Device / Device options / Serialization[デバイス/デバイス・オプション/シリアライゼーション]

シリアライゼーションはプログラムの特殊なモードです。シリアライゼーション・モードがアクティブな時、各デバイスにプログラミングする前に指定した値が自動的にバッファの前もって定義されたアドレスに挿入されます。そして、次から次へとデバイスをプログラムする時、自動的にシリアル番号の値が.変更してデバイスのプログラミングの前にバッファに挿入されます。従って、各々のデバイスが特有のシリアル番号を持つことが出来ます。

シリアライゼーションには3つのタイプがあります:

- インクレメンタル[増加]・モード
- ファイルからのモード
- カスタム・ジェネレーター・モード

ダイアログ **Serialization[シリアライゼーション]** はシリアライゼーションがオンにされた場合にプロジェクト・ファイルで使用される関連したシリアライゼーション位置ファイルのための設定も含まれています。さらに詳しくは “シリアライゼーションとプロジェクト”をご覧下さい。

シリアライゼーションのベーシック・ルール:

- シリアライゼーションは最近選択されたデバイスにのみ関連付けられます。 新しいデバイスを選択すると、シリアライゼーション設定がリセットされます(シリアライゼーションは無効に設定されます)。
- 最近のデバイスのシリアライゼーション設定はデバイスのプロジェクト・ファイル、又は、アプリケーションが閉じられたときのコンフィギュレーション・ファイルと他の設定と共に保存されます。
- シリアライゼーション・エンジンは各デバイス・プログラミングが開始される前に新しい(次の)シリアル番号を要求します(ノート1を参照)。
- 使用されたシリアル番号はシリアル番号の値の後に(*)で示されます。シリアル番号を使用すると、次のデバイス・プログラミングは次のシリアル番号を使用します(ノート1を参照)。

ノート1): *オプションの**Serial number usage if programming action fails[プログラミング動作が失敗した場合のシリアル番号の使用]**により以前のデバイス・プログラミングの結果が失敗した場合、プログラミング前に新しいシリアル番号要求を呼び出すことを抑制できます。:*

- **Reuse generated serial number for next programmed device**[次のプログラムされるデバイスのために生成されたシリアル番号を再使用]オプションが選択されている場合、以前のデバイス操作結果が失敗した場合に新しい(次の)シリアル番号の要求が抑制されます。つまり、使用されたシリアル番号が再び使用され、正常なデバイス・プログラミングが完了するまで同じシリアル番号が使用されます。
- **Throw away (use the serial number only once, regardless result of the programming) Throw away[捨てる](プログラミングの結果に関係なくシリアル番号を1回だけ使用する)**が選択された場合、以前のプログラミング操作の結果に関係なく、各デバイス操作の前に新しいシリアル番号の要求が実行されます。

シリアライゼーションはある種のデバイスのタイプに対してはPG4UWコントロール・プログラムのメイン・バッファ、又は、使用可能な拡張バッファを操作することが出来ます。例えば、データEEPROMメモリーを搭載したマイクロチップPIC16FXXX等、一部のタイプのデバイスで使用可能な拡張バッファで動作します。
どのバッファをシリアライゼーション・ルーチンにより使用するかはダイアログ Serialization[シリアライゼーション]で選択可能です。Buffer[バッファ]設定ボックスが表示されていない場合、現在選択のデバイスのシリアライゼーション・モードは拡張バッファをサポートしていません。

Device / Device options / Serialization / Incremental mode & SQTP
[デバイス/デバイス・オプション/シリアライゼーション/インクレメント・モード & SQTP]

Incremental mode & SQTP[インクレメンタル・モードとSQTP]は各プログラム・デバイスに個別のシリアル番号を割り当てることが出来ます。各デバイスのプログラム操作に対してユーザーにより入力された開始番号が指定されたステップで増加され、そして、各デバイスのプログラミングに先立ち、選択されたフォーマットで指定されたバッファ・アドレスにロードされます。
インクレメンタル・モードのためにユーザーが修正することが出来るオプションには以下の項目があります:

