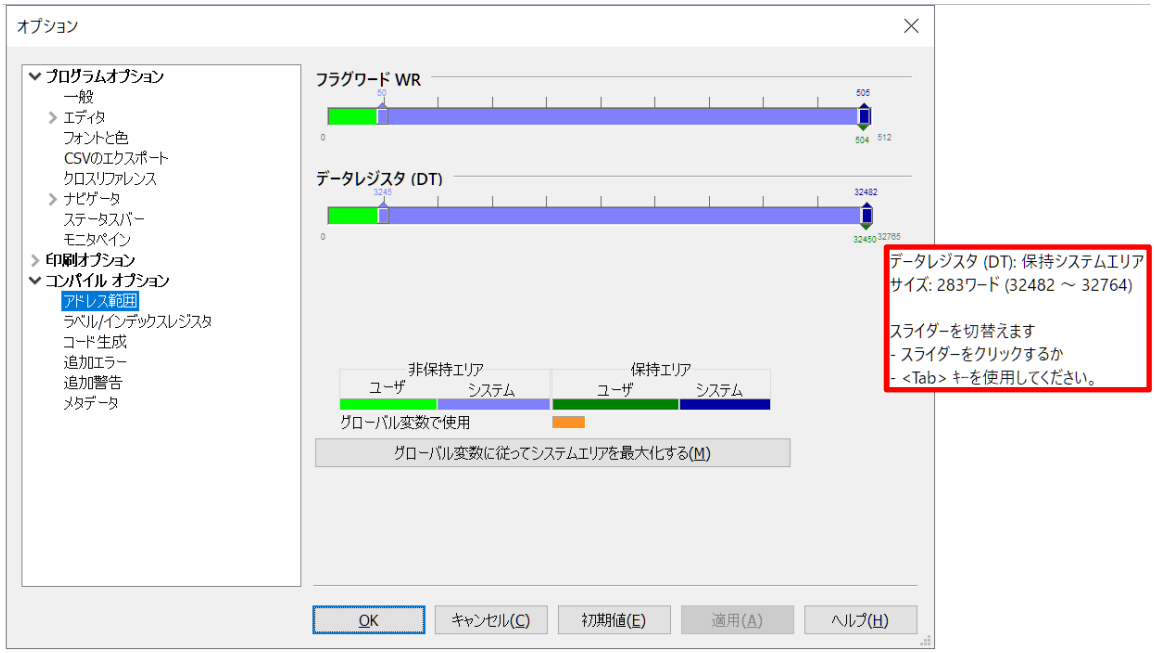


DT の保持エリアに
割り付けられていることがわかります。

●システムレジスタ

No	名称	データ	単位	範囲
5	カウンタ開始アドレス	1008		0 - 1024
6	タイマ/カウンタ 保持エリア開始アドレス	1008		0 - 1024
7	内部リレー保持型エリアの開始アドレス(ワード...	504		0 - 512
8	データレジスタ保持型エリアの開始アドレス	32450		0 - 32765
10	PLCリンク0用リンクリレー保持型エリアの開始...	64		0 - 64
11	PLCリンク1用リンクリレー保持型エリアの開始...	128		64 - 128

しかしながら、DT32450 から保持エリアに設定しているのにプログラムでは DT32482 から割り付けられています。



PLC での保持エリアは DT32450～DT32765 の 316 ワードですが、
上図例では、ユーザエリアとして 32 ワード、システムエリア 284 としてワードの設定になっています。
コンパイル時に自動で割り付けられるエリアとしては、"DT32482"が先頭になるため、
プログラムでは DT32482 から割り付けられます。

PLC へのプログラムダウンロード時、保持型データをクリアしない方法

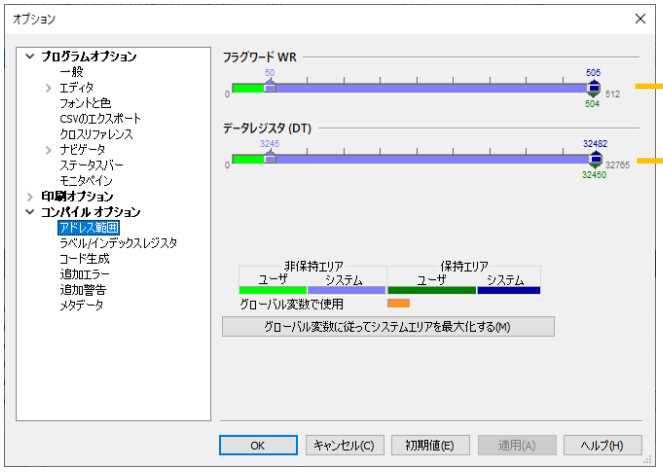
FPWIN Pro7 ではプログラムのダウンロード時にデータをクリアするという設定がデフォルトの状態になっています。
保持型に設定されたデータをクリアしないという設定に変更することも可能です。

ここでは、FP0HC32ET を用い、保持型に設定されたエリア（WR、DT）の例を用いて紹介します。

■システムレジスタ

No	名称	データ	単位	範囲
5	カウンタ開始アドレス	1008		0 - 1024
6	タイマ/カウンタ 保持エリア開始アドレス	1008		0 - 1024
7	内部リレー保持型エリアの開始アドレス(ワード単位)	504		0 - 512
8	データレジスタ保持型エリアの開始アドレス	32450		0 - 32765

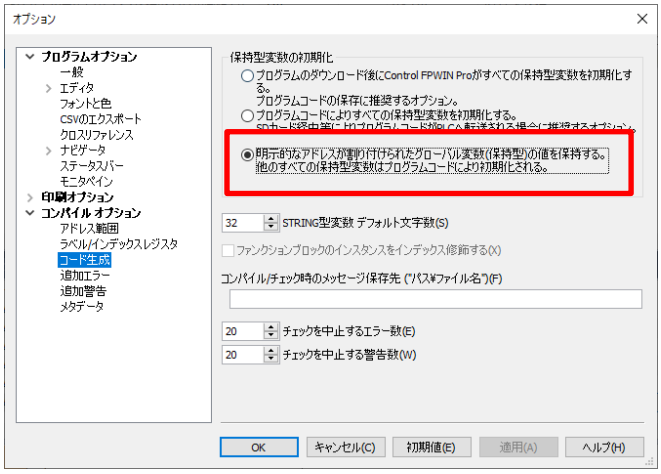
■オプション



フラグワード WR：保持ユーザエリア
サイズ：1 ワード（WR504）

データレジスタ DT：保持ユーザエリア
サイズ：32 ワード（DT32450～32481）

システムレジスタ No.7、No.8 で設定した保持型エリアは、保持ユーザエリアとして設定されます。
ツールバーの「拡張機能」ー「オプション」ー「コンパイルオプション」ー「アドレス範囲」からも設定することができます。

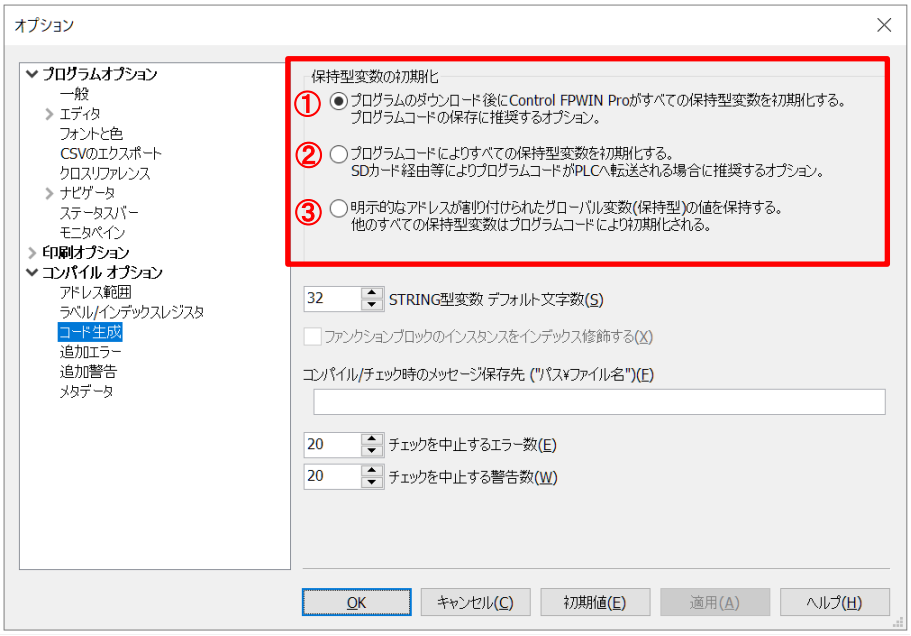


ツールバーの「拡張機能」ー「オプション」ー「コンパイルオプション」ー「コード生成」の項目内の
「明示的なアドレスが～」をチェックすることで、設定した保持ユーザエリアをプロジェクトデータダウンロード時に
データクリアしない設定に変更できます。

保持型変数の初期化

FPWIN Pro7 では保持型変数の初期化を 3 種類から設定することが出来ます。

ツールバーの「拡張機能」ー「オプション」ー「コンパイルオプション」ー「コード生成」を選択すると、以下の画面が表示されます。
保持型変数の初期化は 3 種類からひとつを選択します。
初期値は①です。



①を選択したときのコンパイル結果

```
 0  ST   R9013
 2  F61                (*DCMP*)
                DT32763
                K0
11  AN   R900B
13  F1                 (*DMV*)
                K485253606
                DT32763
20  ED
```

②を選択したときのコンパイル結果

```
 0  ST   R9013
 2  F61                (*DCMP*)
                DT32763
                K441946013
11  AN/  R900B
13  F1                 (*DMV*)
                K441946013
                DT32763
20  F11                (*COPY*)
                K0
                DT32450
                DT32762
27  F11                (*COPY*)
                K0
                WR504
                WR511
34  ED
```

プログラムコードで保持エリア（ユーザ保持エリア＋システム保持エリア）が初期化されています。

③を選択したときのコンパイル結果

```
 0  ST   R9013
 2  F61                (*DCMP*)
                DT32763
                K188947161
11  AN/  R900B
13  F1                 (*DMV*)
                K188947161
                DT32763
20  F11                (*COPY*)
                K0
                DT32482
                DT32762
27  F11                (*COPY*)
                K0
                WR505
                WR511
34  ED
```

プログラムコードでシステム保持エリアのみが初期化されています。

FPWIN Pro7 の機能

POU ヘッダ・ボディの分割表示

FPWIN Pro7 では同一ヘッダ ボディを分割表示することが可能です。

■POU ヘッダ（Prog）

	クラス	変数名	データ型	初期値
1	VAR	iData1	INT	0
2	VAR	iData2	INT	0
⋮				
19	VAR	iData19	INT	0
20	VAR	iData20	INT	0

20 のローカル変数：iData1～iData20 が宣言されています。

	クラス	変数名	データ型	初期値	コメント
1	VAR	iData1	INT	0	
2	VAR	iData2	INT	0	
3	VAR	iData3	INT	0	
4	VAR	iData4	INT	0	
5	VAR	iData5	INT	0	
6	VAR	iData6	INT	0	

左図赤四角部を下方向にドラッグすることで POU ヘッダを分割できます。

●分割後の POU ヘッダ

	クラス	変数名	データ型	初期値	コメント
1	VAR	iData1	INT	0	
2	VAR	iData2	INT	0	
3	VAR	iData3	INT	0	

	クラス	変数名	データ型	初期値	コメント
19	VAR	iData19	INT	0	
20	VAR	iData20	INT	0	

同じ POU ヘッダが分割して表示されています。

●分割した POU ヘッダの結合

	クラス	変数名	データ型	初期値	コメント
1	VAR	iData1	INT	0	
2	VAR	iData2	INT	0	
3	VAR	iData3	INT	0	

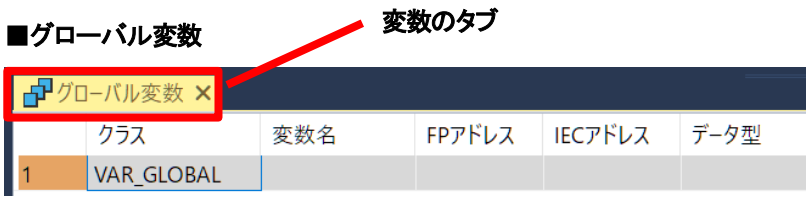
	クラス	変数名	データ型	初期値	コメント
19	VAR	iData19	INT	0	
20	VAR	iData20	INT	0	

分割したヘッダの境目を窓上部までドラッグすることで結合できます。

ホイールクリックによるタブのクローズ

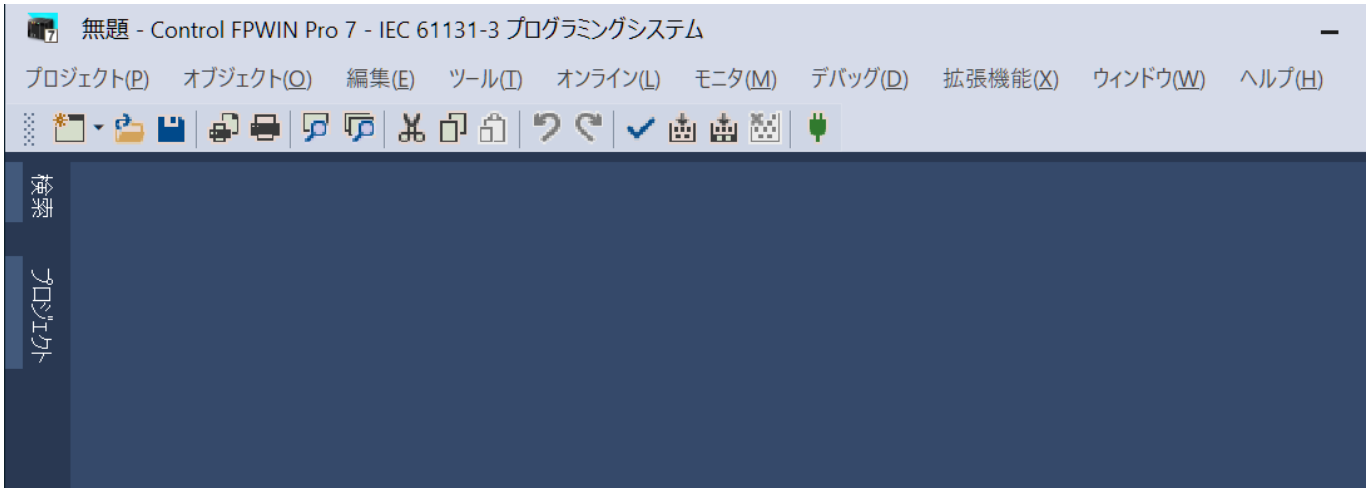
マウスのホイールクリック(ホイールの押し込み)によってタブをクローズすることができます。
ここではグローバル変数のタブを例に説明します。

■グローバル変数



タブにカーソルを合わせた状態で、ホイールクリック(ホイールの押し込み)でタブを閉じます。

■画面全体



タブが閉じました。

ペインレイアウトの任意配置

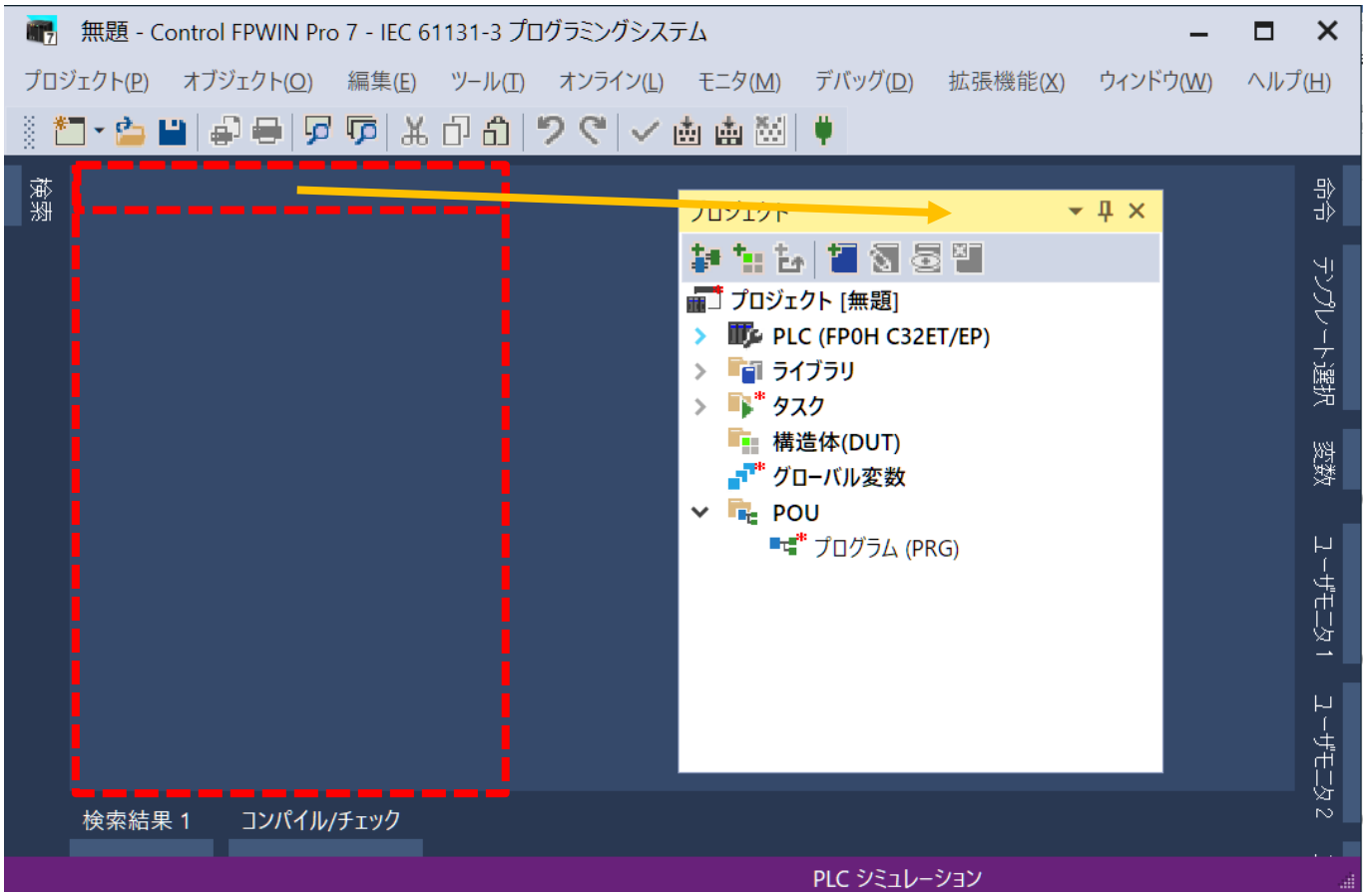
FPWIN Pro7 では各ペインを任意の位置に配置することが可能です。
ここではプロジェクトペインを用いて、任意の位置に配置する例を説明します。

① 赤枠部ピンをクリックして、ペインを固定状態に変更します。

固定状態

自動折りたたみ状態

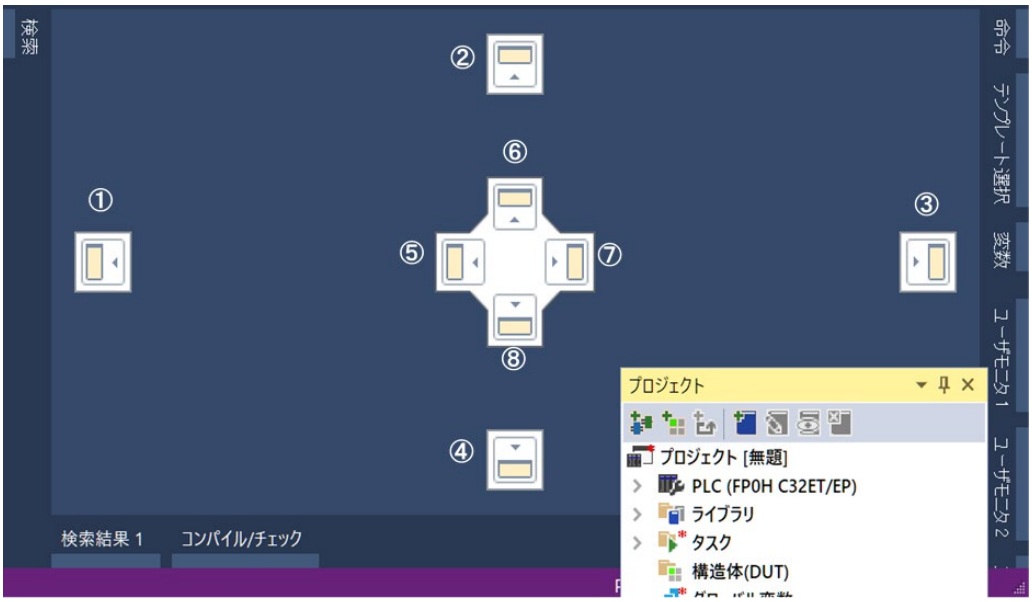
② ペインの枠上部をドラッグすることでペインを任意の位置に配置できます。



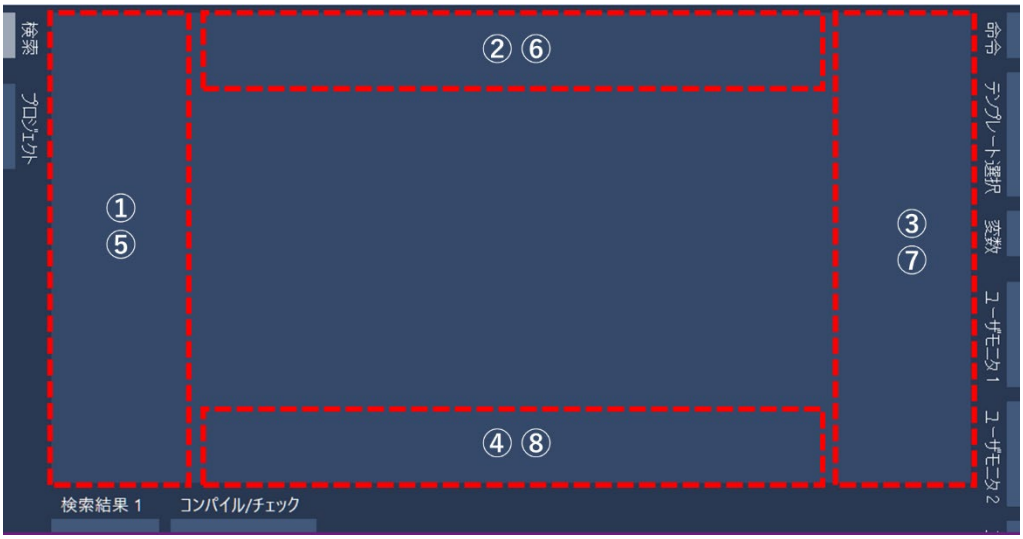
ペインレイアウトの固定

FPWIN Pro7 では各ペインを任意の位置に配置することが可能です。
ここではプロジェクトペインを用いて、任意の位置に配置する例を説明します。

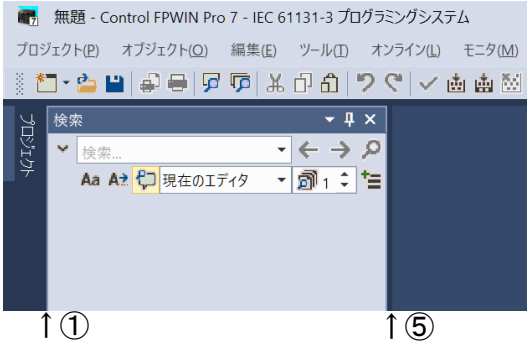
ペインの枠上部をドラッグし画面中央付近にカーソルを移動すると、下記のような画面が表示されます。



①から⑧のいずれかにカーソルを合わせると、下図の対応する数字の位置にペインが固定されます。



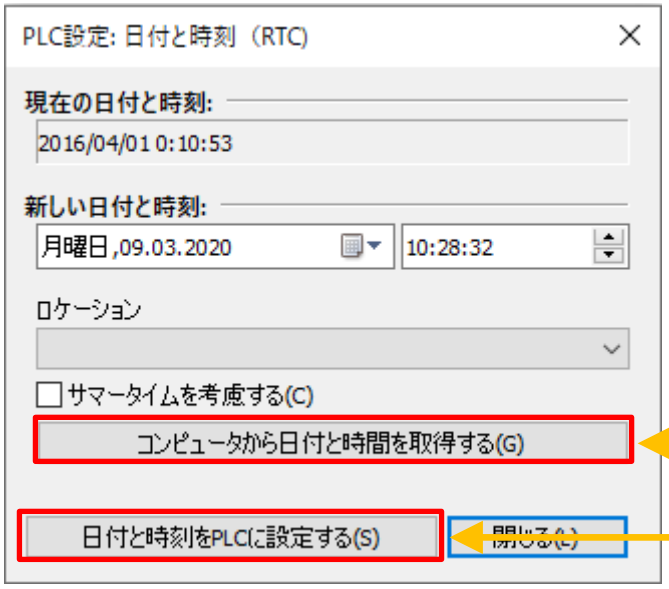
あらかじめ固定されたペインが配置されている場合、①から④は最も外側に、⑤から⑧は最も内側に配置されます。



PC の時計を使用しての PLC 時刻設定

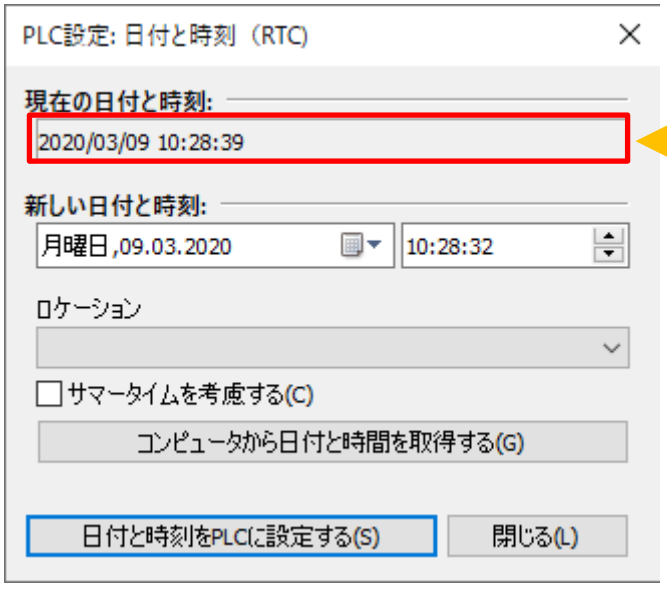
オンラインで接続している PC の時計を PLC の時刻に設定することが出来ます。。

オンラインの状態で、ツールバーの「モニタ」ー「PLC の時刻・日付」ー「PLC 設定：日付と時刻(RTC)」を開きます。



PC の時計の時刻を取得します。

PLC の時刻を更新します。



PLC 時刻が更新されました。

システム変数 sys と SYS の違い

システム変数には小文字「sys_～」で始まるものと大文字「SYS_～」で始まる 2 種類があります。

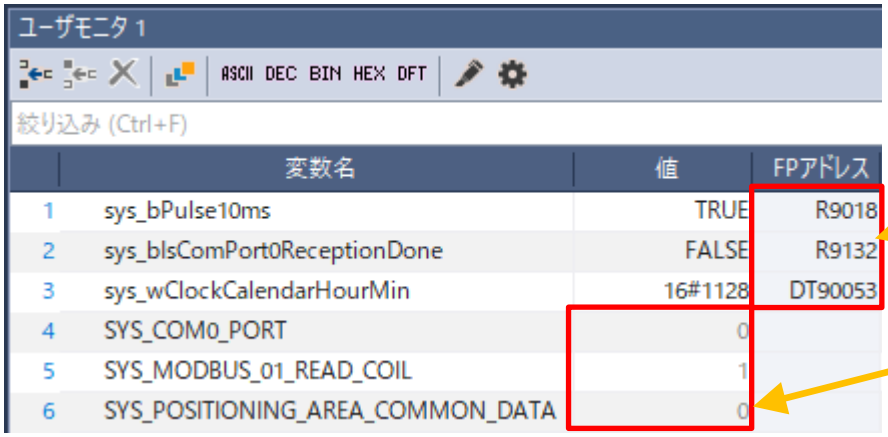
小文字「sys～」で始まるものは、ステータス情報や各種フラグ情報など変化値を割り当てた変数です。
特殊内部リレーや特殊データレジスタが割り当たります。

例）
sys_bPulse10ms
sys_bIsComPort0ReceptionDone
sys_wClockCalendarHourMin

大文字「SYS～」で始まるものは、ポート番号や MODBUS 番号などの固定値を割り当てた変数です。
それ以外にも、位置決めデータエリア番号やメモリエリア番号などが該当します。

例）
SYS_COM0_PORT
SYS_MODBUS_01_READ_COIL
SYS_POSITIONING_AREA_COMMON_DATA

●備考
ユーザモニタでシステム変数に割り当てられた内容を確認できます。



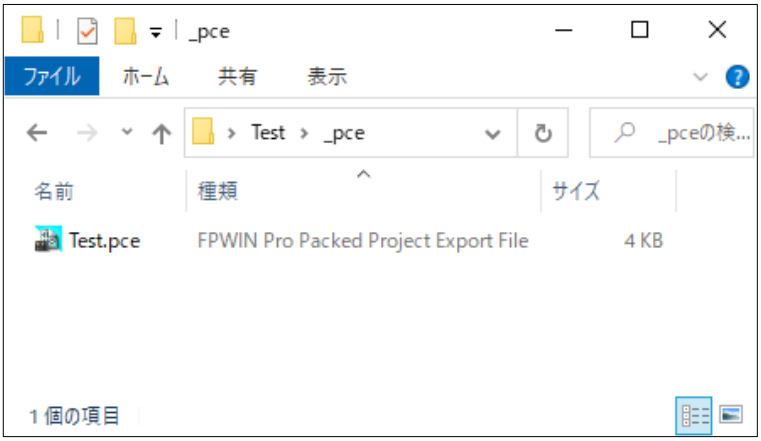
特殊内部リレー・特殊データレジスタが割り当てられています。

固定値が割り当てられています。

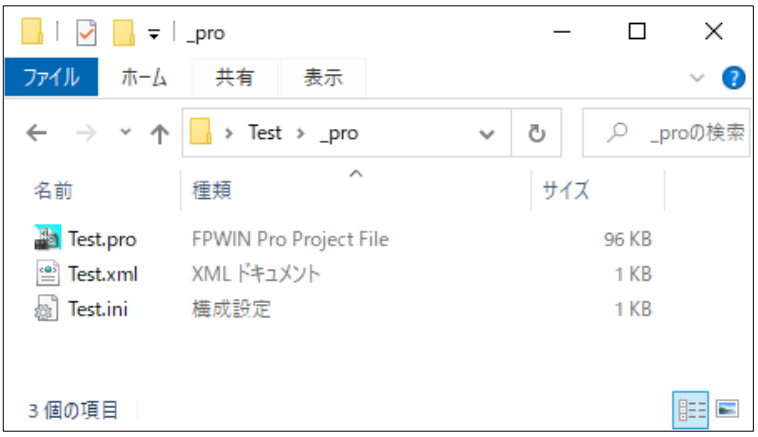
「.pce」と「.pro」とでファイルに保存される情報の違い

プロジェクトファイルを保存する場合、「.pro」のフォーマットの他に「.pce」の圧縮ファイルフォーマットを使用することが出来ます。
Test というプロジェクト名で保存を行った場合を例として説明をします。

