CX2500 Series CODESYS Edition
User Manual
(For Programming)

東京計器株式会社
TOKYO KEIKI INC.
油圧制御システムカンパニー

CCOT-24-016 Rev.1

Contents

1. 安全上の注意

本マニュアルで使用している安全に関する表示の意味は次の通りです。本書に記載した注意事項は、安全上重大な内容を記載していますので、必ず厳守下さい。

なお、ハードウェアに関する安全上の注意については CX2500 機能仕様書(文書番号：CCOT-23-017)を熟読下さい。

 警告	この表示を無視して誤った取扱いをすると、人が死亡、又は重傷を負う可能性が想定される内容が記載されています。
 注意	この表示を無視して誤った取扱いをすると、人が傷害を負ったり、物的損害の発生が想定される内容やその他留意すべき内容が記載されています。

 禁止	この表示は**実施してはいけない**内容が記載されています。
 必須	この表示は本製品を使用する上で**必ず実施する必要がある**内容が記載されています。

 警告  必須	デバッグ中におこなう動作開始/停止や値の強制、並びに出力の強制等は、安全に動作することを操作前の確認を必ずおこなったうえで実施して下さい。確認不十分の場合、操作により事故につながる可能性が有ります。

 注意  禁止	弊社が配布するデバイス定義・機能ドライバファイル及びランタイムファイルを編集し使わないで下さい。CX2500 並びに CODESYS の機能が損なわれたり、誤動作につながる可能性があります。

 警告  禁止	本製品に関する物品の分解や改造等はおこなわないで下さい。破損や誤動作、事故につながる可能性があります。

 警告  必須	本製品の使用に際し、万が一本製品に故障・不具合が発生した場合でもフィールセーフなどの保全機能を機器外部などにユーザー側で必ず持たせて下さい。

 注意  禁止	本製品の仕様内で必ず使用して下さい。製品仕様外での使用は事故につながる可能性がありますので絶対におこなわないでください。

 注意  禁止	通電中に端子に触れたり、配線変更等をしないでください。製品の破損や感電などの事故につながる可能性が有ります。

 注意	CX2500 や CODESYS 等のソフトウェアアップデートにより、製品ソフトの表示・表記が実際のものと本書記載のものが一部異なる場合があります。その際は、実際の表示・表記内容に従って下さい。

2. はじめに

本書は CX2500 シリーズ(以降、本製品と呼称)及び CODESYS®の使用に必要な情報やプログラミングなどについて説明している文書です。

本書および CX2500 については、安全上の注意ならびに下記項目について同意した場合のみご使用下さい。

【厳守項目】

- このマニュアルを熟読して下さい。
  - 本書は本製品を正しく使うための文書のため、必ず最後まで熟読の上で製品を使用して下さい。
- このマニュアルを大切に保管して下さい。
  - 本製品を取り扱う場合、このマニュアルは重要ですので、いつでも参照できるように身近に且つ大切に保管ください。
- このマニュアルを取扱者の手元に届けて下さい。
  - 代理店など、本製品の仲介になる方は、必ずこのマニュアル(URL 情報含む)を実際に取り扱う方の手元に届けて下さい。
- このマニュアルを紛失した場合は直ぐに補充して下さい。
  - 万が一、このマニュアルを紛失した場合は、弊社営業所または購入先までご連絡下さい。
- このマニュアルを断りなく転載することはできません。

【注意項目】

- このマニュアルは予告なく変更する場合があります。
- このマニュアルに記載されている CODESYS の画面表示等はバージョンアップによって一部異なる場合があります。その際は、実際の画面表示で確認下さい。
- CODESYS は 3S-Smart Software Solutions GmbH の登録商標です。
- NXP、NXP ロゴは NXP B.V の商標です。
- 上記以外で、このマニュアルに記載されている会社名、製品名はそれぞれ弊社もしくは第三者の商標や登録商標です。
- 本書および CX2500 仕様の内容を逸脱して製品を使用したことによって生じた不具合故障やその他損害について、弊社は一切責任を負いません。
- 弊社は、本ソフトウェアについて、ソフトの使用やバグ、誤動作や不具合、その他本ソフトウェアにより生じた損害についても一切責任を負いません。
- 弊社は本書に記載されている情報に誤りがないことを保証するものではありません。本書に記載された情報の誤りにより、ユーザーまたは第三者に損害が生じた場合においても弊社は一切責任を負いません。

3. 関連マニュアル

本製品のマニュアルの一覧は下記の通りです。各種目的に応じてご使用下さい。

Table 1 CX2500 シリーズ マニュアル一覧

名称	摘要
CX2500Codesys_UserManual_ForSetup	CODESYS のインストールとそのセットアップ、CX2500 の接続方法などについて記載しています。
CX2500Codesys_UserManual_ForProgramming（本書）	**本製品を扱う上での、CODESYS でのプログラミングやデバッグの基礎的な内容、CX2500 に搭載されている各種機能について記載しています。**

4. 使用可能言語について

CODESYS では、国際電気標準会議(IEC)が制定した IEC61131-3 に規定されている PLC のプログラミング言語で開発することができます。使用できる言語は全部で 5 種類(Table 2)あり、ユーザーが作成するアプリケーションの用途に合わせてそれぞれ選択し使用することができます。

Table 2 IEC61131-3 のプログラミング言語※1

プログラミング言語	記述形式	特長
ラダー図言語 (LD 言語)	グラフィック	従来から日本国内で最も用いられている言語であり、シーケンス処理の記述に適している。
命令リスト言語 (IL 言語)	テキスト	アセンブラに似たローレベル言語であり、高速に動作する小さいモジュールを記述するのに適している。
構造化テキスト言語 (ST 言語)	テキスト	パソコン用プログラム言語に似たテキスト言語で、数式演算の記述又は IF…THEN 文並びに FOR…DO 文のような分岐制御の記述に適している。
ファンクションブロック言語 (FBD 言語)	グラフィック	FB 同士のピンをつなぎ合わせて電子回路図のように記述できる言語であり、連続したデータ処理の記述に向いている。最近では、ラダー図言語と混在記述できるプログラミングツールもある。
シーケンシャルファンクションチャート言語 (SFC 言語)	グラフィック	状態遷移に基づくフローチャートであり、順序制御処理の記述に向いている。

※1 出典：JEMA「PLC アプリケーションの開発効率化検討」

Table 3 プログラミング言語の得意・不得意※1

主な使用状況	LD 言語	IL 言語	ST 言語	FBD 言語	SFC 言語
単純なリレーシーケンス処理	◎	×	△	△	×
数式演算処理	△	×	◎	○	×
状態遷移に基づく順序制御 (ステップシーケンス処理)	△	×	○	×	◎
連続的なアナログ信号処理	△	×	○	◎	×
複雑な情報処理	△	×	◎	△	×
プログラムメモリ制約の厳しい場合	○	◎	△	△	×
最も高速に性能を求められる場合	○	◎	○	○	×
運転方案と対応がとりやすい表現	×	×	○	○	◎
動作を視覚的に確認したい場合	○	×	×	◎	○
注記　記号の意味は、次による。 ◎：最も適している、○：適している、△：困難な場合もある、×：適さない					

5. メイン画面について

CODESYS-IDE のメインウィンドウについて解説します。下図はプロジェクト(使用言語：ST)を開いたときの基本的な画面です。なお、各部の位置や表示可否については、ユーザーが自由にカスタマイズできます。

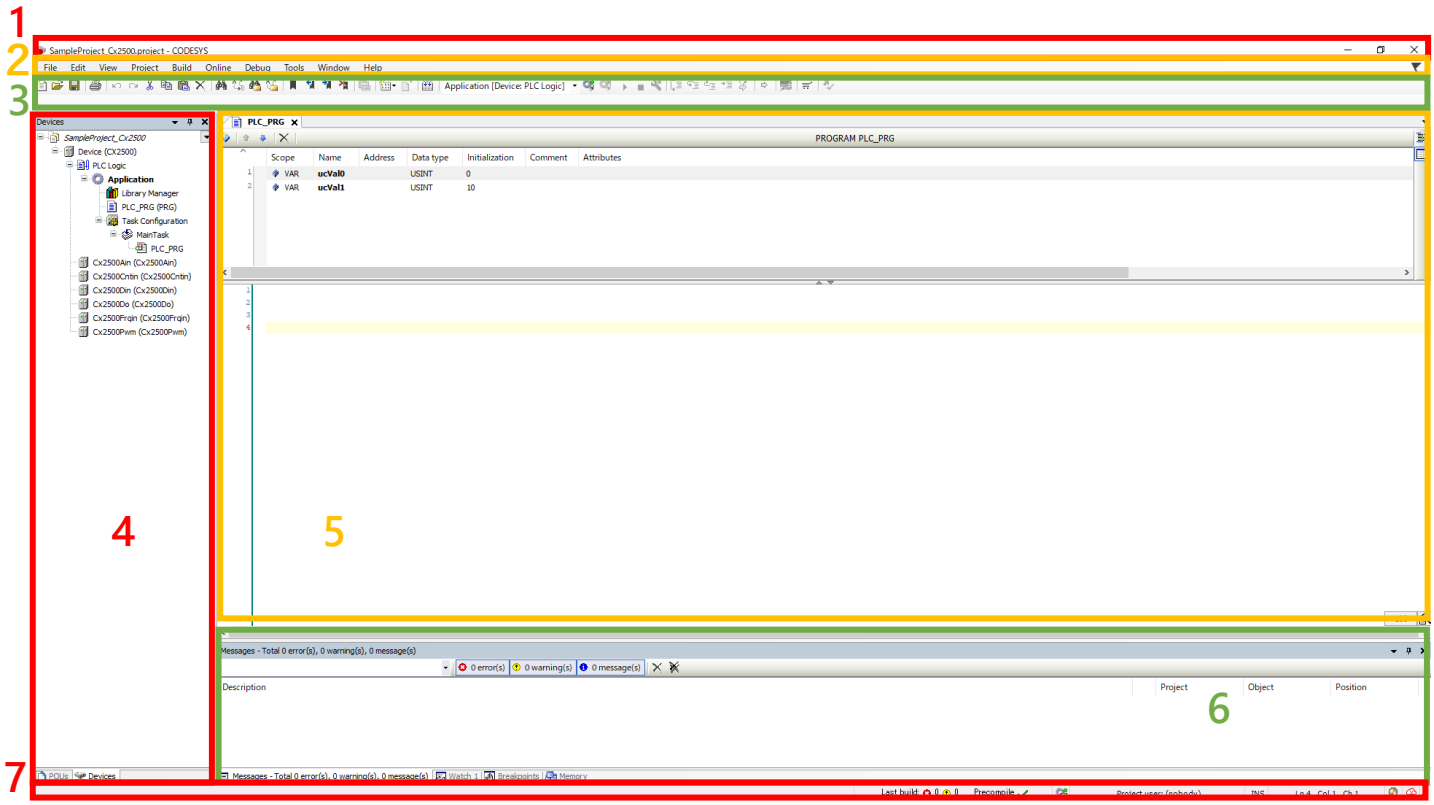


Figure 1 メイン画面

Table 4 メイン画面 機能概要

#	名称	摘要
1	タイトルバー	このエリアには、画面全体のサイズ変更や CODESYS の終了ができる機能があります。
2	メニューバー	このエリアには CODESYS の各種コマンドがあり、目的別にメニュー表示されています。
3	ツールバー	このエリアには、各種コマンドがアイコンで表示されています。アイコンを押すとそれぞれに割り当てられたコマンドがはたらきます。
4	デバイスウィンドウ	このエリアには、現在編集しているプロジェクトのデバイスやプログラム等がツリー状に一覧表示されています。
5	エディタウィンドウ	このエリアでは、変数宣言・プログラムの編集等、ユーザーアプリケーションの編集作業をおこなうことができます。
6	メッセージ・デバッグ情報ウィンドウ	このエリアでは、ビルド結果の詳細やブレークポイントやウォッチ等のデバッグ情報が表示されます。
7	ステータスバー	このエリアには、ビルド結果やアプリの運転状態等の情報が表示されます。

6. プログラミング

6.1. メモリ容量

CX2500(CODESYS 版)について、ユーザーがアプリケーション開発に使用できるメモリは下記の通りです。

Table 5 CX2500(CODESYS 版) ユーザー用メモリ容量

種別	名称	摘要	メモリ容量
ROM	Memory area 0	アプリケーションのソース・実行コード保存領域	1MB
RAM	Memory area 1	アプリケーションの通常変数データ保存領域	84KB
RAM※2	Memory area 2	アプリケーションの保持変数※2 データ保存領域	4KB
	Memory area 3	アプリケーションの持続変数※2 データ保存領域	8KB

※2 保持変数・持続変数については、6.8.12 項を参照。

6.2. デバイスの I/O 更新設定

デバイスの I/O(入出力)データ更新に関する設定をおこないます。I/O とは、本製品において主に Table 6 が該当します。入力データはタスクの始めに、出力データはタスクの終わりにそれぞれ更新がおこなわれます。

そのデータ更新に関する設定をおこないます。設定画面は、デバイスウィンドウから「Device(CX2500)」をダブルクリックし、表示される「Device」→「PLC Settings」(Figure 2)になります。各設定項目については Table 7 を参照下さい。

Table 6 CX2500 I/O

機能名	参照先
デジタル入力	7.3 節
イグニッション入力	7.3 節
周波数入力	7.4 節
2 相カウンタ入力	7.5 節
アナログ入力	7.6 節
内部電源電圧監視入力	7.7 節
基板温度監視入力	7.8 節
デジタル出力	7.9 節
PWM 出力	7.10 節

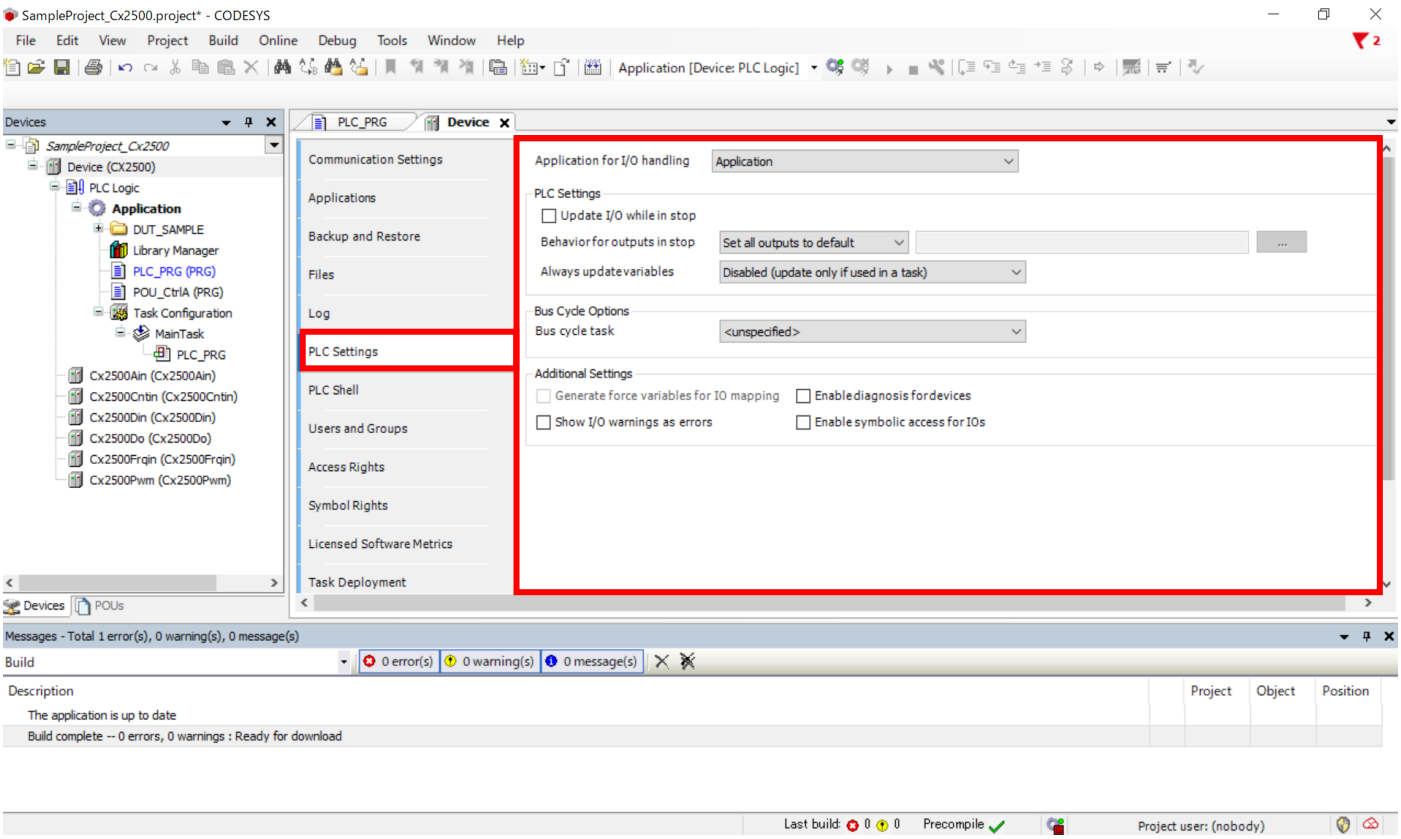


Figure 2 Device PLC Settings

Table 7 PLC Settings 設定項目

<table>
<tr><th>項目</th><th>デフォルト</th><th>摘要</th></tr>
<tr><td>Application for I/O Handling</td><td>「Application」</td><td>I/O データ更新を担当するアプリケーション名。<table><tr><th>選択肢</th><th>摘要</th></tr><tr><td>Application</td><td>ユーザーアプリケーションで I/O 更新処理をおこなう。</td></tr><tr><td>None</td><td>ユーザーアプリケーションで I/O 更新処理をおこなわない。</td></tr></table></td></tr>
<tr><td>PLC Settings Update I/O while in stop</td><td>無効(チェック無し)</td><td>有効時、PLC(CX2500)が動作停止している時も I/O データを更新し続ける。無効時は PLC 動作時のデータで停止される。</td></tr>
<tr><td>Behavior for outputs in stop</td><td>「Set all outputs to Default」</td><td>PLC が動作停止している時の出力状態を設定する。<table><tr><th>選択肢</th><th>摘要</th></tr><tr><td>Keep current values</td><td>動作停止時の値を保持する。</td></tr><tr><td>Set all outputs to default</td><td>I/O が各機能ドライバで設定したデフォルト値にする。</td></tr><tr><td>Execute program</td><td>停止時の出力制御を指定したプログラムで実行する。</td></tr></table></td></tr>
<tr><td>Always update variables</td><td>「Disabled」</td><td>有効時、全ての I/O データをプログラムの使用有無に関わらず更新する。無効時は各 I/O の機能ドライバでデータを更新するか設定する必要がある。</td></tr>
<tr><td>Bus Cycle Options</td><td>「<unspecified>」</td><td>バスサイクルタスクを制御するタスクを選択する。「<unspecified>」を選択した場合は、最も短い周期のタスクになる。</td></tr>
<tr><td>Additional Settings</td><td>全て無効</td><td><table><tr><th>項目</th><th>有効化(チェック有)時の効果</th></tr><tr><td>Enable diagnosis for devices</td><td>有効にすると、ライブラリ「CAADeviceDiagnosis※3」を用いたデバイス診断が可能となる。</td></tr><tr><td>Show I/O warnings as errors</td><td>ビルド時、I/O 構成に関する警告をエラーとして表示する。</td></tr><tr><td>Enable symbolic access for IOs</td><td>各 I/O に変数を自動的に作成し割り当てる。</td></tr></table></td></tr>
</table>

※3 ライブラリ「CAADeviceDiagnosis」とそのデバイス診断機能については、CODESYS オンラインヘルプを参照下さい。

6.3. プログラム作成フロー

プログラムの作成の流れの例を下記に示します。

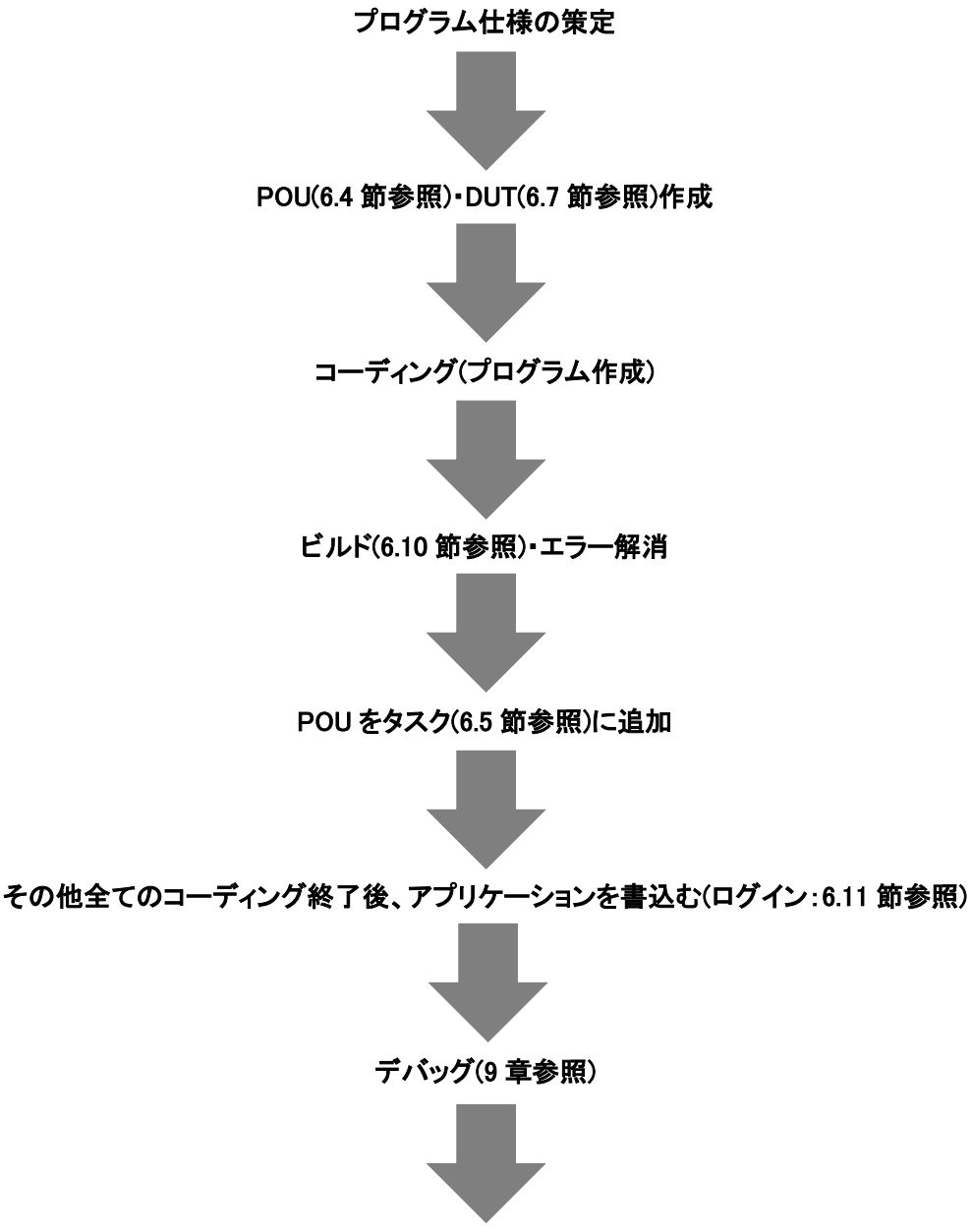


Figure 3 プログラム作成フロー

6.4. POU

POU(Program Organization Unit)は、プログラムの単位を指します。ユーザーはこの POU を作成し(1 つはプロジェクト作成時に自動生成される)、そこにプログラム(処理)を記述していきます。従来のラダーと異なり、IEC61131-3 では POU をアプリケーションの機能・制御毎に複数作成することで、特定の処理がどの POU(機能・制御)に割り当てられているのかが分かりやすくなっています。ただし、1 つの POU に対して記述できる言語は 1 種類に限定されることに留意下さい。

POU には以下の通り、3 種類のタイプが有ります。

Table 8 POU タイプ一覧

タイプ	摘要	呼び出し方の例
Program	・タスクに割り当てることで呼び出すことができる。 ・他の POU からも呼び出すことは可能。 ・POU 内の変数値は保持される。	6.5.2 項
Function block	・呼び出す際は、呼び出し元の POU で Function block のインスタンス(変数宣言部で宣言する)を作成する。 ・それぞれのインスタンス内の変数値は保持される。 ・出力を複数持つことができる。	6.9.5.2 項
Function	・呼び出す際は、Function block のように呼び出し元の POU で宣言する必要は無い。コード部から Function 名で直接呼び出せる。 ・Function 内の変数値は全て保持されない。(呼び出す毎に初期値になる) ・出力は 1 つしか持てない。	6.9.4 項

6.4.1. POU の追加方法

ここでは、プロジェクトへの POU の追加例を示します。

① デバイスウィンドウにて、「Application」を右クリックして下さい。

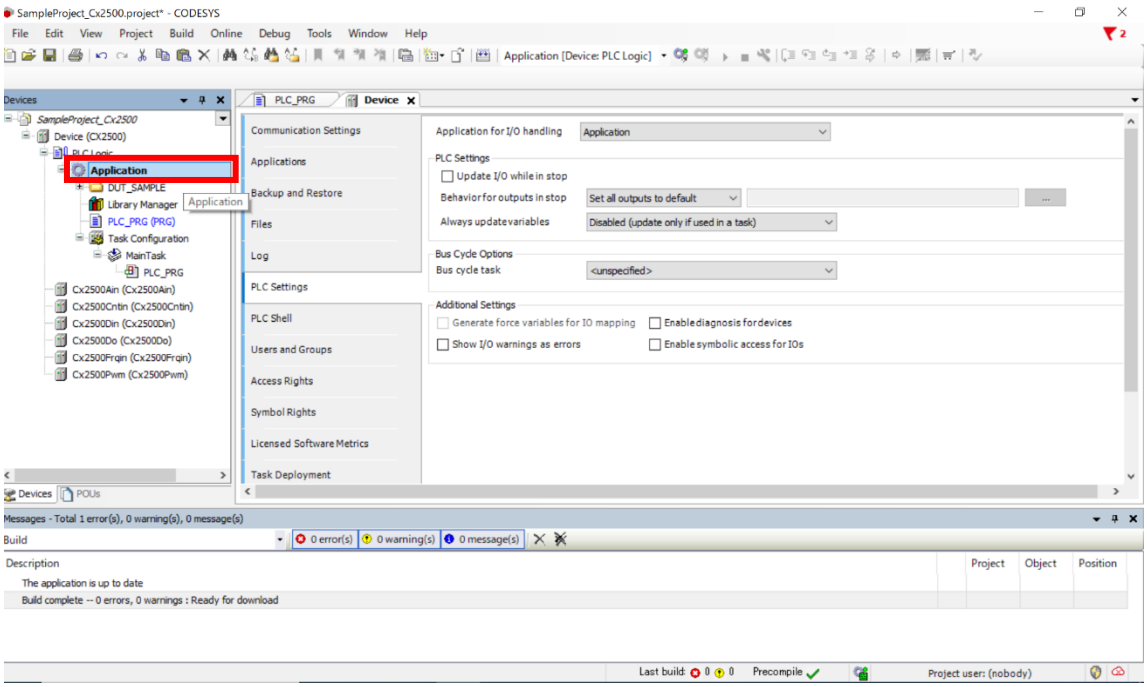


Figure 4 メイン画面 Application の選択

② 表示されたコンテキストメニューから「Add Object」→「POU…」を選択して下さい。

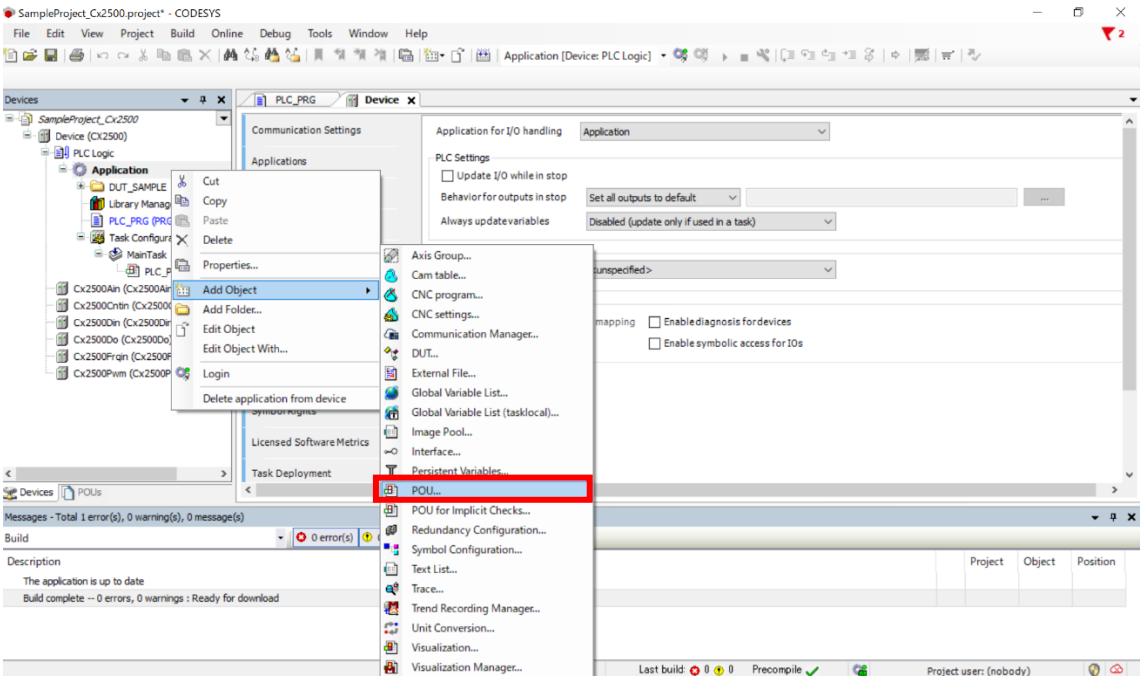


Figure 5 Applcation コンテキストメニュー POU の選択

③ 「Add POU」ウィンドウが表示されるので、以下 3 点を記入・選択し「Add」ボタンを押して下さい。

Table 9 Add POU 設定項目

<table>
<tr><th>設定項目</th><th></th></tr>
<tr><td>Name</td><td>POU 名を入力する。</td></tr>
<tr><td>Type</td><td>以下 3 つのタイプから 1 つを選択する。
<table>
<tr><th>タイプ</th><th>設定値</th></tr>
<tr><td>Program</td><td>特に無し。</td></tr>
<tr><td>Function block</td><td>・Extend : 有効化する(チェックを入れる)と、既に作成されている Function block を拡張することができる。
・Implements : 有効化する(チェックを入れる)と、既に定義されている Function block インターフェースの処理内容を記述できる。
・Final :有効化して(チェックを入れる)作成したものを、別の Function block で拡張できなくする。
・Abstract : 有効化して(チェックを入れる)POU を作成すると、抽象 Function block(入出力のブロックのみ)で作成される。これは上記 Extend 用の Function block として使用されることが一般的。
・Access specifier :以下 2 つから選択する。
(1)INTERNAL:作成した POU 内でのみ呼び出すことができる。
(2)PUBLIC:どの POU からも呼び出すことができる。
・Method implementation language :インターフェースの実装を行う場合の実装言語を選択する。</td></tr>
<tr><td>Function</td><td>Return type(返り値)のデータ型を入力する。</td></tr>
</table></td></tr>
<tr><td>Implementation language</td><td>以下 5 言語から 1 つを選択する。
・Continuous Function Chart(CFC)
・Continuous Function Chart(CFC) – page oriented
・Function Block Diagram(FBD)
・Ladder Logic Diagram(LD)
・Structured Text(ST)
・Sequential Function Chart(SFC)</td></tr>
</table>

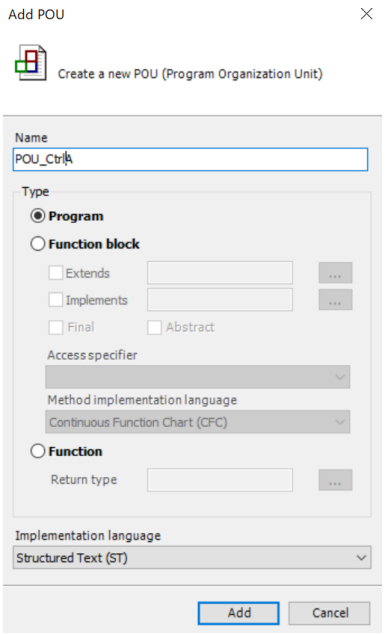


Figure 6 Add POU ウィンドウ

④ 「Application」ツリーに作成した POU が表示されます。

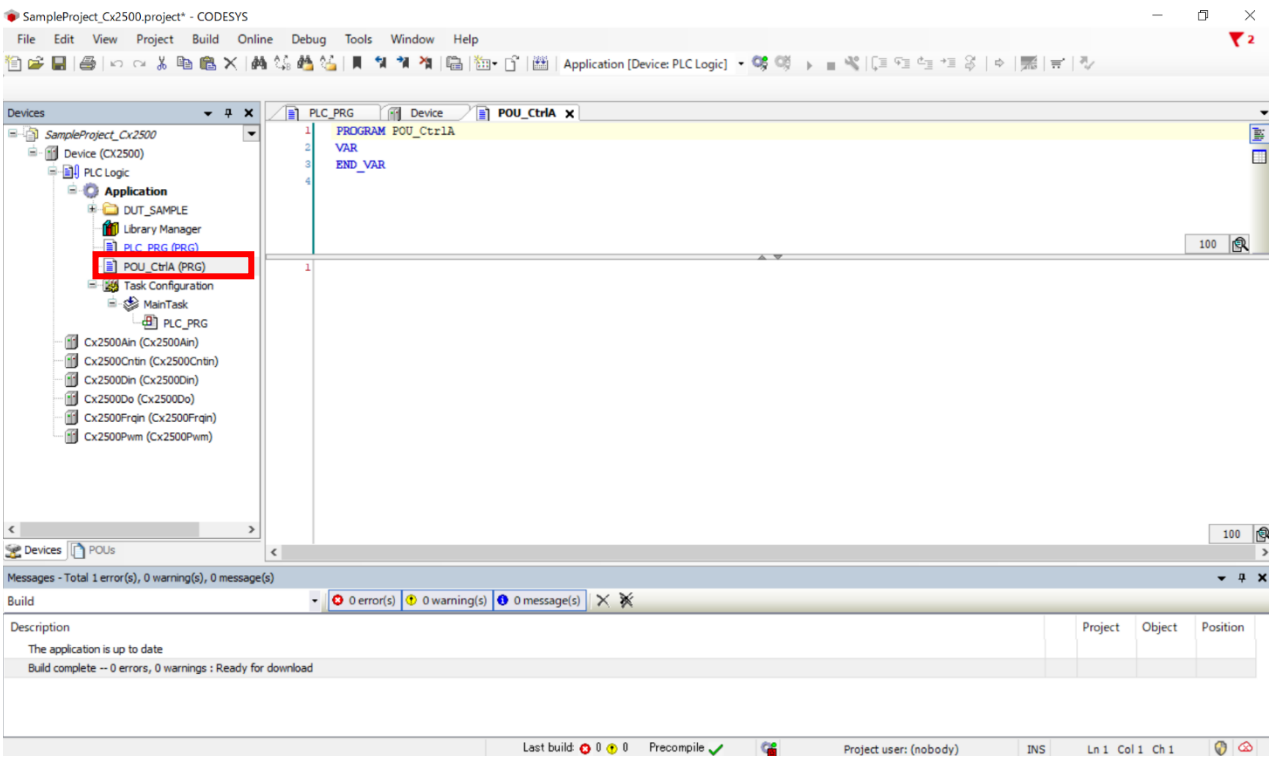


Figure 7 POU 作成後

6.5. タスク

プログラム(POU)を動作させるには、タスクへの割り当てが必要です。タスクは、POU がどのような実行周期・方式で処理するかを管理します。タスクに割り当てられていない POU は実行されません。

タスクには下記のようにタイプが分かれています。なお、最大タスク数は 3 つです。各タスクは、別の実行中のタスクがある場合、それが終わるまで実行されることはありません。タスク周期<タスク実行時間(9.10 節参照)にしてしまうと処理に大きな遅れが出る場合があることに注意して設計して下さい。

Table 10 タスクのタイプ

タイプ	摘要
Cyclic	実行周期が設定でき、その設定した周期でタスクが実行される。
Event	実行イベントとして変数(BOOL 型※4)が設定でき、その変数が 0(FALSE)→1(TRUE)になった時にタスクが 1 度実行される。

※4 BOOL 型については、6.8.1 項を参照下さい。

6.5.1. タスクの追加

タスクの追加方法について下記に示します。

① デバイスウィンドウの「Task Configuration」にカーソルを合わせ、右クリックして下さい。

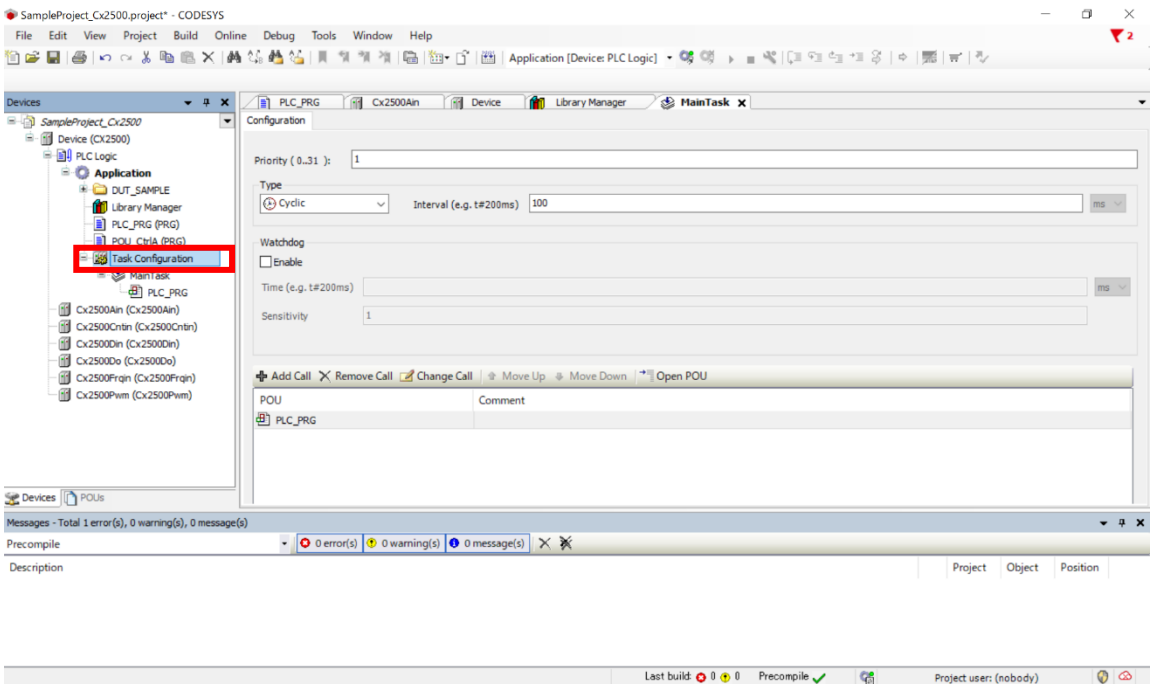


Figure 8 メイン画面 Task Configrarion の選択

② 表示されたコンテキストメニューから「Add Object」→「Task…」を選択して下さい。

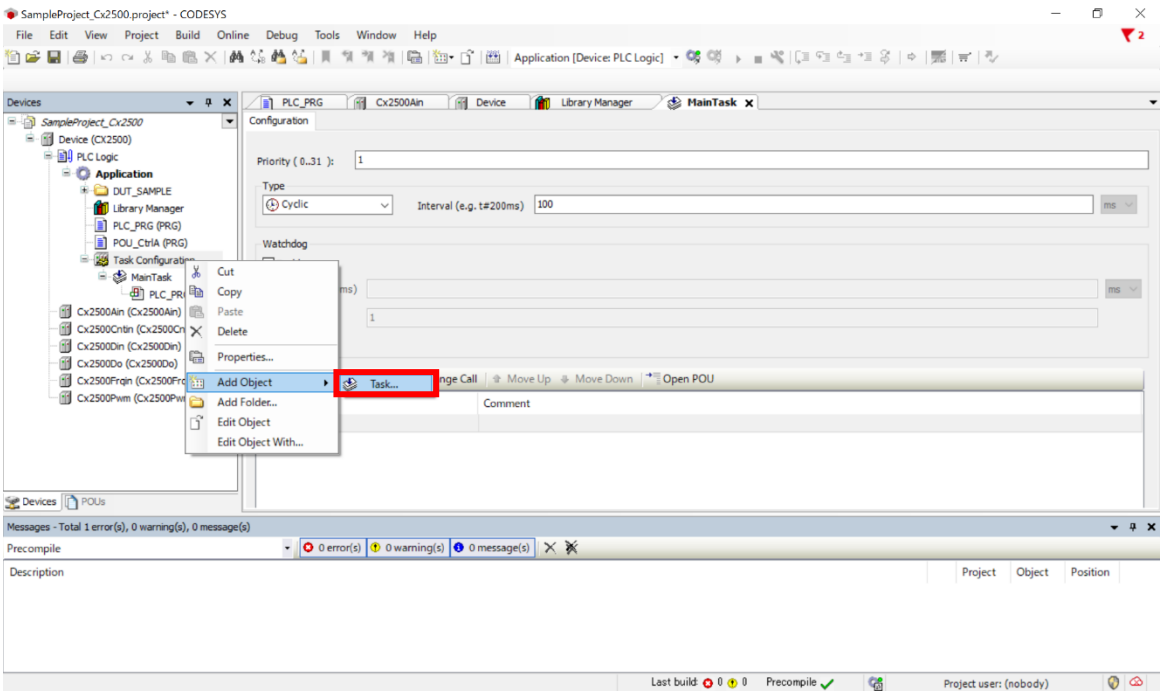


Figure 9 Task Configration コンテキストメニュー Task の選択

③ 「Add Task」ウィンドウが表示されるので、タスク名を入力し「Add」ボタンを押して下さい。

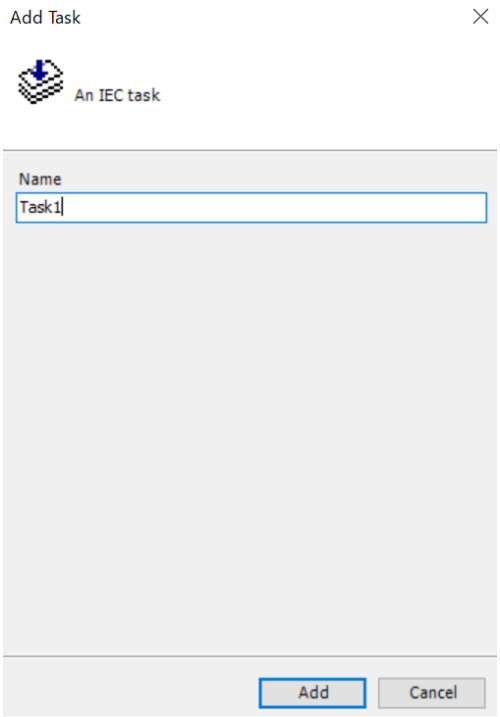
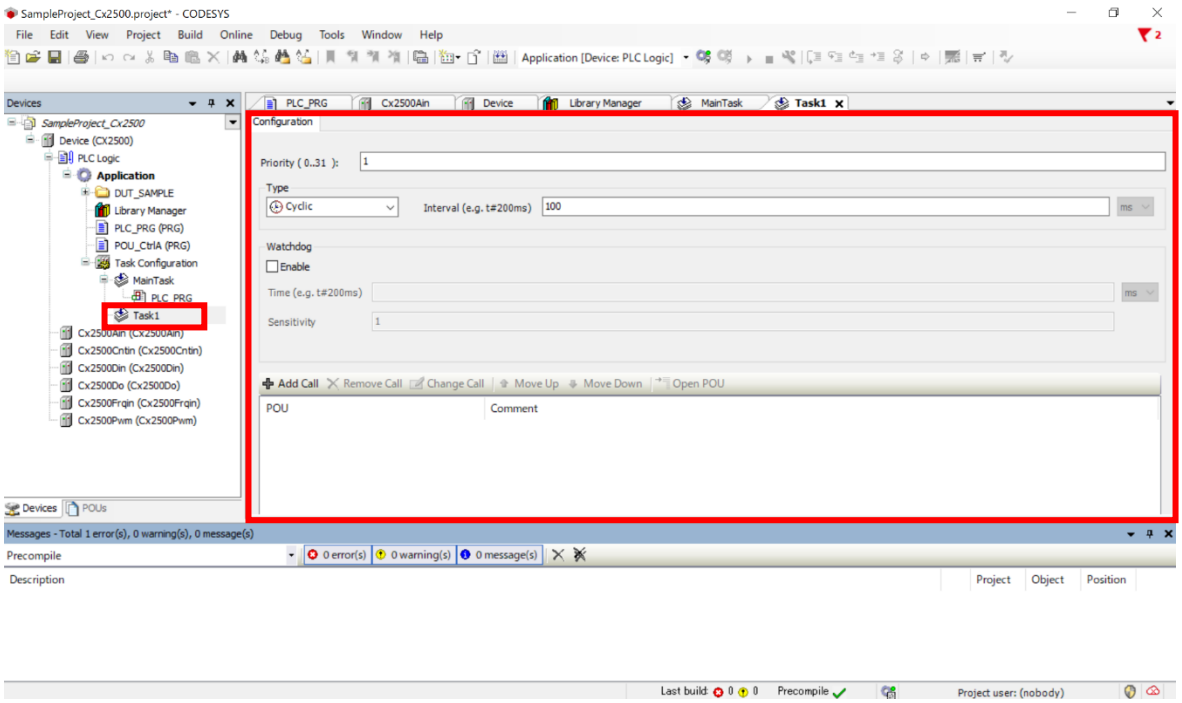


Figure 10 Add Task ウィンドウ

④ タスクが新しく追加され、タスクの設定画面が表示されると完了です。



6.5.2. タスクの設定・POU の割り当て

タスクの設定画面では、以下のような設定項目があります。各種アプリケーションの仕様に合わせ設定して下さい。

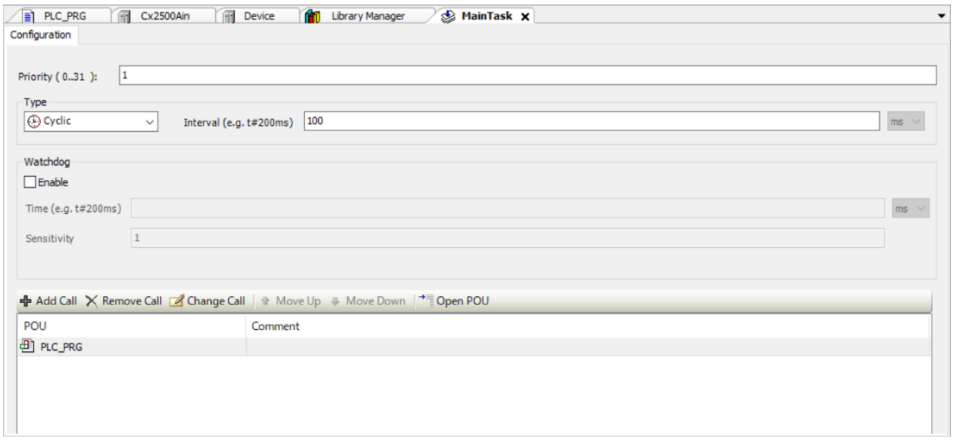


Figure 11 タスク設定画面

Table 11 タスク 設定項目

#	項目	摘要
1	Priority※5	複数のタスクがある場合の当該タスクの実行優先度を設定する。優先度は 0～31 まで設定でき、値が小さいほど優先度は高くなる。
2	Type	タスクのタイプ(Table 10)を設定する。 タイプ: 設定値 Cyclic: Interval(周期時間)を設定する。単位は ms。 Event: Event を設定する。Event に設定できる変数はユーザーが定義した BOOL 型変数に限られる。
3	Watchdog	タスクのウォッチドッグタイマを設定できる。詳細は 6.6 節を参照。
4	POU Settings	タスクに割り当てる POU を設定できる。1 つのタスクに POU を複数割り当てた場合、この実行リストの最上段にある POU から順番に実行される。

※5 本製品はマルチタスク方式ではありません。優先度が高いタスクでも、実行中の別のタスクが終わるまで実行されないことに留意して下さい。

Table 12 POU 割り当て コマンド一覧

コマンド名	摘要
Add Call	タスクに POU を追加する。
Remove Call	タスクの実行リストで選択した POU を削除する。
Change Call	タスクの実行リストで選択した POU を別の POU に変更する。
Move Up	タスクの実行リストで選択した POU を 1 段リストアップ(実行順を 1 つ上げる)する。
Move Down	タスクの実行リストで選択した POU を 1 段リストダウン(実行順を 1 つ下げる)する。
Open POU	タスクの実行リストで選択した POU のエディタ画面を開く。

6.6. ウォッチドッグタイマ

ウォッチドッグタイマはプログラム(アプリケーション)が暴走や異常停止していないかを確認するための機能を指します。この機能を有効にして監視時間を設定した場合、プログラムの実行時間が監視時間を超過した時にウォッチドッグエラーとしてプログラムの実行を停止します。

ウォッチドッグタイマの設定はタスク毎におこなう必要が有ります。ウォッチドッグタイマは下図の通り、タスクの設定画面で設定できます。設定項目としては、機能の有/無効の他に Time・Sensitivity が有ります。

ウォッチドッグタイマエラーの条件とその例を Table 13 に示します。

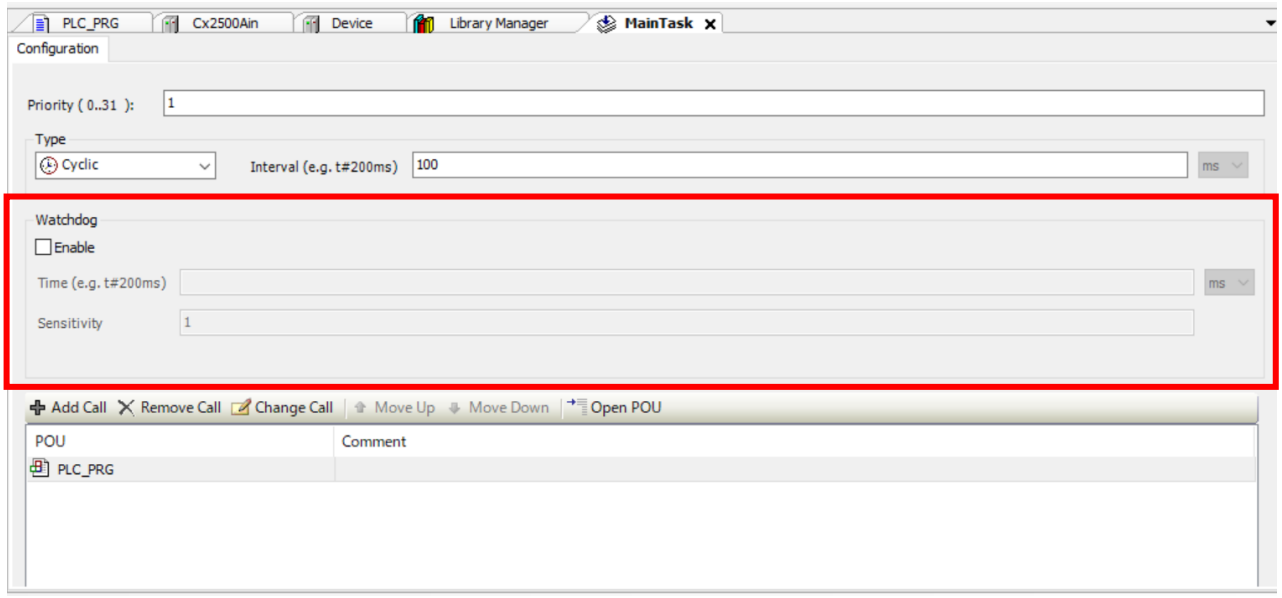


Figure 12 ウォッチドッグタイマ 設定エリア

Table 13 ウォッチドッグタイマ エラー条件とその例

エラー条件	下記いずれかを満たすこと。 (1)プログラムの実行時間が Time※6 × Sensitivity※6 の時間を超えた。 (2)プログラムの実行時間が Sensitivity※6 の回数連続で Time※6 の時間を超えた。
例	Time:20[ms]、Sensitivity:5 を設定した場合のエラー条件は下記いずれかを満たす時。 (1)プログラムの実行時間が 20[ms]×5=100[ms]を超えた。 (2)プログラムの実行時間が 5 回連続で 20[ms]を超えた。

※6 この Time・Sensitivity は、タスク設定画面で Time・Sensitivity に設定した値を指す。

6.7. DUT

DUT(Data Unit Type)は、ユーザーが定義できるデータ型のことです。DUT として定義できる型は以下の通りです。

Table 14 DUT 定義可能なデータ型

データ型	参照先
列挙型	6.8.5 項
型の別名定義	6.8.6 項
構造体	6.8.7 項
共用体	6.8.9 項

6.7.1. DUT の作成方法

DUT の作成方法を下記に示します。

① デバイスウィンドウの「Application」にカーソルを合わせ右クリックして下さい。

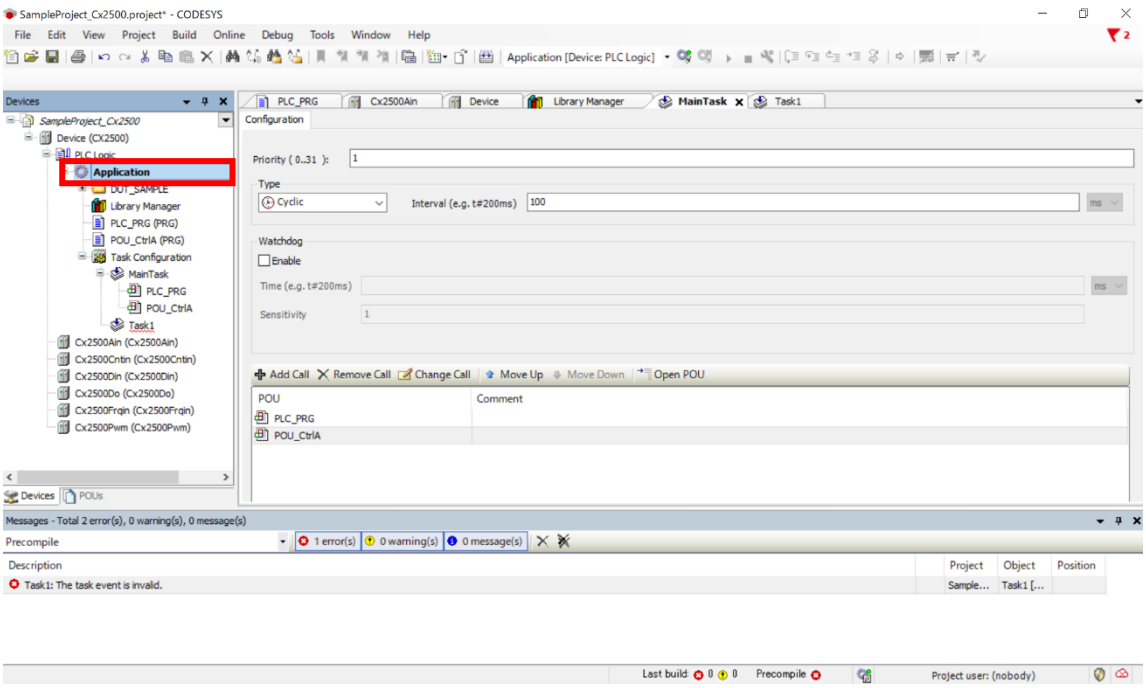


Figure 13 メイン画面 Application の選択

② 表示されるコンテキストメニューから「Add Object」→「DUT…」を選択して下さい。

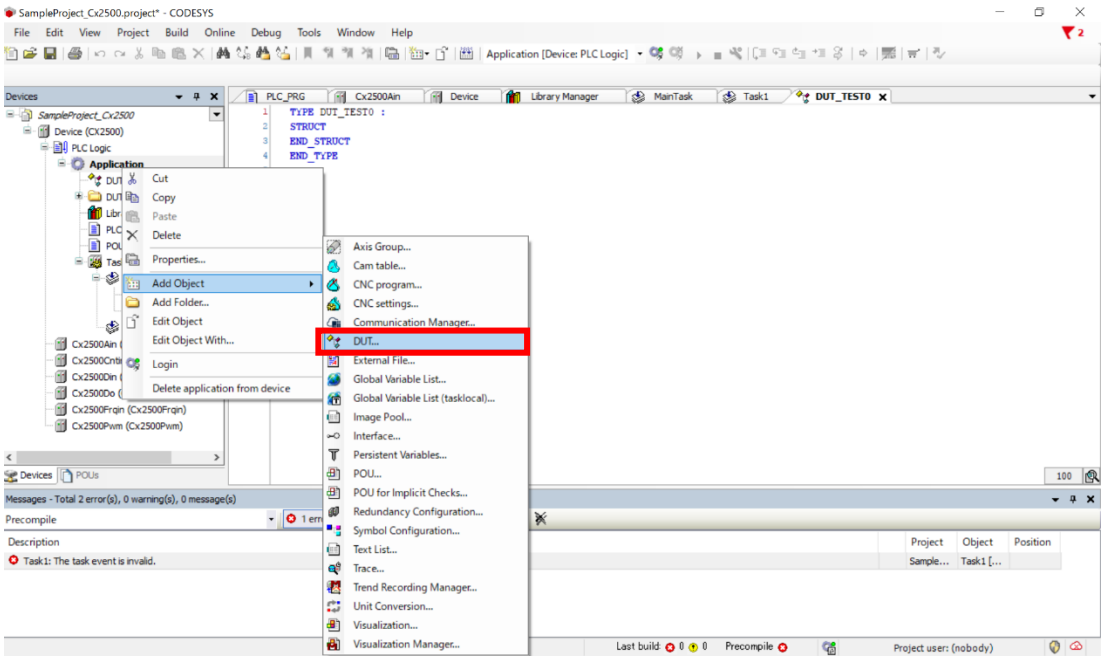


Figure 14 Application コンテキストメニュー DUT の選択

③ 「Add DUT」ウィンドウが表示されるので、DUT の名前と定義したい型を選択し、「Add」ボタンを押して下さい。

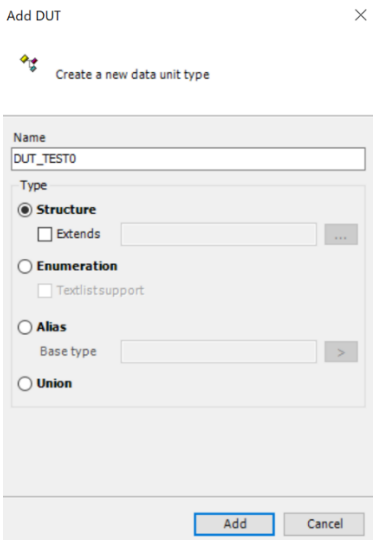


Figure 15 Add DUT ウィンドウ

Table 15 DUT タイプと設定項目

タイプ	設定項目	摘要
Structure (構造体)	Extends	有効化すると、既に定義されている構造体をさらに拡張した構造体を定義できる。対象となる構造体の型を入力するか「...」ボタンを押して型を選択する。
Enumeration (列挙型)	Textlist support	有効化すると、列挙型のメンバが 1 つ追加された状態で定義が作成される。
Alias (型の別名定義)	BaseType	別名定義する型を入力するか「>」ボタンを押して型を選択する。
Union (共用体)	—	型名以外設定項目無し。

④ DUT が新しく追加され、そのエディタウィンドウが表示されれば完了です。エディタウィンドウにて所望のメンバを記述してデータ型を完成させて下さい。

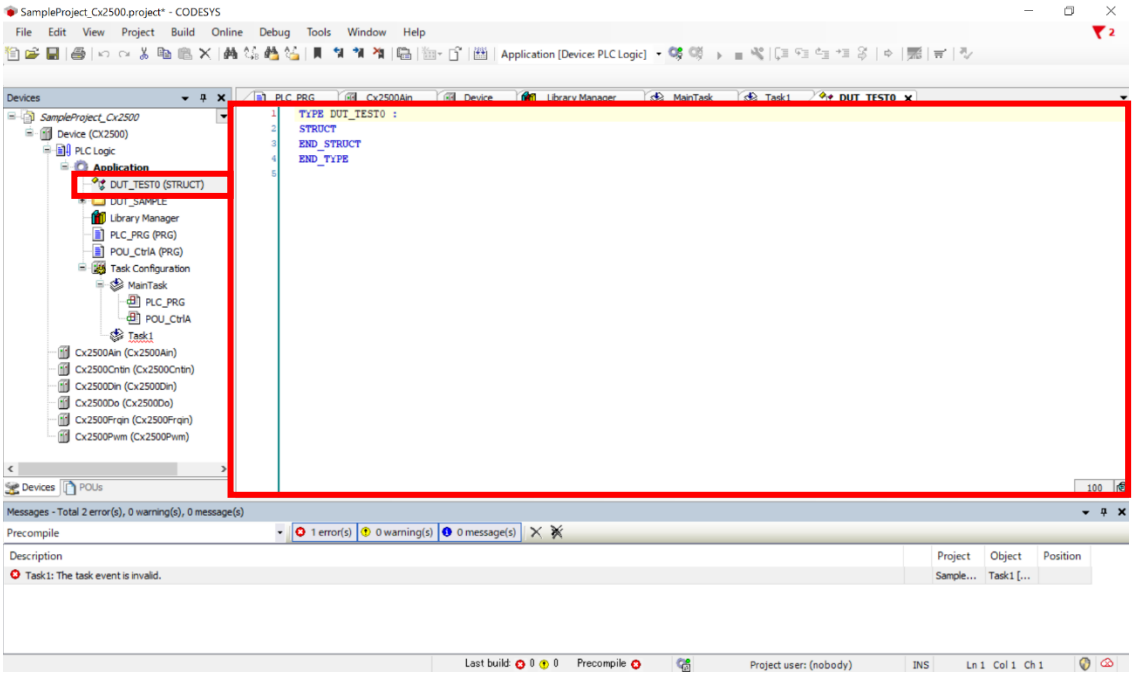


Figure 16　メイン画面　DUT 作成後

6.8. 変数

6.8.1. 一般的な型

変数はデータをメモリに格納するためのもので、ユーザーはそのデータを使って制御(処理)をおこなったりします。変数には「型」があり、この「型」宣言をして変数に格納できる値の範囲を定義する必要が有ります。

Table 16 一般的な変数の型

項目	型名	値の範囲	メモリサイズ[bit]
ブール	BOOL	0、1	1
整数(符号有り)	SINT	-128～127	8
	INT	-32768～32767	16
	DINT	-2^{31}～$2^{31}-1$	32
	LINT	-2^{63}～$2^{63}-1$	64
整数(符号なし)	USINT	0～255	8
	UINT	0～65535	16
	UDINT	0～$2^{32}-1$	32
	ULINT	0～$2^{64}-1$	64
	BYTE	0～255	8
	WORD	0～65535	16
	DWORD	0～$2^{32}-1$	32
	LWORD	0～$2^{64}-1$	64
実数	REAL	-3.402E+38～-1.175E-38 1.175E-38～3.402E+38	32
	LREAL	-1.797E+308～-2.225E-308 2.225E-308～1.797E+308	64
文字列	STRING	(文字数+1)×8	(文字数+1)×8
	WSTRING	(文字数+1)×16	(文字数+1)×16

PLC_PRG

```
1  PROGRAM PLC_PRG
2  VAR
3      bFlg    : BOOL   := FALSE;          //Declaration of "BOOL"
4      ucVal0  : USINT  := 0;              //Declaration of "UINT"
5      rVal    : REAL   := 1.0;            //Declaration of "REAL"
6      sStr    : STRING := 'Tokyokeiki';   //Declaration of "STRING"
7  END_VAR
```

Figure 17 例:一般的な型の宣言

6.8.2. 変数宣言

変数の宣言は POU のエディタ部(下図赤枠内)でおこなう必要があります。 変数の宣言手法としては、主に以下の 3 つがあります。詳細は後述します。

【宣言手法】

- 「Add Variable」で宣言
- 「Textual View」で宣言
- 「Tabular View」で宣言

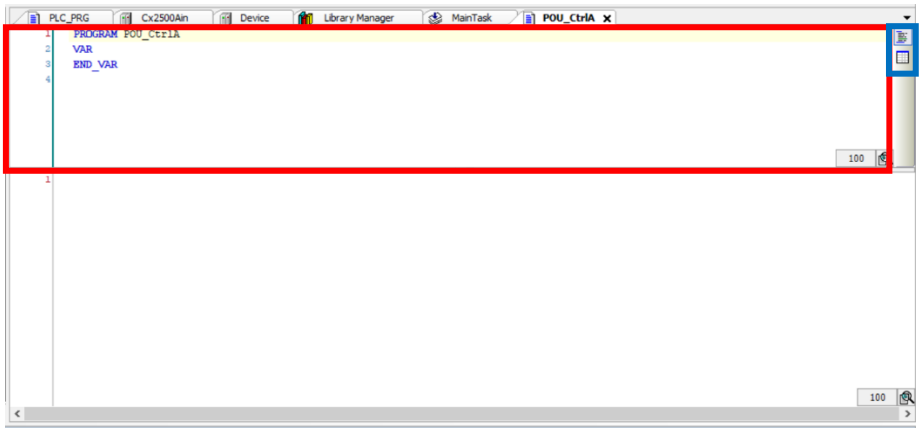


Figure 18 POU エディタ Textual View

Textual・Tabular View の切り替えは青枠のボタンからできます。

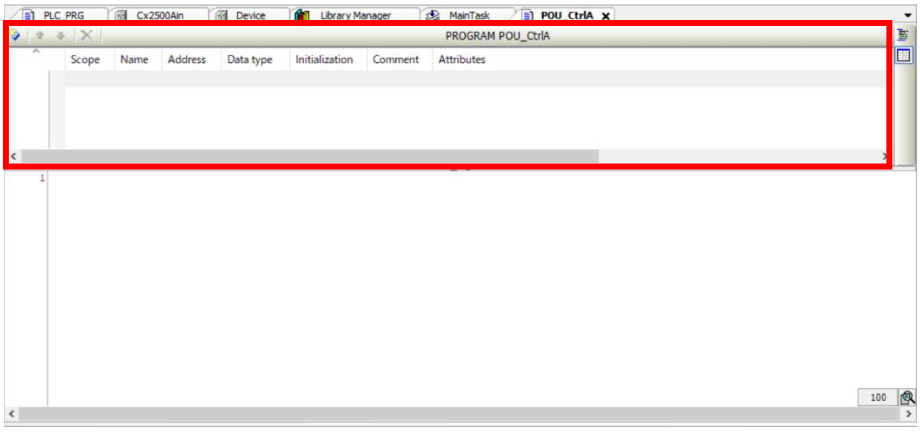


Figure 19 POU エディタ Tabular View

6.8.2.1. 「Add Variable」で宣言

ここでは、「Add Variable」で変数を宣言する手法を記します。

① 変数宣言部で右クリックして下さい。表示されるコンテキストメニューから「Refactoring」→「Add Variable…」を選択します。

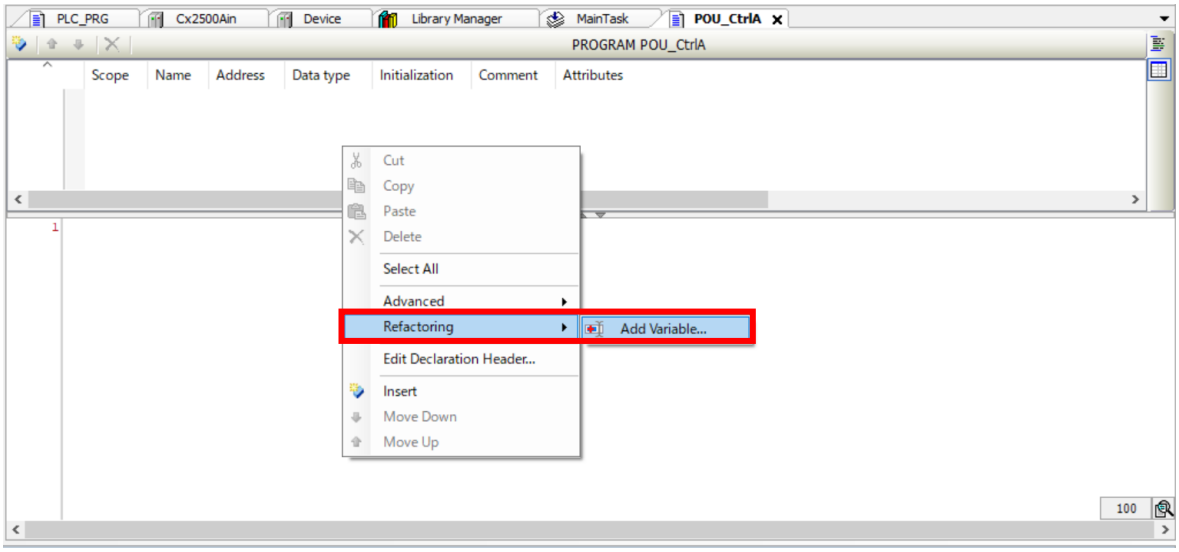


Figure 20 変数宣言部コンテキストメニュー Add Variable

② 「Add variable '??' to '(POU の名前)'」ウィンドウが表示されるので、各設定項目に所望の値を設定し「OK」ボタンを押します。

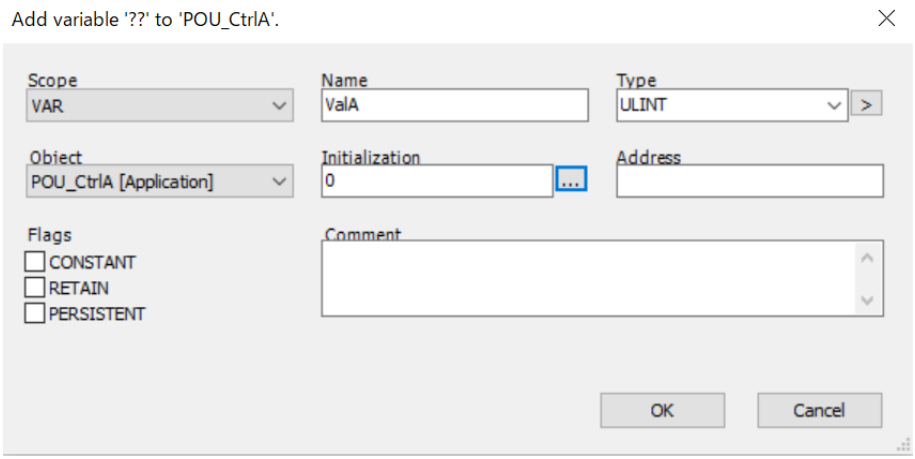


Figure 21 Add variable '??' to '(POU の名前)' ウィンドウ

Table 17 Add Variable 設定項目

設定項目	摘要
Scope	「VAR」を選択する。(ローカル変数の場合)※7
Name	変数名を入力する。
Type	変数の型を入力する。
Object	変数が属する POU 名を選択する。(ローカル変数の場合)※7
Initialization	変数の初期値を入力する。
Address	**値は記入しない。**
CONSTANT	チェックを入れて有効化した場合、この変数は定数として扱われる。定数値は Initialization 欄に入力する。
RETAIN	チェックを入れて有効化した場合、この変数は保持変数※8 として扱われる。
PERSISTENT	チェックを入れて有効化した場合、この変数は持続変数※8 として扱われる。
Comment	変数に関するコメント(メモ)を自由に記入できる。

※7 グローバル変数の場合、宣言の仕方が異なる。6.8.11 項を参照。

※8 保持変数・持続変数については 6.8.12 項を参照。

③ 変数宣言部に変数が追加されます。

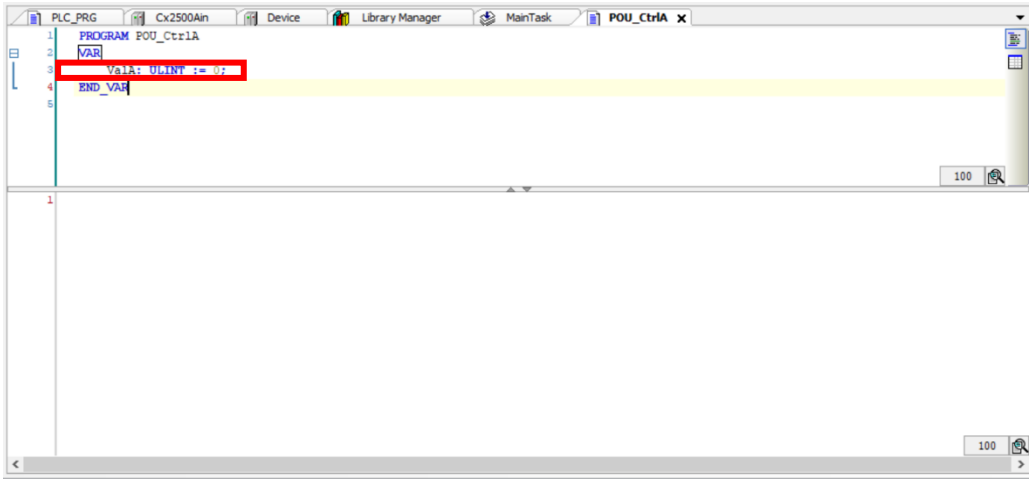


Figure 22 Add Variable 宣言結果

6.8.2.2. 「Textual View」で宣言

ここでは、「Textual View」で変数を宣言する手法を記します。

① 変数宣言部が Tabular View になっている場合は Textual View にして下さい。

② 下図のように、宣言したい変数を直接記入します。これで宣言は完了です。

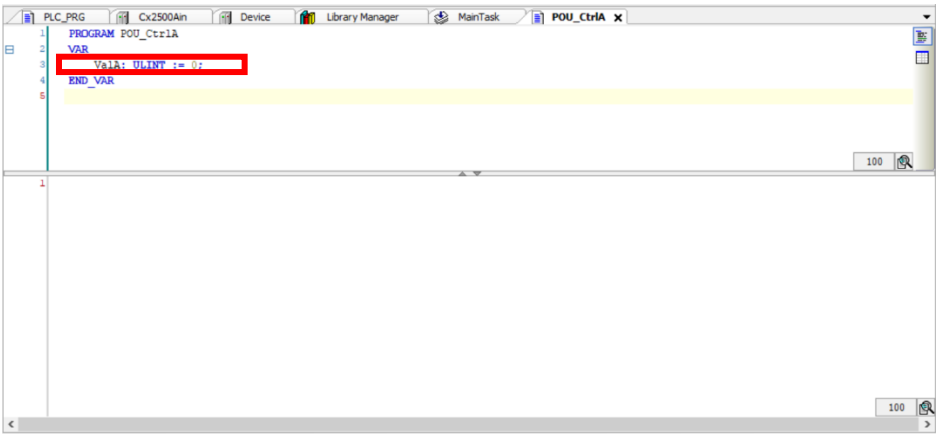


Figure 23 Textual View での宣言

6.8.2.3. 「Tabular View」で宣言

ここでは、「Tabular View」で変数を宣言する手法を記します。

① 変数宣言部が Textual View になっている場合は Tabular View にして下さい。

② 「Insert」ボタンを押して下さい。

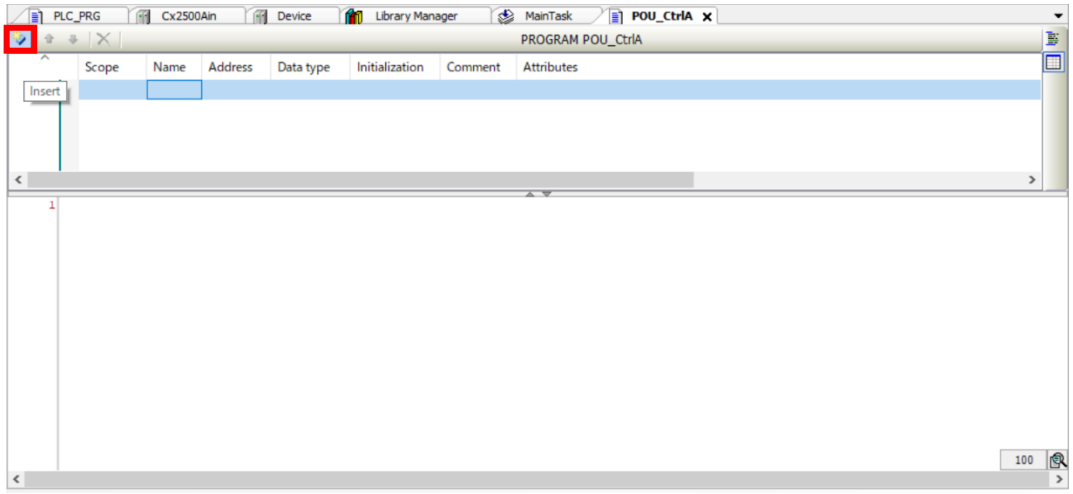


Figure 24 Tabular View Insert の選択

③ 下図のように、仮の変数が表示されるので、各設定項目に所望の値を入力して下さい。

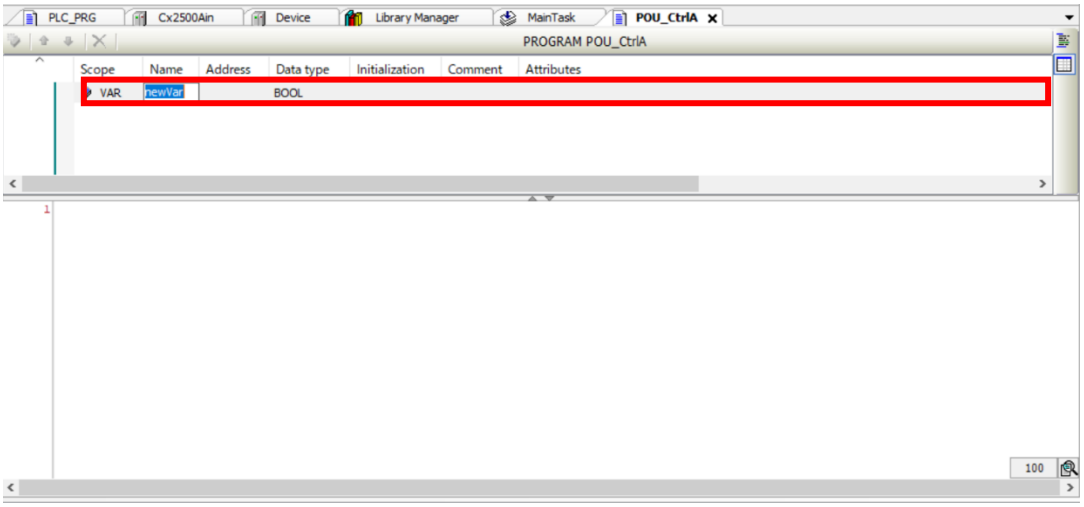


Figure 25 Tabular View 仮変数追加

④ 入力が終わったら変数の宣言完了です。

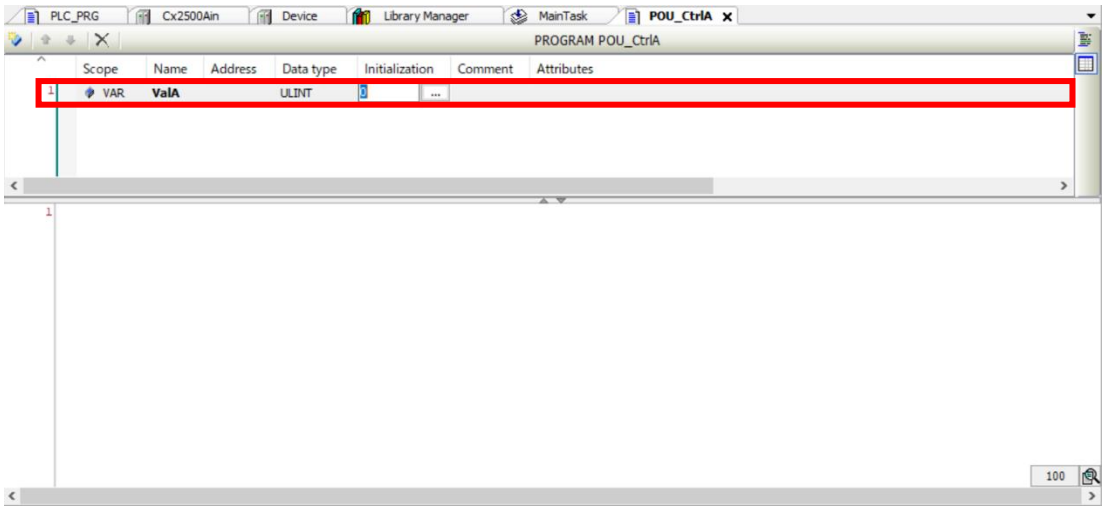


Figure 26 Tabular View 宣言結果

6.8.3. サブレンジ

サブレンジ型で宣言すると、宣言した変数値の範囲を元の型の範囲内よりも絞って指定することができます。

```
11    usBox : UINT(200..1000);          // Declaration of "UINT(range : 200-1000)"
```

Figure 27 宣言例(Textual View):サブレンジ型の宣言(UINT 型で 200～1000 のみ格納可能)

6.8.4. 配列

同じ型のデータを複数格納したい場合、配列を使うと複数のデータを 1 つの変数名称で管理・使用することが可能です。

```
8     usAry : ARRAY [0..10] OF UINT;      //Declaration of "ARRAY(UINT)"
```

Figure 28 宣言例(Textual View):配列の宣言(10 個の INT データ持ちの 1 次元配列)

```
1
2   //Array
3   FOR Idx := 0 TO 10 DO
4       usAry[Idx] := Idx * 2;
5   END_FOR
6
```

Figure 29 呼出例:配列(配列の各要素に値を代入する)

6.8.5. 列挙型

列挙型は、複数の定数を 1 つの型に纏めて管理することができます。例えば、列挙型で変数を宣言した場合、その型に無い値は「誤った値」としてその変数に割り当てることはできないため、エラーの判定が容易になります。また、型に含めた定数に名前を付けることによって、その値の意味が分かりやすくなりコードの可読性が良くなります。

なお、この定義は DUT を作成(6.7.1 項)する必要があります。

```
1  {attribute 'qualified_only'}
2  {attribute 'strict'}
3  TYPE EN_SAMPLE_ENUM0 :
4  (
5      ENUM_SP0 := 0,
6      ENUM_SP1,
7      ENUM_SP2,
8      ENUM_SP3,
9      ENUM_SP_NUMBER
10 );
11 END_TYPE
12
```

(a)列挙型の定義

```
23      enEnum0 : EN_SAMPLE_ENUM0;          //Declaration of "EUNM"
```

(b)a の型を用いての変数宣言

Figure 30 宣言例(Textual View):列挙型の定義

```
//Enum
enEnum0 := EN_SAMPLE_ENUM0.ENUM_SP2;        //Set "ENUM_SP2(=2)" to "ucVal0"
```

Figure 31 呼出例:列挙型(列挙型変数に ENUM_SP2 を代入する)

6.8.6. 型の別名定義

ユーザーは Table 16 などの既にある型を別の名前で定義することができます。分かりやすい名前をつけることによって、その型で宣言した変数がアプリケーション上でどんな用途で使われるのかが分かりやすくなります。

なお、この定義は DUT を作成(6.7.1 項)する必要があります。

呼び出し方については、元々の型と同じ方法になります。

```
1   TYPE ALS_UC : USINT; END_TYPE
1   TYPE ALS_UC_ARRAY : ARRAY[0..3] OF USINT; END_TYPE
```

(a)型の別名定義

```
18      ucTmp0   : ALS_UC;              //Declaration of "USINT"
19      ucTmpAry : ALS_UC_ARRAY;        //Declaration of "ARRAY(USINT)"
```

(b)別名定義した型を用いての変数宣言

Figure 32 宣言例(Textual View):型の別名定義

6.8.7. 構造体

構造体は型の異なる複数のデータを纏めて格納できます。構造体は関連性のあるデータを纏めることでそれらを一括管理できます。

なお、この定義は DUT を作成(6.7.1 項)する必要があります。

```
1   TYPE ST_SAMPLE_STRUCT0 :
2   STRUCT
3       ucMember0 : USINT;
4       ulMember0 : ULINT;
5       ucMemArray0 : ARRAY[0..3] OF USINT;
6   END_STRUCT
7   END_TYPE
8
```

