

COGNEX

DataMan[®] 通信およびプログラミングガイド

10/26/2017

5.7.0.37

法律上の注意

本書で説明する本ソフトウェアは、ライセンスのもとで提供されており、かかるライセンス条項にしたがい、かつ本ページに記されている著作権表示を行うことを条件として使用および複製が許諾されます。本ソフトウェア、本書、あるいはそれらのコピーをライセンスの保有者以外に提供することはできません。本ソフトウェアのあらゆる権利は Cognex Corporation あるいは権限を有する第三者に帰属します。Cognex Corporation は、当社の供給によらない装置上での本ソフトウェアの使用または信頼性についていかなる責任も負いません。Cognex Corporation は、本ソフトウェアの商業性、非侵害または特定目的に対する適合性に関して、明示的にも黙示的にも一切の保証を行いません。

本書の内容は、予告なしに変更することがあります。本書のいかなる内容も、Cognex Corporation が何らかの約束を実行するものと解釈されてはなりません。本書あるいは関連するソフトウェアに含まれる誤りについて、Cognex Corporation はいかなる責任も負いません。

本書で例として使用されている企業名、人名およびデータは、特に断りがない限り架空のものであります。本書のいかなる部分も、Cognex Corporation の書面による許諾なく複製すること、手段・方法・目的を問わず電子的または機械的に転載すること、他のメディアに変換すること、および他の言語に翻訳することを禁じます。

Copyright © 2017. Cognex Corporation. All Rights Reserved.

Cognex の提供するハードウェアおよびソフトウェアの一部は、下記のウェブサイトを示す米国およびその他の国の特許、または出願中の特許で保護されています。<http://www.cognex.com/patents>

下記は Cognex Corporation の登録商標です。

Cognex, 2DMAX, Advantage, AlignPlus, Assemblyplus, Check it with Checker, Checker, Cognex Vision for Industry, Cognex VSOC, CVL, DataMan, DisplayInspect, DVT, EasyBuilder, Hotbars, IDMax, In-Sight, Laser Killer, MVS-8000, OmniView, PatFind, PatFlex, PatInspect, PatMax, PatQuick, SensorView, SmartView, SmartAdvisor, SmartLearn, UltraLight, Vision Solutions, VisionPro, VisionView

下記は Cognex Corporation の商標です。

Cognex ロゴ, 1DMax, 3D-Locate, 3DMax, BGAll, CheckPoint, Cognex VSoC, CVC-1000, FFD, iLearn, In-Sight (クロスヘアデザインおよび記号), In-Sight 2000, InspectEdge, Inspection Designer, MVS, NotchMax, OCRMax, PatMax RedLine, ProofRead, SmartSync, ProfilePlus, SmartDisplay, SmartSystem, SMD4, VisiFlex, Xpand

その他の製品名および商標は、各所有者の商標です。

目次

法律上の注意	2
目次	3
シンボル	5
このマニュアルについて	6
ネットワークの形成	7
DataMan とネットワークの接続	7
DataMan 有線式リーダをネットワークに接続	7
DataMan インテリジェントベースステーションとネットワークの接続	7
コンピュータへの直接接続	8
サブネットを越えたリーダの接続	14
DataMan をネットワークにワイヤレス接続	14
ネットワーク接続に関するトラブルシューティング	17
産業用ネットワークプロトコル	18
EtherNet/IP	18
DMCC	18
リーダ設定コード	18
Setup Tool	19
はじめに	19
オブジェクトモデル	22
Rockwell ControlLogix の例	33
Rockwell CompactLogix の例	46
SLMP プロトコル	56
DMCC	56
リーダ設定コード	56
Setup Tool	57
SLMP プロトコルスキャナ	57
はじめに	57
ネットワーク設定	58
データブロック設定	59
インタフェース	60
サンプル	68
ModbusTCP	72
DMCC	72
リーダ設定コード	72
Setup Tool	73
Modbus TCP ハンドラ	73
はじめに	73
ネットワーク設定	74
データブロック設定	75
インタフェース	76
操作	81
サンプル	84
PROFINET	87
DMCC	87
リーダ設定コード	87

Setup Tool	88
はじめに	88
モジュール	92
操作	97
Siemens の例	99
ワイヤレス DataMan 向けの産業用プロトコル	108
プロトコル操作	108
Ethernet アドレス	108
PLC からのトリガ	108
ソフトイベント	108
DMCC	108
オフラインバッファ処理	109
産業用プロトコルのステータス	109
DataMan アプリケーション開発	111
DMCC の概要	111
コマンド構文	111
コマンドヘッダ構文	111
ヘッダの例	111
DMCC アプリケーション開発	113
DataMan SDK コンテンツ	113
SDK の使用	113
ヘルパユーティリティ	117
ヘルパユーティリティの使用	117
スクリプトベースのデータフォーマット	118
DMCC のサポート	118
エラー管理	121
スクリプトフォーマット	121
Output	131
コードのオートコンプリートとスニペット	133
カスタム通信プロトコル API	135
イベントコールバック	155
イベントタイプ	155
例	156

シンボル

次のシンボルは、安全上の注意事項および追加情報を示します。

 **警告：** 死亡、重症または感電の原因になる危険性があることを示します。

 **注意：** 物的損害につながる可能性があることを示します。

 **注：** 説明に関する追加情報を提供します。

 **ヒント：** 理解を助ける示唆が示されています。

このマニュアルについて

『DataMan 通信・プログラミングガイド』には、DataMan リーダをご使用の環境に組み込む方法に関して、次の情報が記載されています。

- [ネットワーク設定](#)
- [産業用ネットワークプロトコル](#)
- [PLC との統合](#)
- [DataMan Control Commands \(DMCC\) API](#)

ネットワークに接続された DataMan リーダには、複数の方法で画像を取り込みのためにトリガをかけることができます。以下の方法で実行可能です。

- DataMan Setup Tool
- 次を利用してトリガ可能
 - トリガビット
 - 産業用プロトコルを経由/使用してオブジェクトを操作
 - DMCC コマンドの使用

本書には、各方法に関する詳細な説明が記載されています。

ネットワークの形成

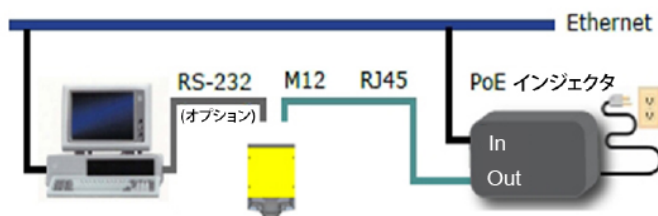
シンプルな Ethernet 接続を経由して DataMan デバイスを接続することができます。DataMan デバイスの IP アドレスおよびサブネットマスクを手動で設定することも、DHCP を使用して自動的に設定することもできます。

DataMan とネットワークの接続

DataMan 有線式リーダをネットワークに接続

PoE インジェクタを使ってリーダの電源を入れます。Cognex で推奨している接続手順は次のとおりです。

1. PoE インジェクタを Ethernet ネットワークに接続します (パッチケーブルの両端共に接続)。
2. 電源コード (AC 230V/110V) を PoE インジェクタにつなげます。
3. リーダと PoE インジェクタを接続します。



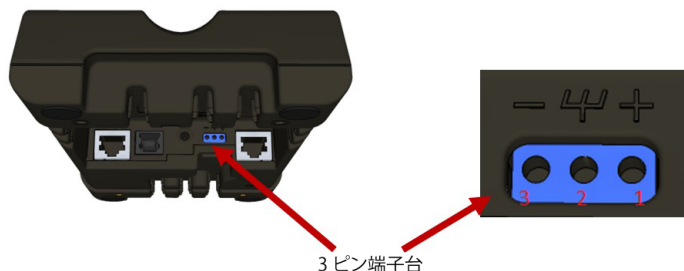
リーダの電源を切るには

1. リーダと PoE インジェクタの接続を切断します。
2. PoE インジェクタから電源コードを抜きます。
3. PoE インジェクタと Ethernet ネットワークの接続を切断します。

DataMan インテリジェントベースステーションとネットワークの接続

1. DMA-IBASE-00 を使用する場合、次の 2 つのオプションの 1 つを使用してベースステーションに電源を投入します。
 - Ethernet ケーブルを直接 PC のネットワークに接続する場合は、24V 電源を使用してベースステーションに電源を投入します。

- PoE 電源アダプタを使用する場合は、PoEでベースステーションに電源を投入できます。**DMA-IBASE-BT-01** ベースステーションを使用する場合は、24V 電源に直接接続してください。**DMA-IBASE-BT-01** には 3 ピン端子台が装備されています。



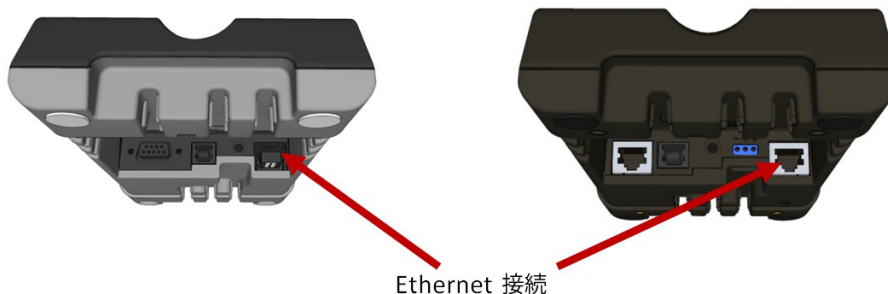
ピン番号	信号
1	+24V
2	シールド
3	GND

① 注：端子台と、DCプラグコネクタ電源を同時に接続しないでください。

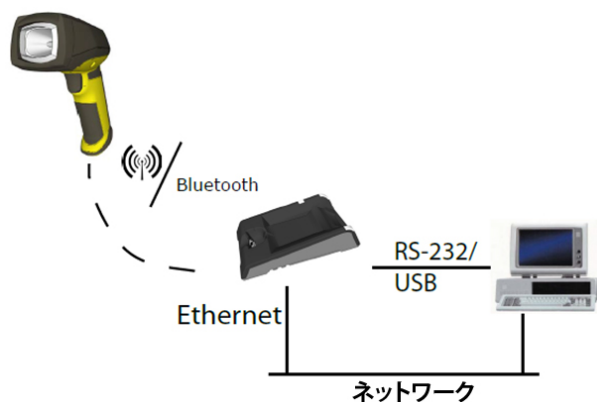
2. Ethernet ケーブルでベースステーションを PC に接続します。

DMA-IBASE-00

DMA-IBASE-BT-01



3. ベースステーションは、Ethernet を経由して接続されると表示され、ワイヤレス (WiFi または Bluetooth) インタフェースを介してデータをリーダにルーティングします。



コンピュータへの直接接続

DataMan デバイスを PC の Ethernet ポートに直接接続するときには、PC および DataMan デバイスの両方を同じサブネットに設定する必要があります。これは、リンクローカルアドレスによって自動的に実行することも、リーダおよび PC を手動で設定することでも

きます。

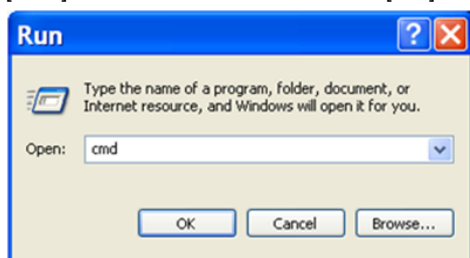
リンクローカルアドレスでは、IP アドレスを自動的に要求して割り当てます。DataMan Setup Tool では、これは DHCP サーバ通信オプションに相当します。これはデフォルトであり、変更を行う必要はありません。

DataMan デバイスが PC と同じサブネット上に入るように手動で設定することもできますし、PC を DataMan と同じサブネット上に入るように設定することもできます。これらのオプションについては、以下の節で詳しく説明しています。

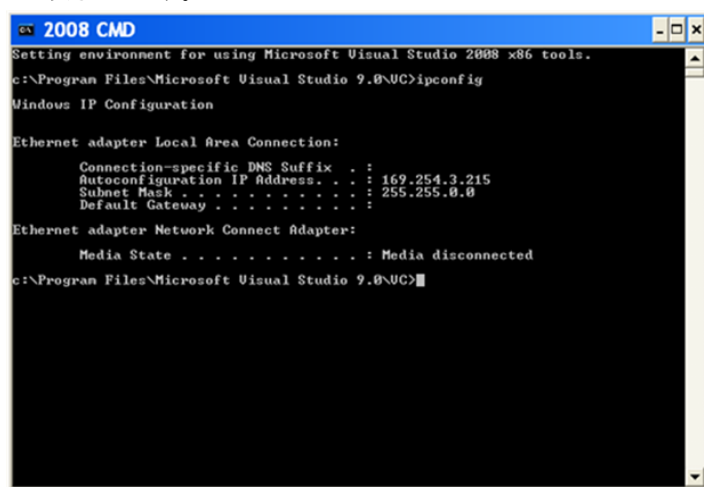
PC と同じサブネットに DataMan を入れるための設定

以下の手順で DataMan デバイスを設定します。

1. **ipconfig** ユーティリティを使用して、お使いの PC の IP アドレスとサブネットマスクを調べます。[スタート] メニューで [ファイル名を指定して実行] をクリックします。
2. [名前] フィールドに「cmd」と入力して [OK] をクリックします。



3. コマンドプロンプトウィンドウに、「ipconfig」と入力して Enter キーを押します。PC 上のすべてのネットワークアダプタのリストが表示されます。



4. PC の IP アドレスとサブネットマスクを記録します。この例では、
 - IP アドレスは 169.254.3.215
 - サブネット マスクは 255.255.0.0

5. [リーダメンテナンス] に移動して、DataMan Setup Tool で使用するデバイスをリストから選択し、[ネットワーク設定] ダイアログを使用して対象の DataMan リーダのネットワーク設定を手動で行います。

ネットワーク設定

☐ DHCPサーバを使用する
☒ 固定 IP アドレスを使用する

IP アドレス

サブネットマスク

デフォルトゲートウェイ

デバイス名

認証

ユーザー名

パスワード

適用

6. DataMan のネットワーク設定を固定するには
- [固定 IP アドレスを使用する] を選択します。
 - PC と同じサブネットに入れる [IP アドレス] と [サブネットマスク] を入力します。この IP アドレスがまだ使用されていないことを確認します (例えば、ping を実行してテストします)。
 - IP アドレスの例: 169.254.135.200
 - サブネットマスク: 255.255.0.0

注: デフォルトのサブネットマスクは 255.255.255.0 です。[DataMan の設定を工場出荷時のデフォルト値にリセットする] 設定コードをスキャンすると、このデフォルト値に戻すことができます。

DataMan で認証が有効になっていない場合は、認証は空欄のままにしておく必要があります。認証はデフォルトで無効になっています。

ネットワーク設定

☐ DHCP サーバを使用する
☒ 固定 IP アドレスを使用する

IP アドレス

サブネットマスク

デフォルトゲートウェイ

デバイス名

認証

ユーザー名

パスワード

適用

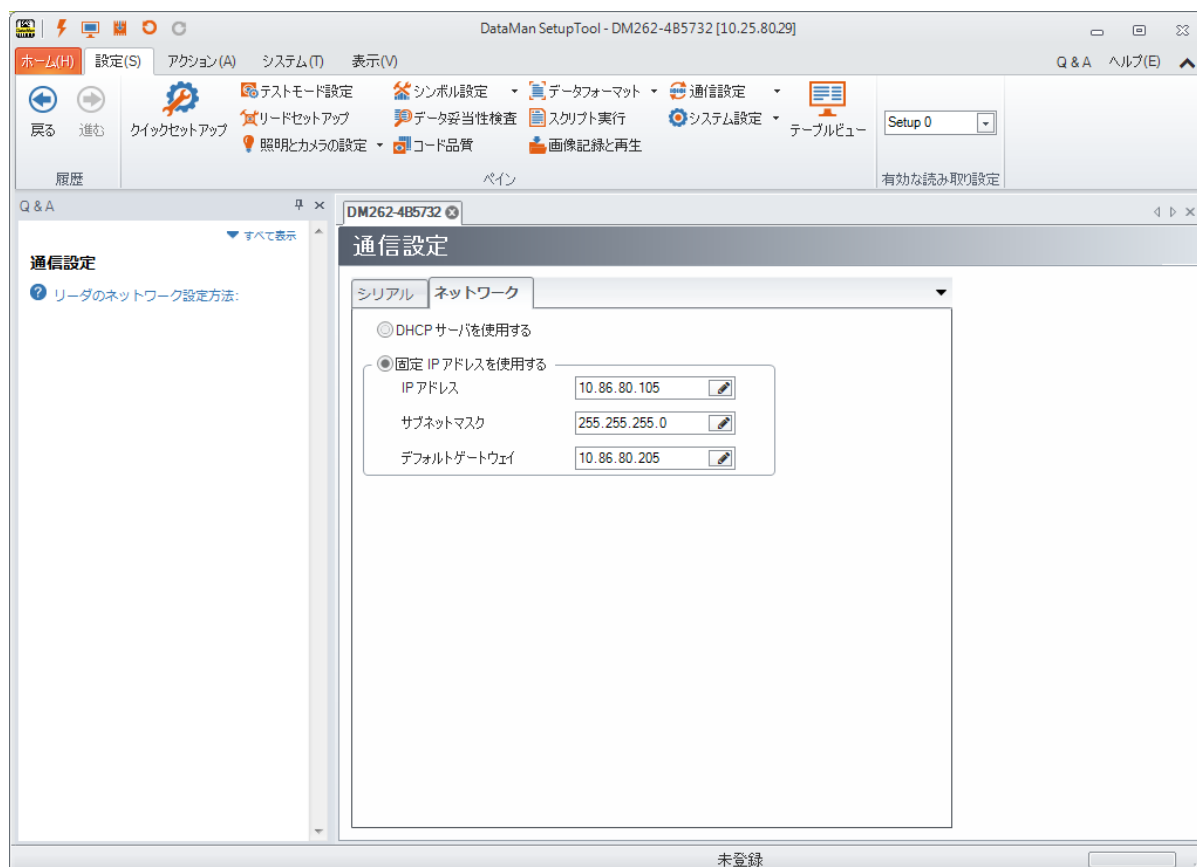
[OK] をクリックします。DataMan デバイスが指定したネットワーク設定に設定され、自動的に再起動されます。

アドレスが解決されると、**[検出したデバイス]** ノードに DataMan リーダが表示されます。表示されるまでに最大 60 秒かかります。

8. 1、2 分経過してもデバイスが表示されない場合は、DataMan Setup Tool の **[接続]** ページにある **[更新]** ボタンを押します。これによって、PC に接続されたか、または同じネットワークに接続された DataMan デバイスがあるかどうか、DataMan Setup Tool で強制的にスキャンされます。

DataMan と同じサブネットに PC を入れるための設定

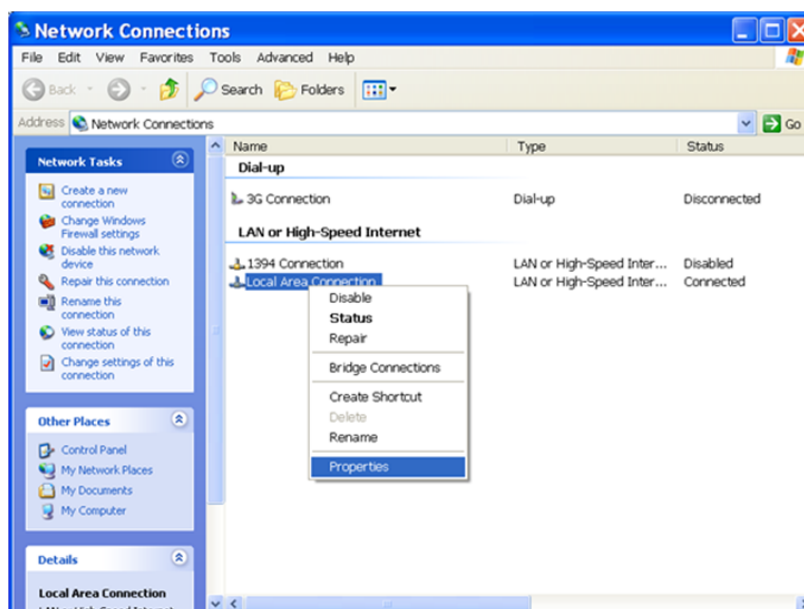
DataMan のネットワーク設定を変更しない場合は、DataMan の IP アドレスとサブネットマスクがすでにわかっているか、または RS-232 を経由して DataMan に接続し、IP アドレスとサブネットマスクを見つける必要があります。DataMan の IP アドレスとサブネットマスクは、**[通信設定]** で見ることができます。



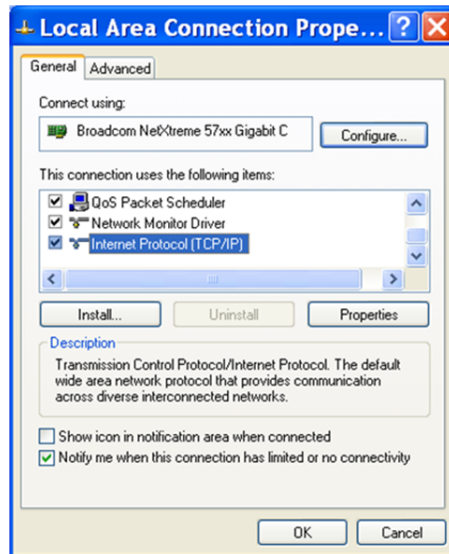
DataMan デバイスの IP アドレスとサブネットマスクがわかったら、PC のネットワーク設定を変更することができます。

次の手順を実行して PC を設定します (ここにある例は Windows XP のものです)。

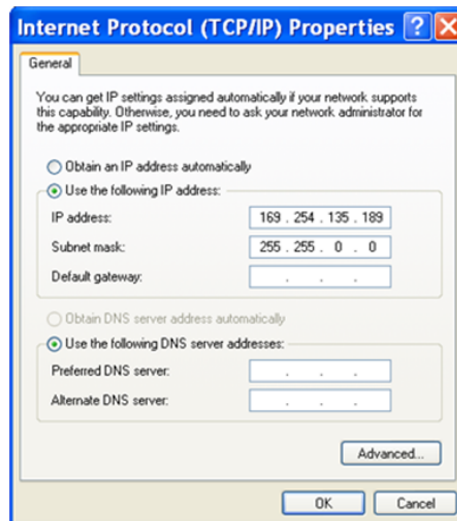
1. [スタート] メニューで [マイ ネットワーク] を右クリックし、[プロパティ] メニューオプションをクリックして [ネットワーク接続] を起動します。
2. DataMan に接続されているネットワークアダプタを右クリックし、[プロパティ] メニューオプションを選択します。



3. [全般] タブをスクロールダウンして [インターネット プロトコル (TCP/IP)] を選択し、[プロパティ] をクリックします。



4. [全般] タブで [次の IP アドレスを使う] オプションを選択し、DataMan と同じサブネットの IP アドレスとサブネットマスクを入力します。[OK] をクリックします。

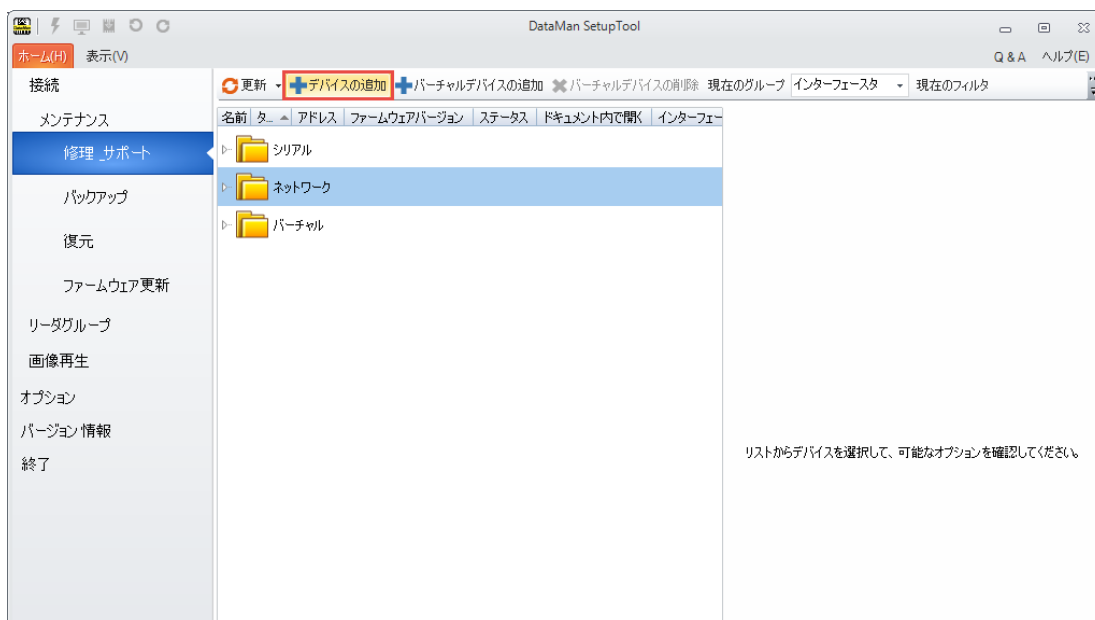


5. [閉じる] をクリックします。PC のネットワーク設定が、指定した新しい値に変わります。
6. DataMan デバイスを再起動します。ネットワークアドレスが解決されると、[接続] ページの [検出したデバイス] ノードに DataMan デバイスが表示されます。
7. 1、2 分経過してもデバイスが表示されない場合は、DataMan Setup Tool の [接続] ページにある [更新] ボタンをクリックします。PC に接続されたか、または同じネットワークに接続された DataMan デバイスがあるかどうか、DataMan Setup Tool でスキャンされます。

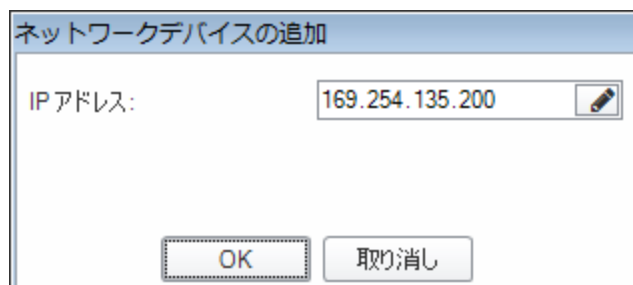
サブネットを越えたリーダの接続

デバイスの IP アドレスがすでにわかっている場合、DataMan Setup Tool の次のオプションを使用して異なるサブネットの DataMan デバイスに接続できます。

1. DataMan Setup Tool の [リーダメンテナンス] ページで [ネットワークデバイスの追加] をクリックします。



2. 対象の DataMan デバイスの実際の IP アドレスを入力します。



3. [OK] をクリックします。[ネットワーク] ノードにリーダが表示されます。新しいノードをダブルクリックするか、またはこれを選択して [接続] ボタンをクリックします。このデバイスが使用可能であれば、リーダが接続されます。

DataMan をネットワークにワイヤレス接続

DataMan リーダをワイヤレスネットワークで接続することもできます。そのためには、Wi-Fi スライドインを使用する必要があります。

アドホック接続

デバイスの工場デフォルトのワイヤレス設定は次のとおりです。

- アドホック接続
- 暗号化なし、認証なし
- SSID: デバイス名

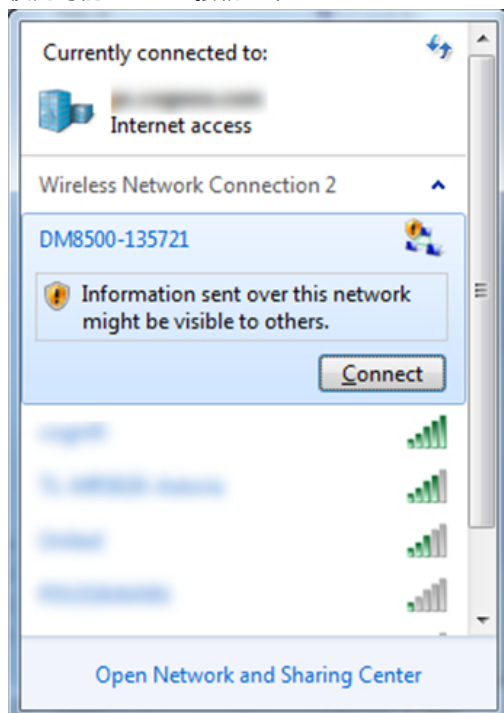
つまり、ベースステーションやルータなしで DataMan に接続できます。

アドホックモード



DataMan リーダをアドホックモードで接続するには、次の手順を実行します。

1. DataMan デバイスを工場出荷時の設定に戻します。
2. 使用可能な Wi-Fi 接続の中から DataMan デバイスを探し、接続します。



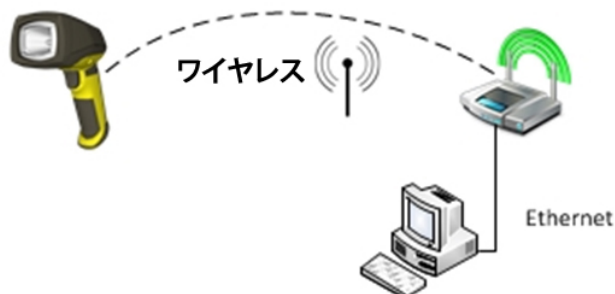
3. DataMan Setup Tool を開きます。
4. デバイスを検索して接続します。
5. DataMan Setup Tool で DataMan デバイスを接続した後、ワイヤレス接続を設定できます。
 - a. 認証: オープンモードのみ選択可能です。
 - b. 暗号化方式: WEP-40 と WEP-104 が使用できます。パスフレーズを入力できます。

インフラストラクチャモード

DataMan Setup Tool ではワイヤレスインフラストラクチャモードも設定できます。

1. DataMan Setup Toolでお使いのデバイスに接続します。
2. [通信設定] の [WiFi] タブで、[ネットワークの種類] コンボボックスから [Infrastructure mode (インフラストラクチャモード)] を選択します。SSID の名前がデバイス名と同じである場合、デバイスの設定が間違っているとして、警告が表示されます。

インフラストラクチャモード



3. 次の認証モードの中から選択します。

認証モード	暗号化モード	要件
オープンシステム	WEP-40、WEP-104	パスフレーズ
WPA-PSK、WPA2-PSK	TKIP、AES、TKIP/AES	パスフレーズ
EAP-TLS (次の節を参照)	TKIP、AES、TKIP/AES	- クライアントの証明書 - CA の証明書 - クライアントのプライベートキー - クライアントのユーザ名
PEAP-MSCHAPV2 (次の節を参照)	TKIP、AES、TKIP/AES	- CA の証明書 - クライアントのユーザ名 - クライアントのパスワード

EAP-TLS 認証モード

暗号化方式: TKIP、AES、TKIP/AES がサポートされています。これらのすべての方式で、複数の PEM ファイルを指定する必要があります。これらのファイルはユーザのローカルシステム管理者が作成し、証明書情報が含まれています。

これらの証明書を使用して、Wi-Fi アクセスポイントとリーダ間の通信が暗号化されます。

必要な証明書は次のとおりです。

- クライアントの証明書。リーダごとに異なる必要があります。公開可能 (例: 企業の Web ページ)。
- CA の証明書 (CA = 認証局)。社内の認証サーバごとに 1 つのファイルが作成されます。公開可能。
- クライアントのプライベートキー。リーダごとに異なる必要があります。公開してはいけません。機密情報として保存し、取り扱う必要があります。

DataMan への証明書ファイルのアップロード

DataMan Setup Tool を使用すれば、これらのファイルを 1 つずつアップロードできます。フィールドの横のフォルダボタンをクリックして該当するファイルを選択すると、ファイルコンテンツがデバイスにアップロードされます。

証明書が指定されている場合は簡潔なメッセージが表示されます。キーまたは証明書が指定されていない場合はフィールドに「<not set> (<未設定>)」と表示されます。

Security Options	
Authentication mode	EAP-TLS
Encryption method	TKIP/AES
Client's certificate	Certificate (1388 bytes)
Client's private key	Private key (637 bytes)
CA's certificate	Certificate (907 bytes)
Client's user name	dm8x00-sn02

DataMan からの証明書ファイルの削除

対応するフィールドの横の赤色の [X] ボタンをクリックすると、デバイスから既存の証明書が削除されます。証明書はデバイスのバックアップに保存されるので、完全に復元することができます。

PEAP-MSCHAPV2 認証モード

暗号化方式: TKIP、AES、TKIP/AES がサポートされています。これらのすべての方式で、CA の証明書、クライアントのユーザ名、パスワードを含む PEM ファイルが必要です。

Security Options	
Authentication mode	PEAP-MSCHAPV2
Encryption method	TKIP/AES
CA's certificate	Certificate (907 bytes)
Client's user name	dm8x00-sn02
Client's password	*****
Client's password again	*****
Client passwords do not match	

証明書ファイル

DataMan Setup Tool では現在、PEM ファイルについて次の制限を適用しています。

- ・フォーマットは業界標準の PEM フォーマット (OpenSSL ツールキットで生成) である必要があります。
- ・PEM は PKCS8 または SSLeay である必要があります。
- ・暗号化されていないプライベートキーと証明書ファイルのみが許可されています。
- ・クライアントのプライベートキーと証明書に含まれるセクションは 1 つである必要があります。CA の証明書には 1 つまたは複数のセクションが含まれていてもかまいません。
- ・ユーザは自身の証明書ファイル内に格納されているユーザ名を把握し、[Client's user name (クライアントのユーザ名)] テキストボックスに同じ名前を正確に入力する必要があります。つまり、Setup Tool は証明書ファイルを参照せず、このユーザ名情報を抽出しません。
ユーザが [Wireless (ワイヤレス)] タブページを離れると、再起動確認ウィンドウがポップアップ表示され、設定がデバイスに保存されます。

ネットワーク接続に関するトラブルシューティング

現在のネットワーク設定で DataMan Setup Tool がお使いのリーダと通信できず、リーダがネットワークデバイスのリストに表示されなくなることがあります。リーダの IP アドレスがわかっている場合は、DataMan Setup Tool の [ネットワークデバイスの追加] オプションを使用します。この方法で DataMan リーダがネットワークデバイスのリストに表示され、DataMan Setup Tool で接続できるようになります。

産業用ネットワークプロトコル

DataMan では、標準の Ethernet プロトコルをベースとする産業用ネットワークプロトコルを使用しています。EtherNet/IP、PROFINET、SLMP (MC プロトコル) および Modbus/TCP プロトコルは、標準の Ethernet よりも信頼性が向上しています。

EtherNet/IP

DataMan は EtherNet/IP™ をサポートしています。これは、Common Industrial Protocol (CIP) をベースとするアプリケーションレベルのプロトコルです。EtherNet/IP™ はより幅広いメッセージ伝達を可能にし、Ethernet 経由でのデータ転送および入出力サービスを実行します。EtherNet/IP ネットワーク上のすべてのデバイスは、属性と呼ばれる一連のデータ値として、データをネットワークに提示します。属性は、関連するデータ値とともにグループ化してセットにすることができ、これらをアセンブリと呼びます。

デフォルトでは、DataMan デバイスは EtherNet/IP プロトコルが無効になっています。このプロトコルを有効にするには、DMCC を送信するか、パラメータコードをスキャンするか、または DataMan Setup Tool を使用します。

① 注： ワイヤレス DataMan リーダをお持ちの場合は、「[ワイヤレス DataMan 向けの産業用プロトコル](#)」の節を参照してください。

DMCC

次のコマンドを使用して、DataMan リーダで EtherNet/IP を有効または無効にすることができます。このコマンドは RS-232 または Telnet 接続を経由して送信することができます。

① 注： DataMan と通信するには、Windows が提供する Telnet クライアントの設定を変更する必要があるため、PuTTY などのサードパーティのクライアントを使用することを推奨します。

有効:

```
||>SET ETHERNET-IP.ENABLED ON
||>CONFIG.SAVE
||>REBOOT
```

無効:

```
||>SET ETHERNET-IP.ENABLED OFF
||>CONFIG.SAVE
||>REBOOT
```

リーダ設定コード

次のリーダ設定コードをスキャンすると、有線式リーダで EtherNet/IP が有効化または無効化されます。

① 注： 変更を反映するにはデバイスを再起動する必要があります。

有効:



無効:



次のリーダ設定コードをスキャンすると、DataMan 8000 ベースステーションで EtherNet/IP が有効化または無効化されます。

① 注： 変更を反映するにはデバイスを再起動する必要があります。

有効:



無効:



Setup Tool

EtherNet/IP を有効にするには、産業用プロトコルの [EtherNet/IP] タブで **[有効]** をオンにします。リーダから切断する前に、[設定の保存] を選択して新しい選択内容を保存してください。

① 注： 新しい設定はリーダの再起動後に反映されます。

はじめに

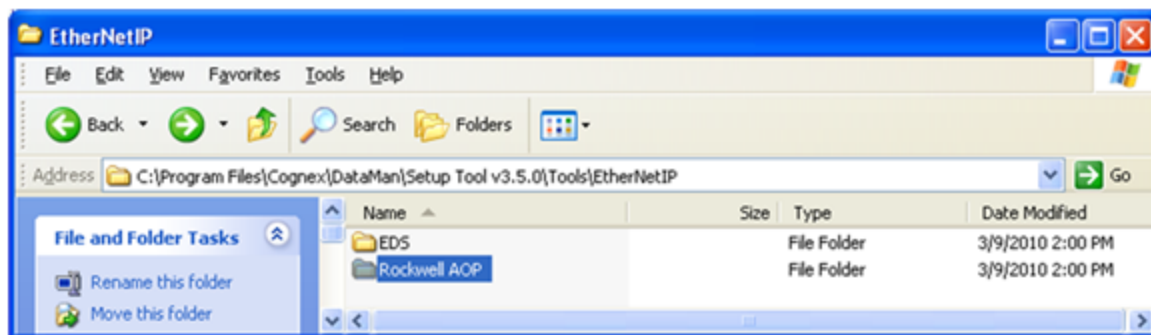
EtherNet/IP を使用する準備として、主に次の手順を実行します。

- Rockwell ソフトウェア ツールがお使いのマシン上にインストールされていることを確認します。
- Rockwell ソフトウェア ツールがお使いの DataMan デバイスを認識するように設定します。
- DataMan リーダ用の DataMan Electronic Data Sheet (EDS) をインストールします。

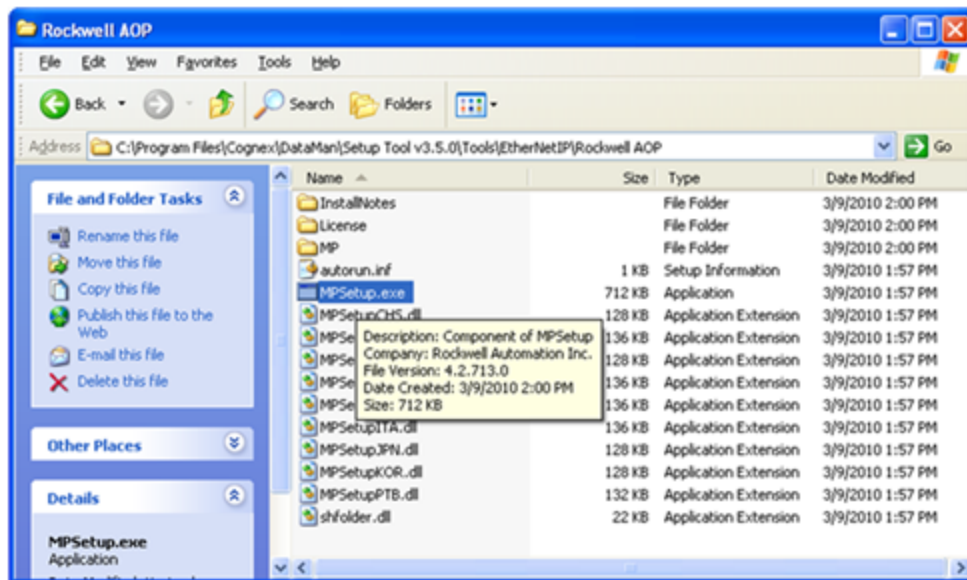
次の手順を実行して、EtherNet/IP を設定します。

1. Rockwell ソフトウェアが PC 上にインストールされていることを確認します。
2. **[Add on Profile (アドオンプロファイル)]** のインストールおよび **[Samples (サンプル)]** のインストールを選択していることを確認します。アドオンプロファイルは、Rockwell ControlLogix または CompactLogix PLC でのみ使用します。
3. 次のディレクトリに移動して、Rockwell アドオンプロファイルをインストールします。

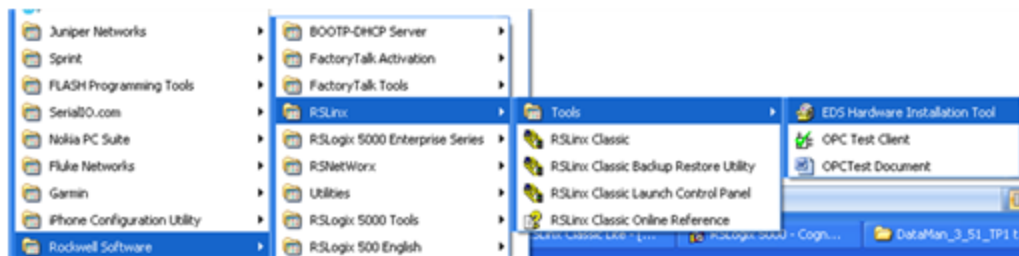
① 注： 使用している DataMan Setup Tool のバージョンに合わせてパスを調整します。



4. Rockwell AOP ディレクトリを選択します。



5. まだ Rockwell AOP をインストールしていない場合は、MPSetup.exe を実行します。
6. [スタート] メニューから [プログラム] → [Rockwell Software (Rockwell ソフトウェア)] → [Tools (ツール)] → RSLinx → [EDS Hardware Install Tool (EDS ハードウェアインストールツール)] と進みます。



7. EDS インストールツールを実行します。

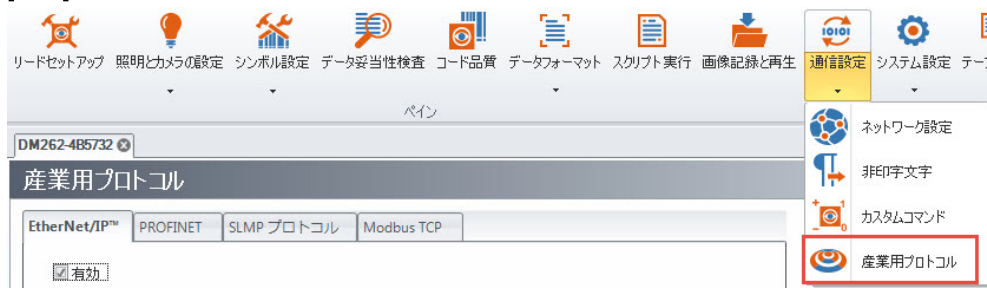
① 注： 既存の EDS ファイルがある場合は、それを先にアンインストールしてから、最新バージョンの EDS をインストールします。

8. DataMan Setup Tool を実行し、DataMan ファームウェアを更新します。

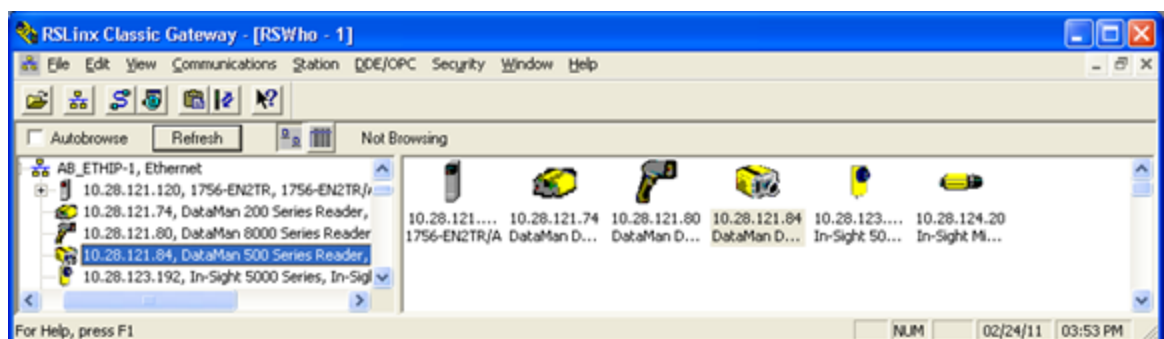
9. [Setup Tool] ビュー → [システム情報] とクリックして、指定したファームウェアがユニットにロードされたかどうかを確認します。



DataMan Setup Tool で [設定] → [通信設定] と選択して [産業用プロトコル] ペインに移動し、[EtherNet/IP™] タブで [有効] チェックボックスをクリックします。

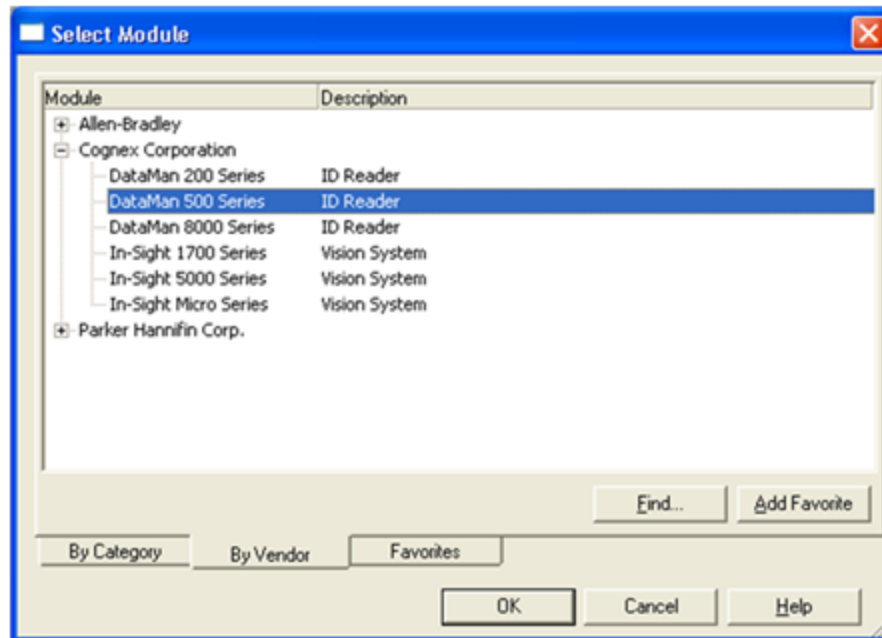


10. 変更を有効にするには、設定を保存して電源を切り、再度入れなおしてください。[システム] に移動して [設定の保存] をクリックします。
11. リーダを再起動します。
12. これで DataMan が RSWHO に表示されるようになりました。



DataMan が表示されても、アイコンがクエスチョンマークである場合は、ESD のインストールを繰り返します。

13. サンプルジョブの 1 つを開き、[Add on Profile (アドオンプロファイル)] を使用して DataMan をプログラムに組み込みます。
14. あるいは、DataMan をモジュールとしてネットワークに追加することもできます。



オブジェクトモデル

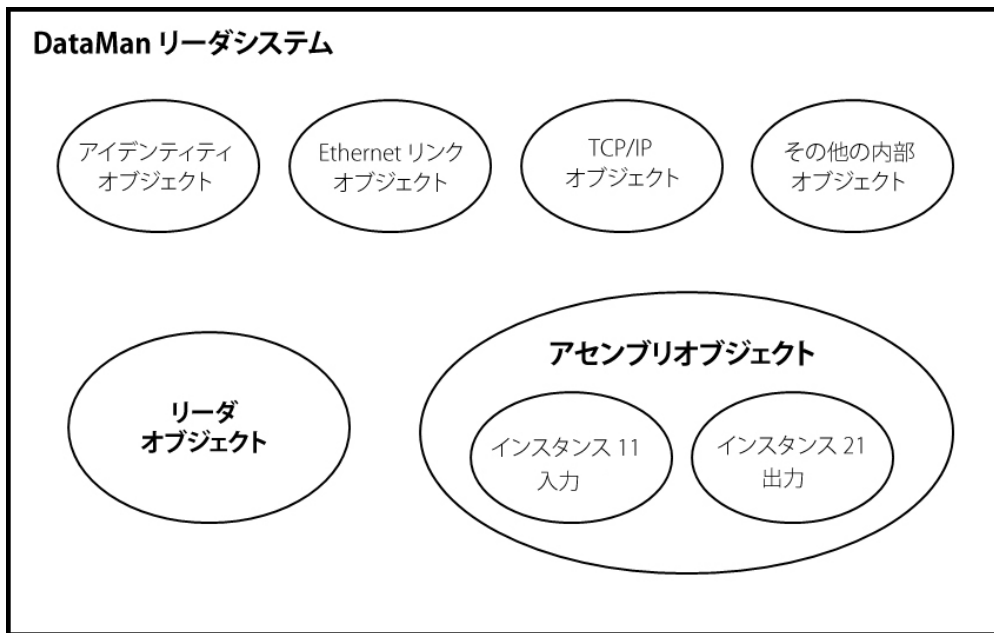
ID リーダオブジェクトは、ベンダー固有のオブジェクトクラスです。つまり、CIP 共通 (パブリック) アーキテクチャの一部ではなく、拡張機能です。これは、Cognex が DataMan デバイスの EtherNet/IP アーキテクチャに追加したカスタムオブジェクトです。このオブジェクトモデルのすべてのデータおよび機能を、DataMan リーダで使用することができます。これにはトリガ、ステータス、イベント、エラー、および結果の各データが含まれます。

ID リーダオブジェクトは、そのベンダー固有のクラスコードで識別します。

DataMan ID リーダオブジェクトのクラスコード: 0x79

オブジェクトは属性 (データ) およびサービス (機能) で構成されます。クラスレベル (クラスのすべてのインスタンスに共通) またはインスタンスレベル (個々のインスタンスに固有) でこれらを定義することができます。これらはすべてのオブジェクトに適用される CIP の仕様によって定義される、共通の属性およびサービスです (多くの場合オプションです)。ベンダー固有のクラス向けに、独自の属性およびサービスもベンダーが定義している場合があります。

ID リーダオブジェクトの属性およびサービスは、明示的なメッセージを介して個別にアクセスすることができます。さらに、ID リーダオブジェクトのいくつかの属性は、DataMan アセンブリオブジェクトに公開されており、暗黙的なメッセージを介して 1 つのグループとしてアクセスすることができます。



属性

DataMan ID リーダオブジェクト (クラスコード: 0x79) には次の属性が備わっています。

属性 ID	アクセスルール	名前	データタイプ	説明
0x9	Set	AcqTriggerEnable	BOOL	0 = EtherNet/IPトリガが無効 1 = EtherNet/IPトリガが有効
0xA	Set	AcqTrigger	BOOL	この属性が 0 から 1 に変わったときに画像を取り込みます。
0xB	Get	AcqStatusRegister	BYTE	Bit0: トリガ待ち Bit1: トリガ応答 Bit2: 画像取り込み中 Bit3: 画像取り込み失敗 Bit4-7: 予約済み
0xC	Set	UserData	BYTE の配列	画像取り込みまたはデコードへの入力として使用できるユーザ定義データ
0xD	Set	BufferResultsEnable	BOOL	true の場合、デコード結果のバッファ処理を有効にします。
0xE	Get	DecodeStatusRegister	BYTE	Bit0: デコード中 Bit1: デコード完了 (反転) Bit2: 結果バッファオーバーラン Bit3: 結果確定 Bit4: 予約済み Bit5: 予約済み Bit6: 予約済み Bit7: 一般的なエラーインジケータ
0xF	Set	ResultsAck	BOOL	クライアントがデコード結果を受信したことを示します

