SIEMENS

SIMATIC

STEP 7 V5.4 の使い方

はじめに

エディション 08/2007
A5E01112981-01

安全上のガイドライン

このマニュアルには、ユーザの安全を保護し、製品および接続された機器の損傷を避けるために遵守すべき注意事項が記載されています。注意事項は、三角形の警告マークで強調されており、危険度に応じて以下の等級に分類されています。

危険
適切な予防措置を講じなければ、極めて高い可能性で、死亡、重傷、または機器の重大な損傷を引き起こす恐れがあります。

警告
適切な予防措置を講じなければ、死亡、重傷、または機器の重大な損傷を引き起こす恐れがあります。

注意
安全警告シンボルと一緒に使用される場合は、適切な予防措置を講じなければ、人体に軽度の傷害を引き起こす恐れがあります。

注意
安全シンボルなしで使用されている場合は、適切な予防措置を講じなければ、機器の損傷を引き起こす恐れがあります。

注
製品、製品の取扱い、マニュアル類の当該事項に関する特に重要な情報を表します。

有資格者

この機器の取り付けと作業を行うことができるのは**有資格者**だけです。このマニュアルに記載される安全上の注意事項の意味において、有資格者とは、定められた保全作業および安全基準に従って、機器とシステムの配線と設置を行うことを許可された人員をいいます。

正しい使用方法

以下の点に注意してください。

警告
この製品とそのコンポーネントは、カタログまたは説明書に記載されている用途でのみ使用できます。また、Siemens が推奨または承認する他の機器およびコンポーネントとの接続においてのみ使用可能です。

この製品は、輸送、保管、セットアップ、取り付けが正しく行われ、適切な操作とメンテナンスが行われた場合にのみ、安全かつ正確に機能します。

商標

SIMATICR と SINECR は、SIEMENS AG の登録商標です。

本書のその他の商標は、第三者が自己の目的のために使用すると、所有者の権利を侵害する恐れを持つ商標である場合があります。

免責事項

本マニュアルの内容は、実際のハードウェアおよびソフトウェアと一致するよう細心の注意を払っています。 しかしながら、相違点をすべて取り除くことはできないため、完全な一致を保証するものではありません。 本マニュアルの内容は定期的に見直され、必要な修正は次回の版で行われます。 ご意見やご要望などありましたら、お知らせください。

Siemens AG
Bereich Automation and Drives
Geschaeftsgebiet Industrial Automation Systems
Postfach 4848, D- 90327 Nuernberg

STEP 7 へようこそ...

SIMATIC スタンダードソフトウェアは、SIMATIC S7-300/400 ステーション用に、ラダーロジック、ファンクションブロックダイアグラム、ステートメントリストを使ってプログラマブルロジックコントロールプログラムを作成するためのソフトウェアです。

このマニュアルについて

このマニュアルでは、SIMATIC STEP 7 の基本事項を説明します。画面に表示されるダイアログボックスや操作手順のうちで最も重要なものを、実践的な演習を使って説明します。これらの演習は、どの章からでも始められるように構成されています。

各章は、説明のセクション (余白の色がグレイ) と手順のセクション (余白の色がグリーン) に分かれています。グリーンの余白に矢印が付いている箇所は、そこから操作手順の説明が始まることを示しています。操作手順の説明は数ページにわたり、最後は関連事項をボックスで囲んで示しています。

マウス、ウィンドウ操作、プルダウンメニューなどを扱ったことがあれば作業が進めやすくなります。また、プログラマブルロジックコントローラの基本原理を熟知していればさらに有利です。

STEP 7 トレーニングコースでは、STEP 7 でオートメーションソリューション全体を作成する方法が説明され、本書の内容よりも詳しい知識を習得することができます。

このマニュアルを使用するための前提条件

このマニュアルで STEP 7 の演習を行うには、以下が必要です。

- Siemens プログラミング装置または PC
- STEP 7 ソフトウェアパッケージとライセンスキー
- SIMATIC S7-300/S7-400 プログラマブルコントローラ
  (第 7 章「プログラムのダウンロードおよびデバッグ」で必要)

STEP 7 のその他のドキュメント

- STEP 7 基本情報
- STEP 7 リファレンス情報

STEP 7 のインストールが完了すると、[スタート] メニューの **[Simatic| S7 マニュアル]** から電子マニュアルを使用できます。この電子マニュアルは、Siemens セールスセンターに注文することもできます。このマニュアルの全情報は、STEP 7 のオンラインヘルプで呼び出すことができます。

それではご健闘をお祈り申し上げます。

SIEMENS AG

目次

第 6 章と第 7 章では、ハードウェアをコンフィグレーションし、作成したプログラムをテストします。

第 8 章から第 11 章では、新しいファンクションの挿入方法について説明します。

1 STEP 7 の基本

1.1 本書で学ぶこと

このマニュアルでは、ラダーロジック、ステートメントリスト、ファンクションブロックダイアグラムを使って STEP 7 でプログラミングする方法を実践的な演習を通して説明します。

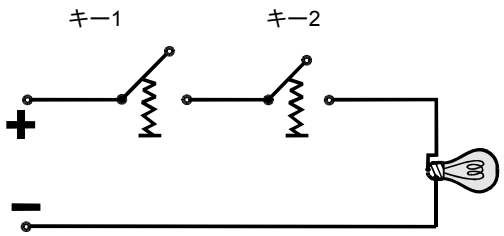
各章では、STEP 7 の様々な使用方法をステップバイステップ方式で詳細に説明します。

バイナリロジックによるプログラムの作成

第 2 章から第 7 章では、バイナリロジックでプログラムを作成します。プログラミングした論理演算を使って、CPU の入力と出力を指定します。

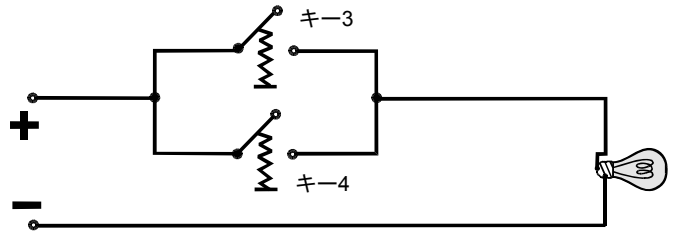
このマニュアルのプログラミング例は、3 種類の基本的な 2 値論理演算を基本としています。

最初の 2 値論理演算(後で実習)は AND ファンクションです。AND ファンクションは、2 つのキーを使用した回路図を表すのに適しています。



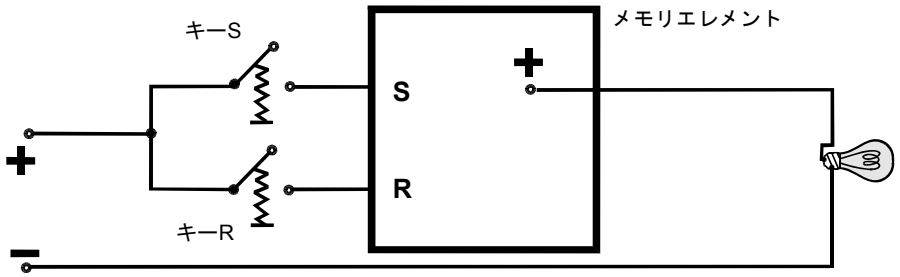
キー1 **と**キー2 の両方を押すと、電球が点灯する

2 番目の 2 値論理演算は OR ファンクションです。 OR ファンクションも回路図で表すことができます。



キー3 **と**キー4 のどちらかを押すと、電球が点灯する

3 番目の 2 値論理演算はメモリエレメントです。SR ファンクションは、回路図内で特定の電圧状態に反応し、その状態に従って動作します。

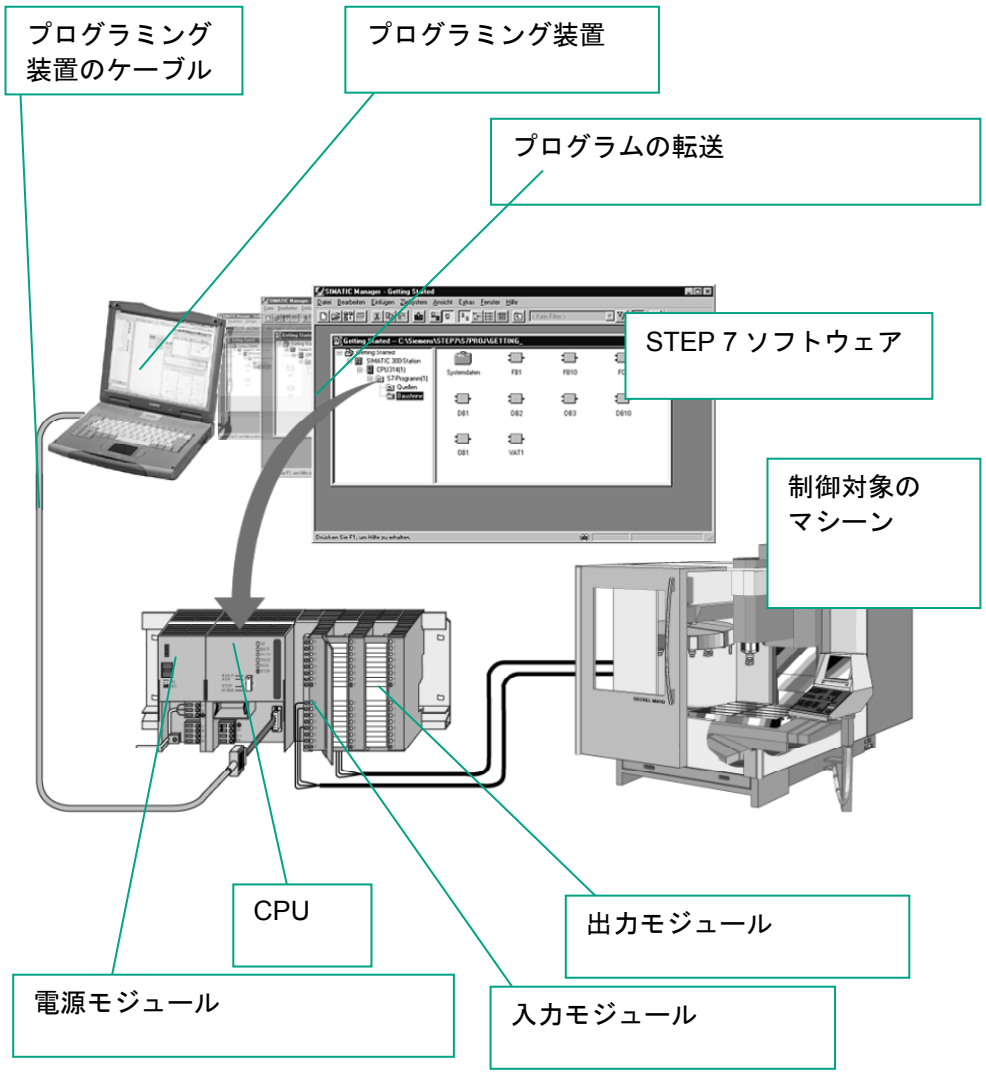


キーS を押すと電球が点灯し、キーR を押すまで点灯している

1.2 ハードウェアとソフトウェアの結合

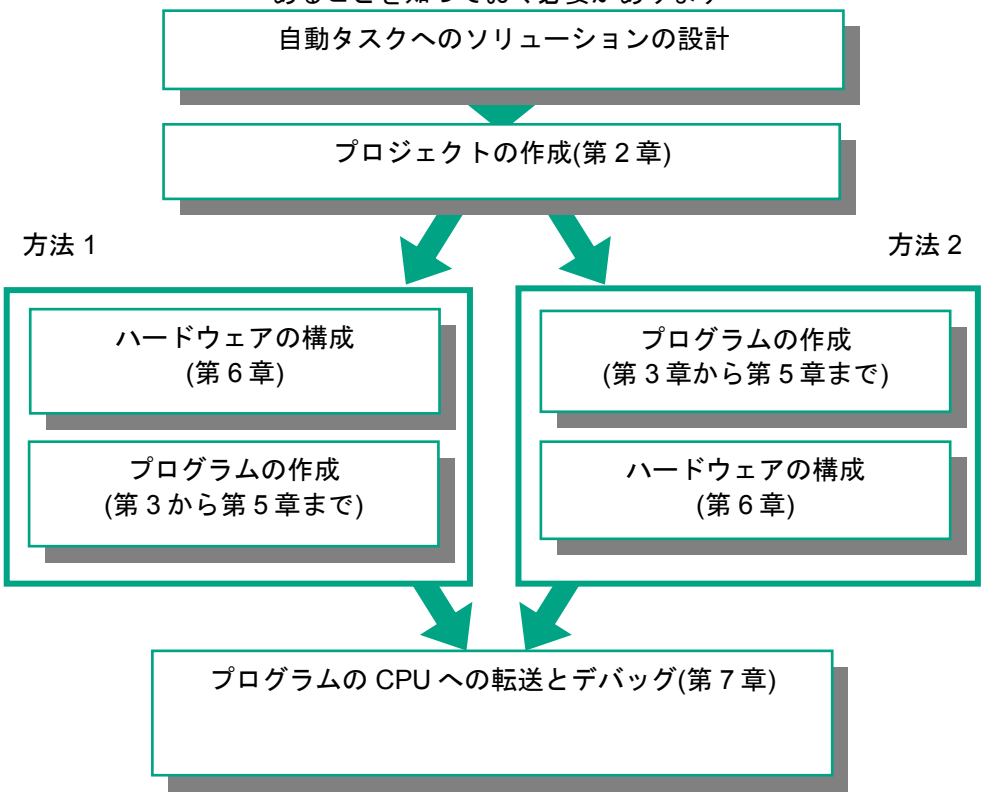
STEP 7 ソフトウェアにより、プロジェクト内で S7 プログラムを作成することができます。S7 プログラマブルコントローラは、電源ユニット、CPU、入出力モジュール (I/O モジュール) で構成されます。

プログラマブルロジックコントローラ(PLC)は、S7 プログラムによりマシンのモニタおよび制御を行います。I/O モジュールのアドレス指定は、S7 プログラムでアドレスを使って行います。



1.3 STEP 7 の基本的な使用方法

プロジェクトを作成する前に、STEP 7 プロジェクトには 2 通りの作成順序があることを知っておく必要があります



多数の入力と出力を使用する総合的なプログラムを作成する場合は、ハードウェアのコンフィグレーションを先に行うことを推奨します。この方法ならば、STEP 7 のハードウェアコンフィグレーションエディタにアドレスが表示されるという利点があります。

プログラムを先に作成する方法では、選択したコンポーネントによってはユーザ自身がアドレスを指定しなければならず、このアドレスを STEP 7 で呼び出すことはできません。

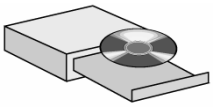
ハードウェアコンフィグレーションでは、アドレスを定義でき、しかもモジュールのパラメータとプロパティを変更することができます。複数の CPU を使用するような場合は、すべての CPU の MPI アドレスを一致させる必要があります。

このマニュアルでは、使用する入力と出力の数が少ないため、ハードウェアコンフィグレーションからではなく、プログラミングから開始します。

1.4 STEP 7 のインストール

プログラミングまたはハードウェアコンフィグレーションのどちらから開始するにしても、最初に STEP 7 をインストールする必要があります。SIMATIC プログラミング装置を使用する場合、STEP 7 はすでにインストールされています。

STEP 7 がインストールされていないプログラミング装置や PC に STEP 7 ソフトウェアをインストールする場合は、ソフトウェアとハードウェアの動作条件を確認してください。これについては、STEP 7 CD の Readme.wri (**<Drive>:/STEP 7 /Disk1**) を参照してください。



STEP 7 を最初にインストールする場合は、STEP 7 CD を CD-ROM ドライブに挿入します。インストールプログラムが自動的に開始します。画面の指示に従います。

インストールが自動的に開始しない場合は、CD-ROM のインストールプログラム (**<Drive>:/STEP 7 /Disk1/setup.exe**) を使用します。

インストールの完了後にコンピュータを再起動すると、Windows のデスクトップに"SIMATIC Manager"アイコンが表示されます。

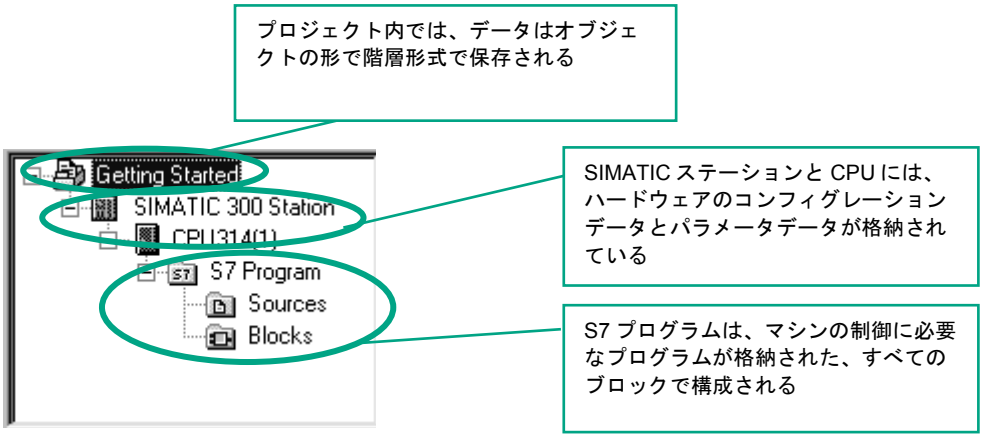
インストール後に"SIMATIC Manager"アイコンをダブルクリックすると、STEP 7 ウィザードが自動的に起動します。

インストールに関するその他の留意事項は、STEP 7 CD の Readme.wri ファイル(**<Drive>:/STEP 7 /Disk1\Readme.wri**) を参照してください。

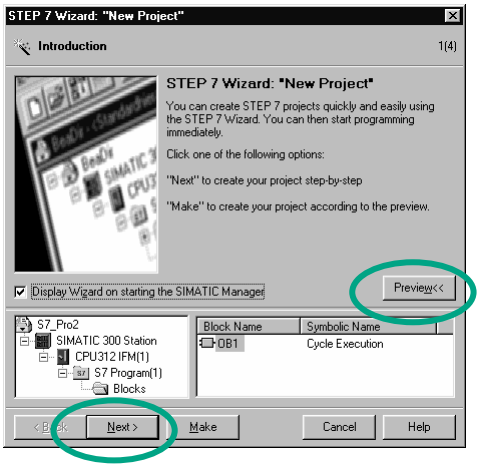
2 SIMATIC Manager

2.1 SIMATIC Manager の起動およびプロジェクトの作成

SIMATIC Manager は、STEP 7 が起動するとアクティブになる基本ウィンドウです。デフォルト設定では、STEP 7 プロジェクトを作成するための STEP 7 ウィザードが起動します。データとプログラムは、プロジェクト構造に従って保存され、規則正しく配置されます。

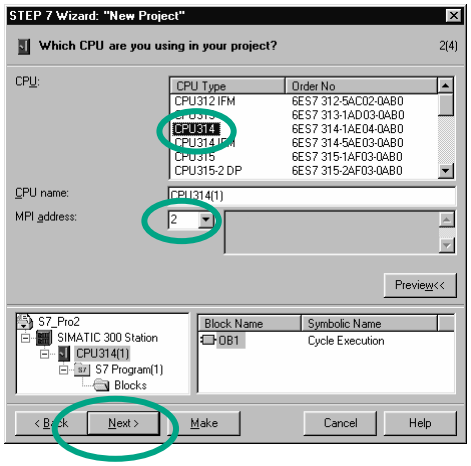


Windows デスクトップの[**SIMATIC Manager**]アイコンをダブルクリックし、ウィザードが自動的に開始しない場合はメニューコマンド[**ファイル|"新規プロジェクト"ウィザード**]を選択します。



[**プレビュー**]では、作成されるプロジェクト構造の表示のオン/オフを切り替えることができます。

次のダイアログボックスへ移動するには、[**次へ**]をクリックします。



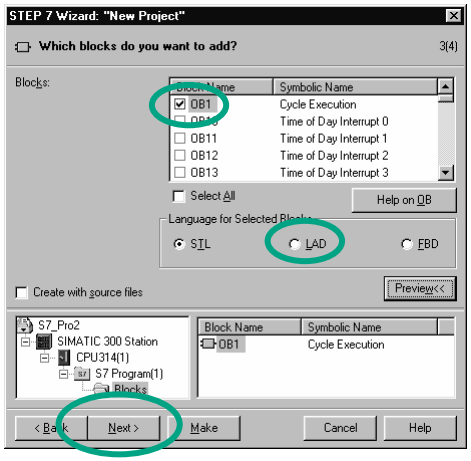
"Getting Started"のサンプルプロジェクトの場合は、CPU 314 を選択します。例は同様に作成されていて、提供されている CPU をいつでも実際に選択することができます。

MPI アドレスのデフォルト設定は 2 です。

設定内容を確認して[**次へ**]をクリックし、次のダイアログボックスへ進みます。

各 CPU には一定のプロパティがあります。たとえば、メモリコンフィグレーションやアドレス領域に関するものです。これは、プログラミングを開始する前に CPU を選択する必要があるからです。

MPI アドレス(マルチポイントインターフェース)は、CPU がプログラミング装置や PC と通信するために必要です。



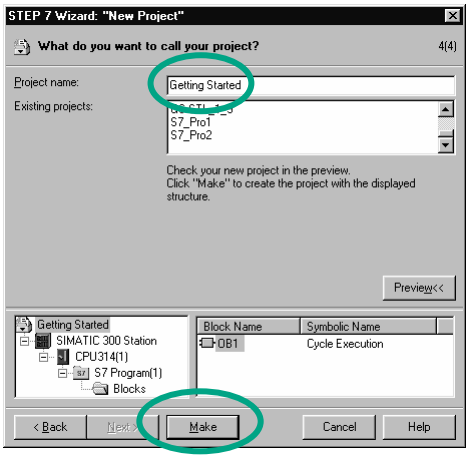
オーガニゼーションブロック **OB1** を選択します(選択していない場合)。

次のプログラム言語の 1 つを選択します。ラダーロジック**(LAD)**、ステートメントリスト**(STL)**、ファンクションブロックダイアグラム**(FBD)**

設定内容を確認して、[**次へ**]をクリックします。

OB1 は、最上位のプログラミングレベルで、S7 プログラムの他のブロックを編成します。

プログラミング言語は後で変更することができます。



[プロジェクト名]欄に示されている名前をダブルクリックして選択し、これを"Getting Started"で上書きします。

[**作成**]をクリックして、プレビューに従って新しいプロジェクトを作成します。

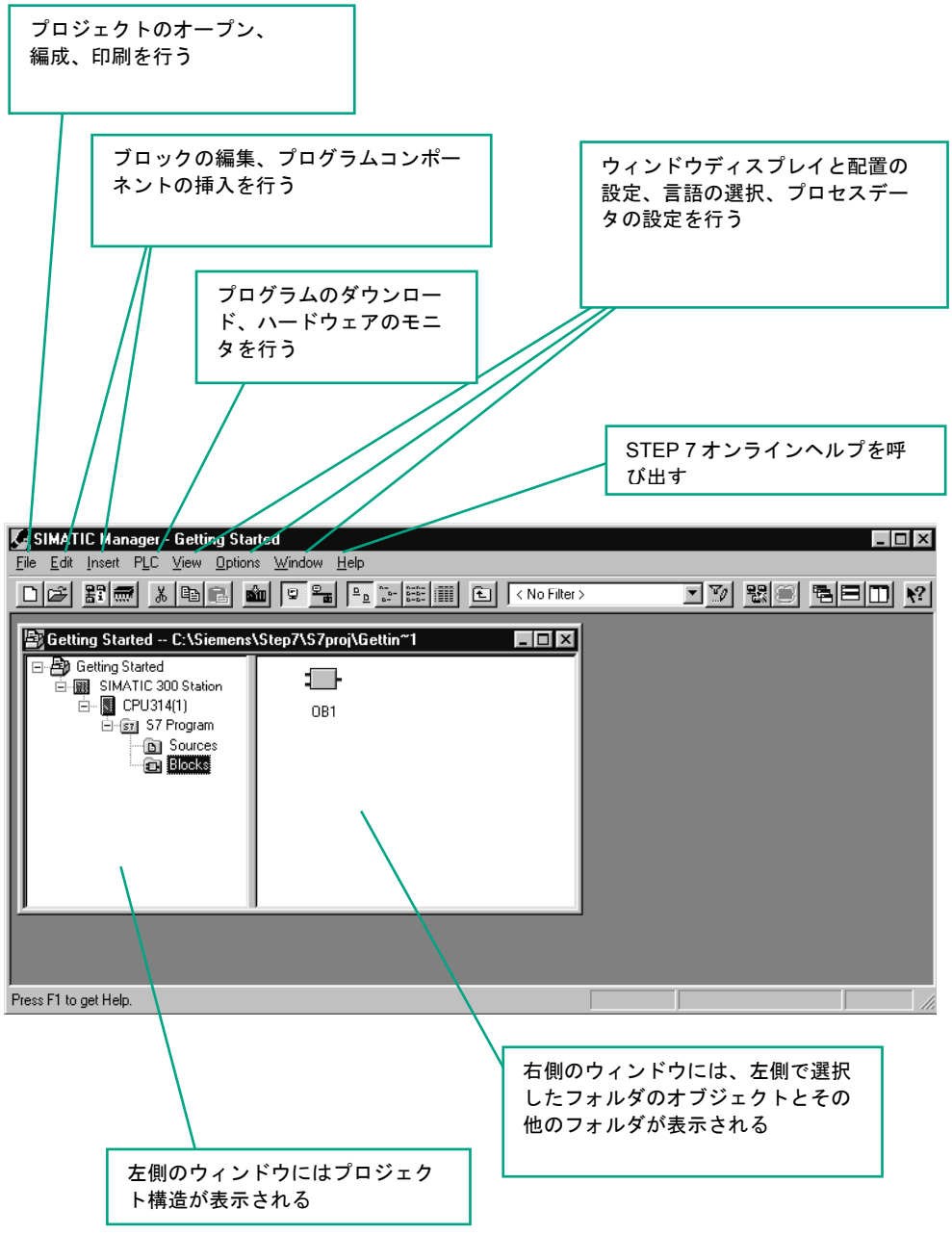
[**作成**]ボタンをクリックすると、SIMATIC Manager が開き、作成した"Getting Started"プロジェクトがウィンドウに表示されます。作成したファイルとフォルダの目的と、それらを効果的に使用する方法を以降のページで説明します。

STEP 7 ウィザードは、プログラムが開始するたびに起動します。これはデフォルト設定ですが、ウィザードの最初に表示されるダイアログボックスで無効にできます。ただし、STEP 7 ウィザードを使わずにプロジェクトを作成する場合、ユーザ自身がプロジェクト内にディレクトリを作成しなければなりません。

詳細については、[**ヘルプ|目次**]の「プロジェクトのセットアップおよび編集」を参照してください。

2.2 SIMATIC Manager のプロジェクト構造、およびオンラインヘルプの呼び出し方

STEP 7 ウィザードが閉じると直ちに、"Getting Started"プロジェクトウィンドウが開いて SIMATIC Manager が表示されます。ここから、STEP 7 のすべてのファンクションとウィンドウを開始できます。