(a)構造体の定義

```
stSmp0 : ST_SAMPLE_STRUCT0;         //Declaration of "STRUCT"
```

(b)a の型を用いての変数宣言

Figure 33 宣言例(Textual View):構造体の定義

```
9   //Struct
10  stSmp0.ucMember0 := 0;
11  stSmp0.ulMember0 := 10;
12  stSmp0.ucMemArray0[0] := 11;
13  stSmp0.ucMemArray0[1] := 22;
14
```

Figure 34 呼出例:構造体(構造体変数の各メンバに数値を代入する)

6.8.8. 構造体(拡張)

既に定義されている構造体をさらに拡張し、別の構造体名で定義することができます。定義は DUT を作成(6.7.1 項)する必要があります。構造体(拡張)は、Table 15 に従って設定し定義して下さい。

ST_SAMPLE_STRUCT1

```
1  TYPE ST_SAMPLE_STRUCT1 EXTENDS ST_SAMPLE_STRUCT0 :
2  STRUCT
3      ucMember1 : USINT;
4      ulMember1 : ULINT;
5      ucMemArray1 : ARRAY[0..3] OF USINT;
6  END_STRUCT
7  END_TYPE
8
```

(a)拡張構造体の定義

```
16      stSmp1 : ST_SAMPLE_STRUCT1;          //Declaration of "STRUCT(EXT)"
```

(b)a の型を用いての変数宣言

Figure 35 宣言例(Textual View):拡張構造体の定義(Figure 33 で定義した構造体を拡張)

```
16  //Struct(Ext)
17  stSmp1.ucMember1 := 0;
18  stSmp1.ulMember1 := 10;
19  stSmp1.ucMemArray1[0] := 11;
20  stSmp1.ucMemArray1[1] := 22;
21
22  stSmp1.ucMember0 := 0;
23  stSmp1.ulMember0 := 10;
24  stSmp1.ucMemArray0[0] := 11;
25  stSmp1.ucMemArray0[1] := 22;
```

Figure 36 呼出例:拡張構造体(構造体変数の各メンバに数値を代入する)

6.8.9. 共用体

共用体は、同じメモリ領域を複数のデータの型で共用できるものを指します。共用体内で宣言した変数をそれぞれメンバと言いますが、それらは同じメモリ領域をシェアしているため、メモリの節約ができます。また、メンバの型はそれぞれ別の型で定義できるため、用途に応じた型で使うことが可能です。

下図は、8 バイトのメモリ領域を「8 個の USINT 型データを持つ配列」と「1 個の ULINT 型データ」が共用できる共用体の定義例です。

なお、この定義は DUT を作成(6.7.1 項)し宣言する必要があります。

```
1  TYPE UN_SAMPLE0 :
2  UNION
3      ucArray : ARRAY[0..7] OF USINT;
4      ulAll   : ULINT;
5  END_UNION
6  END_TYPE
7
```

(a)共用体の定義

```
unSmp : UN_SAMPLE0;                    //Declaration of "UNION"
```

(b)a の型を用いての変数宣言

Figure 37 宣言例(Textual View):共用体の定義

```
36
37  //Union
38  unSmp.ulAll := 7;
39  ucVal0 := unSmp.ucArray[0];      //Set to "ucVal0"
40                                   //unSmp.ucArray[0]'s value is 7.
41
```

Figure 38 呼出例:共用体(共用体変数と USINT 型変数に数値 7 を代入する)

6.8.10. 定数

変数への値の代入などで定数を記述する場合、その種類によって下表の通り記述する必要が有ります。

Table 18 定数 記述方法

項目	型(一般的なもの)	記述方法
ブール	BOOL	TRUE(1)、FALSE(0)
整数	SINT INT DINT LINT USINT UINT UDINT ULINT BYTE WORD DWORD LWORD	進数表記 進数 / 記述方法 / 記述例 2 進数 / 2#(数値) / 2#11001000 8 進数 / 8#(数値) / 8#310 10 進数 / (数値) / 200 16 進数 / 16#(数値) / 16#C8
実数	REAL LREAL	実数表記 例:1.2、3.45E+6
文字列	STRING	‘(文字列)’ 例:’Tokyokeiki’
	WSTRING	“(文字列)” 例:”Tokyokeiki”

6.8.11. グローバル変数・ローカル変数

変数には、宣言する場所によってグローバル変数とローカル変数に分けられます。グローバル変数の宣言手順は、ローカル変数(6.8.2 項)と異なります。

Table 19 グローバル変数とローカル変数

名称	摘要	変数宣言場所
グローバル変数	全ての POU で使用できる変数。変数の値も共有される。	Global Variable List
ローカル変数	宣言した POU 内でのみ使用できる変数。	各 POU

6.8.11.1. グローバル変数の宣言

ここでは、グローバル変数の宣言手順を示します。

① グローバル変数リストがデバイスツリーにない場合、グローバル変数リストを作成します。デバイスウィンドウの「Application」にカーソルを合わせ右クリックして下さい。既にリストを作成している場合は手順⑤から始めて下さい。

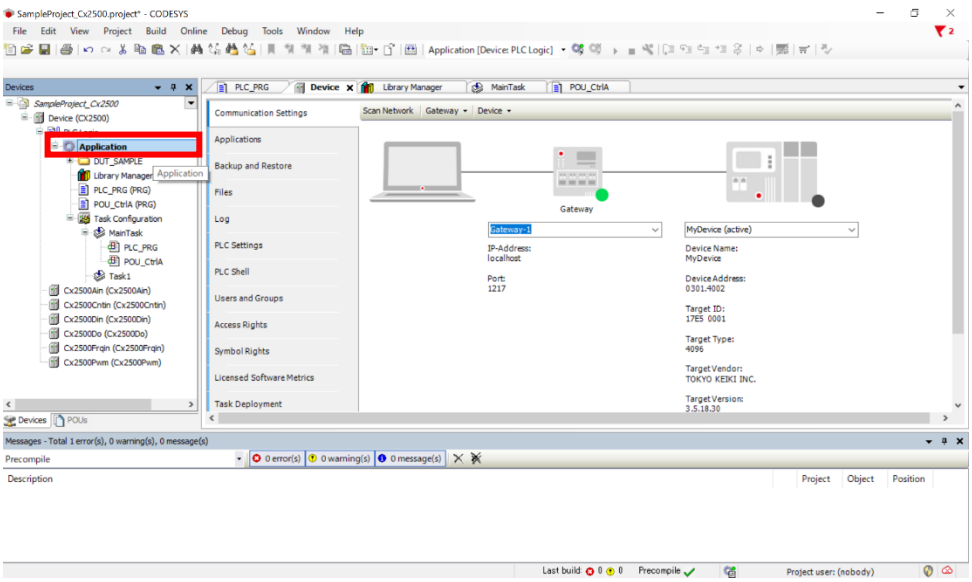


Figure 39 メイン画面 Application の選択

② 表示されたコンテキストメニューから「Add Object」→「Global Variable List…」を選択します。

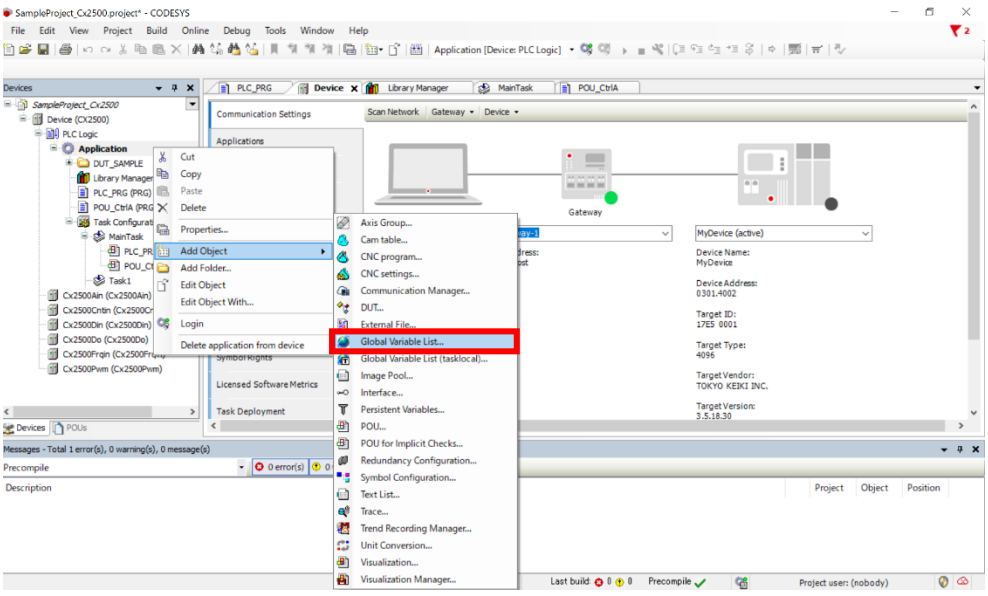


Figure 40 Application コンテキストメニュー Global Variable List の選択

③ 「Add Global Variable List」ウィンドウが表示されるので、リスト名を入力し「Add」ボタンを押して下さい。

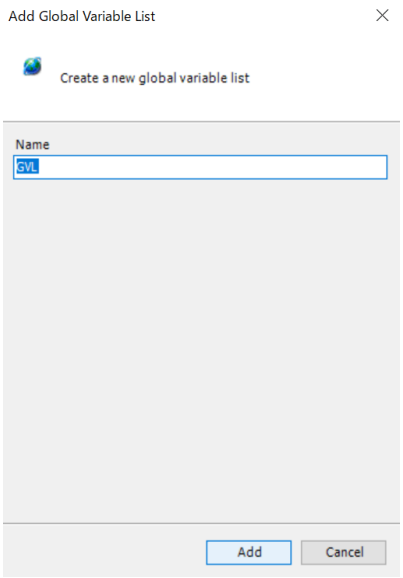


Figure 41 Add Global Variable List ウィンドウ

④ デバイスツリーにグローバル変数リストが追加され、さらにリストのエディタ画面も表示されます。これで、リストの作成は完了です。

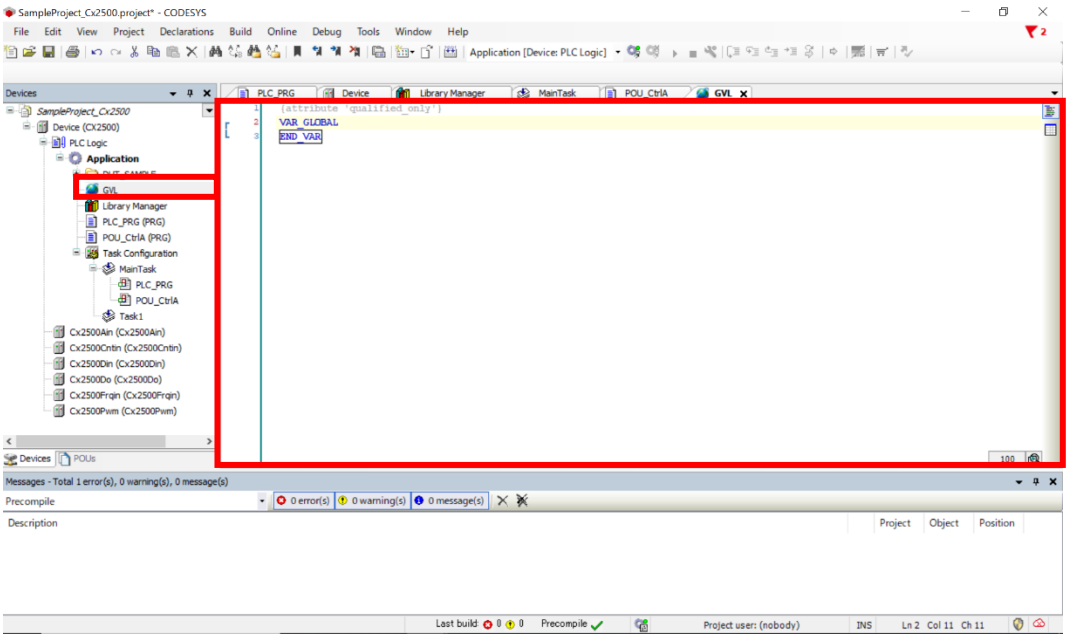


Figure 42 メイン画面 Global Variable List 追加後

⑤ 次に、グローバル変数を宣言します。グローバル変数リストのエディタ画面で所望の変数を宣言します。下図は、初期値 0 の UDINT 型グローバル変数を宣言した時の宣言例です。これで変数の宣言は完了です。

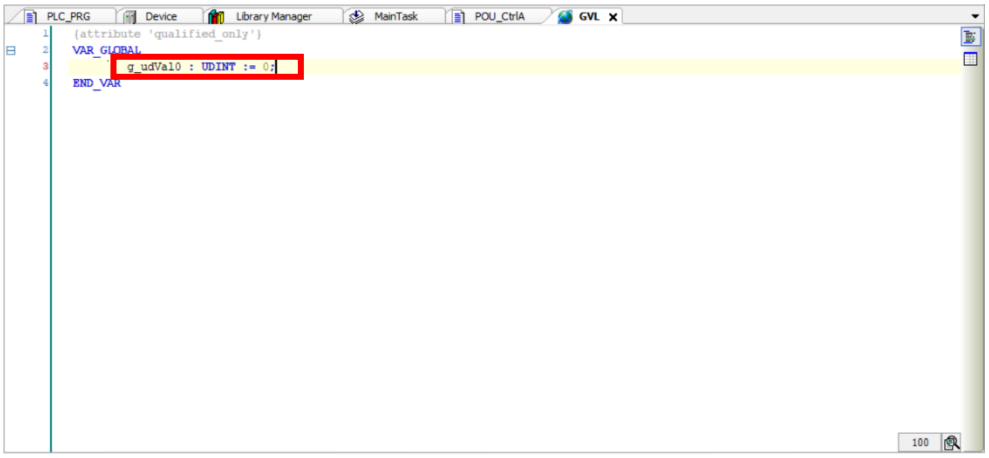


Figure 43 Global Variable List エディタ 変数の宣言

6.8.11.2. グローバル変数の呼出

グローバル変数を POU で呼び出す時の記述方法について記します。ここでは、6.8.11.1 項で宣言したグローバル変数を例に呼び出しをおこないます。

グローバル変数を呼び出す際は(グローバル変数リスト名).(変数名)と記述することで、記述した POU で呼び出し(使用)することができます。

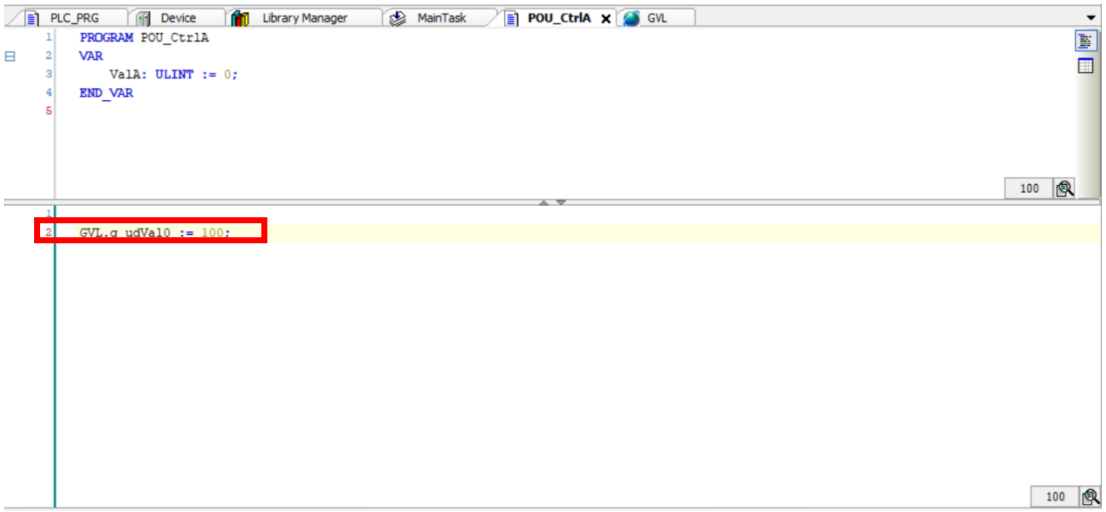


Figure 44 POU エディタ グローバル変数の宣言

6.8.12. 保持変数・持続変数

変数の中には、CX2500 の電源を落とした後でも値を保存し保持できる変数があります。保持変数と持続変数です。なお、電源を落とす際に変数の値を保存するには、必ず Figure 45 の手順で行う必要が有ります。バッテリの電源をいきなり落とす(停電等)と変数値が保存できないことに留意して下さい。

Table 20 保持変数と持続変数

名称	摘要
保持(RETAIN)変数	下記以外では変数の値を保持する。 ・アプリケーションの書き換え ・Reset Cold(コールドリセット)※9 ・Reset Origin(PLC 初期化)※9
持続(PERSISTENT)変数	Reset Origin※9 以外では変数の値を保持する。

※9 これらのリセットについては 9.6 節を参照。

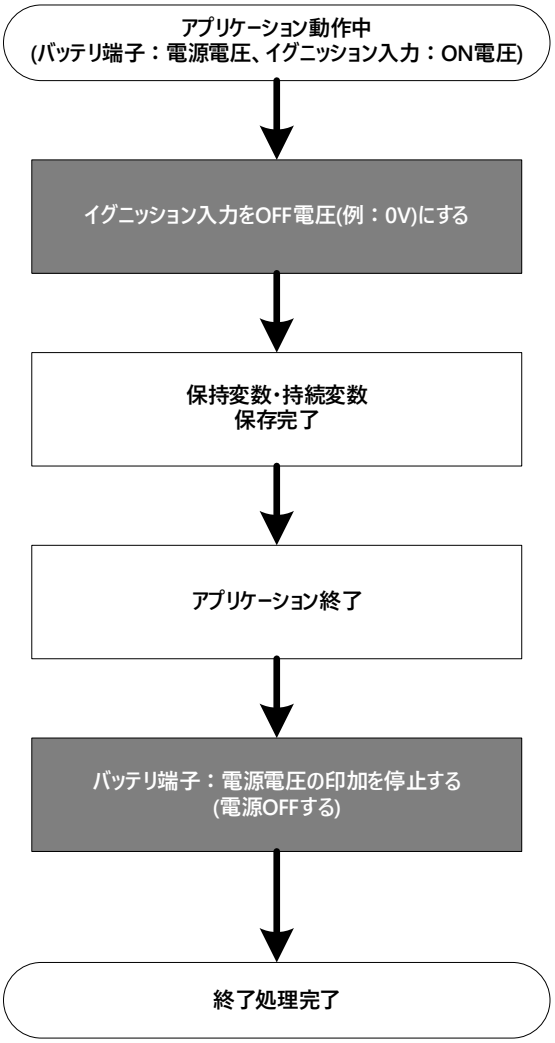


Figure 45 保持変数・持続変数 変数値保存フロー

色付き部がユーザーによって机上若しくは実際の現場で行う必要があるもの

6.8.12.1. 保持変数の宣言手順

保持変数の宣言手順を記します。ここではローカル変数で宣言していますが、グローバル変数でも宣言可能です。

① POU の変数宣言部にて所望の変数を宣言します。Add Variable(6.8.2.1 項参照)で宣言する場合、「Add variable '??' to '(POU の名前)'」ウィンドウで変数の情報を入力する際に下図のように「RETAIN」を有効化(チェックを入れる)必要が有ります。

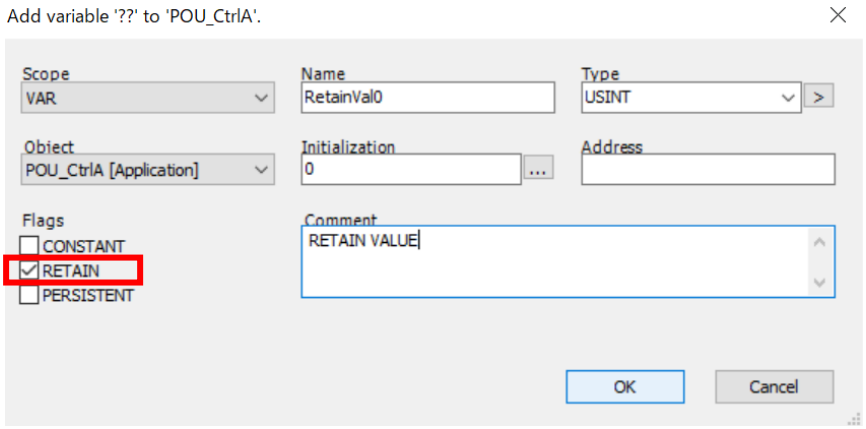


Figure 46 Add Variable 保持変数の宣言

② 変数の情報を入力した後、「OK」ボタンを押すと変数の宣言は完了です。保持変数の場合、変数のスコープは「VAR」ではなく「VAR RETAIN」になります。

```
6   VAR RETAIN
7       RetainVal0 : USINT := 0;    //RETAIN VARIABLE
8   END_VAR
9
```

Figure 47 POU エディタ 保持変数宣言後

6.8.12.2. 持続変数の宣言手順

持続変数は必ずグローバル変数で宣言する必要があります。手順は以下の通りです。

① 「Persistent Variables」がデバイスツリーにない場合、「Persistent Variables」を作成します。デバイスウィンドウの「Application」にカーソルを合わせ右クリックして下さい。既にリストを作成している場合は手順⑤から始めて下さい。

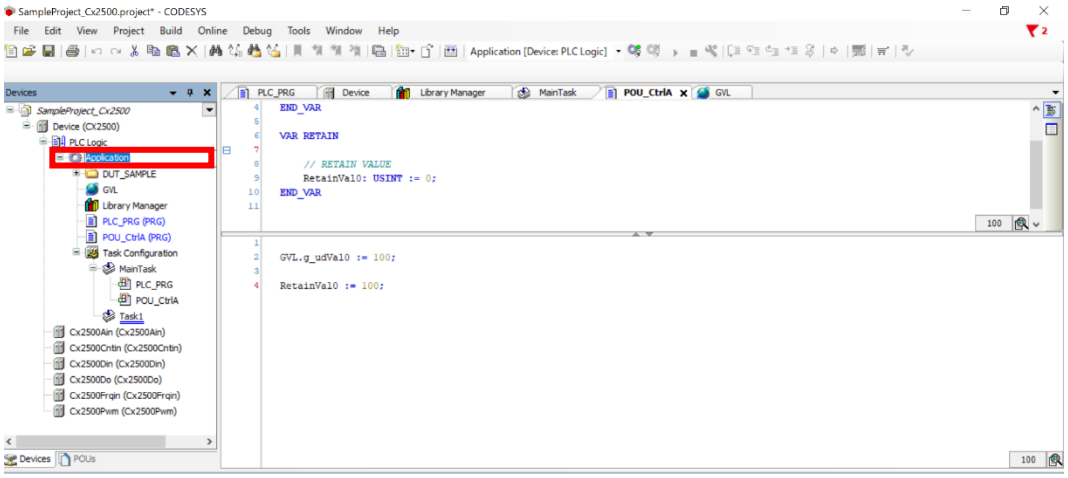


Figure 48 メイン画面 Application の選択

② 表示されるコンテキストメニューから「Add Object」→「Persistent Variables…」を選択して下さい

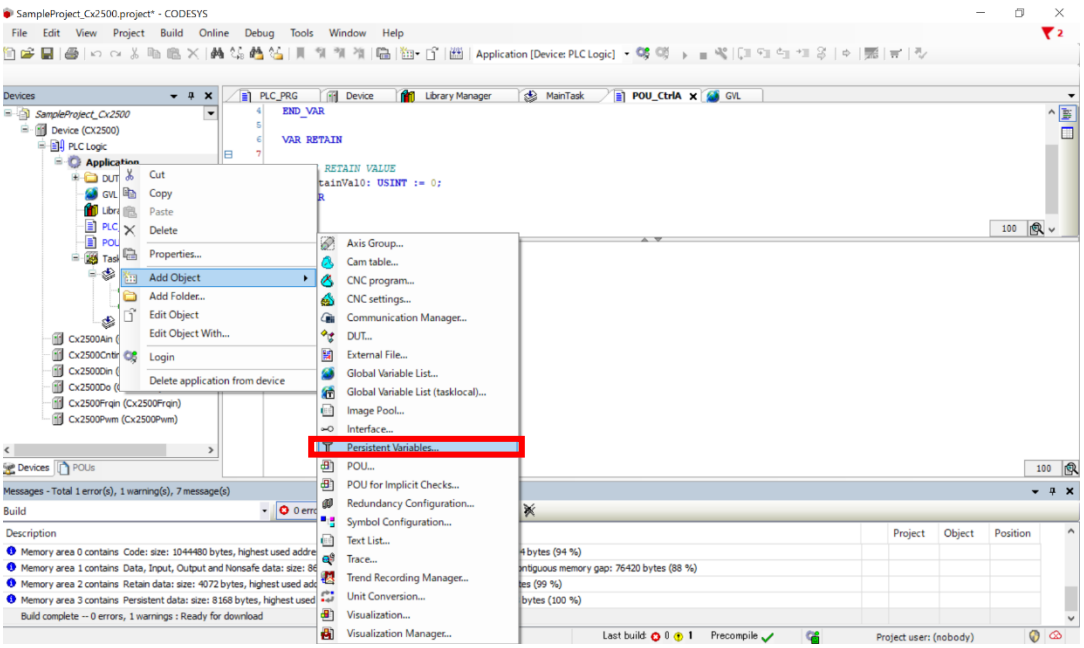


Figure 49 Application コンテキストメニュー Persistent Variables の選択

③ 「Add Persistent Variables」ウィンドウが表示されるので、名称を入力し「Add」ボタンを押して下さい。

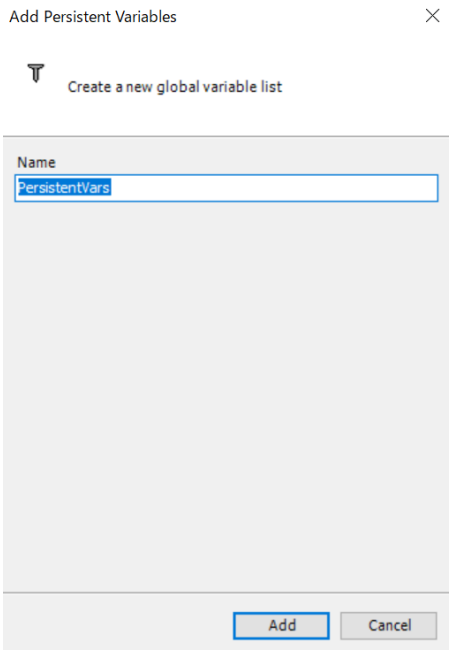


Figure 50 Add Persistent Variables ウィンドウ

④ デバイスツリーに「Persistent Variables」が追加され、さらに「Persistent Variables」のエディタ画面も表示されます。これで、「Persistent Variables」の作成は完了です。

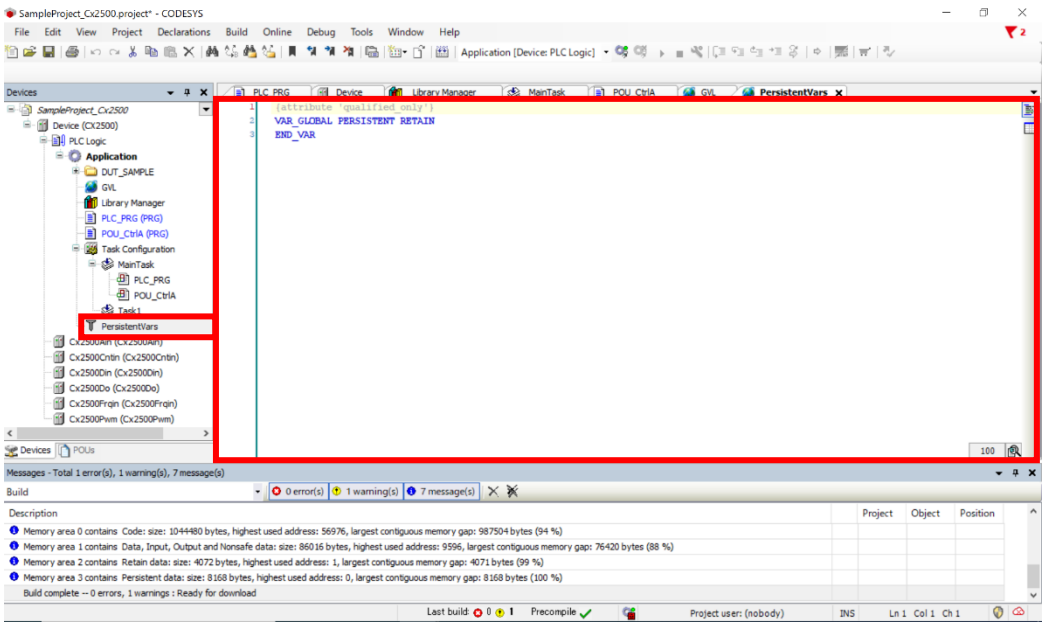


Figure 51 メイン画面 Persistent Variables 追加後

⑤ 持続変数を宣言します。ここでは「Add Variable(6.8.2.1 項)」で宣言する場合の手順を記します。「Persistent Variables」のエディタ画面にて右クリックをして下さい。

⑥ 表示されるコンテキストメニューから「Refactoring」→「Add Variables」を選択して下さい。

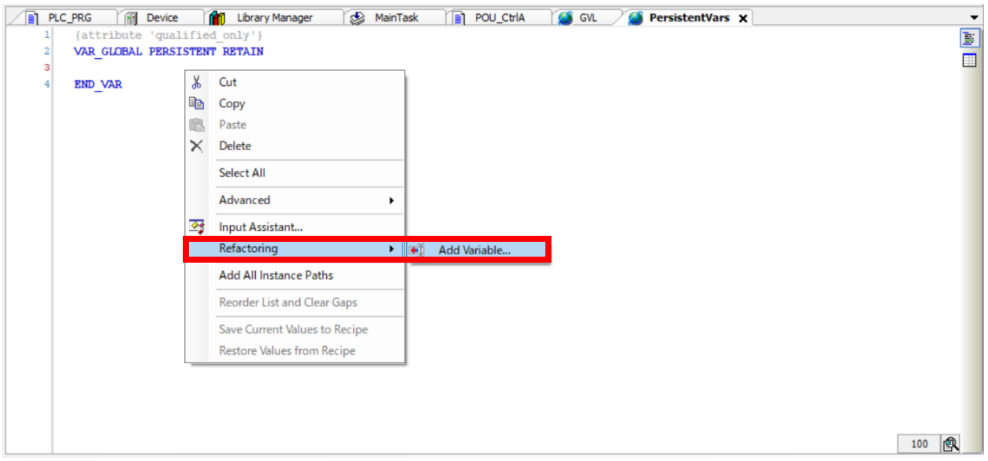


Figure 52 Persistent Variables エディタ コンテキストメニュー

⑦ 「Add variable '??' to '(PersistentVariables 名称)'」ウィンドウにて変数の情報を入力します。この時、「RETAIN」と「PERSISTENT」の 2 つを有効化(チェックを入れる)する必要があります。

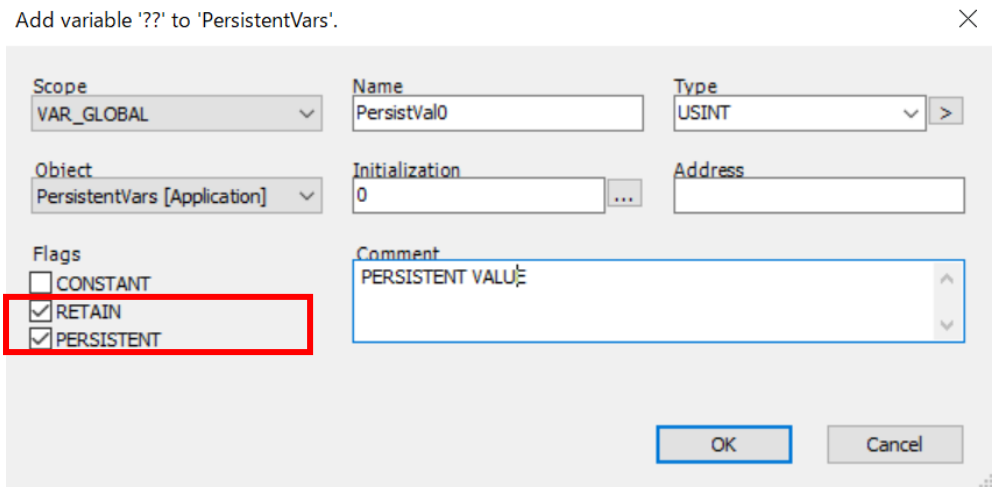


Figure 53 Add Variable 持続変数の宣言

⑧ 情報を入力した後、「OK」ボタンを押すと変数の宣言は完了です。持続変数の場合、変数のスコープは「VAR」ではなく、「VAR GLOBAL PERSISTENT RETAIN」になります。これで、変数の宣言は完了です。

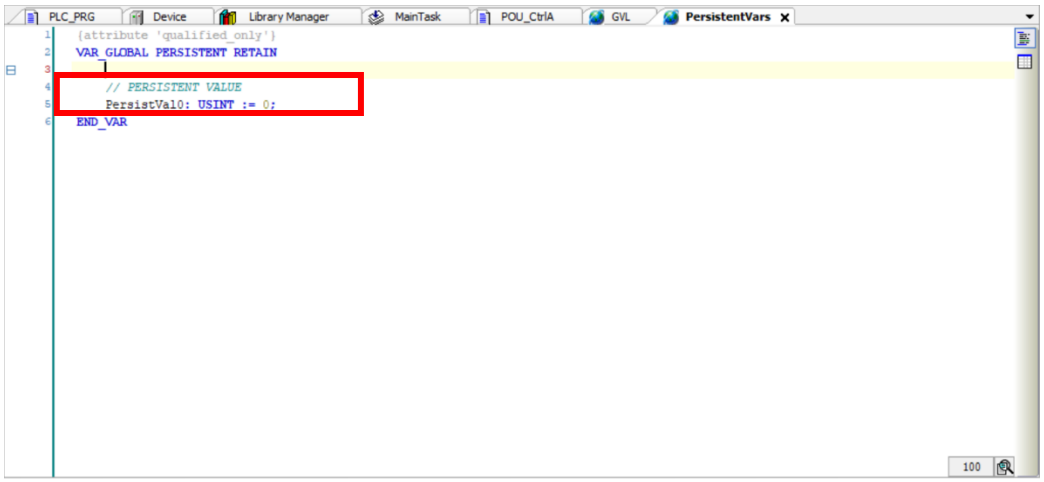


Figure 54 Persistent Variables List 持続変数宣言後

6.8.12.3. 保持変数・持続変数の呼出

ここでは、保持変数・持続変数を POU で呼び出す際のエディタでの記述方法について記します。各変数は下図の通り記述することによって POU で使用可能になります。

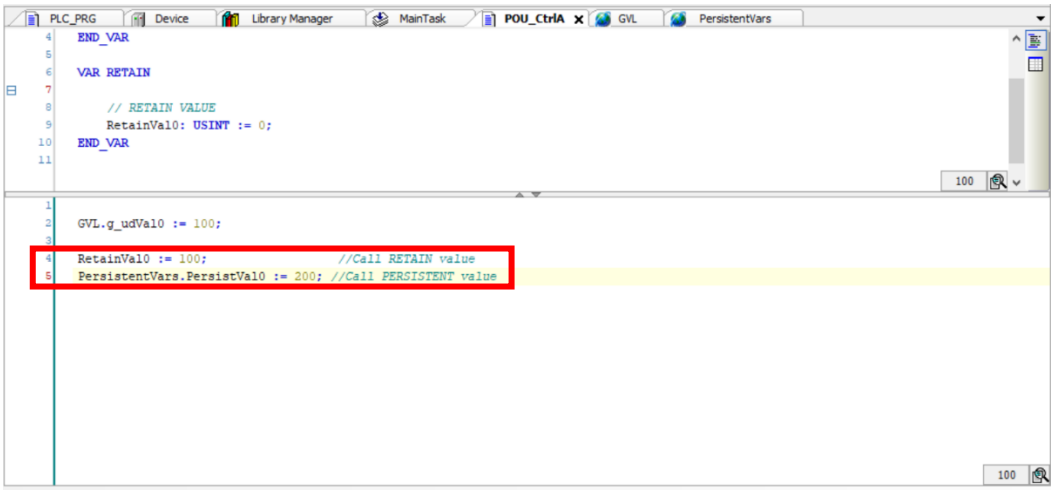


Figure 55 POU エディタ 保持・持続変数の呼出

6.9. ライブラリ

ライブラリは、POU や DUT などのアイテムを纏めた 1 つのファイルのことを指します。これは、これまで作成したような IEC アプリケーション用のプロジェクトとは別のファイルで保存します。これにより、ユーザーはいくつものアプリケーションプロジェクトで利用・流用することができ、工数の削減につなげることができます。

6.9.1. ライブラリマネージャー

アプリケーションプロジェクトには Library Manager が紐づけられています。Library Manager では、そのプロジェクトで使用できるライブラリの一覧及び情報の確認とライブラリの追加・削除などが行えます。

ライブラリマネージャーを開くには、デバイスウィンドウの「Library Manager」をダブルクリックして下さい。Library Manager 画面にてライブラリファイルを左クリックすると画面下部にそのライブラリに含まれるアイテムの情報が表示されます。

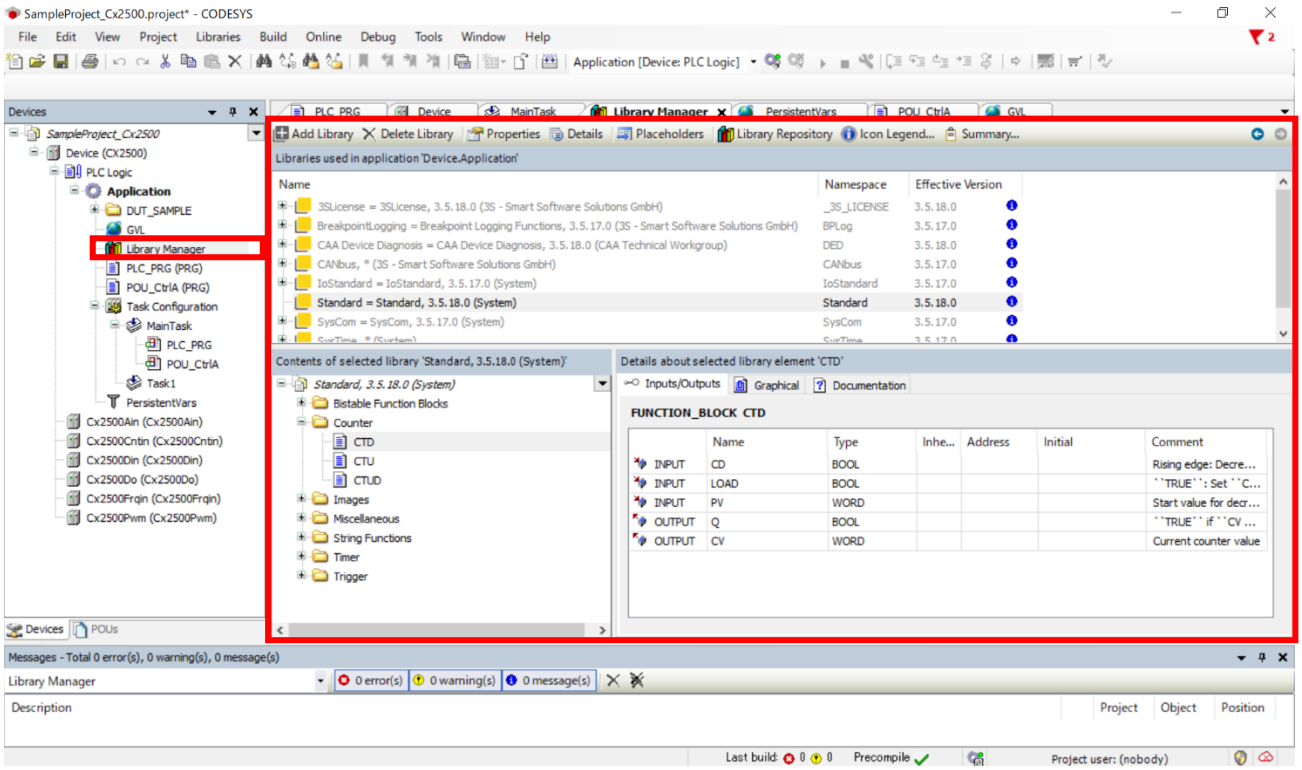


Figure 56 メイン画面 Library Manager

ライブラリの登録・削除は、それぞれ「Add library」・「Delete Library」ボタンを押すことでできます。なお、CODESYS-IDE にインストールされていないライブラリは登録ができません。インストール方法は 6.9.5.2 項を参照して下さい。

6.9.2. 共通ライブラリ

共通ライブラリは、CODESYS にデフォルトでインストールされており、プロジェクトに含まれているライブラリ群を指します。ユーザーが主に使用するライブラリは CANbus・Standard・SysCom・SysTime です。なお、ライブラリの内、RTC のみ当社独自のユーザー定義ライブラリ(Cx2500RtcLibrary)になります。

Table 21 共通ライブラリー覧

名称	摘要
3SLicense	システムライブラリ
BreakpointLogging	ブレークポイント用ライブラリ
CAA Device Diagnosis	デバイス診断用ライブラリ
CANbus	CAN 用ライブラリ(7.12 節参照)
IoStandard	システムライブラリ
Standard	IEC 標準コマンド用ライブラリ
SysCom	RS232C 用ライブラリ(7.11 節参照)
SysTime	タイマカウンタ・RTC 用ライブラリ(7.13、7.14 節参照)

6.9.3. 当社独自ライブラリ

CX2500 の機能の内、RTC をユーザーアプリケーションで利用する際は当社独自ライブラリ(CX2500 Rtc Library)をプロジェクトに紐づける必要があります。利用する際は、6.9.5.2 項を参考にして CX2500 Rtc Library のインストールおよび紐づけを行ってください。

6.9.4. ライブラリアイテムの使用方法

ここでは、Library Manager に紐づけられているライブラリファイル内のアイテムを POU に呼び出して処理に使う方法を記します。例として、SysTime ライブラリの中にある SysTimeCore ライブラリを使用し、CX2500 が起動してからの経過時間(タイマカウンタ)を取得する Function を呼び出します。

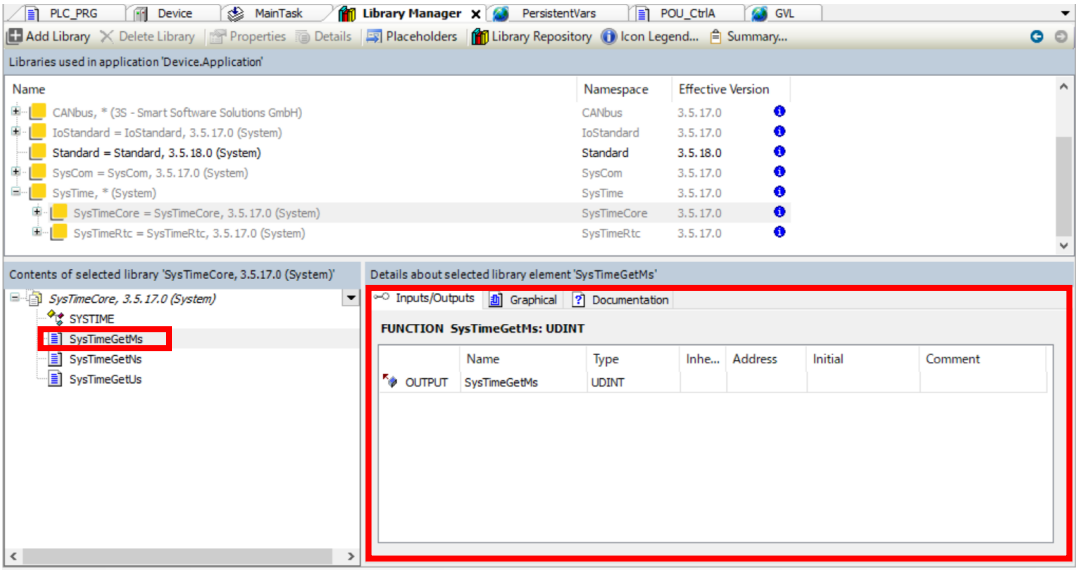


Figure 57 Library Manager SysTimeCore ライブラリ情報

POU のエディタ画面で下記のように(ライブラリ名).(関数名)で記述して下さい。DUT などの他のアイテムでも、同様に(ライブラリ名).(アイテム名)で記述する必要があります。

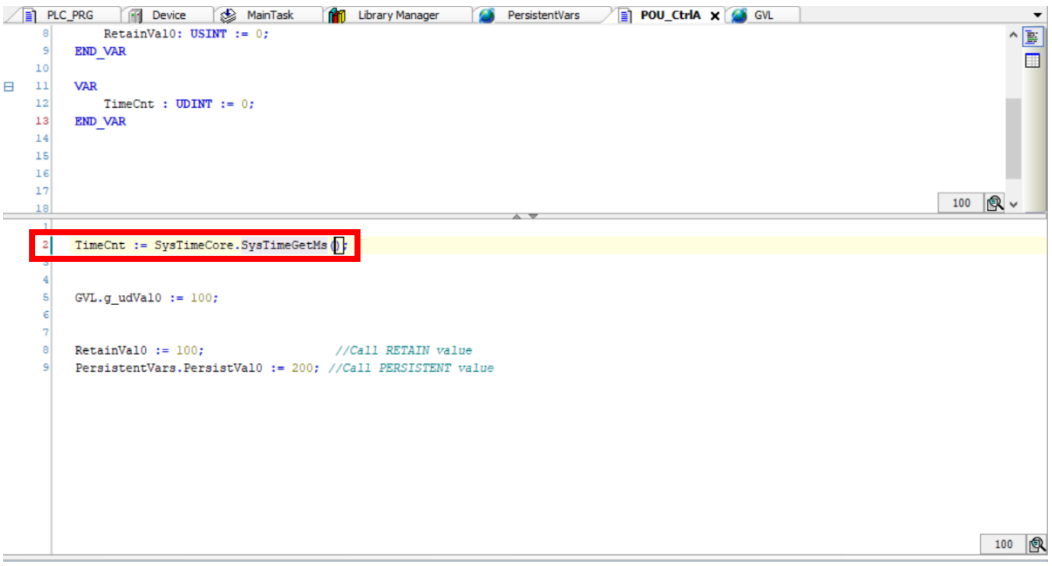


Figure 58 POU エディタ ライブラリ関数の呼出

6.9.5. ユーザー定義のライブラリ

ユーザー定義のライブラリとは、ユーザーが自ら作成したライブラリのことを指します。ライブラリファイル内のアイテムの呼び出す方法は後述します。なお、ユーザー定義ライブラリのアイテム使用方法は 6.9.4 項と同様です。

6.9.5.1. ユーザー定義ライブラリの作成

ここでは、ライブラリの作成方法を記します。

① メニューバーの「File」→「New Project..」を選択して下さい。

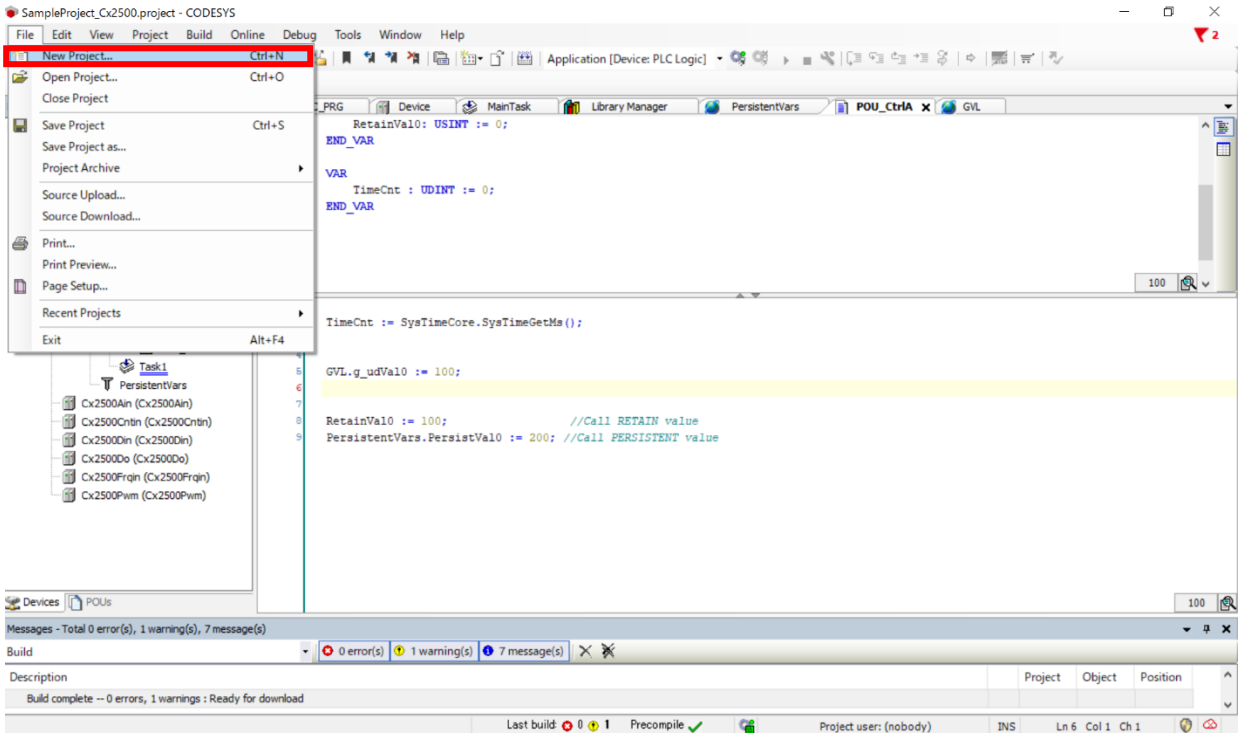


Figure 59 メイン画面 New Project の選択

② 「New Project」ウィンドウにて下記の設定をおこなった後、「OK」ボタンを押して下さい。

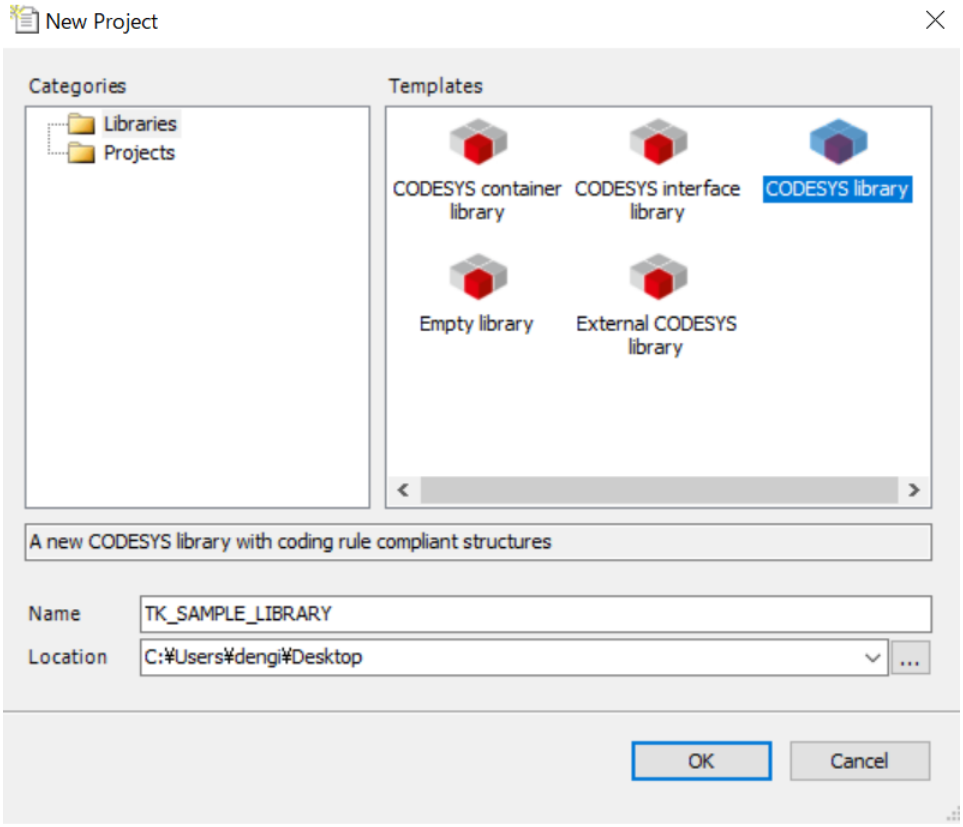


Figure 60　New Project ウィンドウ

項目	設定値
Category	「Libraries」を選択する。
Template	「CODESYS library」を選択する。
Name	ライブラリ名(ユーザー任意)を記入する。
Location	プロジェクトの保存先(ユーザー任意)を「…」ボタンを押して選択する。

③ ライブラリが作成され、ライブラリウィンドウが表示されます。編集の際は下図赤枠の「POUs」タブをクリックすると編集リストが表示されます。

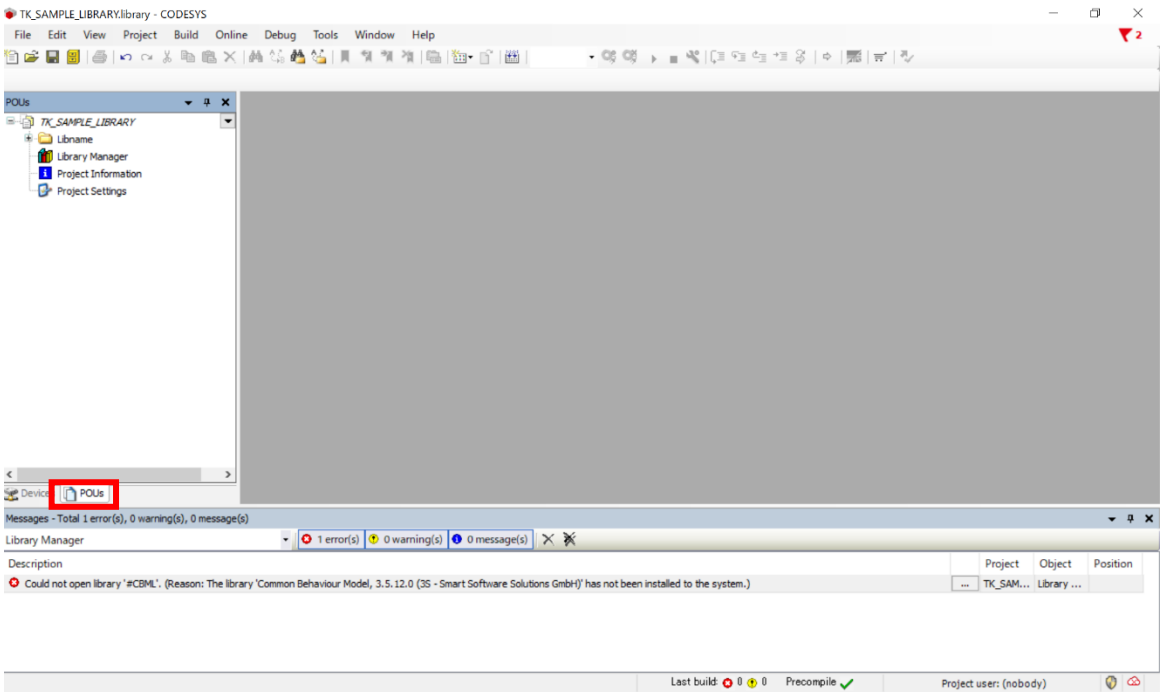


Figure 61 ライブラリ 編集画面

④ ライブラリウィンドウから、「Project Information」をダブルクリックします。すると、「Project Information」ウィンドウが表示されるので、下記の通りライブラリ情報を入力して下さい。これは、Library Manager で当該ライブラリを選択した時に表示されるライブラリ情報です。
入力後、「OK」ボタンを押して下さい。

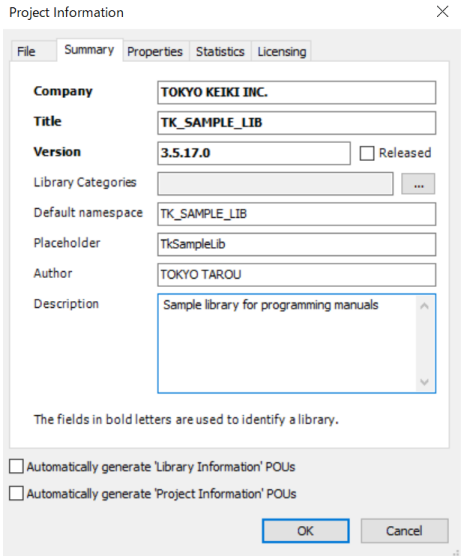


Figure 62 Project Information ウィンドウ

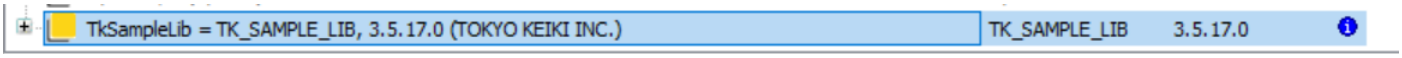


Figure 63 Figure 62 で設定した際の LibraryManager 上での表示

Table 22 Project Information ウィンドウ 設定項目（値は全て任意値）

項目	設定値
Company	ユーザーの社名等を入力する。
Title	ライブラリ名を入力する。
Version	ライブラリのバージョンを入力する。
Library Categories	設定不要。
Default namespace	ライブラリ名等を入力する。 （アイテムを呼び出すときに必要になる。）
Placeholder	ライブラリ名を入力する。
Author	ユーザーの名前等を入力する。
Description	ライブラリに関する説明を入力する。
Automatically generate ‘Library Information’ POUs	無効化(チェックを外す)
Automatically generate ‘Project Information’ POUs	無効化(チェックを外す)

⑤ ここからは、アイテム(POU や DUT 等)をライブラリ上で作成して下さい。ライブラリに入れたいアイテムを作成して下さい。作成方法はアプリケーションプロジェクトでおこなうときと同様です。ここでは例として構造体 ST_SMP_LIB と POUFB_SMP_LIB(POU、Function block)を作成しました。

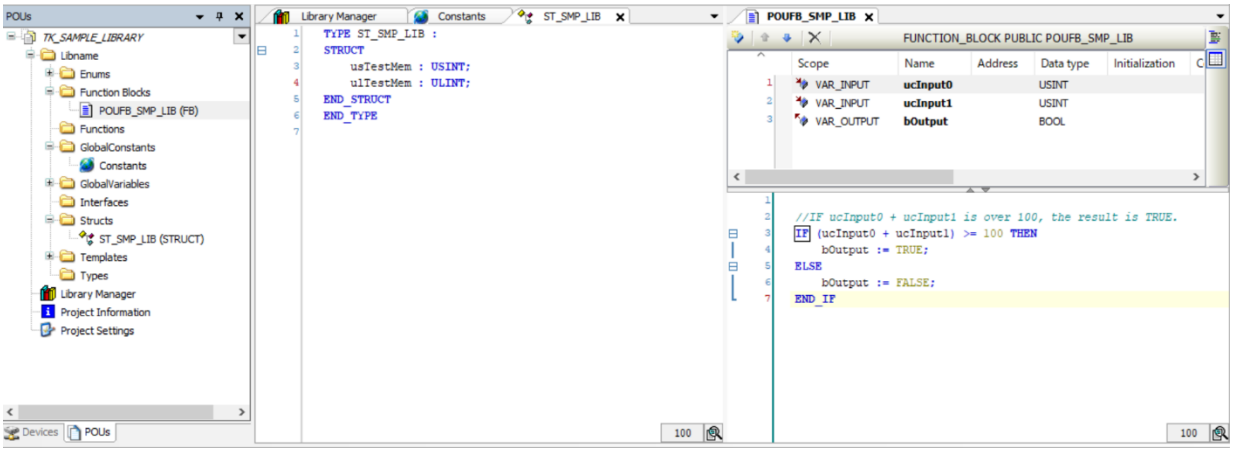


Figure 64 ライブラリ編集例 構造体と Function block の追加

⑥ アイテムを一通り作成し終えたらビルドをおこなって下さい。メニューバーの「Build」→「Check all Pool Objects」を選択して下さい。

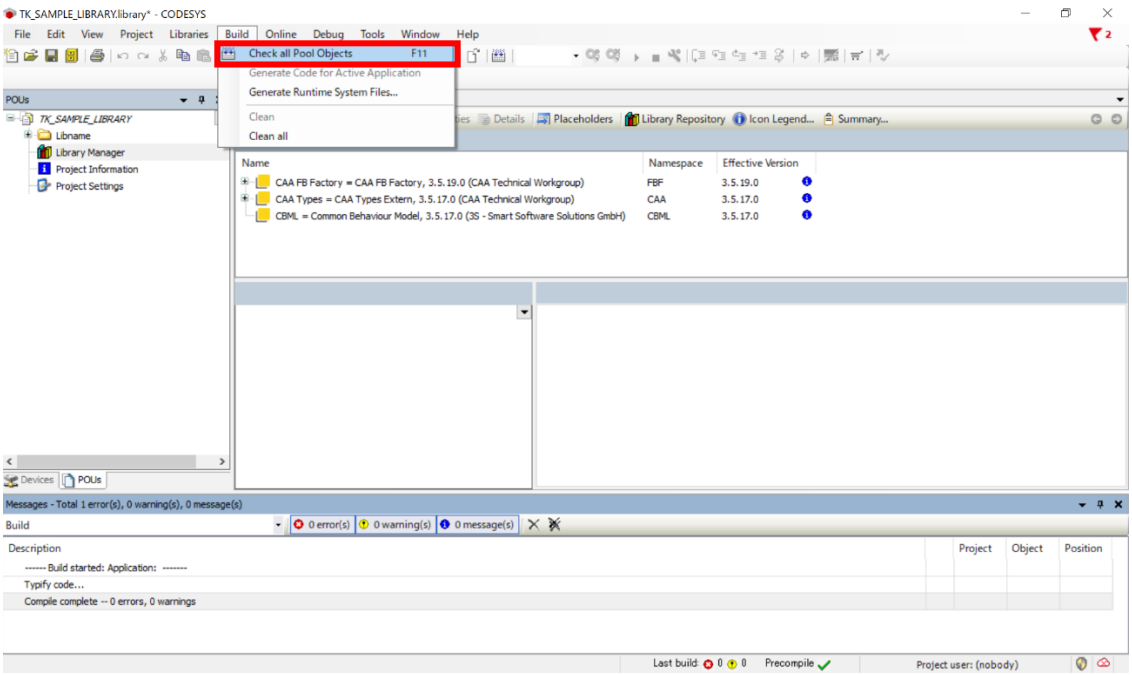


Figure 65 ライブラリのビルド

⑦ ビルド後、エラーが無いことを確認します。エラーが出た場合はその情報に従って修正し、再度ビルドして下さい。

⑧ CODESYS-IDE のライブラリリポジトリにライブラリを登録します。ライブラリリポジトリは CODESYS-IDE で使用するライブラリを登録・保存しています。  アイコンボタンを押して下さい。これでライブラリリポジトリに登録され、プロジェクトへの紐づけが可能になります。

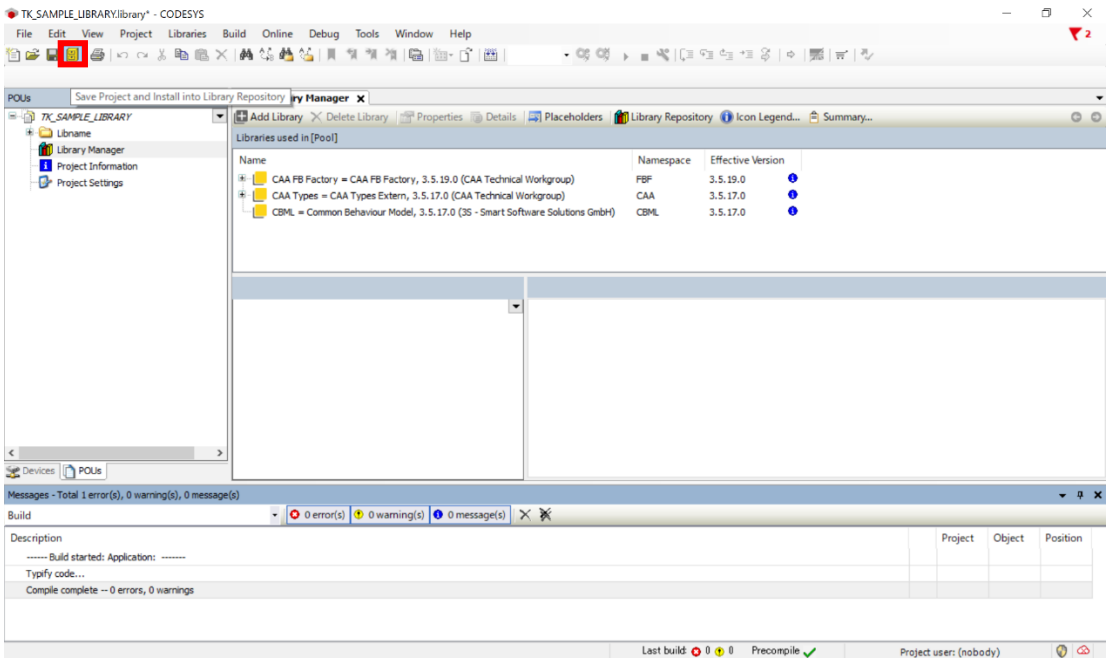


Figure 66 ライブラリリポジトリへの登録

6.9.5.2. ユーザー定義ライブラリの紐づけ

ここでは、ユーザー定義ライブラリのプロジェクトへの紐づけを記します。6.9.5.1 項と異なり、ライブラリリポジトリへの登録が済んでいない場合(違う PC の CODESYS でライブラリを作成していた場合等々)、CODESYS-IDE のライブラリリポジトリにライブラリを登録する必要が有ります。既にライブラリリポジトリへの登録が完了している場合は手順⑤から始めて下さい。

① メニューバーから「Tools」→「Library Repository…」を選択して下さい。

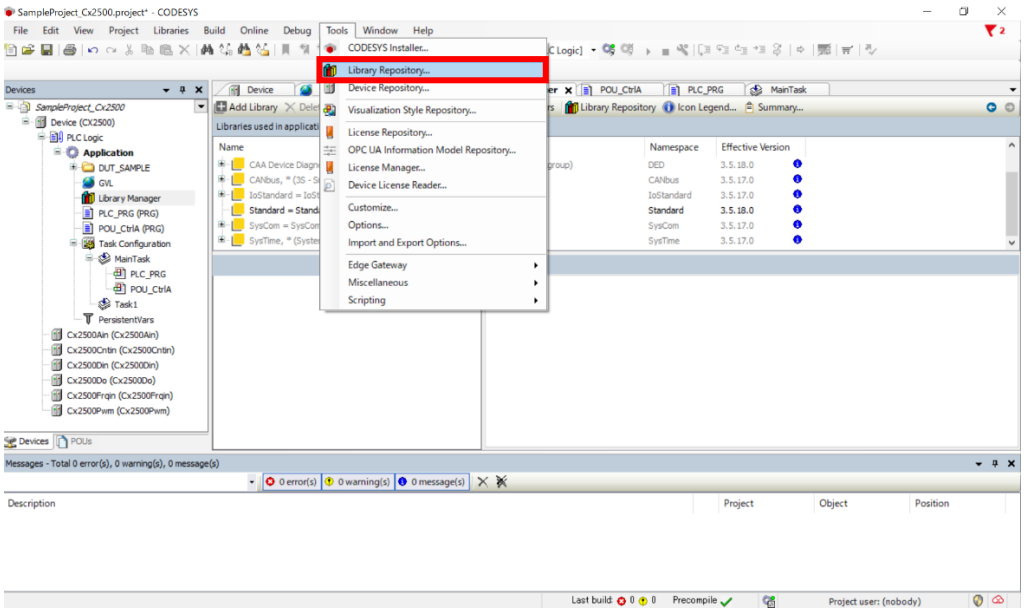


Figure 67 メイン画面 Library Repository の選択

② 「Library Repository」ウィンドウが表示されるので、「Install」ボタンを押します。

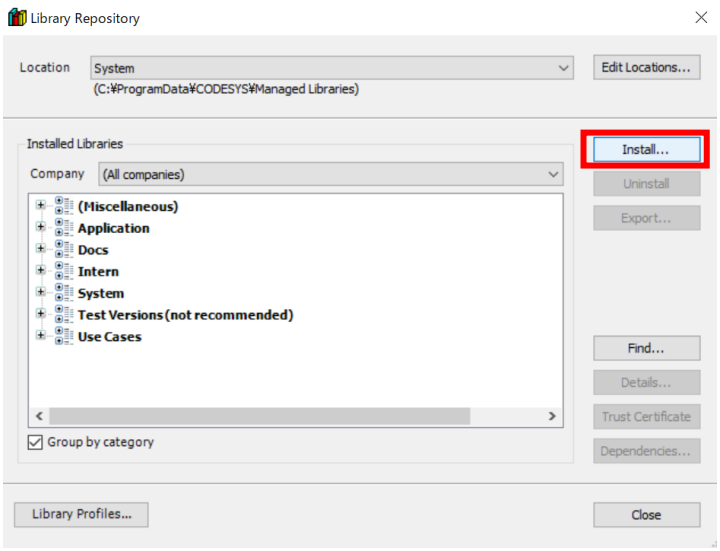


Figure 68 Library Repository ウィンドウ

③　「Select Library」ウィンドウが表示されるので、登録したいライブラリファイルの置き場所を探し、ファイルを選択した上で「開く」ボタンを押して下さい。この例では、登録したいライブラリファイルをデスクトップに置いています。

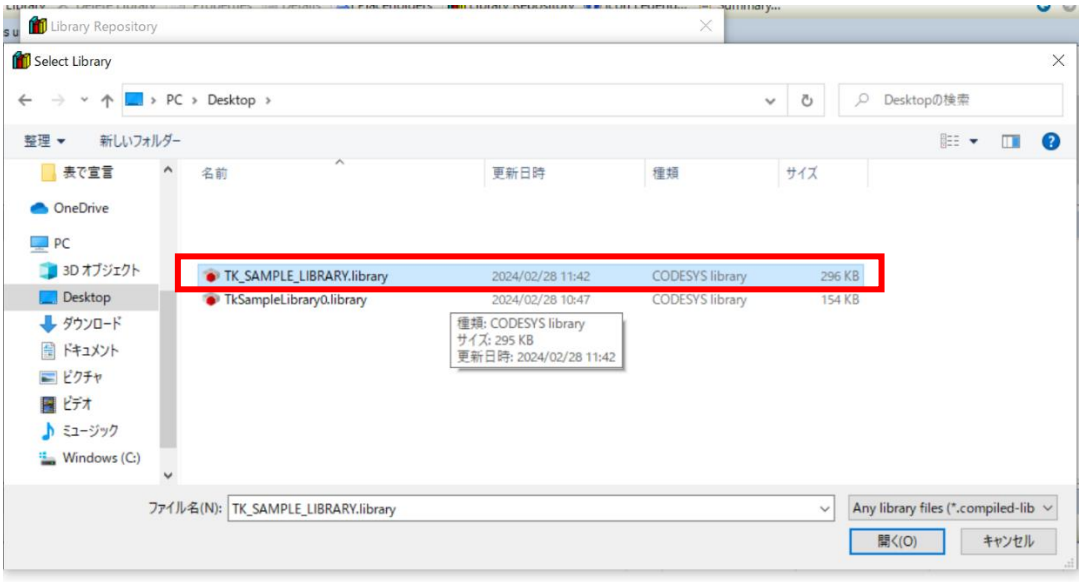


Figure 69　Select Library ウィンドウ　ライブラリの選択

④ 「Library Repository」上にライブラリが登録されます。これでライブラリリポジトリへのライブラリの登録は完了です。

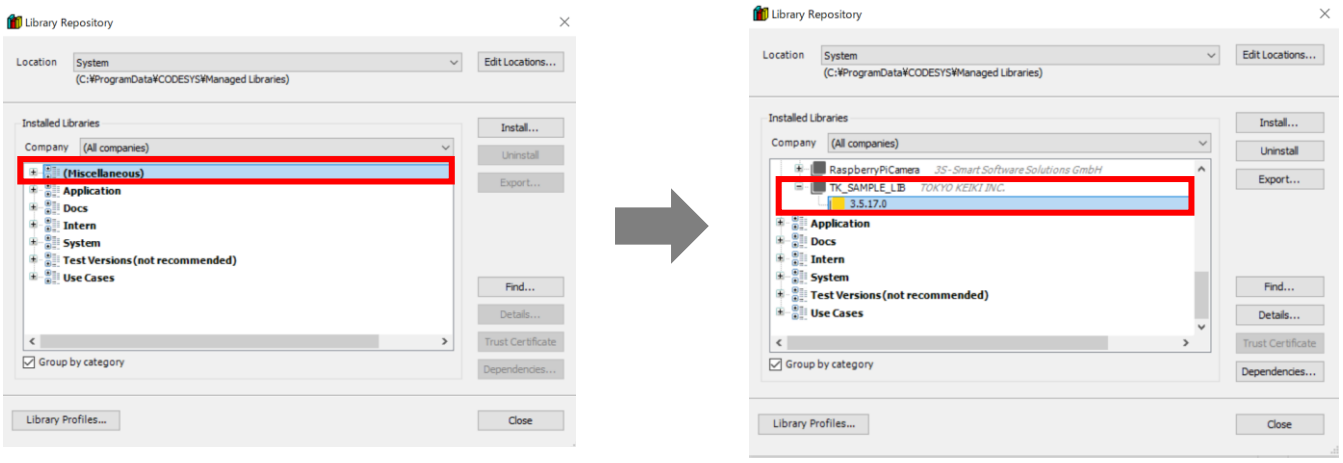


Figure 70 Library Repository ウィンドウ 追加されたライブラリの確認

⑤ ここからは、アプリケーションプロジェクトへのライブラリの紐づけをおこないます。アプリケーションプロジェクトにて、デバイスウィンドウの Library Manager をダブルクリックし Library Manager 画面を開いて下さい。

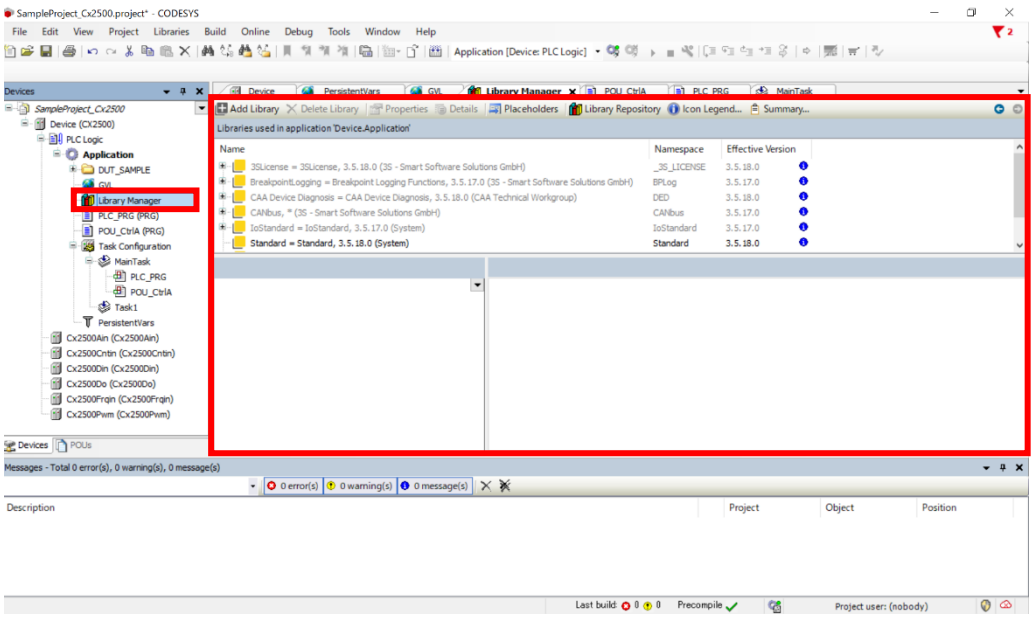


Figure 71 メイン画面 Library Manager の選択

⑥ Library Manager 画面にて、「Add Library」ボタンを押して下さい。

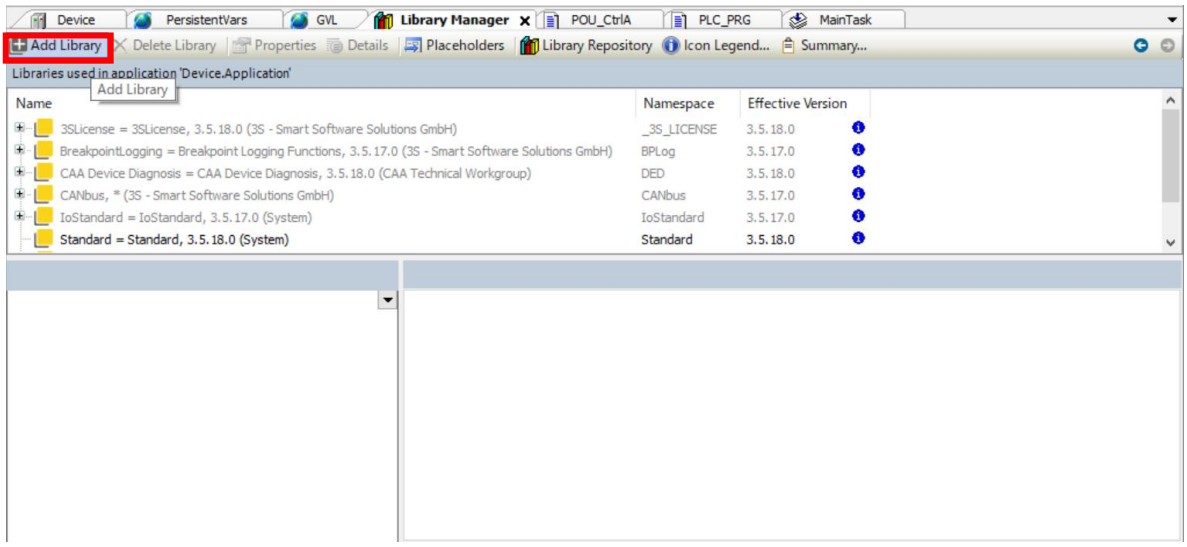


Figure 72 Library Manager Add Library の選択

⑦ 「Add Library」ウィンドウが表示されるので、下記のように紐づけたいライブラリ名を入力してライブラリを選択します。選択した状態で「OK」ボタンを押して下さい。

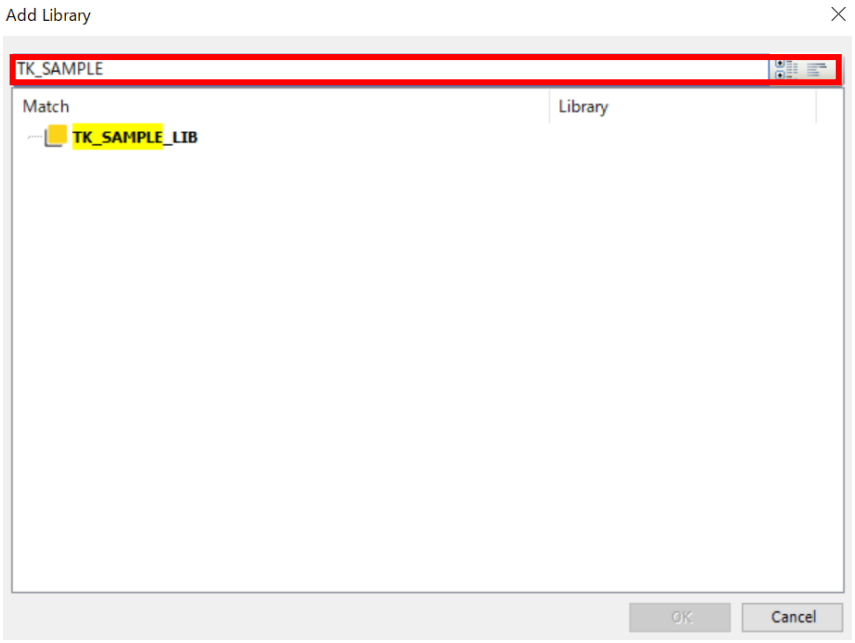


Figure 73 Add Library ウィンドウ ライブラリの検索

⑧ Library Manager 上にライブラリが表示されれば紐づけは完了です。紐づけ後、ライブラリのアイテムを使用できます。

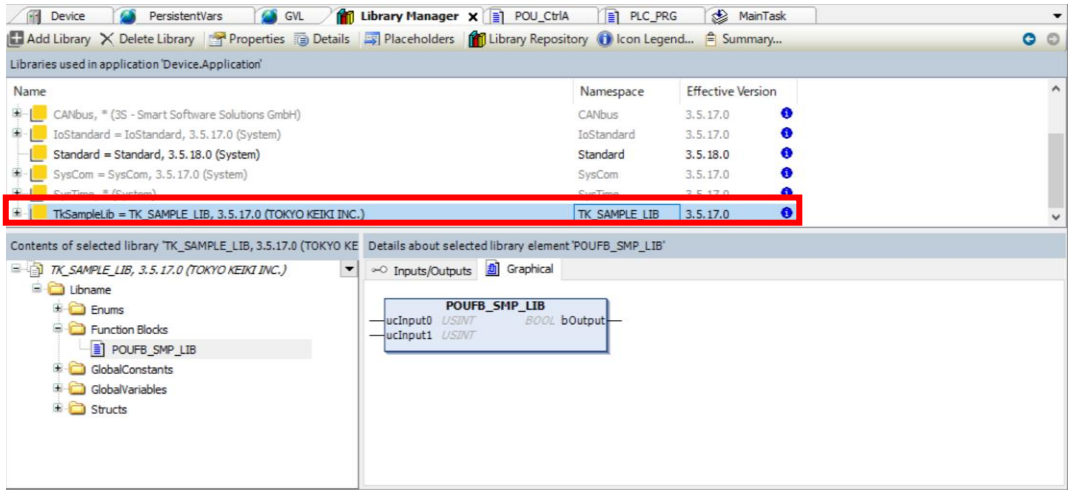


Figure 74 Library Manager ライブラリ追加後

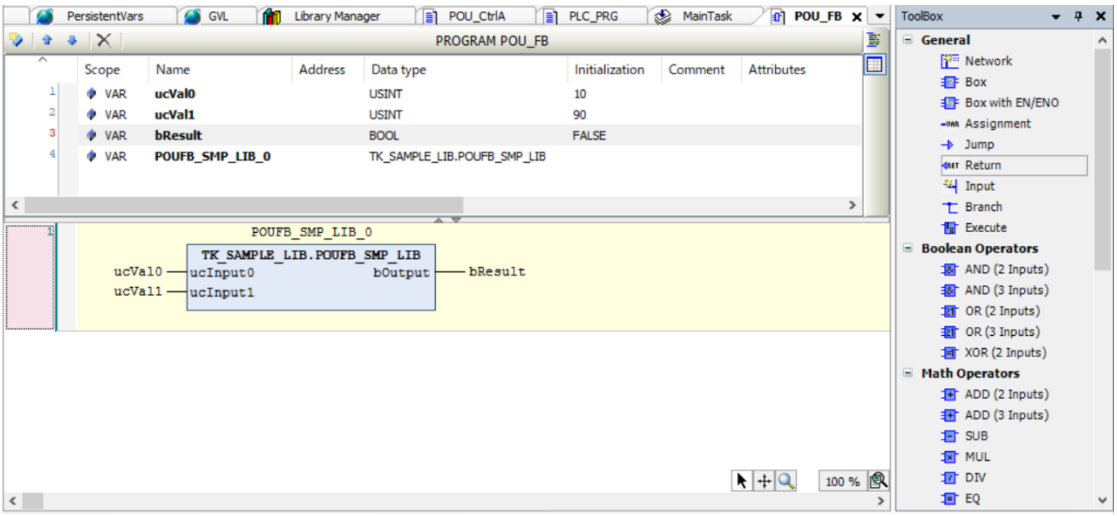


Figure 75 POU エディタ ライブラリ関数(Function block)の呼出例

6.10. ビルド

IEC アプリケーションを作成して CX2500 に書き込む前に、必ずビルド(コード生成)をおこなう必要が有ります。

ビルドは、CODESYS メイン画面の「Build」タブの「Generate Code」からできます。ビルドしてエラーが出た場合はエラー内容に沿って修正して下さい。エラー無しの場合、CX2500 にアプリケーションを書き込むことができます。