属性 ID	アクセスルール	名前	データタイプ	説明
0x10	Get	DecodeResults	構造体	最後のデコード結果
		ResultsID	UINT	デコード結果識別子。 デコードされた画像の TriggerID に対応します。
		ResultCode	UINT	デコード結果のサマリコード値。 Bit0: 1 = 読み取り, 0 = 読み取り失敗 Bit1: 1 = 妥当性検査合格, 0 = 妥当性検査失敗 Bit2: 1 = 検証合格, 0 = 検証失敗 Bit3: 1 = 画像取り込みトリガオーバーラン Bit4: 1 = 画像取り込みバッファオーバーラン Bit5 ~ 15: 予約済み (将来使用)
		ResultExtended	UINT	拡張結果情報
		ResultLength	UINT	結果データの現在のバイト数
		ResultData	BYTE の配列	最後のデコードの結果データ
0x12	Set	SoftEvents	BYTE	SoftEvents は仮想入力として動作します (0 から 1 への遷移でアクションを実行します) Bit0: コード登録 Bit1: 一致文字列の登録 Bit2: 焦点位置の登録 Bit3: 明るさの登録 Bit4: 登録解除 Bit5: 予約済み (将来使用) Bit6: DMCC コマンドの実行 Bit7: 一致文字列の設定
0x15	Get	TriggerID	UNIT	トリガ識別子。次に発行されるトリガの ID です。
0x16	Set	UserDataOption	UINT	オプションのユーザデータ情報
0x17	Set	UserDataLength	UINT	ユーザデータの現在のバイト数
0x18	Get	SoftEventAck	BYTE	SoftEvents の応答。 Bit0: コード登録の応答 Bit1: 一致文字列登録の応答 Bit2: 焦点位置登録の応答 Bit3: 明るさ登録の応答 Bit4: 登録解除の応答 Bit5: 予約済み (将来使用) Bit6: DMCC コマンド実行の応答 Bit7: 一致文字列設定の応答

SoftEvents

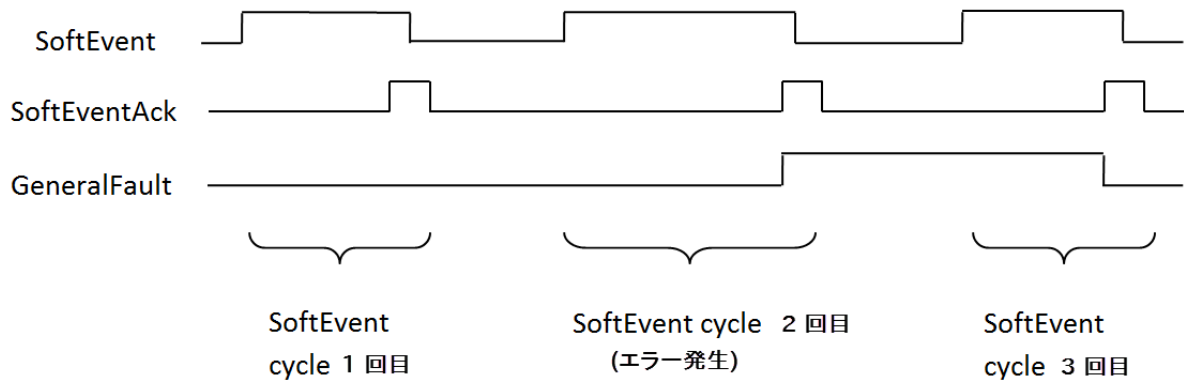
SoftEvents は「仮想」入力として機能します。SoftEvent の値が 0 から 1 に変化すると、そのイベントに関連付けられているアクションが実行されます。アクションが完了すると、対応する SoftEventAck ビットが 1 から 0 に変化して完了を示します。

SoftEvent と SoftEventAck は論理ハンドシェイクを形成します。SoftEventAck が 1 に変化した後、元の SoftEvent を 0 に戻す必要があります。その後、SoftEventAck は自動的に 0 に戻ります。

“ExecuteDMCC” および “SetMatchString” ソフトイベントアクションには、ユーザ提供データが必要です。ソフトイベントを呼び出す前に、このデータを入力アセンブリの UserData および UserDataLength 領域に書き込む必要があります。これらのソフトイベントはいずれも UserData に依存しているため、一度に呼び出すことができるのは 1 つのみです。

一般的なエラーインジケータ

通信関連のエラーが発生すると、“GeneralFault” ビットが 0 から 1 に変化します。現時点で唯一サポートされているエラー条件は、ソフトイベント操作です。ソフトイベント操作が失敗すると、エラービットがセットされます。次にソフトイベント操作が成功するまでは、エラービットはセットされたままです。あるいは、TriggerEnable が 0 にセットされてから 1 に戻るまでセットされたままです。



サービス

ID リーダオブジェクトは次の共通 CIP サービスをサポートしています。

サービスコード	サービス名	説明
0x05	Reset	ID リーダオブジェクトをリセットします。
0x0E	Get_Attribute_Single	指定した属性のコンテンツを返します。
0x10	Set_Attribute_Single	指定した属性を変更します。

ID リーダオブジェクトは、次のベンダー固有のサービスをサポートしています。

サービスコード	サービス名	説明
0x32	Acquire	単一画像取り込みをトリガします。
0x34	SendDMCC	DMCC コマンドをデバイスに送信します。
0x35	GetDecodeResults	DecodeResults 属性のコンテンツを取得します。

Acquire サービス

Acquire サービスによって画像取り込みがトリガされます (画像取り込みシステムで画像取り込みの準備ができている場合)。画像取り込みをトリガできない場合は、次に画像取り込みが成功するまで、AcqStatusRegister の画像取り込み失敗ビットがセットされます。

SendDMCC サービス

SendDMCC サービスは、DMCC コマンド文字列をデバイスに送信します。要求データは、リーダに送信される DMCC コマンド文字列で構成されます。応答データは、DMCC コマンドの結果文字列を含んでいます。さらに、このサービスは呼び出しの結果ステータスを数値で返します。これらの結果コードの大部分は、サービス実行の基本的な成功/失敗に関連しています。ただし、このサービスは実際の DMCC ステータスコードのマッピングも行います。これによって、実際の DMCC の応答文字列を解析しなくても、PLC でサービス要求を解釈することができます。

産業用インターネットプロトコル (EtherNet/IP、Profinet など) を経由して転送される DMCC コマンドは、自動的に wifi リーダにルーティングされます。

wifi リーダの電源がオフになっている、ハイバネーション状態である、または範囲外である場合には、このコマンドを実行することはできません。

サービスリターンコード	説明	DMCC リターンコード
0	成功 - エラーなし	0
1	コマンドが正しくない	-

サービスリターンコード	説明	DMCC リターンコード
4	応答なし - システムがビジー	-
100	不明なエラー	100
101	コマンドが無効	101
102	パラメータが無効	102
103	チェックサムに誤りがある	103
104	リーダの状態が原因でパラメータが拒否/変更された	104

注：この文字列は CIP STRING2 フォーマットでなければなりません (16 ビット整数は、その後ろに続く実際の文字列の長さを示します。NULL 終端 は不要です)。

GetDecodeResults サービス

GetDecodeResults サービスは、ID リーダオブジェクトの DecodeResults 属性からデータを読み取ります。このサービスは、「サイズ」(読み取るバイト数) および「オフセット」(読み取りを開始する DecodeResults 属性へのオフセット) を示すパラメータを取得します。これによって、明示的なメッセージで許容されるデータ量に対してさまざまな制限がある PLC で使用するための柔軟性が、このサービスに与えられます。さらに、暗黙的なメッセージ (アセンブリオブジェクト) では完全には転送できない非常に大きなコードにアクセスできます。

GetDecodeResults 要求データフォーマット

名前	型	説明
サイズ	UINT	読み取る DecodeResults 属性のバイト数。
オフセット	UINT	DecodeResults 属性へのオフセット。読み取りを開始する DecodeResults 属性の最初のバイトを指定します (0 ベースのオフセット)。

画像取り込みシーケンス

DataMan の画像取り込みをトリガする方法はいくつかあります。アセンブリオブジェクトを介して暗黙的に行うことができます。または、ID リーダオブジェクトを介して明示的に行います。明示的なメッセージの使用時には、Acquire サービスにアクセスしてシングルステップで行うことができ、また、ID リーダオブジェクトの属性 (AcqTrigger および AcqStatusRegister) を直接操作して、最終的に DMCC コマンドを介して行うこともできます。ここでは ID リーダの属性について説明しますが、これらの同じ値にアセンブリオブジェクトを介してアクセスすることができます。

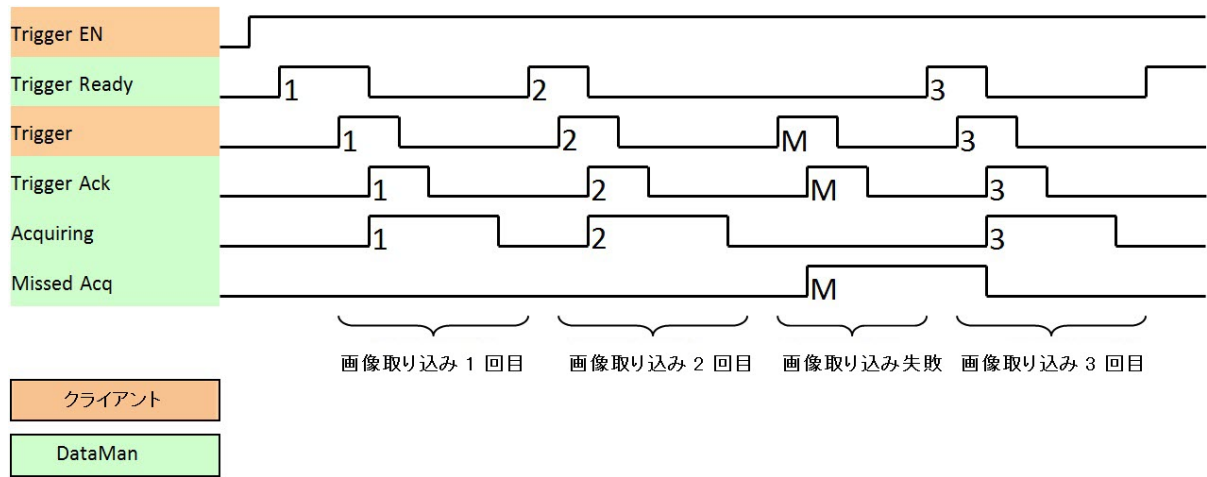
起動時には、AcqTriggerEnable 属性は False です。これを True にセットして、トリガを有効にする必要があります。デバイスでトリガを受け入れる準備ができると、AcqStatusRegister の「トリガ待ち」ビットが True にセットされます。

AcqStatusRegister の「トリガ待ち」ビットが True である間に、ID リーダオブジェクトで AcqTrigger 属性が 0 から 1 に変わったことが確認されるたびに、画像取り込みが開始されます。これをアセンブリオブジェクトを介して設定するとき、同じ状態値が AcqStatusRegister の「トリガ応答」ビットで認識されるまで、この属性は新しい状態で保持される必要があります (これは、ID リーダオブジェクトが変更を必ず認識するために必要なハンドシェークです)。

画像取り込み中に、AcqStatusRegister の「トリガ待ち」ビットはクリアされ、「画像取り込み中」ビットが True にセットされます。画像取り込みが完了すると、画像取り込み中ビットはクリアされます。デバイスで新しい画像取り込みを開始する準備ができると、トリガ待ちビットが再度 True にセットされます。

結果のバッファ処理が有効になっている場合、デバイスは画像取り込みとデコード処理をオーバーラップすることができます。デコード処理が続行されていても、画像取り込みが完了すると、TriggerReady は high にセットされます。これを利用して、全体のトリガレートは高速にすることができます。結果のバッファ処理が有効になっていない場合、画像取り込み操作とデコード操作の両方が完了するまで、TriggerReady ビットは low のままです。

特殊なケースとして、読み取り操作が完了する前にトリガ信号がクリアされ、画像取り込みがキャンセルされることがあります。この場合、視野にコードがなくても自動読み取りモードおよびマニュアルモードでの読み取りがキャンセルできてしまいます。読み取りが誤ってキャンセルされないよう、TriggerAck と ResultsAvailable の両方が True になる (または DecodeComplete の状態が反転する) まで、PLC でトリガ信号が True のまま保持されるようにするとよいでしょう。



トリガメカニズムを強制的にリセットするには、AcqStatusRegister が 0 になるまで、AcqTriggerEnable 属性を False に設定します。その後、AcqTriggerEnable を True に設定して画像取り込みを再び有効にできます。

デコードと結果シーケンス

取り込んだ画像はデコードされます。デコード中は DecodeStatusRegister の Decoding ビットがセットされます。デコードが完了すると、Decoding ビットがクリアされ、DecodeComplete ビットが反転します。

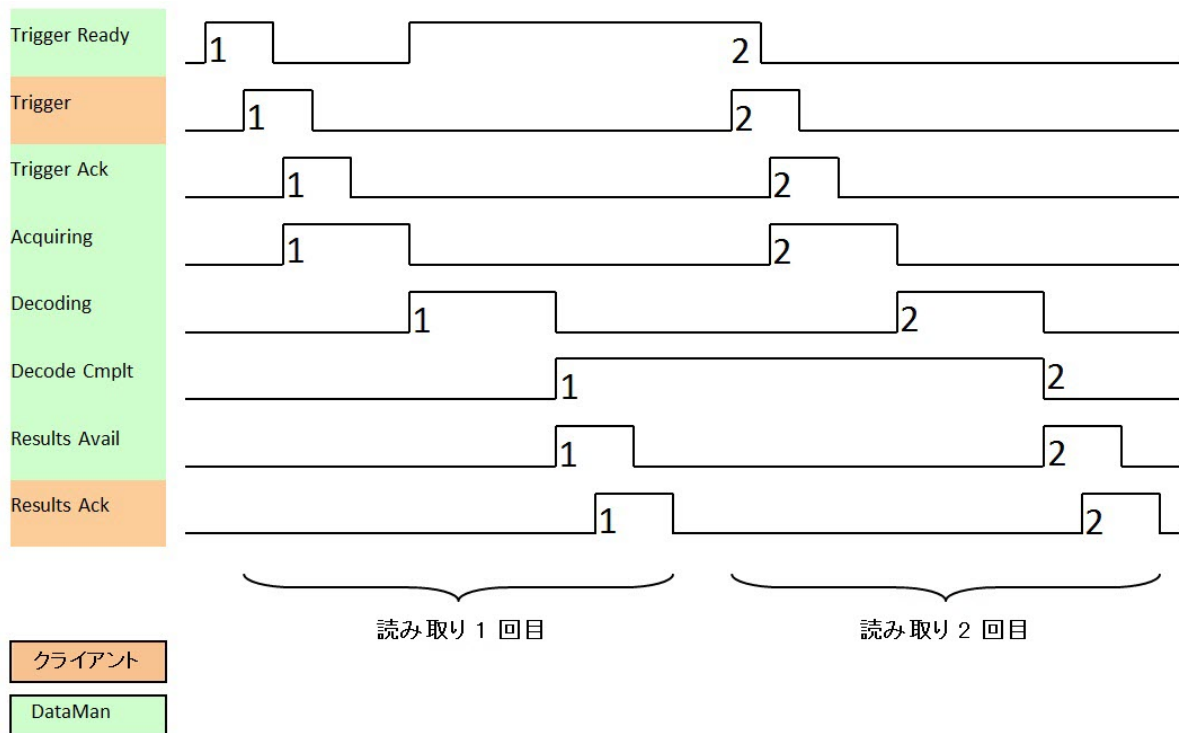
BufferResultsEnable 属性によって、ID リードオブジェクトがデコード結果をどのように処理するかが決まります。

BufferResultsEnable 属性が False にセットされると、即座にデコード結果が DecodeResults 属性に配置され、結果確定が True にセットされます。

BufferResultsEnable 属性が True にセットされると、新しい結果がキューに入れられます。それより前のデコード結果は、クライアントが ResultsAck 属性を True にセットし応答するまで、DecodeResults 属性内に残ります。ResultsAvailable ビットがクリアされた後、クライアントが ResultsAck 属性を False に戻すと、次にキューに入れられた結果を DecodeResults 属性に配置することができます。これは、DataMan リードのクライアント (PLC) によって結果が受信されたことを確認するために必要なハンドシェークです。

DecodeStatusRegister の動作

ビット	ビット名	バッファが無効な場合の結果	バッファが有効な場合の結果
1	Decoding	画像をデコードしているときにセットします。	画像をデコードしているときにセットします。
2	Decode Complete	画像デコードの完了時に反転します。	画像デコードの完了時に反転します。
3	Results Buffer Overflow	ゼロにセットしたままにします。	クライアントが直前の結果の応答に失敗し、デコード結果をキューに入れられなかった場合にセットします。デコード結果が問題なくキューに入ると、クリアされます。
4	Results Available	新しい結果が DecodeResults 属性に配置されたときにセットします。ResultsAck を true にセットし、結果応答が行われるまで、セットされたままになります。	新しい結果が DecodeResults 属性に配置されたときにセットします。ResultsAck を true にセットし、結果応答が行われるまで、セットされたままになります。



結果のバッファ処理

デコード結果のキューを有効にするオプションがあります。有効にした場合、PLC が読み取れるようになるまで有限数のデコード結果データをキューに入れることができます。これは、クライアント (PLC) のパフォーマンスが一時的に低下した場合にデータフローを円滑化するのに役立ちます。

また、結果のバッファ処理が有効になっている場合、デバイスは画像取り込みとデコード処理をオーバーラップすることができます。アプリケーションによっては、これを利用して全体のトリガレートを高速にすることができます。詳細については、前の「画像取り込みシーケンス」の説明を参照してください。

一般的に、結果の送信よりも読み取りの方が高速である場合、バッファ処理が有効であるかないかの主な違いは、どの結果が破棄されるかを定めることです。バッファ処理が有効でない場合は、最新の結果が保持され、それより前の結果 (PLC による読み取りが遅かった) が失われます。基本的に、最新の結果によって前の結果が上書きされます。バッファ処理が有効になっている (かつキューがいっぱいになっている) 場合、結果キューに空きが生じるまで最新の結果が破棄されます。

注： キューがオーバーフローし、バッファ処理が無効になっている場合、TriggerID 値と ResultID 値の差が 1 より大きくなります。この差は、発生したもののキューがいっぱいであるためにキューに入れることができなかった読み取りの数を表します (失われた読み取りの数 = $\text{TriggerID} - \text{ResultID} - 1$)。次の読み取りの後、ResultID 値は通常の動作値である $\text{TriggerID} - 1$ に戻ります。

アセンブリオブジェクト

アセンブリとは、CIP オブジェクトから選択した属性 (データ項目) とデバイス内の属性を結合したものです。デバイスのベンダーは、その必要性に従ってアセンブリを定義します。アプリケーションの要件に従って、データを結合して便利なグループにします。

入力および出力アセンブリの指定は、混乱する可能性があります。DataMan は I/O アダプタクラスのデバイスです。アダプタに関するルールは、入力アセンブリは別のデバイスに関するデータを生成 (送信) し (つまり、DataMan から PLC)、出力アセンブリは別のデバイスからデータを消費 (受信) します (つまり、PLC から DataMan)。基本的に、DataMan は PLC などの別のデバイス用の I/O モジュールとして動作します。

アセンブリオブジェクトは暗黙的なメッセージを使用します。理論上は、暗黙的なメッセージ送信パケットの生データとして送信される、単なるデータのブロックです。これらの暗黙的なメッセージ送信パケットは、事前定義で選択された速度 (100 ミリ秒、200 ミリ秒など) で繰り返し生成 (送信) されます。

DataMan リードには、単一の入力アセンブリと単一の出力アセンブリがあります。これらのアセンブリは、DataMan ID リードオブジェクトの選択した属性 (データ) を結合してグループにし、ネットワーク帯域を最小限にし、それでもなお効率的な制御と処理を行うことができます。これらのアセンブリ内のデータは、ID リードオブジェクトから個別にアクセスすることもできます。ただし、アセンブリオブジェクトを使用する方がはるかに効率的です。この理由から、アセンブリオブジェクトは DataMan リードと PLC 間のランタイム通信の主要な手段となっています。

入力アセンブリ

入力アセンブリは、ステータス情報、処理状態、およびデコード結果を示します。

インスタンス	バイト	ビット 7	ビット 6	ビット 5	ビット 4	ビット 3	ビット 2	ビット 1	ビット 0
11	0	Reserved				MissedAcq	Acquiring	TriggerAck	Trigger Ready
	1	General Fault	Reserved			Results Available	Results Buffer Overrun	Decode Complete Toggle	Decoding
	2	SoftEvent Ack 7	SoftEvent Ack 6	SoftEvent Ack 5	SoftEvent Ack 4	SoftEvent Ack 3	SoftEvent Ack 2	SoftEvent Ack 1	SoftEvent Ack 0
	3-5	Reserved							
	6	TriggerID (16-bit integer)							
	7								
	8								
	9	ResultID (16-bit integer)							
	10								
	11								
	12	ResultExtended (16-bit integer)							
	13								
	14								
	15	ResultDataLength (16-bit integer)							
	16								
	...								
	499	ResultData 483							

出力アセンブリ

出力アセンブリは、制御信号、ソフトウェアイベント信号、およびトリガとデコードに必要なユーザデータを含んでいます。

インスタンス	バイト	ビット 7	ビット 6	ビット 5	ビット 4	ビット 3	ビット 2	ビット 1	ビット 0
21	0	Reserved				Results Ack	Buffer Results Enable	Trigger	Trigger Enable
	1	SoftEvent 7	SoftEvent 6	SoftEvent 5	SoftEvent 4	SoftEvent 3	SoftEvent 2	SoftEvent 1	SoftEvent 0
	2	Reserved							
	3								
	4	UserDataOption (1 6-bit integer)							
	5								
	6	UserDataLength (1 6-bit integer)							
	7								
	8	UserData 0							
	...								
	499	UserData 491							

PCCC オブジェクト

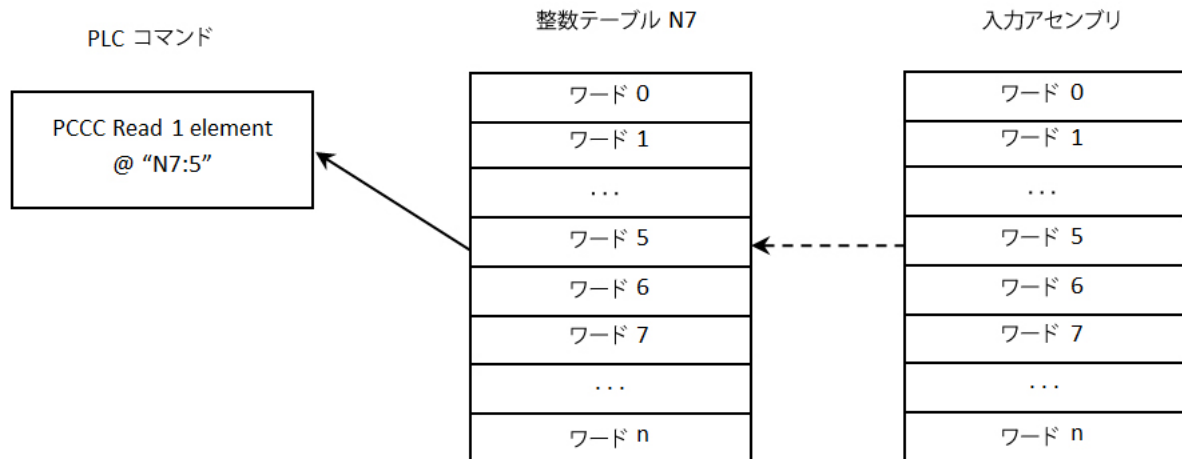
DataMan は、Rockwell PCCC オブジェクトを限定的にサポートしています。これによって、従来の PLC (PLC-5、SLC など) は、そのネイティブの PCCC コマンドセットおよび明示的メッセージを使用して DataMan と通信することができます。PCCC オブジェクトによって、DataMan を Rockwell PLC-5 ロジックコントローラのように動作させることができます。

PCCC コマンドを構成し、従来のロジックコントローラ内にある「データテーブル」を使用して作業します。それぞれのデータテーブルは、指定されたデータタイプ (BYTE、INT、FLOAT など) の配列です。このコマンドは、指定したデータテーブルの 1 つ以上のデータ項目の読み取り/書き込みに適応します。データテーブルおよびテーブル内の項目のインデックスを指定することによって、項目のアドレス指定を行います (インデックスは 0 から開始されます)。たとえば PLC データテーブル内で 6 番目の整数を読み取るには、N7:5 を読み取る PCCC コマンドを送信します。「N」は整数テーブルを指定し、「7」は PLC 内でのテーブル番号であり (各テーブルには固有の数字の識別子があり、ユーザ PLC プログラムが作成されたときに割り当てられます)、「5」はテーブルへのインデックスです (インデックスは 0 から開始することに注意してください)。

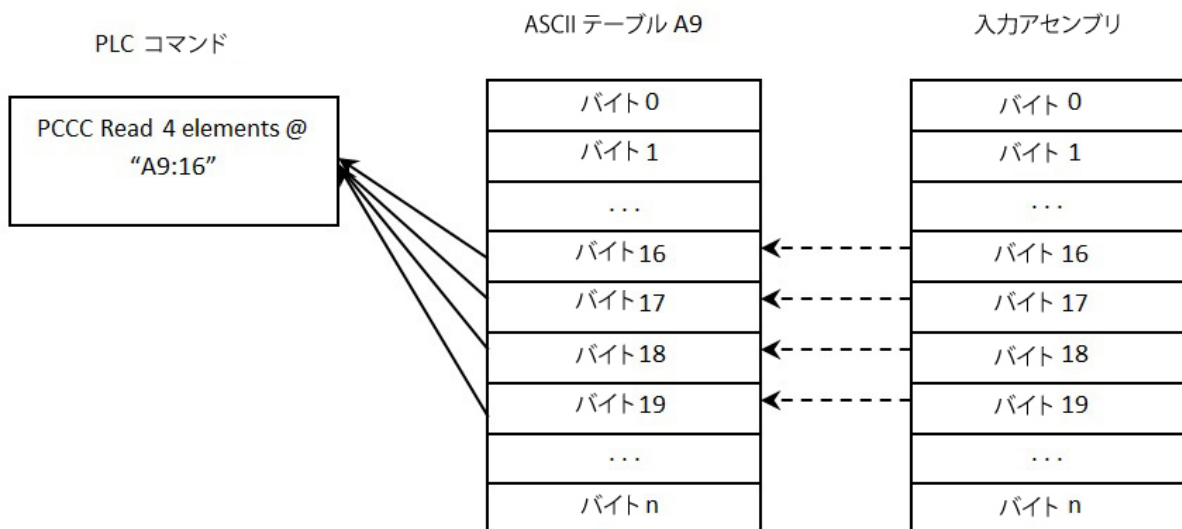
DataMan の PCCC オブジェクトは、読み取りおよび書き込み要求を ID リーダアセンブリ (特殊なケースでは DMCC サービス) にマッピングします。読み取りコマンドは、入力アセンブリからデータを返します (インスタンス 11)。書き込みコマンドは、出力アセンブリにデータを送信します (インスタンス 21)。基本的に、PCCC オブジェクトは PLC-5 データテーブルの外観を示していますが、実際にはアセンブリデータにアクセスしています。現在、この実装は整数データテーブル (N7) および ASCII データテーブル (A9) のみをサポートしています。DMCC に関して、文字列データテーブルの特殊なケースが 1 つあります (ST10:0)。

テーブル	データタイプ	テーブルのサイズ
N7	整数 (16 ビット)	250 要素
A9	ASCII (8 ビット)	500 要素
ST10	文字列	1 要素

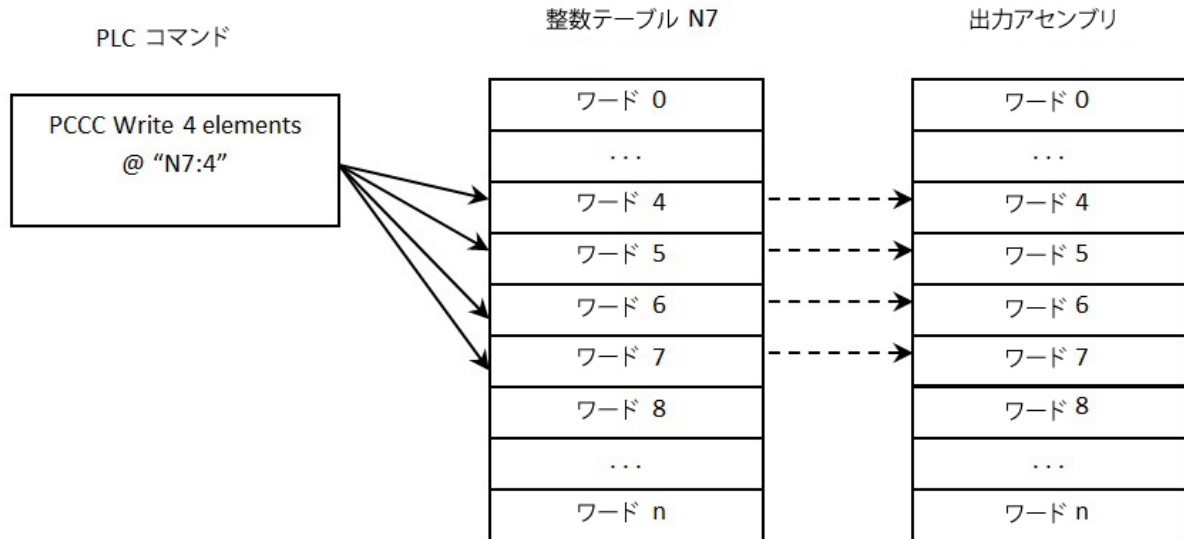
ResultCode の値は、入力アセンブリのワードオフセット 5 (0 からカウント) に位置しています。この値にアクセスするには、次の PLC コマンドを発行します。



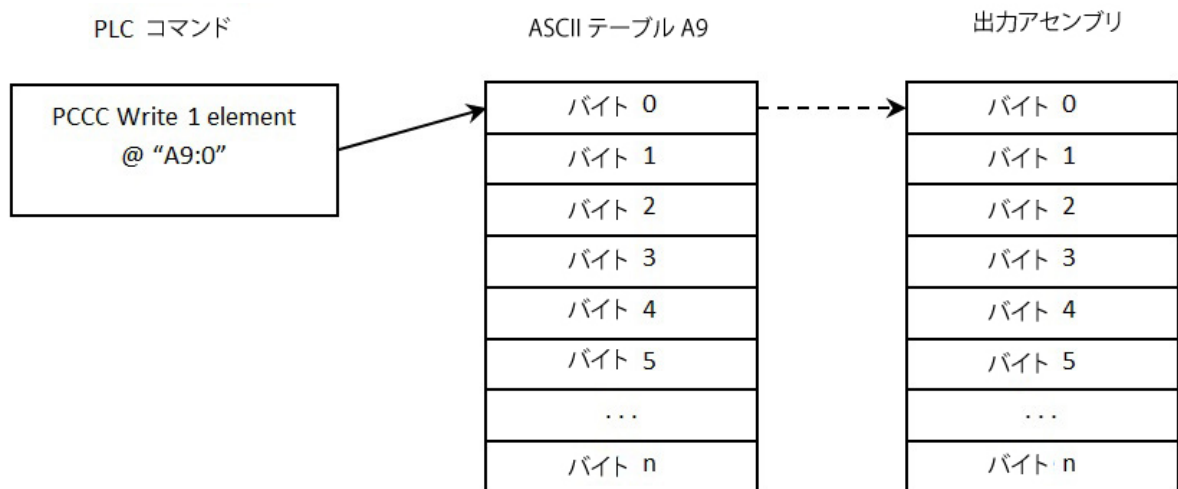
Decode ResultCode は、入力アセンブリのバイトオフセット 16 (0 からカウント) から始まります。結果データの最初の 4 バイトを読み取るには、次の PLC コマンドを発行します。



UserData は、出力アセンブリのワードオフセット 4 (0 からカウント) から始まります。UserData の 4 つのワードを書き込むには、次の PLC コマンドを発行します。

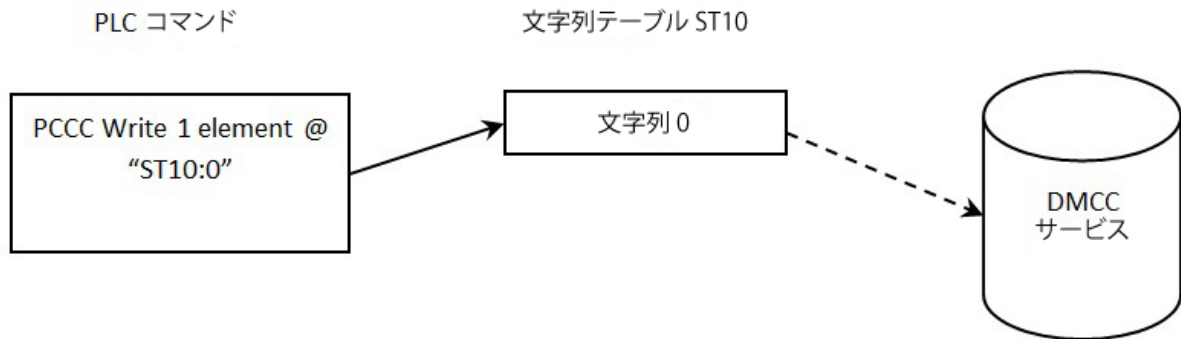


画像取り込みをトリガするビットは、出力アセンブリのバイトオフセット 0 にあります。このバイトを書き込むには、次の PLC コマンドを発行します。



PCCC オブジェクトは、DMCC サービスに対する文字列テーブル要素 (ST10:0) の特殊なケースのマッピングをサポートしています。ST10:0 に書き込まれた文字列は、DMCC サービスに渡されて処理されます。これによって、DMCC コマンドの呼び出しに使用する文字列コマンドを、PCCC で書き込むことができます。

① 注： 文字列テーブルは、1 要素のサイズのみです。その他の要素に書き込むと、エラーが返されます。



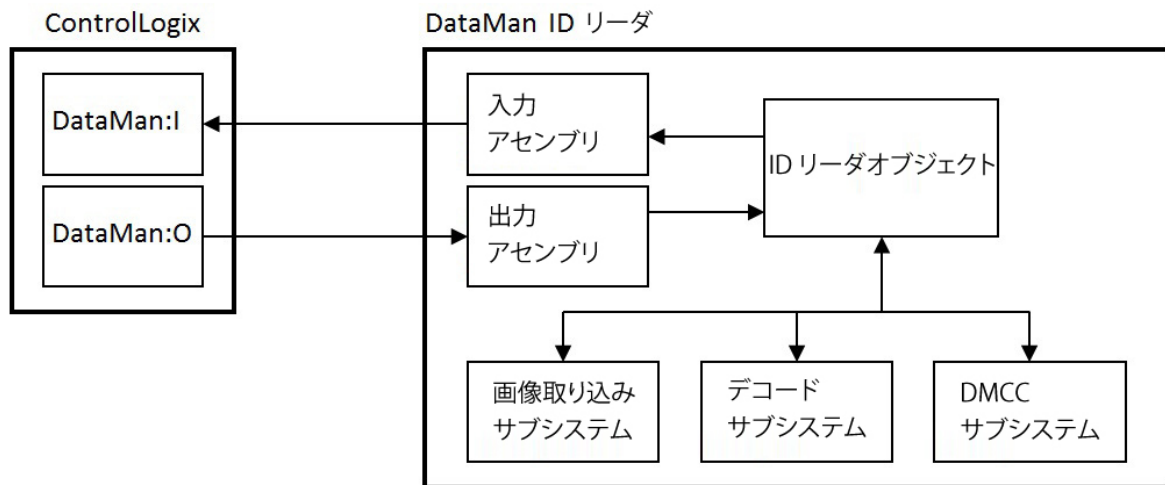
Rockwell ControlLogix の例

暗黙的なメッセージは応答時間が重要なアプリケーション固有の I/O データを送信します。ポイントツーポイントまたはマルチキャストが可能です。明示的メッセージには受信デバイスからの応答が必要です。そのため、明示的メッセージは発生頻度が高くない操作に適しています。DMCC コマンドの送信指示は明示的メッセージの一例です。

暗黙的メッセージ

EtherNet/IP による暗黙的メッセージでは、DataMan リーダの入力および出力を ControlLogix PLC のタグに割り振ることができます。これらの接続の確立後、ユーザ定義の間隔 (10 ミリ秒、50 ミリ秒、100 ミリ秒など) でデータが周期的に転送されます。

次の図に、EtherNet/IP を使用した Ethernet ベースの I/O を示します。



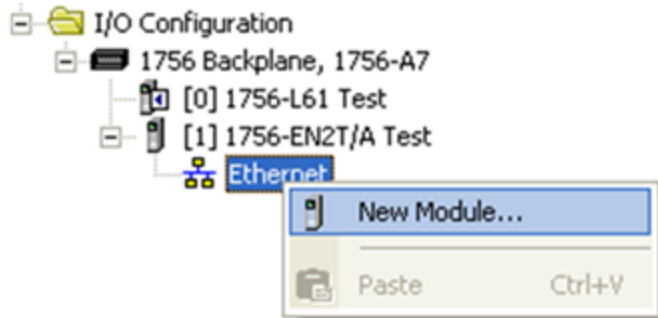
「入力アセンブリ」と「出力アセンブリ」は ID リーダオブジェクトのさまざまな属性 (データ) を割り付けます。入力アセンブリは PLC (PLC 入力) に送信される DataMan リーダのデータ値のコレクションであり、出力アセンブリは PLC (PLC 出力) から DataMan リーダが受信するデータ値のコレクションです。

暗黙的メッセージ接続の確立

DataMan と ControlLogix コントローラの間で EtherNet/IP の暗黙的メッセージ接続を設定するには、まず DataMan リーダを ControlLogix I/O 設定ツリーに追加する必要があります。そのためにはアドオンプロファイルを使用するのが最も効率的です。この例では、アドオンプロファイルがすでにインストールされていることを前提としています。アドオンプロファイルがない場合は、[汎用 EtherNet/IP プロファイルの使用](#)」の節を参照してください。

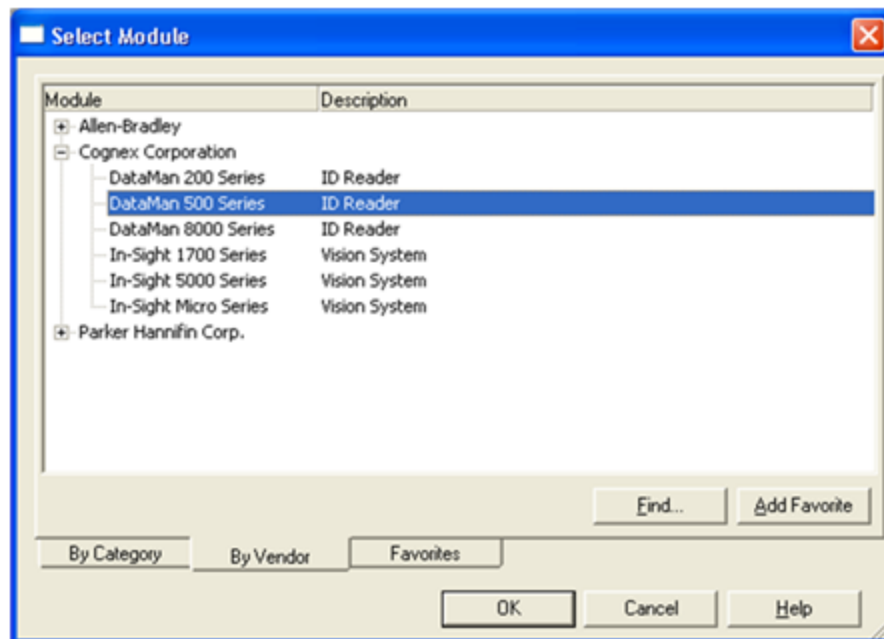
ControlLogix PLC との暗黙的メッセージ接続を確立するには

1. RSLogix5000 を開き、プロジェクトをロードします (または、[ファイル] → [新規] を選択して新規プロジェクトを作成します)。I/O 設定ノードからプロジェクトの Ethernet モジュールの下にある Ethernet ノードを選択し、アイコンを右クリックしてメニューから **[New Module (新規モジュール)]** を選択します。



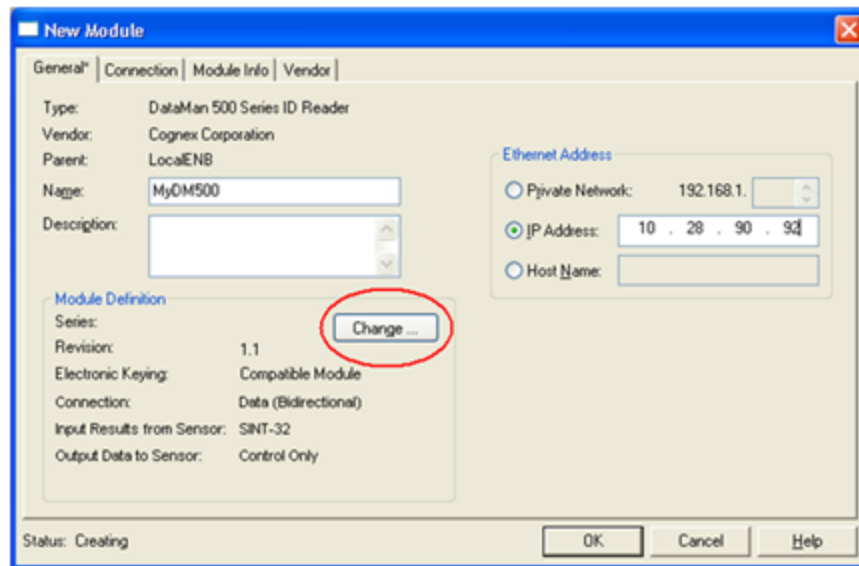
2. [Select Module (モジュール選択)] ダイアログで、リストから使用する DataMan ID リーダのモデルを選択します。

注： このオプションは、DataMan アドオンプロファイルがインストールされている場合のみ使用可能です。



注： この後の手順は、どの DataMan モデルを選択しても同じです。

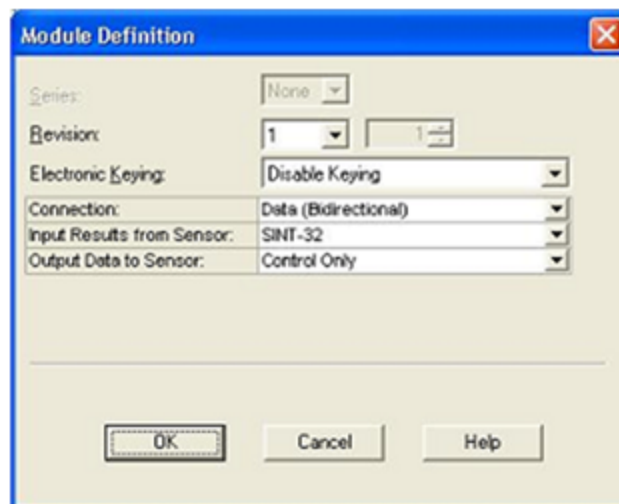
3. 選択すると、DataMan ID リーダシステムの設定ダイアログが表示されます。モジュールに名前を付け、DataMan の IP アドレスを入力します。デフォルトは、制御、ステータス、キー操作を無効にした 32 バイトの結果データで構成された双方向 (送信／受信) 接続です。このデフォルト設定を変更するには、[変更] ボタンを選択します。変更が不要な場合は、次の手順に進みます。



4. 接続設定を変更します。

[変更] ボタンを選択すると、[Module Definition (モジュール定義)] ダイアログが表示されます。このダイアログを使用して、接続設定を変更します。次の変更が可能です。

- DataMan リビジョン
- 電子キー操作
- 接続の種類 (双方向/受信のみ)
- (DataMan からの) 受信データ量
- (DataMan への) 送信データ量



Electronic Keying (電子キー操作): 接続を確立する前に PLC が実行するモジュールタイプのチェックレベルを定義します。

完全一致 - すべてのパラメータが一致する必要があります。一致しない場合、接続は拒否されます。

- ベンダー
- 製品タイプ

- カタログ番号
- メジャーリビジョン
- マイナーリビジョン

Compatible Module (互換モジュール) - 次の基準を満たす必要があります。満たさない場合、挿入されたモジュールで接続が拒否されます。

- モジュールタイプが一致する
- カタログ番号が一致する
- メジャーリビジョンが一致する
- モジュールのマイナーリビジョンは、ソフトウェアに指定されているリビジョン以降である必要があります。

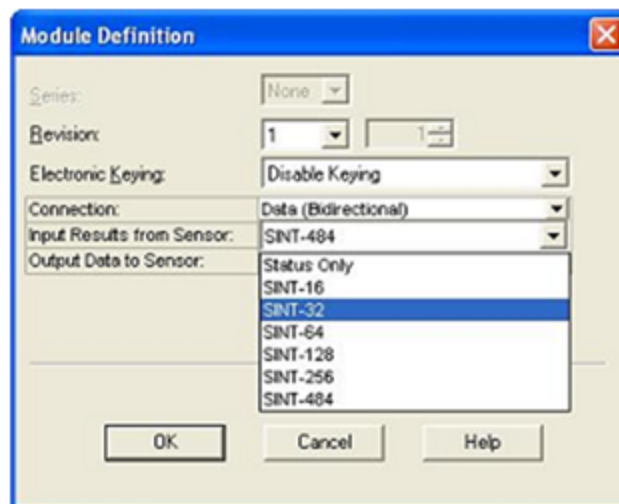
Disable Keying (キー操作無効) - コントローラでキー操作が受け付けられません。

接続: データフローのタイプを定義します。

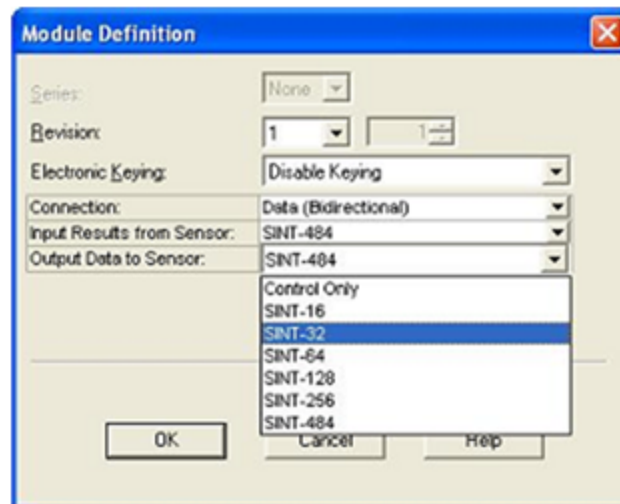
Data (Bidirectional) (データ (双方向)) - (DataMan への) データ送信と (DataMan からの) データ受信を行います。

Input (Results only) (入力 (結果のみ)) - (DataMan からの) データ受信のみ行います。これは一般的に、同じ DataMan デバイスから複数の PLC がデータを受信する必要がある場合に使用します。

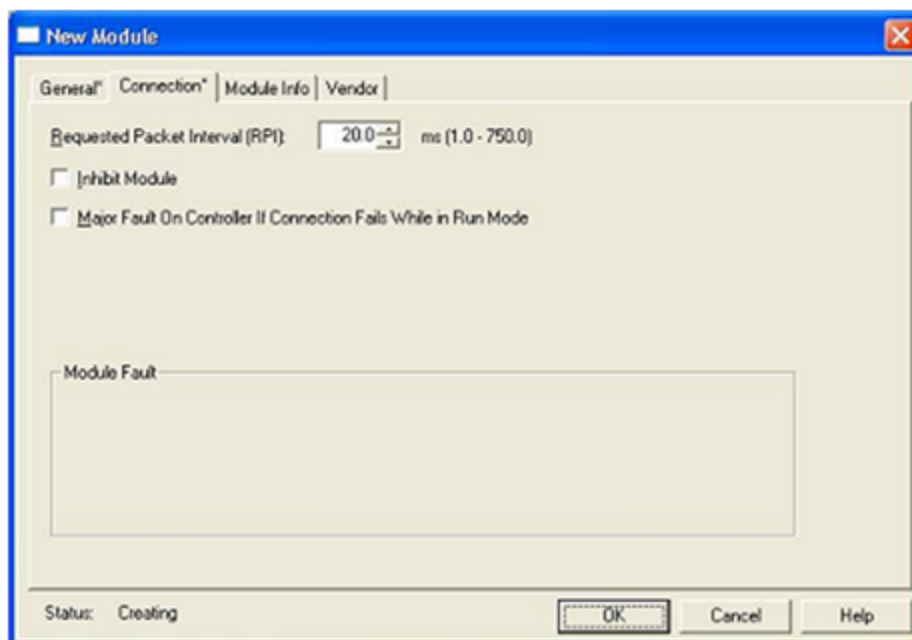
Input Results from Sensor (センサからの結果入力): 接続時に (DataMan から) 受信するデータ量を定義します。最小量はステータスデータのみです。読み取り結果データも受信するように設定することもできます。結果データの受信量は、一定の増分 (16 バイト、32 バイト、64 バイトなど) で定義します。アプリケーションが読み取る最大コードサイズを超えないようにサイズを選択してください。それよりも大きくするとネットワーク帯域幅が浪費され、パフォーマンスが損なわれます。



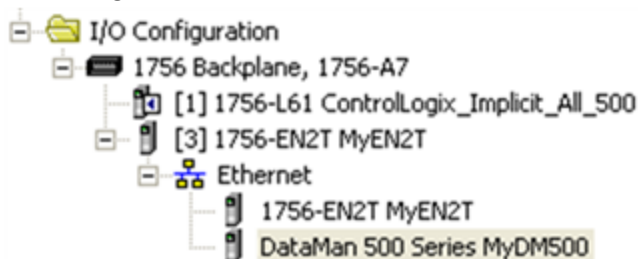
Output Data to Sensor (センサへのデータ出力): 接続時に (DataMan に) 送信するデータ量を定義します。最小量は制御データのみです。ユーザデータも送信するように設定することもできます。ユーザデータの送信量は、一定の増分 (16 バイト、32 バイト、64 バイトなど) で定義します。ユーザデータ出力を有効にするには、DataMan モジュールを右クリックし、[プロパティ] → [変更] → [Output Data to Sensor (センサへの出力データ)] を選択します。



5. 最後に、接続レートを設定します。データの送受信レートは Requested Packet Interval (RPI) として定義されます。RPI は接続を介してデータが送受信される頻度を定義します。ネットワークパフォーマンスを最適化するには、このレートを特定のアプリケーションに確実に必要なレート以上にする必要があります。一般的には、ユーザアプリケーションで予想される最大読み取り率の 1/2 以上にしてください。それよりも小さく設定すると帯域幅が浪費され、処理パフォーマンスが向上しません。
6. [New Module (新規モジュール)] ダイアログの [接続] タブを選択してレートを設定します。



7. ControlLogix にモジュールを追加すると、I/O ツリーが次のように表示されます。



8. DataMan モジュールが I/O ツリーに追加されると、RSLogix 5000 によって DataMan リーダの入力および出力データ (DataMan リーダの入力および出力アセンブリオブジェクト) に対応するタグが作成されます。これらのタグはプロジェクトツリーの [Controller Tags (コントローラタグ)] ノードにあります。

① 注：これらのタグの基本名は、前の手順で I/O 設定に追加した DataMan モジュールの名前です。

The screenshot shows the RSLogix 5000 software interface. On the left is the 'Controller Tags' tree, which includes folders for 'Controller Tags', 'Controller Fault Handler', 'Power-Up Handler', 'Tasks', 'Motion Groups', 'Ungrouped Axes', 'Add-On Instructions', 'Data Types', 'Trends', and 'I/O Configuration'. Under 'I/O Configuration', there is a folder for '1756 Backplane, 1756-A7' containing '1756-L61 ControlLogix_I' and '1756-EN2T MyEN2T'. Under '1756-EN2T MyEN2T', there is a folder for 'Ethernet' containing '1756-EN2T MyEN2T' and 'DataMan 500 Series'.

On the right is a table showing the tags for 'MyDM500.I' and 'MyDM500.O'. The table has columns for 'Name', 'Value', 'Style', and 'Data Type'.