S/N size[S/Nサイズ]
S/Nサイズ・オプションはバッファに書込まれるシリアル値のバイトの数を定義します。S/NサイズではBin(バイナリー)

シリアライゼーション・モードの値は1-8が有効で、そして、ASCII シリアライゼーション・モードでは1-16の値が有効値です。

Addres[アドレス]
アドレス・オプションはシリアル値が書込まれるバッファ・アドレスを指定します。アドレス範囲はデバイスの開始と終了のアドレスの範囲内でなければいけません。アドレスはシリアル値の最後（最上位、又は、最下位）バイトがデバイスの開始と終了のアドレス範囲の中に指定されなければいけませんので、正しく指定されなければいけません。

Start value[スタート値]
スタート値オプションはシリアライゼーションが開始されるイニシャル値を指定します。一般的にシリアライゼーションの最大値は32bit long wordで$1FFFFFFFです。実際のシリアル値が最大値を超えた場合は、シリアル番号の3つの最上位ビットがゼロにセットされます。このアクションの後、数値は常に0 .. $ 1FFFFFFFの間隔内にあります(これはオーバーフロー処理の基本スタイルです)。

Step[ステップ]
ステップ・オプションはシリアル値のインプリメンテーションの増加ステップを指定します。

S/N mode[S/N モード]
S/Nモード・オプションはバッファに書込まなければいけないシリアル値の形式を定義します。２つのオプションが利用できます:
- ASCII
- Bin

ASCII - シリアル番号がASCII文字列としてバッファに書込まれることを意味します。例えば、番号$0528CDはASCIIモードで 30h 35h 32h 38h 43h 44h (‘0’5’2’8’C’D’) としてバッファに書込まれます, 即ち、6バイトです。
Bin - シリアル番号が直接バッファに書込まれることを意味します。もし、シリアル番号が１バイト長以上の場合，2つの可能なバイト・オーダーの１つに書くことができます。バイト・オーダーはSave to buffer[バッファにセーブ]項目で変更することが出来ます。

Style[スタイル]
スタイル・オプションはシリアル番号ベースを定義します。2つのオプションがあります:
- Decimal[デシマル － 10進数]
- Hexadecimal[ヘキサデシマル - 16進数]

Decimal[デシマル]番号は'0' から '9'のキャラクターを使って入力と表示がされます。
Hexadecimal[ヘキサ・デシマル]番号は'A' から'F'のキャラクターを使います。
特別なケースはBinary Dec[バイナリー・デシマル]で、これはBCD番号スタイルを意味します。BCDはデシマル番号がヘキサ・デシマル番号にストアされることを意味します、即ち、各ニブルが0から9までの値を持たなければいけません。AからFの値はBCD番号のニブルとしては使用出来ません。
シリアル開始値とステップの数字を入れるまえに"Style"[スタイル]オプションでベースを選択して下さい。

Save to buffer[バッファへセーブ]
Save to buffer[バッファにセーブ]オプションはバッファに書込むためのシリアル値のバイト・オーダーを指定します。このオプションはBin S/N モード(ASCII モードには役立ちません)に対して使用されます。
2つのオプションが利用出来ます:
- *LSByte first*(インテルのプロセッサーで使用されています)はシリアル番号の最下位バイトを最初にバッファの最下位アドレスに置きます。
- *MSByte first*(モトローラのプロセッサーで使用されています) は最上位バイトを最初にバッファの最下位アドレスに置きます。

Split serial number[スプリット・シリアル番号]
オプションはシリアル番号を個々のバイトにスプリットし、そして、バッファの各N番目のアドレスにバイトを配置することが出来ます。この機能はデバイスのシリアル番号をRETLW、又は、NOP命令のグループとしてプログラムメモリの一部とすることが出来る時、マイクロチップ社のPICデバイスのためのSQTPシリアライゼーション・モードのために特に有用です。詳細については以下のサンプルのサンプル2を参照して下さい。