「Build」タブ・コンテキストメニューの各要素については下表を参照して下さい。

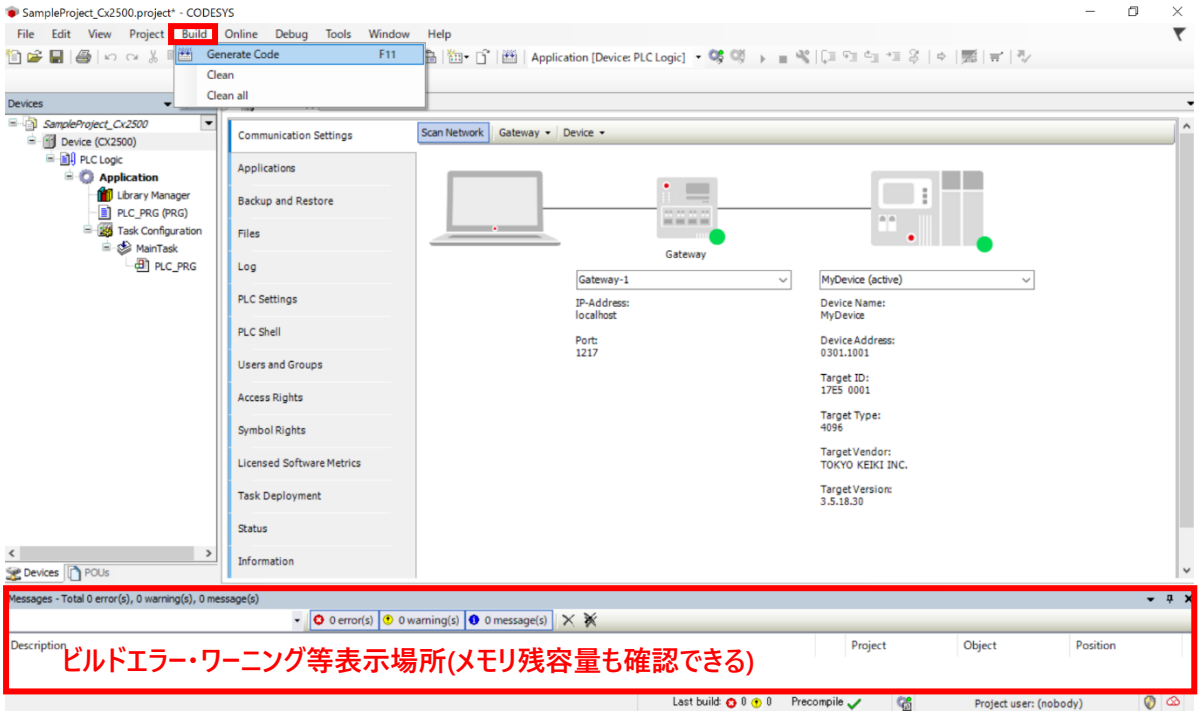


Figure 76 ビルド(コード生成)

Table 23 ビルドメニュー

名称	摘要
Generate Code	IEC アプリケーションのビルド(コード生成)をおこなう。
Clean※10	プロジェクト内のアプリケーションの内、アクティブになっているアプリケーションの前回ビルド情報をクリアする。
Clean all※10	プロジェクト内の全てのアプリケーションのビルド情報をクリアする。

※10 Clean や Clean all は、主にデバイス定義ファイルを更新した時(Setup 編参照)等に使用する。

6.11. ログイン・ログアウト

IEC アプリケーションを作成しビルドまで完了したら、CX2500 にアプリケーションを書き込むことができます。アプリケーションを書き込んでデバッグをおこなっている状態のことをログインと言います。一方、そのログイン状態を終了(CODESYS-IDE と CX2500 との接続を終了する)することをログアウトと言います。

6.11.1. ログイン手順

ここではログインの手順を示します。

① 別紙 Setup 編の「デバイスの接続」を参照し、CX2500 と CODESYS-IDE を接続した状態にして下さい。

② メニューバーの「Online」タブから「Login」を選択して下さい。

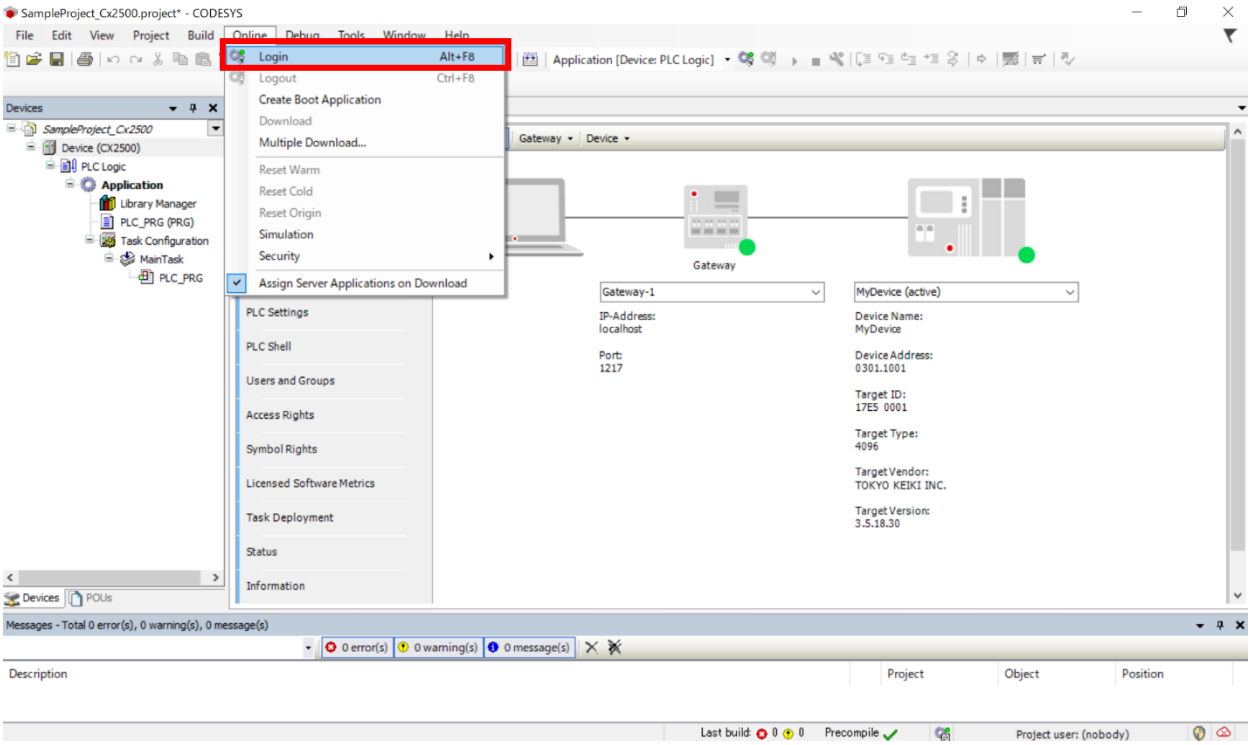


Figure 77 Login の選択

③ 下記のようなアプリケーション書き込みの確認ウィンドウが表示されますので、「Yes」ボタンを押して下さい。なお、既に書き込もうとしているプロジェクトのアプリケーションと同じアプリケーションが CX2500 に書き込まれている場合、このウィンドウは表示されずに手順⑤へ遷移します。

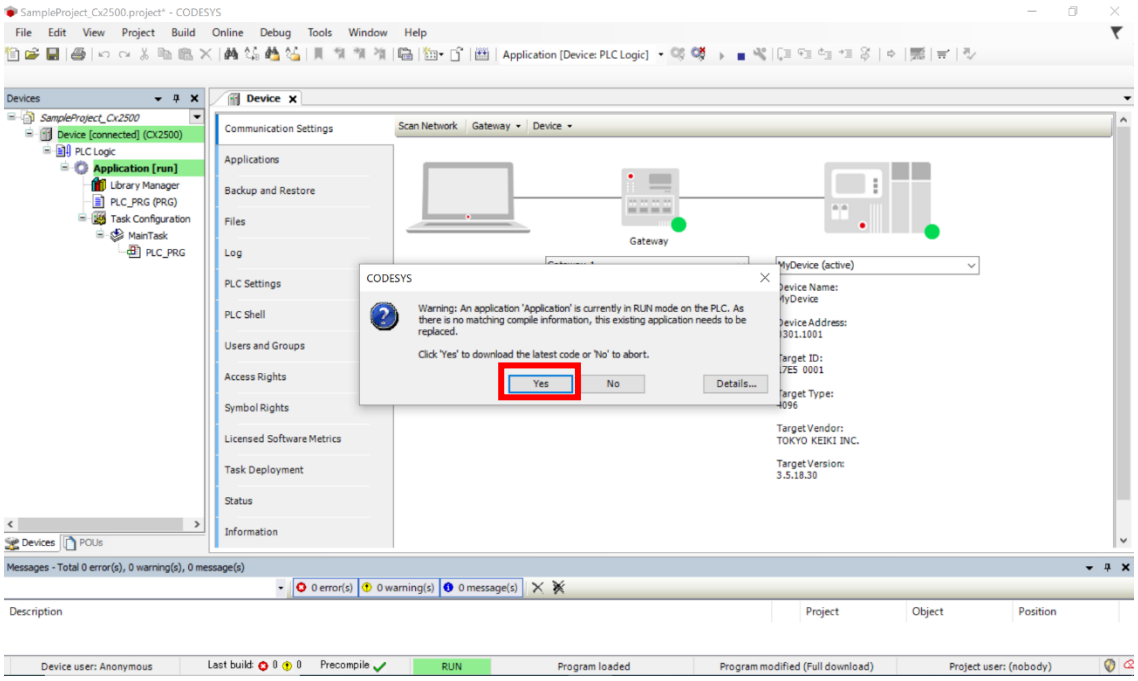


Figure 78 書き込み確認ウィンドウ

④ アプリケーション書込みが始まりますので、書き込みが終了するまで待機します。

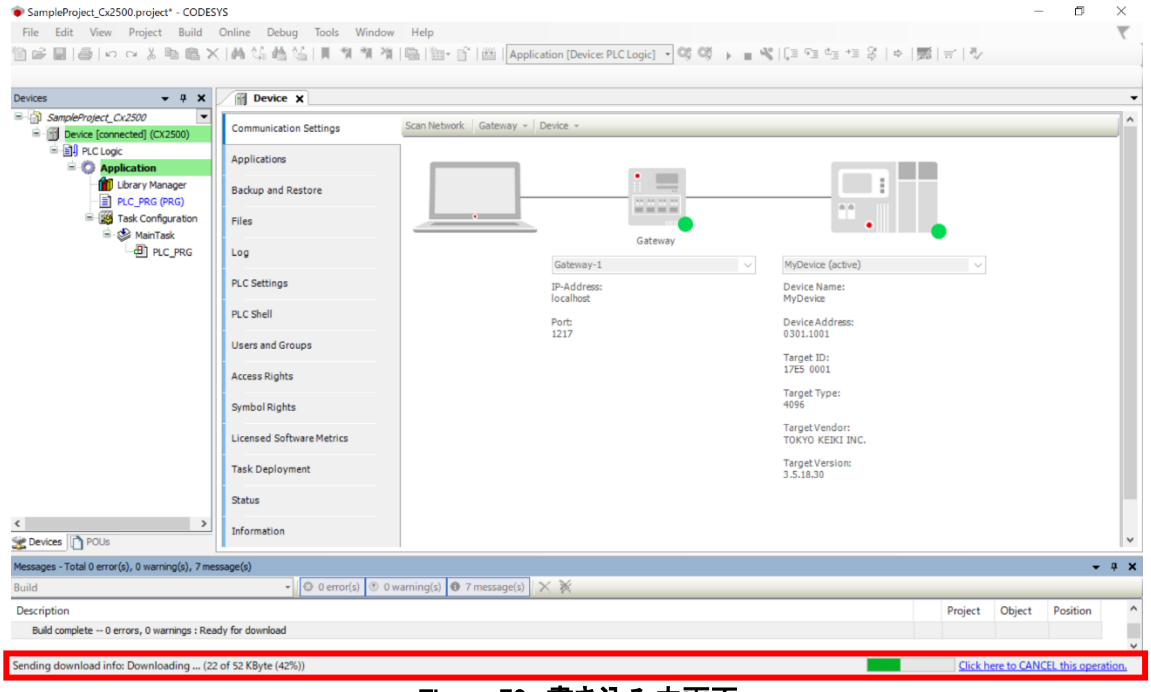


Figure 79 書き込み中画面

⑤ 以下のようなデバッグ画面に遷移するとアプリケーション書込み(ログイン)は完了です。ログイン中のデバッグについては 9 章を参照下さい。

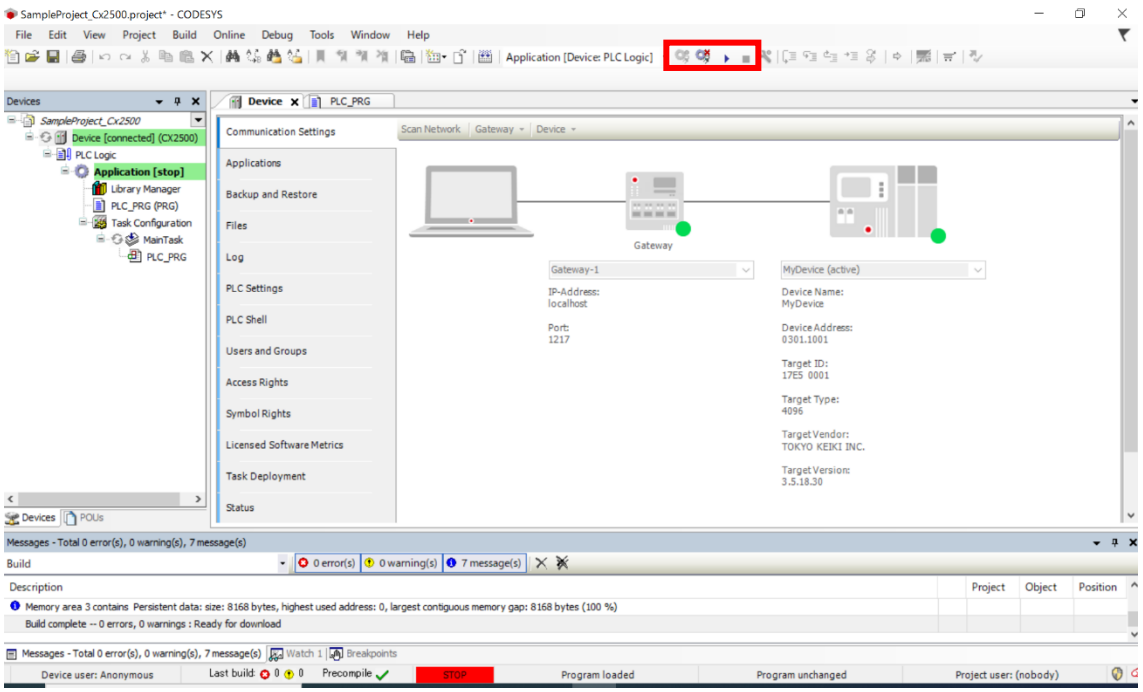


Figure 80 書き込み終了後の画面

赤枠内のデバッグ関連のアイコンボタンが有効化される

6.11.2. ログアウト手順

ログアウトは、メニューバーの「Online」タブから「Logout」を選択して下さい。若しくは、Figure 80 のようなログイン中の画面で、ツールバーのログアウトアイコン「」を押すことでもログアウトができます。なお、CX2500 内のアプリケーション運転状態はログアウト時点での状態を維持することに留意して下さい。

7. CX2500 の機能について

7.1. CX2500 機能一覧

CX2500 の機能一覧は下表の通りです。各機能の使い方は関連の節を参照して下さい。

Table 24 CX2500(CODESYS 版) 機能一覧

項目	機能名	チャネル数	参照先	備考
入力	デジタル入力	23	7.3 節	-
	イグニッション入力	1	7.3 節	-
	周波数入力	8	7.4 節	-
	2 相カウンタ入力	8	7.5 節	-
	アナログ入力	30	7.6 節	-
	内部電源電圧監視入力	-	7.7 節	-
	基板温度監視入力	3	7.8 節	-
出力	デジタル出力	16	7.9 節	-
	PWM 出力	10	7.10 節	-
通信	RS232C	2	7.11 節	
	CAN	5	7.12 節	Ch.3、4 のみ通信速度 250kbps 以下。
その他	タイマカウンタ	-	7.13 節	-
	RTC	1	7.14 節	-

7.2. 機能ドライバについて

機能ドライバとは、CX2500 に搭載されている以下の入出力機能の制御をおこなうためのモジュールです。下記機能を使用する際は、「CX2500Codesys_UserManual_ForSetup」の 5.2 節にあるようにプロジェクトのデバイスに紐づける必要があります。各機能ドライバの共通する設定については 7.2.1 項以降に纏めています。

各ドライバの設定画面については、下図赤枠のように表示されているドライバをそれぞれダブルクリックすると表示されます。

Table 25 機能ドライバと対象機能の対応表

区分	機能名	対応機能ドライバ名
入力	デジタル入力	Cx2500Din
	イグニッション入力	
	周波数入力	Cx2500Frqin
	2 相カウンタ入力	Cx2500Cntin
	アナログ入力	Cx2500Ain
	内部電源電圧監視入力	
	基板温度監視入力	
出力	デジタル出力	Cx2500Do
	PWM 出力	Cx2500Pwm

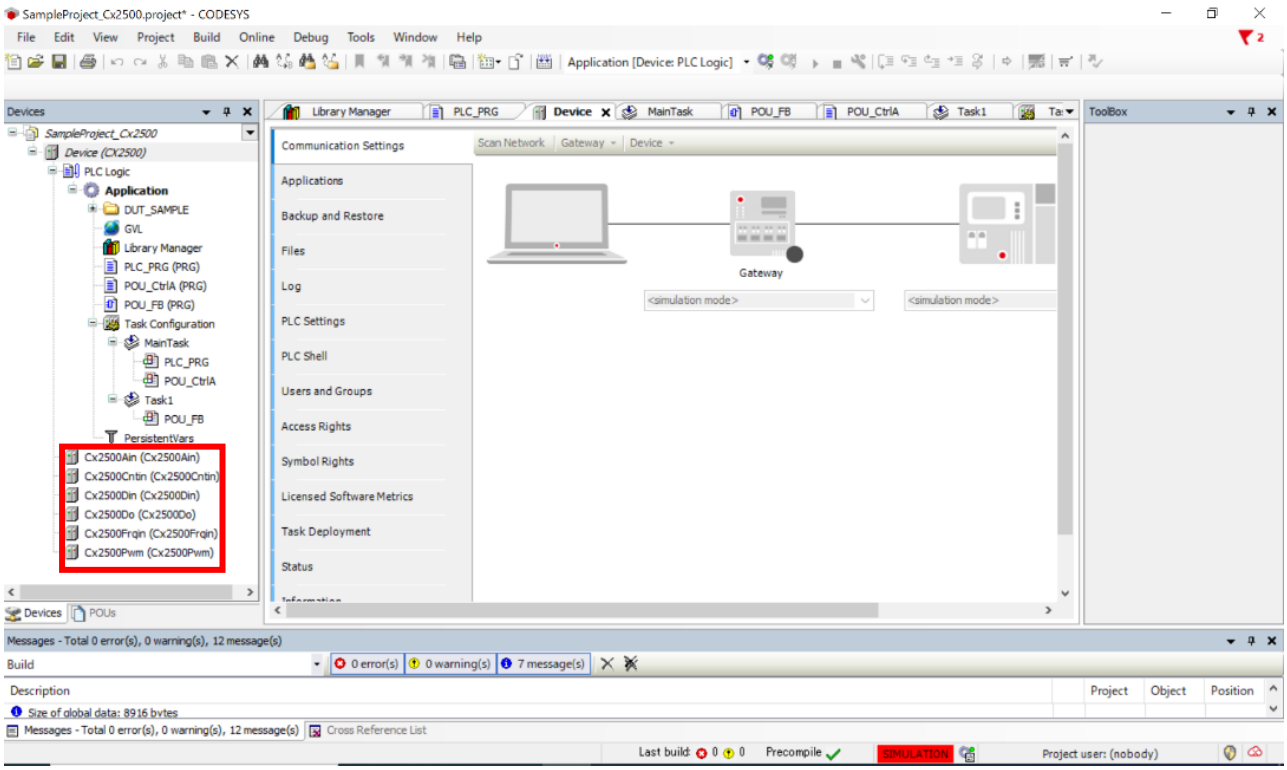


Figure 81 機能ドライバの紐づけ

7.2.1. 機能ドライバ画面

各機能ドライバは、以下の４つのタブ(機能によってはInternal Parameters タブが無い)で構成されています。これらのタブの内、Internal Parameters タブ(7.2.2 項)と Internal I/O Mapping(7.2.3 項)タブはユーザーが設定・制御に使うことになります。

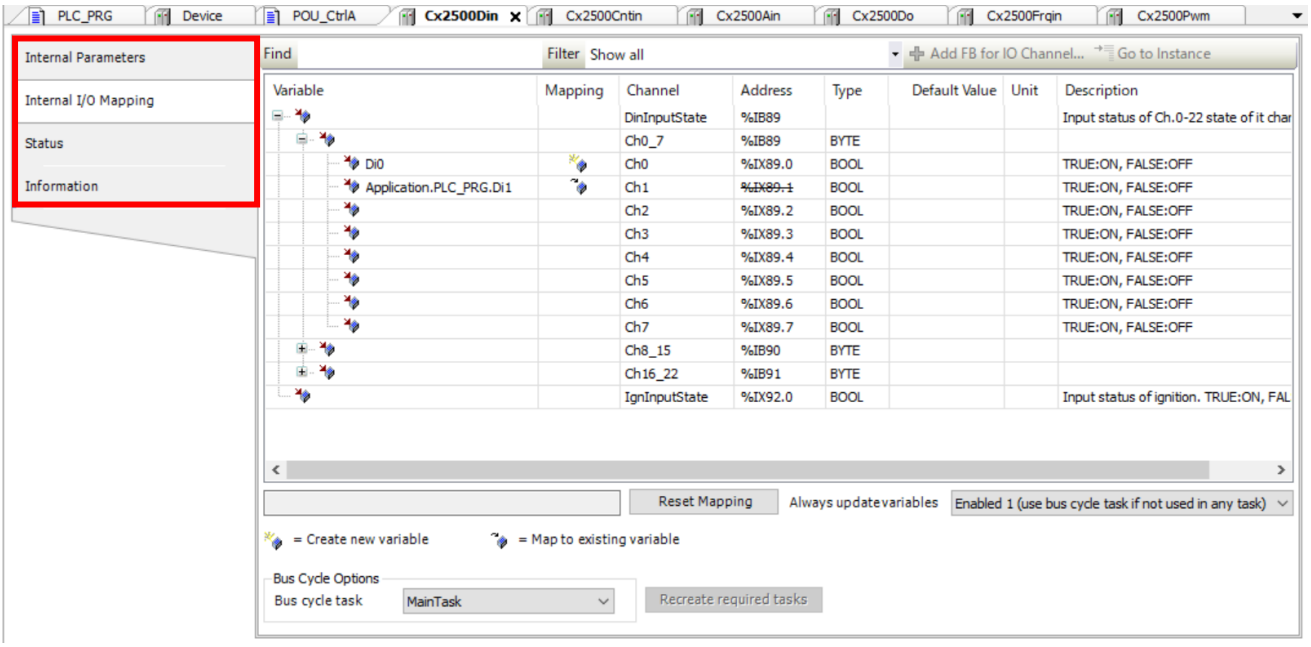


Figure 82 機能ドライバ画面

Table 26 機能ドライバ タブ一覧

タブ名称	摘要
Internal Parameters	機能の初期設定をおこなう必要がある場合に表示されるタブ。 (例)デジタル入力の入力形式選択、PWM 出力のディザ周波数 etc.
Internal I/O Mapping	機能の制御、機能の制御中の状態モニタに使う。
Status	機能ドライバの稼働状態を確認することができる。稼働状態の場合は「Running」と表示される。
Information	機能ドライバの情報(ドライバのバージョン等)を確認できる。

7.2.2. Internal Parameters タブ

Internal Parameters タブでは、機能の初期設定をおこなう必要のある要素がある場合のみ表示されます。

ユーザーはこのタブにある要素の設定を必ずおこなって下さい。設定は、ビルド前に各要素の Value 列に所望の値を入力するだけです。設定値が TRUE 若しくは FALSE を設定する場合は、要素の Value 列をクリックすることで TRUE/FALSE を切り替え・選択することができますユーザーが Value 列で値を設定しない場合、各要素の設定値は「Default Value」列の値になることに注意して下さい。

各機能の Internal Parameter タブの解説ついては、後述する各節を参照して下さい。

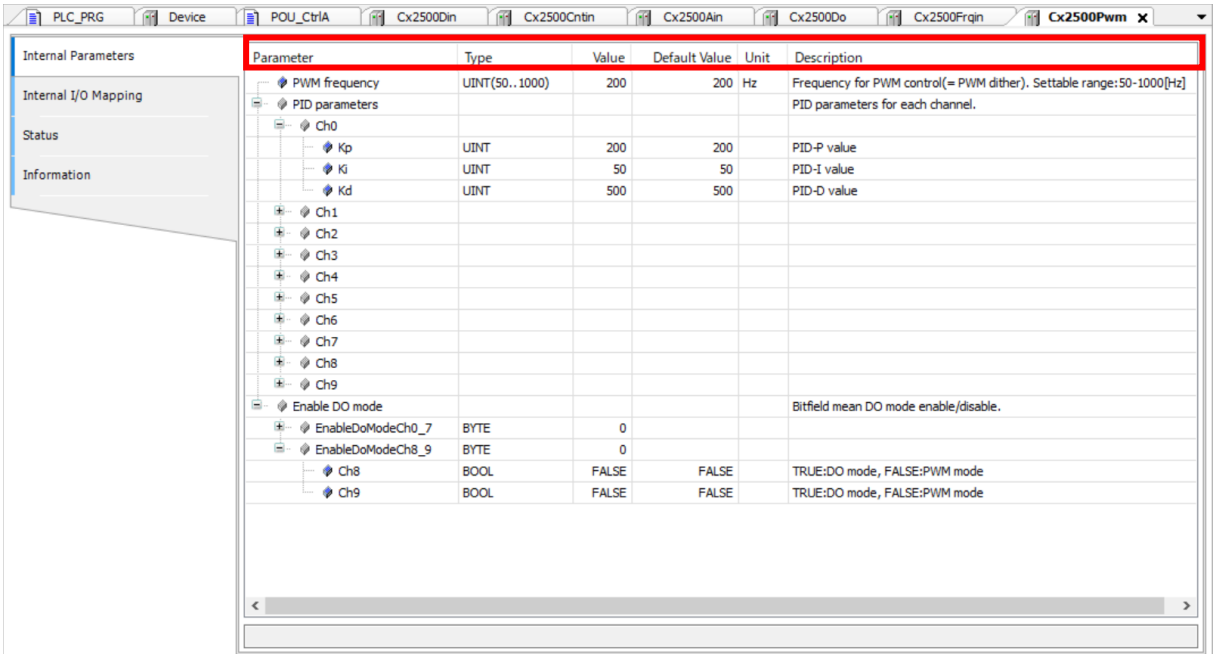


Figure 83 機能ドライバ Internal Parameters タブ選択時の画面

Table 27 Internal Parameters タブ 各列の概要

項目	ユーザー編集可否	摘要
Parameter	×	各要素の名称が表示されている。機能にチャネル毎に設定する要素がある場合はチャネル番号も表示されている。
Type	×	各要素に設定する値のデータ型が表示されている。
Value	○	ユーザー設定値。Value にユーザーが設定した値が、アプリケーション起動時に各要素に設定される。
Default Value	×	各要素のデフォルト値が表示されている。ユーザーが Value を設定しない場合はこの値が要素に設定される。
Unit	×	各要素の単位が表示されている。単位が無い場合は空欄。
Description	○	各要素の説明が表示されている。

7.2.3. Internal I/O Mapping タブ

Internal I/O Mapping タブは、機能の制御・状態モニタをおこなうために使われます。本タブで設定する必要がある項目については 7.2.3.1 項以降に記しています。

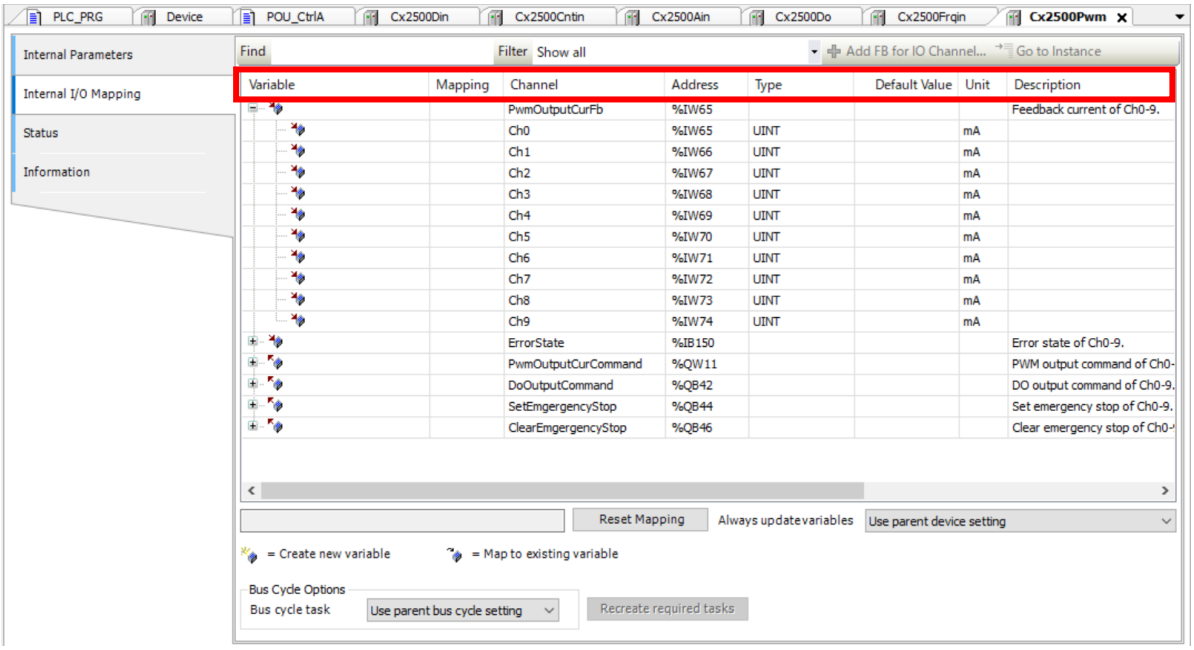


Figure 84 機能ドライバ Internal I/O Mapping タブ選択時の画面

Table 28 Internal I/O Mapping タブ 各列の概要

項目	ユーザー編集可否	摘要
Variable	○	ユーザー設定値。ユーザーアプリケーションで使用する変数を設定する。設定方法は 7.2.3.1 項参照。
Mapping	○	Variable に設定した変数がどこで宣言されて割り当てられたものか確認できる。  :本タブで入力し宣言された変数  :POU 側で宣言された変数
Channel	×	各要素の名称が表示されている。機能にチャネル毎に設定する要素がある場合はその枝にチャネル番号が表示されている。
Address	×	要素に割り当てた変数のアドレスが表示されている。
Type	×	各要素のデータ型が表示されている。
Default Value	○	ユーザー設定値。値を設定した時、アプリケーション起動時の初期値となる。
Unit	×	各要素の単位が表示されている。単位が無い場合は空欄。
Description	×	各要素の説明が表示されている。

7.2.3.1. 変数の割り当て

各機能を制御するには、このタブで各要素に変数を割り当てる必要があります。割り当て方は2通りあります。

【変数割り当て方法】

- Internal I/O Mapping 上で変数を入力する。
  - タブ上の Variable 列に直接入力すると自動的に宣言され要素に割り当てられる。
- POU 上で宣言している変数を入力する。
  - タブ上の Variable 列でクリックすると「…」ボタンが表示されるので、ボタンを押して POU 上で宣言された変数を選択する。その後下図のようになり要素にその変数が割り当てられる。

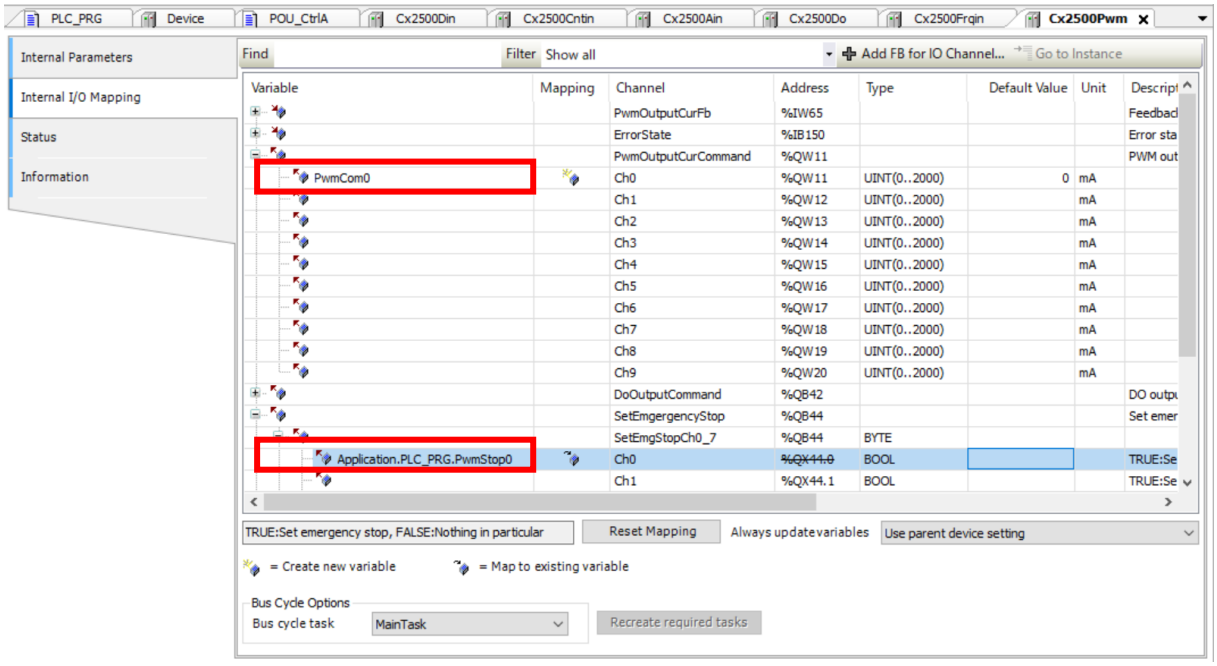


Figure 85 変数割り当て例

7.2.3.2. デフォルト値の設定

変数を割り当てした後、アプリケーション起動時の初期値となるデフォルト値を必ず設定して下さい。設定しない場合、アプリケーション起動後不定値となってしまい、想定しない制御・動作になる場合があります。

デフォルト値の設定は、下図のように変数を割り付けた要素の Default Value 列に値を入力することで設定できます。設定値が BOOL 型(TRUE 若しくは FALSE)の場合は、要素の Default Value 列でクリックすることによって値を設定できます。

なお、変数が以下に該当する場合、この機能ドライバ画面上で Default Value を入力・設定できません。ユーザは各プログラム(POU)上で変数を初期化する必要があります。

【機能ドライバ画面上で Default Value が設定できない変数】

- POU で宣言した変数
- PWM 出力の電流出力指令 FB(PwmOutputCurFb)など、ユーザが CX2500 に設定をおこなう機能に該当しないもの(各機能ドライバの部分 7.3～7.10 節を参照)

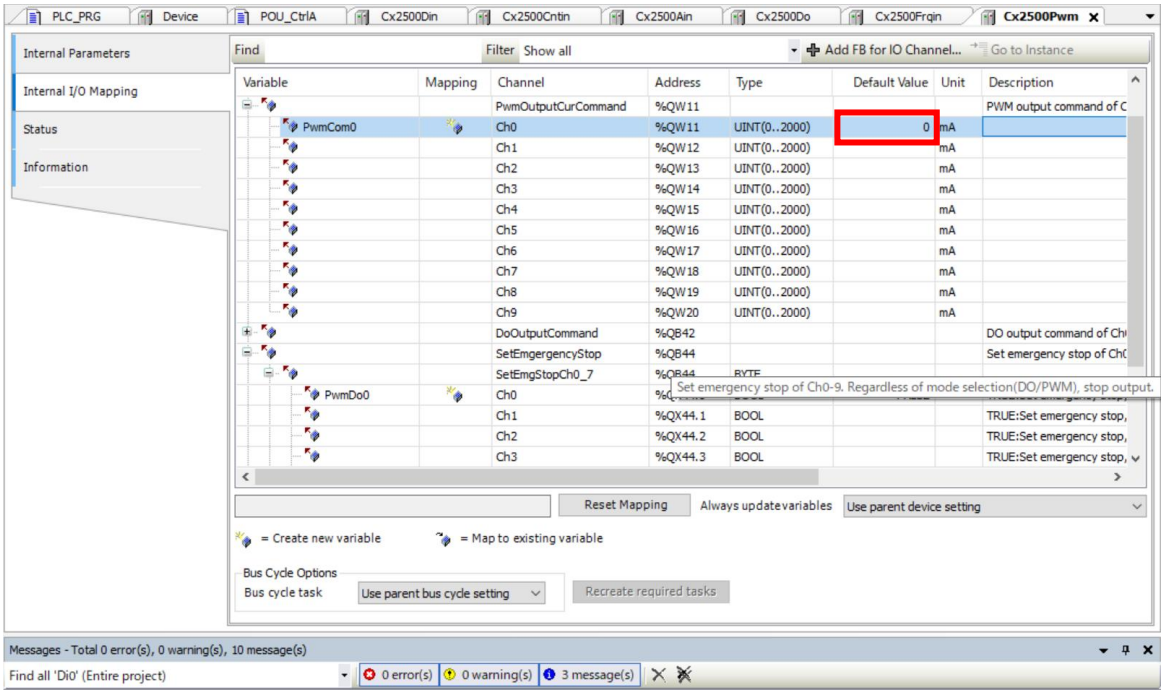


Figure 86 デフォルト値設定例

7.2.3.3. 機能ドライバ制御可否・サイクルの設定

機能ドライバを機能させるには、「変数を常に更新(Always update variables)」と「バスサイクルタスク(Bus cycle task)」の設定が必要になります。それぞれ選択タブから設定を選択して下さい。デフォルトはどちらも親デバイス(Device(CX2500))の設定に従うことになっています。

下図の例では、それぞれの設定を「親デバイスに従う」(ユーザーアプリケーションで処理に使用している変数のみ更新)にしています。Di2 はユーザーアプリケーションの POU で宣言していますが、処理に使用していません。その為制御値(ここではデジタル入力 Ch.2 の入力状態)が取得されません。

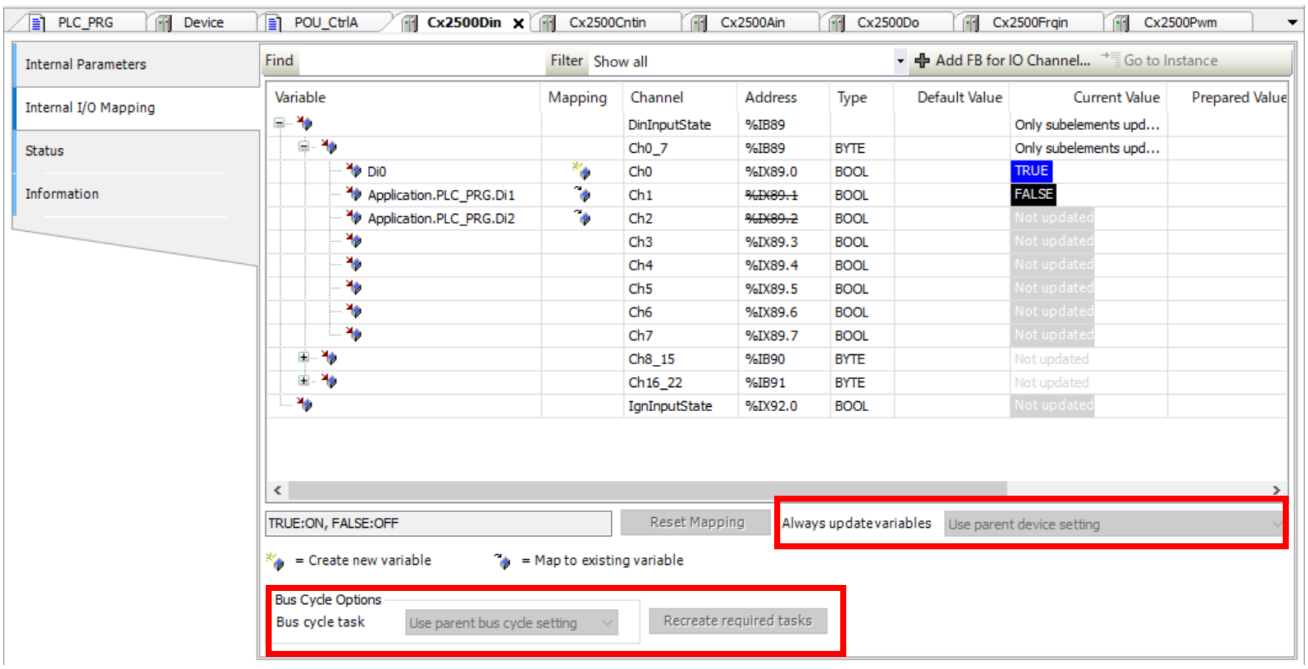


Figure 87 機能ドライバ 「変数を常に更新」と「バスサイクルタスク」の設定箇所

Table 29 変数を常に更新(Always update variables) 選択肢

選択肢	摘要
Use parent device setting	選択すると親デバイス設定(6.2 節の Always update variables で設定したもの)に従い、機能の各要素の制御値設定・取得する範囲を設定する。
Enabled 1 (use bus cycle task if not used in any task)※11	選択すると、ユーザーアプリケーションのタスクで使用していない変数、変数を割り当てていない要素も含め全ての要素の制御値設定・取得を毎サイクルおこなう。

⚠ 注意 ※11	この設定にすると、ユーザーアプリケーションの処理に使っていない要素にも数値が設定されてしまう。変数を割り当ててないものは起動時不定値になる。そのため、例えば PWM 出力値の場合、処理に使用していないチャネルの出力電流指令値に 2000mA 等の大きな値が設定されてしまう可能性があり、負荷が接続されている場合にユーザーが想定していない動作となり極めて危険である。よって、特別な事情が無い限り親デバイスの設定を Disable にした上で「Use parent device setting」を選択すること。

Table 30 バスサイクルタスク(Bus cycle task)※12 選択肢

選択肢	摘要
Use parent bus cycle setting	選択すると親デバイスのバスサイクル(6.2 節の Bus Cycle Options で設定したもの)と同じ周期で、機能の制御値設定・取得がおこなわれる。
(ユーザーが定義したタスク名)	選択したタスクのバスサイクルで機能の制御値設定・取得がおこなわれる。

※12 バスサイクルタスクを短い周期のタスク(例えば、数 ms 等)に極力設定しないでください。そうした場合、IO ドライバの入出力値は CX2500 内部で高頻度に更新処理をおこなうため、他の処理に遅れが生じる可能性があります。これは、IO ドライバに限らず通常のタスク時間設定の際も留意して下さい。

7.2.3.4. アプリケーションで使用しない機能ドライバの設定

機能ドライバの内、ユーザーで使用しない機能がある場合は機能ドライバの制御を無効化すると、アプリケーション内部で処理する時間を短縮することが可能です。ユーザーは、アプリケーションで使用しない機能ドライバについて、最低限下記設定をおこなう必要があります。

【設定値】

- Table 7 の「Always update variables」で「Disabled」を選択する。
- Table 29 で「Use parent device setting」を選択する。

7.3. デジタル入力・イグニッション入力

デジタル入力とイグニッション入力は機能ドライバ Cx2500Din を用いることで使用することができます。

7.3.1. Internal Parameters タブ

デジタル入力で初期設定する要素は入力形式の選択(Input selection)のみです。下表の解説を参考に設定して下さい。なお、イグニッション入力に関する設定要素はありません。

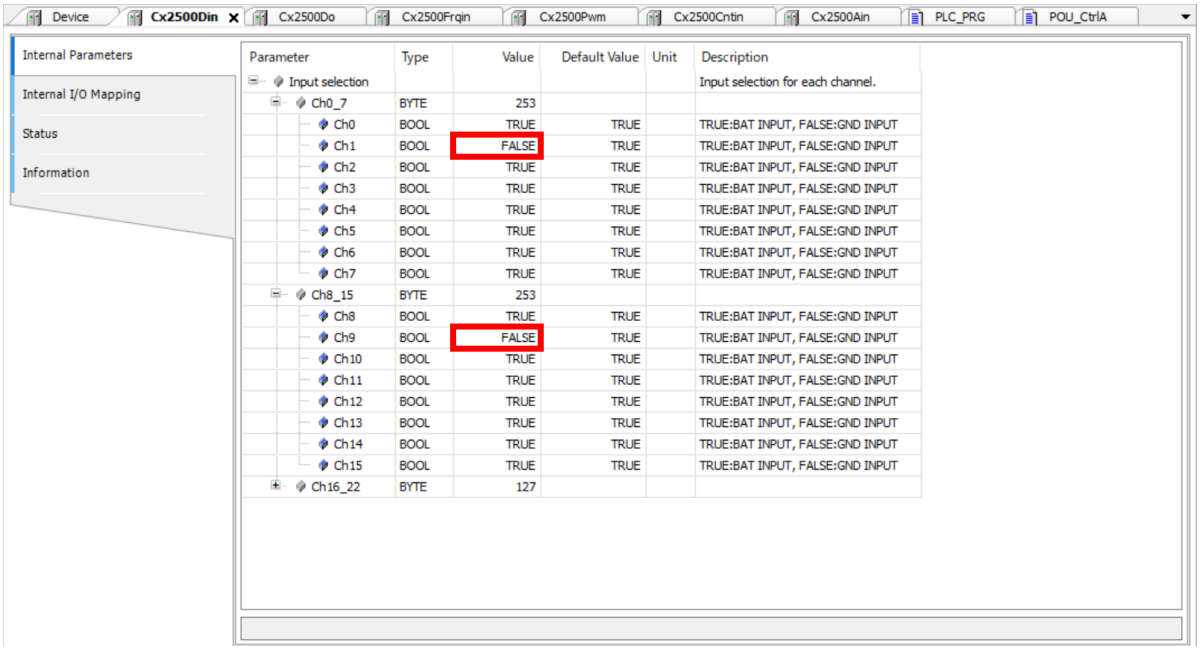


Figure 88 Cx2500Din Internal Parameters タブ画面

Ch.1 と Ch.9 をローサイド入力で設定した時の例(それ以外はハイサイド入力)

Table 31 デジタル入力 Internal Parameter タブ 要素一覧

要素名(項目)	データ型	設定範囲	摘要
Input selection	BOOL	TRUE/FALSE	・デジタル入力の入力形式(ハイサイド/ローサイド)の選択。 ・チャネル毎に設定できる。

設定値	チャネル設定
TRUE	ハイサイド入力
FALSE	ローサイド入力

7.3.2. Internal I/O Mapping タブ

Internal I/O Mapping タブではデジタル入力とイグニッション入力の入力状態を取得できます。なお、本機能ドライバでは変数の Default Value をドライバ画面上で直接設定することはできません。各 POU 上で初期値を設定して下さい。

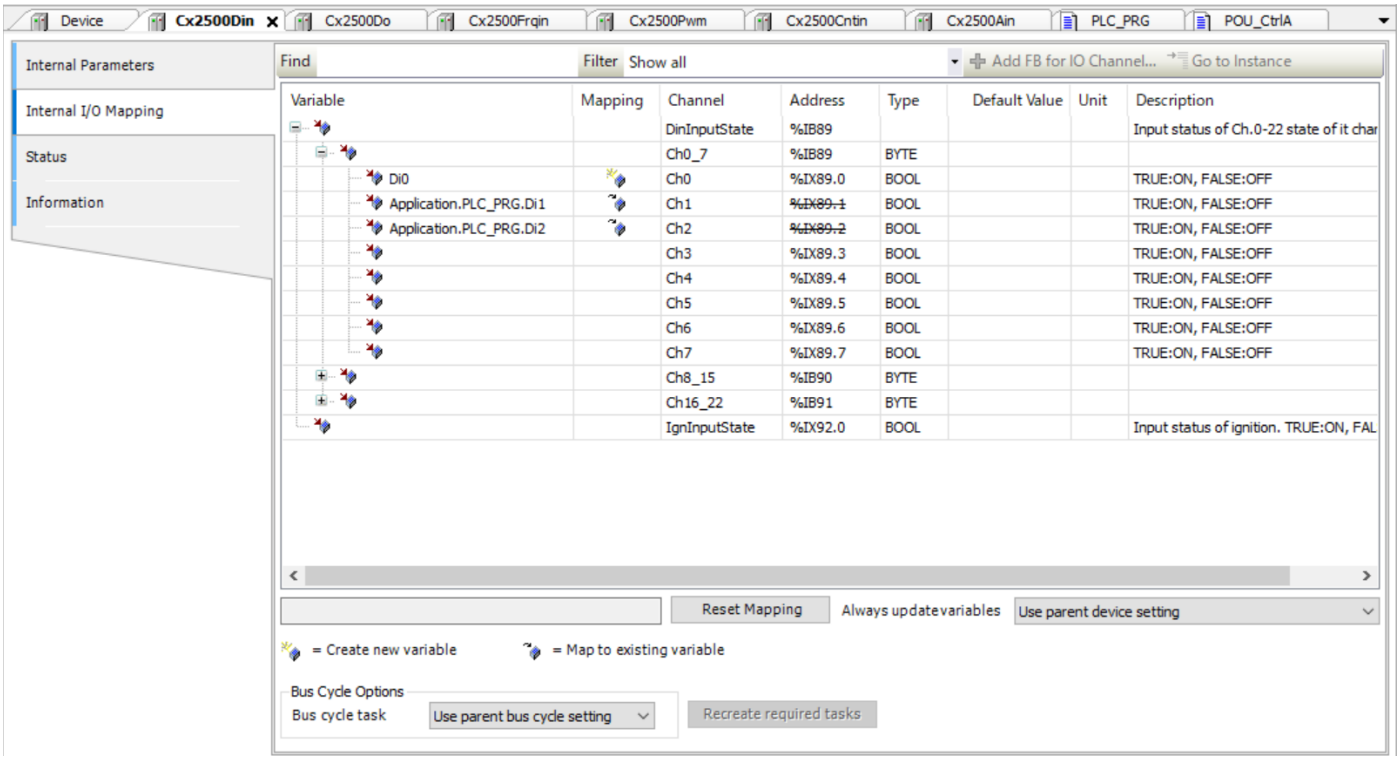


Figure 89 Cx2500Din Internal I/O Mapping タブ画面

Table 32 デジタル入力・イグニッション入力 Internal I/O Mapping タブ 要素一覧

要素名(Channel 列)	データ型	取得/設定値範囲	摘要
DinInputState	BOOL	TRUE/FALSE	・デジタル入力の入力状態(ON/OFF)。 ・チャネル毎に取得できる。 取得値: TRUE = ON, FALSE = OFF
IgnInputState	BOOL	TRUE/FALSE	・イグニッション入力の入力状態(ON/OFF)。 取得値: TRUE = ON, FALSE = OFF

7.4. 周波数入力

周波数入力は機能ドライバ Cx2500Frqin を用いることによって使用することができます。なお、周波数入力は初期設定する要素はありません。そのため、Internal Parameter タブもありません。

7.4.1. Internal I/O Mapping タブ

Internal I/O Mapping タブでは周波数入力各チャネルの入力周波数と ON/OFF 入力状態を取得できます。なお、本機能ドライバでは、変数の Default Value をドライバ画面上で直接設定することはできません。各 POU 上で初期値を設定して下さい。

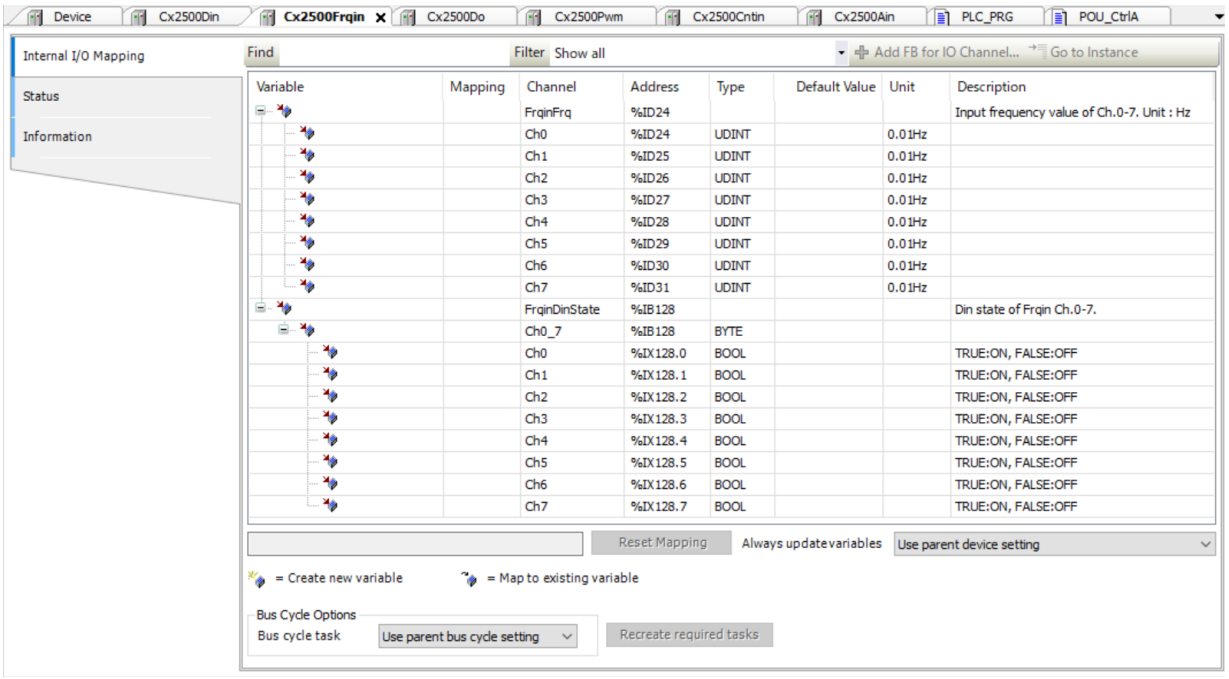


Figure 90 Cx2500Frqin Internal I/O Mapping タブ画面

Table 33 周波数入力 Internal I/O Mapping タブ 要素一覧

要素名(Channel 列)	データ型	取得/設定値範囲	摘要
FrqinFrq	UDINT	0.00～4000.00	・周波数入力の入力された周波数[0.01Hz]。 ・チャネル毎に取得できる。 ・ハードウェア仕様外の周波数が入力された場合、取得値は不定になる。
FrqinDinState	BOOL	TRUE/FALSE	・周波数入力の入力状態(ON/OFF)。 ・チャネル毎に取得できる。

取得値	チャネル状態
TRUE	ON
FALSE	OFF

7.5. 2 相カウンタ入力

2 相カウンタ入力は機能ドライバ Cx2500Cntin を用いることによって使用することができます。なお、2 相カウンタ入力は起動時にユーザーが設定する要素はありません。そのため、Internal Parameter タブもありません。

7.5.1. Internal I/O Mapping タブ

Internal I/O Mapping タブでは、2 相カウンタ入力のカウント値設定/取得や ON/OFF 入力状態を取得できます。なお、下記以外の機能については、変数の Default Value をドライバ画面上で直接設定することはできません。各 POU 上で初期値を設定して下さい。

【ドライバ画面上で Default Value を設定できる機能】

- EnableSetCount(2 相カウンタへのカウント値設定許可)
- CntinCnt_Set(2 相カウンタ入力の各チャネルへ設定するカウンタ値)

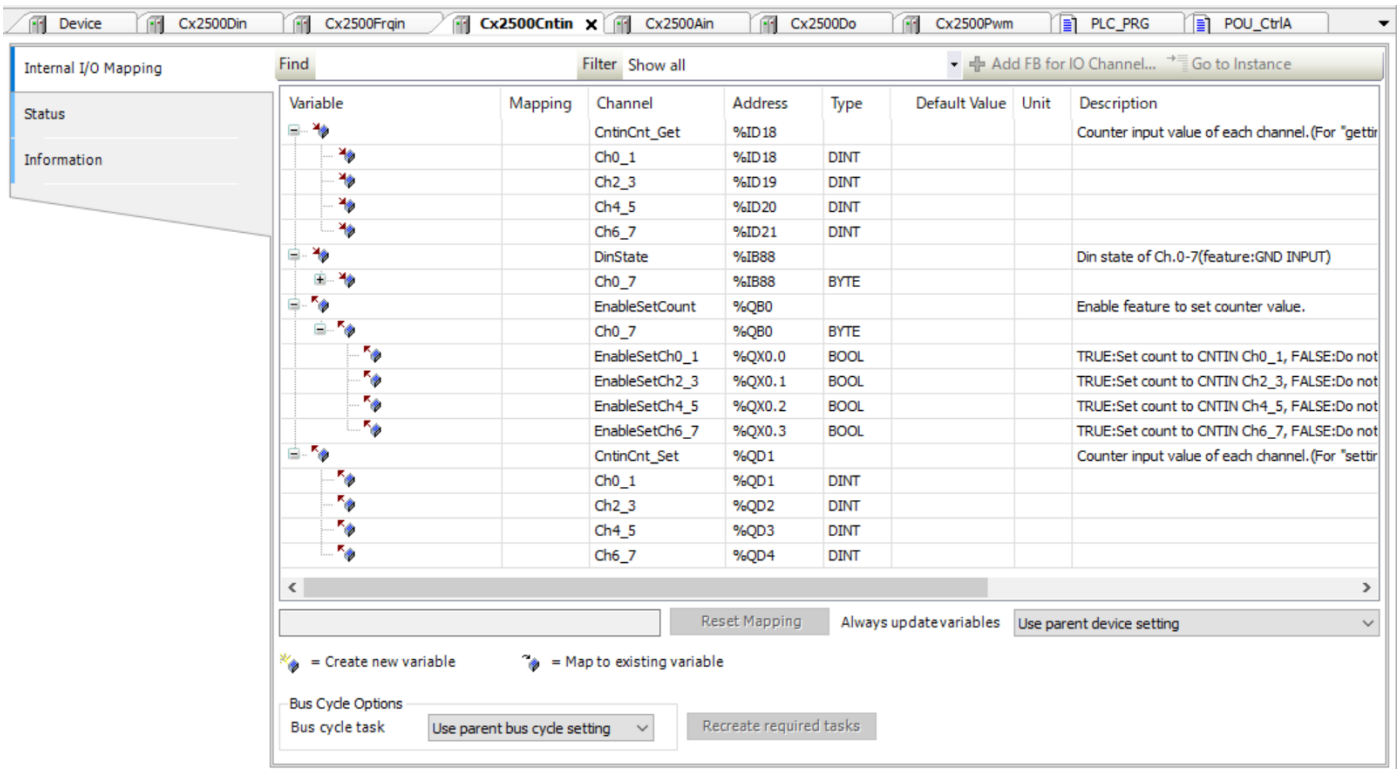


Figure 91 Cx2500Cntin Internal I/O Mapping タブ画面

Table 34 2 相カウンタ入力 Internal I/O Mapping タブ 要素一覧

要素名(Channel 列)	データ型	取得/設定値範囲	摘要
CntinCnt_Get	DINT	-2^{31}～$2^{31}-1$	・2 相カウンタ入力のカウント値(取得値)。 ・チャネル毎に取得できる。 ・カウント値(取得値)は左記の上/下限値に達した場合、それ以上の値へカウントアップ/ダウンしない。
DinState	BOOL	TRUE/FALSE	・2 相カウンタ入力の入力状態(ON/OFF)。 ・チャネル毎に取得できる。 取得値 / チャネル状態 TRUE / ON FALSE / OFF
EnableSetCount	BOOL	TRUE/FALSE	・2 相カウンタ入力へのカウント値設定許可。 ・値が TRUE である間、CntinCnt_Set に設定したカウント値を CX2500 に設定する。 設定値 / 処理内容 TRUE / 対応するチャネルへ CntinCnt_Set のカウンタ値を CX2500 に設定する。 FALSE / 対応するチャネルへのカウンタ値設定はおこなわない。
CntinCnt_Set	DINT	-2^{31}～$2^{31}-1$	・2 相カウンタ入力の各チャネルへ設定するカウント値。 ・チャネル毎に設定できる。 ・要素 EnableSetCount の対応するチャネルが TRUE であるときにカウンタ値が設定される。 ・左記設定値範囲外の値は設定できない。

7.5.2. カウンタ値セットの流れ

2 相カウンタ入力の各チャネルへカウント値をセットするには、プログラム上で下記のような流れでおこなう必要が有ります。ただし、EnableSetCount が TRUE の間毎サイクルカウンタ値が設定されてしまうので、毎サイクル設定する必要が無い場合は設定後にプログラム上で EnableSetCount を必ず FALSE にセットして下さい。

(1)要素 CntinCnt_Set へ所望のカウント値をセットする。
カウント値を設定したいチャネルの CntinCnt_Set にカウント値をセットする。

↓

(2) EnableSetCount を TRUE にセットする。
所望のチャネルの EnableSetCount を TRUE にセットする。

↓

(3)CX2500 にカウント値がセットされる。
カウント値が実際に設定されたかは要素 CntinCnt_Get で確認する。

↓

(4)EnableSetCount を FALSE にセットする。

Figure 92 2 相カウンタ入力 カウント値の設定フロー

7.6. アナログ入力

アナログ入力は機能ドライバ Cx2500Ain を用いることによって使用することができます。

7.6.1. Internal Parameters タブ

アナログ入力で初期設定する要素は入力形式の選択(Input selection)のみです。下表の解説を参考に設定して下さい。

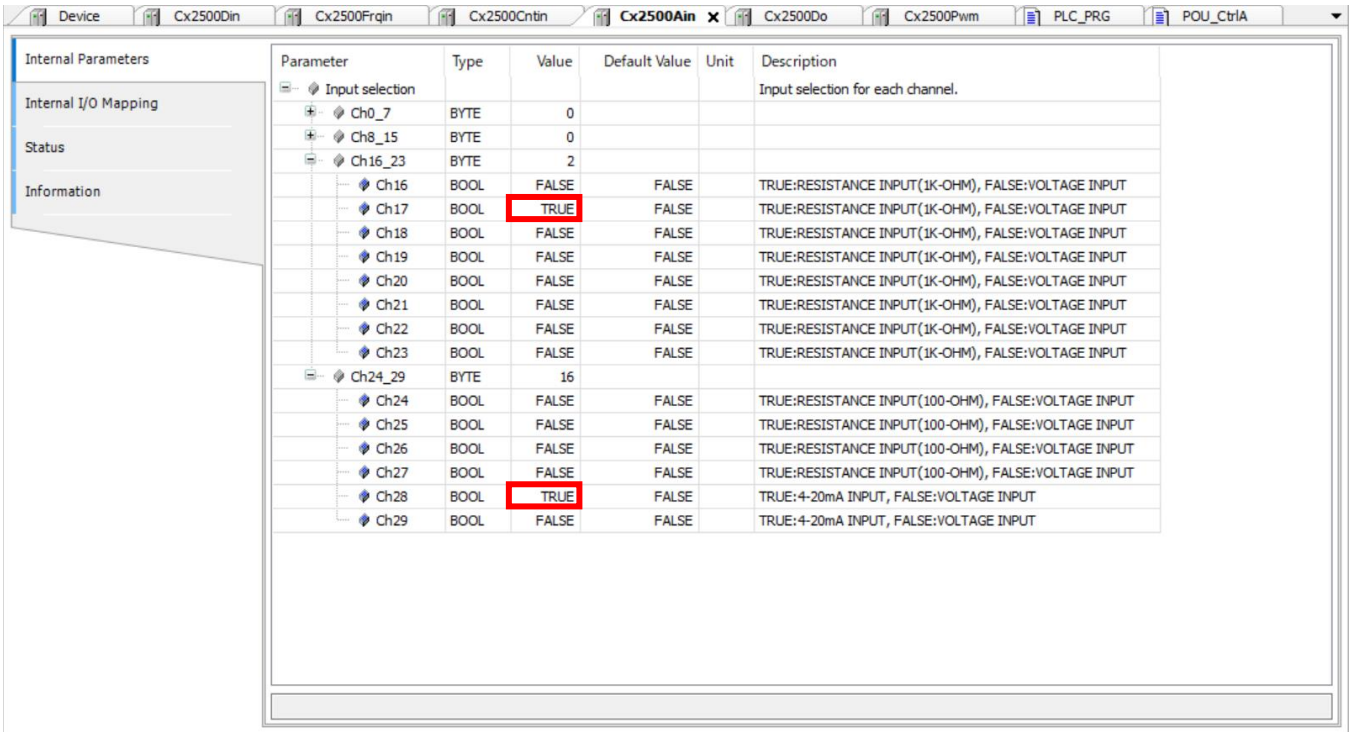


Figure 93 Cx2500Ain Internal Parameters タブ画面

Ch.17 を抵抗(1kΩ)入力、Ch.28 を 4-20mA 入力で設定した時の例(それ以外は 0-5V 電圧入力)

Table 35 アナログ入力 Internal Parameter タブ 要素一覧

要素名(項目)	データ型	設定範囲	摘要
Input selection	BOOL	TRUE/FALSE	・アナログ入力の入力形式の選択。 ・チャネル毎に設定できる。

設定値	チャネル設定
TRUE	Ch.0～15 :ON/OFF 入力(電圧入力 0～32V) Ch.16～23 :抵抗入力(内部 1kΩプルアップ) Ch.24～27 :抵抗入力(内部 100Ωプルアップ) Ch.28～29 :4-20mA 入力
FALSE	電圧入力(0～5V)

7.6.2. Internal I/O Mapping タブ

Internal I/O Mapping タブでは、アナログ入力の AD 値や ON/OFF 入力状態(Ch.0～15)の取得が可能です。なお、本機能ドライバでは、変数の Default Value をドライバ画面上で直接設定することはできません。各 POU 上で初期値を設定して下さい。

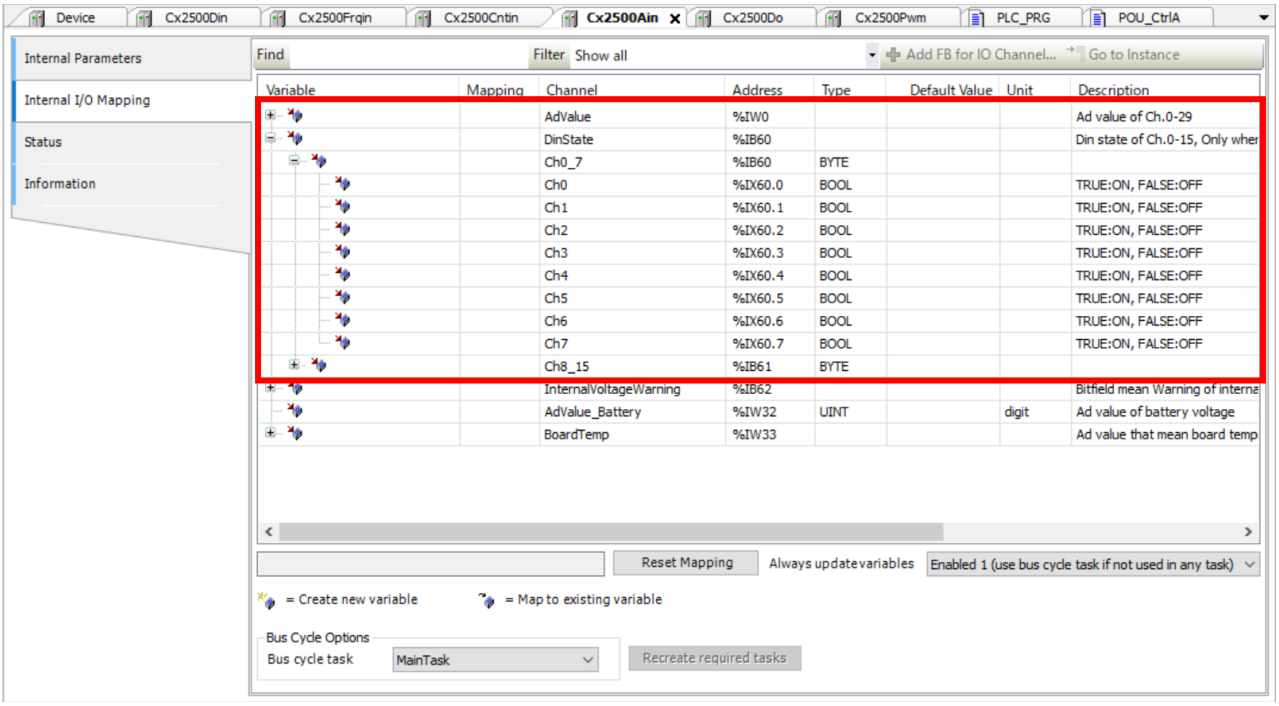


Figure 94 Cx2500Ain Internal I/O Mapping タブ画面(赤枠:アナログ入力部)

Table 36 アナログ入力 Internal I/O Mapping タブ 要素一覧

要素名(Channel 列)	データ型	取得/設定値範囲	摘要
AdValue	UINT	0～4095	・アナログ入力の AD 値[digit]。 ・チャネル毎に取得できる。
DinState	BOOL	TRUE/FALSE	・アナログ入力の Ch.0～15 の入力状態(ON/OFF)。 ・ただし、Input selection でチャネルの設定を TRUE(ON/OFF 入力)にしていないと取得できないことに注意。

取得値	チャネル状態
TRUE	ON
FALSE	OFF

7.7. 内部電源電圧監視入力

内部電源電圧監視入力は製品内部電源の異常監視するための機能です。機能ドライバ Cx2500Ain を用いることによって使用することができます。

7.7.1. Internal I/O Mapping タブ

機能ドライバ Cx2500Ain の Internal I/O Mapping タブでは、内部電源電圧監視の異常モニタ値やバッテリ電圧 AD 値の取得が可能です。なお、本機能ドライバでは、変数の Default Value をドライバ画面上で直接設定することはできません。各 POU 上で初期値を設定して下さい。

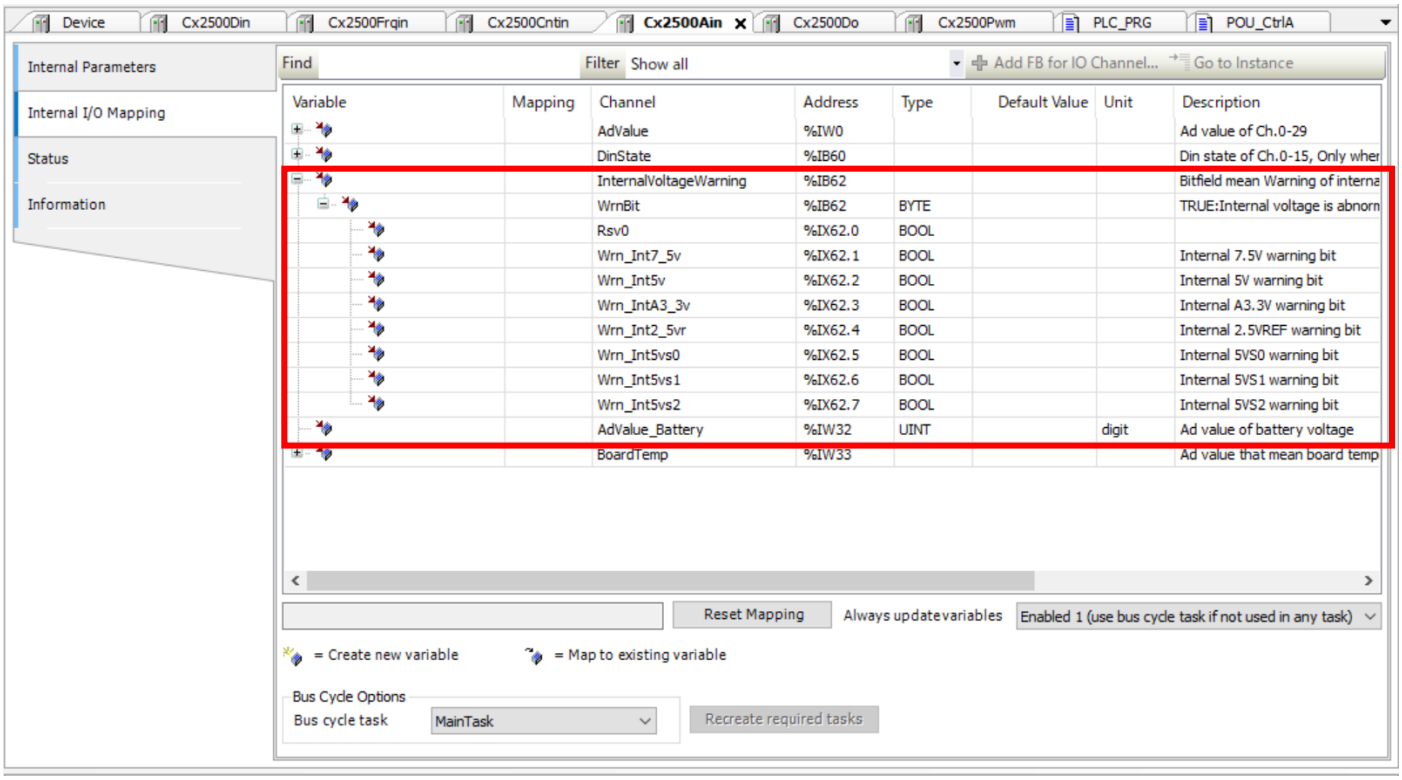


Figure 95 Cx2500Ain Internal I/O Mapping タブ画面(赤枠:内部電源電圧監視入力部)

Table 37 内部電源電圧監視入力 Internal I/O Mapping タブ 要素一覧

<table>
<tr><th>要素名(Channel 列)</th><th>データ型</th><th>取得/設定値
範囲</th><th>摘要</th></tr>
<tr><td>InternalVoltageWarning</td><td>BOOL</td><td>TRUE/FALSE</td><td>・各内部電源電圧の異常モニタ(TRUE/FALSE)。
・各要素の意味
<table><tr><th>要素名</th><th>摘要</th></tr><tr><td>Rsv0</td><td>予約ビット</td></tr><tr><td>Wrn_Int7_5v</td><td>内部電源 7.5V 異常モニタビット</td></tr><tr><td>Wrn_Int5v</td><td>内部電源 5V 異常モニタビット</td></tr><tr><td>Wrn_IntA3_3V</td><td>内部電源 3.3V 異常モニタビット</td></tr><tr><td>Wrn_Int2_5vr</td><td>内部電源 2.5V 異常モニタビット</td></tr><tr><td>Wrn_Int5vs0</td><td>センサ用電源 Ch.0 異常モニタビット</td></tr><tr><td>Wrn_Int5vs1</td><td>センサ用電源 Ch.1 異常モニタビット</td></tr><tr><td>Wrn_Int5vs2</td><td>センサ用電源 Ch.2 異常モニタビット</td></tr></table>
・取得値の意味
<table><tr><th>取得値</th><th>チャネル状態</th></tr><tr><td>TRUE</td><td>内部電源電圧が異常</td></tr><tr><td>FALSE</td><td>正常</td></tr></table></td></tr>
<tr><td>AdValue_Battery</td><td>UINT</td><td>0～4095</td><td>・バッテリ電圧の AD 値[digit]。
・AD 値から電圧値への換算式の一例は下記の通り。

【AD 値・電圧値換算式】
$AD = \frac{V}{V_{MCU}} \times AD_{MAX} \times R$
AD :AD 生値[digit]
AD_{MAX} :12bitAD 最大値。4095[digit]。
V_{MCU} :MCU の電源電圧。3.3[V]。
V :内部電源電圧[V]
R :分圧比(下表)

・監視対象ごとの分圧比 R の値(詳細は機能仕様書を参照。)
<table><tr><th>項目</th><th>R の値(TYP)</th></tr><tr><td>BAT</td><td>0.048</td></tr><tr><td>7.5V</td><td>0.248</td></tr><tr><td>5V、3.3VA、2.5VREF、5VS[0-2]</td><td>0.313</td></tr></table></td></tr>
</table>

7.8. 基板温度監視入力

基板温度監視入力は製品内部の基板上に搭載したサーミスタ(RT)による温度モニタ機能です。機能ドライバ Cx2500Ain を用いることによって使用することができます。

7.8.1. Internal I/O Mapping タブ

機能ドライバ Cx2500Ain の Internal I/O Mapping タブでは、基板温度 AD 値の取得が可能です。なお、本機能ドライバでは、変数の Default Value をドライバ画面上で直接設定することはできません。各 POU 上で初期値を設定して下さい。

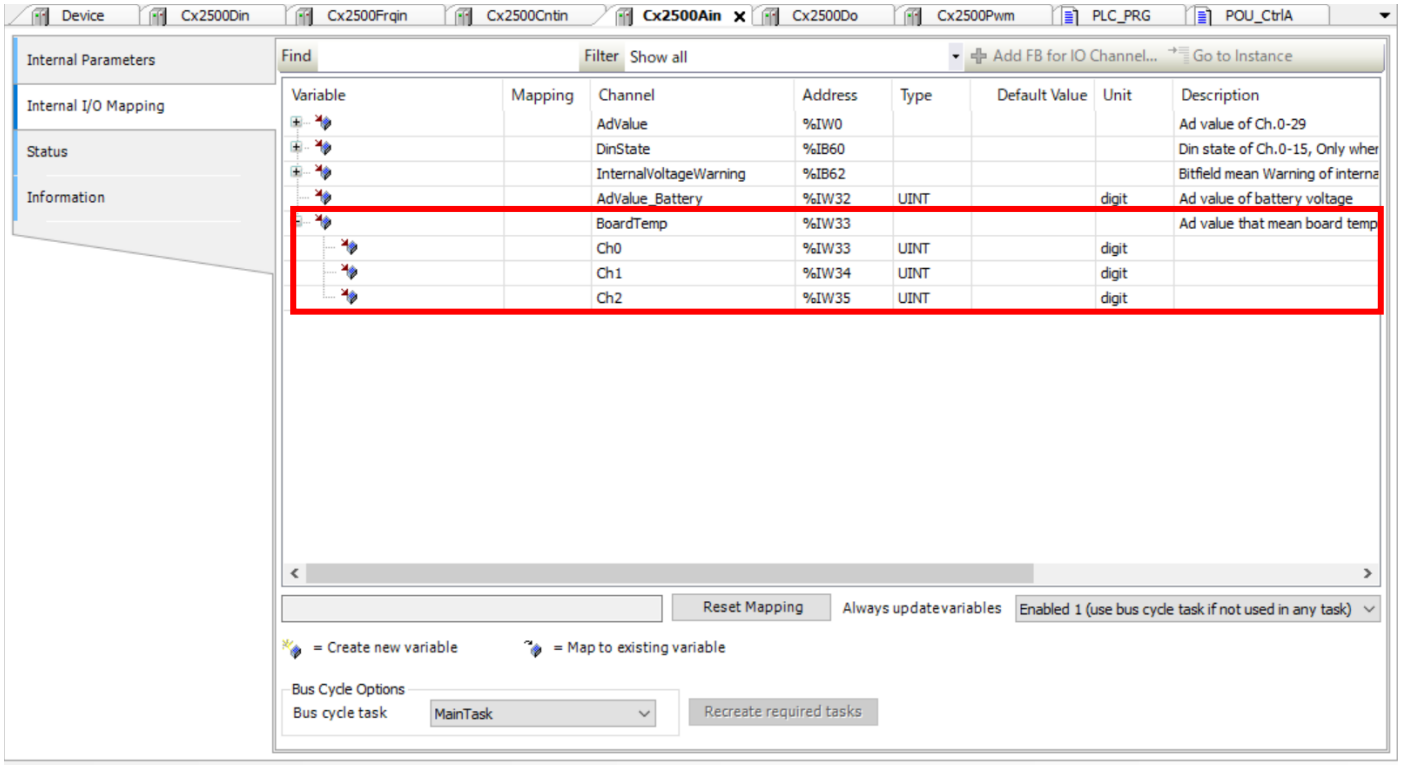


Figure 96 Cx2500Ain Internal I/O Mapping タブ画面(赤枠:基板温度監視入力)

Table 38 基板温度監視入力 Internal I/O Mapping タブ 要素一覧

<table>
<tr><th>要素名(Channel 列)</th><th>データ型</th><th>取得/設定値範囲</th><th>摘要</th></tr>
<tr><td>BoardTemp</td><td>UINT</td><td>0～4095</td><td>・製品基板上に搭載したサーミスタから得られる周囲温度の AD 値[digit]。
・チャネル毎に取得できる。
・取得値は AD 値のため、温度については下式に従い換算する。
<table>
<tr><th>チャネル番号</th><th>製品筐体内サーミスタ位置</th></tr>
<tr><td>Ch.0</td><td>MCU 周辺温度</td></tr>
<tr><td>Ch.1</td><td>パワー素子温度</td></tr>
<tr><td>Ch.2</td><td>コネクタ付近温度</td></tr>
</table>
</td></tr>
</table>

【AD 値→温度換算式】

(1)サーミスタの抵抗値：R[kΩ]

$$R = R_P \times (\frac{AD_{MAX} \times V_{MCU}}{V_{MCU} \times AD} - 1)^{-1}$$

(2)温度：T[℃]

$$T = (B^{-1} \times \ln\left(\frac{R}{R_0}\right) + {T_0}^{-1})^{-1} - 273$$

R_P	：入力プルアップ抵抗。4.7[kΩ]。
AD_{MAX}	：12bitAD 最大値。4095[digit]。
AD	：AD 値[digit]。
V_{MCU}	：MCU の電源電圧。3.3[V]。
B	：サーミスタ B 定数。3500[K]。
R_0	：常温 25[℃]時のサーミスタ抵抗値。10[kΩ]。
T_0	：常温 25[℃]+273=298[K]。

7.9. デジタル出力

デジタル出力は機能ドライバ Cx2500Do を用いることで使用することができます。

7.9.1. Internal Parameters タブ

デジタル出力で初期設定する要素は出力形式の選択(Output selection)のみです。下表の解説を参考に設定して下さい。

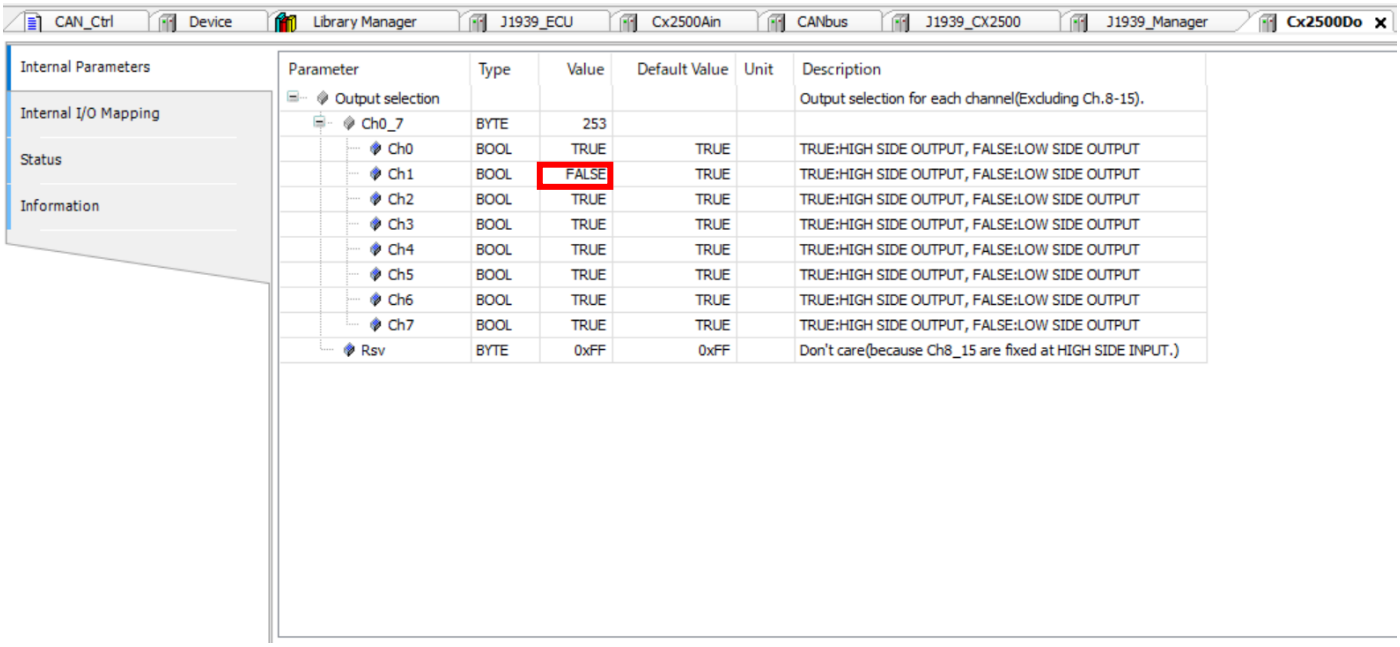


Figure 97 Cx2500Do Internal Parameters タブ画面

Ch.1 をローサイド出力で設定した時の例(それ以外はハイサイド出力)

