マイクロネット

リモート共有メモリ
TcpIpDrv

ユーザーズ ガイド

Rev. 1.0.0.0

INplc

はじめに

■ 本ドキュメントについて

本書は、INplc で TCP/IP 通信をご利用いただくための INplc-Driver 「TcpIpDrv」 の取扱説明書です。
なお、利用にあたっては、TCP/IP 通信について理解されていることが前提となります。

関連するドキュメントは、以下のとおりです。

- TcpIpDrv セットアップガイド　【TcpIpDrv_Setup.pdf】
- INplc ユーザーズマニュアル　【INplc ユーザーズマニュアル.pdf】
- INplc クイックスタートガイド　【MULTIPROG クイックスタートガイド.pdf】

> 📖 本書では原則として、Windows XP、INplc-SDK(Pro+)の操作手順および画像を使用しています。
> お使いの OS や INplc-SDK のエディション等により、記載されている操作手順や画像などが異なる場合がありますので、適時読み替えてご利用ください。

■ 事前設定

1) ネットワークの設定

INplc-Controller に本ドライバで使用するネットワークが設定されていない場合、セットアップガイド【TcpIpDrv_Setup.pdf】の『ハードウェアの設定』の説明に従って、ネットワークの設定を行ってください。

第2章　ドライバの設定

開発 PC（INplc-SDK）から、INplc-Controller に接続し、TcpIpDrv の起動および動作設定を行います。

2.1. IO・FWL 起動設定ツールの起動

1. 開発 PC 側スタートメニューのすべてのプログラムから［INplc］➢［INplcTool］を選択します。

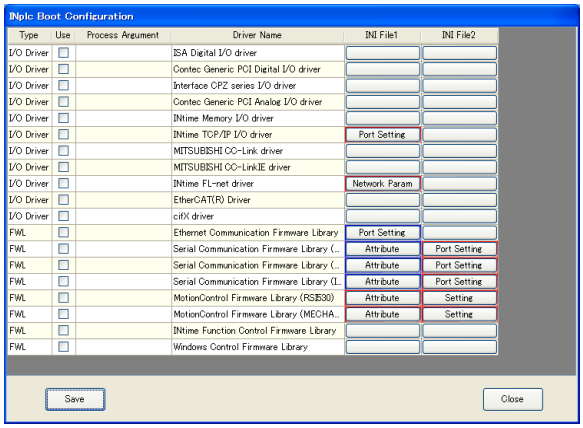
2. INplc-Controller 選択ダイアログが表示されるので、一覧から INplc-Driver を適用する INplc-Controller を選択して［接続］ボタンをクリックします。

3. INplc Configuration Panel が表示されるので、［IO・FWL 起動設定］アイコンをダブルクリックします。

IO・FWL起動設定

※ **INplc-Controller が稼働中（PLC プログラムの実行中）の場合は、IO・FWL 起動設定ツールを起動できません。稼働中の場合は、MULTIPROG から PLC プログラムを停止してください。**

以上で、IO・FWL 起動設定ツールの起動は完了です。
表示される画面から、各ドライバ設定の編集ができます。



2.2. ドライバのロード設定の編集

ドライバのロード時の設定を編集します。
画面の［Driver Name］列に［**TCP/IP Communication Driver**］と表記されている行の各項目を編集してください。

1) INplc でドライバをロードする設定

［**Use**］列のチェックボックスを ON にすることで、TcpIpDrv が INplc にロードされるようになります。

2.3. 動作設定ファイルの編集

［**INI File**］列の［**Port Setting**］ボタンをクリックすることで、動作設定ファイルが開きます。
画面の［Driver Name］列に［**TCP/IP Communication Driver**］と表記されている行の各項目を編集してください。

※ 基本的に、デフォルト値を変更する必要はありません。

```
[MEMORY]
MemorySize=4096

[TCPPORT]
TcpPort01=9090
```

設定項目は以下のとおりです：

No.	セクション	キー	説明
1	MEMORY	MemorySize	通信に使用するメモリサイズ（4096～65536 バイト）を指定します。 メモリは、4K バイト区切りで生成されます。 例）5000 と設定した場合、8192 バイトで生成されます。
2	TCPPORT	TcpPort01	通信に使用するポートを設定します。

2.4. 設定の適用

変更した設定内容を、INplc-Controller に適用します。
注意）設定の適用は、他の I/O ドライバおよびファームウェアライブラリの設定も適用されます。

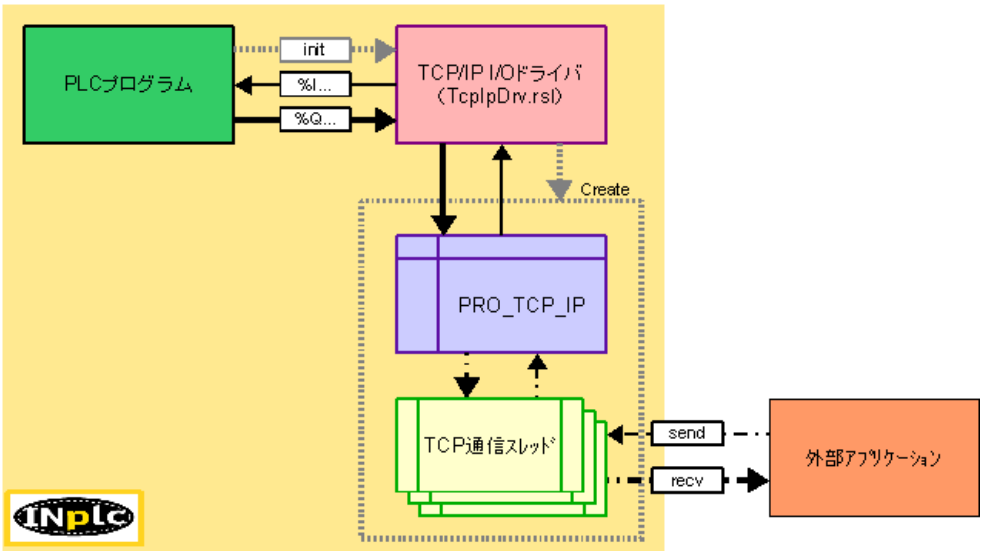
1. IO・FWL 起動設定ツールの画面左下の［Save］ボタンをクリックします。