次のスプリット・オプションが利用可能:
- チェック・ボックス “**Split serial number**” –スプリット機能のターンオン/オフ

・ **Split gap** – スプリット・シリアル番号のフラグメント間に置くバイト数を指定
・ **S/N fragment size**–シリアル番号はこのオプションにより指定されたサイズでフラグメントに分割されます。

サンプル:
サンプル 1:
アドレス7FFFAHでAT29C040デバイスにシリアル番号を書く、シリアル番号のサイズは4バイト、開始値は16000000H、インクリメンタル・ステップは1、シリアル番号の形式はバイナリ、そして、最下位バイトはデバイスのシリアル番号の下位アドレスに配置されます。

上記に記載のシリアライゼーションを作成するにはシリアライゼーション・ダイアログで次の設定をする必要があります:
モード: インクリメンタル・モード
S/N size: bytes
S/N mode: Bin
Style: Hex
Save to buffer: LS Byte first
Address: 7FFFCH
Start value: 16000000H
Step: 1

次の値がデバイスに書き込まれます:
1番目のデバイス
アドレス　データ
007FFF0　xx xx xx xx xx xx xx xx xx xx xx xx 00 00 00 16
2番目のデバイス
アドレス　データ
007FFF0　xx xx xx xx xx xx xx xx xx xx xx xx 01 00 00 16
3番目のデバイス
アドレス　データ
007FFF0　xx xx xx xx xx xx xx xx xx xx xx xx 02 00 00 16等々
”xx” はデバイスにプログラムされるユーザー・データ
シリアル番号はシリアル数サイズが4バイトのためデバイスのアドレス7FFFCH から7FFFFHに書き込まれます。

サンプル 2:
次のサンプルはシリアル番号がマイクロチップPIC16F628デバイスに対してRETLW命令にスプリットされる時のSQTPシリアライゼーション・モードの使用方法を示します。

ノート:シリアル・クイック・ターン・プログラミング(SQTP)はマイクロチップ社のPICマイクロコントローラのシリアル・プログラミングのために、マイクロチップ社に指定された標準方式です。 マイクロチップPICデバイスを使用すると各マイクロコントローラに固有のシリアル番号をプログラムすることができます。この数はエントリコード、パスワード、又は、ID番号として使用することが出来ます。

シリアライゼーションはリテラル・データとして、シリアル番号のバイトで、RETLW(リターンリテラルW)命令を連続使用して行われます。シリアライズするには、インクリメンタル・モードのシリアライゼーション、又は、From file mode シリアライゼーションを使用することが出来ます。

インクリメンタル・シリアライゼーションはシリアル番号を分割するSplit機能を提供します。シリアル番号分割機能は偶数又は奇数バイトに分割された増加数を使用することが出来、シリアル番号の各バイトの間にRETLW命令コードが挿入されます。

"From file"シリアライゼーションは独自のシリアル番号ファイルを使用しています。このファイルは色々なシリアル番号で構成することが出来ます。この番号はSQTPに適した形式を持つことが出来ます。たとえばRETLW b1　RETLW b2 等です。注意：PG4UWのシリアル・ファイル形式はマイクロチップ社のMPLABによって生成されるSQTPシリアル・ファイルとは互換性がありません。

サンプル 2a:
Microchip PIC16F628デバイスに対してシリアライゼーションの分割を使用するとRETLW命令で分割します。

PIC16F628は14ビット幅命令ワードを持っています。RETLW命令は14ビット・オペコードを持っています:

説明		MSB 14-Bit word LSB
RETLW	リテラルをWで返す	11 01xx kkkk kkkk

xxは00に置き換えることが出来、そして、kはデータ・ビット、即ち、シリアル番号バイト

RETLW命令のオペコードはKKはデータ・バイト(シリアル番号バイト)であるヘキサデシマル 34KKH です。

4つのRETLW命令の一部としてシリアル番号1234ABCDHをデバイスPICに書き込むと仮定しましょう。 シリアル番号の最も高いバイトが最上位バイトです。アドレス40Hのデバイス・プログラム・メモリにシリアル番号を書きたいとします。 シリアル番号はこの状況で非常に便利に分割しました。 シリアル番号分割のないシリアライゼーションは次の番号をバッファとデバイスに書き込みます:

アドレス　　データ
0000080　CD AB 34 12 xx xx xx xx xx xx xx xx xx xx xx xx

ノート: アドレス80Hはバッファがバイト構成を持っており、PICはワード構成を有していますので、プログラム・メモリのアドレス40Hと同等であるためです。バッファがワード構成 x16を持っている場合、アドレスは40Hと番号1234ABCDHは次のようにバッファに配置されます:

アドレス　　データ
0000040　ABCD 1234 xxxx xxxx xxxx xxxx xxxx xxxx

RETLW命令を使いたいと仮定しますとバッファは:
アドレス　　データ
0000040　34CD 34AB 3434 3412 xxxx xxxx xxxx xxxx

これは次のステップで行うことが出来ます:
A) アドレス40Hに4つのRETLW命令をメイン・バッファに書き込みます（これは手作業によるバッファの編集、又は、適切な内容のファイルをロードすることによって行うことができます）。各RETLW命令の下位8ビットは重要ではありません、それは各RETLW命令の下位8ビットには、シリアライゼーションの正しいシリアル番号のバイトが書き込まれる為です。

デバイスのプログラムを開始する前のバッファの内容は例えば以下の様に見えます:

アドレス　　データ
0000040　3400 3400 3400 3400 xxxx xxxx xxxx xxxx

各RETLW命令の8ビットはゼロです。それらは如何なる値も持つことが出来ます。

B) 次のようなシリアライゼーション・オプションをセット:

S/N size:	4 Bytes
Address:	40H
Star value:	1234ABCDH
Step:	1
S/N mode:	BIN
Style:	HEX
Save to buffer:	LS Byte first
Split serial number:	checked
Split gap:	1 byte(s)
S/N fragment size:	1 byte(s)

上述のスプリット設定はシリアル番号をバイト単位で分割して2バイト毎にバッファすることを意味します。正しいシリアル番号はデバイス・プログラミング動作が開始する前に厳密に設定されます。

最初のデバイスがプログラミングされる時のシリアル番号のバッファ内容は:
アドレス　データ
0000040　34CD 34AB 3434 3412 xxxx xxxx xxxx xxxx
2番目のデバイスは:

アドレス　データ
0000040　34CE 34AB 3434 3412 xxxx xxxx xxxx xxxx
次のデバイスは同じフォーマットのシリアル番号を持ち、各デバイスに対して1でインクリメントされます。

Example 2.b
Microchip PIC24FJ256デバイスのためのNOP命令を持ったシリアライゼーション・スプリットの使用

デバイス　PIC24FJ256は24ビット幅の命令ワードを持っています。NOP命令はコード00xxxxhを持っています。Microchip MPLAB®で指定されているSQTPシリアライゼーションと同じ方法のシリアライゼーションを使用するとします:

次のステップでこれを行うことが出来ます:
A) PG4UWのメイン・バッファのアドレス800hにNOP命令(00xxxxh)を書き込みます。これは編集バッファを手動、又は、正しい内容を持ったファイルをロードすることによって行うことが出来ます。PG4UW/バッファ内のアドレス800hはPIC24Fxxxプログラム・メモリのアドレス200hと同等です。詳細についてはPG4UWのPIC24FJ256デバイス用のデバイス情報を見て下さい。