Table 39 デジタル出力 Internal Parameters タブ 要素一覧

要素名(項目)	データ型	設定範囲	摘要
Output selection	BOOL	TRUE/FALSE	・デジタル出力の出力形式(ハイサイド/ローサイド)の選択。 ・チャネル毎に設定できる。(ただし、Ch.8～15 についてはハイサイド出力固定のため設定項目無し)

設定値	チャネル設定
TRUE	ハイサイド出力
FALSE	ローサイド出力

7.9.2. Internal I/O Mapping タブ

Internal I/O Mapping タブではデジタル出力の出力設定、及び出力状態の取得ができます。なお、下記以外の機能については、変数の Default Value をドライバ画面上で直接設定することはできません。各 POU 上で初期値を設定して下さい。

【ドライバ画面上で Default Value を設定できる機能】

- DoOutputCommand(デジタル出力の出力指示設定)

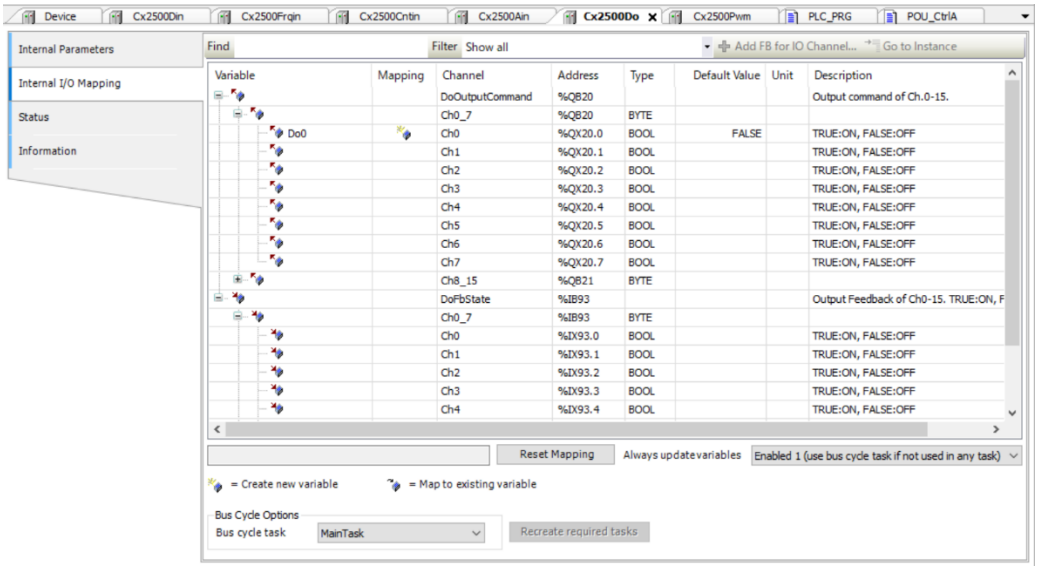


Figure 98 Cx2500Do Internal I/O Mapping タブ画面

Table 40 デジタル出力 Internal I/O Mapping タブ 要素一覧

要素名(Channel 列)	データ型	取得/設定値範囲	摘要
DoOutputCommand	BOOL	TRUE/FALSE	・デジタル出力の出力指示設定(ON/OFF)。 ・チャネル毎に設定できる。 設定値: TRUE → チャネル設定: ON 設定値: FALSE → チャネル設定: OFF
DoFbState※13	BOOL	TRUE/FALSE	・デジタル出力の出力状態(ON/OFF)。 ・チャネル毎に取得できる。 取得値: TRUE → チャネル状態: ON 取得値: FALSE → チャネル状態: OFF

※13 DoFbState は各チャネルの出力端子電圧(HIGH/LOW)で出力状態(ON/OFF)を判定している。そのため、端子に負荷を接続していないと電圧を正確に読み取れず、正しい出力状態を判定できないことに注意。

7.10. PWM 出力

PWM 出力は機能ドライバ Cx2500Pwm を用いることで使用することができます。

7.10.1. Internal Parameters タブ

PWM 出力で初期設定する要素は下表の通りです。下表の解説を参考に設定して下さい。

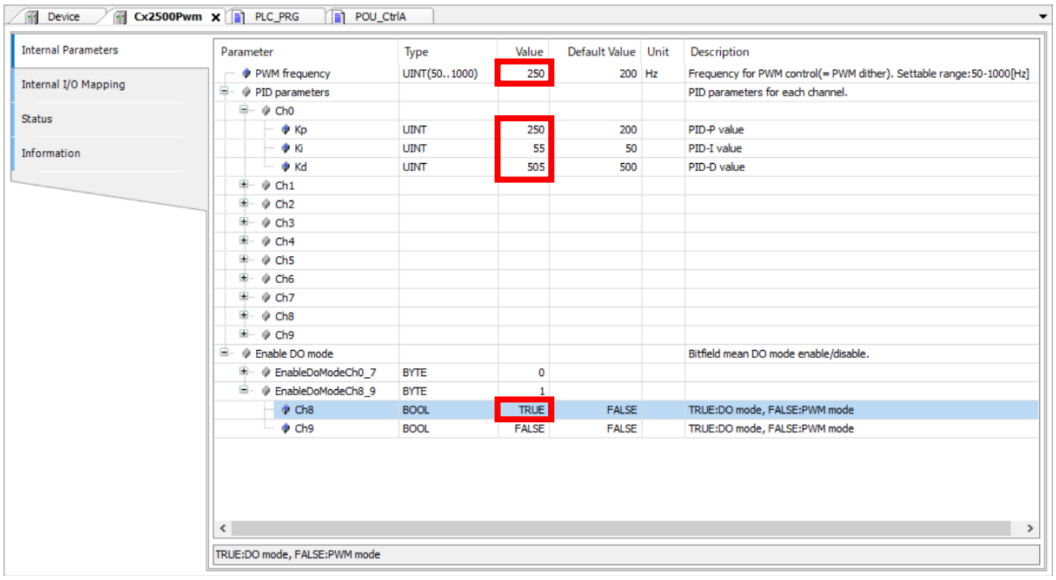


Figure 99 Cx2500Pwm Internal Parameterss タブ画面

上図はディザ周波数と Ch.0 の PID 値、Ch.8 の出力モードをデフォルト値から変更した時の例

Table 41 PWM 出力 Internal Parameters タブ 要素一覧

<table>
<tr><th>要素名(項目)</th><th>データ型</th><th>設定範囲</th><th>摘要</th></tr>
<tr><td>PWM frequency</td><td>UINT(50..1000)</td><td>50～1000</td><td>・PWM 出力全チャネルのディザ周波数(PWM 制御周波数)[Hz]。
・全チャネル共通のため、チャネル毎に個別に設定することはできない。</td></tr>
<tr><td>PID parameters</td><td>各種 UINT</td><td>各種 0～65535</td><td>・出力チャネルの PID パラメータの設定。
・チャネル毎に設定できる。
<table>
<tr><th>要素名</th><th>摘要</th></tr>
<tr><td>Kp</td><td>PID 制御用の P(比例)値</td></tr>
<tr><td>Ki</td><td>PID 制御用の I(積分)値</td></tr>
<tr><td>Kd</td><td>PID 制御用の D(微分)値</td></tr>
</table></td></tr>
<tr><td>Enable_DO_mode</td><td>BOOL</td><td>TRUE/FALSE</td><td>・出力チャネルの出力制御モード(7.10.3 項参照)の設定。
・チャネル毎に設定できる。
<table>
<tr><th>設定値</th><th>チャネル設定</th></tr>
<tr><td>TRUE</td><td>DO 出力モード</td></tr>
<tr><td>FALSE</td><td>PWM 出力モード</td></tr>
</table></td></tr>
</table>

7.10.2. Internal I/O Mapping タブ

Internal I/O Mapping タブでは PWM 出力の出力設定・状態取得等が可能です。なお、下記以外の機能については、変数の Default Value をドライバ画面上で直接設定することはできません。各 POU 上で初期値を設定して下さい。

【ドライバ画面上で Default Value を設定できる機能】

- PwmOutputCurCommand(各チャネルの出力電流指示設定)
- DoOutputCommand(各チャネルの DO 出力指示設定)
- SetEmergencyStop(出力チャネルへの緊急停止時設定)
- ClearEmergencyStop(緊急停止状態の出力チャネルへの緊急停止解除設定)

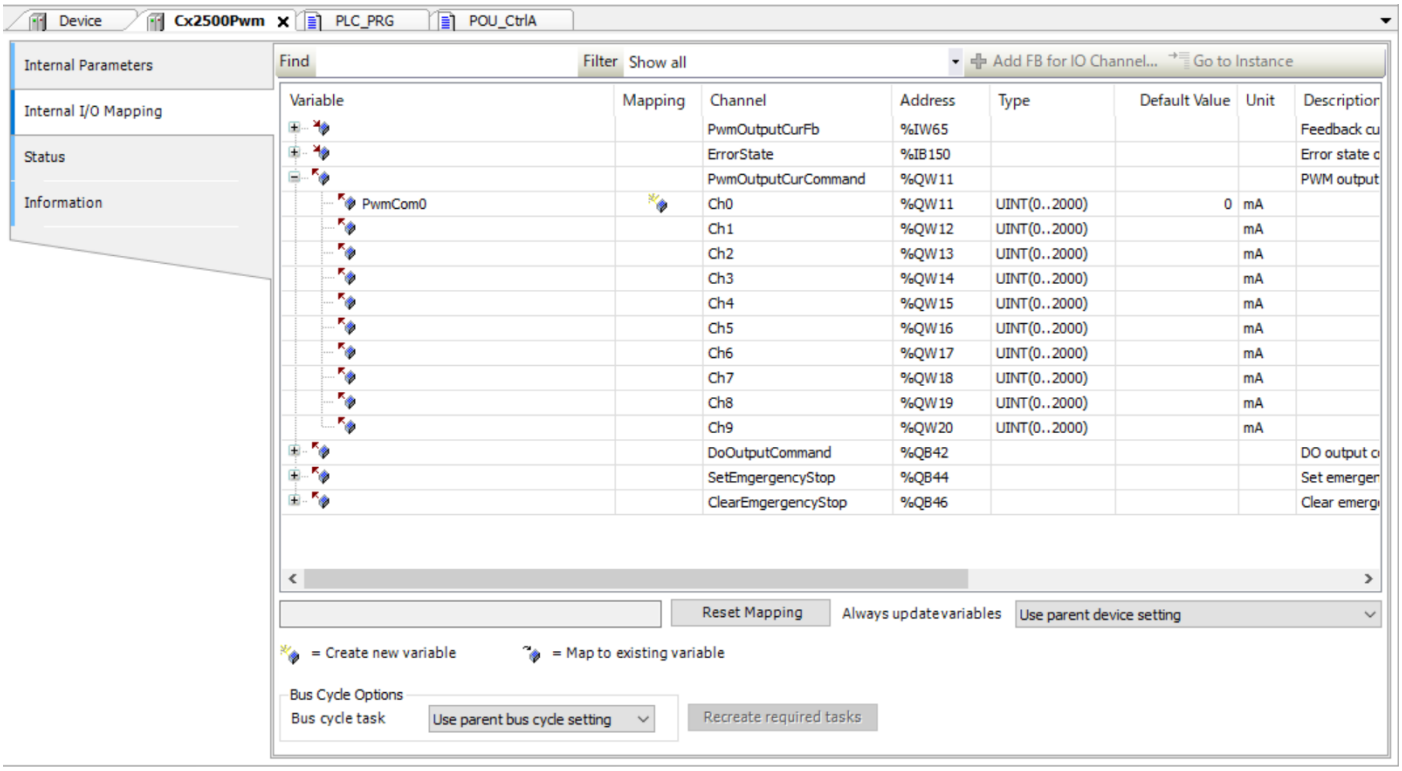


Figure 100 Cx2500Pwm Internal I/O Mapping タブ画面

Table 42 PWM 出力 Internal I/O Mapping タブ 要素一覧(1/2)

<table>
<tr><th>要素名(Channel 列)</th><th>データ型</th><th>取得/設定値
範囲</th><th>摘要</th></tr>
<tr><td>PwmOutputCurFb</td><td>UINT</td><td>0～65535</td><td>・PWM 出力の出力電流フィードバック[mA]。
・チャネル毎に取得できる。</td></tr>
<tr><td>ErrState</td><td>BOOL</td><td>TRUE/FALSE</td><td>・PWM 出力のチャネルステータス。
・チャネル毎に取得できる。
・エラーについては 7.10.5 項を参照。
・各要素の意味
<table>
<tr><th>要素名</th><th>摘要</th></tr>
<tr><td>EmergencyStop</td><td>緊急停止ステータス</td></tr>
<tr><td>ShortCircuit</td><td>短絡エラーステータス</td></tr>
<tr><td>WireBreak</td><td>断線エラーステータス</td></tr>
</table>
・エラー状態
<table>
<tr><th>取得値</th><th>ステータス</th></tr>
<tr><td>TRUE</td><td>エラー</td></tr>
<tr><td>FALSE</td><td>正常</td></tr>
</table></td></tr>
<tr><td>PwmOutputCurCommand</td><td>UINT(0..2000)</td><td>0～2000</td><td>・各チャネルの出力電流指示設定[mA]。
・チャネル毎に設定できる。
・DO モードのチャネルはこの要素をセットしても CX2500 には設定されない。
・下記組み合わせのチャネルは同時に出力できず、それぞれ一方のみ出力できる。
・Ch.0 と Ch.1
・Ch.2 と Ch.3
・Ch.4 と Ch.5
・Ch.6 と Ch.7
・Ch8 と Ch.9</td></tr>
<tr><td>DoOutputCommand</td><td>BOOL</td><td>TRUE/FALSE</td><td>・各チャネルの DO 出力指示設定(TRUE/FALSE)。
・チャネル毎に設定できる。
・PWM モードのチャネルはこの要素をセットしても CX2500 には設定されない。
・PwmOutputCurCommand と同様に隣り合うチャネルは同時出力できないことに注意。
<table>
<tr><th>設定値</th><th>チャネル設定</th></tr>
<tr><td>TRUE</td><td>DO モード:ON</td></tr>
<tr><td>FALSE</td><td>DO モード:OFF</td></tr>
</table></td></tr>
</table>

Table 43 PWM 出力 Internal I/O Mapping タブ 要素一覧(2/2)

<table>
<tr><th>要素名(Channel 列)</th><th>データ型</th><th>取得/設定値
範囲</th><th>摘要</th></tr>
<tr><td>SetEmergencyStop</td><td>BOOL</td><td>TRUE/FALSE</td><td>・出力チャネルへの緊急停止指示設定(TRUE/FALSE)。
・TRUE の間は、毎バスサイクル指定チャネルを緊急停止し続けることに注意。
<table>
<tr><th>設定値</th><th>チャネル設定</th></tr>
<tr><td>TRUE</td><td>出力強制停止・緊急停止状態へ移行</td></tr>
<tr><td>FALSE</td><td>処理無し</td></tr>
</table></td></tr>
<tr><td>ClearEmergencyStop</td><td>BOOL</td><td>TRUE/FALSE</td><td>・緊急停止状態の出力チャネルへの緊急停止解除指示設定(TRUE/FALSE)。
・TRUE の間は、毎バスサイクル指定チャネルの緊急停止を解除し続けることに注意。
<table>
<tr><th>設定値</th><th>チャネル設定</th></tr>
<tr><td>TRUE</td><td>エラー解除・出力可能状態</td></tr>
<tr><td>FALSE</td><td>処理無し</td></tr>
</table></td></tr>
</table>

7.10.3. DO モード

PWM 出力チャネルの出力制御モードは 2 つあります。これらは機能ドライバ Cx2500Pwm の Internal Parameter タブ・Enable_DO_mode 要素で設定します。各チャネルは、この Enable_DO_mode 要素で設定したモードでしか起動後出力制御できません。そのため、チャネルに接続する負荷の用途に合わせて出力モードを選択し使用して下さい。

Table 44 PWM 出力 出力モード一覧

モード名	摘要
PWM モード	・チャネルを PWM 出力として使用できる。 ・PwmOutputCurCommand 要素の設定が有効になる。 ・DoOutputCommand 要素の設定が無効になる。
DO モード	・チャネルをデジタル出力(ハイサイド出力)として使用できる。 ・DoOutputCommand 要素の設定が有効になる。 ・PwmOutputCurCommand 要素の設定が無効になる。

7.10.4. 初期状態

PWM 出力のチャネルは安全性を考慮し、起動時全て緊急停止状態にしています(ErrState 要素で確認可能)。そのため、各チャネルは緊急停止状態を解除(ClearEmergencyStop 要素を TRUE 設定)しないと出力できないことに注意して下さい。

7.10.5. エラー時の処理と解除方法

PWM 出力には短絡エラーと断線エラーという 2 つのエラーがあります。それぞれエラーを検知した時、検知したチャネルは出力を停止し、緊急停止状態(ErrState 要素の EmergencyStop が TRUE)に移行します。緊急停止状態に移行した後に再度出力をおこないたい場合、まずは緊急停止状態の解除(ClearEmergencyStop 要素を TRUE にセット)をおこなう必要があります。緊急停止状態を解除すると ErrState 要素の EmergencyStop が FALSE になります。解除後、出力設定された値で出力を自動で再開されることに注意して下さい。

7.11. RS232C

ユーザーは RS232C(シリアル通信)を使用することが可能です。機能を使用するには、ライブラリ SysCom が登録されている必要があります。使用できる関数は下記の通りです。
また、SysCom ライブラリについては CODESYS オンラインヘルプにも記載があります。そちらも合わせて参照して下さい。

Table 45 RS232C(SysCom ライブラリ) 関数一覧

機能区分	関数名	摘要
ポート制御	SysComOpen2	シリアル通信ポートをオープンする。
	SysComClose	シリアル通信ポートをクローズする。
受信	SysComRead	内部バッファから受信したメッセージを取得する。
送信	SysComWrite	メッセージを送信する。
非対応	SysComGetSettings	非対応
	SysComGetSettings2	
	SysComOpen	
	SysComOpen3	
	SysComPurge	
	SysComSetSettings	
	SysComSetSettings2	
	SysComSetTimeout	

7.11.1. 列挙型

RS232C で使用する SysCom ライブラリで定義されている列挙型は下記の通りです。ただし、本製品非対応のものは除きます。

型名	SYS_COM_BAUDRATE		
摘要	RS232C のボーレートを表す。		
列挙子	名前	値	説明
	SYS_BR_4800	4800	4800bps
	SYS_BR_9600	9600	9600bps
	SYS_BR_19200	19200	19200bps
	SYS_BR_38400	38400	38400bps
	SYS_BR_57600	57600	57600bps
	SYS_BR_115200	115200	115200bps

型名	SYS_COM_PARITY		
摘要	RS232C のパリティチェックの種類を表す。		
列挙子	名前	値	説明
	SYS_NOPATIRY	0	パリティチェック無効
	SYS_ODDPARITY	1	奇数パリティチェック
	SYS_EVENPARITY	2	偶数パリティチェック

型名	SYS_COM_PORTS		
摘要	RS232C のポート番号を表す。		
列挙子	名前	値	説明
	SYS_COMPORT_NONE	0	ポート無し(非対応)
	SYS_COMPORT1	1	ポート 1(RS232C の Ch.0 にあたる)
	SYS_COMPORT2	2	ポート 2(RS232C の Ch.1 にあたる)
	SYS_COMPORT3	3	ポート 3(非対応)
	SYS_COMPORT4	4	ポート 4(非対応)

型名	SYS_COM_STOPBITS		
摘要	RS232C のストップビット長を表す。		
列挙子	名前	値	説明
	SYS_ONESTOPBIT	1	1 ビット
	SYS_ONE5STOPBITS	2	(非対応)
	SYS_TWOSTOPBITS	3	2 ビット

型名	SYS_COM_TIMEOUT		
摘要	RS232C の送受信タイムアウト待ち時間を表す。		
列挙子	名前	値	説明
	SYS_NOWAIT	0	タイムアウト待ち時間無し
	SYS_INFINITE	0xFFFFFFFF	タイムアウト待ち時間無限(処理待ちし続ける)
備考	CX2500 内部処理の仕様上、非対応。この列挙型を使用した関数の呼び出しの際には任意値を設定すること。		

7.11.2. 構造体

RS232C で使用する SysCom ライブラリで定義されている構造体は下記の通りです。ただし、本製品非対応のものは除きます。

型名	SysComSettings		
摘要	RS232C ポートの通信パラメータ設定用構造体		
メンバ	型	名前	説明
	SYS_COM_PORTS	sPort	RS232C ポート番号
	SYS_COM_STOPBITS	byStopBits	ストップビット
	SYS_COM_PARITY	byParity	パリティ
	SYS_COM_BAUDRATE	ulBaudrate	ボーレート
	SYS_COM_TIMEOUT	ulTimeour	タイムアウト待ち時間(非対応)
	UDINT	ulBufferSize	受信バッファサイズ(非対応)

型名	SysComSettingsEx		
摘要	RS232C ポートの通信拡張機能パラメータ設定用構造体		
メンバ	型	名前	説明
	BYTE	byByteSize	(非対応)
	BOOL	bBinary	
	BOOL	bOuttxCtsFlow	
	BOOL	bOuttxDsrFlow	
	BOOL	bDtrControl	
	BOOL	bDsrSensitivity	
	BOOL	bRtsControl	
	BOOL	bTXContinueOnXoff	
	BOOL	bOutX	
	BOOL	bInX	
	BYTE	byXOnChar	
	BYTE	byXoffChar	
	WORD	wXonLim	
	WORD	wXoffLim	
備考	SysComOpen2 関数の引数ではあるが、CX2500 内部処理の仕様上、本構造体のメンバは使用されない。よって、関数の引数として使用する際は任意値を設定すること。		

7.11.3. 関数

RS232C で使用できる関数について、それぞれ下記に示します。ただし、本製品非対応のものは除きます。

関数名	SysComOpen2			
摘要	RS232C ポートをオープンする。			
引数 (INPUT)	型	名前	説明	
	POINTER_TO SysComSettings	pSettings	通信パラメータ	
	POINTER_TO SysComSettingsEx	pSettingsEx	通信拡張機能用パラメータ(非対応)	
	POINTER_TO RTS_IEC_RESULT	pResult	ERR_OK(0x0)	:オープン成功
			ERR_PARAMETER(0x2)	:オープン失敗(引数異常)
			ERR_NOTINITIALIZED(0x3)	:オープン済み
引数 (IN_OUT)	無し			
返り値 (OUTPUT)	型	名前	説明	
	RTS_IEC_HANDLE	SysComOpen2	オープンしたポートのポート番号	
備考	・本関数呼び出し後受信が開始される。 ・構造体 pSettings のメンバ sPort には必ず SYS_COMPORT1 又は SYS_COMPORT2 を入れること。 ・構造体 pSettingsEx は C2500 内部処理に不使用の為、関数を呼び出す際は任意値をそれぞれ設定すること。			

関数名	SysComClose			
摘要	RS232C ポートをクローズする。			
引数 (INPUT)	型	名前	説明	
	RTS_IEC_HANDLE	hCom	ポート番号(1 又は 2 を設定すること)	
引数 (IN_OUT)	無し			
返り値 (OUTPUT)	型	名前	説明	
	RTS_IEC_RESULT	SysComClose	ERR_OK(0x0)	:クローズ成功
			ERR_PARAMETER(0x2)	:クローズ失敗(引数異常)
備考	・送受信を中止、これまでに受信したバッファをクリアし、ポートをクローズする。 ・引数 hCom には必ず 1 又は 2 を設定すること。			

関数名	SysComRead		
摘要	内部バッファから受信したメッセージを取得する。		
引数 (INPUT)	型	名前	説明
	RTS_IEC_HANDLE	hCom	ポート番号(1 又は 2 を設定すること)
	POINTER_TO_BYTE	pbyBuffer	受信データ取得用データバッファのポインタ
	UDINT	ulSize	取得したい受信データサイズ
	SYS_COM_TYMEOUT	ulTimeout	受信タイムアウト待ち時間(非対応)
	POINTER_TO RTS_IEC_RESULT	pResult	ERR_OK(0x0) :取得成功
			ERR_FAULED(0x1) :取得失敗(受信バッファオーバーフロー、通信エラー)
			ERR_PARAMETER(0x2) :取得失敗(引数異常)
引数 (IN_OUT)	無し		
返り値 (OUTPUT)	型	名前	説明
	UDINT	SysComRead	取得できたデータのサイズ
備考	・引数 hCom には必ず 1 又は 2 を設定すること。 ・CX2500 内の受信バッファサイズは 512。ユーザーが本関数を呼び出す前にそのサイズより多いデータを受信していた場合、CX2500 は受信バッファオーバーフローエラーとなる。 ・本関数呼び出し後、引数 pResult が ERR_FAILED になった場合はエラーが発生しており、内部で送受信ができない状態になっている。再度送受信をおこなう場合はポートクローズ→オープンをおこなう必要がある。		

関数名	SysComWrite			
摘要	メッセージの送信設定・送信をおこなう。			
引数 (INPUT)	型	名前	説明	
	RTS_IEC_HANDLE	hCom	ポート番号(1 又は 2 を設定すること)	
	POINTER_TO_BYTE	pbyBuffer	送信データバッファのポインタ	
	UDINT	ulSize	送信したいデータサイズ	
	SYS_COM_TYMEOUT	ulTimeout	送信タイムアウト待ち時間(非対応)	
	POINTER_TO RTS_IEC_RESULT	pResult	ERR_OK(0x0)	:送信設定成功
			ERR_FAILED(0x1)	:送信設定失敗(受信バッファオーバーフロー、通信エラー)
			ERR_PARAMETER(0x2)	:送信設定失敗(引数異常)
			ERR_PENDING(0xA)	:送信設定成功(備考参照)
引数 (IN_OUT)	無し			
返り値 (OUTPUT)	型	名前	説明	
	UDINT	SysComWrite	送信設定できたデータのサイズ	
備考	・引数 hCom には必ず 1 又は 2 を設定すること。 ・CX2500 内の送信バッファサイズは 512。そのサイズより多いデータを送信設定しようとした場合、pResult は ERR_PENDING となる。バッファサイズを超えている分のデータは、先に送信設定されているデータが送信完了した後、CX2500 が自動で送信する。 ・本関数で多量のデータを速い周期で呼び出し続ける場合、CX2500 の送信処理が完了していない場合がある。その場合も ERR_PENDING となる。この場合、送信周期を長くしたり、1 度に送るデータ量を少なくする必要がある。 ・本関数呼び出し後、引数 pResult が ERR_FAILED になった場合はエラーが発生しており、内部で送受信ができない状態になっている。再度送受信をおこない場合はポートクローズ→オープンをおこなう必要がある。			

7.11.4. ポートオープン・クローズ

RS232C でメッセージの送受信を行うには、SysComOpen2 関数を呼び出し、ポートオープンする必要があります。ポートオープン後、メッセージ受信待ち状態になり送受信が可能になります。

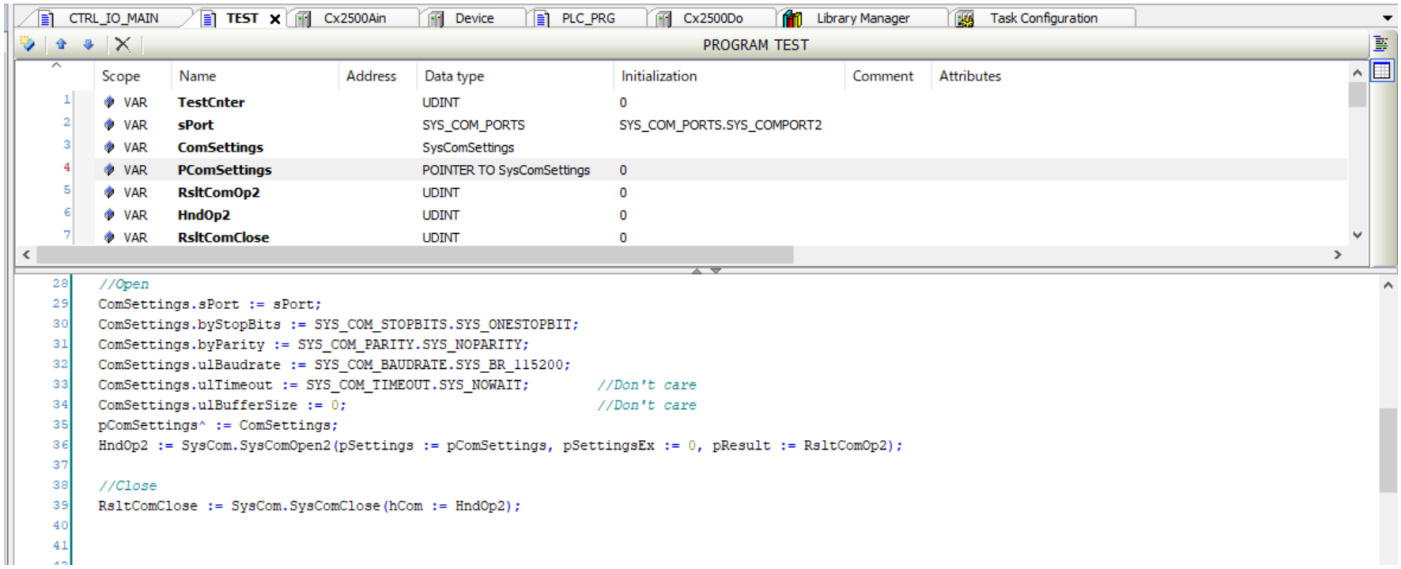


Figure 101 RS232C ポートオープン・クローズ呼出例(ST 言語)

7.11.5. ポートクローズ

通信エラーなどで受信・送信ができなくなった場合、SysComClose 関数を呼び出しポートクローズして下さい。ポートクローズ後、エラークリアされます。また、それまで内部で保持していた送受信バッファのデータは全て 0 クリアされることに留意して下さい。ポートクローズの呼び出し例は Figure 101 を参照して下さい。

7.11.6. 受信

CX2500 は内部で RS232C の受信バッファ(サイズ：512 バイト)を持っています。そのため、CX2500 はポートオープン後、メッセージを対向機から受信するとその内部バッファにメッセージを保存します。ただし、513 バイト以上のデータは内部で保有できません。その際は受信バッファオーバーフローエラーとなり SysComRead 関数を呼び出しても受信できなくなります。よって、ユーザーはバッファオーバーフローになる前に適度に SysComRead 関数を呼び出してバッファからメッセージを取り出して下さい。

受信バッファオーバーフローエラー含め、受信できなくなった場合、ポートクローズ→オープンをおこなって再度受信を試みるようにして下さい。

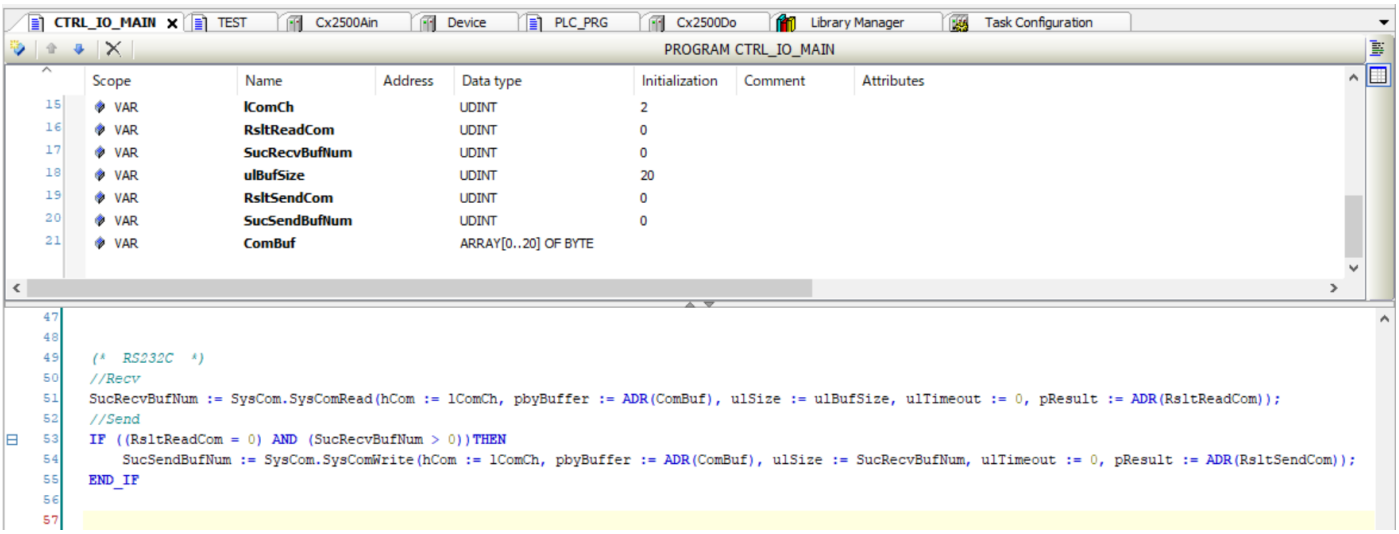


Figure 102 RS232C 送受信呼出例(ST 言語)

7.11.7. 送信

メッセージを送信(SysComWrite 関数の呼び出し)は受信バッファオーバーフローや通信エラーが発生していない場合におこなうことができます。エラー等で送信できなかった場合、ポートクローズ→オープンをおこなってから再送信を試みるようにして下さい。送信関数の呼び出し例は Figure 102 を参照して下さい。

7.12. CAN

CAN 通信を利用するには、下記いずれかを利用する必要があります。ユーザーアプリケーションで使用する通信プロトコルに合わせ選択し使用して下さい。本節では、CANBus ライブラリを使用する場合について解説しています。フィールドバスを使用する際は、本節ではなく 8 章を参照して下さい。

また、CANBus ライブラリ、各種フィールドバスについては、CODESYS オンラインヘルプにも記載があります。そちらも合わせて参照して下さい。

Table 46 CAN 利用手段

<table>
<tr><th>手段</th><th>摘要</th></tr>
<tr><td>フィールドバス
(8 章参照)</td><td>・ユーザーアプリケーションの通信プロトコルが J1939 や CANopen でおこなう場合に使用する。
・ユーザーは初期設定や送受信で関数を呼び出す必要が無い。各種設定は IO ドライバのように専用画面で簡単に設定できる。
・各プロトコル用のファイル(下記)が用意できれば、それを読み込むだけで送受信メッセージが簡単に定義・設定できる。
<table>
<tr><th>プロトコル名</th><th>ファイル名</th></tr>
<tr><td>J1939</td><td>DBC(CAN データベース)ファイル※14</td></tr>
<tr><td>CANopen</td><td>EDS ファイル※15</td></tr>
</table>
</td></tr>
<tr><td>CANBus
ライブラリ</td><td>・CANBus ライブラリの関数をユーザーが手動で呼び出して CAN 通信制御をおこなう。
・機能を使用するには、ライブラリ CANBus(CAA CanL2)が必要。使用できる関数は Table 48 参照。
・ユーザーアプリケーションの通信プロトコルが J1939 や CANopen でない場合に使用する。
・制御の自由度が高い分、ユーザーアプリケーションで初期設定を含め緻密な制御が求められる。</td></tr>
</table>

※14 DBC ファイルは CAN メッセージフレームの各データに識別名称などを結び付けることによって、生の CAN データ値をユーザーが読みやすくすることができるものです。DBC ファイルは Vector 社製 CANalyzer(CANdb++)などの生成できるツールを入手して作成する必要があります。

※15 EDS(Electronic Data Sheet)ファイルは CANopen などのフィールドネットワークで接続したいデバイスの接続情報(メッセージ構成・IO 情報他)を含んだファイルのことです。こちらも CX2500 と通信させるデバイスの EDS ファイルをデバイス生産会社から入手するか、CANalyzer など EDS ファイルを作成できるツールを用いて独自に作成する必要があります。

Table 47 CAN 設定可能ボードレート

チャネル番号	ボードレート
0～2	125kbps、250kbps、500kbps、1Mbps
3、4	125kbps、250kbps

Table 48 CAN(CAA CanL2 ライブラリ) 関数一覧

機能区分	関数名	摘要
ドライバ関連	DriverOpenH	ヒープメモリを割り当てたドライバを作成し、CAN ネットワークをオープンする。
	DriverClose	ドライバ(ネットワーク)を削除する。
レシーバー関連	CreateIdAreaReceiver	エリア(CAN-ID 範囲指定)レシーバー(ハンドラ)を作成する。
	RegisterIdArea	エリアレシーバーに受信できる CAN-ID を追加する。
	UnregisterIdArea	エリアレシーバーから受信できる CAN-ID の条件を削除する。
	CreateMaskReceiver	マスクレシーバー(ハンドラ)を作成する。
	CreateSingleIdReceiver	特定 CAN-ID のみ受信するレシーバー(ハンドラ)を作成する。
	DeleteReceiver	レシーバーを削除する。
メッセージハンドラ関連	CreateMessage	メッセージハンドラを作成する。
	CloneMessage	指定のメッセージを複製する。
	FreeMessage	指定のメッセージハンドラを削除する。
メッセージ取得	Read	受信したメッセージを取得する。
メッセージ送信	Write	メッセージを送信する。
ステータス取得関連	GetDiagnosis	バスの診断(チャネルステータス)情報を取得する。
	GetBaudrate	ボードレートを取得する。
	GetBusState	バスステータスを取得する。
	GetLostCounter	ロストメッセージの総数を取得する。
	GetReceiveCounter	受信したメッセージの総数を取得する。
	GetReceiveErrorCounter	受信エラーカウンタ値を取得する。
	GetReceivePoolSize	受信待ちのバッファ数を取得する。
	GetReceiveQueueLength	受信済みのメッセージ数を取得する。
	GetTransmitCounter	送信済みメッセージの総数を取得する。
	GetTransmitErrorCounter	送信エラーカウンタ値を取得する。
	GetTransmitPoolSize	送信設定待ちバッファ数を取得する。
	GetTransmitQueueLength	送信完了待ちメッセージ数を取得する。
	GetMessageDataPointer	指定メッセージのデータポインタを取得する。
	GetMessageId	指定メッセージの CAN-ID を取得する。
	GetMessageLength	指定メッセージのデータ長を取得する。
	GetMsgCount	受信バッファに残っているメッセージ数を取得する。
	GetNetId	指定したメッセージのネットワーク ID(チャネル番号)を取得する。
	LostMessage	指定 ID のロストメッセージ数を取得する。
	IsSendingActive	指定チャネルの送信バッファが Empty(送信可能状態)であるか判定する。
	Is29BitIdMessage	指定のメッセージが拡張 ID のメッセージかどうか判定する。

	IsRTRMessage	指定メッセージがリモートフレームかどうか判定する。
	IsTransmitMessages	前回送信設定したメッセージの送信が完了したか判定する。
非対応	DriverOpenP	非対応
	DriverGetSize	
	GetTimeStamp	
	GetBusload	
	GetBusAlarm	
	ResetBusAlarm	
	DisableSyncService	
	EnableSyncService	
	GetCiAState	
	SetCiAState	

7.12.1. 列挙型

CAN で使用する CAA CANL2I ライブラリで定義されている列挙型は下記の通りです。

型名	BUSSTATE		
摘要	CAN バスのステータスを表す。		
列挙子	名前	値	説明
	UNKNOWN	0	未定義(ドライバ非オープン)
	ERR_FREE	1	エラー無し
	ACTIVE	2	エラーアクティブ
	WARNING	3	エラーワーニング
	PASSIVE	4	エラーパッシブ
	BUSOFF	5	バスオフ

型名	ERROR		
摘要	CAA CANL2I ライブラリ関数の返り値を表す。		
列挙子	名前	値	説明
	NO_ERROR	0	エラー無し
	NO_29BIT_ID	10202	29BIT-ID(拡張 ID)サポート非対応
	WRONG_BAUDRATE	10203	ボードレート非対応
	NO_MEMORY	10204	内部メモリの空き無し
	INVALID_NETID	10205	ネットワーク ID(CAN チャネル番号)が不正値
	INVALID_PRIORITY	10206	優先度が不正値
	INVALID_DRIVER_HANDLE	10207	ドライバハンドラが不正値
	INVALID_MESSAGE_HANDLE	10208	メッセージハンドラが不正値
	INVALID_ID_HANDLE	10209	ID ハンドラが不正値
	NO_DRIVER	10210	CAN ドライバが利用不可
	SENDING_ERROR	10211	送信エラー
	NO_SYNC_SERVICE	10212	SYNC(同期)サービス非対応
	NO_SYNC_PRODUCER	10213	SYNC プロデューサー非対応
	NO_SYNC_CONSUMER	10214	SYNC コンシューマー非対応
	NO_SYNC_EVENT	10215	SYNC イベント非対応
	NO_SYNC_WINDOW	10216	SYNC ウィンドウ非対応
	NO_SYNC_FOREWARNTIME	10217	SYNC 早期警告非対応
	WRONG_PARAM	10224	パラメータが不正値
	INVALID_HANDLE	10249	ハンドラが不正値

7.12.2. 構造体

CAN で使用する CAA CANL2I ライブラリで定義されている構造体は下記の通りです。

型名	DIAGNOSIS_INFO		
摘要	CAN ネットワーク 診断(チャネルステータス)情報を表す構造体		
メンバ	型	名前	説明
	UINT	uiBaudrate	ボードレート[kbps]
	USINT	usiBusload	バス負荷率[%](非対応)
	BOOL	xBusAlarm	バスアラーム(非対応)
	BUSSTATE	eBusState	バスステータス
	CAA.COUNT	ctTxCounter	送信バッファセット完了総数
	CAA.COUNT	ctTxErrorCounter	送信エラーカウンタ値
	CAA.COUNT	ctRxCounter	受信済みメッセージ総数
	CAA.COUNT	ctRxErrorCounter	受信エラーカウンタ値
	CAA.COUNT	ctLostCounter	メッセージロスト総数
	CAA.COUNT	ctFreeRxMessages	受信待ちバッファ数
	CAA.COUNT	ctMessagesRxQueue	受信済みメッセージ数
	CAA.COUNT	ctFreeTxMessages	送信設定待ちバッファ数
	CAA.COUNT	ctMessagesTxQueue	送信完了待ちメッセージ数

7.12.3. 関数

CAN で使用できる関数について、それぞれ下記に示します。ただし、本製品非対応のものは除きます。

関数名	DriverOpenH			
摘要	ヒープメモリを割り当てたドライバを作成し、CAN ネットワークをオープンする。			
引数 (INPUT)	型	名前	説明	
	USINT	usiNetId	ネットワーク ID(チャネル番号:0～4)	
	UINT	uiBaudrate	ボードレート[kbps]	
	BOOL	xSupport29Bits	拡張 ID をサポートするか TRUE :サポートする FALSE :サポートしない	
	CAA.COUNT	ctMessages	ドライバに割り当てる送信バッファ数(最大 10。それ以上は 10 で制限される)	
	POINTER TO ERROR	peError	NO_ERROR(0)	:オープン成功(エラー無し)
			WRONG_BAUDRATE(10203)	:引数のボードレートが不正値
			NO_MEMORY(10204)	:内部メモリ空き無し
			INVALID_NETID(10205)	:引数のネットワーク ID が不正値
			NO_DRIVER(10210)	:ネットワークオープン失敗
			INVALID_HANDLE(10249)	:ネットワークオープン失敗
			MBM_NOMORE_MEMORY(30103)	:内部メモリ空き無し
引数 (IN_OUT)	無し			
返り値 (OUTPUT)	型	名前	説明	
	CAA.HANDLE	DriverOpenH	作成されたドライバハンドラ	

関数名	DriverClose			
摘要	作成したドライバ(ネットワーク)を削除する。			
引数 (INPUT)	型	名前	説明	
	CAA.HANDLE	hDriver	ドライバハンドラ	
引数 (IN_OUT)	無し			
返り値 (OUTPUT)	型	名前	説明	
	ERROR	DriverClose	NO_ERROR(0)	:エラー無し
			INVALID_HANDLE(10249)	:引数のハンドラが不正値

関数名	CreateIdAreaReceiver			
摘要	エリア(CAN-ID 範囲指定)レシーバー(ハンドラ)を作成する。			
引数 (INPUT)	型	名前	説明	
	CAA.HANDLE	hDriver	ドライバハンドラ	
	BOOL	xAlwaysNewest	**非対応(FALSE を設定すること)**	
	CB.EVENT	eEvent	**非対応(0 を設定すること)**	
	BOOL	xEnableSyncWindow	**非対応(FALSE を設定すること)**	
	POINTER TO ERROR	peError	NO_ERROR(0)	:エラー無し
			NO_MEMORY(10204)	:内部メモリ空き無し又は引数 xAlwaysNewest が不正値
			INVALID_DRIVER_HANDLE(10207)	:引数のハンドラが不正値
			NO_DRIVER(10210)	:引数のドライバが不正値
			WRONG_PARAM(10224)	:引数 eEvent が不正値
			MBM_NOMORE_MEMORY(30103)	:内部メモリ空き無し
引数 (IN_OUT)	無し			
返り値 (OUTPUT)	型	名前	説明	
	CAA.HANDLE	CreateIdAreaReceiver	作成されたレシーバーハンドラ	
備考	エリア指定のレシーバーは、拡張 ID が非対応であることに注意。			

関数名	RegisterIdArea		
摘要	エリア指定のレシーバーに受信できる CAN-ID を追加する。		
引数 (INPUT)	型	名前	説明
	CAA.HANDLE	hReceiverId	レシーバーハンドラ
	CL2I.COBID	cobIdStart	レシーバーに登録したい CAN-ID の最小値
	CL2I.COBID	cobIdEnd	レシーバーに登録したい CAN-ID の最大値
	BOOL	xRTRValue	リモートフレームサポート TRUE :リモートフレームを受信する FALSE :通常のフレームを受信する
	BOOL	xRTRMask	リモートフレームマスク TRUE :受信時リモートフレームかどうか検証する FALSE :受信時リモートフレームの検証をしない
	BOOL	X29BitIdValue	**FALSE を設定する**
	BOOL	X29BitIdMask	**TRUE を設定する**
	BOOL	xTransmitValue	TRUE :CX2500 が送信したメッセージ FALSE :CX2500 が送信したメッセージではない
	BOOL	xTransmitMask	TRUE :このレシーバーで受信したメッセージが、CX2500 が送信したメッセージであるか判定する FALSE :このレシーバーで受信したメッセージが、CX2500 が送信したメッセージであるか判定しない
引数 (IN_OUT)	無し		

返り値 (OUTPUT)	型	名前	説明	
	ERROR	RegisterIdArea	NO_ERROR(0)	:エラー無し
			NO_29BIT_ID(10202)	:引数 x29BitIdValue・x29BitIdMask が上記設定値以外
			INVALID_ID_HANDLE(10209)	:引数のハンドラが不正値
			WRONG_PARAM(10224)	:引数が不正値 (cobIdStart > cobIdEnd or ハンドラ不正)

関数名	UnregisterIdArea		
摘要	エリア指定のレシーバーから受信できる CAN-ID の条件を削除する。		
引数 (INPUT)	型	名前	説明
	CAA.HANDLE	hReceiverId	レシーバーハンドラ
	CL2I.COBID	cobIdStart	レシーバーから削除したい CAN-ID の最小値
	CL2I.COBID	cobIdEnd	レシーバーから削除したい CAN-ID の最大値
	BOOL	xRTRValue	リモートフレームサポート TRUE :リモートフレームを受信する FALSE :通常のフレームを受信する
	BOOL	xRTRMask	リモートフレームマスク TRUE :受信時リモートフレームかどうか検証する FALSE :受信時リモートフレームの検証をしない
	BOOL	X29BitIdValue	**FALSE を設定する**
	BOOL	X29BitIdMask	**TRUE を設定する**
	BOOL	xTransmitValue	TRUE :CX2500 が送信したメッセージ FALSE :CX2500 が送信したメッセージではない
	BOOL	xTransmitMask	TRUE :このレシーバーで受信したメッセージが、CX2500 が送信したメッセージであるか判定する FALSE :このレシーバーで受信したメッセージが、CX2500 が送信したメッセージであるか判定しない
引数 (IN_OUT)	無し		
返り値 (OUTPUT)	型	名前	説明
	ERROR	UnregisterIdArea	NO_ERROR(0) :エラー無し
			INVALID_ID_HANDLE(10209) :引数のハンドラが不正値
			WRONG_PARAM(10224) :引数が不正値 (cobIdStart > cobIdEnd)

関数名	CreateMaskReceiver		
摘要	マスクレシーバー(ハンドラ)を作成する。		
引数 (INPUT)	型	名前	説明
	CAA.HANDLE	hDriver	ドライバハンドラ
	CL2I.COBID	cobIdValue	CAN-ID
	CL2I.COBID	cobIdMask	CAN-ID マスク
	BOOL	xRTRValue	リモートフレームサポート TRUE :リモートフレームを受信する FALSE :通常のフレームを受信する
	BOOL	xRTRMask	リモートフレームマスク TRUE :受信時リモートフレームかどうか検証する FALSE :受信時リモートフレームの検証をしない
	BOOL	X29BitIdValue	拡張 ID サポート TRUE :拡張 ID を受信する FALSE :標準 ID を受信する
	BOOL	X29BitIdMask	拡張 ID マスク TRUE :受信時拡張 ID かどうか検証する FALSE :受信時拡張 ID の検証をしない
	BOOL	xTransmitValue	TRUE :CX2500 が送信したメッセージ FALSE :CX2500 が送信したメッセージではない
	BOOL	xTranmitMask	TRUE :このレシーバーで受信したメッセージが、CX2500 が送信したメッセージであるか判定する FALSE :このレシーバーで受信したメッセージが、CX2500 が送信したメッセージであるか判定しない
	BOOL	xAlwaysNewest	受信メッセージの取得順序 TRUE :最新の受信メッセージから取得する。 FALSE :最も古い受信メッセージから取得する。
	CB.EVENT	eEvent	**非対応(0 を設定すること)**
	BOOL	xEnableSyncWindow	**非対応(FALSE を設定すること)**
	POINTER TO ERROR	peError	NO_ERROR(0) :エラー無し
			NO_MEMORY(10204) :内部メモリ空き無し
			INVALID_DRIVER_HANDLE(10207) :引数のハンドラが不正値
			NO_DRIVER(10210) :引数のハンドラが不正値
			WRONG_PARAM(10224) :引数 eEvent が不正値
			MBM_NOMORE_MEMORY(30103) :内部メモリ空き無し

返り値 (OUTPUT)	型	名前	説明
	CAA.HANDLE	CreateMaskReceiver	作成されたレシーバーハンドラ

関数名	CreateSingleIdReceiver		
摘要	特定 CAN-ID のみ受信するレシーバー(ハンドラ)を作成する。		
引数 (INPUT)	型	名前	説明
	CAA.HANDLE	hDriver	ドライバハンドラ
	CL2I.COBID	cobId	CAN-ID
	BOOL	xRTR	リモートフレームサポート TRUE :リモートフレームを受信する FALSE :通常のフレームを受信する
	BOOL	X29BitId	拡張 ID サポート TRUE :拡張 ID を受信する FALSE :標準 ID を受信する
	BOOL	xTransmit	TRUE :CX2500 が送信したメッセージをこのレシーバーで受信する FALSE :CX2500 が送信したメッセージはこのレシーバーで受信しない
	BOOL	xAlwaysNewest	受信メッセージの取得順序 TRUE :最新の受信メッセージから取得する。 FALSE :最も古い受信メッセージから取得する。
	CB.EVENT	eEvent	**非対応(0 を設定すること)**
	BOOL	xEnableSyncWindow	**非対応(FALSE を設定すること)**
	POINTER TO ERROR	peError	NO_ERROR(0) :エラー無し
			NO_MEMORY(10204) :内部メモリ空き無し
			INVALID_DRIVER_HANDLE(10207) :引数のハンドラが不正値
			NO_DRIVER(10210) :引数のハンドラが不正値
			WRONG_PARAM(10224) :引数 eEvent が不正値
			MBM_NOMORE_MEMORY(30103) :内部メモリ空き無し
引数 (IN_OUT)	無し		

返り値 (OUTPUT)	型	名前	説明
	CAA.HANDLE	CreateSingleIdReceiver	作成されたレシーバーハンドラ

関数名	DeleteReceiver		
摘要	レシーバーを削除する。		
引数 (INPUT)	型	名前	説明
	CAA.HANDLE	hReceiverId	レシーバーハンドラ
引数 (IN_OUT)	無し		

返り値 (OUTPUT)	型	名前	説明	
	ERROR	DeleteReceiver	NO_ERROR(0)	:エラー無し
			INVALID_ID_HANDLE(10209)	:引数のハンドラが不正値
			NO_DRIVER(10210)	:引数のハンドラが不正値

関数名	CreateMessage			
摘要	メッセージハンドラを作成する。			
引数 (INPUT)	型	名前	説明	
	CAA.HANDLE	hDriver	ドライバハンドラ	
	CL2I.COBID	cobID	CAN-ID	
	USINT	usiLength	メッセージのデータ長	
	BOOL	xRTR	リモートフレームのサポート	
			TRUE	:リモートフレーム
			FALSE	:リモートフレームではない
	BOOL	X29BitID	拡張 ID のサポート	
			TRUE	:拡張 ID
			FALSE	:標準 ID
	POINTER TO ERROR	peError	NO_ERROR(0)	:エラー無し
			INVALID_DRIVER_HANDLE(10207)	:引数のドライバハンドラが不正値
引数 (IN_OUT)	無し			

返り値 (OUTPUT)	型	名前	説明
	CAA.HANDLE	CreateMessage	作成されたメッセージハンドラ

関数名	CloneMessage		
摘要	指定のメッセージを複製する。		
引数 (INPUT)	型	名前	説明
	CAA.HANDLE	hMessage	メッセージハンドラ
	POINTER TO ERROR	peError	NO_ERROR(0) :エラー無し
			INVALID_MESSAGE_HANDLE(10208) :引数のハンドラが不正値
引数 (IN_OUT)	無し		
返り値 (OUTPUT)	型	名前	説明
	CAA.HANDLE	CloneMessage	複製されたメッセージハンドラ

関数名	FreeMessage		
摘要	指定のメッセージハンドラを削除する。		
引数 (INPUT)	型	名前	説明
	CAA.HANDLE	hMessage	メッセージハンドラ
引数 (IN_OUT)	無し		
返り値 (OUTPUT)	型	名前	説明
	ERROR	FreeMessage	NO_ERROR(0) :エラー無し
			INVALID_MESSAGE_HANDLE(10208) :引数のハンドラが不正値

関数名	Read		
摘要	受信したメッセージを取得する。		
引数 (INPUT)	型	名前	説明
	CAA.HANDLE	hReceiverId	レシーバーハンドラ
	POINTER TO CAA.COUNT	pctMsgLeft	Read 関数呼出し後、受信バッファに残っている受信済みのメッセージ数
	POINTER TO ERROR	peError	NO_ERROR(0) :エラー無し
			INVALID_NETID(10205) :引数のハンドラが不正値
			INVALID_ID_HANDLE(10209) :引数のハンドラが不正値
			NO_DRIVER(10210) :引数のハンドラが不正値
引数 (IN_OUT)	無し		
返り値 (OUTPUT)	型	名前	説明
	CAA.HANDLE	Read	取得した受信メッセージのハンドラ
備考	Read 後、受信データを保存したら、FreeMessage 関数を呼び出し、リソースを解放すること。		

関数名	Write		
摘要	メッセージを送信する。		
引数 (INPUT)	型	名前	説明
	CAA.HANDLE	hDriver	ドライバハンドラ
	CAA.HANDLE	hMessage	メッセージハンドラ
	USINT	usPriority	送信優先度(1～8、数字が小さいほど優先度高)
	BOOL	xEnableSyncWindow	**非対応(FALSE を設定すること)**
引数 (IN_OUT)	無し		
返り値 (OUTPUT)	型	名前	説明
	ERROR	Write	NO_ERROR :エラー無し
			INVALID_NETID(10205) :引数のドライバハンドラが不正値
			INVALID_PRIORITY(10206) :引数の優先度が不正値
			INVALID_DRIVER_HANDLE(10207) :引数のドライバハンドラが不正値
			INVALID_MESSAGE_HANDLE(10208) :引数のメッセージハンドラが不正値
			NO_DRIVER(10210) :引数のドライバハンドラが不正値

関数名	GetDiagnosis		
摘要	バスの診断(チャネルステータス)情報を取得する。		
引数 (INPUT)	型	名前	説明
	CAA.HANDLE	hDriver	ドライバハンドラ
	POINTER TO DIAGNOSIS_INFO	pDiagnosisInfo	診断情報
引数 (IN_OUT)	無し		

返り値 (OUTPUT)	型	名前	説明
	ERROR	GetDiagnosis	NO_ERROR(0) :エラー無し

備考	引数 hDriver が不正値(例:ドライバハンドラではない値)の場合でも、返り値 GetDiagnosis には NO_ERROR が入る。また、pDiagnosisInfo にも不定値がセットされてしまうことに注意。これは CX2500 ではなく CODESYS 自体の動作仕様である。

関数名	GetBaudrate		
摘要	ボードレートを取得する。		
引数 (INPUT)	型	名前	説明
	CAA.HANDLE	hDriver	ドライバハンドラ
	POINTER TO ERROR	peError	NO_ERROR(0) :エラー無し
			INVALID_DRIVER_HANDLE(10207) :引数のハンドラが不正値
			NO_DRIVER(10210) :引数のハンドラが不正値
引数 (IN_OUT)	無し		
返り値 (OUTPUT)	型	名前	説明
	UINT	GetBaudrate	ボードレート[kbps]

関数名	GetBusState		
摘要	バスステータスを取得する。		
引数 (INPUT)	型	名前	説明
	CAA.HANDLE	hDriver	ドライバハンドラ
	POINTER TO ERROR	peError	NO_ERROR(0) :エラー無し
			INVALID_DRIVER_HANDLE(10207) :引数のハンドラが不正値
			NO_DRIVER(10210) :引数のハンドラが不正値
引数 (IN_OUT)	無し		
返り値 (OUTPUT)	型	名前	説明
	BUSSTATE	GetBusState	バスステータス

関数名	GetLostCounter		
摘要	ロストメッセージの総数を取得する。		
引数 (INPUT)	型	名前	説明
	CAA.HANDLE	hDriver	ドライバハンドラ
	POINTER TO ERROR	peError	NO_ERROR(0) :エラー無し
			INVALID_DRIVER_HANDLE(10207) :引数のハンドラが不正値
			NO_DRIVER(10210) :引数のハンドラが不正値
引数 (IN_OUT)	無し		
返り値 (OUTPUT)	型	名前	説明
	CAA.COUNT	GetLostCounter	ロストメッセージ総数

関数名	GetReceiveCounter		
摘要	受信したメッセージの総数を取得する。		
引数 (INPUT)	型	名前	説明
	CAA.HANDLE	hDriver	ドライバハンドラ
	POINTER TO ERROR	peError	NO_ERROR(0) :エラー無し
			INVALID_DRIVER_HANDLE(10207) :引数のハンドラが不正値
			NO_DRIVER(10210) :引数のハンドラが不正値
引数 (IN_OUT)	無し		
返り値 (OUTPUT)	型	名前	説明
	CAA.COUNT	GetReceiveCounter	受信メッセージ総数

関数名	GetReceiveErrorCounter		
摘要	受信エラーカウンタ値を取得する。		
引数 (INPUT)	型	名前	説明
	CAA.HANDLE	hDriver	ドライバハンドラ
	POINTER TO ERROR	peError	NO_ERROR(0) :エラー無し
			INVALID_DRIVER_HANDLE(10207) :引数のハンドラが不正値
			NO_DRIVER(10210) :引数のハンドラが不正値
引数 (IN_OUT)	無し		
返り値 (OUTPUT)	型	名前	説明
	CAA.COUNT	GetReceiveErrorCounter	受信エラーカウンタ値

関数名	GetReceivePoolSize		
摘要	受信待ちのバッファ数を取得する。		
引数 (INPUT)	型	名前	説明
	CAA.HANDLE	hDriver	ドライバハンドラ
	POINTER TO ERROR	peError	NO_ERROR(0) :エラー無し
			NO_DRIVER(10210) :引数のハンドラが不正値
			INVALID_HANDLE(10249) :引数のハンドラが不正値
引数 (IN_OUT)	無し		
返り値 (OUTPUT)	型	名前	説明
	CAA.COUNT	GetReceivePoolSize	受信待ちバッファ数

関数名	GetReceiveQueueLength		
摘要	受信済みのメッセージ数を取得する。		
引数 (INPUT)	型	名前	説明
	CAA.HANDLE	hDriver	ドライバハンドラ
	POINTER TO ERROR	peError	NO_ERROR(0) :エラー無し
			INVALID_NETID(10205) :引数のハンドラが不正値
			NO_DRIVER(10210) :引数のハンドラが不正値
			INVALID_HANDLE(10249) :引数のハンドラが不正値
引数 (IN_OUT)	無し		
返り値 (OUTPUT)	型	名前	説明
	CAA.COUNT	GetReceiveQueueLength	受信済みメッセージ数

関数名	GetTransmitCounter		
摘要	送信済みメッセージの総数を取得する。		
引数 (INPUT)	型	名前	説明
	CAA.HANDLE	hDriver	ドライバハンドラ
	POINTER TO ERROR	peError	NO_ERROR(0) :エラー無し
			INVALID_DRIVER_HANDLE(10207) :引数のハンドラが不正値
			NO_DRIVER(10210) :引数のハンドラが不正値
引数 (IN_OUT)	無し		
返り値 (OUTPUT)	型	名前	説明
	CAA.COUNT	GetTransmitCounter	送信済みメッセージ総数

関数名	GetTransmitErrorCounter		
摘要	送信エラーカウンタ値を取得する。		
引数 (INPUT)	型	名前	説明
	CAA.HANDLE	hDriver	ドライバハンドラ
	POINTER TO ERROR	peError	NO_ERROR(0) :エラー無し
			INVALID_DRIVER_HANDLE(10207) :引数のハンドラが不正値
			NO_DRIVER(10210) :引数のハンドラが不正値
引数 (IN_OUT)	無し		
返り値 (OUTPUT)	型	名前	説明
	CAA.COUNT	GetTransmitErrorCounter	送信エラーカウンタ値

関数名	GetTransmitPoolSize		
摘要	送信設定待ちバッファ数を取得する。		
引数 (INPUT)	型	名前	説明
	CAA.HANDLE	hDriver	ドライバハンドラ
	POINTER TO ERROR	peError	NO_ERROR(0) :エラー無し
			NO_DRIVER(10210) :引数のハンドラが不正値
			INVALID_HANDLE(10249) :引数のハンドラが不正値
引数 (IN_OUT)	無し		
返り値 (OUTPUT)	型	名前	説明
	CAA.COUNT	GetTransmitPoolSize	送信設定待ちバッファ数

関数名	GetTransmitQueueLength		
摘要	送信完了待ちメッセージ数を取得する。		
引数 (INPUT)	型	名前	説明
	CAA.HANDLE	hDriver	ドライバハンドラ
	POINTER TO ERROR	peError	NO_ERROR(0) :エラー無し
			INVALID_NETID(10205) :引数のハンドラが不正値
			NO_DRIVER(10210) :引数のハンドラが不正値
			INVALID_HANDLE(10249) :引数のハンドラが不正値
引数 (IN_OUT)	無し		
返り値 (OUTPUT)	型	名前	説明
	CAA.COUNT	GetTransmitQueue Length	送信完了待ちバッファ数

関数名	GetMessageDataPointer		
摘要	指定メッセージのデータポインタを取得する。		
引数 (INPUT)	型	名前	説明
	CAA.HANDLE	hMessage	メッセージハンドラ
	POINTER TO ERROR	peError	NO_ERROR(0) :エラー無し
			INVALID_MESSAGE_HANDLE(10208) :引数のハンドラが不正値
引数 (IN_OUT)	無し		
返り値 (OUTPUT)	型	名前	説明
	POINTER TO CL2I.DATA	GetMessageDataPointer	データポインタアドレス

関数名	GetMessageId		
摘要	指定メッセージの CAN-ID を取得する。		
引数 (INPUT)	型	名前	説明
	CAA.HANDLE	hMessage	メッセージハンドラ
	POINTER TO ERROR	peError	NO_ERROR(0) :エラー無し
			INVALID_MESSAGE_HANDLE(10208) :引数のハンドラが不正値
引数 (IN_OUT)	無し		
返り値 (OUTPUT)	型	名前	説明
	CL2I.COBID	GetMessageId	CAN-ID

関数名	GetMessageLength		
摘要	指定メッセージのデータ長を取得する。		
引数 (INPUT)	型	名前	説明
	CAA.HANDLE	hMessage	メッセージハンドラ
	POINTER TO ERROR	peError	NO_ERROR(0) :エラー無し
			INVALID_MESSAGE_HANDLE(10208) :引数のハンドラが不正値
引数 (IN_OUT)	無し		
返り値 (OUTPUT)	型	名前	説明
	USINT	GetMessageLength	データ長

関数名	GetMsgCount			
摘要	受信バッファに残っているメッセージ数を取得する。			
引数 (INPUT)	型	名前	説明	
	CAA.HANDLE	hReceiverId	レシーバーハンドラ	
	POINTER TO ERROR	peError	NO_ERROR(0)	:エラー無し
			INVALID_NETID(10205)	:引数のハンドラが不正値
			INVALID_ID_HANDLE(10209)	:引数のハンドラが不正値
			NO_DRIVER(10210)	:引数のハンドラが不正値
引数 (IN_OUT)	無し			
返り値 (OUTPUT)	型	名前	説明	
	CAA.COUNT	GetMsgCount	受信メッセージ残数	

関数名	GetNetId			
摘要	指定したメッセージのネットワーク ID(チャネル番号)を取得する。			
引数 (INPUT)	型	名前	説明	
	CAA.HANDLE	hMessage	メッセージハンドラ	
	POINTER TO ERROR	peError	NO_ERROR(0)	:エラー無し
			INVALID_MESSAGE_HANDLE(10208)	:引数のハンドラが不正値
引数 (IN_OUT)	無し			
返り値 (OUTPUT)	型	名前	説明	
	USINT	GetNetId	ネットワーク ID(チャネル番号:0～4)	

関数名	LostMessages		
摘要	指定 ID のロストメッセージ数を取得する。		
引数 (INPUT)	型	名前	説明
	CAA.HANDLE	hReceiverId	レシーバーハンドラ
	POINTER TO ERROR	peError	NO_ERROR(0) :エラー無し
			INVALID_ID_HANDLE(10209) :引数のハンドラが不正値
引数 (IN_OUT)	無し		
返り値 (OUTPUT)	型	名前	説明
	UINT	LostMessages	ロストメッセージ数

関数名	IsSendingActive		
摘要	指定チャネルの送信バッファが Empty(送信可能状態)であるか判定する。		
引数 (INPUT)	型	名前	説明
	CAA.HANDLE	hDriver	ドライバハンドラ
	POINTER TO ERROR	peError	NO_ERROR(0) :エラー無し
			INVALID_DRIVER_HANDLE(10207) :引数のハンドラが不正値
引数 (IN_OUT)	無し		
返り値 (OUTPUT)	型	名前	説明
	BOOL	IsSendingActive	TRUE :送信不可状態(エラーなど)
			FALSE :送信可能状態

関数名	Is29BitIdMessage			
摘要	指定のメッセージが拡張 ID のメッセージかどうか判定する。			
引数 (INPUT)	型	名前	説明	
	CAA.HANDLE	hMessage	判定したいメッセージハンドラ	
	POINTER TO ERROR	peError	NO_ERROR(0)	:エラー無し
			INVALID_MESSAGE_HANDLE (10208)	:引数のハンドラが不正値
引数 (IN_OUT)	無し			
返り値 (OUTPUT)	型	名前	説明	
	BOOL	Is29BitIdMessage	TRUE	:拡張 ID
			FALSE	:拡張 ID ではない

関数名	IsRTRMessage			
摘要	指定メッセージがリモートフレームかどうか判定する。			
引数 (INPUT)	型	名前	説明	
	CAA.HANDLE	hMessage	判定したいメッセージハンドラ	
	POINTER TO ERROR	peError	NO_ERROR(0)	:エラー無し
			INVALID_MESSAGE_HANDLE(10208)	:引数のハンドラが不正値
引数 (IN_OUT)	無し			
返り値 (OUTPUT)	型	名前	説明	
	BOOL	IsRTRMessage	TRUE	:リモートフレーム
			FALSE	:リモートフレームではない

関数名	IsTransmitMessage		
摘要	レシーバーで受信したメッセージが CX2500 から送信されたメッセージであるかを判定する。		
引数 (INPUT)	型	名前	説明
	CAA.HANDLE	hMessage	判定したいメッセージハンドラ
	POINTER TO ERROR	peError	NO_ERROR(0) :エラー無し
			INVALID_MESSAGE_HANDLE(10208) :引数のハンドラが不正値
引数 (IN_OUT)	無し		
返り値 (OUTPUT)	型	名前	説明
	BOOL	IsTransmitMessage	TRUE :CX2500 から送信されたメッセージである
			FALSE :CX2500 から送信されたメッセージではない
備考	CX2500 が送信したメッセージを受信するには、各レシーバー作成の際に ID 等と共に xTransmit 系引数(xTransmit・xTransmitValue・xTransmitMask)を TRUE にして作成する必要がある。		

7.12.4. 初期設定

CAN 通信をおこなう際、ユーザーは用途に応じて下記のハンドラを作成する必要があります。なお、ドライバハンドラについては他のどのハンドラよりも必ず先に作成しなければなりません。

各ハンドラの作成方法については 7.12.4.1 項以降に後述します。

Table 49 CAN 各種ハンドラ

名称	摘要
ドライバハンドラ	・作成すると、CX2500 の CAN チャネルの制御が可能になる。 ・これを作成しないと、レシーバーハンドラやメッセージハンドラの作成ができないことに注意。 ・使用するチャネルに対してそれぞれ 1 つ作成。
レシーバーハンドラ	・CX2500 が CAN メッセージを受信するためのレシーバーを指す。 ・作成後 CX2500 内部で CAN メッセージの受信が自動でおこなわれる。 ・1 つのドライバハンドラ(1 チャネル)に対して複数作成可能。
メッセージハンドラ	・CX2500 で CAN メッセージの送受信するために必要なメッセージ構造体。 ・1 つのドライバハンドラ(1 チャネル)に対して複数作成可能。

7.12.4.1. ドライバハンドラの作成

ユーザーアプリケーションで CAN を使用するには、CX2500 の使用したい CAN チャネル番号に対して 1 つドライバを作成する必要があります。ドライバ(ハンドラ)作成によって、CAN メッセージの送受信制御やバスステータス情報管理が可能になります。 ドライバハンドラは他のハンドラを作成する前に必ず作成して下さい。

Table 50 ドライバハンドラ 作成用関数一覧

関数名	摘要
DriverOpenH	CX2500 のヒープメモリを割り当て、CAN ドライバを作成する。
DriverOpenP	CX2500 の RAM メモリを割り当て、CAN ドライバを作成する。

```
(*  Initialization  *)
IF bCanInit = FALSE THEN

(*  Open Driver *)
hCanDriver := CL2.DriverOpenH(usiNetId := 1, uiBaudrate := 250, xSupport29Bits := TRUE, ctMessages := ctTxMsg, peError := ADR(eCanErr));

(*  Make Receiver   *)
IF eCanErr = ERROR.NO_ERROR THEN
    hReceiver := CL2.CreateMaskReceiver(
        hDriver:= hCanDriver,
        cobIdValue := 0,
        cobIdMask := 0,
        xRTRValue := 0,
        xRTRMask := 0,
        x29BitIdValue := FALSE,
        x29BitIdMask := TRUE,
        xTransmitValue := FALSE,
        xTransmitMask := TRUE,
        xAlwaysNewest := FALSE,
        eEvent := CB.EVENT.NO_EVENT,
        xEnableSyncWindow := FALSE,
        peError := ADR(eCanErr)
    );

    //Initialization Completed
    bCanInit := TRUE;
END_IF
```

Figure 103 ドライバハンドラおよびレシーバーハンドラ 作成・呼び出し例(ST 言語)

この例でのレシーバーハンドラは全ての標準フォーマット CAN-ID を受信する設定

7.12.4.2. レシーバーハンドラの作成

CAN メッセージを受信するには、ユーザーアプリケーションにてレシーバーを作成する必要が有ります。レシーバーには下記の種類があり、いずれかを作成した後に CX2500 内部でメッセージ受信が自動的におこなわれます。アプリケーションの用途に合わせレシーバー(ハンドラ)を選択・作成して下さい(呼び出し例は Figure 103)。なお、レシーバーは各チャネル複数作成可能です。ただし、各チャネル最大 16 個までです(フィールドバスも同様。もし定義してもメッセージを受信できない場合は送受信メッセージの種類数を調整して下さい)。レシーバー作成に関する注意は Table 51 を参照して下さい。

Table 51 CAN レシーバーの種類

名称	作成用関数名	摘要
ID エリア指定レシーバー	CreateIdAreaReceiver	・ユーザーが指定した範囲内の CAN-ID のみ受信をおこなうレシーバー。 ・CAN-ID の範囲指定はレシーバー作成後、RegisterIdArea 関数を呼び出すことでおこなうことができる。 ・受信できる CAN-ID の範囲は RegisterIdArea 関数の引数 cobIdStart～cobIdEnd となる。 ・ID 範囲内の CAN-ID1 つにつき作成可能レシーバー数をランタイム内部で 1 つ消費してしまうことに注意。レシーバーハンドラ自体は、範囲指定した CAN-ID が複数の場合でも、1 つしか作成されない。この 1 つのハンドラに範囲指定した CAN-ID を満たすメッセージが受信・格納される。 ・拡張 ID フォーマットは非対応。
マスクレシーバー	CreateMaskReceiver	・複数の CAN-ID を受信可能なレシーバー。 ・エリア指定レシーバーと異なり、範囲ではなくビットマスクで受信できる CAN-ID をフィルタする。例は次ページ参照。 ・x29BitIdMask を FALSE に設定してこのレシーバーを作成した場合、x29BitIdMask を TRUE で設定した場合に比べ、作成可能レシーバー数を 1 つ多く消費することに注意。ただし、レシーバーハンドラ自体は、いずれの場合でも 1 つしか作成されない。この 1 つのハンドラに Mask 条件をクリアした CAN-ID を満たすメッセージが受信・格納される。
シングル ID レシーバー	CreateSingleIdReceiver	・特定の 1 つの CAN-ID のみ受信をおこなうレシーバー。

(例)マスクレシーバー

CAN-ID の Value 及び Mask(CreateMaskReceiver 関数の引数)を下記の通り設定した場合、受信 ID1～3 の内、レシーバーが受信・受信バッファに保存できるのは検証ビットの Value と値が同じ受信 ID3 のみ。

項目	Bit												ID 値
	11	10	9	8	7	6	5	4	3	2	1	0	
cobIdValue	0	1	1	1	0	1	1	1	1	0	0	1	0x779
cobIdMask	0	0	0	1	0	0	1	0	0	0	1	1	0x123
受信検証 Bit	×	×	×	○	×	×	○	×	×	×	○	○	-
受信 ID1	0	1	1	0	1	1	0	1	1	0	1	0	0x6DA
受信 ID2	0	1	1	1	0	1	1	1	1	0	1	1	0x77B
受信 ID3	0	1	0	1	0	1	1	0	0	1	0	1	0x565

赤色：検証 NG、青色：検証 OK

7.12.4.3. メッセージハンドラの作成

CX2500 から CAN メッセージを送信する場合、メッセージ情報を格納するメッセージハンドラを事前に作成する必要が有ります。メッセージハンドラは CreateMessage 関数を呼び出すことで作成できます。作成したメッセージハンドラは Write 関数を呼び出してメッセージを送信するか、FreeMessage 関数を呼び出すことで解放されることに留意して下さい。

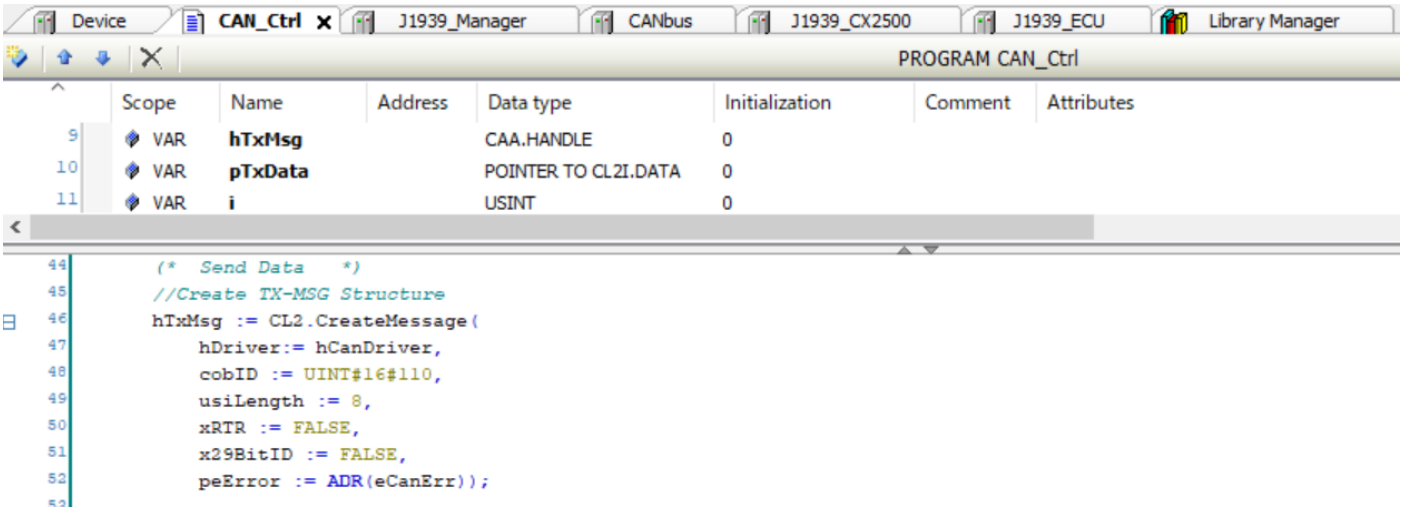


Figure 104 メッセージハンドラ 作成・呼出例(ST 言語)

7.12.5. 受信

CAN メッセージの受信は、レシーバーハンドラ作成後、CX2500 内部で自動的におこなわれます。受信したメッセージをユーザーアプリケーションで取り出すには、Read 関数を呼び出す必要が有ります。Read 関数を呼び出すと、受信メッセージハンドラが返り値として得られ、メッセージ内の情報をアプリケーションで使用できます。使用後は FreeMessage 関数を呼び出してメッセージハンドラ(リソース)を解放するようにして下さい。

なお、CX2500 内部で受信したメッセージをユーザーアプリケーションで取得しない状態が続くと、受信バッファが満杯になりそれ以上メッセージを受信・保存することができなくなる場合があります。定期的に受信メッセージの取得をおこなって下さい。

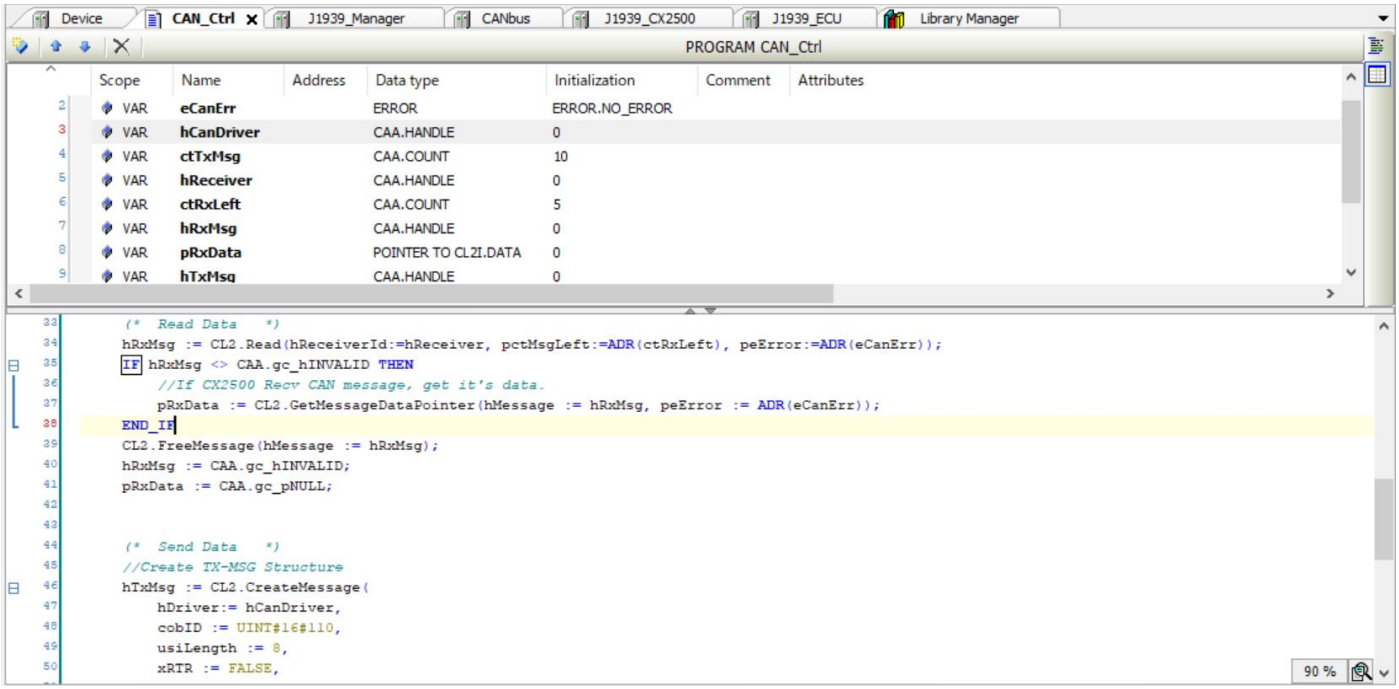


Figure 105 CAN 受信処理例(ST 言語)

7.12.6. 送信

CAN メッセージの送信は Write 関数を呼び出すことでおこなうことができます。なお、送信にはメッセージハンドラが必要になりますので、CreateMessage 関数を事前に呼び出しメッセージ情報を設定の上送信処理をおこなって下さい。

なお、CreateMessage 関数で作成したメッセージハンドラは Write 関数が正常に処理された場合(返り値が NO_ERROR の場合)、内部で自動的に FreeMessage 関数が呼び出されて解放されます。この場合、ユーザは FreeMessage を呼び出す必要はありません。Write 関数が失敗した場合(返り値が NO_ERROR 以外の場合)、ユーザ自身で必ず FreeMessage 関数を呼び出してメモリを解放して下さい。下図 64～67 行目がその処理例になります。

また、メッセージハンドラとして使用した変数は使用後、必ず下図(70、71 行目)のようにリセット処理をして下さい。リセットせずに続けて使用した場合、予期せぬ挙動になる場合があります。