Name	Value	Style	Data Type
MyDM500.I	{...}		CC:DataMan
MyDM500.I.Status	{...}		CC:DataMan
MyDM500.I.Status.TriggerReady	0	Decimal	BOOL
MyDM500.I.Status.TriggerAck	0	Decimal	BOOL
MyDM500.I.Status.Acquiring	0	Decimal	BOOL
MyDM500.I.Status.MissedAcq	0	Decimal	BOOL
MyDM500.I.Status.Decoding	0	Decimal	BOOL
MyDM500.I.Status.DecodeCompleted	0	Decimal	BOOL
MyDM500.I.Status.ResultsBufferOverrun	0	Decimal	BOOL
MyDM500.I.Status.ResultsAvailable	0	Decimal	BOOL
MyDM500.I.Status.GeneralFault	0	Decimal	BOOL
MyDM500.I.Status.TrainCodeAck	0	Decimal	BOOL
MyDM500.I.Status.TrainMatchStringAck	0	Decimal	BOOL
MyDM500.I.Status.TrainFocusAck	0	Decimal	BOOL
MyDM500.I.Status.TrainBrightnessAck	0	Decimal	BOOL
MyDM500.I.Status.UntrainAck	0	Decimal	BOOL
MyDM500.I.Status.ExecuteDmccAck	0	Decimal	BOOL
MyDM500.I.Status.SetMatchStringAck	0	Decimal	BOOL
MyDM500.I.Status.TriggerID	0	Decimal	INT
MyDM500.I.Status.ResultID	0	Decimal	INT
MyDM500.I.Status.ResultCode	0	Decimal	INT
MyDM500.I.Status.ResultExtended	0	Decimal	INT
MyDM500.I.Status.ResultLength	0	Decimal	INT
MyDM500.I.ResultData	{...}	ASCII	SINT[484]
MyDM500.O	{...}		CC:DataMan
MyDM500.O.Control	{...}		CC:DataMan
MyDM500.O.Control.TriggerEnable	0	Decimal	BOOL
MyDM500.O.Control.Trigger	0	Decimal	BOOL
MyDM500.O.Control.ResultsBufferEnable	0	Decimal	BOOL
MyDM500.O.Control.ResultsAck	0	Decimal	BOOL
MyDM500.O.Control.TrainCode	0	Decimal	BOOL
MyDM500.O.Control.TrainMatchString	0	Decimal	BOOL
MyDM500.O.Control.TrainFocus	0	Decimal	BOOL
MyDM500.O.Control.TrainBrightness	0	Decimal	BOOL
MyDM500.O.Control.Untrain	0	Decimal	BOOL

タグは 2 つのグループ (ステータスとコントロール) に分けられています。ステータスグループは (DataMan から) 受信したすべてのデータを表し、コントロールグループは (DataMan に) 送信するすべてのデータを表します。

これらのタグは DataMan アセンブリオブジェクトコンテンツのシンボリック表現です。これらのタグ値にアクセスするには、PLC ラダーに書き込みます。PLC ラダーはこれらのタグ値を監視または変更することで、DataMan アセンブリオブジェクトのコンテンツの監視および変更を行います。

① 注：DataMan とこれらの PLC タグ値には (設定した RPI に基づく) 遅延が生じます。すべての PLC ラダーはその遅延を考慮に入れて書き込まれる必要があります。

暗黙的メッセージ接続データへのアクセス

上記の節では、ControlLogix と DataMan ID リーダの間で暗黙的メッセージ接続を確立する方法の詳細について説明しました。この例では、DataMan アドオンプロファイルを使用することを前提としています。アドオンプロファイルでは、接続データを表す ControlLogix タグが自動的に生成されます。

生成されたタグは 2 つのグループ (ステータスとコントロール) に分けられています。ステータスグループは (DataMan から) 受信したすべてのデータを表し、コントロールグループは (DataMan に) 送信するすべてのデータを表します。

次に、ステータスタググループについて説明します。これは ControlLogix が DataMan リーダから受信したデータです。

MyDM500.I	{...}	CC:DataMan
MyDM500.I.Status	{...}	CC:DataMan
MyDM500.I.Status.TriggerReady	0 Decimal	800L
MyDM500.I.Status.TriggerAck	0 Decimal	800L
MyDM500.I.Status.Acquiring	0 Decimal	800L
MyDM500.I.Status.MissedAcq	0 Decimal	800L
MyDM500.I.Status.Decoding	0 Decimal	800L
MyDM500.I.Status.DecodeCompleted	0 Decimal	800L
MyDM500.I.Status.ResultsBufferOverrun	0 Decimal	800L
MyDM500.I.Status.ResultsAvailable	0 Decimal	800L
MyDM500.I.Status.GeneralFault	0 Decimal	800L
MyDM500.I.Status.TrainCodeAck	0 Decimal	800L
MyDM500.I.Status.TrainMatchStringAck	0 Decimal	800L
MyDM500.I.Status.TrainFocusAck	0 Decimal	800L
MyDM500.I.Status.TrainBrightnessAck	0 Decimal	800L
MyDM500.I.Status.UntrainAck	0 Decimal	800L
MyDM500.I.Status.ExecuteDmccAck	0 Decimal	800L
MyDM500.I.Status.SetMatchStringAck	0 Decimal	800L
MyDM500.I.Status.TriggerID	0 Decimal	INT
MyDM500.I.Status.ResultID	0 Decimal	INT
MyDM500.I.Status.ResultCode	0 Decimal	INT
MyDM500.I.Status.ResultExtended	0 Decimal	INT
MyDM500.I.Status.ResultLength	0 Decimal	INT
MyDM500.I.ResultData	{...} ASCII	SINT[484]

- **TriggerReady:** DataMan リーダが新規トリガを受け入れ可能であることを示します。このタグは、コントロールタグ “TriggerEnable” がセットされ、センサが現在画像取り込み中でない場合に True です。
- **TriggerAck:** DataMan リーダがトリガされた (コントロールタグ “Trigger” が True にセットされた) ことを示します。このタグは Trigger タグがクリアされるまでセットされたままです。
- **Acquiring:** DataMan リーダが Trigger ビットセットまたは外部トリガによって現在画像取り込み中であることを示します。
- **MissedAcq:** DataMan リーダが取り込みトリガに失敗したことを示します。次の取り込みが成功するとクリアされます。
- **Decoding:** DataMan リーダが取り込んだ画像をデコードしていることを示します。
- **DecodeCompleted:** デコードが完了するとタグ値が反転します (1→0 または 0→1)。
- **ResultsBufferOverrun:** 結果キューがいっぱいになったため、DataMan リーダが一連のデコード結果を破棄したことを示します。次の結果セットが正常にキューに入った場合にクリアされます。
- **ResultsAvailable:** 一連のデコード結果が使用可能な状態になった (ResultID、ResultCode、ResultLength、ResultsData タグに有効なデータが含まれている) ことを示します。
- **GeneralFault:** エラーが発生 (ソフトイベント “SetMatchString” または “ExecuteDMCC” エラーが発生) したことを示します。
- **TrainCodeAck:** ソフトイベント “TrainCode” が完了したことを示します。
- **TrainMatchStringAck:** ソフトイベント “TrainMatchString” が完了したことを示します。
- **TrainFocusAck:** ソフトイベント “TrainFocus” が完了したことを示します。
- **TrainBrightnessAck:** ソフトイベント “TrainBrightness” が完了したことを示します。
- **UnTrainAck:** ソフトイベント “UnTrain” が完了したことを示します。
- **ExecuteDmccAck:** ソフトイベント “ExecuteDMCC” が完了したことを示します。
- **SetMatchStringAck:** ソフトイベント “SetMatchString” が完了したことを示します。

- **TriggerID:** 発行される次のトリガの値。発行されたトリガと後で受信する対応する結果データの照合に使用されます。
- **ResultID:** 結果を生成したトリガが発行されたときの TriggerID の値。TriggerID と結果データの照合に使用されます。
- **ResultCode:** この結果セットの成功/失敗を示します。
 - ビット 0、1= 読み取り 0= 読み取り失敗
 - ビット 1、1= 妥当性検査合格 0= 妥当性検査失敗 (または妥当性検査未実施)
 - ビット 2、1= 検証合格 0= 検証失敗 (または検証未実施)
 - ビット 3、1= 画像取り込みトリガオーバーラン
 - ビット 4、1= 画像取り込みバッファオーバーフロー (結果バッファオーバーフローとは異なる)
 - ビット 5~15、予約済み (将来使用)
- **ResultExtended:** 現時点では未使用です。
- **ResultLength:** ResultData タグに含まれる結果データのバイト数。

- **ResultData:** デコード結果データ。

次に、コントロールタググループについて説明します。これは ControlLogix から DataMan リーダに送信したデータです。

Name	Value	Style	Data Type
MyDM200:0	{...}		CC:DataMan...
MyDM200:0.Control	{...}		CC:DataMan...
MyDM200:0.Control.TriggerEnable	0	Decimal	BOOL
MyDM200:0.Control.Trigger	0	Decimal	BOOL
MyDM200:0.Control.ResultsBufferEnable	0	Decimal	BOOL
MyDM200:0.Control.ResultsAck	0	Decimal	BOOL
MyDM200:0.Control.TrainCode	0	Decimal	BOOL
MyDM200:0.Control.TrainMatchString	0	Decimal	BOOL
MyDM200:0.Control.TrainFocus	0	Decimal	BOOL
MyDM200:0.Control.TrainBrightness	0	Decimal	BOOL
MyDM200:0.Control.Untrain	0	Decimal	BOOL
MyDM200:0.Control.ExecuteDMCC	0	Decimal	BOOL
MyDM200:0.Control.SetMatchString	0	Decimal	BOOL
MyDM200:0.Control.UserDataOption	0	Decimal	INT
MyDM200:0.Control.UserDataLength	0	Decimal	INT
MyDM200:0.UserData	{...}	ASCII	SINT[484]

- **TriggerEnable:** このタグをセットすると EtherNet/IP トリガが有効になります。このフィールドをクリアすると EtherNet/IP トリガが無効になります。
- **Trigger:** このタグをセットすると、次の条件が満たされた場合に取り込みがトリガされます。
 - TriggerEnable タグがセットされている。
 - 取り込み/デコードが現在進行中ではない。
 - デバイスはトリガの準備ができています。
- **ResultsBufferEnable:** セットされている場合デコード結果がキューに入れられます。現在の結果の応答があるたびに、PLC によって応答されるまで、結果がキューから取り出されます (使用可能な状態になる)。Decode ID、Decode Result、Decode ResultsData のフィールドは、ResultsAck フィールドによって応答され設定されるまで変わりません。DataMan リーダは ResultsValid ビットをクリアすることで応答に対応します。ResultsAck フィールドがクリアされると、次のデコード結果セットが格納されます。

- **ResultsAck:** ResultsAck タグは、PLC が最新の結果を読み取ったことに知らせるために使用されます。ResultsAck がセットされると、ResultsAvailable タグはクリアされます。結果のバッファ処理が有効になっている場合、ResultsAck タグが再度クリアされた時点で次の結果セットが格納されます。
- **TrainCode:** このタグを 0 から 1 に変更すると、コード登録操作が呼び出されます。
- **TrainMatchString:** このタグを 0 から 1 に変更すると、一致文字列の登録操作が呼び出されます。
- **TrainFocus:** このタグを 0 から 1 に変更すると、焦点位置の登録操作が呼び出されます。
- **TrainBrightness:** このタグを 0 から 1 に変更すると、明るさの登録操作が呼び出されます。
- **Untrain:** このタグを 0 から 1 に変更すると、登録解除操作が呼び出されます。
- **ExecuteDMCC:** このタグを 0 から 1 に変更すると、DMCC 操作が呼び出されます。このソフトイベントの起動前に、有効な DMCC コマンド文字列を UserData に書き込んでおく必要があります。
- **SetMatchString:** このタグを 0 から 1 に変更すると、一致文字列の設定操作が呼び出されます。このソフトイベントの起動前に、一致文字列データを UserData に書き込んでおく必要があります。
- **UserDataOption:** 現時点では未使用です。
- **UserDataLength:** UserData タグに含まれるユーザデータのバイト数。
- **UserData:** このデータは取り込みやデコードをサポートするために DataMan リードに送信されます。

注：

RSLogix 5000 で DataMan モジュールを設定することで、**UserData** を出力アセンブリに手動で追加する必要があります。CompactLogix または ControlLogix PLC で次の手順を実行します。

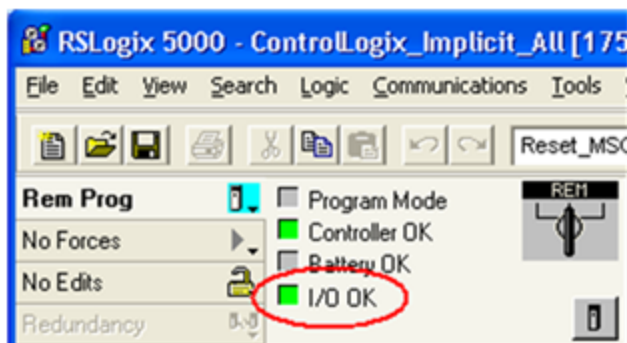
1. DataMan モジュールを右クリックして **[プロパティ]** を選択します。
- ① 2. **[Module Definition (モジュール定義)]** で **[変更]** をクリックします。
3. [Module Definition (モジュール定義)] ウィンドウが表示されます。**[Output Data to Sensor (センサーへの出力データ)]** ドロップダウンメニューで「**SINT-484**」を選択します。
4. [OK] をクリックします。RSLogix 500 でモジュール定義が更新されます。出力アセンブリコントローラのタグに、出力アセンブリの一部として UserData が表示されます。

暗黙的メッセージ接続の動作の確認

DataMan リードが I/O デバイスとして ControlLogix プロジェクトに追加されました。このプロジェクトをコントローラにダウンロードすると、I/O 接続が確立されます。接続が正常に確立された後、要求された RPI で周期データ転送が開始されます。

適切な I/O 接続を確認するには、次の手順を実行します。

1. 作成済みのプロジェクトを ControlLogix コントローラにダウンロードします。
2. ダウンロードを完了すると、プロジェクトの I/O インジケータが「I/O OK」になります。これは I/O 接続が正常に完了したことを示しています。



I/O データの適切な双方向転送を確認するには、RSLogix でコントローラタグに移動し、TriggerEnable ビットの状態を 0 から 1 に変更します。

- MyDM200:0	{...}		CC.DataMan...
- MyDM200:0.Control	{...}		CC.DataMan...
MyDM200:0.Control.TriggerEnable	1	Decimal	BOOL
MyDM200:0.Control.Trigger	0	Decimal	BOOL
MyDM200:0.Control.ResultsBufferEnable	0	Decimal	BOOL
MyDM200:0.Control.ResultsAck	0	Decimal	BOOL
MyDM200:0.Control.TrainCode	0	Decimal	BOOL
MyDM200:0.Control.TrainMatchString	0	Decimal	BOOL
MyDM200:0.Control.TrainFocus	0	Decimal	BOOL
MyDM200:0.Control.TrainBrightness	0	Decimal	BOOL
MyDM200:0.Control.Untrain	0	Decimal	BOOL
MyDM200:0.Control.ExecuteDMCC	0	Decimal	BOOL
MyDM200:0.Control.SetMatchString	0	Decimal	BOOL
+ MyDM200:0.Control.UserDataOption	0	Decimal	INT
+ MyDM200:0.Control.UserDataLength	0	Decimal	INT

3. TriggerReady タグが 1 に変更されます。
4. これでトリガが有効になりました。Trigger タグを 0 から 1 に変更すると、DataMan リードで画像が取り込まれます。現在の TriggerID の値は 1 です。次に発行されるトリガの結果は、対応する ResultID である 1 とともに返されます。

5. 取り込み/デコードが完了すると、DecodeCompleted タグが反転し、ResultsAvailable タグが 1 になります。ここに示す例では、読み取りが正常に行われ (ResultCode ビット 0 = 1)、読み取りで 16 バイトのデータが返されています (ResultLength=16)。データは ResultData タグにあります。

MyDM200.I	{...}		CC:DataMan...
MyDM200.I.Status	{...}		CC:DataMan...
MyDM200.I.Status.TriggerReady	1	Decimal	BOOL
MyDM200.I.Status.TriggerAck	0	Decimal	BOOL
MyDM200.I.Status.Acquiring	0	Decimal	BOOL
MyDM200.I.Status.MissedAcq	0	Decimal	BOOL

MyDM500.I	{...}		CC:DataMan...
MyDM500.I.Status	{...}		CC:DataMan...
MyDM500.I.Status.TriggerReady	1	Decimal	BOOL
MyDM500.I.Status.TriggerAck	0	Decimal	BOOL
MyDM500.I.Status.Acquiring	0	Decimal	BOOL
MyDM500.I.Status.MissedAcq	0	Decimal	BOOL
MyDM500.I.Status.Decoding	0	Decimal	BOOL
MyDM500.I.Status.DecodeCompleted	1	Decimal	BOOL
MyDM500.I.Status.ResultsBufferOverflow	0	Decimal	BOOL
MyDM500.I.Status.ResultsAvailable	1	Decimal	BOOL
MyDM500.I.Status.GeneralFault	0	Decimal	BOOL
MyDM500.I.Status.TrainCodeAck	0	Decimal	BOOL
MyDM500.I.Status.TrainMatchStringAck	0	Decimal	BOOL
MyDM500.I.Status.TrainFocusAck	0	Decimal	BOOL
MyDM500.I.Status.TrainBrightnessAck	0	Decimal	BOOL
MyDM500.I.Status.UntrainAck	0	Decimal	BOOL
MyDM500.I.Status.ExecuteDmccAck	0	Decimal	BOOL
MyDM500.I.Status.SetMatchStringAck	0	Decimal	BOOL
+ MyDM500.I.Status.TriggerID	297	Decimal	INT
+ MyDM500.I.Status.ResultID	296	Decimal	INT
+ MyDM500.I.Status.ResultCode	1	Decimal	INT
+ MyDM500.I.Status.ResultExtended	0	Decimal	INT
+ MyDM500.I.Status.ResultLength	5	Decimal	INT
+ MyDM500.I.ResultData	{...}	ASCII	SINT[484]

明示的メッセージ

暗黙的メッセージとは異なり、明示的メッセージは特定のデバイスに送信され、そのデバイスが常にメッセージに応答します。そのため、明示的メッセージは発生頻度が低い操作に適しています。明示的メッセージは、ID リードオブジェクトの属性（データ）の読み取りおよび書き込みに使用できます。また、画像の取り込み、DMCC コマンドの送信、結果データの取得にも使用できます。

DMCC コマンドの発行

DataMan ID リードに送信される明示的メッセージの例として、DMCC コマンドの実行があります。ControlLogix から DataMan への明示的メッセージの送信には MSG 命令を使用します。明示的メッセージを使用して DMCC メッセージを起動する方法は 2 通りあります。PCCC オブジェクトを使用する方法と、ID リードオブジェクトの “SendDMCC” サービスを使用する方法です。この例では、SendDMCC サービスを使用する方法について説明します。

EtherNet/IP での伝送には CIP STRING2 フォーマット (16 ビット長の値の後に実際の文字列が続く。Null 終了文字は含まれない) が必要です。しかし、Logix では少し異なるフォーマット (32 ビット長の値の後に実際の文字列が続く。Null 終了文字は含まれない) で文字列が格納されます。そのため、一部のサンプルラダーでは、2 つの異なる文字列フォーマットの変換が行われます。

① 注：この例では、DataMan の明示的メッセージの動作を示すことを目的としています。これと同じ操作をより効率的なラダーに書き込むこともできますが、学習にはあまり役立ちません。

6. ControlLogix の [Controller Tags (コントローラタグ)] ダイアログで次のタグを追加します。

Name	Data Type	Style
Send_DMCC_Command	BOOL	Decimal
DMCC_Command_String	STRING	
DMCC_Result_String	STRING	
Message_Data	SINT[128]	Decimal
Message_Result	SINT[128]	Decimal
Message_Pending	BOOL	Decimal
MSG_DMCC	MESSAGE	

Send_DMCC_Command: コマンドを開始するための boolean のフラグ。

DMCC_Command_String: 実行する DMCC コマンドを含む文字列。

DMCC_Result_String: DMCC コマンド結果を受信する文字列。

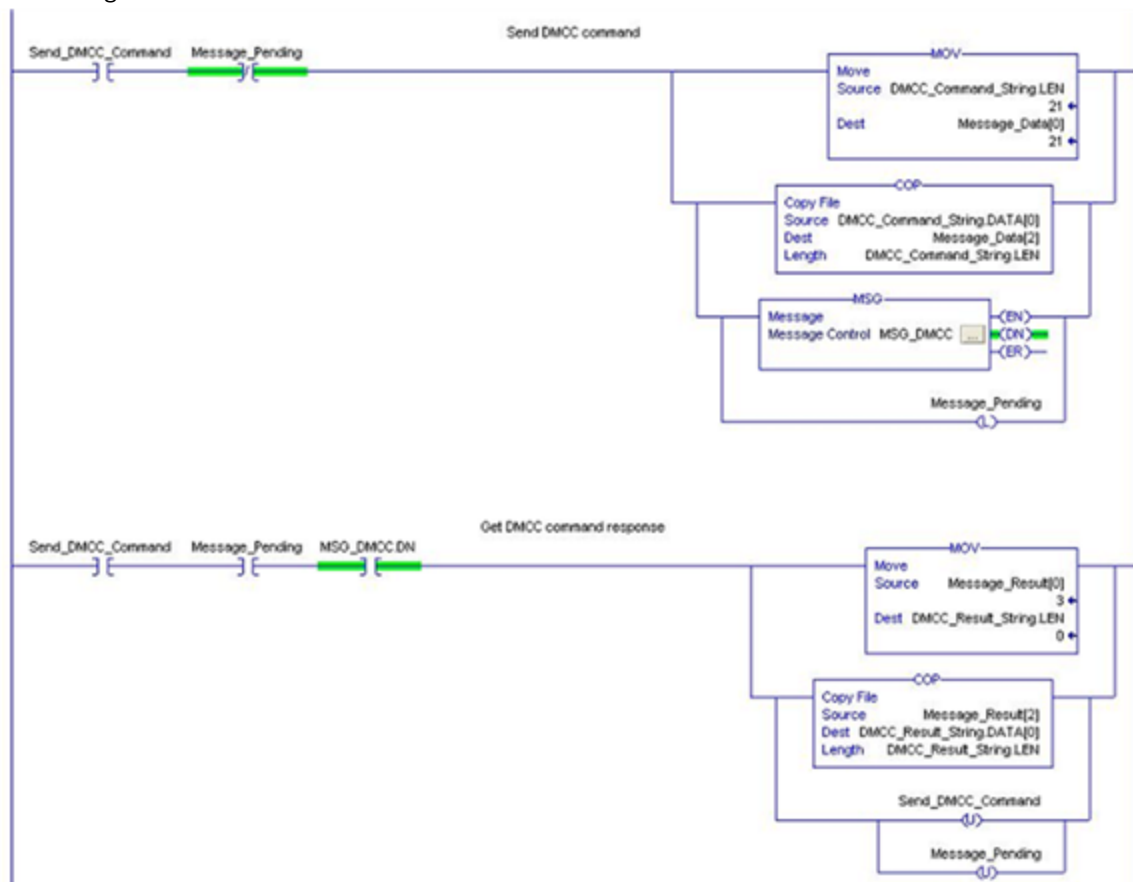
Message_Data: MSG 命令によって送信するデータを格納する一時バッファ。

Message_Result: MSG 命令によって受信するデータを格納する一時バッファ。

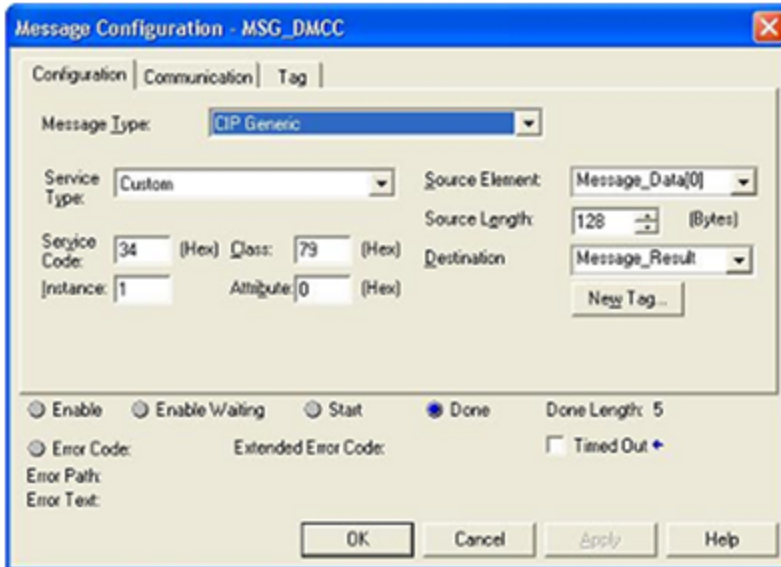
Message_Pending: メッセージが処理中であることを示すための boolean のフラグ。

MSG_DMCC: Logix の MSG 命令に必要なデータ構造。

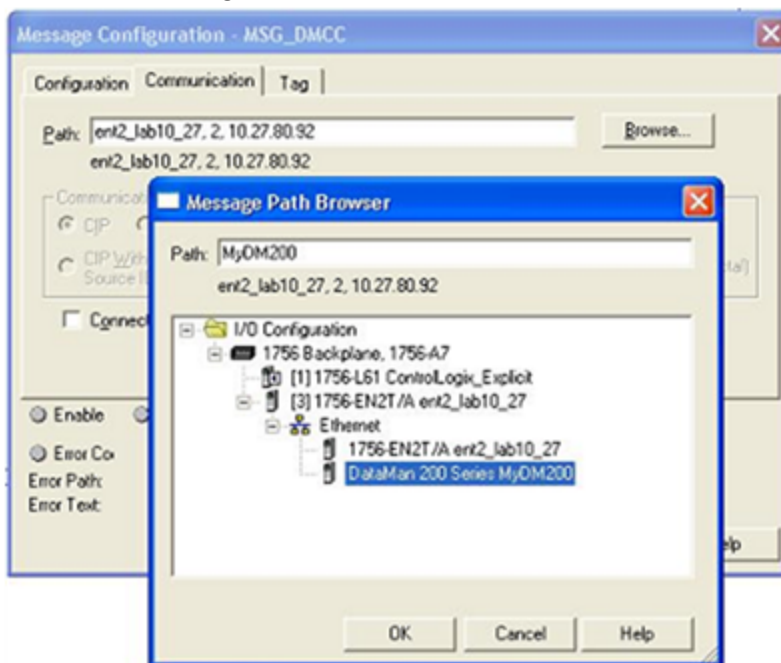
7. ControlLogix プロジェクトの MainRoutine に次の 2 つのラングを追加します。



8. MSG 命令を編集します。「CIP Generic (CIP 汎用)」を選択し、サービス 0x34 で“SendDMCC”、クラス 0x79 で「ID リーダオブジェクト」に設定し、[インスタンス] を 1 に設定します。ソースを“Message_Data”、出力先を“Message_Result”に設定します。



9. MSG 命令の [通信] タブで、プロジェクトの I/O 設定ツリーに追加した DataMan を参照して選択します。これで、明示的メッセージの送信先が Logix に認識されます。



10. ControlLogix にダウンロードし、「実行モード」にします。

11. 操作するには

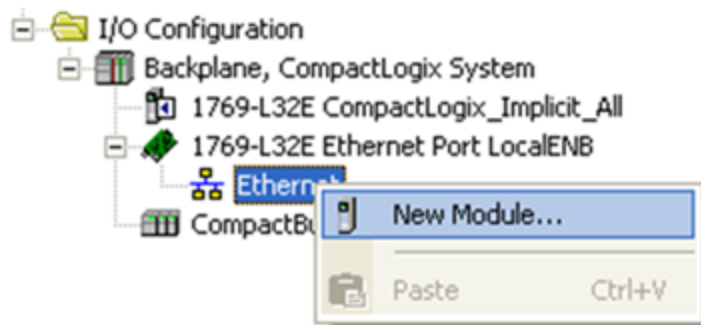
- DMCC コマンドを “DMCC_Command_String” タグに配置します (例: “||>GET TRIGGER.TYPE\$r\$I”)。文字列の末尾を \$r\$I にすることに注意してください。Logix では CRLF がこのように表現されます。
- “Send_DMCC_Command” タグを 1 に反転します。
- “Send_DMCC_Command” タグが 0 に戻れば、実行完了です。DMCC コマンド結果は “DMCC_Result_String” で確認できます。

Rockwell CompactLogix の例

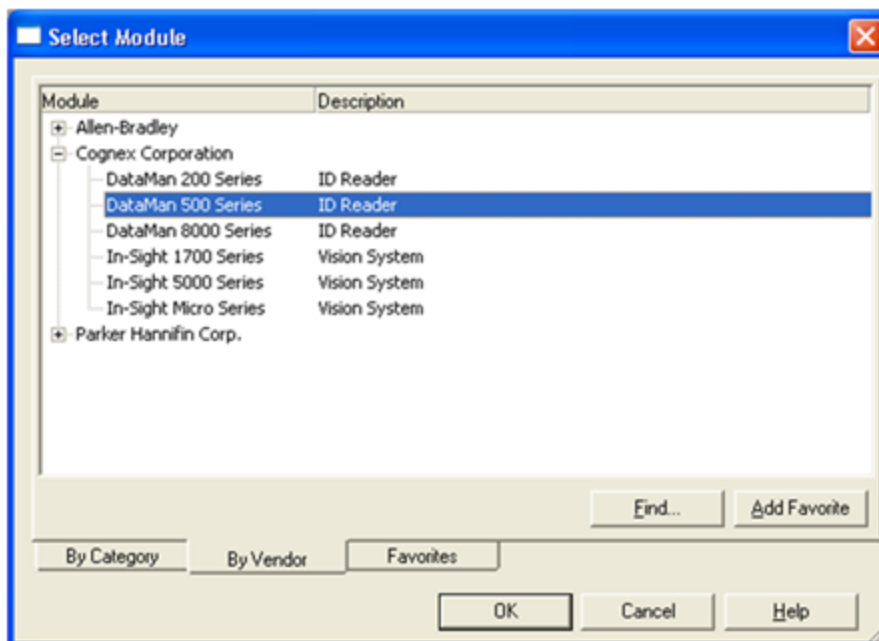
CompactLogix のプログラミングは ControlLogix とほぼ同様です。ControlLogix の例は CompactLogix システムでも同じように使用することができます。ただし、プロジェクト I/O ツリーへの DataMan デバイスの追加にはわずかな差異があります。

CompactLogix プロジェクトと ControlLogix プロジェクトでは、I/O 設定ツリーが少し異なります。Ethernet 接続の面での違いは、CompactLogix プロセッサモジュールには Ethernet ロジックモジュールが組み込まれている点です。I/O 設定ツリーには、バックプレーンの別のモジュールであるかのように表示されます。このモジュールは ControlLogix の Ethernet モジュールとまったく同じように設定されています。

DataMan モジュールの追加方法は CompactLogix でも ControlLogix でも同じです。I/O 設定ツリーで Ethernet ノードを右クリックし、[New Module (新規モジュール)] を選択します。



[Select Module (モジュール選択)] ダイアログで、リストから使用する DataMan ID リーダのモデルを選択します。



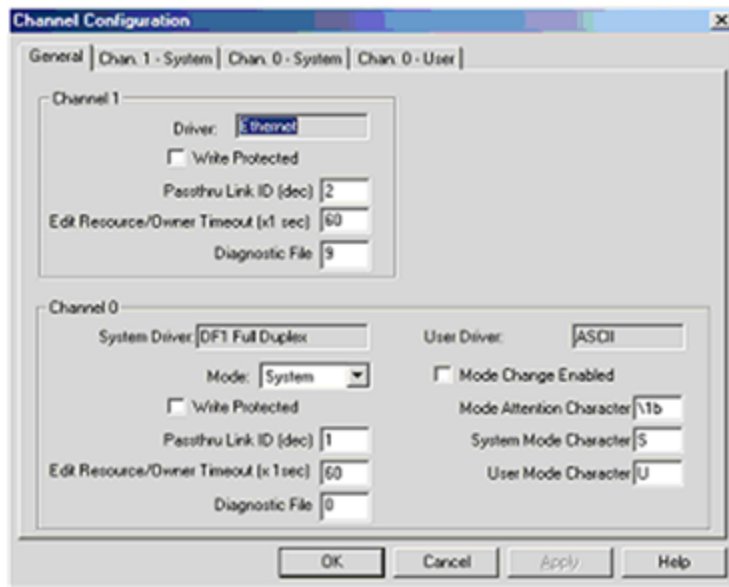
選択すると、DataMan ID リーダシステムの設定ダイアログが表示されます。これ以降の設定およびプログラミングの手順は、前の [ControlLogix](#) の節に記載されている手順と同じです。

Rockwell SLC 5/05 の例

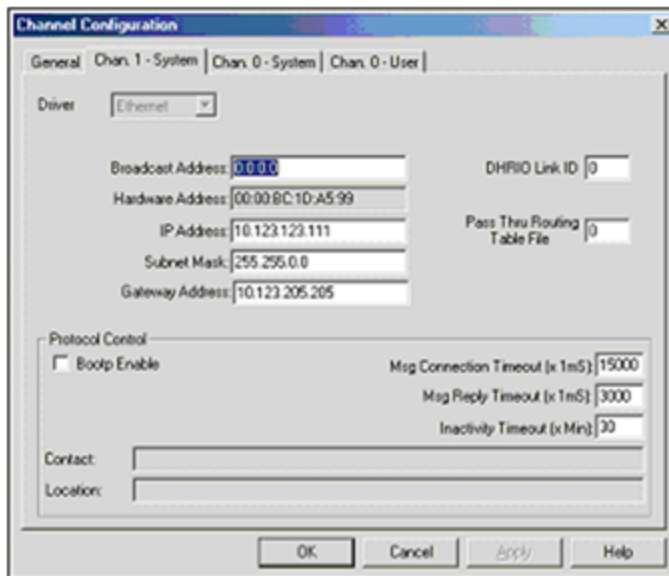
この節では、DataMan リーダと PLC の間での PCCC (PC³) 通信の設定について説明します。この例では Allen-Bradley SLC5/05 と Rockwell 500 ソフトウェアを使用します。

PLC での Ethernet 通信の設定

1. RSLogix 500 ソフトウェアプログラムから .RSS ファイルを開き、[Channel Configuration (チャンネル設定)] ダイアログ ([プロジェクトフォルダ] → [Controller Folder (コントローラフォルダ)] → [Channel Configuration (チャンネル設定)]) を開きます。



2. Allen-Bradley SLC では 2 つのチャンネルを設定できます。チャンネル 1 (Ethernet) とチャンネル 0 (DF1 全二重 - シリアル) です。[Chan.1 - System (チャンネル 1 - システム)] タブをクリックします。
3. 必要に応じてチャンネル 1 (Ethernet) を設定します。適切な設定についてはネットワーク管理者にお問い合わせください。



4. 必要に応じて[タイムアウト]を設定します。

メッセージ命令 (MSG)

アプリケーション内でメッセージ命令を設定できるようになりました。メッセージ作成のための拡張指示については、RSLogix 500 のドキュメントを参照してください。

メッセージ (MSG) 命令内では次の設定パラメータを設定することができます。



- **Type:** Peer-To-Peer。変更できません。
- **Read/Write:** DataMan リーダで実行する機能を選択します。「Read」では DataMan からデータを取得し、「Write」では DataMan にデータを送信します。
- **Target Device:** DataMan リーダと通信する PLC5 を選択します。これによって SLC が使用する通信プロトコルを指定します。DataMan リーダは ControlLogix コントローラのように機能します (**Rockwell のマニュアル 13862** を参照)。
- **Local/Remote:** DataMan リーダが SLC と同じネットワーク上にある場合は「Local」を、別のネットワーク上にある場合は「Remote」を選択します。リモート通信では、セカンダリネットワークへのゲートウェイとして機能する別のデバイスを通じてメッセージを直接送信する必要があります。通常、Allen-Bradley ControlLogix コントローラがその役割を果たします (ゲートウェイを通じて他のネットワーク上のデバイスのアドレスを指定する方法については、**Rockwell** のドキュメントを参照してください)。
- **Control Block:** これは MSG 命令でデータ (IP アドレス、メッセージタイプなど) を格納するために使用する一時整数ファイルです。通常、送信されるユーザデータではありません。
- **Control Block Length:** これは MSG 命令によって自動的に計算されます。
- **Setup Screen:** 「Setup Screen」を選択すると、[Message Instruction Setup (メッセージ命令の設定)] ダイアログが開きます。

[MSG Instruction Setup (メッセージ命令の設定)] 画面では次の設定パラメータを設定することができます。

[This Controller (このコントローラ)] セクション:

- **[Communication Command (通信コマンド)]**: 最初の画面 (MSG 命令画面) と同じコマンド (READ/WRITE) である必要があります。
- **[Data Table Address (データテーブルアドレス)]**: これは SLC でデータの書き込み先 (READ) または送信元 (WRITE) になるデータファイルの場所です (MSG 命令画面と同様)。このインスタンスでは、'N7:0' の 'N' は整数ファイル、'7' はファイル番号 7、'0' はそのファイルへのオフセット (この場合は 0 番目の要素から開始) を示します。次の図に、RSLogix 500 メイン画面からアクセスした整数テーブルの例を示します。
- **[Size in Elements (要素のサイズ)]**: 読み取り対象の要素 (個々のデータ) の数です。この例では、2 つの要素を読み取ります。
- **[チャンネル]**: SLC の設定によって異なります。SLC では、チャンネル 1 は Ethernet ポートです。
- **[Message Timeout (メッセージのタイムアウト)]**: DataMan リーダが応答できる適切な時間の長さを選択します。DataMan がこの時間内に応答しない場合、MSG 命令はエラーになります。この画面からは、このパラメータを変更することはできません。メッセージのタイムアウトは、[Channel 1 setup (チャンネル 1 の設定)] ダイアログで入力したパラメータによって決定されます。
- **[Data Table Address (データテーブルアドレス)]**: これは DataMan リーダでデータの読み取り元または書き込み先となる場所です。このインスタンスでは、'N7:1' の 'N' はデータが整数型 (16 ビット) であること、'7' は DataMan によって無視されること (データは常に出力アセンブリに書き込まれ、入力アセンブリから読み込まれる)、'1' はターゲットバッファの先頭からの要素オフセットを示します。例えば、メッセージが READ の場合、'N7:2' では 3 番目の整数 (SLC はゼロベースのインデックスであるため、'2' は 3 番目の要素を指します) を入力アセンブリから読み取ります (READ では DataMan の入力アセンブリからデータを取得するため)。メッセージが WRITE の場合、'N7:12' では出力アセンブリの 13 の整数位置に (16 ビットの) 整数値を書き込みます。

注: ST10:0 という宛先アドレスは、DMCC コマンドを DataMan リーダに送信する特別な用途のためにあります。ST10:0 に送信される文字列は DMCC コマンドとして解釈されます。



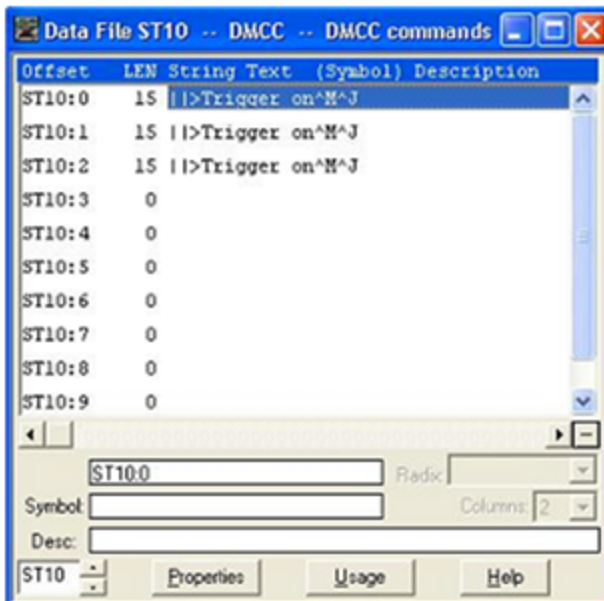
- **[Local/Remote (ローカル/リモート)]**: アプリケーションに応じて、[ローカル] または [リモート] に設定されます。
- **MultiHop (マルチホップ)**: この設定は以前に入力した情報によって異なります。In-Sight 通信を正常に行うため、この時点では [YES(あり)] にする必要があります。

SLC 5/05 からの DMCC コマンドの送信

1. 必要に応じて SLC5/05 を設定します。
2. DMCC コマンドを格納する文字列テーブルを作成します。



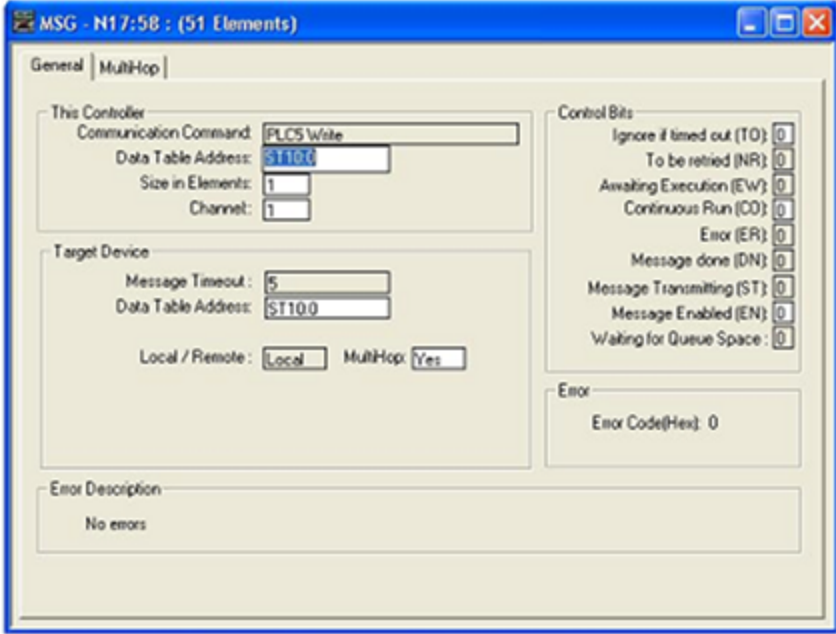
3. 必要な DMCC コマンド文字列をデータファイルに追加します。



4. 新しいメッセージ (MSG) 命令をラダーロジックに追加し、次の例のように設定します。



5. [MSG Setup (MSG 設定)] 画面で次のように設定します。



This Controller (このコントローラ)	パラメータ	説明
Data Table Address (データテーブルアドレス)	ST10:0	前に作成した文字列テーブル (ST) からの最初の要素です。
Size in Elements (要素のサイズ)	1	常に 1 に設定します。PCCC MSG で同時に送信できる文字列は 1 つ (1 つのコマンド) のみです。
チャンネル	1	コントローラの Ethernet チャンネルに設定します。

ターゲットデバイス	パラメータ	説明
Message Timeout (メッセージのタイムアウト)	(チャンネル設定ダイアログから)	
Data Table Address (データテーブルアドレス)	ST10:0	これは宛先アドレスです。DMCC コマンドの場合は常に ST10:0 です。

6. [MultiHop (マルチホップ)] タブをクリックし、必要に応じて設定します (DataMan の IP アドレス)。
7. すべて設定し終えたら、MSG ウィンドウを閉じます。
8. ラダーロジックを保存してコントローラにダウンロードし、オンラインでコントローラを実行モードに設定します。
9. DataMan リーダへのメッセージ送信をトリガします。

メッセージ命令の結果



命令への入力が high にセットされると、メッセージ命令の有効化 (EN) ビットが 1 にセットされます。DataMan が DMCC コマンドの受信と正常実行したと応答すると、完了 (DN) ビットが 1 にセットされます。エラー (ER) ビットが有効になった (1 にセットされた)

場合、メッセージ命令に問題が発生したことを示します。エラーが発生した場合、MSG 命令の設定画面をクリックします。エラーコードはウィンドウ下部に表示されます。

汎用 EtherNet/IP プロファイルの使用

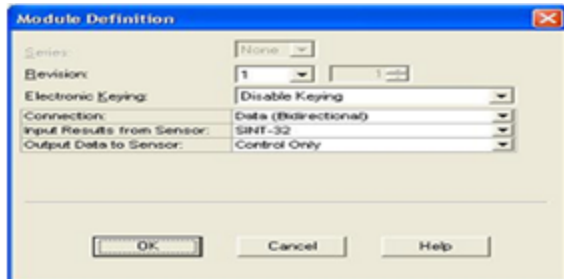
アドオンプロファイルのないデバイス向けに、Rockwell は汎用 EtherNet/IP プロファイルを用意しています。このプロファイルによって、特定の製品アドオンプロファイルの自動タグ生成機能がなくても暗黙的メッセージ接続を作成することができます。

汎用の暗黙的メッセージ接続の確立

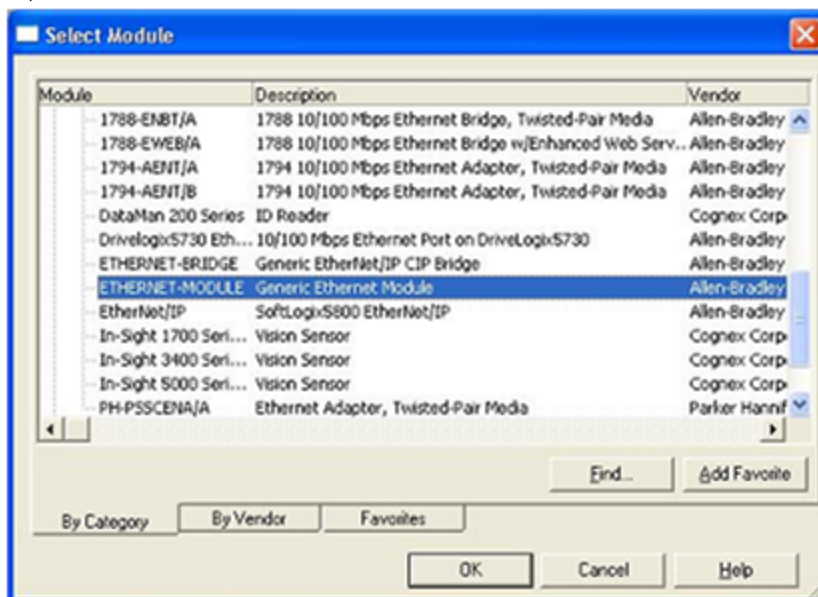
DataMan と ControlLogix コントローラの間で EtherNet/IP の暗黙的メッセージ接続を設定するには、まず DataMan リーダを ControlLogix I/O 設定ツリーに追加する必要があります。これは Rockwell が提供する汎用プロファイルで可能です。

ControlLogix PLC との汎用の暗黙的メッセージ接続を確立するには

1. RSLogix5000 を開き、プロジェクトをロードします (または、[ファイル] → [新規] を選択して新規プロジェクトを作成します)。
2. I/O 設定ノードからプロジェクトの Ethernet モジュールの下にある Ethernet ノードを選択し、アイコンを右クリックしてメニューから [New Module (新規モジュール)] を選択します。

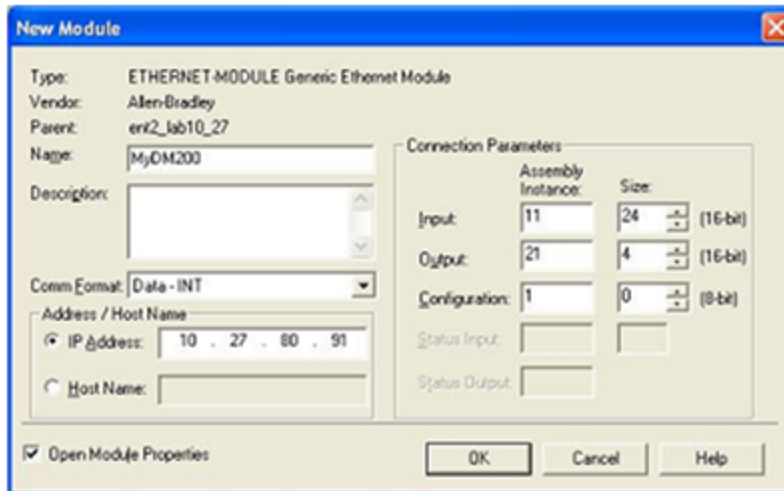


3. [Select Module (モジュール選択)] ダイアログで、Allen-Bradley の「Generic Ethernet Module (汎用 Ethernet モジュール)」を選択します。

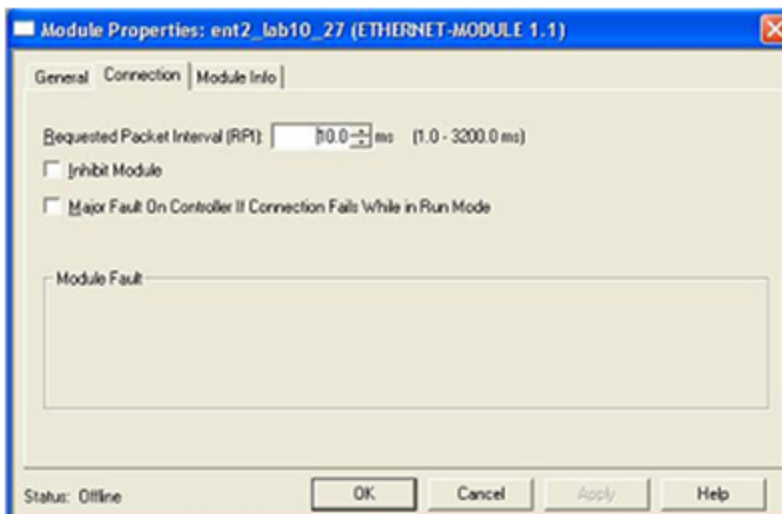


4. 選択すると、汎用 Ethernet モジュールの設定ダイアログが表示されます。次のように設定します。

- モジュールに名前を付けます。
- DataMan の IP アドレスを入力します。
- 通信フォーマットを「Data - INT」に設定します。モジュールにデータを 16 ビット整数の配列として扱うように指示します。
- 入力アセンブリ: インスタンス 11 を設定します。サイズを PLC が受信する入力アセンブリデータ数に設定します。基本的な「ステータス」データでは 8 個の整数が必要です。それを超える数は実際のデコード結果データです。下記の例では、サイズが 24 (ステータス用に 8 + 結果データ用に 16) に設定されています。この接続では、ステータス情報と 32 バイトの結果データを受信します。
- 出力アセンブリ: インスタンス 21 を設定します。サイズを 4 個の整数に設定します。このサイズは、必要なすべての「コントロール」データを DataMan に送信するのに十分です。
- 設定アセンブリ: インスタンス 1 を設定します。サイズを 0 (使用しない) に設定します。



5. 最後に接続レートを設定します。データの送受信レートは Requested Packet Interval (RPI) として定義されます。RPI は接続でデータが送受信される頻度を定義します。ネットワークパフォーマンスを最適化するには、このレートをアプリケーションが確実に必要とするレート以上にする必要があります。どのような場合でも、PLC ラダープログラムの中央スキャンレートの $\frac{1}{2}$ 未満に設定しないでください。それよりも小さく設定すると帯域幅が浪費され、処理パフォーマンスが向上しません。



6. ControlLogix に汎用モジュールを追加すると、I/O ツリーが次のように表示されます。



7. 汎用モジュールが I/O ツリーに追加されると、RSLogix 5000 によって DataMan リーダの入力および出力データ (DataMan リーダの入力および出力アセンブリオブジェクト) に対応するタグが作成されます。これらのタグはプロジェクトツリーの [Controller Tags (コントローラタグ)] ノードにあります。

注：これらのタグの基本名は、前に I/O 設定に追加した汎用モジュールに付けた名前です。

Name	Value	Style	Data Type
+ MyDM200:C	{...}		AB:ETHERNET_MODULE:C:0
- MyDM200:I	{...}		AB:ETHERNET_MODULE_INT_48bytes:I:0
+ MyDM200:I.Data	{...}	Decimal	INT[24]
- MyDM200:O	{...}		AB:ETHERNET_MODULE_INT_8bytes:O:0
+ MyDM200:O.Data	{...}	Decimal	INT[4]

タグは 3 つのグループ (設定 “MyDM200:C”、入力 “MyDM200:I”、出力 “MyDM200:O”) に分けられています。設定タグは無視してかまいません (使用しません)。入力タグは (DataMan から) 受信するすべてのデータを表します。出力タグは (DataMan に) 送信するすべてのデータを表します。

これらのタグは、DataMan アセンブリオブジェクトのコンテンツのデータテーブル表現です。これらのタグ値にアクセスするには、PLC ラダーに書き込みます。PLC ラダーはこれらのタグ値を監視または変更することで、DataMan アセンブリオブジェクトのコンテンツの監視および変更を行います。

注：DataMan とこれらの PLC タグ値には (設定した RPI に基づく) 遅延が生じます。すべての PLC ラダーはその遅延を考慮に入れて書き込まれる必要があります。