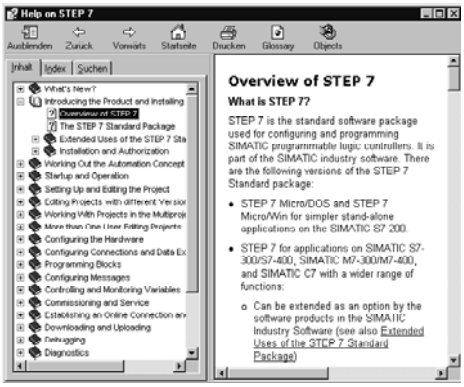


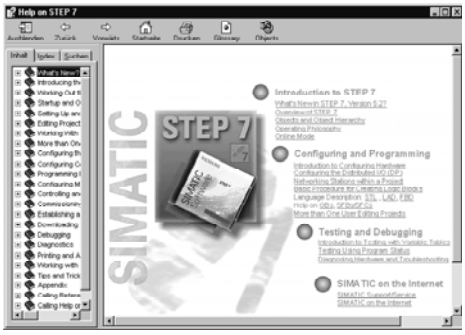
STEP 7 でのヘルプの呼び出し

F1

方法 1:
任意のメニューコマンドにカーソルを置き、**F1** キーを押します。そのメニューコマンドに関するヘルプが表示されます。

方法 2:
メニューを使って STEP 7 オンラインヘルプを開きます。
左側のウィンドウには様々なヘルプトピックが記載された目次ページが開き、右側のウィンドウには選択したトピックが表示されます。
目的のトピックへ移動するには、[**目次**]リストの**+**記号をクリックします。これで、選択したトピックの内容が右側のウィンドウに表示されます。
[**インデックス**]と[**検索**]により、検索文字列を入力し、特定のトピックを検索することができます。



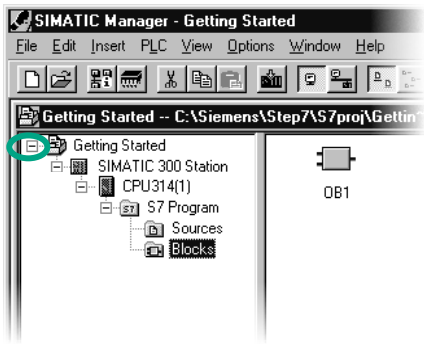


方法 3:
STEP 7 オンラインヘルプの[開始ページ]アイコンをクリックして情報ポータルを開きます。このポータルには、オンラインヘルプの主要トピックなどへのアクセスがまとめられています。

- STEP 7 の入門
- コンフィグレーションとプログラミング
- テストとデバッグ
- インターネットでの SIMATIC

方法 4:
ツールバーにある疑問符のボタンをクリックします。マウスの形がヘルプカーソルの形に変わります。次に特定のオブジェクトをクリックすると、オンラインヘルプが起動します。

プロジェクト構造での移動



作成したプロジェクトは、選択した S7 ステーションおよび CPU と共に表示されます。

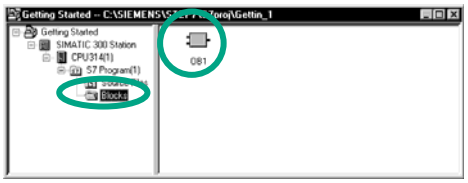
+または–をクリックして、フォルダを開いたり、閉じたりします。

他のファンクションを後で開始する場合は、右側のウィンドウに表示されるシンボルをクリックします。

S7 Program (1)フォルダをクリックします。このフォルダには、必要なプログラムコンポーネントすべてが入っています。

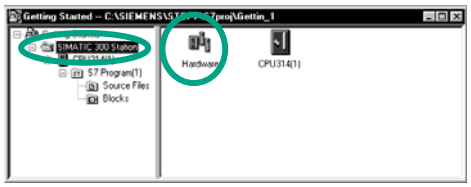
第 3 章では、シンボルコンポーネントを使ってアドレスシンボル名を指定します。

ソースファイルプログラムの保存には、ソースファイルコンポーネントを使用します。このコンポーネントは、本書では扱いません。



Blocks フォルダをクリックします。このフォルダには、作成した **OB1** が入っていますが、ブロックはすべてこのフォルダに格納されます。

ここで、ラダーロジック、ステートメントリスト、ファンクションブロックダイアグラムでプログラミングを開始します(第 4 章と第 5 章)。



SIMATIC 300 Station フォルダをクリックします。すべてのハードウェア関連プロジェクトデータがこのフォルダに入っています。

第 6 章のハードウェアコンポーネントを使用して、プログラマブルコントローラのパラメータを指定します。

オートメーションタスク用に追加の SIMATIC ソフトウェア、たとえばオプションパッケージ PLCSIM (ハードウェアシミュレーションプログラム)や S7 Graph (グラフィックプログラム言語)が必要な場合、これらも STEP 7 に統合されます。たとえば SIMATIC Manager を使用すると、S7 Graph ファンクションブロックなどの関連オブジェクトを直接開くことができます。

詳細は、[**ヘルプ|目次**]の「オートメーションの概念の理解」と「プログラム構造設計の基本」を参照してください。

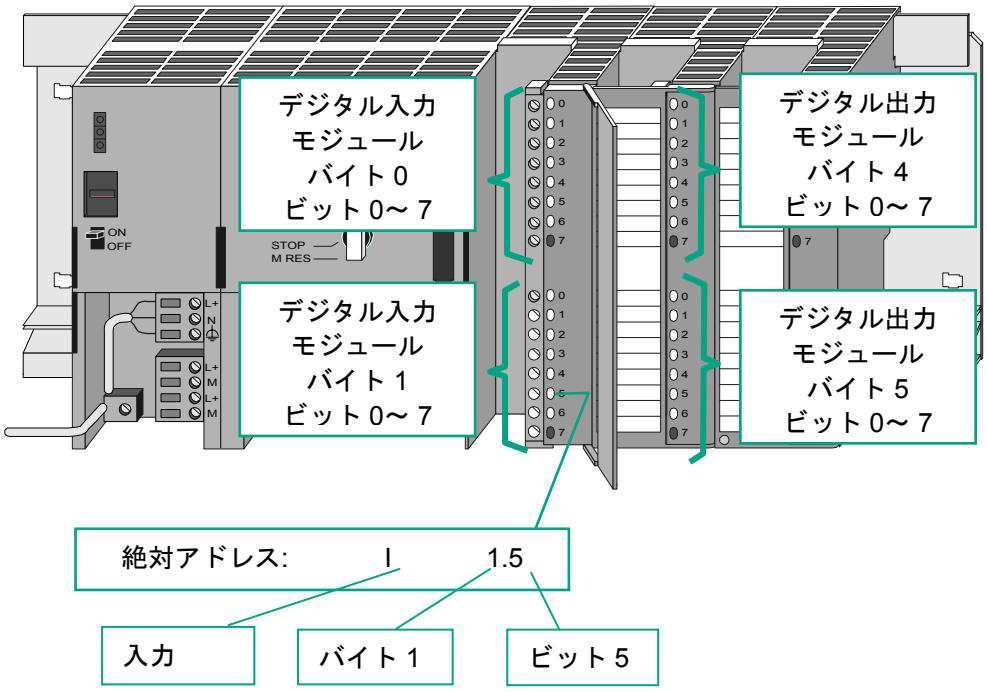
オプションパッケージの詳細については、SIMATIC カタログ ST 70 の「完全統合オートメーション用のコンポーネント」を参照してください。

3　シンボルによるプログラミング

3.1　絶対アドレス

入力と出力には、ハードウェアコンフィグレーションによりあらかじめ絶対アドレスが割り付けられます。このアドレスは、直接 (つまり絶対形式で) 指定されます。

絶対アドレスは、ユーザが選択したシンボル名に置き換えることができます。



S7 プログラムでアドレス指定する入力と出力がそれほど多くない場合は、絶対プログラミングを使用することをお勧めします。

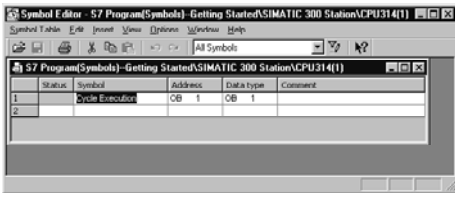
3.2 シンボルによるプログラミング

シンボルテーブルでは、すべての絶対アドレスにシンボル名とデータタイプを割り付けます。このアドレスは、後でプログラムで指定することになり、たとえば、入力 I 0.1 のシンボル名は Key 1 のようになります。この名前は、プログラム全体に適用され、グローバル変数と呼ばれています。

シンボルによるプログラミングを使用すれば、作成した S7 プログラムが格段に読みやすくなります。

シンボルエディタの使用

プロジェクトウィンドウ "Getting Started" で、**S7 Program (1)** へ移動してダブルクリックし、**Symbols** コンポーネントを開きます。



シンボルテーブルは、この段階では定義済みのオーガニゼーションブロック OB1 で構成されています。

	Status	Symbol	Address	Data type
1		Cycle Execution	OB 1	OB 1
2				

[**周期的に実行**] をクリックし、"Main Program" で上書きします。

	Status	Symbol	Address	Data type
1		Main Program	OB 1	OB 1
2		Green Light	Q 4.0	BOOL

行 2 に "Green Light" および "Q 4.0" と入力します。データタイプは自動的に追加されます。

Comment

行 1 または 2 のコメント列をクリックし、シンボルに関するコメントを入力します。**Enter** を押して行の入力を完了します。新しい行が追加されます。

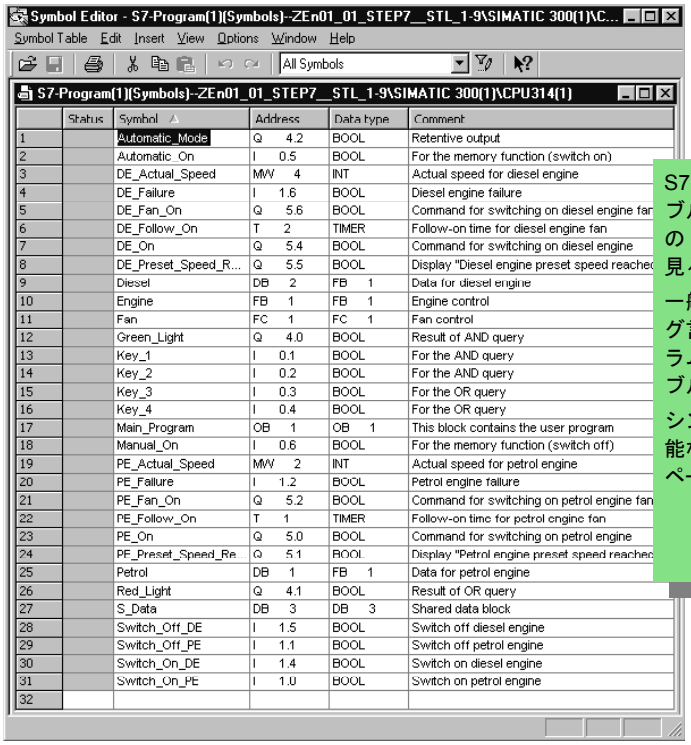
	Status	Symbol	Address	Data type
1		Main Program	OB 1	OB 1
2		Green Light	Q 4.0	BOOL
3		Red Light	Q 4.1	BOOL

行 3 に "Red Light" および "Q 4.1" と入力し、Enter を押して入力を完了します。

上記の方法で、プログラムで必要な入力と出力の絶対アドレスにシンボル名を割り付けることができます。

入力内容と変更内容をシンボルテーブルに保存し、ウィンドウを閉じます。

"Getting Started"プロジェクト全体では多数の名前があるため、第 4 章 1 節で"Getting Started"プロジェクトにシンボルテーブルをコピーしてもかまいません。



	Status	Symbol	Address	Data type	Comment
1		Automatic_Mode	Q 4.2	BOOL	Retentive output
2		Automatic_On	I 0.5	BOOL	For the memory function (switch on)
3		DE_Actual_Speed	MW 4	INT	Actual speed for diesel engine
4		DE_Failure	I 1.6	BOOL	Diesel engine failure
5		DE_Fan_On	Q 5.6	BOOL	Command for switching on diesel engine fan
6		DE_Follow_On	T 2	TIMER	Follow-on time for diesel engine fan
7		DE_On	Q 5.4	BOOL	Command for switching on diesel engine
8		DE_Preset_Speed_R...	Q 5.5	BOOL	Display "Diesel engine preset speed reached
9		Diesel	DB 2	FB 1	Data for diesel engine
10		Engine	FB 1	FB 1	Engine control
11		Fan	FC 1	FC 1	Fan control
12		Green_Light	Q 4.0	BOOL	Result of AND query
13		Key_1	I 0.1	BOOL	For the AND query
14		Key_2	I 0.2	BOOL	For the AND query
15		Key_3	I 0.3	BOOL	For the OR query
16		Key_4	I 0.4	BOOL	For the OR query
17		Main_Program	OB 1	OB 1	This block contains the user program
18		Manual_On	I 0.6	BOOL	For the memory function (switch off)
19		PE_Actual_Speed	MW 2	INT	Actual speed for petrol engine
20		PE_Failure	I 1.2	BOOL	Petrol engine failure
21		PE_Fan_On	Q 5.2	BOOL	Command for switching on petrol engine fan
22		PE_Follow_On	T 1	TIMER	Follow-on time for petrol engine fan
23		PE_On	Q 5.0	BOOL	Command for switching on petrol engine
24		PE_Preset_Speed_Re	Q 5.1	BOOL	Display "Petrol engine preset speed reached
25		Petrol	DB 1	FB 1	Data for petrol engine
26		Red_Light	Q 4.1	BOOL	Result of OR query
27		S_Data	DB 3	DB 3	Shared data block
28		Switch_Off_DE	I 1.5	BOOL	Switch off diesel engine
29		Switch_Off_PE	I 1.1	BOOL	Switch off petrol engine
30		Switch_On_DE	I 1.4	BOOL	Switch on diesel engine
31		Switch_On_PE	I 1.0	BOOL	Switch on petrol engine
32					

S7 プログラムのシンボルテーブルは、ステートメントリストの "Getting Started" サンプルで見ることができます。

一般に、選択したプログラミング言語に関係なく、S7 プログラム 1 つに対してシンボルテーブル 1 つを作成します。

シンボルテーブルでは、印刷可能なすべての文字 (特殊文字やスペースなど) を使用できます。

シンボルテーブルに自動的に追加されたデータタイプにより、CPU で処理される信号のタイプが決まります。STEP 7 では、次のデータタイプを使用します。

BOOL BYTE WORD DWORD	このタイプのデータは、ビットで構成される。1 ビット (タイプ BOOL) ～ 32 ビット (DWORD)
CHAR	このタイプのデータは、ASCII 文字セットの 1 文字になる
INT DINT REAL	数値の処理に使用できる(たとえば、計算式の計算など)
S5TIME TIME DATE TIME_OF_DAY	このタイプのデータは、STEP 7 の異なる時刻と日付を表す(たとえば、日付を設定したり、タイマの時刻を入力するなど)

詳細は、[**ヘルプ> 目次**] のトピック「ブロックのプログラミング」および「シンボルの定義」を参照してください。

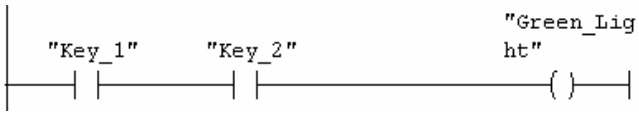
4 OB1 でのプログラムの作成

4.1 LAD/STL/FBD プログラムウィンドウのオープン

ラダーロジック、ステートメントリスト、ファンクションブロックダイアグラムの選択

STEP 7 では標準言語であるラダーロジック(LAD)、ステートメントリスト(STL)、ファンクションブロックダイアグラム(FBD) で S7 プログラムを作成します。この章でもそうですが、通常は使用する言語を決定する必要があります。

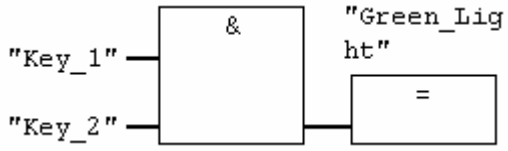
ラダーロジック(LAD)
電気工業関係などのユーザに適している



ステートメントリスト(STL)
コンピュータ関係のユーザに適している

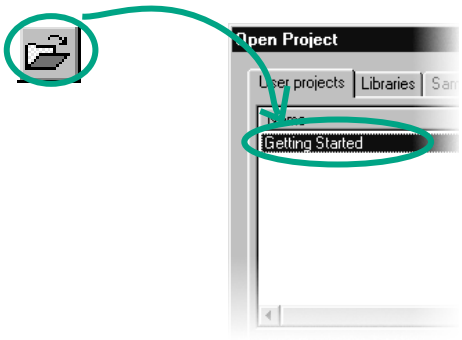
```
A     "Key_1"
A     "Key_2"
=     "Green_Light"
```

ファンクションブロックダイアグラム(FBD)
回路を扱うユーザに適している



ブロック OB1 は、プロジェクトウィザードでプロジェクトを作成したときに選択した言語で開きます。 ただし、デフォルトのプログラミング言語はいつでも変更することができます。

シンボルテーブルのコピーおよび OB1 のオープン



"Getting Started"プロジェクトは、必要に応じて開くことができます。それには、ツールバーの[**開く**]ボタンをクリックし、作成した"Getting Started"プロジェクトを選択し、[**OK**] で確定します。

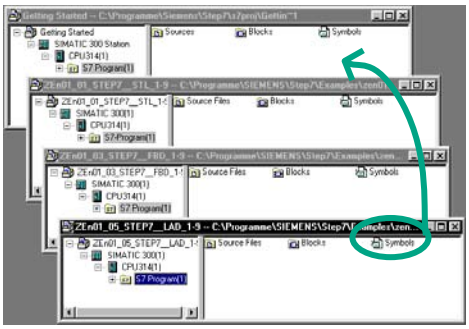
使用するプログラム言語に応じて、[サンプルプロジェクト]タブで次のプロジェクトのいずれかを開きます。

- ZEn01_05_STEP7__**LAD**_1-9
- ZEn01_01_STEP7__**STL**_1-9

または

- ZEn01_03_STEP7__**FDB**_1-9

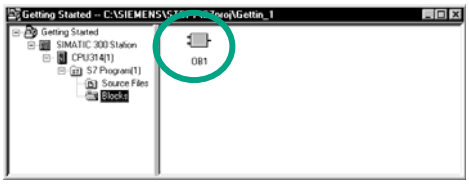
ここでは、3 つのサンプルプロジェクトすべてを表示することができます。



"ZEn01_XXX"で **Symbols** コンポーネントへ移動し、これを"Getting Started"プロジェクトウィンドウの **S7 Program** フォルダへドラッグ&ドロップしてコピーします。

ウィンドウ"ZEn01_XXX“を閉じます。

ドラッグ＆ドロップは、オブジェクトをマウスでクリックし、マウスボタンを押したまま移動させることです。選択した位置でマウスボタンを放すと、そのオブジェクトがペーストされます。



"Getting Started"プロジェクトの **OB1** をダブルクリックします。LAD/STL/ FBD プログラムウィンドウが開きます。

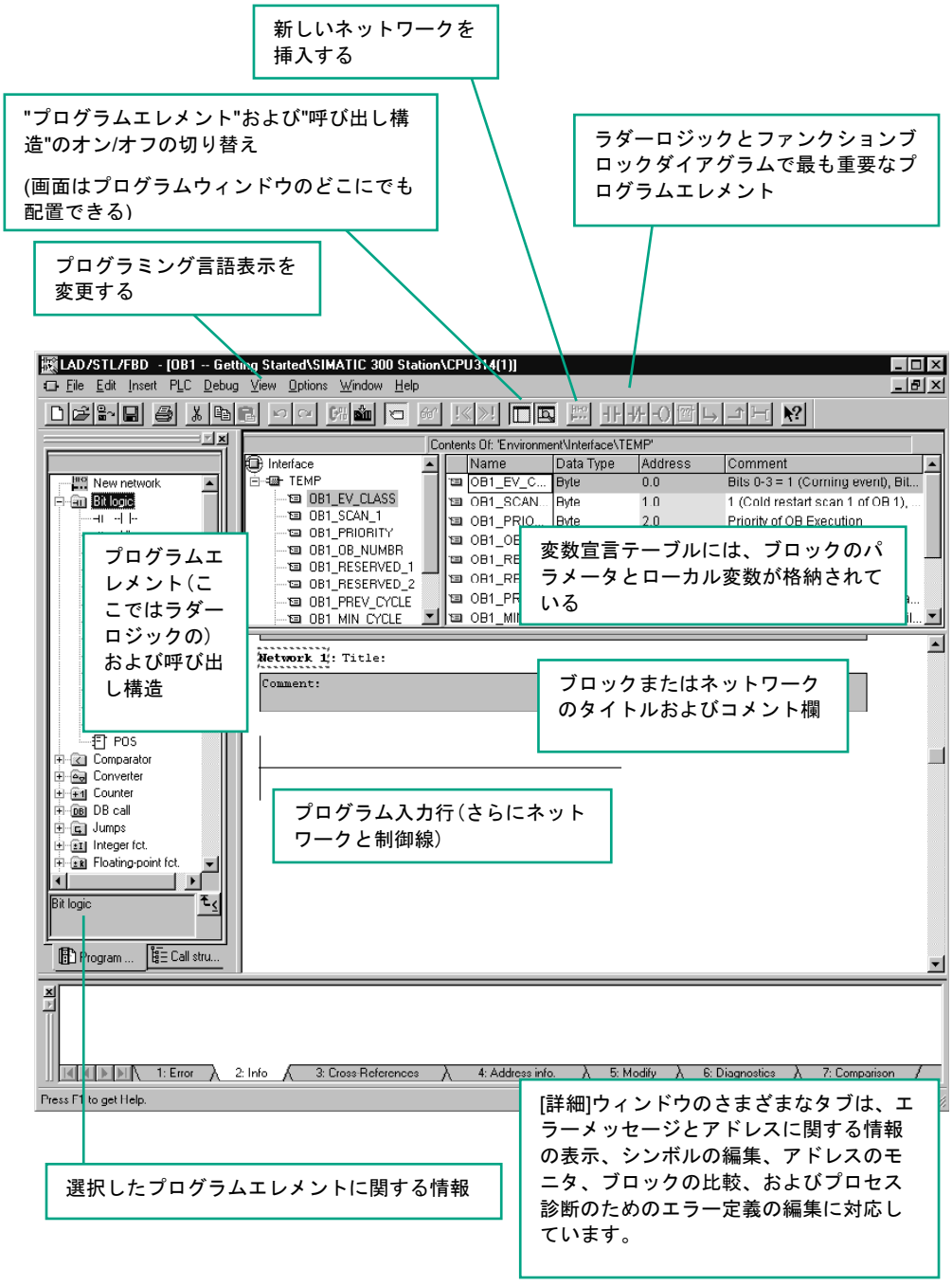
STEP 7 では、OB1 は CPU によって周期的に処理されます。CPU は 1 行ずつ読み取り、プログラムコマンドを実行します。CPU がプログラムの先頭行に戻ると、1 サイクルが完了します。1 サイクルの所要時間をスキャンサイクルタイムと呼びます。

ラダーロジックを選択した場合は第 4 章 2 節を、ステートメントリストは第 4 章 3 節を、ファンクションブロックダイアグラムは第 4 章 4 節をお読みください。

詳細は、[**ヘルプ|目次**]の「ブロックのプログラミング」および「ブロックおよびライブラリの作成」を参照してください。

LAD/STL/FBD プログラムウィンドウ

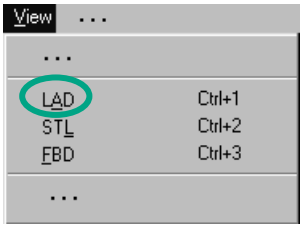
ブロックのプログラミングは LAD/STL/FBD プログラムウィンドウで行います。ここではラダーロジックの画面を示します。



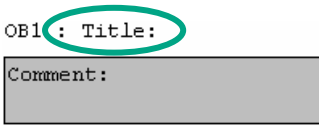
4.2 ラダーロジックによる OB1 のプログラミング

次の節では、直列回路、並列回路、セット/リセットメモリファンクションをラダーロジック (LAD) でプログラミングします。

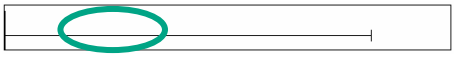
ラダーロジックによる直列回路のプログラミング



必要に応じて、[**表示**]メニューで **LAD** をプログラミング言語に設定します。



OB1 の[**タイトル**]領域をクリックし、「周期的に処理されるメインプログラム」のように入力します。



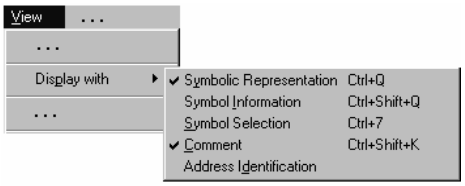
最初のエレメントの制御線を選択します。

ツールバーのボタンをクリックし、a 接点を挿入します。

同様の方法で、2 つ目の a 接点を挿入します。

制御線の右端にコイルを挿入します。

直列回路には a 接点とコイルのアドレスが示されていません。

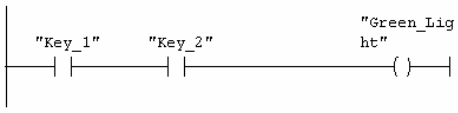


シンボル表示が有効になっていることを確認します。

??.?をクリックし、シンボル名"Key_1"を入力します(引用符で囲みます)。代わりに、表示されるプルダウンリストから名前を選択できます。**Enter** を押して確定します。

2 番目の a 接点にシンボル名"Key_2"を入力します。

コイルに"Green_Light"と入力します。



これで、直列回路のプログラミングが完了しました。

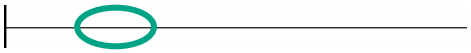
赤で表示されているシンボルがなければ、ブロックを保存します。

シンボルテーブルにないシンボルや、構文エラーがあるシンボルは赤で表示されます。

ラダーロジックによる並列回路のプログラミング

Network 1 を選択します。

新しいネットワークを挿入します。

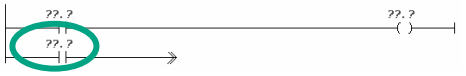


制御線を再度選択します。

a 接点とコイルを挿入します。

制御線の縦線を選択します。

並列ブランチを挿入します。



並列ブランチに a 接点を追加します。

ブランチを閉じます(必要ならば、下部の矢印を選択する)。

並列回路にはアドレスが表示されていません。

直列回路と同じ方法で、シンボルアドレスを割り付けます。

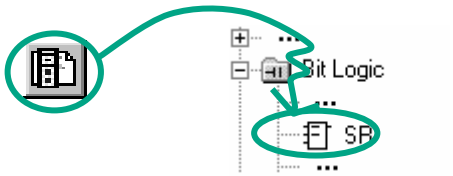
上の a 接点を"Key_3"で上書きし、下の a 接点を"Key_4"で上書きし、コイルを"Red_Light"で上書きします。

ブロックを保存します。

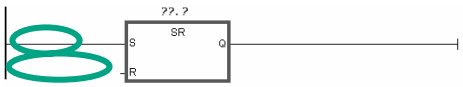
ラダーロジックによるメモリファンクションのプログラミング

Network 2 を選択し、別のネットワークを挿入します。

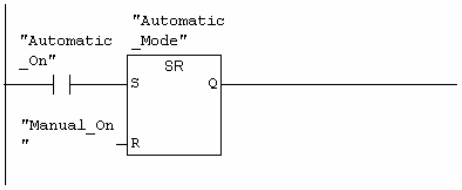
制御線を再度選択します。



Bit Logic のプログラムエレメントカタログで **SR** エレメントへ移動します。このエレメントをダブルクリックして、挿入します。



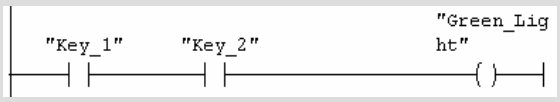
入力 S と R の前にそれぞれ a 接点を挿入します。



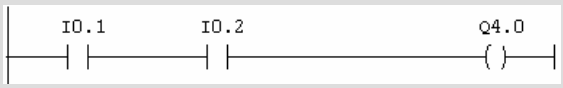
SR エレメントに次のシンボル名を入力します。
上の接点には"Automatic_On"
下の接点には"Manual_On"
SR エレメントには"Automatic_Mode"