デバイスのプログラムを開始する前のアドレス800hのNOPでのバッファの内容は例えば以下の様に見えます:
アドレス　　データ
0000800　00 00 00 00 00 00 00 00 xx xx xx xx xx xx xx xx
xx – はバイト値を意味します。

B) 次のようなシリアライゼーション・オプションをセット:
S/N size:　3 bytes
Address:　800h
Start value:　123456h
Step:　1
S/N mode:　BIN
Style:　HEX
Save to buffer:　LS byte first
Split serial number: checked
Split gap:　2 byte(s)
S/N fragment size:　2 byte(s)

上述のスプリットの設定はフラグメント間の2バイトのギャップで16ビット(2/バイト)サイズのフラグメントにシリアル番号のスプリットをバッファします。正しいシリアル番号はデバイスのプログラミング操作が開始される前にしっかりと設定されます。

最初のデバイスがプログラミングされる時のシリアル番号のバッファ内容は:
アドレス　　データ
0000800　56 34 00 00 12 00 00 00 xx xx xx xx xx xx xx xx

2番目のデバイスは:
アドレス　　データ
0000800　57 34 00 00 12 00 00 00 xx xx xx xx xx xx xx xx

次のデバイスは各デバイスに対して1でインクリメントされる同じフォーマットのシリアル番号を持ちます。

サンプル 3:
次のサンプルはシリアル番号スプリット・ギャップが2と3に設定されている代わりサンプル2aと同じシリアライゼーション・オプションを使用しています。

スプリット・ギャップが2バイトにセットされている時、バッファ内容は次のように見えます:

バイト・バッファ構成:
アドレス　データ
0000080　CD xx xx AB xx xx 34 xx xx 12 xx xx xx xx xx xx

ワード16 バッファ構成:
アドレス　データ
0000040　xxCD ABxx xxxx xx34 12xx xxxx xxxx xxxx

スプリット・ギャップが3バイトにセットさている時、バッファ内容は次の様に見えます:

バイト・バッファ構成:
アドレス　データ
0000080　CD xx xx xx AB xx xx xx 34 xx xx xx 12

ワード16 バッファ構成:
アドレス　データ
0000040　xxCD xxxx xxAB xxxx xx34 xxxx xx12 xxx

ノート: シリアライゼーション・オプションの効果がわからない時は、バッファに書き込まれる実際のシリアル番号をテストすることが可能です。テストは次のステップで行うことが出来ます:

1.ダイアログ"シリアライゼーション"で希望するシリアライゼーションを選択し、OKボタンで確認します。
2.ダイアログ”デバイス操作オプション”でインサーション・テストとDevice IDチェックを無効にします。
3.ZIFソケットにデバイスが装着されていないことを確認して下さい。
4.デバイス・プログラム操作を実行(ある種のデバイスではプログラミングの開始前にプログラミング・オプションを選択する必要があります)
5.プログラミング操作が終了後(殆どの場合、デバイスが装着されていませんのでエラーとなります)、どこにシリアル番号が置くかはアドレスのメイン・バッファ(View/Edit/バッファ)で見て下さい。

ノート: シリアライゼーションのためのアドレスは常に制御プログラムが現在のデバイスに使用している実際のデバイスとバッファの構成に対して割り当てられます。もし、バッファ構成がバイトorg (x8)であれば、シリアライゼーション・アドレスはバイト・アドレス。もし、バッファ構成がバイトより広い、即ち、16ビット・ワード(x16)ならシリアライゼーション・アドレスはワード・アドレスになります。.