「.pce」フォーマットの場合



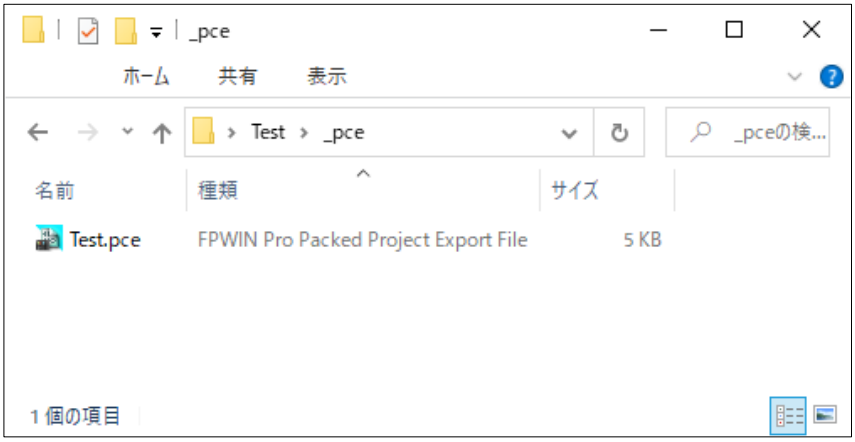
「.pro」フォーマットの場合



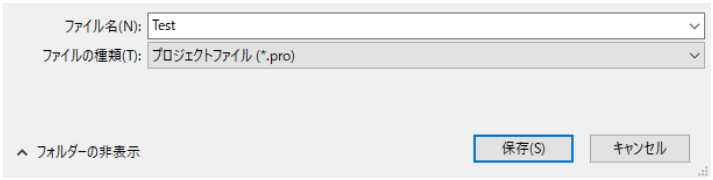
「.pce」フォーマットをで保存を行った場合、以下の特徴があります。
・保存したファイルサイズ「.pro」フォーマットと比較して小さくなる。
・「.xmi」「.ini」(「.sul」※上図未記載)等のファイルを含む。
・プロジェクトは未コンパイル状態となる。
・設定や POU 等の各種タブはすべて閉じた状態となる。

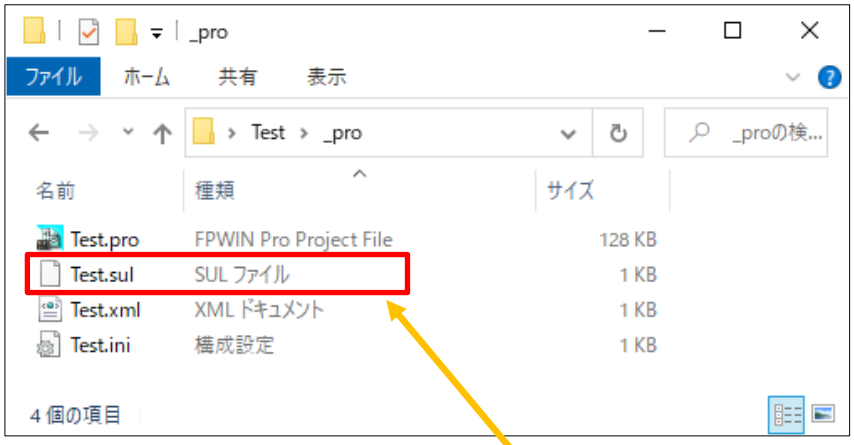
ライブラリを含む「.pce」を「.pro」で保存すると.sul が生成される

ライブラリを含むプロジェクトを圧縮フォーマット「.pce」で保存した場合、「.pce」内にライブラリファイルが圧縮されて保存されます。
そのプロジェクトをフォーマット「.pro」で保存した場合、ライブラリファイル「.sul」が自動生成されます。



「.pce」を「.pro」フォーマットで保存。





「.pro」が保存された階層フォルダにライブラリファイル「.sul」が保存されます。

この時、保存したプロジェクト上でのライブラリのパスは、上記で「.sul」が保存されたフォルダに自動で設定されます。

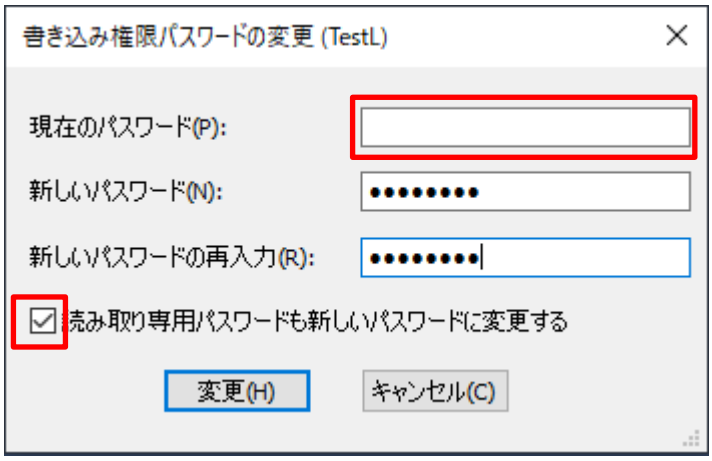
ライブラリ編集時のパスワード設定

ライブラリには「書き込み権限パスワード」と「読み取り専用パスワード」の 2 種類のパスワードを設定することが出来ます。

プロジェクトペインの「ライブラリ」ー「作成/インストールしたライブラリ」よりパスワード設定ウインドウを開きます。

書き込み権限パスワード　:ライブラリの編集に必要
読み取り専用パスワード　:ライブラリの閲覧に必要

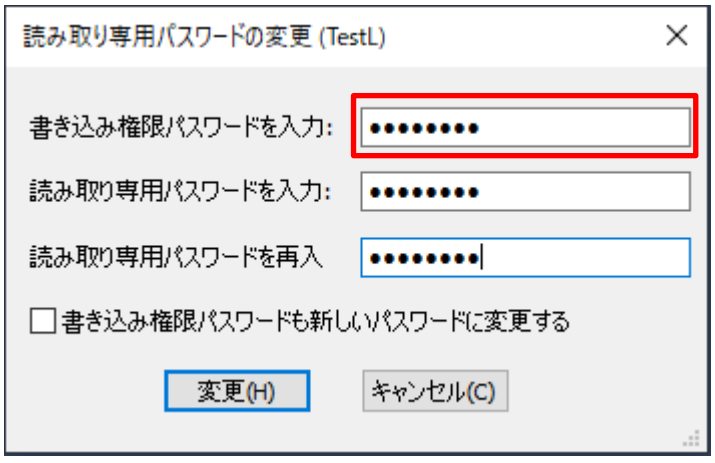
■書き込み権限パスワードの設定



初期状態では、パスワードは設定されていません。
初めてパスワードを設定する場合は「現在のパスワード」を空欄のまま「新しいパスワード」を入力してください。

チェックを入れて「変更」をすると、
「読み取り専用パスワード」に「書き込み権限パスワード」と同じパスワードが自動で設定されます。

■読み取り専用パスワードの設定



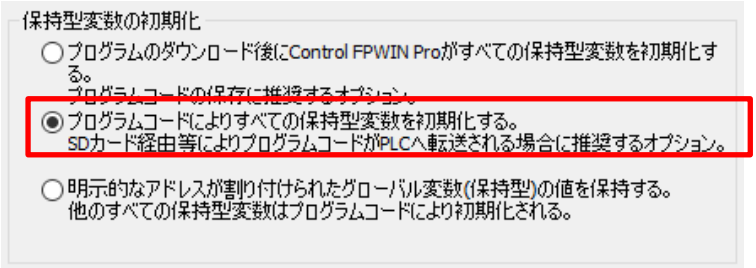
「読み取り専用パスワード」を設定するには、
「書き込み権限パスワード」をあらかじめ設定しておく必要があります。

※注意
パスワードを忘れてしまった場合は 2 度とライブラリの内容を読み出せなくなってしまします。
パスワードは大切に保管しておく必要があります。

SD カード運転ファイルの生成

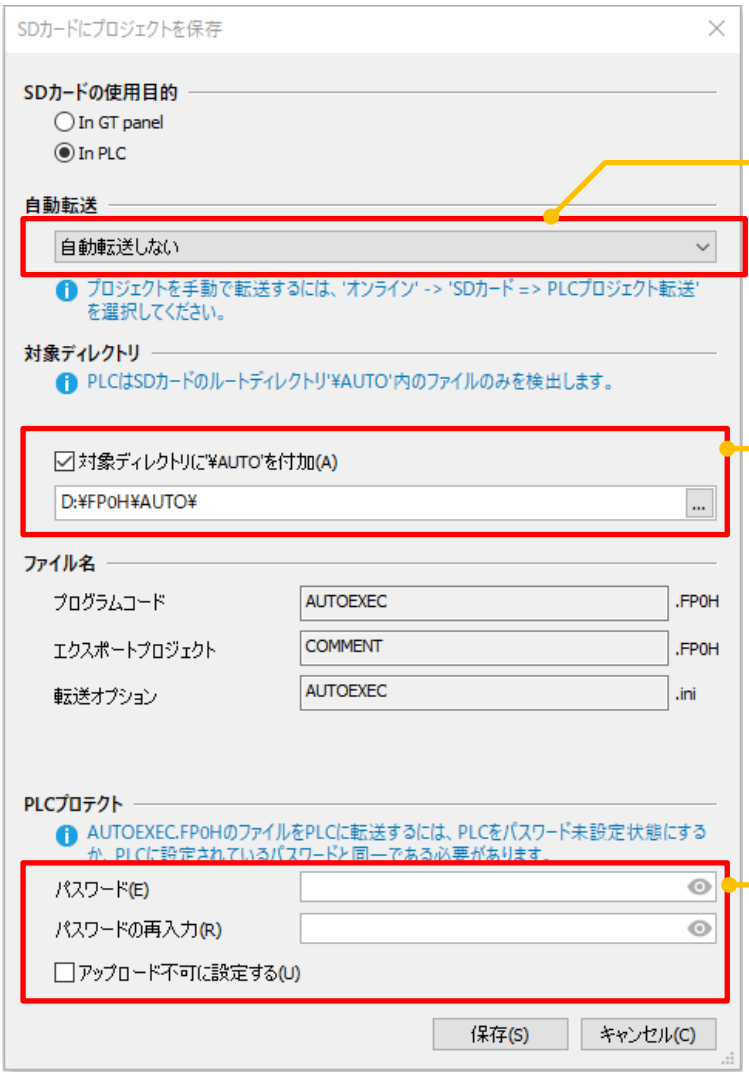
メニューバーの「プロジェクト」ー「名前を付けて保存」ー「SD カードのプロジェクト」より、SD カード運転ファイルを生成することができます。

SD カード運転を行う場合、保持型変数の初期化設定を「プログラムコードよりすべての保持型変数を初期化する。」に設定してください。デフォルトの「プログラムのダウンロード後に～」の設定で SD カードからプログラムが転送された場合、保持型変数の初期化が実行されません。



FPWIN Pro7 からダウンロードを行う場合は保持型変数の初期化をソフトから行いますが、
SD カード運転時は FPWIN Pro7 と未接続状態のためこの処理を行うことが出来ません。
上記設定を行うことで、保持型変数の初期化を行うプログラムコードを自動生成し初期化処理を行います。

●SD カードプロジェクト保存ウインドウ



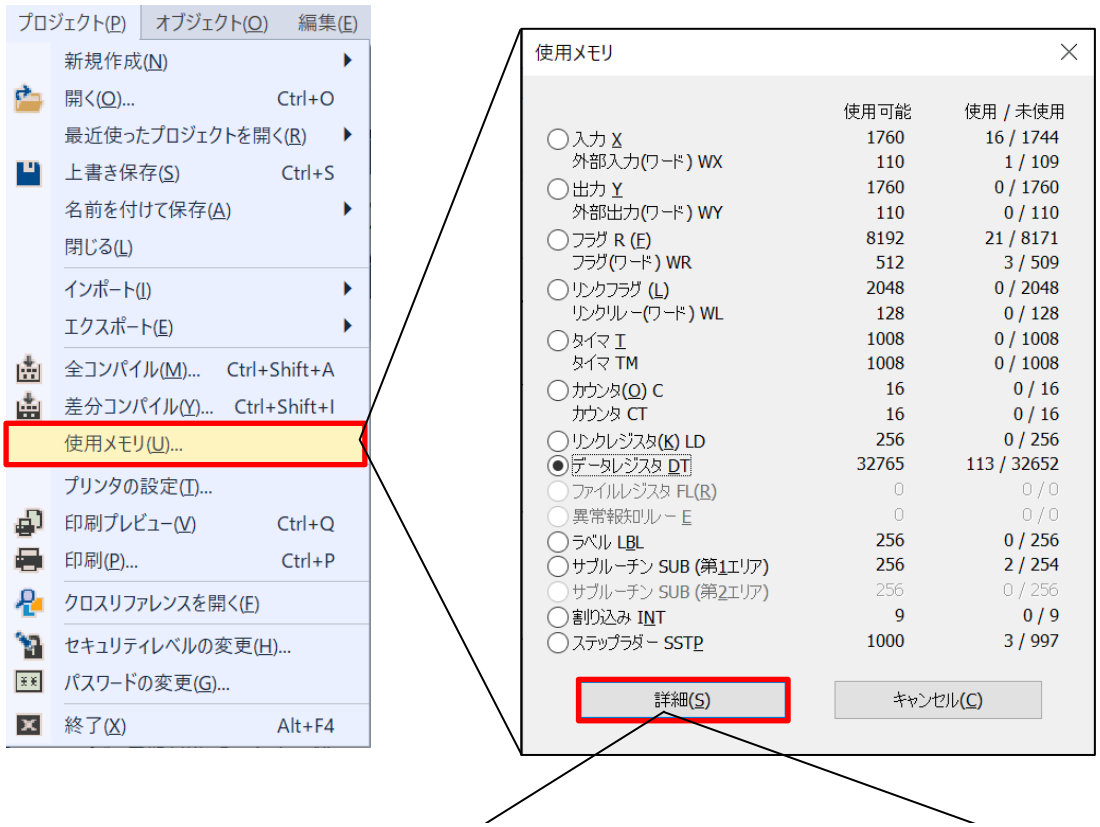
自動転送タイプを「自動転送しない」、
「PROG モードで電源を投入した場合」、
「PROG モードから RUN モードに切り替えた場合」
の中から選択します。

自動運転ファイルの保存場所を指定します。
自動実行ファイルはフォルダ名「AUTO」内に
格納される必要がありますが、☑ を入れることで
「AUTO」フォルダを自動生成してくれます。

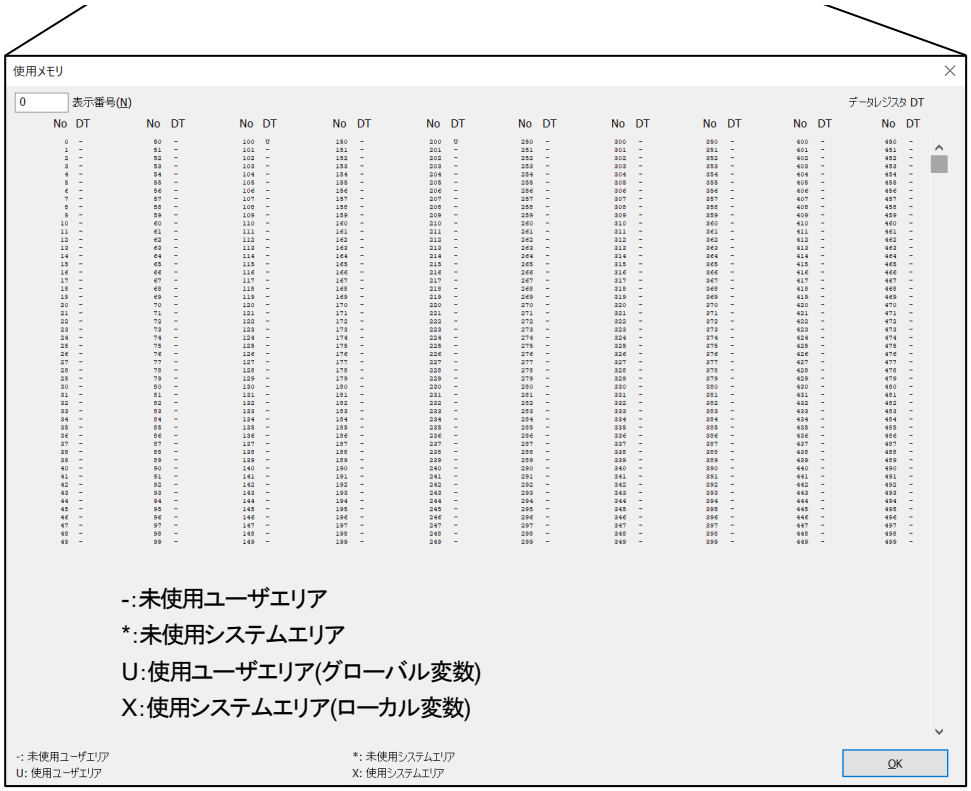
PLC 側にパスワードが設定されている場合、
自動実行ファイルにも同じパスワードを設定
する必要があります。

使用メモリの確認

メニューバーの「プロジェクト」ー「使用メモリ」を選択すると、使用/未使用メモリエリアの一覧が表示されます。



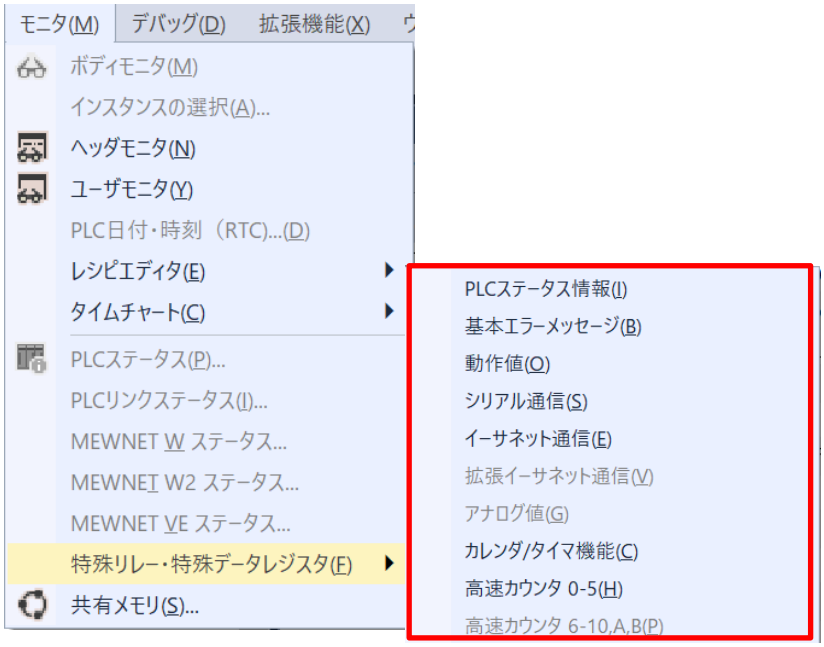
[詳細]ボタンを押すと、“使用メモリ”ダイアログボックスで選択した項目に対する詳細情報が表示されます。



特殊データレジスタの確認

各 PLC に割り当てられている特殊データレジスタや特殊内部リレーの状態をモニタすることができます。

メニューバーの「モニタ」ー「特殊リレー・特殊データレジスタ」から確認をすることできます。
「PLC ステータス情報」や「基本エラーメッセージ」の様に機能ごとにモニタが分かれています。



例えば、「PLC ステータス情報」をクリックすると下図のように対応する特殊データレジスタが確認できます。

	変数名	値	FPアドレス	
1	sys_bTrue	TRUE	R9010	常時ONリレー
2	sys_bFalse	FALSE	R9011	常時OFFリレー
3	sys_bIsFirstScan	FALSE	R9013	イニシャルパルスリレー(ON)
4	sys_bIsNotFirstScan	TRUE	R9014	イニシャルパルスリレー(OFF)
5	sys_bIsFirstScanOfSfcStep	FALSE	R9015	ステップラダー イニシャルパルスリレー(ON)
6	sys_bScanPulse	TRUE	R9012	スキャンパルスリレー
7	sys_bPulse10ms	TRUE	R9018	0.01秒クロックパルスリレー
8	sys_bPulse20ms	TRUE	R9019	0.02秒クロックパルスリレー
9	sys_bPulse100ms	FALSE	R901A	0.1秒クロックパルスリレー
10	sys_bPulse200ms	TRUE	R901B	0.2秒クロックパルスリレー
11	sys_bPulse1s	FALSE	R901C	1秒クロックパルスリレー
12	sys_bPulse2s	FALSE	R901D	2秒クロックパルスリレー
13	sys_bPulse1min	FALSE	R901E	1分クロックパルスリレー
14	sys_iRingCounter_2ms5	-19122	DT90019	RINGカウンタ 2.5ms
15	sys_iRingCounter_10usXX	-4440	DT90020	リングカウンタ値。分解能はシステム定数'SYS_RINGCOUNTER_
16	sys_bIsPlcInRunMode	TRUE	R9020	RUN モードフラグ
17	sys_bIsValueForced	FALSE	R9029	強制中フラグ
18	sys_bIsExternalInterruptEnabled	FALSE	R902A	外部割り込み許可フラグ
19	sys_bIsFirstScanAfterDownloadChangesDuringRun	FALSE	R9034	RUNモード中プログラム編集フラグ

ユーザモニタに D&D で変数を一括登録

宣言された変数をドラッグ&ドロップ(D&D)でユーザモニタに一括登録することができます。
変数ペインまたは、変数宣言エディタのいずれかからこの操作を行うことができます。

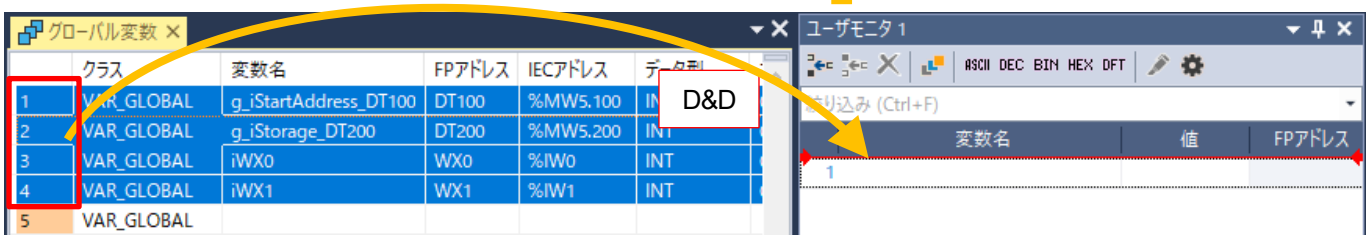
■グローバル変数

	クラス	変数名	FPアドレス	IECアドレス	データ型	初期値
1	VAR_GLOBAL	g_iStartAddress_DT100	DT100	%MW5.100	INT	0
2	VAR_GLOBAL	g_iStorage_DT200	DT200	%MW5.200	INT	0
3	VAR_GLOBAL	iWX0	WX0	%IW0	INT	0
4	VAR_GLOBAL	iWX1	WX1	%IW1	INT	0

変数ペイン/変数宣言エディタ内の変数を選択し、ユーザモニタの任意の場所にドラッグ &ドロップします。
挿入場所はユーザモニタの赤矢印部分(下図参照)になります。

●変数ペインから一括登録する

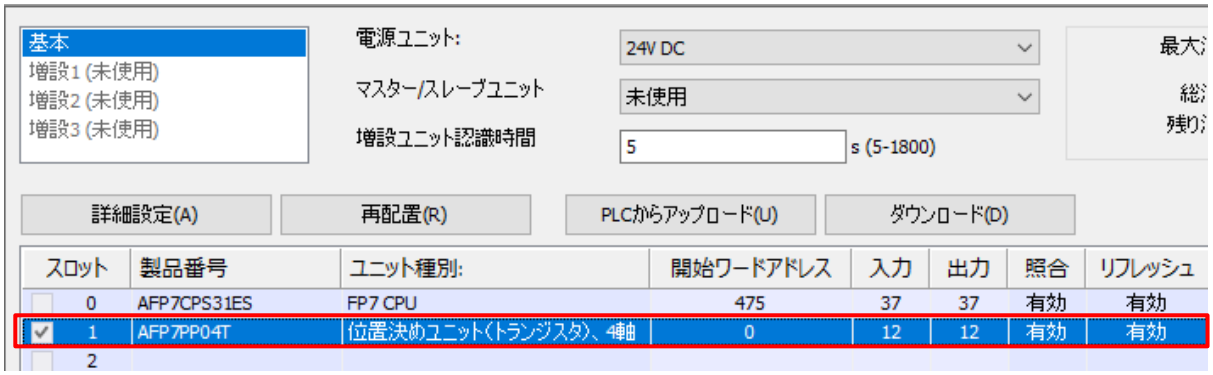
●変数宣言エディタから一括登録する



※変数宣言エディタの左端列を選択してドラッグして下さい。

ユニットメモリの変数宣言

ユニットメモリ(UM)の登録方法について説明します。
今回はスロット 1 に装着した FP7 位置決めユニットを例にして説明します。



ユニットメモリを変数で宣言する場合、FP アドレスに「S＊:UM＊＊＊＊＊」と登録します。
S＊にスロット No.、UM＊＊＊＊＊にユニットメモリ番号を入力します。

1 軸の実行中/実行完了テーブル No と、1 軸の現在値を登録する場合、
グローバル変数に次のように登録を行います。

■グローバル変数

	クラス	変数名	FPアドレス	IECアドレス	データ型	初期値
1	VAR_GLOBAL	g_iExecutionTableNo	S1:UM00438	%MW20.1.1080	INT	0
2	VAR_GLOBAL	g_diElapsedValue	S1:DUM0043C	%MD20.1.1084	DINT	0

現在値情報はダブルワードとなるためこの場合は、ダブルユニットメモリ「DUM＊＊＊＊＊」と登録します。

ユーザモニタでの連続したデバイス番号の登録

ユーザモニタには、連続したデバイス番号の登録が可能です。

●ユーザモニタ

DT0 の左隣にある ◢ を押すと折りたたむこともできます。

上記のように DT0 と DT100 の間に “-”（ハイフン）を入れることで登録されます。

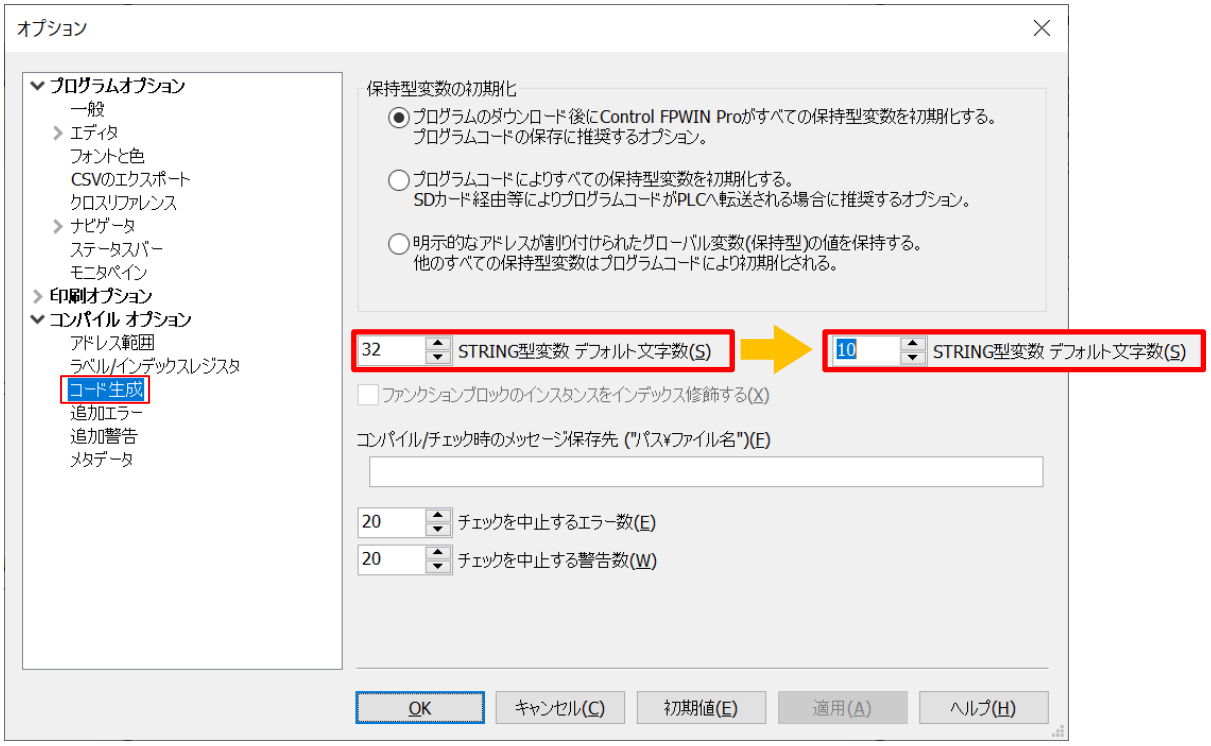
STRING 型宣言時の文字数初期値の変更

STRING 型を宣言した際、下図のようにデータ型には STRING[32]と入力されます。
これにより宣言した STRING 型変数の格納文字数は 32 となります。この値はデフォルトの値です。

■POU ヘッダ（Prog）

	クラス	変数名	データ型	初期値
1	VAR	sString0	STRING[32]	''

メニューバーの「拡張機能」ー「オプション」ー「コンパイルオプション」ー「コード生成」から、
STRING 型変数のデフォルト文字数を変更できます。



STRING 型をする場合、文字数が 32 も必要でない場合は先に変えておくと便利です。

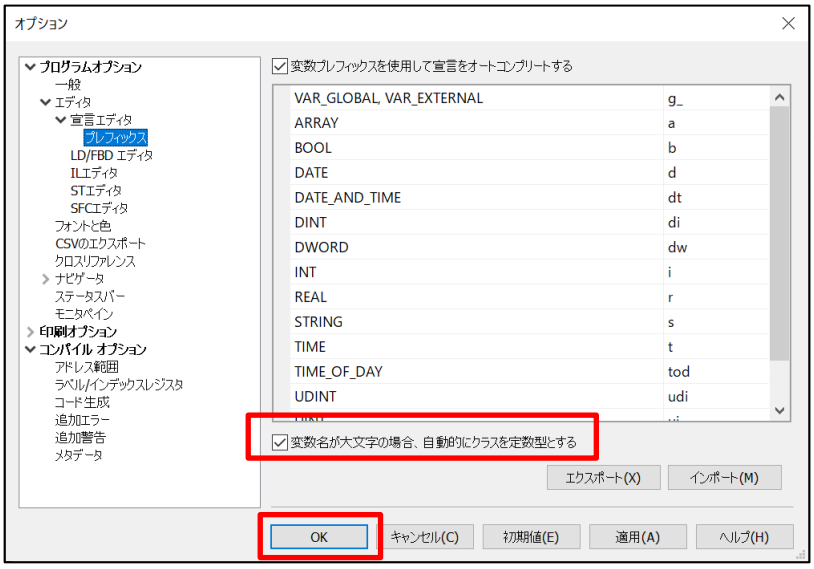
変更後に宣言した「sString1」は、下図のように STRING[10]がデフォルトになりました。

	クラス	変数名	データ型	初期値
1	VAR	sString0	STRING[32]	''
2	VAR	sString1	STRING[10]	''

VAR_CONSTANT の自動宣言

「クラス」を手動で変更することなく大文字で変数名を宣言することで、VAR_CONSTANT を自動で宣言することができます

メニューバーの「拡張機能」ー「オプション」ー「エディタ」ー「宣言エディタ」ー「プレフィックス」から、
「変数名が大文字の場合、自動的にクラスを定数型とする」に☑を入れて、「OK」ボタンを押します。



■POU ヘッダ

	クラス	変数名	データ型	初期値
1	VAR	iVARIABLE	INT	0
2	VAR			

変数名はプレフィックス以外の部分をすべて大文字で入力します。

	クラス	変数名	データ型	初期値
1	VAR_CONSTANT	iVARIABLE	INT	0
2	VAR			

VAR_CONSTANT で宣言されます。

未使用変数削除

未使用変数削除で削除したくないが、プログラムで使用しない（使用する予定の）変数は、
LD 言語のコンパイル対象外ネットワークまたは ST 言語のコメントに入力しておくことで、削除されません。

例）GT の予約エリア（DT0-2,WR0-2）をプログラムでは使用しないが、宣言はしておきたい場合などに使用できます。

■グローバル変数

	クラス	変数名	FPアドレス	IECアドレス	データ型	初期値
1	VAR_GLOBAL	g_aiGTReserveArea	DT0	%MW5.0	ARRAY [0..2] OF INT	[3(0)]
2	VAR_GLOBAL	g_awGTReserveArea	WR0	%MW0.0	ARRAY [0..2] OF WORD	[3(0)]