2. INplc の再起動メッセージが表示されるので、［はい］を選択して再起動を行うことで、設定が適用されます。
   ※ この再起動はソフトウェアの再起動です。INplc-Controller 本体の再起動ではありません。

第3章 プログラミング

3.1. 通信構成

本 INplc-Driver では、以下のような通信構成になっています。

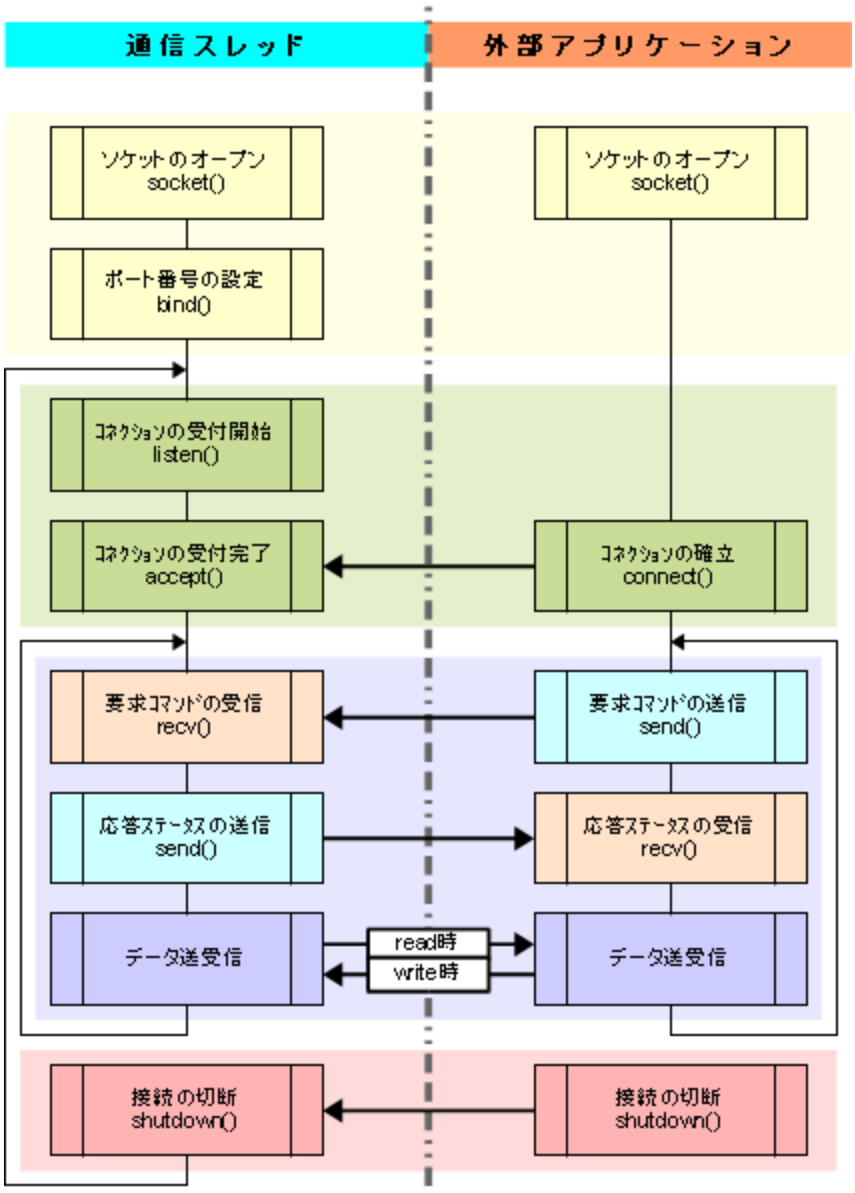


名称	内容
TCP/IP I/O ドライバ	MULTIPROG から PLC プログラムのダウンロードが完了した時に、外部アプリケーションとの通信を行うスレッド、共有メモリを生成します。 I/O アドレス設定に、**物理入力設定(%I...)**がされている場合、共有メモリから読込まれたデータを取得します。**物理出力設定（%Q...）**がされている場合、設定データを共有メモリに書込みます。
PRO_TCP_IP	I/O ドライバで使用する共有メモリです。 カタログ名は「**PRO_TCP_IP**」で、共有メモリサイズはデフォルト［**4096**］バイトです。
TCP 通信スレッド	外部アプリケーションとの通信を行うスレッドです。 カタログ名は「**TCP_ポート番号**」です。通信ポートはデフォルト［**9090**］です。
外部アプリケーション	TCP/IP 通信を行うアプリケーションです。