ブロックを保存し、ウィンドウを閉じます。

絶対アドレス指定とシンボルアドレス指定の違いを確認したい場合は、メニューコマンド[**表示|表示|シンボル表現**]を無効にしてください。



例:
LAD のシンボルアドレス指定:



例:
LAD の絶対アドレス指定

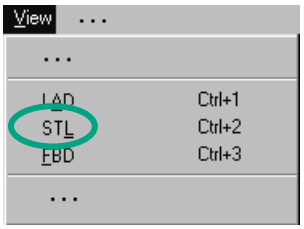
LAD/STL/FBD プログラムウィンドウでは、シンボルアドレス指定の改行を変更できます。それには、メニューコマンド[**オプション|カスタマイズ**]を使って、[LAD/FBD]タブの[アドレス欄の幅]を選択します。 ここで、10～26 文字の間で改行を設定できます。

詳細は、[**ヘルプ|目次**]の「ブロックのプログラミング」、「ロジックブロックの作成」、「ラダー命令の編集」を参照してください。

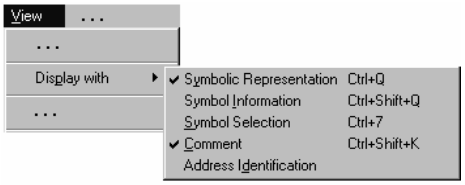
4.3 ステートメントリストによる OB1 のプログラミング

次の節では、AND 命令、OR 命令、メモリ命令セット/リセットをステートメントリスト (STL) でプログラミングします。

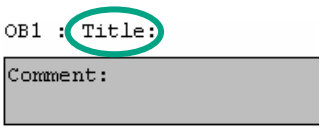
ステートメントリストによる AND 命令のプログラミング



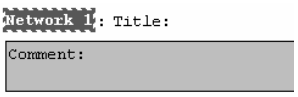
必要に応じて、[**表示**]メニューで **STL** をプログラミング言語に設定します。



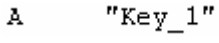
シンボル表示が有効になっていることを確認します。



OB1 の[**タイトル**]領域をクリックし、「周期的に処理されるメインプログラム」のように入力します。



最初のステートメントの領域を選択します。



プログラムの先頭行で A (AND)、スペースを入力し、次にシンボル名 "Key_1" (引用符で囲む)を入力します。

Enter を押してその行を完了します。カーソルが次の行へジャンプします。

```
A     "Key_1"
A     "Key_2"
=     "Green_Light"
```

同様の方法で、AND 命令を完成させます。右記のようにします。

これで AND 命令のプログラミングが完了しました。赤で表示されているシンボルがなければ、ブロックを保存します。

シンボルテーブルにないシンボルや、構文エラーがあるシンボルは赤で表示されます。

シンボル名は、シンボルテーブルから直接挿入することもできます。**??.?**をクリックし、メニューコマンド[**挿入 |シンボル**]を選択します。プルダウンリストをスクロールし、目的の名前が見つかったらそれを選択します。シンボル名が自動的に追加されます。

ステートメントリストによる OR 命令のプログラミング

Network 1 を選択します。

新しいネットワークを挿入し、入力領域を再度選択します。

```
O     "Key_3"
```

O (OR)とシンボル名"Key_3" を入力します(AND 命令と同じ方法を使用する)。

```
O     "Key_3"
O     "Key_4"
=     "Red_Light"
```

OR 命令を完成させて、保存します。

ステートメントリストによるメモリ命令のプログラミング

Network 2 を選択し、別のネットワークを挿入します。

```
A       "Automatic_On"
```

先頭行に、命令 A とシンボル名 "Automatic_On"を入力します。

```
A       "Automatic_On"
S       "Automatic_Mode"
A       "Manual_On"
R       "Automatic_Mode"
```

メモリ命令を完成させて、保存します。ブロックを閉じます。

絶対アドレス指定とシンボルアドレス指定の違いを確認したい場合は、メニューコマンド[**表示|表示|シンボル表現**]を無効にしてください。

```
A       "Key_1"
A       "Key_2"
=       "Green_Light"
```

例:
STL のシンボルアドレス指定

```
A       I       0.1
A       I       0.2
=       Q       4.0
```

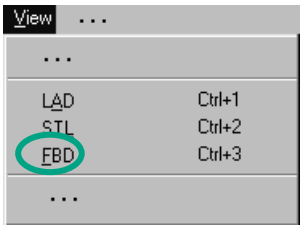
例:
STL の絶対アドレス指定

詳細は、[**ヘルプ|目次**]の「ブロックのプログラミング」、「ロジックブロックの作成」、「STL ステートメントの編集」を参照してください。

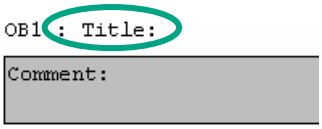
4.4 ファンクションブロックダイアグラムによる OB1 のプログラミング

次の節では、AND ファンクション、OR ファンクション、メモリファンクションをファンクションブロックダイアグラム (FBD) でプログラミングします。

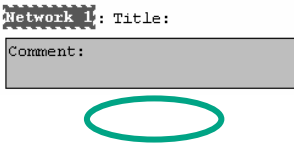
ファンクションブロックダイアグラムによる AND ファンクションのプログラミング



必要に応じて、[**表示**]メニューで **FBD** をプログラミング言語に設定します。

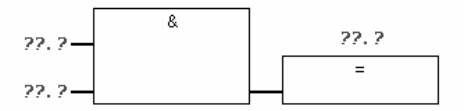


OB1 の[**タイトル**]領域をクリックし、「周期的に処理されるメインプログラム」のように入力します。

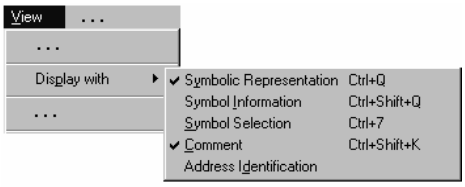


AND ファンクションの入力領域を選択します(コメント欄の下)。

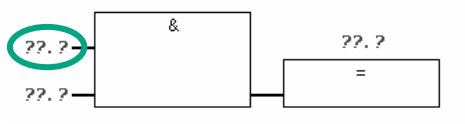
AND ボックス (&) と代入 (=) を挿入します。



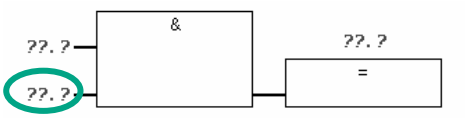
AND ファンクションにエレメントのアドレスは表示されていません。



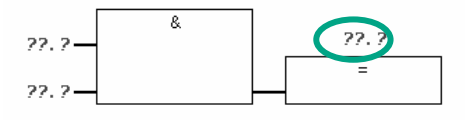
シンボル表示が有効になっていることを確認します。



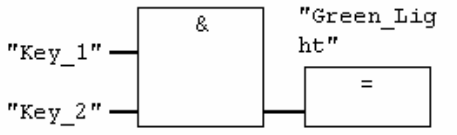
??.?をクリックし、 シンボル名 "Key_1"を入力します(引用符で囲みます)。代わりに、表示されるプルダウンリストから名前を選択できます。
Enter を押して確定します。



2 番目の入力にシンボル名"Key_2"を入力します。



代入に"Green_Light"を入力します。



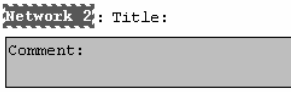
これで AND ファンクションのプログラミングが完了しました。

赤で表示されているシンボルがなければ、ブロックを保存します。

シンボルテーブルにないシンボルや、構文エラーがあるシンボルは赤で表示されます。

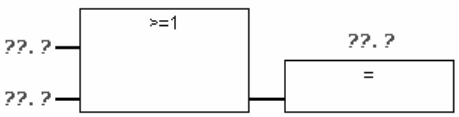
ファンクションブロックダイアグラムによる OR ファンクションのプログラミング

新しいネットワークを挿入します。

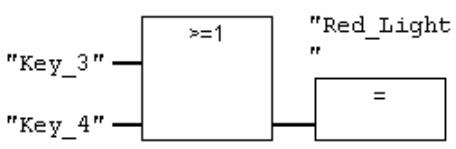


OR ファンクションの入力領域を再度選択します。

OR ボックス (≥1) と代入 (=) を挿入します。



OR ファンクションにはアドレスが表示されていません。AND ファンクションと同じ方法を実行します。

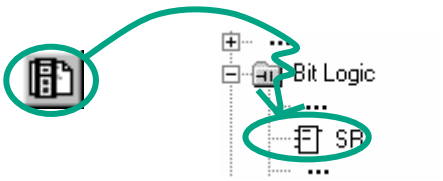


上の入力に"Key_3"を、下の入力に"Key_4"を、代入に"Red_Light"と入力します。

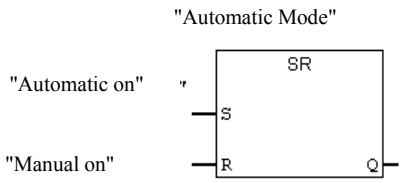
ブロックを保存します。

ファンクションブロックダイアグラムによるメモリファンクションのプログラミング

Network 2 を選択し、別のネットワークを挿入します。入力領域(コメント欄の下)を再度選択します。



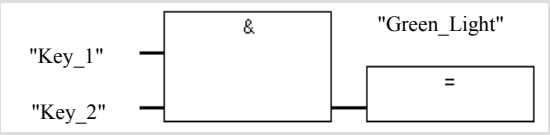
Bit Logic のプログラムエレメントカタログで **SR** エレメントへ移動します。このエレメントをダブルクリックして、挿入します。



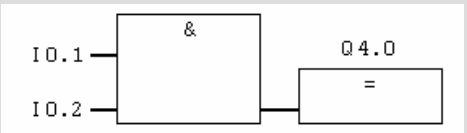
SR エレメントに次のシンボル名を入力します。
セットには"Automatic_On"リセットには"Manual_On"メモリビットには"Automatic_Mode"

ブロックを保存し、ウィンドウを閉じます。

絶対アドレス指定とシンボルアドレス指定の違いを確認したい場合は、メニューコマンド[**表示|表示|シンボル表現**]を無効にしてください。



例:
FBD のシンボルアドレス指定



例:
FBD の絶対アドレス指定

LAD/STL/FBD プログラムでは、シンボルアドレス指定の改行を変更できます。それには、メニューコマンド[**オプション|カスタマイズ**]を使って、[LAD/FBD]タブの[アドレス欄の幅]を選択します。ここで、10～26 文字の間で改行を設定できます。

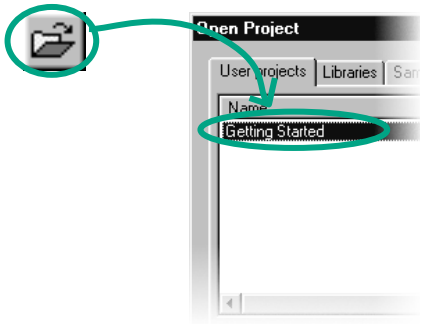
詳細は、[**ヘルプ|目次**]の「ブロックのプログラミング」、「ロジックブロックの作成」、「FBD ステートメントの編集」を参照してください。

5 ファンクションブロックとデータブロックによるプログラムの作成

5.1 ファンクションブロック(FB)の作成およびオープン

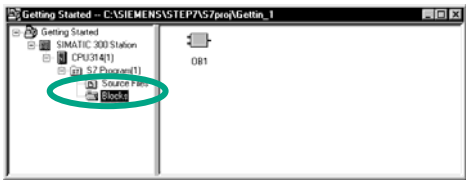
ファンクションブロック(FB)は、プログラム階層でオーガニゼーションブロックの下にあります。これには、OB1 で何度も呼び出すことができるプログラムの一部が含まれます。ファンクションブロックのすべての仮パラメータとスタティックデータは別々のデータブロック(DB)に保存され、このデータブロックはファンクションブロックに割り付けられます。

ファンクションブロック(FB1、シンボル名"Engine"、3-3 ページのシンボルテーブルを参照)は、よく知られている LAD/STL/FBD プログラムウィンドウでプログラミングします。プログラミングには、第 4 章 (OB1 のプログラミング) と同じプログラミング言語を使用します。



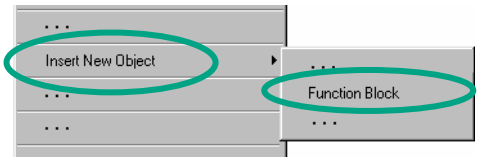
シンボルテーブルは、既に"Getting Started"プロジェクトにコピーされている必要があります。コピーされていない場合は、4-2 ページ「シンボルテーブルのコピー」で対処方法を読んでからこの節に戻ってください。

必要に応じて、"Getting Started" プロジェクトを開きます。

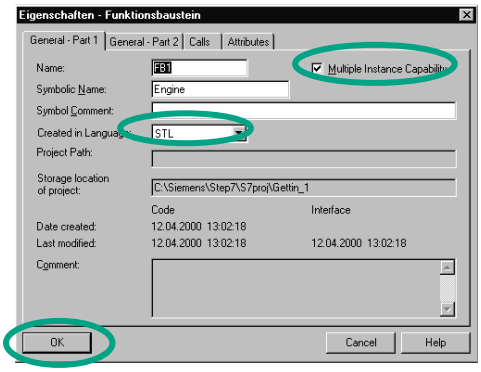


Blocks フォルダへ移動し、このフォルダを開きます。

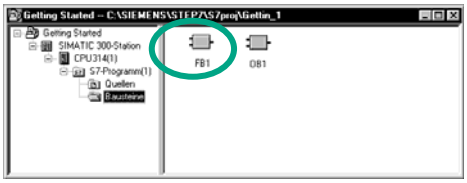
ウィンドウの右側をマウスの右ボタンでクリックします。



ポップアップメニューが表示され、メニューバーの最も重要なコマンドが示されます。**ファンクションブロック**を新規オブジェクトとして挿入します。



[プロパティ - ファンクションブロック]ダイアログボックスで、ブロックの作成に使用する言語を選択し、チェックボックス**[マルチプルインスタンス FB]**をオンにし、残りの設定を**[OK]**で確定します。



ファンクションブロック **FB1** が Blocks フォルダに挿入されます。

FB1 をダブルクリックして、LAD/STL/FBD プログラムウィンドウを開きます。

選択したプログラミング言語がラダーロジックの場合は第 5 章 2 節を、ステートメントリストは第 5 章 3 節を、ファンクションブロックダイアグラムは第 5 章 4 節をお読みください。

詳細は、**[ヘルプ|目次]**の「ブロックのプログラミング」および「ブロックおよびライブラリの作成」を参照してください。

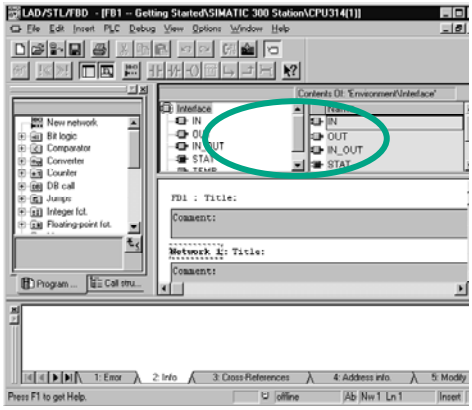
5.2 ラダーロジックによる FB1 のプログラミング

ここでは、ガソリンエンジンやディーゼルエンジンの制御と監視を行うファンクションブロックを 2 つのデータブロックを使ってプログラミングする方法を説明します。

「エンジン固有」の信号はすべてブロックパラメータとしてオーガニゼーションブロックからファンクションブロックへ渡されるため、入力/出力パラメータ (宣言 "in" と "out") として変数宣言テーブルにリストする必要があります。

直列回路、並列回路、メモリファンクションを STEP 7 で入力する方法については、すでに説明しました。

変数の宣言/定義 最初



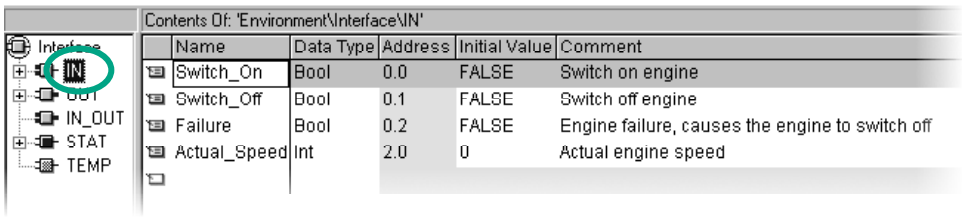
LAD/STL/FBD プログラムウィンドウが開き、オプション**[表示|LAD]** (プログラム言語)が有効になっています。

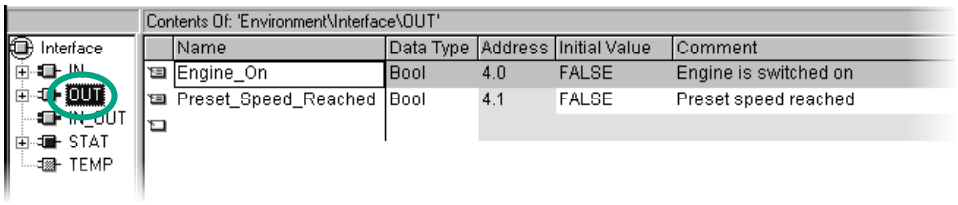
FB1 をダブルクリックしてプログラムウィンドウを開いたため、FB1 がヘッダーに表示されています。

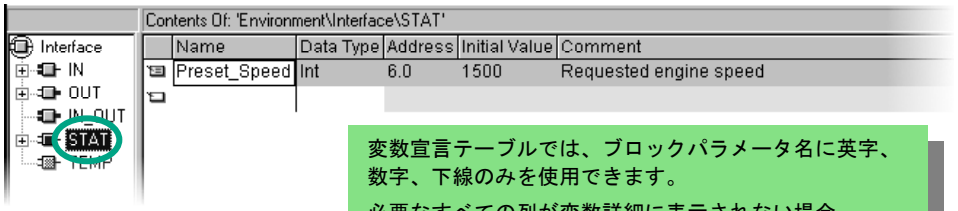
変数宣言領域は、変数概要(左の画面)および変数詳細ビュー(右の画面)から構成されます。

変数概要で宣言タイプ"IN"、"OUT"、および"STAT"を逐次選択し、その後の宣言を変数詳細に入力します。

変数概要で対応するセルをクリックし、その後の図からエントリを適用します。表示されるプルダウンリストからデータタイプを選択できます。



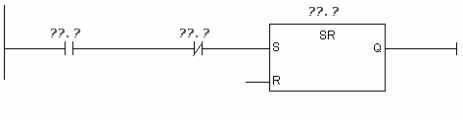




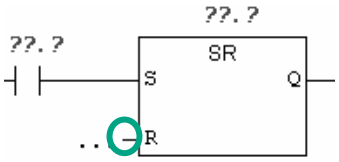
変数宣言テーブルでは、ブロックパラメータ名に英字、数字、下線のみを使用できます。

必要なすべての列が変数詳細に表示されない場合、ショートカットメニューコマンド(マウスを右クリック)を使用して表示することができます。

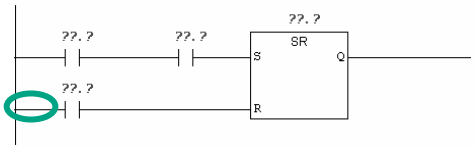
エンジンの始動/停止の切り替えのプログラミング



Network 1 に a 接点、b 接点、SR エレメントを直列で挿入します。それには、ツールバーの対応するボタンか、プログラムエレメントカタログを使用します。

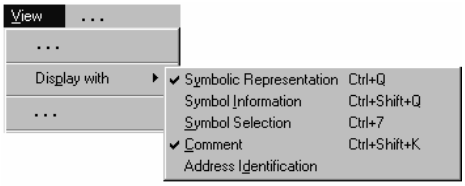


入力 R の前の制御線を選択します。



別の a 接点を挿入します。この接点の前の制御線を選択します。

b 接点を a 接点に並列に挿入します。

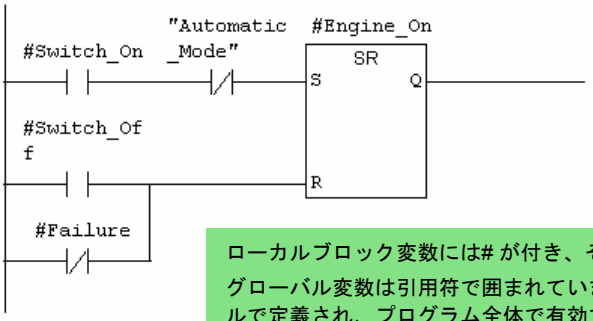


シンボル表示が有効になっていることを確認します。

疑問符を選択し、変数宣言テーブルから対応する名前を入力します (# 記号が自動的に割り付けられる)。

b 接点のシンボル名"Automatic_Mode" を直列回路に挿入します。

プログラムを保存します。

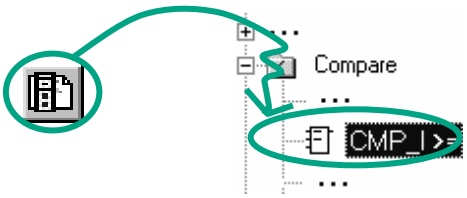


ローカルブロック変数には# が付き、そのブロックでのみ有効です。

グローバル変数は引用符で囲まれています。 この変数は、シンボルテーブルで定義され、プログラム全体で有効です。

信号状態"Automatic_Mode"は別の SR エレメントにより OB1 に定義され (Network 3、4-7 ページを参照)、FB1 で紹介されます。

速度監視のプログラミング



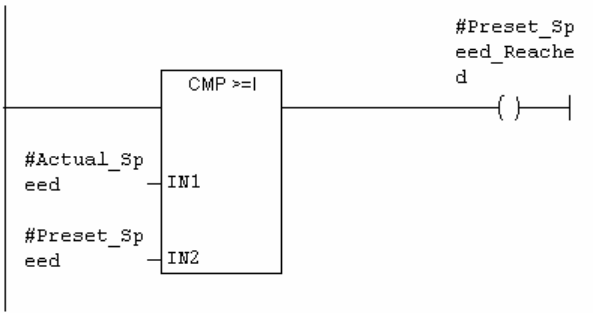
新しいネットワークを挿入し、制御線を選択します。

プログラムエレメントカタログで **Compare** ファンクションへ移動し、**CMP>=I** を挿入します。

さらに制御線にコイルを挿入します。

疑問符を選択し、コイルとコンパレータに変数宣言テーブルの名前を入力します。

プログラムを保存します。



エンジンはいつ始動/停止するか？

変数#Switch_On の信号状態が"1" になり、同時に変数"Automatic_Mode"が"0"になると、エンジンが始動します。このファンクションは、"Automatic_Mode"が否定される(b 接点)まで有効になりません。

変数#Switch_Off が"1"になるか、または変数#Fault が"0"になると、エンジンは停止します。このファンクションは、#Fault (#Fault は"アクティブゼロ"信号で、正常な状態では"1"になり、エラーが発生すると"0"になる) が否定されると、再度有効になります。

コンパレータはどのような方法でエンジン速度を監視するか？

コンパレータは変数#Actual_Speed と#Setpoint_Speed を比較し、その結果を #Setpoint_Speed_Reached (信号状態"1") に割り付けます。

詳細は、**[ヘルプ|目次]**の「ブロックのプログラミング」、「ロジックブロックの作成」、「変数宣言の編集」を参照するか、「LAD 命令の編集」を参照してください。

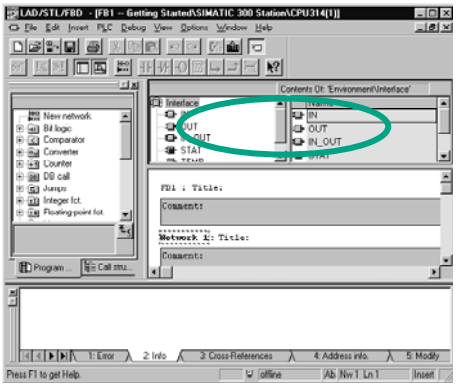
5.3 ステートメントリストによる FB1 のプログラミング

ここでは、ガソリンエンジンやディーゼルエンジンの制御と監視を行うファンクションブロックを 2 つのデータブロックを使ってプログラミングする方法を説明します。

「エンジン固有」の信号はすべてブロックパラメータとしてオーガニゼーションブロックからファンクションブロックへ渡されるため、入力/出力パラメータ (宣言"in"と"out") として変数宣言テーブルにリストする必要があります。

AND 命令、OR 命令、セット/リセットメモリ命令を STEP 7 で入力する方法については、すでに説明しました。

変数の宣言/定義 最初



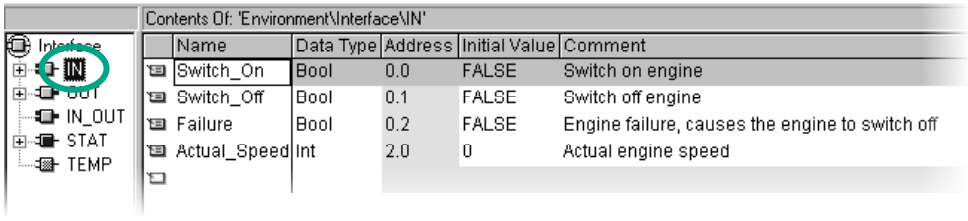
LAD/STL/FBD プログラムウィンドウが開き、オプション**[表示|STL]**(プログラム言語)が有効になっています。

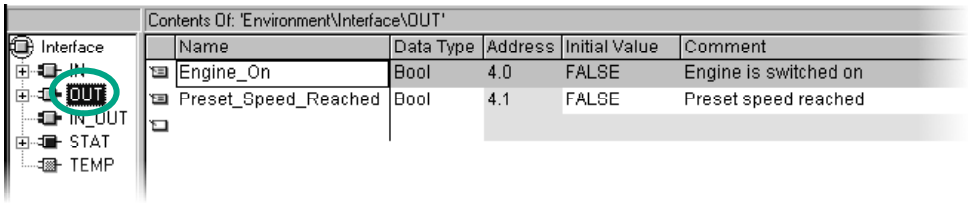
FB1 をダブルクリックしてプログラムウィンドウを開いたため、FB1 がヘッダーに表示されています。

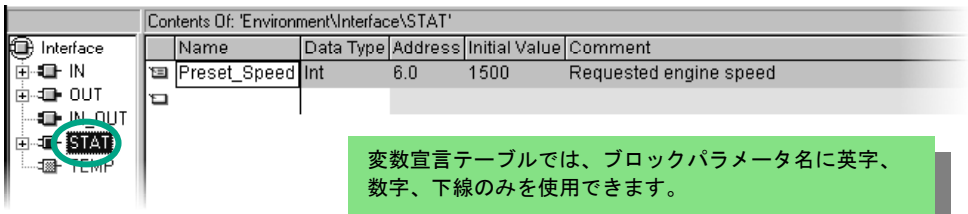
変数宣言領域は、変数概要(左の画面)および変数詳細ビュー(右の画面)から構成されます。

変数概要で宣言タイプ"IN"、"OUT"、および"STAT"を逐次選択し、その後の宣言を変数詳細に入力します。

変数概要で対応するセルをクリックし、その後の図からエントリを適用します。表示されるプルダウンリストからデータタイプを選択できます。

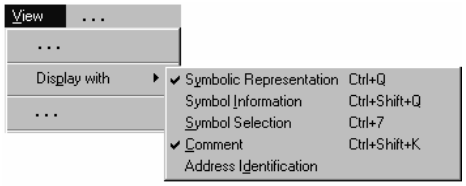






変数宣言テーブルでは、ブロックパラメータ名に英字、数字、下線のみを使用できます。

エンジンの始動/停止の切り替えのプログラミング



シンボル表示が有効になっていることを確認します。

```
A     #Switch_On
AN    "Automatic_Mode"
S     #Engine_On
O     #Switch_Off
ON    #Failure
R     #Engine_On
```