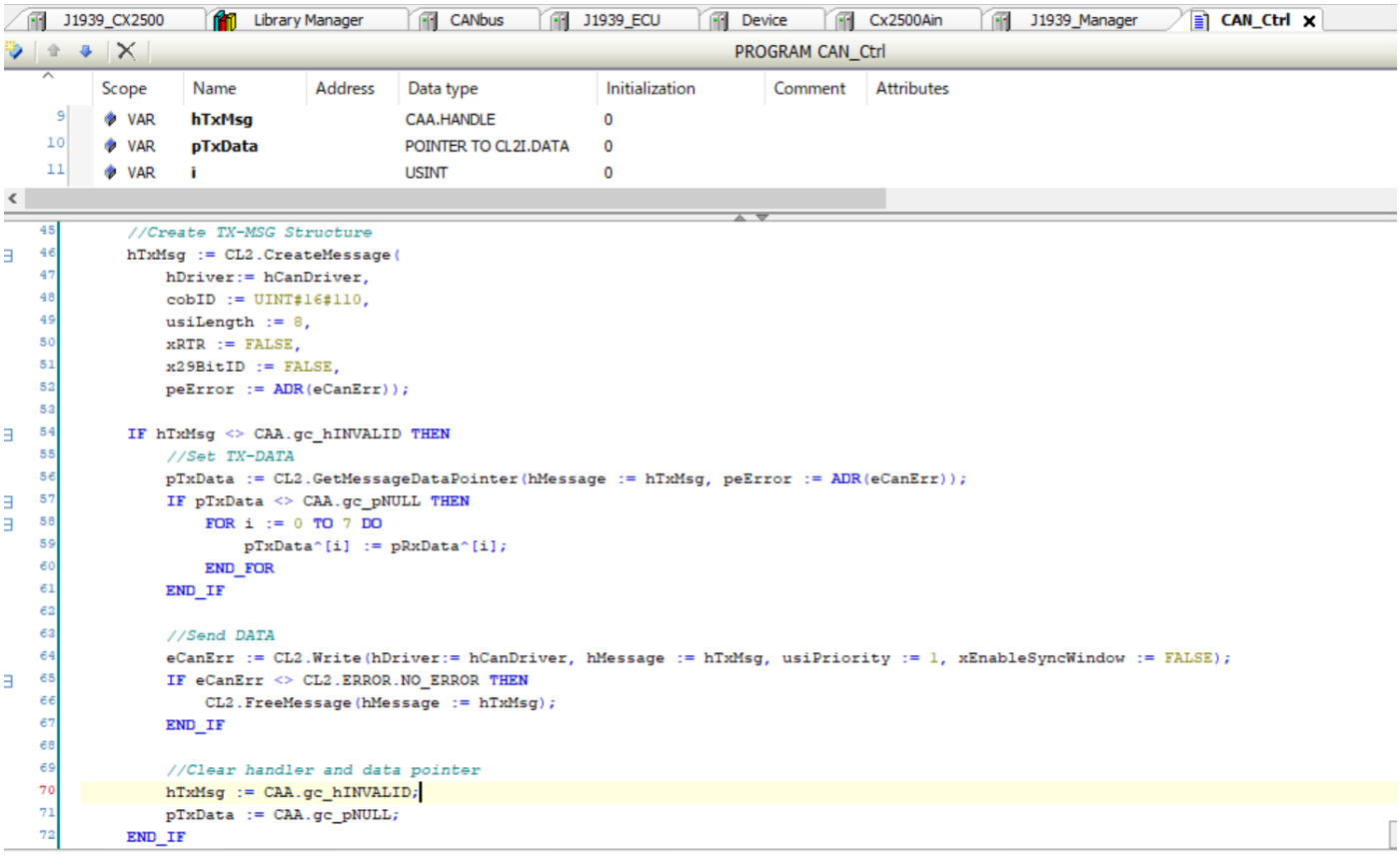


Figure 106 CAN 送信処理例(ST 言語)

7.13. タイマカウンタ

本機能は、ランタイム起動(CX2500 に電源投入)してからの経過時間を取得できます。この経過時間は、IDE でのユーザー操作によるアプリケーション停止中(ブレークポイント等)も時間が加算され続けることに留意して下さい。本機能の使用には、ライブラリ SysTime(SysTimeCore)が登録されている必要が有ります。使用できる関数は下記の通りです。

また、SysTime ライブラリについては CODESYS オンラインヘルプにも記載があります。そちらも合わせて参照して下さい。

Table 52 タイマカウンタ(SysTimeCore ライブラリ) 関数一覧

機能区分	関数名	摘要
経過時間取得	SysTimeGetMs	ランタイムが起動してからの経過時間[ms]を取得する。
	SysTimeGetUs	ランタイムが起動してからの経過時間[μs]を取得する。
非対応	SysTimeGetNs	非対応。

7.13.1. 型の別名定義

タイマカウンタで使用する SysTimeCore ライブラリで定義されている型は下記の通りです。

Table 53 SysTimeCore ライブラリ 型の別名定義一覧

名称	ベース型名	摘要
SYSTIME	ULINT	経過時間の取得などに使用される。

7.13.2. 関数

タイマカウンタで使用できる関数について、それぞれ下記に示します。ただし、本製品非対応のものは除きます。

関数名	SysTimeGetMs		
摘要	ランタイムが起動してからの経過時間[ms]を取得する。		
引数(INPUT)	無し		
引数(IN_OUT)	無し		
返り値(OUTPUT)	型	名前	説明
	UDINT	SysTimeGetMs	経過時間[ms]

関数名	SysTimeGetUs		
摘要	ランタイムが起動してからの経過時間[μs]を取得する。		
引数 (INPUT)	無し		
引数 (IN_OUT)	型	名前	説明
	SYSTIME	pUsTime	経過時間[μs]
返り値 (OUTPUT)	型	名前	説明
	RTS_IEC_RESULT	SysTimeGetUs	ERR_OK(0x0) :取得成功

7.14. RTC

ユーザーは CX2500 内蔵の RTC(リアルタイムクロック)の計時及び時刻を設定・取得することが可能です。機能を使用するには、当社独自ライブラリ CX2500RtcLibrary(Cx2500 Rtc Library)が登録されている必要が有ります。使用できる関数は下記の通りです。

Table 54 RTC(CX2500RtcLibrary ライブラリ) 関数一覧

機能区分	関数名	摘要
カレンダ設定	SetDate_Rtc	RTC へカレンダ(時刻)を設定する。
カレンダ取得	GetDate_Rtc	RTC からカレンダ(現在時刻)を取得する。

7.14.1. 列挙型

RTC で使用する CX2500RtcLibrary ライブラリで定義されている列挙型は下記の通りです。

型名	EN_RTC_DAY_OF_WEEK_IEC		
摘要	曜日を表す。		
列挙子	名前	値	説明
	DW_SUN	0	日曜日
	DW_MON	1	月曜日
	DW_TUE	2	火曜日
	DW_WED	3	水曜日
	DW_THU	4	木曜日
	DW_FRI	5	金曜日
	DW_SAT	6	土曜日

7.14.2. 構造体

RTC で使用する CX2500RtcLibrary ライブラリで定義されている構造体は下記の通りです。

型名	ST_RTC_TIME_IEC		
摘要	時刻をあらわす構造体。		
メンバ	型	名前	説明
	USINT(0..59)	ucSec	秒(0～59)
	USINT(0..59)	ucMin	分(0～59)
	USINT(0..23)	ucHour	時間(0～23)
	EN_RTC_DAY_OF_WEEK_IEC	ucWeek	曜日(0(Sunday)～7(Saturday))
	USINT(1..31)	ucDay	日(1～31)
	USINT(1..12)	ucMonth	月(1～12)
	USINT(0..99)	ucYear	年(例:2024 の場合は下二桁の 24 を設定)

7.14.3. 関数

RTC で使用できる関数について、それぞれ下記に示します。

関数名	SetDate_Rtc		
摘要	RTC から時刻を設定する。		
引数 (INPUT)	型	名前	説明
	ST_RTC_TIME_IEC	stRtcTime	設定したい時刻
引数 (IN_OUT)	無し		
返り値 (OUTPUT)	型	名前	説明
	RTS_IEC_RESULT	SetDate_Rtc	ERR_OK(0x0) :設定成功
			ERR_FAILED(0x1) :設定失敗(引数異常・RTC エラー)

関数名	GetDate_Rtc		
摘要	RTC から現在時刻を取得する。		
引数 (INPUT)	無し		
引数 (IN_OUT)	型	名前	説明
	ST_RTC_TIME_IEC	stRtcTime	取得した時刻
返り値 (OUTPUT)	型	名前	説明
	RTS_IEC_RESULT	GetDate_Rtc	ERR_OK(0x0) :取得成功
			ERR_FAILED(0x1) :取得失敗(RTC エラー)
備考	時刻が設定されていない、若しくは長期放電により時刻が消失した場合は取得できる値が不定値となることに注意。		

7.14.4. 初期状態とバックアップ時間

RTC には初期設定(時刻設定)がされていません。そのため、ユーザーはアプリケーション内で任意のタイミングにカレンダを設定する必要が有ります。

また、時刻を RTC に設定した後、CX2500 に長い間電源を入れないと、RTC へ設定した時刻が失われます。電源を落としてから時刻が失われるまでのバックアップ時間の目安については、CX2500 の機能仕様書を参照して下さい。

なお、初期状態・時刻が失われた状態で時刻を取得すると取得値は不定値になりますので留意して下さい。

8. フィールドバスについて

8.1. 概要

CX2500 は CAN フィールドバス(J1939・CANopen)を使用することができます。フィールドバスは、通信機能を有する機器間で情報をやり取りするために接続するバスシステムを指します。

CODESYS では、CAN ライブラリ(7.12 節)を使っても CAN 通信をおこなうことができます。ただ、フィールドバス機能を使うと、下記のような利点があります。

CANopen については、サンプルプロジェクトが CODESYS Store にて無料配布されていますので、ダウンロードして参考コードとして確認することができます。また、マニュアルに記載無き情報については CODESYS オンラインヘルプなどを参照して下さい。

【主な利点】

- ユーザーアプリケーションの通信プロトコルが J1939 や CANopen でおこなう場合に便利。
- ユーザーは初期設定や送受信で CAN ライブラリのように関数を呼び出す必要が無い。各種設定は IO ドライバのように専用画面で簡単に設定できる。
- 各プロトコル用のファイル(下記)が用意できれば、それを読み込むだけで同一バス内のデバイス向け送受信メッセージが簡単に定義・設定できる。
  - J1939 : DBC ファイル※14
  - CANopen : EDS ファイル※15

8.2. 共通設定

ここでは、プロトコル問わず、フィールドバスを使う際の共通設定について記します。フィールドバスを使用する際、機能ドライバのようにプロジェクトのデバイスにフィールドバスアイテムを紐づける必要があります。紐づける必要があるのは以下の通りです。手順については 8.2.1 項を参照して下さい。

Table 55 フィールドバス デバイスに紐づけるアイテム

名称	デフォルト名称		摘要
	J1939	CANopen	
バスマスタ	CANbus	CANbus	フィールドバスを使用するチャネルや CAN の基本的な通信設定・ステータスモニタができる。
プロトコルマネージャー	J1939_Manager	CANopen_Manager※16	各プロトコルの基本設定やデバイス用の DBC/EDS ファイルなどをインポート/エクスポートすることができる。
CAN デバイス	J1939_ECU	(各デバイスに依る)	バス内のノードとなるデバイス情報。デバイスがおこなう送受信メッセージの設定およびモニタをおこなうことができる。

※16 CAN デバイスがローカルデバイス(スレーブ)の場合は不要。その場合、CAN デバイスはバスマスタから直接紐づける。

8.2.1. フィールドバスの紐づけ

ここではフィールドバスセットの紐づけ手順を示します。例として J1939 でおこなっていますが、CANopen でも同様の手順をおこなうことで設定できます。

(1) まず、バスマスタの紐づけをおこないます。デバイスウィンドウの「Device」にカーソルを合わせ右クリックして下さい。

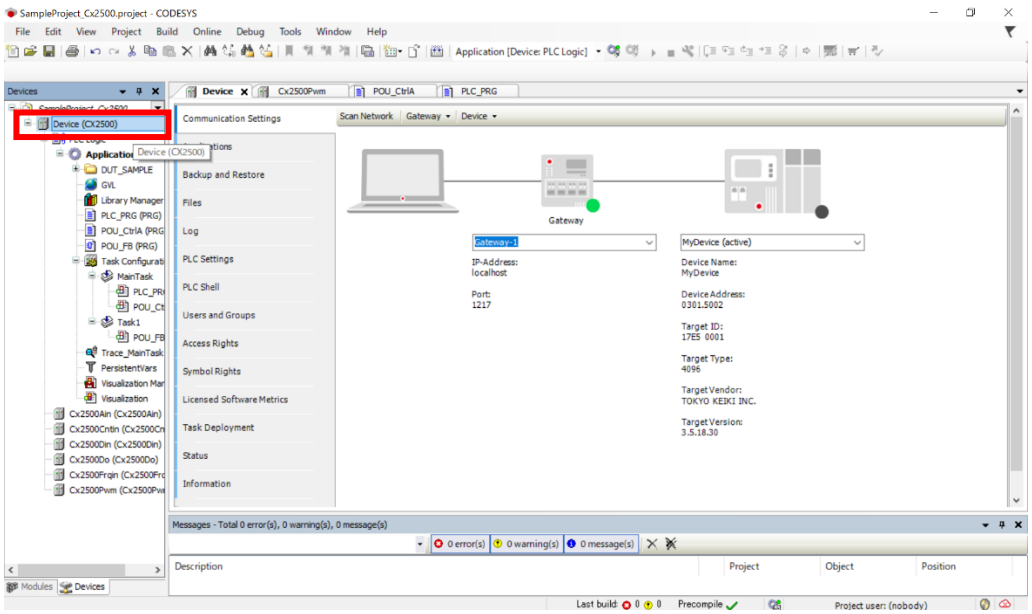


Figure 107 デバイスウィンドウ Device の選択

(2) 表示されるコンテキストメニューから「Add Device…」を選択して下さい。

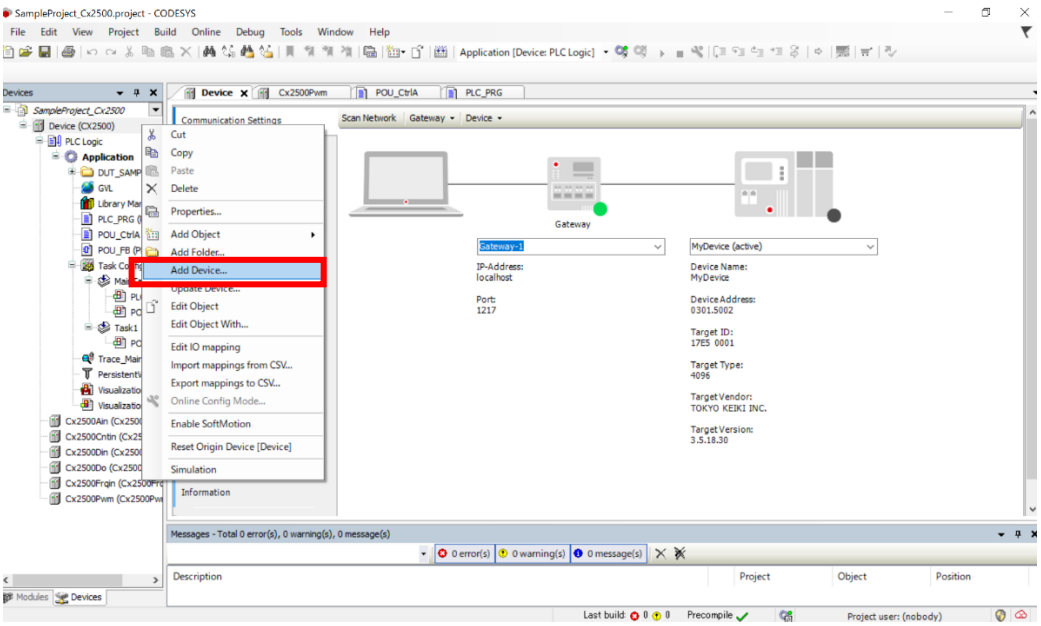


Figure 108 Device Add Device の選択

(3) 「Add Device」ウィンドウが表示されるので、デバイス一覧の中から「CANbus」を選択して「Add Device」ボタンを押して下さい。

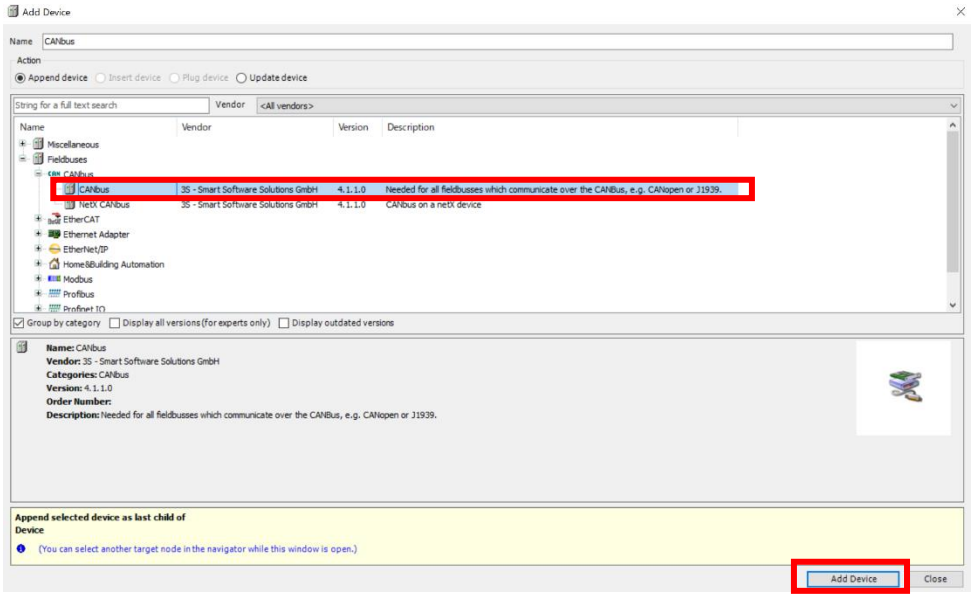


Figure 109 Add Device ウィンドウ CANbus の選択

(4) 「Add Device」ウィンドウを閉じると、デバイスウィンドウ上に CANbus が追加されています。

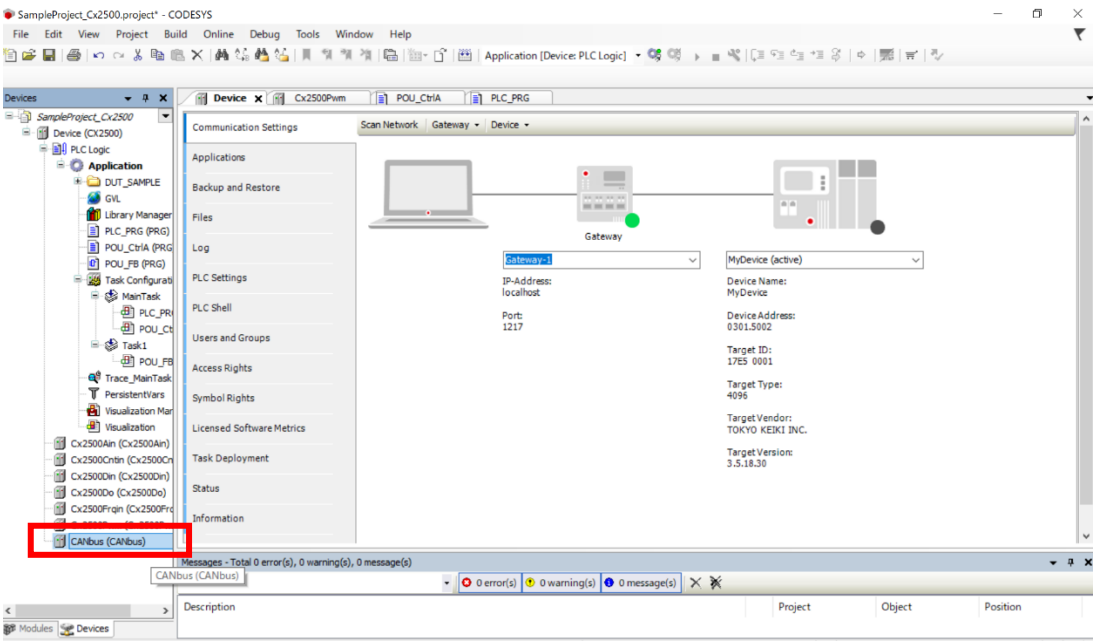


Figure 110 デバイスウィンドウ CANbus 追加

(5) 次に、プロトコルマネージャー(J1939_Manager)を追加します。追加した「CANbus」にカーソルを合わせ右クリックして下さい。

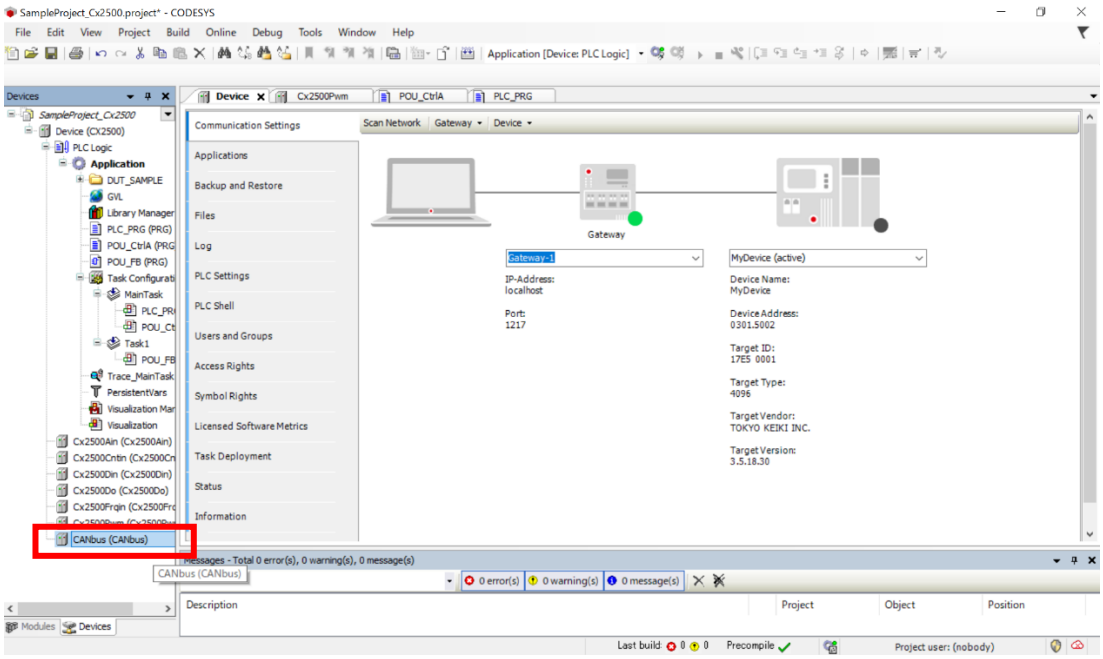


Figure 111 デバイスウィンドウ CANbus の選択

(6) 表示されるコンテキストメニューから「Add Device…」を選択して下さい。

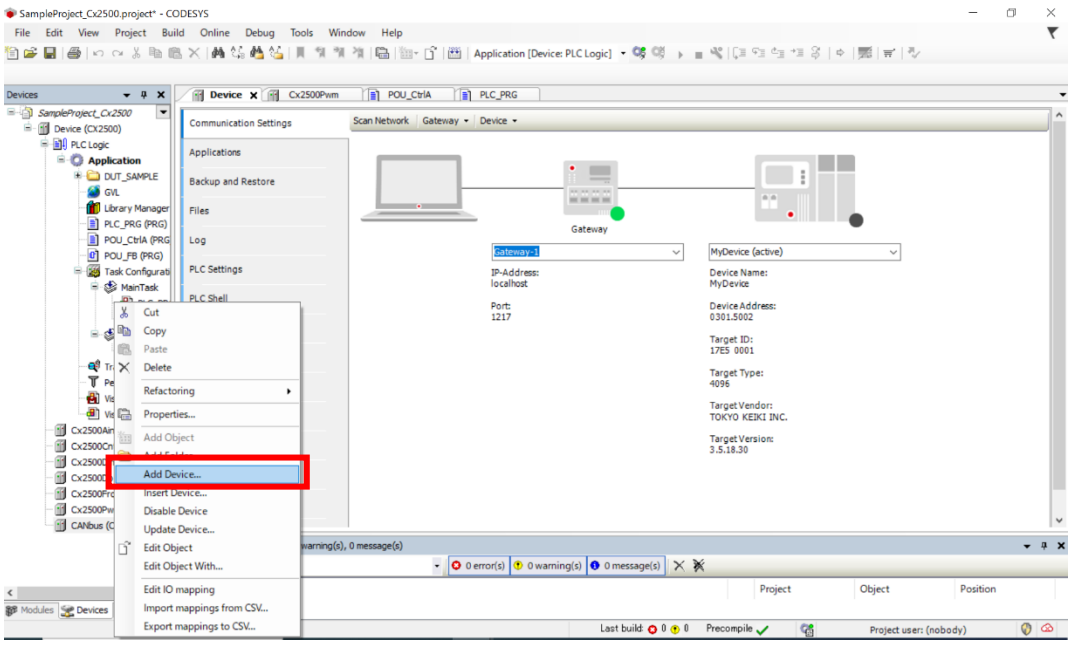


Figure 112 CANbus Add Device の選択

(7) 「Add Device」ウィンドウが表示されるので、デバイス一覧から「J1939_Manager」を選択して「Add Device」ボタンを押して下さい。

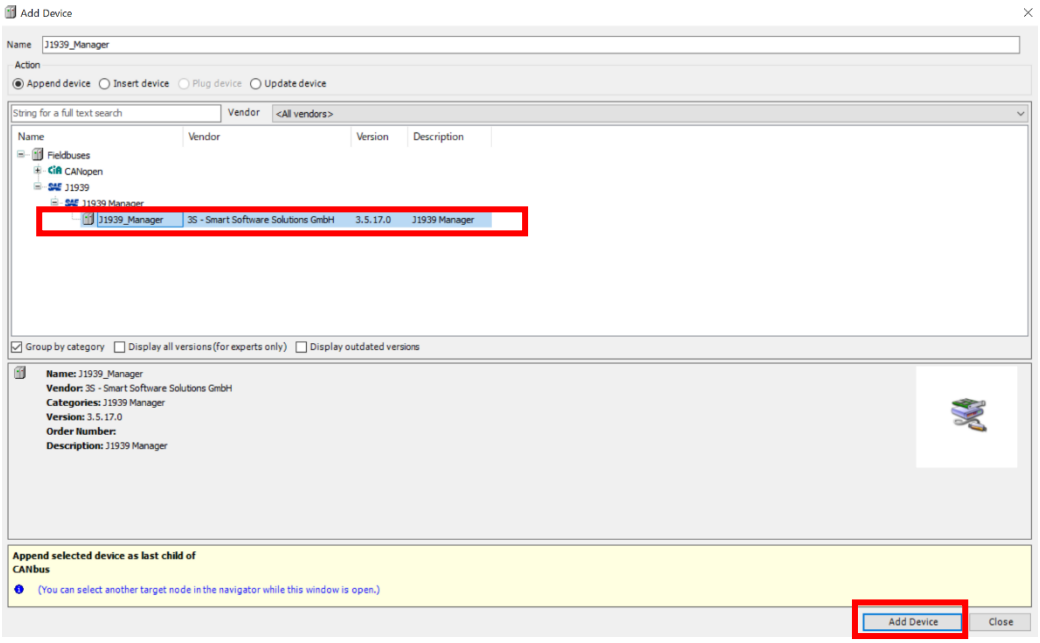


Figure 113 Add Device ウィンドウ J1939_Manager の選択

(8) 「Add Device」ウィンドウを閉じると、デバイスウィンドウの「CANbus」下に「J1939_Manager」が追加されています。

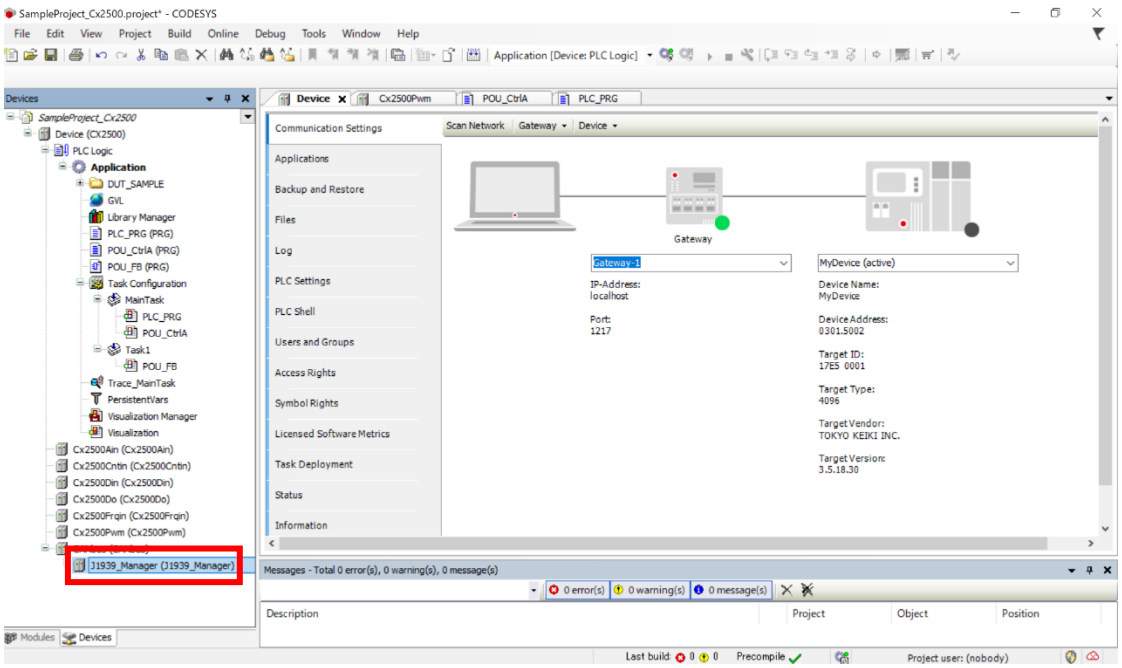


Figure 114 デバイスウィンドウ J1939_Manager の追加

(9) 次に、フィールドバスチャネルのネットワークに接続するデバイスを登録します。このデバイスは CX2500 の通信相手又は CX2500 にあたります。追加した「J1939_Manager」にカーソルを合わせ右クリックして下さい。

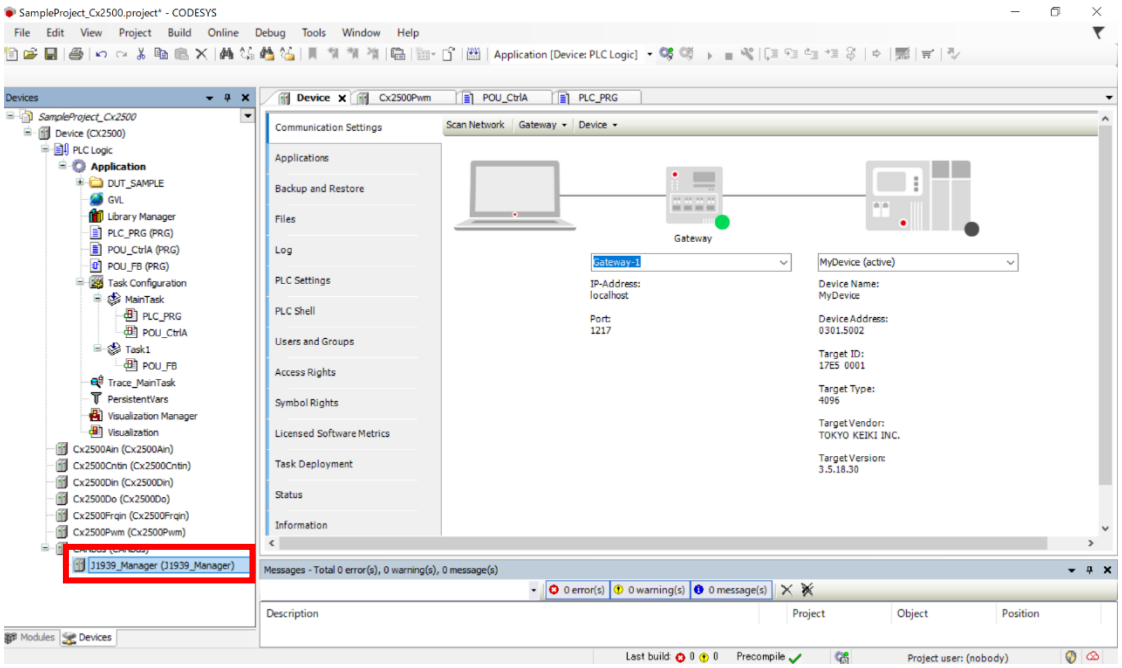


Figure 115 デバイスウィンドウ J1939_Manager の選択

(10) 表示されるコンテキストメニューから「Add Device…」を選択して下さい。

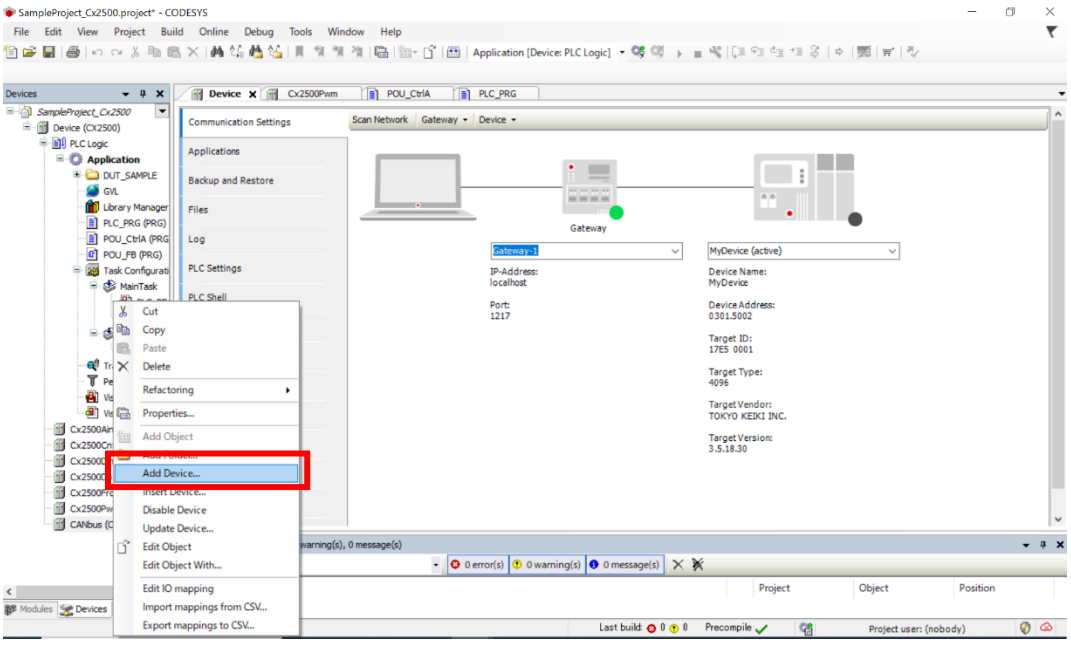


Figure 116 J1939_Manager Add Device の選択

(11) 「Add Device」ウィンドウが表示されるので、デバイス一覧から「J1939_ECU」を選択して「Add Device」ボタンを押して下さい。なお、CANopen の場合、CODESYS にデフォルトでインストール(登録)されているデバイスとは別に、CANopen デバイス(EDS ファイル)を新しく CODESYS-IDE に追加する場合は、別紙「CX2500Codesys_UserManual_ForSetup」4.4 節の要領で EDS ファイルのインストール(登録)をおこなってください。

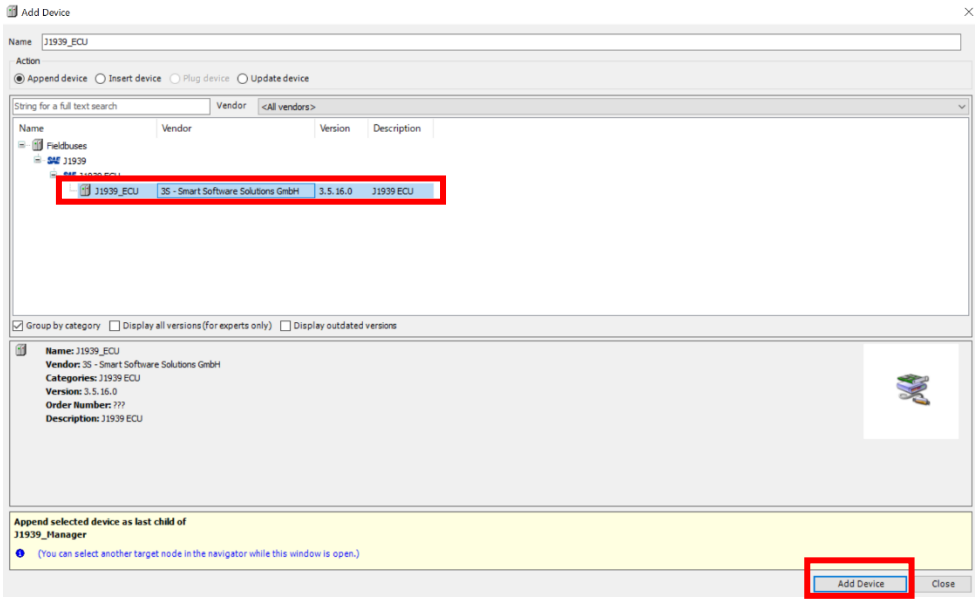


Figure 117 Add Device ウィンドウ J1939_ECU の選択

(12)　「Add Device」ウィンドウを閉じると、「J1939_Manager」下に「J1939_ECU」が追加されています。なお、同じデバイスを複数追加した場合は図のように「デバイス名_(番号)」の名称で追加されます。デバイス名称は各デバイスのコンテキストメニューのプロパティからおこなうことができます。

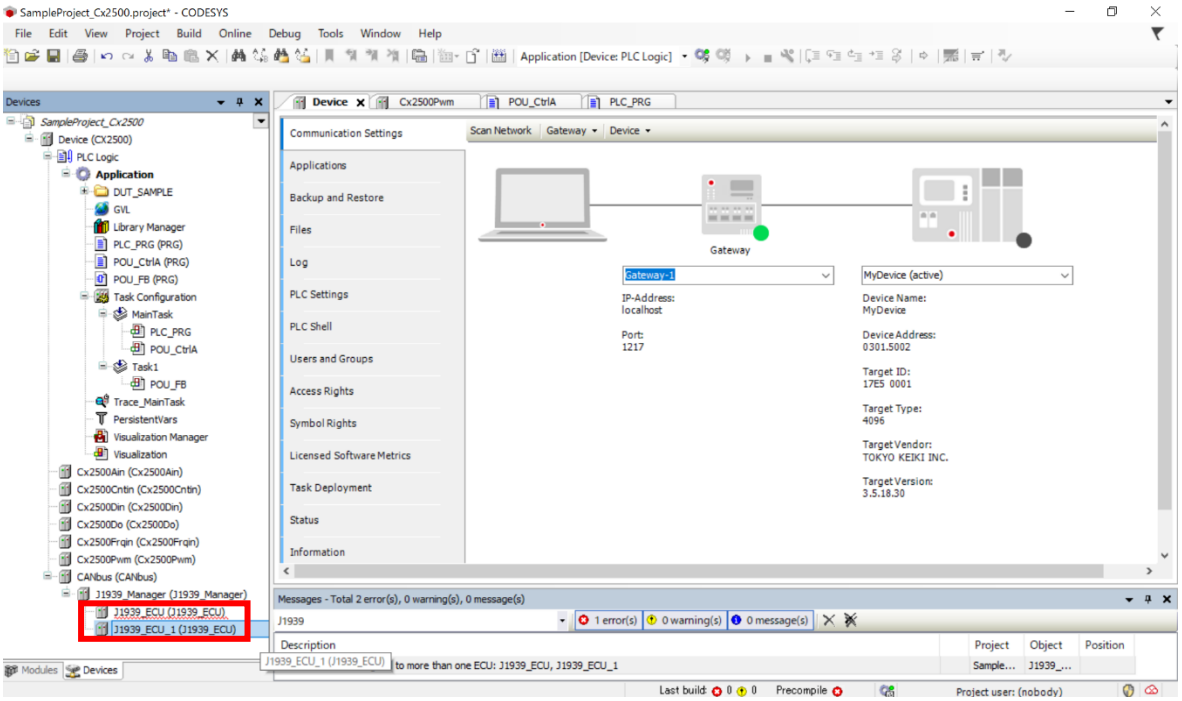


Figure 118　デバイスウィンドウ　CAN デバイス追加

8.3. J1939

8.3.1. CANbus

バスマスタ CANbus エディタ画面について説明します。バスマスタのエディタ画面は、エディタウィンドウにあるバスマスタをダブルクリックすると表示されます。

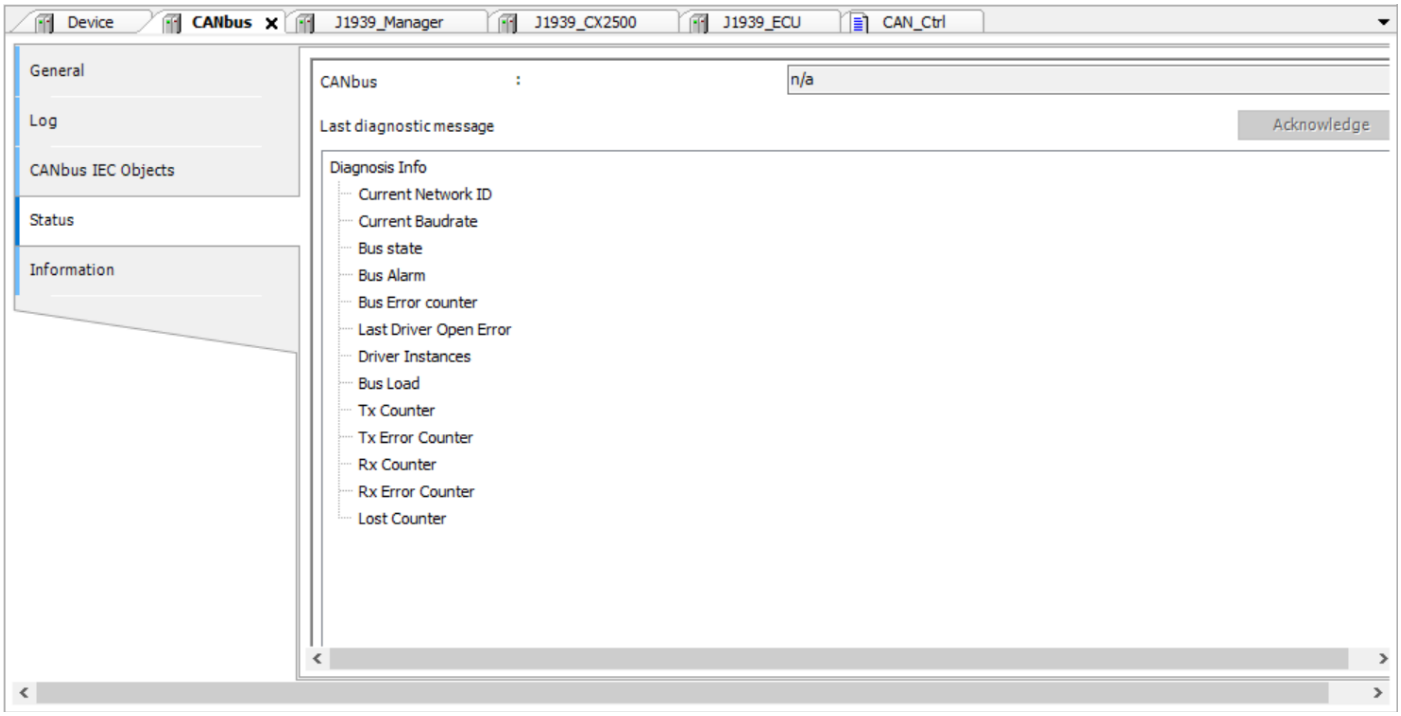


Figure 119 CANbus エディタ画面

Table 56 CANbus タブ一覧

タブ名称	摘要
General	CAN バスに関する設定をおこなうことができる。
Log	デバッグ中にバスマスタ CANbus の動作ログを確認できる。
CANbus IEC Objects	このタブ内で定義されているオブジェクト名称を使うことでユーザーアプリケーションから CANbus の一部情報にアクセスできる。
Status	デバッグ中にバスステータス情報を確認できる。
Information	バスマスタ CANbus のバージョン情報などを確認できる。

8.3.1.1. General タブ

バスマスタ CANbus の General タブでは、使用する CX2500 のチャネルとボードレートを設定して下さい。

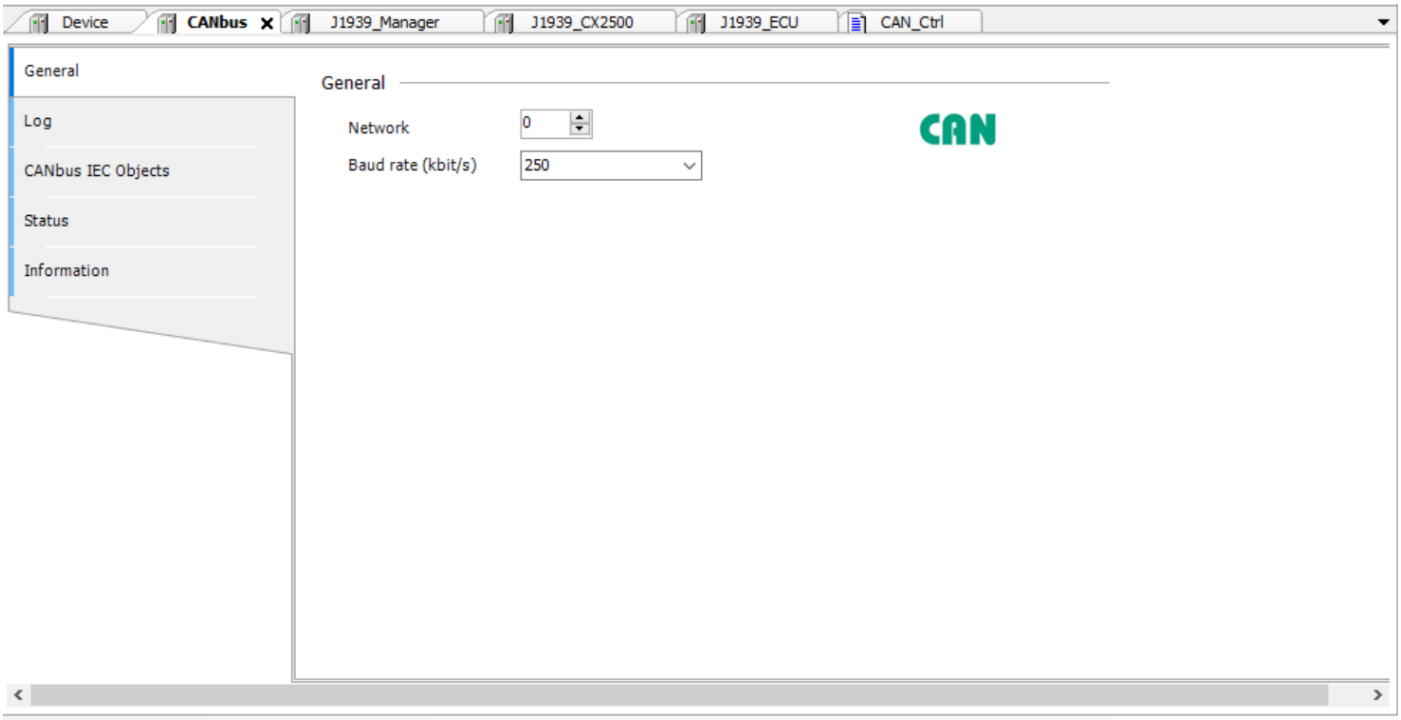


Figure 120 CANbus General タブ

Table 57 CANbus General タブ 設定項目

設定項目	摘要
Network	フィールドバスで使用する CX2500 の CAN チャネル番号(0～4)を入力する。
Baud rate(kbit/s)	選択タブからアプリケーションで使用するボードレートを選択する。 なお、設定可能なボードレートは Table 47 と同じとなる。

8.3.1.2. Status タブ

このタブでは、デバッグ中、下図のように CAN バスステータス情報が表示されます。

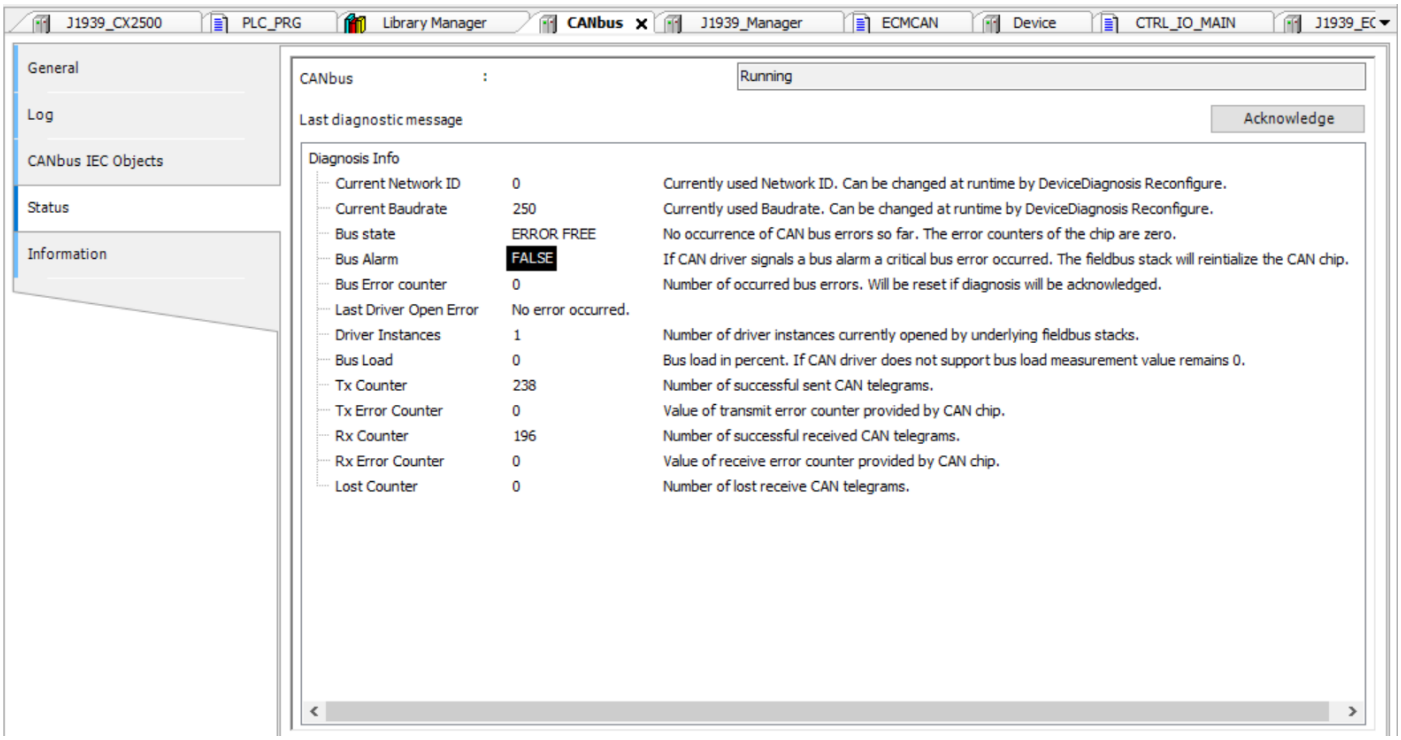


Figure 121 CANbus Status タブ(デバッグ中の表示例)

8.3.2. J1939_Manager

プロトコルマネージャーJ1939_Manager エディタ画面について説明します。プロトコルマネージャーのエディタ画面は、エディタウィンドウにあるプロトコルマネージャーをダブルクリックすると表示されます。

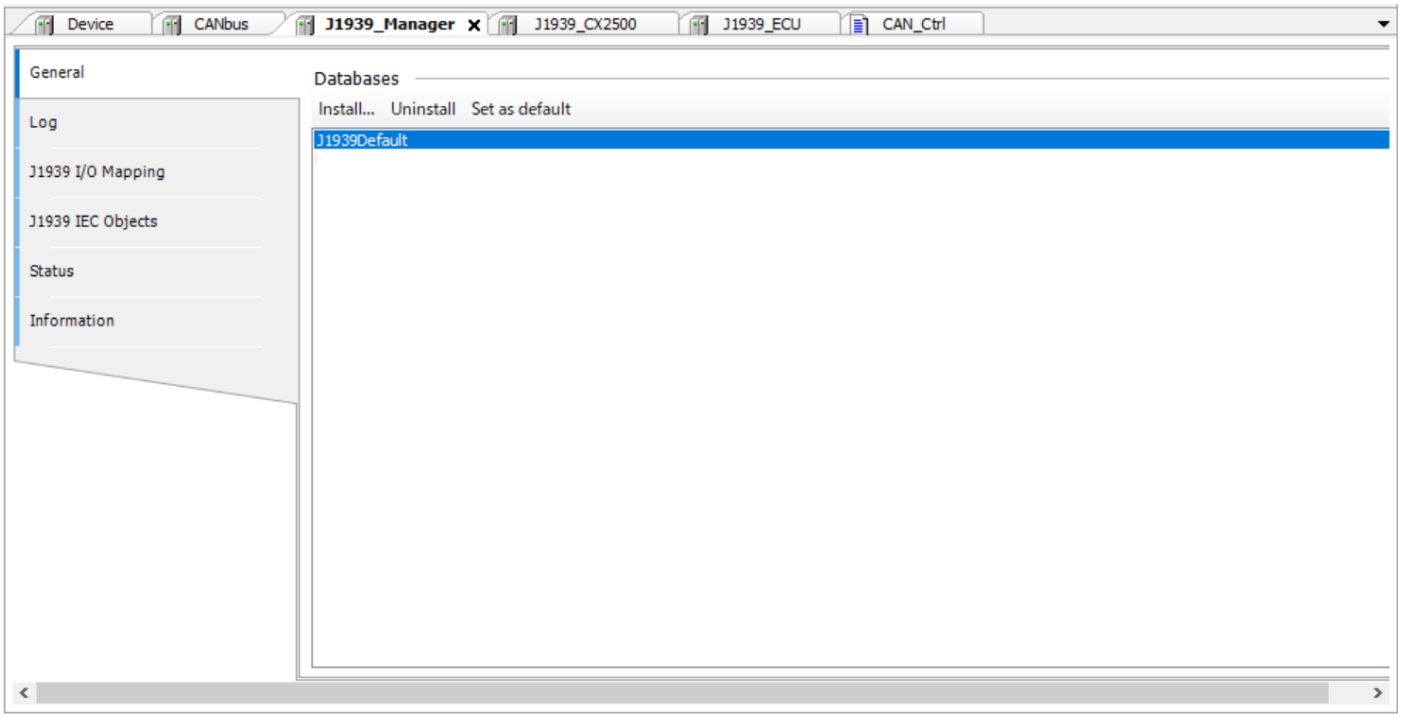


Figure 122 J1939_Manager エディタ画面

Table 58 J1939_Manager タブ一覧

タブ名称	摘要
General	DBC ファイルの登録/削除をおこなうことができる。ファイルを登録することで J1939 フィールドバスでの各 CAN デバイスで、送受信メッセージを簡単に設定できる。
Log	デバッグ中にプロトコルマネージャーJ1939_Manager の動作ログを確認できる。
J1939 I/O Mapping	フィールドバスを制御する周期時間となるバスサイクルを設定できる。
J1939 IEC Objects	このタブ内で定義されているオブジェクト名称を使うことでユーザーアプリケーションから J1939_Manager の一部情報にアクセスできる。
Status	デバッグ中に J1939_Manager の動作良否を確認できる。
Information	プロトコルマネージャーJ1939_Manager のバージョン情報などを確認できる。

8.3.2.1. General タブ

このタブでは、DBC ファイルの登録/削除をおこなうことができます。DBC ファイルを登録することで、J1939 フィールドバスでの各 CAN デバイスで、送受信メッセージを簡単に設定することができます。登録の際はタブ画面上の「Install」ボタンを押してファイルをインストール(登録)できます。登録した DBC ファイルを削除する際は、同画面上の削除対象ファイルを選択した上で「Uninstall」ボタンを押すことで削除できます。

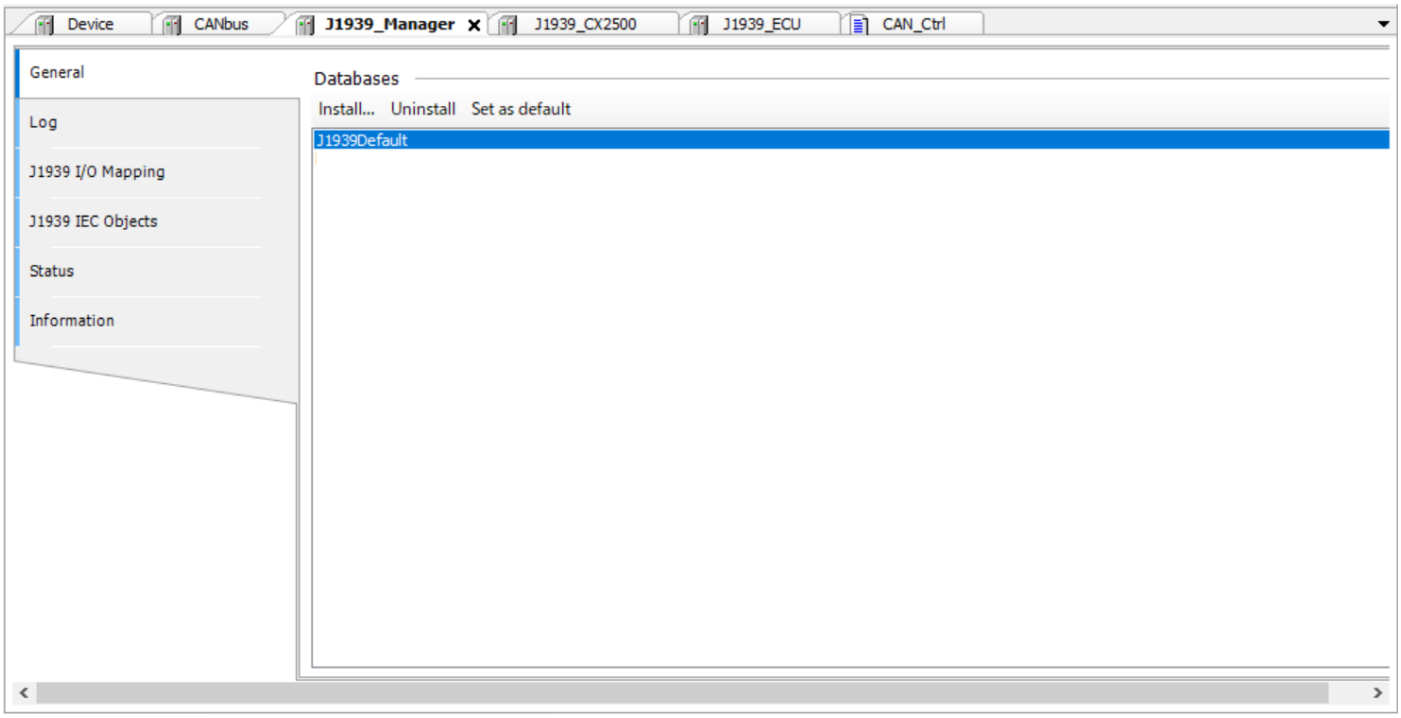


Figure 123 J1939_Manager General タブ

8.3.2.2. J1939 I/O Mapping タブ

このタブでは、フィールドバス内の送受信メッセージやステータス情報を制御する周期時間を設定して下さい。

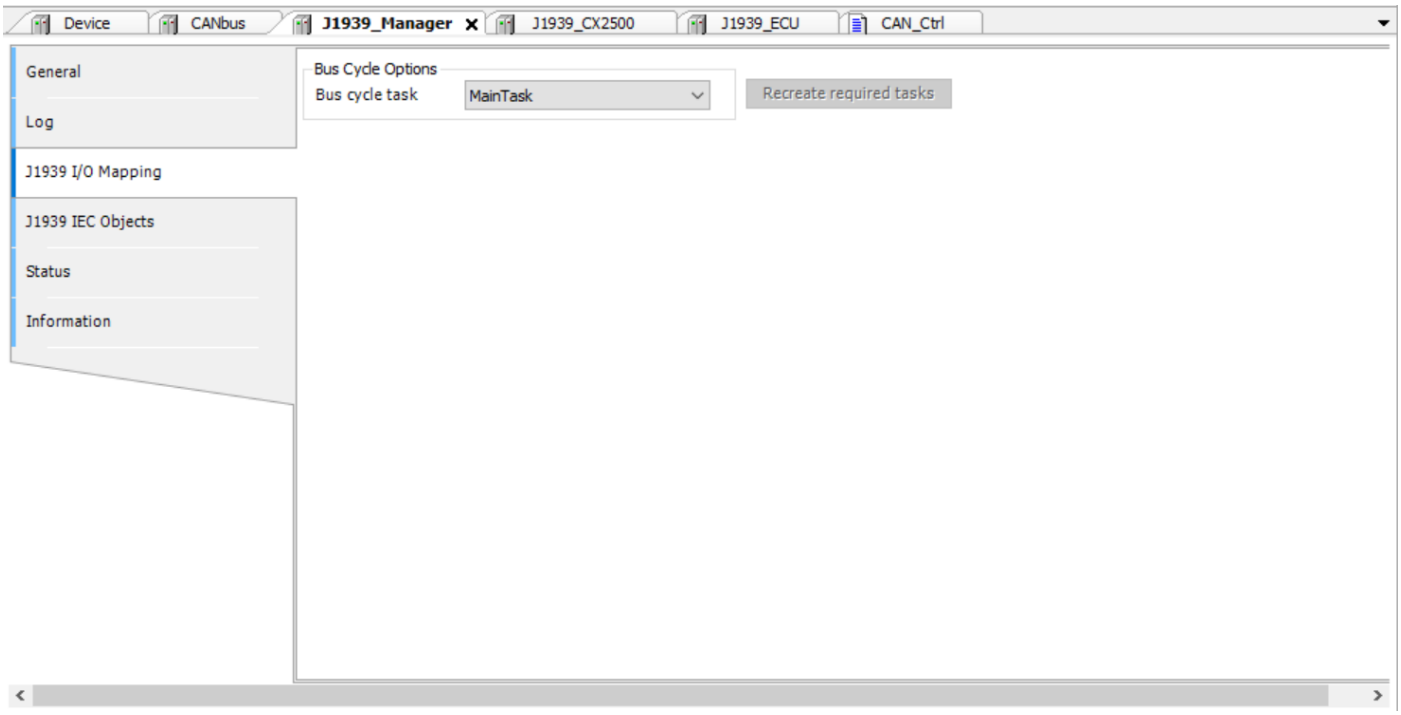


Figure 124 J1939_Manager J1939 I/O Mapping タブ

8.3.3. CAN デバイス

J1939 の CAN デバイスエディタ画面について説明します。CAN デバイスのエディタ画面は、エディタウィンドウにある CAN デバイスをダブルクリックすると表示されます。

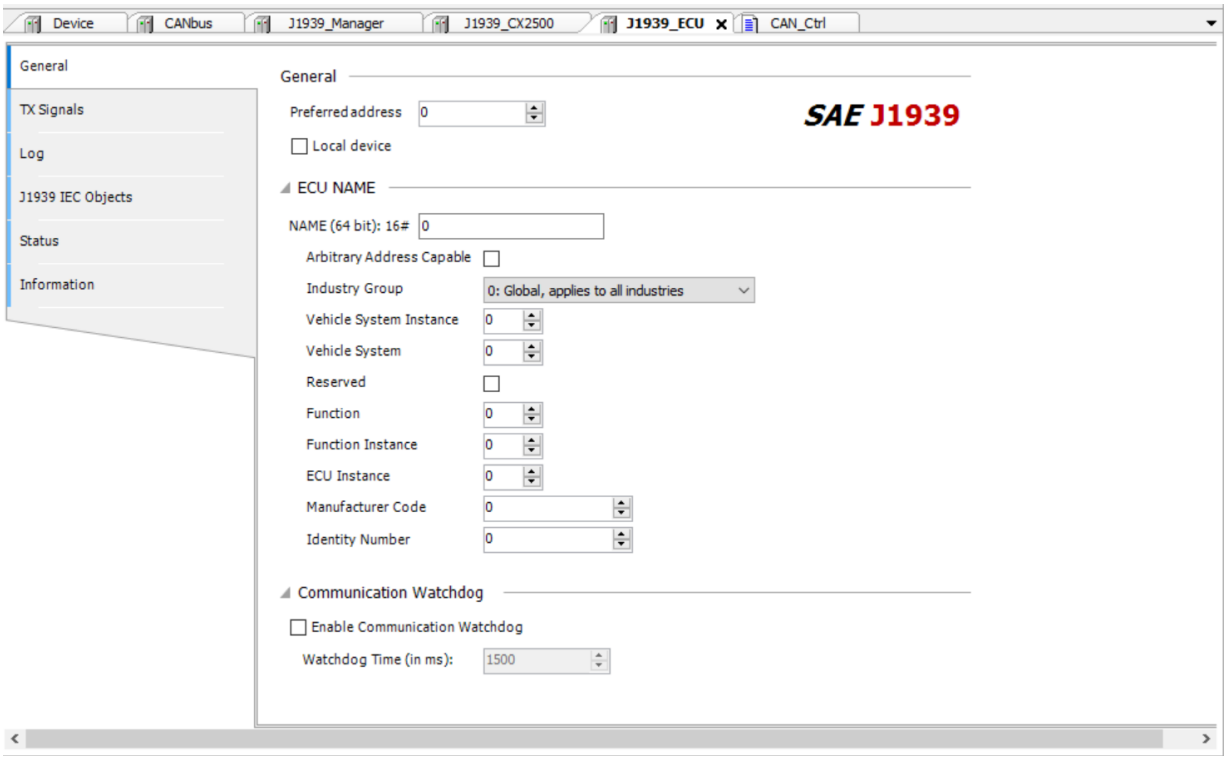


Figure 125 CAN デバイス(J1939) エディタ画面

Table 59 J1939 CAN デバイス タブ一覧

タブ名称	摘要
General	J1939 プロトコルにおけるパラメータ(CAN-ID など)を設定できる。
TX Signals	CAN デバイスに関わる送受信メッセージの登録および送信設定ができる。
P2P RX Signals	General で Local device にチェックを入れると表示される。他の ECU が受信する必要がある全ての PGN が表示される。
Log	デバッグ中に CAN デバイスの動作ログを確認できる。
J1939 I/O Mapping	上記 Signals タブで定義したメッセージのデータに対して、ユーザーアプリケーションで使う変数を紐づけることができる。メッセージが登録されていないとこのタブは表示されない。
J1939 IEC Objects	このタブ内で定義されているオブジェクト名称を使うことでユーザーアプリケーションから CAN デバイスの一部情報にアクセスできる。
Status	デバッグ中、J1939 プロトコルにおける診断ステータス情報(DM1 – Diagnostic Message1)を確認できる。
Information	CAN デバイスのバージョン情報などを確認できる。

8.3.3.1. General タブ

このタブは、J1939 プロトコルにおけるパラメータを設定できます。各種ユーザーアプリケーション所望の値になるように設定して下さい。

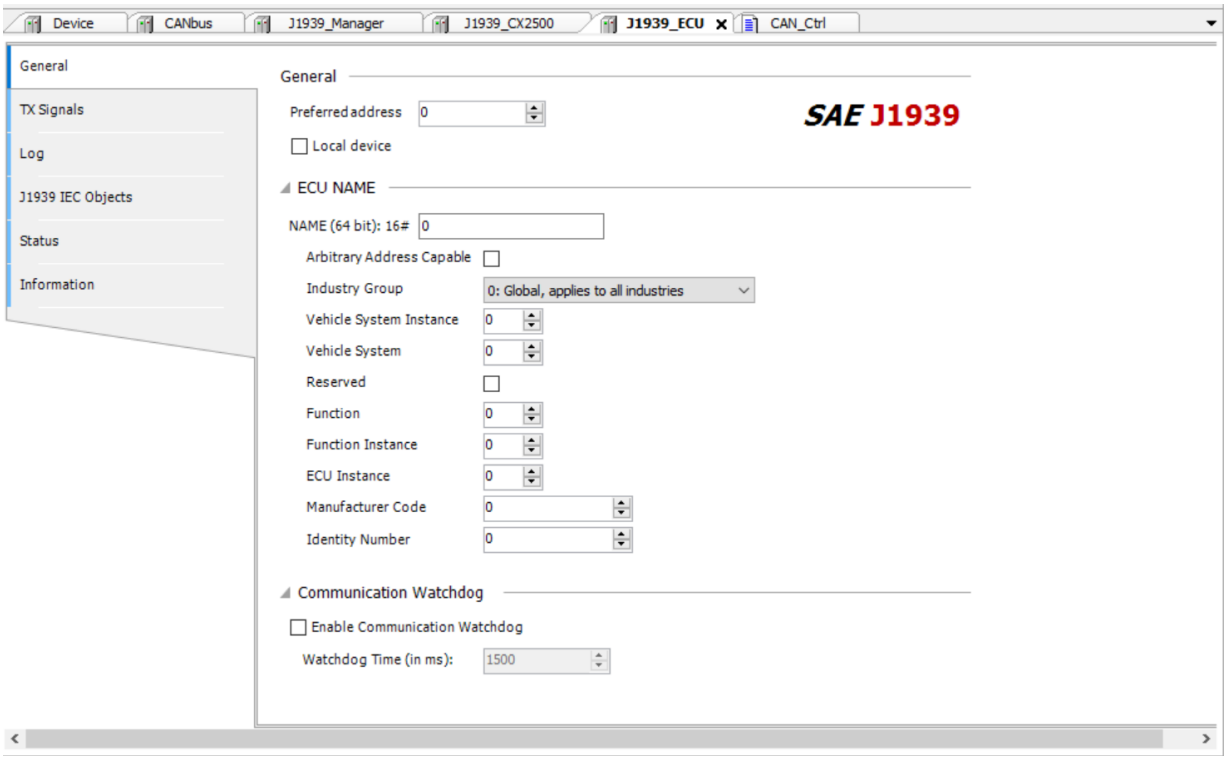


Figure 126 CAN デバイス(J1939) General タブ

Table 60 CAN バス General タブ 設定項目一覧

区分	名称	摘要
General	Preferred address	CAN デバイスのソースアドレスを入力する。
	Local device	チェックの有無によって、TxSignals タブで登録できるメッセージが変わる。 ☑ :CX2500 が他 ECU(デバイス)へ送信するメッセージを登録できる。 ☐ :CX2500 が他 ECU から受信するメッセージを登録できる。
ECU NAME※17	NAME	ECU NAME を入力する。もしくは、以降のパラメータを入力することにより自動で数値が設定される。
	Arbitrary Address Capable	チェックを入れると、他 ECU でアドレスが競合した場合に、CAN デバイスを別のアドレスに変更することを試みる。
	Industry Group	ECU NAME に関する設定。所望の値を入力する。
	Vehicle System Instance	
	Vehicle System	
	Reserved	
	Function	
	Function Instance	
	ECU Instance	
	Manufacturer Code	
	Identity Number	
Communication Watchdog※18	Enable Communication Watchdog	チェックを入れると、CAN デバイスに関するメッセージが Watchdog Time の時間内に送受信されたかを監視する。時間内に送受信されていない場合は、デバイスウィンドウ上の CAN デバイスのアイコン表示が△になる。
	Watchdog Time	送受信可否監視時間[ms]を入力する。

※17 ECU NAME については、J1939 プロトコル規格書などを参照。

※18 Local Device が有効の場合は表示されない。

8.3.3.2. TX Signals タブ

TX Signals タブでは、送受信メッセージの登録・削除をおこなうことができます。ただし、General タブの設定項目 Local Device の設定により登録できるメッセージの種類が異なります。

Table 61 TX Signals タブの役割

Local Device 設定	TX Signal タブ役割
☑	CX2500 が他の ECU(デバイス)へ送信するメッセージを登録できる。
☐	CX2500 が他の ECU(デバイス)から受信するメッセージを登録できる。

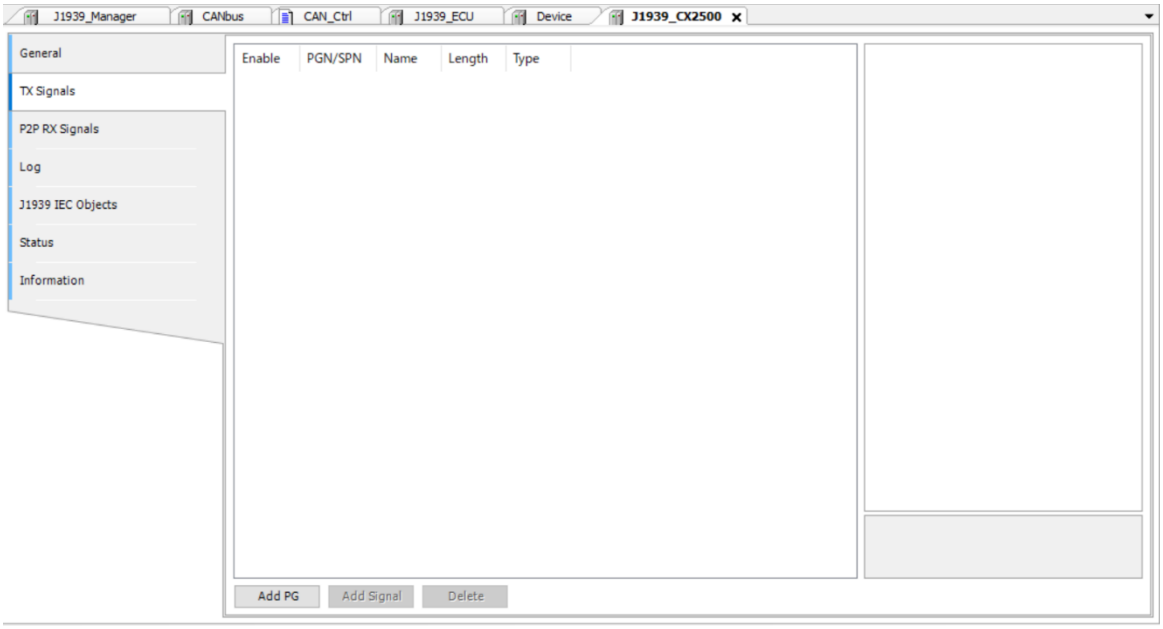


Figure 127 CAN デバイス(J1939) TX Signals タブ

【メッセージの登録】

ここでは、メッセージ登録の手順を示します。メッセージの登録は送受信に限らず同様です。

(1) TX Signals タブ画面にて、「Add PG」ボタンを押して下さい。

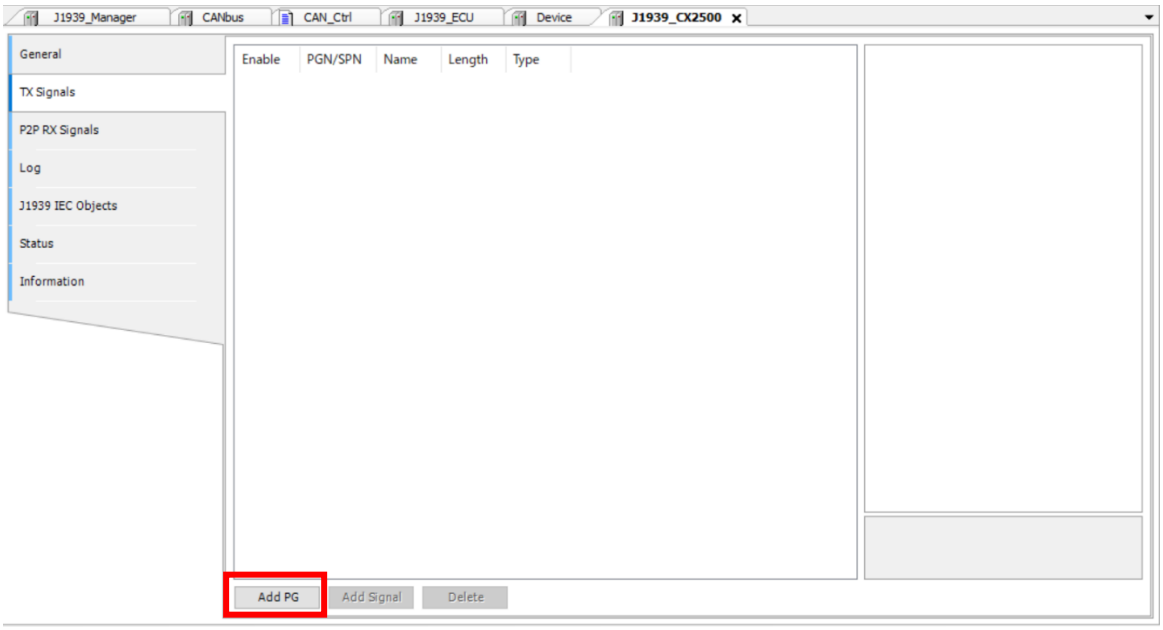


Figure 128 メッセージの登録 Add PG ボタン

(2) 「Add Parameter Group」ウィンドウが表示されます。このウィンドウにはタブが 2 つあります。事前にプロトコルマネージャーで DBC ファイルを登録していれば、「Datebase」タブに DBC ファイルに登録されている CAN メッセージの一覧が表示されます。所望のメッセージを選択して「Add PG」ボタンを押して下さい。もし、DBC ファイルが無かったり、DBC ファイル内に登録されていないメッセージを登録したい場合は「Custom」タブで所望の値を入力することでメッセージを登録することができます。

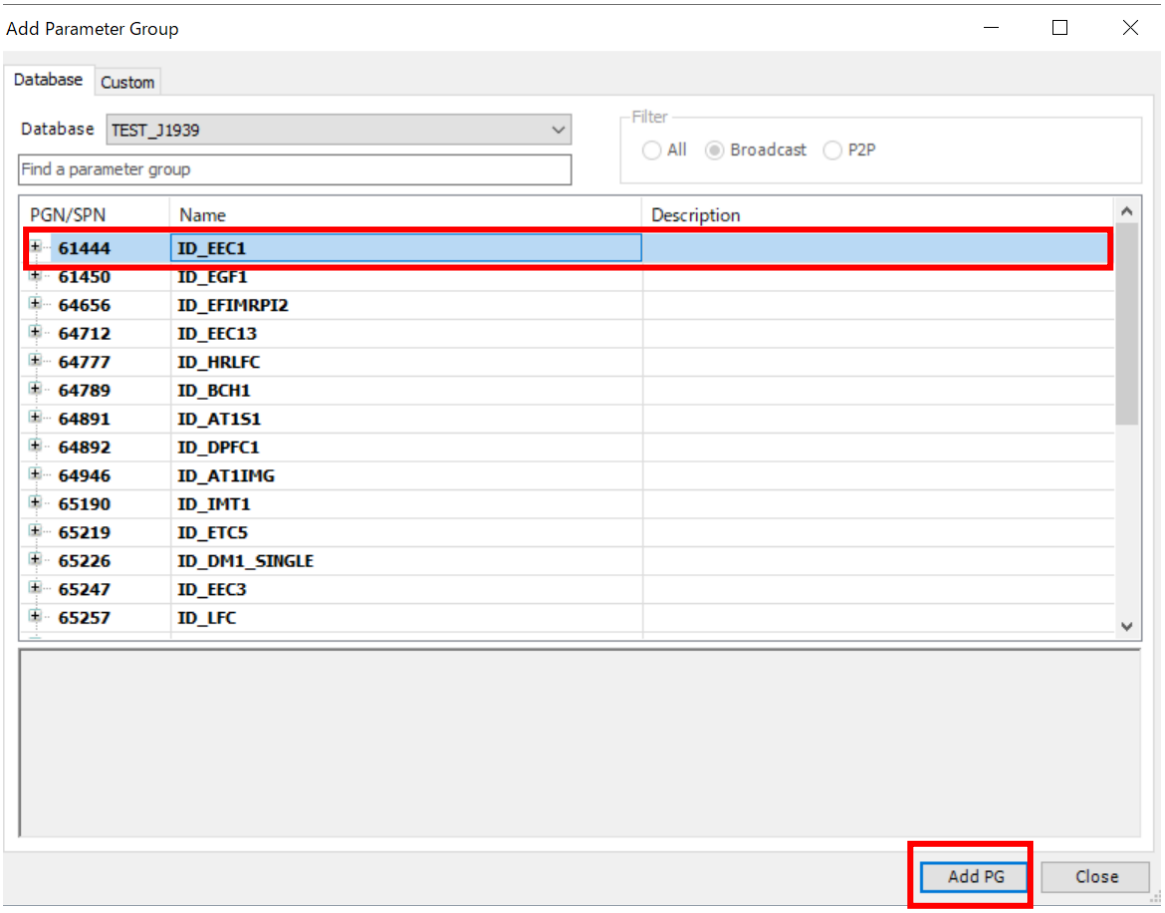


Figure 129 Add Parameter Group ウィンドウ Database タブ

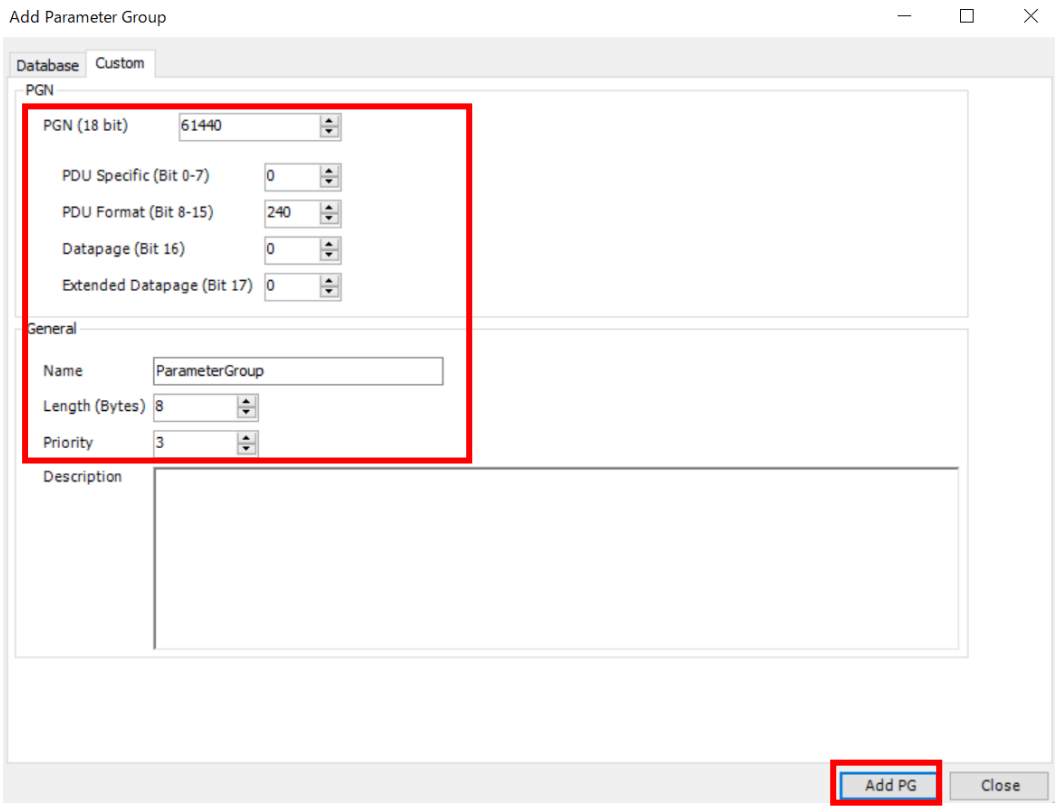


Figure 130 Add Parameter Group ウィンドウ Custom タブ

(3) TX Signals にメッセージが登録されます。登録することで、CAN デバイスの I/O Mapping タブで作業がおこなうことができたり、送信メッセージの場合は送信方法設定ができるようになります。また、登録したメッセージを選択して「Add Signal」ボタンを押すことで、メッセージデータの信号ラベルを登録することができます。信号ラベルを登録すると、メッセージデータ内の指定領域において、ユーザーアプリケーションでそのデータを設定・取得することができるようになります。(DBC ファイルであらかじめ登録していれば、メッセージ登録した段階で信号ラベルが既に登録されています)