1) TCP 通信スレッドと外部アプリケーションとの通信フロー

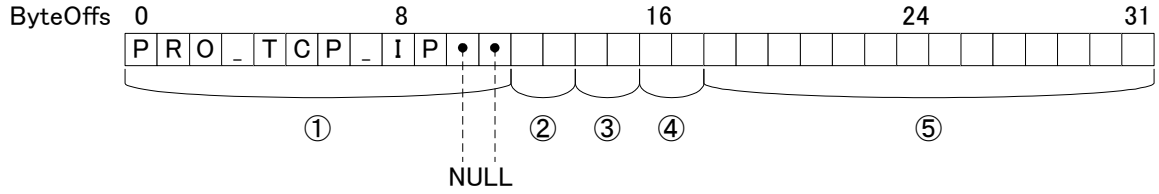
通信スレッドは外部アプリケーションとのコネクション受付開始後、受信した要求コマンドから対応する処理を行います。

- read 要求の場合、共有メモリの内容を外部アプリケーションに送信します。
- write 要求の場合、受信した情報を共有メモリに書込みます。



2) 要求コマンドのパケット構成

要求コマンドは 32 バイトで構成されています。



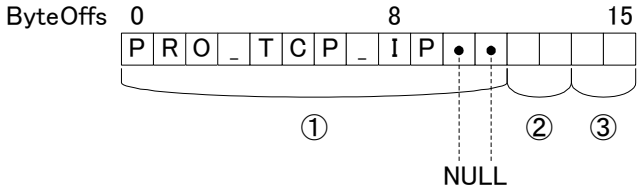
①	サブヘッダ（“PRO_TCP_IP”）
②	要求コード（1:read 要求 / 2:write 要求）
③	アクセスする共有メモリの開始アドレス（0～4095）
④	アクセスする共有メモリのバイトサイズ（1～4096）
⑤	予備（将来用）

宣言例：

```
typedef struct _SOCK_REQ_COMMAND_INFO
{
    char        SubHead[12];
    WORD        ReqCode;
    WORD        StartAddr;
    WORD        ByteSize;
    WORD        Reserve[7];
} *LPREQCMDINFO, REQCMDINFO;
```

3) 応答ステータスのパケット構成

応答ステータスは 16 バイトで構成されています。



①	サブヘッダ（“PRO_TCP_IP”の後に NULL（’¥0’）を追加してください）
②	応答コード（0:正常終了 / 0 以外:エラー）
③	予備（将来用）

応答コード:

11	受信サイズエラー
21	サブヘッダ不一致
31	ReqCode が不正値
41	StartAddr が不正値
51	ByteSize が不正値

宣言例：

```
typedef struct _SOCK_RES_COMMAND_INFO
{
    char            SubHead[12];
    WORD            ResCode;
    WORD            Reserve;
} *LPRESCMDINFO, RESCMDINFO;
```

4) 送受信データ

送受信データは、要求コマンドで指定したバイトサイズで送受信を行います。

3.2. I/O ドライバ

I/O ドライバのデータは、I/O グループの設定を行うことにより、INplc の I/O メモリ領域に割り付けられます。
入力データは［I エリア］へ、出力データは［Q エリア］にそれぞれ割り付けられます。

I/O データの割り付けの詳細、I/O グループの設定方法につきましては、各 I/O ドライバの種類の中で説明します。

本 INplc-Driver の I/O ドライバには以下の種類があります：

種類	機能
通信データ読み込み	設定されたアドレス位置から、通信用共有メモリ「**PRO_TCP_IP**」のデータを取得（Read）します。
通信データ書き込み	設定されたアドレス位置から、通信用共有メモリ「**PRO_TCP_IP**」にデータを設定（Write）します。

1) 通信データ読み込み

① I/Oデータ割り付け例

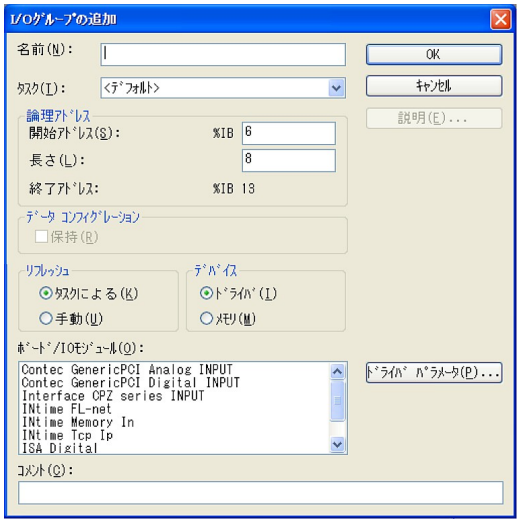
データの割り付けは、通信用共有メモリ「**PRO_TCP_IP**」の指定された開始アドレスから、サイズ（長さ）分のデータがそのままIエリアに割り付けられます。

■ **入力10バイト**・・・　［Iエリア］の 10バイト分 の領域を占有します。

オフセット（バイト）	Iエリア
0	**通信用共有メモリ（指定した開始アドレス+ 0）**
+1	**通信用共有メモリ（指定した開始アドレス + 1）**
+2	**通信用共有メモリ（指定した開始アドレス + 3）**
・ ・ ・	・ ・ ・
+7	**通信用共有メモリ（指定した開始アドレス + 7）**
+8	**通信用共有メモリ（指定した開始アドレス + 8）**
+9	**通信用共有メモリ（指定した開始アドレス + 9）**

② I/O グループの設定

MULTIPROG の IO_Configuration の I/O グループの追加ダイアログで、以下のように設定してください。



名前	任意の名前を設定してください。
タスク	入力データを更新するタイミングを同期するタスクを選択します。 指定したタスクの実行時に、最初の処理として入力データを更新します。
開始アドレス	入力データを取得する I エリアのアドレスを指定します。
長さ	占有するバイト数を指定します。**最大［4096］**バイトです。 通信用共有メモリのアクセス範囲を超えないように注意してください。
ボード/IO モジュール	一覧から「**TCP/IP**」を選択します。 選択後、［ドライバ パラメータ］ボタンをクリックして、ドライバの詳細設定を行います。 ☞ ドライバパラメータにつきましては、P.10「ドライバパラメータの設定」をご覧ください。