汎用の暗黙的メッセージ接続データへのアクセス

上記の節では、汎用モジュールプロファイルを使用して ControlLogix と DataMan ID リーダの間で暗黙的メッセージ接続を確立する方法の詳細について説明しました。DataMan アドオンプロファイルとは異なり、汎用プロファイルではアセンブリオブジェクト内の個々のデータ項目を表す名前付きタグが自動生成されません。代わりに、定義した接続のサイズに従ってデータ配列が生成されます。

アセンブリオブジェクト内の個々のデータ項目にアクセスするには、汎用プロファイルによって提供されるタグ配列内で適切なタグオフセットとデータサブタイプ (必要な場合) を手動で選択する必要があります。アセンブリオブジェクトを定義するベンダードキュメントを手動で参照する必要があるため、これは煩雑で間違いの元になります。

注：入力タグ “MyDM200:I.Data[0]” の始まりは、DataMan 入力アセンブリの始まりに直接対応しています。同様に、出力タグ “MyDM200:O.Data[0]” の始まりは、DataMan 出力アセンブリの始まりに直接対応しています。

例

入力アセンブリ “TriggerReady”: 入力アセンブリのワード 0 のビット 0。DataMan の入力タグ配列からワード 0 のビット 0 を選択します。

MyDM200:I	{ ... }		AB:ETHERNET_MODULE_INT_48Bytes:I:0
MyDM200:I.Data	{ ... }	Decimal	INT[24]
MyDM200:I.Data[0]	0	Decimal	INT
MyDM200:I.Data[0].0	1	Decimal	BOOL
MyDM200:I.Data[0].1	0	Decimal	BOOL
MyDM200:I.Data[0].2	0	Decimal	BOOL
MyDM200:I.Data[0].3	0	Decimal	BOOL
MyDM200:I.Data[0].4	0	Decimal	BOOL
MyDM200:I.Data[0].5	0	Decimal	BOOL
MyDM200:I.Data[0].6	0	Decimal	BOOL
MyDM200:I.Data[0].7	0	Decimal	BOOL
MyDM200:I.Data[0].8	0	Decimal	BOOL
MyDM200:I.Data[0].9	0	Decimal	BOOL
MyDM200:I.Data[0].10	0	Decimal	BOOL
MyDM200:I.Data[0].11	0	Decimal	BOOL
MyDM200:I.Data[0].12	0	Decimal	BOOL
MyDM200:I.Data[0].13	0	Decimal	BOOL
MyDM200:I.Data[0].14	0	Decimal	BOOL
MyDM200:I.Data[0].15	0	Decimal	BOOL

入力アセンブリ “ResultLength”: 入力アセンブリのワード 7。DataMan の入力タグ配列からワード 7 を選択します。

MyDM200:I	{ ... }		AB:ETHERNET_MODULE_INT_48Bytes:I:0
MyDM200:I.Data	{ ... }	Decimal	INT[24]
MyDM200:I.Data[0]	0	Decimal	INT
MyDM200:I.Data[1]	0	Decimal	INT
MyDM200:I.Data[2]	0	Decimal	INT
MyDM200:I.Data[3]	0	Decimal	INT
MyDM200:I.Data[4]	0	Decimal	INT
MyDM200:I.Data[5]	0	Decimal	INT
MyDM200:I.Data[6]	0	Decimal	INT
MyDM200:I.Data[7]	0	Decimal	INT
MyDM200:I.Data[8]	0	Decimal	INT

出力アセンブリ “Trigger”: 出力アセンブリのワード 0 のビット 1。DataMan の出力タグ配列からワード 0 のビット 1 を選択します。

SLMP プロトコル

SLMP プロトコルは標準の Ethernet ハードウェアおよびソフトウェアを使用して、I/O データ、アラーム、診断のやり取りを行います。これは、Ethernet またはシリアル接続を介して Q、iQ、および L シリーズの PLC と通信するために三菱電機が提供している標準化された通信フォーマットです。DataMan では SLMP プロトコルを Ethernet でのみサポートしています。

デフォルトでは、DataMan では SLMP プロトコルは無効になっています。プロトコルを有効にするには、Setup Tool を使用するか、DMCC を送信、もしくはパラメータコードをスキャンします。

注：ワイヤレス DataMan リーダをお持ちの場合は、「[ワイヤレス DataMan 向けの産業用プロトコル](#)」の節を参照してください。

DMCC

次のコマンドを使用して SLMP プロトコルを有効または無効にすることができます。コマンドの発行は RS-232 または Telnet 接続を介して行います。

注：DataMan と通信するには、Windows が提供する Telnet クライアントの設定を変更する必要があるので、PuTTY などの Cognex 社以外のクライアントを使用することを推奨します。

有効:

```
||>SET SLMP-PROTOCOL.ENABLED ON
||>CONFIG.SAVE
||>REBOOT
```

無効:

```
||>SET SLMP-PROTOCOL.ENABLED OFF
||>CONFIG.SAVE
||>REBOOT
```

リーダ設定コード

次のリーダ設定コードをスキャンすると、有線式リーダで SLMP プロトコルが有効または無効になります。

注：変更を反映するにはデバイスを再起動する必要があります。

有効:



無効:



次のリーダ設定コードをスキャンすると、DataMan 8000 ベースステーションで SLMP プロトコルが有効または無効になります。

注：変更を反映するにはデバイスを再起動する必要があります。

有効:



無効:



Setup Tool

SLMP プロトコルを有効にするには、[産業用プロトコル] ペインの [SLMP Protocol (SLMP プロトコル)] タブで [有効] をオンにします。リーダから切断する前に、[設定の保存] を選択して新しい選択内容を保存してください。

注： 新しい設定を反映するにはリーダを再起動する必要があります。

SLMP プロトコルスキャナ

DataMan の SLMP プロトコルは、スキャナとも呼ばれるクライアント型デバイスとして実装されています。すべての通信は、読み取りおよび書き込み要求の形式で DataMan リーダによって開始されます。PLC は、読み取りおよび書き込み要求に応答するパッシブサーバとして動作します。PLC は通信を開始できないため、PLC に対して必要なアクションや情報 (トリガや読み取り結果の取得など) を定期的に問い合わせ (スキャン) します。

はじめに

デフォルトでは、SLMP プロトコルは DataMan リーダで有効になっていません。このプロトコルを有効にして、PLC と正しく情報をやり取りするようにプロトコル設定パラメータを設定する必要があります。プロトコル設定は DataMan Setup Tool で実行します。

1. Windows の [スタート] メニューから DataMan Setup Tool を起動します。
2. [設定] → [通信設定] で [産業用プロトコル] をクリックします。

3. [SLMP Protocol (SLMP プロトコル)] タブを選択します。

DM262-4B5732

産業用プロトコル

EtherNet/IP™ PROFINET **SLMP プロトコル** Modbus TCP

☒ 有効

IP アドレス: . . . ネットワーク番号: 0

ホストポート: 0x 3000 PC 番号 [16進]: 0x FF

タイムアウト [ミリ秒]: 1000 宛先モジュール:

ポーリング間隔 [ミリ秒]: 1000 0x3FF = ローカルステーション

PLC シリーズ: QCPU

名前	選択したデバイス	オフセット	デバイス数	説明
コントロール	なし	0	0	画像制御ブロックの開始アドレスです。
ステータス	なし	0	0	画像ステータスブロックの開始アドレスです。
PLC 入力	なし	0	0	ユーザーデータブロックの開始アドレスです。
PLC 出力	なし	0	0	検査結果ブロックの開始アドレスです。
コマンド	なし	0	0	コマンドストリングの開始アドレスです。
コマンドの実...	なし	0	0	コマンド実行結果データの開始アドレスです。

ステータス:

4. プロトコルを有効にして、適切な設定を行います。

SLMP プロトコル設定は、ネットワーク情報の定義およびやり取りを行うデータの定義の 2 つの局面で構成されています。すべての設定パラメータは、[SLMP Protocol (SLMP プロトコル)] タブからアクセスします。

PLC のアドレスに合わせて [IP アドレス] を変更する必要があります。また、使用しているネットワークと異なる場合は、[ネットワーク番号]、[PC 番号]、[宛先モジュール] も変更します。

注： [システム] → [設定の保存] を選択し、SLMP プロトコルに加えた設定を保存してください。また、新しい設定を反映するには、リーダを再起動する必要があります。

ネットワーク設定

ネットワーク設定では、DataMan リーダが PLC との接続を確立するために必要なすべての情報を定義します。

名前	デフォルト	範囲	説明
IP アドレス	<空>	任意の有効な IP アドレス	接続先の PLC の IP アドレス

名前	デフォルト	範囲	説明
ホストポート (16 進数)	3000	任意の 16 進数のポート番号 1000 ~ FFFF	PLC 上の SLMP プロトコルチャンネルのポート番号
タイムアウト (ミリ秒)	1000	5 ~ 30000	PLC から SLMP プロトコルのメッセージへの応答時間 (単位: ミリ秒)。
ポーリング間隔 (ミリ秒)	1000	10 ~ 30000	PLC からのコントロールブロックの連続ポーリング間の要求時間 (単位: ミリ秒)。
PLC シリーズ	QCPU	QCPU または LCPU	使用するフレームタイプを定義します。現時点では 3E のみサポートされています。
ネットワーク番号	0	0 ~ 239	通信する SLMP プロトコルのネットワーク番号 (0 = ローカルネットワーク)
PC 番号	0xFF	1 ~ 120 = CC-Link IE フィールドネットワークアダプタのステーション 126 = CC-Link IE フィールドネットワークの マスタステーション 255 = ローカルステーションに直接接続	宛先モジュールの指定したネットワークのステーション識別子。
宛先モジュール	0x3FF	0x3ff = ローカルステーション (デフォルト) 0x3d0 = CPU 制御システム 0x3d1 = CPU 待機システム 0x3d2 = システム A CPU 0x3d3 = システム B CPU 0x3e0 = CPU 1 0x3e1 = CPU 2 0x3e2 = CPU 3 0x3e3 = CPU 4	接続先デバイスのモジュール識別子。

データブロック設定

データブロック設定では、DataMan リーダと PLC の間でやり取りされるデータを定義します。使用可能なデータブロックは 6 個です。それぞれのブロックにはあらかじめ定義された機能があります。

すべてのデータブロックが必要なわけではありません。アプリケーションに必要なデータブロックのみを設定してください。通常、ほとんどのデータフローを制御する、コントロールブロックとステータスブロックを定義します。ただし、これらのブロックすら不要な場合もあります。

データブロックを設定するには、PLC デバイスタイプ (メモリタイプ)、デバイスのオフセット、データブロックに含まれるデバイス数を定義します。デバイスタイプまたはデバイス数を定義しないと、そのブロックは使用されません (そのブロックのデータは交換されません)。

ブロック名	サポートされるデバイスタイプ	オフセット	デバイス数
コントロール	<なし>、D、W、R、ZR、M、X、Y、L、F、B	0 ～ 65535	タイプが <なし> の場合: 0 ビットタイプの場合: 32 ワードタイプの場合: 2 (読み取り専用)
ステータス	<なし>、D、W、R、ZR、M、X、Y、L、F、B	0 ～ 65535	タイプが <なし> の場合: 0 ビットタイプの場合: 32 ワードタイプの場合: 2 (読み取り専用)
PLC 入力	なし、D、W、R、ZR	0 ～ 65535	0 ～ 960
PLC 出力	なし、D、W、R、ZR	0 ～ 65535	0 ～ 960
コマンド	なし、D、W、R、ZR	0 ～ 65535	0 ～ 960
コマンド 結果	なし、D、W、R、ZR	0 ～ 65535	0 ～ 960

インタフェース

この節では、SLMP プロトコルを介して PLC で表示される DataMan リーダのインタフェースについて説明します。インタフェースモデルは、3 つの論理ペアに分けられた 6 個のデータブロックで構成されています。

- ・ コントロールとステータス
- ・ 入力データと出力データ
- ・ 文字列コマンドと文字列応答

すべてのブロックが必要なわけではありません。特定のアプリケーションに適したブロックを選択できます。ただし、コントロールおよびステータスは一般的にほとんどのアプリケーションに含まれています。

アプリケーションで使用する各インタフェースブロックの開始アドレスとデバイスタイプを定義できます。未定義ブロックのデータは交換されません。値の変更がブロック内の 1 つのフィールドのみでも、転送 (読み取りまたは書き込み) ではブロック全体が送信されます。プロトコル実装では、できるだけ多くの値の変化を論理的に 1 回の転送にまとめることで、ネットワーク使用量が最小化されます。

コントロールブロック

コントロールブロックにはビットタイプのデータが含まれます。ただし、PLC ではブロックがビットメモリまたはワードメモリに存在するよう定義されていることがあります。このブロックは、PLC からリーダーに送信される制御信号で構成されます。PLC はこれを使用してアクションを開始し、特定のデータ転送に対して応答します。

ビット 7	ビット 6	ビット 5	ビット 4	ビット 3	ビット 2	ビット 1	ビット 0
Reserved				Results ¥Ack	Buffer Results Enable	Trigger	Trigger Enable
ビット 15	ビット 14	ビット 13	ビット 12	ビット 11	ビット 10	ビット 9	ビット 8
Reserved							
ビット 23	ビット 22	ビット 21	ビット 20	ビット 19	ビット 18	ビット 17	ビット 16
Reserved						Initiate StringCmd	SetUser Data
ビット 31	ビット 30	ビット 29	ビット 28	ビット 27	ビット 26	ビット 25	ビット 24
SoftEvent 7	SoftEvent 6	SoftEvent 5	SoftEvent 4	SoftEvent 3	SoftEvent 2	SoftEvent 1	SoftEvent 0

コントロールブロックのフィールドの説明

ビット	名前	説明
0	Trigger Enable	このフィールドは「Trigger」ビットを介してトリガを有効にするためにセットします。このフィールドをクリアすると、ネットワークトリガのメカニズムが無効になります。
1	Trigger	このビットをセットすると、画像取り込みがトリガされます。 注： 画像取り込みをトリガする前に、「TriggerReady」ビットが high にセットされている必要があります。
2	Buffer Results Enable	このビットがセットされている場合、各読み取り結果セット (ResultID、ResultCode、ResultLength、ResultData フィールド) に応答があるまで「出力ブロック」に保持されます。応答があった後、次の読み取り結果セットをバッファから取得できるようになります。前の読み取りセットの応答がある前に新しい読み取り結果が届いた場合、新しい読み取りセットはリーダのバッファのキューに入れられます。リーダのバッファに保持できる読み取り結果セットは最大 6 個です。応答ハンドシェイクシーケンスの説明については、「 操作 」の節を参照してください。
3	ResultsAck	最新の結果 (ResultID、ResultCode、ResultLength、ResultData フィールド) を受信したことに応答するために、PLC によってセットされます。リーダはこのビットが 0 から 1 に遷移したことを認識すると、ResultsAvailable ビットをクリアします。これが PLC とリーダ間の論理ハンドシェイクを形成します。結果のバッファ処理が有効になっている場合、応答を受けてキューで待機中の次の結果がバッファから取り出されます。応答ハンドシェイクシーケンスの説明については、「 操作 」の節を参照してください。
4 ~ 15	Reserved	将来使用
16	SetUserData	新しい UserData が使用可能であることを示すために、PLC によってセットされます。リーダは新しい UserData を読み取った後、転送が完了したことを示すために SetUserDataAck をセットします。これが PLC とリーダ間の論理ハンドシェイクを形成します。
17	Initiate StringCmd	新しい StringCommand が使用可能であることを示すために、PLC によってセットされます。リーダはコマンドを処理した後、コマンドの結果が使用可能であることを示すために StringCmdAck をセットします。これが PLC とリーダ間の論理ハンドシェイクを形成します。
18 ~ 23	Reserved	将来使用
24 ~ 31	SoftEvents	ビットは仮想ディスクリット入力として機能します。ビットが 0 から 1 に遷移すると、関連付けられたアクションが実行されます。リーダはアクションを実行した後、アクションが完了したことを示すために対応する SoftEventAck をセットします。これが PLC とリーダ間の論理ハンドシェイクを形成します。 Bit0: コード登録 Bit1: 一致文字列の登録 Bit2: 焦点位置の登録 Bit3: 明るさの登録 Bit4: 登録解除 Bit5: 予約済み (将来使用) Bit6: DMCC コマンドの実行 Bit7: 一致文字列の設定

ステータスブロック

ステータスブロックにはビットタイプのデータが含まれます。ただし、PLC ではブロックがビットメモリまたはワードメモリに存在するよう定義されていることがあります。このブロックは、リーダから PLC に送信されるステータス信号で構成されます。リーダはこれを使用してステータスを示し、特定のデータ転送のハンドシェイクを行います。

ビット 7	ビット 6	ビット 5	ビット 4	ビット 3	ビット 2	ビット 1	ビット 0
Reserved				Missed Acq	Acquiring	Trigger Ack	Trigger Ready
ビット 15	ビット 14	ビット 13	ビット 12	ビット 11	ビット 10	ビット 9	ビット 8
General Fault	Reserved			Results Available	Results Buffer Overrun	DecodeComplete Toggle	Decoding
ビット 23	ビット 22	ビット 21	ビット 20	ビット 19	ビット 18	ビット 17	ビット 16

Reserved						String Cmd Ack	SetUser DataAck
ビット 31	ビット 30	ビット 29	ビット 28	ビット 27	ビット 26	ビット 25	ビット 24
SoftEvent Ack 7	SoftEvent Ack 6	SoftEvent Ack 5	SoftEvent Ack 4	SoftEvent Ack3	SoftEvent Ack 2	SoftEvent Ack 1	SoftEvent Ack 0

ステータスブロックのフィールドの説明

ビット	名前	説明
0	Trigger Ready	リーダが新規「トリガ」を受け入れ可能であることを示します。リーダは、TriggerEnable がセットされ、新規トリガの受け入れ準備ができているときにこのビットをセットします。
1	Trigger Ack	「Trigger」がセットされたことをリーダが認識していることを示します。「Trigger」ビットがクリアされるまで、このビットはセットされたままです。
2	Acquiring	リーダが画像取り込み中であることを示すためにセットされます。
3	Missed Acq	リーダが要求された画像取り込みトリガに失敗したことを示します。次の画像取り込みが発行されると、このビットはクリアされます。
4 ~ 7	Reserved	予約済み
8	Decoding	リーダが画像のデコード中であることを示すためにセットされます。
9	Decode Complete Toggle	新しい結果データが使用可能であることを示します。新しい結果データが使用可能になるたびに、ビットの状態が (0 から 1 へ、または 1 から 0 へ) 反転します。
10	Results Buffer Overrun	PLC が以前の結果に回答していないためリーダが一連の読み取り結果を破棄したことを示すためにセットされます。次の結果データセットが正常にバッファのキューに入れられた時点でクリアされます。このビットは、結果のバッファ処理が有効である場合のみ意味を持ちます。
11	Results Available	新しい結果データが使用可能であることを示すためにセットされます。追加の新しい読み取り結果が使用可能であっても、ResultsAck で応答されるまで、このビットはセットされたままです。
12 ~ 14	Reserved	予約済み
15	General Fault	Ethernet 通信エラーが発生したことを示すためにセットされます。現時点では、ソフトイベント操作によってのみ使用されています。次のソフトイベントが成功するまで、または TriggerEnable が low にセットされ再び high に設定されるまで、このビットはセットされたままです。
16	SetUser DataAck	リーダが新しい UserData を受信したことを示すためにセットされます。対応する SetUserData ビットがクリアされるまで、このビットはセットされたままです。これが PLC とリーダ間の論理ハンドシェイクを形成します。
17	String Cmd Ack	リーダが最新の文字列コマンドの処理を完了し、コマンド応答が使用可能であることを示すためにセットされます。対応する InitiateStringCmd ビットがクリアされるまで、このビットはセットされたままです。これが PLC とリーダ間の論理ハンドシェイクを形成します。
18 ~ 23	Reserved	予約済み
24 ~ 31	SoftEvent Ack	リーダがソフトイベントアクションを完了したことを示すためにセットされます。対応する SoftEvent ビットがクリアされるまで、このビットはセットされたままです。これが PLC とリーダ間の論理ハンドシェイクを形成します。 Bit0: コード登録応答 Bit1: 一致文字列登録の応答 Bit2: 焦点位置の登録応答 Bit3: 明るさの登録応答 Bit4: 登録解除応答 Bit5: 予約済み (将来使用) Bit6: DMCC コマンド実行応答 Bit7: 一致文字列設定応答

入力データブロック

入力データブロックにはワードタイプのデータが含まれます。これは PLC からリーダに送信されるデータです。このブロックは、画像取り込みまたはデコード操作への入力として使用できるユーザ定義データで構成されます。

ワード 0	ワード 1	ワード 2 ～ N
Reserved	UserDataLength	UserData

入力データブロックのフィールドの説明

ワード	名前	説明
0	Reserved	予約済み
1	UserDataLength	UserData フィールドに実際に含まれる有効なデータのバイト数。
2 ～ N	UserData	画像取り込みまたはデコードへの入力として使用できるユーザ定義データ。

出力データブロック

出力データブロックにはワードタイプのデータが含まれます。これはリーダから PLC に送信されるデータです。このブロックは主に読み取り結果データで構成されます。

ワード 0	ワード 1	ワード 2	ワード 3	ワード 4	ワード 5 ～ N
Reserved	TriggerID	ResultID	ResultCode	ResultLength	ResultData

出力データブロックのフィールドの説明

ワード	名前	説明
0	Reserved	予約済み
1	TriggerID	トリガ識別子。発行される次のトリガの識別子。発行されたトリガと後で受信する結果データの照合のために使用されます。これと同じ値が対応する読み取りの ResultID として返されます。
2	ResultID	結果セット識別子。対応するトリガが発行されたときの TriggerID の値。トリガと対応する結果データの照合に使用されます。
3	ResultCode	読み取りの成功または失敗を示す結果セットを生成します。 Bit0: 1= 読み取り、0= 読み取り失敗 Bit1: 1= 妥当性検査合格、0= 妥当性検査失敗 Bit2: 1= 検証合格、0= 検証失敗 Bit3: 1= 画像取り込みトリガオーバーラン Bit4: 1= 画像取り込みバッファオーバーラン Bit5 ～ 15: 予約済み (将来使用)
4	ResultDataLength	ResultData フィールドに実際に含まれる有効なデータのバイト数。
5 ～ N	ResultData	この画像取り込みまたはデコードによる結果データ。

文字列コマンドブロック

文字列コマンドブロックにはワードタイプのデータが含まれます。これは PLC からリーダに送信されるデータです。このブロックは、文字列ベースのコマンド (DMCC) をリーダに送信するために使用します。

① 注：リーダのトリガと同時に、リーダ設定を変更する文字列コマンドを送信しないでください。画像取り込みまたはデコードの実行中に設定を変更すると、予期しない結果が生じる可能性があります。

ワード 0	ワード 1 ～ N
Length	String Command

文字列コマンドブロックのフィールドの説明

ワード	名前	説明
0	Length	StringCommand フィールドに含まれる有効なデータのバイト数。
1 ～ N	String Command	実行するコマンドを含む ASCII テキスト文字列。NULL 終端は不要です。

文字列コマンド結果ブロック

文字列コマンド結果ブロックにはワードタイプのデータが含まれます。これはリーダーから PLC に送信されるデータです。このブロックは、文字列ベースのコマンド (DMCC) からの応答を PLC に送信するために使用します。

ワード 0	ワード 1	ワード 2 ～ N
ResultCode	Length	String Command Result

文字列コマンド結果ブロックのフィールドの説明

ワード	名前	説明
0	Result Code	コマンドの成功または失敗を示すコード値。具体的な値については、コマンドリファレンスを参照してください (Windows の [スタート] メニューまたは DataMan Setup Tool の [ヘルプ] メニューからアクセスできます)。
1	Length	StringCommand フィールドに含まれる有効なデータのバイト数。
2 ～ N	String Command Result	実行するコマンドを含む ASCII テキスト文字列。NULL 終端は不要です。

操作

SLMP プロトコルはコマンド/応答ベースのプロトコルです。すべての通信は DataMan リーダから始まります。リーダーは制御ビットの変更を検出するために、読み取り要求を PLC に定期的に送信する必要があります。

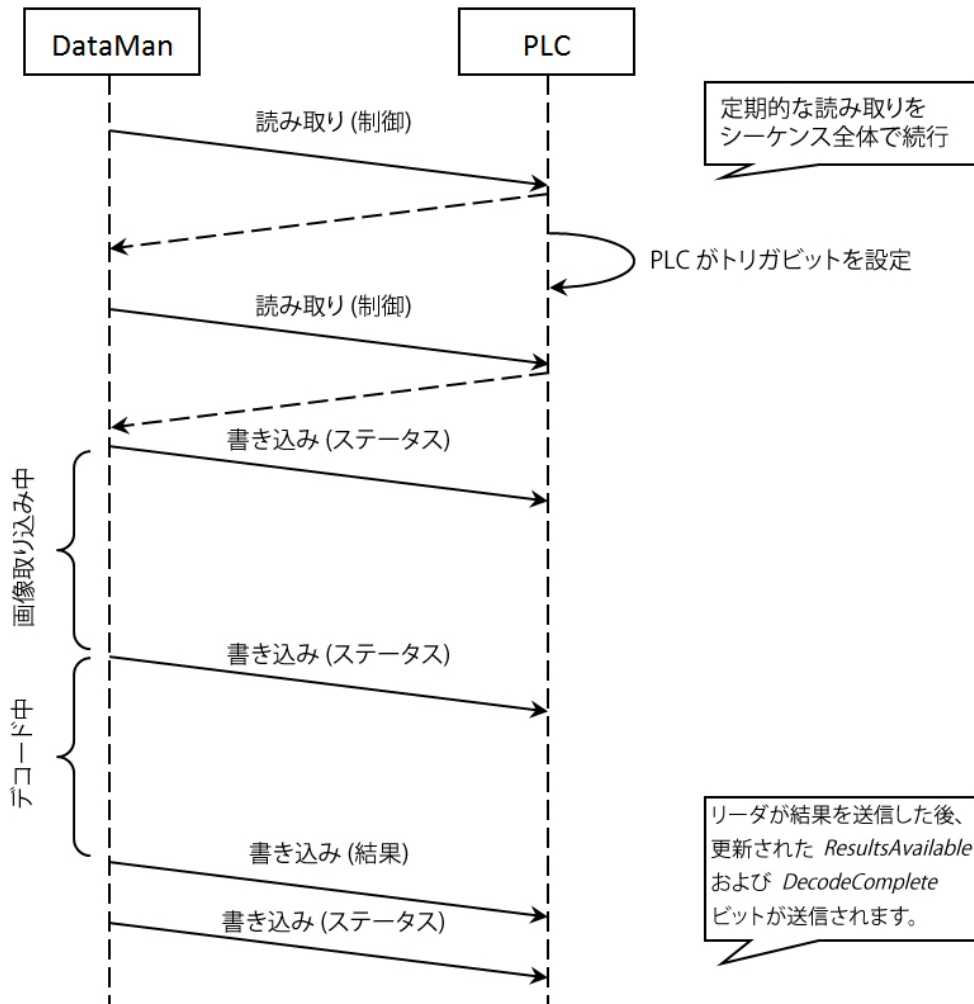
スキャン

PLC はアクションまたは制御データ転送を開始するために、コントロールブロックの特定のビットの状態を変更します。通信を開始できるのがリーダーだけなので、リーダーは定期的にスキャン (PLC からのコントロールブロックの読み取り) を実行します。このレートはユーザが定義します。

リーダーは各スキャン後にコントロールブロックのビットの状態における変更を処理します。一部の状態変更では、PLC との通信がさらに必要になります。例えば、応答ビット値の更新や、新しい文字列コマンドの読み取りなどです。これらの追加の通信はリーダーによって自動的に処理されます。その他の状態変更では、読み取りのトリガ、ソフトイベントの実行などのアクティビティが開始されます。リーダーは要求されたアクションを実行し、後で結果を報告します。

値の変更がブロック内の 1 つのフィールドのみでも、転送 (読み取りまたは書き込み) ではインタフェースブロック全体が送信されます。プロトコル実装では、できるだけ多くの値の変化を論理的に 1 回の転送にまとめることで、ネットワーク使用量が最小化されます。

一般的なシーケンス図



ハンドシェーク

いくつかのアクション (トリガ、結果の転送、ソフトイベントや文字列コマンドの実行など) がリーダと PLC の間での論理ハンドシェークを通じて行われます。これは、トランザクションの両側が相手方の操作の状態を認識するためです。ネットワーク伝送の遅延によって、転送データおよび信号には有限の時間遅延が生じます。このハンドシェークをしないと、トランザクションの一方が他方の信号状態変化を検出できない可能性があります。開始信号と対応する応答信号の両方がある操作では、この基本的なハンドシェーク手続きを使用します。

手順では 4 通りのハンドシェークを行います。

1. 信号アサート
2. 信号への応答
3. 信号アサート解除
4. アサート解除への応答

要求側デバイスは、アクションを要求する信号をアサートします (ビットを 0 から 1 にセット)。ターゲットデバイスが信号を検出し、要求された操作を完了すると、対応する応答をアサートします (ビットを 0 から 1 にセット)。要求側デバイスが応答を検出すると、元の信号をアサート解除します (1 から 0)。最後に、ターゲットデバイスが元の信号がアサート解除されたことを検出し、応答をアサート解

除します (ビットを 0 から 1)。正常に機能させるには、両側がアサートとアサート解除の完全なサイクル (0 から 1、1 から 0) を認識する必要があります。サイクルが完了するまで、要求側デバイスでは後続の要求を開始しません。

画像取り込みシーケンス

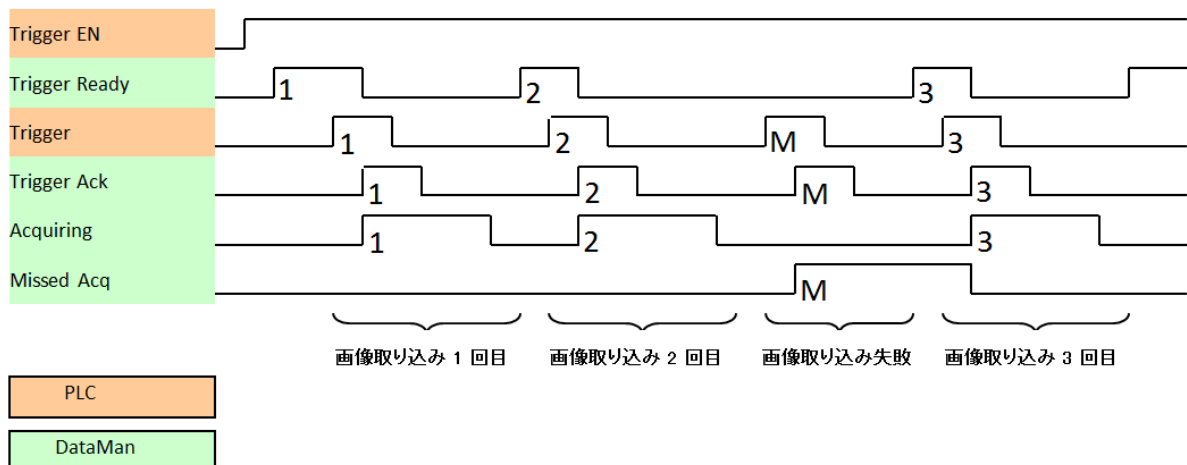
DataMan での画像取り込みをトリガする方法はいくつかあります。まず、SLMP プロトコルで「Trigger」ビットをセットするか、トリガ文字列コマンドを発行する方法です。また、DMCC コマンド (Telnet) や IO 配線によってトリガ信号を与える方法もあります。ここでは、「Trigger」ビットを使用する方法について説明します。

起動時の TriggerEnable は False です。SLMP プロトコルの「Trigger」ビットを介してトリガを有効にするには、これを True にセットする必要があります。デバイスでトリガ受け入れ準備ができると、リーダは TriggerReady ビットを True にセットします。

TriggerReady ビットが True のとき、リーダは「Trigger」ビットの 0 から 1 への変化を検出するたびに読み取りを開始します。同じ状態値が TriggerAck ビットで認識されるまで、「Trigger」ビットは新しい状態で保持される必要があります (これはリーダがトリガを認識するために必要なハンドシェークです)。

画像取り込み中、TriggerReady ビットはクリアされ、「Acquiring」ビットが True にセットされます。画像取り込みが完了すると、「Acquiring」ビットがクリアされます。デバイスで次の画像取り込みの準備ができると、TriggerReady ビットが再び True にセットされます。

結果のバッファ処理が有効になっている場合、画像取り込み操作とデコード操作のオーバーラップが可能です。デコード処理中に画像取り込みが完了すると、TriggerReady は high にセットされます。これを利用して、全体のトリガレートを高速にすることができます。結果のバッファ処理が有効になっていない場合、画像取り込み操作とデコード操作の両方が完了するまで、TriggerReady ビットは low のままです。



トリガメカニズムを強制的にリセットするには、TriggerReady が False にセットされるまで TriggerEnable を False にセットします。その後、TriggerEnable を True にセットして画像取り込みを再び有効にできます。

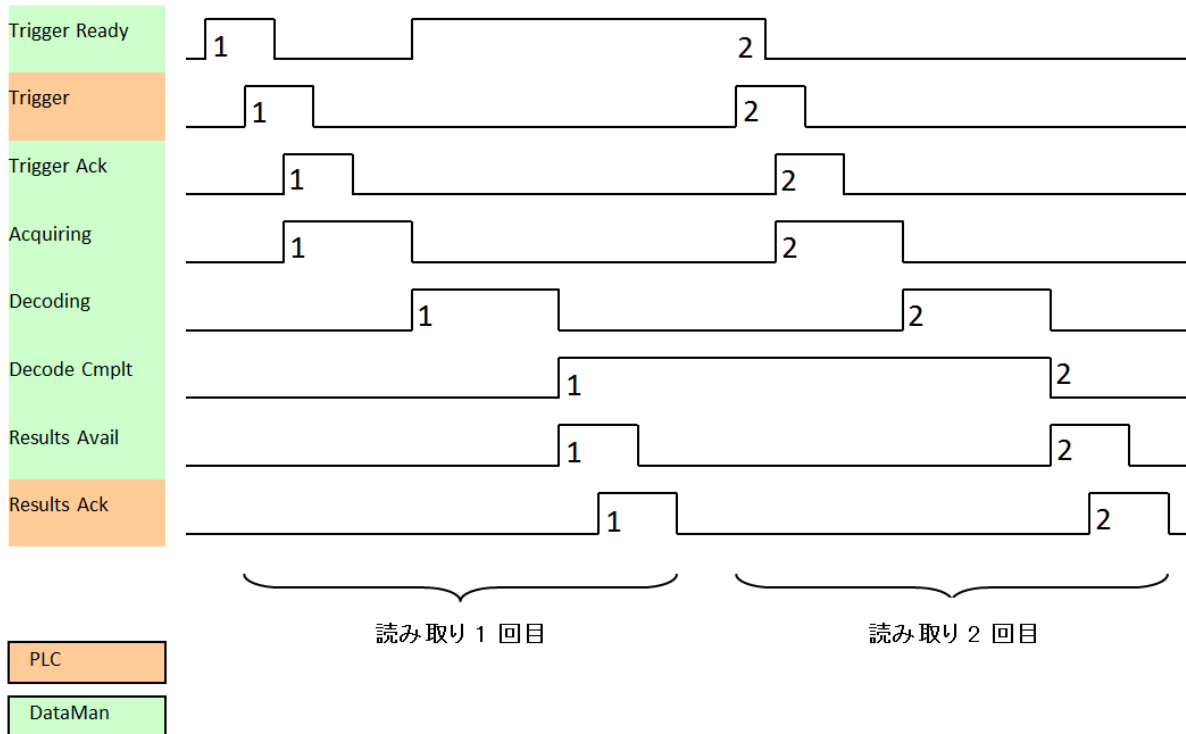
特殊なケースとして、読み取り操作が完了する前に「Trigger」信号がクリアされ、画像取り込みがキャンセルされることがあります。この場合、視野にコードがなくても自動読み取りモードおよびマニュアルモードでの読み取りがキャンセルできてしまいます。読み取りが誤ってキャンセルされないよう、TriggerAck と ResultsAvailable の両方が True になる (または DecodeComplete の状態が反転する) まで、PLC でトリガ信号が True のまま保持されるようにするとよいでしょう。

デコードまたは結果シーケンス

画像が取り込まれるとデコードが実施されます。デコード中は「Decoding」ビットがセットされます。デコード操作が完了すると、「Decoding」ビットがクリアされます。ResultsBufferEnable によって、リーダがデコード結果をどのように処理するかが決まります。

ResultsBufferEnable が False にセットされている場合、読み取り結果は直ちに出力データブロックに置かれ、ResultsAvailable が True にセットされて、DecodeComplete が反転します。

ResultsBufferEnable が True にセットされている場合、新しい結果はバッファのキューに入れられ、DecodeComplete が反転します。前の読み取り結果は PLC からの応答があるまで出力データブロックに残ります。応答ハンドシェークの後、キューにまだ結果が残っている場合は、次の結果セットが出力データブロックに置かれ、ResultsAvailable が True にセットされます。



結果のバッファ処理

読み取り結果のキューを有効にするオプションがあります。有効にした場合、PLC が読み取れるようになるまで有限数の結果データセットをキューに入れることができます。これは、PLC のパフォーマンスが一時的に低下した場合にデータフローを円滑化するのに役立ちます。

また、結果のバッファ処理が有効になっている場合、画像取り込み操作とデコード操作のオーバーラップが可能です。アプリケーションによっては、これを利用して全体のトリガレートを高速にすることができます。詳細については、前の「[画像取り込みシーケンス](#)」の説明を参照してください。

一般的に、結果の送信よりも読み取りの方が高速である場合、バッファ処理が有効であるかないかの主な違いによって、どの結果が破棄されるかが決まります。バッファ処理が有効でない場合は、最新の結果が保持され、それより前の結果 (PLC による読み取りが遅かった) が失われます。最新の結果によって前の結果が上書きされます。バッファ処理が有効になっている (かつキューがいっぱいになっている) 場合、結果キューに空きが生じるまで最新の結果が破棄されます。

注： キューがオーバーフローし、バッファ処理が無効になっている場合、TriggerID 値と ResultID 値の差が 1 より大きくなります。この差は、キューがいっぱいであるためにキューに入れることができなかった読み取りの数を表します (失われた読み取りの数 = TriggerID - ResultID - 1)。次の読み取りの後、ResultID 値は通常の動作値である TriggerID - 1 に戻ります。

SoftEvents

SoftEvents は「仮想」入力として機能します。SoftEvent ビットの値が 0 から 1 に変化すると、そのイベントに関連付けられているアクションが実行されます。アクションが完了すると、対応する SoftEventAck ビットが 0 から 1 に変化して完了を示します。

SoftEvent と SoftEventAck は論理ハンドシェークを形成します。SoftEventAck が 1 に変化した後、元の SoftEvent が 0 に戻ります。その場合、SoftEventAck は自動的に 0 に戻ります。

警告： リーダのトリガと同時に、リーダー設定を変更するソフトイベントを実行しないでください。画像取り込みまたはデコードの実行中に設定を変更すると、予期しない結果が生じる可能性があります。

① 注：“ExecuteDMCC” および “SetMatchString” ソフトイベントアクションには、ユーザ提供データが必要です。ソフトイベントを呼び出す前に、このデータを入力データブロックの UserData および UserDataLength 領域に書き込む必要があります。これらのソフトイベントはいずれも UserData に依存しているため、一度に呼び出すことができるのは 1 つのみです。

文字列コマンド

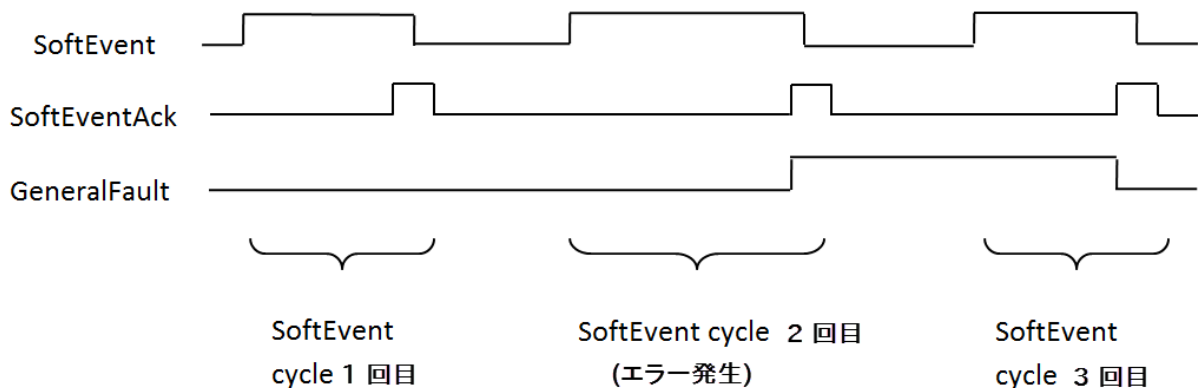
DataMan SLMP プロトコル実装には、文字列コマンド機能が含まれています。この機能により、SLMP プロトコル接続で文字列ベースの DMCC コマンドを実行できます。DMCC コマンドは文字列コマンドブロックを介してリーダに送信されます。DMCC コマンドの結果は文字列コマンド結果ブロックを介して返されます。コマンドの開始と完了通知には、コントロールブロックとステータスブロックの信号ビットを使用します。

DMCC コマンドを実行するには、コマンド文字列を文字列コマンドブロックのデータフィールドに配置します。コマンド文字列は標準の ASCII テキストで構成されます。コマンドフォーマットはシリアル (RS-232) または Telnet 接続で使用するものとまったく同じです。文字列を NULL 文字で終端する必要はありません。代わりに、文字列の長さ (ASCII 文字の数) を文字列コマンドブロックの length フィールドに配置します。

DMCC コマンドを実行すると、結果文字列は文字列コマンド結果ブロックで返されます。元のコマンドと同様に、結果文字列もシリアルまたは Telnet で返される場合と同じフォーマットの ASCII 文字で構成されています。また、終端 NULL 文字はありません。代わりに、結果の長さがコマンド文字列結果の length フィールドで返されます。コマンド文字列結果ブロックには、数値による結果コードも含まれています。そのため、テキスト文字列を解析しなくても、コマンドの成功または失敗を判断できます。結果コードの値は DMCC のドキュメントで定義されています。

一般的なエラーインジケータ

SLMP プロトコル通信に関するエラーが発生すると、“GeneralFault” ビットが 0 から 1 に変化します。現時点で唯一サポートされているエラー条件は、ソフトイベント操作です。ソフトイベント操作が失敗すると、エラービットがセットされます。次のソフトイベント操作が成功するまで、または TriggerEnable が 0 に設定され再び 1 にセットされるまで、このビットはセットされたままです。



サンプル

DataMan Setup Tool のインストーラに、三菱 (GX Works2) ソフトウェアで作成された PLC プログラムのサンプルが含まれています。このシンプルなプログラムにより、DataMan ID リーダの機能と適切な操作を明確に確認できます。三菱 PLC のより高度な機能と効率的なプログラミング実践を使用して、同様の操作を実行することができます。ただし、そのような高度なプログラムは、デモンストラーションにはあまり役立ちません。

機能

サンプルアプリケーションでは次の操作を確認できます。

1. 読み取りのトリガ
2. 読み取り結果の取得
3. 文字列コマンド (DMCC) の実行

4. ソフトイベント操作の実行

- a. コード登録
- b. 一致文字列の登録
- c. 焦点位置の登録
- d. 明るさの登録
- e. 登録解除
- f. DMCC の実行
- g. 一致文字列の設定

「メイン」プログラムに、これらの各操作を呼び出すための PLC ラダーラングがあります。ラングの制御ビットを 0 から 1 に反転することで操作を呼び出します。関連するサブルーチンが呼び出され、操作が実行されます。操作が完了すると、サブルーチンによって制御ビットが 0 に戻されます。



読み取りのトリガ

サンプルには「Continuous Trigger」と「Single Trigger」の 2 つのトリガオプションがあります。名前のとおり、「Continuous Trigger」ビットを有効にすると、連続する読み取り操作が呼び出されます。有効にした「Continuous Trigger」コントロールビットは、無効にするまで有効なままです。「Single Trigger」コントロールビットでは、1 回の読み取り操作が呼び出されます。このコントロールビットは、読み取り完了時に自動的にクリアされます。

トリガサブルーチンは主に PLC とリーダ間のトリガハンドシェイク操作を管理します。「Trigger」コントロールビットがセットされると、PLC はリーダからの対応する TriggerAck ステータスビットを待機し、「Trigger」コントロールビットがリセットされます。[「操作」](#)の節のハンドシェイクの説明を参照してください。

トリガサブルーチンには遅延タイムが含まれています。これは操作には不要ですが、デモンストレーションのため読み取り間に人工的な調整可能な遅延を挿入するために存在します。

読み取り結果の取得

このサンプルでは、読み取りのトリガ操作と読み取り結果の取得操作を意図的に分けています。これは、PLC が読み取りトリガ元ではない状況にも対応するためです。例えば、リーダがハードウェアトリガを使用するように設定されている場合があります。その場合、必要なのは結果取得サブルーチンのみです。

トリガサブルーチンと同様に、結果取得サブルーチンは PLC とリーダ間の結果ハンドシェイク操作を管理します。ただし、結果データは内部ストレージにもコピーされます。ルーチンは ResultsAvailable ステータスビットが有効になるのを待機し、結果データを内部ストレージにコピーした後、ResultsAck ハンドシェイクを実行します。[「操作」](#)の節のハンドシェイクの説明を参照してください。

読み取り結果は、ResultCode、ResultLength、ResultData で構成されています。ResultCode 値の詳細については、[「出力データブロックのフィールドの説明」](#)の節を参照してください。ResultLength フィールドは、ResultData フィールドに実際の結果データが何バイト存在するかを示します。サブルーチンは結果を内部ストレージにコピーする前に、このバイト長をワード長に変換します。

結果取得サブルーチンは読み取り統計 (読み取り成功、読み取り無効などの数) を収集します。これは操作には不要ですが、デモンストレーションのために行われます。

文字列コマンド (DMCC) の実行

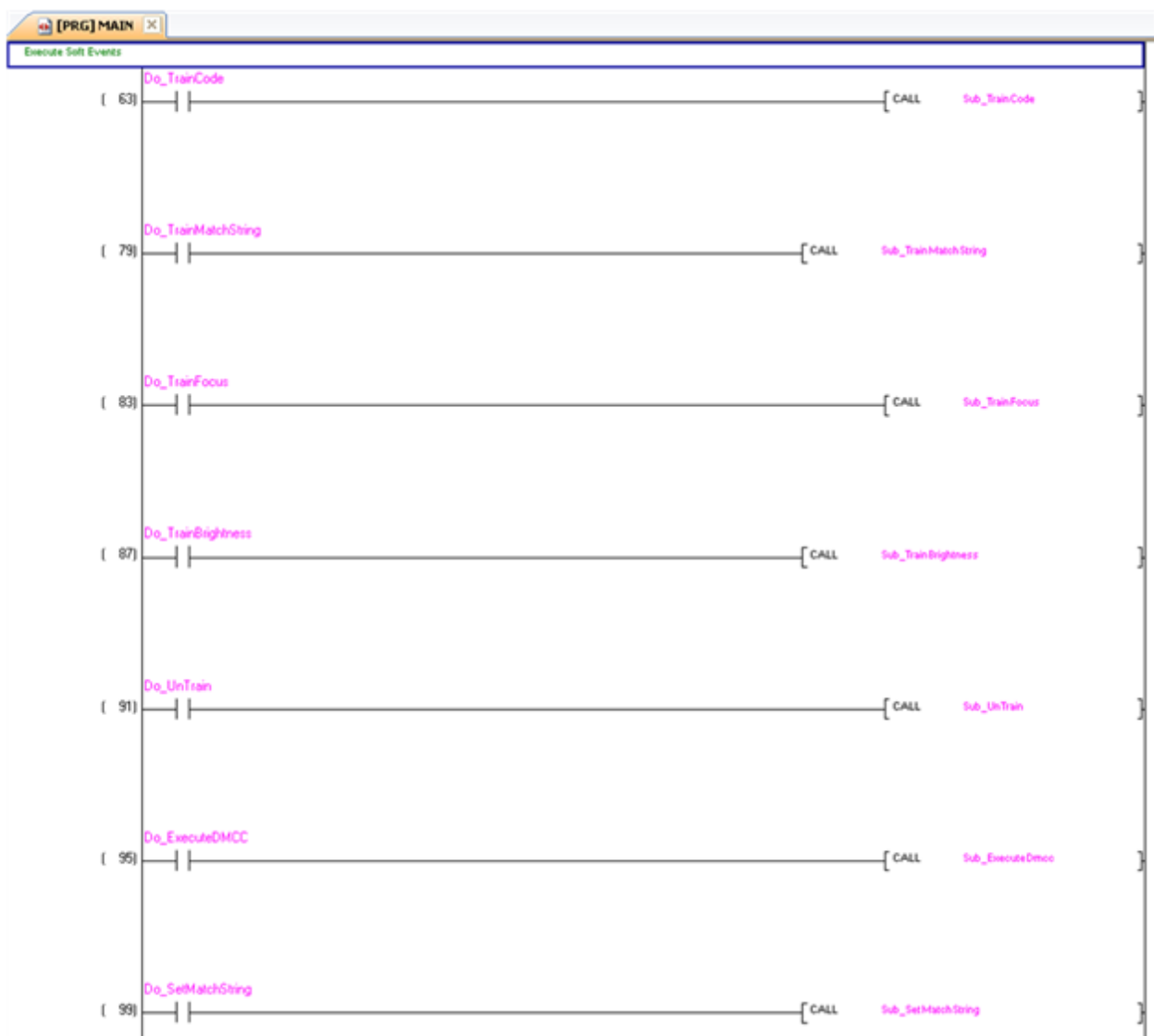
文字列コマンド機能を使用すると、PLC から DMCC コマンドを簡単に呼び出すことができます。コマンドフォーマットとコマンド結果フォーマットは、シリアルまたは Telnet での DMCC 操作に使用するものとまったく同じです。

このサブルーチンはサンプルの DMCC コマンド (||>GET CAMERA.EXPOSURE) を文字列コマンドブロックにコピーした後、PLC とリーダ間の文字列コマンドハンドシェイク操作を管理し、リーダでコマンドの呼び出しとコマンド結果の取得を行います。このメカニズムで、任意の有効な DMCC コマンドを呼び出すことができます。DataMan のコマンドリファレンスドキュメントを参照してください (Windows の [スタート] メニューまたは DataMan Setup Tool の [ヘルプ] メニューからアクセスできます)。

ソフトイベントの実行

ソフトイベントを使用して、定義済みのアクションを呼び出します。各ソフトイベントは本質的に仮想入力信号であると言えます。それぞれのソフトイベントサブルーチンは PLC とリーダ間のハンドシェイク操作を管理し、定義済みアクションを呼び出します。SoftEvent ビットが 0 から 1 に反転すると、関連付けられているアクションが呼び出されます。その後、サブルーチンはアクション完了を示す SoftEventAck ビットがリーダから送信されるのを待機します。ハンドシェイクの説明については、「[操作](#)」の節を参照してください。

注：“ExecuteDMCC” および “SetMatchString” ソフトイベントは入力データブロックを使用します。これらの 2 つのイベントのサブルーチンは関連データを入力データブロックの UserData フィールドにコピーした後、UserData サブルーチン呼び出してデータをリーダに転送します。ユーザーデータが転送されて初めて、実際のソフトイベントアクションが呼び出されます。これらのイベントのいずれかを呼び出す前に、ユーザーデータが転送される必要があります。



① 注：“TrainMatchString” ソフトイベントでは、登録メカニズムが準備されるだけです。実際の登録は、次の読み取り操作で行われます。そのため、“TrainMatchString” の後にトリガを発行する必要があります。

ModbusTCP

Modbus はアプリケーションレイヤプロトコルです。様々なタイプのバスまたはネットワークに接続されているデバイス間に、クライアント/サーバ通信を提供します。Modbus は要求/応答プロトコルであり、そのサービスは機能コードで指定します。

Modbus TCP は、TCP/IP を使用して Modbus プロトコルを提供します。システムポート 502 は、Modbus 通信用に予約済みです。標準の Ethernet ハードウェアおよびソフトウェアを使用して、I/O データおよび診断のやり取りを行います。DataMan は Modbus TCP サーバ機能のみを備えています。