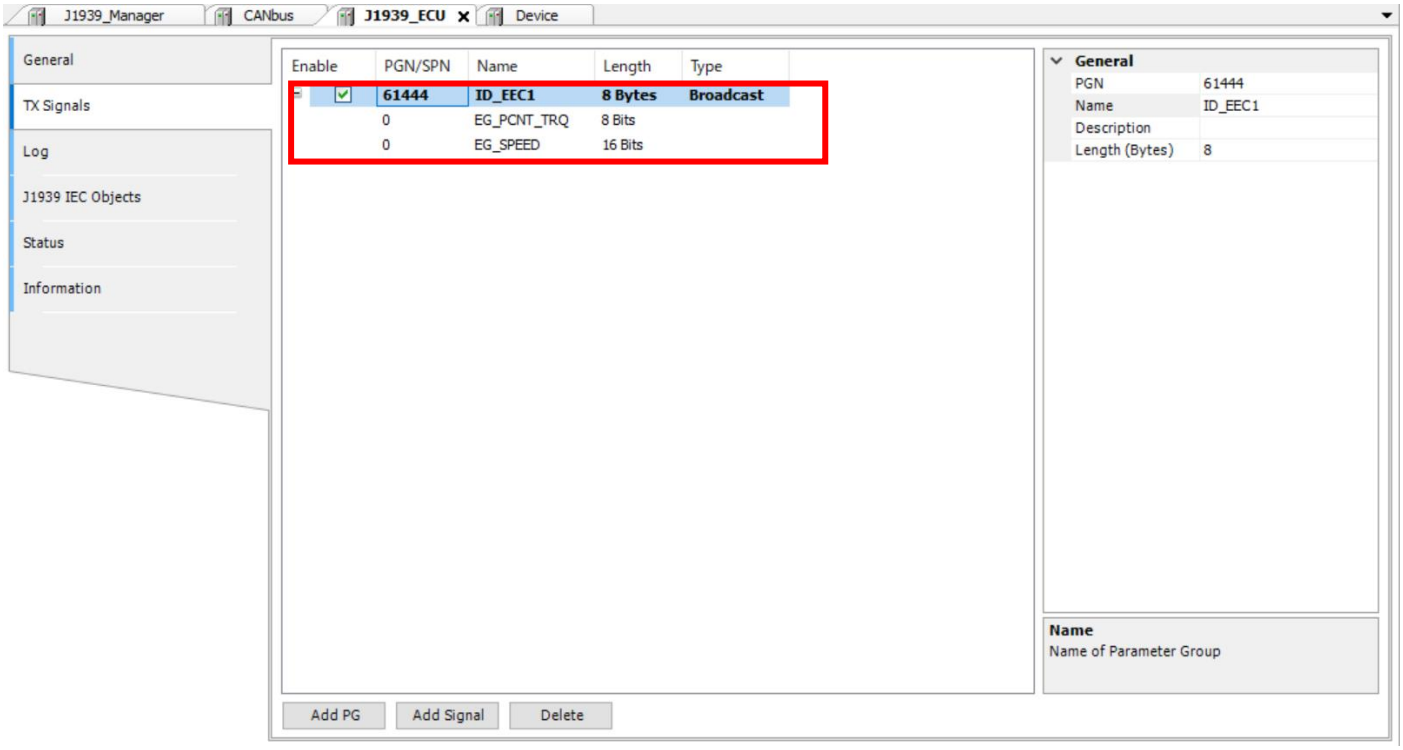


Figure 131 TX Signals タブ メッセージ登録完了後

【送信方法設定】

デバイスに送信メッセージを登録できる設定の場合、各メッセージの送信方法の設定もおこなうことができます。所望のメッセージを選択して表示されるメニュー内の値を編集することで設定できます。

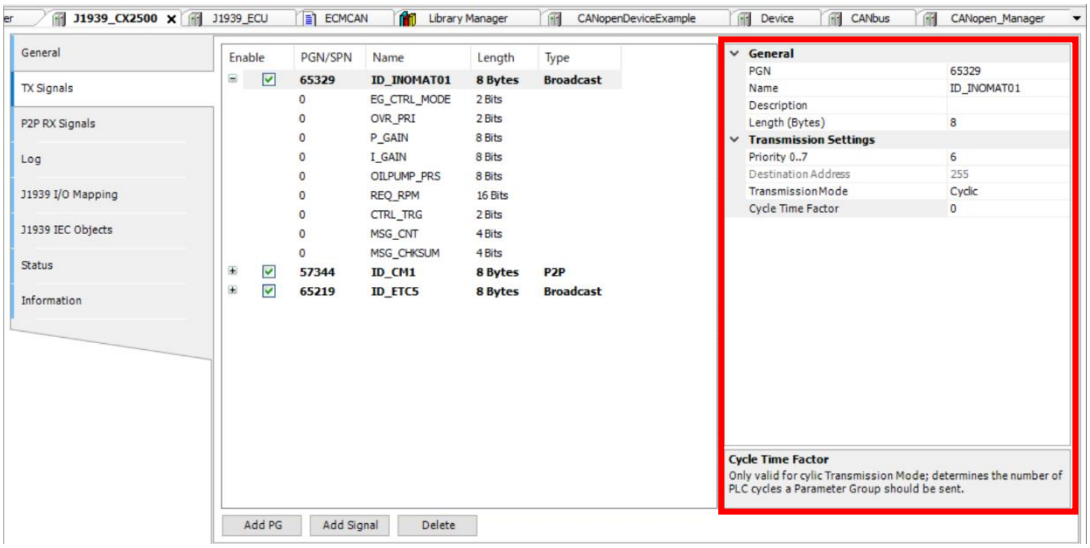


Figure 132 TX Signals タブ 送信方法設定メニュー

Table 62 送信方法設定メニュー

区分	名称	摘要
General	PGN※19	パラメータグループ番号
	Name※19	パラメータグループの名称
	Description※19	パラメータグループの説明
	Length(Bytes)※19	パラメータグループのメッセージデータの長
Transmission Settings	Priority※19	送信優先度
	Destination Address※19	ターゲットアドレス(P2P メッセージのみ有効)
	Transmission Mode	送信方法を選択タブから選択する。 項目 / 摘要 Change Of State / メッセージ内のデータがユーザーアプリケーションで変更された時送信する。 Cyclic※20 / 周期的に送信する。 On Request / Request PGN メッセージを受信した時に送信する。 Application Triggered / 非対応
	Cycle Time Factor※20	Transmission Mode を Cyclic にした場合に有効になる。メッセージの送信周期を設定する。 送信周期時間:「J1939 Bus Cycle Task[ms]」 × 「Cycle Time Factor」

※19 これらの設定項目については、メッセージ登録した時点で既に設定されているため、再編集は不要となる。

※20 フィールドバスに限らず、他のタスクや CX2500 内部処理のため、Cyclic で設定した周期から定常的に遅れて送信されることが想定されます。その際は、タスクを短周期にしたり、Cycle Time Factor を微調整して対応をお願い致します。

8.3.3.3. P2P RX Signals タブ

このタブは、General で Local device にチェックを入れると表示されます。このデバイスの他にも別の ECU(デバイス)が受信する必要がある PGN がある場合、このタブで受信メッセージを登録することができます。メッセージの登録手順は TX Signals タブと同様です。

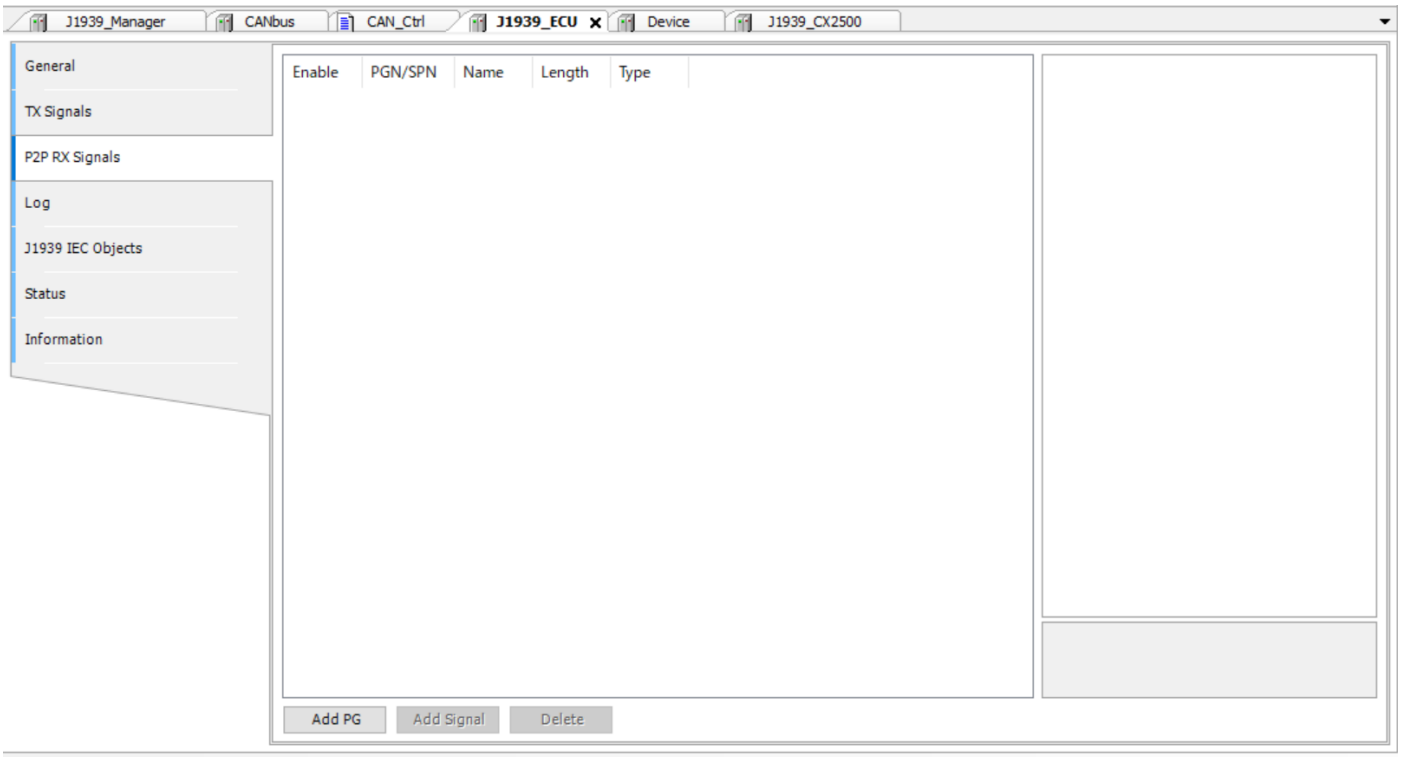


Figure 133 CAN デバイス(J1939) P2P RX Signals タブ

8.3.3.4. J1939 I/O Mapping タブ

このタブは、Signals タブでメッセージを登録した場合に表示されます。各メッセージのデータが一覧表示されており、IO ドライバと同様にこれらに変数を紐づけてユーザーアプリケーションで使用することができます。

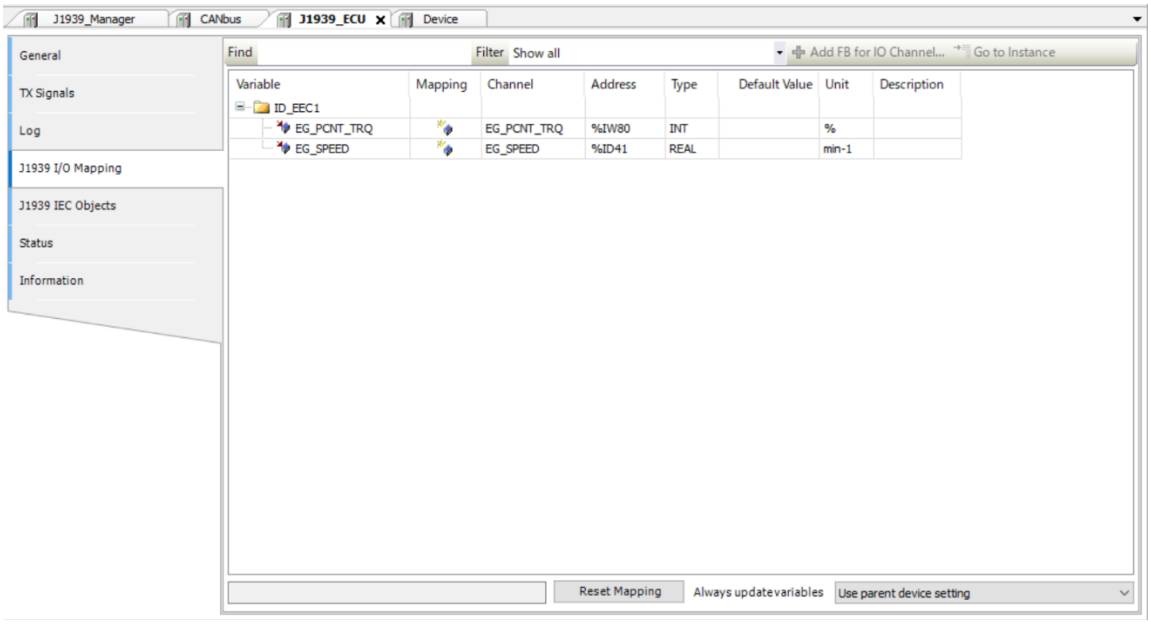


Figure 134 CAN デバイス(J1939) J1939 I/O Mapping タブ

8.3.3.5. Status タブ

このタブは J1939 プロトコルにおける診断情報を表示します。CX2500 が診断メッセージ(DM1 – Diagnostic Message1)を受信した場合、メッセージ内容に従って下図のように診断情報が表示されます。診断メッセージデータの詳細については、J1939 プロトコルの規格書を参照して下さい。

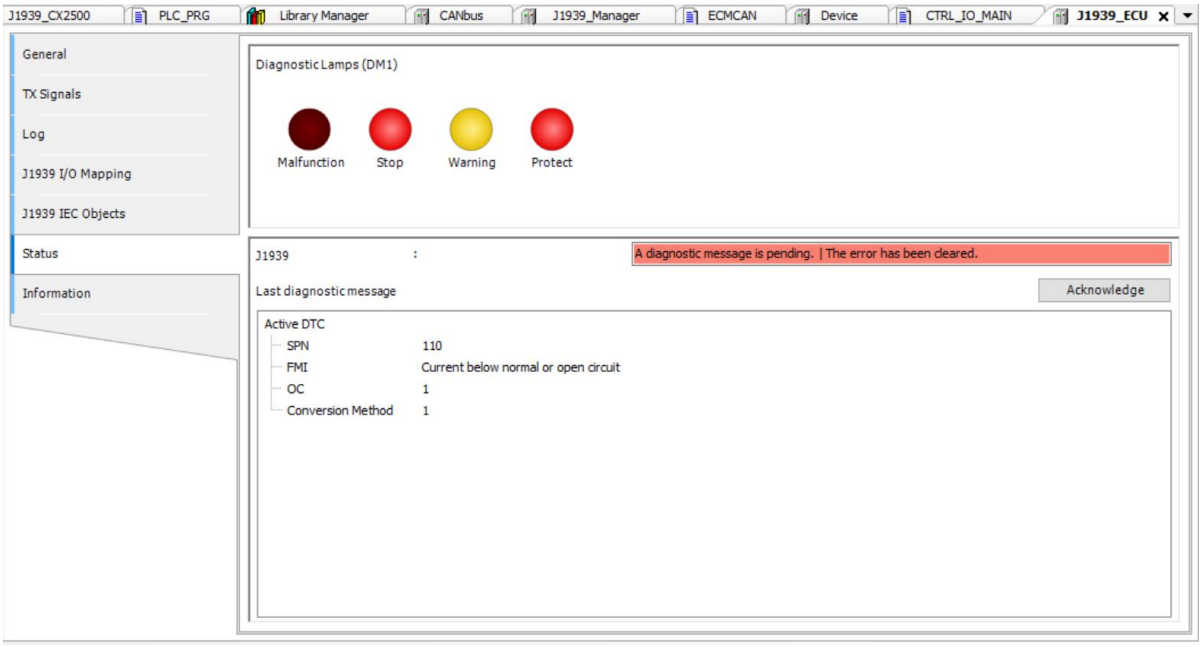


Figure 135 CAN デバイス(J1939) Status タブ(デバッグ中の表示例)

8.4. CANopen

8.4.1. CANbus

CANopen をフィールドバスで使う際のバスマスタ CANbus の設定・モニタ等は J1939 を使う場合と同様です。8.3.1 項を参照して下さい。

8.4.2. CANopen_Manager

プロトコルマネージャーCANopen_Manager エディタ画面について説明します。プロトコルマネージャーのエディタ画面は、エディタウィンドウにあるプロトコルマネージャーをダブルクリックすると表示されます。

プロトコルマネージャーはリモートデバイス(CANopen マスタ)として CX2500 を使用する場合に紐づけが必要になります。ローカルデバイス(CANopen スレーブ)として CX2500 を使用する場合は CANopen Manager は不要です。その際は、バスマスタ CANbus から直接 CAN デバイスを紐づけて下さい。

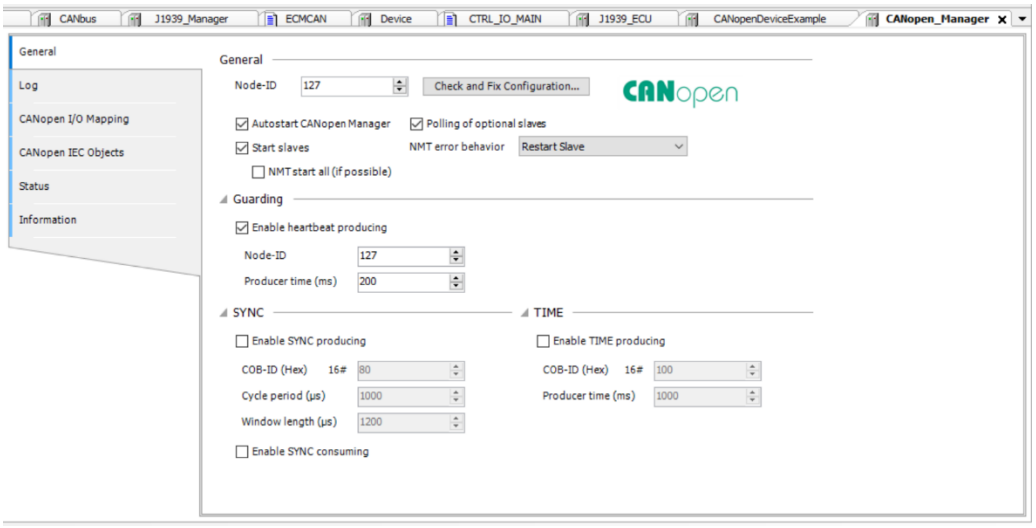


Figure 136 CANopen Manager エディタ画面

Table 63 CANopen_Manager タブ一覧

タブ名称	摘要
General	CANopen プロトコルに関するパラメータの設定ができる。
Log	デバッグ中にプロトコルマネージャーCANopen_Manager の動作ログを確認できる。
CANopen I/O Mapping	フィールドバスを制御する周期時間となるバスサイクルを設定できる。
CANopen IEC Objects	このタブ内で定義されているオブジェクト名称を使うことでユーザーアプリケーションから CANopen_Manager の一部情報にアクセスできる。
Status	デバッグ中に CANopen_Manager の動作良否を確認できる。
Information	プロトコルマネージャーCANopen_Manager のバージョン情報などを確認できる。

8.4.2.1. General タブ

このタブは、CANopen プロトコルにおけるパラメータを設定できます。各種ユーザーアプリケーション所望の値になるように設定して下さい。

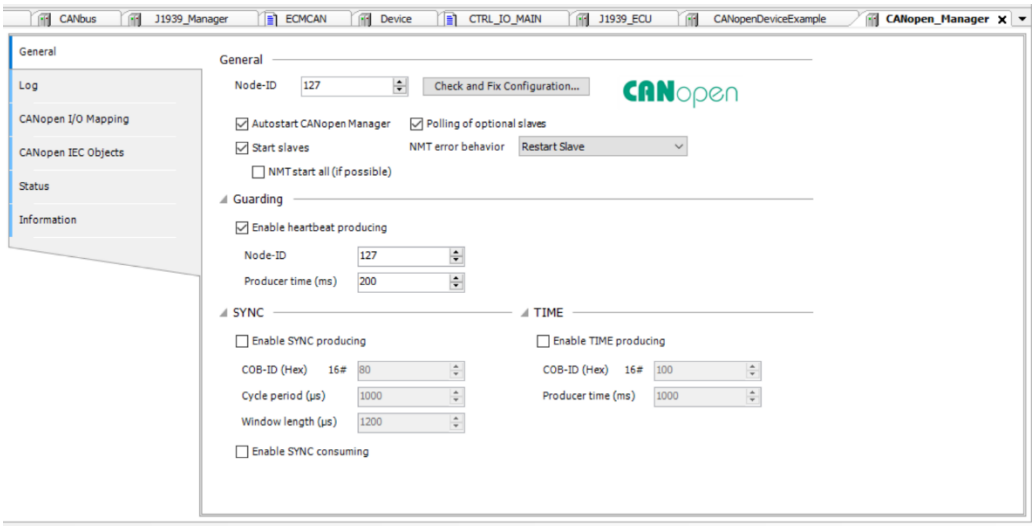


Figure 137 CANopen Manager General タブ

Table 64 CANopen_Manager General タブ設定項目

区分	名称	摘要
General	Node-ID	CANopen マスタデバイスとなる CX2500 のノード番号を入力する。
	Autostart CANopen Manager	**チェックを入れること。**チェックを入れることでプロトコルマネージャーが自動で起動する。
	Polling of optional slaves	チェックを入れると、スレーブデバイスが応答しない場合に 1 秒ごとにマスタからスレーブへ問い合わせる。
	Start slaves	**チェックを入れること。**チェックを入れると、プロトコルマネージャーが紐づけたスレーブデバイスの起動設定処理を自動でおこなう。
	NMT error behavior	NMT(Network Management Object)エラー時のスレーブデバイスの動作を選択タブから選択する。 Restart slave :NMT エラー時スレーブデバイスを再起動する。 Stop slave :NMT エラー時スレーブデバイスを停止する。
	NMT start all(if possible)	チェックを入れると、プロトコルマネージャーが NMT Start All コマンドを使用して全てのスレーブデバイスを開始する。
Guarding	Enable heartbeat producing	チェックを入れると、マスタデバイスが Producer time の間隔でハートビートを送信する。
	Node-ID	ハートビート用のノード ID を入力する。
	Producer time(ms)	ハートビート送信間隔[ms]を入力する。
SYNC	Enable SYNC producing	チェックを入れると、マスタデバイスが Cycle period の間隔で SYNC メッセージを送信する。
	COB-ID(Hex)	SYNC メッセージの ID を入力する。
	Cycle period(μs)	SYNC メッセージの送信間隔[μs]を入力する。
	Window length(μs)	同期 PDO のタイムフレーム長[μs]を入力する。
	Enable SYNC consuming	チェックを入れると、スレーブデバイスのいずれかが SYNC 生成をおこなう必要がある。
TIME	Enable TIME producing	チェックを入れると、プロトコルマネージャーは TIME メッセージを送信する。
	COB-ID(Hex)	TIME メッセージの ID を入力する。
	Producer time(ms)	TIME メッセージ送信間隔[ms]を入力する。

8.4.2.2. CANpen I/O Mapping タブ

このタブでは、フィールドバス内の送受信メッセージやステータス情報を制御する周期時間を設定して下さい。

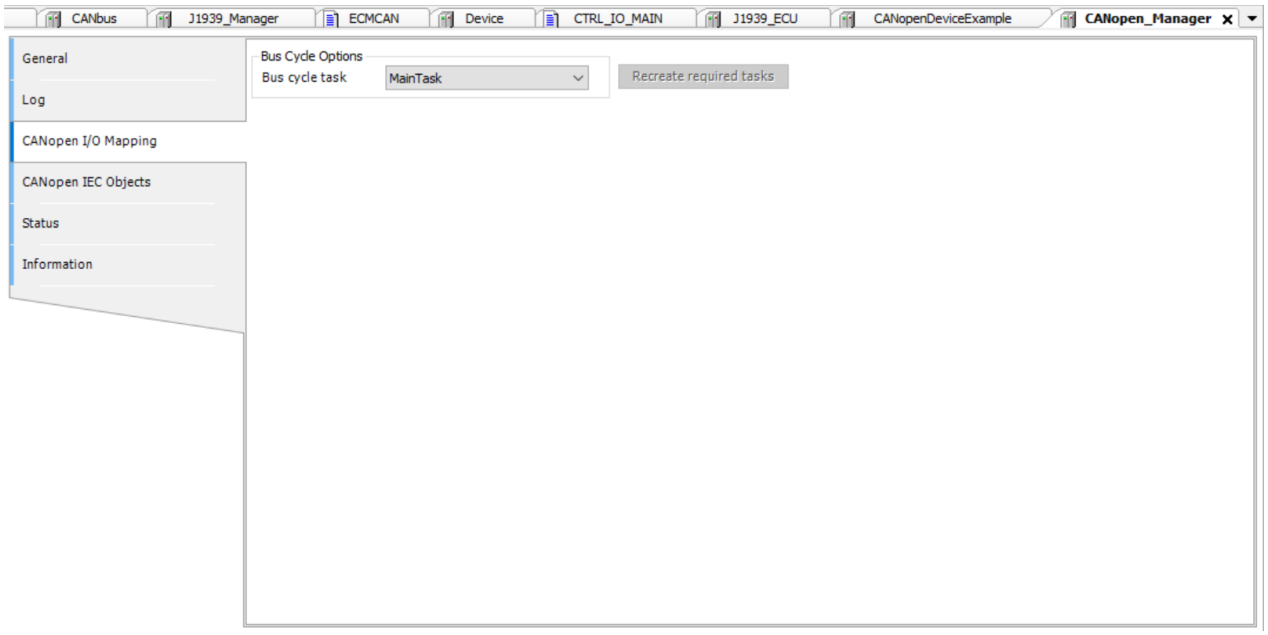


Figure 138 CANopen Manager CANopen I/O Mapping タブ

8.4.3. CAN デバイス

CANopen の CAN デバイスエディタ画面について説明します。CAN デバイスのエディタ画面は、エディタウィンドウにある CAN デバイスをダブルクリックすると表示されます。

CANopen の CAN デバイスは、EDS ファイルのデータやデバイスによって下記タブ名などが異なる場合があることに留意して下さい。

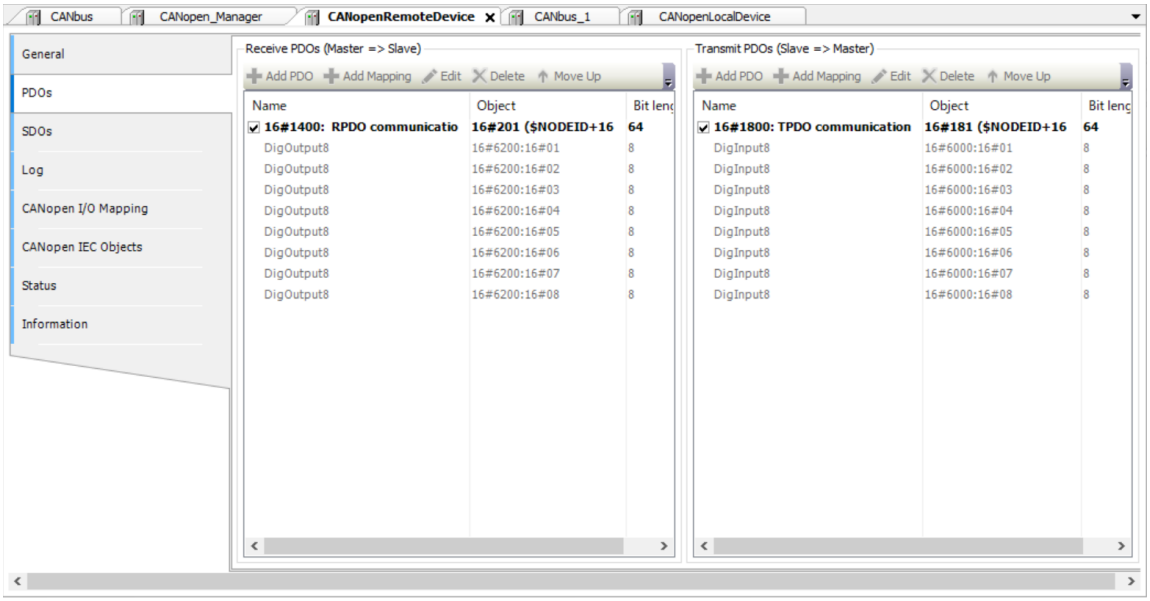


Figure 139 CAN デバイス(CANopen) エディタ画面

Table 65 CANopen CAN デバイス タブ一覧

タブ名称	摘要
General	CANopen プロトコルにおけるデバイスパラメータを設定できる。
PDOs	各デバイスの PDO 設定ができる。
SDOs 又は Object Dictionary	CX2500 がマスタデバイスの場合、CANopen_Manager に紐づけたスレーブデバイスは SDOs タブとなり、SDO 設定をおこなうことができる。 CX2500 がスレーブデバイスの場合、Object Dictionary タブとなり、スレーブデバイス(CX2500)の Object Dictionary 設定をおこなうことができる。
Log	デバッグ中に CAN デバイスの動作ログを確認できる。
CANopen I/O Mapping	上記 Object タブで定義したメッセージのデータに対して、ユーザーアプリケーションで使う変数を紐づけることができる。メッセージが登録されていないとこのタブは表示されない。
CANopen IEC Objects	このタブ内で定義されているオブジェクト名称を使うことでユーザーアプリケーションから CAN デバイスの一部情報にアクセスできる。
Status	CANopen プロトコルにおけるステータス情報を表示する。
Information	CAN デバイスのバージョン情報などを確認できる。

8.4.3.1. General タブ

このタブは、CANopen プロトコルにおけるデバイスパラメータを設定できます。このタブは、CX2500 が CANopen マスタかスレーブかによって設定画面及び項目が異なります。CODESYS にデフォルトでインストール(登録)されているスレーブデバイスとは別に、CANopen デバイス(EDS ファイル)を新しく CODESYS-IDE に追加する場合は、別紙「CX2500Codesys_UserManual_ForSetup」4.4 節の要領で EDS ファイルのインストール(登録)をおこなってください。

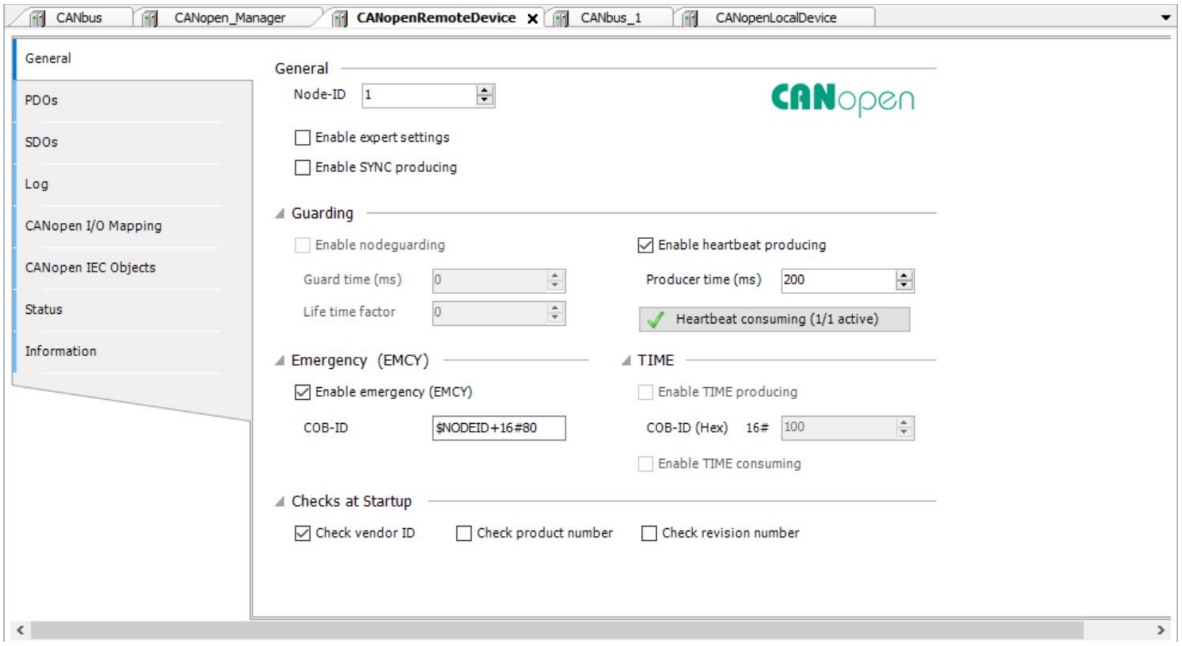


Figure 140 CAN デバイス(CANopen マスタ) General タブ

Table 66 CAN デバイス CANopen マスタ 設定項目(デフォルト時)

区分	名称	摘要
General	Node-ID	スレーブデバイスのノード ID を入力する。
	Enable expert settings	チェックを入れると、対象デバイスの EDS ファイル内に定義されている全設定が表示される。
	Enable SYNC producing	CANopen_Manager の Enable SYNC progucing が無効化されている時、チェックを入れると対象のスレーブデバイスで SYNC 生成をが有効になる。
Guarding	Enable nodeguarding	チェックを入れると、スレーブデバイスが応答しない場合にマスタデバイスはノードガードメッセージを送信する。
	Guard time(ms)	ノードガードメッセージの送信周期[ms]を入力する。
	Life time factor	一定時間(Guard time×Life time factor)スレーブデバイスが応答しない場合、デバイスはノードガードエラーを検知する。
	Enable heartbeat producing	チェックを入れると、デバイスは Producer time で設定した周期でハートビートメッセージを送信する。
	Producer time(ms)	ハートビートメッセージの送信周期[ms]を入力する。
	Heartbeat consuming	ボタンを押すとハートビート監視するスレーブデバイスを設定できる。
EMCY	Enable Emergency	チェックを入れると、デバイスで内部エラーが発生した場合に EMCY メッセージを送信する。
	COB-ID	EMCY メッセージの ID を入力する。
TIME	Enable TIME producing	チェックを入れると、デバイスは TIME メッセージを送信する。
	COB-ID(Hex)	TIME メッセージの ID を入力する。
	Enable TIME consuming	チェックを入れると、デバイスは TIME メッセージの処理をおこなう。
Checks at Startup	Check Vendor ID	デバイス起動時にベンダーID を EDS ファイルの設定と合っているか確認する。
	Check product number	デバイス起動時に製品番号が EDS ファイルの設定と合っているか確認する。
	Check revision number	デバイス起動時にリビジョン番号が EDS ファイルの設定と合っているか確認する。

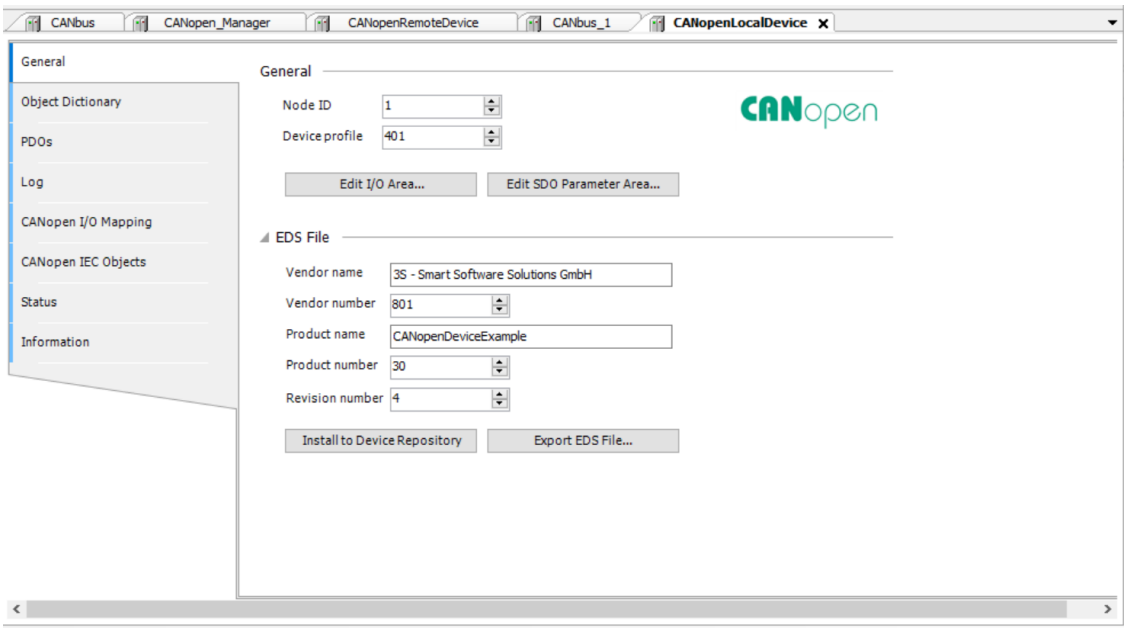


Figure 141 CAN デバイス(CANopen スレーブ) General タブ

Table 67 CAN デバイス CANopen スレーブ 設定項目

区分	名称	摘要
General	Node ID	スレーブデバイスのノード ID を入力する。
	Device profile	デバイスのプロファイル番号を入力する。
	Edit I/O Area..	ボタンを押すと、IO 領域の編集をおこなうことができる。編集ウィンドウについては CODESYS オンラインヘルプを参照。
	Edit SDO Parameter Area..	ボタンを押すと、SDO パラメータの編集をおこなうことができる。編集ウィンドウについては CODESYS オンラインヘルプを参照。
EDS File	Vendor name	ベンダー名
	Vendor number	ベンダー番号
	Product name	製品名
	Product number	製品番号
	Revision number	リビジョン番号
	Install to Device Repository	ボタンを押すと、EDS ファイルをインポートし、CAN デバイスに EDS ファイルの設定を反映することができる。
	Export EDS File..	ボタンを押すと、エディタ画面で設定した情報を元に EDS ファイルを生成・エクスポートする。

8.4.3.2. PDOs タブ

このタブでは、CAN デバイスの PDO 設定をおこなうことができます。

CX2500 がマスタデバイスの時、CANopen_Manager に紐づけたスレーブデバイスの PDOs タブは以下の通りになります。PDOs タブには、EDS ファイルから読み取られた PDO メッセージが自動追加されて一覧表示されます。この場合の PDS タブでは PDO メッセージの ID や送受信の種類について編集をおこなうことができます。編集は所望のメッセージを選択した上で「Edit」ボタンを押して下さい。すると、PDO Properties ウィンドウが表示されるので、所望の値を設定し「OK」ボタンを押して下さい。

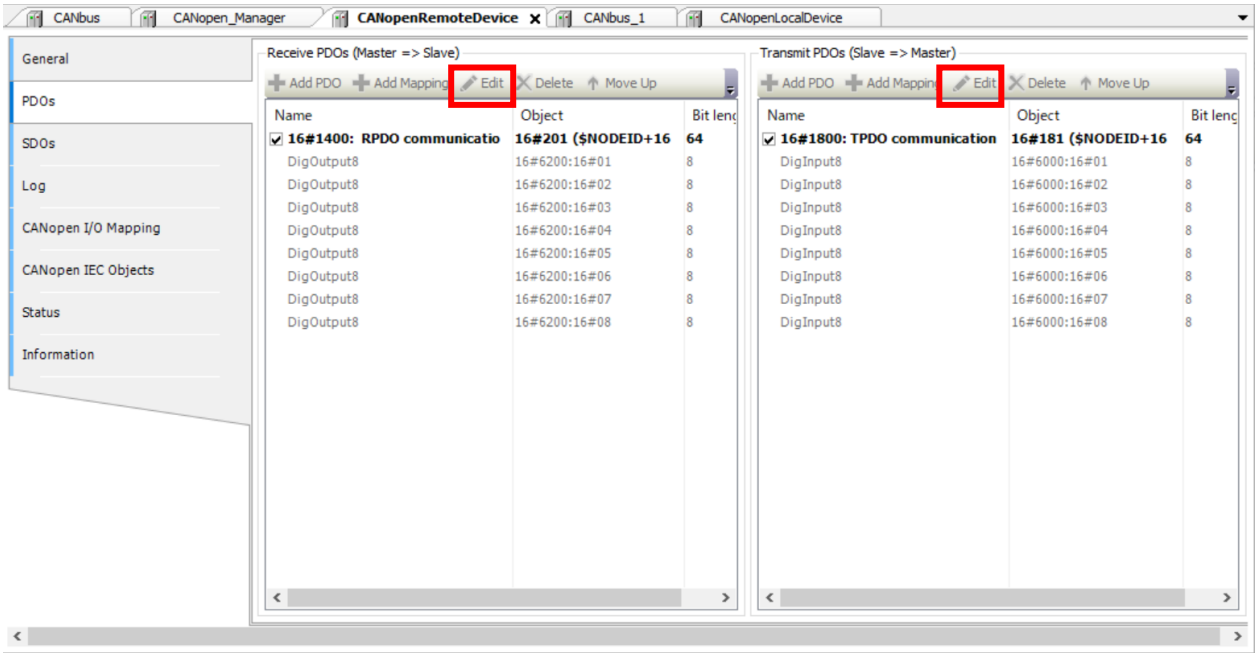


Figure 142 CAN デバイス(CANopen) リモートデバイス PDOs タブ

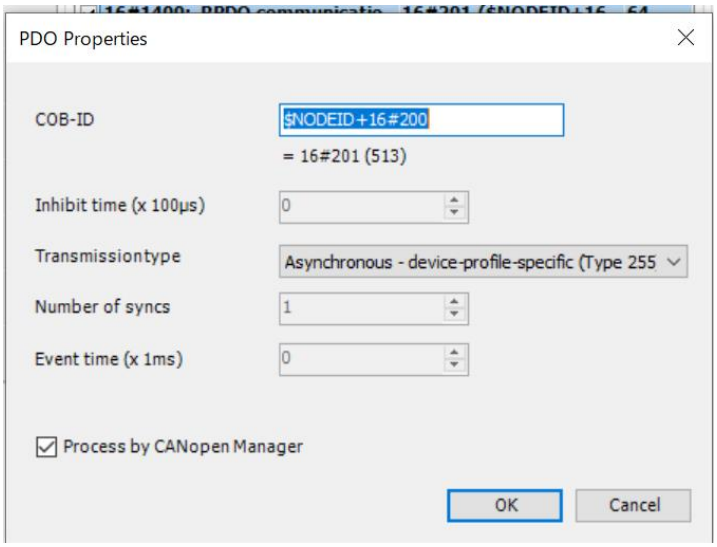


Figure 143 リモートデバイス PDO Properties ウィンドウ

Table 68 リモートデバイス PDO Properties ウィンドウ 設定項目

<table>
<tr><th>項目</th><th>摘要</th></tr>
<tr><td>COB-ID</td><td>対象の PDO メッセージの ID を設定する。</td></tr>
<tr><td>RTR</td><td>送信 PDO メッセージのみ表示される。チェックを入れると、リモートフレームを使用して PDO 問い合わせをおこなう。</td></tr>
<tr><td>Inhibit time(×100 μs)</td><td>非対応</td></tr>
<tr><td>Transmissiontype</td><td>PDO の送信方法を選択タブから設定する。
<table>
<tr><th>Transmissiontype</th><th>PDO メッセージの送信法</th></tr>
<tr><td>Acyclic-synchronous</td><td>同期送信</td></tr>
<tr><td>Cyclic-synchronous</td><td>バスサイクル×Number if syncs の周期で送信</td></tr>
<tr><td>Synchronous-only RTR</td><td>送信 PDO メッセージのみ。リモート送信要求時のみ送信(同期)。</td></tr>
<tr><td>Asynchronous-only RTR</td><td>送信 PDO メッセージのみ。リモート送信要求時のみ送信(非同期)。</td></tr>
<tr><td>Asynchronous-manufacturer specific</td><td>非対応</td></tr>
<tr><td>Asynchronous-device profile</td><td>CiA デバイスプロファイルに従って送信。</td></tr>
</table></td></tr>
<tr><td>Number if syncs</td><td>Transmissiontype が Cyclic-synchronous 設定時のみ有効化される。PDO の送信間隔(バスサイクル時間×Number if syncs)を入力する。</td></tr>
<tr><td>Event time(×1ms)</td><td>非対応</td></tr>
<tr><td>Process by CANopen Manager</td><td>チェックを入れること。チェックを外すと、CANopen_Manager(マスタデバイス)が PDO の処理をおこなわないため、送受信もおこなわれない。</td></tr>
</table>

CX2500 がローカルデバイス(CANopen スレーブ)の時の PDOs タブは以下の通りとなります。この場合の PDOs タブでは主に Table 69 の操作ができます。

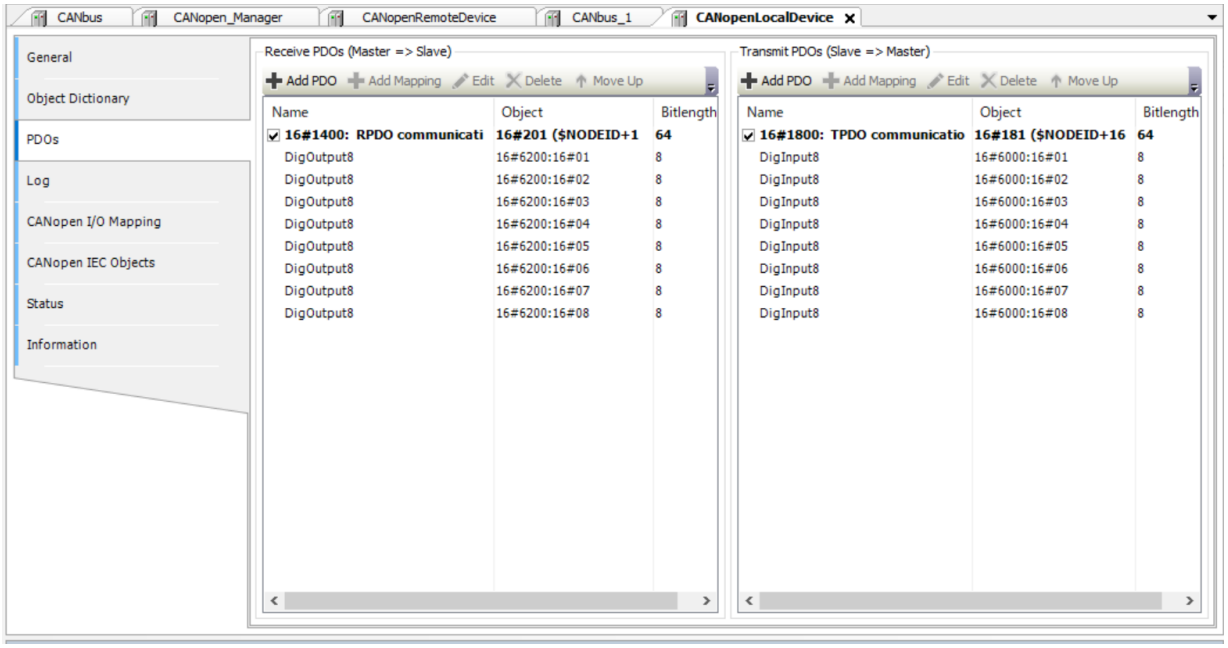


Figure 144 CAN デバイス(CANopen) ローカルデバイス PDOs タブ

Table 69 ローカルデバイス PDOs タブ 主な操作

操作名	対応するボタン名称	編集ウィンドウ
PDO メッセージの追加	Add PDO	Figure 145
PDO メッセージ編集	Edit	Figure 146
PDO メッセージデータの追加	Add Mapping	Figure 147
PDO メッセージデータの編集	Edit	Figure 147

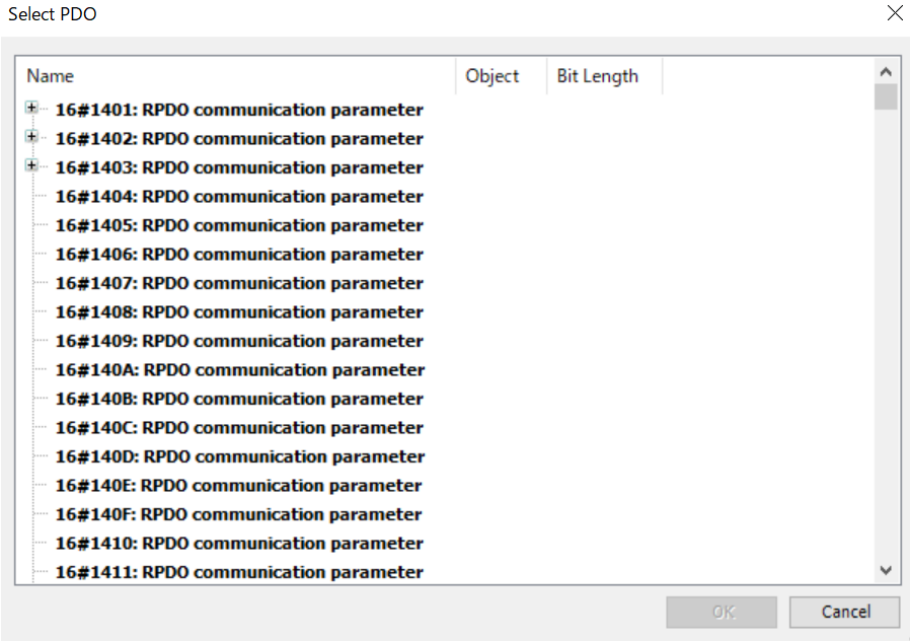


Figure 145 Select PDO ウィンドウ(PDO メッセージの追加)

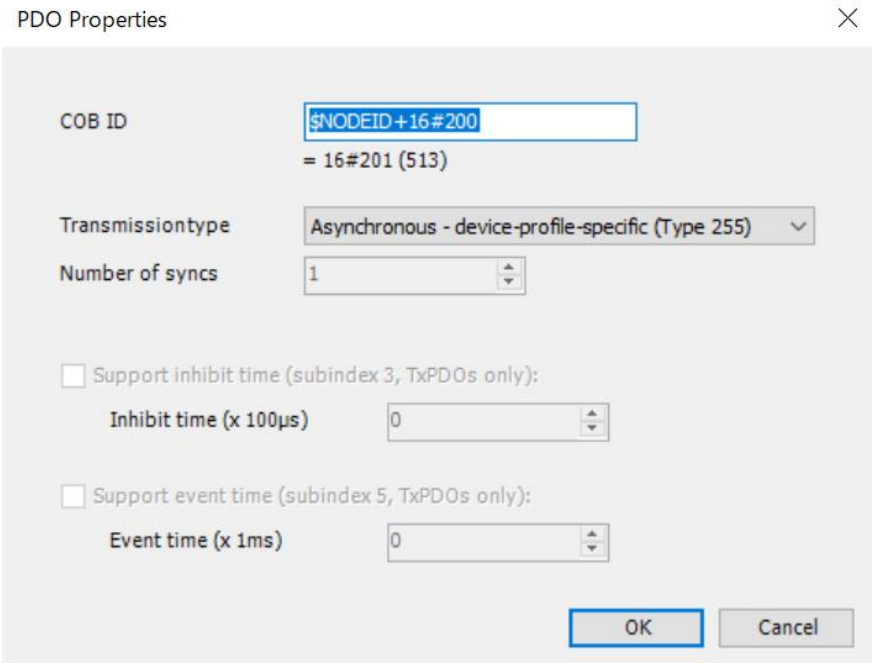


Figure 146 PDO Properties ウィンドウ(PDO メッセージの編集)

Table 70 ローカルデバイス PDO Properties ウィンドウ設定項目

項目	摘要
COB-ID	対象の PDO メッセージの ID を設定する。
Transmissiontype	PDO の送信方法を選択タブから設定する。

Transmissiontype	PDO メッセージの送信法
Acyclic-synchronous	同期送信
Cyclic-synchronous	バスサイクル×Number if syncs の周期で送信
Synchronous-only RTR	送信 PDO メッセージのみ。リモート送信要求時のみ送信(同期)。
Asynchronous-only RTR	送信 PDO メッセージのみ。リモート送信要求時のみ送信(非同期)。
Asynchronous-manufacturer specific	**非対応**
Asynchronous-device profile	CiA デバイスプロファイルに従って送信。

Support inhibit time	**非対応**
Support event time	**非対応**

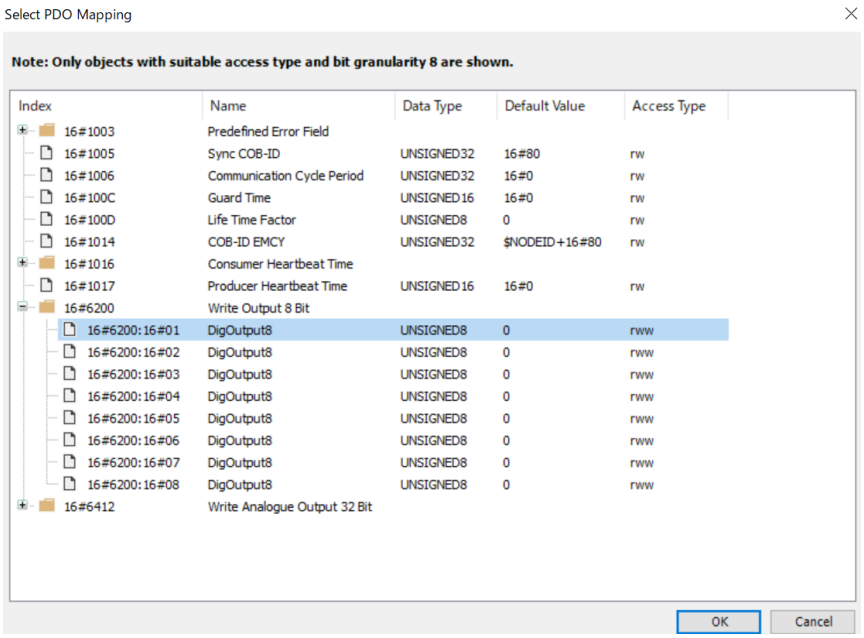


Figure 147 Select PDO Mapping ウィンドウ(PDO メッセージデータの追加)

8.4.3.3. SDOs/Object Dictionary タブ

CX2500 がマスタデバイスの時、CANopen_Manager に紐づけたスレーブデバイスの SDOs タブは以下の通りになります。SDOs タブには、EDS ファイルから読み取られた SDO メッセージが自動追加されて一覧表示されます。SDOs タブでは、SDO のオブジェクト追加や送信順序(図中の Move Up/Move Down ボタン)を設定することができます。なお、オブジェクト追加は「Add SDO」ボタンを押すと、「Select Item from Object Directory」ウィンドウが開くので、EDS ファイルで登録され表示されているオブジェクトを選択して「OK」ボタンを押すことで追加されます。

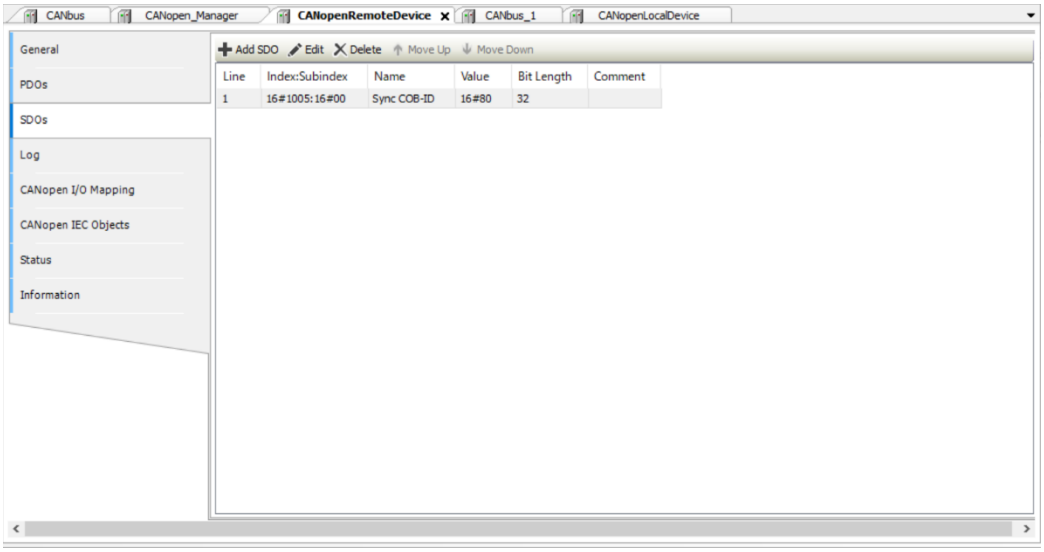


Figure 148 リモートデバイス SDOs タブ

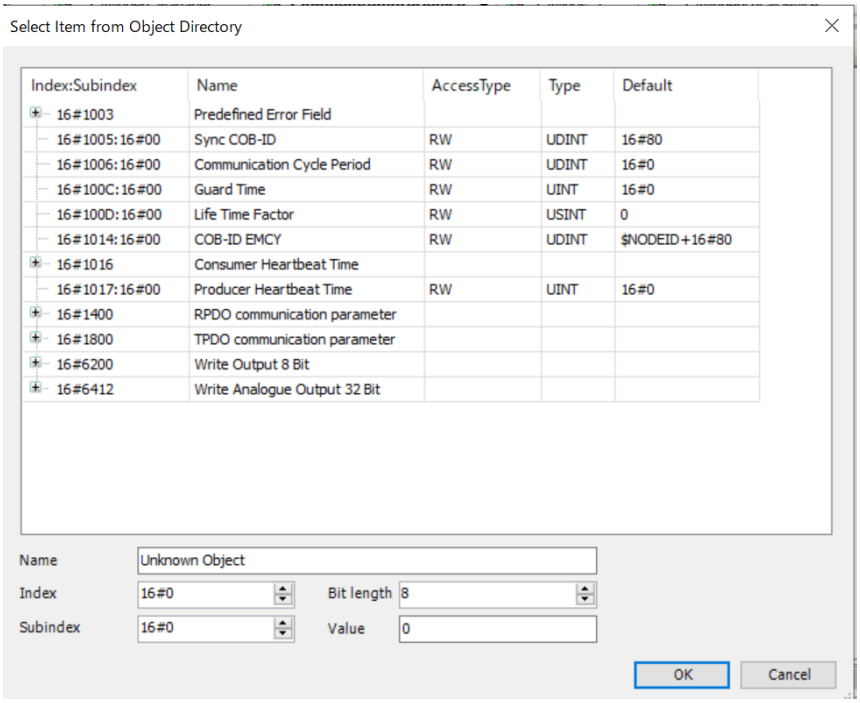


Figure 149 Select Item from Object Directory ウィンドウ

CX2500 がローカルデバイス(CANopen スレーブ)の時、CAN デバイスには Object Dictionary タブが表示されます。Object Dictionary タブには、EDS ファイルから読み取られたオブジェクトが自動追加されて一覧表示されます。

Object Dictionary タブでは、新しいオブジェクトの追加・編集をおこなうことができます。新しいオブジェクト追加の際は、「Add Object」ボタンを押すと「Add Object」ウィンドウが開くので、所望のオブジェクトを選択して「Add Object」ボタンを押して下さい。

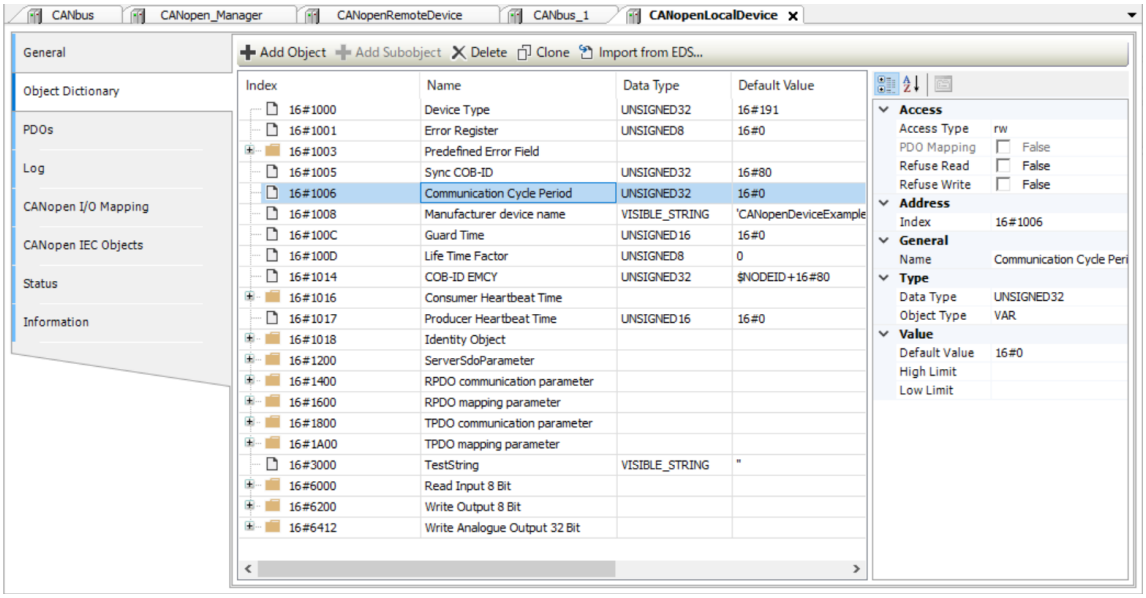


Figure 150 ローカルデバイス Object Dictionary タブ

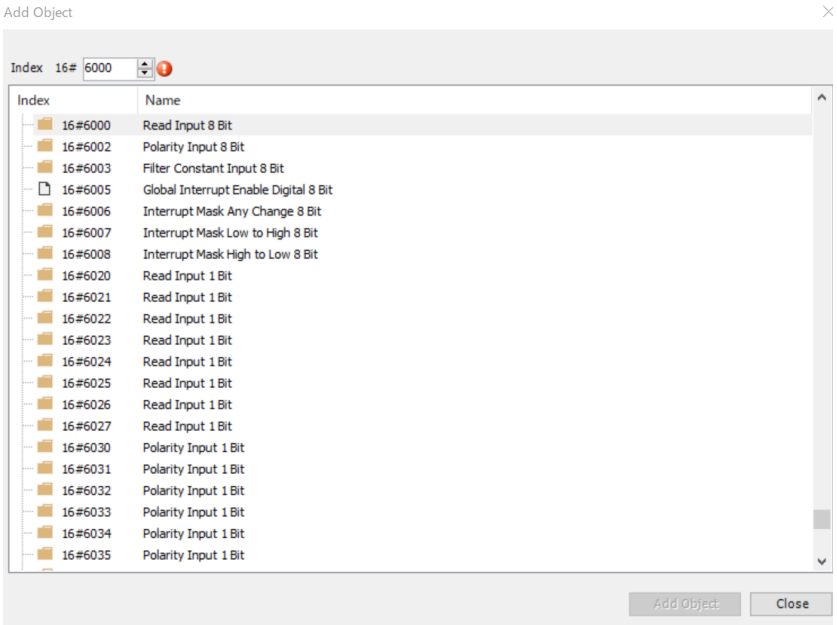


Figure 151 Object Dictionary タブ Add Object ウィンドウ

8.4.3.4. CANopen I/O Mapping タブ

このタブは、PDOs・SDOs タブでメッセージを登録した場合に表示されます。各メッセージのデータが一覧表示されており、IO ドライバと同様にこれらに変数を紐づけてユーザーアプリケーションで使用することができます。

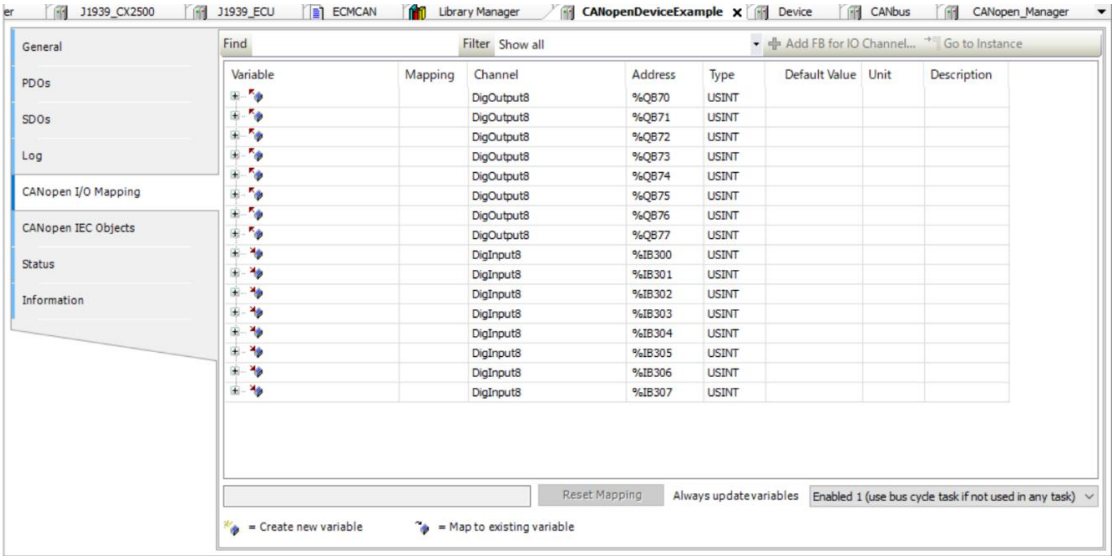


Figure 152 CAN デバイス(CANopen) CANopen I/O Mapping タブ

8.4.3.5. Status タブ

このタブは CANopen プロトコルにおけるステータス情報を表示します。表示されるステータス情報については、CANopen プロトコルの規格書などを参照して下さい。

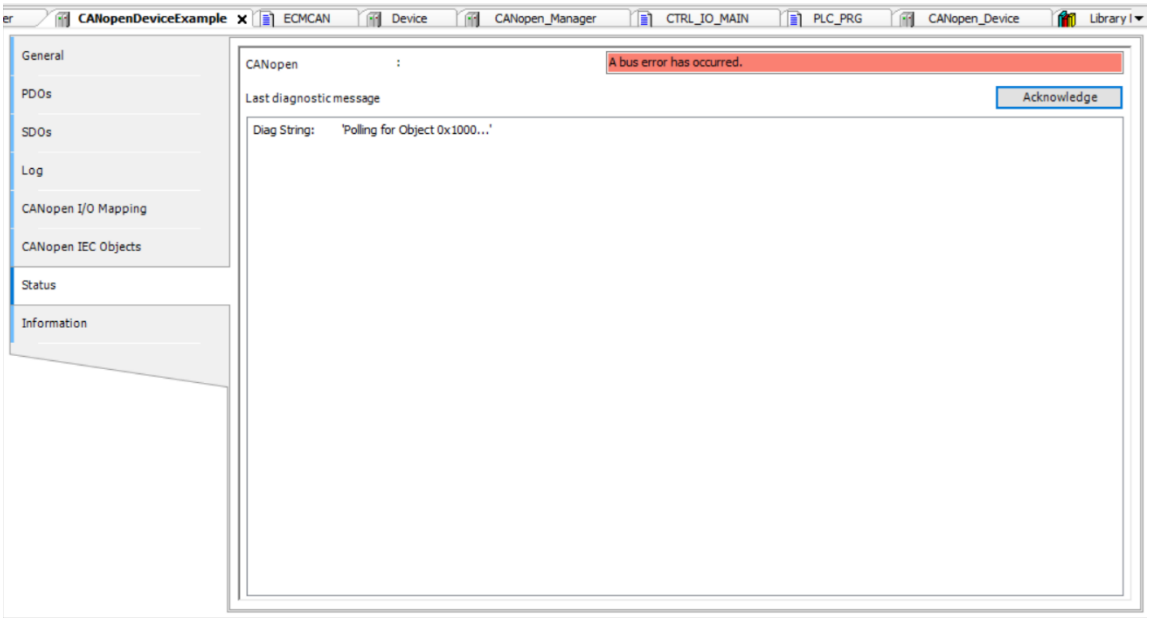


Figure 153 CAN デバイス(CANopen) Status タブ

9. デバッグ機能について

9.1. 基本画面

ログイン(6.11.1 項)完了後、CODESYS-IDE はデバッグモード(ログイン中と同義)へ遷移します。下図は、遷移後のデバッグ画面の一例です。

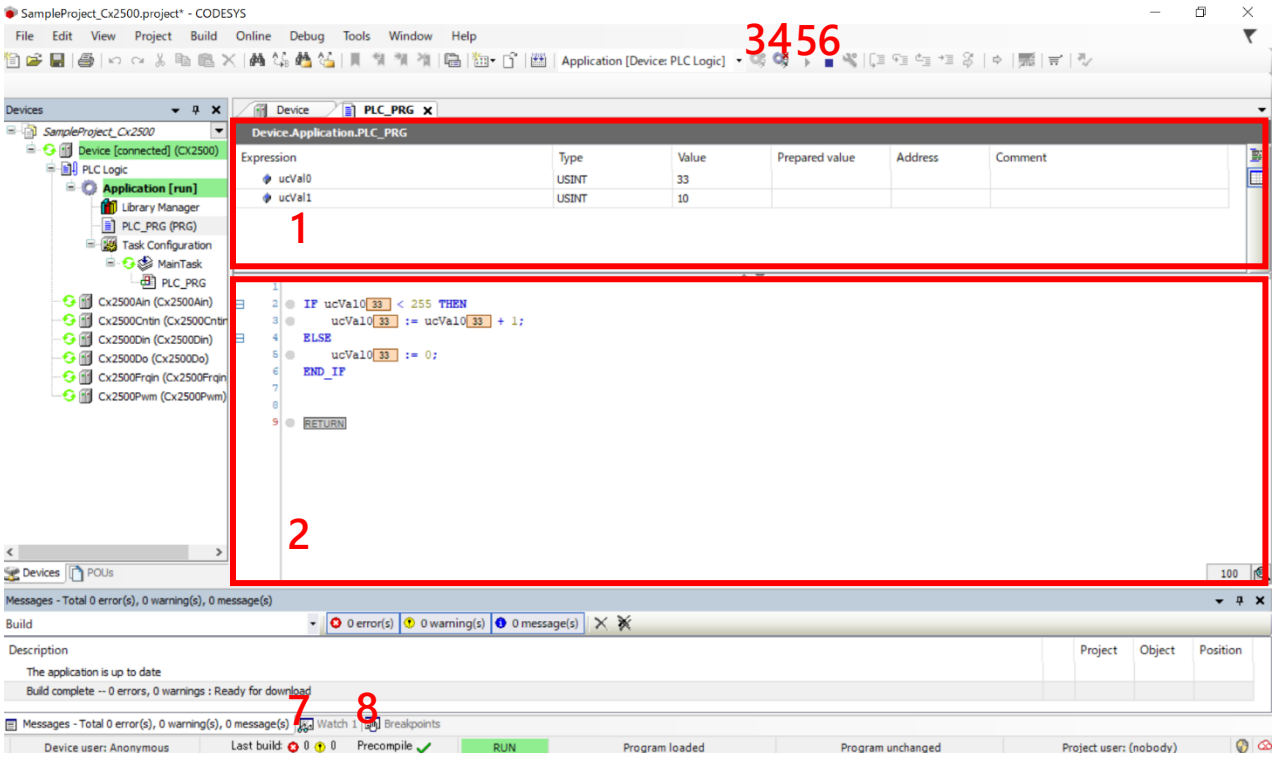


Figure 154 デバッグモード メイン画面

Table 71 デバッグモード メイン画面 機能概要

#	機能名	摘要
1	Variables Monitor	このエリアでは、表示しているソースファイルで定義した変数値をモニタしたり、強制的に値を書き込むことができる。
2	Code Monitor	このエリアでは、表示しているソースファイルで記述したコード及び現在の変数値をモニタできる。ブレークポイントなども設定可能。
3	Login	アイコンを押すとアプリケーションを書き込み、デバッグを開始する。
4	Logout	アイコンを押すとデバッグを終了する。
5	Start	アイコンを押すとアプリケーションは動作を開始する。
6	Stop	アイコンを押すとアプリケーションは動作を停止する。
7	Watch	デバッグ中に監視したい変数とその値等を確認できる。
8	Breakpoints	アプリケーション上に設定したブレークポイントの情報を確認できる。

9.2. アプリケーションの動作開始・停止

アプリケーションの動作開始・停止は下記の手順でおこなうことができます。

9.2.1. 動作開始

下記にデバッグモードにおけるアプリケーションの動作開始の手順を示します。

① 下図の通り、デバッグモードでアプリケーションが停止していることを確認して下さい。

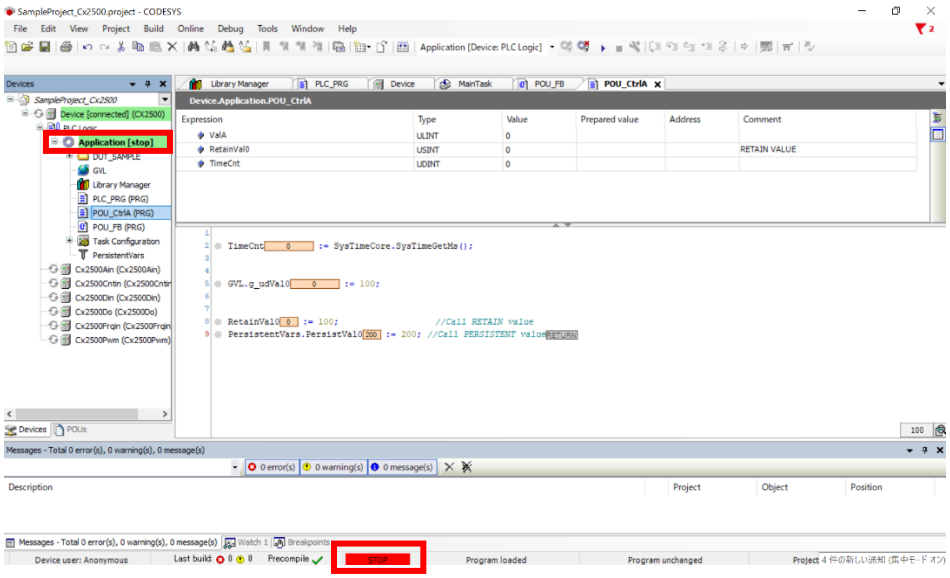


Figure 155 デバッグモード 動作停止状態

② ツールバーの「 」アイコン(運転開始)を押して下さい。

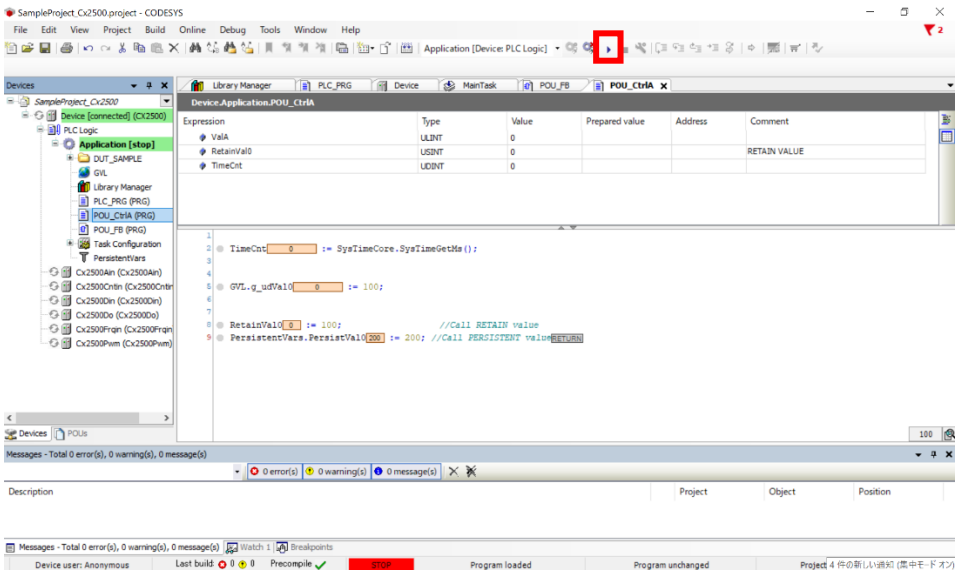


Figure 156 デバッグモード 運転開始アイコンの位置

③ 画面の動作ステータスが「RUN」となりアプリケーションの動作が開始されます。

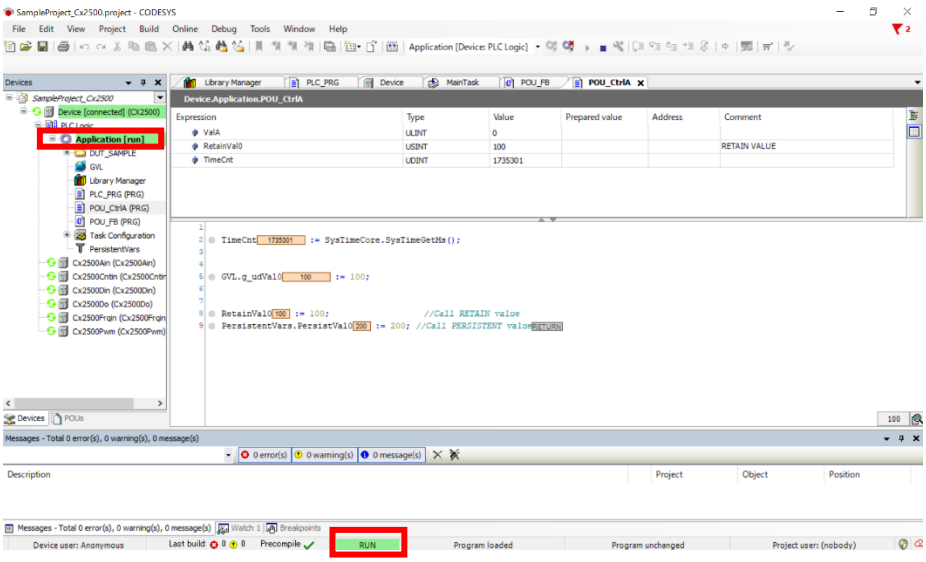


Figure 157 デバッグモード 動作中状態

9.2.2. 動作停止

下記にデバッグモードにおけるアプリケーションの動作停止の手順を示します。

① 下図の通り、デバッグモードでアプリケーションが動作中であることを確認して下さい。

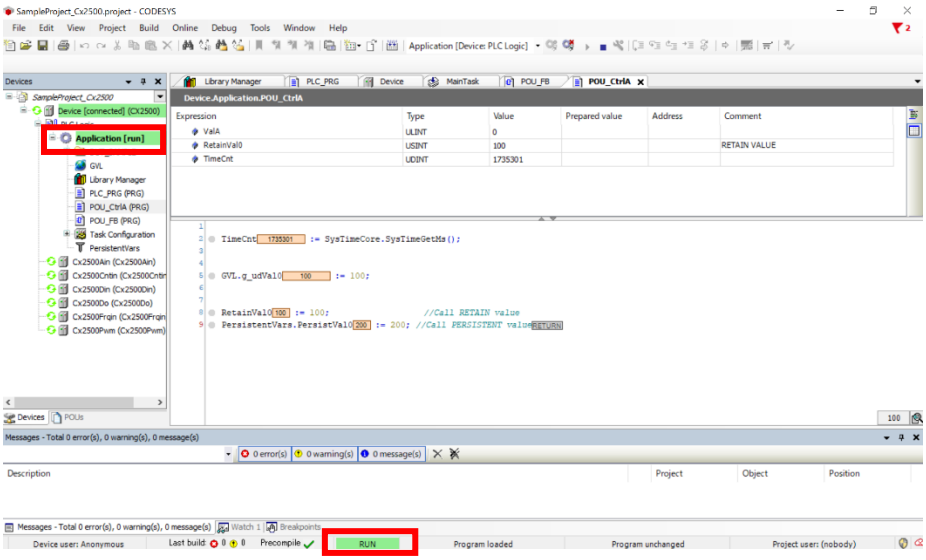


Figure 158 デバッグモード 動作中状態

② 「■」アイコン(運転停止)を押して下さい。

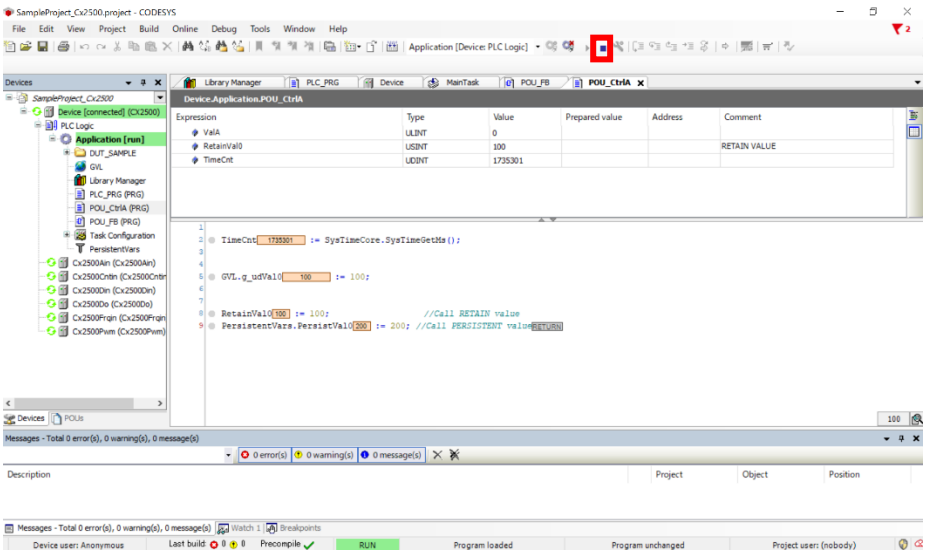


Figure 159 デバッグモード 運転停止アイコンの位置

③ 画面の動作ステータスが「STOP」となりアプリケーションの動作が停止します。

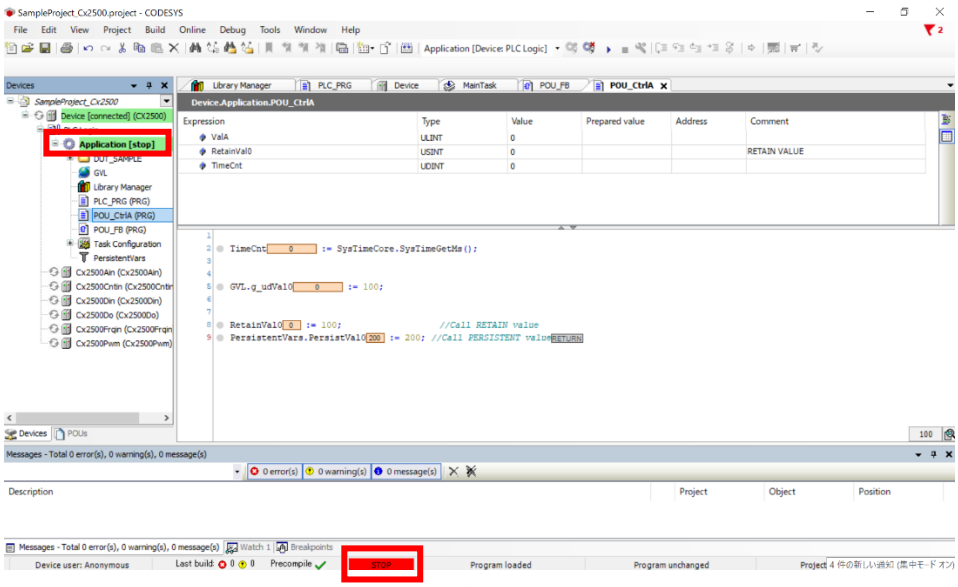


Figure 160 デバッグモード 動作停止状態

9.3. シングルサイクル

CODESYS-IDE では、アプリケーションを 1 サイクルだけ実行する機能があります。

デバッグモード中にメニューバーの「Debug」→「Single Cycle」を選択すると、各タスクを 1 サイクル実行しアプリケーションが停止します。

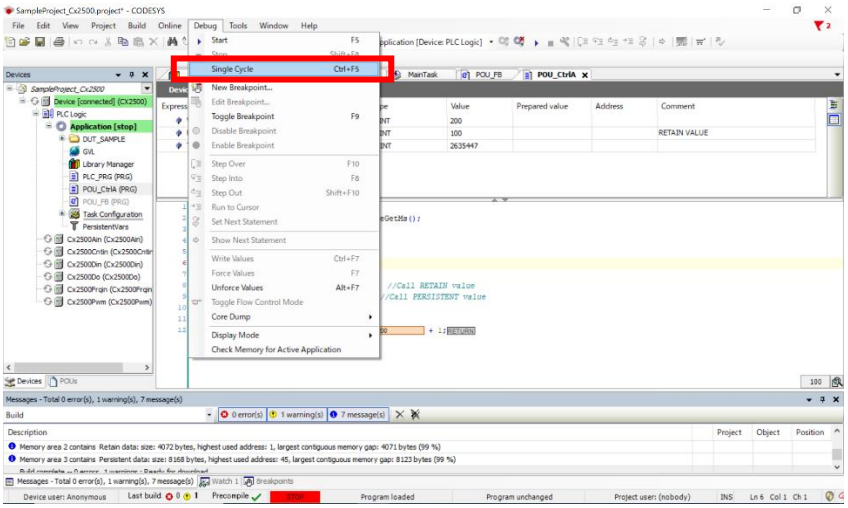


Figure 161 シングルサイクルの選択

9.4. 変数モニタ

デバッグモードでは、各 POU の Variable Monitor(変数宣言部)や Code Monitor(コード部)、グローバル変数リスト画面で変数の現在値が表示されます。下図は、その一例として ST の画面を挙げます。LD や FBD の場合は、実行中(TRUE)のアイテム(接点など)・接続ラインが黒→青色に変わります。