☞🕮 MULTIPROG の操作方法につきましては、INplc-SDK 付属の INplc ユーザーズマニュアルや MULTIPROG のヘルプ等を参照してください。

③ ドライバパラメータの設定

I/O グループの追加ダイアログの［ドライバ パラメータ］ボタンをクリックすると、ドライバ固有の設定ダイアログが表示されます。

設定項目は、以下のとおりです：

項目	解説
通信用共有メモリの設定 	● 開始アドレス 通信用共有メモリへアクセスするアドレスのオフセット（0～4095）を指定します。 ※ アクセスする領域が書き込み側と重ならないように注意してください。

2) 通信データ書き込み

① I/Oデータ割り付け例

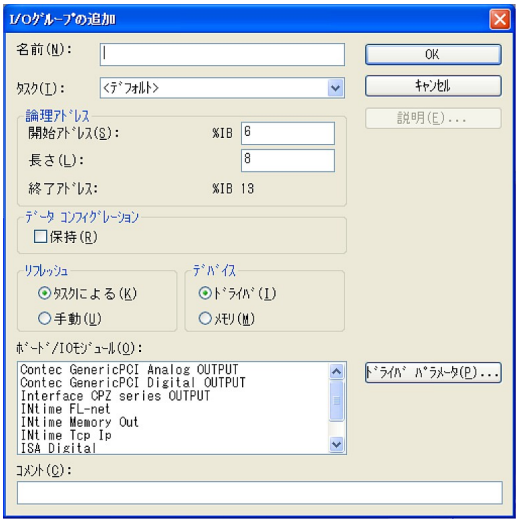
データの割り付けは、通信用共有メモリ「**PRO_TCP_IP**」の指定された開始アドレスから、サイズ（長さ）分のデータがそのままQエリアに割り付けられます。

■ **出力10バイト**・・・　［Qエリア］の 10バイト分 の領域を占有します。

オフセット（バイト）	Qエリア
0	**通信用共有メモリ（指定した開始アドレス+ 0）**
+1	**通信用共有メモリ（指定した開始アドレス + 1）**
+2	**通信用共有メモリ（指定した開始アドレス + 3）**
・ ・ ・	・ ・ ・
+7	**通信用共有メモリ（指定した開始アドレス + 7）**
+8	**通信用共有メモリ（指定した開始アドレス + 8）**
+9	**通信用共有メモリ（指定した開始アドレス + 9）**

② I/O グループの設定

MULTIPROG の IO_Configuration の I/O グループの追加ダイアログで、以下のように設定してください。



名前	任意の名前を設定してください。
タスク	出力データを更新するタイミングを同期するタスクを選択します。 指定したタスクの実行時に、最後の処理として出力データを更新します。
開始アドレス	出力データを設定する Q エリアのアドレス位置を指定します。
長さ	占有するバイト数を指定します。**最大［4096］**バイトです。 通信用共有メモリのアクセス範囲を超えないように注意してください。
ボード/IO モジュール	一覧から「**TCP/IP**」を選択します。 選択後、［ドライバ パラメータ］ボタンをクリックして、ドライバの詳細設定を行います。 ドライバパラメータにつきましては、P.12「ドライバパラメータの設定」をご覧ください。

☞🕮 MULTIPROG の操作方法につきましては、INplc-SDK 付属の INplc ユーザーズマニュアルや MULTIPROG のヘルプ等を参照してください。

③ ドライバパラメータの設定

I/O グループの追加ダイアログの［ドライバ パラメータ］ボタンをクリックすると、ドライバ固有の設定ダイアログが表示されます。

設定項目は、以下のとおりです：

項目	解説
通信用共有メモリの設定 	● 開始アドレス 通信用共有メモリへアクセスするアドレスのオフセット（0～4095）を指定します。 ※ アクセスする領域が読み込み側と重ならないように注意してください。

④ Stop 時の出力設定

PLC プログラムを停止した際の、出力信号の保持状態を設定します。

注意）この設定は、I/O ドライバ共通の設定です。すべての I/O ドライバに適用されます。

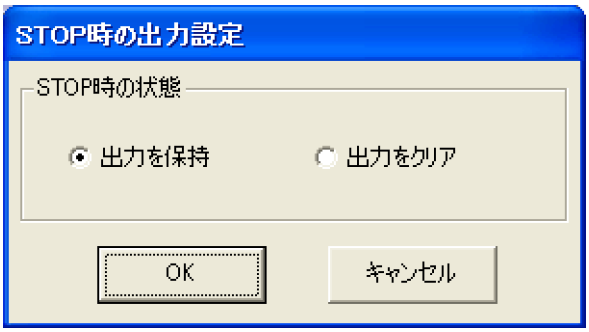
1. 開発 PC 側スタートメニューのすべてのプログラムから［INplc］➢［INplcTool］を選択します。

2. INplc-Controller 選択ダイアログが表示されるので、一覧から INplc-Driver を適用する INplc-Controller を選択して［接続］ボタンをクリックします。

3. INplc Configuration Panel が表示されるので、［Stop 時の出力設定］アイコンをダブルクリックします。

※ **INplc-Controller が稼働中（PLC プログラムの実行中）の場合は、設定ツールを起動できません。稼働中の場合は、MULTIPROG から PLC プログラムを停止してください。**