Network 1 に対応する命令を入力します。

ローカルブロック変数には#が付き、そのブロックでのみ有効です。

グローバル変数は引用符で囲まれています。この変数は、シンボルテーブルで定義され、プログラム全体で有効です。

信号状態"Automatic_Mode"は別の SR エレメントにより OB1 に定義され(Network 3、4-10 ページを参照)、FB1 で紹介されます。

速度監視のプログラミング

```
L     #Actual_Speed
L     #Preset_Speed
>=I
=     #Preset_Speed_Reached
```

新しいネットワークを挿入し、対応する命令を入力します。プログラムを保存します。

エンジンはいつ始動/停止するか？

変数#Switch_On の信号状態が"1" になり、同時に変数"Automatic_Mode"が"0"になると、エンジンが始動します。このファンクションは、"Automatic_Mode" が否定される (b 接点) まで有効になりません。

変数#Switch_Off が"1"になるか、または変数#Fault が"0"になると、エンジンは停止します。このファンクションは、#Fault (#Fault は"アクティブゼロ"信号で、正常な状態では"1" になり、エラーが発生すると"0"になる) が否定されると、再度有効になります。

コンパレータはどのような方法でエンジン速度を監視するか？

コンパレータは変数#Actual_Speed と#Setpoint_Speed を比較し、その結果を #Setpoint_Speed_Reached (信号状態"1") に割り付けます。

詳細は、**[ヘルプ|目次]**の「ブロックのプログラミング」、「ロジックブロックの作成」、「変数宣言の編集」を参照するか、「STL 命令の編集」を参照してください。

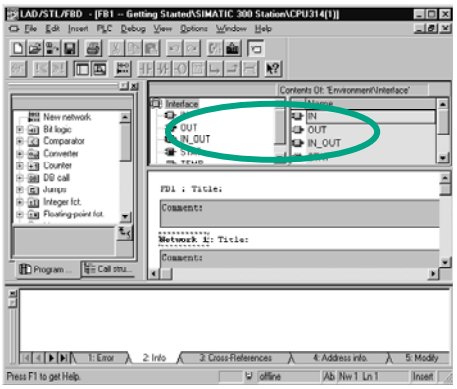
5.4 ファンクションブロックダイアグラムによる FB1 のプログラミング

ここでは、ガソリンエンジンやディーゼルエンジンの制御と監視を行うファンクションブロックを 2 つのデータブロックを使ってプログラミングする方法を説明します。

「エンジン固有」の信号はすべてブロックパラメータとしてオーガニゼーションブロックからファンクションブロックへ渡されるため、入力/出力パラメータ (宣言"in"と"out") として変数宣言テーブルにリストする必要があります。

AND 命令、OR 命令、メモリファンクションを STEP 7 で入力する方法については、すでに説明しました。

変数の宣言/定義 最初



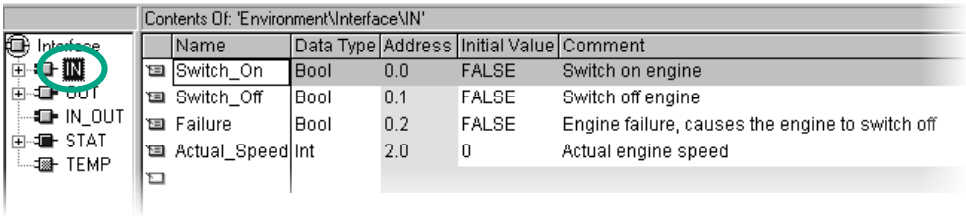
LAD/STL/FBD プログラムウィンドウが開き、オプション**[表示|FBD]** (プログラム言語)が有効になっています。

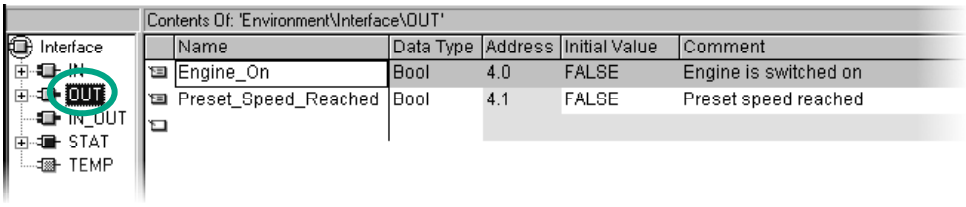
FB1 をダブルクリックしてプログラムウィンドウを開いたため、FB1 がヘッダーに表示されています。

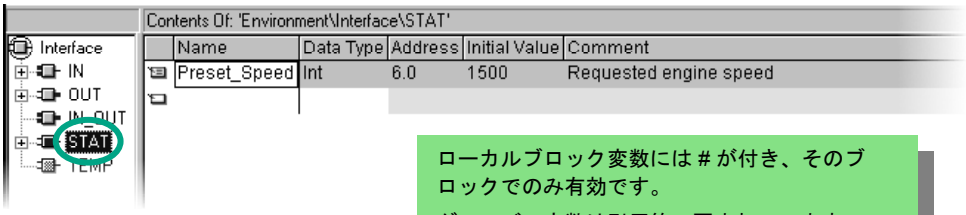
変数宣言領域は、変数概要(左の画面)および変数詳細ビュー(右の画面)から構成されます。

変数概要で宣言タイプ"IN"、"OUT"、および"STAT"を逐次選択し、その後の宣言を変数詳細に入力します。

変数概要で対応するセルをクリックし、その後の図からエントリを適用します。表示されるプルダウンリストからデータタイプを選択できます。



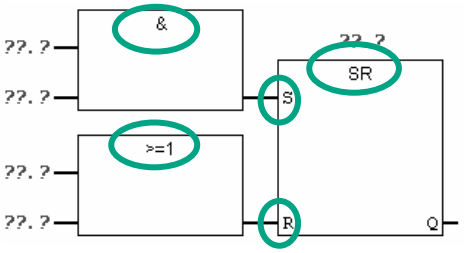




ローカルブロック変数には # が付き、そのブロックでのみ有効です。

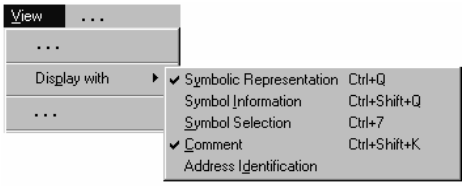
グローバル変数は引用符で囲まれています。この変数は、シンボルテーブルで定義され、プログラム全体で有効です。

エンジンの始動/停止の切り替えのプログラミング



プログラムエレメントカタログ(Bit Logic フォルダ) を使って Network 1 に SR ファンクションを挿入します。

入力 S (Set)に AND ボックスを、入力 R(Reset)に OR ボックスを追加します。



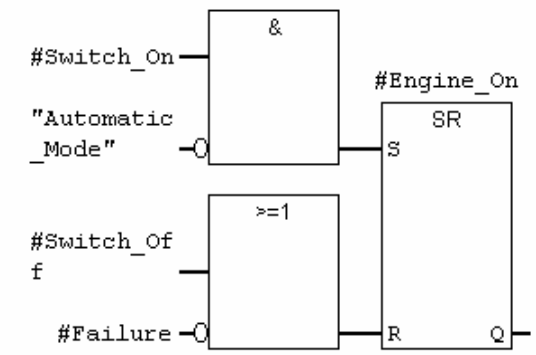
シンボル表示が有効になっていることを確認します。

??.?をクリックし、 変数宣言テーブルから対応する名前を入力します(# 符号が自動的に割り付けられます)。

AND ファンクションの入力の 1 つを必ずシンボル名"Automatic_Mode"でアドレス指定します。

入力"Automatic_Mode"と#Fault をツールバーの対応するボタンで否定します。

プログラムを保存します。

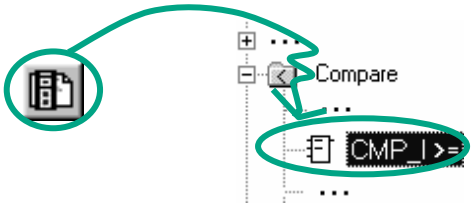


ローカルブロック変数には#が付き、そのブロックでのみ有効です。

グローバル変数は引用符で囲まれています。この変数は、シンボルテーブルで定義され、プログラム全体で有効です。

信号状態"Automatic_Mode"は別の SR エレメントにより OB1 に定義され(Network 3、4-14 ページを参照)、FB1 で紹介されます。

速度監視のプログラミング

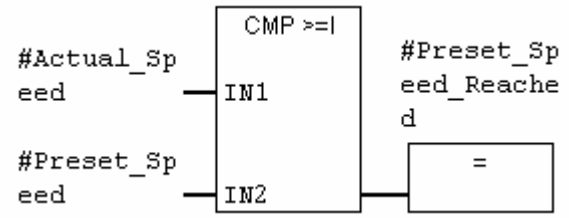


新しいネットワークを挿入し、入力領域を選択します。

プログラムエレメントカタログで **Compare** ファンクションへ移動し、**CMP>=I** を挿入します。

出力割り付けをコンパレータに追加し、変数宣言テーブルの名前で入力をアドレス指定します。

プログラムを保存します。



エンジンはいつ始動/停止するか？

変数#Switch_On の信号状態が"1" になり、同時に変数"Automatic_Mode"が"0"になると、エンジンが始動します。このファンクションは、"Automatic_Mode" が否定される(b 接点)まで有効になりません。

変数#Switch_Off が"1"になるか、または変数#Fault が"0"になると、エンジンは停止します。このファンクションは、#Fault (#Fault は"アクティブゼロ"信号で、正常な状態では"1"になり、エラーが発生すると"0"になる) が否定されると、再度有効になります。

コンパレータはどのような方法でエンジン速度を監視するか？

コンパレータは変数#Actual_Speed と#Setpoint_Speed を比較し、その結果を #Setpoint_Speed_Reached (信号状態"1")に割り付けます。

詳細は、**[ヘルプ|目次]**の「ブロックのプログラミング」、「ロジックブロックの作成」、「変数宣言の編集」を参照するか、「FBD 命令の編集」を参照してください。

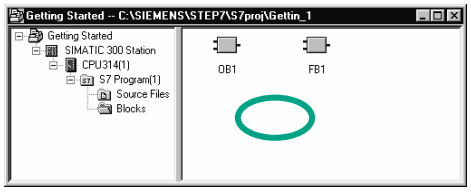
5.5 インスタンスデータブロックの作成および現在値の変更

ファンクションブロック FB1("Engine")のプログラミング、変数宣言テーブルでのエンジン固有パラメータの定義が完了しました。

ファンクションブロックの呼び出しを OB1 でプログラミングできるように、対応するデータブロックを生成する必要があります。インスタンスデータブロック(DB)は常にファンクションブロックに割り付けられます。

このファンクションブロックは、ガソリンエンジンやディーゼルエンジンの制御と監視を行います。エンジンの各セットポイント速度は 2 つのデータブロックに保存され、このデータブロックで現在値(#Setpoint_Speed)が変更されます。

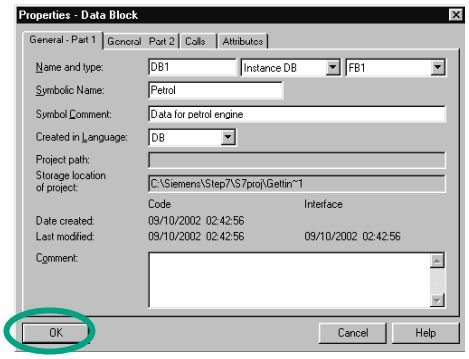
ファンクションブロックを一元的に 1 度だけプログラミングすれば、作成するプログラムの量を減らすことができます。



"Getting Started"プロジェクトで SIMATIC Manager を開きます。

Blocks フォルダへ移動し、ウィンドウの右側をマウスの右ボタンでクリックします。

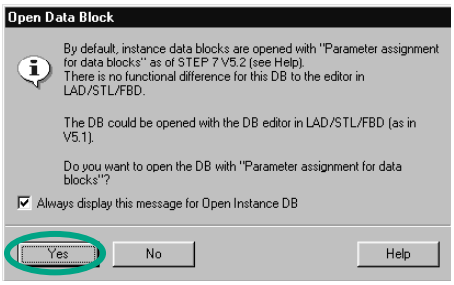
右マウスボタンでポップアップメニューを表示し、**データブロック**を挿入します。



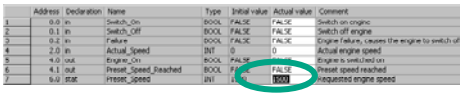
名前 DB1 を[プロパティデータブロック]ダイアログボックスに適用してから、隣りのプルダウンリストでアプリケーション"インスタンス DB"を選択し、割り付けられるファンクションブロック"FB1"ブロックの名前に適用します。[**OK**]をクリックして、[プロパティ]ダイアログボックスに表示されるすべての設定を適用します。

データブロック **DB1** が"Getting Started"プロジェクトに追加されます。

DB1 をダブルクリックして開きます。

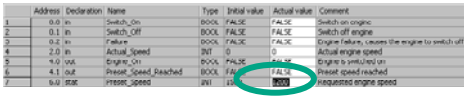


その後のダイアログを**[はい]**で確定して、パラメータをインスタンスデータブロックに割り付けます。



ガソリンエンジンの値"1500"を[現在値]列の"Setpoint_Speed"行に入力します。これで、このエンジンの最高速度が定義されました。

DB1 を保存し、プログラムウィンドウを閉じます。



DB1 の場合と同様に、FB1 に別のデータブロック DB2 を生成します。

ディーゼルエンジンの現在値"1200"を入力します。

DB2 を保存し、プログラムウィンドウを閉じます。

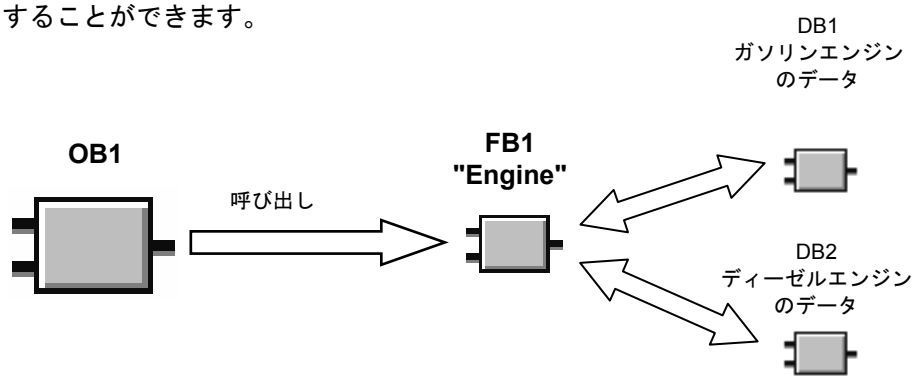
現在値を変更することにより、1 つのファンクションブロックで 2 つのエンジンを制御する準備ができました。追加のデータブロックを作成すれば、制御するエンジンの数を増やすことができます。

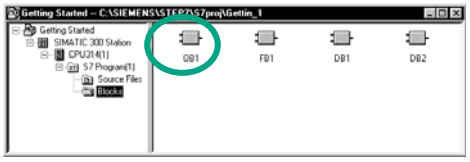
次にしなければならないことは、OB1 のファンクションブロックに対する呼び出しのプログラミングです。それには、使用しているプログラム言語に従い、ラダーロジックは第 5 章 6 節を、ステートメントリストは第 5 章 7 節を、ファンクションブロックダイアグラムは第 5 章 8 節をお読みください。

詳細は、**[ヘルプ|目次]**の「ブロックのプログラミング」および「データブロックの作成」を参照してください。

5.6 ラダーロジックによるブロック呼び出しのプログラミング

ファンクションブロックに対してプログラミングしたことはすべて、このブロックを OB1 で呼び出ししない限り役に立ちません。データブロックは各ファンクションブロックに対して使用され、このように両方のエンジンを制御することができます。

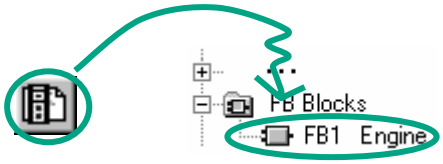




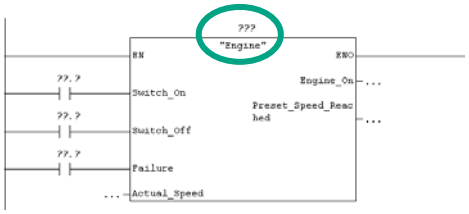
"Getting Started" プロジェクトで SIMATIC Manager を開きます。

Blocks フォルダへ移動し、**OB1** を開きます。

ネットワーク 3 を選択してから、LAD/STL/FBD プログラムウィンドウにネットワーク 4 を挿入します。

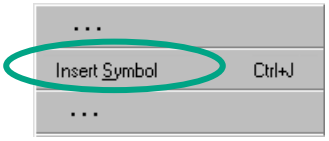


プログラムエレメントカタログで **FB1** へ移動し、ダブルクリックして挿入します。

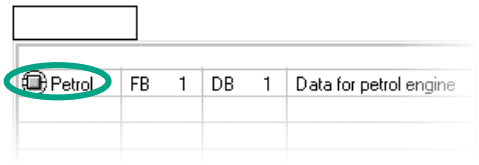


Switch_On、Switch_Off および Fault の前に a 接点を挿入します。

"Engine"の上の**???** をクリックし、その位置にカーソルを置いたまま、右マウスボタンで入力フレームをクリックします。

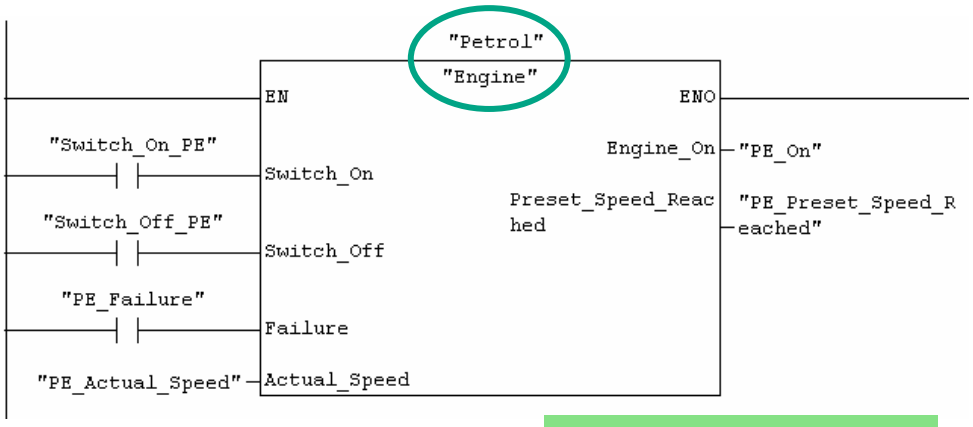


ショートカットメニューの**[シンボルの挿入]**を右マウスボタンでクリックします。プルダウンリストが表示されます。



データブロック **Petrol** をダブルクリックします。このブロック名は引用符で囲まれて、入力フレームに自動的に入力されます。

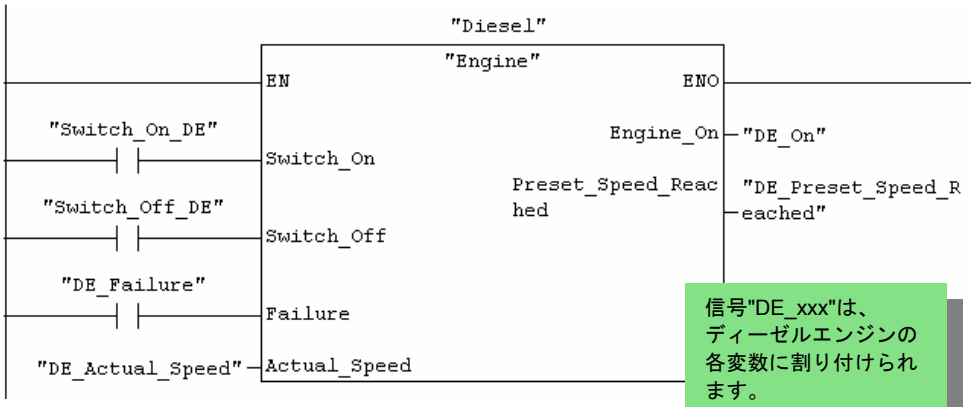
疑問符をクリックし、疑問符を入力してから、プルダウンリストの対応するシンボル名を使ってファンクションブロックのその他のパラメータをアドレス指定します。



エンジン固有の入力/出力変数("in"と"out"で宣言)が FB "Engine"に表示されます。

信号"PE_xxx"は、ガソリンエンジンの各変数に割り付けられます。

新しいネットワークで、ファンクションブロック"Engine" (FB1)の呼び出しをデータブロック"Diesel" (DB2)を使ってプログラミングし、プルダウンリストの対応するアドレスを使用します。



プログラムを保存し、ブロックを閉じます。

オーガニゼーションブロック、ファンクションブロック、データブロックを使ってプログラム構造を作成する場合、階層構造の上のブロック(たとえばOB1)で下位ブロック(FB1など)の呼び出しをプログラミングする必要があります。作成手順は常に同じです。

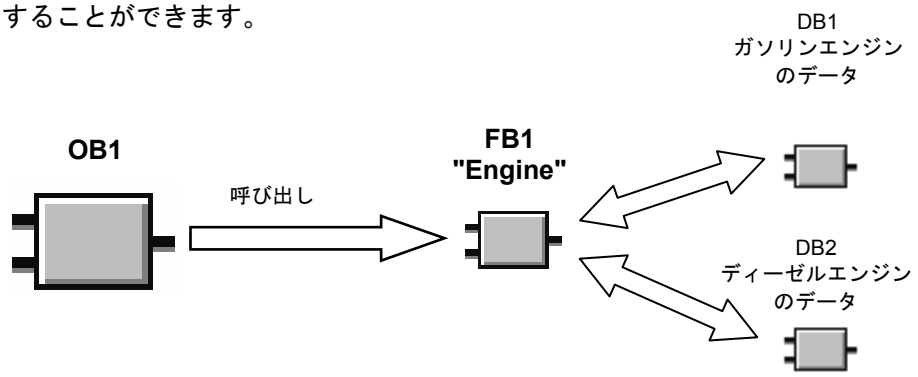
また、様々なブロックシンボル名をシンボルテーブルで指定することもできます(たとえば、FB1の名前に"Engine"、DB1の名前に"Petrol"など)。

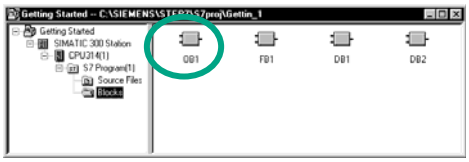
プログラミングしたブロックはいつでもアーカイブしたり、印刷することができます。対応するファンクションは、SIMATIC Managerのメニューコマンド**[ファイル|アーカイブ]**または**[ファイル|印刷]**で見つけることができます。

「言語の説明: LAD」と「プログラム制御命令」の下のトピック「参照ヘルプの呼び出し」で**[ヘルプ|目次]**を使って詳細な説明を見つけることができます。

5.7 ステートメントリストによるブロック呼び出しのプログラミング

ファンクションブロックに対してプログラミングしたことはすべて、このブロックを OB1 で呼び出ししない限り役に立ちません。データブロックは各ファンクションブロックに対して使用され、このように両方のエンジンを制御することができます。





"Getting Started" プロジェクトで SIMATIC Manager を開きます。

Blocks フォルダへ移動し、**OB1** を開きます。

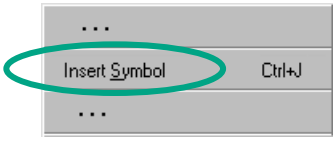
ネットワーク 3 を選択してから、LAD/STL/FBD プログラムウィンドウにネットワーク 4 を挿入します。

```
CALL  "Engine" , "Petrol"
  Switch_On            :=
  Switch_Off           :=
  Failure              :=
  Actual_Speed         :=
  Engine_On            :=
  Preset_Speed_Reached:=
```

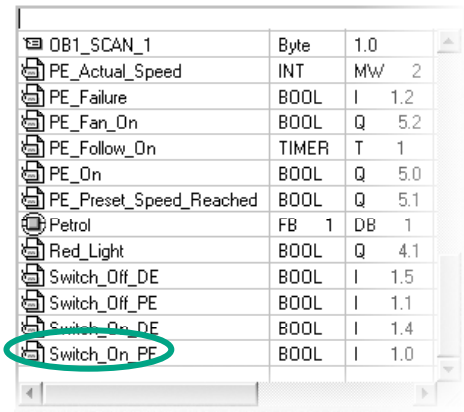
コードセクションで **CALL "Engine", "Petrol"**とタイプし、**Enter** キーを押します。

ファンクションブロック"Petrol"の全パラメータが表示されます。

Switch_On の等号記号の後ろにカーソルを置き、マウスの右ボタンを押します。



ショートカットメニューの[シンボルの挿入]を右マウスボタンでクリックします。プルダウンリストが表示されます。



OB1_SCAN_1	Byte	1.0
PE_Actual_Speed	INT	MW 2
PE_Failure	BOOL	I 1.2
PE_Fan_On	BOOL	Q 5.2
PE_Follow_On	TIMER	T 1
PE_On	BOOL	Q 5.0
PE_Preset_Speed_Reached	BOOL	Q 5.1
Petrol	FB 1	DB 1
Red_Light	BOOL	Q 4.1
Switch_Off_DE	BOOL	I 1.5
Switch_Off_PE	BOOL	I 1.1
Switch_On_DE	BOOL	I 1.4
Switch_On_PE	BOOL	I 1.0

Switch_On_PE をクリックします。この名称はプルダウンリストに表示され、自動的に引用符で囲まれます。

```
CALL  "Engine" , "Petrol"
 Switch_On            :="Switch_On_PE"
 Switch_Off           :="Switch_Off_PE"
 Failure              :="PE_Failure"
 Actual_Speed         :="PE_Actual_Speed"
 Engine_On            :="PE_On"
 Preset_Speed_Reached:="PE_Preset_Speed_Reached"
```

プルダウンリストを使って、必要なアドレスすべてを変数に割り付けます。

信号"PE_xxx"は、ガソリンエンジンの各変数に割り付けられます。

```
CALL  "Engine" , "Diesel"
 Switch_On            :="Switch_On_DE"
 Switch_Off           :="Switch_Off_DE"
 Failure              :="DE_Failure"
 Actual_Speed         :="DE_Actual_Speed"
 Engine_On            :="DE_On"
 Preset_Speed_Reached:="DE_Preset_Speed_Reached"
```

新しいネットワークでファンクションブロック"Engine"(FB1)の呼び出しをデータブロック"Diesel"(DB2)を使ってプログラミングします。これから後の手順は他の呼び出しと同じです。

プログラムを保存し、ブロックを閉じます。

オーガニゼーションブロック、ファンクションブロック、データブロックを使ってプログラム構造を作成する場合、階層構造の上のブロック(たとえば OB1)で下位ブロック(FB1 など)の呼び出しをプログラミングする必要があります。作成手順は常に同じです。

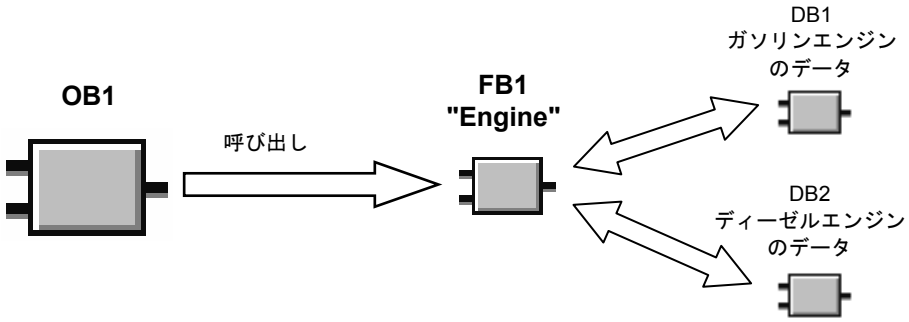
また、様々なブロックシンボル名をシンボルテーブルで指定することもできます(たとえば、FB1 の名前に"Engine"、DB1 の名前に"Petrol"など)。