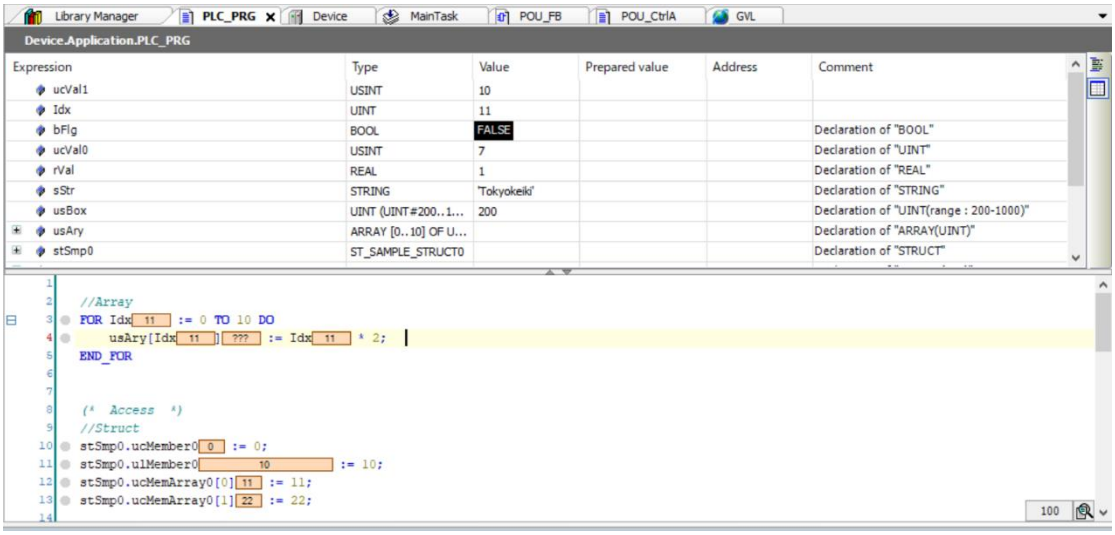


Figure 162 デバッグモード POU 画面

9.4.1. 変数値表記方式の変更

変数モニタでは値の表記形式(2・10・16 進数)を変更できます。ここでは、2 進数→16 進数への変更を例に手順を示します。

① メニューバーから、「Debug」→「Display Mode」を選択すると、表記方式の候補が表示されるので所望のものを選択して下さい。下図の✓印は、現在の表記方式が Decimal(10 進数)であることを意味します。

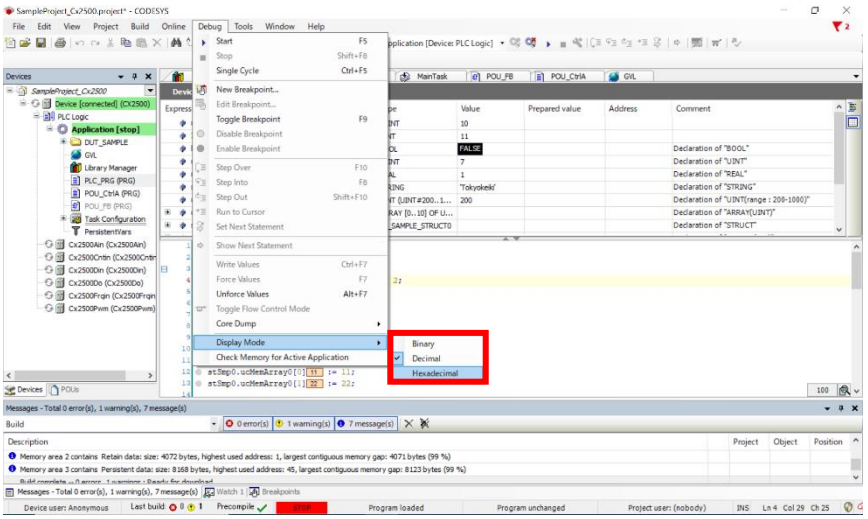


Figure 163 Display Mode の選択

② 選択後、表記が変更されます。

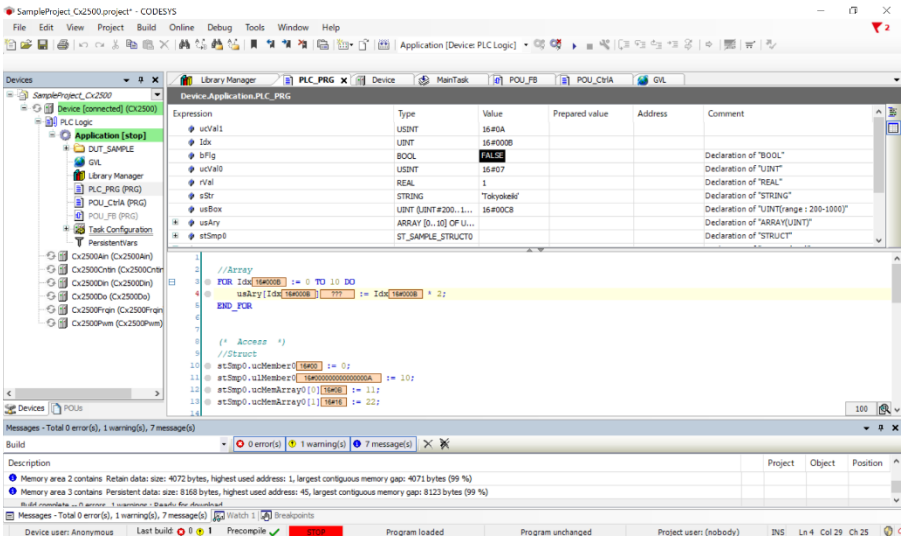


Figure 164 変数モニタ 16 進数表記

9.4.2. ウォッチ

通常、デバッグモード時の変数値は 9.4 節の通り POU や変数リストでモニタできます。ただ、別々の POU にある変数などを同時に見ることが難しい場合があります。その場合はウォッチリストにモニタしたい変数を登録することで実現できます。ウォッチリストは最大 4 つ利用できます。

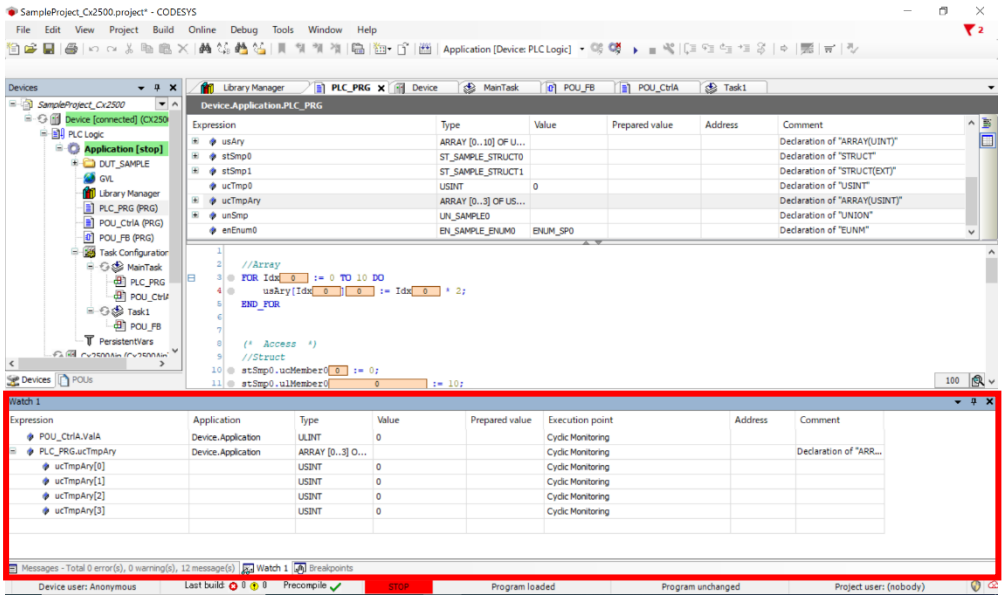


Figure 165 ウォッチリストの位置

メイン画面に表示されていない場合は、メニューバーの「View」→「Watch」からウォッチリストを選択して下さい。

9.4.2.1. 変数の登録

ウォッチリストへ変数を登録する手順を示します。

① デバッグモードにてウォッチリストに登録したい変数にカーソルを合わせ、右クリックして下さい。

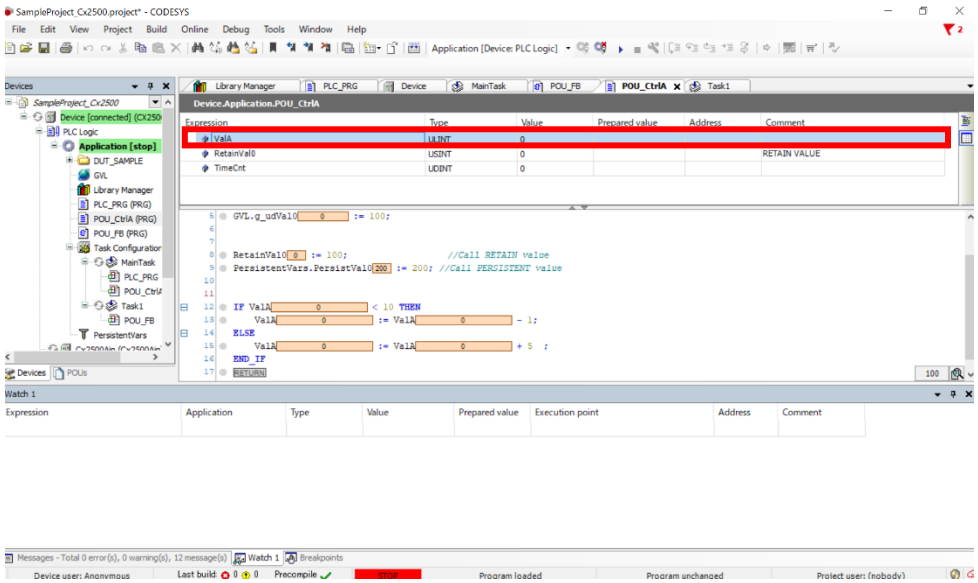


Figure 166 リストに登録したい変数の選択

② 表示されるコンテキストメニューから「Add to Watchlist」を選択して下さい。

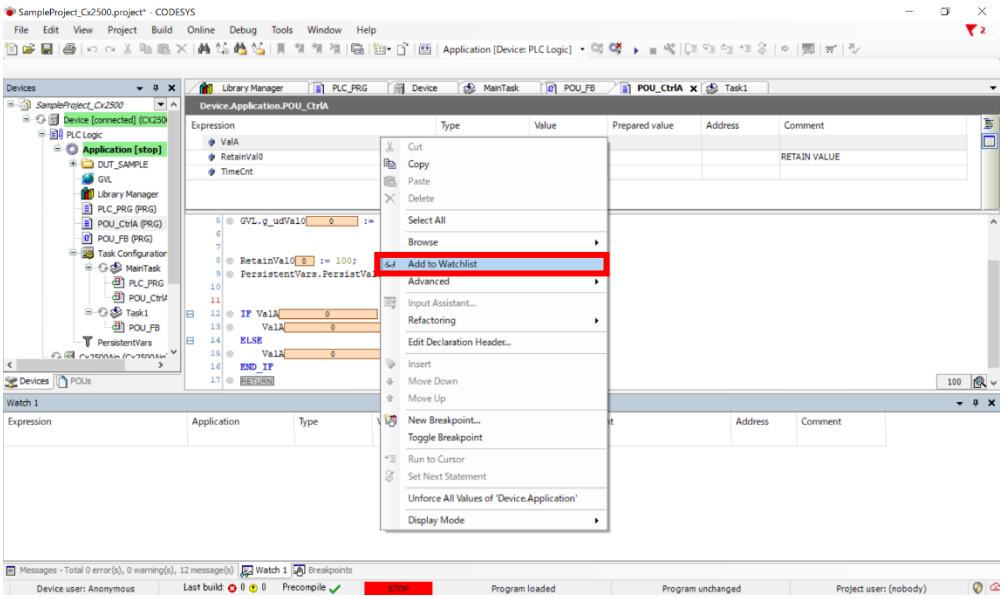


Figure 167 Add Watchlist の選択

③ ウォッチリストに所望の変数が登録されます。

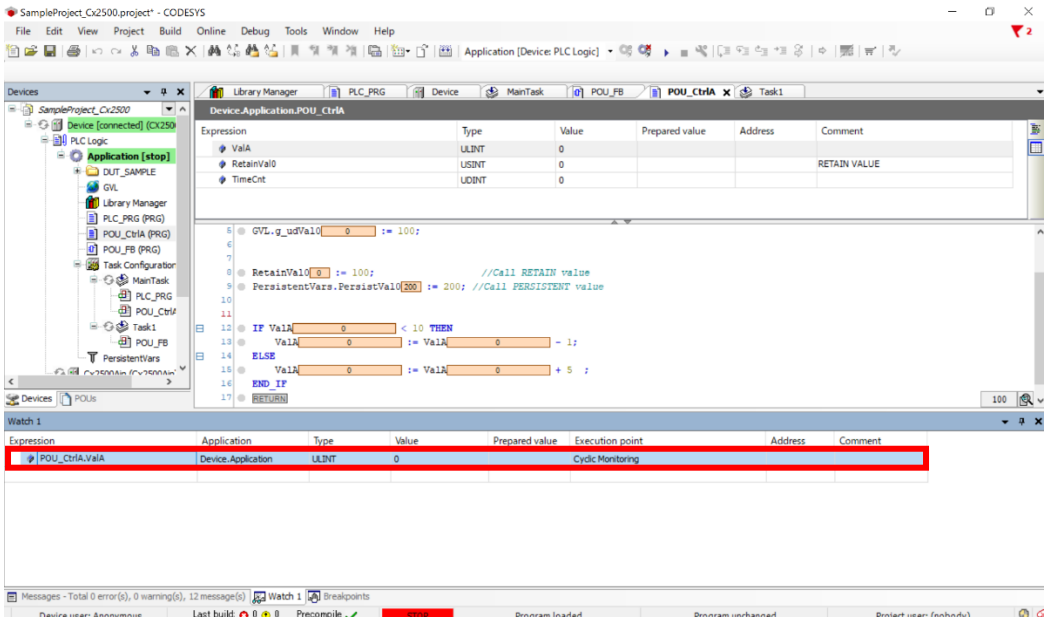


Figure 168 変数登録後

9.4.2.2. 変数の削除

ウォッチリストから変数を削除する手順を示します。

① デバッグモードにてウォッチリストから削除したい変数にカーソルを合わせ、右クリックして下さい。

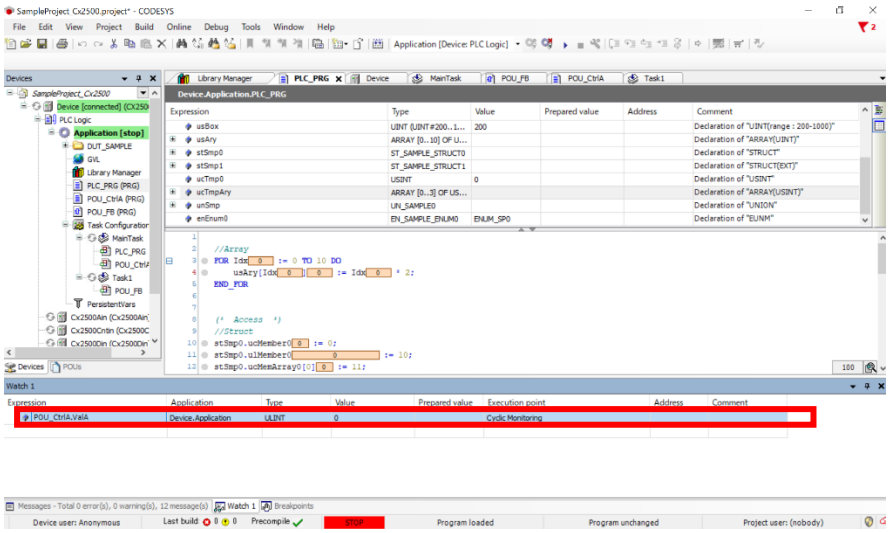


Figure 169 リストから削除したい変数の選択

② 表示されるコンテキストメニューから「Delete」を選択して下さい。

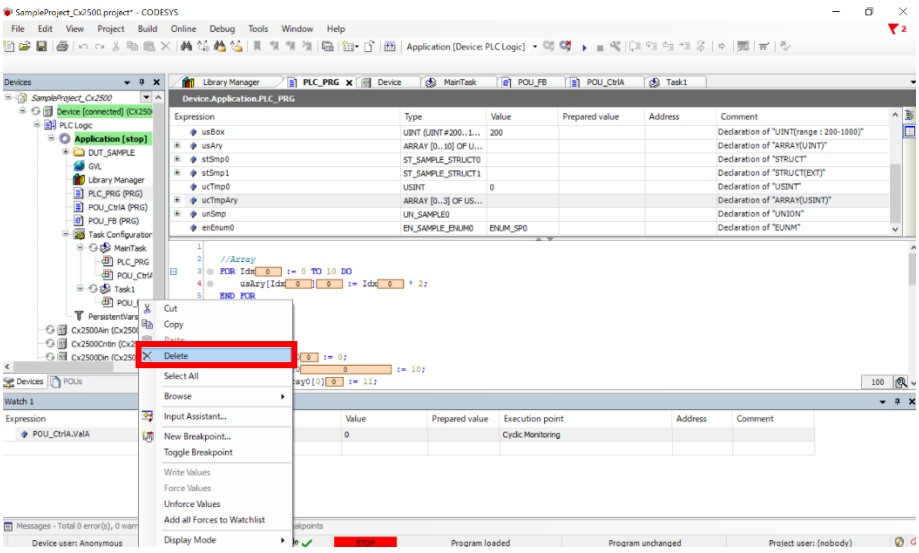


Figure 170 Delete の選択

③ これで変数がウォッチリストから削除されます。

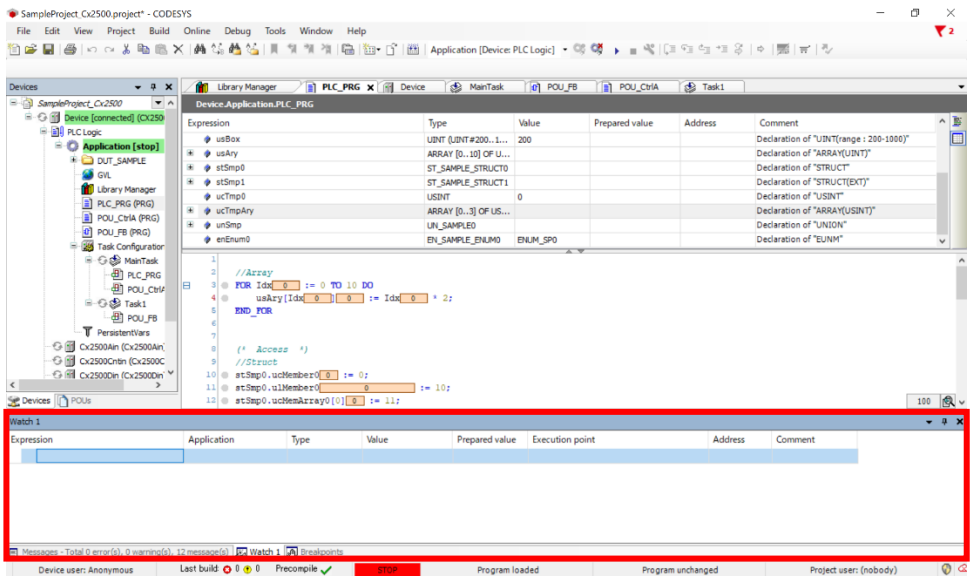


Figure 171 変数削除後

9.5. 値の書き込み・強制

ユーザーは、デバッグモード中に変数値を任意で設定できます。デバッグモード中の値の設定方法としては以下 2 通りの手法があります。値の設定は、実機の動作に重大な影響を与える可能性があります。設定の際は、必ずアプリケーション動作に与える影響を念入りに考慮・検討を重ね、問題無いと判断した上で実施して下さい。

9.5.1～9.5.3 項に変数宣言部での設定・解除手順を記載しております。値の書き込み・強制はウォッチリストでも同様に設定可能です。

Table 72 デバッグモード中の変数値設定方法

名称	摘要
値の書き込み	アプリケーション動作開始時若しくは動作中に値の書き込みをした時に 1 度だけ対象の変数に値を設定します。この方法の場合、次のサイクルやアプリケーション動作によっては、変数値が上書きされる可能性があります。
値の強制	毎サイクル変数に値を書き込みます。これにより、変数の値を強制解除(9.5.3 項参照)するまで保持し続けます。

9.5.1. 値の書き込み

ここでは、変数への値の書き込み手順を示します。

① 書き込みたい変数の変数宣言部にて Prepared value 列に所望の値を入力します。

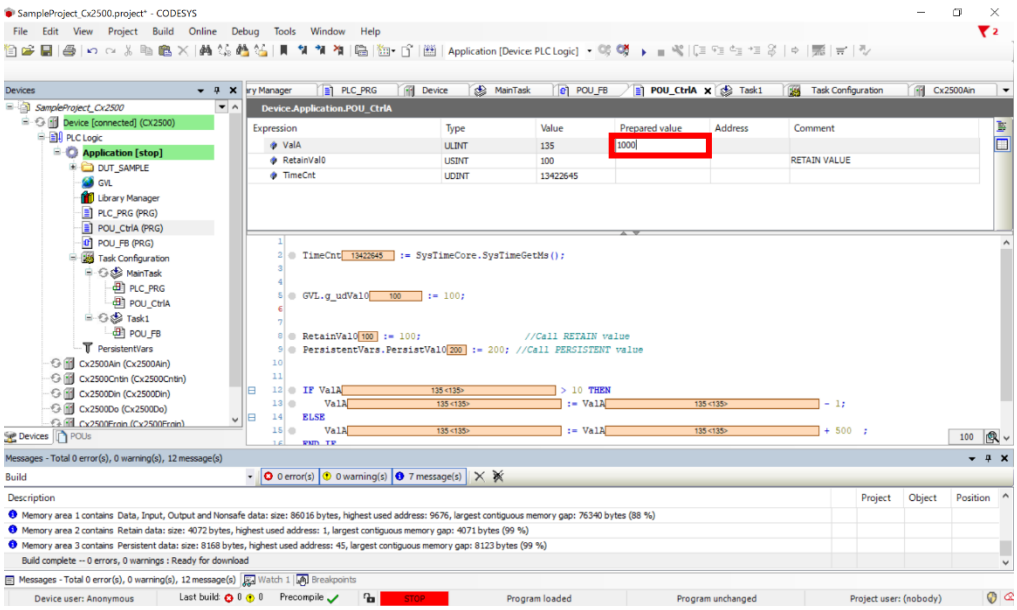


Figure 172 値の書き込み Prepare value への値の入力

② メニューバーから「Debug」→「Write Values」を選択して下さい。

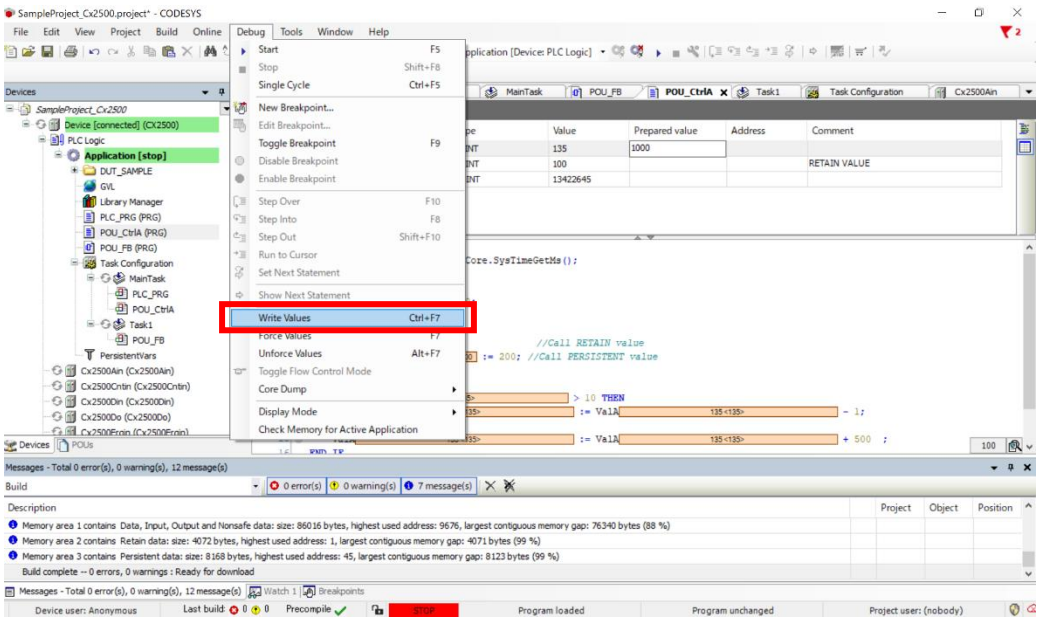


Figure 173 Write Values の選択

③ 所望の変数へ値が書き込まれます。

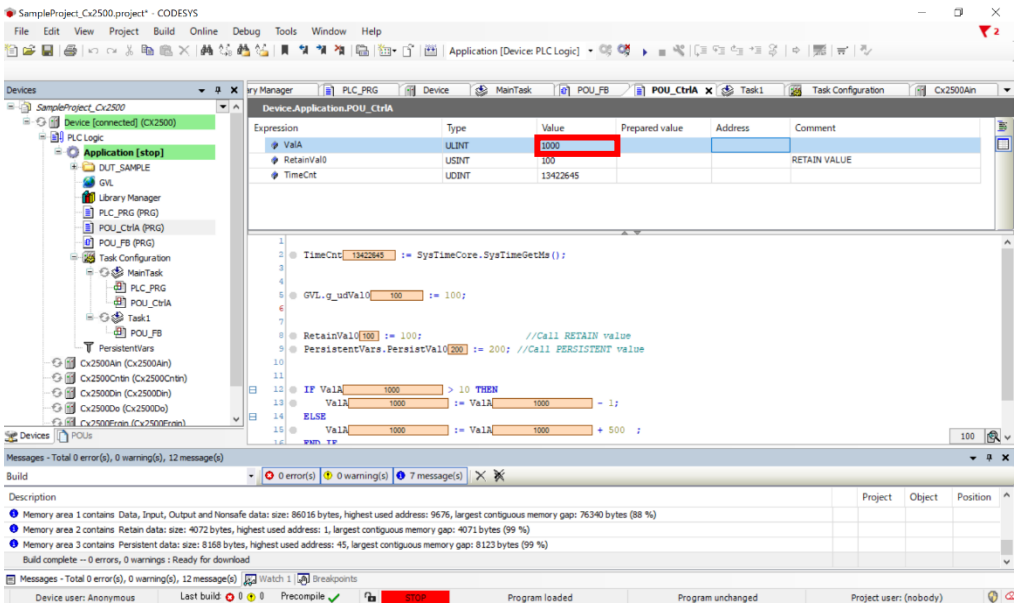


Figure 174 値書き込み完了後

9.5.2. 値の強制

ここでは、変数の値の強制手順を示します。値を強制した後は 9.5.3 項のように強制を解除しないと通常のアプリケーション動作による変数値に戻らないことに留意して下さい。

① 値を強制したい変数の変数宣言部にて Prepared value 列に所望の値を入力します。

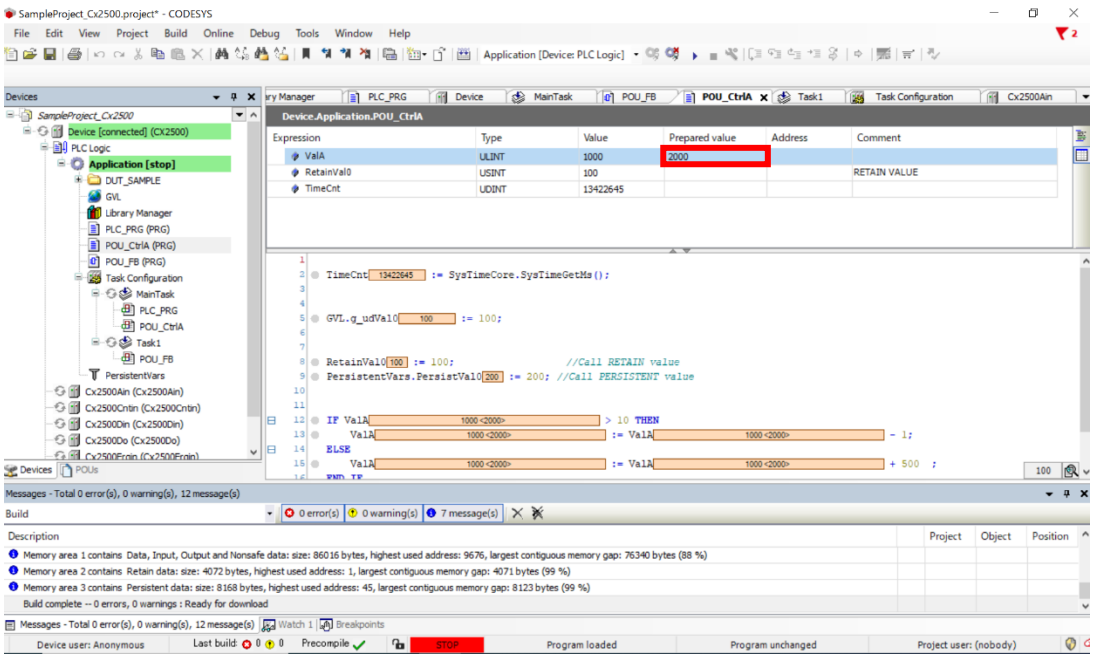


Figure 175 値の強制 Prepare value への値の入力

② メニューバーから「Debug」→「Force Values」を選択して下さい。

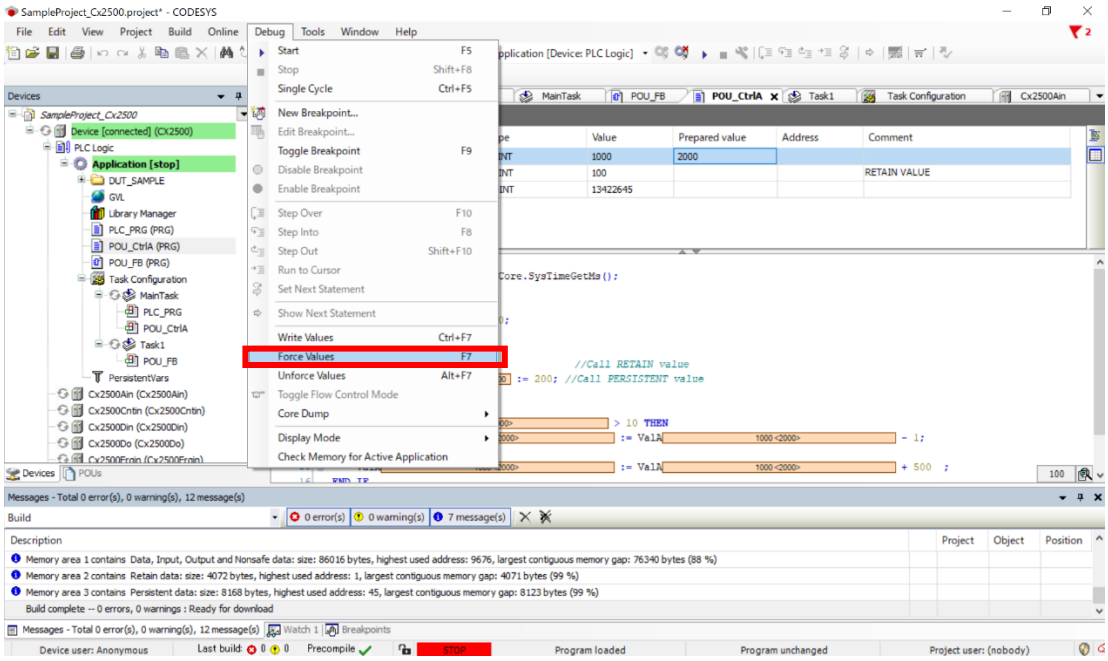


Figure 176 Force Values の選択

③ 所望の変数へ値が書き込まれ、強制されます。強制中は図のようなアイコンが変数の隣に表示されます。

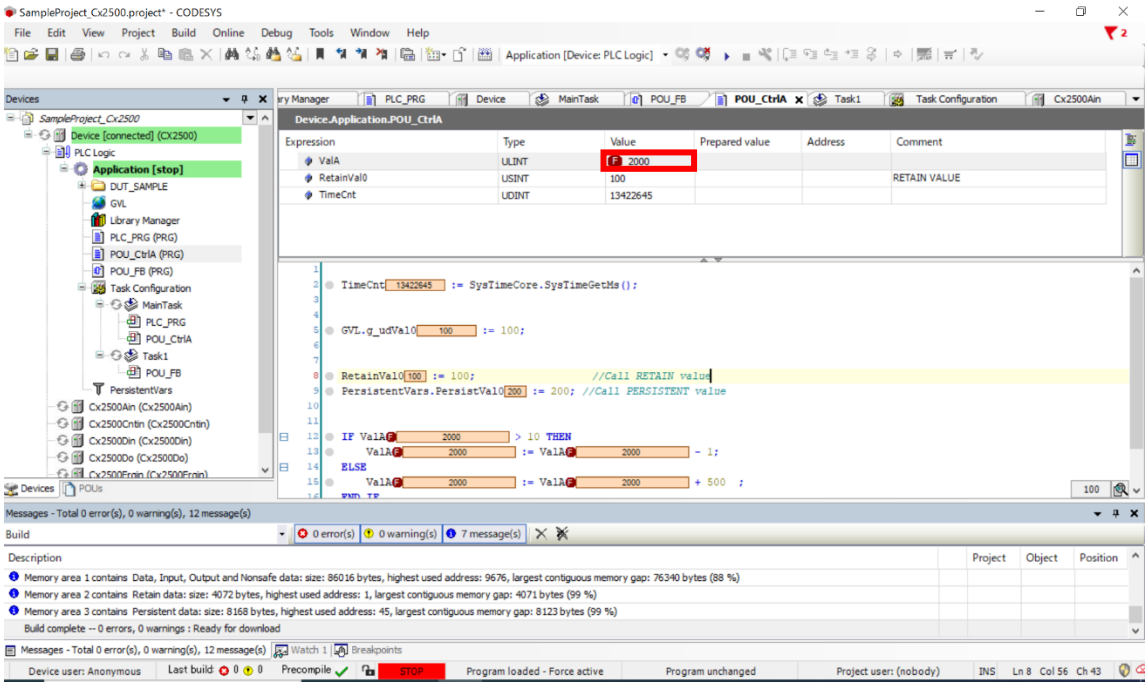


Figure 177 値の強制完了後

9.5.3. 値の強制解除

変数値を強制した場合、値を元に戻すには強制を解除する必要が有ります。値の強制解除をすると、強制されている変数全ての解除が行われます。ここでは、その手順を示します。

① メニューバーから「Debug」→「Unforce Values」を選択して下さい。

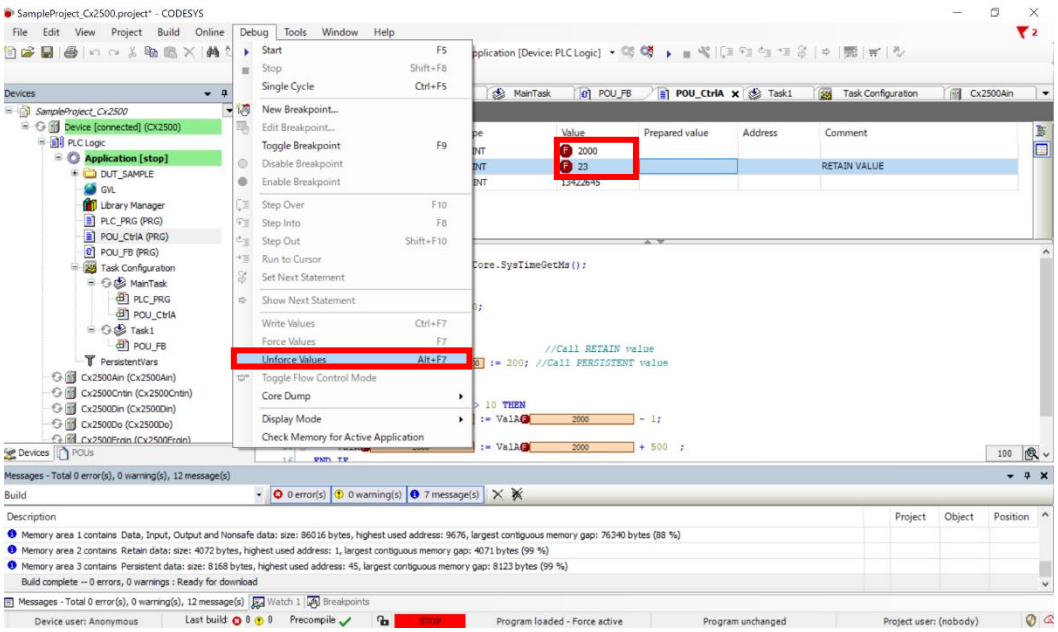


Figure 178 Unforced Values の選択

② 値が強制されている変数の強制が解除されます。解除と共に強制中のアイコンは消失します。

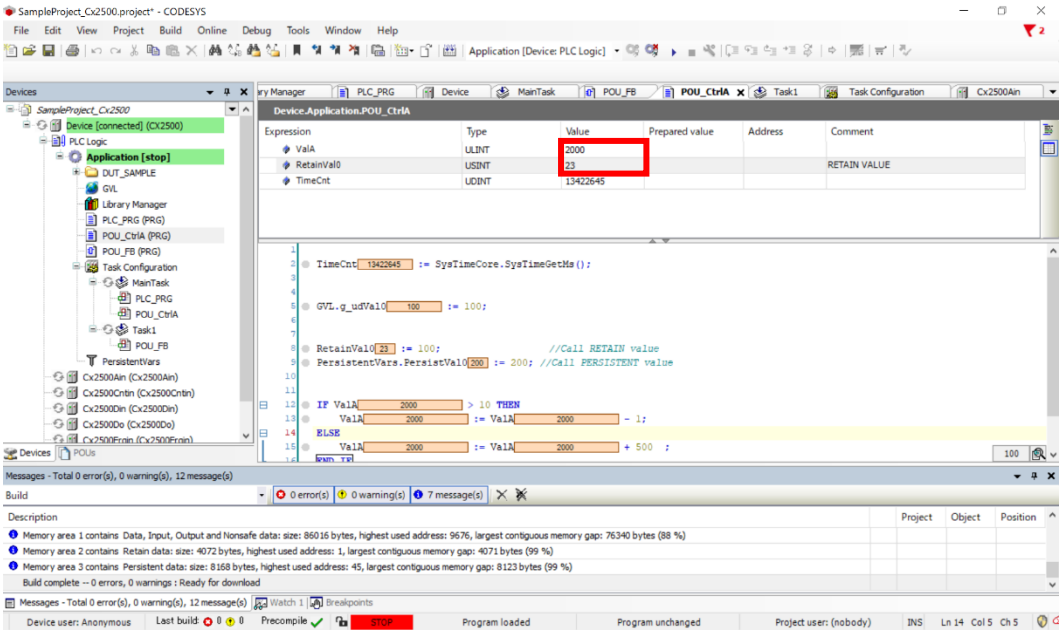


Figure 179 変数強制解除完了後

9.6. リセット

アプリケーションをリセットすることで変数などを初期化できます。CODESYS-IDE でできるリセットは 3 種類(Reset Warm・Reset Cold・Reset Origin)あります。各リセット等で初期化されるデータについては Table 73 を参照して下さい。

Table 73 各操作における初期化有無 ○:初期化されない(値が保持される)、×:初期化される

操作名	通常変数	保持変数	持続変数	アプリケーション	RTC 時刻
動作停止	○	○	○	○	○
Reset Warm (ウォームリセット)	×	○	○	○	○
Reset Cold (コールドリセット)	×	×	○	○	○
Reset Origin (PLC 初期化)	×	×	×	×	○
Download (アプリ書込)	×	×	○	–※21	○
電源再起動	×	○	○	○	○

※21 新しいアプリケーションを書き込むので、初期化ではなくアプリケーションが更新されます。

9.6.1. リセット手順

CODESYS-IDE でおこなえるリセットの手順を示します。ここでは Reset Warm を例に示しますが、Reset Cold や Reset Origin についても手順は同様になります。

なお、Reset Origin については Table 73 の通りリセット実施後アプリケーションも CX2500 から削除されます。再びアプリケーションを動作させるには、ログイン中の場合はログアウトをおこない、再度ログインをおこなう必要が有ります。

① デバッグモード中にメニューバーから「Online」→「Reset Warm」を選択して下さい。

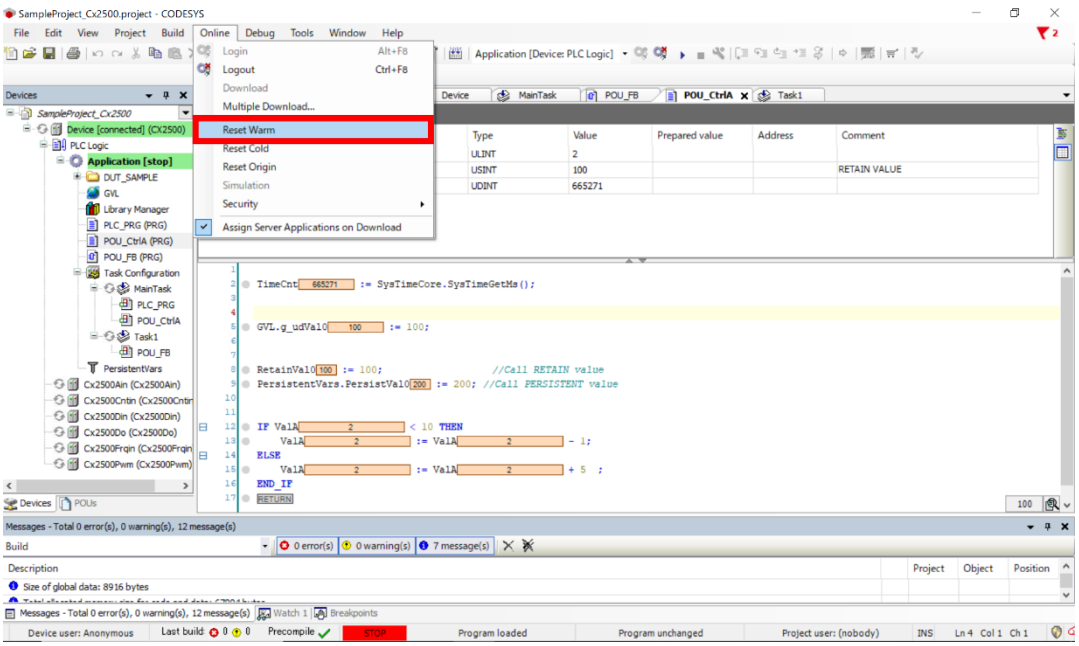


Figure 180 Reset Warm の選択

② 確認ウィンドウが表示されるので、「はい(Y)」ボタンを押して下さい。

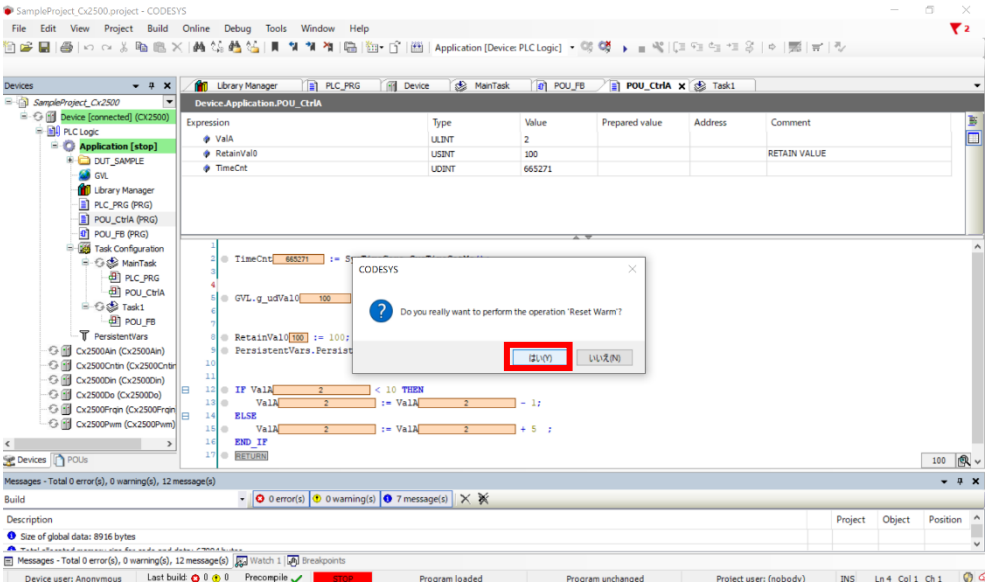


Figure 181 Reset Warm の実施前確認ウィンドウ

③ Reset Warm により、値がリセットされてアプリケーション動作が停止します。これでリセット完了です。

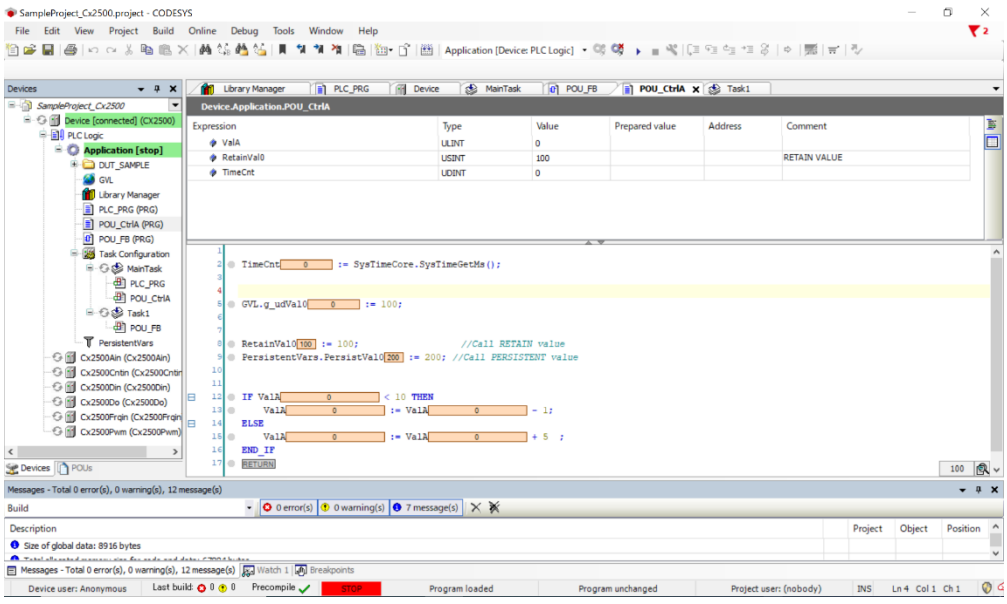


Figure 182 Warm Reset 後の POU

9.7. ブレークポイント

ブレークポイントは、デバッグモード中にプログラムを任意の位置で強制的に停止させる機能です。プログラムの途中でユーザーが自由に停止できるため、処理中の変数値変化等を詳細に確認することができます。ブレークポイントは最大 4 つまで同時に設定(有効化)可能です。なお、デバッグモードを終了(ログアウト)すると、設定したブレークポイントは次のログイン時に無効化されること(次のログインまでのアプリケーションを変更していた場合はブレークポイント削除)に留意して下さい。

CODESYS で利用できるブレークポイントには下記の 2 種類あります。それぞれの設定方法については後述します。

Table 74 ブレークポイント種類

名称	摘要	設定方法(参照先)
通常ブレークポイント	条件問わず、設定した位置の処理を行うときに毎回停止する。	9.7.1 項
条件付きブレークポイント	通常のブレークポイントと異なり、ユーザーが設定した特定条件を満たす場合のみ停止する。	9.7.2 項

Table 75 ブレークポイントステータス

アイコン	状態
	ブレークポイント設定可能
	ブレークポイント設定中(無効状態)
	通常ブレークポイント設定中(有効状態)
	条件付きブレークポイント設定中(有効状態)
	ブレークポイント停止中

9.7.1. ブレークポイントの設定(通常)

通常のブレークポイントの設定手順を下記に示します。

① デバッグモードにして、ブレークポイントを設定したい行を選択し右クリックを押して下さい。選択された行は図のように黄色に変わります。

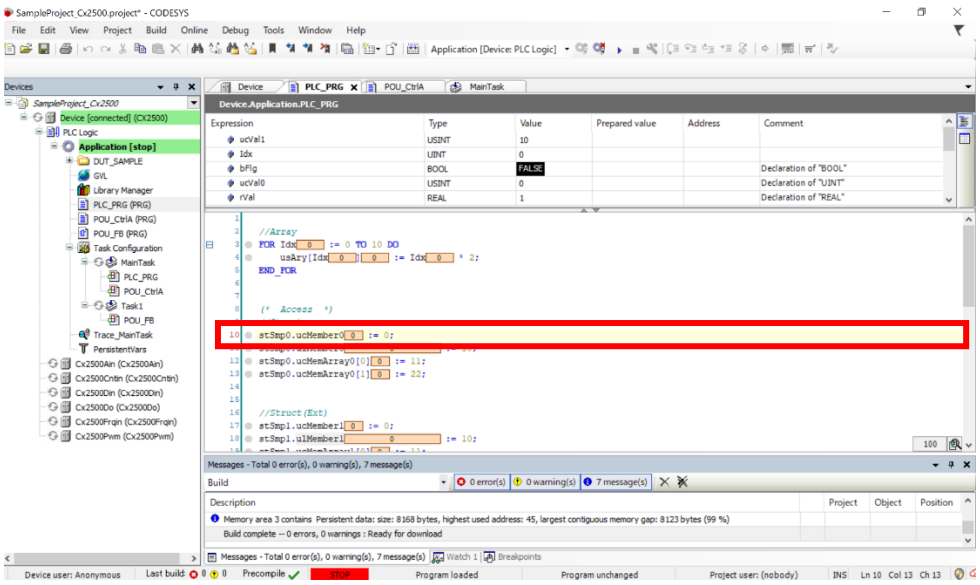


Figure 183 ブレークポイントを設定したい行の選択

② 表示されるコンテキストメニューから「Toggle Breakpoint」を選択して下さい。

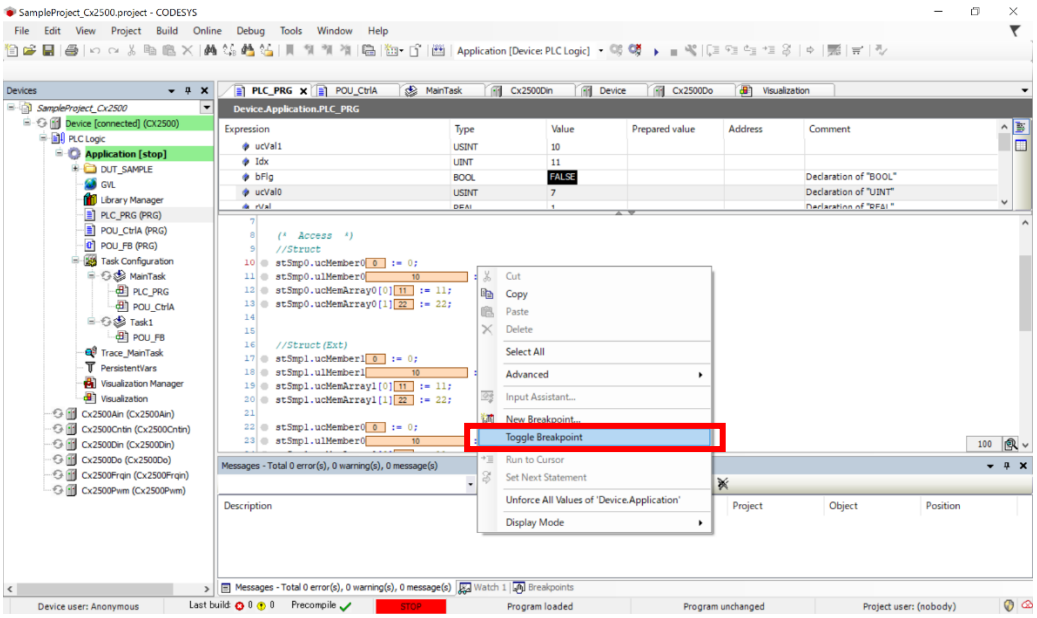


Figure 184 Toggle Breakpoint の選択

③ アイコンがブレークポイント設定中(有効状態)に変わると設定完了です。

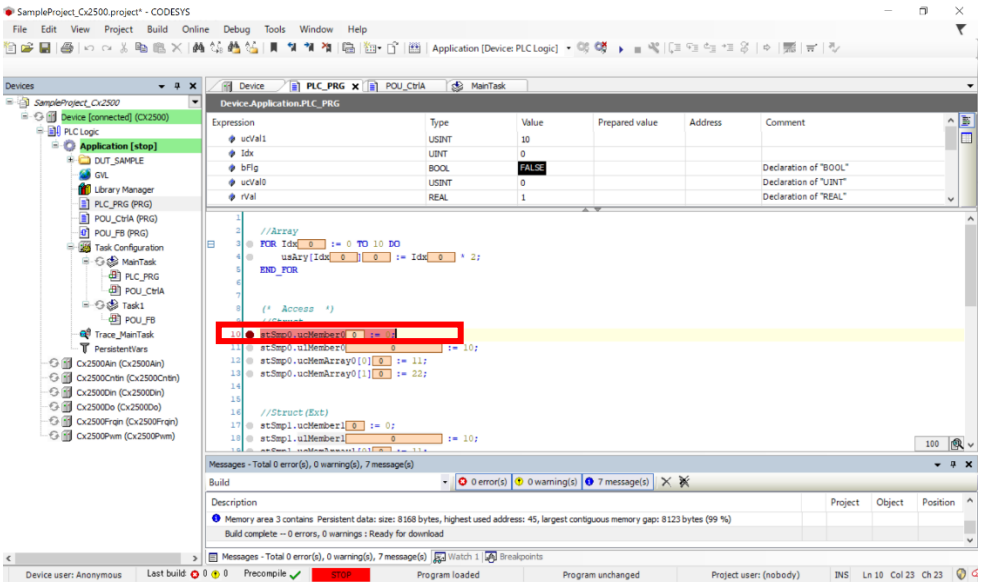


Figure 185 通常ブレークポイント設定完了後

9.7.2. ブレークポイントの設定(条件付き)

ここでは、条件付きブレークポイントの設定手順について示します。

① 条件付きブレークポイントを設定したい行を選択し、右クリックして下さい。

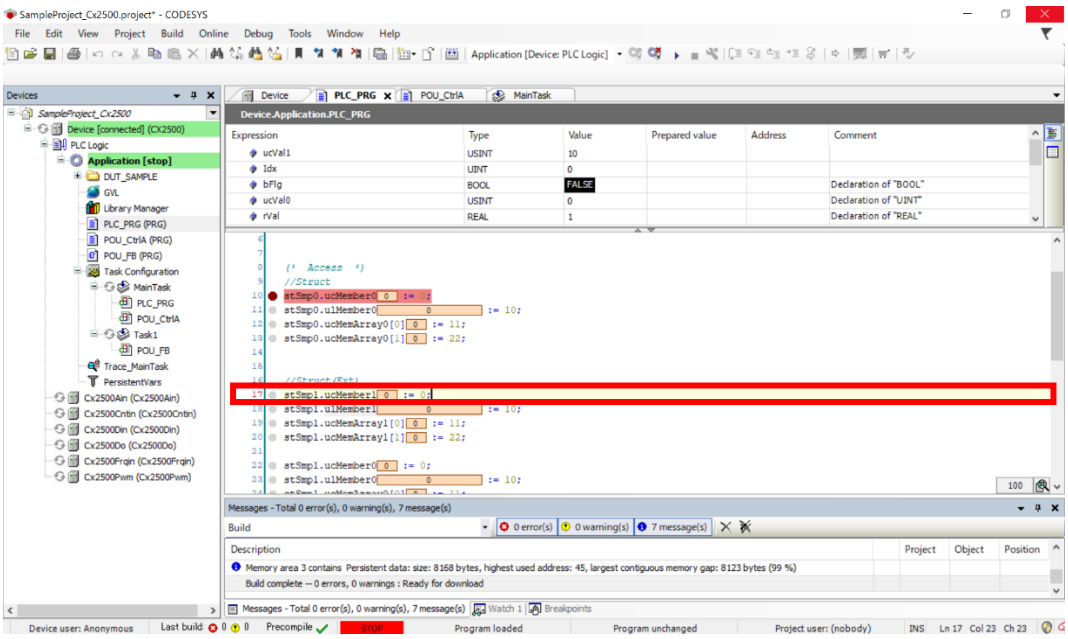


Figure 186 ブレークポイントを設定したい行の選択

② 表示されるコンテキストメニューから「New Breakpoint...」を選択して下さい。

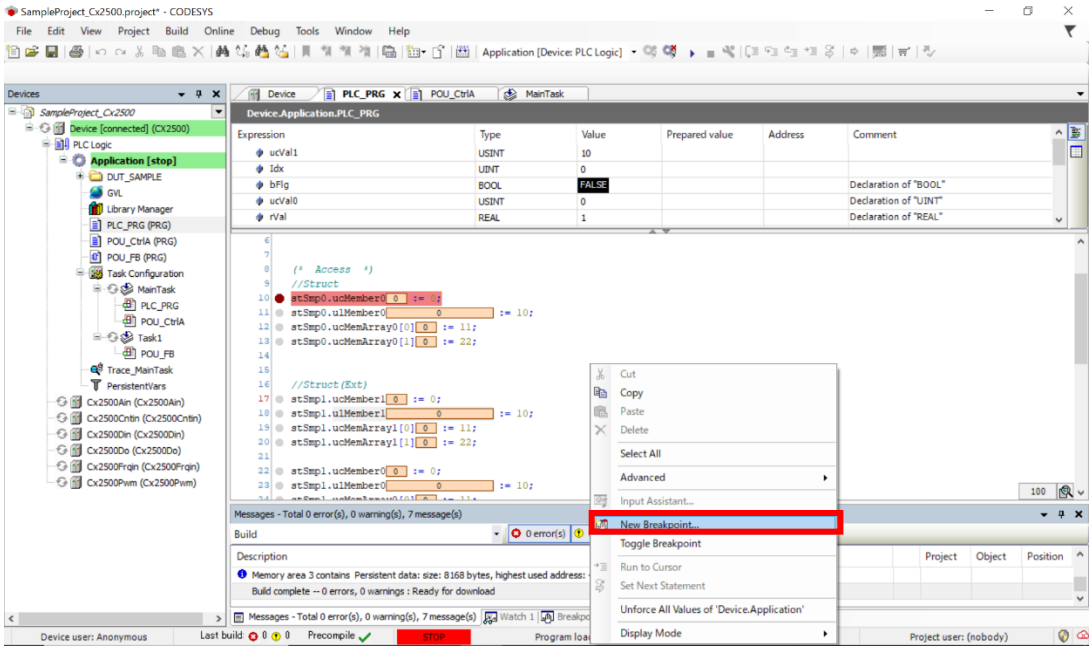


Figure 187 New Breakpoint の選択

③ 「Breakpoint Properties」ウィンドウが表示されるので、「Condition」タブを選択してブレークポイント停止条件を設定します。条件を設定したら「OK」ボタンを押して下さい。なお、ここでは例としてブレークポイントを設定する行に 5 回入った時に停止する条件を設定しています。

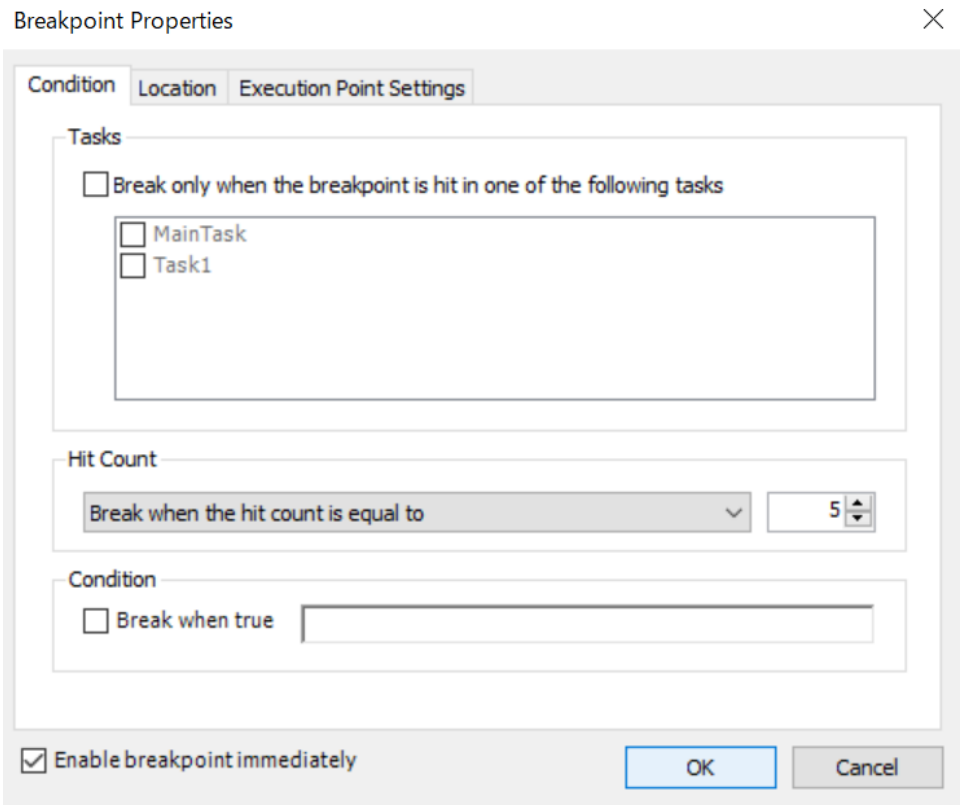


Figure 188 Breakpoint Properties ウィンドウ Condition タブ

Table 76 Breakpoint Properties ウィンドウ Condition タブの設定項目

設定項目	摘要
Tasks	チェックを入れたタスクが全て 1 度以上実行されたのちにブレークポイントで停止する。
Hit Count	タブで選択した回数(条件)分指定行に入った時にブレークポイントで停止する。
Condition	「Break when true」にチェックを入れると、その右隣の欄に入力した変数値が true の時にブレークポイントで停止する。
Enable breakpoint immediately	チェックを入れるとブレークポイントが設定後即有効化される。

④ 図のようにブレークポイントが設定されれば完了です。

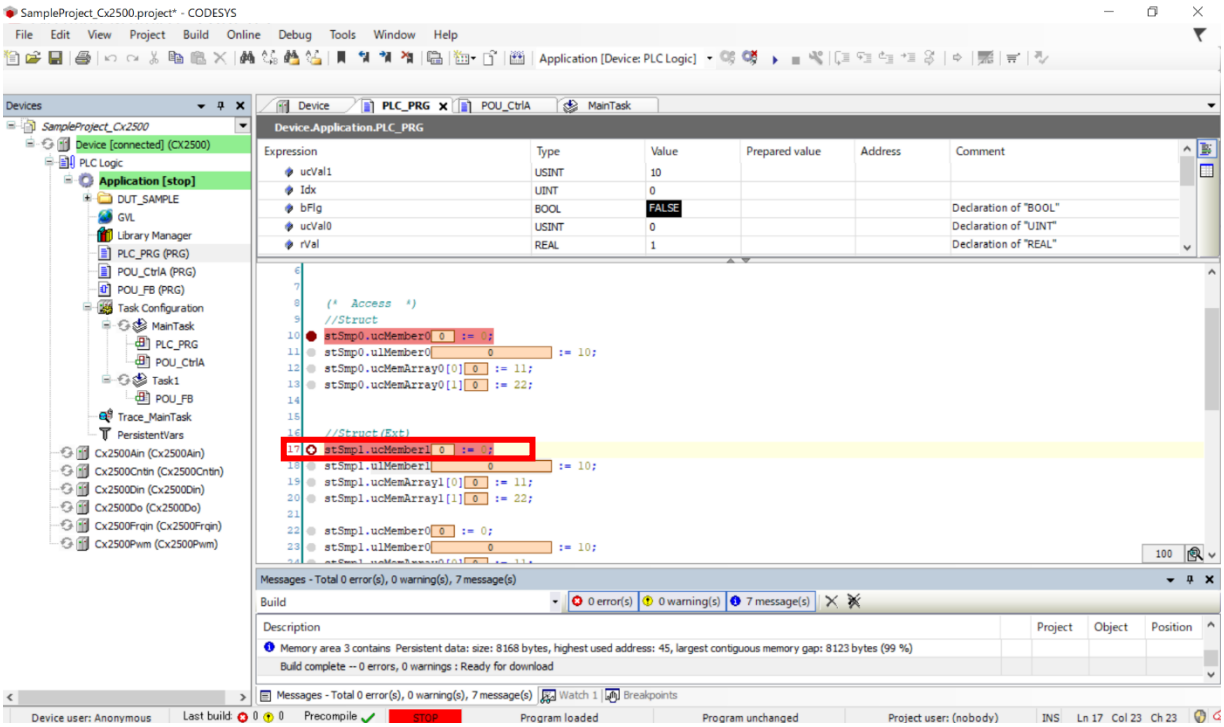


Figure 189 条件付きブレークポイント設定完了後

9.7.3. ブレークポイントの無効化設定

ここでは設定して有効化状態(指定位置でプログラムが停止する)のブレークポイントを無効化(指定位置でプログラムは停止しない)する手順を示します。

① 無効化したいブレークポイントがある行を選択し右クリックして下さい。

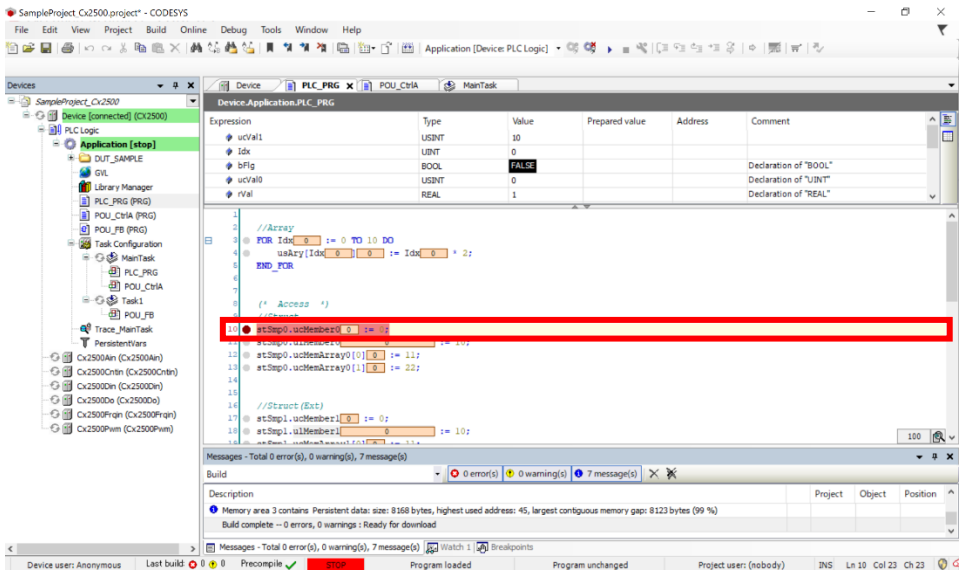


Figure 190 ブレークポイントを無効化したい行の選択

② 表示されるコンテキストメニューから「Disable Breakpoint」を選択して下さい。

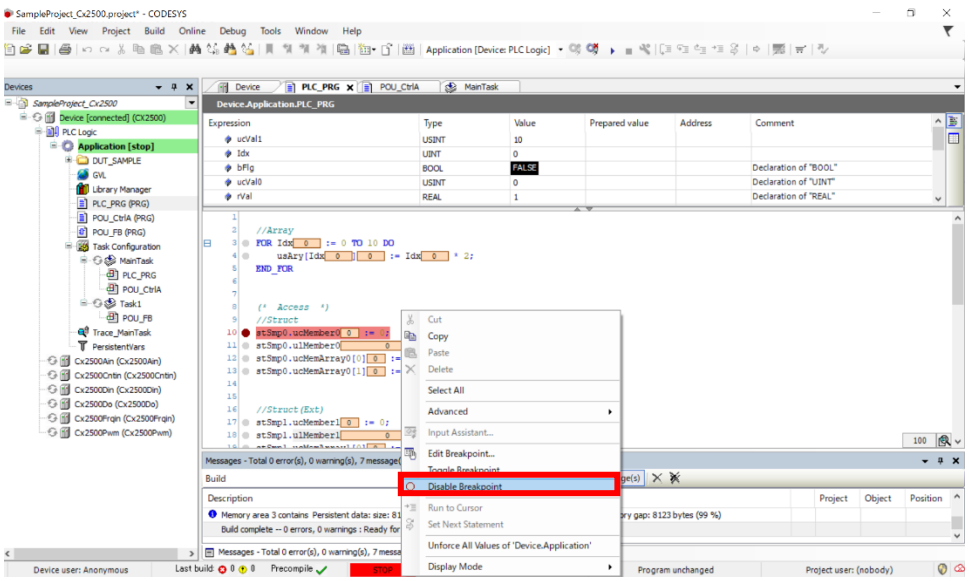


Figure 191 Disable Breakpoint の選択

③ 図のようにブレークポイントのアイコンが無効化状態になれば完了です。なお、無効化状態から有効化状態に変えたいときは、同様の手順で行います。ただし、手順②では「Enable Breakpoint」を選択して下さい。

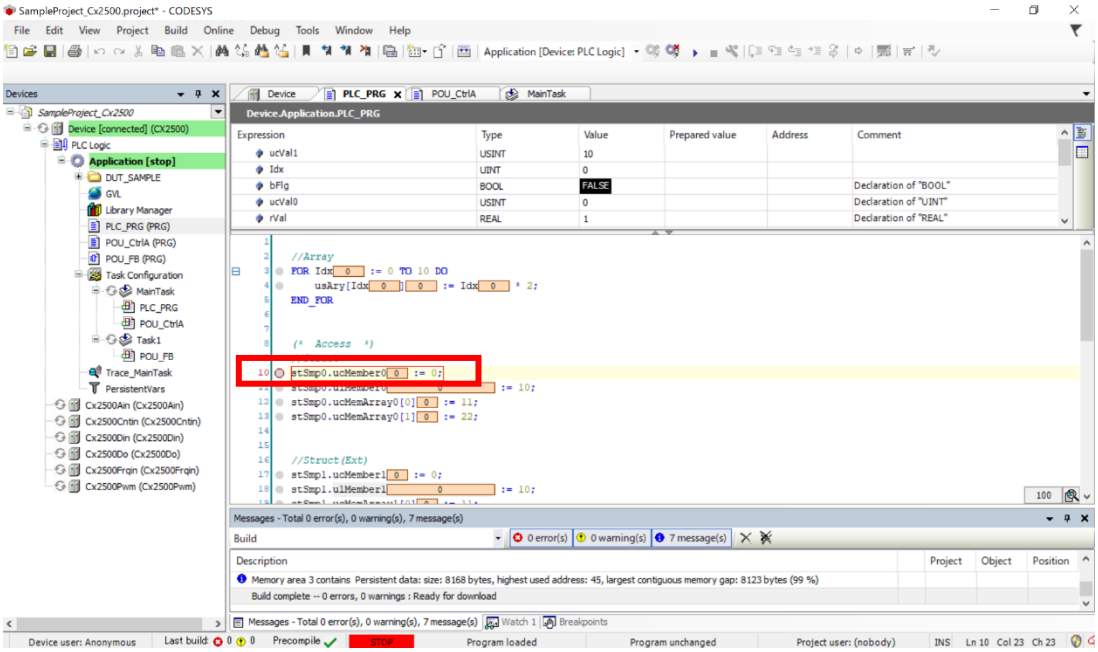


Figure 192 ブレークポイント無効化完了後

9.7.4. ブレークポイントの設定削除

設定したブレークポイントを削除する手順を下記に示します。ブレークポイントの無効化(9.7.3 項)と異なり、ブレークポイントを削除するとこれまでのブレークポイント到達数などの情報も削除されてしまうことに留意して下さい。

① 削除したいブレークポイントがある行を選択した状態で右クリックして下さい。

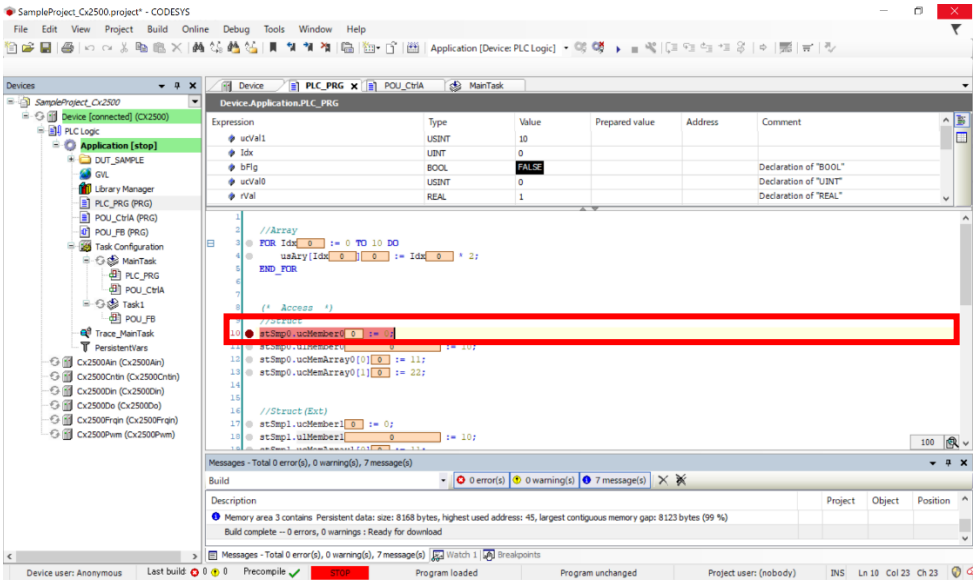


Figure 193 ブレークポイントを削除したい行の選択

② 表示されるコンテキストメニューから「Toggle Breakpoint」を選択して下さい。

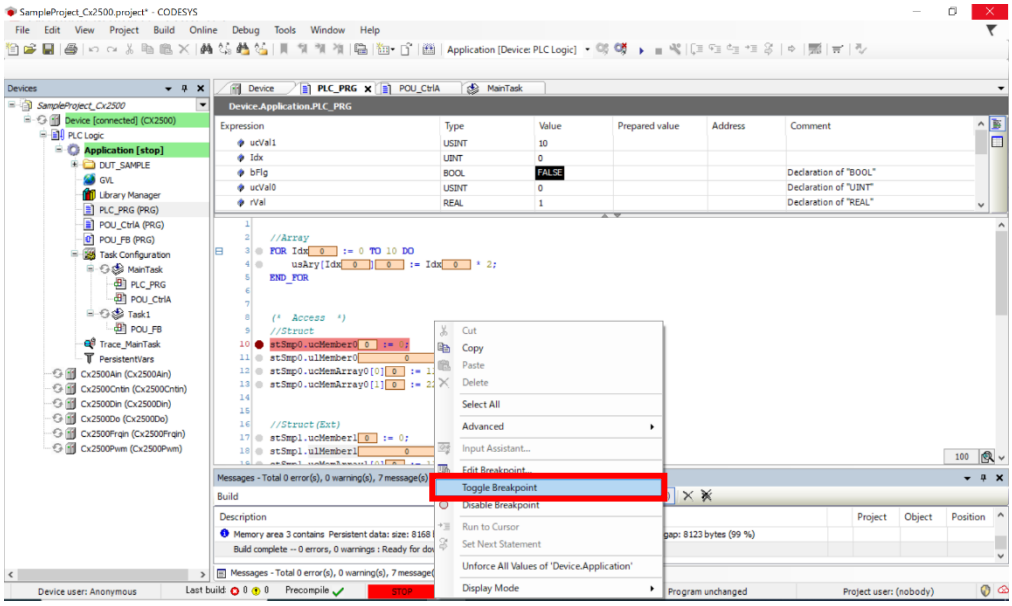


Figure 194 Toggle Breakpoint の選択

③ 図のようにブレークポイントアイコンがブレークポイント設定可能状態になっていれば、ブレークポイントの削除は完了です。

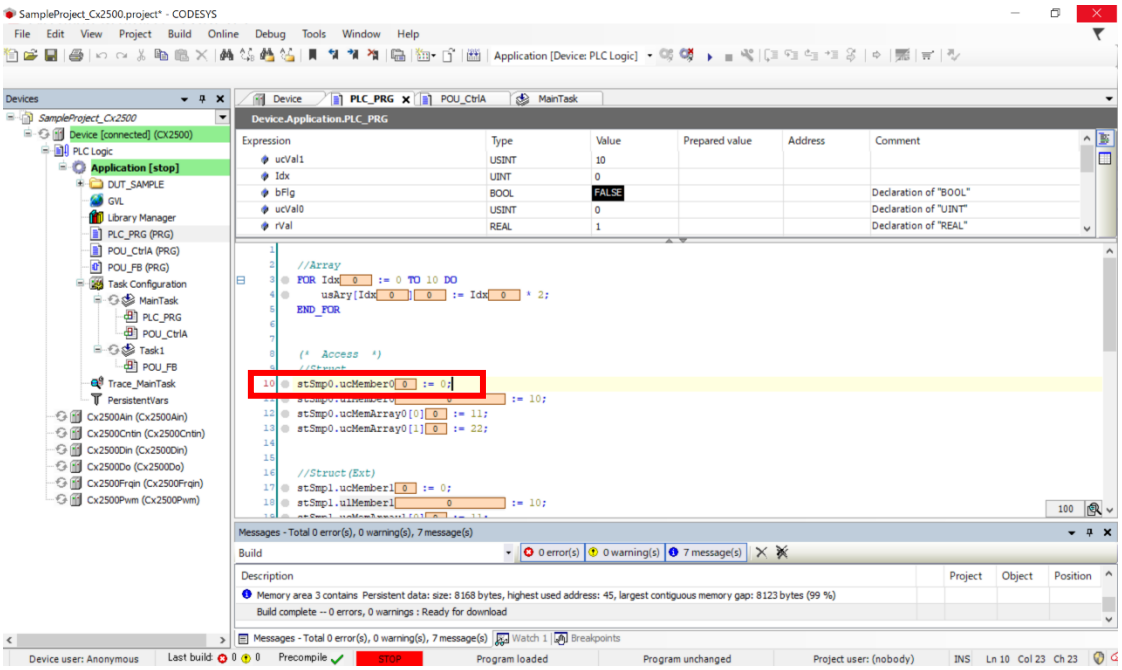


Figure 195 ブレークポイント削除完了後

9.7.5. ブレークポイント中の操作

ブレークポイント停止中、メニューバーの「Debug」から下記の操作ができます。

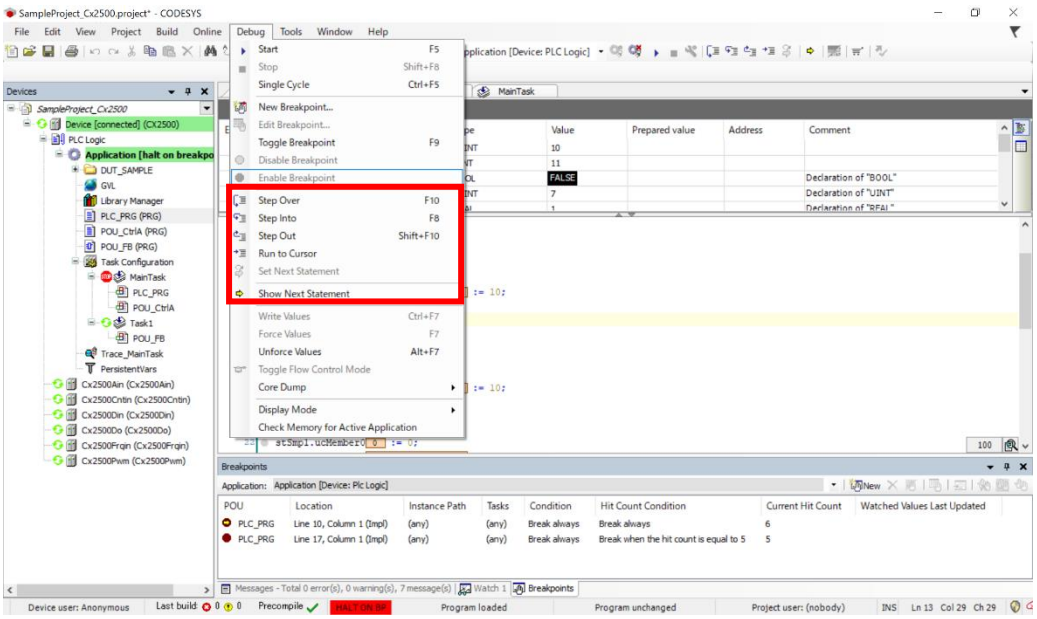


Figure 196 ブレークポイント 操作コマンド

Table 77 ブレークポイント中の操作

操作名	摘要
Step Over	ブレークポイントで停止している行の処理が実行される。この行に関数など別 POU 呼び出しで合った場合、呼び出し先の処理をすべて実行した上で元の POU の次の行で停止する。
Step Into	ブレークポイントで停止している行の処理が実行される。Step Over と異なり、この行が関数など別 POU の呼び出しであった場合はその呼び出し先にジャンプし、その最初の行で停止する。
Step Out	ブレークポイント以降の全ての処理を実行し次のプログラム(POU)の先頭で停止、もしくは呼び出し元 POU の次の行で停止する。
Run to Cursor	ブレークポイント以外の任意の行を選択した状態でこの操作をおこなった時、その行のまえまでアプリケーションの動作が行われて停止する。
Set Next Statement	ブレークポイント以外の任意の行を選択した状態でこの操作をおこなった時、ブレークポイントからその行までの処理は実行されずに一気にジャンプして停止する。
Show Next Statement	現在の停止位置にフォーカスされる。

9.7.6. ブレークポイントステータス

ブレークポイントステータス画面で設定しているブレークポイントのリストを確認できます。また、各ブレークポイントが動作中何度機能したか確認することもできます。

下図のようにブレークポイントステータス画面が表示されていない場合、メニューバーから「View」→「Breakpoints」を選択すると画面が表示されます。

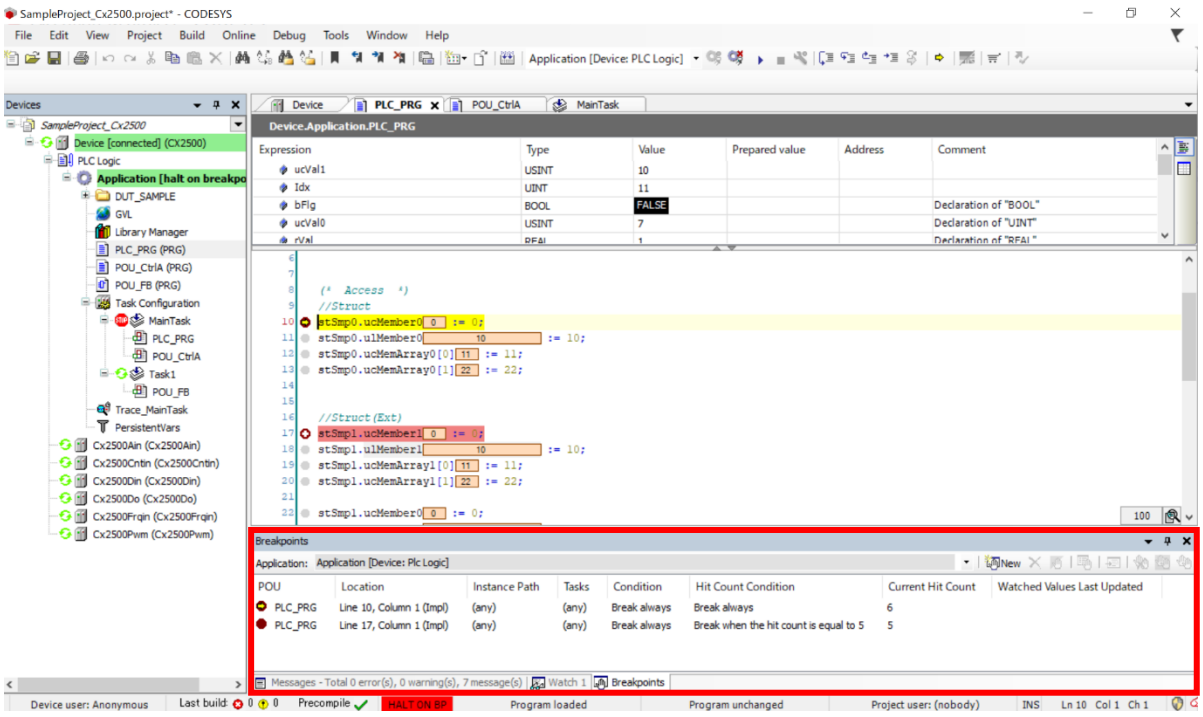


Figure 197 ブレークポイントステータス画面

9.8. デバイスログ

デバイスログは、デバッグ中のデバイス状態をモニタすることができます。これにより、デバイス(PLC)の内部イベントや例外を含むエラー等を確認可能です。デバイスログは、デバイスウィンドウから「Log」を選択することでログ画面が表示され、確認することができます。各ログ画面中のボタンについては Table 78 を参照して下さい。