■POU ボディ（LD 言語）

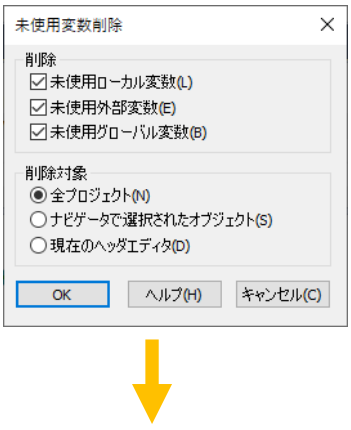
コンパイル対象外ネットワークに入力

■POU ボディ（ST 言語）

コメント内に変数を入力

```
(* g_aiGTReserveArea g_awGTReserveArea *)
```

POU ヘッダ内を右クリックして「未使用変数削除」から未使用変数の削除を行います。



LD でコンパイル対象外／ST でコメント入力した変数はコンパイル対象外の変数としてヘッダ内に残ります。

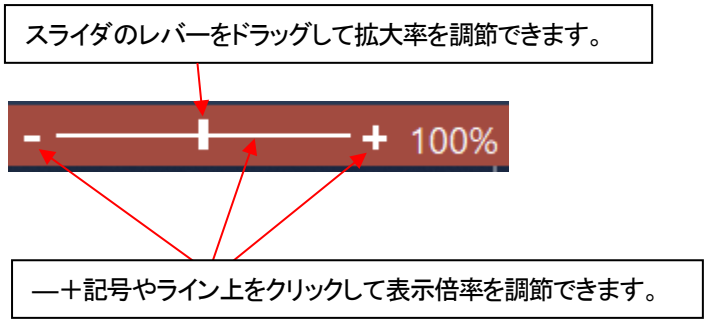
	クラス	変数名	FPアドレス	IECアドレス	データ型	初期値
1	VAR_GLOBAL	g_aiGTReserveArea	DT0	%MW5.0	ARRAY [0..2] OF INT	[3(0)]
2	VAR_GLOBAL	g_awGTReserveArea	WR0	%MW0.0	ARRAY [0..2] OF WORD	[3(0)]

エディタ画面の拡大縮小　便利な機能

PC の画面が小さくて見えにくい時にエディタ面が拡大縮小できる機能は便利です。
拡大率を変更するには、主に以下の 2 つの方法があります。

1. 編集画面右下の拡大率調整バーを操作する
2. エディタ画面上で CTRL キー＋マウスホイール操作で拡大縮小する

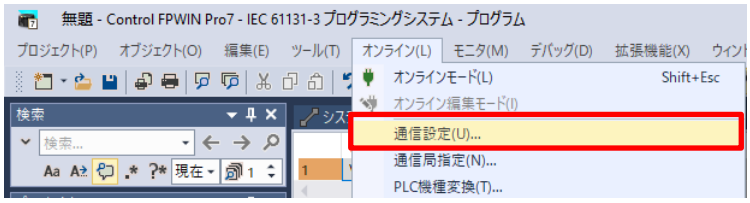
モニター中の画面も拡大・縮小可能です。



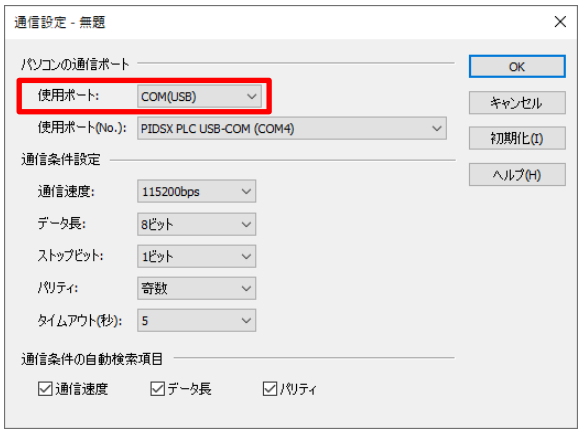
オンライン通信設定接続先の自動検出

Control FPWIN Pro7 で、Eterhnet を使用してオンラインの通信設定を行う際、接続先の自動検出を行うことができます。

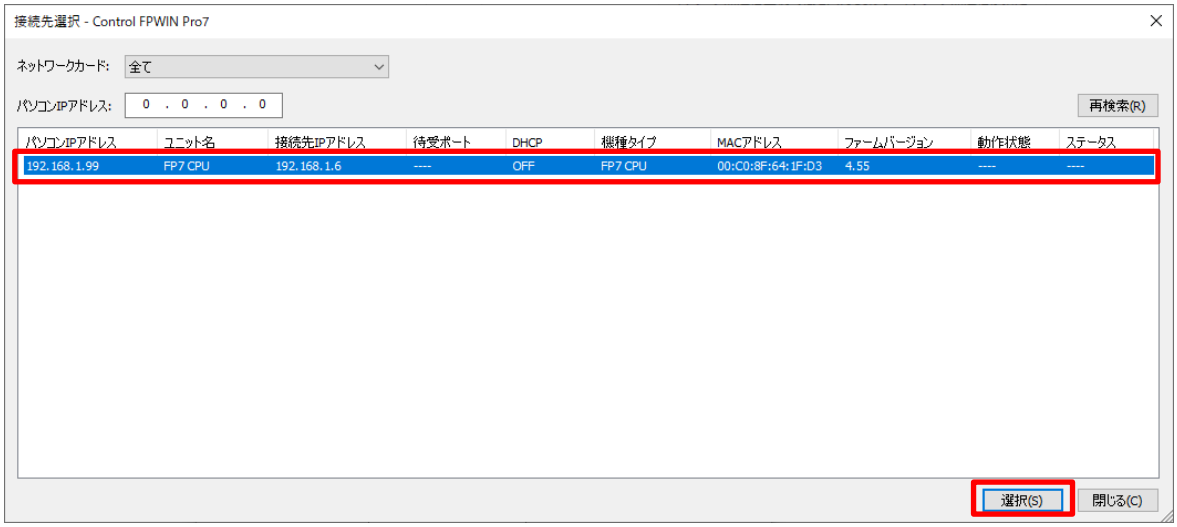
操作を行う前に、Control FPWIN Pro7 をインストールしたパソコンと、検索対象の機器が LAN ケーブルで接続されていることを確認してください。
ツールバーの「オンライン」から「通信設定」を選択します。



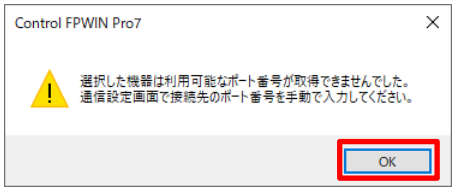
「通信設定」ダイアログが開きます。
「通信ポート」を「Ethernet」に変更し、「接続先選択」を選択します。



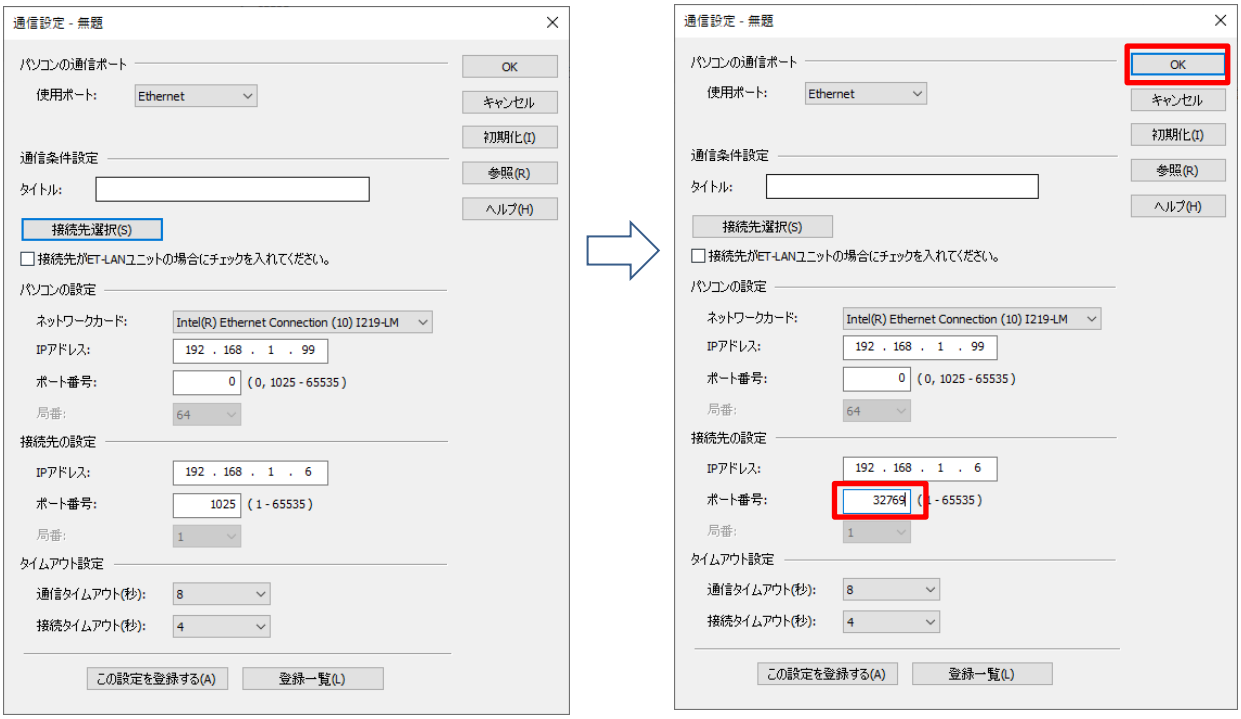
ダイアログが開き、Ethernet 接続されている PLC が検索されますので、接続を行う機器を選択して、「選択」をクリックします。



対象機器が FP7 の場合、下図のポップアップが出ますので「OK」をクリックします。

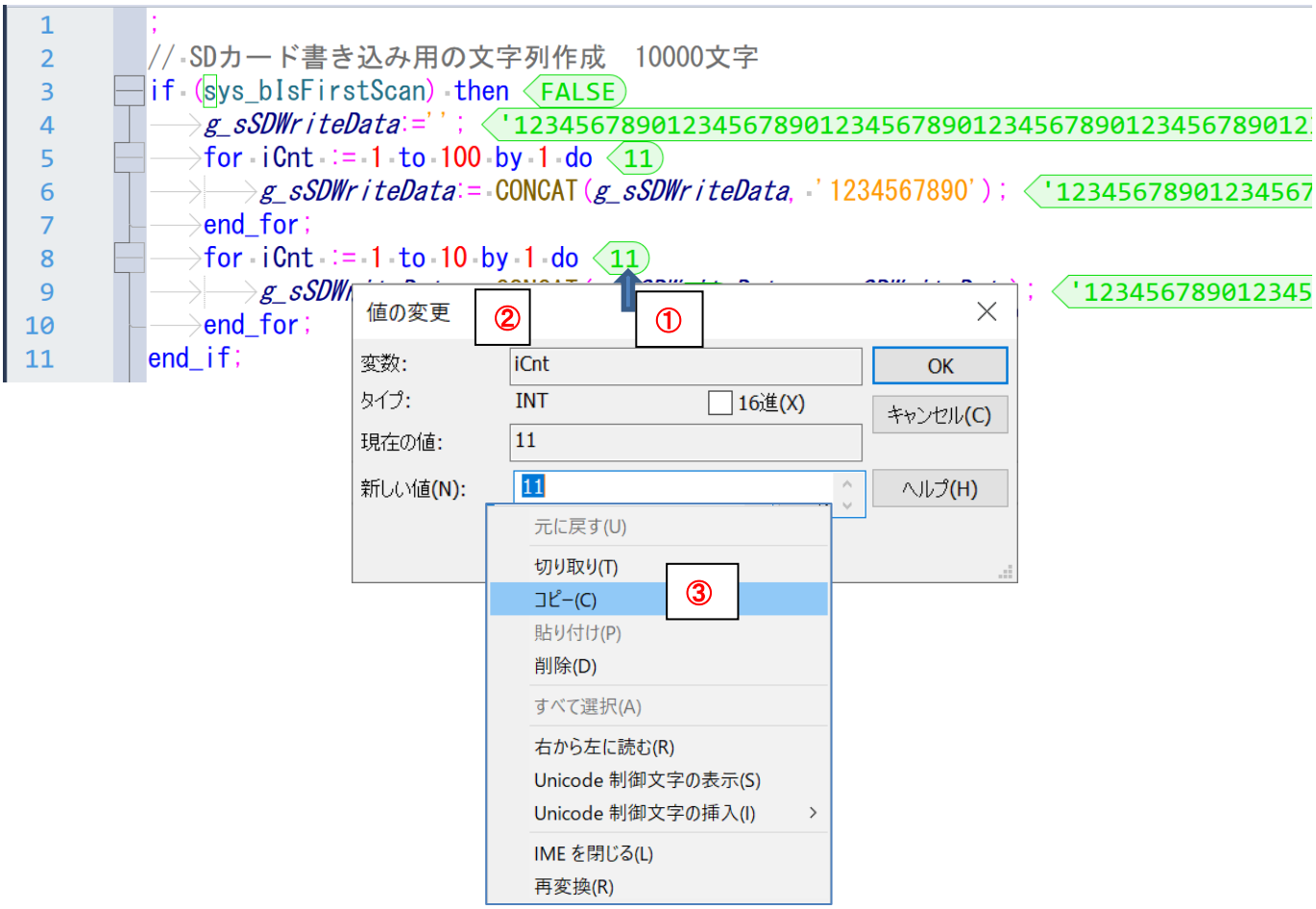


「通信設定」ダイアログに、パソコン側と PLC 側の IP アドレスが入力され、ポート番号は暫定のものが仮入力されます。
パソコン側は空きポート、PLC 側はシステムコネクションで設定されたポートを確認して入力してください。
下例では、PLC 側のポート番号に、32769 を使用しています。
「OK」をクリックすることで、通信設定が完了します。

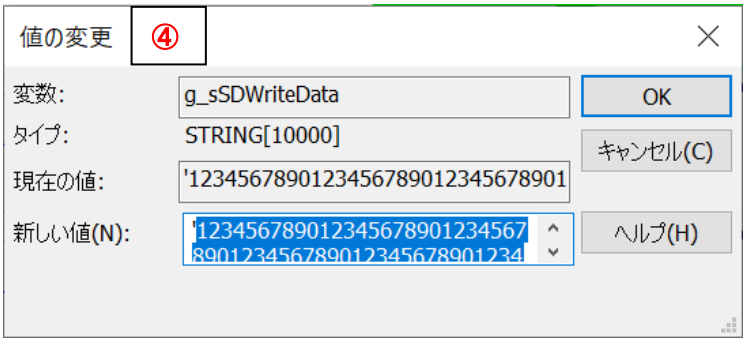


プログラム編集画面でのモニタデータの取得と保存　便利な機能

プログラムを PLC にダウンロードするとプログラム編集画面に表示されている変数のモニタが自動的に開始されます。
この変数のモニター値をクリックすると(下図の①)、値の変更ダイアログが表示され(下図の②)、別の値に変更することが可能です。
このダイアログで、新しい値にはダイアログを開いた時のモニタ値が自動的にコピーされているので、
これを右クリックしてコンテキストメニューを表示させ、コピーする(下図の③)ことができます。
これらのモニタ結果をテスト結果のエビデンスとして残したい場合やモニタ結果をエディタなどで詳細に確認する場合、コピーすることによりテキストエディタやエクセルなどに張り付けて確認・整理・記録することができます。
String 型の変数に対しても同様の操作が可能です。(下図の④string 型変数の値の変更ダイアログ)

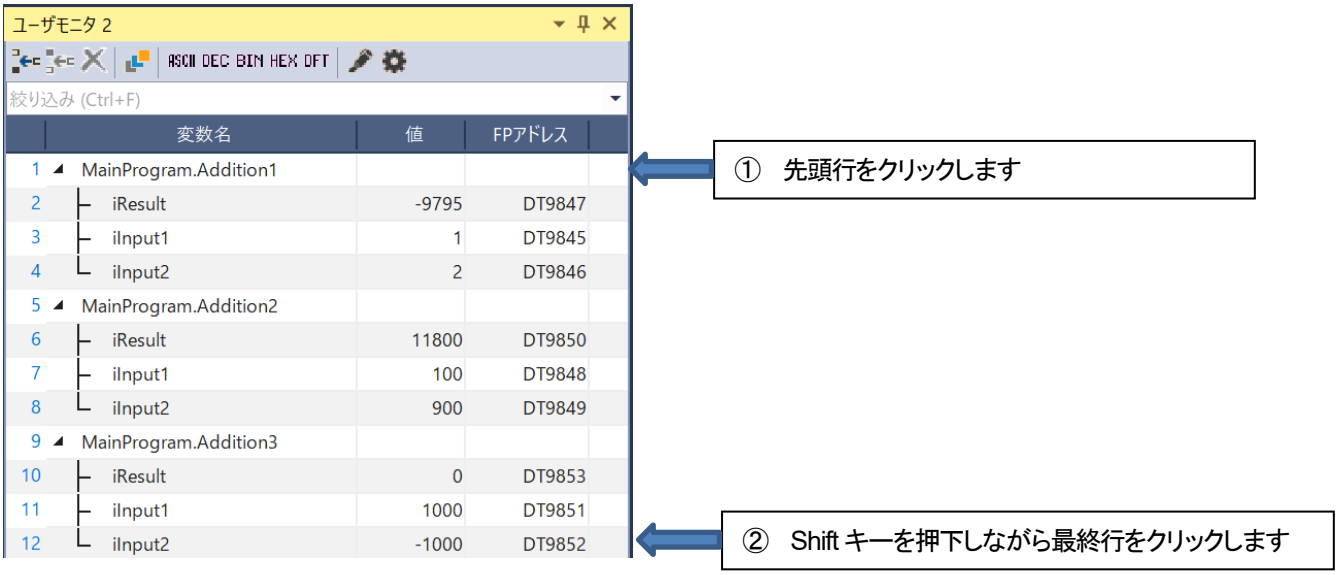


STRING 型変数の値の変更ダイアログ



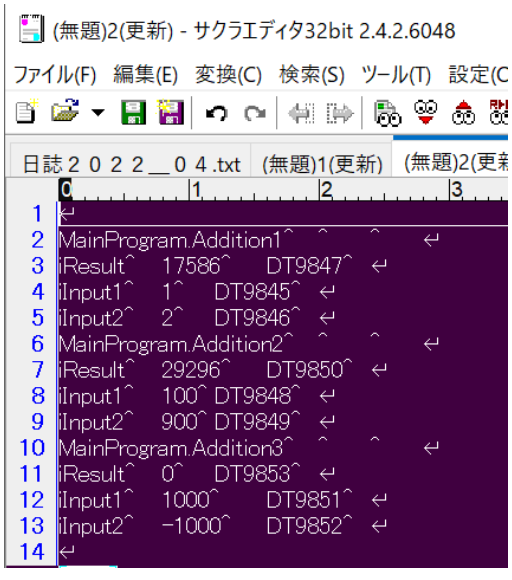
ユーザモニタ画面でのモニタデータの取得と保存　便利な機能

下図は、ユーザモニタ画面にファンクションブロック Addition のインスタンス Addition1 と Addition2 と Addition3 を登録してモニターしている状態です。これらのモニター結果をテスト結果のエビデンスとして残したい場合やモニター結果をエディタなどで詳細に確認する場合、全体あるいは部分を選択しコピーすることによりテキストエディタやエクセルなどに張り付けて確認・整理・記録することができます。



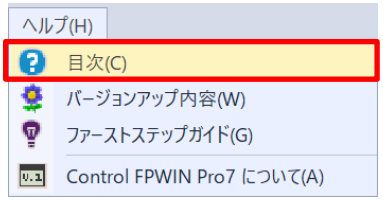
全体が選択できました。選択状態で[Ctrl]＋[c]でコピーするか、右クリック→コンテキストメニュー→コピーでコピーしてください。

右図は、コピーした結果をテキストエディタに張り付けた結果です
数千文字の文字変数も残すことができます。



PRO7（V7.7.0.0）ヘルプの仕様変更について

Control FPWIN Pro V.7.7.0.0 (2023-06 リリース)より、新しいヘルプのプラットフォームが追加されました。
Pro7 のヘルプメニューの目次を選択すると、インターネット接続されている場合は、以下の web サイトがブラウザで表示します。
インターネットに接続していない場合は、ローカルコンピュータのインストール先からブラウザでオフラインヘルプを表示します。。
これらの選択は、ツールソフトウェアが自動的に行います。
インストールしたユーザインターフェィスの言語によって、利用できる言語が変わります。



オンライン時
https://infohub.industry.panasonic.eu/documentation/fpwin/ja/#/topic/t-0000014302
オフライン時
C:¥Program Files (x86)¥Panasonic-ID SUNX Control¥Control FPWIN Pro 7¥OfflineHelp¥src¥data¥fpwin¥ja

オンラインヘルプを使用することで、常に最新の情報を参照できます。

ヘルプ内のサンプルプログラム利用

Control FPWIN Pro7 のヘルプには、各命令の説明にサンプルプログラムが掲載されています
命令の動作を確認してみたい時に、これらのサンプルを簡単にコピーして、Pro7 のプロジェクトに貼り付け確かめることができます。

ここでは、MOD 命令のの例をコピー&ペーストして、動作確認してみましょう。
下図は、MOD 命令のヘルプに記載されている例です。
下図赤枠部のボタンをクリックすると、それぞれの場所のデータがクリップボードにコピーされ
対応する編集領域に貼り付けるとそれらのデータが使用できるようになります。

・POU ヘッダのデータ → POU ヘッダ部へ貼り付け
・LD ボディのデータ → POU の LD 編集領域へ貼り付け
・ST ボディのデータ → POU の ST 編集領域へ貼り付け

（注）上記のデータを異なる領域に貼り付けることは可能ですが、意味のない事ですので間違わない様にしてください。

例

POUヘッダ

POUヘッダには、このプログラムで使用するすべての入力変数と出力変数を宣言します。 POUヘッダは全プログラム言語で使用できます。

	クラス	変数名	データ型	初期値	コメント
0	VAR	dividend	INT	11	
1	VAR	divisor	INT	4	
2	VAR	remainder	INT	0	11 divided by 4 = 2 with remainder of 3 3 is written into output variable

LDボディ

この例では、変数を使用しています。入力変数に定数を使用することもできます。被除数(11)が、除数(4)で除算されます。除算の余り(3)が、出力変数 **remainder**に書き込まれます。

STボディ

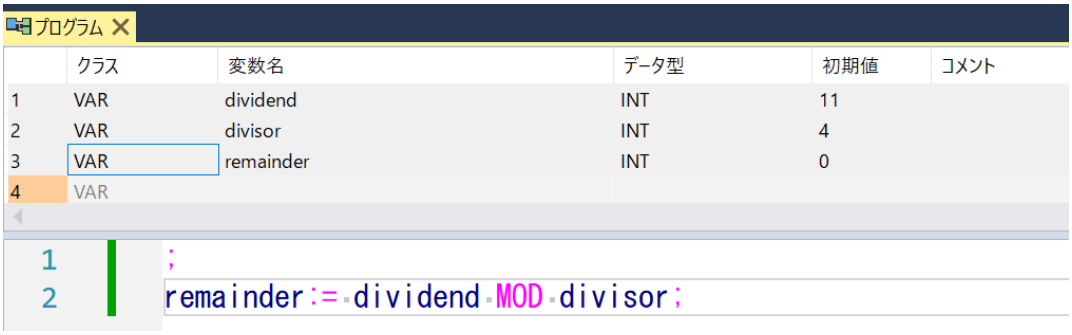
```
remainder:= dividend MOD divisor;
```

■コピーと貼り付けの実行と動作確認

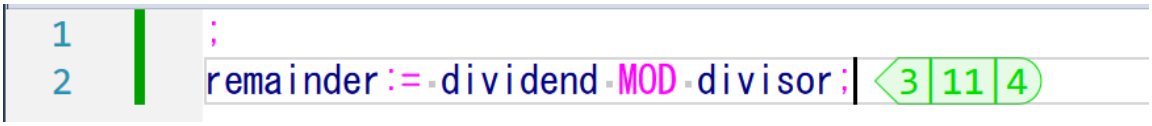
ST の POU に上図の POU ヘッダと ST ボディを貼り付けて実行してみましょう

・POU ヘッダのデータ → POU ヘッダ部へ貼り付け。（右クリック → 貼り付け、または CTRL + V）
・ST ボディのデータ → POU の ST 編集領域へ貼り付け（同上）

下図は、貼り付けた結果です。これをシミュレータにダウンロードして実行してみてください。



■ダウンロード&実行結果



11 mod 4 は、 11 / 4 の余りの計算なので、結果が 3 になっています。

（ご注意）

・サンプルプログラムで使用されている変数名などは、接頭語の指定がない場合があります。

・誤植などにより、サンプルプログラムのコンパイルエラーが発生する場合は、デバッグが必要になる場合があります。

システム変数・システム定数

■システム変数とは

PLC のシステムリレー（特殊リレー）、システムデータ（特殊データレジスタ）、各高機能ユニット用の制御リレー（X or Y）を変数名で表したものです。Control FPWIN Pro7 のシステムで決まっている変数名です。

■システム変数の名前

システムリレー（特殊リレー）は、sys_b に続いて意味のある複数の単語（大文字小文字）で構成されています。

システムデータ（特殊データ）は、sys_i、 sys_ui、 sys_w 等のデータの型で始まり、意味のある複数の単語（大文字小文字）で構成されています。

■システム変数のメリット

変数宣言の必要がなく、全ユーザで名前が共通化されています。

変数名なので、ユニット配置によって番号が変動する制御フラグも変更なしに使用できます。

名称なので、読めば意味がわかります。入力時に入力補助があるので、途中まで記憶していたら使用できます。

■システム変数の種類（カテゴリ）と内容の確認方法

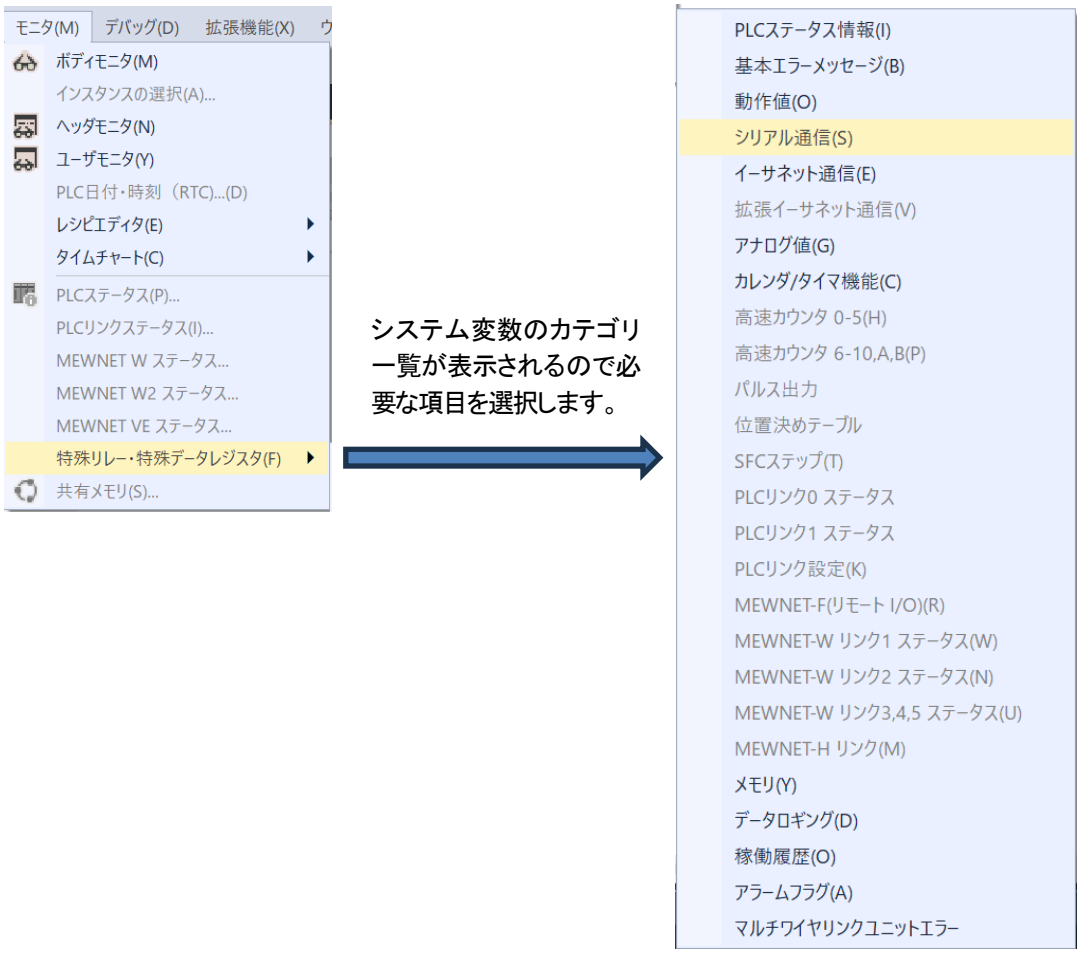
ツールのモニタ機能を使用して確認することができます。

ツールソフトウェアのモニタ機能

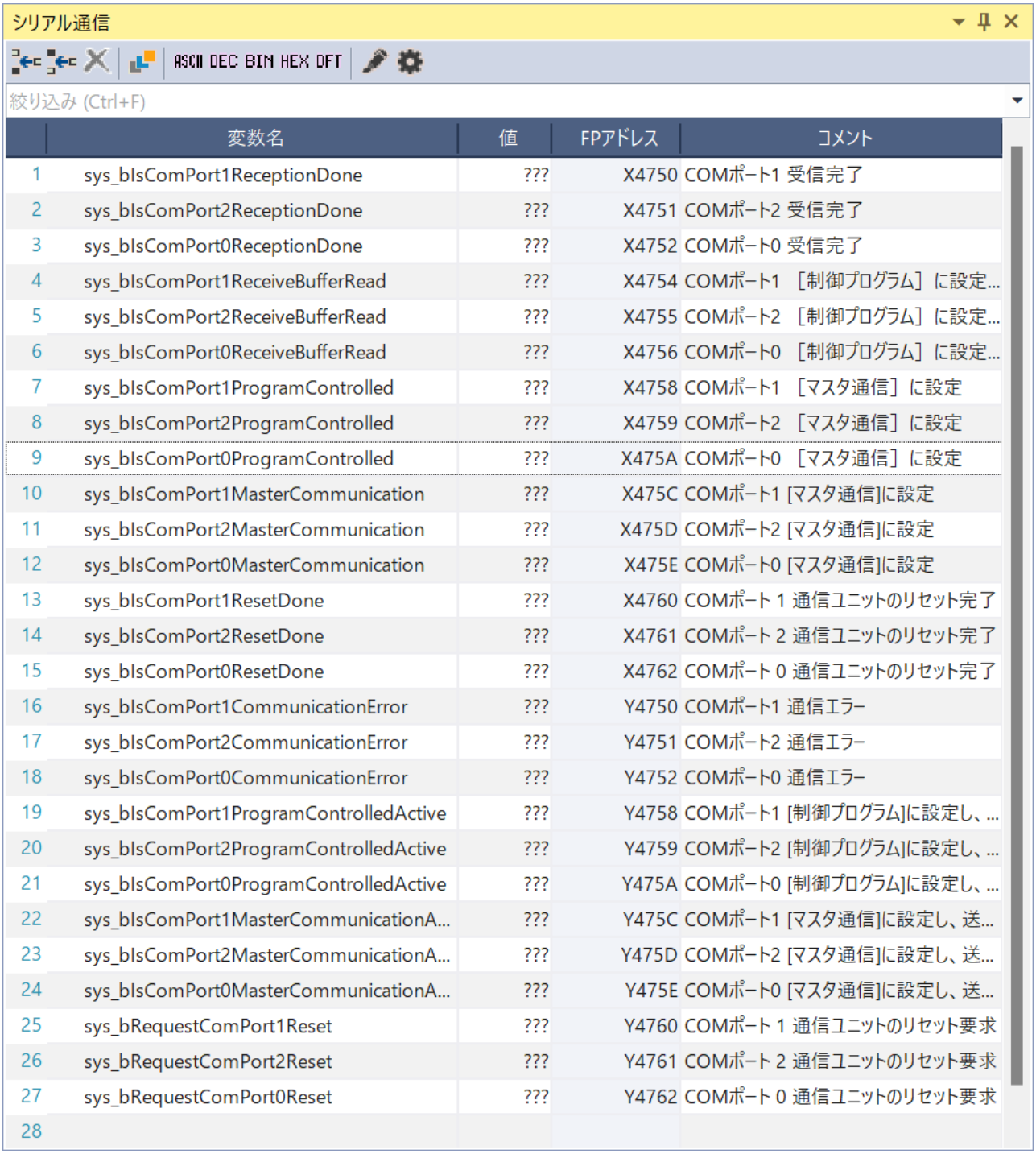
以下の操作手順で、カテゴリを選択し、ユーザモニタ画面にカテゴリ内のシステム変数一覧を表示できます。