デフォルトでは、DataMan では Modbus TCP プロトコルは無効になっています。プロトコルを有効にするには、Setup Tool を使用するか、DMCC の送信、もしくはパラメータコードをスキャンします。

注： ワイヤレス DataMan リーダをお持ちの場合は、「[ワイヤレス DataMan 向けの産業用プロトコル](#)」の節を参照してください。

DMCC

次のコマンドを使用して、Modbus TCP を有効または無効にすることができます。コマンドの発行は RS-232 または Telnet 接続を介して行います。

注： DataMan と通信するには、Windows が提供する Telnet クライアントの設定を変更する必要があるので、PuTTY などのサードパーティのクライアントを使用することを推奨します。

有効:

```
||>SET MODBUSTCP.ENABLED ON
||>CONFIG.SAVE
||>REBOOT
```

無効:

```
||>SET MODBUSTCP.ENABLED OFF
||>CONFIG.SAVE
||>REBOOT
```

リーダ設定コード

次のリーダ設定コードをスキャンすると、有線式リーダで Modbus TCP が有効または無効になります。

注： 変更を反映するにはデバイスを再起動する必要があります。

有効:



無効:



次のリーダ設定コードをスキャンすると、DataMan 8000 ベースステーションで Modbus TCP が有効または無効になります。

注： 変更を反映するにはデバイスを再起動する必要があります。

有効:



無効:



Setup Tool

Modbus TCP を有効にするには、[産業用プロトコル] ペインの [Modbus TCP] タブで **[有効]** をオンにします。リーダから切断する前に、[設定の保存] を選択して新しい選択内容を保存してください。

① 注： 新しい設定を反映するにはリーダを再起動する必要があります。

Modbus TCP ハンドラ

DataMan の Modbus TCP は、サーバタイプのデバイスとして実装されます。すべての通信は、読み取りおよび書き込み要求の形式で PLC によって開始されます。PLC は、読み取りおよび書き込み要求をアクティブに送信するクライアントとして動作します。

はじめに

デフォルトでは、Modbus TCP は DataMan リーダで有効になっていません。このプロトコルを有効にして、PLC と正しく情報をやり取りするようにプロトコル設定パラメータを設定する必要があります。プロトコル設定は DataMan Setup Tool で実行します。

1. Windows の [スタート] メニューから DataMan Setup Tool を起動します。
2. [通信設定] で [産業用プロトコル] をクリックします。
3. [Modbus TCP] タブを選択します。

DM262-4B5732 x

産業用プロトコル

EtherNet/IP™

PROFINET

SLMP プロトコル

Modbus TCP

☒ 有効

ポート

502

最大接続

3

待機状態のタイムアウト

120

☐ 文字列のバイトスワップ☐ 保持レジスタ限定モード

名前	アドレススペース	オフセット	数量	説明
コントロール	コイル	0	32	画像制御ブロックの開始アドレスです。
ステータス	ディスクリート入力	0	32	画像ステータスブロックの開始アドレスです。
PLC 入力	保持レジスタ	2000	2005	ユーザーデータブロックの開始アドレスです。
PLC 出力	入力レジスタ	2000	2005	検査結果ブロックの開始アドレスです。
コマンド	保持レジスタ	1000	1000	コマンドストリングの開始アドレスです。
コマンドの実...	入力レジスタ	1000	1000	コマンド実行結果データの開始アドレスです。

ステータス:

4. プロトコルを有効にして、適切な設定を行います。

Modbus TCP 設定は、ネットワーク情報の定義およびやり取りを行うデータの定義の 2 つの局面で構成されています。すべての設定パラメータは、[Modbus TCP] タブからアクセスします。

注： [システム] → [設定の保存] を選択し、Modbus TCP 設定に加えた変更を保存してください。また、新しい設定を反映するには、リーダを再起動する必要があります。

ネットワーク設定

ネットワーク設定では、DataMan リーダが PLC との接続を確立するために必要なすべての情報を定義します。ほとんどの場合にデフォルト値を使用でき、変更の必要はありません。

名前	デフォルト	範囲	説明
ポート	502	固定	このリーダで Modbus TCP がアクセス可能なポート番号。
最大接続数	3	1 ~ 6	Modbus TCP の同時接続の最大数。
待機状態のタイムアウト (秒)	120	1 ~ 3600	このタイムアウト時間経過後 Modbus TCP 接続が閉じられます。この時間内に Modbus TCP 接続で受信されたトラフィックがない場合、この接続は自動的に閉じられます。

文字列のバイトスワップ	False	True / False	文字列のバイトスワップが有効になります。True に設定した場合、文字列を形成する各レジスタ内のバイトがスワップされます。
保持レジスタ限定モード	False	True / False	保持レジスタ限定モードが有効になります。True に設定した場合、すべてのデータブロックが保持レジスタ領域にマッピングされます。

データブロック設定

データブロック設定では、DataMan リーダと PLC の間でやり取りされるデータを定義します。使用可能なデータブロックは 6 個です。それぞれのブロックにはあらかじめ定義された機能があります。

DataMan では Modbus TCP サーバ操作のみをサポートしています。標準のサーバ操作では、設定オプションは 2 つのみです。デフォルトでは、「コントロール」および「ステータス」データブロックは、ビットアドレス領域 (コイルおよびディスクリート入力) 内にあります。必要に応じて、これらのデータブロックの 1 つまたは両方を、レジスタアドレス領域 (保持レジスタおよび入力レジスタ) に存在するように再定義することができます。すべてのデータブロック設定は固定です。

標準のブロック設定

ブロック名	アドレス領域	オフセット	Schneider フォーム アドレス指定	数量
コントロール	コイルまたは保持レジスタ	0	000000 ~ 000031 400000 ~ 400001	コイルの場合: 32 保持レジスタの場合: 2
ステータス	ディスクリート入力または入力レジスタ	0	100000 ~ 100031 300000 ~ 300001	ディスクリート入力の場合: 32 入力レジスタの場合: 2
PLC 入力	保持レジスタ	2000	402000 ~ 404004	1 ~ 2005
PLC 出力	入力レジスタ	2000	302000 ~ 304004	1 ~ 2005
コマンド	保持レジスタ	1000	401000 ~ 401999	1 ~ 1000
コマンドの実行結果	入力レジスタ	1000	301000 ~ 301999	1 ~ 1000

① 注: ここでは 6 桁のゼロベースの Schneider 形式を使用します。

DataMan は代替ブロック設定もサポートしています。多くの PLC は、ModbusTCP の「保持レジスタ」アドレス領域へのアクセスのみ可能です。代替の「保持レジスタ限定」設定は、これらの PLC に対応するためにあります。この代替設定では、リーダのすべての入力および出力データが ModbusTCP の「保持レジスタ」アドレス領域にマッピングされます。この代替モードを有効にするには、次のオプションの 1 つを実行します。

- ModbusTCP 設定画面で [Holding Register Only (保持レジスタ限定)] ボックスをオンにします。
- DMCC コマンドを使用して、代替モード (保持レジスタ限定モード) のオン/オフを切り替えます。

有効:

```
||>SET MODBUSTCP-HOLDING-ONLY ON
```

無効:

```
||>SET MODBUSTCP-HOLDING-ONLY OFF
```

- リーダプログラミングコードをスキャンします。
 - ワイヤレスリーダの場合

有効:



無効:



- ・コード付きリーダの場合

有効:



無効:



「保持レジスタ限定」ブロック設定

ブロック名	アドレス領域	オフセット	Schneider フォームアドレス指定	数量
コントロール	保持レジスタ	0	400000 ～ 400001	2
ステータス	保持レジスタ	5000	405000 ～ 405001	2
PLC 入力	保持レジスタ	2000	402000 ～ 404004	1 ～ 2005
PLC 出力	保持レジスタ	7000	407000 ～ 409004	1 ～ 2005
コマンド	保持レジスタ	1000	401000 ～ 401999	1 ～ 1000
コマンドの実行結果	保持レジスタ	6000	406000 ～ 406999	1 ～ 1000

① 注：ここでは 6 桁のゼロベースの Schneider 形式を使用します。

インタフェース

この節では、Modbus TCP を介して PLC で表示される DataMan リーダのインタフェースについて説明します。インタフェースモデルは、3 つの論理ペアに分けられた 6 個のデータブロックで構成されています。

- ・コントロールとステータス
- ・入力データと出力データ
- ・文字列コマンドと文字列応答

コントロールブロック

制御ブロックにはビットタイプのデータが含まれます。このブロックは、PLC からリーダに送信される制御信号で構成されます。PLC はこれを使用してアクションを開始し、特定のデータ転送に対する応答を行います。

ビット 7	ビット 6	ビット 5	ビット 4	ビット 3	ビット 2	ビット 1	ビット 0
Reserved				ResultsAck	Buffer Results Enable	Trigger	Trigger Enable

ビット 15	ビット 14	ビット 13	ビット 12	ビット 11	ビット 10	ビット 9	ビット 8
Reserved							
ビット 23	ビット 22	ビット 21	ビット 20	ビット 19	ビット 18	ビット 17	ビット 16
Reserved						Initiate StringCmd	SetUser Data
ビット 31	ビット 30	ビット 29	ビット 28	ビット 27	ビット 26	ビット 25	ビット 24
SoftEvent 7	SoftEvent 6	SoftEvent 5	SoftEvent 4	SoftEvent 3	SoftEvent 2	SoftEvent 1	SoftEvent 0

コントロールブロックのフィールドの説明

ビット	名前	説明
0	TriggerEnable	このフィールドは「Trigger」ビットによりトリガを有効にするためにセットされます。このフィールドをクリアすると、ネットワークトリガのメカニズムが無効になります。
1	Trigger	このビットをセットすると、画像取り込みがトリガされます。画像取り込みをトリガする前に、「TriggerReady」ビットを high にセットするようにしてください。
2	BufferResults Enable	このビットがセットされている場合、各読み取り結果 (ResultID、ResultCode、ResultLength、ResultData フィールド) に応答があるまで「Output Block」に保持されます。応答があった後、次の読み取り結果セットをバッファから取得できるようになります。前の読み取りセットの応答がある前に新しい読み取り結果が届いた場合、新しい読み取りセットはリーダーのバッファのキューに入れます。リーダーのバッファに保持できる読み取り結果は最大 6 個です。応答ハンドシェイクシーケンスの説明については、「 操作 」の節を参照してください。
3	ResultsAck	最新の結果 (ResultID、ResultCode、ResultLength、ResultData フィールド) を受信したことに応答するために、PLC によってセットされます。リーダーはこのビットが 0 から 1 に遷移したことを認識すると、ResultsAvailable ビットをクリアします。これが PLC とリーダー間の論理ハンドシェイクを形成します。結果のバッファ処理が有効になっている場合、応答を受けてキューで待機中の次の結果がバッファから取り出されます。応答ハンドシェイクシーケンスの説明については、「 操作 」の節を参照してください。
4 ~ 15	Reserved	予約済み
16	SetUserData	新しい UserData が使用可能であることを示すために、PLC によってセットされます。リーダーは新しい UserData を読み取った後、転送が完了したことを示すために SetUserDataAck をセットします。これが PLC とリーダー間の論理ハンドシェイクを形成します。
17	Initiate StringCmd	新しい StringCommand が使用可能であることを示すために、PLC によってセットされます。リーダーはコマンドを処理した後、コマンドの結果が使用可能であることを示すために StringCmdAck をセットします。これが PLC とリーダー間の論理ハンドシェイクを形成します。
18 ~ 23	Reserved	予約済み
24 ~ 31	SoftEvents	ビットは仮想ディスクリート入力として機能します。ビットが 0 から 1 に遷移すると、関連付けられたアクションが実行されます。リーダーはアクションを実行した後、アクションが完了したことを示すために対応する SoftEventAck を設定します。これが PLC とリーダー間の論理ハンドシェイクを形成します。 Bit0: コード登録 Bit1: 文字列一致の登録 Bit2: 焦点位置の登録 Bit3: 明るさの登録 Bit4: 登録解除 Bit5: 予約済み (将来使用) Bit6: DMCC コマンドの実行 Bit7: 文字列一致の設定

ステータスブロック

ステータスブロックにはビットタイプのデータが含まれます。このブロックは、リーダーから PLC に送信されるステータス信号で構成されます。リーダーはこれを使用してステータスを示し、特定のデータ転送のハンドシェイクを行います。

ビット 7	ビット 6	ビット 5	ビット 4	ビット 3	ビット 2	ビット 1	ビット 0
予約済み				Missed Acq	Acquiring	Trigger Ack	Trigger Ready
ビット 15	ビット 14	ビット 13	ビット 12	ビット 11	ビット 10	ビット 9	ビット 8
General Fault	予約済み			Results Available	Results Buffer Overrun	Decode Complete Toggle	Decoding
ビット 23	ビット 22	ビット 21	ビット 20	ビット 19	ビット 18	ビット 17	ビット 16
予約済み						String Cmd Ack	SetUser DataAck
ビット 31	ビット 30	ビット 29	ビット 28	ビット 27	ビット 26	ビット 25	ビット 24
SoftEvent Ack 7	SoftEvent Ack 6	SoftEvent Ack 5	SoftEvent Ack 4	SoftEvent Ack3	SoftEvent Ack 2	SoftEvent Ack 1	SoftEvent Ack 0

ステータスブロックのフィールドの説明

ビット	名前	説明
0	TriggerReady	リーダが新規「Trigger」を受け入れ可能であることを示します。リーダは、TriggerEnable がセットされ、新規トリガの受け入れ準備ができているときにこのビットをセットします。
1	TriggerAck	「Trigger」がセット済みであることをリーダが認識していることを示します。Trigger ビットがクリアされるまで、このビットはセットされたままです。
2	Acquiring	リーダが画像取り込み中であることを示すためにセットされます。
3	MissedAcq	リーダが要求された画像取り込みトリガに失敗したことを示します。次の画像取り込みが発行されると、このビットはクリアされます。
4 ～ 7	Reserved	予約済み
8	Decoding	リーダが画像のデコード中であることを示すためにセットされます。
9	DecodeComplete Toggle	新しい結果データが使用可能であることを示します。新しい結果データが使用可能になるたびに、ビットの状態が (0 から 1 へ、または 1 から 0 へ) 反転します。
10	ResultsBuffer Overrun	PLC が以前の結果に回答していないためリーダが一連の読み取り結果を破棄したことを示すためにセットされます。次の結果データセットが正常にバッファのキューに入れられた時点でクリアされます。このビットは、結果バッファ処理が有効である場合のみ意味を持ちます。
11	ResultsAvailable	新しい結果データが使用可能であることを示すためにセットされます。追加の新しい読み取り結果が使用可能であっても、ResultsAck で応答されるまで、このビットはセットされたままです。
12 ～ 14	Reserved	予約済み
15	GeneralFault	Ethernet 通信エラーが発生したことを示すために設定されます。現時点では、ソフトイベント操作によってのみ使用されています。次のソフトイベントが成功するまで、または TriggerEnable が low に設定され再び high にセットされるまで、このビットはセットされたままです。
16	SetUserDataAck	リーダが新しい UserData を受信したことを示すためにセットされます。対応する SetUserData ビットがクリアされるまで、このビットはセットされたままです。これが PLC とリーダ間の論理ハンドシェイクを形成します。
17	String CmdAck	リーダが最新の文字列コマンドの処理を完了し、コマンド応答が使用可能であることを示すためにセットされます。対応する InitiateStringCmd ビットがクリアされるまで、このビットはセットされたままです。これが PLC とリーダ間の論理ハンドシェイクを形成します。

18 ~ 23	Reserved	予約済み
24 ~ 31	SoftEvent Ack	リーダがソフトイベントアクションを完了したことを示すためにセットされます。対応する SoftEvent ビットがクリアされるまで、このビットはセットされたままです。これが PLC とリーダ間の論理ハンドシェークを形成します。 Bit0: コード登録応答 Bit1: 一致文字列の登録応答 Bit2: 焦点位置の登録応答 Bit3: 明るさの登録応答 Bit4: 登録解除応答 Bit5: 予約済み (将来使用) Bit6: DMCC コマンド実行の応答 Bit7: 一致文字列設定の応答

入力データブロック

入力データブロックは、PLC からリーダに送信されます。このブロックは、画像取り込みまたはデコード操作への入力として使用できるユーザ定義データで構成されます。

ワード 0	ワード 1	ワード 2 ~ N
Reserved	UserDataLength	UserData

入力データブロックのフィールドの説明

ワード	名前	説明
0	Reserved	予約済み
1	UserDataLength	UserData フィールドに実際に含まれる有効なデータのバイト数。
2 ~ N	UserData	画像取り込みまたはデコードへの入力として使用できるユーザ定義データ。

出力データブロック

出力データブロックは、リーダから PLC に送信されます。このブロックは主に読み取り結果データで構成されます。

ワード 0	ワード 1	ワード 2	ワード 3	ワード 4	ワード 5 ~ n
Reserved	TriggerID	ResultID	ResultCode	ResultLength	ResultData

出力データブロックのフィールドの説明

ワード	名前	説明
0	Reserved	予約済み
1	TriggerID	トリガ識別子。発行される次のトリガの識別子。発行されたトリガと後で受信する結果データの照合のために使用されます。これと同じ値が対応する読み取りの ResultID として返されます。
2	ResultID	結果識別子。対応するトリガが発行されたときの TriggerID の値。トリガと対応する結果データの照合に使用されます。

3	ResultCode	この結果セットを生成した読み取りの成功または失敗を示します。 Bit0: 1= 読み取り、0= 読み取り失敗 Bit1: 1= 妥当性検査合格、0= 妥当性検査失敗 Bit2: 1= 検証合格、0= 検証失敗 Bit3: 1= 画像取り込みトリガオーバーラン Bit4: 1= 画像取り込みバッファオーバーラン Bit5 ~ 15: 予約済み (将来使用)
4	ResultData Length	ResultData フィールドに実際に含まれる有効なデータのバイト数。
5 ~ n	ResultData	この画像取り込みまたはデコードによる結果データ。16 ビットレジスタごとに 2 文字を含む ASCII テキストとしてフォーマットされます。終端 NULL 文字はありません。

文字列コマンドブロック

文字列コマンドブロックは、PLC からリーダに送信されます。このブロックは、文字列ベースのコマンド (DMCC) をリーダに送信するために使用します。

① 注：リーダのトリガと同時に、リーダ設定を変更する文字列コマンドを送信しないでください。画像取り込みまたはデコードの実行中に設定を変更すると、予期しない結果が生じる可能性があります。

ワード 0	ワード 1 ~ n
Length	String Command

文字列コマンドブロックのフィールドの説明

ワード	名前	説明
0	Length	StringCommand フィールドに含まれる有効なデータのバイト数。
1 ~ N	String Command	実行するコマンドを含む ASCII テキスト文字列。NULL 終端は不要です。

文字列コマンド結果ブロック

文字列コマンド結果ブロックは、リーダから PLC に送信されます。このブロックは、文字列ベースのコマンド (DMCC) からの応答を PLC に送信するために使用します。

ワード 0	ワード 1	ワード 2 ~ N
Result Code	Length	String Command Result

文字列コマンド結果ブロックのフィールドの説明

ワード	名前	説明
0	ResultCode	コマンドの成功または失敗を示すコード値。具体的な値については、コマンドリファレンスを参照してください (Windows の [スタート] メニューまたは DataMan Setup Tool の [ヘルプ] メニューからアクセスできます)。
1	Length	StringCommand フィールドに含まれる有効なデータのバイト数。
2 ~ N	String Command Result	実行するコマンドを含む ASCII テキスト文字列。NULL 終端は不要です。

操作

Modbus TCP は要求/応答ベースのプロトコルです。すべての通信は PLC から始まります。リーダーがサーバとして動作します。

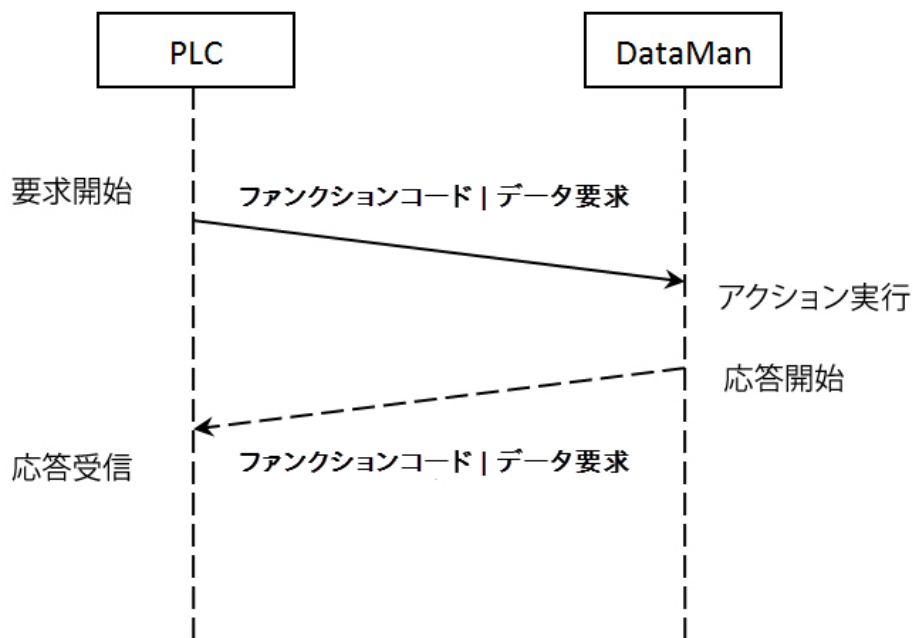
要求

PLC はアクションまたは制御データ転送を開始するために、コントロールブロックの特定のビットの状態を変更して要求をリーダーに送信します。

リーダーは各要求の後にコントロールブロックのビットの状態変更に応じて処理を行います。一部の状態変更では、PLC との通信がさらに必要になります。例えば、応答ビット値の更新や、新しい文字列コマンドの読み取りなどです。これらの追加の通信はリーダーによって自動的に処理されます。その他の状態変更では、読み取りのトリガ、ソフトイベントの実行などのアクティビティが開始されます。

リーダーは要求されたアクションを実行し、後で結果を報告します。

一般的なシーケンス図



ハンドシェーク

いくつかのアクション (トリガ、結果の転送、ソフトイベントや文字列コマンドの実行など) がリーダーと PLC の間での論理ハンドシェークを通じて行われます。これは、トランザクションの両者が相手方の操作の状態を認識するためです。ネットワーク伝送の遅延によって、転送データおよび信号には有限の時間遅延が生じます。このハンドシェークがないと、トランザクションの一方が他方の信号状態変化を検出できない可能性があります。開始信号と対応する応答信号の両方がある操作では、この基本的なハンドシェーク手続きを使用します。

手順では 4 通りのハンドシェークを行います。

1. 信号アサート
2. 信号への応答
3. 信号アサート解除
4. アサート解除への応答

要求側デバイスは、アクションを要求する信号をアサートします (ビットを 0 から 1 にセット)。ターゲットデバイスが信号を検出し、要求された操作を完了すると、対応する応答をアサートします (ビットを 0 から 1 にセット)。要求側デバイスが応答を検出すると、元の

信号をアサート解除します (1 から 0)。最後に、ターゲットデバイスが元の信号がアサート解除されたことを検出し、応答をアサート解除します (1 から 0)。正常に機能させるには、両側がアサートとアサート解除の完全なサイクル (0 から 1、1 から 0) を認識する必要があります。サイクルが完了するまで、要求側デバイスでは後続の要求を開始しません。

画像取り込みシーケンス

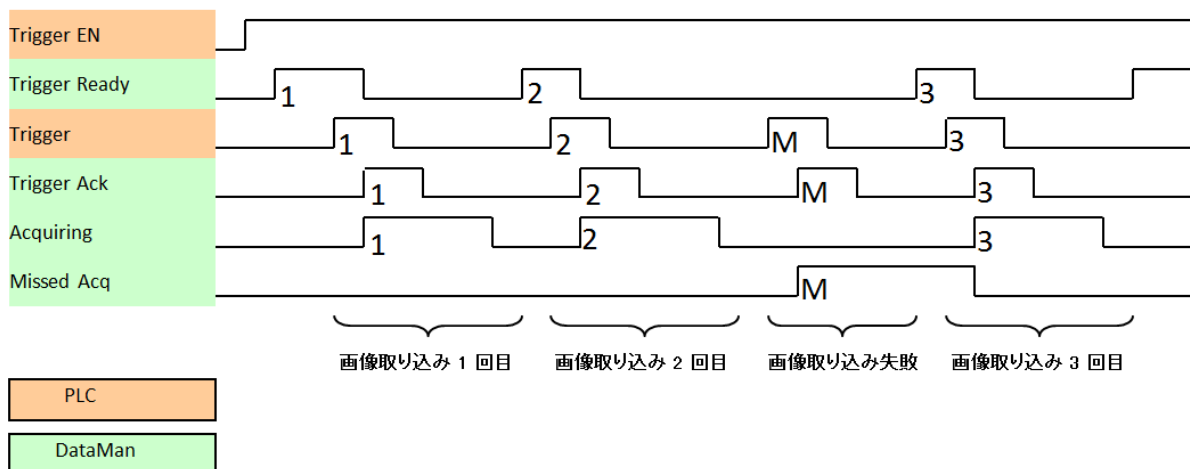
DataMan での画像取り込みをトリガする方法はいくつかあります。「Trigger」ビットをセットするか、トリガ文字列コマンドを発行します。また、DMCC コマンド (Telnet) や IO 配線によるトリガ信号を使用する方法もあります。ここでは、「Trigger」ビットを使用する方法について説明します。

起動時の TriggerEnable は False です。「Trigger」ビットをによりトリガを有効にするには、これを True にセットする必要があります。デバイスでトリガ受け入れ準備ができると、リーダは TriggerReady ビットを True にセットします。

TriggerReady ビットが True のとき、リーダは「Trigger」ビットの 0 から 1 への変化を検出するたびに読み取りを開始します。同じ状態値が TriggerAck ビットで認識されるまで、「Trigger」ビットは新しい状態で保持される必要があります (これはリーダがトリガを必ず認識するために必要なハンドシェークです)。

画像取り込み中、TriggerReady ビットはクリアされ、「Acquiring」ビットが True にセットされます。画像取り込みが完了すると、「Acquiring」ビットがクリアされます。デバイスで次の画像取り込みの準備ができると、TriggerReady ビットが再び True にセットされます。

結果のバッファ処理が有効になっている場合、画像取り込み操作とデコード操作のオーバーラップが可能です。デコード処理中に画像取り込みが完了すると、TriggerReady は high にセットされます。これを利用して、全体のトリガレートを高速にすることができます。結果のバッファ処理が有効になっていない場合、画像取り込み操作とデコード操作の両方が完了するまで、TriggerReady ビットは low のままです。



トリガメカニズムを強制的にリセットするには、TriggerReady が False にセットされるまで TriggerEnable を False にセットします。その後、TriggerEnable を True にセットして画像取り込みを再び有効にできます。

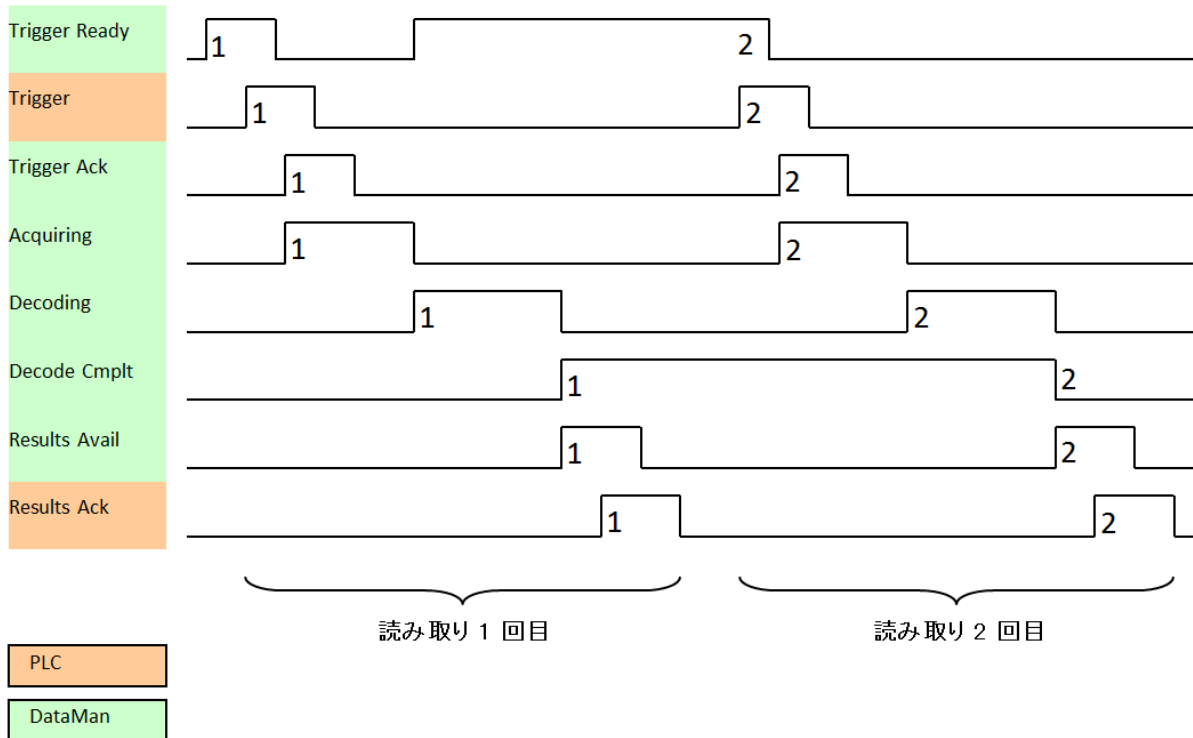
特殊なケースとして、読み取り操作が完了する前に「Trigger」信号がクリアされ、画像取り込みがキャンセルされることがあります。この場合、視野にコードがなくても自動読み取りモードおよびマニュアルモードでの読み取りがキャンセルできてしまいます。読み取りが誤ってキャンセルされないよう、TriggerAck と ResultsAvailable の両方が True になる (または DecodeComplete の状態が反転する) まで、PLC でトリガ信号が True のまま保持されるようにするとよいでしょう。

デコードまたは結果シーケンス

画像が取り込まれるとデコードが実施されます。デコード中は「Decoding」ビットがセットされます。デコード操作が完了すると、「Decoding」ビットがクリアされます。ResultsBufferEnable によって、リーダがデコード結果をどのように処理するかが決まります。

ResultsBufferEnable が False にセットされている場合、読み取り結果は直ちに出力データブロックに置かれ、ResultsAvailable が True にセットされて、DecodeComplete が反転します。

ResultsBufferEnable が True にセットされている場合、新しい結果はバッファのキューに入れられ、DecodeComplete が反転します。前の読み取り結果は PLC からの応答があるまで出力データブロックに残ります。応答ハンドシェイクの後、キューにまだ結果が残っている場合は、次の結果セットが出力データブロックに置かれ、ResultsAvailable が True にセットされます。



結果のバッファ処理

読み取り結果のキューを有効にするオプションがあります。有効にした場合、PLC が読み取れるようになるまで有限数の結果データセットをキューに入れることができます。これは、PLC のパフォーマンスが一時的に低下した場合にデータフローを円滑化するのに役立ちます。

また、結果のバッファ処理が有効になっている場合、画像取り込み操作とデコード操作のオーバーラップが可能です。アプリケーションによっては、これを利用して全体のトリガレートを高速にすることができます。詳細については、「画像取り込みシーケンス」の説明を参照してください。

一般的に、結果が PLC に転送されるよりも読み取りの方が高速である場合、一部のデータが失われます。バッファ処理が有効であるかないかの主な違いによって、どの結果が破棄されるかが決まります。バッファ処理が有効でない場合は、最新の結果が保持され、それより前の結果は (PLC の速度が不十分で読み取れなかったために) 失われます。最新の結果によって前の結果が上書きされます。バッファ処理が有効になっている (かつキューがいっぱいになっている) 場合、結果キューに空きが生じるまで最新の結果は破棄されます。

注： キューがオーバーフローし、バッファ処理が無効になっている場合、TriggerID 値と ResultID 値の差が 1 より大きくなります。この差は、キューがいっぱいであるためにキューに入れることができなかった読み取りの数を表します (失われた読み取りの数 = TriggerID - ResultID - 1)。次の読み取りの後、ResultID 値は通常の動作値である TriggerID - 1 に戻ります。

SoftEvents

SoftEvents は「仮想」入力として機能します。SoftEvent ビットの値が 0 から 1 に変化すると、そのイベントに関連付けられているアクションが実行されます。アクションが完了すると、対応する SoftEventAck ビットが 0 から 1 に変化して完了を示します。

SoftEvent と SoftEventAck は論理ハンドシェイクを形成します。SoftEventAck が 1 に変化した後、元の SoftEvent が 0 に戻ります。その場合、SoftEventAck は自動的に 0 に戻ります。

① 注：リーダのトリガと同時に、リーダ設定を変更するソフトイベントを実行しないでください。画像取り込みまたはデコードの実行中に設定を変更すると、予期しない結果が生じる可能性があります。

“ExecuteDMCC” および “SetMatchString” ソフトイベントアクションには、ユーザ提供データが必要です。ソフトイベントを呼び出す前に、このデータを入力データブロックの UserData および UserDataLength 領域に書き込む必要があります。これらのソフトイベントはいずれも UserData に依存しているため、一度に呼び出すことができるのは 1 つのみです。

文字列コマンド

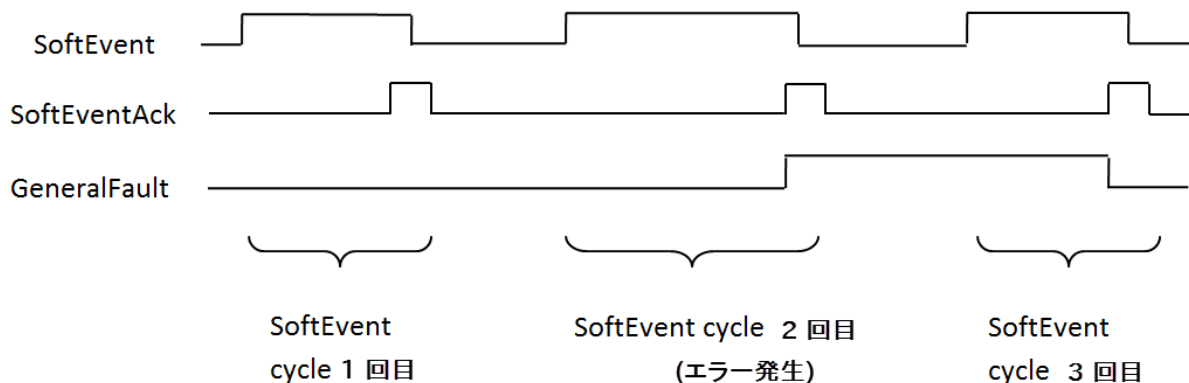
DataMan の Modbus TCP プロトコル実装には、文字列コマンド機能が含まれています。この機能で、文字列ベースの DMCC コマンドを実行することができます。DMCC コマンドは文字列コマンドブロックを介してリーダに送信されます。DMCC コマンドの結果は文字列コマンド結果ブロックを介して返されます。コマンドの開始と完了通知には、コントロールブロックとステータスブロックの信号ビットを使用します。

DMCC コマンドを実行するには、コマンド文字列を文字列コマンドブロックのデータフィールドに配置します。コマンド文字列は標準の ASCII テキストで構成されます。コマンドフォーマットはシリアル (RS-232) または Telnet 接続で使用するものとまったく同じです。文字列を NULL 文字で終端する必要はありません。代わりに、文字列の長さ (ASCII 文字の数) を文字列コマンドブロックの length フィールドに配置します。

DMCC コマンドを実行すると、結果文字列が文字列コマンド結果ブロックで返されます。元のコマンドと同様に、結果文字列もシリアルまたは Telnet で返される場合と同じフォーマットの ASCII 文字で構成されています。また、終端 NULL 文字はありません。代わりに、結果の長さがコマンド文字列結果の length フィールドで返されます。コマンド文字列結果ブロックには、数値による結果コードも含まれています。そのため、テキスト文字列を解析しなくても、コマンドの成功または失敗を判断できます。結果コードの値は DMCC のドキュメントで定義されています。

一般的なエラーインジケータ

通信関連の障害が発生すると、“GeneralFault” ビットが 0 から 1 に変化します。現時点で唯一サポートされているエラー条件は、ソフトイベント操作です。ソフトイベント操作が失敗すると、エラービットがセットされます。次のソフトイベント操作が成功するまで、または TriggerEnable が 0 にセットされ再び 1 にセットされるまで、このビットはセットされたままです。



サンプル

DataMan Setup Tool インストーラに、CoDeSys v2.3 ソフトウェアで作成された PLC プログラムのサンプルが 2 つ含まれています。これらのサンプルは、Wago 750-841 PLC 上で設計およびテストが行われています。これらのシンプルなプログラムにより、DataMan ID リーダの機能と適切な操作を明確に確認できます。Wago PLC より高度な機能と効率的なプログラミング実践を使用して、同様の操作を実行することができます。

ただし、そのような高度なプログラムは、デモンストレーションにはあまり役立ちません。サンプルでは、DataMan リーダとの通信に使用される技術におけるさまざまなアプローチを紹介しています。

① 注：すべてのサンプルは、「コントロール」データブロックがコイル領域にマッピングされ、「ステータス」データブロックがディスクリット入力領域にマッピングされる場合にのみ機能するように設計されています。

ApplicationLayer のサンプル

このサンプルは、DataMan リーダと PLC との間に一般のデータ転送を実現しています。「コントロール」、「ステータス」、および「出力領域」のメモリ領域は、PLC でクローン化され、必要に応じて、または周期的に同期されます。各データ領域は、「ETHERNETMODBUSMASTER_TCP」の自身のインスタンスと同期されます。これによって 3 つの TCP 接続が同時に開かれます。このサンプルを動作させるために、DataMan リーダの Modbus TCP に関する設定で [最大接続] が少なくとも 3 に設定されていることを確認してください。

機能

サンプルアプリケーションでは次の操作を確認できます。

1. 32 ビット「コントロール」レジスタデータを PLC からリーダーに転送する。
2. 32 ビット「ステータス」レジスタデータをリーダーから PLC に転送する。
3. 「出力データ」をリーダーから PLC に転送する。

すべてのアクションは、リーダーへの接続が存在する場合に開始されます。

「コントロール」レジスタデータの転送

ローカルの PLC データに変更があった場合、すべてのデータが転送されます。ローカルの PLC データは、視覚的に操作できます。

① 注：同期はリーダーから PLC には実施されないで、接続の構築後や、別の Modbus TCP クライアントがコントロールレジスタを同時に操作する場合に、ローカルの PLC データが正しくない可能性があります。この時間枠中に「コントロール」レジスタを操作しなかった場合、タイムアウト設定によって接続が切断される可能性があります。

ステータスレジスタデータの転送

すべてのデータは周期的に転送されます。ポーリング間隔は視覚的に指定することができます。

出力データの転送

すべてのデータは周期的に転送されます。ポーリング間隔は視覚的に指定することができます。

DataManControl のサンプル

このサンプルでは、次のサブセクションで名前が付けられている機能の 1 つを実行する手順を、順を追って示します。この時間的な順序の概要を説明するため、プログラミング言語として「シーケンシャルファンクションチャート」が選択されています。

機能

サンプルアプリケーションでは次の操作を確認できます。

1. 読み取りのトリガ
2. 読み取り結果の取得
3. 文字列コマンド (DMCC) の実行
4. ソフトイベント操作の実行
 - a. コード登録
 - b. 一致文字列の登録
 - c. 焦点位置の登録
 - d. 明るさの登録
 - e. 登録解除
 - f. DMCC の実行
 - g. 一致文字列の設定

Main プログラムに、これらの各操作を呼び出すための変数があります。これらの操作を呼び出すには、コントロールブールを直接切り替えるか、または視覚的に (赤 =0、緑 =1)、0 から 1 に反転します。関連するサブルーチンが呼び出され、操作が実行されます。操作 4 が完了すると、サブルーチンによってコントロールビットが 0 に戻されます。

読み取りのトリガ

このサンプルでは「Continuous Trigger」を説明します。名前のとおり、「xTrigger」ビットを有効にすると、連続する読み取り操作が呼び出されます。有効にした「xTrigger」コントロールビットは、無効にするまで有効なままです。

トリガサブルーチンは主に PLC とリーダ間のトリガハンドシェイク操作を管理します。「ResultAck」および「Trigger」ビットがリセットされ、「TriggerEnable」ビットがセットされて、PLC はリーダからの対応する TriggerReady ステータスビットを待機し、トリガビットがセットされます。「[操作](#)」の節のハンドシェイクの説明を参照してください。

読み取り結果の取得

このサンプルでは、読み取りのトリガ操作と読み取り結果の取得操作を意図的に分けています。これは、PLC が読み取りトリガ元ではない状況にも対応するためです。例えば、リーダがハードウェアトリガを使用するように設定されている場合があります。その場合、必要なのは結果取得サブルーチンのみです。

トリガサブルーチンと同様に、結果取得サブルーチンは PLC とリーダ間の結果ハンドシェイク操作を管理します。ルーチンは ResultsAvailable ステータスビットが有効になるのを待ち、結果データを内部ストレージにコピーした後、ResultsAck ハンドシェイクを実行します。「[操作](#)」の節のハンドシェイクの説明を参照してください。

読み取り結果は、ResultCode、ResultLength、ResultData で構成されています。ResultCode 値の詳細については、「[出力データブロックのフィールドの説明](#)」の節を参照してください。ResultLength フィールドは、ResultData フィールドに実際の結果データが何バイト存在するかを示します。サブルーチンは結果を内部ストレージにコピーする前に、このバイト長をワード長に変換します。

文字列コマンド (DMCC) の実行

文字列コマンド機能を使用すると、PLC から DMCC コマンドを簡単に発行することができます。コマンドフォーマットとコマンド結果フォーマットは、シリアルまたは Telnet での DMCC 操作に使用するものとまったく同じです。

このサブルーチンはサンプルの DMCC コマンド (||>GET DEVICE.TYPE) を文字列コマンドブロックにコピーした後、PLC とリーダ間の文字列コマンドハンドシェイク操作を管理し、リーダでコマンドの呼び出しとコマンド結果の取得を行います。このメカニズムで、任意の有効な DMCC コマンドを呼び出すことができます。DataMan のコマンドリファレンスドキュメントを参照してください (Windows の [スタート] メニューまたは Setup Tool の [ヘルプ] メニューからアクセスできます)。

ソフトイベントの実行

ソフトイベントを使用して、定義済みのアクションを呼び出します。各ソフトイベントは本質的に仮想入力信号であると言えます。それぞれのソフトイベントサブルーチンは PLC とリーダ間のハンドシェイク操作を管理し、定義済みアクションを呼び出します。SoftEvent ビットが 0 から 1 に反転すると、関連付けられているアクションが呼び出されます。その後、サブルーチンはアクション完了を示す SoftEventAck ビットがリーダから送信されるのを待機します。ハンドシェイクの説明については、「[操作](#)」の節を参照してください。

注：“ExecuteDMCC” および “SetMatchString” ソフトイベントは入力データブロックを使用します。これらの 2 つのイベントのサブルーチンは関連データを入力データブロックの UserData フィールドにコピーした後、UserData サブルーチン呼び出してデータをリーダに転送します。ユーザデータが転送されて初めて、実際のソフトイベントアクションが呼び出されます。これらのイベントのいずれかを呼び出す前に、ユーザデータが転送される必要があります。

注：“TrainMatchString” ソフトイベントでは、登録メカニズムが準備されるだけです。実際の登録は、次の読み取り操作で行われます。そのため、“TrainMatchString” の後にトリガを発行する必要があります。

PROFINET

PROFINET は、産業用オートメーションアプリケーションで使用されるアプリケーションレベルのプロトコルです。このプロトコルは、標準の Ethernet ハードウェアおよびソフトウェアを使用して、I/O データ、アラーム、および診断のやり取りを行います。

DataMan は PROFINET I/O をサポートしています。これは、PROFINET 通信規格に含まれている 2 つの「ビュー」の 1 つです。PROFINET I/O は周期的にデータ転送を実行して、Ethernet 経由でプログラマブルロジックコントローラ (PLC) とデータをやり取りします。この規格の 2 つめの「ビュー」である PROFINET CBA (コンポーネントベースオートメーション) はサポートされていません。

DataMan PROFINET 通信モデルを Cognex In-Sight ファミリに厳密に適合させるため、慎重に取り組んできました。In-Sight の使用経験のあるお客様は、慣れ親しんだ快適な方法で DataMan を使用して作業できることがお分かりいただけます。

デフォルトでは、DataMan では PROFINET プロトコルは無効になっています。このプロトコルを有効にするには、DMCC を送信するか、パラメータコードをスキャンするか、または DataMan Setup Tool を使用します。

注： ワイヤレス DataMan リーダをお持ちの場合は、「[ワイヤレス DataMan 向けの産業用プロトコル](#)」の節を参照してください。

DMCC

次のコマンドを使用して、PROFINET を有効または無効にすることができます。コマンドの発行は RS-232 または Telnet 接続を介して行います。

注： DataMan と通信するには、Windows が提供する Telnet クライアントの設定を変更する必要があるため、PuTTY などのサードパーティのクライアントを使用することを推奨します。

有効:

```
||>SET PROFINET.ENABLED ON
||>CONFIG.SAVE
||>REBOOT
```

無効:

```
||>SET PROFINET.ENABLED OFF
||>CONFIG.SAVE
||>REBOOT
```

リーダ設定コード

次のリーダ設定コードをスキャンすると、有線式リーダで PROFINET が有効または無効になります。

注： 変更を反映するにはデバイスを再起動する必要があります。

有効:



無効:



次のリーダ設定コードをスキャンすると、DataMan 8000 ベースステーションで PROFINET が有効または無効になります。

注： 変更を反映するにはデバイスを再起動する必要があります。

有効:

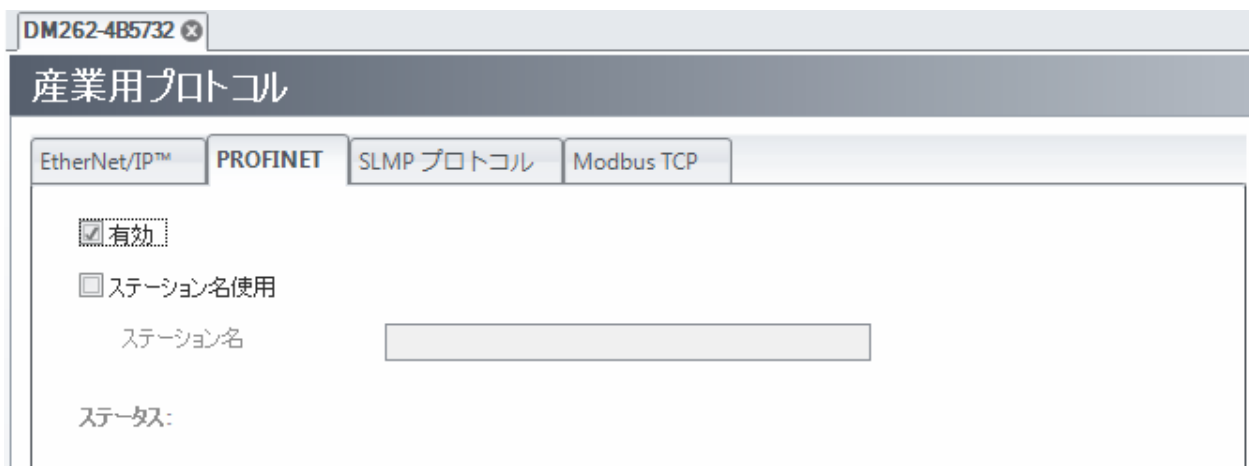


無効:



Setup Tool

PROFINET プロトコルを有効にするには、[産業用プロトコル] ペインの [PROFINET] タブで **[有効]** にチェックを入れます。



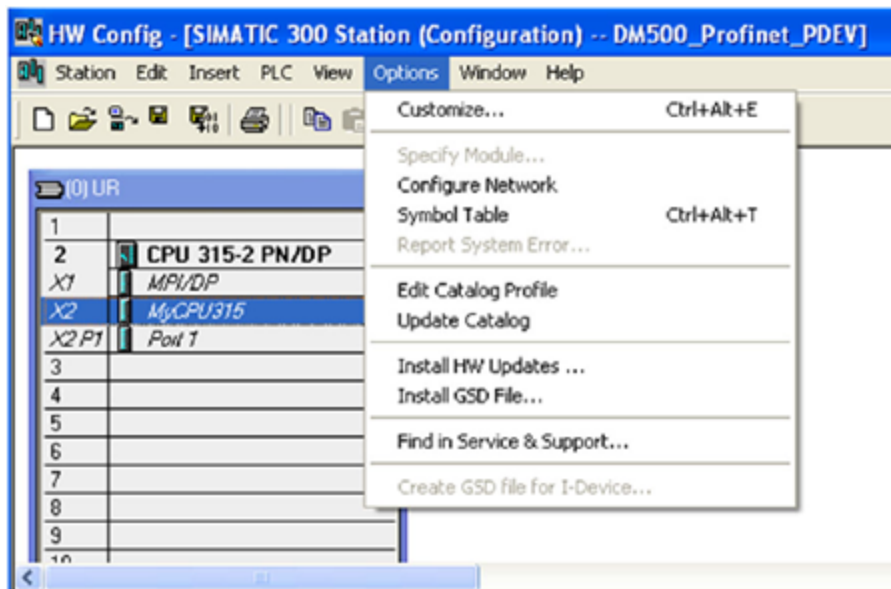
はじめに

PROFINET を使用する準備として、次の主な手順を実行します。

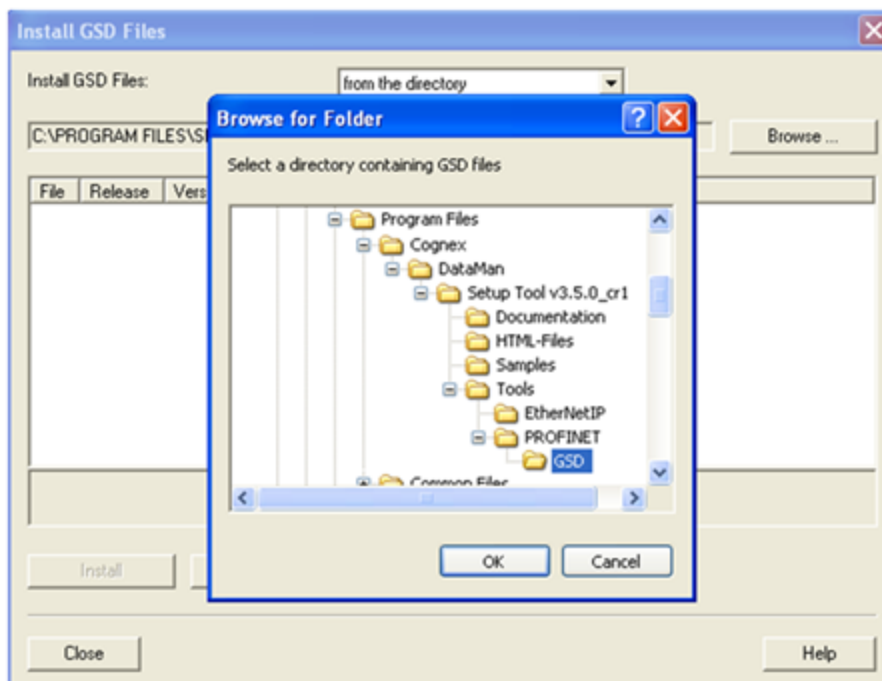
- Siemens Step 7 プログラミングソフトウェア (SIMATIC) がインストールされていることを確認します。
- Siemens ソフトウェアツールがお使いの DataMan デバイスを認識するように設定します。
- Generic Station Description (GSD) ファイルをインストールします。

次の手順を実行して PROFINET を設定します。

1. SIMATIC がお使いのマシン上にインストールされていることを確認します。
2. Windows の **[スタート]** メニューから SIMATIC Manager を起動します。
3. すでにプロジェクトが設定されている場合は、[Cancel (キャンセル)] を選択して [New Project (新しいプロジェクト)] ウィザードをスキップします。プロジェクトがない場合は、ウィザードに従って新しいプロジェクトを作成します。
4. Manager でプロジェクトが開いたら、[Hardware (ハードウェア)] アイコンをダブルクリックして [HW Config (ハードウェア設定)] ダイアログを開きます。メインメニューから、[Options (オプション)] → [Install GSD File (GSD ファイルをインストール)] を選択します。