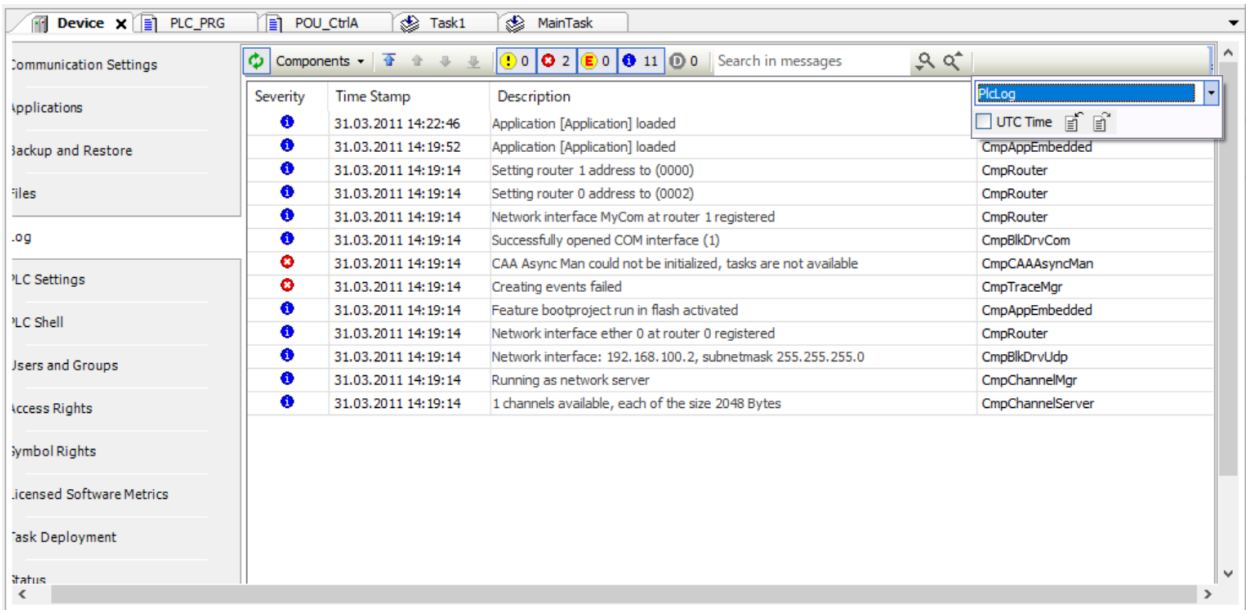


Figure 198 Device タブ Log 画面

Table 78 デバイスログ 各部概要

#	名称	摘要
1	定期更新ボタン	このボタンを押した状態にした時(上図のように背景が青くなる)、デバイスからのログを定期的に画面に表示・更新する。ただし、ログは一定量超過すると削除される。
2	コンポーネント選択タブ	ログ画面に表示されているものの内、このタブで選択したコンポーネント(機能)の情報のみ表示する。
3	情報選択ボタン	各矢印ボタンを押すことで、ログ画面に表示される情報をひとつずつ確認できる。
4	Warning	このボタンを押した状態にした時、ログ画面に Warning を表示する。
5	Error	このボタンを押した状態にした時、ログ画面に Error を表示する。
6	Exception	このボタンを押した状態にした時、ログ画面に Exception を表示する。
7	Information	このボタンを押した状態にした時、ログ画面に Information を表示する。
8	エラー検索エリア	検索エリアに値を入力した時、その値と合致するログのみ画面に表示する。
9	ログインポートボタン※22	このボタンを押すと、エクスポートしたログをインポートし、ログの確認ができる。
10	ログエクスポートボタン※22	このボタンを押すと、画面に表示されるログをエクスポートする。

※22 このボタンはログ画面上の右上部(図上青枠部)をクリックすると表示される。

9.9. トレース

トレースは、デバッグモード中に変数値の変化をグラフ上で視覚的にモニタできる機能です。モニタ履歴は保存することができ、保存したファイルを読み込むことで履歴の確認をおこなうことができます。

9.9.1. トレースの作成

プロジェクトにトレースを作成します。作成手順の例を以下に示します。

① デバイスウィンドウにて、「Application」にカーソルを合わせ右クリックして下さい。

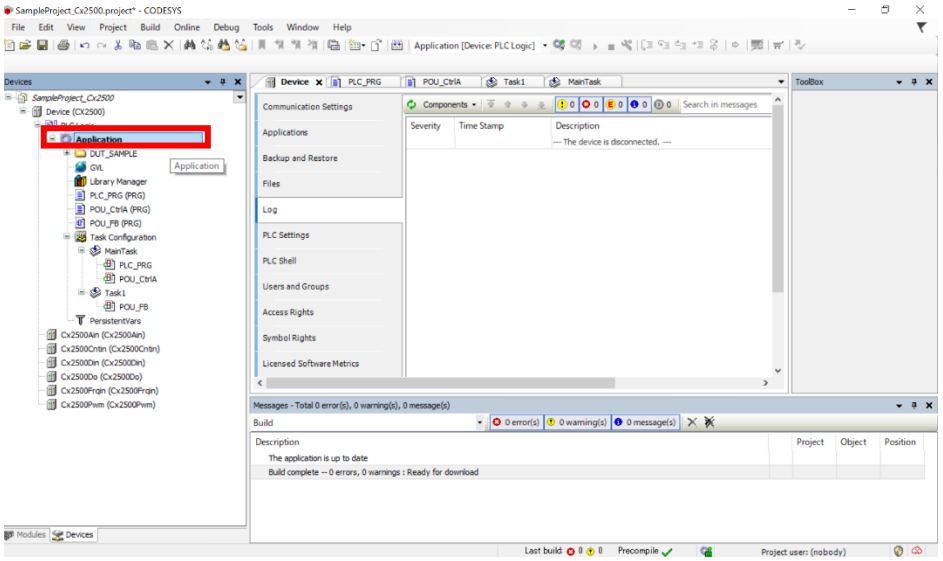


Figure 199 トレース作成 Application の選択

② 表示されるコンテキストメニューから「Add Object」→「Trace..」を選択して下さい。

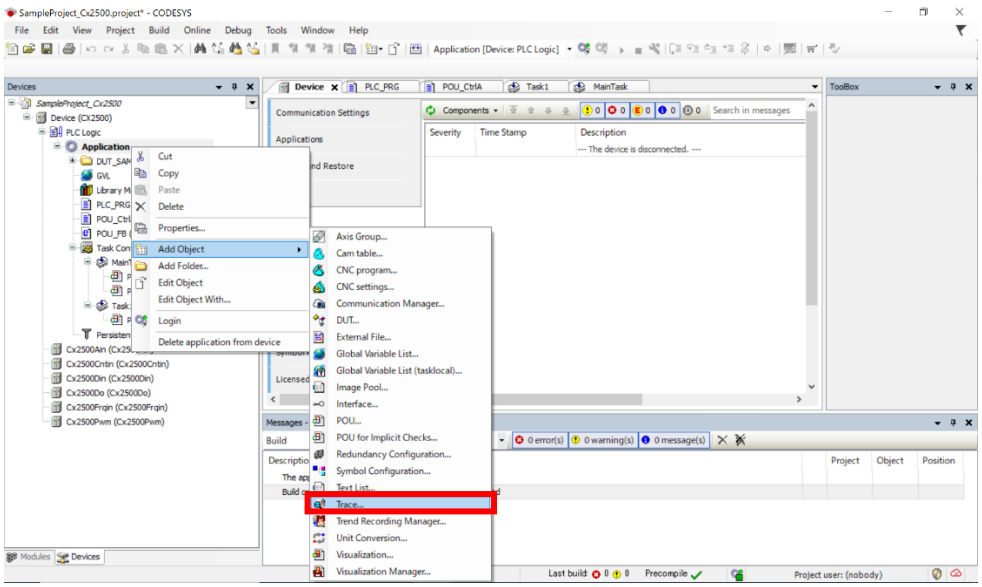


Figure 200 Trace の選択

③ 「Add Trace」ウィンドウが表示されるので、下記のようにトレース名とモニタしたいタスク名を入力・選択し、「Add」ボタンを押して下さい。

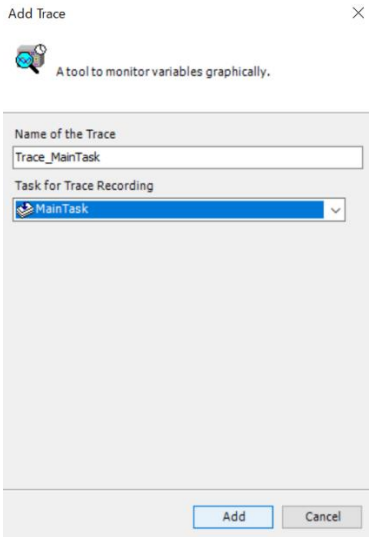


Figure 201 Add Trace ウィンドウ

④ 下図のようにデバイスウィンドウにトレースが追加され、また、トレース画面が表示されます。これでトレースの作成は完了です。

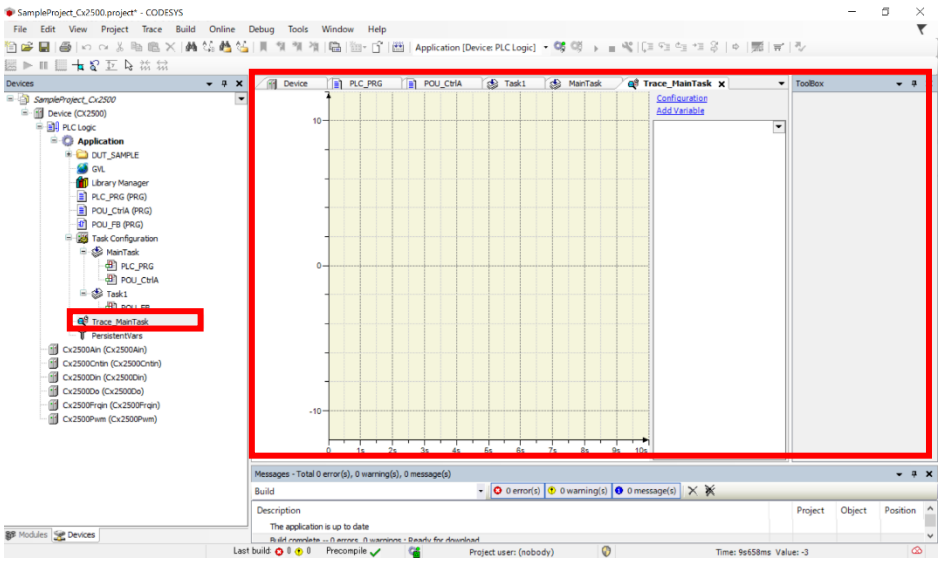


Figure 202 トレース追加完了後

9.9.2. トレースの設定

ここではトレースを開始するための設定をおこないます。

① トレース画面右上の「Configuration」をクリックして下さい。

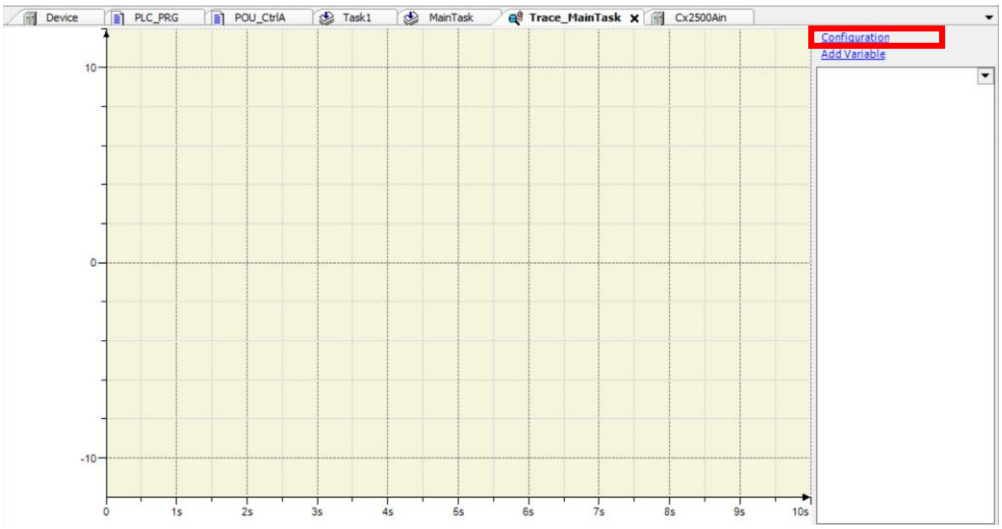


Figure 203 トレース画面 Configuration の選択

② 「Trace Configuration」ウィンドウが表示されるので、各種所望の値を設定し「OK」ボタンを押して下さい。

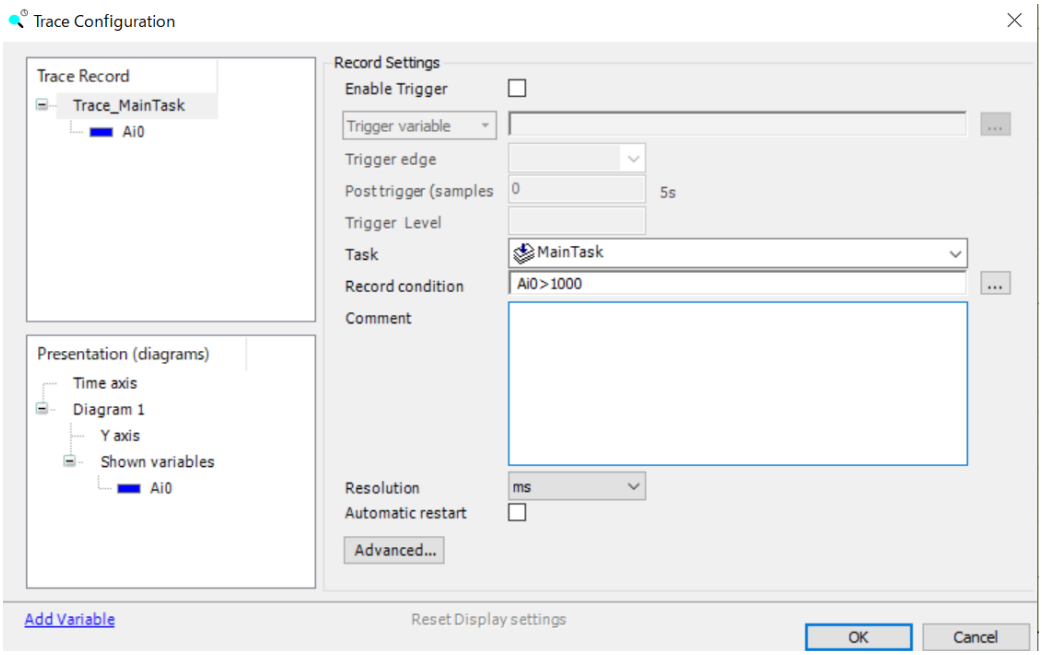


Figure 204 Trace Configuration ウィンドウ

Table 79 Trace Configuration ウィンドウ トレース設定 設定項目

設定項目	摘要
Enable Trigger	チェックを入れるとトレース開始をトリガ起因にできる。Trigger variable、Trigger edge を満たした時にトレースを開始できる。
Trigger variable	トリガ条件となる変数を入力、もしくは「…」ボタンを押して選択する。
Trigger edge	トリガ条件となるエッジの種類を選択する。
Post trigger(samples)	トリガにてトレース開始した際のサンプリング数を設定する。
Trigger level	Trigger variable が BOOL 型の変数出ない時、トリガエッジがかかる数値を設定する。
Task	トレースするタスクを設定する。
Record condition	トレース開始およびトレース継続する条件を設定できる。
Comment	このトレースについてのコメントを記入できる。
Resolution	トレースする時間の分解能を設定できる。
Automatic restart	チェックを外した状態にする。
Advanced	ボタンを押すと「Advanced Trace Settings」ウィンドウが表示される。詳細は Table 80 を参照。

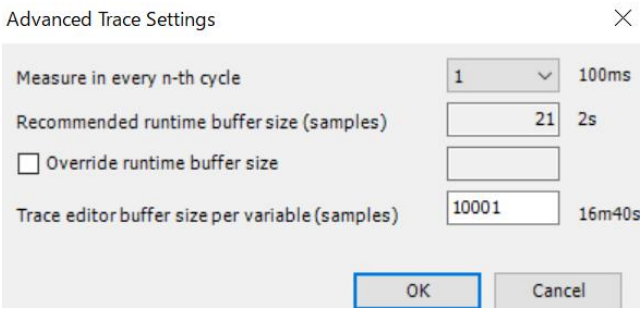


Figure 205 Advanced Trace Settings ウィンドウ

Table 80 Trace Configuration Advanced 設定項目

設定項目	摘要
Measure in every n-th cycle	トレースのサンプリング時間を選択タブから設定できる。数値をタブから選択すると、その右隣に対応するサンプリング時間(=設定値×タスク周期時間)が表示される。
Recommended runtime buffer size(samples)	ユーザー設定不要。上記サンプリング時間に従い、ランタイム内で必要となるバッファサイズが表示される。
Override runtime buffer size	チェックを入れると、ランタイムのバッファサイズを変更できる。大きさは、上記バッファサイズの 2 倍以上で設定する必要がある。
Trace editor buffer size per variable(samples)	各変数のサンプリング最大数を設定できる。

③ トレース画面でモニタしたい変数を登録します。アナログ入力の Ch.0 に割り当てた変数 Ai0 を例におこないます。トレース画面右上の「Add Variable」をクリックして下さい。

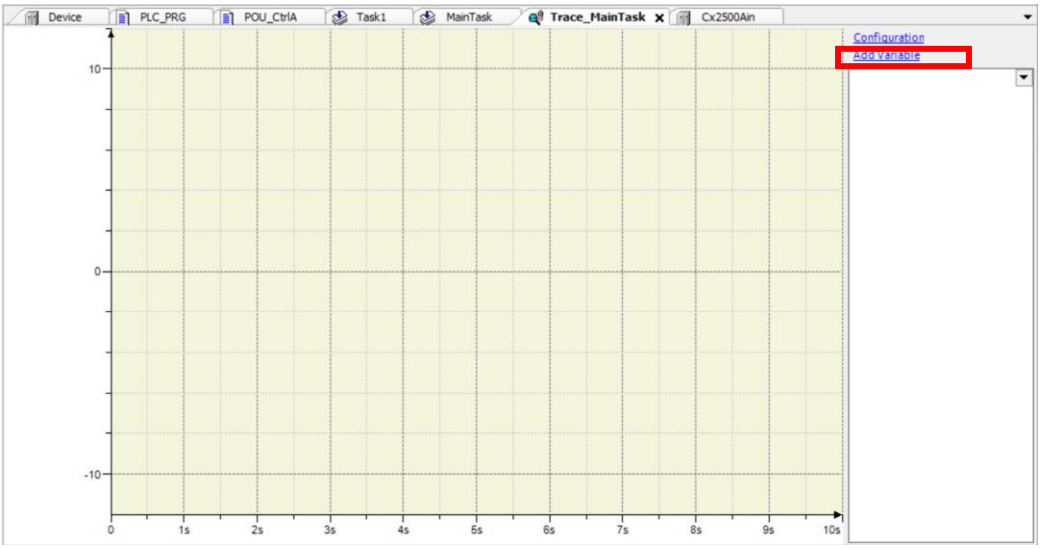


Figure 206 トレース画面 Add Variables の選択

④ 「Trace Configuration」ウィンドウが表示されるので、下記のように所望の値を設定して「OK」ボタンを押して下さい。

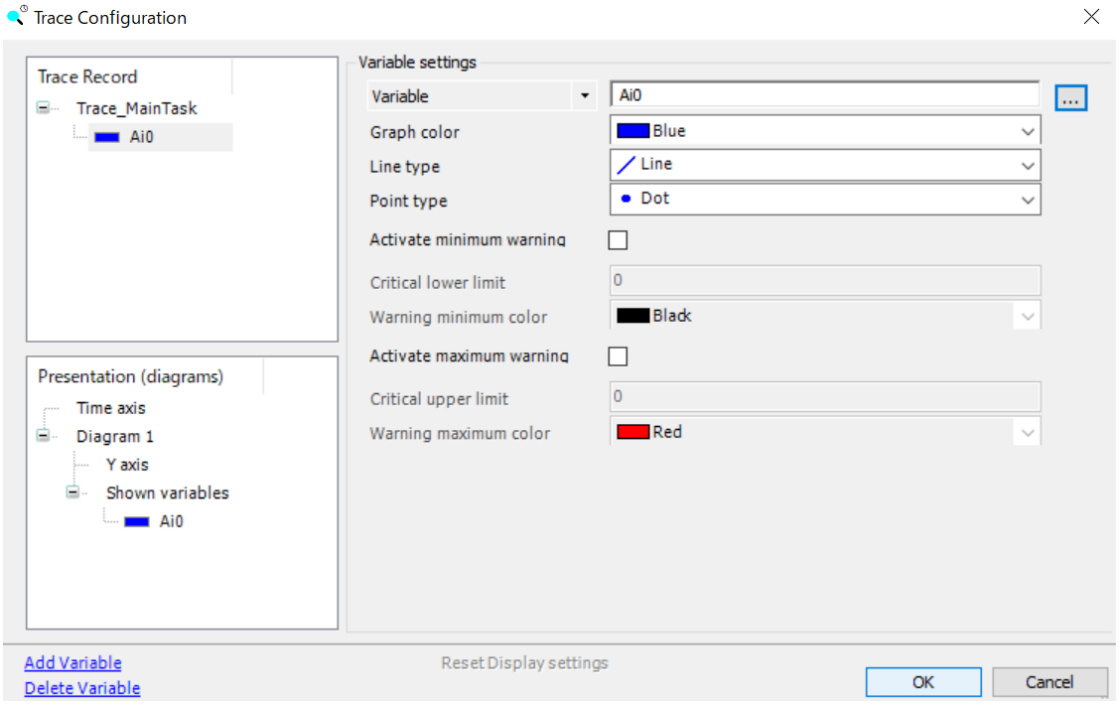


Figure 207 Trace Configuration 変数登録ウィンドウ

Table 81 Trace Configuration 変数登録 設定項目

設定項目	摘要
Variable	モニタ画面に追加したい変数を記入、もしくは「…」ボタンを押して所望の変数を選択する。
Graph color	対象の変数の線色を設定できる。
Line type	対象の変数の線種を設定できる。
Point type	対象の変数の点種を設定できる。
Activate minimum warning	チェックを入れると、変数値がグラフの下限値を下回った時に警告が表示される。
Critical lower limit	上記警告を表示するグラフの下限値を入力することで設定できる。
Warning minimum color	上記警告表示の色を設定できる。
Activate maximum color	チェックを入れると、変数値がグラフの上限値を上回った時に警告が表示される。
Critical upper limit	上記警告を表示するグラフの上限値を入力することで設定できる。
Warning maximum color	上記警告表示の色を設定できる。

⑤ トレース画面右上に追加した変数が表示されれば登録は完了です。

Figure 208 トレース画面 変数登録後

9.9.3. トレースの開始・終了

ここではトレースの開始・終了について示します。開始・停止手順は下記の通りです。

【開始手順】

① デバッグモード中にメニューバーから「Trace」→「Download Trace」を選択して下さい。

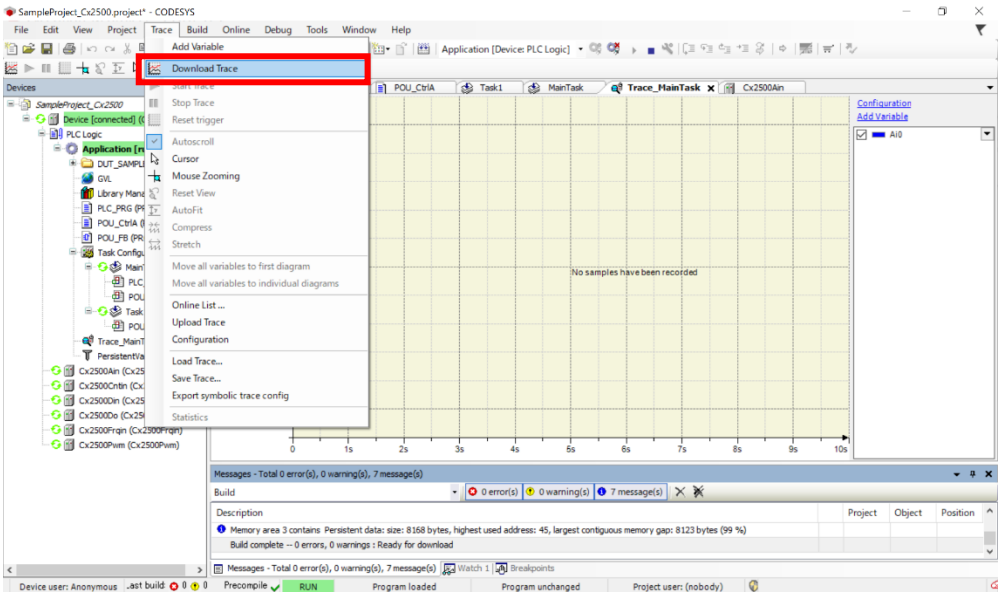


Figure 209 Download Trace の選択

② 9.9.2 項で設定したトレース開始条件を満たすとトレースが自動で始まります。

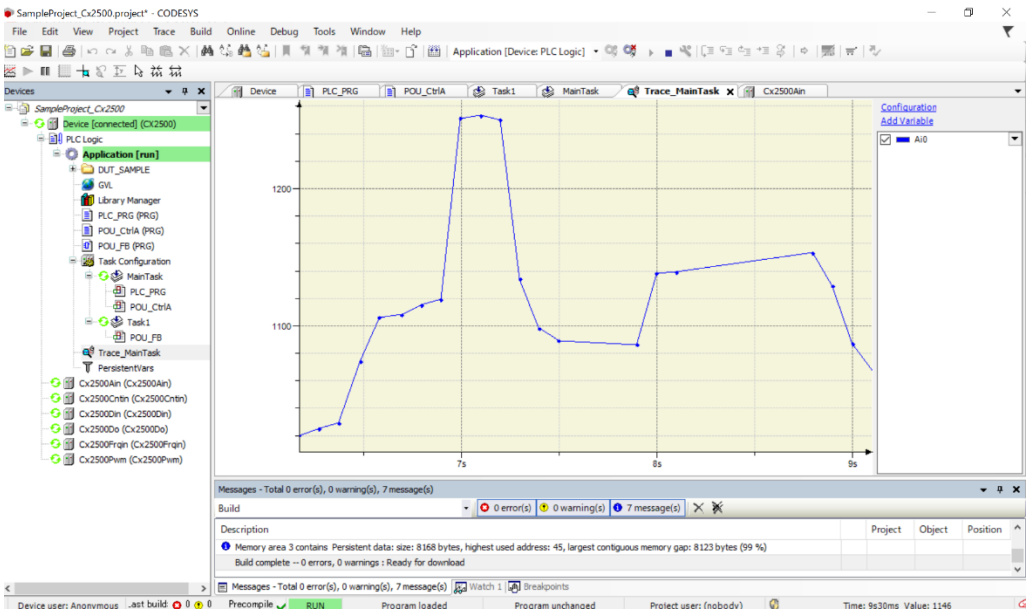


Figure 210 トレース中の画面

【終了手順】

① トレース中に、メニューバーから「Trace」→「Stop Trace」を選択して下さい。

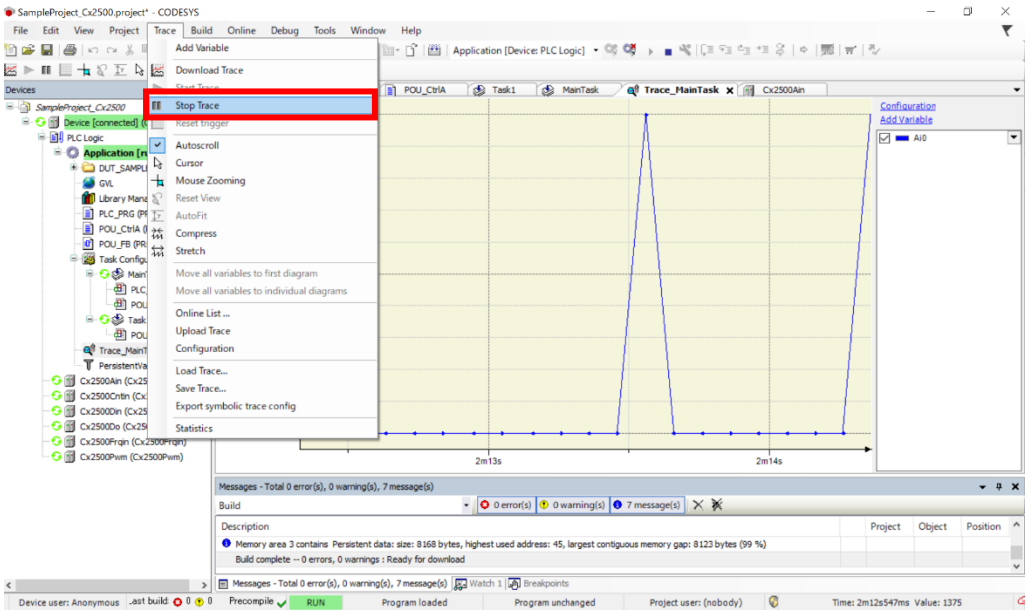


Figure 211 Stop Trace の選択

② トレースが停止します。メイン画面右下のステータスバーに「Trace stopped」と表示されます。

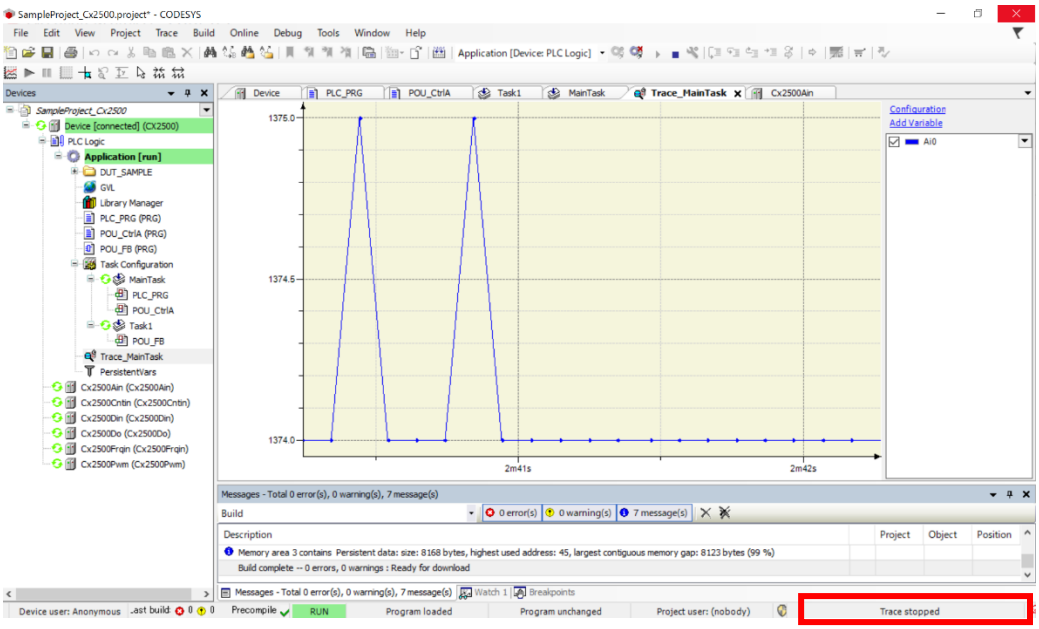


Figure 212 トレース停止後の画面

9.9.4. トレースの保存

ここではトレースの保存手順を示します。

① トレースを停止した状態で、メニューバーから「Trace」→「Save Trace」を選択して下さい。

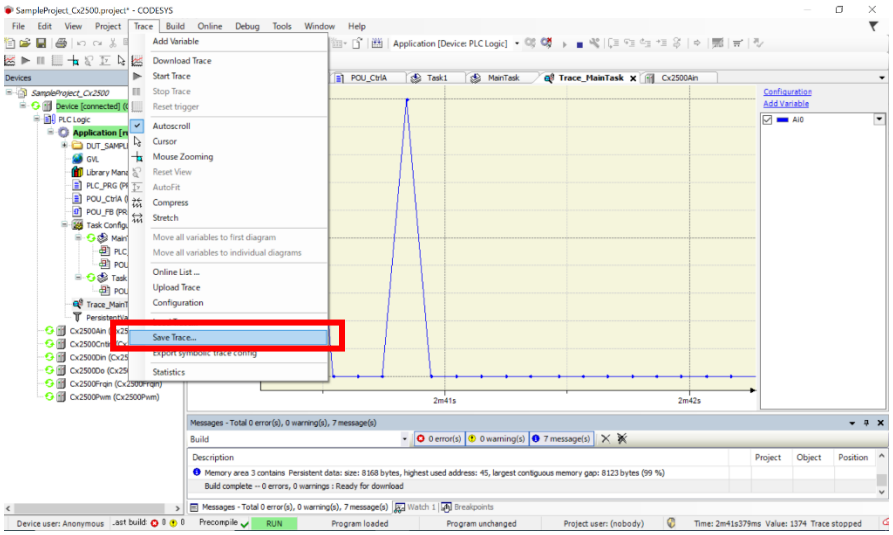


Figure 213 Save Trace の選択

② 「Save Trace」ウィンドウが表示されるので、任意の保存場所とファイル名を入力して「保存」ボタンを押して下さい。これでトレースの保存は完了です。

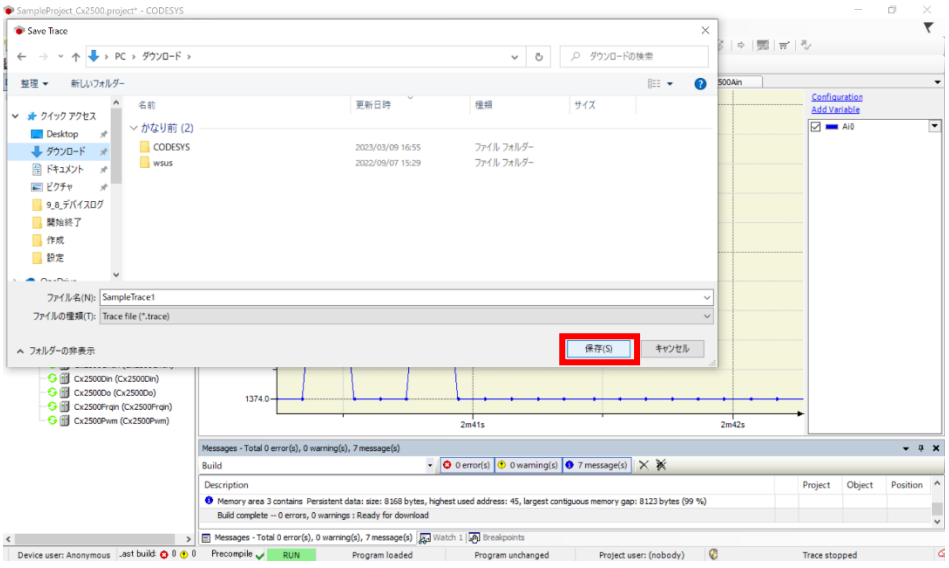


Figure 214 トレースファイルの保存

9.9.5. トレース履歴の読み込み

ここでは、保存したトレースファイルの読み込み手順を示します。なお、トレース中にトレースファイルを読み込んだ場合、トレースは停止し採取中であったデータは削除されることに留意して下さい。

① メニューバーから「Trace」→「Load Trace」を選択して下さい。

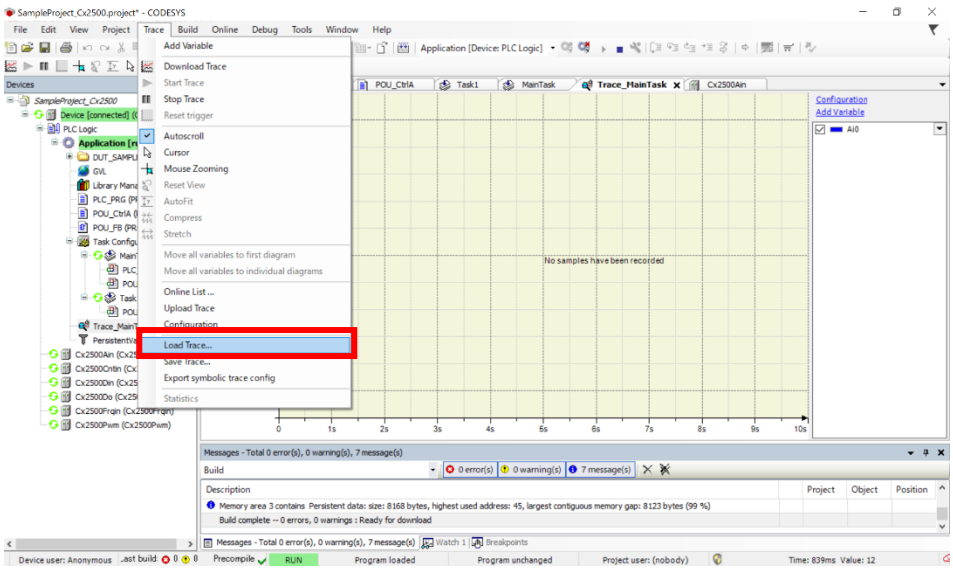


Figure 215 Load Trace の選択

② 「Load Trace」ウィンドウが表示されるので、読み込みたいトレースファイルを選択して「開く」ボタンを押して下さい。

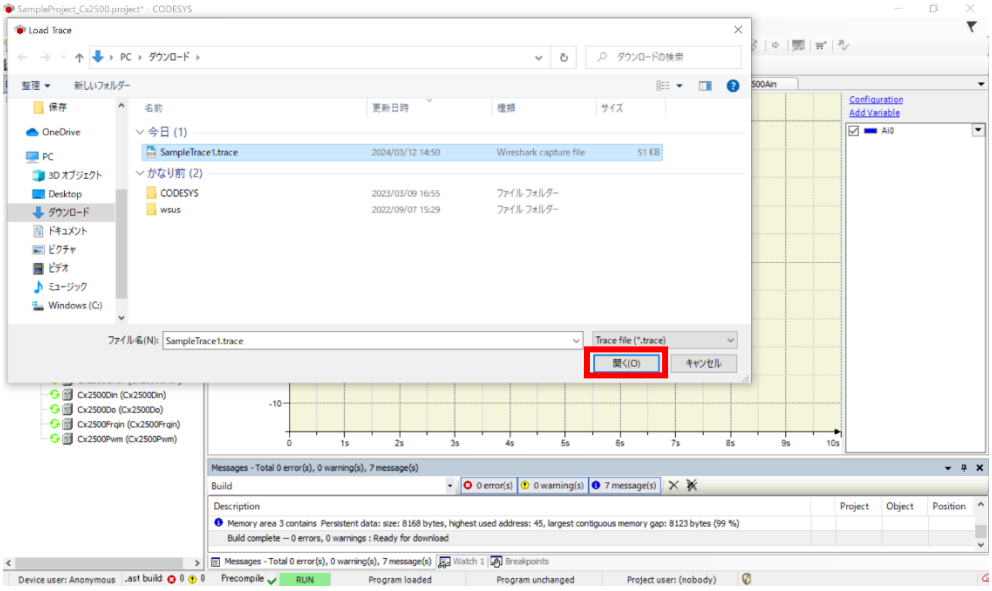


Figure 216 トレースファイルの選択

③ 読み込みが完了し、トレース画面上にデータが表示されたら読み込みは完了です。

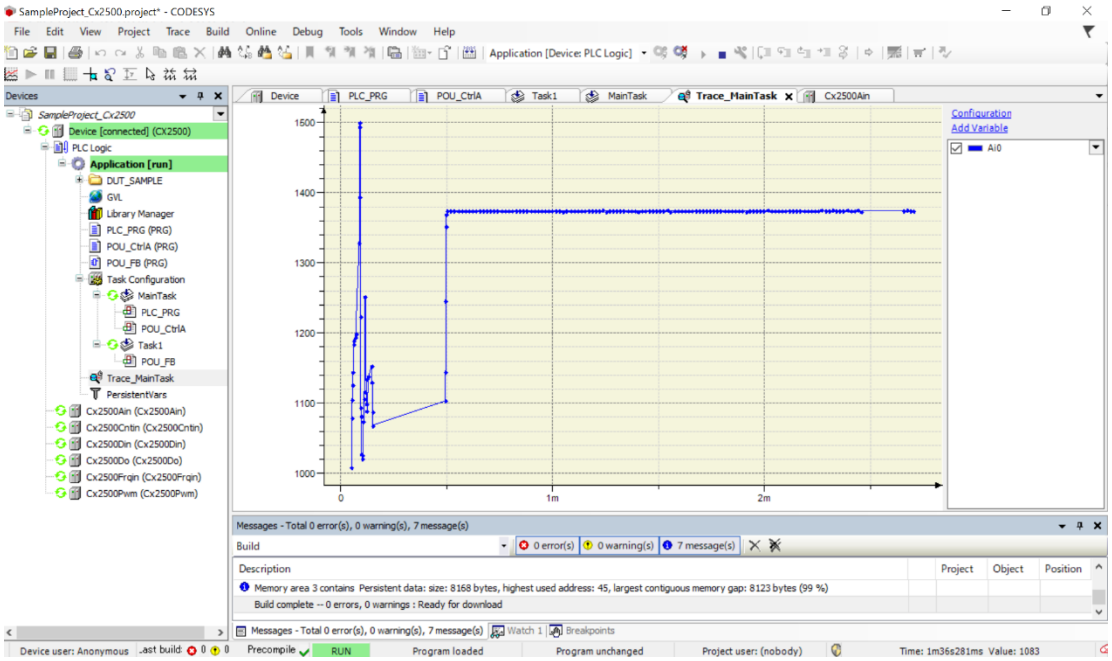


Figure 217 トレースファイル読み込み完了後

9.10. タスクステータス監視

各タスクがアプリケーション内でどの程度処理に時間を要しているか確認することができます。

確認は、デバッグモード中にデバイスウィンドウの「Task Configuration」をダブルクリックすると表示される画面でできます。各項目に関する説明を Table 82 に示します。

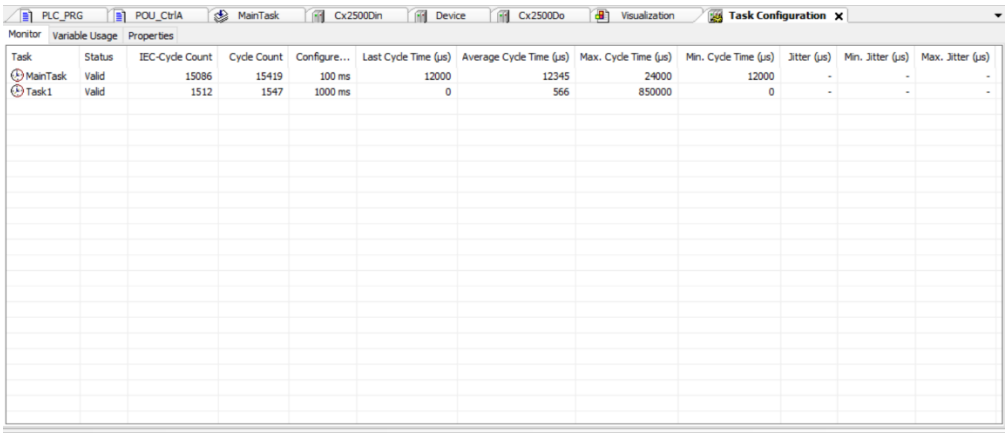


Figure 218 Task Configuration 画面

Table 82 タスクステータス監視 項目一覧

項目名	摘要
Task	アプリケーションで作成したタスク名が表示される。
Status	タスクのステータスが表示される。
IEC-Cycle Count	アプリケーション起動後、動作中にタスクが呼び出された回数が表示される。
Cycle Count	タスクが呼び出された回数。ただし、上記と異なりアプリケーション起動後にアプリケーション・タスクが動作を停止していてもサイクル数は加算されることに留意。
Configured Cycle Period	ユーザーが設定したタスクを呼び出す周期が表示される。
Last Cycle Time(µs)※23	最後に測定されたタスクの実行時間が表示される。
Average Cycle Time(µs)※23	全てのサイクルにおけるタスクの平均実行時間が表示される。
Max. Cycle Time(µs)※23	全てのサイクルにおけるタスクの最長実行時間が表示される。
Min. Cycle Time(µs)※23・※24	全てのサイクルにおけるタスクの最短実行時間が表示される。
Jitter(µs)	**非対応。**
Min. Jitter(µs)	**非対応。**
Max. Jitter(µs)	**非対応。**

※23 タスクの実行時間が 1ms 未満の場合、製品仕様上画面には 0 が表示されることに留意。また、これらの数値はアプリケーションが停止している時も含まれる(CODESYS-IDE の仕様)。

※24 Min. Cycle Time(µs)が 0ms より大きい値の時に実行時間 0ms(1ms 未満)が計測された場合 Min. Cycle Time(µs)は 0 になる。ただ、CODESYS-IDE の仕様上、Min. Cycle Time(µs)=0 の時には 0 より大きい(1ms 以上)実行時間が観測された場合に、その値に表示を更新してしまう。これにより、実行時間が短い時に Min. Cycle Time(µs)の値が例えば 0⇔2000µs を行き来するような場合があるが CODESYS-IDE の仕様である。

9.11. シミュレーション(オフラインデバッグ)

CODESYS-IDE では CX2500 と実際に接続せずとも、シミュレーションモードとして仮想環境でデバッグをおこなうことができます。シミュレーションモードへの移行とログイン手順について下記に示します。

なお、シミュレーションモードで CX2500 と実際に接続してデバッグをおこなうことはできません。その際にはシミュレーションモードを終了してから実施して下さい。シミュレーションモードの終了は手順①の「Simulation」を再度選択すると終了できます。

① ログインしていない状態で、メニューバーから「Online」→「Simulation」を選択して下さい。

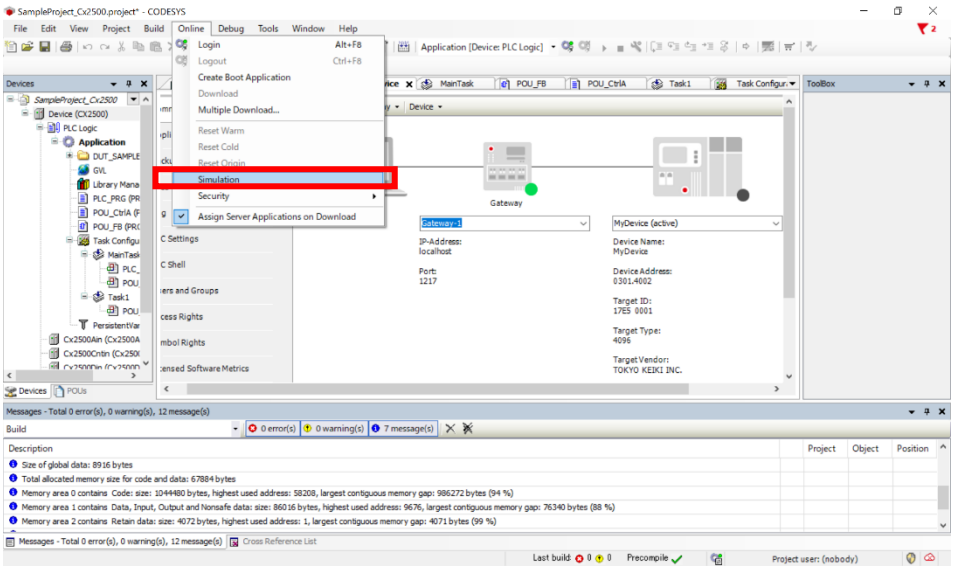


Figure 219 Simulation の選択

② 下図のようなシミュレーションモードに切り替わります。

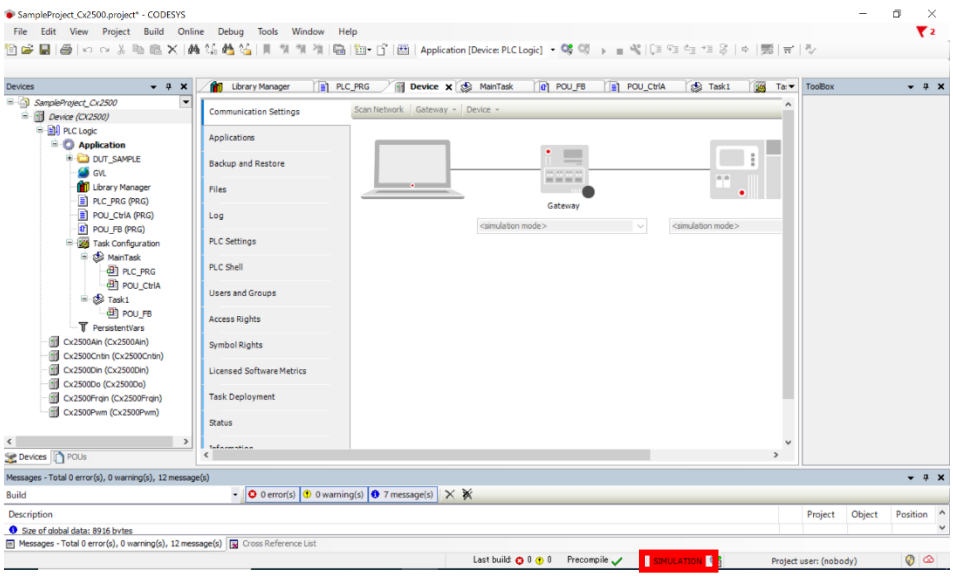


Figure 220 シミュレーションモード移行完了後の画面

③ メニューバーから「Online」→「Login」を選択して下さい。

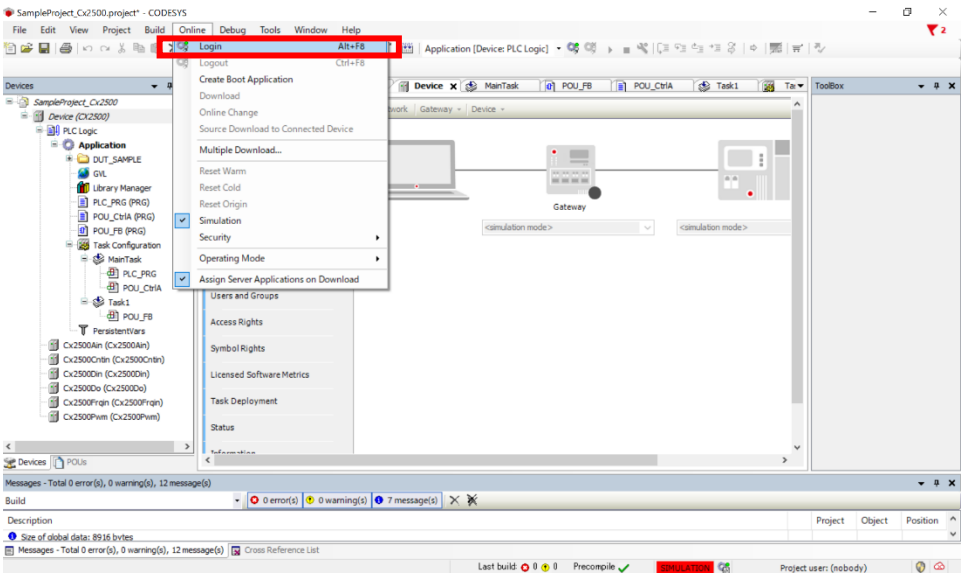


Figure 221 シミュレーションモード Login の選択

④ ログインの書き込み確認ウィンドウが表示されますので、「Yes」ボタンを押して下さい。

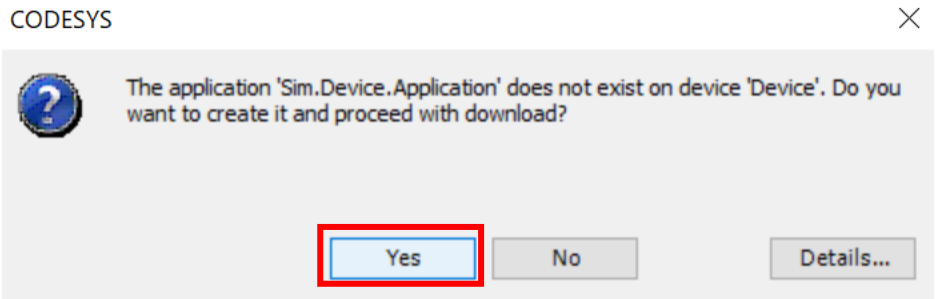


Figure 222 シミュレーションモード ログイン確認画面

⑤ デバッグモードに遷移します。これでシミュレーションモードでのデバッグ準備は完了です。デバッグ手法については通常の場合と同様です。なお、シミュレーションモードの場合、デバイスは図のようなアイコンが表示されます。機能ドライバも同様で、シミュレーションモードでは周期的な入力値の取得はおこなえません。

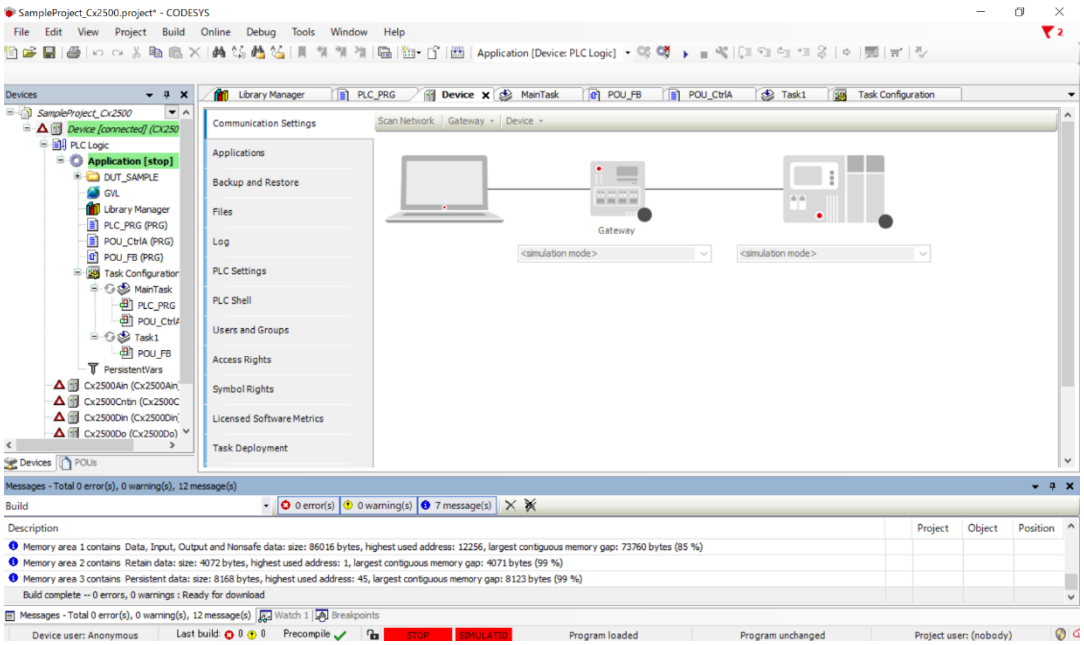


Figure 223 シミュレーションモード ログイン完了後画面

9.12. Visualization

Visualization は、プログラムの変数値をランプやメーターなどの仮想のアイテムと CODESYS-IDE 上で紐づけることによって、視覚的にデバッグすることができる機能です。なお、本製品は Web ブラウザ上に表示させる Web Visualization には対応していません。

9.12.1. Visualization 作成

ここでは、Visualization を作成する手順を示します。

① デバイスウィンドウにて、「Application」にカーソルを合わせ右クリックして下さい。

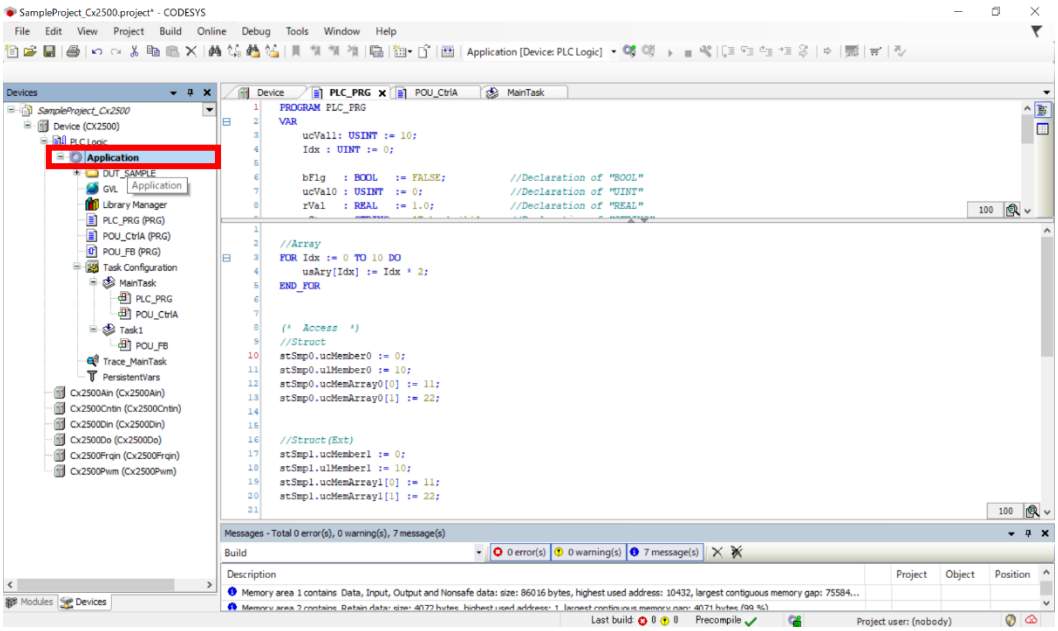


Figure 224 Visualization Application の選択

② 表示されるコンテキストメニューから「Add Object」→「Visualization…」を選択して下さい。

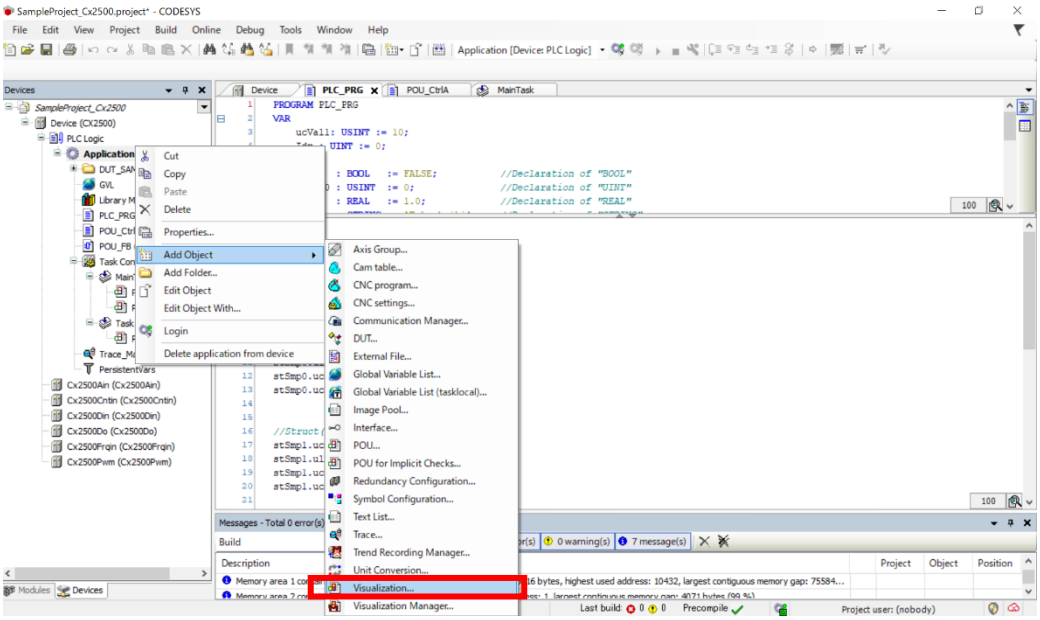


Figure 225 Visualization の選択

③ 「Add Visualization」ウィンドウが表示されるので、Name に任意の名称を入力し VisuSymbols にチェックを入れた状態で「Add」ボタンを押して下さい。

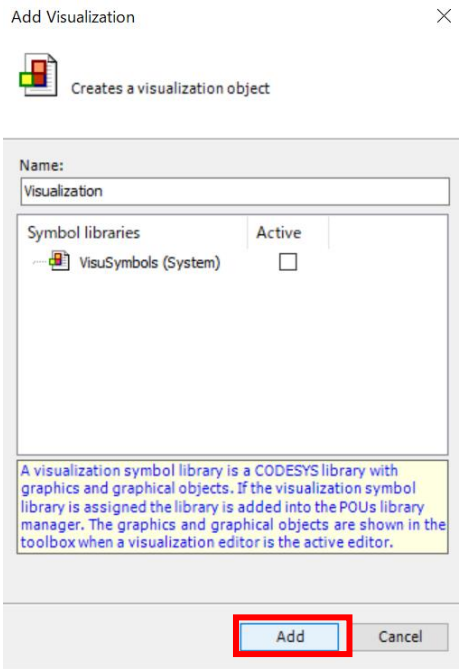


Figure 226 Add Visualization ウィンドウ

④ Visualization と Visualization Manager がデバイスウィンドウに追加され、メイン画面上に Visualization エディタ画面が表示されれば作成は完了です。

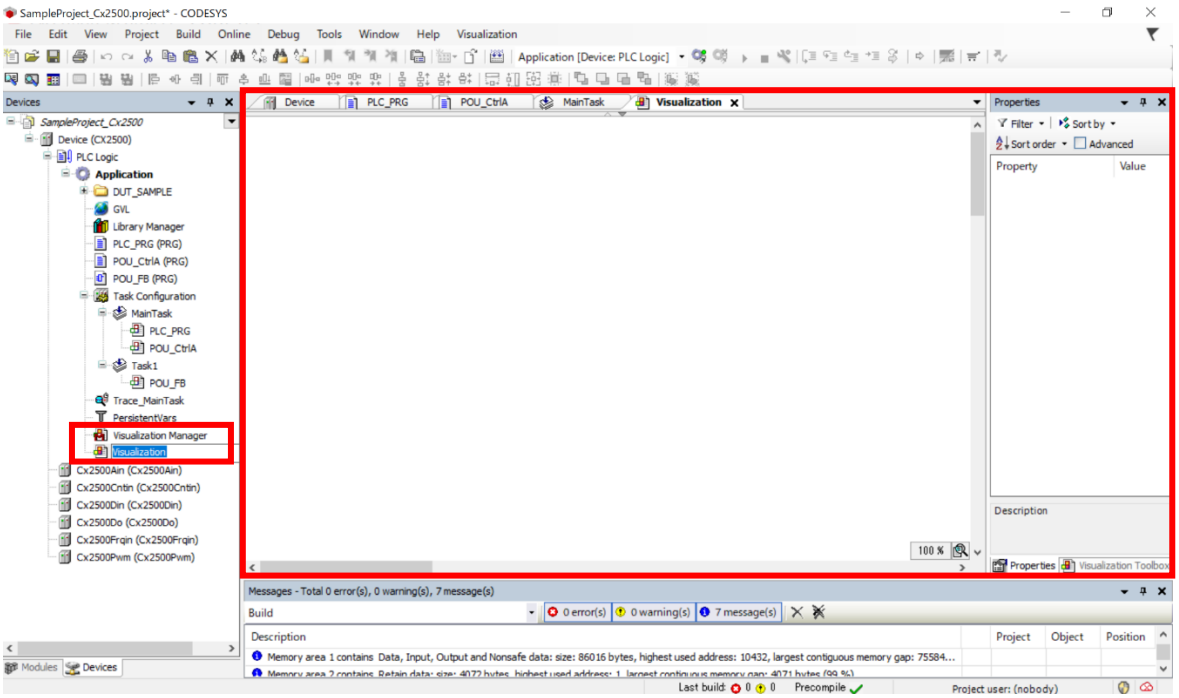


Figure 227 Visualization 追加完了後画面

9.12.2. 変数とアイテムの紐づけ例

ここでは、HMI 画面の作成、つまり、Visualization 上で以下の変数とアイテムを紐づけてデバッグの準備をおこなう例を示します。今回、変数はそれぞれの機能ドライバで定義していますが、通常のコード上の変数についても紐づけの方法は同様になります。

なお、他の Visualization アイテムについては CODESYS オンラインヘルプを参照して下さい。

Table 83 紐づけする変数とアイテムの組み合わせ

変数名	アイテム
Di0(デジタル入力 Ch.0 の入力値)	Power Switch
Do0(デジタル出力 Ch.0 の出力値)※25	Lamp

※25 本例では、Di0 が TRUE(ON 状態)の時に Do0 が ON にするプログラムを POU で作成していることとする。

① まず、Di0 の紐づけをおこないます。Visualization エディタ画面にて、Visualization Toolbox から Lamps/Switches/Bitmaps を選択し、表示される Visualization アイテムから Power Switch をエディタ画面にドラッグ＆ドロップして下さい。

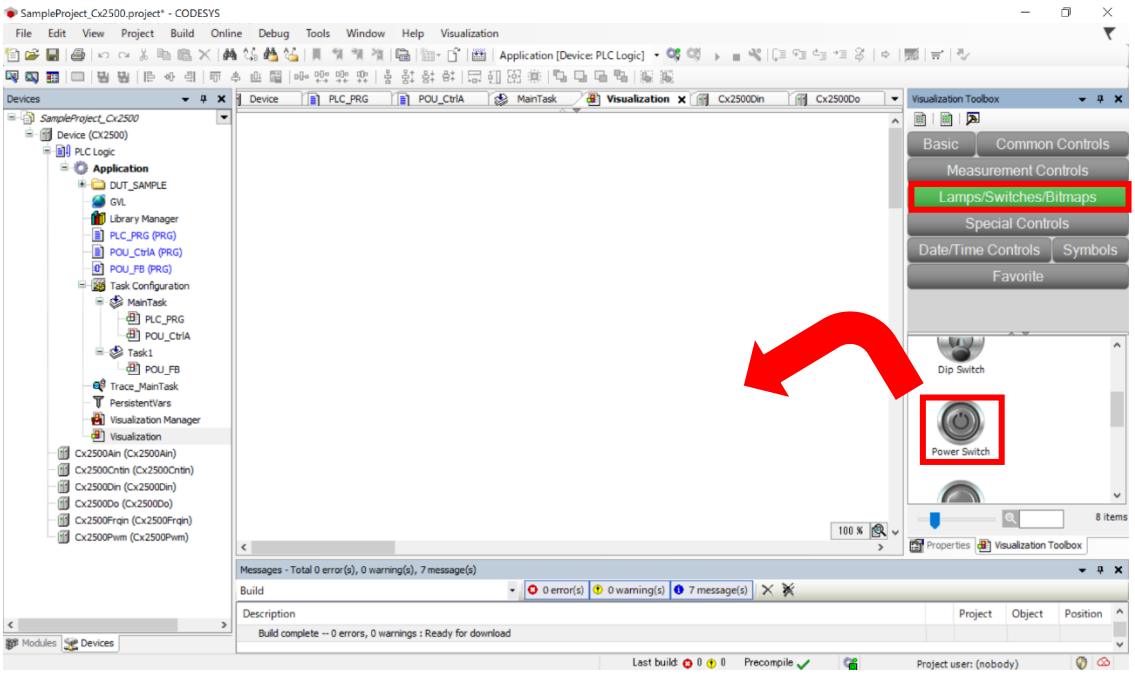


Figure 228 Visualization アイテム Power Switch の選択

② エディタ画面上に追加されたアイテムを選択すると、アイテムプロパティウィンドウが表示されます。そのウィンドウの「Position」の「Variable」に Di0 と入力(もしくは「...」ボタンを押して Di0 を選択)して下さい。

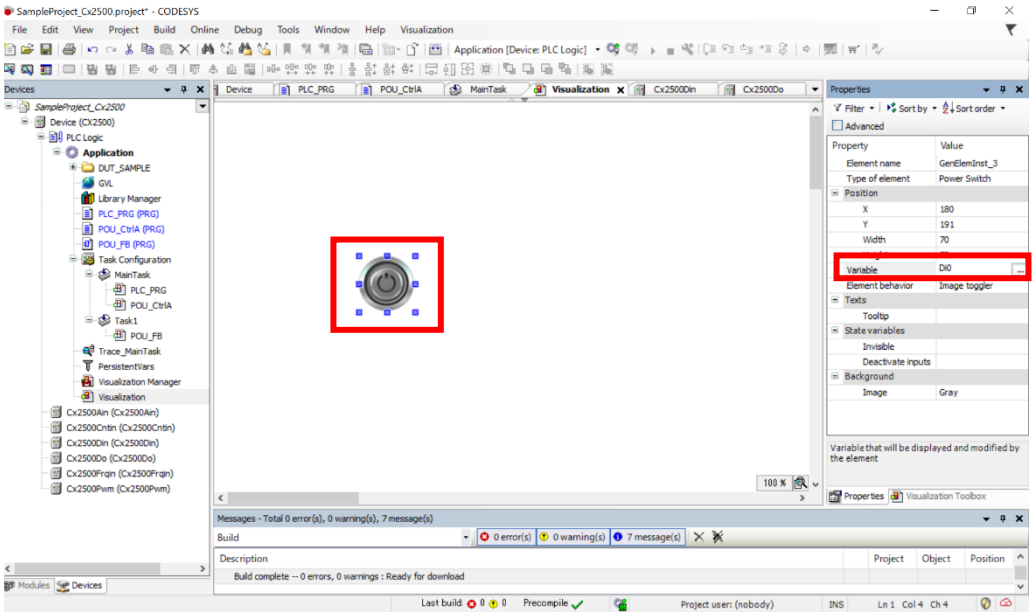


Figure 229 アイテムと変数 Di0 の紐づけ

③ 次に、Do0 の紐づけをおこないます。Visualization エディタ画面にて、Visualization Toolbox から Lamps/Switches/Bitmaps を選択し、表示される Visualization アイテムから Lamp をエディタ画面にドラッグ&ドロップして下さい。

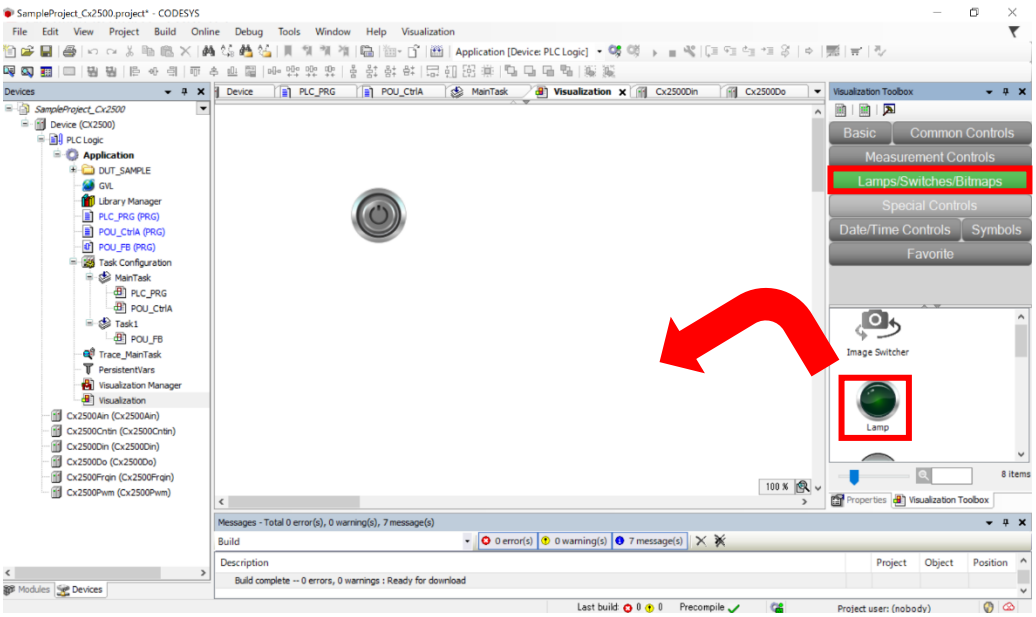


Figure 230 Visualization アイテム Lamp の選択

④ エディタ画面上に追加されたアイテムを選択すると、アイテムプロパティウィンドウが表示されます。そのウィンドウの「Position」の「Variable」に Do0 と入力(もしくは「…」ボタンを押して Do0 を選択)して下さい。これで紐づけは完了です。

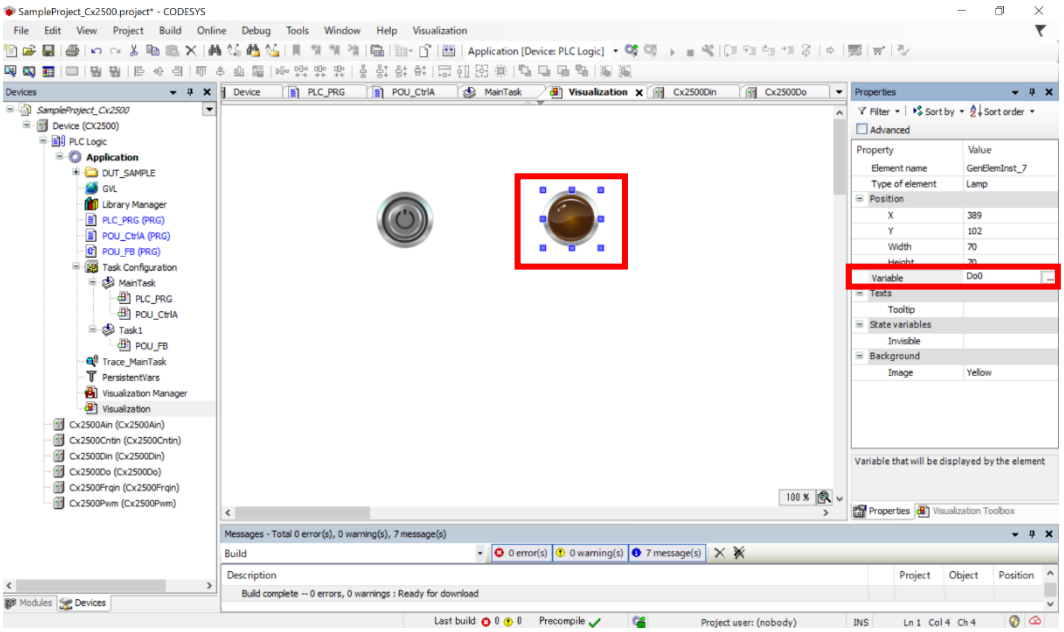


Figure 231 アイテムと Do0 の紐づけ

9.12.3. Visualization 実行例

ここでは、9.12.3 項で作成した Visualization をデバッグモードで実行する例を示します。

① デバイスと接続しログインし、アプリケーションの動作を開始して下さい。

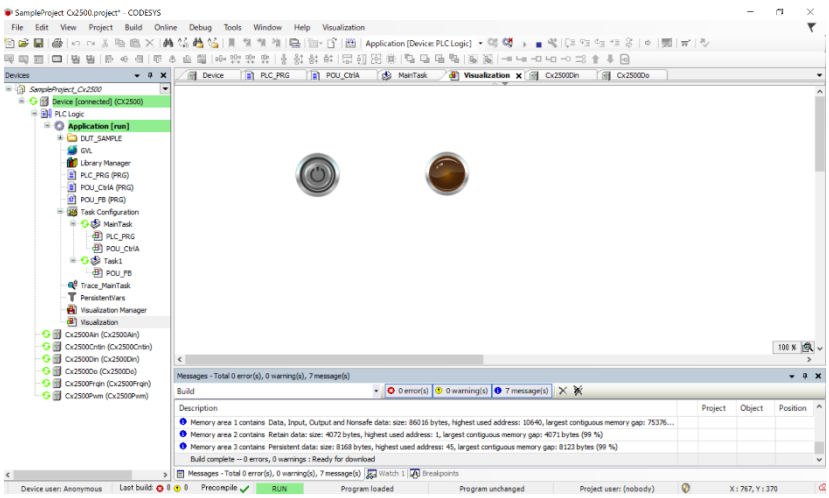
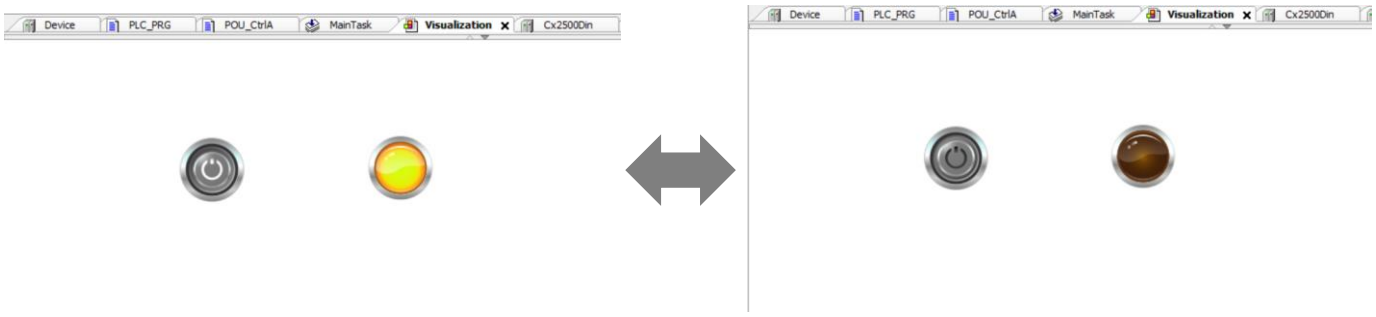


Figure 232 デバッグモード中画面

② 動作を開始して CX2500 のデジタル入力 Ch.0 の入力状態を変化させると下記のようにスイッチとランプが状態に応じて変化することが確認できます。



Di0 が TRUE(ON 状態)　　Di0 が FALSE(OFF 状態)

Figure 233 Di0 の状態変化に伴う Visualization アイテムの様子

10. Tips

10.1. プログラム中に IDE と通信できなくなった時は・・・

プログラム中やデバッグ中に IDE と通信できなくなった時は、以下のような可能性が考えられます。製品破損認定の前に、それぞれの場合に従って確認して下さい。

【通信できなくなった時に考えうる原因と対処】

- 通信ラインの接触不良
  - 取扱説明書「CX2500Codesys_UserManual_ForSetup」の 8.1 節にある対処法を試して下さい。

- その他
  - 取扱説明書「CX2500Codesys_UserManual_ForSetup」の 7.2 節を参考にユーザーの CX2500 にダウンロードされているバージョンのランタイムを弊社 HP からダウンロードし、CX2500 にランタイムを書き込んでください。
  - または、10.1.1 項のセーフモードを実施してください。

10.1.1. セーフモード起動

CX2500 を再起動しても IDE と通信できない場合、CX2500 をセーフモードで起動する必要があります。セーフモードで起動すると、下記の通りユーザーアプリケーションは保持されていますが、CX2500 は通電中、新たなアプリケーションの書き込みを待ち続けます。

下図の通り、CX2500 の起動前に指定の入力(条件 A)を ON 状態にした上で起動(イグニッション入力を ON)にするとセーフモードで起動できます。セーフモードで起動した場合、デバイスログにセーフモードで起動した旨のメッセージが表示されます。

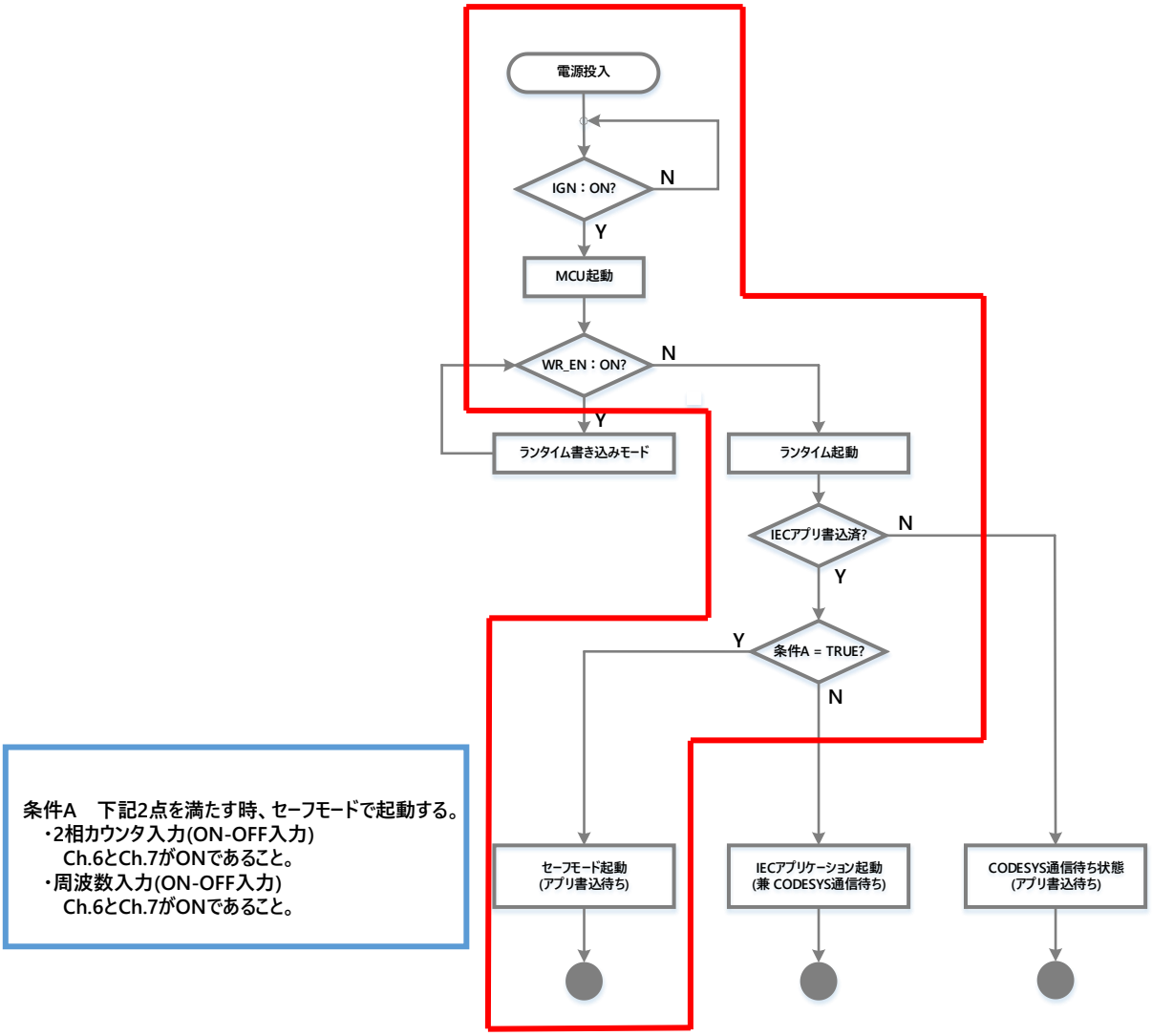


Figure 234　CX2500　起動フロー(赤枠がセーフモード起動フロー)

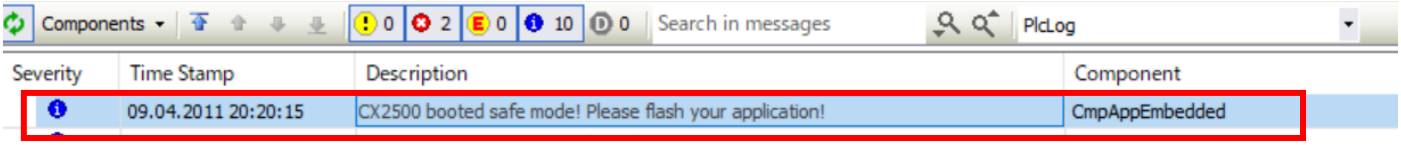


Figure 235　デバイスログ　セーフモード起動時のメッセージ

11. Revision history

Rev	Date	Substantive changes
0	2024.06.11	Initial release
1	2024.10.1	● 6.9.3 項 当社独自ライブラリ ・新規追加。 ● 6.9.4 項 ・旧 6.9.2.1 項の内容であり、項目番号が左記より変更。 ● 6.9.5 項 ・旧 6.9.3 項の内容であり、項目番号が左記より変更。 ● 7.2.3.4 項 アプリケーションで使用しない機能ドライバの設定 ・新規追加。 ● 7.12.3 項 ・Write 関数の返り値の誤記を修正。(SENDING_ERROR の削除) ・IsSendingActive 関数の返り値の誤記を修正。 ● 7.12.4.2 項 ・マスクレシーバー例の表中の誤記を修正。 ● 7.14 節 ・使用ライブラリ変更につき、関数他関連機能を変更。 ● 8.1 節 ・旧 Table.55 削除。