Device / Device options / Serialization / Classic From file mode
[デバイス/デバイス・オプション/シリアライゼーション/クラシック・フロム・ファイル・モード]
Classic From-file mode[クラシック・フロム・ファイル・モード]を使用する場合、シリアライゼーション・ファイルにはシリアル値が直接含まれています。シリアライゼーション・データはシリアライゼーション・ファイルからファイルに指定されたアドレスのバッファに直接読み込まれます。クラシック・フロム・ファイル・モードはPG4UWコントロール・プログラムのメイン・ウインドウとj情報ウインドウに**"From File"**シリアライゼーションとしてパネル"Serialization"で表示されます。
2つのユーザー・オプションがあります:

Start label[スタート・ラベル]
開始ラベルは入力ファイルの開始ラベルを定義します。 ファイルからのシリアル値は定義された開始ラベルから読み取りを開始します。

File name[ファイル名]
Classis from file[クラシック・フロム・ファイル]のためのシリアライゼーションの入力ファイルは正しい形式でなければいけません。

File format[ファイル・フォーマット]
Classic From-fileシリアライゼーション入力ファイルはテキスト形式です。このファイルはバッファ・アドレスとバッファに書き込むデータを定義するバイトのアドレスと配列が含まれます。入力ファイルにはテキスト形式の形式があり、その構造は次のとおりです:

[label*1*]　addr byte*0* byte*1* .. byte*n* ...
[label*n*]　addr byte*0* byte*1* .. byte*m* , addr byte*0* byte*1* ... byte*k*
　　¥_____/　¥_____/
　　|　　|
　ベーシック・パート　　オプショナル・パート
; Comment[コメント]

意味は:

ベーシック・パート
基本部分はバッファ・アドレスとバッファに書き込むバイトの配列を定義します。基本部分は常にラベルの行の後に定義する必要があります。

オプショナル・パート
オプショナル部分は2番目のバイトの配列とバッファに書き込むバッファ・アドレスを定義します。オプショナル部分の一部はデータの基本部分の後に定義することができます。

label*1*, label*n* – ラベル
ラベルは入力ファイルの各行の識別子です。これらはファイルの各行のアドレス指定に使用されます。ラベルはユニークでなければいけません。ファイルの行をアドレス指定するとは、ユーザーが入力する必要な開始ラベルはシリアル値の読み込みを開始する入力ファイルでの行を定義します。

addr -
Addrはアドレスに続くデータを書き込むバッファ・アドレスを定義します。

byte*0*..byte*n*, byte*0*..byte*m*, byte*0*..byte*k* -
バイト配列byte0..byten、byte0..bytemとbyte0..bytekはバッファに書き込むために割り当てられるデータを定義しています。アドレスに続く1つのデータ・フィールドの最大バイト数は64バイトです。データ・バイトはアドレスaddrからaddr+nまでのバッファに書き込まれます。
特定のバイトをバッファに書き込むプロセスは次のとおりです:
byte*0* to addr
byte*1* to addr + *1*
byte2 to addr + 2
....
byte*n* to addr + *n*

Optional part[オプショナル部分]は最初のデータ部分から文字“,“(カンマ)で区切られ、その構造は最初のデータ部分と同じです。即ち、アドレスとそれに続くデータバイトの配列です。

特別使用の文字:
[ ] - ラベルは角括弧の中に定義する必要があります。
',' – データのベーシック・パート[基本部分]とオプショナル部分を区切る文字‘;’ - セミコロン文字はコメントの先頭を意味します。„ ; „ から行末までの全ての文字は無視されます。コメントは個々の行、又は、定義行の最後に置くことができます。

ノート:
- *ラベル名は '['と ']'を除く全ての文字を含めることができます。ラベル名は大文字と小文字を区別しないように分析されます。即ち、文字 'a'は 'A'と同じで 'b'は 'B'と同じです。*
- *入力ファイルの全てのアドレスとバイト番号の値は16進数です。*
- *許容されるアドレス値のサイズは1～4バイトです。*
- *1行のデータ配列の許容サイズは1から64バイトの範囲です。1行に2つのデータ配列がある場合、それらのサイズの合計は最大80バイト迄です。*
- *正しいアドレスをセットするように注意してください。アドレスはデバイスの開始アドレスとデバイス終了アドレスの範囲内で定義する必要があります。アドレスが範囲外の場合、警告ウィンドウが表示されシリアライゼーションは無効にセットされます。*
- *シリアライゼーションのためのアドレスは制御プログラムが現在のデバイスに使用している実際のデバイス構成とバッファ構成に常に割り当てられます。バッファ構成がバイト構成の場合(x8)、シリアライゼーション・アドレスはバイト・アドレスになります。バッファ構成がバイトよりも広い場合、例えば、16ビットワード(x16)の場合、シリアライゼーション・アドレスはワード・アドレスになります。*