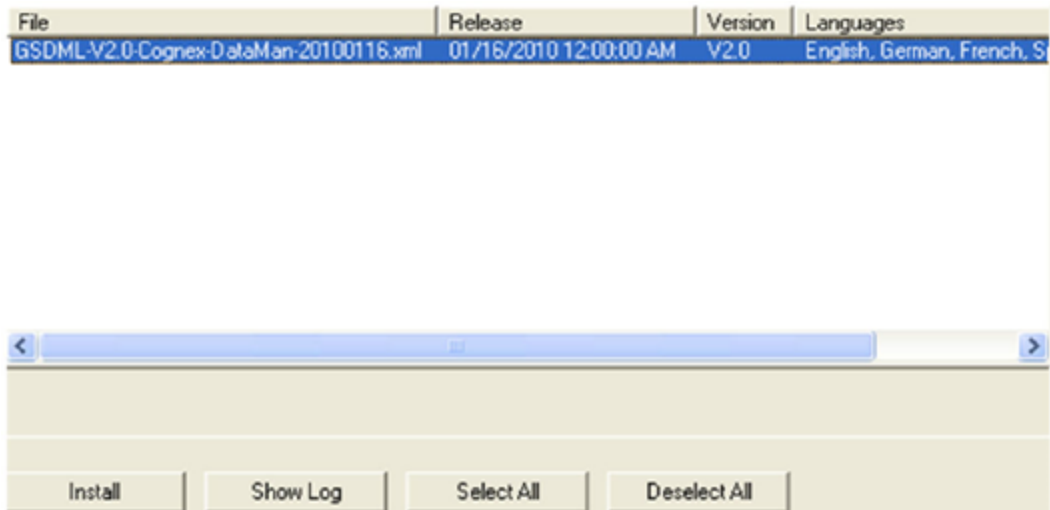


5. GSD ファイルがインストールされた場所 (または、GSD ファイルを Web からダウンロードした場合は、ファイルを保存した場所) を参照します。

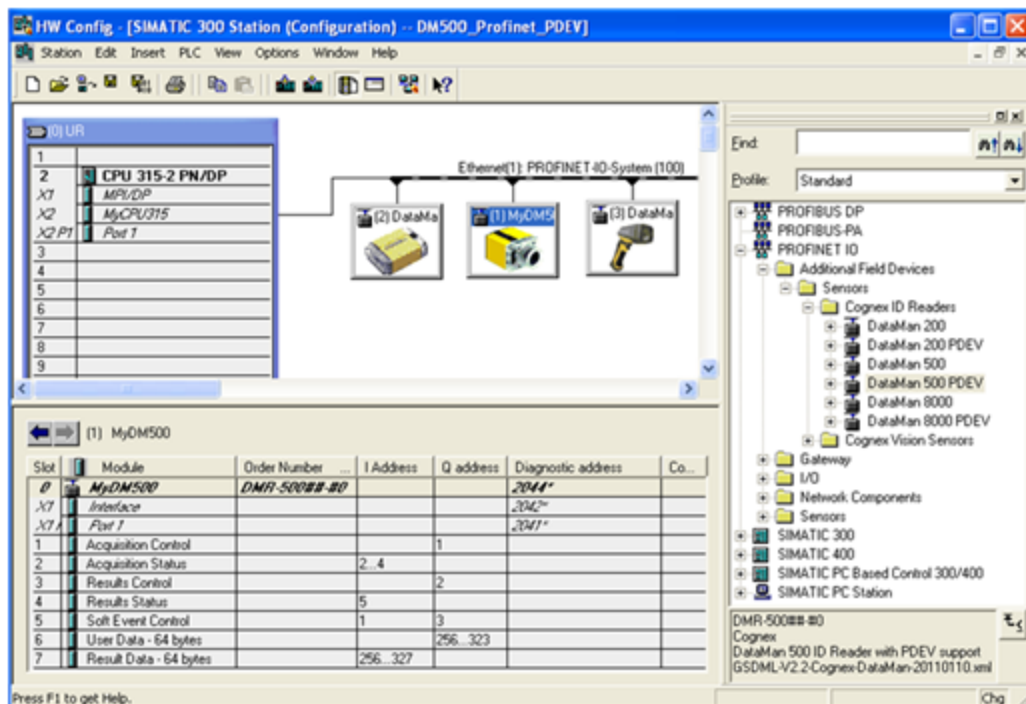


6. インストールする GSD ファイルを選択し、表示された指示に従ってインストールを完了します。

注: リストには複数の GSD ファイルがある場合があります。インストールするファイルが不確かな場合は、最新の日付のファイルを選択します。



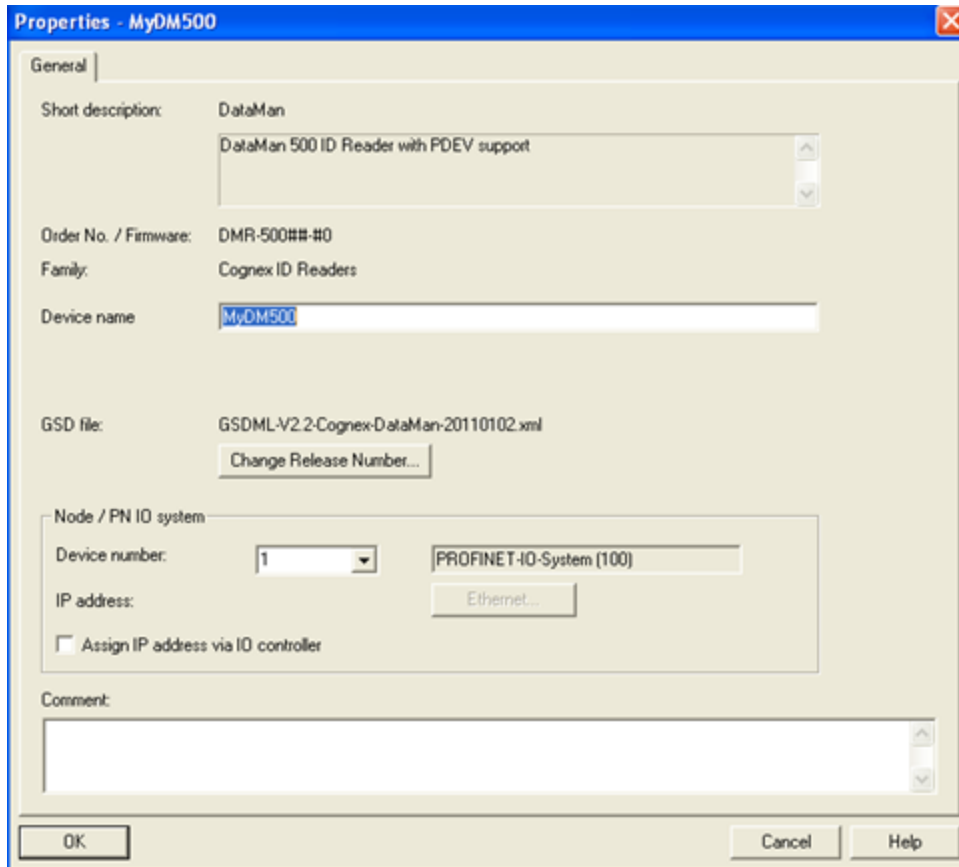
7. DataMan デバイスをプロジェクトに追加します。これによって DataMan がハードウェアカタログで使用可能になります。SIMATIC Hardware Config ツールを起動します。
8. メインメニューで [View (表示)] → [Catalog (カタログ)] と選択します。
9. カタログが表示されます。[Cognex ID Readers] ノードまで [PROFINET IO] ツリーを展開します。
10. 左のペインで DataMan リーダをマウスの左ボタンでドラッグして、[PROFINET IO] ネットワークシンボル上にドロップします。



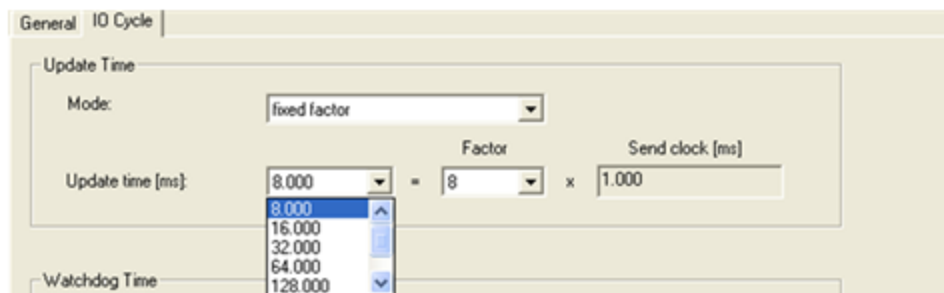
HW Config ツールが自動的に DataMan I/O モジュールをメモリ領域にマッピングします。

注： デフォルトでは、64 バイトのユーザデータおよび 64 バイトの結果データモジュールが挿入されます。これらの両方のモジュールで複数のサイズが使用可能です。パフォーマンスを最適化するには、アプリケーションの実際のデータ要件に厳密に適合するモジュールサイズを使用します。モジュールを変更するには、テーブル内で削除し、カタログから適切なサイズのモジュールを挿入するだけです。

11. [DataMan] アイコンを右クリックして [Object Properties (オブジェクトのプロパティ)] を選択します。
12. リーダに名前を付けます。これは実際の DataMan リーダの名前と一致させる必要があります。この名前が一意であることと、DNS 命名規約に準拠している必要があります。詳細については、SIMATIC ソフトウェアのヘルプを参照してください。
13. お使いの DataMan リーダが独自の固定 IP を使用するように設定している場合は、[Assign IP address via IO controller (IP アドレスを IO コントローラ経由で割り当てる)] ボックスのチェックを外します。設定していない場合、PLC で IP アドレスを割り当てるには、[Ethernet] ボタンを選択して適切なアドレスを設定します。

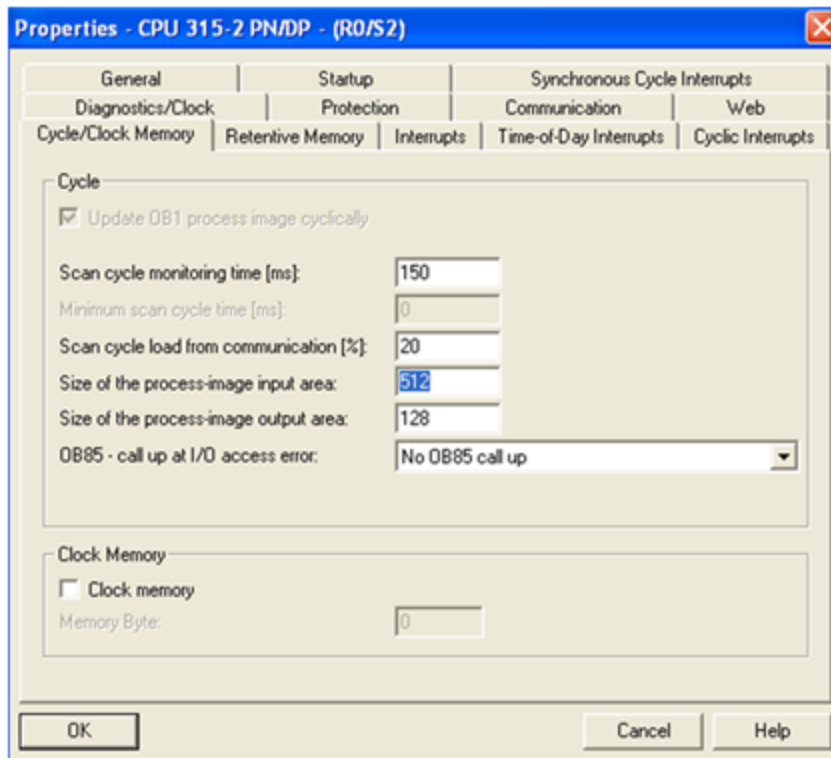


14. [IO Cycle (IO サイクル)] タブで、アプリケーションの適切なサイクル更新レートを選択します。



15. デフォルトでは、SIMATIC ソフトウェアがユーザデータおよび結果データモジュールをオフセット 256 にマッピングします。これは、デフォルトの処理画像領域サイズである 128 の範囲外です。つまり、デフォルトでは、これらのモジュール内のデータは、BLKMOV などの一部の SFC からアクセスできません。解決策としては、処理画像領域内でより低いオフセットにモジュールを再マッピングするか、または処理画像領域を拡大してこれらのモジュールを含めます。

処理画像領域の拡大を選択した場合は、モジュールサイズとデフォルトの 256 オフセットを合わせたものに十分なサイズにします。

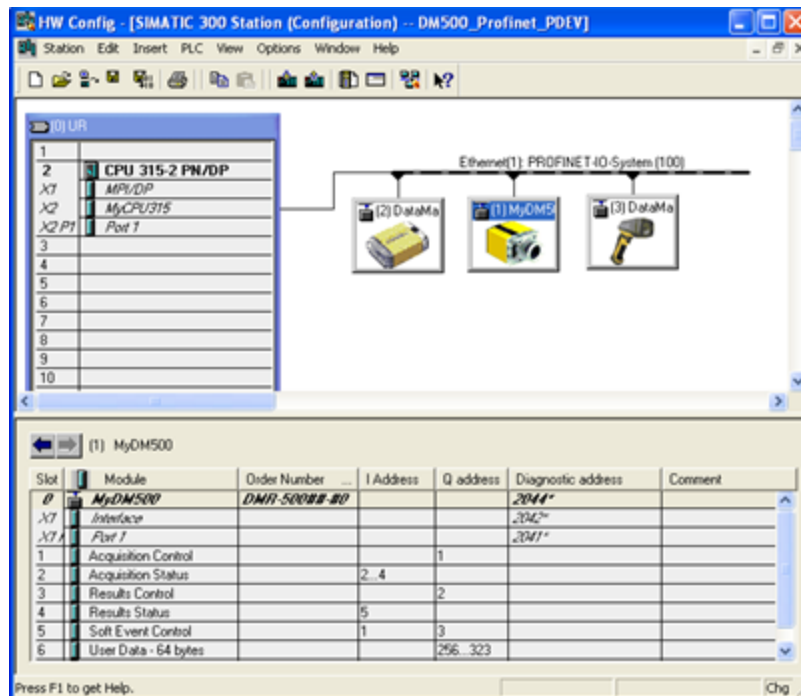


注： 処理画像を拡大すると、PLC スキャンのサイクル時間に関するパフォーマンスに影響する可能性があります。スキャン時間が重要となる場合は、許容可能な最小モジュールサイズを使用して、手動で処理画像内により低いオフセットに再マッピングします。

モジュール

DataMan への PROFINET の実装は、次の 7 つの I/O モジュールで構成されます。

1. 画像取り込みコントロールモジュール
2. 画像取り込みステータスモジュール
3. 結果コントロールモジュール
4. 結果ステータスモジュール
5. ソフトイベントコントロールモジュール
6. ユーザデータモジュール
7. 結果データモジュール



画像取り込みコントロールモジュール

画像取り込みを制御します。このモジュールは、PLC から DataMan デバイスに送信されたデータで構成されます。

スロット番号: 1

合計モジュールサイズ: 1 バイト

ビット	名前	説明
0	TriggerEnable	このビットをセットすると PROFINET 経由のトリガが有効になります。このビットをクリアすると、トリガが無効になります。
1	Trigger	このビットをセットすると、次の条件が満たされた場合に取り込みがトリガされます。 - TriggerEnable がセットされている - 現在処理中の画像取り込みがない - デバイスでトリガの準備ができています
2 ~ 7	Reserved	将来の使用のために予約済み

画像取り込みステータスモジュール

画像取り込みの現在のステータスを示します。このモジュールは、DataMan デバイスから PLC に送信されたデータで構成されます。

スロット番号: 2

合計モジュールサイズ: 3 バイト

ビット	名前	説明
0	TriggerReady	デバイスが新規トリガを受け入れ可能であることを示します。「TriggerEnable」がセットされ、デバイスで新規トリガの受け入れ準備ができています。場合、ビットは True です。
1	TriggerAck	DataMan が新規トリガを受信したことを示します。「Trigger」ビットが True である限り、このビットは True のままになります (つまり、トリガビットに連動します)。
2	Acquiring	DataMan が現在画像を取り込んでいることを示します。

3	MissedAcq	DataMan が画像取り込みを正常に実行できなかったことを示します。次に画像取り込みが成功したときに、ビットはクリアされます。
4～7	Reserved	将来の使用のために予約済み
8～23	TriggerID	発行される次のトリガの ID の値です (16 ビット整数)。発行されたトリガと後で受信する対応する結果データの照合に使用されます。これと同じ値が、結果データの ResultID で返されます。

注： 画像取り込みコントロールモジュールからトリガされた画像取り込みを実行できなかった場合にのみ、画像取り込みステータスレジスタの MissedAcq ビットがセットされます。

結果コントロールモジュール

結果データの処理を制御します。このモジュールは、PLC から DataMan デバイスに送信されたデータで構成されます。

スロット番号: 3

合計モジュールサイズ: 1 バイト

ビット	名前	説明
0	ResultsBufferEnable	「ResultData」のキューを有効にします。有効にすると、応答までの間 (新しい結果が届いた場合でも)、現在の結果データがそのまま保持されます。新しい結果がキューに入ります。現在の結果の応答があるたびに、結果の次のセットがキューから取り出されます (結果データモジュール内で使用可能になります)。DataMan は、「ResultsAvailable」ビットをクリアして応答に対応します。「ResultsAck」ビットがクリアされると、読み取り結果の次のセットがポストされ、「ResultAvailable」が True にセットされます。結果のバッファ処理が有効になっていない場合は、新たに受信された読み取り結果によって、結果データモジュールの内容が上書きされます。
1	ResultsAck	このビットは、PLC が最新の結果データを問題なく読み取ったことを確認するために使用されます。True にセットすると、「結果確定」ビットがクリアされます。結果のバッファ処理が有効になっている場合、結果データの次のセットがキューから取り出され、ResultsAvailable が再度 True にセットされます。
2～7	Reserved	将来の使用のために予約済み

結果ステータスモジュール

画像取り込みおよび結果のステータスを示します。このモジュールは、DataMan デバイスから PLC に送信されたデータで構成されます。

スロット番号: 4

合計モジュールサイズ: 1 バイト

ビット	名前	説明
0	Decoding	DataMan が取り込んだ画像をデコードしていることを示します。
1	DecodeComplete	新しい結果が使用可能になると、デコード操作の完了時にビットが反転します (0 から 1 または 1 から 0)。
2	ResultBufferOverrun	結果キューがいっぱいになったため、DataMan が一連の読み取り結果を破棄したことを示します。次の結果セットが正常にキューに入った場合にクリアされます。
3	ResultsAvailable	読み取り結果の新しいセットが使用可能であること (つまり、結果データモジュールの内容が有効であること) を示します。結果に対して応答があるとクリアされます。
4～6	Reserved	将来の使用のために予約済み
7	GeneralFault	エラーが発生したこと (つまり、ソフトイベント「一致文字列の設定」または「DMCC の実行」エラーが発生したこと) を示します。

ソフトイベントコントロールモジュール

ソフトイベントを開始し、完了の応答を受信するために使用されます。これは双方向の I/O モジュールであることに注意してください。PLC から送信されたモジュールデータがソフトイベントを開始します。DataMan デバイスによって送信されたモジュールデータは、完了の応答を行います。

スロット番号: 5

合計モジュールサイズ: 1 バイト (入力) および 1 バイト (出力)

PLC から DataMan に書き込まれたデータ:

ビット	名前	説明
0	コード登録	0 から 1 のビット遷移によって、コード登録操作が呼び出されます。
1	一致文字列の登録	0 から 1 のビット遷移によって、一致文字列の登録操作が呼び出されます。
2	焦点位置の登録	0 から 1 のビット遷移によって、焦点位置の登録操作が呼び出されます。
3	明るさの登録	0 から 1 のビット遷移によって、明るさの登録操作が呼び出されます。
4	登録の解除	0 から 1 のビット遷移によって、登録解除操作が呼び出されます。
5	予約済み	将来の使用のために予約済み
6	DMCC の実行	0 から 1 のビット遷移によって、DMCC 操作が呼び出されます。このイベントを呼び出す前に、有効な DMCC コマンド文字列を先に「ユーザデータ」に配置する必要があります。
7	一致文字列の設定	0 から 1 のビット遷移によって、一致文字列の設定操作が呼び出されます。このイベントを呼び出す前に、一致文字列データを先に UserData に配置する必要があります。

DataMan から PLC に書き込まれたデータ:

ビット	名前	説明
0	コード登録応答	「コード登録」操作が完了したことを示します。
1	一致文字列登録の応答	「一致文字列の登録」操作が完了したことを示します。
2	焦点位置登録応答	「焦点位置の登録」操作が完了したことを示します。
3	明るさ登録応答	「明るさの登録」操作が完了したことを示します。
4	登録解除応答	「登録の解除」操作が完了したことを示します。
5	予約済み	将来の使用のために予約済み
6	DMCC 実行の応答	「DMCC の実行」操作が完了したことを示します。
7	一致文字列設定の応答	「一致文字列の設定」操作が完了したことを示します。

ユーザデータモジュール

画像取り込み、デコード、およびその他の特殊な操作に対応するために PLC から DataMan に送信されるデータです。現在このモジュールは、「DMCC の実行」および「一致文字列の設定」ソフトイベントのみをサポートしています。

注： ユーザデータモジュールは、実際には 5 バージョンあります。指定したアプリケーションを使用するため、インスタンスを 1 つだけ設定することができます。「ユーザデータオプション」および「ユーザデータの長さ」フィールドは、各モジュールで同じです。「ユーザデータ」フィールドは、選択したモジュールに応じてサイズが変わります。アプリケーションが必要とする量のデータをやり取りするために、十分な大きさのモジュールを選択します。

スロット番号: 6

合計モジュールサイズ:

4 + 16 (16 バイトのユーザデータ)

4 + 32 (32 バイトのユーザデータ)

4 + 64 (64 バイトのユーザデータ)

4 + 128 (128 バイトのユーザデータ)

4 + 250 (250 バイトのユーザデータ)

バイト	名前	説明
0 ~ 1	UserDataOption	現在、「一致文字列の設定」ソフトイベントによってのみ使用されています。文字列を割り当てるターゲットコードを指定します (16 ビット整数)。 0: 文字列をすべてのターゲットに割り当てる 1: 文字列を 2D コードに割り当てる 2: 文字列を QR コードに割り当てる 3: 文字列を 1D / スタック / ポスタルコードに割り当てる
2 ~ 3	UserDataLength	「ユーザデータ」フィールドに実際に含まれている有効なデータのバイト数 (16 ビット整数)。
4 ~	UserData	画像取り込み、デコード、およびその他の特殊な操作に対応するために PLC から DataMan に送信されるデータです (バイトの配列)。

結果データモジュール

DataManから PLC に送信される読み取り結果データです。

リーダで画像取り込みに問題があった場合は、結果データモジュールで PROFINET 以外の関連データを詳しく調べます。「結果コード」部分には、リーダで発生したトリガオーバーランやバッファオーバーフローに関する情報が含まれています。

注： 結果データモジュールは、実際には 5 バージョンあります。指定したアプリケーションを使用するため、インスタンスを 1 つだけ設定することができます。「ResultID」、「ResultCode」、「ResultExtended」、および「ResultLength」の各フィールドは、それぞれのモジュールで同じです。「ResultData」フィールドは、選択したモジュールに応じてサイズが変わります。アプリケーションが必要とする量の結果データをやり取りするために、十分な大きさのモジュールを選択します。

スロット番号: 7

合計モジュールサイズ:

8 + 16 (16 バイトの ResultData)

8 + 32 (32 バイトの ResultData)

8 + 64 (64 バイトの ResultData)

8 + 128 (128 バイトの ResultData)

8 + 246 (246 バイトの ResultData)

バイト	名前	説明
0 ~ 1	ResultID	これらの結果を生成したトリガが発行されたときの、「TriggerID」の値です。トリガと対応する結果データの照合に使用されます (16 ビット整数)。
2 ~ 3	ResultCode	これらの結果を生成した読み取りの成功または失敗を示します (16 ビット整数)。 ビット 0, 1= 読み取り, 0= 読み取り失敗 ビット 1, 1= 妥当性検査合格, 0= 妥当性検査失敗 (または妥当性検査未実施) ビット 2, 1= 検証合格, 0= 検証失敗 (または検証未実施) ビット 3, 1= 画像取り込みトリガオーバーラン ビット 4, 1= 画像取り込みバッファオーバーフロー ビット 5 ~ 15 予約済み
4 ~ 5	ResultExtended	現在使用されていません (16 ビット整数)。
6 ~ 7	ResultLength	「ResultData」フィールドに含まれている読み取りデータの実際のバイト数 (16 ビット整数)。
8 ~	ResultData	デコードされた読み取り結果データ (バイトの配列)

操作

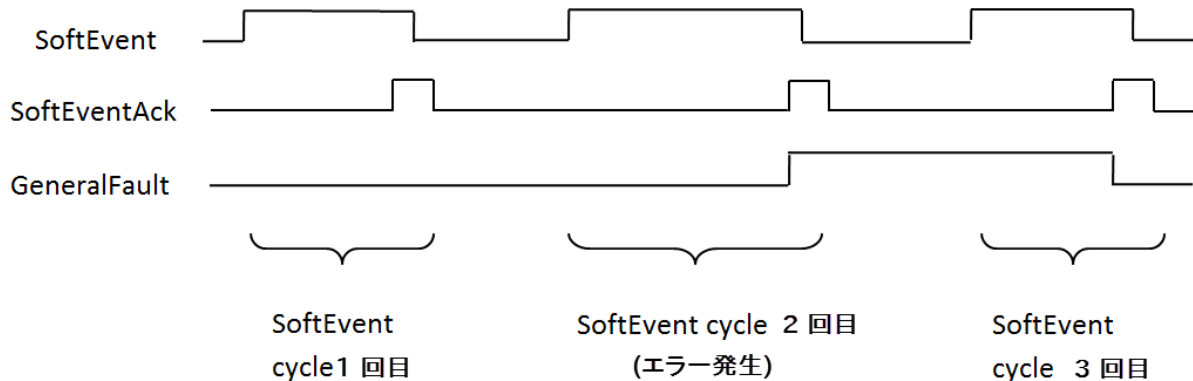
SoftEvents

SoftEvents は「仮想」入力として機能します。SoftEvent の値が 0 から 1 に変化すると、そのイベントに関連付けられているアクションが実行されます。アクションが完了すると、対応する SoftEventAck ビットが 0 から 1 に変化して完了を示します。対応する SoftEvent ビットが 0 に設定されると、応答ビットも 0 に戻ります。

“ExecuteDMCC” および “SetMatchString” ソフトイベントアクションには、ユーザ提供データが必要です。ソフトイベントを呼び出す前に、このデータを UserData モジュールの UserData および UserDataLength 領域に書き込む必要があります。これらのソフトイベントはいずれも UserData に依存しているため、一度に呼び出すことができるのは 1 つのみです。

一般的なエラーインジケータ

通信関連のエラーが発生すると、“GeneralFault” ビットが 0 から 1 に変化します。現時点で唯一サポートされているエラー条件は、ソフトイベント操作です。ソフトイベント操作が失敗すると、エラービットがセットされます。次のソフトイベント操作まで、またはトリガを無効にして再度有効にするまでの間、エラービットはセットされたままです。



画像取り込みシーケンス

DataMan の画像取り込みをトリガする方法はいくつかあります。画像取り込みコントロールモジュールの Trigger ビットを操作して明示的に行うことが可能であり、外部の IO 配線の入力によってトリガでき、最後に、DMCC コマンドを介してトリガすることができます。ここでは、画像取り込みコントロールモジュールのビット操作について説明します。

起動時には、「TriggerEnable」ビットは False になっています。これを True にセットして、トリガを有効にする必要があります。デバイスでトリガを受け入れる準備ができると、「TriggerReady」ビットが True にセットされます。

「TriggerReady」ビットが True である間に、リーダで「Trigger」ビットが 0 から 1 に変わったことが確認されるたびに、画像取り込みが開始されます。クライアント (PLC) は、同じ状態値がトリガ応答ビットで認識されるまで、新しい状態で保持される必要があります (これはリーダが変化を認識するために必要なハンドシェイクです)。

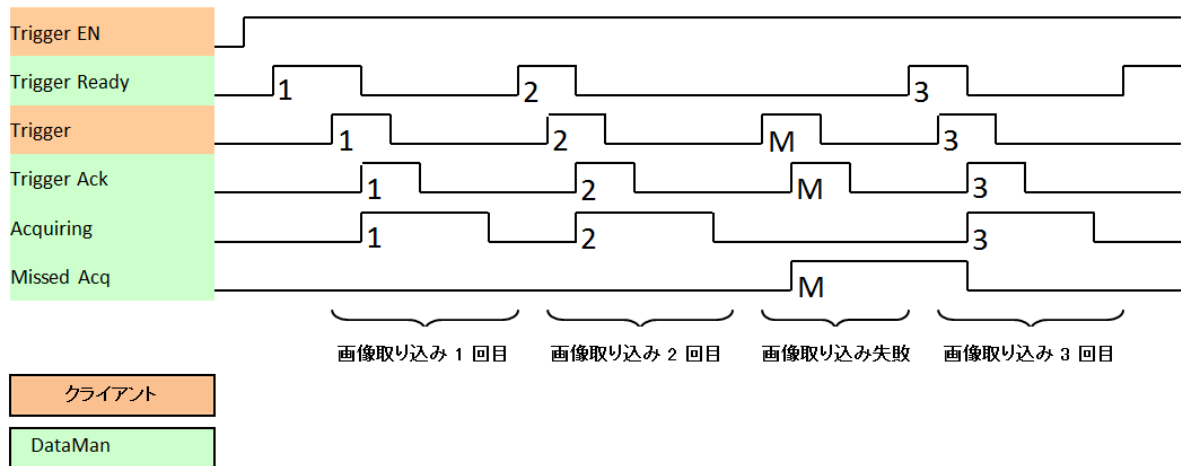
画像取り込み中、TriggerReady ビットはクリアされ、Acquiring ビットが True にセットされます。画像取り込みが完了すると、Acquiring ビットはクリアされます。デバイスで新しい画像取り込みを開始する準備ができると、TriggerReady が再度 True にセットされます。

結果のバッファ処理が有効になっている場合、画像取り込み操作とデコード操作のオーバーラップが可能です。デコード処理中に画像取り込みが完了すると、TriggerReady は high にセットされます。これを利用して、全体のトリガレートを高速にすることができます。結果のバッファ処理が有効になっていない場合、画像取り込み操作とデコード操作の両方が完了するまで、TriggerReady ビットは low のままです。

トリガシステムを強制的にリセットするには、TriggerReady ビットが 0 になるまで、TriggerEnable ビットを False に設定します。その後、TriggerEnable を True にセットして画像取り込みを再度有効にします。

特殊なケースとして、読み取り操作が完了する前にトリガ信号がクリアされ、画像取り込みがキャンセルされることがあります。この場合、視野にコードがなくても自動読み取りモードおよびマニュアルモードでの読み取りがキャンセルできてしまいます。読み取りが誤っ

でキャンセルされないよう、TriggerAck と ResultsAvailable の両方が True になる (または DecodeComplete の状態が反転する) まで、PLC で Trigger 信号が True のまま保持されるようにするとよいでしょう。



デコードまたは結果シーケンス

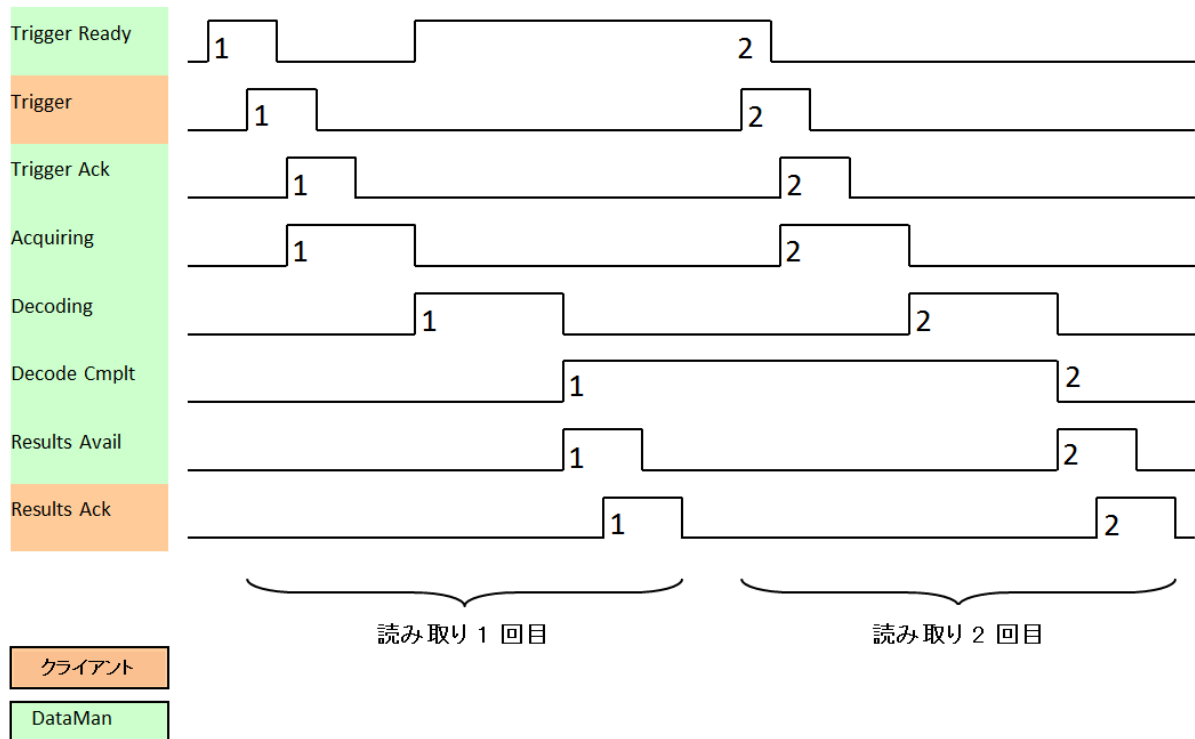
取り込んだ画像はデコードされます。デコード中は結果ステータスモジュールの「Decoding」ビットがセットされます。デコードが完了すると、デコード中ビットがクリアされ、「DecodeComplete」ビットが反転します。

「ResultsBufferEnable」ビットによって、リーダがデコード結果をどのように処理するかが決まります。ResultsBufferEnable が False にセットされると、即座にデコード結果が結果モジュールに配置され、ResultsAvailable が True にセットされます。

ResultsBufferEnable が True にセットされると、新しい結果がキューに入れられます。それより前のデコード結果は、「ResultsAck」ビットを True にセットしているクライアントによって応答されるまで、結果モジュール内に残ります。ResultsAvailable ビットがクリアされた後、クライアントがResultsAck ビットを False に戻すと、次にキューに入れられた結果を結果モジュールに配置することができます。これは、DataManの クライアント (PLC) によって結果が受信されたことを確認するために必要なハンドシェークです。

DecodeStatusRegister の動作

ビット	ビット名	バッファが無効な場合の結果	バッファが有効な場合の結果
1	Decoding	画像をデコードしているときにセットします。	画像をデコードしているときにセットします。
2	DecodeComplete	画像デコードの完了時に反転します。	画像デコードの完了時に反転します。
3	ResultsBuffer Overflow	ゼロにセットしたままにします。	クライアントが直前の結果の応答に失敗したために、デコード結果をキューに入れられなかった場合にセットします。デコード結果が問題なくキューに入ると、クリアされます。
4	ResultsAvailable	新しい結果が結果モジュールに配置されたときにセットします。結果応答を true にセットすることによって結果に応答が行われるまで、セットされたままになります。	新しい結果が結果モジュールに配置されたときにセットします。ResultsAck を true にセットすることによって結果に応答が行われるまで、セットされたままになります。



結果のバッファ処理

デコード結果のキューを有効にするオプションがあります。有効にした場合、PLC が読み取れるようになるまで有限数のデコード結果データをキューに入れることができます。これは、クライアント (PLC) のパフォーマンスが一時的に低下した場合、または読み取りアクティビティが急増した場合に、データフローを円滑化するのに役立ちます。

また、結果のバッファ処理が有効になっている場合、画像取り込み操作とデコード操作のオーバーラップが可能です。アプリケーションによっては、これを利用して全体のトリガレートを高速にすることができます。詳細については、前の「画像取り込みシーケンス」の説明を参照してください。

一般的に、結果の送信よりも読み取りの方が高速である場合、バッファ処理が有効であるかないかの主な違いによって、どの結果が破棄されるかが決まります。バッファ処理が有効でない場合は、最新の結果が保持され、それより前の結果は (PLC の速度が不十分で読み取れなかったために) 失われます。基本的に、最新の結果によって前の結果が上書きされます。バッファ処理が有効になっている (かつキューがいっぱいになっている) 場合、結果キューに空きが生じるまで最新の結果は破棄されます。

注： キューがオーバーフローし、バッファ処理が無効になっている場合、TriggerID 値と ResultID 値の差が 1 より大きくなります。この差は、キューがいっぱいであるためにキューに入れることができなかった読み取りの数を表します (失われた読み取りの数 = TriggerID - ResultID - 1)。次の読み取りの後、ResultID 値は通常の動作値である TriggerID - 1 に戻ります。

Siemens の例

この節では、Siemens S7-300 PLC で DataMan を使用する例をいくつか紹介します。ユーザが S7-300 および SIMATIC プログラミングソフトウェアを熟知していることを前提としています。

シンボルテーブル

必須ではありませんが、DataMan I/O モジュール要素用のシンボルを定義すると非常に便利です。コードがずっと読みやすくなり、誤りが少なくなります。このサンプルテーブルは、DataMan リードの標準のインスタンス用に定義されたシンボルを示しています。DataMan I/O モジュールは、プロジェクト内の異なるアドレスに位置する場合もあることに注意してください。必ず I/O モジュールの特定のオフセットに基づいて、シンボル定義を調整してください。

	Status	Symbol	Address	Data type	Comment
1		Reader_TriggerEnable	Q 1.0	BOOL	
2		Reader_Trigger	Q 1.1	BOOL	
3		Reader_TriggerReady	I 2.0	BOOL	
4		Reader_TriggerAck	I 2.1	BOOL	
5		Reader_Acquiring	I 2.2	BOOL	
6		Reader_MissedAck	I 2.3	BOOL	
7		Reader_TriggerID	MV 3	WORD	
8		Reader_BufferEnable	Q 2.0	BOOL	
9		Reader_ResultsAck	Q 2.1	BOOL	
10		Reader_Decoding	I 5.0	BOOL	
11		Reader_DecodeComplete	I 5.1	BOOL	
12		Reader_ResultsOverrun	I 5.2	BOOL	
13		Reader_ResultsAvailable	I 5.3	BOOL	
14		Reader_GeneralFault	I 5.7	BOOL	
15		Reader_TrainCode	Q 3.0	BOOL	
16		Reader_TrainMatchString	Q 3.1	BOOL	
17		Reader_TrainFocus	Q 3.2	BOOL	
18		Reader_TrainBrightness	Q 3.3	BOOL	
19		Reader_UnTrain	Q 3.4	BOOL	
20		Reader_ExecuteDmcc	Q 3.6	BOOL	
21		Reader_SetMatchString	Q 3.7	BOOL	
22		Reader_TrainCodeAck	I 1.0	BOOL	
23		Reader_TrainMatchStrAck	I 1.1	BOOL	
24		Reader_TrainFocusAck	I 1.2	BOOL	
25		Reader_TrainBrightAck	I 1.3	BOOL	
26		Reader_UnTrainAck	I 1.4	BOOL	
27		Reader_ExecuteDmccAck	I 1.6	BOOL	
28		Reader_SetMatchStringAck	I 1.7	BOOL	
29		Reader_UserDataOption	QW 256	WORD	
30		Reader_UserDataLength	QW 258	WORD	
31		Reader_ResultID	MV 256	WORD	
32		Reader_ResultCode	MV 258	WORD	
33		Reader_ResultExtended	MV 260	WORD	
34		Reader_Resultlength	MV 262	WORD	

Press F1 to get Help.

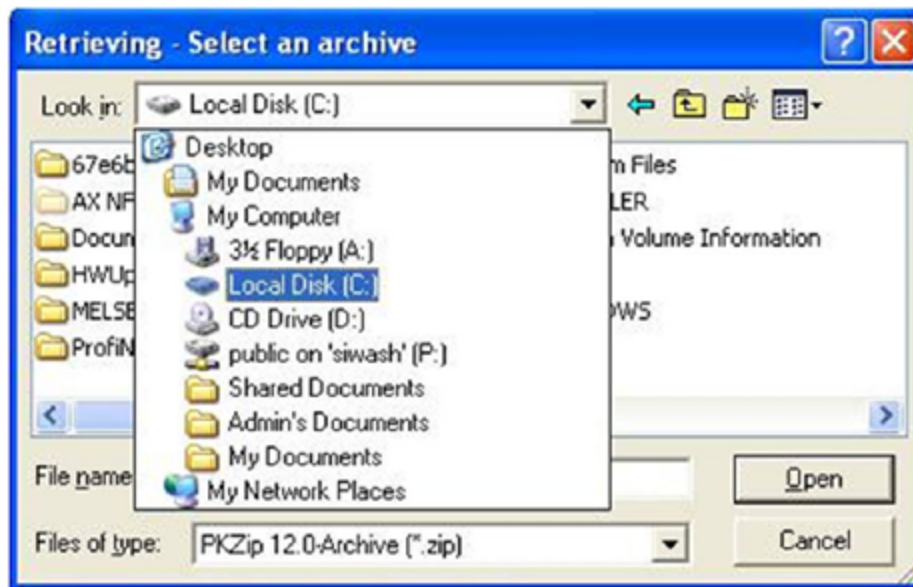
トリガと結果取得

サンプルプログラム "DM200_SampleRead" を実行して、プログラム例を完了します。

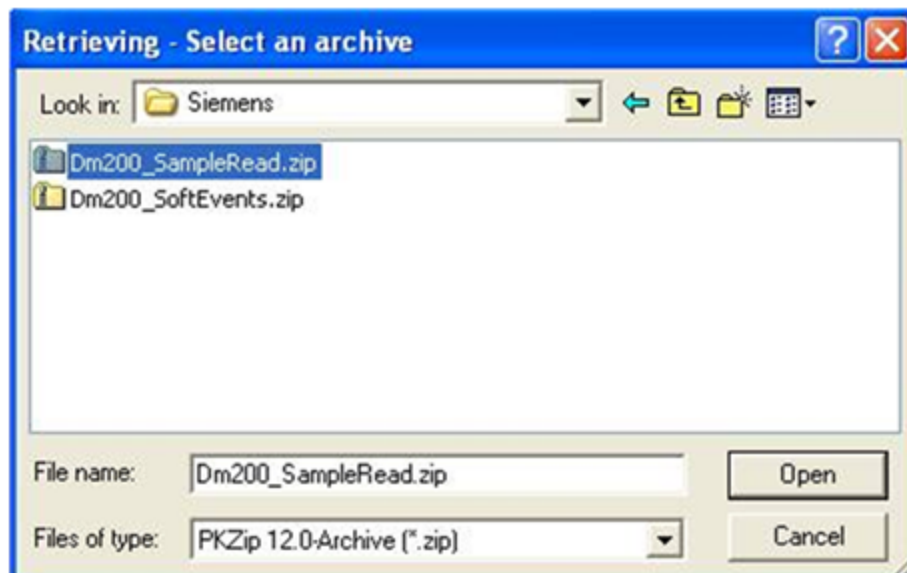
① 注： このサンプルは、PROFINET が有効になっている DataMan リードで使用できます。

次の手順を実行してこのプログラムをインストールします。

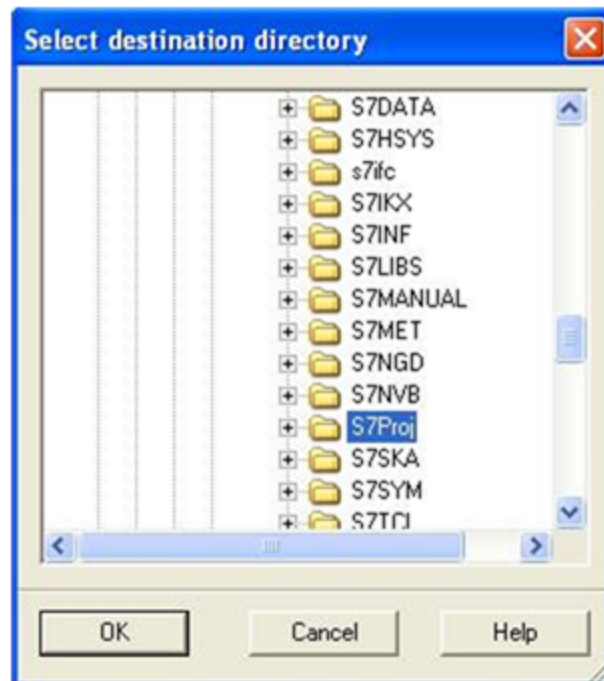
1. SIMATIC Manager ソフトウェアを起動します。
2. 開いているアプリケーションがあれば閉じます。
3. メインメニューから [File (ファイル)] → [Retrieve (取得)] と選択します。
4. PC 上を参照してサンプルファイルを探します。



5. [Siemens] フォルダを探して zip ファイルを選択します。



6. 保存先のディレクトリを選択して、プロジェクトを PC 上に保存します。



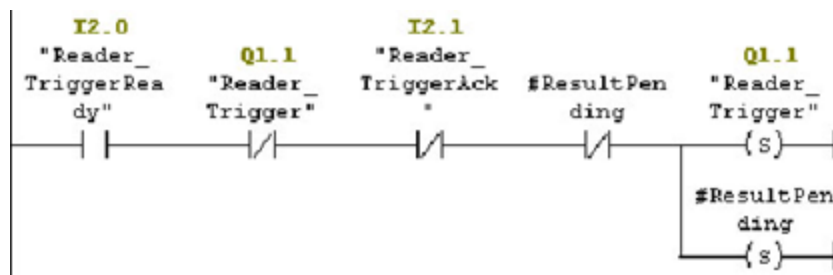
7. Siemens ソフトウェアがサンプルアーカイブを抽出し、使用可能にします。
8. 基本的なことを要約すると、結果の読み取りおよび取得のプロセスは、次のような構成になっています。
9. アプリケーション内で、読み取り結果を保存する領域を定義します。結果データを保存できる方法および場所に関するオプションが多数あります。サンプルでは、アプリケーションで関心の対象である、結果データモジュールのフィールドを含んでいるデータブロック (DB) を定義します。

Address	Name	Type	Initial value	Comment
0.0		STRUCT		
+0.0	ID	INT	0	ID of this read result
+2.0	Flags	INT	0	Flags indicating success or failure
+4.0	Length	INT	0	Number of data bytes in the array Value[]
+6.0	Value	ARRAY[1..63]		Code value read
*1.0		CHAR		
=70.0		END_STRUCT		

10. リーダを有効にします。



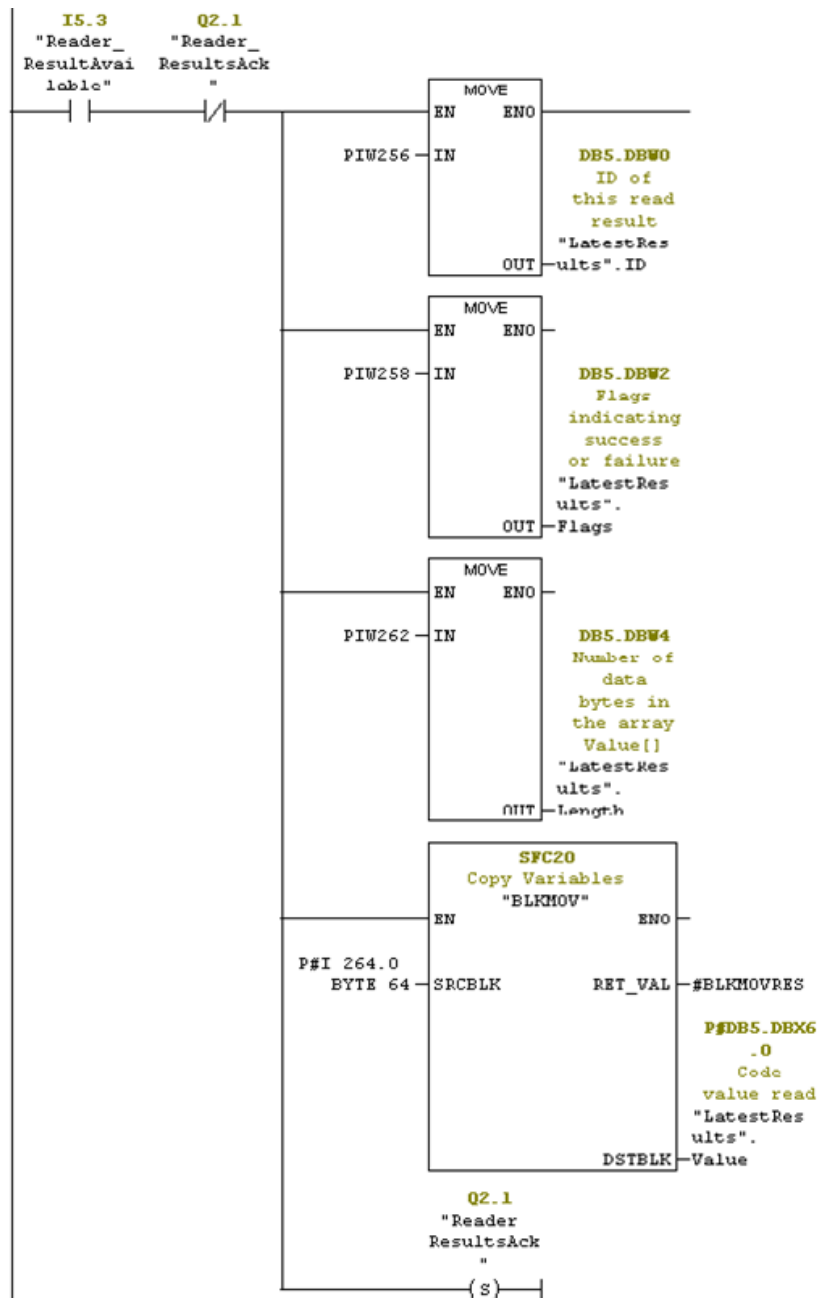
11. トリガ信号を設定し、読み取りが保留中であることを示すセマフォをセットします。