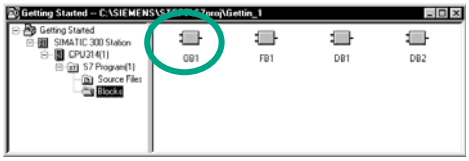
プログラミングしたブロックはいつでもアーカイブしたり、印刷することができます。対応するファンクションは、SIMATIC Manager のメニューコマンド**[ファイル|アーカイブ]**または**[ファイル|印刷]**で見つけることができます。

「言語の説明: STL」と「プログラム制御命令」の下のトピック「参照ヘルプの呼び出し」で**[ヘルプ|目次]**を使って詳細な説明を見つけることができます。

5.8 ファンクションブロックダイアグラムによるブロック呼び出しのプログラミング

ファンクションブロックに対してプログラミングしたことはすべて、このブロックを OB1 で呼び出さない限り役に立ちません。データブロックは各ファンクションブロックに対して使用され、このように両方のエンジンを制御することができます。

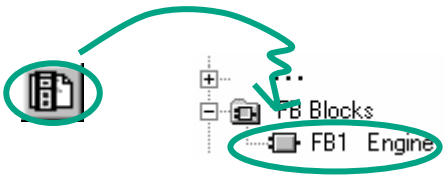




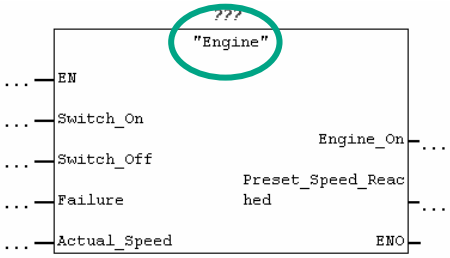
"Getting Started"プロジェクトで SIMATIC Manager を開きます。

Blocks フォルダへ移動し、**OB1** を開きます。

ネットワーク 3 を選択してから、LAD/STL/FBD プログラムウィンドウにネットワーク 4 を挿入します。

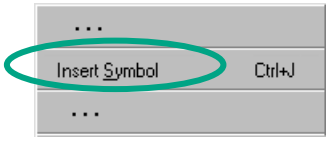


プログラムエレメントカタログで **FB1** へ移動し、このブロックを挿入します。

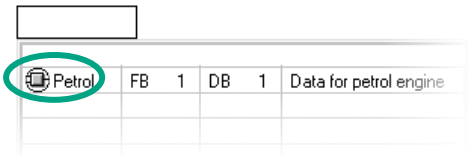


エンジン固有の入力/出力変数がすべて表示されます。

"Engine"の上の**???** をクリックし、その位置にカーソルを置いたまま、右マウスボタンで入力フレームをクリックします。

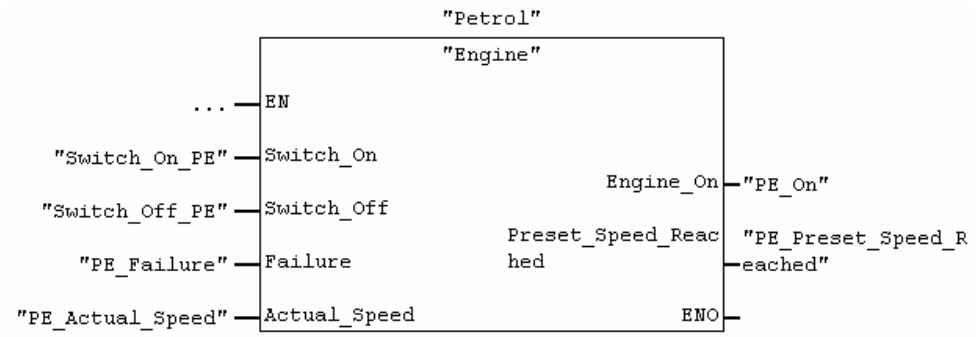


ショートカットメニューの**[シンボルの挿入]**を右マウスボタンでクリックします。プルダウンリストが表示されます。



データブロック **Petrol** をダブルクリックします。このブロック名は引用符で囲まれて、入力フレームに自動的に入力されます。

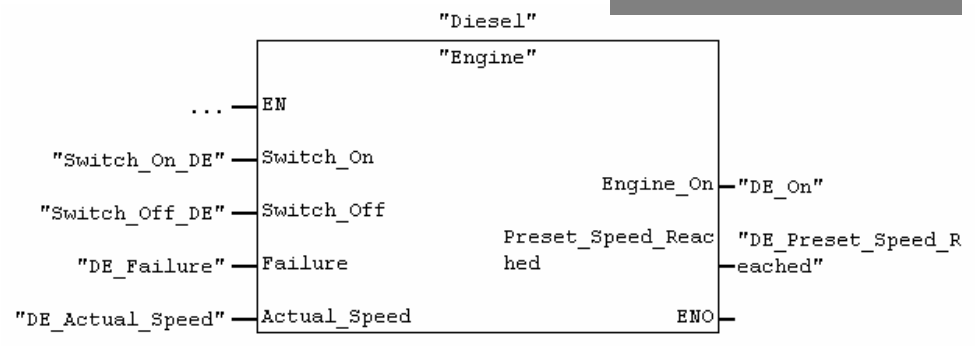
疑問符をクリックし、プルダウンリストの対応するシンボル名を使ってファンクションブロックのその他のパラメータをアドレス指定します。



信号"PE_xxx"は、ガソリンエンジンの各変数に割り付けられます。

新しいネットワークで、ファンクションブロック"Engine" (FB1)の呼び出しをデータブロック"Diesel" (DB2)を使ってプログラミングし、プルダウンリストの対応するアドレスを使用します。

信号"DE_xxx"は、ディーゼルエンジンの各変数に割り付けられます。



プログラムを保存し、ブロックを閉じます。

オーガニゼーションブロック、ファンクションブロック、データブロックを使ってプログラム構造を作成する場合、階層構造の上のブロック(たとえば OB1)で下位ブロック(FB1 など) の呼び出しをプログラミングする必要があります。作成手順は常に同じです。

また、様々なブロックシンボル名をシンボルテーブルで指定することもできます (たとえば、FB1 の名前に"Engine"、DB1 の名前に"Petrol"など)。

プログラミングしたブロックはいつでもアーカイブしたり、印刷することができます。対応するファンクションは、SIMATIC Manager のメニューコマンド**[ファイル|アーカイブ]**または**[ファイル|印刷]**で見つけることができます。

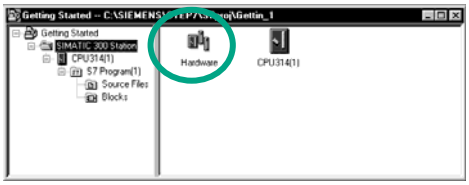
「言語の説明: FBD」と「プログラム制御命令」の下のトピック「参照ヘルプの呼び出し」で**[ヘルプ|目次]**を使って詳細な説明を見つけることができます。

6　基本ラックのコンフィグレーション

6.1　ハードウェアのコンフィグレーション

SIMATIC ステーションでプロジェクトを作成したら、次にハードウェアのコンフィグレーションを行えます。第 2 章 2 節で STEP 7 ウィザードを使ってプロジェクト構造を作成したので、ハードウェアのコンフィグレーションを行うための条件が整いました。

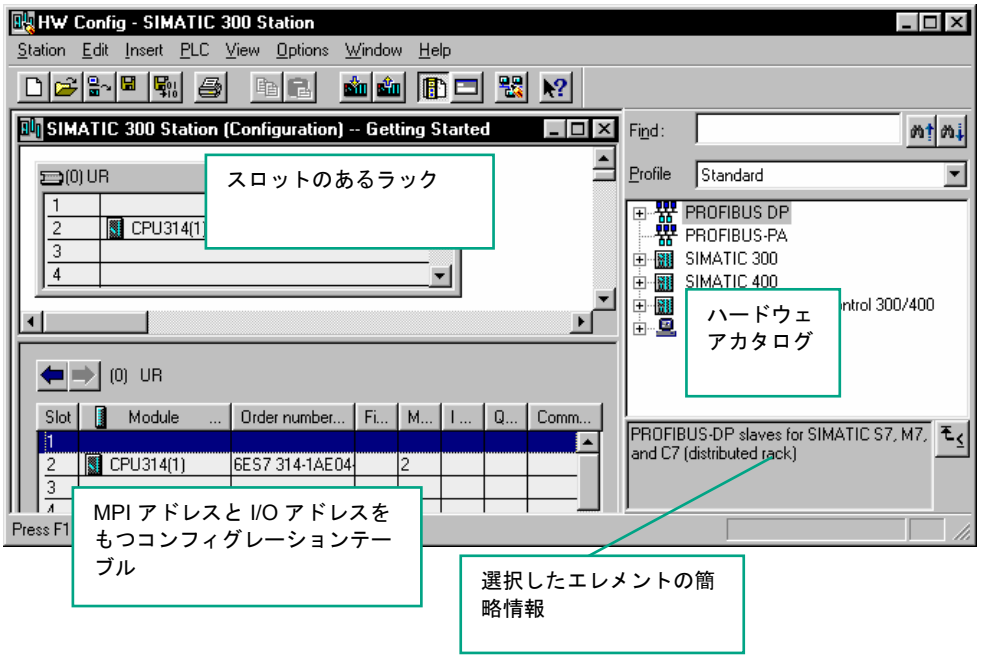
ハードウェアのコンフィグレーションは STEP 7 で行います。これらのコンフィグレーションデータは、後で「ダウンロード」によってプログラマブルコントローラに転送されます (第 7 章を参照)。

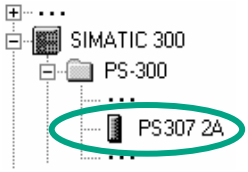


SIMATIC マネージャで "Getting Started" プロジェクトを開き、ここから作業を開始します。

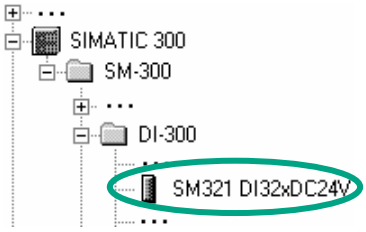
SIMATIC 300 Station フォルダを開き、**Hardware** シンボルをダブルクリックします。

[HW Config] ウィンドウが開きます。プロジェクトの作成時に選択した CPU が表示されます。"Getting Started" プロジェクトの場合は CPU 314 です。

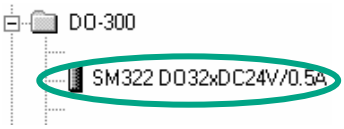




まず最初に電源モジュールが必要です。カタログの **PS307 2A** をスロット 1 へドラッグ＆ドロップします。



入力モジュール (DI, デジタル入力) **SM321 DI32xDC24V** へ移動し、これをスロット 4 に挿入します。スロット 3 は空にしておきます。



同様に、出力モジュール **SM322 DO32xDC24V/0.5A** をスロット 5 に挿入します。

プロジェクト内のモジュールのパラメータ (たとえばアドレス) を変更するには、そのモジュールをダブルクリックします。ただし、変更を行うのは、この変更によるプログラマブルコントローラへの影響がはっきりしている場合だけにしてください。

"Getting Started" プロジェクトには変更は必要ありません。

Slot	Module	Order Number	MPI Address	I Add...	Q...	Comment
1	PS307 2A	6ES7 307-1BA00-0AA0				
2	CPU314(1)	6ES7 314-1AE04-0AB0	2			
3						
4	DI32xDC24V	6ES7 321-1BL00-0AA0		0...3		
5	DO32xDC24V/0.5A	6ES7 322-1BL00-0AA0			4...7	
6						
7						
8						
9						
10						
11						

メニューコマンド [**保存とコンパイル**] を使って、データを CPU に転送できるようにします。

"HW Config" アプリケーションを閉じると、"System Data" シンボルが Blocks フォルダに表示されます。

さらに、メニューコマンド [**ステーション > 一貫性チェック**] により、コンフィグレーションのエラーをチェックすることができます。エラーが発生した場合でも、STEP 7 には有効な解決策が用意されています。

詳細は、[**ヘルプ> 目次**] の「ハードウェアのコンフィグレーション」および「基本ラックのコンフィグレーション」を参照してください。

7　プログラムのダウンロードおよびデバッグ

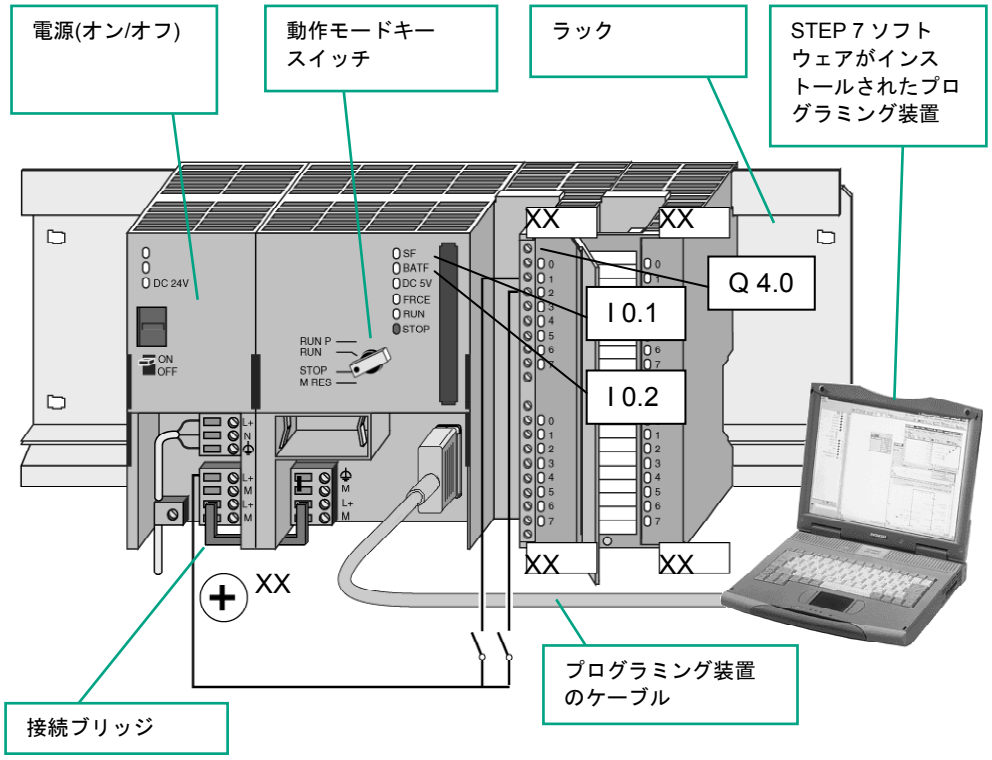
7.1　オンライン接続の確立

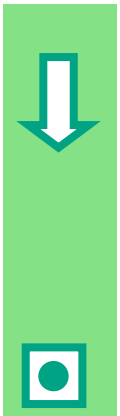
付属の"GS-LAD_Example"プロジェクトまたは"Getting Started"プロジェクトを使用して、コンフィグレーションを作成し、簡単なテストを行いましたが、次にプログラムをプログラマブルロジックコントローラ(PLC)にダウンロードし、デバッグする方法を説明します。

以下の作業はすでに完了しています。

- "Getting Started"プロジェクトのハードウェアコンフィグレーション(第 6 章を参照)
- インストールマニュアルの手順によるハードウェアのセットアップ

直列回路の例(AND ファンクション):
出力 Q 4.0 は、Key I 0.1 と Key I 0.2 を押さなければ点灯しません(ダイオード Q 4.0 はデジタル出力モジュールで点灯します)。テストコンフィグレーションとして、ケーブルと CPU を下記の方法でセットアップします。





ハードウェアのコンフィグレーション

モジュールのレールへの取り付けは、次の順序に従って行います。

- モジュールをバスコネクタに接続します。
- モジュールをレールに引っかけて、下方へ回転させます。
- モジュールを所定の位置にネジで止めます。
- 残りのモジュールを取り付けます。
- すべてのモジュールの取り付けが終わったら、CPU にキーを差し込みます。

使用するハードウェアが前ページの図と異なる場合でも、テストを実行することができます。この場合、入力と出力のアドレス指定を守りさえすれば問題なく実行できます。
STEP 7 は、ユーザのプログラムをデバッグするさまざまな方法を提供します。たとえば、プログラムステータスを使用する方法や変数テーブルによる方法です。

基本ラックのコンフィグレーションの詳細については、「S7-300, Hardware and Installation / Module Specifications」および「S7-400 / M7-400 – Hardware」を参照してください。

7.2 プログラムのプログラマブルコントローラへのダウンロード

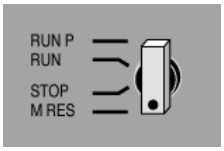
プログラムをダウンロードするには、オンライン接続を確立しておく必要があります。

電圧の適用

ON/OFF スイッチを使って電源を入れます。CPU のダイオード"DC 5V"が点灯します。

動作モードスイッチを STOP にします (STOP になっていない場合)。
"STOP"LED が赤く点灯します。

CPU のリセットおよび RUN への切り替え



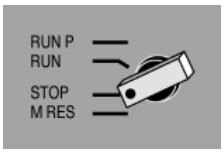
動作モードスイッチを **MRES** にして、3 秒ほどそのままにしておきます。赤の"STOP"LED がゆっくりと点滅し始めます。

スイッチを解除し、3 秒以上待ってから再度 **MRES** にします。"STOP"LED が素早く点滅し、CPU がリセットされます。

"STOP"LED が素早く点滅し始めない場合は、上記の手順を繰り返してください。

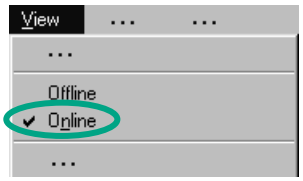
メモリリセットにより CPU 上の全データが消去され、 CPU は初期状態になる

プログラムの CPU へのダウンロード

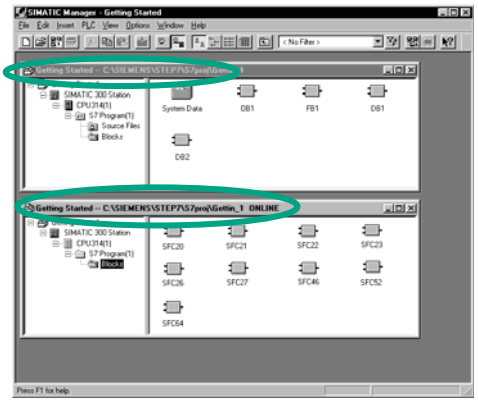


動作モードスイッチを"STOP"にします。これで、プログラムのダウンロードが実行されます。

SIMATIC Manager を起動し、[開く]ダイアログボックスで"Getting Started" プロジェクトを開きます (まだ開いていない場合)。



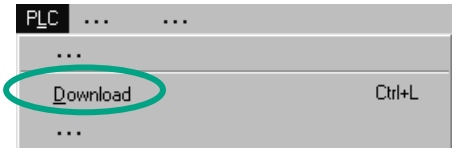
[Getting Started Offline]ウィンドウのほかに、[Getting Started ONLINE]ウィンドウも開きます。オンラインまたはオフラインの状態は、ヘッダーで色別に表示されます。



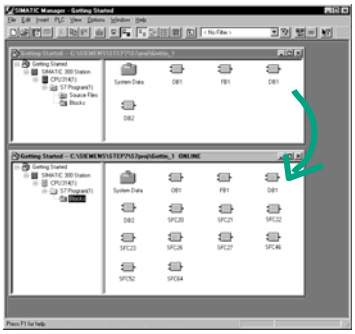
両方のウィンドウで **Blocks** フォルダを開きます。

オフラインウィンドウにより、プログラミング装置に関する状況が表示されます。オンラインウィンドウにより、CPU に関する状況が表示されます。

メモリをリセットを実行しても、システムファンクション (SFC) が CPU から削除されることはありません。CPU は、これらの SFC をオペレーティングシステムに提供します。SFC はダウンロードする必要はありませんが、削除することはできません。



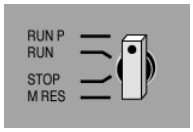
オフラインウィンドウで **Blocks** フォルダを選択し、メニューコマンド **[PLC|ダウンロード]**を使ってプログラムを CPU にダウンロードします。プロンプトに**[OK]**で応答します。



プログラムブロックは、ダウンロード時にオンラインウィンドウに表示されます。

メニューコマンド**[PLC|ダウンロード]**は、ツールバーのボタンから呼び出したり、マウスの右ボタンでポップアップメニューを表示させて選択することもできます。

CPU の起動および動作モードのチェック



動作モードスイッチを **RUN-P** にします。緑の"RUN"LED が点灯し、赤の"STOP"LED が消えます。CPU は動作可能状態になります。

緑の LED が点灯したら、プログラムのテストを開始できます。

赤の LED が点灯したままの場合は、エラーが発生しています。この場合、診断バッファを調べてエラーを診断しなければならないことがあります。

各ブロックのダウンロード

エラーに迅速に対応するためには、ドラッグ&ドロップ機能を使って、ブロックを個々に CPU に転送します。
ブロックをダウンロードする場合、CPU の動作モードスイッチは"RUN-P"または"STOP"になっていなければなりません。"RUN-P"モードでダウンロードされたブロックは、直ちにアクティブになります。したがって、以下に留意する必要があります。

- 正常なブロックをエラーのあるブロックで上書きすると、プラント障害が発生します。これを防ぐには、ブロックはテストしてからダウンロードするようにします。
- ブロックをダウンロードする順序(下位ブロックから上位ブロックへ)を守らないと、CPU は"STOP"モードになります。これを防ぐには、プログラム全体を CPU にダウンロードします。

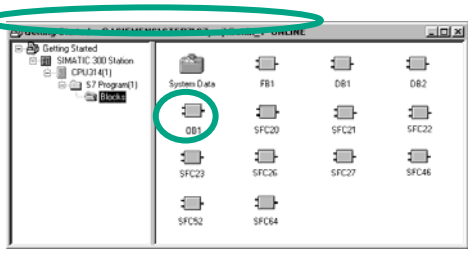
オンラインでのプログラミング

一般に、ブロックをテストする場合、CPU にダウンロードしたブロックを変更しなければならない場合があります。変更するには、目的のブロックをオンラインウィンドウでダブルクリックして、LAD/STL/FBD プログラムウィンドウを開きます。次に、ブロックを通常通りにプログラミングします。プログラミングしたブロックは、CPU で直ちにアクティブになります。

詳細は、**[ヘルプ|目次]**の「ダウンロードとアップロード」、「オンライン接続の確立と CPU の設定」を参照してください。

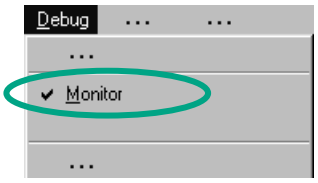
7.3 プログラムステータスによるプログラムのテスト

プログラムステータス機能により、ブロックでプログラムのテストを実行できます。このためには、CPU とのオンライン接続を確立し、CPU を RUN モードまたは RUN-P モードにし、プログラムをダウンロードしておく必要があります。



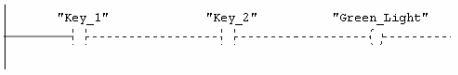
プロジェクトウィンドウ"Getting Started ONLINE"で **OB1** を開きます。

LAD/STL/ FBD プログラムウィンドウが開きます。



[デバッグ|モニタ] を有効にします。

ラダーロジックによるデバッグ

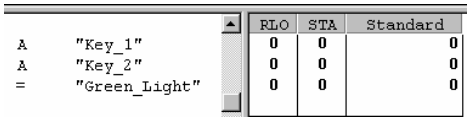


Network 1 の直列回路がラダーロジックで表示されます。制御線が Key 1(I 0.1)までの実線で示されます。つまり、電力は既に回路に適用されています。

ファンクションブロックダイアグラムによるデバッグ

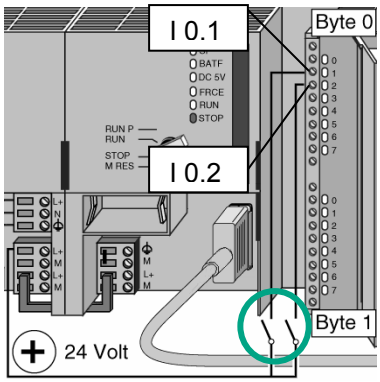
信号状態が、"0"および"1"により示されます。点線は、論理演算結果がないことを意味します。

ステートメントリストによるデバッグ



ステートメントリストでは、表形式で示されます。– 論理演算結果(RLO)
– ステータスビット(STA)
– 標準ステータス(STANDARD)

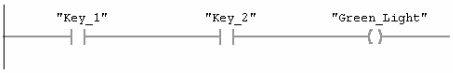
[オプション|カスタマイズ]により、テスト実行中におけるプログラミング言語の表示法を変更できます。

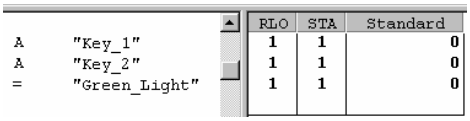


テストコンフィグレーションの両方のキーを押します。

入力モジュールの入力 I 0.1 と I 0.2 のダイオードが点灯します。

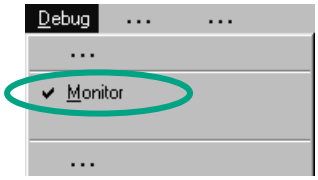
出力モジュールの Q 4.0 のダイオードが点灯します。





グラフィックプログラミング言語であるラダーロジックとファンクションブロックダイアグラムでは、プログラミングされたネットワークの色の変化によりテスト結果を追跡できます。この色の変化は、その時点まで、論理演算結果が true であることを示します。

ステートメントリストでは、論理演算結果が true の場合に STA 列と RLO 列の表示が変化します。



[デバッグ|モニタ]を無効にして、ウィンドウを閉じます。

SIMATIC Manager のオンラインウィンドウを閉じます。

大量のプログラムを一度に CPU にダウンロードして実行することは避けてください。エラーが発生した場合、考えられる原因が多すぎるために診断が難しくなります。全体を把握できるように、ブロックを個々にダウンロードしてからテストする方法をお勧めします。

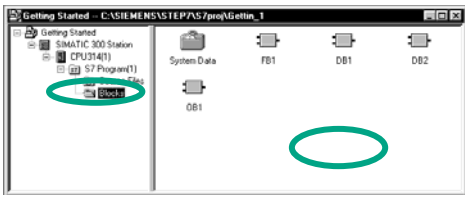
詳細は、**[ヘルプ|目次]**の「デバッグ」および「プログラムステータスによるテスト」を参照してください。

7.4 変数テーブルによるプログラムのテスト

プログラム変数は、モニタし修正することによりテストできます。このためには、CPU とのオンライン接続を確立し、CPU を RUN-P モードにし、プログラムをダウンロードしておく必要があります。

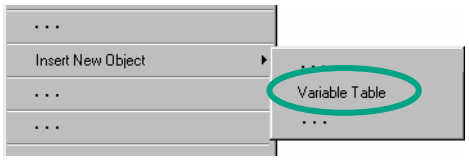
プログラムステータスによるテストと同様に、変数テーブルで Network 1 (直列回路または AND ファンクション)の入力と出力をモニタすることができます。さらに、実際の速度を事前設定することにより FB 1 のエンジン速度のコンパレータをテストすることもできます。