[モニタメニュー]→[特殊リレー・特殊データレジスタ]→カテゴリー覧の表示

オンライン中は、自動的にモニタ動作が始まり、オフライン中でも開いて内容を確認することができます。



シリアル通信のカテゴリを選択した場合のユーザモニタ画面
左から順に[変数名]、[値]、[FP アドレス]、[コメント]が表示されます。各項目をクリックすると表示が昇順や降順にソートされて表示されます。例えば、FP アドレスを番号順に表示したい場合は、[FP アドレス]の項目をクリックしてみてください。
変数名称がすべて表示されない場合など、各項目間の縦線をドラッグして列の幅を調整することができます。
(注)この画面は、オフラインで開いているので、値が???表示になっています。



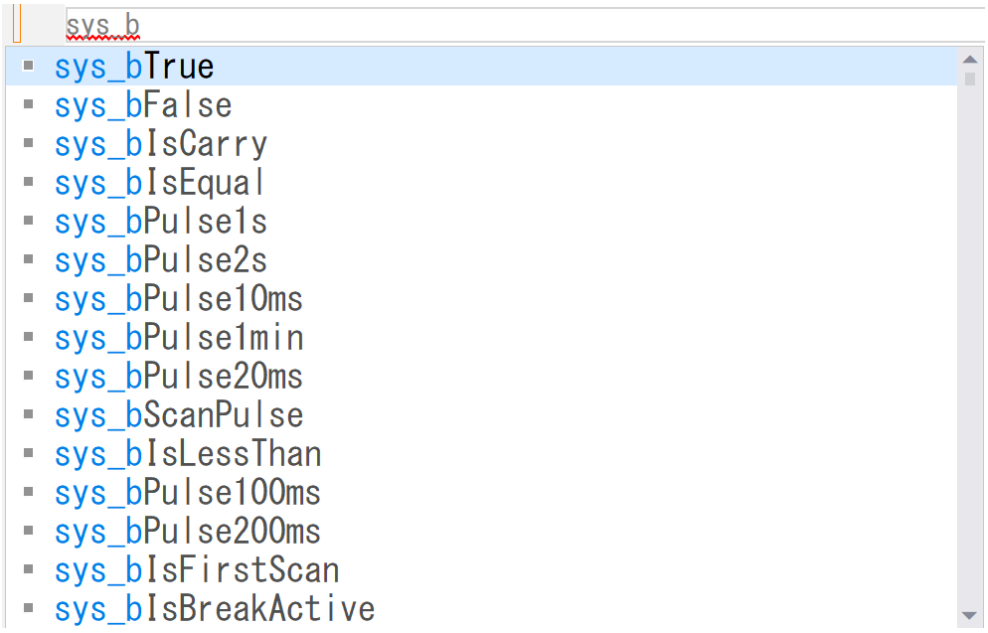
	変数名	値	FPアドレス	コメント
1	sys_bIsComPort1ReceptionDone	???	X4750	COMポート1 受信完了
2	sys_bIsComPort2ReceptionDone	???	X4751	COMポート2 受信完了
3	sys_bIsComPort0ReceptionDone	???	X4752	COMポート0 受信完了
4	sys_bIsComPort1ReceiveBufferRead	???	X4754	COMポート1 ［制御プログラム］に設定...
5	sys_bIsComPort2ReceiveBufferRead	???	X4755	COMポート2 ［制御プログラム］に設定...
6	sys_bIsComPort0ReceiveBufferRead	???	X4756	COMポート0 ［制御プログラム］に設定...
7	sys_bIsComPort1ProgramControlled	???	X4758	COMポート1 ［マスタ通信］に設定
8	sys_bIsComPort2ProgramControlled	???	X4759	COMポート2 ［マスタ通信］に設定
9	sys_bIsComPort0ProgramControlled	???	X475A	COMポート0 ［マスタ通信］に設定
10	sys_bIsComPort1MasterCommunication	???	X475C	COMポート1 [マスタ通信]に設定
11	sys_bIsComPort2MasterCommunication	???	X475D	COMポート2 [マスタ通信]に設定
12	sys_bIsComPort0MasterCommunication	???	X475E	COMポート0 [マスタ通信]に設定
13	sys_bIsComPort1ResetDone	???	X4760	COMポート 1 通信ユニットのリセット完了
14	sys_bIsComPort2ResetDone	???	X4761	COMポート 2 通信ユニットのリセット完了
15	sys_bIsComPort0ResetDone	???	X4762	COMポート 0 通信ユニットのリセット完了
16	sys_bIsComPort1CommunicationError	???	Y4750	COMポート1 通信エラー
17	sys_bIsComPort2CommunicationError	???	Y4751	COMポート2 通信エラー
18	sys_bIsComPort0CommunicationError	???	Y4752	COMポート0 通信エラー
19	sys_bIsComPort1ProgramControlledActive	???	Y4758	COMポート1 [制御プログラム]に設定し、...
20	sys_bIsComPort2ProgramControlledActive	???	Y4759	COMポート2 [制御プログラム]に設定し、...
21	sys_bIsComPort0ProgramControlledActive	???	Y475A	COMポート0 [制御プログラム]に設定し、...
22	sys_bIsComPort1MasterCommunicationA...	???	Y475C	COMポート1 [マスタ通信]に設定し、送...
23	sys_bIsComPort2MasterCommunicationA...	???	Y475D	COMポート2 [マスタ通信]に設定し、送...
24	sys_bIsComPort0MasterCommunicationA...	???	Y475E	COMポート0 [マスタ通信]に設定し、送...
25	sys_bRequestComPort1Reset	???	Y4760	COMポート 1 通信ユニットのリセット要求
26	sys_bRequestComPort2Reset	???	Y4761	COMポート 2 通信ユニットのリセット要求
27	sys_bRequestComPort0Reset	???	Y4762	COMポート 0 通信ユニットのリセット要求
28				

■プログラミング時のシステム変数入力補助

以下の2つの方法があります

① オートコンプリート機能を利用する方法

名前をはっきり覚えていなくても、途中まで入力すれば、その文字列で始まる変数名の一覧が表示されて選択入力することができます。下図では、sys_b と入力しています。



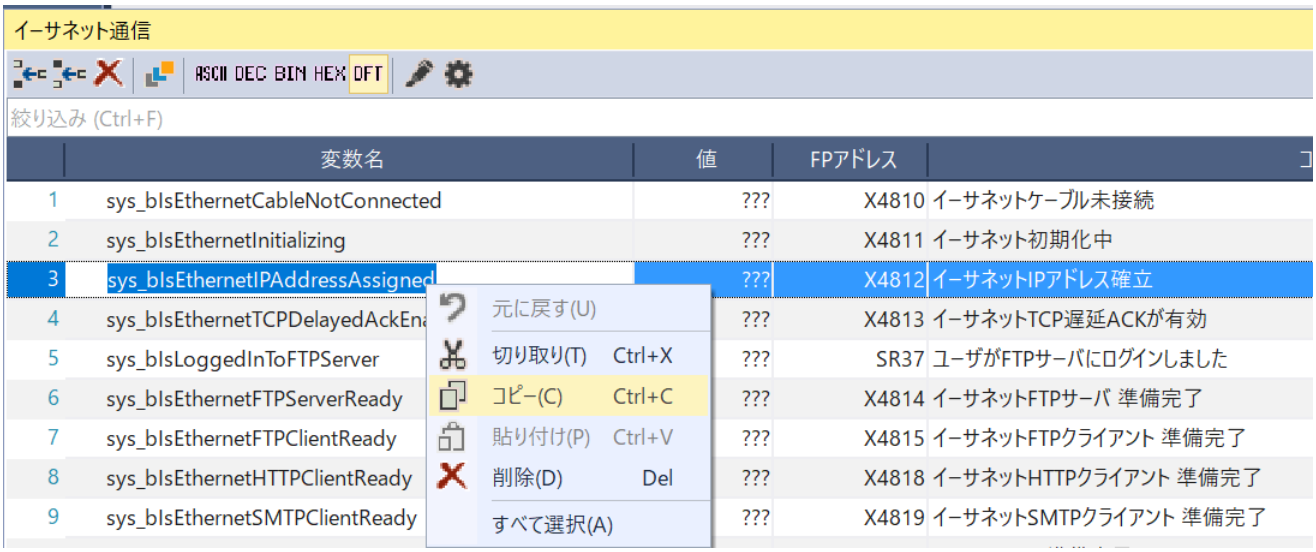
② モニタ機能を利用する方法

リレー番号を確認して入力したい場合、モニタ機能を使用して確認しモニタ画面で変数をコピーしてプログラム編集画面に貼り付けることができます。

下図は、イーサネット通信カテゴリのシステム変数モニタ画面です。

イーサネットIPアドレス確立リレーをダブルクリックして選択後、右クリックしてコピーし、編集画面に貼り付けることができます。

（CTRL+C → CTRL+V でコピーペーストすることもできます。）



■システム定数とは、

システム定数は、Pro7 の命令の入力等に指定する為の数値データであり、名前を付けて覚えやすいようにしてあります。

PLC オリジナルの命令と同じ機能のシステム命令であっても、Pro7 の命令として入力などに指定するパラメータの値は少し変わっていることがあるので注意が必要です。 PLC のオリジナル命令の入力値から推測して、数値そのものを入力すると

エラーになったり、期待通りに動作しないことがあります。

■システム定数の名前

SYS_で始まるすべて大文字の複数の単語で構成された文字列です。(具体例:SYS_COM1_PORT)

基本的に命令入力で指定するパラメータの定数で、具体的な名前がツール内で定義されています。

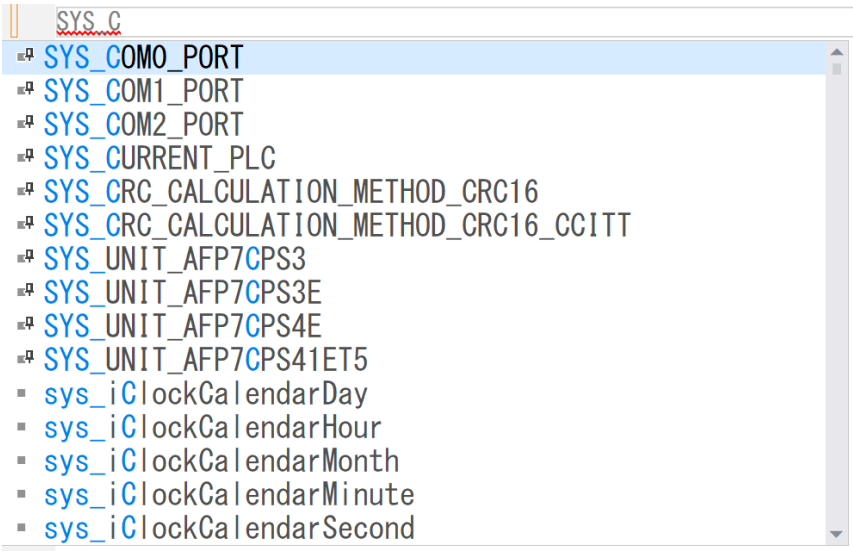
個別命令の説明中に指定可能なシステム変数が説明されています。

■システム変数のメリット

意味のある単語で構成されているので、プログラムが見やすくなります。

■プログラミング時のシステム定数入力補助

システム定数の先頭の数文字を入力すると、その候補リストが表示され選択することができます。



■システム定数の具体例

BCC 計算命令の計算パターン指定
SYS_BCC_CALCULATION_METHOD_ADD
SYS_BCC_CALCULATION_METHOD_SUB
SYS_BCC_CALCULATION_METHOD_XOR
SYS_BCC_CALCULATION_METHOD_CRC16
SYS_CRC_CALCULATION_METHOD_CRC16
SYS_CRC_CALCULATION_METHOD_CRC16_CCITT

シリアル通信命令での COM ポート指定
SYS_TOOL_PORT
SYS_COM0_PORT
SYS_COM1_PORT
SYS_COM2_PORT
SYS_COM3_PORT
SYS_COM4_PORT

MODBUS 通信命令でのコマンド指定
SYS_MODBUS_01_READ_COIL
SYS_MODBUS_02_READ_INPUT
SYS_MODBUS_03_READ_HOLDING_REGISTERS
SYS_MODBUS_04_READ_INPUT_REGISTERS
SYS_MODBUS_05_FORCE_COIL
SYS_MODBUS_06_PRESET_REGISTER
SYS_MODBUS_15_FORCE_COILS
SYS_MODBUS_16_PRESET_REGISTERS

イーサネット通信関連命令でのコネクション番号指定
SYS_ETHERNET_USER_CONNECTION_1
SYS_ETHERNET_USER_CONNECTION_2
SYS_ETHERNET_USER_CONNECTION_3
……
SYS_ETHERNET_USER_CONNECTION_16

プログラムの記述

TIME 型のデータの中身

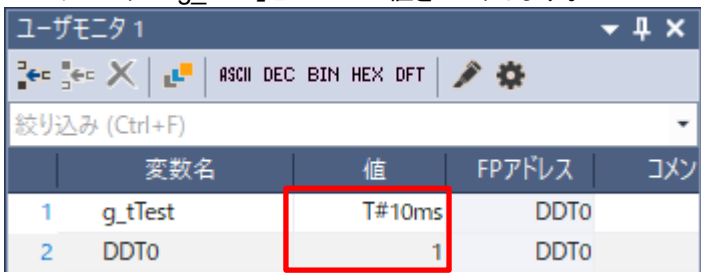
FPWIN Pro7 では TIME 型は 10ms 単位のため、10ms で割った値がデータレジスタに格納されます。

■グローバル変数

	クラス	変数名	FPアドレス	IECアドレス	データ型	初期値
1	VAR_GLOBAL	g_tTest	DDT0	%MD5.0	TIME	T#10ms

●ユーザモニタ

ユーザモニタで「g_tTest」と DDT0 の値をモニタします。



TIME 型では T#10ms のとき、DDT0 は DINT 型として 1 が格納されています。

FPWINPro7 では TIME 型は 10ms 単位のため、10ms で除した値がデータレジスタに格納されます。

g_tTest の値が T#5s のとき、DDT0 には 500 が格納されています。

5,000ms÷10ms＝500 が格納されています。

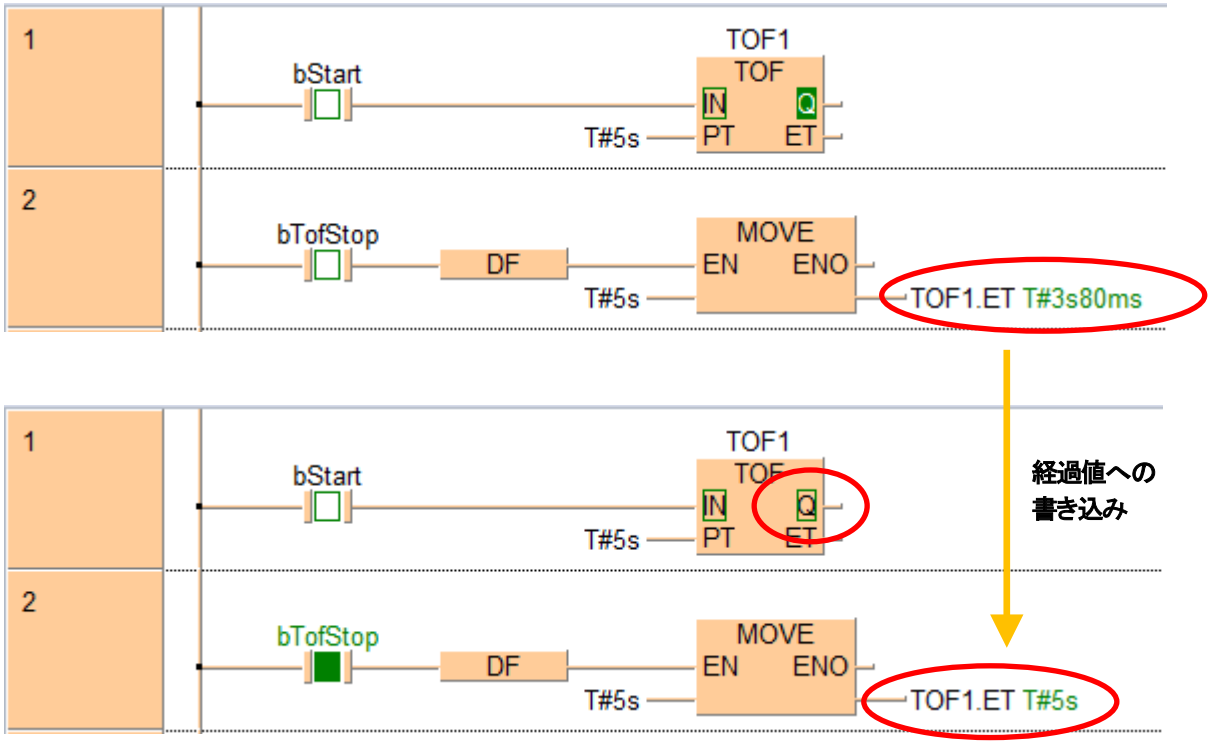
TOF の強制停止

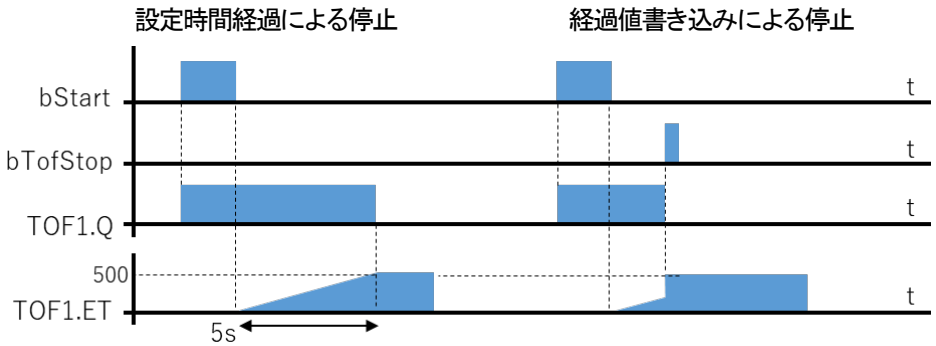
TOF の出力信号を、設定時間の経過前に強制的に OFF することができます。
ここでは、経過値:ET へ設定時間を書き込む例で紹介します。

■POU ヘッダ(Prog)

	クラス	変数名	データ型	初期値
1	VAR	bStart	BOOL	FALSE
2	VAR	bTofStop	BOOL	FALSE
3	VAR	TOF1	TOF	

■POU ボディ(Prog)





ユニットメモリのインデックスについて

ユニットメモリのスロット番号とメモリアドレスは、インデックス修飾を使用して指定をすることができます。
ここでは例として、FP7 位置決めユニットの指定したスロット番号＋軸番号の現在値を読み出すプログラムを使用します。

■POU ヘッダ(Prog)

	クラス	変数名	データ型	初期値
1	VAR	diIndexValueDIX	DINT	0
2	VAR	diIndexValueDIY	DINT	0
3	VAR	diSlotNo	DINT	1
4	VAR	diAxisNo	DINT	1
5	VAR	diElapsedValue	DINT	0

■POU ボディ(Prog)

```
diIndexValueDIX := diSlotNo - 1;
diIndexValueDIY := (diAxisNo - 1) * 16#0040;

DIX := diIndexValueDIX;
DIY := diIndexValueDIY;

diElapsedValue := DIXS1:DIYDUM0043C;
```

上記プログラムの例では、初期値として DIX に 0、DIY に 0 を指定することで、
スロット番号 1 の UM0043C(1 軸現在値データ)のユニットメモリを指定することができます。

※参考:1 軸現在値のユニットメモリアドレス

UM 0043C -UM 0043D	1 軸の現在値	機械原点を基準とする現在値が、パルス単位で格納されます。原点復帰完了時には、"0"にリセットされます。現在値更新機能を実行した場合も、値は更新されません。 単位:Pulse

また、DIX に 2(diSlitNo＝3)、DIY に 3(diAxisNo＝4)を指定することで、
スロット番号 3 の UM004FC(4 軸現在値データ)のユニットメモリを指定することができます。

グローバル変数の内部メモリとしての使い方

グリーバル変数に FP アドレスを登録しなければ、各 POU でグローバルで共通の内部メモリとして使用できます。
ここでは、2 つの POU:Test1－Test2 の両方で使用する BOOL 型変数の例として紹介します。

■グローバル変数

	クラス	変数名	FPアドレス	IECアドレス	データ型	初期値
1	VAR_GLOBAL	g_bInternalMemory			BOOL	FALSE

FP アドレスを登録しない

■POU ヘッダ（Prog:Test1）

	クラス	変数名	データ型	初期値
1	VAR_EXTERNAL	g_bInternalMemory	BOOL	FALSE

■POU ボディ（Prog:Test1）

■POU ヘッダ（Prog:Test2）

	クラス	変数名	データ型	初期値
1	VAR_EXTERNAL	g_bInternalMemory	BOOL	FALSE
2	VAR	bOutput	BOOL	FALSE

■POU ボディ（Prog:Test2）

POU:Test1 での”g_bInternalMemory”の状態が反映されます。

定数変数(VAR_CONSTANT)を使用したプログラム

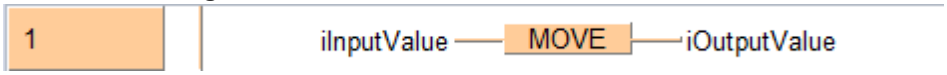
定数変数 VAR_CONSTANT を使用すると DT 等のレジスタに割り付けられることなく、直接定数を指定できます。
ただし、定数変数はメモリを取得しないため、転送先に VAR_CONSTANT を指定するとエラーになります。

ここでは FP0H を用い、保持型に設定されたエリア(DT)の例を用いて紹介します。

■POU ヘッダ(Prog)

	クラス	変数名	データ型	初期値
1	VAR_CONSTANT	iInputValue	INT	1234
2	VAR	iOutputValue	INT	0

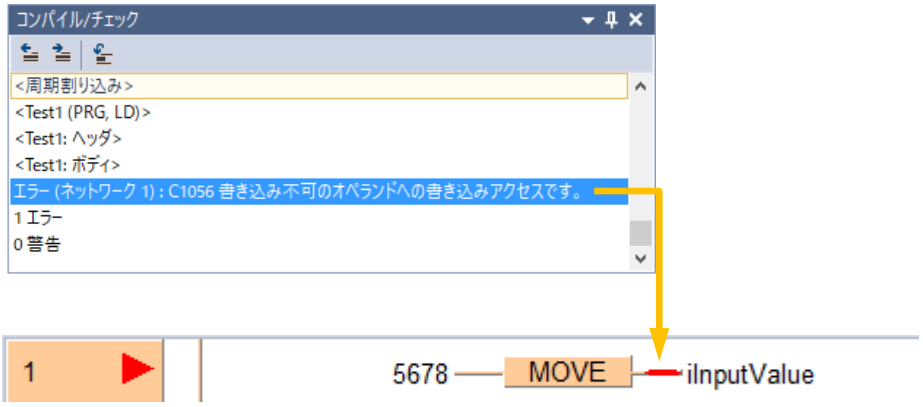
■POU ボディ(Prog)



■コンパイル後のプログラム(FPWIN GR7 ラダー表記)

定数が直接指定されていることが分かります。

注意！！
定数変数 VAR_CONSTANT を出力先に指定するとコンパイル時に以下のエラーが発生します。

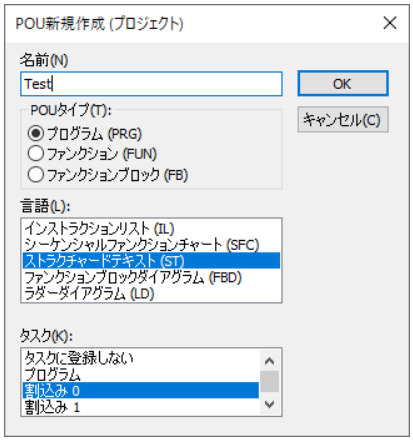


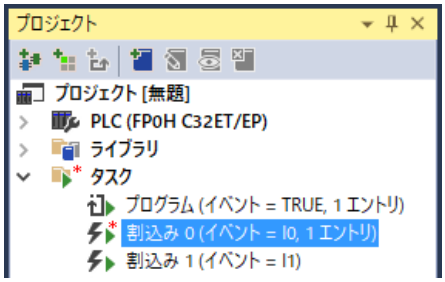
割り込みプログラムの使用方法(FP7 以外の場合)

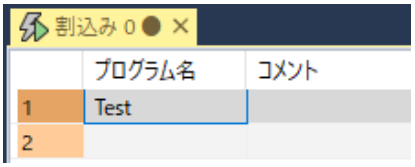
割り込みタスクに POU を登録することで、割り込み制御命令をプログラムコード上に自動的に記述します。
ここでは、FP0H の「割込み入力:X0」を使用して「割込み 0」にプログラムを記述する例を紹介します。

「プロジェクトペイン」ー「システムレジスタ」ー「高速カウンタ,パルス出力,割込み入力,パルスキャッチ入力」から
「割り込み入力:X0ー>割込み 0」を「立上がりエッジ」に設定します。

「POU 新規作成(プロジェクト)」より、「タスク(K)」の「割込み 0」を選択することで
「割込み 0」のタスクに POU を登録することができます。





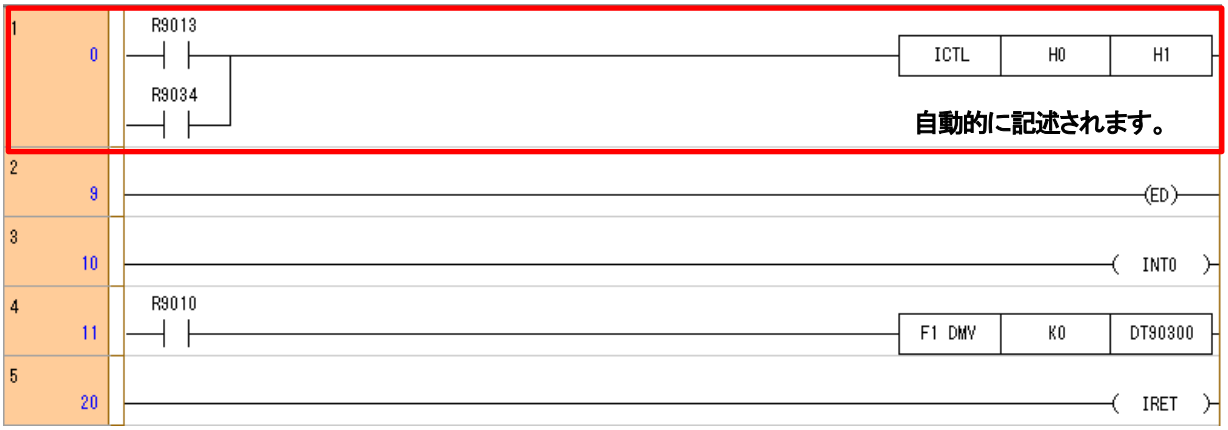


ここで登録した POU 内に記述したプログラムが割り込みプログラムとして動作します。

■POU ボディ(Prog:Test)

```
sys_diHscChannel0ElapsedValue := 0;
```

■コンパイル後のプログラム(FPWIN GR7 ラダー表記)

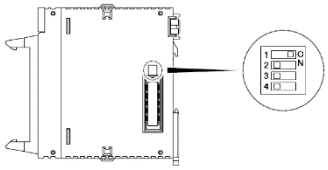


自動的に記述する ICTL 命令は、割り込みタスクに 1 つでも POU を登録した時点で、全割り込みの許可設定になります。
タイミングにより、許可/禁止を行いたい場合は、別途 ICTL 命令の記述が必要です。

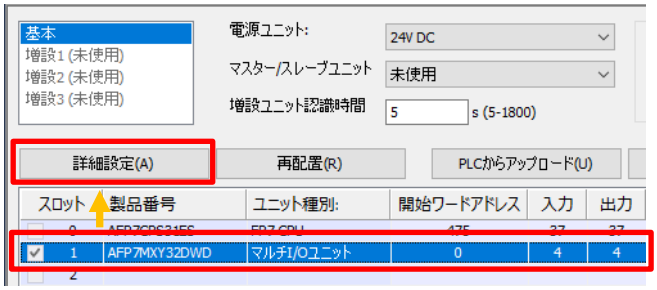
割り込みプログラムの使用方法（FP7 の場合）

割り込みタスクに POU を登録することで、割り込み制御命令をプログラムコード上に自動的に記述します。
FP7 ではマルチ入出力ユニットを装着することで、割り込み入力信号またはカウンタ比較一致フラグにて
割込みプログラムを動かすことができます。

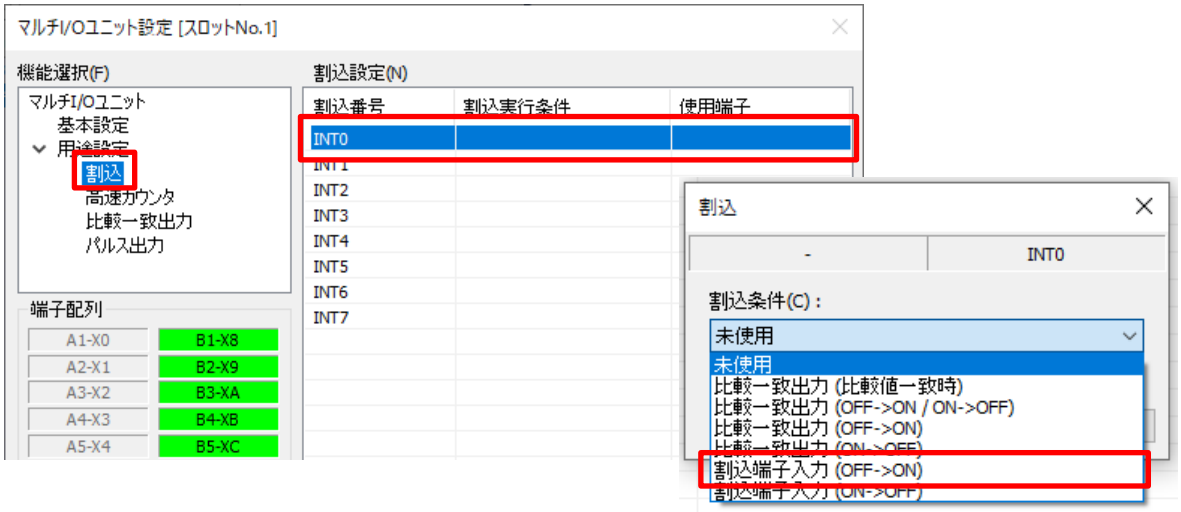
●注意
マルチ入出力ユニットで割り込み機能を使用するには右図の様に
ユニット側面のスイッチ（No.1）を ON にする必要があります。



プロジェクトペインの「IO 割付」からマルチ I/O ユニットを登録し、詳細設定を選択します。



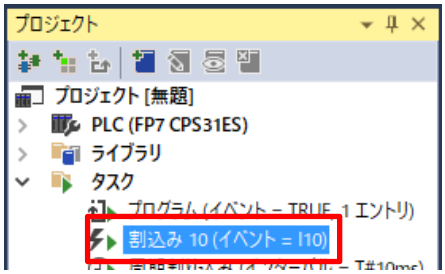
「マルチ I/O ユニット設定」画面より「割込」を選択し、割込入力を割り付ける番号をダブルクリックします。
割り込み条件を選択します。ここでは「割込端子入力（OFF－＞ON）」を選択します。