12. トリガ信号に応答があったら即座に、トリガ信号をクリアします。

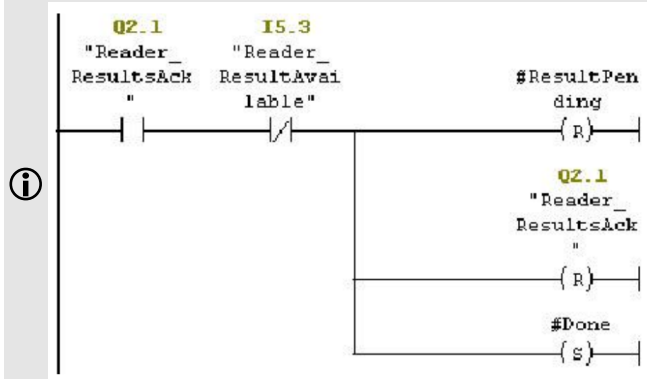


13. 結果が使用可能になったら即座に、結果データのコピーを保存し、結果応答信号をセットします。



14. リーダに結果応答信号が表示されたら、結果応答をクリアし、読み取り保留中セマフォをクリアして、読み取りプロセスが完了したことを示します。

注：リーダに PLC の「ResultsAck」信号が表示されると即座に、「ResultsAvailable」がクリアされます。



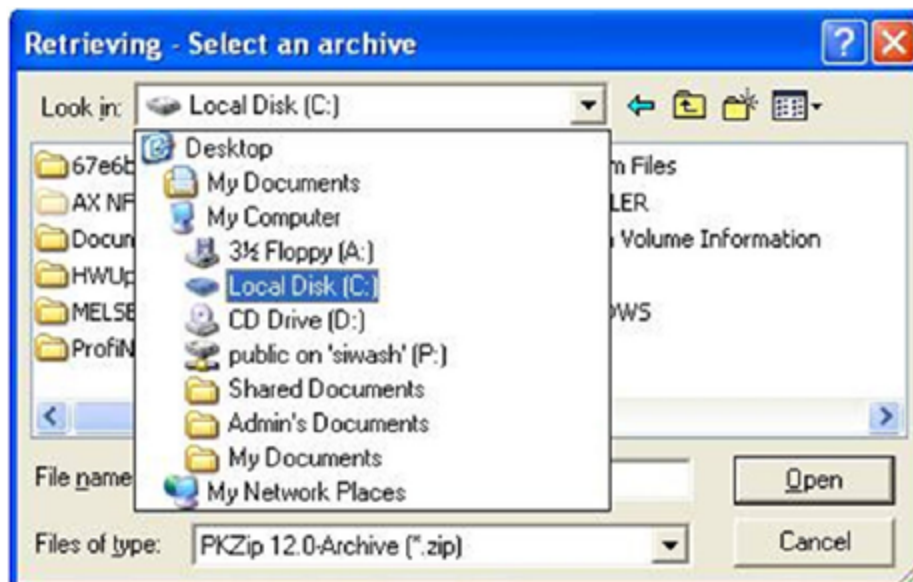
ソフトイベントの使用

サンプルプログラム "DM200_SoftEvents" を実行して、プログラム例を完了します。

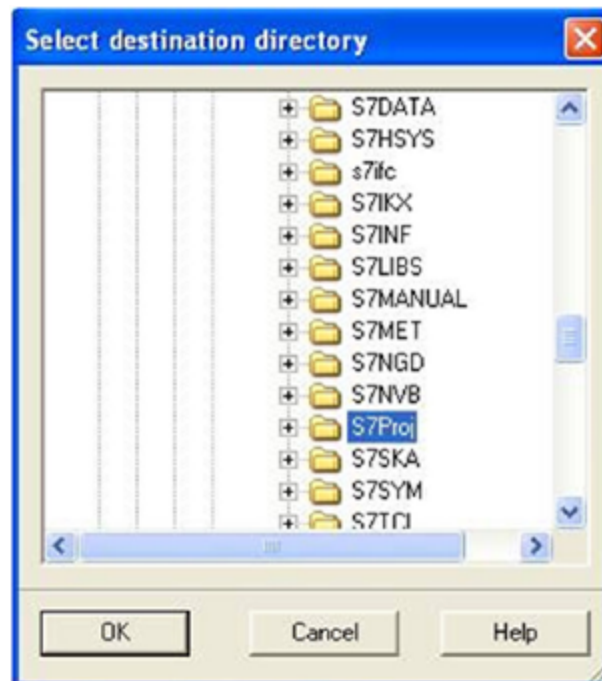
① 注：このサンプルは、PROFINET が有効になっている DataMan リードで使えます。

次の手順を実行してこのプログラムをインストールします。

1. SIMATIC Manager ソフトウェアを起動します。
2. 開いているアプリケーションがあれば閉じます。
3. メインメニューから [File (ファイル)] → [Retrieve (取得)] と選択します。
4. PC 上を参照してサンプルファイルを探します。



5. [Siemens] フォルダを探して Dm200_SoftEvents.zip を選択します。
6. 保存先のディレクトリを選択して、プロジェクトを PC 上に保存します。



7. Siemens ソフトウェアがサンプルアーカイブを抽出し、使用可能にします。

ソフトイベントは、1 つの制御ビットを操作するだけで、アクティビティを呼び出す手段です。各ビットのアクティビティはあらかじめ定義されています (詳細については、「[SoftEvents](#)」の節を参照してください)。「DMCC の実行」および「一致文字列の設定」を例外として、すべてのソフトイベントは同じ方法で呼び出すことができます。「DMCC の実行」および「一致文字列の設定」では、イベントを呼び出す前に、ユーザデータモジュールにアプリケーションデータを読み込む追加の手順が必要です。

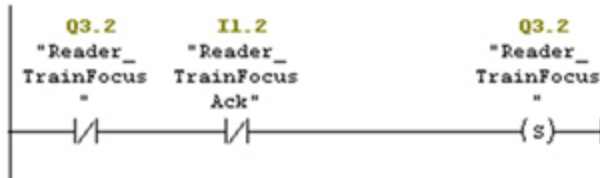
基本的なことを要約すると、ソフトイベント呼び出しのプロセスは、次のような構成になっています。

FC3: 焦点位置の登録

「焦点位置の登録」操作を開始し、完了まで監視。

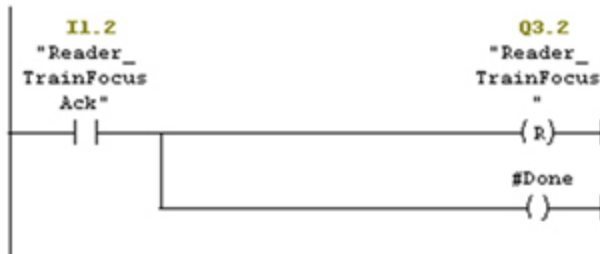
ネットワーク1: タイトル:

「焦点位置の登録」信号を発行。



ネットワーク2: タイトル:

リーダからの応答があり次第、「焦点位置の登録」信号を解放。



ネットワーク3: タイトル:

戻り FV ステータスを設定。

リーダから応答を受信した後のみ TRUE が返されます。その他の場合は FALSE が返されます。



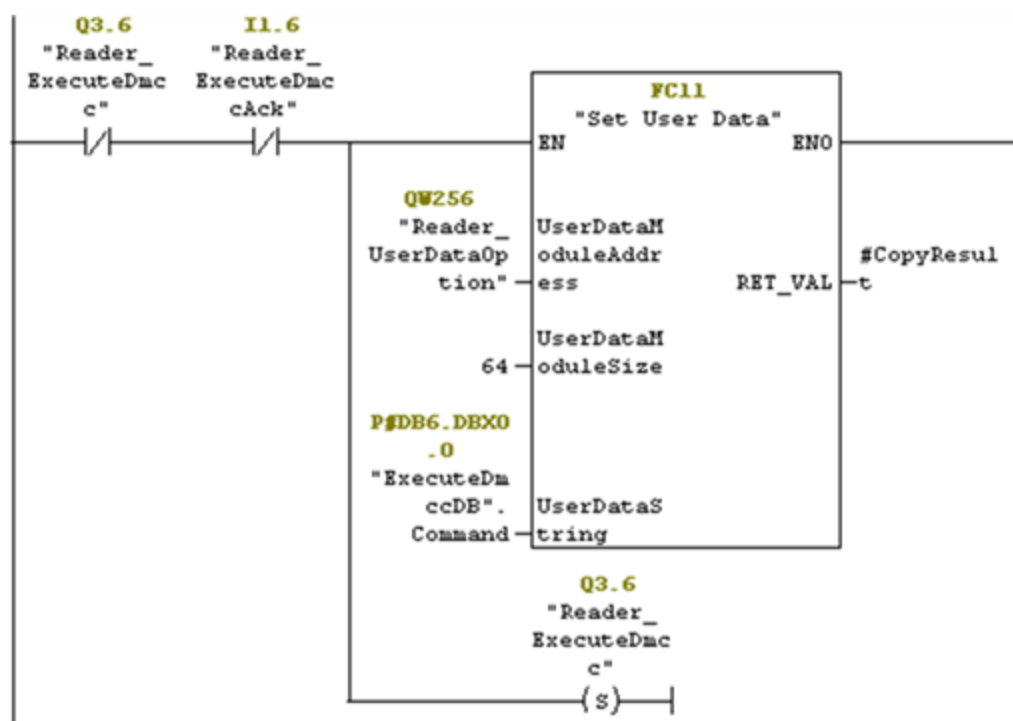
DMCC コマンドの実行

完全なサンプルプログラムについては、サンプルプログラム "DM200_SoftEvents" を参照してください (インストール方法については、[「ソフトイベントの使用」](#)の節を参照してください)。

注: このサンプルは、PROFINET が有効になっている DataMan リーダで使用できます。

「ExecuteDMCC の実行」は、イベントを呼び出す前に、ユーザーデータモジュールに適切な DMCC コマンド文字列を読み込む追加の手順が必要です。ソフトイベントメカニズムには、DMCC 応答データを返す手段が備わっていない (エラーの表示を除く) ことに注意してください。そのため、このメカニズムを DMCC “||>GET…” コマンドで使用することはできません。

DMCC コマンド実行のプロセスは、その他すべてのソフトイベントと同じですが (上記の例を参照)、異なる点は、ソフトイベントを呼び出す手順にコマンド文字列をユーザーデータモジュールにコピーすることも含んでいることです。この例では、コマンド文字列がデータブロック内に存在します。この例を拡張して、コピー機能でインデックス値によって参照できるコマンド文字列の配列を、データブロックに使用することができます。それによって、アプリケーションが必要とするすべての DMCC コマンドを事前定義し、インデックスによって簡単に呼び出すことができます。



「Set User Data」(FC11) 機能は、提供された文字列を単純にユーザデータモジュールにコピーします。実際の STL コードについては、サンプルプログラムを参照してください。

ワイヤレス DataMan 向けの産業用プロトコル

DataMan ワイヤレスリーダで産業用イーサネットプロトコルのいずれかを使用するときには、特殊な事項を考慮する必要があります。産業用プロトコル接続は、PLC とワイヤレスデバイスとの間ではなく、PLC と DataMan ベースステーションとの間に確立されます。

DataMan ワイヤレスリーダは、次の状態になる場合があります。

- 電源がオフになる
- ハイバネーションモードになる
- 物理的に圏外になる

自動電源オフおよびハイバネーションは、電力を維持し、再充電の間隔を延ばす機能です。

次の節では、DataMan ベースステーション (これを經由して DataMan ワイヤレスリーダに情報をルーティングします) と通信するために必要な設定について詳細を説明します。

プロトコル操作

ワイヤレスリーダを使用できない場合があるため、産業用イーサネットプロトコルはベースステーションで実行されます。ベースステーションは Ethernet ネットワークに直接接続され、(リーダが使用不能の場合も) 常にオンラインです。したがって、リーダが使用できないときも、イーサネットプロトコルは中断されません。そのため、いくつか特別な注意が必要です。

Ethernet アドレス

イーサネットプロトコルはベースステーションから実行されるので、プロトコル設定時にベースステーションの Ethernet アドレスが必要です。例えば、RSLogix5000 ソフトウェアパッケージを使用して EtherNet/IP を設定する場合、DM8000 リーダを選択しますが、(リーダではなく) ベースステーションの IP アドレスを入力する必要があります。プロトコルを使用している PLC とベースステーションとの通信が必要です。

PLC からのトリガ

ワイヤレスリーダが使用できない場合、ワイヤレスリーダを PLC からトリガすることはできません。PLC はリーダからのすべての読み取り結果を受信します。また、ソフトイベントと DMCC のコマンドがサポートされています (リーダが使用できる場合)。

ソフトイベント

PLC からトリガされたソフトイベントがサポートされています。ただし、リーダが使用できない場合、これらのイベントは失敗します。このようなエラーは「GeneralFault」ビットで表されます。

注：ソフトイベントに割り当てられている操作の中には、ハンドヘルド型の DataMan リーダでは一般的にサポートされていないものがあります (例:「コード登録」、「焦点位置の登録」、「明るさの登録」)。サポートされていない操作が行われた場合も「General Fault」信号が設定されます。

DMCC

PLC から発行される DMCC コマンドがサポートされています。ただし、リーダが使用できない場合、これらのイベントは失敗します。使用中の産業用イーサネットプロトコルによって、「General Fault」信号またはエラーステータスのリターンコードでエラーが示されます。リーダが使用できないことを示す新しい DMCC エラーリターンコード「105」が確立されています。

① 注： PLC からワイヤレスリーダに DMCC コマンドを発行する場合は、DMCC の応答フォーマットを変更しないでください (「COM.DMCC-RESPONSE」コマンドを発行しないでください)。これを発行すると、ワイヤレスリーダで DMCC が動作しなくなる可能性があります。このコマンドは、Telnet などの非産業用イーサネット接続で使用できます。

オフラインバッファ処理

ワイヤレスリーダがベースステーションから物理的に圏外になる場合、すべての読み取りが自動的にバッファ処理されます。バッファ容量はさまざまな要素によって異なります。ただし、一般的には数百件の読み取りが保存可能です。リーダがハイバネーション (スリープモード) になる場合、バッファ処理された読み取りも保存されます。リーダがベースステーションの有効範囲内に戻ると、バッファ処理されていた読み取りが自動的にダウンロードされます。

ユーザのアプリケーションでこのオフラインバッファ処理が予期される場合、産業用イーサネットプロトコルのバッファ処理も有効にする必要があります。プロトコルのバッファ処理により、PLC に返される読み取り結果のレートを PLC が制御できます。これによって、読み取り結果が速すぎて PLC が圧迫されるのを防止します。プロトコルのバッファ処理の有効化と使用の詳細については、本書の個々の産業用プロトコルの節を参照してください。

① 注： ベースステーションは最大 500 件の結果をバッファ処理できます。

産業用プロトコルのステータス

産業用プロトコルの Status フィールド (Setup Tool の [産業用プロトコル] タブ) には、ステータスダイアログで最後に記録されたメッセージが表示されます。このメッセージは新たなメッセージが記録されるまで表示されます。産業用プロトコルが有効になっていない時は、ステータスフィールドに何も表示されません。

ステータスメッセージはプロトコルによって異なり、中にはメッセージを記録しないプロトコルもあります。次の表に、表示するプロトコルのメッセージの概要を示します。

プロトコル/リーダ	メッセージ	説明
Wi-Fi リーダ	Industrial ethernet protocols shutdown (産業用イーサネットプロトコルがシャットダウンされました)	プロトコルが実行されていない場合に表示されます。
	Proxy %s connection opening (プロキシ %s 接続を開いています)	Wi-Fi リーダの接続を開いています。s == 接続タイプ ("Command", "Result", または "User DMCC")
	Proxy %s open (プロキシ %s が開いています)	接続が開いています。s == 接続タイプ ("Command", "Result", または "User DMCC")
	Proxy %s not open (プロキシ %s が開いていません)	接続を開けませんでした。s == 接続タイプ ("Command", "Result", または "User DMCC")
	Proxy %s closed (プロキシ %s が閉じています)	接続が閉じています。s == 接続タイプ ("Command", "Result", または "User DMCC")
Modbus TCP	Modbus TCP starting... (Modbus TCP を起動しています)	プロトコルの起動中です。
	Modbus TCP: Server Setup Failed (Modbus TCP: サーバ設定に失敗しました)	サーバを起動できませんでした。
	Modbus TCP running (Modbus TCP を実行しています)	プロトコルの実行中です。
	Modbus TCP: Add block %s failed (Modbus TCP: %s ブロックの追加に失敗しました)	データブロックの設定エラー
	Modbus TCP control block %s (Modbus TCP 制御ブロック %s)	ModbusTCP データブロックのステータス。s == "info" または "error"

プロトコル/リーダ	メッセージ	説明
SLMP プロトコル	Opening SLMP scanner connection %s:%x (SLMP スキャナ接続 %s:%x を開いています)	SLMP 接続を開始しています。s==IP アドレス、x==ポート番号
	SLMP scanner connection established %s:%x (SLMP スキャナ接続 %s:%x が確立されました)	SLMP 接続が確立されました。s==IP アドレス、x==ポート番号
	SLMP scanner connection %s:%x rejected (SLMP スキャナ接続 %s:%x が拒否されました)	PLC が接続を拒否しました。s==IP アドレス、x==ポート番号
	SLMP scanner connection %s:%x faulted (SLMP スキャナ接続 %s:%x が失敗しました)	その他の接続エラーが発生しました。s==IP アドレス、x==ポート番号
	SLMP scanner poll rate set too low (SLMP スキャナのポーリングレートが低すぎます)	設定されている SLMP ポーリングレートが低すぎます。
	SLMP scanner connection normal %s:%x (SLMP スキャナ接続 %s:%x は正常です)	SLMP 接続が復元されました。s==IP アドレス、x==ポート番号
	SLMP Scanner control block %s (SLMP スキャナコントロールブロック %s)	SLMP データブロックのステータス。s == “info” または “error”
	SLMP Scanner failed transferring '%s' block (SLMP スキャナで '%s' ブロックの転送に失敗しました)	通信エラー。データブロックの転送に失敗しました。s==ブロック名

DataMan アプリケーション開発

DataMan Control Commands (DMCC) は、COMポートまたはEthernet接続を経由して直接またはカスタムアプリケーションを使用してプログラマ的にDataMan リーダを設定し、制御するメソッドです。

DMCC コマンドの全リストを参照するには、Windows の [スタート] メニューをクリックして [Cognex] → [DataMan Setup Tool v x.x] → [ドキュメント] → [コマンドリファレンス] とクリックします。あるいは、Setup Tool のヘルプメニューからコマンドリファレンスを開くことができます。

DMCC の概要

DataMan Control Commands (DMCC) は、COMポートまたはEthernet接続を経由して直接またはカスタムアプリケーションを使用してプログラマ的にリーダーを設定し、制御するメソッドです。ご使用中の DataMan リーダに応じて、COM ポート接続は RS232 または USB、Ethernet 対応リーダーの場合は Telnet プロトコルを使用します。デフォルトでは、Ethernet 対応リーダーは TCP ポート番号 23 を経由して通信するように設定されますが、必要に応じて、DataMan Setup Tool を使用して異なるポート番号を割り当てることができます。

Windows に付属する通信アプリケーションである HyperTerminal を使用して、ご使用中のリーダーに接続できるかどうかをテストすることを推奨します。

コマンド構文

すべての DMCC コマンドは ASCII 印刷可能文字のストリームの形式となっており、次の構文を使用します。

コマンドヘッダ コマンド[引数] フッタ

例:

```
||>trigger on\CR\LF
```

コマンドヘッダ構文

```
||checksum:command-id>
```

オプションはすべてコロンで区切られた ASCII テキストです。ヘッダ-オプションブロックのないヘッダは、デフォルトのヘッダを使用します。

checksum

0: チェックサムなし (デフォルト)

1: フッタの前の最後のバイトは、XOR のバイト

command-id

応答で報告される整数コマンドシーケンスです。

ヘッダの例

例	説明
>	デフォルトのヘッダ
0:123>	チェックサムがなく、123 の ID を示すヘッダ
1>	command および data の後にチェックサムを示すヘッダ

コマンド

キーボードで入力可能な ASCII 文字列のコマンドの後に、データが続く場合があります。すべてのコマンド名およびパブリックパラメータデータは、大文字と小文字を区別しません。ヘッダ-フッタブロック内では、コマンドを 1 つだけ発行できます。コマンド、パラメータ、および引数は、スペース文字で区切られます。

コマンド

アクションを示す短縮名です。通常使用されるコマンドは、GET または SET の後にパラメータと値が続きます。

パラメータ

デバイス設定を指定する短縮名です。パラメータ名は、1 つのレベルの構造編成でピリオド('.') で区切られた類似のコマンドのグループで構成されています。

引数

Boolean: ON または OFF

Integer: 123456

String: 引用符 (") で囲まれた ASCII テキスト文字列です。文字列の内容が関数に渡されて、文字列を最終的なフォーマットに変換します。バックスラッシュでエスケープする必要のある文字は、引用符 (¥"), バックスラッシュ (¥¥), パイプ (¥|), タブ (¥t), CR (¥r), LF (¥n) です。

フッタ

フッタは、復帰改行です (¥CR¥LF または ¥r¥n として知られています)。

リーダの応答

リーダは、複数の応答フォーマットのいずれか 1 つを持ちます。応答フォーマットの選択は、SET COM.DMCC-RESPONSE コマンドを使用して設定します。

サイレント: (0、デフォルト) リーダから応答は送信されません。無効なコマンドは、フィードバックなしで無視されます。コマンド応答は、ヘッダもフッタもなく間をスペースで区切った ASCII テキストで送信されます。

拡張: (1) このリーダの応答には、コマンドフォーマットに類似する「ヘッダ データ フッタ」ブロックが含まれます。

注: リーダは DMCC コマンドのストリームを処理できますが、通常は、応答を待機するか、または連続するコマンド間に遅延を挿入すると、信頼性が高まります。

```
||checksum:command-id[status]
```

checksum

応答は、リーダに送信されたコマンドと同じチェックサムフォーマットを使用します。

0: チェックサムなし

1: フッタの前の最後のバイトは、バイトの XOR

command-id

リーダに送信された *command-id* は、応答ヘッダで返されます。

status

ASCII テキストフォーマットでの整数です。

0: エラーなし

1: リーダで文字列の読み取りを開始

100: 未確認のエラー

101: コマンドが無効

102: パラメータが無効

103: チェックサムに誤りがある

104: リーダの状態が原因でパラメータが拒否/変更された

105: リーダを使用できない (オフライン)

例

コマンド	サイレント応答	拡張応答	説明
>GET SYMBOL.DATAMATRIX\r\n	ON	[0]ON\r\n	DataMatrix シンボルが利用可能か。
>SET SYMBOL.DATAMATRIX ON\r\n	応答なし	[0]\r\n	DataMatrix シンボルを有効にします。
>TRIGGER ON\r\n	デコードしたデータ または読み取り失 敗応答	[0]\r\n [1] デコードし たデータまたはbase64形式の 読み取り失敗応答	トリガコマンド

DMCC アプリケーション開発

より大規模なオートメーションシステムにリーダを組み込むために、アプリケーションプログラミングインタフェースとして DMCC を使用することができます。

DataMan SDK (以下、SDK と表します) を使用することもできます。以下の節では、SDK のインストール、そのコンテンツ、SDK サンプルアプリケーションの構築、および SDK に備えられているユーティリティソースコードに関する詳細な情報を提供します。

注: 独自のアプリケーションをゼロから作成し、シリアルポートを経由して DataMan リーダと通信するには、ポートが SerialPort クラスのインスタンスである場合は、port.DtrEnable = true と設定していることを確認してください。

DataMan SDK コンテンツ

DataMan SDK は、SDK バイナリファイルとそのドキュメント、そしてヘルパユーティリティとサンプルアプリケーションのソースコードで構成されています。

バイナリファイルは 2 つのプラットフォームで使用可能です。1 つは Microsoft .Net (PC) 用、もう 1 つは Microsoft .Net Compact Framework (CF) 用です。各ファイルの名前は、それが属するプラットフォーム (PC/CF) に対応します。各プラットフォームには 2 つのコンポーネントがあり、1 つは DataMan SDK コア自体 (Cognex.DataMan.SDK)、もう 1 つは SDK で使用可能なデバイスを検出するためのもの (Cognex.DataMan.Discovery) です。

ソースプログラムは、完全な Microsoft Visual Studio プロジェクトの形式で提供されます。SDK サンプルアプリケーションをビルドするには、Microsoft Visual Studio でサンプルコードのソリューションを開き、[ソリューションのビルド] を選択します。

SDK の使用

代表的な DataMan SDK アプリケーションでの通常の手順

1. デバイスを検出します (前もってデバイスのアドレスがわかっている場合は省略できます)。
2. 必要とするイベントをサブスクライブします (例: 結果文字列到着イベント)。
3. デバイスに接続します。
4. DMCC コマンドをデバイスに送信します (例: trigger)。
5. 受信した結果データを処理します (例: 結果文字列の表示)。

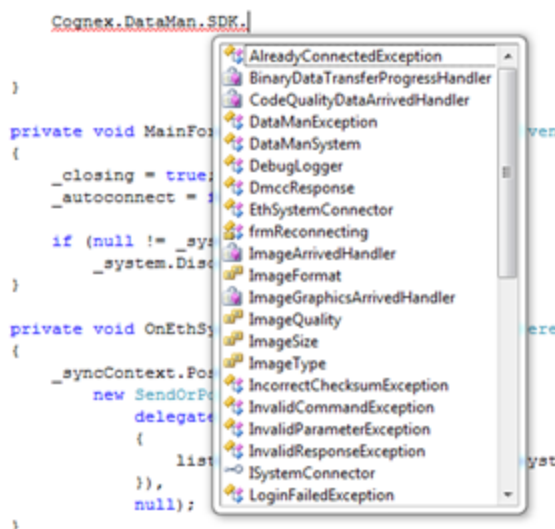
DataMan SDK ライブラリのアクセス

SDK を独自の目的に使用するには、次の手順を実行します。

1. Microsoft Visual Studio で [Create Solution/Project (ソリューション/プロジェクトの作成)] をクリックします。
2. [プロジェクト] 領域の [参照] を右クリックして [参照の追加] を選択します。
3. ポップアップウィンドウで [参照] タブをクリックし、バイナリファイルをインストールまたはコピーしたディレクトリ内で Cognex.DataMan.SDK.*.dll ファイルを検索します (ここで * は作業中のプラットフォーム、PC または CF です)。
4. コードの先頭に次の行を追加することができます。

```
using Cognex.DataMan.SDK;
```

この名前空間で SDK に属する各種要素を検索します。次の画像に示されるように、IntelliSense に表示されます。



DataMan デバイスのリスト化

すでに SDK を使用しているプロジェクトでは、次の追加の手順が必要です。

1. [プロジェクト] 領域の [参照] を右クリックして [参照の追加] を選択します。
2. ポップアップウィンドウで [参照] タブをクリックし、バイナリファイルをインストールまたはコピーしたディレクトリ内で Cognex.DataMan.Discovery.*.dll ファイルを検索します (ここで * は作業中のプラットフォーム、PC または CF です)。
3. コードの先頭に次の行を追加します。

```
using Cognex.DataMan.Discovery;
```

これらの名前空間で SDK に属する各種要素を検索します。これらは IntelliSense に表示されます。

ここで、デバイスの検出に Ethernet またはシリアル通信 (RS232/USB) のいずれか、あるいは両方を選択することもできます。

4. Ethernet 経由でデバイスを検出します。
 - a. 新しい EthSystemDiscoverer を作成します。

```
EthSystemDiscoverer ethSystemDiscoverer = new EthSystemDiscoverer();
```

- b. その SystemDiscovered イベントをサブスクライブします。

```
ethSystemDiscoverer.SystemDiscovered += new
EthSystemDiscoverer.SystemDiscoveredHandler (OnEthSystemDiscovered);
```

- c. EthSystemDiscoverer.SystemDiscoveredHandler 型のイベントハンドラを作成します。
 - d. イベントハンドラの引数は、EthSystemDiscoverer.SystemInfo です。これらの SystemInfo オブジェクトは、リーダとの接続に必要な情報を含んでいます。これらの SystemInfo オブジェクトを自身のコレクションに格納することができます。
 - e. デバイスの検出を開始するには、ethSystemDiscoverer.Discover() メソッドを呼び出します。
5. シリアル通信 (RS232/USB) 経由でのデバイスを検出します。
- a. 新しい SerSystemDiscoverer を作成します。

```
SerSystemDiscoverer serSystemDiscoverer = new SerSystemDiscoverer();
```

- b. その SystemDiscovered イベントにサブスクライブします。

```
serSystemDiscoverer.SystemDiscovered += new
SerSystemDiscoverer.SystemDiscoveredHandler (OnSerSystemDiscovered);
```

- c. SerSystemDiscoverer.SystemDiscoveredHandler 型のイベントハンドラを作成します。
- d. イベントハンドラの引数は、SerSystemDiscoverer.SystemInfo です。これらの SystemInfo オブジェクトは、リーダとの接続に必要な情報を含んでいます。これらの SystemInfo オブジェクトを自身のコレクションに格納することができます。
- e. デバイスの検出を開始するには、serSystemDiscoverer.Discover() メソッドを呼び出します。

① 注： デバイスが検出されるたびに SystemDiscovered イベントが発行されます (デバイスの起動後、または Discover() メソッドに応答した場合)。

イベントへのサブスクライブ

お使いのアプリケーションで結果のようなイベントに反応する場合、関連するイベントにサブスクライブする必要があります。通信状態の変化に関連するイベントもいくつかあります。

次の例では、文字列の読み取りおよび画像到着のイベントをサブスクライブします。

```
mySystem.XmlResultArrived += new XmlResultArrivedHandler (OnXmlResultArrived);
mySystem.ImageArrived += new ImageArrivedHandler (OnImageArrived);
```

① 注： 結果コンポーネントの順序は常に同じとは限らないので、コンポーネントの同期が重要である場合は、DataManUtils コンポーネント経由で提供される ResultCollector ユーティリティクラスを使用します (詳細については、「[ヘルプユーティリティ](#)」の節を参照してください)。

DataMan デバイスへの接続

Ethernet デバイス

次の手順を実行して Ethernet デバイスに接続します。

1. お使いのデバイスとのコネクタを作成します。

```
EthSystemConnector myConn = new EthSystemConnector(deviceIP);
```

ここで deviceIP は既知の IP アドレスまたは EthSystemDiscoverer で検出された IP アドレスです。

2. ユーザ名とパスワードを指定します。

```
myConn.UserName = "admin";
myConn.Password = "パスワードもしくは空白";
```

3. 作成したコネクタを含めて新しい DataManSystem インスタンスを作成します。

```
DataManSystem mySystem = new DataManSystem(myConn);
```

4. DataManSystem インスタンスの Connect() メソッドを呼び出します。

```
mySystem.Connect();
```

5. 必要に応じて接続されているかどうかを確認します。

```
if (mySystem.IsConnected)
```

6. 接続を切断するには、次を呼び出します。

```
mySystem.Disconnect();
```

① 注：現在すべてのデバイスでユーザ名 admin を使用しています。パスワードが必要な場合は、空の文字列を使用することができます。

シリアルデバイス

次の手順を実行してシリアルデバイスに接続します。

1. お使いのデバイスとのコネクタを作成します。

```
SerSystemConnector myConn = new SerSystemConnector(PortName, Baudrate);
```

ここで PortName および Baudrate は、既知のシリアル接続パラメータか、または SerSystemDiscoverer によって指定されます。

2. 作成したコネクタを含めて新しい DataManSystem インスタンスを作成します。

```
DataManSystem mySystem = new DataManSystem(myConn);
```

3. DataManSystem インスタンスの Connect() メソッドを呼び出します。

```
mySystem.Connect();
```

4. 必要に応じて接続されているかどうかを確認します。

```
if (mySystem.IsConnected)
```

5. 接続を切断するには、次を呼び出します。

```
mySystem.Disconnect();
```

DataMan デバイスへの DMCC コマンドの送信

各種コマンドをリーダーに送信するには、SendCommand () を使用します。使用可能なコマンドについては、『DMCC コマンドリファレンス』を参照してください。

SendCommand () には必須パラメータが 1 つあり、それはコマンド文字列そのものです。これには 2 つのオプションパラメータもあります。1 つはコマンドのデフォルトのタイムアウトを無効にし、もう 1 はコマンドを使用して追加のバイトを渡します。

次は、DMCC コマンドを送信する例です。

```
DmccResponse response = mySystem.SendCommand("GET_DEVICE.TYPE");
```

① 注：応答の内容は、応答オブジェクトの PayLoad プロパティに含まれています。コマンド文字列に DMCC ヘッダまたはフッタが指定されていないことにも注意してください。

SendCommand() または GetLiveImage() などの関数は、非同期実装も含んでいます。これらを使用する場合は、Begin/End プレフィックスの付いた適切な関数名を検索します。これらの関数は二つ一組で使用されます。Begin プレフィックスの付いた関数は、End プレフィックスの付いた関数で使用する可能な IAsyncResult を返します。

DataMan デバイスからの静止画像とライブ画像の表示

静止画像を表示するには、DataManSystem.GetLastReadImage () を使用するか、または ImageArrived イベントにサブスクライブして画像を取得します。

ライブ画像を表示するには、次の手順を実行します。

1. リーダをライブ表示モードに設定します。

```
mySystem.SendCommand("SET LIVEIMG.MODE 2");
```

2. 次を使用して、画像用にデバイスを定期的にポーリングします。

```
mySystem.GetLiveImage(ImageFormat, ImageSize, ImageQuality);
```

サンプルアプリケーションのソースで、実装例を参照してください。その例では、GUI のロックを防止する新しいポーリングスレッドが作成されています。

ライブ表示モードをオフにするには、次を使用します。

```
mySystem.SendCommand("SET LIVEIMG.MODE 0")
```

ヘルパユーティリティ

幾つかのヘルパ関数が DataManUtils と呼ばれるプロジェクトの SDK にソースコードとして用意されています。主な機能を下記で説明します。

Gui

結果画像を指定したコントロールにフィットさせたり、ビットマップデータとバイト配列を相互に変換するなどの画像操作関数を提供します。

追加のクラスは、画像解析および SVG レンダリングのための SVG ヘルパを提供します。SVG フォーマットの結果コンポーネントは、コードが検出された画像の領域をマークするのに使われます。

ResultCollector

結果コンポーネントの順序は常に同じとは限りません。たとえば、XML の結果が先に到着するときはあれば、画像が先に到着するときもあります。この問題は、ResultCollector を使用して解決することができます。

ユーザは、結果の構成要素 (たとえば、画像、SVG グラフィックス、および xml 読み取り結果) を指定し、ResultCollector の ComplexResultArrived イベントにサブスクライブする必要があります。

ResultCollector は結果コンポーネントを待機します。結果が完成すると、ComplexResultArrived イベントが発行されます。結果が完成していないが、タイムアウトした (タイムアウト値は ResultTimeOut プロパティで設定できます)、または ResultCollector のバッファがいっぱいの場合 (バッファ長は ResultCacheLength プロパティで設定できます)、PartialResultDropped イベントが発行されます。どちらのイベントもそのイベント引数で使用可能な結果コンポーネントを提供し、これを使用して複雑な結果を処理することができます (たとえば、結果履歴の保持や、画像、グラフィックスおよび結果文字列の表示など)。

DmccEscaper

DMCC コマンド文字列をエスケープするか、エスケープされないようにするために使用できます。

FileLogger

開発中に、シンプルなロガークラスを使用することができます。ここで提供するその他すべてのユーティリティと同様に、PC および CF 両方のプラットフォーム上で機能します。

ヘルパユーティリティの使用

ヘルパユーティリティは 2 つのプロジェクトに含まれています。両方のプロジェクトが同じソースプログラムを参照しますが、1 つは Microsoft .Net Compact Framework (DataManUtilsCF) 用に作成されたもので、もう 1 つは PC の Microsoft .Net

Framework (DataManUtilsPC) 用です。これらのユーティリティに備わっている機能を使用するには、適切な DataManUtils プロジェクトをソリューションに含め、機能を使用するプロジェクト内で参照します。

スクリプトベースのデータフォーマット

DataMan Setup Tool ではさまざまなデータフォーマットを組み合わせ使用でき、リーダに出力チャンネルで多様なアクション (ピープ音、LED 点滅、出力 1 のプルアップなど) を実行させることができます。

スクリプトベースのフォーマットには主に次の 2 つの利点があります。

- 柔軟な結果設定
- 結果を返す前のリーダイベントの設定

注： スクリプトベースフォーマットにより、ユーザが実行できる操作は 2 つのカスタムイベントの実行とシステムイベントの上書きに制限されます。

DMCC のサポート

スクリプトベースフォーマットには、次の DMCC コマンドを使用できます。

コマンド	範囲	説明
GET/SET FORMAT.MODE	[0..1]	フォーマットモードを選択します。 • 0 = 基本フォーマット • 1 = スクリプトベースフォーマット
SCRIPT.LOAD	length	ホストからリーダにフォーマットスクリプトをロードします。
SCRIPT.SEND	-	リーダからホストにフォーマットスクリプトを送信します。

スクリプト関数を使用して、スクリプトパーサに DMCC のサポートを追加できます。DMCC 実装はスクリプト関数にラッピングされます。スクリプト関数はグローバルスコープで定義されています。DMCC 関数は次の 3 つのカテゴリに分割できます。

- コマンド (ピープ音の発行、再起動など)
- プロパティのセッタ関数
- プロパティのゲッタ関数

関数はスクリプトエンジンの可変引数機能を使用します。関数の引数型が DMCC コマンドで定義されている正しい型と比較されます。引数の数または引数の型が正しくない場合、エラーステータスが返されます。

関数はそのステータスを表すプロパティとともにオブジェクトを返します。コマンドによって応答が返された場合、応答プロパティによってアクセスできます。ステータスコードは DMCC コマンドのものと同じです。

関数が正常に実行されると、ステータス値ゼロが返されます。スクリプト例外は使用されません。

注：

- データフォーマットスクリプト関数は、出力遅延時間または距離の経過後に実行されます。

① すべてのスクリプト関数は独立したスレッドで実行され、実行は相互に排他的です。あるスクリプト関数が別のスクリプト関数によって中断されることはありません。

- スクリプトでは、[ON|OFF] の代わりに [0,1] または [true,false] を使用します。

DMCC 実装に基づき、複数値応答であっても、次のように常に 1 つの文字列として応答が返されます。

```
var foo = dmccGet("DECODER.ROI");
```

set コマンドでは、次のように複数の適切な型のパラメータがサポートされます。

```
dmccSet("DECODER.ROI", 16, 1280, 16, 1024);
```

例

次の例では、dmccSet 関数を使用して、ビーブ信号の発行、画像保管用の FTP サーバの IP アドレスの設定、出力応答への MAC アドレスの追加を行います。

```
function onResult (decodeResults, readerProperties, output)
{
    var myoutput;
    var result_tmp = dmccCommand("BEEP", 1, 1);

    result_tmp = dmccSet("FTP-IMAGE.IP-ADDRESS", "192.168.23.42");
    if(result_tmp.status !=0)
    {
        throw("FATAL: failed to set the ftp server address");
    }
    var mac = dmccGet("DEVICE.MAC-ADDRESS");

    myoutput = 'Result="' + decodeResults[0].content + '"', MAC='+mac.response;
    output.content = myoutput;
}
```

IP アドレス用の DMCC の set コマンドが失敗した場合、ゼロ以外のステータスが返され、スクリプト例外がスローされて DataMan Setup Tool により報告されます。

注： 上記の例のように Throw() コマンドを使用すると、異常な状況 (例外) の発生を報告するために Setup Tool のエラーログにエラーが表示されます。エラーログにアクセスするには、Setup Tool のメニューバーで [システム] → [デバイスログの表示] の順にクリックします。

補助関数

次のグローバル補助関数も使用可能です。

- エスケープシーケンスをデコードするための関数
- 文字列引数を base64 にエンコードするための関数

decode_sequences 関数

このグローバル関数は、エスケープシーケンスをデコードするために使用します。この関数は、デコードされたエスケープシーケンスを含む文字列を返します。戻り値を使用して、HID 接続で送信される結果にキーボードコントロールコマンドを追加できます。

パラメータ	型	説明
encodedString	string	キーボードエスケープシーケンスを含む文字列値。

Alt キー、Ctrl キー、または Shift キーの組み合わせをシミュレートするために、次の 4 つのエスケープシーケンスを使用できます。

- **¥ALT** - <ALT キー> シーケンス
- **¥CTRL** - <CTRL キー> シーケンス
- **¥SHIFT** - <SHIFT キー> シーケンス
- **¥K** - 特殊キー

注： バックスラッシュに続くキーは必ず大文字にしてください。小文字では特殊キーの組み合わせとして認識されません。

対応しているキーシーケンスおよびキー

次のリストに、現在サポートしているキーまたはキーの組み合わせを示します。

- ALT-A ~ ALT-Z
- CTRL-A ~ CTRL-Z
- CTRL-F1 ~ CTRL-F12
- SHIFT-F1 ~ SHIFT-F12
- F1 ~ F12
- ALT-F1 ~ ALT-F12
- PageUp, PageDown, Home, End, 矢印キー (上、下、左、右)、Insert, Delete, Backspace, Tab, Esc, Print Screen, GUI (左、右) キー
- これらのエスケープシーケンスは次のとおりです。
 - PageUp -> ¥KPup;
 - PageDown -> ¥KPdn;
 - Home -> ¥KHome;
 - End -> ¥KEnd;
 - 上矢印 -> ¥KUar;
 - 下矢印 -> ¥KDar;
 - 左矢印 -> ¥KLar;
 - 右矢印 -> ¥KRar;
 - Insert -> ¥KIns;
 - Delete -> ¥KDel;
 - Backspace -> ¥KBksp;
 - Tab -> ¥KTab;
 - Esc -> ¥KEsc;
 - Print Screen -> ¥KPrtScr;
 - Left GUI -> ¥KLGui;
 - Right GUI -> ¥KRGui;

例

先頭または末尾に Ctrl-B キーボードコントロールコマンドを追加するには次の例が使用できます。

```
var ctrl_b = decode_sequences("\\Ctrl-B;");

function onResult (decodeResults, readerProperties, output)
{
    if (decodeResults[0].decoded)
    {
        output.content = ctrl_b+decodeResults[0].content+ctrl_b;
    }
}
```

注：入力文字列では、エスケープシーケンスを開始するためのバックスラッシュもエスケープする必要があります。シーケンス、そしてシーケンスと同じプレフィックスを区別するために、文末のセミコロンは必ず付けてください。セミコロンが欠けていると、例えば ¥KF11 が「F11 キーを押す」か「F1 キーに続いて 1 を押す」かが検知できず、シーケンスが勝手に解釈されてしまいます。誤った、または不十分なシーケンスが使用されると、エスケープシーケンスを示す 2 文字は無視されます。この場合はエスケープシーケンスの最初の 2 文字が飛ばされて、残りの文字が送信されます。例えば "¥ALT-M11;" という無効なシーケンスの結果表示は "LT-M11;" となります。

encode_base64 関数

このグローバル関数は、文字列引数を base64 にエンコードするために使用します。エンコード結果は文字列オブジェクトとして返されます。

パラメータ	型	説明
inputString	string	base64 にエンコードする入力文字列。

エラー管理

スクリプトがロードされた、またはコード解析関数が呼び出された場合、スクリプトエラーが発生することがあります。これらのエラーは次の場所に表示されます。

- デバイスログ
- [スクリプトベースフォーマット] ウィンドウのエラーボックス

スクリプトフォーマット

スクリプトベースフォーマットが有効になっている場合、ユーザ定義の JavaScript モジュールによってデータフォーマットが実行されます。解析関数はデフォルトで onResult になっており、DecodeResult オブジェクトの配列、ReaderProperties オブジェクト (トリガモード、統計)、Output オブジェクトの 3 つのオブジェクトを引数として呼び出されます。シングルコードおよびマルチコードで、その結果のエントリポイントはこの 1 つだけです。

クラス階層の構造は次のとおりです。

```
function onResult [decodeResults, readerProperties, output]
```

```
DecodeResult
SymbologyProperties
  Point
ValidationResult
  GS1Validation
  DoDValidation
QualityMetrics
Metric
ReaderProperties
  Trigger
  Statistics
```

```
Output
Event
```

下記に示す関連オブジェクトの詳細な説明を参照してください。

onResult 関数

デコードイベントのイベントハンドラであり、ゼロまたは 1 つ以上のデコード結果が生成されます。

プロパティ	型	説明
decodeResults	DecodeResult[]	入力。DecodeResult オブジェクトの配列。1 つのデコード結果にそのデコード試行に関するすべての情報が含まれます。
readerProperties	ReaderProperties	入力。実際のデコード結果に関連付けられていないリーダのプロパティ。
output	Output	出力。出力文字列の変更またはイベント発生のために更新が必要なオブジェクト。

onGenerateFTPFilename 関数

この関数を使用して、FTP サーバに送信されるファイルの名前を生成できます。

プロパティ	型	説明
decodeResults	DecodeResult[]	入力。DecodeResult オブジェクトの配列。1 つのデコード結果にそのデコード試行に関するすべての情報が含まれます。
readerProperties	ReaderProperties	入力。実際のデコード結果に関連付けられていないリーダのプロパティ。
output	Output	出力。出力文字列の変更またはイベント発生のために更新が必要なオブジェクト。

アップロードされる画像ファイル名は、スクリプトの文字列戻り値から取得されます。次に例を示します。

```
function onGenerateFTPFilename(decodeResults, readerProperties, output)
{
    var ftp_filename = readerProperties.name + "-";
    ftp_filename += readerProperties.trigger.index + "-" + decodeResults
    [0].image.index;
    return ftp_filename;
}
function onGenerateFTPPCMReportFilename(decodeResults, readerProperties, output)
{
    var ftp_filename = readerProperties.name + "-";
    ftp_filename += readerProperties.trigger.index + "-" + decodeResults
    [0].image.index;
    return ftp_filename;
}
```

DecodeResult オブジェクト

次の表は、DecodeResult オブジェクトとその型、プロパティの詳細を示しています。

デコード結果

1 つのデコード結果の詳細を示します。

プロパティ	型	説明
decoded	boolean	デコードが成功した場合は True です。
content	string	未処理の読み取り結果。

decodeTime	integer	デコード時間 (単位: ミリ秒)。
triggerTime	integer	トリガ時間 (単位: ミリ秒)。
timeout	integer	トリガタイムアウト (単位: ミリ秒)。
symbology	SymbologyProperties	このプロパティの値は、下記のシンボルプロパティの表にあります。
image	ImageProperties	このプロパティ (別名、取り込み属性) の値は、下記の画像プロパティの表にあります。
validation	ValidationResult	このプロパティの値は、下記のバリデーション結果の表にあります。
metrics	QualityMetrics	このプロパティの値は、下記の QualityMetrics の表にあります。
readSetup	integer	使用されたReadSetupのインデックス。
source	string	画像をデコードするデバイスの名前。
annotation	string	マスタースレーブのトリガに関する結果の注記。
label	string	シンボルのラベル。

シンボルプロパティ

デコード結果を表すシンボルのプロパティです。

プロパティ	型	説明
name	string	シンボルの名前。
id	string	シンボル識別子 (ISO15424 による)。
quality	integer	コード全体の品質測定。範囲は [0, 100]。リード・ソロモン符号によるエラー訂正を使用するシンボル (例: DataMatrix、QR コード、AztecCode、MaxiCode、DotCode、PDF417、4-state のポスタルコード) では、UEC (エラー訂正不使用率) が報告されます。リニアシンボルでは、全体のコード品質が示されます。数字が大きいほど高品質です。
moduleSize	float	モジュールサイズ(単位はモジュールあたりのピクセル数、ppm)。
size	Point	列×行のシンボルのサイズ。シンボルに該当しない場合、値は -1 に設定されます。
corners	Point の配列	四つのコーナーの座標を指定します。Point プロパティ型の詳細は下記の Point の表にあります。 コーナーの座標は次の順に返されます。 非鏡像シンボルの場合 - コーナー 0: シンボルの左上 - コーナー 1: シンボルの右上 - コーナー 2: シンボルの左下 - コーナー 3: シンボルの右下 非鏡像 DataMatrix および Vericode を除き、コーナー 0 は、隣り合う 2 辺のセルの実線が交わり、コーナー 1 ~ 3 が反時計回りに続きます。 鏡像のシンボルの場合、コーナーはそれに対応してミラーリングされます。
center	Point	中央の座標を指定します。Point プロパティ型の詳細は下記の Point の表にあります。
angle	float	コードの向き (度)。

PtpTimeStamp

画像取り込みのピアツーピアのタイムスタンプ値。

プロパティ	型	説明
-------	---	----

s	integer	画像取り込みタイムスタンプの秒プロパティ。
ns	integer	画像取り込みタイムスタンプのナノ秒プロパティ。

Point

Point は、2 次元平面上で点を定義する整数 x および y 座標の順序対です。

プロパティ	型	説明
x	integer	x 座標
y	integer	y 座標

ImageProperties オブジェクト

次の表は、ImageProperties オブジェクトとその型、プロパティの詳細を示しています。

画像のプロパティ

取り込んだ画像のプロパティです。

プロパティ	型	説明
index	integer	トリガ内の画像のインデックス。
FoV	Rect	視野。センサの左上隅を基準とする画像センサの使用領域。Rect プロパティ型の詳細は下記の Rect の表にあります。
RoI	Rect	対象領域。センサを基準として実際に使用する視野の一部。Rect プロパティ型の詳細は下記の Rect の表にあります。
exposureTime	integer	露光時間 (単位: マイクロ秒)。
gain	integer	カメラのゲイン。
autoExposure	boolean	自動露光を使用する場合は True です。
illEnabled	boolean	内部照明が有効な場合は True です。
illIntensity	integer	内部照明の明るさ。
extillEnabled	boolean	外部照明が有効な場合は True です。
extillIntensity	integer	外部照明の明るさ。
targetBrightness	integer	自動露光の場合の目標となる明るさ。
focusLength	integer	フォーカス値 (単位: ミリメートル)。該当しない場合は 0 です。
setupIndex	integer	リードセットアップ (読み取り設定) の現在のインデックス。
inputStates	boolean の配列	トリガが起動されたときの入力ラインの状態。
filterTime	integer	フィルタリングの継続時間 (単位: ミリ秒)。
creationTime	integer	作成時刻。
creationTicks	integer	画像作成時刻に対応するエンコーダのティック。
ptpTimeStamp	ptpTimeStamp	PtP 画像取り込みのタイムスタンプ。
id	integer	この画像の数値識別子。

Rect

Rect は矩形の幅、高さ、位置を記述します。

プロパティ	型	説明
top	integer	センサの左上隅を基準として top 値を指定します。
bottom	integer	センサの左上隅を基準として bottom 値を指定します。
left	integer	センサの左上隅を基準として left 値を指定します。
right	integer	センサの左上隅を基準として right 値を指定します。

ValidationResult オブジェクト

次の表は、ValidationResult オブジェクトとその型、プロパティの詳細を示しています。

ValidationResult

バリデーションのすべての詳細を示します。

プロパティ	型	説明
state	integer	バリデーションの状態は次のとおりです。 - notTried - fail - pass このプロパティのフォーマットは“validation.state.notTried”です。
method	integer	バリデーションの方法は次のとおりです。 - none - gs1 - iso - dod_uid - pattern - matchString このプロパティのフォーマットは“validation.method.none”です。
matchString	string	このプロパティは設定済みの一致文字列とともに返されます。一致文字列バリデーションを有効にする必要があります。
failurePos	integer	バリデーションエラーの位置。
failureCode	integer	バリデーションのエラーコード。
failureMsg	string	バリデーションエラーの原因を説明するエラーメッセージ。
gs1	GS1 Validation	GS1 Validation プロパティ型の詳細は下記の GS1 Validation の表にあります。
dod_uid	DoD Validation	DoD Validation プロパティ型の詳細は下記の DoD Validation の表にあります。

GS1Validation

GS1 Validation の詳細です。

プロパティ	型	説明
AI00	string	出荷梱包シリアル番号 (Serial Shipping Container Code)。
AI01	string	固定の商品識別コード (Global Trade Item Number)
AI01	string	可変の商品識別コード (GTIN)
AI01	string	POS でスキャンされる可変の商品識別コード (GTIN)
AI01	string	POS でスキャンされない可変の商品識別コード (GTIN)
AI02	string	物流単位に含まれる、固定の商品識別コード
AI02	string	物流単位に含まれる、可変の商品識別コード
AI10	string	バッチまたはロット番号
AI11	string	製造年月日
AI12	string	支払期限日

AI13	string	包装年月日
AI15	string	品質保持期限日(賞味期限日)
AI16	string	販売期限日
AI17	string	有効期限日
AI20	string	リニューアル商品／規格変更品識別番号
AI21	string	シリアル番号
AI240	string	製造元が割り当てた追加の商品識別番号
AI241	string	顧客製品番号
AI242	string	カスタム製品番号
AI243	string	包装のコンポーネント識別番号
AI250	string	部品参照番号(二次的なシリアル番号)
AI251	string	元の商品／原材料参照番号
AI253	string	グローバル文書識別番号 (GDTI)
AI254	string	GLN 拡張フィールド
AI255	string	クーポン識別番号 (GCN)
AI30	string	数量
AI31nn AI32nn AI35nn AI36nn	string	取引方法
AI33nn AI34nn AI35nn AI36nn	string	輸送方法
AI337n	string	平方メートルあたり重量 (Kg)
AI37	string	物流単位の入り数
AI390n	string	支払金額またはクーポン金額 - 単一通貨地域
AI391n	string	支払金額と ISO 通貨コード
AI392n	string	計量商品金額 - 単一通貨地域
AI393n	string	計量商品金額 - ISO 通貨コード
AI394n	string	クーポンの割引率
AI400	string	顧客発注番号
AI401	string	積荷番号 (GINC)
AI402	string	出荷識別番号 (GSIN)
AI403	string	追跡コード
AI410	string	グローバル・ロケーション・ナンバー (GLN)、出荷先(納品先)コード
AI411	string	グローバル・ロケーション・ナンバー (GLN)、請求先コード
AI412	string	グローバル・ロケーション・ナンバー (GLN)、商品仕入先企業コード
AI413	string	グローバル・ロケーション・ナンバー (GLN)、最終納品場所コード
AI414	string	グローバル・ロケーション・ナンバー (GLN)、物理的位置のコード
AI415	string	グローバル・ロケーション・ナンバー (GLN)、請求書発行者コード
AI420	string	宛先の郵便番号 (単一郵政組織内)
AI421	string	宛先の郵便番号 (3 桁の ISO 国コード)
AI422	string	取引商品の原産国