変数テーブルの作成

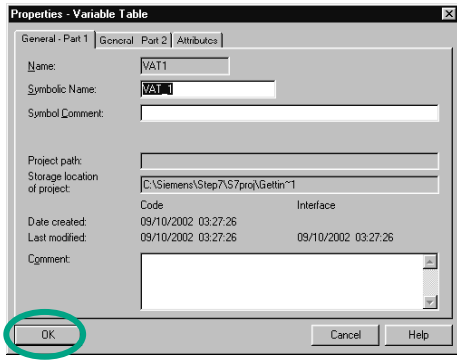


SIMATIC Manager でプロジェクトウィンドウ"Getting Started Offline"を開きます。

Blocks フォルダへ移動し、ウィンドウの右側をマウスの右ボタンでクリックします。

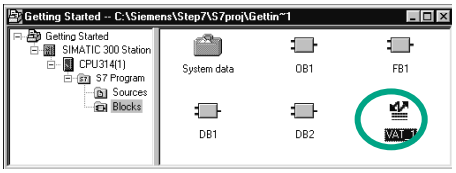


右マウスボタンでポップアップメニューを表示し、そこから**変数テーブル**を挿入します。



[OK]をクリックして[プロパティ]ダイアログボックスを閉じて、デフォルト設定を適用します。

代わりに、シンボル名を変数テーブルに割り付け、シンボルのコメントを入力することができます。

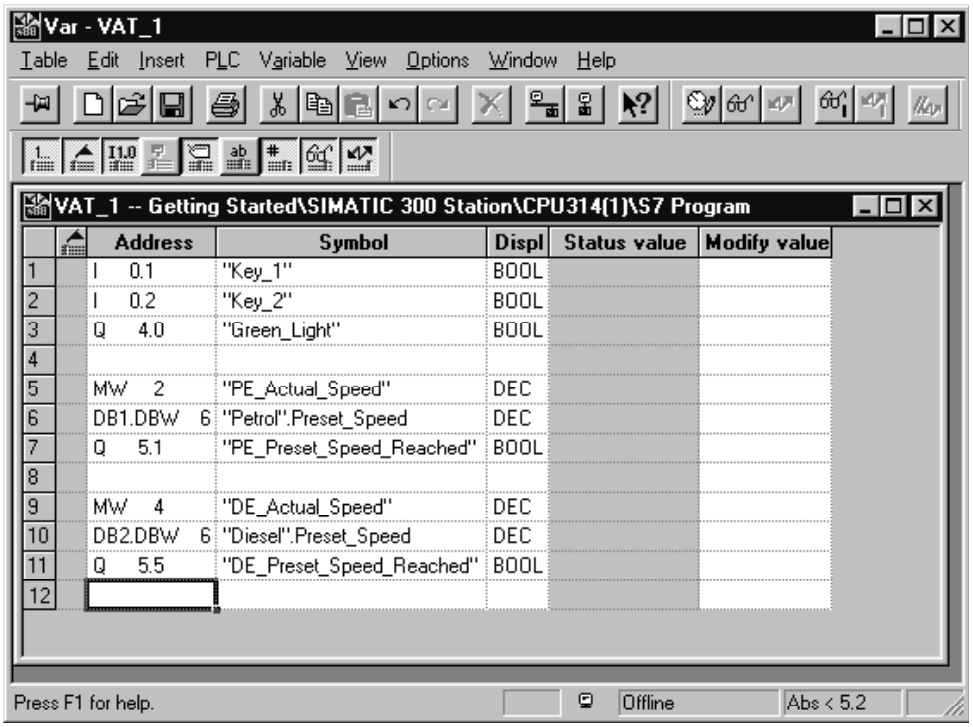


VAT1(変数テーブル)が Blocks フォルダに作成されます。

VAT1 をダブルクリックして開きます。[変数のモニタと修正]ウィンドウが開きます。

変数テーブルは最初は空の状態です。"Getting Started"のシンボル名またはアドレスを、下記の図に従って入力します。**Enter** キーを押して入力を完了すると、詳細項目が追加されます。

すべての速度値のステータスフォーマットを DEC (10 進)フォーマットに変換します。それには、対応するセルをクリックし、マウスの右ボタンを使用して DEC フォーマットを選択します。

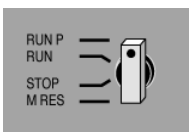


	Address	Symbol	Displ	Status value	Modify value
1	I 0.1	"Key_1"	BOOL		
2	I 0.2	"Key_2"	BOOL		
3	Q 4.0	"Green_Light"	BOOL		
4					
5	MW 2	"PE_Actual_Speed"	DEC		
6	DB1.DBW 6	"Petrol".Preset_Speed	DEC		
7	Q 5.1	"PE_Preset_Speed_Reached"	BOOL		
8					
9	MW 4	"DE_Actual_Speed"	DEC		
10	DB2.DBW 6	"Diesel".Preset_Speed	DEC		
11	Q 5.5	"DE_Preset_Speed_Reached"	BOOL		
12					

変数テーブルを保存します。

変数テーブルのオンラインでの切り替え

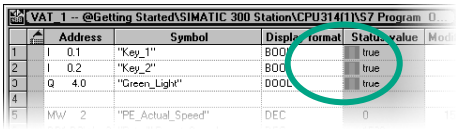
コンフィグレーションされた CPU との接続の確立します。CPU の動作モードがステータスバーに表示されます。



CPU のキースイッチを **RUN-P** に設定します(RUN-P にしていない場合)。

変数のモニタ

ツールバーの**[変数のモニタ]**ボタンをクリックします。

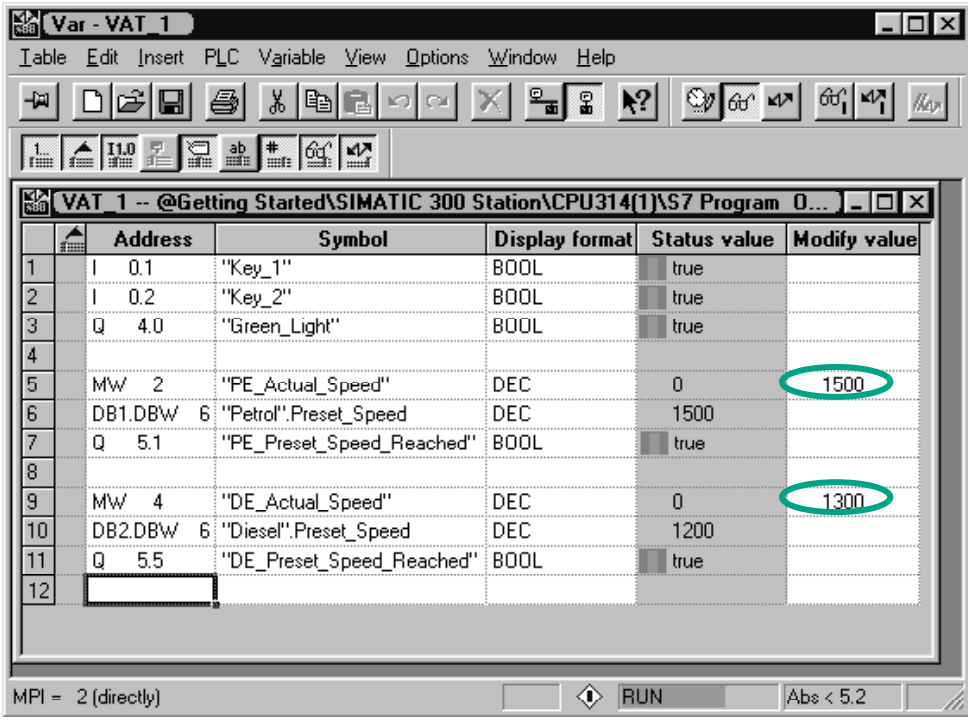


テストコンフィグレーションの Key 1 と Key 2 を押して、変数テーブルの結果をモニタします。

変数テーブルのステータス値が false から true に変更されます。

変数の修正

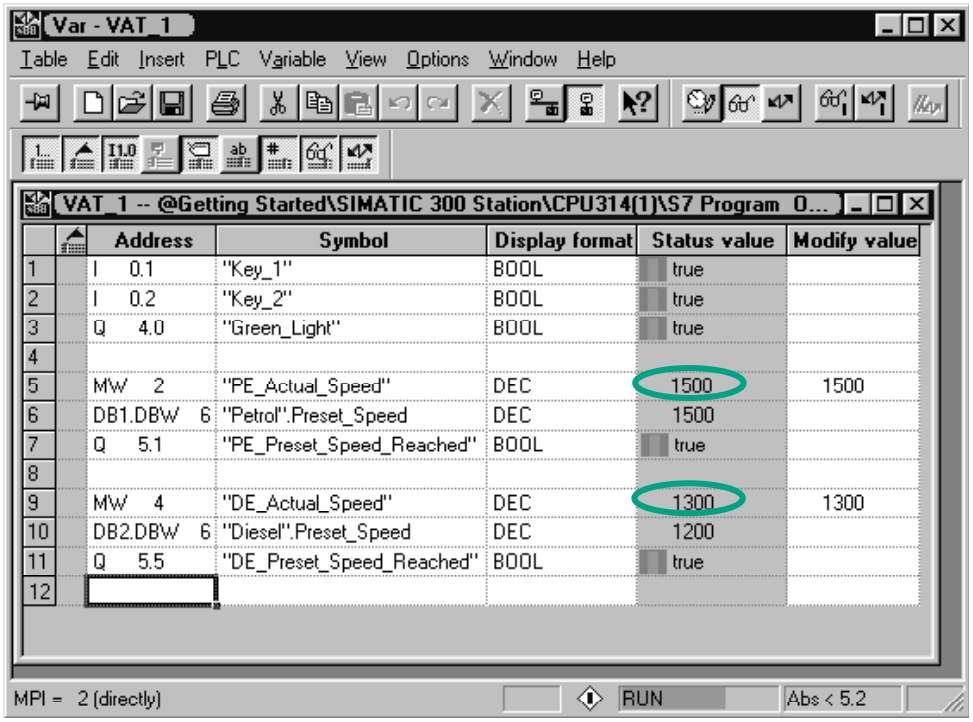
[修正値]列のアドレス MW2 に"1500"を、MW4 に"1300"を入力します。



修正値を CPU に転送します。

これらの値は、転送後に CPU で処理されます。比較の結果が表示されます。

変数のモニタを停止し (ツールバーのボタンを再度クリックする)、ウィンドウを閉じます。照会に対して**[YES]**または**[OK]**で肯定応答します。



	Address	Symbol	Display format	Status value	Modify value
1	I 0.1	"Key_1"	BOOL	true	
2	I 0.2	"Key_2"	BOOL	true	
3	Q 4.0	"Green_Light"	BOOL	true	
4					
5	MW 2	"PE_Actual_Speed"	DEC	1500	1500
6	DB1.DBW 6	"Petrol".Preset_Speed	DEC	1500	
7	Q 5.1	"PE_Preset_Speed_Reached"	BOOL	true	
8					
9	MW 4	"DE_Actual_Speed"	DEC	1300	1300
10	DB2.DBW 6	"Diesel".Preset_Speed	DEC	1200	
11	Q 5.5	"DE_Preset_Speed_Reached"	BOOL	true	
12					

変数テーブルが非常に大きい場合、画面スペースの関係ですべてを表示できません。
大きな変数テーブルがある場合、STEP 7 を使用して 1 つの S7 プログラムに複数のテーブルを作成することをお勧めします。変数テーブルをユーザ固有のテスト要件にぴったり合わせることができます。
変数テーブルには、ブロックの場合と同じように名前を個々に割り付けることができます(たとえば、VAT1 ではなく OB1_Network1)。シンボルテーブルを使って、新しい名前を割り付けます。

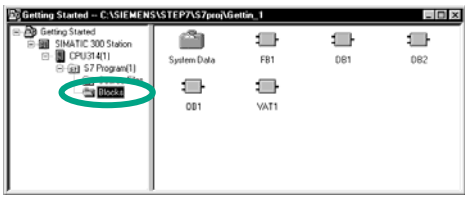
詳細は、**[ヘルプ|目次]**の「変数テーブルによるテスト」の「デバッグ」を参照してください。

7.5 診断バッファの評価

極端な場合、S7 プログラムの処理中に CPU が STOP になったり、プログラムのダウンロード後に CPU が RUN になることがあります。この場合は、診断バッファにリストされているイベントから原因を突き止めることができます。

そのためには、CPU とのオンライン接続を確立し、CPU を STOP モードにしておく必要があります。

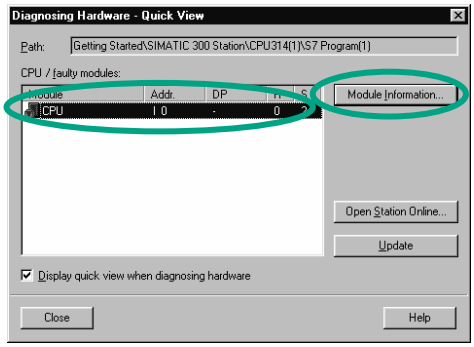
CPU の動作モードスイッチを STOP にします。



SIMATIC Manager でプロジェクトウィンドウ"Getting Started Offline"を開きます。

Blocks フォルダを選択します。

プロジェクトに複数の CPU がある場合、STOP になっている CPU を調べます。



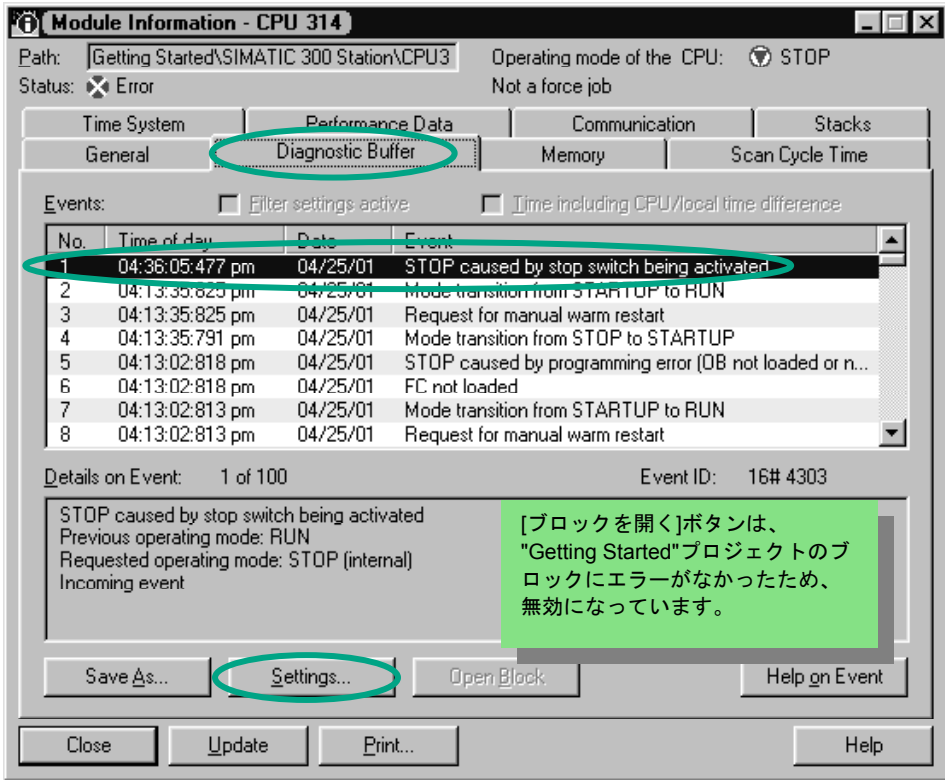
使用可能なすべての CPU が [ハードウェアの診断] ダイアログボックスにリストされます。STOP モードの CPU が強調表示されます。

"Getting Started" プロジェクトでは、CPU は 1 つだけ表示されます。

[モジュール情報] をクリックして、この CPU の診断バッファの評価を行います。

接続されている CPU が 1 つの場合、この CPU のモジュール情報は、メニューコマンド**[PLC|モジュール情報]**によって照会できます。

[モジュール情報] ウィンドウには、CPU のプロパティとパラメータに関する情報が表示されます。ここで、[診断バッファ] タブを選択して、STOP 状態になった原因を調べます。



最新のイベント (番号 1) は、リストの一番上に表示されます。STOP 状態の原因が表示されます。SIMATIC Manager 以外のすべてのウィンドウを閉じます。

CPU が STOP モードになった原因がプログラミングエラーの場合、イベントを選択し、[ブロックを開く] ボタンをクリックします。

LAD/STL/FBD プログラムウィンドウでブロックが開き、障害のあるネットワークが強調表示されます。

この章では、プロジェクトの作成、完成したプログラムのデバッグを実行して "Getting Started" サンプルプロジェクトが完成しました。次の章では、演習を選択し実行することで、さらに知識を深めることができます。

詳細については、**[ヘルプ|目次]**の「モジュール情報の呼び出し」を参照してください。

8　ファンクションのプログラミング

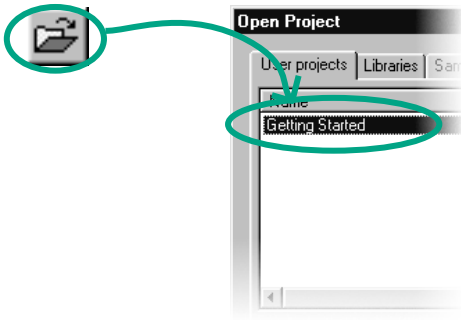
8.1　ファンクション(FC)の作成および操作

ファンクションは、ファンクションブロックと同様に、プログラム階層のオーガニゼーションブロックの下にあります。ファンクションを CPU で処理するには、階層の上位にあるブロックで呼び出す必要があります。ただし、データブロックは、この方法で呼び出す必要はありません。

ファンクションでは、パラメータも変数宣言テーブルにリストされますが、スタティックローカルデータはリストされません。

ファンクションは、ファンクションブロックと同様に LAD/STL/FBD プログラムウィンドウを使ってプログラミングできます。

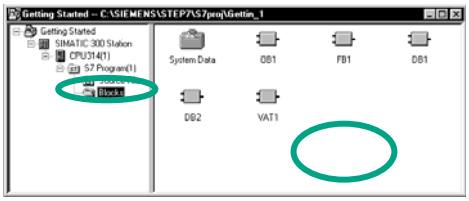
ラダーロジック、ファンクションブロックダイアグラム、ステートメントリスト(第 4 章と第 5 章を参照) の各プログラミング言語、シンボルによるプログラミング (第 3 章を参照) については既に理解されているでしょう。



第 1 章から第 7 章まで "Getting Started" サンプルプロジェクトを使用してきた場合は、そのプロジェクトを開きます。

上記以外の場合は、SIMATIC Manager のメニューコマンド[**ファイル|"新規プロジェクト"ウィザード**]を使って新しいプロジェクトを作成します。それには、第 2 章 1 節の手順を実行し、プロジェクト名の "Getting Started Function" を変更します。

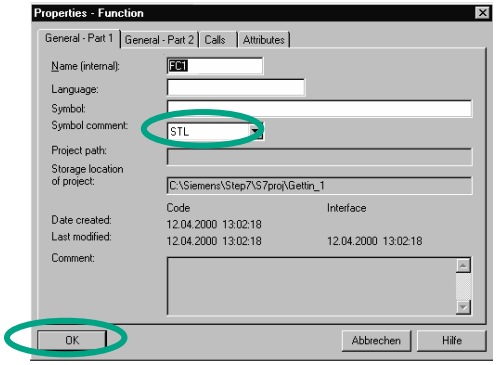
"Getting Started" プロジェクトを引き続き使用します。ただし、新しいプロジェクトを使って各ステップを実行することもできます。



Blocks フォルダへ移動し、このフォルダを開きます。

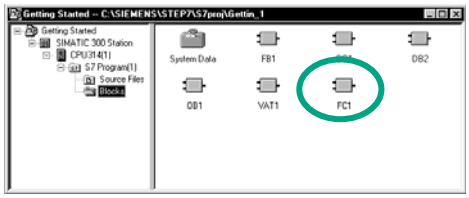
ウィンドウの右半分で右マウスボタンをクリックします。

ポップアップメニューから**ファンクション**(FC) を挿入します。



[プロパティ - ファンクション]ダイアログボックスで FC1 を選択し、さらにプログラミング言語を選択します。

その他のデフォルト設定を[**OK**]で確定します。



Blocks フォルダにファンクション FC1 が追加されます。

FC1 をダブルクリックして開きます。

ファンクションブロックとは異なり、ファンクションのスタティックデータは、変数宣言テーブルで定義することはできません。

ファンクションブロックで定義されたスタティックデータは、ブロックを閉じても保持されます。スタティックデータには、「速度」限界値で使用されるメモリビットなどがあります (第 5 章を参照)。

ファンクションをプログラミングする場合は、シンボルテーブルのシンボル名を使用できます。

詳細は、[**ヘルプ|目次**]の「オートメーションの概念の理解」、「プログラム構造の定義の基本」、「ユーザプログラムのブロック」を参照してください。

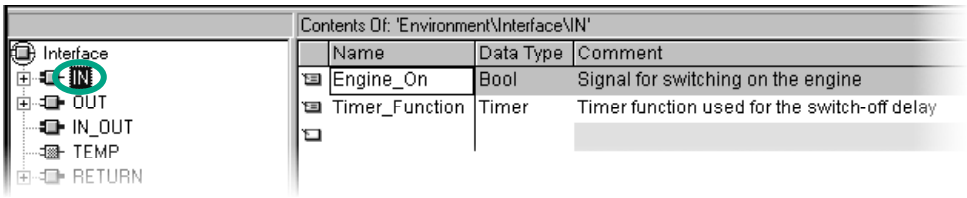
8.2 ファンクションのプログラミング

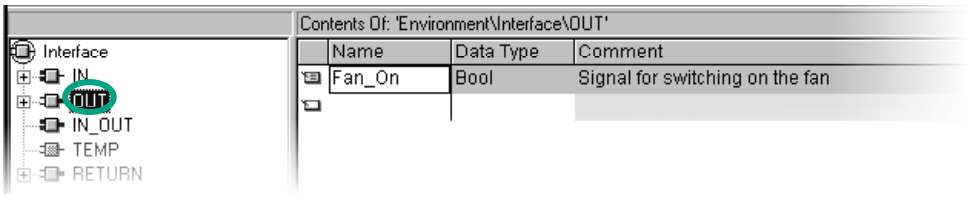
この節では、タイマファンクションのプログラミングを行います。タイマファンクションにより、エンジンの始動後すぐにファンが動作し(第 5 章を参照)、エンジンの停止後 4 秒間だけ動作するようにできます(オフディレイ)。

前述のように、ファンクションの入力/出力パラメータ("in" と "out" 宣言)は変数詳細ビューで指定する必要があります。

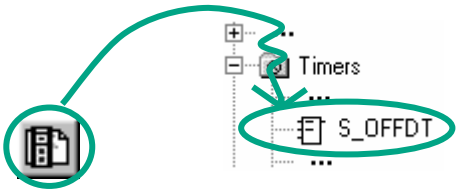
LAD/STL/FBD プログラムウィンドウが開きます。この変数詳細ビューは、ファンクションブロックの変数詳細ビューと同様に使用します(第 5 章を参照)。

次の宣言を入力します。



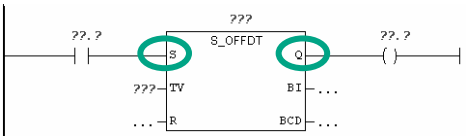


ラダーロジックによるタイマファンクションのプログラミング



ラダー命令を入力するために、制御線を選択します。

プログラムエレメントカタログで **S_OFFDT** (オフディレイタイマの起動) を選択します。

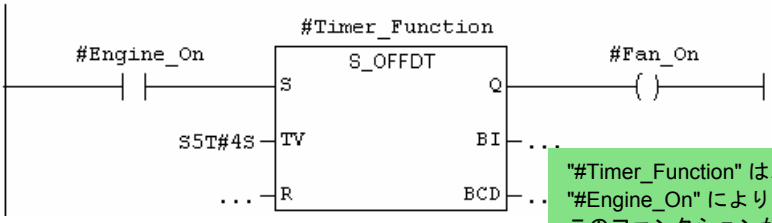


入力 **S** の前に a 接点を挿入します。
出力 **Q** の後にコイルを挿入します。

疑問符を選択し、"#"を入力してから対応する名前を選択します。

S_OFFDT の入力 TV で遅延時間を設定します。S5T#4s は、定数がデータタイプ S5Time#(S5T#) で定義されており、4 秒 (4s) 間続くことを意味します。

ファンクションを保存して、ウィンドウを閉じます。



"#Timer_Function" は、入力パラメータ "#Engine_On" により開始します。後でこのファンクションが OB1 で呼び出される場合、これはガソリンエンジンのパラメータに一度、ディーゼルエンジンのパラメータに一度提供されます(たとえば、"PE_Follow_on"の T1)。これらのパラメータの名前を後でシンボルテーブルに入力します。

ステートメントリストによるタイマファンクションのプログラミング

```
A     #Engine_On
L     S5T#4S
SF    #Timer_Function
A     #Timer_Function
=     #Fan_On
```

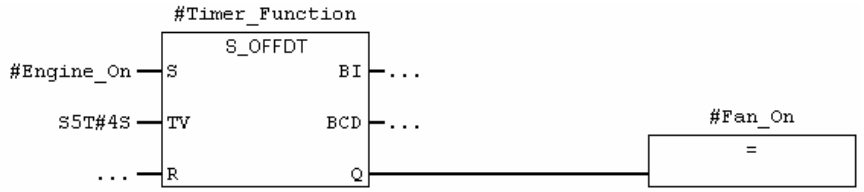
ステートメントリストでプログラミングを行う場合、ネットワークの下の入力領域を選択し、右に示すようにステートメントを入力します。

ファンクションを保存して、ウィンドウを閉じます。

ファンクションブロックダイアグラムによるタイマファンクションのプログラミング

ファンクションブロックダイアグラムでプログラミングを行う場合、ネットワークの下の入力領域を選択し、タイマファンクションの FBD プログラムを次に示すように入力します。

ファンクションを保存して、ウィンドウを閉じます。



タイマファンクションを処理するには、ブロック階層で上のレベルにあるブロック (このマニュアルの例では OB1) でこのファンクションを呼び出す必要があります。

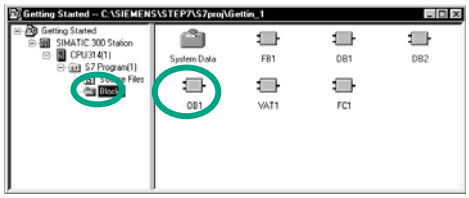
詳細は、[**ヘルプ|目次**]の「リファレンスヘルプの呼び出し」、「STL、FBD、または LAD 言語の説明」、「タイマ命令」を参照してください。

8.3 OB1 でのファンクションの呼び出し

ファンクション FC1 の呼び出しが、OB1 のファクションブロックの呼び出しに対して同様に実行されます。ファンクションのすべてのパラメータは、OB1 でガソリンエンジンまたはディーゼルエンジンの対応するアドレスに提供されます。

これらのアドレスはシンボルテーブルでは定義されていないので、アドレスのシンボル名をここで追加します。

アドレスは、STEP7 ステートメントの一部で、命令の実行対象となるプロセッサを指定します。アドレスには、絶対アドレスとシンボルアドレスがあります。



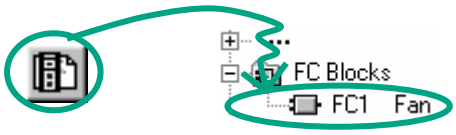
SIMATIC Manager で "Getting Started" プロジェクトまたは新規プロジェクトを開きます。