4. Stop 時の出力設定ツールが表示されるので、STOP 時の状態を選択して［OK］ボタンをクリックします。



5. INplc の再起動メッセージが表示されるので、［はい］を選択して再起動を行うことで設定が適用されます。

※ この再起動はソフトウェアの再起動です。INplc-Controller 本体の再起動ではありません。

3.3. 基本的な使用手順

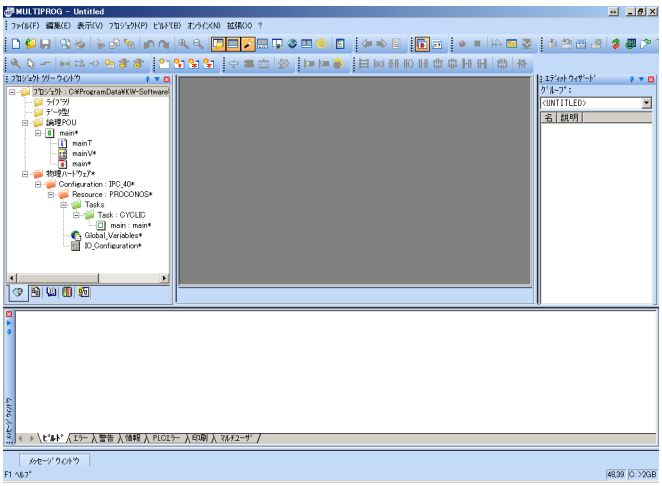
ラダーダイアグラム（LD）のプログラムで使用する手順を説明します。

1) PLCプロジェクト作成

MULTIPROG を起動して、新規に PLC プロジェクトを作成します。

1. 開発 PC 側スタートメニューのすべてのプログラムにある、KW-Software グループ内 MULTIPROG グループの中から［MULTIPROG］を選択します。
2. MULTIPROG が起動するので、ツールバーの［新規プロジェクト］ アイコン をクリックします。
3. テンプレート一覧から［INplc_LD］を選択します。

新しいプロジェクトが作成され、画面左側のプロジェクトツリーウィンドウにプロジェクトツリーが表示されます。



2） IO_Configuration の設定

PLC プログラムで使用する I/O の設定を行います。
プロジェクトツリーの［IO_Configuration］をダブルクリックして、I/O コンフィグレーションダイアログを表示します。

① INPUT グループの設定

1. I/O コンフィグレーションダイアログの［INPUT］タブから［追加］ボタンをクリックします。

2. I/O グループの追加ダイアログが表示されるので、次のように設定します。

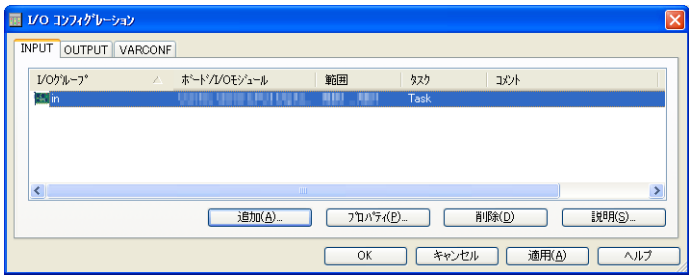
名前	「**in**」と入力します。
タスク	一覧から「**Task**」を選択します。
開始アドレス	「**0**」と入力します。
長さ	「**2**」と入力します。（今回は、入力 2 バイト分のみ使用する）
ボード/IO モジュール	一覧から「**TCP/IP**」を選択します。

3. I/O グループの追加ダイアログの［ドライバ パラメータ］ボタンをクリックします。
設定ダイアログが表示されるので、次のように設定してから［OK］ボタンをクリックします。

開始アドレス	「**0**」と入力します。

4. I/O グループの追加ダイアログに戻るので、［OK］ボタンをクリックします。

I/O コンフィグレーションダイアログの一覧に設定したグループが表示されます。



② OUTPUT グループの設定

1. I/O コンフィグレーションダイアログの［OUTPUT］タブから［追加］ボタンをクリックします。

2. I/O グループの追加ダイアログが表示されるので、次のように設定します。

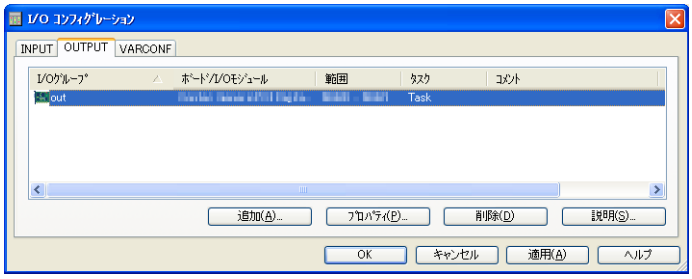
名前	「**out**」と入力します。
タスク	一覧から「**Task**」を選択します。
開始アドレス	「**0**」と入力します。
長さ	「**2**」と入力します。（今回は、出力 2 バイト分のみ使用する）
ボード/IO モジュール	一覧から「**TCP/IP**」を選択します。