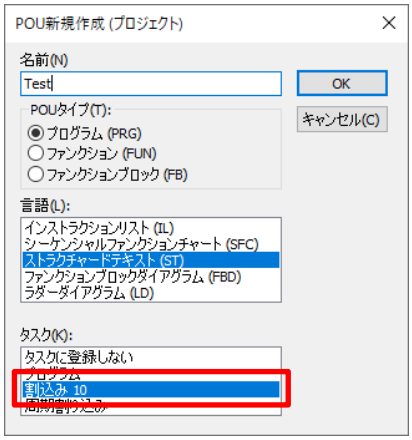
割込設定画面に割込実行条件が登録されます。

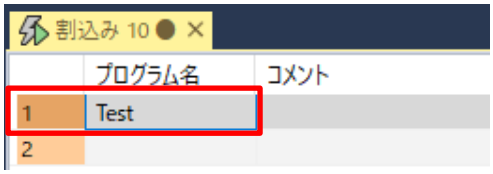
※「使用端子」は「IO 割付」の「開始ワードアドレス」と、「マルチ I/O ユニット」設定内の「割込番号」によって変わります。

割込実行条件を登録後、プロジェクトをコンパイルすると、タスクに割り込みタスク「割込み 10」が登録されます。



「POU 新規作成(プロジェクト)」より、「タスク(K)」の「割込み 10」を選択することで
「割込み 10」のタスクに POU を割り当てることができます。





ここで登録した POU 内に記述したプログラムが
割り込みプログラムとして動作します。

■グローバル変数

	クラス	変数名	FPアドレス	IECアドレス	データ型	初期値
1	VAR_GLOBAL	g_diCountValue	S1:DUM00110	%MD20.1.272	DINT	0

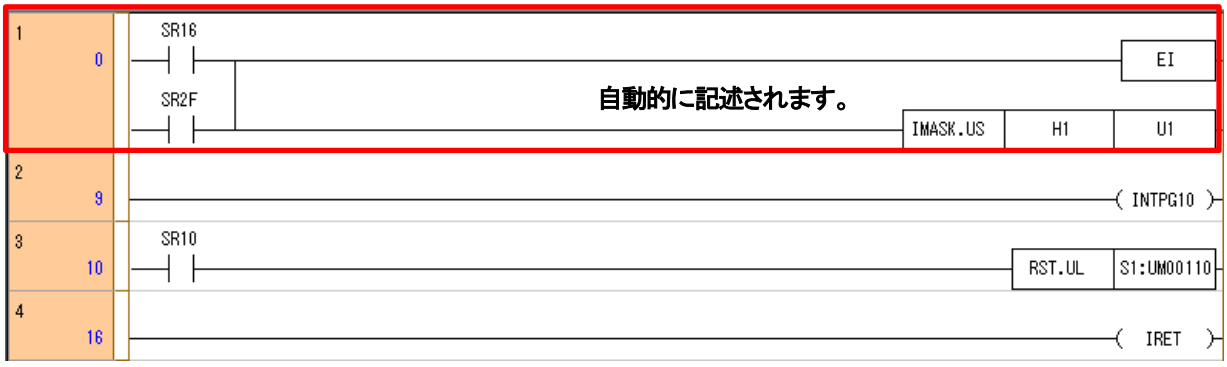
■POUヘッダ(Prog:Test)

	クラス	変数名	データ型	初期値
1	VAR_EXTERNAL	g_diCountValue	DINT	0

■POU ボディ(Prog:Test)

```
g_diCountValue := 0 ;
```

■コンパイル後のプログラム(FPWIN GR7 ラダー表記)



自動的に記述する割り込み制御命令では、登録した割り込み設定はすべて許可設定になります。
タイミングにより、許可/禁止を行いたい場合は、別途「FP_INTERRUPT_DISABLE 命令」や
「FP_INTERRUPT_ACTIVATE 命令」の記述が必要です。

数字の桁区切り方法

数字の桁数が大きい場合、プログラム上で桁数区切りを入れることができます。
また、モニタ上では自動で 3 桁ごとに区切りが入ります。

■POU ヘッダ(Prog)

	クラス	変数名	データ型	初期値
1	VAR	diOutput	DINT	0

■POU ボディ(Prog)

桁区切りは 3 桁ごとに"_"を入力することを推奨します。

```
diOutput := 123_456_789;    123_456_789
```

●補足

```
diOutput := 123456789;    123_456_789
```

プログラム上で桁区切りを入力しなくても、モニタ上は自動で桁区切りが入ります。

定数に 2 進数指定が可能

FPWIN Pro7 では命令の定数に 2 進数指定が可能です。
ここでは、”MOVE”命令を用いて 2 進数の定数を WY0 に出力する例を用いて紹介します。

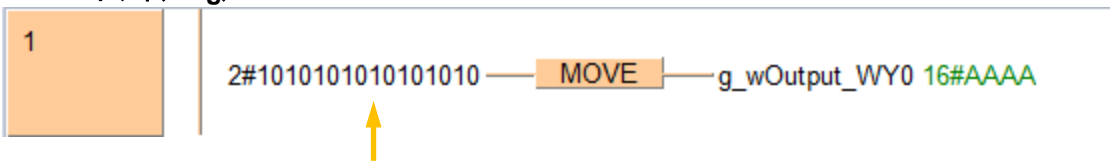
■グローバル変数

	クラス	変数名	FPアドレス	IECアドレス	データ型	初期値
1	VAR_GLOBAL	g_wOutput_WY0	WY0	%QW0	WORD	0

■POU ヘッダ(Prog)

	クラス	変数名	データ型	初期値
1	VAR_EXTERNAL	g_wOutput_WY0	WORD	0

■POU ボディ(Prog)



定数を 2 進数で指定することが可能

■コンパイル後のプログラム(FPWIN GR7 ラダー表記)

当然、コンパイル後のプログラムは 16 進数での指定に変更されます。
2 進数→16 進数の返還を FPWIN Pro7 が自動的に行います。

■POU ボディ(ST の場合)

```
g_wOutput_WY0 := 2#1010101010101010;    16#AAAA
```

WORD 型から BOOL 型(16 ビット)要素への変換

WX0、WY0、WR0 のように 1 ワード(16 点)の情報を 16bit の配列変換を用いて bit 単位で取り込みます。
ここでは、グローバル変数に登録した WX0 の 1 ワードの情報を bit 単位で変数に取り込む例で紹介します。

■グローバル変数

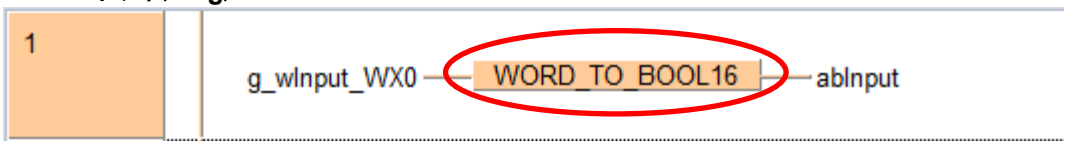
	クラス	変数名	FPアドレス	IECアドレス	データ型	初期値
1	VAR_GLOBAL	g_wInput_WX0	WX0	%IW0	WORD	0

■POU ヘッダ(Prog)

	クラス	変数名	データ型	初期値
1	VAR_EXTERNAL	g_wInput_WX0	WORD	0
2	VAR	abInput	ARRAY [0..15] OF BOOL	[16(FALSE)]

“0～15”の 16 ビットの配列変数を宣言します。

■POU ボディ(Prog)



“WORD_TO_BOOL”ライブラリを用いて 1 ワードデータを BOOL 型 16 ビットに変換します。

16 ビット要素の指定方法

abInput[0]　**配列変数名の後に角括弧で目的のビット位置を 0～15 で指定します。**

■コンパイル後のプログラム(FPWIN GR7 ラダー表記)

配列変数(ARRAY)の配列指定方法

配列変数は INT 型、UINT 型、DINT 型、UDINT 型変数を用いることで間接的に指定することができます。
ここでは、INT 型変数を使用して入力毎に配列変数 0～9 まで順にデータを繰り返し書き込む例で紹介します。

■POU ヘッダ(Prog)

	クラス	変数名	データ型	初期値
1	VAR	aiDataTable	ARRAY [0..9] OF INT	[10(0)]
2	VAR	bStart	BOOL	FALSE
3	VAR	iElementNo	INT	0
4	VAR	iInputElement	INT	0

10 個の連続した配列変数を指定

■POU ボディ(Prog)

```
IF (DF(bStart)) THEN
  aiDataTable[iElementNo] := iInputElement;
  iElementNo := iElementNo + 1;
  IF (iElementNo = 10) THEN
      iElementNo := 0;
  END_IF;
END_IF;
```

配列変数名[INT 型変数]

配列変数名[整数データ型変数]の指定をすることで間接的に配列変数を指定することができます。

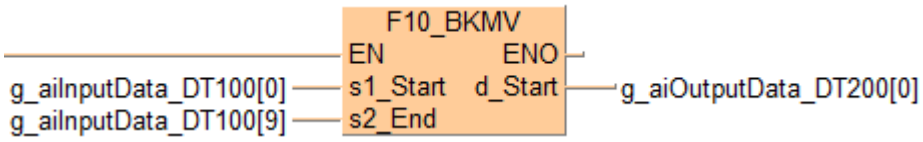
F10(BKMV)拡張命令

FPWIN Pro7 には本来の F10(BKMV)命令に拡張性を持たせた命令があります。
拡張性を持たせた命令は 3 つあります。

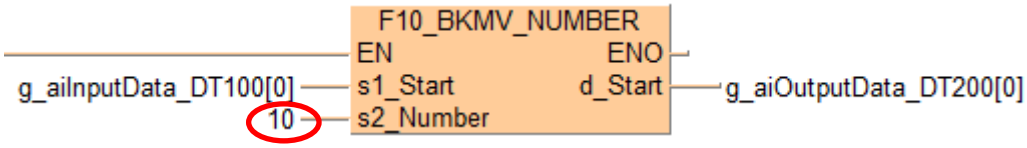
従来、F10(BKMV)命令は「DT100～DT109(10 ワード分)を、DT200 を先頭に転送する」といった仕様です。

■POU ヘッダ(Prog)

	クラス	変数名	FPアドレス	IECアドレス	データ型	初期値
1	VAR_GLOBAL	g_aiInputData_DT100	DT100	%MW5....	ARRAY [0..9] OF INT	[10(0)]
2	VAR_GLOBAL	g_aiOutputData_DT200	DT200	%MW5....	ARRAY [0..9] OF INT	[10(0)]

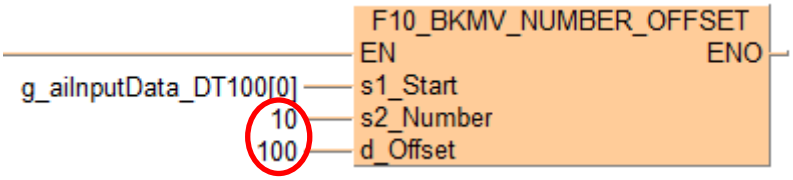


■拡張命令①F10_BKMV_NUMBER



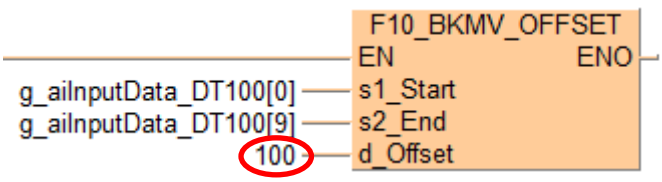
「DT100 から 10 ワード分を、DT200 を先頭に転送する」といったことができます。
F10_BKMV_NUMBER を使用することで、
「どこから何ワード分、どこを先頭に」といった転送ができるようになっています。

■拡張命令②F10_BKMV_NUMBER_OFFSET



「DT100 から 10 ワード分を、DT100 から 100 オフセットした先頭に転送する」といったことができます。
F10_BKMV_NUMBER_OFFSET を使用することで、
「どこから何ワード分、指定元アドレスからオフセットした先頭に」といった転送ができるようになっています。

■拡張命令③F10_BKMV_OFFSET



「DT100 から DT109 を、DT100 から 100 オフセットしたを先頭に転送する」といったことができます。
F10_BKMV_OFFSET を使用することで、
「どこからどこまでを、指定元アドレスからオフセットした先頭に」といった転送ができるようになっています。

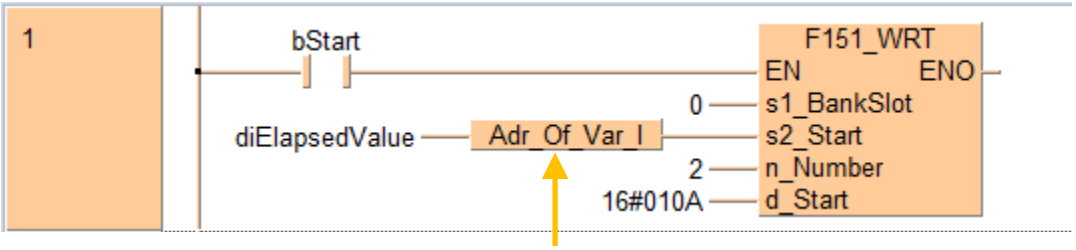
変数の先頭アドレスを取得する必要のある F 命令使用方法

F151 命令等、デバイスの先頭アドレスを必要とする命令に、2 ワード変数を割り付けたい場合、
“Adr_Of_Var”を使用することで可能になります。
ここでは、FP0H 位置決めユニットの CH0 経過値を変更する例で紹介します。

■POU ヘッダ(Prog)

	クラス	変数名	データ型	初期値
1	VAR	bStart	BOOL	FALSE
2	VAR	diElapsedValue	DINT	0

■POU ボディ(Prog)

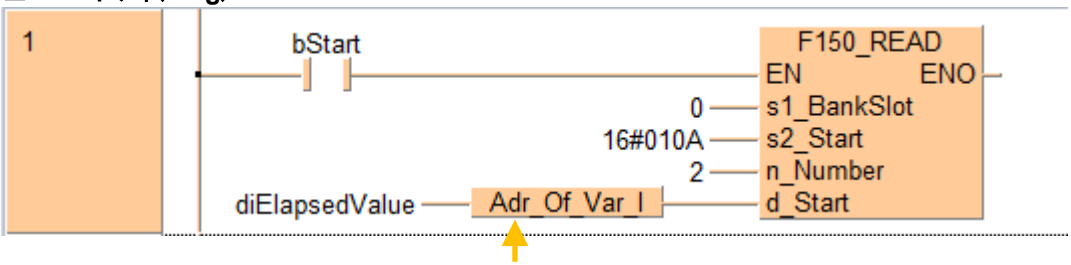


FP TOOL ライブラリ“Adr_of_Var_I”を用いて、2 ワード変数を指定することにより可能になります。
上図例では F151(WRT)命令を例に紹介しましたが、F150(READ)命令を使用する場合も”Adr_of_Var_I”を使用します。
イメージ的に”Adr_of_Var_O”を使用しがちなので注意が必要です。

■POU ヘッダ(Prog)

	クラス	変数名	データ型	初期値
1	VAR	bStart	BOOL	FALSE
2	VAR	diElapsedValue	DINT	0

■POU ボディ(Prog)



F150(READ)命令でも”Adr_of_Var_I”を使用します。

●注意
“Adr_Of_Var”は FP シリーズ命令と対で使用する必要があります。

●備考(ST 言語による記述例)

```
IF (DF(bStart)) THEN
   F151_WRT(s1_BankSlot := 0,
            s2_Start := AdrLast_Of_Var(diElapsedValue),
            n_Number := 2,
            d_Start := 16#010A);
END_IF;
```

ST 言語においては、”Adr_Of_Var_I”と”Adr_Of_Var_O”の区別はなく、”Adr_Of_Var”のみとなります。

DWORD 型変数の上位・下位 WORD を WORD 型変数へ転送する①

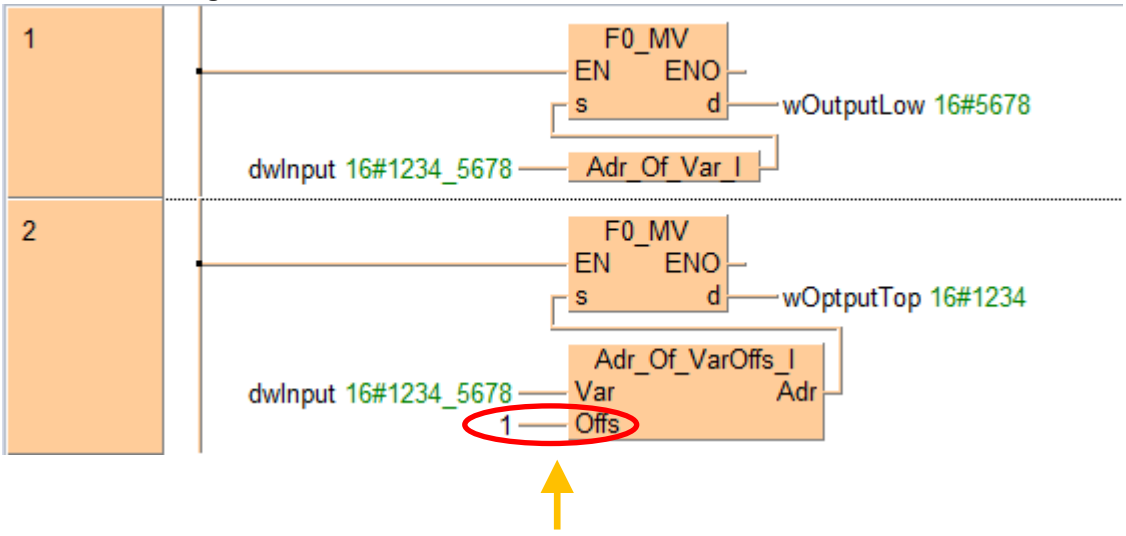
DWORD 型変数の上位もしくは下位を指定して WORD 型変数へ転送することができます。

ここでは、Adr_of_Var_I を使用した例で紹介します。

■POU ヘッダ(Prog)

	クラス	変数名	データ型	初期値
1	VAR	dwInput	DWORD	16#12345678
2	VAR	wOutputLow	WORD	0
3	VAR	wOptputTop	WORD	0

■POU ボディ(Prog)



上位の指定には"Adr_Of_VarOffs_I"を使用します。
Offs＝1 とすることにより、下位アドレスの"1 つ上"、
つまり上位アドレスを指定することができます。

●備考(ST 言語による記述例)

```
F0_MV(s := Adr_Of_Var(dwInput),d => wOptputTop);
F0_MV(s := Adr_Of_VarOffs(Var := dwInput,Offs := 1),d => wOutputLow);
```

※ST 言語においては、"Adr_Of_Var_I"と"Adr_Of_Var_O"の区別はなく、"Adr_Of_Var"のみとなります。

●注意
"Adr_Of_Var"、"Adr_Of_VarOffs"は FP シリーズ命令と対で使用する必要があります。

DWORD 型変数の上位･下位 WORD を WORD 型変数へ転送する②

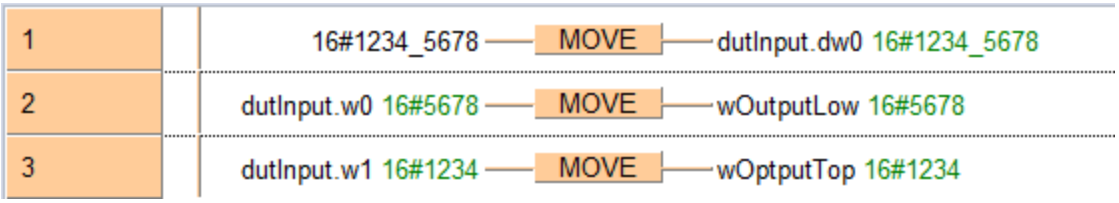
WORD 型変数を DWORD 型変数の上位もしくは下位を指定して転送することができます。

ここでは、DWORD_OVERLAPPING_DUT を使用した例を紹介します。

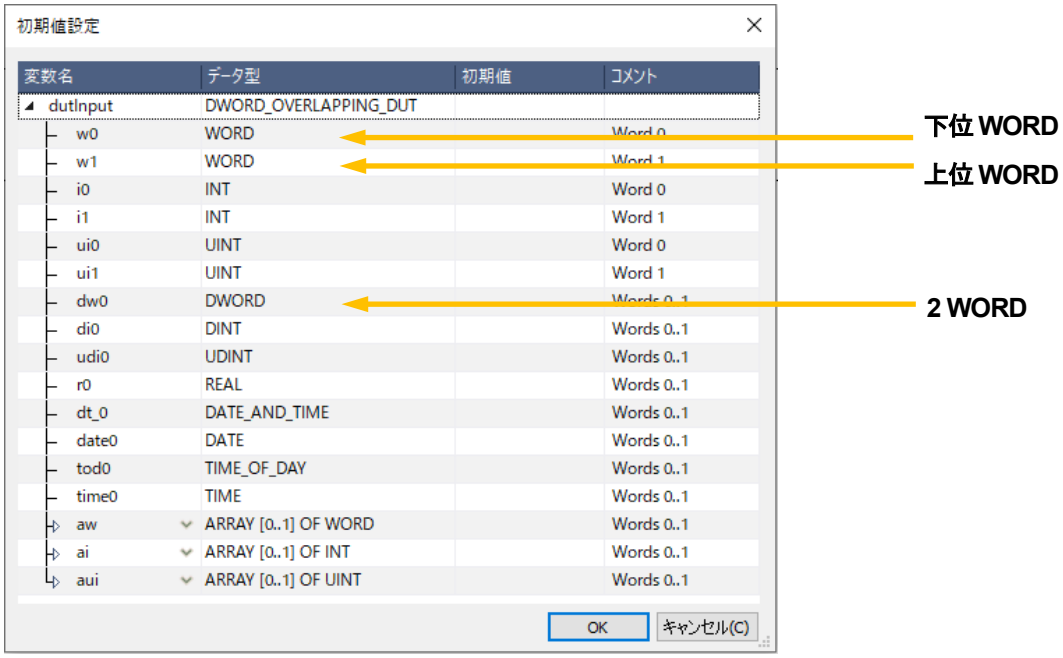
■POU ヘッダ(Prog)

	クラス	変数名	データ型	初期値
1	VAR	dutInput	DWORD_OVERLAPPING_DUT	
2	VAR	wOutputLow	WORD	0
3	VAR	wOptputTop	WORD	0

■POU ボディ(Prog)



■DWORD_OVERLAPPING_DUT



“DWORD_OVERLAPPING_DUT”を使用することにより、
2WORD 変数を 1WORD ずつの型に分けたり、配列に置き換えたりすることができる為、
2WORD データを 1 ワードずつ個別に指定したい場合に便利です。

●備考(ST 言語による記述例)

```
dutInput.dw0 := 16#1234_5678;
dutInput.w0 := wOutputLow;
dutInput.w1 := wOptputTop;
```

WORD 型変数を DWORD 型変数の上位・下位 WORD へ転送する①

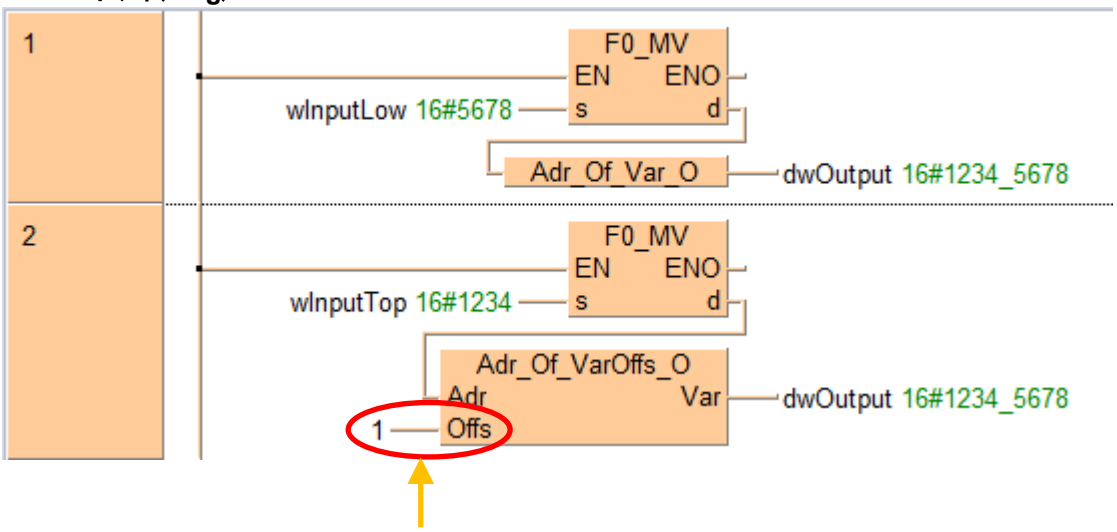
WORD 型変数を DWORD 型変数の上位もしくは下位を指定して転送することができます。

ここでは、Adr_of_Var_O を使用した例で紹介します。

■POU ヘッダ(Prog)

	クラス	変数名	データ型	初期値
1	VAR	dwOutput	DWORD	0
2	VAR	wInputLow	WORD	16#5678
3	VAR	wInputTop	WORD	16#1234

■POU ボディ(Prog)



上位の指定には”Adr_Of_VarOffs_O”を使用します。
Offs＝1 とすることにより、下位アドレスの”1 つ上”、
つまり上位アドレスを指定することができます。

●備考(ST 言語による記述例)

```
FO_MV(s := wInputLow,d => Adr_Of_Var(dwOutput));
FO_MV(s := wInputTop,d => Adr_Of_VarOffs(Var := dwOutput,Offs := 1));
```

※ST 言語においては、”Adr_Of_Var_I”と”Adr_Of_Var_O”の区別はなく、”Adr_Of_Var”のみとなります。

●注意
“Adr_Of_Var”、“Adr_Of_VarOffs”は FP シリーズ命令と対で使用する必要があります。

WORD 型変数を DWORD 型変数の上位･下位 WORD へ転送する②

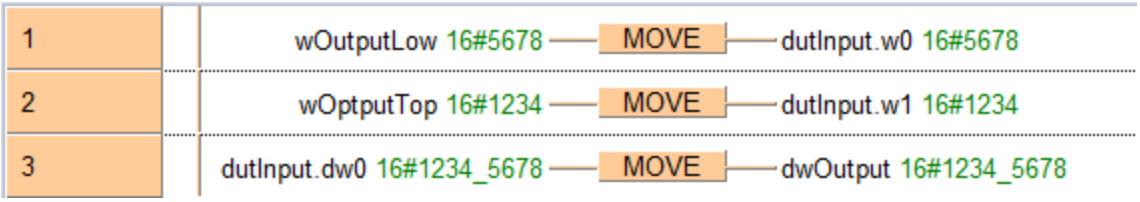
WORD 型変数を DWORD 型変数の上位もしくは下位を指定して転送することができます。

ここでは、DWORD_OVERLAPPING_DUT を使用した例で紹介します。

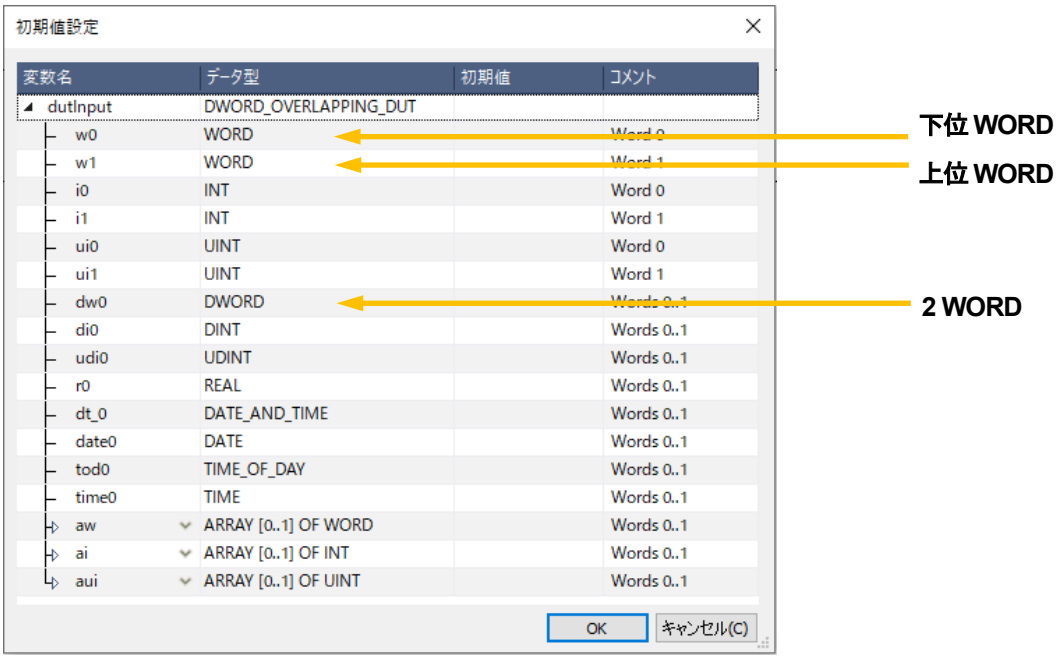
■POU ヘッダ(Prog)

	クラス	変数名	データ型	初期値
1	VAR	dutInput	DWORD_OVERLAPPING_DUT	
2	VAR	wOutputLow	WORD	16#5678
3	VAR	wOptputTop	WORD	16#1234
4	VAR	dwOutput	DWORD	0

■POU ボディ(Prog)



■DWORD_OVERLAPPING_DUT



“DWORD_OVERLAPPING_DUT”を使用することにより、
2WORD 変数を 1WORD ずつの型に分けたり、配列に置き換えたりすることができる為、
2WORD データを 1 ワードずつ個別に指定したい場合に便利です。

●備考(ST 言語による記述例)

```
dutInput.w0 := wInputLow;
dutInput.w1 := wInputTop;
dwOutput := dutInput.dw0;
```

AdrDT_Of_Offs の使用例

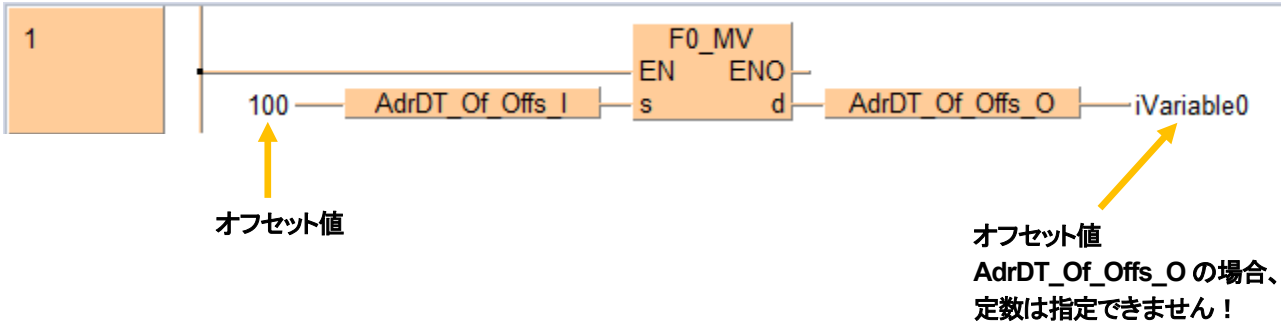
AdrDT_Of_Offs_I で指定した DT から Fun 命令に値を入力、AdrDT_Of_Offs_O で出力ができます。
このファンクションは FP シリーズ命令と対で使用する必要があります。

ここでは、F0（MV）命令を使用した例で紹介します。

■POU ヘッダ（Prog）

	クラス	変数名	データ型	初期値
1	VAR	iVariable0	INT	0

■POU ボディ（Prog）



●プログラム動作

iVariable0 が”200”の場合、

DT100 の値を DT200 に転送するという動作になります。

●備考（ST 言語による記述例）

```
F0_MV(AdrDT_Of_Offs(100),AdrDT_Of_Offs(iVariable0));
```

※ST 言語においては、”AdrDT_Of_Offs_I”と”AdrDT_Of_Offs_O”の区別はなく、“AdrDT_Of_Offs”のみとなります。

AdrLast_Of_Var_I の使用例

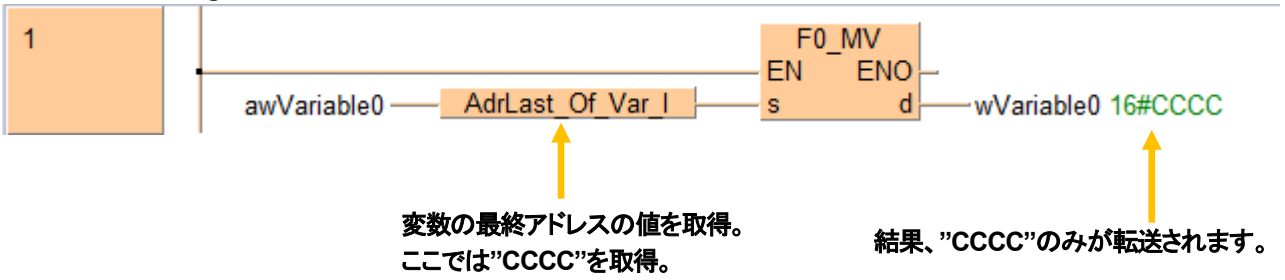
AdrLast_Of_Var_I を使用することで、変数の最終アドレスを F 命令に指定することができます。
このファンクションは F 命令と対で使用する必要があります。