Blocks フォルダへ移動し、**OB1** を開きます。

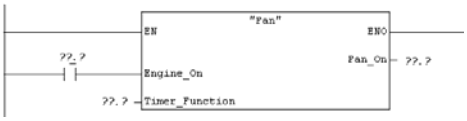
LAD/STL/FBD プログラムウィンドウが開きます。

ラダーロジックによる呼び出しのプログラミング

LAD ビューが表示されています。ネットワーク番号 5 を選択し、新規ネットワーク番号 6 を挿入します。



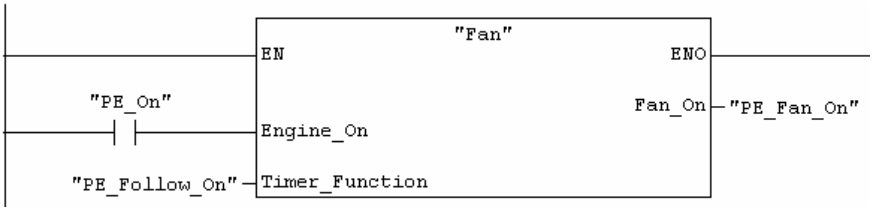
プログラムエレメントカタログの FC1 でファンクションを挿入します。



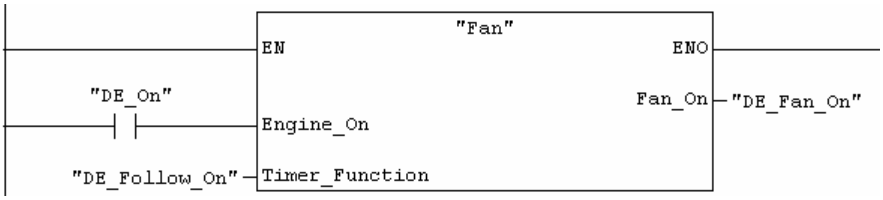
"Engine_On" の前に a 接点を挿入します。

メニューコマンド[**表示|表示|シンボル表示**]を使用すれば、シンボルアドレスと絶対アドレスを切り替えることができます。

FC1 呼び出しの疑問符をクリックして、シンボル名を挿入します。



ディーゼルエンジンのアドレスを使って、Network 7 のファンクション FC1 の呼び出しをプログラミングします。これは、前のネットワークと同じ方法でプログラミングできます (ディーゼルエンジンのアドレスはシンボルテーブルにすでに追加している)。



ブロックを保存して、ウィンドウを閉じます。

メニューコマンド[表示|**表示**|シンボル情報]を有効にして、各ネットワークの個々のアドレスに関する情報を表示します。

複数のネットワークを画面に表示するには、メニューコマンド[**表示|表示| コメント**]を無効にし、必要に応じて[**表示|表示|シンボル情報**]も無効にします。

メニューコマンド[**表示|倍率**]により、ネットワークの表示サイズを変更できます。

ステートメントリストによる呼び出しのプログラミング

```
Network 6 : Controlling the Fan for the Petrol Engine
    CALL  "Fan"
     Engine_On      :="PE_On"
     Timer_Function:="PE_Follow_On"
     Fan_On         :="PE_Fan_On"

Network 7 : Controlling the Fan for the Diesel Engine
     CALL  "Fan"
      Engine_On      :="DE_On"
      Timer_Function:="DE_Follow_On"
      Fan_On         :="DE_Fan_On"
```

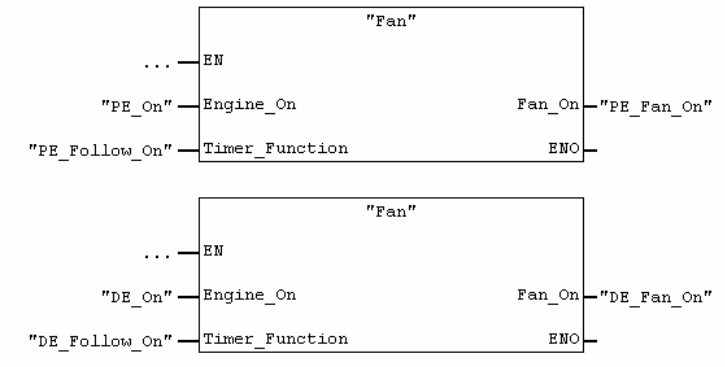
ステートメントリストでプログラミングを行う場合は、ネットワークの下の入力領域を選択し、右に示すようにSTL ステートメントを入力します。

呼び出しを保存して、ウィンドウを閉じます。

ファンクションブロックダイアグラムによる呼び出しのプログラミング

ファンクションブロックダイアグラムでプログラミングを行う場合、ネットワークの下の入力領域を選択し、次に示す FBD 命令を入力します。

呼び出しを保存して、ウィンドウを閉じます。



ファンクションの呼び出しは、例では無条件の呼び出しとしてプログラミングされます。つまり、このファンクションは常に処理されます。

オートメーションタスクの要件に従い、入力や先行する論理演算などの特定の条件により異なるファンクションまたはファンクションブロックに対して、呼び出しを作成することができます。ボックスの EN 入力と ENO 出力を使って条件をプログラミングします。

詳細は、[**ヘルプ|目次**]の「リファレンスヘルプの呼び出し」、「LAD、FBD、または STL 言語の説明」を参照してください。

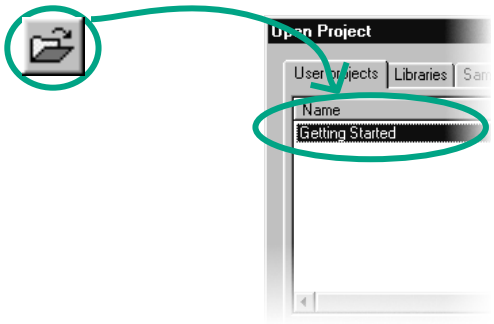
9　共有データブロックのプログラミング

9.1　共有データブロックの作成およびオープン

全データを保存するだけの内部メモリビットが CPU にない場合、特定のデータを共有データブロックに保存することができます。

共有データブロックのデータは、他のすべてのブロックで使用できます。一方、インスタンスデータブロックは特定のファンクションブロックに割り付けられ、そのデータはファンクションブロックでローカルにしか使用できません (第 5 章 5 節を参照)。

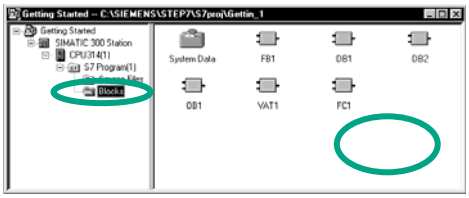
ラダーロジック、ファンクションブロックダイアグラム、ステートメントリスト(第 4 章と第 5 章を参照) の各プログラミング言語、シンボルによるプログラミング (第 3 章を参照) については、既に理解されているでしょう。



第 1 章から第 7 章まで "Getting Started" サンプルプロジェクトを使用してきた場合は、そのプロジェクトを開きます。

上記以外の場合は、SIMATIC Manager のメニューコマンド[**ファイル|"新規プロジェクト"ウィザード**]を使って新しいプロジェクトを作成します。それには、第 2 章 1 節の手順を実行し、プロジェクト名の "Getting Started Function" を変更します。

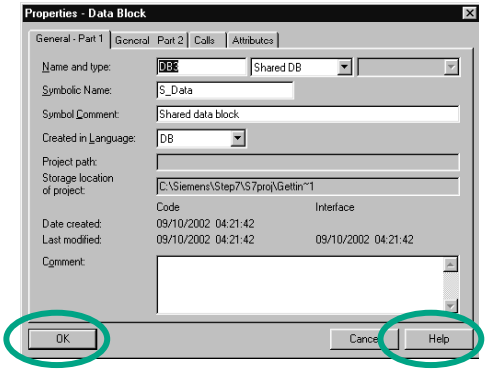
"Getting Started" プロジェクトを引き続き使用します。ただし、新しいプロジェクトを使って各ステップを実行することもできます。



Blocks フォルダへ移動し、このフォルダを開きます。

ウィンドウの右半分で右マウスボタンをクリックします。

ポップアップメニューから**データブロック**(DB)を挿入します。



[プロパティ – データブロック]ダイアログボックスで、[**OK**]をクリックしてデフォルト設定を受け取ります。

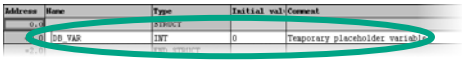
詳細を知りたい場合は、[ヘルプ]ボタンを使用します。

データブロック DB3 が **Blocks** フォルダに追加されます。

DB3 をダブルクリックして開きます。

留意事項: 第 5 章 5 節では、"ファンクションブロックを参照するデータブロック" オプションを有効にすることにより、インスタンスデータブロックを生成しました。その一方、"データブロック" を使用して共有データブロックを作成します。

データブロックでの変数のプログラミング



[Name] 列に "PE_Actual_Speed" と入力します。

ポップアップメニューのメニューコマンド[**基本タイプ|INT**]を使って、右マウスボタンをクリックしてタイプを選択します。

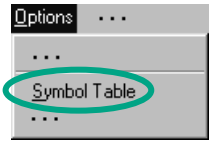
下の例では、3 つの共有データが DB3 に定義されています。これらのデータを適宜変数宣言テーブルに入力します。

Address	Name	Type	Initial val	Comment
0.0		STRUCT		
+0.0	DB_VAR	INT	0	Temporary placeholder variable
=2.0		END_STRUCT		

データブロック"PE_Actual_Speed"と"DE_Actual_Speed"の実際の速度に関する変数は、メモリワード MW2 (PE_Actual_Speed)および MW4 (DE_Actual_Speed)と同様に扱われます。これについては、次の章で説明します。

共有データブロックを保存します。

シンボルの割り付け



データブロックにはシンボル名を割り付けることもできます。

シンボルテーブルを開き、データブロック DB3 のシンボル名"S_Data"を入力します。

第 4 章でシンボルテーブルをサンプルプロジェクト(zEn01_02_STEP7__STL_1-10, zEn01_06_STEP7__LAD_1-10 または zEn01_04_STEP7__FBD_1-10)から "Getting Started" プロジェクトへコピーした場合、ここでシンボルを追加する必要はありません。

Symbol	Address	Data Type	Comment
...	...	...	...
S_Data	DB 3	DB 3	Shared data block

シンボルテーブルを保存して、[シンボルエディタ]ウィンドウを閉じます。

共有データブロックも閉じます。

変数宣言テーブルの共有データブロック:

メニューコマンド [**表示|データビュー**] により、共有データブロックのデータタイプ INT の現在値をテーブルで変更できます (第 5 章 5 節を参照)。

シンボルテーブルの共有データブロック:

シンボルテーブル内の共有データブロックのデータタイプは、インスタンスデータブロックとは異なり、常に絶対アドレスをとります。例では、データタイプは"DB3"です。インスタンスデータブロックでは、対応するファンクションブロックは常にデータタイプとして指定されます。

詳細は、[**ヘルプ|目次**]の「ブロックのプログラミング」および「データブロックの作成」を参照してください。

10 マルチプルインスタンスのプログラミング

10.1 上位ファンクションブロックの作成およびオープン

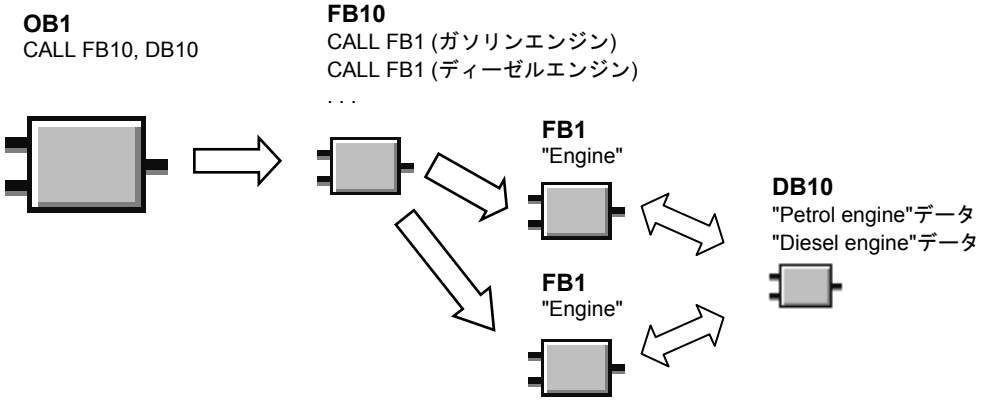
第 5 章では、エンジンを制御するためのプログラムをファンクションブロック "Engine" (FB1) を使って作成しました。オーガニゼーションブロック OB1 で呼び出されたファンクションブロック FB1 は、データブロック "Petrol" (DB1) と "Diesel" (DB2) を使用しました。これらのデータブロックにはそれぞれ、エンジンに関する異なるデータ (たとえば #Setpoint_Speed) が格納されています。

オートメーションタスクのエンジンを制御するために他のプログラム、たとえば菜種油エンジンや水素エンジンなどの制御プログラムが必要だと仮定します。

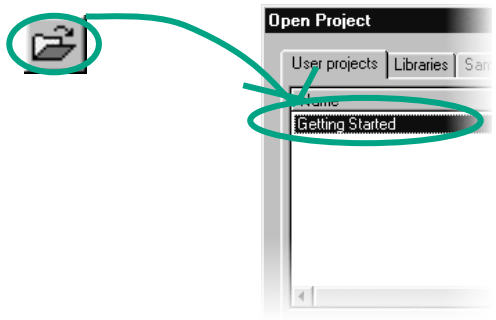
今までに学んだ手順に従い、追加の各エンジン制御プログラムに対して FB1 を使用し、このエンジンに対するデータがあるたびに新規データブロックを割り付けます。たとえば、菜種油エンジンに FB1 と DB3、水素エンジンに FB1 と DB4 などです。新規エンジン制御プログラムを作成すると、ブロックの数は著しく増加します。

一方、マルチプルインスタンスを使用する場合は、ブロック数を減らすことができます。それには、新規に上位ファンクションブロック(例では FB10)を作成し、"ローカルインスタンス"として変更されない FB1 を呼び出します。下位の FB1 が、各呼び出しに対してデータを上位の FB10 のデータブロック DB10 に格納します。つまり、FB1 にデータブロックを割り付ける必要はありません。すべてのファンクションブロックが、1 つのデータブロック(ここでは DB10)を参照します。

データブロック DB1 と DB2 は、データブロック DB10 に統合されます。それには、FB10 の静的ローカルデータで FB1 を宣言する必要があります。

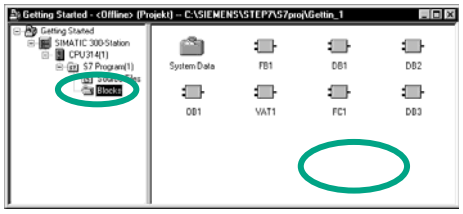


ラダーロジック、ファンクションブロックダイアグラム、ステートメントリスト(第 4 章と第 5 章を参照) の各プログラミング言語、シンボルによるプログラミング (第 3 章を参照) については、既に理解されているでしょう。



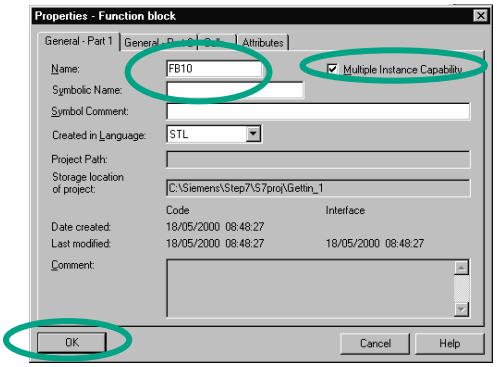
第 1 章から第 7 章まで "Getting Started" サンプルプロジェクトを使用してきた場合は、そのプロジェクトを開きます。

上記以外の場合は、SIMATIC Manager で次のプロジェクトを開きます。
ZEn01_05_STEP7__**LAD**_1-9 (ラダーロジックの場合)
ZEn01_01_STEP7__**STL**_1-9 (ステートメントリストの場合)
ZEn01_03_STEP7__**FBD**_1-9 (ファンクションブロックダイアグラムの場合)



Blocks フォルダへ移動し、このフォルダを開きます。

ウィンドウの右半分で右マウスボタンをクリックし、ポップアップメニューからファンクションブロックを挿入します。



ブロック名を FB10 に変更し、プログラミング言語を選択します。

マルチプルインスタンス FB を有効にして(必要な場合)、**[OK]**をクリックしてデフォルト設定を受け入れます。

FB10 が Blocks フォルダに追加されています。**FB10** をダブルクリックして開きます。

マルチプルインスタンスは、ファンクションブロック (たとえば値制御プログラムなどにも) に対して作成できます。マルチプルインスタンスを作成する場合は、呼び出し側と呼び出される側の両方のファンクションブロックにマルチプルインスタンス機能がなければなりません。

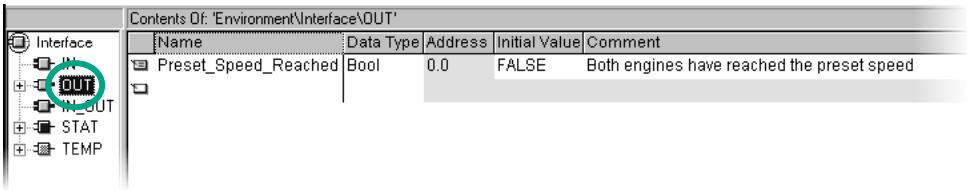
詳細は、**[ヘルプ|目次]**の「ブロックのプログラミング」および「ブロックおよびライブラリの作成」を参照してください。

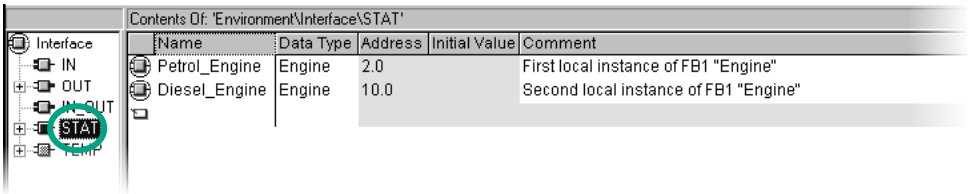
10.2 FB10 のプログラミング

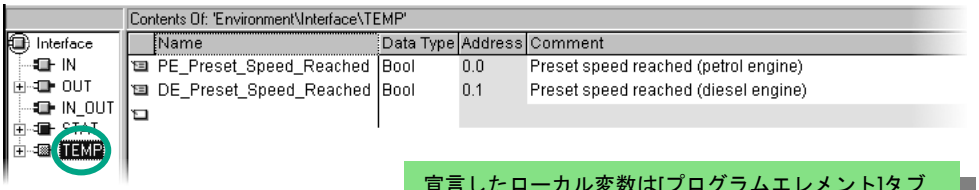
FB1 を FB10 の"ローカルインスタンス"として呼び出すには、変数詳細ビューでスタティック変数を FB1 の呼び出しごとに異なる名前で宣言する必要があります。ここではデータタイプは FB1 ("Engine")です。

変数の宣言/定義

LAD/STL/FBD プログラムウィンドウで FB10 が開きます。その後のイメージの宣言を変数詳細ビューに転送します。それには、宣言タイプ "OUT"、"STAT"、および "TEMP" を逐次選択し、変数詳細ビューでエントリを作成します。プルダウンリストから宣言タイプ "STAT" のデータタイプに "FB <nr>" を選択し、キャラクタ文字列 "<nr>" を "1" で置き換えます。

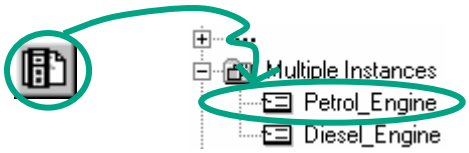






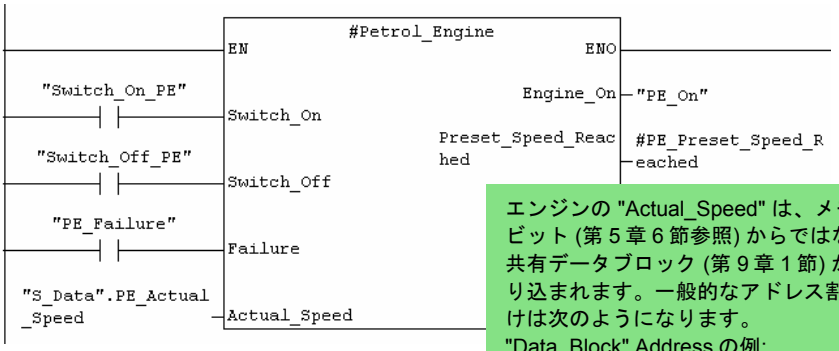
宣言したローカル変数は[プログラムエレメント]タブの[マルチプルインスタンス]に表示されます。

ラダーロジックによる FB10 のプログラミング



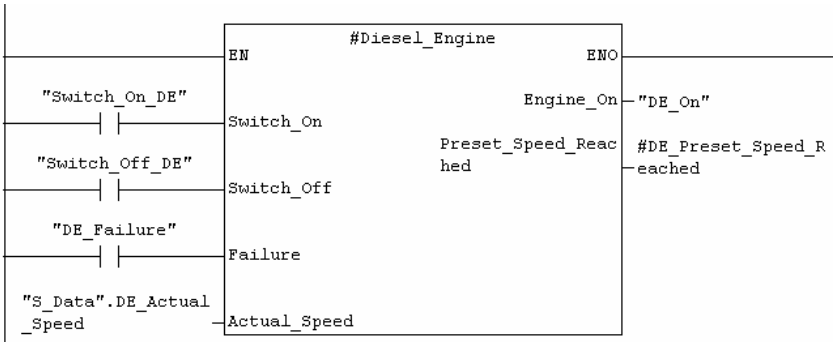
呼び出し"Petrol_Engine"をマルチプルインスタンスブロック"Petrol_Engine"として Network 1 に挿入します。

必要な a 接点を挿入し、シンボル名を指定して呼び出しを完成させます。



エンジンの "Actual_Speed" は、メモリビット (第 5 章 6 節参照) からではなく、共有データブロック (第 9 章 1 節) から取り込まれます。一般的なアドレス割り付けは次のようになります。
"Data_Block".Address の例:
"S_Data".PE_Actual_Speed

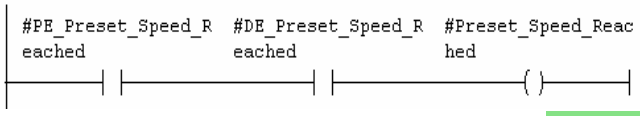
新しいネットワークを挿入し、ディーゼルエンジンの呼び出しをプログラミングします。方法は Network 1 と同じです。



新しいネットワークを挿入し、対応するアドレスで直列回路をプログラミングします。プログラムを保存し、ブロックを閉じます。

それぞれのテンポラリ変数を使用します。左に表示されるシンボルにより、プルダウンメニューのテンポラリ変数を認識できます。

プログラムを保存し、ブロックを閉じます。



一時変数("PE_Setpoint_Reached" と "DE_Setpoint_Reached") が出力パラメータ "Setpoint_Reached" に挿入され、その後、OB1 で処理されます。

ステートメントリストによる FB10 のプログラミング

```
CALL  #Petrol_Engine
 Switch_On            :="Switch_On_PE"
 Switch_Off           :="Switch_Off_PE"
 Failure              :="PE_Failure"
 Actual_Speed         :="S_Data".PE_Actual_Speed
 Engine_On            :="PE_On"
 Preset_Speed_Reached:=#PE_Preset_Speed_Reached

CALL  #Diesel_Engine
 Switch_On            :="Switch_On_DE"
 Switch_Off           :="Switch_Off_DE"
 Failure              :="DE_Failure"
 Actual_Speed         :="S_Data".DE_Actual_Speed
 Engine_On            :="DE_On"
 Preset_Speed_Reached:=#DE_Preset_Speed_Reached

A     #PE_Preset_Speed_Reached
A     #DE_Preset_Speed_Reached
=     #Preset_Speed_Reached
```

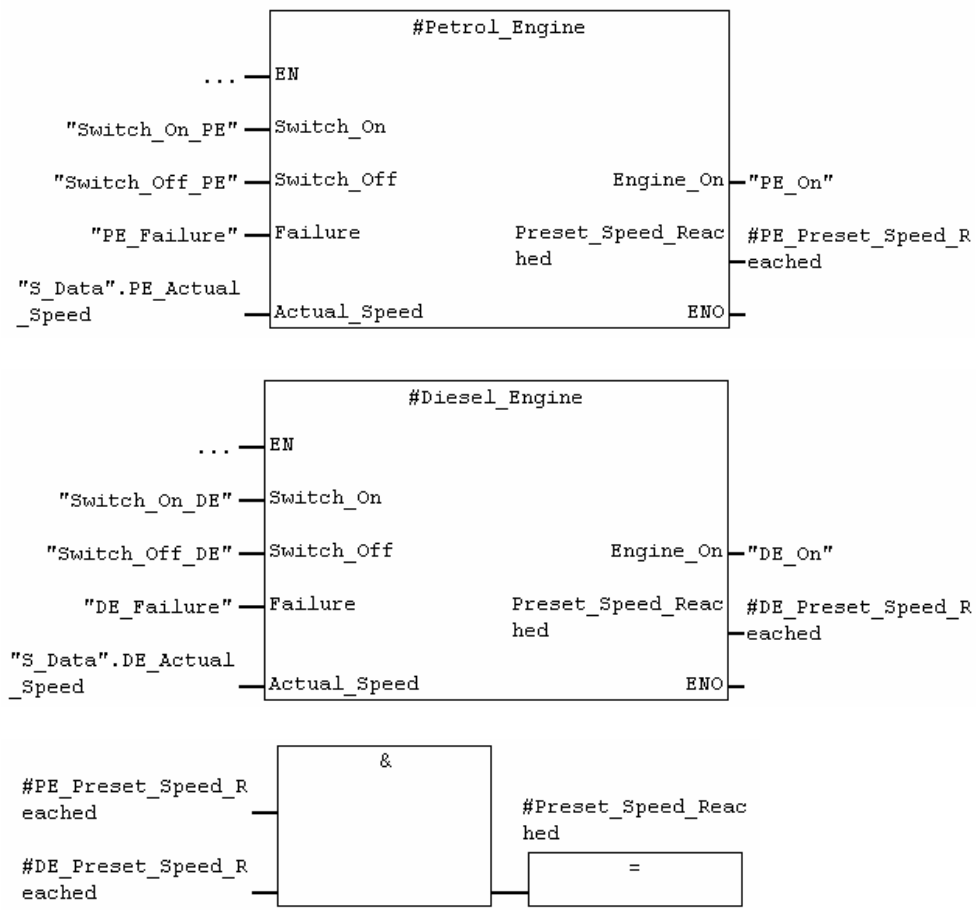
ステートメントリストでプログラミングを行う場合、ネットワークの下の入力領域を選択し、右に示すように STL 命令を入力します。

プログラムを保存し、ブロックを閉じます。

ファンクションブロックダイアグラムによる FB10 のプログラミング

ファンクションブロックダイアグラムでプログラミングを行う場合、ネットワークの下の入力領域を選択し、次に示す FBD 命令を入力します。

プログラムを保存し、ブロックを閉じます。



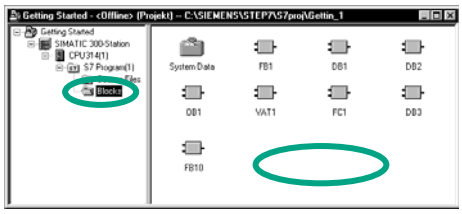
FB10 で両方の FB1 呼び出しを編集するには、FB10 自体を呼び出す必要があります。

マルチプルインスタンスは、ファンクションブロックに対してのみプログラミングできます。ファンクション (FC) に対してはプログラミングできません。

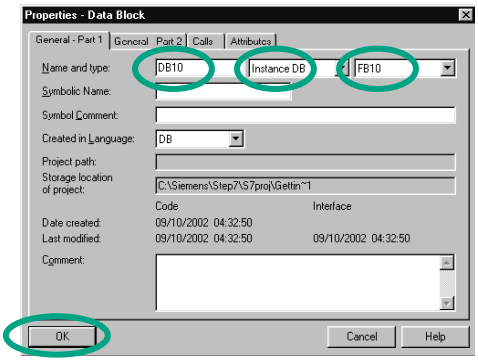
詳細は、[**ヘルプ|目次**]の「ブロックのプログラミング」、「ロジックブロックの作成、「変数宣言テーブルのマルチプルインスタンス」を参照してください。