3. I/O グループの追加ダイアログの［ドライバ パラメータ］ボタンをクリックします。
   設定ダイアログが表示されるので、次のように設定してから［OK］ボタンをクリックします。

開始アドレス	「**2048**」と入力します。

4. I/O グループの追加ダイアログに戻るので、［OK］ボタンをクリックします。

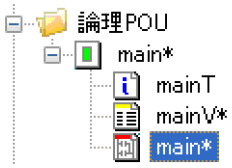
I/O コンフィグレーションダイアログの一覧に設定したグループが表示されます。



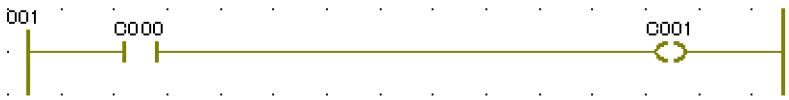
3) プログラムコードの作成

取得した入力データを、そのまま出力データとして設定するプログラムを作成します。

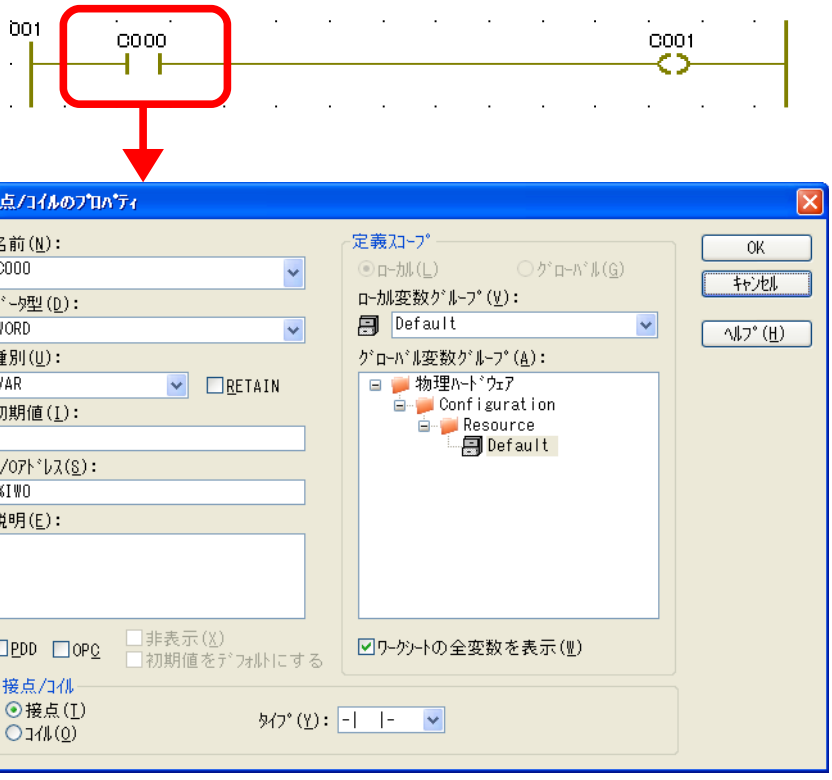
1. プロジェクトツリーから［main］POU のコードワークシートを開きます。



2. コードワークシート上をクリックして［+］カーソルを置き、ツールバーの［LD 回路を挿入］アイコン を クリックします。LD 回路が挿入されます。

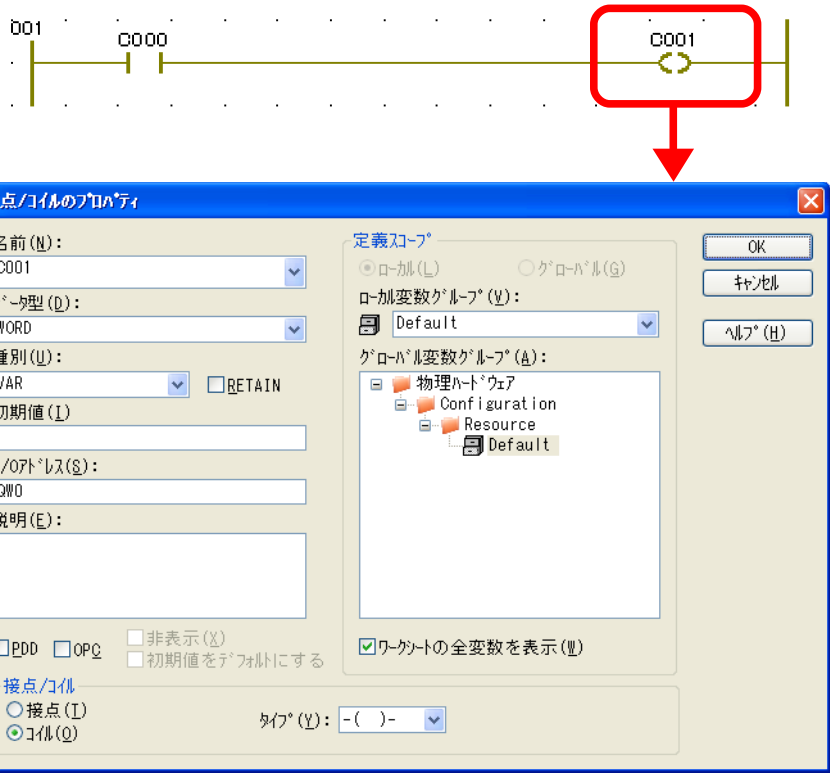


3. LD 回路の 接点：C000 をダブルクリックして、次のように設定して［OK］ボタンをクリックします。



名前	「**C000**」とします。
データ型	2 バイト分の入力データをこの変数で取得するので「**WORD**」を選択します。
種別	ローカル変数「**VAR**」を選択します。
I/O アドレス	2 バイト分の入力データを取得するので「**%IW0**」と入力します。

4. LD 回路の コイル：C001 をダブルクリックして、次のように設定して［OK］ボタンをクリックします。



名前	「**C001**」とします。
データ型	2 バイト分の出力データをこの変数で設定するので「**WORD**」を選択します。
種別	ローカル変数「**VAR**」を選択します。
I/O アドレス	2 バイト分の出力データを設定するので「**%QW0**」と入力します。