ここでは、F0(MV)命令を使用した例で紹介します。

■POU ヘッダ(Prog)

	クラス	変数名	データ型	初期値
1	VAR	awVariable0	ARRAY [0..2] OF WORD	[16#AAAA,16#BBBB,16#CCCC]
2	VAR	wVariable0	WORD	0

■POU ボディ(Prog)



●備考(ST 言語による記述例)

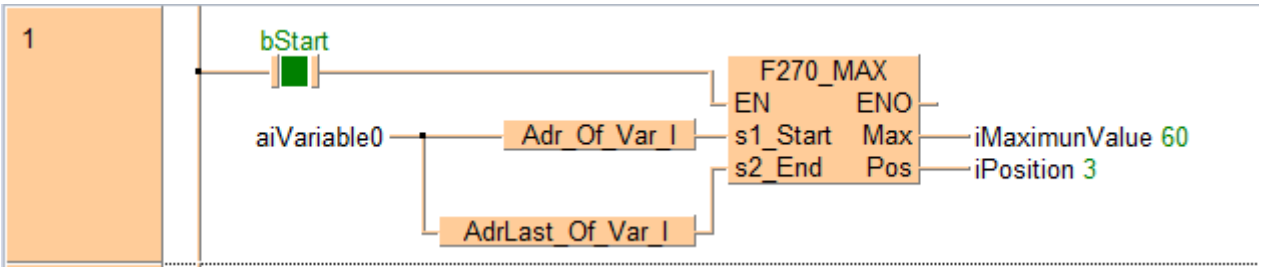
```
F0_MV(AdrLast_Of_Var(awVariable0),wVariable0);
```

※ST 言語においては、"AdrLast_Of_Var_I"と"AdrLast_Of_Var_O"の区別はなく、"AdrLast_Of_Var"のみとなります。

●サンプル例
■POU ヘッダ(Prog)

	クラス	変数名	データ型	初期値
1	VAR	bStart	BOOL	FALSE
2	VAR	aiVariable0	ARRAY [0..5] OF INT	[10,20,30,60,50,40]
3	VAR	iMaximunValue	INT	0
4	VAR	iPosition	INT	0

■POU ボディ(Prog)



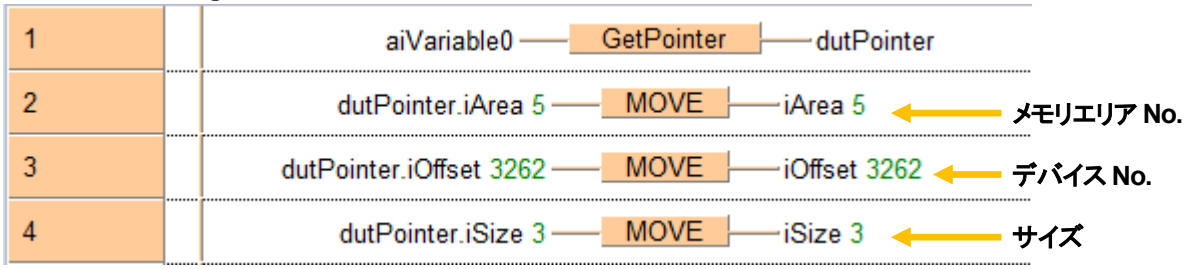
GetPointer の使用例

GetPointer を使用することによって、変数が PLC デバイスのどこに割付いているのかが分かります。

■POU ヘッダ(Prog)

	クラス	変数名	データ型	初期値
1	VAR	dutPointer	POINTER	
2	VAR	aiVariable0	ARRAY [0..2] OF INT	[3(0)]
3	VAR	iArea	INT	0
4	VAR	iOffset	INT	0
5	VAR	iSize	INT	0

■POU ボディ(Prog)

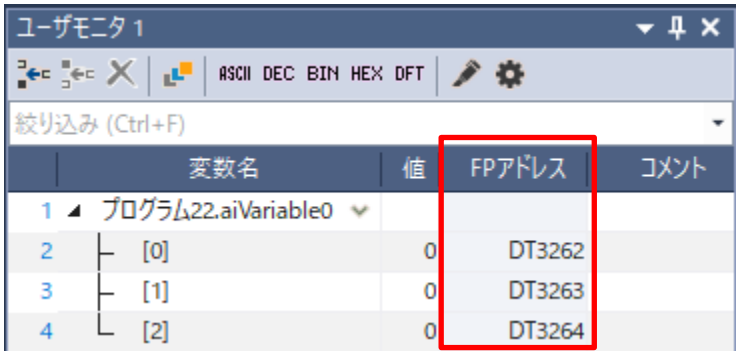


●使用可能メモリエリア

メモリエリア	変数名	メモリエリア No.
フラグ	SYS_MEMORY_AREA_WR	0
データレジスタ	SYS_MEMORY_AREA_DT	5
ファイルレジスタ	SYS_MEMORY_AREA_FL	9
出力フラグ	SYS_MEMORY_AREA_Y	10,200

上図例では INT 型に 3 ワードの配列変数 aiVariable0 が、DT3262 から 3 ワードに格納されているということが分かります。

●参考:FP アドレス割当確認(ユーザモニタより)



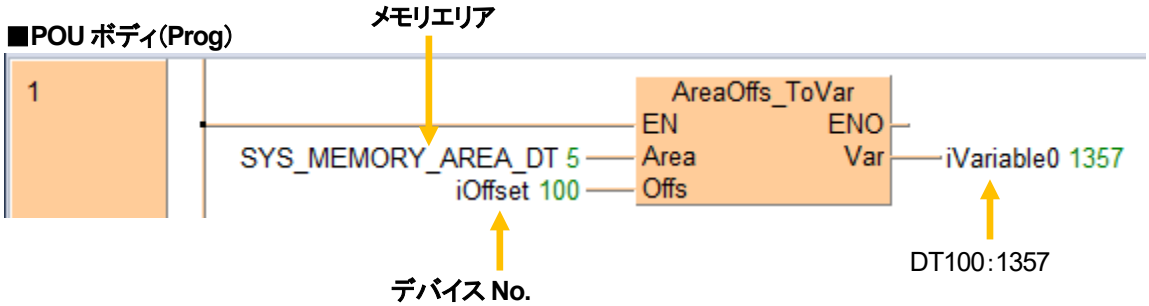
AreaOffs_ToVar の使用例

AreaOffs_ToVar を使用することによって、指定した PLC デバイスから変数にデータを読み出すことができます。

■POU ヘッダ(Prog)

	クラス	変数名	データ型	初期値
1	VAR	iOffset	INT	0
2	VAR	iVariable0	INT	0

■POU ボディ(Prog)



●使用可能メモリエリア

メモリエリア	変数名	メモリエリア No.
フラグ	SYS_MEMORY_AREA_WR	0
データレジスタ	SYS_MEMORY_AREA_DT	5
ファイルレジスタ	SYS_MEMORY_AREA_FL	9

上図例では DT100＝1357 の値を”iVariable0”に読み出しているということになります。
iOffset の値を変更することで目的 DT エリアを iVariable0 に読み出すことができます。

●備考:ST 言語による記述例

```
iVariable0 := AreaOffs_ToVar(Area := SYS_MEMORY_AREA_DT, Offs := iOffset);
```

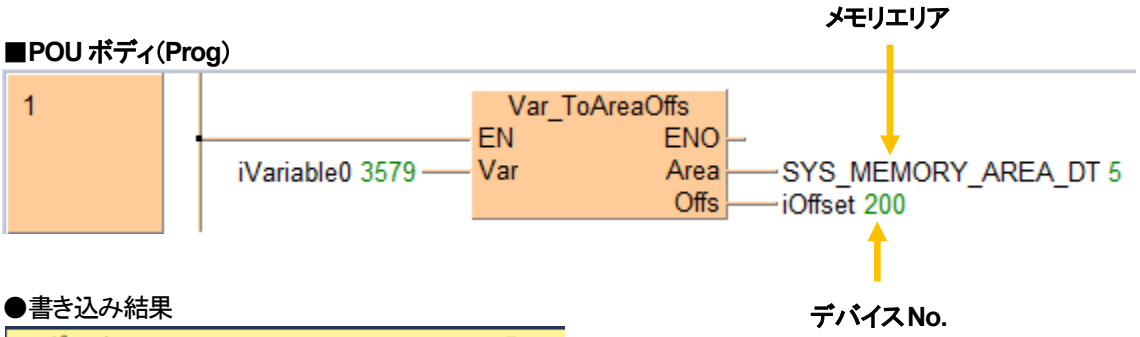
Var_ToAreaOffs の使用例

Var_ToAreaOffs を使用することによって、変数の値を指定した PLC デバイス(DT、FL)に書き込むことができます。

■POU ヘッダ(Prog)

	クラス	変数名	データ型	初期値
1	VAR	iOffset	INT	0
2	VAR	iVariable0	INT	0

■POU ボディ(Prog)



●書き込み結果

●使用可能メモリエリア

メモリエリア	変数名	メモリエリア No.
フラグ	SYS_MEMORY_AREA_WR	0
データレジスタ	SYS_MEMORY_AREA_DT	5
ファイルレジスタ	SYS_MEMORY_AREA_FL	9

上図例では、iVariable0 の値(3579)を DT200 に書き込んでいるということになります。
iOffset の値を変更することで目的の DT エリアに iVariable0 の値を転送することができます。

●備考　DT 言語による記述例

```
Var_ToAreaOffs(Var := iVariable0, Area => SYS_MEMORY_AREA_DT, Offs => iOffset);
```

Size_Of_Var の使用例

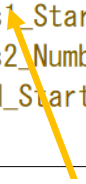
Size_Of_Var を使用することによって、変数のサイズ(何ワードなのか)を取得することができます。

■POU ヘッダ(Prog)

	クラス	変数名	データ型	初期値
1	VAR	bVariable0	BOOL	FALSE
2	VAR	diVariable0	DINT	0
3	VAR	diVariable1	DINT	0
4	VAR	iSize	INT	0

■POU ボディ(Prog)

```
IF (bVariable0) THEN
 iSize := Size_Of_Var(diVariable0);   2        0
 F10_BKMV_NUMBER(s1_Start := Adr_Of_Var(diVariable0),     0
                 s2_Number := iSize,     2
                 d_Start => Adr_Of_Var(diVariable1));    0
END_IF;
```



この場合、DINT 型なので結果は"2"(ワード)となります。

●注意
Siza_Of_Var が使用できるデータ型は INT,DINT,WORD,DWORD,REAL,STRING,TIME,DUT,ARRAY です。

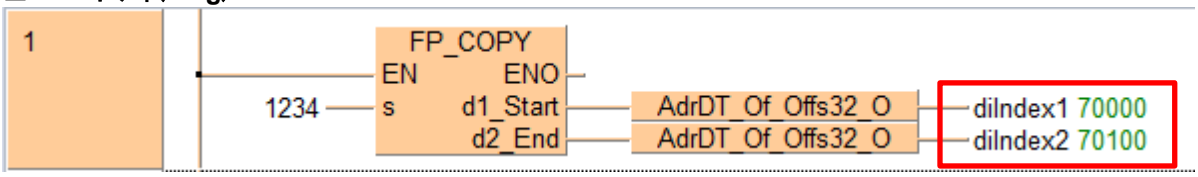
AdrDT_Of_Offs32 の使用例

AdrDT_Of_Offs32 命令はデータレジスタのアドレスを 32 ビットオフセット指定することが出来ます。
この命令を使用することで、16 ビット整数で指定できない要素番号のデータレジスタを修飾できます。

■POU ヘッダ（Prog）

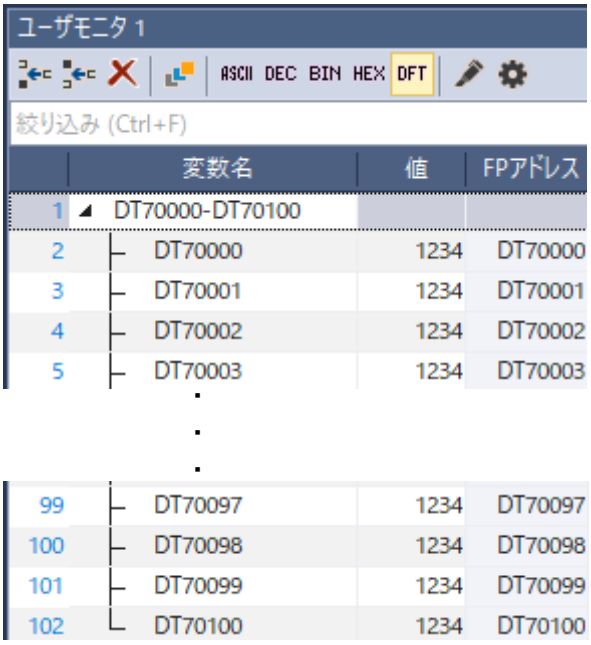
	クラス	変数名	データ型	初期値
1	VAR	dIIndex1	DINT	70000
2	VAR	dIIndex2	DINT	70100

■POU ボディ（Prog）



DT70000～DT70100 に値:1234 が転送されます。

●ユーザモニタ



FP7 I/O 割付の取り込み

FP7 において、装着されたユニットの I/O No.をスロット No.の指定だけで取り込むことにより、
汎用的なファンクションブロック(FB)およびファンクション(FUN)を作成することができます。

ここでは、任意の位置にあるアナログ入力ユニット(AD4H)の、任意チャンネルの入力値を取り込む FUN の例で紹介します。

■POU ヘッダ(FUN:AD4H_DATA_READ)

	クラス	変数名	データ型	初期値
1	VAR_INPUT	iSlotNumeber	INT	0
2	VAR_INPUT	diChannelNumber	DINT	0
3	VAR_OUTPUT	iData	INT	0
4	VAR	dutInputData	BOOL32_OVERLAPPING_DUT	
5	VAR	diWxData	DINT	0

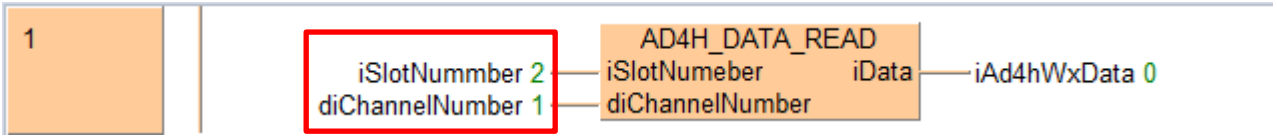
■POU ボディ(FUN:AD4H_DATA_READ)

```
FP_GET_IO_START_OFFSET(s_Slot := iSlotNumeber, d => diWxData);
dutInputData.di0 := AreaOffs32_ToVar(iArea := SYS_MEMORY_AREA_WX,
                                     diOffs := (diChannelNumber * 2) + diWxData);
iData := dutInputData.i0;
```

■POU ヘッダ(Prog)

	クラス	変数名	データ型	初期値
1	VAR	iAd4hWxData	INT	0
2	VAR	iSlotNummber	INT	0
3	VAR	diChannelNumber	DINT	0

■POU ボディ(Prog)



FUN の入力で任意のスロット No／チャンネル No を指定することで、グローバル変数登録を行わなくても
入力値の読み出しが可能です。
上記動作例では、スロット No.2 の Ch.1 の入力値を読み込んでいます。

実アドレスを使用せずに入力リレーを取り込む

実アドレスを使用せずに入力リレーを取り込むことで、汎用的なファンクションブロック、ファンクションを作成することができます。
ここでは、WX5 の 7 ビット目を指定して読み出す例で紹介します。

■POU ヘッダ(FUN:BitDataRead)

	クラス	変数名	データ型	初期値
1	VAR_INPUT	bReadStart	BOOL	FALSE
2	VAR_INPUT	iWXAddress	INT	0
3	VAR_INPUT	iBitAddress	INT	0
4	VAR	wWXData	WORD	0

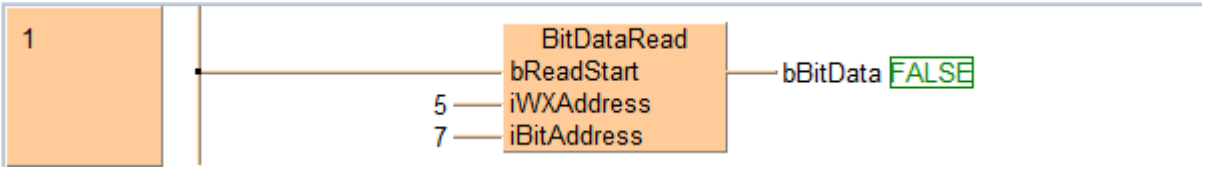
■POU ボディ(FUN:BitDataRead)

```
wWXData := AreaOffs_ToVar(Area := SYS_MEMORY_AREA_WX, Offs := iWXAddress);
IF (bReadStart) THEN
   FP_TEST_BIT(s := wWXData, n_Pos := iBitAddress, d => BitDataRead);
END_IF;
```

■POU ヘッダ(Prog)

	クラス	変数名	データ型	初期値
1	VAR	bBitData	BOOL	FALSE

■POU ボディ(Prog)



X57 の ON/OFF 信号を取り込むというファンクションになります。

例えば、FP7・FP0H 位置決めユニットのファンクションブロックを作成する場合など、
上記のように FP_TEST_BIT 命令を使用することで汎用的なファンクションブロックを作成することも可能になります。

実アドレスを使用せずに出力リレーを指定する

実アドレスを使用せずに出力を指定することで、汎用的なファンクションブロック、ファンクションを作成することができます。
ここでは、WY0 のビット 7 を ON/OFF する FB 例で紹介します。

■POU ヘッダ（FB:BitDataWrite）

	クラス	変数名	データ型	初期値
1	VAR_INPUT	bWriteStart	BOOL	FALSE
2	VAR_INPUT	iWYAddress	INT	0
3	VAR_INPUT	iBitAddress	INT	0
4	VAR	wWYdata	WORD	0

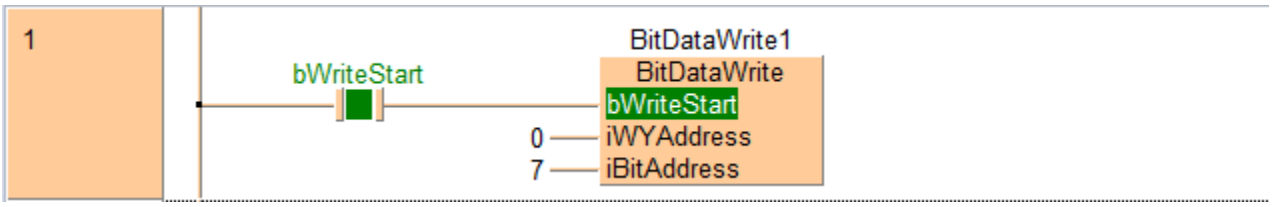
■POU ボディ（FB:BitDataWrite）

```
wWYdata := AreaOffs_ToVar(Area := SYS_MEMORY_AREA_WY, Offs := iWYAddress);
IF (DF(bWriteStart)) THEN
   FP_SET_BIT(n_Pos := iBitAddress, d := wWYdata);
END_IF;
IF (DFN(bWriteStart)) THEN
   FP_RESET_BIT(n_Pos := iBitAddress, d := wWYdata);
END_IF;
Var_ToAreaOffs(Var := wWYdata, Area => SYS_MEMORY_AREA_WY, Offs => iWYAddress);
```

■POU ヘッダ（Prog）

	クラス	変数名	データ型	初期値
1	VAR	BitDataWrite1	BitDataWrite	
2	VAR	bWriteStart	BOOL	FALSE

■POU ボディ（Prog）



“bWriteStart”の ON/OFF で Y7 が ON/OFF するというファンクションブロックになります。

●書き込み結果

例えば、FP7・FP0H 位置決めユニットのファンクションブロックを作成する場合など、
上記のように FP_SET_BIT, FP_RSET_BIT 命令を使用することで汎用的なファンクションブロックを
作成することも可能になります。

Elem_OfArray1D の使用例

Elem_OfArray1D を使用することによって、配列変数の配列数を取得することができます。

■POU ヘッダ(Prog)

	クラス	変数名	データ型	初期値	コメント
1	VAR	bVariable0	BOOL	FALSE	
2	VAR	aiVariable0	ARRAY [0..7] OF INT	[8(0)]	
3	VAR	aiVariable1	ARRAY [0..7] OF INT	[8(0)]	
4	VAR	iVariable0	INT	0	

■POU ボディ(Prog)

```
IF (DF(bVariable0)) THEN
   iVariable0 := Elem_OfArray1D(Array1D := aiVariable0);   8
   F10_BKMV_NUMBER(s1_Start := Adr_Of_Var(aiVariable0),
                   s2_Number := iVariable0,    8
                   d_Start => Adr_Of_Var(aiVariable1));
END_IF;
```

BKMV で転送するための転送データのワード数を計算しています。

●注意

Elem_OfArray1D によって得られる結果はあくまでも配列数です。

■補足

結果 =8

```
iVariable0 := Elem_OfArray1D(Array1D := aiVariable0);   8
```

INT 型の配列変数

結果 =8

```
iVariable0 := Elem_OfArray1D(Array1D := adiVariable0);   8
```

DINT 型の配列変数

※データ型が 2 ワードであっても結果は同じです。

Elem_OfArray2D の使用例

Elem_OfArray2D を使用することによって、配列変数の 1 次元および 2 次元の要素数を取得することができます。

■POU ヘッダ(Prog)

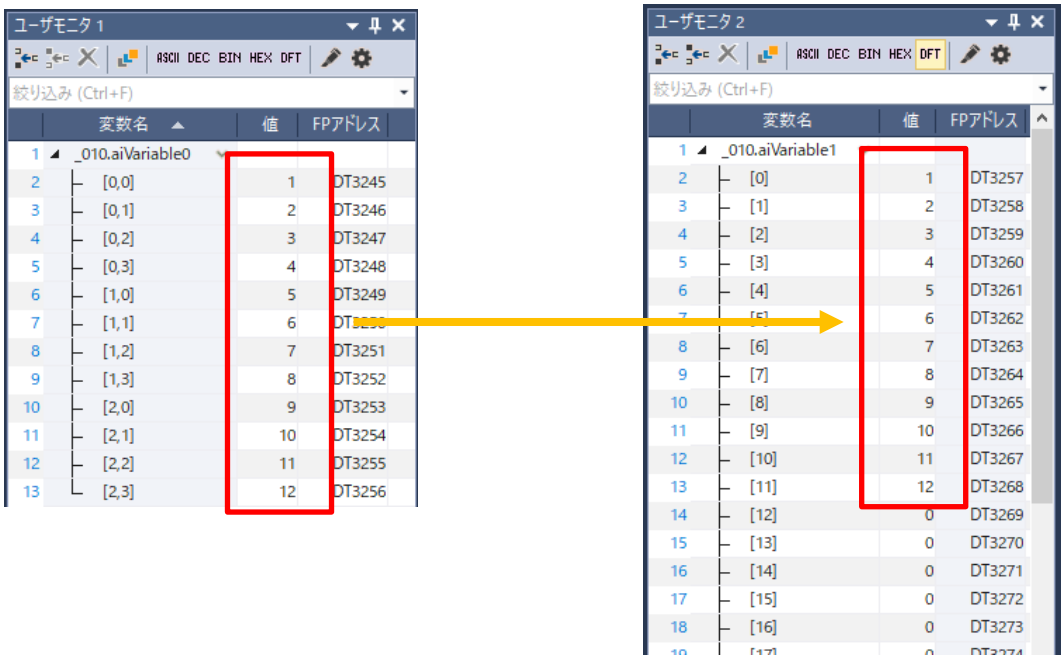
	クラス	変数名	データ型	初期値
1	VAR	bVariable0	BOOL	FALSE
2	VAR	aiVariable0	ARRAY [0..2,0..3] OF INT	[12(0)]
3	VAR	aiVariable1	ARRAY [0..99] OF INT	[100(0)]
4	VAR	iVariable0	INT	0
5	VAR	iVariable1	INT	0

■POU ボディ(Prog)

```
IF (DF(bVariable0)) THEN
   Elem_OfArray2D(Array2D := aiVariable0,
                  NrOfElem1 => iVariable0, NrOfElem2 => iVariable1);   3        4
   F10_BKMV_NUMBER(s1_Start := Adr_Of_Var(aiVariable0),
                   s2_Number := iVariable0 * iVariable1,    3        4
                   d_Start => Adr_Of_Var(aiVariable1));
END_IF;
```

BKMV で転送するための転送データのワード数を計算しています。

転送元の合計配列数(12 ワード)分転送されています。



●注意

Elem_OfArray2D によって得られる結果はあくまでも要素数です。

上記の結果は、iNumber_of_Elements0＝3、iNumber_of_Elements1＝4 となります。

Elem_OfArray3D の使用例

Elem_OfArray3D を使用することによって、3 次元配列変数の要素数を取得することができます。

■POU ヘッダ(Prog)

	クラス	変数名	データ型	初期値
1	VAR	bVariable0	BOOL	FALSE
2	VAR	aiVariable0	ARRAY [0..1,0..2,0..3] OF INT	[24(0)]
3	VAR	aiVariable1	ARRAY [0..99] OF INT	[100(0)]
4	VAR	iNumber_of_Elements0	INT	0
5	VAR	iNumber_of_Elements1	INT	0
6	VAR	iNumber_of_Elements2	INT	0

■POU ボディ(Prog)

```
IF (DF(bVariable0)) THEN
   Elem_OfArray3D(Array3D := aiVariable0,
                  NrOfElem1 => iNumber_of_Elements0,    2
                  NrOfElem2 => iNumber_of_Elements1,    3
                  NrOfElem3 => iNumber_of_Elements2);   4
   F10_BKMV_NUMBER(s1_Start := Adr_Of_Var(aiVariable0),
                   s2_Number := iNumber_of_Elements0    2
                                * iNumber_of_Elements1   3
                                * iNumber_of_Elements2,    4
                   d_Start => Adr_Of_Var(aiVariable1));
END_IF;
```

BKMV で転送するための転送データのワード数を計算しています。

●注意

Elem_OfArray3D によって得られる結果はあくまでも要素数です。

上記の結果は、iNumber_of_Elements0＝2、iNumber_of_Elements1＝3、iNumber_of_Elements2＝4 となります。

●備考

Array3D が 1 次元配列の場合

・入力 Array3D の配列要素数が、出力 NrOfElem1 に出力されます。

・出力 NrOfElem2 には、値"1"が出力されます。

・出力 NrOfElem3 には、値"1"が出力されます。

Array3D が 2 次元配列の場合

・入力 Array3D の配列から、1 次元の要素数が出力 NrOfElem1 に、2 次元の要素数が出力 NrOfElem2 に出力されます。

・出力 NrOfElem3 には、値"1"が出力されます。

配列変数の F 命令への記述方法①

データテーブルを必要とする F 命令には配列変数を直接指定することができます。

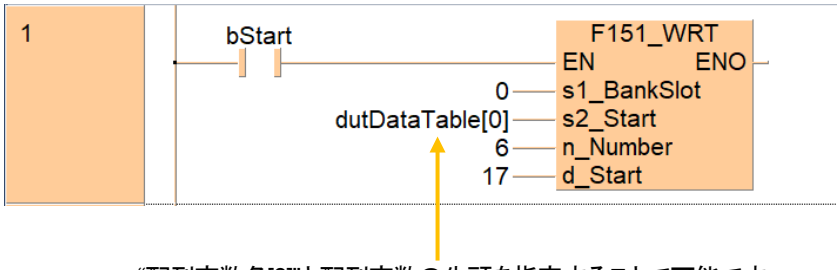
ここでは、FP0H 位置決めユニットの E 点制御用のデータテーブルを書き込む例を用いて紹介します。

■POU ヘッダ（Prog）

	クラス	変数名	データ型	初期値	コメント
1	VAR	bStart	BOOL	FALSE	
2	VAR	dutDataTable	ARRAY [0..4] OF DINT	[16#02000058,1000,10000,100,10000]	

	変数名	データ型	初期値	コメント
	dutDataTable	ARRAY [0..4] OF DINT	[16#02000058,1000,10000,100,10000]	
制御コード	[0]	DINT	16#02000058	
初期速度（pps）	[1]	DINT	1000	
目標速度（pps）	[2]	DINT	10000	
加減速速度（ms）	[3]	DINT	100	
位置指令値（Pulse）	[4]	DINT	10000	

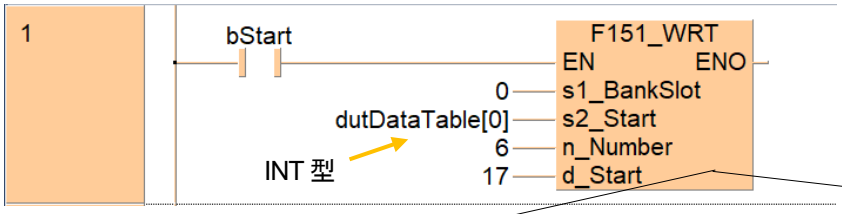
■POU ボディ（Prog）



“配列変数名[0]”と配列変数の先頭を指定することで可能です。

●注意

配列変数の型と F 命令入力オペランドの型が一致していなければなりません。



	クラス	変数名	データ型	初期値
1	VAR_INPUT	s1	ANY16	0
2	VAR_INPUT	s2	ANY16	0
3	VAR_INPUT	n	INT	0
4	VAR_INPUT	d	ANY16	0

上記の例の場合、

配列変数の型と F151 命令の入力オペランドが同じ 1 ワード（INT ＝ ANY16）になっていることがわかります。

配列変数の F 命令への記述方法②

“Adr_Of_Var”を用いることで、型が異なるデータテーブル(配列変数)と F 命令の入力オペランドを接続することができます。

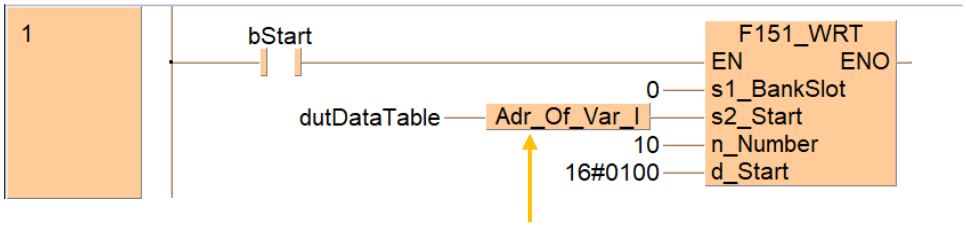
ここでは、FP0H 位置決めユニットに E 点制御用のデータテーブルを書き込む例を用いて紹介します。

■POU ヘッダ(Prog)

	クラス	変数名	データ型	初期値	コメント
1	VAR	bStart	BOOL	FALSE	
2	VAR	dutDataTable	ARRAY [0..4] OF DINT	[16#02000058,1000,10000,100,10000]	

	変数名	データ型	初期値	コメント
	dutDataTable	ARRAY [0..4] OF DINT	[16#02000058,1000,10000,100,10000]	
制御コード	[0]	DINT	16#02000058	
初期速度(pps)	[1]	DINT	1000	
目標速度(pps)	[2]	DINT	10000	
加減速速度(ms)	[3]	DINT	100	
位置指令値(Pulse)	[4]	DINT	10000	

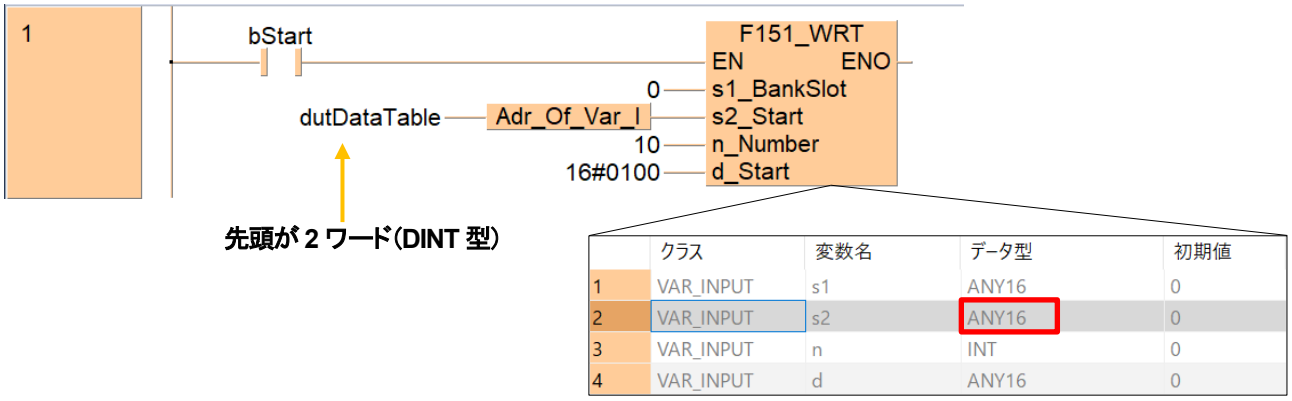
■POU ボディ(Prog)



FP TOOL ライブラリ”Adr_Of_Var_I”を用いて、構造体の先頭デバイスを指定することで可能です。

●注意

配列変数の型と PLC の F 命令入力オペランドの型が一致していなければなりません。



	クラス	変数名	データ型	初期値
1	VAR_INPUT	s1	ANY16	0
2	VAR_INPUT	s2	ANY16	0
3	VAR_INPUT	n	INT	0
4	VAR_INPUT	d	ANY16	0

上記の例の場合、
配列変数の型が 2 ワード(DINT)、F151 命令の入力オペランドが 1 ワード(INT ＝ ANY16)になっているため
“Adr_Of_Var”により、データテーブル先頭の下位 1 ワード目のアドレスを抽出して入力オペランドに指定する必要があります。

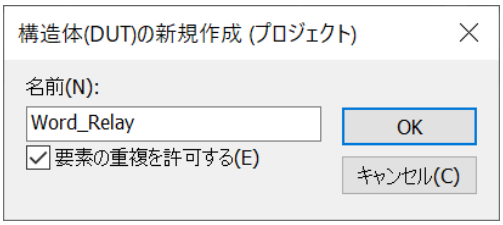
重複したデータアクセス用に重複したエレメントを持つ DUT

重複したエレメントを持つDUTを作成することで、同一のDUT 名でダブルワード・ワード・ビットの指定ができます。
簡単に言えば、1つのDUT名でWR0 とR0～RFを指定することができます。

ここでは、WX0 の値をWR のデバイスに取り込み、リレーに反映させる例で説明します。

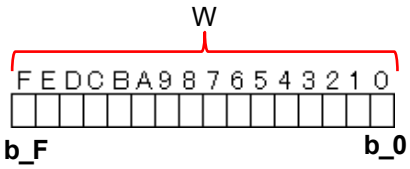
■DUTヘッダ

	変数名	データ型	初期値
1	w	WORD	0
2	b_0	BOOL	FALSE
3	b_1	BOOL	FALSE
4	b_2	BOOL	FALSE
5	b_3	BOOL	FALSE
6	b_4	BOOL	FALSE
7	b_5	BOOL	FALSE
8	b_6	BOOL	FALSE
9	b_7	BOOL	FALSE
10	b_8	BOOL	FALSE
11	b_9	BOOL	FALSE
12	b_A	BOOL	FALSE
13	b_B	BOOL	FALSE
14	b_C	BOOL	FALSE
15	b_D	BOOL	FALSE
16	b_E	BOOL	FALSE
17	b_F	BOOL	FALSE



“要素の重複を許可する”にチェックを付けます!