10.3 DB10 の生成および現在値の調整

新規のデータブロック DB10 により、データブロック DB1 と DB2 が置き換えられます。ガソリンエンジンおよびディーゼルエンジンに必要なデータは DB10 に格納され、後で OB1 の FB10 の呼び出しに必要となります(第 5 章 6 節「OB1 の FB1 の呼び出し」を参照)。



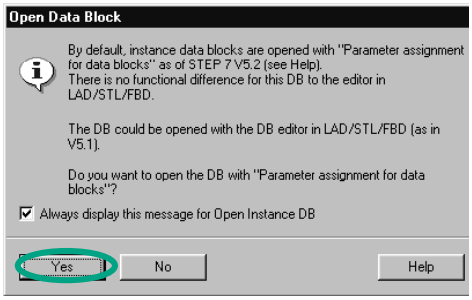
SIMATIC Manager でポップアップメニューを使用して、"Getting Started" プロジェクトの **Blocks** フォルダにデータブロック DB10 を作成します。



それには、[プロパティデータブロック]ダイアログボックスでデータブロックの名前を DB10 に適用してから、隣りのプルダウンリストでアプリケーション "インスタンス DB" を選択します。右のプルダウンリストで、割り付けられるファンクションブロック"FB10"を選択し、さらに[**OK**]で残りの設定を確定します。

データブロック DB10 が"Getting Started"プロジェクトに追加されました。

DB10 をダブルクリックします。



その後のダイアログボックスで、[**はい**] で応答してインスタンス DB を開きます。ここでメニューコマンド **[表示| データ表示]** を選択します。

データビューには、FB1 (ローカルインスタンス) の 2 つの呼び出しの「内部変数」など、DB10 の各変数が表示されます。

宣言ビューには、FB10 で宣言された変数が表示されます。

ディーゼルエンジンの現在値を "1300" に変更し、ブロックを保存してから閉じます。

	Address	Declaration	Name	Type	Initial value	Actual value	Comment
1	0.0	out	Preset_Speed_Reached	BOOL	FALSE	FALSE	Both engines have reached the preset speed
2	2.0	stat:in	Petrol_Engine.Switch_On	BOOL	FALSE	FALSE	Switch on engine
3	2.1	stat:in	Petrol_Engine.Switch_Off	BOOL	FALSE	FALSE	Switch off engine
4	2.2	stat:in	Petrol_Engine.Failure	BOOL	FALSE	FALSE	Engine failure, causes the engine to switch off
5	4.0	stat:in	Petrol_Engine.Actual_Speed	INT	0	0	Actual engine speed
6	6.0	stat:out	Petrol_Engine.Engine_On	BOOL	FALSE	FALSE	Engine is switched on
7	6.1	stat:out	Petrol_Engine.Preset_Speed_Reached	BOOL	FALSE	FALSE	Preset speed reached
8	8.0	stat	Petrol_Engine.Preset_Speed	INT	1500	1500	Requested engine speed
9	10.0	stat:in	Diesel_Engine.Switch_On	BOOL	FALSE	FALSE	Switch on engine
10	10.1	stat:in	Diesel_Engine.Switch_Off	BOOL	FALSE	FALSE	Switch off engine
11	10.2	stat:in	Diesel_Engine.Failure	BOOL	FALSE	FALSE	Engine failure, causes the engine to switch off
12	12.0	stat:in	Diesel_Engine.Actual_Speed	INT	0	0	Actual engine speed
13	14.0	stat:out	Diesel_Engine.Engine_On	BOOL	FALSE	FALSE	Engine is switched on
14	14.1	stat:out	Diesel_Engine.Preset_Speed_Reached	BOOL	FALSE	FALSE	Preset speed reached
15	16.0	stat	Diesel_Engine.Preset_Speed	INT	1500	1300	Requested engine speed

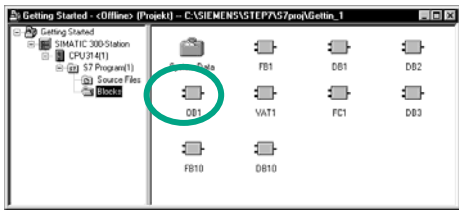
すべての変数は、DB10 の変数宣言テーブルに含まれます。前半ではファンクションブロック "Petrol_Engine" を呼び出すための変数を確認し、後半ではファンクションブロック "Diesel_Engine" を呼び出すための変数を確認することができます(第 5 章 5 節を参照)。

FB1 の "内部" 変数には、"Switch_On" などのシンボル名があります。ローカルインスタンスの名前は、これらの名前の前にあります。たとえば、"Petrol_Engine.Switch_On" となります。

詳細は、[**ヘルプ|目次**]の「ブロックのプログラミング」および「データブロックの作成」を参照してください。

10.4 OB1 での FB10 の呼び出し

FB10 呼び出しは、本書のサンプルでは OB1 で作成しました。この呼び出しは、OB1 での FB1 のプログラミングおよび呼び出しを行った際に使用したファンクションと同じファンクションです (第 5 章 6 節を参照)。マルチプルインスタンスを使用すれば、第 5 章 6 節でプログラミングした Networks 4 と 5 を取りかえることができます。



FB10 をプログラミングしたプロジェクトで **OB1** を開きます。

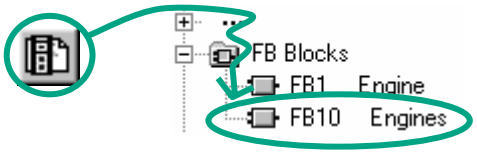
シンボル名の定義

LAD/STL/FBD プログラムウィンドウが開きます。メニューコマンド[**オプション|シンボルテーブル**]を使ってシンボルテーブルを開き、ファンクションブロック FB10 とデータブロック DB10 のシンボル名をシンボルテーブルに入力します。

シンボルテーブルを保存し、ウィンドウを閉じます。

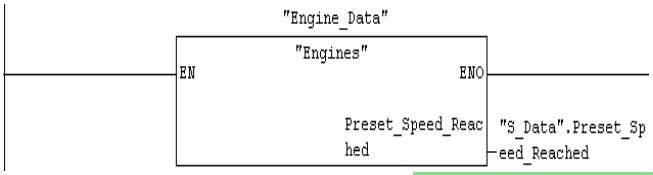
Symbol	Address	Data Type	Comment
...	...	...	...
Engines	FB 10	FB 10	Example of multiple instances
Engine_Data	DB 10	FB 10	Instance data block for FB10 10
...	...	...	...

ラダーロジックによる呼び出しのプログラミング



OB1 の最後に新しいネットワークを挿入し、**FB10** ("Engines")の呼び出しを追加します。

対応するシンボル名を指定して呼び出しを完成させます。
ここでは FB10 を介して FB1 を呼び出すので、OB1 の FB1 呼び出しは削除します (第 5 章 6 節の Networks 4 と 5)。
プログラムを保存し、ブロックを閉じます。



FB10 ("Engines")の出力信号"Setpoint_Reached"が、共有データブロックの変数に渡されます。

ステートメントリストによる呼び出しのプログラミング

ステートメントリストでプログラミングを行う場合は、新しいネットワークの下の入力領域を選択し、次に示すように STL 命令を入力します。それには、プログラムエレメントカタログで[**FB ブロック|FB10 エンジン**]を使用します。

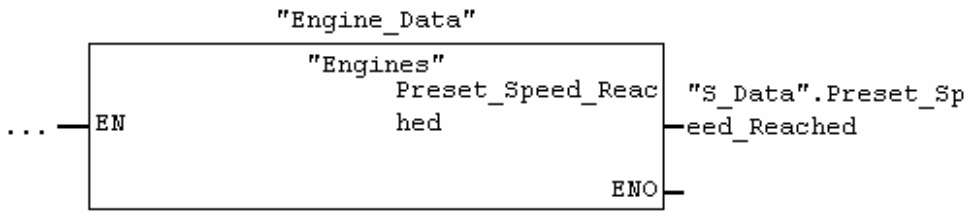
ここでは FB10 を介して FB1 を呼び出すので、OB1 の FB1 呼び出しは削除します (第 5 章 6 節の Networks 4 と 5)。

プログラムを保存し、ブロックを閉じます。

```
CALL  "Engines" , "Engine_Data"
 Preset_Speed_Reached:="S_Data".Preset_Speed_Reached
```

ファンクションブロックダイアグラムによる呼び出しのプログラミング

ファンクションブロックダイアグラムでプログラミングを行う場合は、新しいネットワークの下の入力領域を選択し、次に示すように FBD 命令を入力します。それには、プログラムエレメントカタログで[**FB ブロック|FB10 エンジン**]を使用します。
ここでは FB10 を介して FB1 を呼び出すので、OB1 の FB1 呼び出しは削除します (第 5 章 6 節の Networks 4 と 5)。
プログラムを保存し、ブロックを閉じます。



オートメーションタスクに対して追加のエンジン制御プログラムが必要な場合(ガソリンエンジン、水素エンジン用など)、これらを同様にマルチプルインスタンスとしてプログラミングし、FB10 から呼び出すことができます。

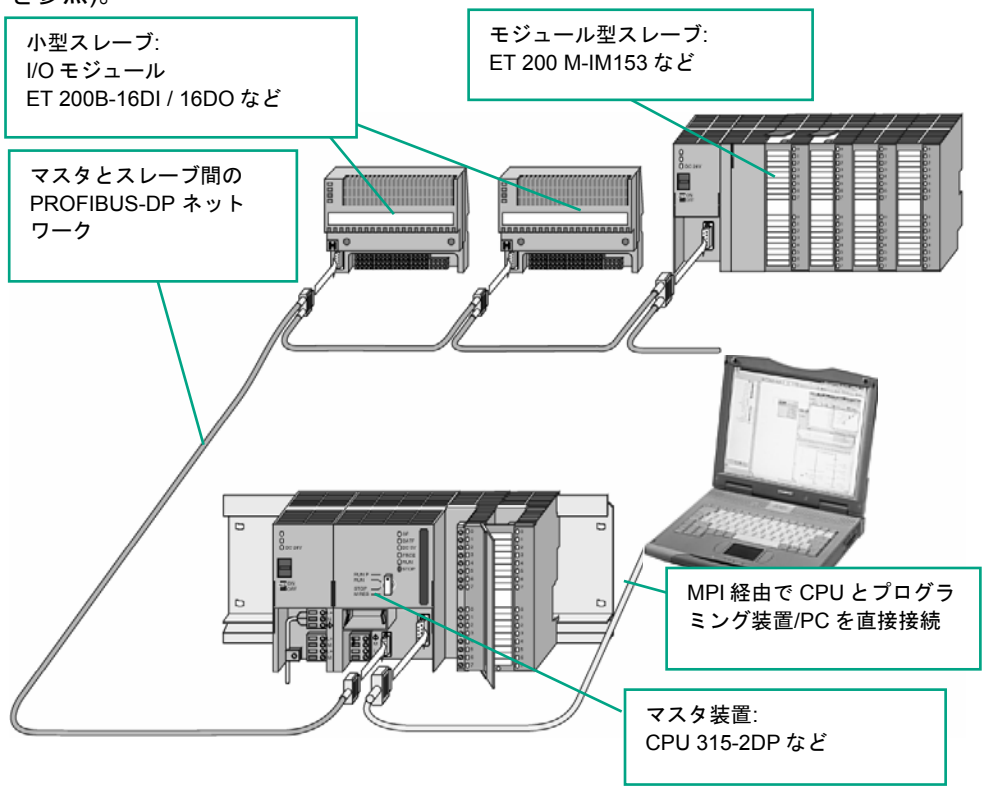
それには、追加のエンジンを FB10 ("Engines") の変数宣言テーブルで宣言し、FB10 で FB1 呼び出しをプログラミングします (プログラムエレメントカタログのマルチプルインスタンス)。新規シンボル名を、たとえばシンボルテーブルの始動プロシージャおよび停止プロシージャに対して定義することができます。

詳細は、[**ヘルプ|目次**]の「リファレンスヘルプの呼び出し」、「STL、FBD、または LAD 言語の説明」を参照してください。

11　リモート I/O のコンフィグレーション

11.1 PROFIBUS DP によるリモート I/O のコンフィグレーション

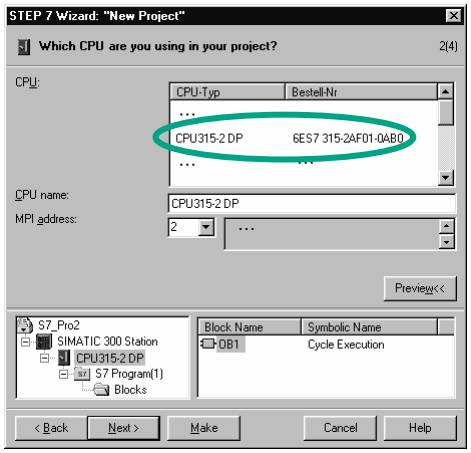
従来のコンフィグレーションを用いたオートメーションシステムは、センサやアクチュエータをケーブル接続し、それをプログラマブルロジックコントローラ本体に直接組み込んでいました。この方法では往々にして、配線に手間がかかります。
リモートコンフィグレーションを使用すれば、入出力モジュールをセンサやアクチュエータの近くに配置できるので、配線の手間を大幅に省くことができます。プログラマブルロジックコントローラ、I/O モジュール、フィールド装置は PROFIBUS DP を使って相互に接続することができます。
従来のコンフィグレーションのプログラミング方法については、第 6 章を参照してください。基本コンフィグレーションおよびリモートコンフィグレーションでは、その作成方法に違いはありません。使用するモジュールをハードウェアカタログから選択し、それらのモジュールをラックに配置し、ユーザの条件に合わせてモジュールのプロパティを設定するだけです。
プロジェクトの作成および基本コンフィグレーションのプログラミングに関する知識がある方は、この章の内容は容易に理解できます (第 2 章 1 節および第 6 章を参照)。



新規プロジェクトの作成

SIMATIC Manager を起動します。作業が円滑に進むように、開いているオブジェクトはすべて閉じておきます。

新規プロジェクトを作成します。

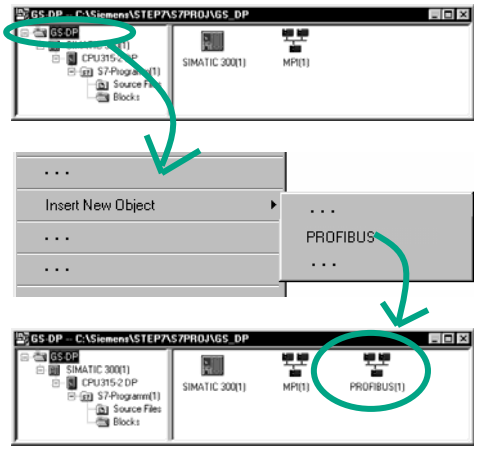


対応するダイアログボックスで **CPU 315-2DP** を選択します(PROFIBUS-DP ネットワーク装備の CPU)。

第 2 章 1 節の手順を実行し、プロジェクトの名前として"GS-DP" (Getting Started – Distributed I/O)を割り付けます。

ここでユーザ自身のコンフィグレーションを作成したい場合は、CPU を指定します。この場合、リモート I/O に対応した CPU でなければなりません。

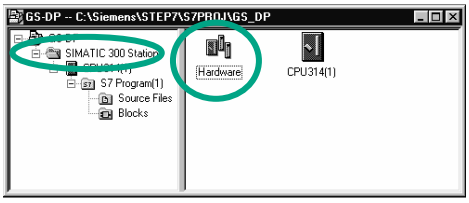
PROFIBUS ネットワークの挿入



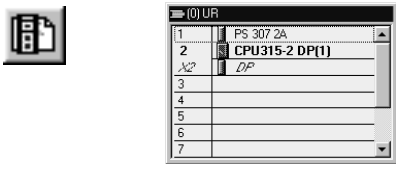
GS-DP フォルダを選択します。

ウィンドウの右半分でマウスの右ボタンを使用して、**PROFIBUS** ネットワークを挿入します。

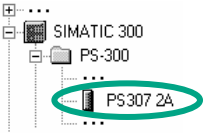
ステーションのコンフィグレーション



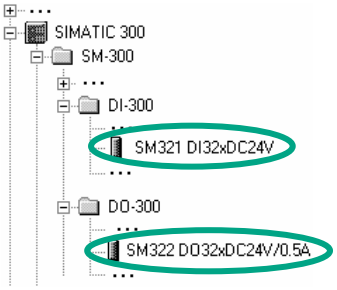
SIMATIC 300 Station フォルダを選択し、**Hardware** をダブルクリックします。
"HW Config" ウィンドウが開きます(第 6 章 1 節を参照)。



CPU 315-2 DP がラックに表示されています。必要に応じて、メニューコマンド[**表示|ハードウェアカタログ**]またはツールバーのボタンを使ってハードウェアカタログを開きます。



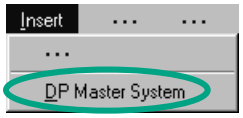
電源モジュール **PS307 2A** をスロット 1 へドラッグ&ドロップします。



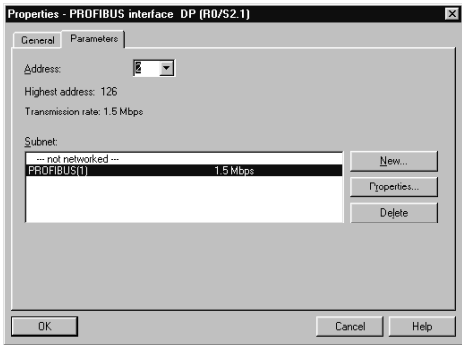
同様の方法で、I/O モジュール **DI32xDC24V** と **DO32xDC24V/0.5A** をスロット 4 と 5 に挿入します。

リモート I/O に対応する CPU と同じラックに、他の CPU を挿入することもできます (説明は省略)。

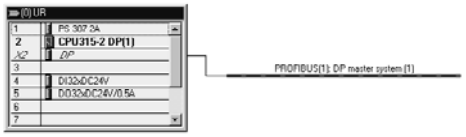
DP マスタシステムのコンフィグレーション



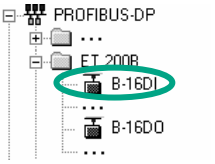
スロット 2.1 の DP マスタを選択し、**DP マスタシステム**を挿入します。



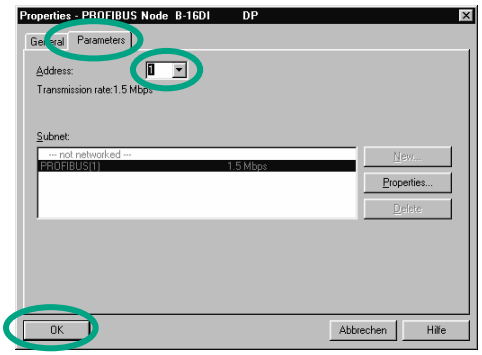
示されたアドレスを表示されたダイアログボックスに適用します。[サブネット]フィールドで"PROFIBUS(1)"を選択してから、[**OK**]で設定を適用します。



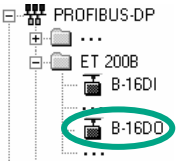
マスタシステムに配置するオブジェクトは、左マウスボタンを押したままドラッグすれば移動できます。



ハードウェアカタログ内のモジュール **B-16DI** をマスタシステムに挿入します(オブジェクトをマスタシステムへドラッグし、カーソルの形が"+"記号に変わったらオブジェクトをドロップします)。

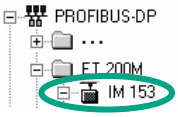


挿入したモジュールのノードアドレスは、[プロパティ]ダイアログボックスの[パラメータ]タブで変更できます。アドレスが示されるので、[**OK**]をクリックして確定します。



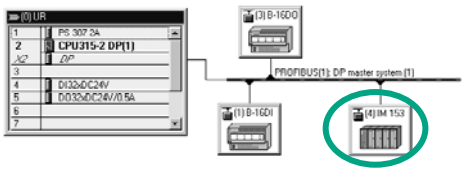
同様の方法で、モジュール **B-16DO** をマスタシステムにドラッグ&ドロップします。

ノードアドレスは、ダイアログボックスで自動的に調整されます。[**OK**] をクリックして、このエントリを確定します。

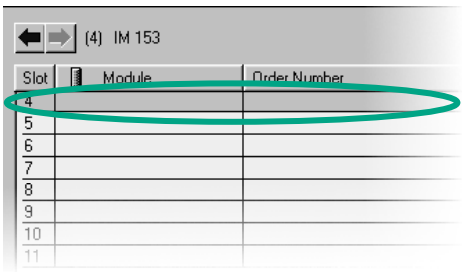


インターフェースモジュール **IM153** をマスタシステムにドラッグ&ドロップし、[**OK**]をクリックしてノードアドレスを確定します。

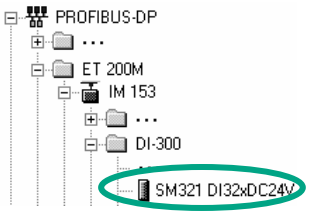
本書のサンプルでは、デフォルトノードアドレスを使用します。ただし、これらのアドレスは、ユーザの条件に合うようにいつでも変更できます。



ネットワークで **ET200M** を選択します。
ET200M 用の空スロットが下部のコンフィグレーションテーブルに表示されます。



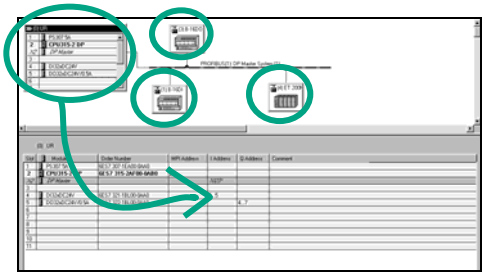
ここではスロット **4** を選択します。



ET200M 自体に I/O モジュールを追加することができます。たとえば、スロット 4 に **DI32xDC24V** モジュールを選択し、このモジュールをダブルクリックして挿入します。

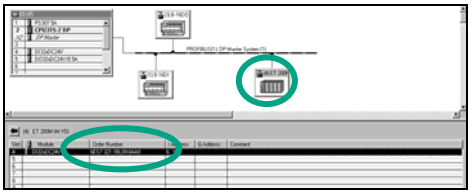
ハードウェアカタログを使用する際は、正しいフォルダを選択していることを常に確認してください。たとえば、ET200M のモジュールを選択する場合は、ET200M フォルダを使用します。

ノードアドレスの変更



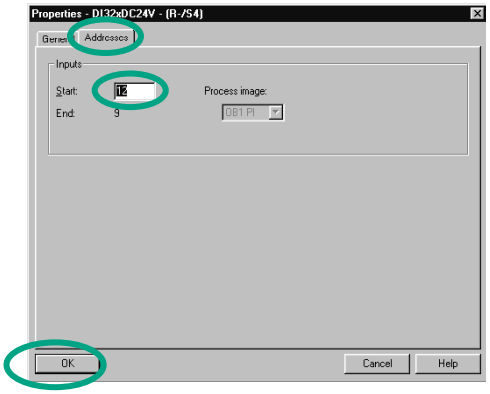
本書のサンプルでは、ノードアドレスを変更する必要はありません。ただし、一般的には変更しなければならない場合がよくあります。

その他のノードを 1 つずつ選択し、入力アドレスと出力アドレスをチェックします。"Configuring Hardware" アプリケーションがすべてのアドレスを調整するので、重複割り付けは起こりません。



ET200M のアドレスを変更する場合を考えましょう。

ET200M を選択し **DI32xDC24V**、(スロット 4)をダブルクリックします。



[プロパティ] ダイアログボックスの [アドレス] タブで入力アドレスを 6 から **12** に変更します。
[**OK**] をクリックして、ダイアログボックスを閉じます。

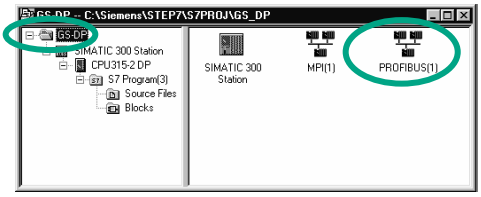
最後に、リモート I/O コンフィグレーションを**保存し、コンパイル**します。

ウィンドウを閉じます。

メニューコマンド [**保存とコンパイル**] では、コンフィグレーションの一貫性チェックが自動的に実行されます。エラーがなければ、システムデータが生成され、プログラマブルコントローラにダウンロードすることができます。

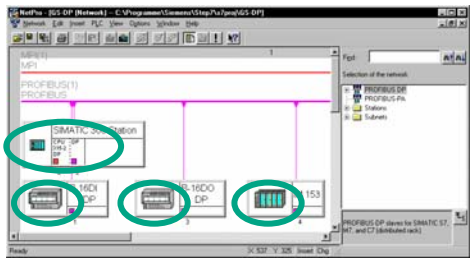
[**保存**]を使用すると、エラーがあってもコンフィグレーションを保存できます。ただし、そのコンフィグレーションをプログラマブルコントローラにダウンロードすること

オプション: ネットワークのコンフィグレーション



リモート I/O は、オプションパッケージ "Configuring Networks" を使ってコンフィグレーションすることもできます。

SIMATIC Manager でネットワーク PROFIBUS (1)をダブルクリックします。



[NetPro] ウィンドウが開きます。

追加の DP スレーブを、ネットワークオブジェクトのカタログから PROFIBUS DP へドラッグ&ドロップできます。

エレメントをコンフィグレーションする場合は、そのエレメントをダブルクリックします。[ハードウェアのコンフィグレーション] ウィンドウが開きます。

メニューコマンド[**ステーション|一貫性チェック**] ([ハードウェアのコンフィグレーション]ウィンドウ)と[**ネットワーク|一貫性チェック**] ([ネットワークのコンフィグレーション]ウィンドウ)を使用すれば、コンフィグレーションを保存する前にエラーをチェックできます。エラーがあれば表示され、可能な対策が示されます。

詳細は、[**ヘルプ|目次**]の「ハードウェアのコンフィグレーション」および「リモート I/O のコンフィグレーション」を参照してください。

おめでとうございます! 『Getting Started』マニュアルを修了し、STEP 7 の最も重要な用語、手順、およびファンクションを学習しました。これで最初のプログラムを開始できます。

プロジェクトに取り組む中で、STEP 7 の特定のファンクションを調べたくなったり、操作方法を忘れてしまった場合は、「STEP 7 のヘルプ」を参照してください。詳細な情報を得ることができます。

STEP 7 に関する知識を深めたい場合は、専門的なトレーニングコースが多数用意されています。最寄の弊社担当部署までお問い合わせください。

プロジェクトの大成功をお祈り申し上げます!

Siemens AG

付録 A

本書で使用するサンプルプロジェクトの概要

- **ZEn01_02_STEP7__STL_1-10:**
  第 1 章から第 10 章で使用。STL プログラミング言語のシンボルテーブルが含まれる。
- **ZEn01_01_STEP7__STL_1-9:**
  第 1 章から第 9 章で使用。STL プログラミング言語のシンボルテーブルが含まれる。
- **ZEn01_06_STEP7__LAD_1-10:**
  第 1 章から第 10 章で使用。LAD プログラミング言語のシンボルテーブルが含まれる。
- **ZEn01_05_STEP7__LAD_1-9:**
  第 1 章から第 9 章で使用。LAD プログラミング言語のシンボルテーブルが含まれる。
- **ZEn01_04_STEP7__FBD_1-10:**
  第 1 章から第 10 章で使用。FBD プログラミング言語のシンボルテーブルが含まれる。
- **ZEn01_03_STEP7__FBD_1-9:**
  第 1 章から第 9 章で使用。FBD プログラミング言語のシンボルテーブルが含まれる。
- **ZEn01_07_STEP7__Dist_IO:**
  第 11 章のリモート I/O で使用。

索引

と

ね

の

は

ふ

へ

ま

も

ら

り