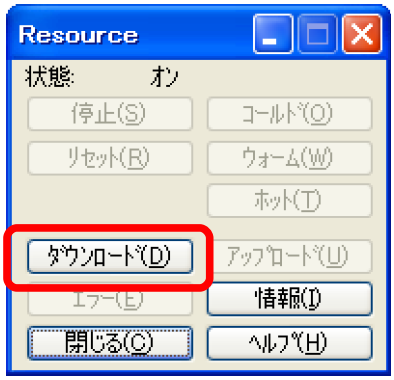
5. ツールバーの [メイク] アイコン  をクリックして、プロジェクトのコンパイルを行います。

コンパイルが正常に完了したら、プログラムコードの作成は完了です。

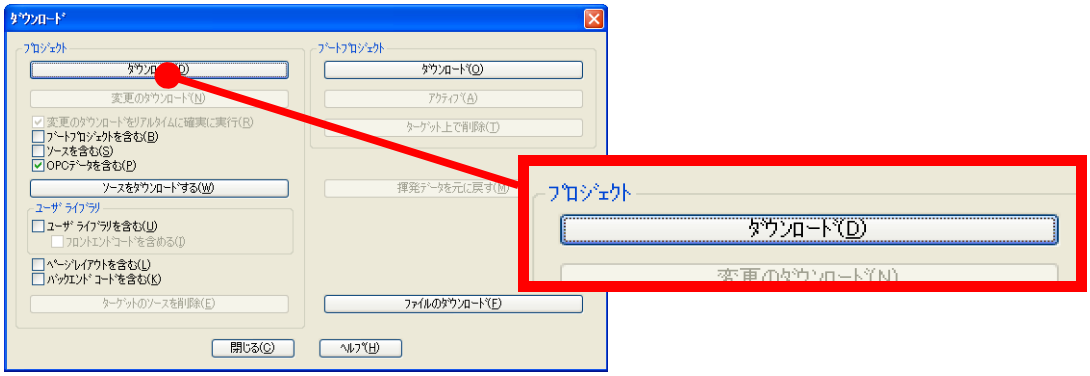
4) プロジェクトの実行・モニタリング

プロジェクトを INplc-Controller にダウンロードして実行し、モニタリングを行います。

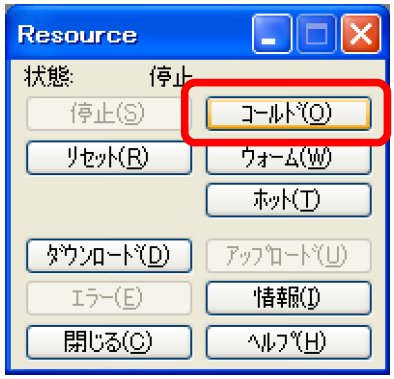
1. オンライン メニューの [プロジェクト コントロール] を選択してリソースダイアログを開き、［ダウンロード］ボタンをクリックします。



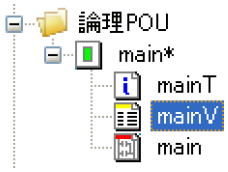
2. ダウンロードダイアログが表示されるので、［プロジェクト］の［ダウンロード］ボタンをクリックします。



3. ダウンロードが完了したら、リソースダイアログの［コールド］ボタンをクリックして、PLC プログラムを稼動状態にします。



4. ツールバーの [デバッグのオン/オフ] アイコン をクリックしてオンライン モードに切り替えて、［main］POU のローカル変数ワークシートを表示します。



5. ローカル変数ワークシートから、オンライン値を見ることができます。
通信用共有メモリの読み込み位置に値が入ると、接点：C000 から コイル：C001 にデータが受け渡され、通信用共有メモリの書き込み位置に値が設定されます。
例えば、アプリケーションから通信用共有メモリ「PRO_TCP_IP」の 1 バイト目に 1 を設定すると、の 2048 バイト目に 1 が反映されます。

	名前	オンライン値	型
	⊟ **Default**		
	C000	16#00B4	WORD
	C001	16#00B4	WORD

付　録

■ ドライバの制限

- TCP/IP 通信に使用する通信ポートは、デフォルト［**9090**］です。
- 通信用共有メモリのサイズは、デフォルト［**4096**］バイトです。

■ 起動時・停止時・停電時の入出力状態

起動時、停止時、停電時の入出力状態は以下のとおりです：

INplc コントローラの状態	共有メモリの状態
INplc コントローラ起動直後 （BootProjecrt なし）	入出力なし状態
INplc コントローラ起動直後 （BootProjecrt あり）	ダウンロードされたプログラムの実行状態の入出力
停止時 （Stop 時の出力設定：出力を保持）	ダウンロードされたプログラムの実行状態の入出力
停止時 （Stop 時の出力設定：出力をクリア）	入出力なし状態
停電時	入出力なし状態