●Word_Relay(DUT)内部詳細

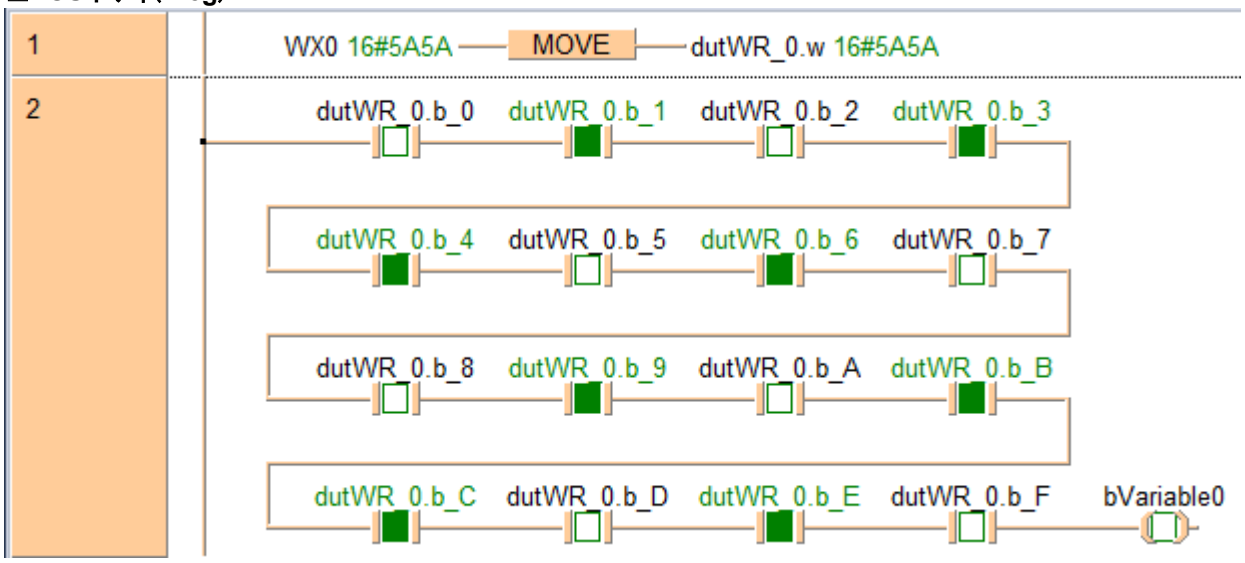


“要素の重複を許可する”ことにより、
同じワードメモリ内でBOOL型16個が割付きます。

■POUヘッダ（Prog）

	クラス	変数名	データ型	初期値
1	VAR	dutWR_0	Word_Relay	
2	VAR	bVariable0	BOOL	FALSE

■POUボディ（Prog）



WX0を”WR_0”という変数名でワード型とビット型にして取り込めているのがわかります。

BOOL16_OVERLAPPING_DUT の使い方

FP Tool Library として”BOOL16_OVERLAPPING_DUT”が用意されています。
例えば、WR0 として使用したり R0,R1,R2･･･RF といったリレーとして使用したりする場合に有効です。

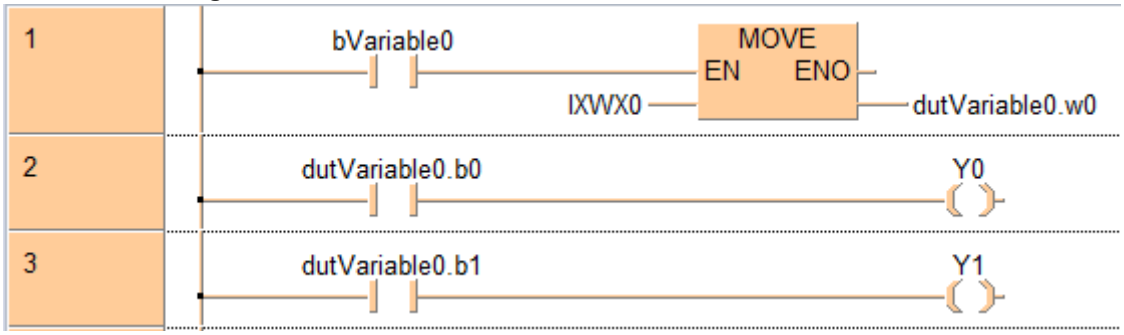
ここでは、IXWX0 のワードデータをリレー(ビット)単位の情報として抜き出す例を用いて紹介します。

■POU ヘッダ(Prog)

	クラス	変数名	データ型	初期値
1	VAR	dutVariable0	BOOL16_OVERLAPPING_DUT	
2	VAR	bVariable0	BOOL	FALSE

データ型に”BOOL16_OVERLAPPING_DUT”を指定

■POU ボディ(Prog)

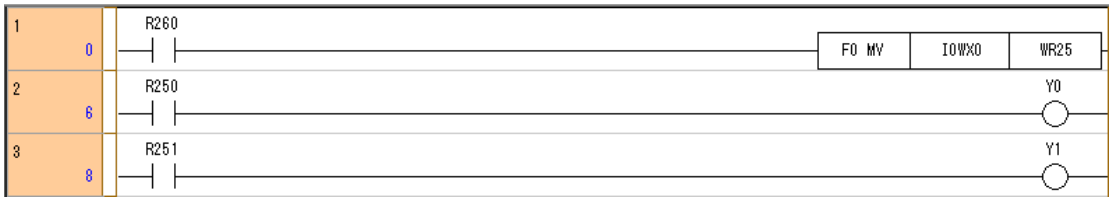


■BOOL16_OVERLAPPING_DUT[構造体(DUT)]

	変数名	データ型	初期値	コメント
1	w0	WORD		Word 0
2	i0	INT		Word 0
3	ui0	UINT		Word 0
4	b0	BOOL		Word 0 - Bit 0
5	b1	BOOL		Word 0 - Bit 1
6	b2	BOOL		Word 0 - Bit 2
7	b3	BOOL		Word 0 - Bit 3
8	b4	BOOL		Word 0 - Bit 4
9	b5	BOOL		Word 0 - Bit 5
10	b6	BOOL		Word 0 - Bit 6
11	b7	BOOL		7
12	b8	BOOL		8
13	b9	BOOL		Word 0 - Bit 9
14	b10	BOOL		Word 0 - Bit 10
15	b11	BOOL		Word 0 - Bit 11
16	b12	BOOL		Word 0 - Bit 12
17	b13	BOOL		Word 0 - Bit 13
18	b14	BOOL		Word 0 - Bit 14
19	b15	BOOL		Word 0 - Bit 15
20	ab	ARRAY [0..15] OF BOOL		Word 0 - Bits 0..15

1 ワード

■コンパイル後のプログラム(FPWIN GR7 ラダー表記)



dutVariable0 が WR25 に割付いており、R250･R251 のビットに展開されていることがわかります。

STRING16_OVERLAPPING_DUT の使い方

FP Tool Libraryとして”STRING16_OVERLAPPING_DUT”が用意されています。
例えば、文字列を 16 進数として使用したい場合に有効です。

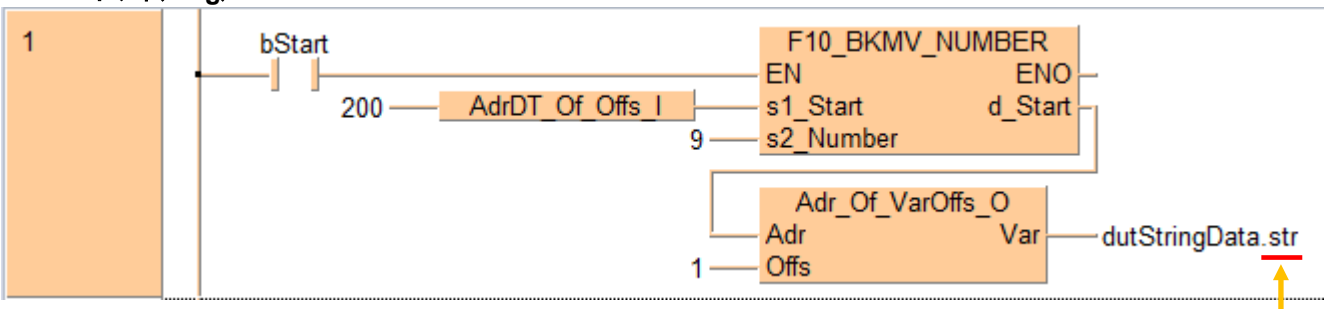
ここでは、受信データ(DT201～DT208)’1234567890ABCDEF’の文字列データを
16 進数(ASCII 文字)の情報として抜き出す例を用いて紹介します。

■POU ヘッダ(Prog)

	クラス	変数名	データ型	初期値
1	VAR	dutStringData	STRING16_OVERLAPPING_DUT	
2	VAR	bStart	BOOL	FALSE

データ型に”STRING16_OVERLAPPING_DUT”を指定

■POU ボディ(Prog)



“str”へ転送

■STRING16_OVERLAPPING_DUT[構造体(DUT)]

	変数名	データ型	初期値	コメント
1	wMaxLen	WORD		Word 0: Maximum number of characters (16)
2	wActLen	WORD		Word 1: Current nubmer of characters
3	wChars_2_1	WORD		Word 2: The characters 1 and 2
4	wChars_4_3	WORD		Word 3: The characters 3 and 4
5	wChars_6_5	WORD		Word 4: The characters 5 and 6
6	wChars_8_7	WORD		Word 5: The characters 7 and 8
7	wChars_10_9	WORD		Word 6: The characters 9 and 10
8	wChars_12_11	WORD		Word 7: The characters 11 and 12
9	wChars_14_13	WORD		Word 8: The characters 13 and 14
10	wChars_16_15	WORD		Word 9: The characters 15 and 16
11	iMaxLen	INT		Int 0: Maximum number of characters (16)
12	iActLen	INT		
13	iChars_2_1	INT		
14	iChars_4_3	INT		
15	iChars_6_5	INT		
16	iChars_8_7	INT		
17	iChars_10_9	INT		
18	iChars_12_11	INT		
19	iChars_14_13	INT		
20	iChars_16_15	INT		
21	aw	ARRAY [0..9] OF WORD		
22	ai	ARRAY [0..9] OF INT		
23	str	STRING[16]		

ユーザモニタ 1

ASCII DEC BIN HEX DFT

絞り込み (Ctrl+F)

	変数名	値	FPアドレス	コメント
1	Test.dutStringData.wChars_2_1	16#3231	DT3254	Word 2: The characters 1 and 2
2	Test.dutStringData.wChars_4_3	16#3433	DT3255	Word 3: The characters 3 and 4
3	Test.dutStringData.wChars_6_5	16#3635	DT3256	Word 4: The characters 5 and 6
4	Test.dutStringData.wChars_8_7	16#3837	DT3257	Word 5: The characters 7 and 8
5	Test.dutStringData.wChars_10_9	16#3039	DT3258	Word 6: The characters 9 and 10
6	Test.dutStringData.wChars_12_11	16#4241	DT3259	Word 7: The characters 11 and 12
7	Test.dutStringData.wChars_14_13	16#4443	DT3260	Word 8: The characters 13 and 14
8	Test.dutStringData.wChars_16_15	16#4645	DT3261	Word 9: The characters 15 and 16

16 進数(ASCII 文字)で取り出せます

STRING_TO_INT と STRING_TO_STEPSARVER

FPWIN Pro7 では F 命令を使用せずに文字列を INT 型、WORD 型等に変換することが出来ます。

ここでは、”STRING_TO_INT”と”STRING_TO_INT_STEPSARVER”を例に紹介します。

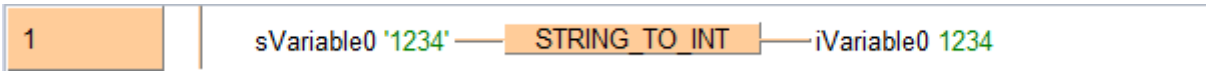
■POU ヘッダ（Prog）

	クラス	変数名	データ型	初期値
1	VAR	sVariable0	STRING[32]	'1234'
2	VAR	iVariable0	INT	0

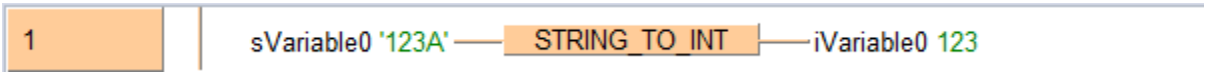
■POU ボディ（Prog）

”STRING_TO_INT”

●データが INT 型の場合



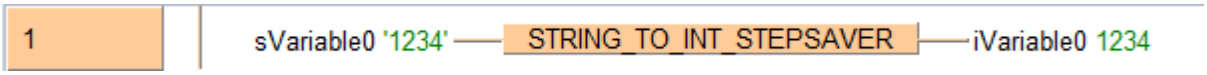
●データが INT 型でない場合



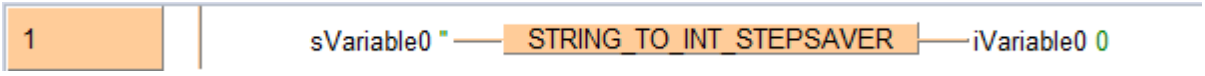
※INT 型でない文字の前まで変換を行い、エラーは発生しません。

”STRING_TO_INT_STEPSARVER”

●データが INT 型の場合



●データが INT 型でない場合



※INT 型でない文字が入力された時点で演算エラーが発生します。

”STRING_TO_INT”

約270ステップを使用したプログラムで構成されており、FPシリーズが持っているF命令は使用していません。
データの先頭から変換を行い、INT型でないデータを認識した時点で変換を中止するため、演算エラーは発生しません。

”STRING_TO_INT_STEPSARVER”

内部でF76（ABIN）命令を使用して変換を行っています。
プログラム容量は数ステップと小さいですが、INT 型でないデータを入力した時点で演算エラーが発生します。

■備考

DINT, WORD, DWORD 型への各変換命令も”_STEPSARVER”が付く命令は内部で F 命令を使用しているため
異なるデータ型の文字が入力されると演算エラーとなります。

STRING_TO_DINT_STEPSARVER　　:F78（DABI）命令。
STRING_TO_WORD_STEPSARVER　:F72（AHEX）命令。
STRING_TO_DWORD_STEPSARVER　:F72（AHEX）命令。

四則演算結果の WORD 型変数への格納

F 命令を使用することで、WORD 型に四則演算結果を格納できます。

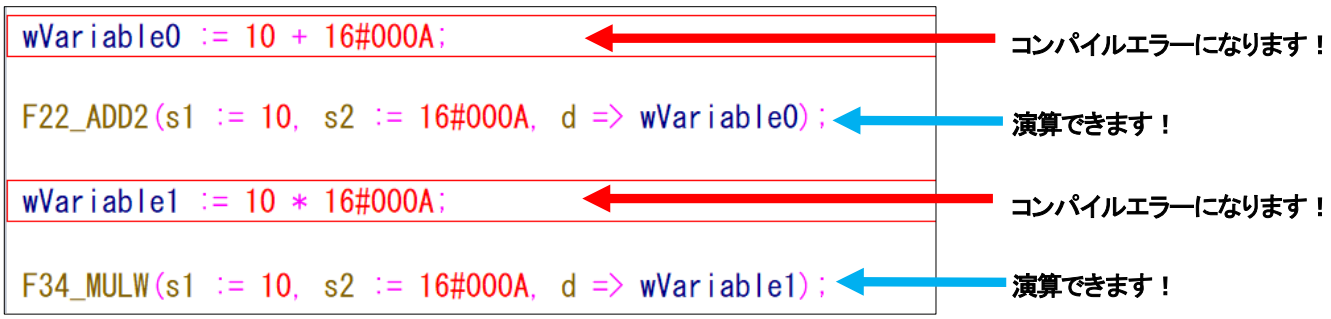
IEC命令では算術演算子＋,－,＊,／の格納先にはINT, UINT, DINT, UDINT型しか使用できません。
格納先にWORD型を指定したい場合にはF命令を使用します。

■POUヘッダ(Prog)

	クラス	変数名	データ型	初期値
1	VAR	wVariable0	WORD	0
2	VAR	wVariable1	WORD	0

■POUボディ(Prog)

```
wVariable0 := 10 + 16#000A;
F22_ADD2(s1 := 10, s2 := 16#000A, d => wVariable0);
wVariable1 := 10 * 16#000A;
F34_MULW(s1 := 10, s2 := 16#000A, d => wVariable1);
```



DUT の初期値を使用した応用事例

自作したDUTには初期値を設定することができますが、その初期値を使用することで、プログラムを簡素化することができます。

ここでは、FP0H位置決めユニットにE点制御用のデータテーブルを書き込む例を用いて紹介します。

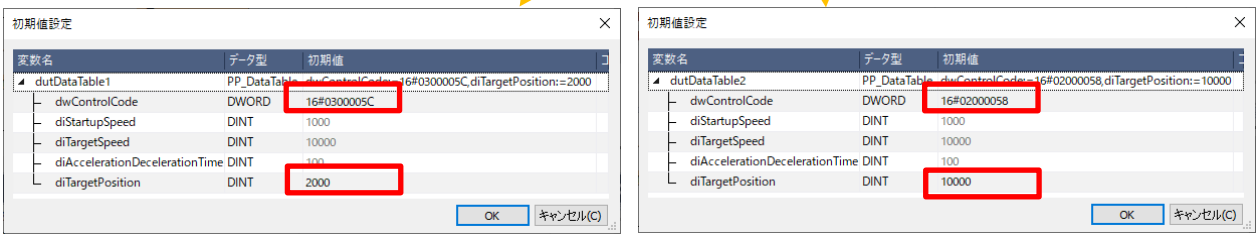
■DUTヘッダ

	変数名	データ型	初期値
1	dwControlCode	DWORD	0
2	diStartupSpeed	DINT	1000
3	diTargetSpeed	DINT	10000
4	diAccelerationDecelerationTime	DINT	100
5	diTargetPosition	DINT	0

固定値として設定したいパラメータに DUT ヘッダの初期値を設定。

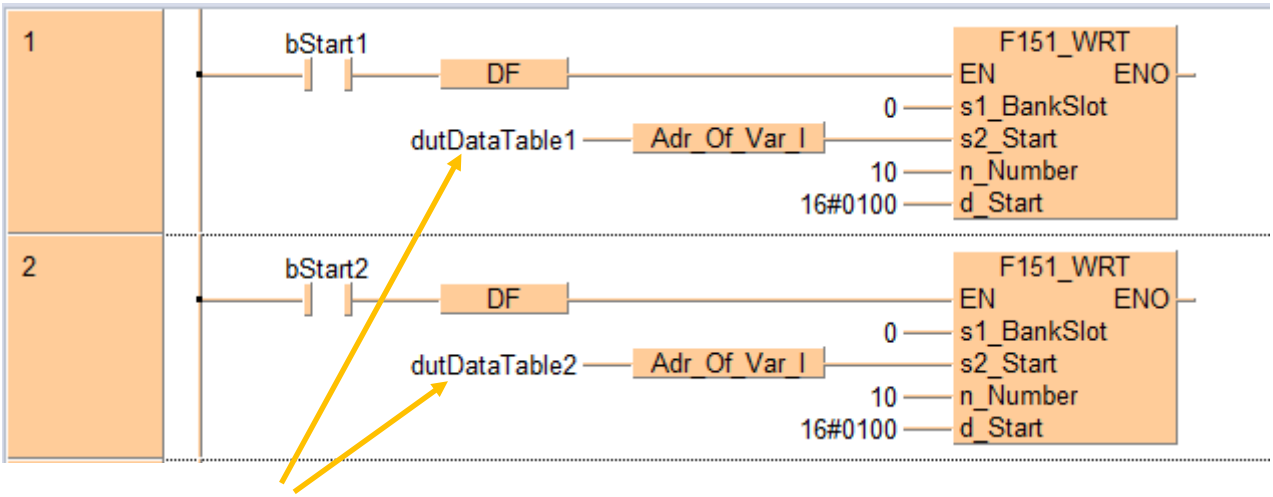
■POUヘッダ（Prog）

	クラス	変数名	データ型	初期値
1	VAR	dutDataTable1	PP_DataTable	dwControlCode:=16#0300005C,diTargetPosition:=2000
2	VAR	dutDataTable2	PP_DataTable	dwControlCode:=16#02000058,diTargetPosition:=10000
3	VAR	bStart1	BOOL	FALSE
4	VAR	bStart2	BOOL	FALSE



変更したいパラメータを POU ヘッダでの初期値として値を設定。

●POUボディ（Prog）



値の異なるデータテーブルを複数個作成できる。

FP アドレスを割り付けたグローバル変数の挿入

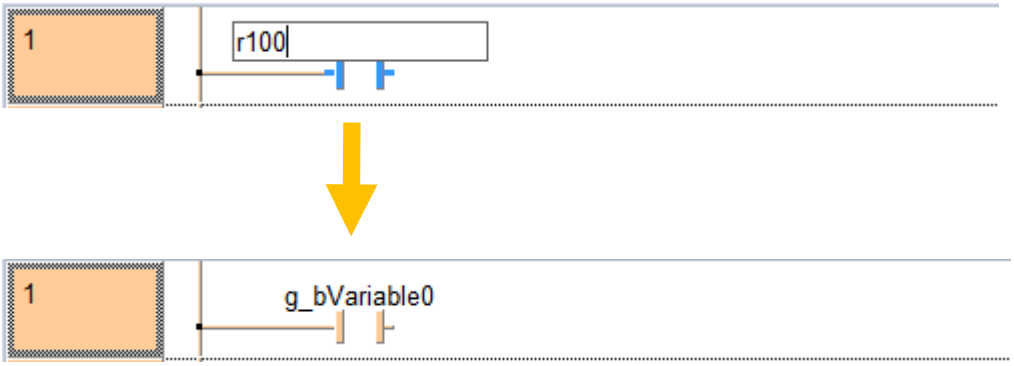
LD、ST ボディで変数名を入力、コピーせずにグローバル変数を挿入する方法を紹介します。

■グローバル変数

	クラス	変数名	FPアドレス	IECアドレス	データ型	初期値
1	VAR_GLOBAL	g_bVariable0	R100	%MX0.10.0	BOOL	FALSE

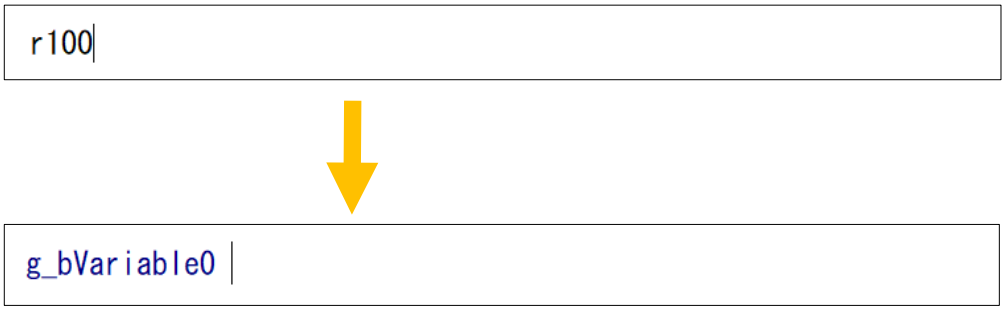
■POU ボディ(Prog) LD 言語の場合

FP アドレスを直接入力し、「Enter」キーを押します。(R100 大文字でも可)



■POU ボディ(Prog) ST 言語の場合

FP アドレスを直接入力し、「Space」キーを押します。(R100 大文字でも可)

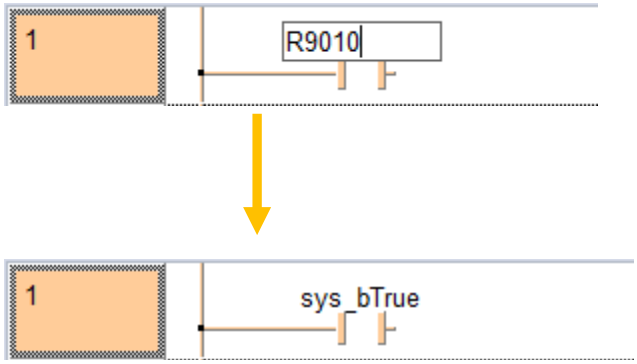


特殊内部リレーをアドレスで挿入

特殊内部リレーを変数ペインからコピーせずに、アドレス入力で直接挿入する方法を紹介します。

■LD 言語の場合

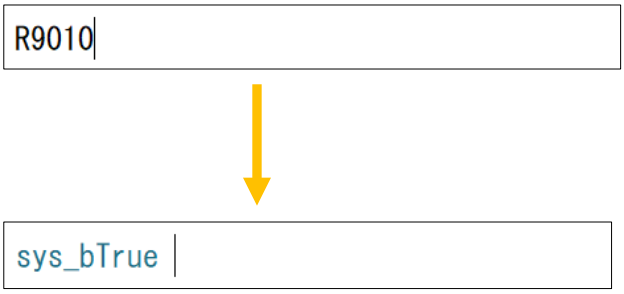
常時 ON リレー:R9010(FP7 の場合は SR10)と入力し「Enter」キーを押す。



■ST 言語の場合

1s クロックパルスリレー

常時 ON リレー:R9010(FP7 の場合は SR10)と入力し「Space」キーを押す。



PRO7 と WH で使用する文字列データの構造と GR7 でのプログラミング

WH で使用する文字列データの構造は、PRO7 で使用する文字列データの構造と同じです。
（先頭 1 ワードが文字列格納領域サイズ、2 ワード目が格納文字数、3 ワード目以降文字列データ）

GR7 でのプログラミング注意点

GR7 のシリアル通信（汎用通信）で受信するデータを WH でモニターする場合、
受信先頭アドレス = DT_nとした場合、WH で指定する文字列変数タグのアドレス =DT_n+1 となるように指定してください。
PLC の DT_n-1 は実在するアドレスとなるように設定してください。（n に 0 は指定しないでください）
DT_n-1 は、PRO7 で使用する文字列と同じ構造にするために必要です。
WH で文字列をモニターする場合、DT_n-1 には、何が入っていてもかまいませんが、文字列格納領域サイズに相当する値を入れておくことを推奨します。

GR7 でプログラムする場合は、本構造に留意して、文字列演算には下記の命令群を使用してください。

ESSET (文字定数→ASCII コード変換:格納領域サイズ付き)
EPRINT(文章作成:格納領域サイズ付き)
ETIMEstr (日付・時刻文字列変換:格納領域サイズ付き)
ESCMP (文字列比較:格納領域サイズ付き)
ESADD (文字列加算:格納領域サイズ付き)
ELEN (文字列長取得:格納領域サイズ付き)
ESSRC (文字列検索:格納領域サイズ付き)
ERIGHT (文字列右側取り出し:格納領域サイズ付き)
ELEFT (文字列左側取り出し:格納領域サイズ付き)
EMIDR (文字列任意位置からの読出し:格納領域サイズ付き)
EMIDW (文字列任意位置からの書換:格納領域サイズ付き)
ESREP (文字列置き換え:格納領域サイズ付き)

SD カード内ファイルへの文字データ書き込みサンプル

SD カードに文字列変数を書き込む場合、FP_SD_WRITE_LINE 命令を使用します。
この命令は、一度に最大 4096 文字を書き込むことができます。4096 文字を超える文字列データは、4096 文字ごとに
複数回にわたって書き込み処理を行います。
書き込みには複数スキャンかかりますので、完了するまで次の書き込みはできません。
SD カードが PLC に装着されていなければ実行できません。また SD カードの蓋が閉まっていないと実行できません。
・書き込み実行前には、以下のフラグを確認する必要があります。
　　SD カード装着確認フラグ sys_bIsSDCardAttached ： TRUE 装着、FALSE 未装着
　　SD カード蓋閉確認フラグ sys_bIsSDCardSlotOpen ： TRUE 開、FALSE 閉
・また、下記の SD カードアクセス中フラグをチェックして、二重の書き込み要求を避ける必要があります。
　　SD カードアクセス中フラグ sys_bIsSDMemoryAccessActive ： TRUE アクセス中
・書き込み完了待ちの際は、以下のフラグをチェックして完了チェックをすることができます。
　　SD カードアクセス完了フラグ sys_bIsSDMemoryAccessDone ： TRUE 完了
・書き込み完了を確認したら、異常が発生していないかを以下のフラグで確認できます。
　　SD カードアクセスエラーフラグ sys_bIsSDMemoryAccessError ： TRUE 異常発生