AI423	string	一次加工国
AI424	string	加工国
AI425	string	分解・解体国
AI426	string	商品の一貫加工国コード
AI427	string	原産地で国より一つ下の単位を表すコード
AI7001	string	NATO ストック番号 (NSN)
AI7002	string	UN/ECE 枝肉番号およびカット規格コード
AI7003	string	有効期限(日時)
AI7004	string	期待値(アクティブポテンシー)
AI7005	string	漁獲海域
AI7006	string	最初の冷凍日
AI7007	string	収穫日
AI7008	string	漁業目的用の魚種
AI7009	string	漁具タイプ
AI7010	string	生産方法
AI703s	string	加工者番号と 3 桁の ISO 国コード
AI710 AI711 AI712 AI713	string	国家保険償還番号 (NHRN):
AI8001	string	ロール状商品の情報 - 幅、長さ、芯の直径、方向、継ぎ目
AI8002	string	携帯電話識別子
AI8003	string	リターナブル資産識別コード (GRAI)
AI8004	string	資産管理識別コード (GIAI)
AI8005	string	計測単位あたりの価格
AI8006	string	商品構成部品の管理番号
AI8007	string	国際銀行口座番号 (IBAN)
AI8008	string	製造日時
AI8010	string	部品／構成品識別番号 (CPID)
AI8011	string	部品／構成品識別シリアル番号
AI8012	string	ソフトウェアバージョン
AI8017 AI8018	string	サービス提供者識別番号 (GSRN)
AI8019	string	サービスインスタンス番号 (SRIN)
AI8020	string	支払伝票番号
AI8110	string	北米用クーポンコード
AI8111	string	クーポンのポイント数
AI8200	string	商品関連情報 URL
AI90	string	貿易相手国間で相互に合意した情報
AI91-99	string	企業の内部使用データ

DoD Validation

DoD Validation の詳細です。

プロパティ	型	説明
enterpriseID	string	企業識別子
serialNum	string	シリアル番号
partNum	string	部品番号
uniqueItemID	string	固有のアイテム識別子
batchNum	string	バッチ番号

QualityMetrics オブジェクト

次の表は、QualityMetrics オブジェクトとその型、プロパティの詳細を示しています。Metric プロパティ型の詳細は下記の Metric の表にあります。次の指標はすべて、DataMan Setup Tool の [シンボル設定] ペインにあるすべての基準で使用できます。

品質測定

すべての計測品質パラメータを記述します。

プロパティ	型	説明
singleScanInt ¹⁾	Metric	シングルスキャンの完全性。raw メンバは -1 に設定されます。シングルスキャンの完全性は、バーコードを 1 回のみスキャンしてバーコードをデコードする際の容易さを示す一般的な計測値です。シンプルなデコーダの処理方法を表すことを目的としています。そのようなアルゴリズムは一般的に高度ではなく、バーコードの複数の位置でシンボルに損傷がある場合はデコード可能性が低下します。singleScanInt 指標はコードの品質全体の計測値であるため、これが低い場合はさまざまな問題が存在している可能性があります。
symbolContrast ¹⁾	Metric	ISO15415 でのシンボルのコントラスト。シンボルのコントラストは、明るいセルと暗いセルの差を計測してグレースケール値で表したものです。コントラストが高いとコードのデコードが容易になり、コントラストが低いと明るいセルと暗いセルの区別が困難なためデコードを適切に行えないことがあります。コントラストが低い場合、照明が不十分である、印刷部分と背景が類似しているためコードの読み取りが困難になる、または印字品質が不十分である可能性があります。
cellContrast ^{2) 3)}	Metric	セルのコントラスト。セルのコントラストは、セルの明るい部分と暗い部分の差を計測してグレースケール値で表したものです。コントラストが高いとコードのデコードが容易になり、コントラストが低いとセルの明るい部分と暗い部分の区別が困難なためデコードを適切に行えないことがあります。コントラストが低い場合、照明が不十分である、またはマークされた領域とマークされていない領域が類似しているためコードの読み取りが困難になる可能性があります。
axialNonUniformity ^{1) 2) 3)}	Metric	軸の歪み度合い。軸の歪み度合いは、各軸に沿ったグリッドセルの間隔の差異を計測したものです。最善のケースでは、この値はゼロになります。これは、グリッドのセルの中央の間隔がすべての方向で均一であることを示します。軸の歪み度合いがひどい場合、コードの印刷プロセスに問題があり、コードが引き伸ばされたり圧縮されたりしている可能性があります。
printGrowth ^{1) 2)}	Metric	印刷太り。印刷太りは、明るいまたは暗い部分がセルをどの程度占めているかを計測したものです。印刷太りが大きい場合はセルが境界を超えていること、印刷太りが小さい場合はセルが使用可能なスペースを完全に占めていないことを示します。いずれの場合も、(印刷太りが大きい場合は隣接セルの読み取りが困難になる、または印刷太りが小さい場合はセルそのものの読み取りが困難になることで) 問題を引き起こします。その結果、印刷太りはゼロに近いことが理想的です。印刷太りが大きいまたは小さいことは通常、コードの印刷プロセスに問題があることを示しています。例えば、刻印されたマークが磨耗してマークが小さくなっていたり、プリンタがラベルにインクを過剰に付着させマークが大きくなったりしている可能性があります。
UEC ^{1) 2) 3)}	Metric	エラー訂正不使用率。エラー訂正不使用率では、コードに印刷されているが使用されていないエラーチェック訂正データの量が計測されます。UEC カウントが大きい場合は、コードを読み取るためにエラー訂正データがほとんど (またはまったく) 不要であることを意味し、良好な状態です。UEC 値が小さい場合は、印刷の不具合、画像処理の不具合、コードの間違い、コードの損傷を示している可能性があります。

プロパティ	型	説明
modulation ^{2) 3)}	Metric	変調。変調では、コードの明るいセルと暗いセルを区別する容易さが計測されます。コントラストとやや類似しており、変調が大きい方が好ましく、変調が小さいと明るいセルと暗いセルの区別が困難になることがあります。変調が小さい場合、照明が不十分である、印刷部分と背景が類似しているためコードの読み取りが困難になる、または印字品質が不十分である可能性があります。
fixedPatternDamage ^{2) 3)}	Metric	固定パターン損傷。固定パターン損傷は、コード外部の固定パターン (実線の検出パターンと交互のクロックパターン) の無損傷の度合いを示す計測値です。固定パターンに損傷がある場合、コードの検出が困難になり、デコードできない可能性があります。固定パターン損傷スコアが不適切である場合は通常、コードに損傷や汚れがある、またはクワイエットゾーンが正しく確保されていないことを示します。
gridNonUniformity ^{2) 3)}	Metric	グリッドの歪み度合い。グリッドの歪み度合いでは、全体のグリッドに基づく最適なセル配置と実際の配置の差異が計測されます。軸の歪み度合いの計測値と同様に、グリッド全体の引き伸ばしまたは圧縮を計測する代わりに、個々のセルの正しい位置からの逸脱の程度が計測されます。グリッドの歪み度合いは通常、印刷プロセスがセルの配置と一致していないことを示します。
extremeReflectance ²⁾	Metric	最大反射率。この指標では、コードの印刷背景の明るさが計測されます。値が大きすぎる場合は照明または画像処理に問題があり、コードが消失して読み取り困難になる可能性があります。値が小さい場合はコードを照らす照明が不足しており、コントラストが不十分で読み取り困難になる可能性があります。最大反射率のグレードが不適切である場合は、ホットスポットなどの照明位置に関する問題を示していることもあります。
reflectMin ¹⁾	Metric	最小反射率。この指標では、バーコードの暗い部分の暗さが計測されます。値が小さい場合はコードの暗い部分が暗いこと、値が大きい場合は暗くないことを示します。値が小さすぎる場合、照明が不十分、または露光時間が短すぎることを示している可能性があります。値が大きすぎる場合、ホットスポット、過剰な照明、露光時間が長すぎることを示している可能性があります。プリンタのインク付着量の不足などの印刷品質の問題が、最小反射率のグレードで示されることもあります。
edgeContrastMin ^{1) 2)}	Metric	エッジの最小コントラストでは、シンボルの最大コントラストに対するエッジの最小コントラストの比率が計測されます。この指標は、バーの損傷 (シンボルにおいてコントラストのばらつきが小さくなる) など、シンボル内のアーティファクトを見つけることを目的としています。グレードが低い場合、光学系の焦点ずれ、不十分な照明、不適切な印刷を示している可能性があります。
multiScanInt ^{*1)}	Metric	マルチスキャンの完全性。マルチスキャンの完全性は、バーコードを複数回のスキャンしてシンボルをデコードする際の容易さを示す一般的な計測値です。この指標では、特定のバーコードに対してどの程度高度なデコーダでデコードを実行するかが計測されます。multiScanInt 指標はコードの品質全体の計測値であるため、これが低い場合はさまざまな問題が存在している可能性があります。

1) これらのコード品質測定は 1D コードで使用できます。

2) これらのコード品質測定は DataMatrix コードおよび QR コードで使用できます。

3) これらのコード品質測定は AIMDPM 検証を実行する検証機で使用できます。

Metric

1 つの計測パラメータの品質を記述します。

プロパティ	型	説明
raw	float	raw 指標
grade	string	品質グレード、グレード範囲は A ~ F (A が最高)

リーダのプロパティ

次の表は、リーダのプロパティの詳細を示しています。

ReaderProperties

実際のデコード結果に関連付けられないリーダのプロパティ。

プロパティ	型	説明
name	string	画像をデコードするデバイスの名前。
trigger	Trigger	Trigger プロパティ型の詳細は下記の Trigger の表にあります。
stats	Statistics	Statistics プロパティ型の詳細は下記の Statistics の表にあります。
inputstr	string	このプロパティは標準フォーマットの <Input String> データフォーマットトークンと同じ機能を果たします。InputString 機能 (DMCC を通じてのみ設定可能) を介してリーダに送信された文字列を保持します。

Trigger

トリガイベント開始の詳細を記述します。

プロパティ	型	説明
type	integer	次のトリガタイプがあります。 • シングル • 自動読み取り • マニュアル • バースト • セルフ • 連続 このプロパティのフォーマットは “trigger.type.single” です。
index	integer	一意のトリガ識別子。
burstLength	integer	バーストトリガ時の画像の数。
interval	integer	トリガ間隔 (単位: マイクロ秒)。
delayType	integer	次のトリガ遅延タイプがあります。 • なし • 時間 • 距離 このプロパティのフォーマットは “trigger.delayType.none” です。
startDelay	integer	トリガ開始遅延 (単位: Trigger.delayTime.time を使用する場合はミリ秒、Trigger.delayTime.distance を使用する場合はミリメートル)。
endDelay	integer	トリガ終了遅延 (単位: Trigger.delayTime.time を使用する場合はミリ秒、Trigger.delayTime.distance を使用する場合はミリメートル)。
creationTime	integer	作成時刻。
creationTicks	integer	トリガ信号時刻に対応するエンコードのティック。
groupIndex	integer	グループをトリガしたリーダの一意のトリガ識別子プロパティ。
endTime	integer	トリガイベント終了時間 (単位: ミリ秒)。
endTicks	integer	トリガ終了イベント時間のエンコードのティックカウンタ。

Statistics

リーダーの動作情報。

プロパティ	型	説明
reads	integer	デコードしたシンボルの合計数。
noReads	integer	トリガを受信したものの、シンボルのデコードが行われなかった合計回数。
triggers	integer	Reads 合計+NoReads合計+missedTriggers によって計算されたトリガの合計数。
bufferOverflows	integer	画像バッファがいっぱいだったためバッファ処理されなかった画像の数。
triggerOverruns	integer	取り込みシステムがビジーだったため失敗したトリガの数。
itemCount	integer	バッファ処理された画像の読み取り無効が許可されている場合の、読み取り無効の数。
passedValidations	integer	データバリデーションに合格した読み取りの数。
failedValidations	integer	データバリデーションに失敗した読み取りの数。

Output

Output はデコード後の結果およびイベントを記述します。個々のプロトコルターゲットに対して異なる結果を指定することが可能です。出力オブジェクトには、string 型のターゲット固有のプロパティがあります。出力プロパティの名前はターゲットプロトコル名と同じです。ターゲット固有の出力が割り当てられていない場合、output.content プロパティで取得したデフォルトの結果になります。

プロパティ	型	説明
content	string	デコード結果として送信される文字列。
events	event	アクティブ化されている出力イベントは次のとおりです。DecodeEvents プロパティ型の詳細は下記の DecodeEvents の表にあります。
SetupTool*	string	デコード結果として Setup Tool に送信される文字列。
Serial*	string	デコード結果としてシリアルおよび USB 接続に送信される文字列。
Telnet*	string	デコード結果として Telnet 接続に送信される文字列。
Keyboard*	string	デコード結果として HID 接続に送信される文字列。5.2 では使用できません。
FTP*	string	デコード結果として FTP 接続に送信される文字列。
PS2*	string	デコード結果として PS2 接続に送信される文字列。5.2 では使用できません。
NetworkClient*	string	デコード結果として NetworkClient 接続に送信される文字列。
IndustrialProtocols*	string	デコード結果として接続先 PLC に送信される文字列。

*これらのプロパティは、output.content プロパティで設定された出力情報を抑制します。

ここに、プロトコル固有のフォーマット機能を示します。

```
function onResult (decodeResults, readerProperties, output)
{
    if (decodeResults[0].decoded)
    {
        var mymsg = decodeResults[0].content;
        // output['Serial'] is identical to output.Serial
        output['Serial'] = "serial:"+mymsg;
        output.Telnet = "telnet:"+mymsg;

        output.content = mymsg;
    }
    else
    {
        output.content = "bad read";
    }
}
```

① 注： 特に指定されていないチャンネルの出力は通常のコンテンツテキストです。次に例を示します。

```
function onResult (decodeResults, readerProperties, output)
{
    if (decodeResults[0].decoded)
    {
        /* save decoded result to variable */
        var mymsg = decodeResults[0].content;

        /* output to telnet channel a different result
        */
        output.Telnet = "telnet:" + mymsg;

        /* to all other channel output the saved
        result */
        output.content = mymsg;
    }
    else
    {
        /* On bad read output to all channels the same
        */
        output.content = "bad read";
    }
}
```

DecodeEvents

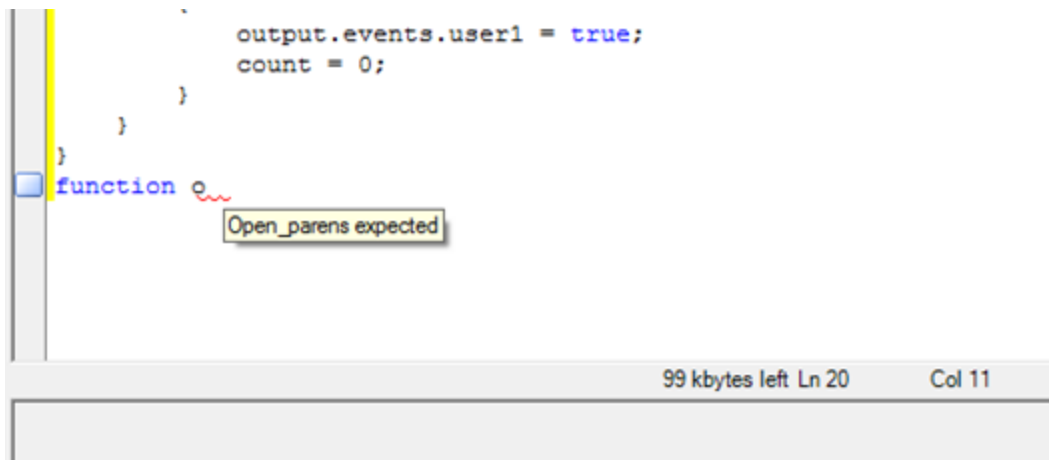
デコードの後に発生するイベントを記述します。

プロパティ	型	説明
system	integer	これらはシステム生成イベントです。 0 = なし 1 = 読み取り成功 2 = 読み取り失敗 3 = バリデーションエラー*
user1	boolean	ユーザイベント 1 が発生する場合は True です。
user2	boolean	ユーザイベント 2 が発生する場合は True です。

* 読み取り成功とバリデーションエラーの切替のみサポートされています。

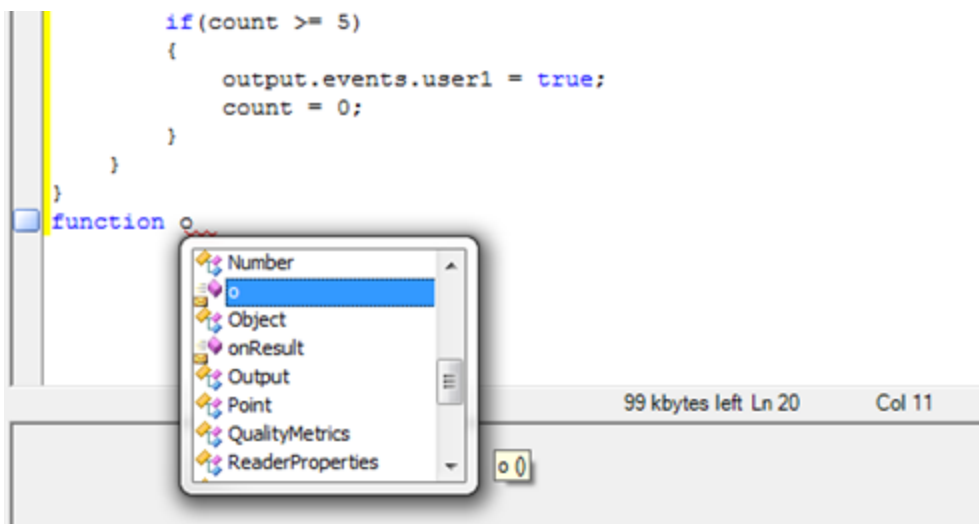
コードのオートコンプリートとスニペット

スクリプトエディタにはコードのオートコンプリート機能があり、記述中のコードに関する情報がポップアップメッセージに表示されます。ポップアップメッセージは通常、特殊文字 (ピリオド、開き括弧、閉じ括弧など) を入力しているときに表示されます。これらのメッセージは Ctrl+Space キーの組み合わせで手動で也可以使用することもできます。



コードのオートコンプリートは次のように動作します。

- コードの自動入力 (Ctrl-Space)
- 関数リストの入力 (Ctrl-Shift-Space)



エディタ上部のツールバーには、エディタ内で実行可能な次のアクションが表示されます。

- 切り取り (Ctrl-x)
- コピー (Ctrl-c)
- 貼り付け (Ctrl-v)
- ワードコンプリート (Ctrl-k を押して w を押す)
- スニペットの挿入 (Ctrl-k を押して x を押す)

スニペット

エディタには例として設定済みのコードの断片がが用意されています。エディタで右クリック、ツールバー、Ctrl-k と x キーの組み合わせを使用することで、これらのスニペットを挿入できます。

カスタム通信プロトコル API

カスタム通信スクリプトの作成は、DataMan Setup Tool でアクセス可能な boolean の VT エントリによってアクティブ化できます。複数のメソッドを 1 つの通信オブジェクトにまとめます。各通信チャンネルで通信オブジェクトのインスタンスを作成します。

プライベート DMCC コマンドを使用してカスタム通信プロトコル API を有効にすると、スクリプトのコンテキストによって次の機能および要件が追加されます。

- 通信オブジェクトのコンストラクタ関数 `CommHandler` には、スクリプトに含める必要のある関数の一覧が含まれています。
 - `onConnect`
 - `onDisconnect`
 - `onExpectedData`
 - `onTimer`
 - `onUnexpectedData`
 - `onError`
 - `onEncoder`

これらの関数をカスタム通信関数が呼び出すので、これらを実装しておく必要があります。

- リーダスクリプトエンジンが提供し、リーダによって実装されるメンバ関数は次の 5 つです。
 - `send`
 - `close`
 - `setTimer`
 - `expectFramed`
 - `setEncoder`

これらの関数を使用して、リーダで別のシステムとのやり取りを許可する JavaScript コードを書くことができます。特に、他のシステムにメッセージを返信するコードや、基本的なスクリプト作成でサポートされていないコードを書くことができます。

利点

このスクリプトエンジンは、関数の実行とオブジェクトの作成に同じコンテキストを使用します。これによって、スクリプトベースのフォーマットと、グローバルにアクセス可能なオブジェクトを使用するカスタム通信スクリプトとの間でデータを共有することができます。

関数のリスト

通信メンバ関数を使用して、次のメソッドプロトタイプを定義して実装します。

CommHandler – 通信オブジェクト用のコンストラクタ関数です。コンストラクタは、ユーザメソッドを通信スクリプトに実装する新しい通信ハンドラオブジェクトを返す必要があります。構築するとすぐに、リーダメソッドが通信ハンドラオブジェクトに追加されます。コンストラクタの実装は必須であり、存在しない場合はエラーがスローされます。ソフトウェアバージョン 5.5 以降では、コンストラクタ関数の呼び出しで次の引数を使用できるようになりました。

- **localName:** 接続のローカル名です。ネットワーク接続のローカル名は、"<READER_IP>:<PORT>" です。Telnet 接続の例として、デフォルトの telnet ポート 23 での "10.82.80.156:23" があります。ネットワーククライアント接続の例としては、"10.82.80.156:57350" があります。シリアル接続のローカル名は "COM1" または "COM USB" です。

- **onConnect** – 接続時に状態を初期化します (ネットワーク接続が確立されたか、シリアル接続時にプロトコルスタックが開始された時)。メソッドが実装されていない場合には、エラーが発生します。メソッドには引数 1 つと戻り値が含まれます。
 - **peerName** – 接続のピア名です。ネットワーク接続のピア名は、"<PEER_IP>:<PORT>" です。Telnet 接続のピア名の例としては "10.82.80.71:19772" があり、「ネットワーククライアント」接続の場合は "10.82.80.71:1000" となります。ここで、ホストポートは 1000 に設定されています。シリアル接続のピア名は "COM1" または "COM USB" です。
 - **return** – この接続のハンドラをアクティブ化するかどうかを、boolean の戻り値で定義します。
 - **true**: 通信プロトコルのカスタマイズを有効にします。したがって、DataMan デバイスとの通信に独自のプロトコルを使用する場合は、true を返します。
 - **false**: このピアに独自のプロトコルを必要としない場合は、false を返します。
- **onDisconnect** – 接続チャンネルを閉じるときに呼び出されるクリーンアップメソッドです。
- **onExpectedData** – 設定プロパティと一致するデータが届いた場合に呼び出されるメソッドです。このメソッドには引数が 1 つ含まれています。
 - **inputString** – ヘッダと終端を除いたデータにマッチした受信フレーム。
 - **return** – データを入力バッファから削除するかどうかを指定します。
 - **true**: バッファ内の値を保持します。
 - **false**: バッファをクリアします。
- **onTimer** – タイマの値が期限切れになっています。
- **onUnexpectedData** – 受信したデータが要件に一致していません。boolean の戻り値で、データを入力バッファから削除するかどうかを指定します。このメソッドには引数が 1 つ含まれています。
 - **inputString** – 受信したデータです。
 - **return** – データを入力バッファから削除するかどうかを指定します。
 - **true**: バッファ内の値を保持します。
 - **false**: バッファをクリアします。
- **onError** – ファームウェアでエラーが発生し、それが報告されます。メソッドの実装は必須です。このメソッドには引数が 1 つ含まれ、戻り値はありません。
 - **errorMsg** – トリガのオーバーラン ("Trigger Overrun")、バッファのオーバーラン ("Buffer Overflow") およびファームウェアによって報告される一般的なエラーに関するエラーメッセージです。
- **onEncoder** – 設定されたエンコーダの距離に達した場合に実行されます。この距離は **setEncoder** メソッドで設定することができます。このメソッドには引数も戻り値も含まれません。
- **send** – データをチャンネルに送信し、送信した文字数を返します。このメソッドは引数を 1 つ指定して呼び出す必要があります。
 - データの引数は文字列です。
- **close** – 通信オブジェクトの接続を有効な状態で終了します (たとえば、TCP/IP ソケットを閉じます)。UART では、直後に **onConnect** が呼び出されます。
- **setTimer** – **onTimer** の実行時に、1 回限りのタイマ値を設定します。タイマは再初期化したり中断することができます。
 - **double** 型の秒単位でのタイムアウト、内部の解像度はマイクロ秒 (1e-6 秒) です。ゼロの値は実行中のタイマを中断します。アクティブなタイマは上書きされます。

- **expectFramed** – **onExpectedData** に渡すデータおよび **onUnexpectedData** メソッドを通信リスナに伝達します。ランタイムに一致するパラメータを変更することができます。次の 3 つの引数が必要です。
 - **header**: string 型、空 ("") の場合があります。
 - **terminator**: string 型、空 ("") の場合があります。
 - **max length**: integer 型、入力メッセージの最大長を指定し、一致をチェックします (必須)。
- **setEncoder** – 距離の引数の単位はミリメートルです。Setup Tool の [システム設定] → [パルスエンコーダ] でエンコーダを設定します。距離の代わりにエンコーダのティックを使用する場合は、パラメータ "Resolution (mm)" の値を 1 に設定します。
 - **distance (double)** – **onEncoder** メソッドが呼び出されるエンコーダの距離です。

このメソッドは、オブジェクトのパブリックセクションに実装する必要があります。

例

カスタム通信プロトコルオブジェクトの API 使用法

この下記の例では、カスタム通信プロトコルオブジェクトの API 使用法を示しています。この例では、接続からのカスタムコマンド読み取りを実装します。このコマンドは "#" ヘッダで構成され、";¥r" で終了します (たとえば、シリアル PuTTY 接続)。カスタムの **stop** コマンドを使用して停止できるタイマメッセージを、定期的にタイマが送信します。タイマハンドラは、switch コマンドで 1 回変更できます。

```

function CommHandler()
{
    // private properties and methods:
    var num_trigger = 0;
    var num_send;

    // public properties and methods:
    function onTimeout()
    {
        num_send = this.send(my_name + ': timer callback\r\n');
        this.setTimer(1.0);
    }

    function onTimeout2()
    {
        today = new Date();
        var msg = today.getSeconds() * 1000 + today.getMilliseconds();

        num_send = this.send(my_name + ': time is:' + msg + '\r\n');
        dmccCommand("TRIGGER", true);
        this.setTimer(1.0);
    }

    function replace_crlf(input_str)
    {
        return input_str.replace(/\r/g, '\\r').replace(/\n/g, '\\n');
    }

    return {
        onConnect: function (peerName)
        {
            my_name = peerName;
            // we may ignore the connection
            if(my_name == "COM1")
                return false;

            num_send = this.send(my_name + ": connected\r\n");

            this.expectFramed("#", ";\r\n", 64);
            return true;
        },
        onDisconnect: function ()
        {
        },
        onExpectedData: function (inputString) {
            var msg = 'ok';
            this.expectFramed("#", ";\r\n", 64);
            if (inputString == "name")
            {
                msg = dmccGet("DEVICE.NAME");
                msg = msg.response;
            }

            else if(inputString == "trigger")

```

```

{
    this.send(my_name + ': issue a trigger...\r\n');
    dmccCommand("TRIGGER", true);

    msg = 'done';
}
else if (inputString == "close")
{
    this.close();
}
else if (inputString == "stop")
{
    this.setTimer(0.0);
}
else if (inputString == "start")
{
    this.setTimer(10.0);
}
else if (inputString == "switch")
{
    this.onTimer = onTimeout2;
}
else if (inputString == "time")
{
    today = new Date();
    msg = today.getSeconds() * 1000 + today.getMilliseconds();
}
else
{
    msg = "unknown command:" + replace_crlf(inputString);
}

num_send = this.send(my_name + ':' + msg + "\r\n");
return inputString.length;
},

onUnexpectedData: function (inputString) {
    this.expectFramed("#", ";\r\n", 128);
    msg = replace_crlf(inputString);

    num_send = this.send(my_name + ':' + msg + "?\r\n");

    return true;
},

onTimer: onTimeout
};
}

```

一般的な使用例: ハートビート

リーダがアイドル状態の場合に、定期的なハートビートメッセージを送信します。

```
// Data Formatting:

var comm_handler = new Array(0);

// Converts read data to all upper case.Single code only.
function onResult (decodeResults, readerProperties, output) {
    if (decodeResults[0].decoded) {
        output.content = decodeResults[0].content+'\r\n';
        for (var i = 0; i < comm_handler.length; i++)
        {
            comm_handler[i].resetHeartBeat();
        }
    }
}

// Communication:

// Heart beat example without disturbing the DMCC communication function CommHandler() {
    var beat_timer = 10.0; // beat timer in sec
    var peer_name;
    return {
        onConnect: function (peerName)
        {
            peer_name = peerName;
            this.resetHeartBeat(); // initial timer
            this.expectFramed("\0", "\0", 128); // some pattern
            unlikely to happen
            comm_handler.push(this); // register the handler for
            results
            // enable the handler for this connection:
            return true;
        },
        onDisconnect: function ()
        {
            var index = comm_handler.indexOf(this)
            comm_handler.splice(index, 1);
        },
        onError: function (errorMsg)
        {
        },
        onExpectedData: function (inputString) {
            return false;
        },
        onUnexpectedData: function (inputString) {
            return false;
        },
    },
}
```

```

onTimer: function () {
    today = new Date();

    var msg = today.getSeconds() * 1000 + today.getMilliseconds();
    num_send = this.send(peer_name + ': time is:' + msg + '\r\n');
    this.resetHeartBeat(); // schedule next timer event [sec]
},

resetHeartBeat: function () {
    this.setTimer(beat_timer); // schedule next timer event [sec]
}

};
}

```

一般的な使用例: 実時間タイムスタンプ

カスタムの DMCC コマンドを実装して実時間を設定します (1970 年 1 月 1 日から開始する現在の時間を秒単位で、date + "%s" コマンドによる出力として送信します)。出力の先頭に実時間タイムスタンプを追加します。

```

// communication script
var time_offset=0;

function CommHandler()
{
    var peer_name;

    return {
        onConnect: function (peerName)
        {
            peer_name = peerName;
            this.expectFramed("||;1>SET TIME.NOW ", "\r\n", 128); //
            some pattern unlikely to happen
            // enable the handler for this connection:
            return true;
        },
        onDisconnect: function ()
        {
        },
        onError: function (errorMsg)
        {
        },
        onExpectedData: function (inputString) {
            realTime = parseInt(inputString)*1000;
            localTime = (new Date()).getTime();
            time_offset = realTime - localTime;
            this.send("||[0]\r\n");
            return true;
        },
    },
}

```

```

        onUnexpectedData: function (inputString) {
            return false;
        },
        onTimer: function () {
        }
    };
}

// data formatting script
function onResult (decodeResults, readerProperties, output)
{
    var d = new Date();
    var real = new Date(time_offset+d.getTime());

    output.content = real.toString() + " " + decodeResults[0].content + "\r\n";
}

```

各種 CR リリースで実装されたカスタムプロトコル cmf400 Profibus ゲートウェイとの通信

```

var CMF400_PROTOCOL_STATUS =
{
    RUNNING: {value:0, name:"Running"},
    SYNCRONIZING:{value:1, name:"Sync"},
    CONFIGURING:{value:2, name:"Config"},
    STOPPED: {value:3, name:"Stop"}
};
// make the enum non-modifyable
Object.freeze(CMF400_PROTOCOL_STATUS);

var cmf400_protocol_stx = '\x02'; // header
var cmf400_protocol_etx = '\x03'; // termination

// VT Parameter to be converted into script configuration constant values:
// "/Communication/Interfaces/COM1/Protocol")
var vt_param_comif_com1_protocol = 1;
// "/Communication/Protocols/CMF400/Profibus node number"), 3);
var vt_param_profibus_node_number = 1;
// "/Communication/Protocols/CMF400/Profibus mode"), 3);*/
var vt_param_profibus_mode = 1;

// TODO: how to configure parameter, where to store them with a out of stock firmware?
var cmf400_protocol_profibus_node_number = 1;

var cmf400_protocol_profibus_mode = 1;

var cmf400_protocol_test_diagnostic_enabled = 0;

var cmf400_protocol_test_diagnostic = 'TEST';

// Protocol strings
var cmf400_gateway_init = '+Gateway-Init+';
var cmf400_gateway_ident_ok = '+GW SOK TSICDPS';
var cmf400_gateway_ident_no = '+GW SNO TSICDPS';
var cmf400_gateway_run = '+GW-RUN+';
var cmf400_gateway_error = '+GW-ERR';

```

```
// Formatting helper function
function zero_prefix(num, size)
{
    var s = "000000000" + num;
    return s.substr(s.length - size);
}

function CommHandler()
{
    // The current protocol state
    var cmf400_status = CMF400_PROTOCOL_STATUS.STOPPED;

    function _configTimedOut()
    {
        if (cmf400_status == CMF400_PROTOCOL_STATUS.CONFIGURING)
        {
            cmf400_status = CMF400_PROTOCOL_STATUS.STOPPED;
            this.setTimer(30.0);
            onTimer = _onSync;
        }
    }

    function _onSync()
    {
        if (cmf400_status == CMF400_PROTOCOL_STATUS.SYNCRONIZING)
        {
            this.send(cmf400_protocol_stx + cmf400_gateway_init +
                cmf400_protocol_etx);
            this.setTimer(1.0);
            onTimer = _onSync;
        }
    }

    function _onTimer()
    {
        if (cmf400_status == CMF400_PROTOCOL_STATUS.STOPPED)
        {
            cmf400_status = CMF400_PROTOCOL_STATUS.SYNCRONIZING;
            return;
        }
    }

    return {
        onConnect: function (peerName)
        {
            expectFramed("", cmf400_protocol_etx, 510); //
            is 510 an arbitrary limit?
            cmf400_status = CMF400_PROTOCOL_
                STATUS.SYNCRONIZING;
            this.onTimer = _onSync;
            this.setTimer(0.0001);
            return true;
        }
    };
}
```

```

    },
    onDisconnect: function ()
    {

    },
    onExpectedData: function (inputData)
    {
        data = inputData.slice(1,inputData.length-1);
        if (cmf400_status == CMF400_PROTOCOL_STATUS.SYNCRONIZING)
        {
            if (data == cmf400_gateway_ident_ok || data ==
                cmf400_gateway_ident_no)
            {
                cmf400_status = CMF400_PROTOCOL_
                    STATUS.CONFIGURING;
                var msg = cmf400_protocol_stx;

                msg += "+GW S000 H000";
                msg += " X" + zero_prefix(vt_param_
                    comif_com1_protocol, 3);
                msg += " N" + zero_prefix(vt_param_
                    profibus_node_number, 3);
                msg += " M" + zero_prefix(vt_param_
                    profibus_mode, 3);
                msg += cmf400_protocol_etx;
                this.send(msg);
                this.onTimer = _configTimedOut;
                this.setTimer(10.0);

            }

        }
        if (data == cmf400_gateway_error)
        {
            cmf400_status = CMF400_PROTOCOL_
                STATUS.STOPPED;
            this.setTimer(30.0);
            this.onTimer = _onTimer;

        }
        else if (data == cmf400_gateway_run) // missing check for
            status, e.g. CMF400_PROTOCOL_STATUS.CONFIGURING?
        {
            cmf400_status = CMF400_PROTOCOL_STATUS.RUN;
            this.setTimer(0);
            this.onTimer = _onTimer;

        }
        return true;
    },
    onUnexpectedData: function (inputData)
    {
        // ignore all unexpected data
        return true;
    },
    onTimer:_onSync
};
}

```

```
function onResult (decodeResults, readerProperties, output)
{
    //assuming single code
    var content = cmf400_protocol_stx+decodeResults[0].content+cmf400_protocol_etx;
    output.content = content;
}
```

重み付け文字列入力をデコードした文字列とともに渡す

```
// the constructor:

var input_string = "";

function CommHandler()
{
    // private properties and methods:

    var num_trigger = 0;
    var my_name;
    var num_send = 99;

    function privFunc ()
    {
    }

    // public properties and methods:

    return {
        onConnect: function (peerName)
        {
            my_name = peerName;
            num_send = this.send(my_name + ": connected\r\n");
            num_send = this.expectFramed("\x02", "\x03", 128);
            return true;
        },
        onDisconnect: function ()
        {
        },
        onExpectedData: function (inputString) {
            input_string = inputString;
            return true;
        },
        onUnexpectedData: function (inputString) {
            return true;
        }
    };
}
```

```
//Empty data formatting entry point function
function onResult (decodeResults, readerProperties, output)
{
    if (decodeResults[0].decoded)
    {
        output.content = input_string + decodeResults[0].content + "\r\n"
        input_string = "";
    }
}
}
```

FMPCS プロトコル

```
// This must be in the global scope, otherwise, it is undefined
var bConnected = false;

dmccSet('TRIGGER.TYPE', 0);
dmccSet('SYMBOL.4STATE-IMB', 1);
dmccSet('SYMBOL.DATAMATRIX', 1);
dmccSet('SYMBOL.I2O5', 1);
dmccSet('SYMBOL.PDF417', 1);
dmccSet('SYMBOL.POSTNET', 1);

function CommHandler()
{
    var tray = "0000";
    var speed = 0;

    var package_id_expr = new RegExp("I([0-9]{9})");
    var package_idtray_expr = new RegExp('^I([0-9]{9}),T([0-9]{4})');
    var config_msg_expr = new RegExp('^CS([0-9]{3}),M([ab]),L([0-9]{4})$');

    var ErrorToId = {
        'Buffer Overflow':101,
        'Trigger Overrun':102
    };

    return {
        onConnect: function (peerName)
        {
            if(peerName == "COM1" || bConnected)
                return false;
            this.expectFramed("", "\r", 128);
            this.send(dmccGet('DEVICE.FIRMWARE-VER').response +
                ', "Cognex ' + dmccGet('DEVICE.TYPE').response + '"\r\n');
            this.send('Ha, "DataMan READY"\r\n');
            bConnected = true;
            return true; // activate this connection
        },
        onError: function (msg) // TODO: this is new!
        {

```

```

var errno = ErrorToId[msg];
if (!errno)
  errno = 100;
this.send('E' + errno + ', "' + msg + '"\r\n');
},
// We delay sending the result until trigger off to be sure that the
package id is received.
setResult: function (decodeResults) {
  storedDecodeResults = decodeResults;
},
onDisconnect: function ()
{
  bConnected = false;
},
onExpectedData: function (input)
{
  var input = input.replace(/\n/g, '');
  switch(input.charAt(0).toUpperCase())
  {
    case 'B':
      dmccCommand("TRIGGER", true);
      break;
    case 'E':
      dmccCommand("TRIGGER", false);
      break;
    case 'I':
      var match = package_idtray_
        expr.exec(input);
      if(!match)
      {
        match = package_id_
          expr.exec(input);
      }
      packageID = match[1];
      if(match[2])
      {
        tray = match[2];
      }
      else
      {
        tray = "0000";
      }
      break;
  }
}

```

```

        case 'C':
            var match = config_msg_expr.exec(input);
            if (match.length == 4)
            {
                speed = parseInt(match[1], 10);
                mode = match[2];
                lengthLimit = parseInt(match[3],
                    10);
            }
            break;
        case 'P':
            this.send('Q\r\n');
            break;
        case 'Q':
            // pong response, not used
            break;
    }
    return true;
},
onUnexpectedData: function (input) {
    return true;
}
};
}

```

通信ハンドラによって設定されたグローバル変数に基づいて、データフォーマットで結果をフォーマット設定します。

```

var packageID = "000000000"; // reset the package id
var mode = 'a';
var lengthLimit = 9999;

function getFixedPsocId(id_)
{
    var id = id_;
    switch (id.charAt(1))
    {
        case 'd':
            id = "[D0";
            break;
        case 'X':
            switch (id.charAt(2))
            {

```

```

        case '0':
        case '1':
            id = "[P0";
            break;
        case '2':
        case '3':
            id = "[L0";
            break;
        case '5':
        case '6':
        case '7':
        case '8':
        case '9':
        case 'A':
            id = "[00";
            break;
    }
    break;
}
return id;
}
function onResult (decodeResults, readerProperties, output)
{
    var my_decode_results = new Array();

    for(var i = 0; i < decodeResults.length; i++)
    {
        if(!decodeResults[i].decoded)
            continue;
        switch (decodeResults[i].symbolology.name)
        {
            case 'Interleaved 2 of 5':
                // b=throw away 6 digit I2of5 ending in 9
                if ((mode == 'b' && decodeResults
                    [i].content.length == 6 && decodeResults
                    [i].content.charAt(5) == '9'))
                    continue;
            case 'Data Matrix':
                if (decodeResults[i].content.length >
                    lengthLimit)
                    continue;
            case 'PDF417':
                if (decodeResults[i].content.length >
                    lengthLimit)
                    continue;
            default:
                my_decode_results.push(decodeResults[i]);
        }
    }
}

```

```

var msg = 'D' + packageID + ',S,W,V';
if (my_decode_results.length == 0)
{
    msg += ',?';
    output.content = "no result";
}
else
{
    for(var i = 0; i < my_decode_results.length; i++)
    {
        msg += ',' + getFixedPsocId(decodeResults
            [i].symbology.id);
        switch (my_decode_results[i].symbology.name)
        {
            case 'Data Matrix':
            case 'PDF417':
                msg += encode_base64(my_decode_results
                    [i].content);
                break;
            case 'POSTNET':
            case 'PLANET':
            case 'XYZ OneCode':
            case 'Interleaved 2 of 5':
            default:
                msg += my_decode_results
                    [i].content;
        }
    }
}
packageID = "0000000000"; // reset the package id
output.Telnet = output.Serial = msg + '\r\n';
}

```

一致文字列の入力、スクリプトからの出力 (30x)

```

function CommHandler()
{
    return {
        onConnect: function (peerName)
        {
            this.expectFramed('\x02', '\x03', 256);
            return true;
        },
        onDisconnect: function ()
        {
        },
        onExpectedData: function (inputString) {

```

```

    if (inputString.length >= 11)
    {
        var new_match_string = inputString.substr(11,
            inputString.length);
        for (var i = 1; i <= 3; i++) {
            dmccSet("DVALID.PROG-TARG", i);
            dmccSet("DVALID.MATCH-STRING", new_match_
                string);
        }
        // The following DMCC command resets all statistic values
        // the CR reset only a view of them
        dmccCommand("STATISTICS.RESET");
    }
    this.send("DEBUG:"+inputString + "\r\n");
    return true;
},
onUnexpectedData: function (inputString) {
    return true;
},
onTimer:function (inputString) {
}
};
}

```

データフォーマットで出力を通信ハンドラオブジェクトにデリゲートする

```

var comm_handler = new Array(0);

// Converts read data to all upper case.Single code only.
function onResult (decodeResults, readerProperties, output)
{
    output.content = '';
    output.SetupTool = decodeResults[0].content;
    if (decodeResults[0].decoded) {
        for (var i = 0; i < comm_handler.length; i++)
        {
            comm_handler[i].sendResultTelegram(decodeResults);
        }
    }
}

```

```

// Parameter:
var system_id = '\x43'; // the system ID
var heartbeat_time_s = 5.0; // heartbeat timer in sec [0-50] (0 is disabled)
var append_crlf = true; // wether to

function CommHandler()
{
    function getChecksum(data)
    {
        var sum = 0;
        for(var i = 0; i < data.length; i++)
            sum += data.charCodeAt(i);
        return 0x7F - (sum % 0x7f);
    }

    var TelegramState = {
        WAIT4CONTENT:{value:0, name:"Wait For Content"},
        CHECKSUM: {value:1, name:"Header Received"}
    };

    var errorCodes = {
        undef_index:0x31,
        multi_index:0x32,
        index_in_use:0x33,
        telegram_error:0x34,
        trigger_overrun:0x40,
        buffer_overflow:0x41,
    };

    var filler = '#';
    var separator = ',';

    var telegram_types = {
        heartbeat:{type:'F', content: system_id+'\xf7'},
        init_resp:{type:'J', content: system_id},
    };

    // initialization:J
    // index: S

    var telegram;
    var status;
    var index;
    var all_index = new Array();

    return {

        sendResultTelegram: function (decodeResults)
        {
            var data = system_id;
            var length = 0;

```

```

        if (!index)
        {
this.sendErrorTelegram(errorCodes.undef_index);

        index = '9999';
    }

    data += index;
    for (var i = 0; i < decodeResults.length; i++) {
        length = decodeResults[i].content.length;
        data += String.fromCharCode(length / 256, length % 256);
    }

    data += separator + filler;
    length = 0;
    for (var i = 0; i < decodeResults.length; i++) {
        length += decodeResults[i].content.length;
        data += decodeResults[i].content;
    }
    if (length & 0x1)
        data += filler;

    data += String.fromCharCode(getChecksum(data));
    this.sendTelegram({type: system_id, content: data});

    index = null; // invalidate the used index
},
sendErrorTelegram: function (errcode)
{
    var errtel = {type:'F', content: system_id+String.fromCharCode
(errcode)}

    this.sendTelegram(errtel);
},
sendTelegram: function (telegram)
{
    var data = telegram.type + telegram.content;
    data = '\x02'+data+String.fromCharCode(getChecksum(data))+'\03';
    this.send(data);
    if (append_crlf)
        this.send('\r\n');
},
checkTelegram: function(data, checksum)
{
    var exp_checksum = getChecksum(data);
    if (checksum != exp_checksum) {
this.sendErrorTelegram(errorCodes.telegram_error);
    } else {
        switch (data[0])
        {
            case 'I':

```

```

this.sendTelegram(telegram_types.init_resp);

        this.setTimer(0.0); // disable the
        heartbeat timer
        all_index = new Array(0);
        break;
        case 'S':
        if (index) {

this.sendErrorTelegram(errorCodes.multi_index);

                break;

        }
        index = data.substr(1, 4);
        if (all_index.indexOf(index) >= 0)
this.sendErrorTelegram(errorCodes.index_in_use);

                else

                        all_index.push(index);

                break;

        default:

                break;

        }
    }
},

onConnect: function (peerName)
{
    status = TelegramState.WAIT4CONTENT;
    this.expectFramed('\x02', '\x03', 203);
    this.setTimer(heartbeat_time_s);
    index = null;
    comm_handler.push(this);
    all_index = new Array();
    return true;
},

onDisconnect: function ()
{
    var index = comm_handler.indexOf(this)
    comm_handler.splice(index,1);
},

onExpectedData: function (inputString) {
    switch (status)
    {

    case TelegramState.WAIT4CONTENT:

        this.expectFramed('', '', 1); // actually, disable framing
        telegram = inputString;
        status = TelegramState.CHECKSUM;
        break;

    case TelegramState.CHECKSUM:

        this.expectFramed('\x02', '\x03', 203); // enable framing
        for the next telegram
        this.checkTelegram(telegram, inputString.charCodeAt(0));
        status = TelegramState.WAIT4CONTENT;
        break;
    }
}

```

```

        default:
            throw("unknown state");
        }
        return true;
    },
    onUnexpectedData: function (inputString) {
        this.expectFramed('\x02', '\x03', 203); // enable framing for the
        next telegram
        status = TelegramState.WAIT4CONTENT;
        return true;
    },
    onTimer:function (inputString) {
        this.sendTelegram(telegram_types.heartbeat);
        this.setTimer(heartbeat_time_s);
    }
};
}

```

イベントコールバック

コールバックメカニズムによって、トリガおよび入力イベント用のハンドラを登録することができます。これらのイベント用のハンドラは、次のように `registerHandler` メソッドを使用して登録することができます。

```
callback_handle registerHandler(eventid, callback, ...)
```

registerHandler 関数では次の引数が必要です。

- eventid – 登録対象のイベントタイプの識別子
- callback – イベントに対して実行する関数オブジェクト

使用可能なイベント識別子は、"Callback" という名前の constant オブジェクトに定義されます。オプションの引数を使用して、感度をフィルタするなどのイベントを設定することができます。

コールバックの再登録に使用する必要があるハンドラが返されます。ハンドラを再登録するには、次のように **deregisterHandler** 関数を使用します。

```
deregisterHandler(callback_handle)
```

- callback_handle – registerHandler メソッドによって返されるハンドルです。

データフォーマットで使用するなど、グローバルスコープ内でコールバックハンドラを登録することができます。

イベントタイプ

登録可能で現在有効なイベントは、"onInput" および "onTrigger" イベントです。

onInput イベント: 入力信号およびボタンの変更時に、コールバック関数を呼び出します。オプションの第 3 の引数で、特定の入力用のフィルタを設定することができます。オブジェクト "ConstInput" で入力用のマスクを定義します。

- Input0:
- Input1:
- Input2:
- Input3:
- Input4:

- Input5:
- Input6:
- InputAll
- BnTrig
- BnTune

入力マスクを結合することができます。入力値は 1 ミリ秒 の精度でサンプリングされます。onInput イベントのコールバック関数には、入力の新しい状態用の引数が 1 つ含まれています。

onTrigger イベント: トリガ開始およびトリガ終了イベントでコールバック関数を実行します。onTrigger イベントのコールバック関数には 2 つの引数が含まれます。1 つ目の引数はトリガオブジェクト、2 つ目の引数はトリガの boolean 状態であり、トリガ開始が true、トリガ終了が false です。

例

この例では、次の 3 つのイベントハンドラを定義します。

- onInput0 – input0 信号およびスイッチボタンに反応
- onInput1 – input1 信号に反応
- onTrigger – トリガイベントに反応

```
function CommHandler()
{
    return {
        onConnect: function (peerName)
        {
            this.peer = peerName;
            this.input1 = registerHandler(Callback.onInput,
            this.onInput0.bind(this),
            ConstInput.Input0|ConstInput.BnTrig);
            this.input2 = registerHandler(Callback.onInput,
            this.onInput1.bind(this), ConstInput.Input1);
            this.ontrigger = registerHandler(Callback.onTrigger,
            this.onTrigger.bind(this));
            return true;
        },
        onDisconnect: function ()
        {
            deregisterHandler(this.input1);
            deregisterHandler(this.input2);
            deregisterHandler(this.ontrigger);
        },
        onTrigger: function (trigger, state) {
            if (state)
                this.send("call onTrigger: started trigger with index " +
                trigger.index + "\r\n");

            else
                this.send("call onTrigger: end trigger with index " +
                trigger.index + "\r\n");
        }
    };
}
```

```

    },
    onInput0: function (inputs) {
        this.send("call onInput0 for '" + this.peer + "', inputs=" + inputs +
            "\r\n");
    },
    onInput1: function (inputs) {
        this.send("call onInput1 for '" + this.peer + "', inputs=" + inputs +
            "\r\n");
    }
};
}

```

イベントシーケンス、input1 on、input0 on、input0 off、input1 off、software trigger、switch on、switch off を使用して、ターミナルで次の出力を取得します。

```

call onInput1 for 'COM1, inputs=2
call onTrigger: start trigger with index 9
call onInput0 for 'COM1, inputs=1
call onTrigger: end trigger with index 9
call onInput0 for 'COM1, inputs=0
NO-READ
call onInput1 for 'COM1, inputs=0
call onTrigger: start trigger with index 10
NO-READ
call onTrigger: end trigger with index 10
call onInput0 for 'COM1, inputs=4096
call onTrigger: start trigger with index 11
call onInput0 for 'COM1, inputs=0
call onTrigger: end trigger with index 11
NO-READ

```

次の例では on Input1 イベントでハンドラを登録し、グローバル変数で状態を格納します。データフォーマットによって入力の状態が出力されます。

```

var ginputs = false;

registerHandler(Callback.onInput, onInput, ConstInput.Input1);

// Default script for data formatting
function onResult (decodeResults, readerProperties, output)
{
    output.content = "Input:"+ginputs+" \r\n";
}

function onInput(inputs)
{
    ginputs = (inputs & ConstInput.Input1) ? true : false;
}

```