■ 外部アプリケーションサンプル

Winsock を使って INplc と通信を行う Windows アプリケーションのサンプルコードを記載します。

```
#include "stdafx.h"
#include <stdlib.h>
#include <string.h>
#include <windows.h>
#include <winsock2.h>

// 要求伝文(ヘッダ)
typedef struct _SOCK_REQ_COMMAND_INFO
{
  char        SubHead[12];            // サブヘッダ
  WORD        ReqCode;                // 要求コード
  WORD        StartAddr;              // 開始アドレス
  WORD        ByteSize;               // バイトサイズ
  WORD        Reserve[7];             // 予備
} *LPREQCMDINFO, REQCMDINFO;

// 応答伝文(ヘッダ)
typedef struct _SOCK_RES_COMMAND_INFO
{
  char        SubHead[12];            // サブヘッダ
  WORD        ResCode;                // 応答コード
  WORD        Reserve;                // 予備
} *LPRESCMDINFO, RESCMDINFO;


void memdump(void * data,size_t size)
{
  UINT        lp1,lp2;
  char        buff[80];
  char        line[80];

  for(lp1=0;lp1<(size+15)/16;lp1++){
      line[0]='\0';
      sprintf(buff,"%03X0: ",lp1);
```

```
        strcat(line,buff);
        for(lp2=0;lp2<16;lp2++){
            if((lp1*16+lp2)<size){
                sprintf(buff,"%02X ",*((BYTE *)data+lp1*16+lp2));
                strcat(line,buff);
            }
            else
                strcat(line,"   ");
        }
        for(lp2=0;(lp2<16)&&((lp1*16+lp2)<size);lp2++){
            if((*((BYTE *)data+lp1*16+lp2)>=32)&&(*((BYTE *)data+lp1*16+lp2)<=127)){
                sprintf(buff,"%c",*((BYTE *)data+lp1*16+lp2));
                strcat(line,buff);
            }
            else
                strcat(line,".");
        }
        strcat(line,"\x0d\x0a");
        printf(line);
    }
}


int main(int argc, char* argv[])
{
    int                 i;
    int                 iID = 0;
    int                 iVal;
    char                szKey[32];
    char                cIp[20];
    BYTE                bData[0xFFFF];
    WORD                wPort;
    WSADATA             wsaData;
    SOCKET              sock;
    struct sockaddr_in  SockAddr;
    REQCMDINFO          stReqCmd;
    RESCMDINFO          stResCmd;

    printf("--------------------------\n    TcpIpDrv.rsl sample\n--------------------------\n");
    while (1)
    {
        printf("\nSet menu (10:Menu list) >> ");

        gets(szKey);
        if ( szKey[0] == 0x00 ){
            continue;
        }

        iID = atoi(szKey);
        memset(&stReqCmd, 0, sizeof(stReqCmd));
        // サブヘッダの設定
        strcpy(stReqCmd.SubHead, "PRO_TCP_IP");

        switch(iID){
            case 10:    // メニューリスト表示
                printf( "** [TcpIpDrv_Test Menu] ****************************\n" );
                printf( "** ID NAME          | ID NAME          \n" );
                printf( "**  1 connect       |  2 discoonect    \n" );
                printf( "**  3 read          |  4 write         \n" );
                printf( "** 99 End \n" );
                break;

            case 1:     // connect
                printf("Server IP address  --> ");
                gets(cIp);

                printf("Server Port No     --> ");
                gets(szKey);
                wPort = (WORD)atoi(szKey);

                // WinSockの初期化
                WSAStartup(MAKEWORD(2,0), &wsaData);
                // ソケットの生成
                sock = socket(AF_INET, SOCK_STREAM, 0);
                // ソケットアドレス情報の作成
                memset(&SockAddr, 0, sizeof(SockAddr));
                SockAddr.sin_family             = AF_INET;
                SockAddr.sin_port               = htons(wPort);
                SockAddr.sin_addr.S_un.S_addr   = inet_addr(cIp);

                // connect
                if( SOCKET_ERROR == connect(sock, (struct sockaddr *)&SockAddr, sizeof(SockAddr)) )
```

```
            {
                printf("connect error %x\n", WSAGetLastError());
                closesocket(sock);
                return 0;
            }
            break;

        case 2:     // diconnect
            // 通信の無効
            shutdown(sock, 2);
            // ソケットのクローズ
            closesocket(sock);
            break;

        case 3:     // read
            // ReqCode
            stReqCmd.ReqCode = 1;
            // StartAddr
            printf("StartAddr          --> ");
            gets(szKey);
            stReqCmd.StartAddr = (WORD)atoi(szKey);
            // ByteSize
            printf("ByteSize           --> ");
            gets(szKey);
            stReqCmd.ByteSize = (WORD)atoi(szKey);

            // 要求伝文の送信
            send(sock, (char*)&stReqCmd, sizeof(stReqCmd), 0);
            // 応答伝文の受信
            recv(sock, (char*)&stResCmd, sizeof(stResCmd), 0);
            // データの受信
            recv(sock, (char*)bData, stReqCmd.ByteSize, 0);
            // データの表示
            memdump(bData, stReqCmd.ByteSize);

            break;

        case 4:     // write
            // ReqCode
            stReqCmd.ReqCode = 2;
            // StartAddr
            printf("StartAddr          --> ");
            gets(szKey);
            stReqCmd.StartAddr = (WORD)atoi(szKey);
            // ByteSize
            printf("ByteSize           --> ");
            gets(szKey);
            stReqCmd.ByteSize = (WORD)atoi(szKey);
            // 設定データ値
            printf("Set Value          --> ");
            gets(szKey);
            iVal = atoi(szKey);
            // データの作成
            for(i = 0; i < stReqCmd.ByteSize; i++)
            {
                bData[i] = (BYTE)iVal;
            }

            // 要求伝文の送信
            send(sock, (char*)&stReqCmd, sizeof(stReqCmd), 0);
            // 応答伝文の受信
            recv(sock, (char*)&stResCmd, sizeof(stResCmd), 0);
            // データの送信
            send(sock, (char*)bData, stReqCmd.ByteSize, 0);

            break;

        case 99:    // End
            break;
    }

    // End
    if(iID == 99)
        break;
  }

  // WinSockの終了
  WSACleanup();

  return 0;
}
```

株式会社 マイクロネット