以下のサンプルプログラムを参考にしてください。

```
(*
SD カード書き込みサンプルプログラム
【機能】
g_sSDWriteData に格納されている 1 万文字のデータを
SD カードに"Test.txt"ファイルとして書き込みます。

実行要求:　　bExecution → TRUE　再実行は、→ FALSE → TRUE
正常終了時:bDone → TRUE
異常終了時:bError → TRUE
*)

if DF(bExecution) then// SD カード書き込み処理開始
    iSDWTstage:=1;
    end_if;
if DFN(bExecution) then
    // SD カード書き込み要求無
    iSDWTstage:=0;
end_if;

case iSDWTstage of
// 無処理
0:
    bDone:=false;
    bError:=false;
```

```
// SD カード書き込み前チェック
1:    ;
// SD カード装着と蓋のチェック
      if (NOT(sys_bIsSDCardAttached) or sys_bIsSDCardSlotOpen) then
            // 書き込み状態変数を異常終了状態にセット
            iSDWTstage:=4;
      // 問題なければ、変数初期化
      else
            iWTCounter:= 3;               // 書き込み回数は 3 回  10000/4096 = 2.44
            iTotalLength:=10000;          //全書き込み文字数
            iStartPosition:= 1;// 書き込み開始位置
            iSDWTstage:=2;                // 書き込み要求
            bDone:=false;
      end_if;

      if DF(bExecution) then
        // SD カード書き込み処理開始
        iSDWTstage:=1;
      end_if;
      if DFN(bExecution) then
        // SD カード書き込み要求無
        iSDWTstage:=0;
      end_if;

// SD カード書き込み要求
2:    ;
      // SD カード書き込み中でなければ、書き込み要求
      if NOT(sys_bIsSDMemoryAccessActive) then
      iTotalLength:=iTotalLength-4096;
      if (iTotalLength>=0) then
      iGetLength:=4096;
      else
      iGetLength:=iTotalLength+4096;
      end_if;
      // 書き込み文字列 4096 文字の抽出
      sSDWrite := MID(IN := g_sSDWriteData, L := iGetLength, P := iStartPosition);
      // 書き込み要求  書き込みファイルは test.txt
      FP_SD_WRITE_LINE(s_In := sSDWrite, d_FullFileName := 'test.txt');
      // 書き込み完了待ちへ
      iSDWTstage:=3;
      end_if;
```

```
// SD カード書き込み完了待ち
3:    ;
      // SD カード書き込み完了チェック
      if sys_bIsSDMemoryAccessDone then
            // SD カード書き込み異常チェック
            if sys_bIsSDMemoryAccessError then
                  // 書き込み異常終了
                  iSDWTstage:=4;
            else
                  // 書き込み正常終了
                  iWTCounter:= iWTCounter-1;
                  // 全データの書き込み完了？
                  if (iWTCounter<>0) then
                        iStartPosition:=iStartPosition+4096;
                        // 書き込み要求処理へ
                        iSDWTstage:=2;
                  else
                        // 全データ書き込み完了
                        iSDWTstage:=5;
                  end_if;
            end_if;
      end_if;
4:    ;
      // 書き込み異常検知
      bError:=true;
5:    ;
      // 全データ書き込み完了
      bDone:=true;
else  ;
end_case;
```

構造体(共用体)を用いた型変換

共用体を使用すると型(構造)の異なる変数の内部を別の型でアクセスすることができます。
ここで共用体と呼ぶものは、DUT (重複した要素をもつ DUT)のことです。
この構造体定義を使用すると、同じメモリを別々の型の変数として二重に定義できるので、
型変換の命令を使用しなくても簡単に異なる型の異なるデータ型の変数としてアクセスすることができます。

システムで用意されている共用体の例
- BOOL16_OVERLAPPING_DUT
- BOOL32_OVERLAPPING_DUT
- BOOL64_LREAL_OVERLAPPING_DUT
- BOOL64_OVERLAPPING_DUT
- WORD_OVERLAPPING_DUT
- DWORD_OVERLAPPING_DUT
- LWORD_LREAL_OVERLAPPING_DUT
- LWORD_OVERLAPPING_DUT
- STRING16_OVERLAPPING_DUT

これらの構造体はシステムで構造体型変数として用意されていますが、各種のメンバ定義がされているため
使用時に不要なメンバなども含まれる場合、自分で定義した構造体を使用しても全く問題ありません。

具体例

汎用シリアル通信において、受信した文字列の受信数を確認する場合や文字列の中身を HEX データとして取り扱いたい場合に STRING 型の文字列変数メンバとワード型の配列変数メンバを重複定義して使用すると便利です

構造体の定義例

dutCOM_RECEIVE_DATA

	変数名	データ型	初期値
1	awComReceiveData	ARRAY [0..51] OF WORD	[52(0)]
2	sComReceiveData	STRING[4096]	''

上記では、
52 要素のワード型配列 awComReceiveData と
4096 文字の文字列データ sComReceiveData を重複定義しています。

下記のようなプログラムで、文字列型の構造体メンバ sComReceiveData に受信データを読み出したのち
dutCom1ReceiveData.sComReceiveData:=ReceiveCharacters(SYS_COM1_PORT);

受信データを文字列としてだけでなくワード型配列メンバ awComReceiveData 変数を使用してアクセスすることが可能です。

文字列変換時のゼロパディング

STRING 型への変換命令のうち、「＊＊_TO_STRING_LEADING_ZEROS」命令を使用することで、変換後の文字列のデータサイズに応じてゼロパディング（ゼロ埋め）を行うことができます。

※＊＊には INT,UINT,DINT,UDINT の整数データ型を指定でき、それにより命令も分かれています。
・INT_TO_STRING_LEADING_ZEROS　　INT をゼロパディングで文字列に変換
・UINT_TO_STRING_LEADING_ZEROS　　UINT をゼロパディングで文字列に変換
・DINT_TO_STRING_LEADING_ZEROS　　DINT をゼロパディングで文字列に変換
・UDINT_TO_STRING_LEADING_ZEROS　　UDINT をゼロパディングで文字列に変換

データ型 宣言	変換結果
STRING[1]	‘1’
STRING [2]	‘01’
STRING [3]	‘001’
STRING [4]	‘0001’
・ ・	・ ・
STRING [10]	‘0000000001’
STRING [11]	‘00000000001’
・ ・	・ ・

参考:ツール上でのモニタ状態
・STRING[1]

```
sStringData := INT_TO_STRING_LEADING_ZEROS(1); '1'
```

・STRING[2]

```
sStringData := INT_TO_STRING_LEADING_ZEROS(1); '01'
```

・STRING[3]

```
sStringData := INT_TO_STRING_LEADING_ZEROS(1); '001'
```

・STRING[4]

```
sStringData := INT_TO_STRING_LEADING_ZEROS(1); '0001'
```

・STRING[10]

```
sStringData := INT_TO_STRING_LEADING_ZEROS(1); '0000000001'
```

・STRING[11]

```
sStringData := INT_TO_STRING_LEADING_ZEROS(1); '00000000001'
```

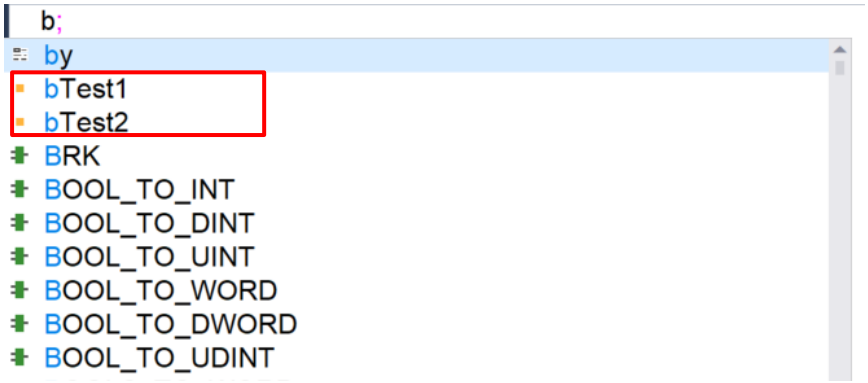
オートコンプリート機能

ここでは、Control FPWIN Pro7 のオートコンプリート機能について紹介します。

オートコンプリート機能を使用することでプログラム作成の際に、
宣言した変数や、システム変数が予測で表示されるようになります

ST ボディでの例
ローカル変数

ST ボディ

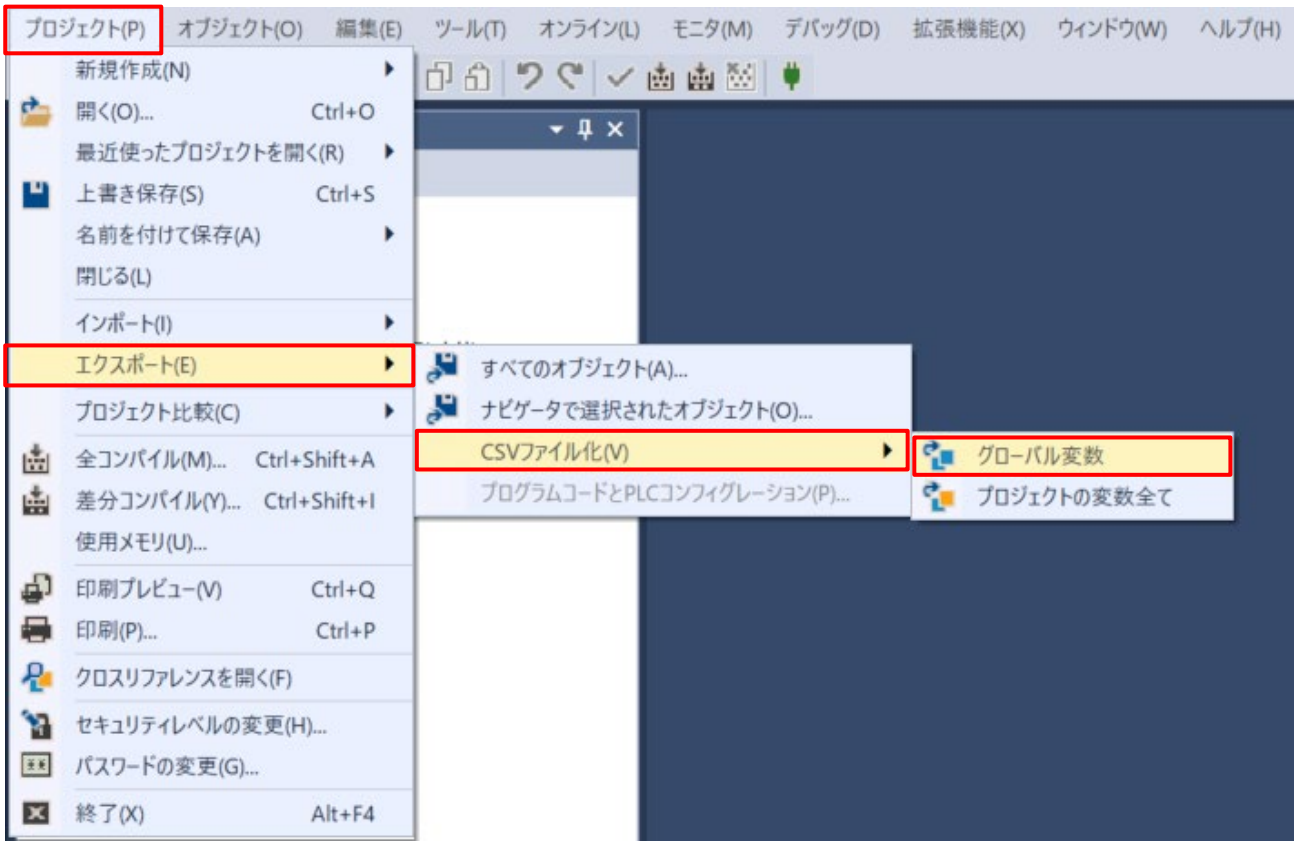


上記のようにローカル変数で「bTest1」「bTest2」が宣言されている場合、ST ボディで「b」と入力すると、
予測で「bTest1」「bTest2」が表示されます。

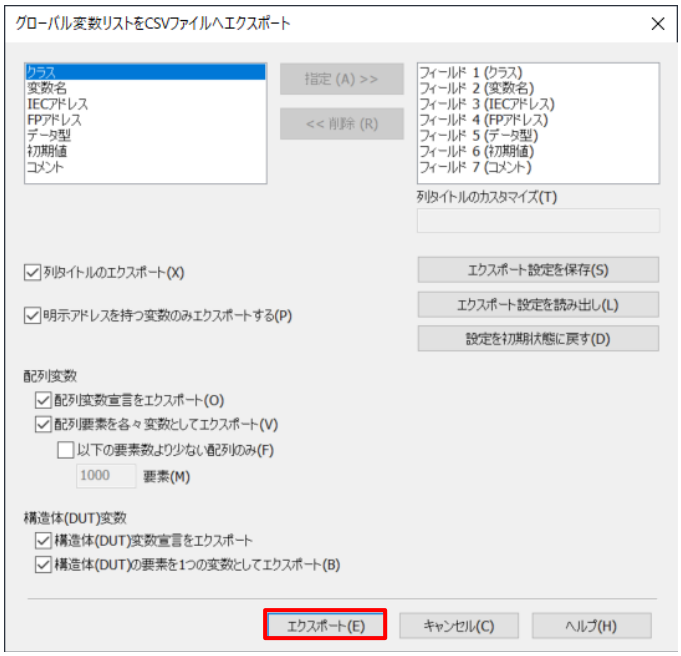
グローバル変数を csv ファイルにエクスポート

FPWIN Pro7 で定義したグローバル変数を csv ファイルにエクスポートする方法を紹介します。

メニューバー内「プロジェクト」→「エクスポート」→「csv ファイル化」→「グローバル変数」を選択します。



「グローバル変数リストを CSV ファイルへエクスポート」のダイアログ内にて、
出力したい内容を選択し、「エクスポート」をクリックすることでグローバル変数をエクスポートすることが可能です。



BOOL の反転

ここでは、BOOL の ON/OFF（TRUE/FALSE）状態を反転させるプログラムを紹介します。

使用する変数


プログラム

	クラス	変数名	データ型	初期値	コメント
1	VAR	bTest	BOOL	FALSE	
2	VAR	bFlag	BOOL	FALSE	
3	VAR				

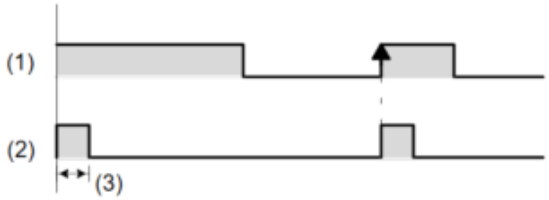
サンプルプログラム

```
IF (DF(bTest)) THEN
   bFlag:=NOT bFlag;
END_IF;
```

上記サンプルプログラム内の ‘bTest’ を ON することで、‘bFlag’ の状態を切り替えることができます。
例）
bFlag：ON の場合　→ bFlag：OFF
bFlag：OFF の場合　→ bFlag：ON

DFI

DFI 命令を使用することで、第 1 スキャンでも、入力信号の立上りエッジを検出することができます。

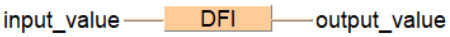


(1) 入力信号
(2) 出力信号
(3) 1スキャン

STボディ

```
output_value:=DFI(input_value);
```

LDボディ



PLC の電源投入前に ON しているセンサなどの入力信号でも、立上りエッジとして検出することができます。

外部機器に PING を送信するサンプルプログラム

PC から発行する PING コマンドは、指定された IP アドレスに対して、ICMP の ECHO Request メッセージを送信し、
送信先から ICMP の ECHO Reply メッセージを受信します。これにより、生存確認や応答時間の確認などを行います。

PLC の PING 発行命令は、命令入力として指定されたコネクションで指定されている相手局 IP アドレスに対して ICMP プロトコルの ECHO Request メッセージを送信します。指定されたコネクションの情報は、相手局の IP アドレスのみ使用されます。
PING 発行命令で指定するコネクションは、TCP でも UDP でもどちらでも良いですし、そのコネクションで相手先と何らかの通信を実施しているか否かは無関係です。
PING 発行命令で使用できるコネクションは、使用許可設定されたコネクションで下記の○のついた設定のコネクションのみです。
（理由:相手局の IP アドレスが設定されているため）

○	TCP	クライアント	相手局指定	IP アドレス＋ポート番号
○	TCP	サーバ	相手局特定	IP アドレス＋ポート番号
×	TCP	サーバ	相手局任意	IP アドレス指定なし
○	UDP		相手局特定	IP アドレス
×	UDP		相手局任意	IP アドレス＝0

これらの○のコネクションが何らかの通信に使用されているかどうかは PING 発行命令の動作には無関係です、
コネクションを通信に使用しない場合は、UDP（相手局特定）設定が、無駄な動作が少なくなります。
コネクションを通信に使用しない場合で、TCP を指定したい場合は、「自動オープンしない」設定をしてください。

複数の機器に PING を発行するには、複数のコネクション設定（宛先の IP アドレス）が必要です。
また、別々の機器に対して同時に送信することはできませんので、命令を順番に実行する必要があります。

具体例
FP7 (IP = 192.168.1.6) → WH (192.168.1.5) への PING 発行の例（ユーザコネクション 1 を使用）
FP_ETHERNET_PING 命令を使用します。

ユーザコネクション 1 の設定（下図）
　ここでは、UDP を選択、相手局 IP アドレスに 192.168.1.5 を設定

変数定義とプログラム

プログラム

	クラス	変数名	データ型	初期値	コメント
1	VAR	bEnable	BOOL	FALSE	
2	VAR	iRequest	INT	0	
3	VAR	dutEthernet	FP_ETHERNET_PING_...		
4	VAR	bPingOK	BOOL	FALSE	
5	VAR				

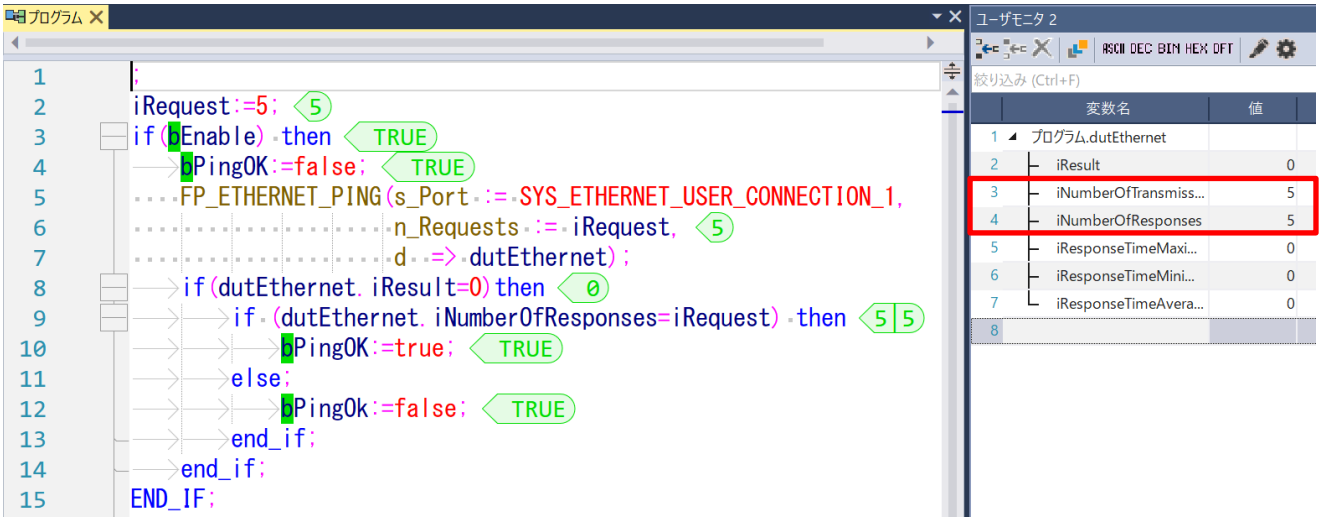
```
1  ;
2  iRequest:=5;
3  if(bEnable) then
4      bPingOK:=false;
5      FP_ETHERNET_PING(s_Port := SYS_ETHERNET_USER_CONNECTION_1,
6                       n_Requests := iRequest,
7                       d => dutEthernet);
8      if(dutEthernet.iResult=0)then
9          if (dutEthernet.iNumberOfResponses=iRequest) then
10             bPingOK:=true;
11         else;
12             bPingOk:=false;
13         end_if;
14     end_if;
15 END_IF;
```

bEnable を true にすると FP_ETHERNET_PING 命令が実行されて、結果が dutEthernet 構造体に格納されます。
実行完了には、時間がかかります。
実行結果コードが FFFF から 0 になったら、送信回数と応答回数の一致を確認してください。

pPINGREQ 命令の出力オペランド	FP_ETHERNET_PING 命令の出力構造体 dutEthernet	実行結果内容	説明
[D]	iResult	実行結果コード	FFFF:実行中、0 正常終了
[D+1]	iNumberOfTransmissions	送信回数	
[D+2]	iNumberOfResponses	応答回数	
[D+3]	iResponseTimeMaximum	応答時間(最大)	0～1000(ms) 応答時間はすべ て 10ms 単位です。 10ms 以下の場合 0 が格納されます。
[D+4]	iResponseTimeMinimum	応答時間(最小)	
[D+5]	iResponseTimeAverage	応答時間(平均)	

実行結果

ICMP の ECHO Request を 5 回送信して、ICMP の ECHO Reply を 5 回受信しています。



IP アドレスを設定するサンプルプログラム

ここでは、PLC の内蔵イーサネット通信の IP アドレス設定方法について説明します。
設備をエンドユーザに納品後、納品先での環境に合わせて IP アドレスを設定していただく必要のある場合などに有効です。
ここでは、Control FPWIN Pro7 FP_IPV4_SET_ADDRESS 命令を使用したサンプルプログラムを紹介します。

■タグの設定例

データ	タイプ	タグ名	タグURI
Panasonic FP/FP7:prot1 Model: FP7	Container		
g_bIPaddressSet	boolean	g_bIPaddressSet	1?R?0.0x00?boolean
g_sIPaddress	string[32]	g_sIPaddress	1?DT?0.0?string-32

■PLC プログラム例

IP アドレス設定開始前のモニタ画面

```
1
2   if DF(g_bIPaddressSet) then FALSE
3       //FP_IPv4_SET_ADDRESS命令の入力パラメータの設定
4       sNewIPaddress:='IP='; ''
5       sNewIpaddress:=CONCAT(sNewIPaddress, ''                    ''
6                             g_sIPaddress, //WHからの入力IPアドレス ''
7                             ',MASK=255.255.255.0,GWIP=0.0.0.0');
8
9       //IPアドレスの設定
10      FP_IPV4_SET_ADDRESS(sAddress:=sNewIpaddress, bError=>bError); ''
11  end_if;
```

IP アドレス設定と設定要求ボタンクリック後のモニタ画面

```
1
2   if DF(g_bIPaddressSet) then TRUE
3       //FP_IPv4_SET_ADDRESS命令の入力パラメータの設定
4       sNewIPaddress:='IP='; 'IP=192.168.1.100,MASK=255.255.255.0,GWIP=0.0.0.0'
5       sNewIpaddress:=CONCAT(sNewIPaddress, 'IP=192.168.1.100,MASK=255.255.255.0,GWIP=0.0.0.0
6                             g_sIPaddress, //WHからの入力IPアドレス '192.168.1.100'
7                             ',MASK=255.255.255.0,GWIP=0.0.0.0');
8
9       //IPアドレスの設定
10      FP_IPV4_SET_ADDRESS(sAddress:=sNewIpaddress, bError=>bError); 'IP=192.168.1.100,MASK=
11  end_if;
```

【IP アドレス設定命令使用時の注意事項】

IP アドレス設定命令(IPv4SET または FP_IPV4_SET_ADDRESS 命令)は、イーサネットの設定自体を変更していないので PLC の電源断や PROG 切り替え後の設定書き換え後に再度 RUN となった場合は、コネクション設定値が優先されます。
その場合もオンライン設定値を有効にするためには、RUN 後に自動的に IP アドレス設定命令を実行して IP アドレスを再設定するプログラムが必要になります。

一定時間ごとに処理を実行するプログラム

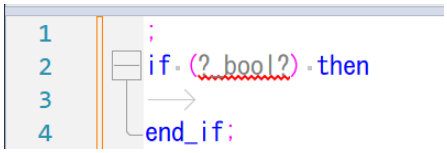
ST 言語で一定時間ごとに1回処理を実行するプログラムは、どのように記述すれば良いでしょうか？
Control FPWIN GR7 で同様のプログラムを書く場合、PLC の特殊リレーやタイマ命令を使用してタイミングを作ります。
Control FPWIN Pro7 の場合も同様ですが、特殊リレーを使用するにはシステム変数を指定する必要があります。
一定時間ごとに実行するには、下記のような方法があります。

1:特殊リレーを使用する
2:タイマ命令を使用する
3:リングカウンタを使用する
4:一定周期実行のプログラムを使用する(割り込み実行プログラム)

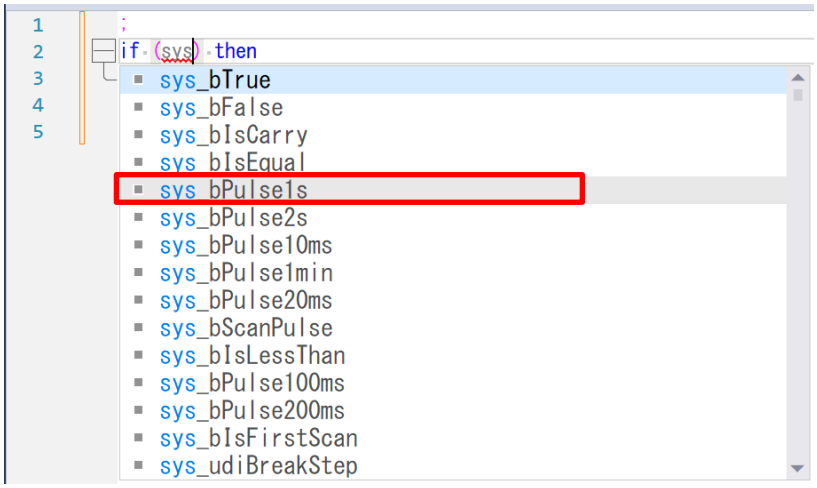
ここでは、PLC の特殊リレーを使用して作成する方法を説明いたします。
PLC(FP7)のクロックリレーには下記のようなものがあります。右側には PRO7 での変数名を記載しています。

リレー番号	説明	Pro7 システム変数名
SR18	0.01 秒クロックパルスリレー	sys_bPulse10ms
SR19	0.02 秒クロックパルスリレー	sys_bPulse20ms
SR1A	0.1 秒クロックパルスリレー	sys_bPulse100ms
SR1B	0.2 秒クロックパルスリレー	sys_bPulse200ms
SR1C	1 秒クロックパルスリレー	sys_bPulse1s
SR1D	2 秒クロックパルスリレー	sys_bPulse2s
SR1E	1 分クロックパルスリレー	sys_bPulse1min

【プログラム例】
if に続いてスペースを入力しますと、下図のように if 文の構文が自動入力されます。



(?_bool?)の部分が条件式を入力する部分で、bool 型の値を求められています。
条件式の値が true の場合に then 以下の処理が実行され、false の場合には、実行されません。
下図は、(?_bool?)の「?_bool?」部分が選択された状態で、sys と入力したときの入力補助画面です。
下図赤枠部分に 1 秒クロックパルスリレーが表示されているので、これを選択します。



条件式に 1 秒クロックパルスリレーが入力できました。

```
1  ;
2  if (sys_bPulse1s) then
3  ⟶
4  end_if;
```

このままでは、1 秒クロックパルスリレーが ON の間ずっと処理を実行してしまいますので、最初の(の前に立ち上がりを検知する命令 DF を入力します。

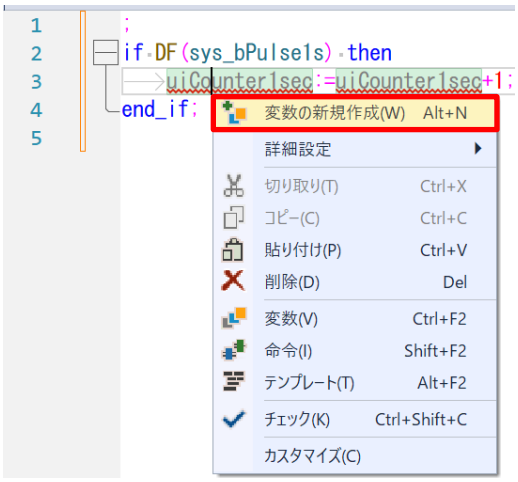
```
1  ;
2  if DF(sys_bPulse1s) then
3  ⟶
4  end_if;
```

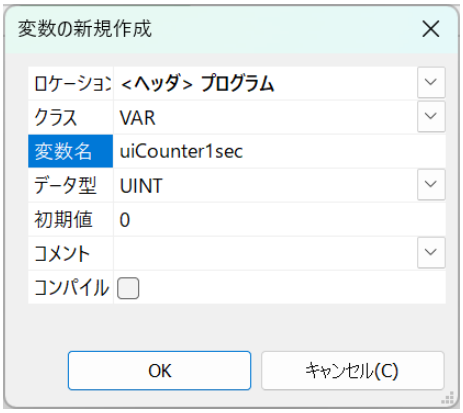
これで、1 秒に 1 回処理が可能になりました。3 行目に処理(uiCounter1sec 変数に+1 する)を入力します。
uiCounter1sec:=uiCounter1sec+1; を入力します。
下図は、入力結果です。変数名の下に赤波線が入っているのは、変数が未登録であることを示しています。

```
1  ;
2  if DF(sys_bPulse1s) then
3  ⟶uiCounter1sec:=uiCounter1sec+1;
4  end_if;
```

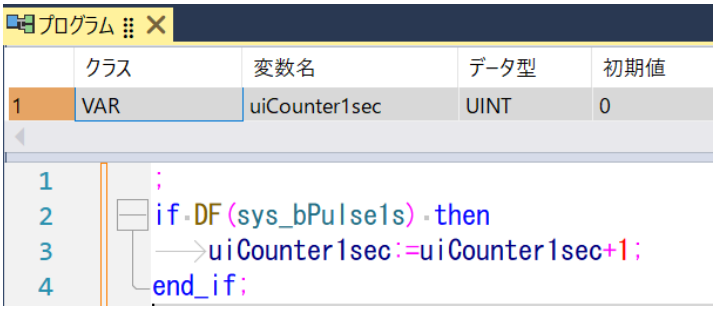
いずれかの変数名 uiCounter1sec を右クリックして変数を新規登録します。

変数名の上で、右クリックするとメニューが表示され、一番上の変数の新規作成をクリックします。
右下図の変数の新規作成ダイアログが表示され、クラス、データ型、初期値などの候補が表示されます。
このままで問題ないので、OK をクリックして作成します。

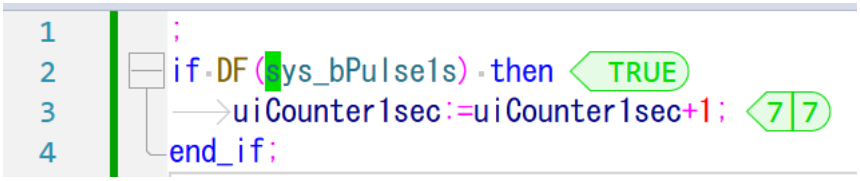




変数がヘッダ部に登録され、未定義警告の赤波線もなくなりました。



シミュレータにダウンロードして uiCounter1sec が1秒ごとに+1 されることをモニタで確認してください。
下図は、実行画面です。uiCounter1sec のモニタ値が 1 秒ごとに+1 されていくのが確認できます。



memo

改訂履歴

発行日時	番号	改訂内容
2020 年 9 月	gts209ja	初版
2022 年 1 月	gts209ja-01	新規記事追加
2023 年 3 月	gts209ja-02	新規記事追加
2024 年 3 月	gts209ja-03	新規記事追加

●技術に関するお問い合わせは
FAデバイス技術相談窓口
0120-394-205
※受付時間／9:00 ～ 17:00(12:00～13:00、弊社休業日を除く)
Webサイト industrial.panasonic.com/ac/
パナソニック インダストリー株式会社 産業デバイス事業部
〒574-0044 大阪府大東市諸福7丁目1番1号

No.GTS209ja-03