Classic From file[クラシック・フロム・ファイル]シリアライゼーションの典型的な入力ファイルの例]:
[nav*1*] A7890 78 89 56 02 AB CD ; comment*1*
[nav*2*] A7890 02 02 04 06 08 0A
[nav*3*] A7890 08 09 0A 0B A0 C0 ; comment*2*

[nav*4*] A7890 68 87 50 02 0B 8D
[nav*5*] A7890 A8 88 59 02 AB 7D

;次の行には2番目の定義も含まれます
[nav*6*] A7890 18 29 36 42 5B 6D , FFFF6 44 11 22 33 99 88 77 66 55 16

; これは最後の行です - ファイルの最後

この例のファイルでは labels „nav1“, „nav2“, ...“nav6“の6つのシリアル値が定義されています。各値はアドレス$A7890のバッファに書き込まれます。 全ての値のサイズは6バイトです。 „nav6“ラベルの行はまたアドレス$FFFF6にバッファリングされサイズが10バイトである第2の定義値を持っています。即ち、この値の最後のバイトはアドレス$FFFFFに書き込まれます。

ノート: シリアライゼーションのアドレスは制御プログラムが現在のデバイスに使用している実際のデバイス構成とバッファ構成に常に割り当てられます。バッファ構成がバイト構成の場合、(x8)、シリアライゼーション・アドレスはバイト・アドレスになります。バッファ構成がバイトよりも広い場合、例えば、 16ビットワード(x16)の場合、シリアライゼーション・アドレスはワード・アドレスになります。

Device / Device options / Serialization / Playlist From file mode
[デバイス/デバイス・オプション/シリアライゼーション/ファイルからのモード]
Playlist From-file mode[プレイリスト・ファイルからのモード]を使用するとシリアライゼーション・ファイルは含まれるシリアル値を直接は持っていません。ファイルはシリアライゼーション・データが含まれている外部ファイルの名前のリストが含まれています。シリアライゼーション・データはこれらの外部データ・ファイルから読み出され、各ファイルは1つのシリアライゼーション・ステップ(1つのデバイスがプログラムされる)を意味します。Playlist From-file mode[プレイリスト・ファイルからのモード]はPG4UW制御プログラムのメイン・ウィンドウと情報ウィンドウに"**From-file-pl[**プレイリスト・ファイルからのモード]"シリアライゼーションとして"Serialization[シリアライゼーション]"パネルに表示されます。

ファイル・フォーマット
From-file[ファイルから]シリアライゼーション・プレイリスト・ファイルはシリアライゼーション・データを持ったファイル名のリストを含みます。そのファイル・フォーマットはクラシック・シリアライゼーション・ファイル・フォーマットに似ています。ファイル・フォーマットの違いはプレイリスト・ファイルにおいては次の通りです:

1. playlistファイルはファイルの最初に空白行でない特別なヘッダを持つ必要があります。そのヘッダ行のフォーマットはテキスト形式です。
FILETYPE=PG4UW SERIALIZATION PLAYLIST FILE

2. 各シリアル・データ・バッチは別の行で次のフォーマットで表わされます。
[label x] datafilename

labelx - ラベルを現わします。
ラベルは入力ファイルの各行が空白でない事を示すための識別子です。これらはファイルの各行をアドレス指定するために使用されます。ラベルはファイル内でユニークである必要があります。ファイルの行のアドレス指定はユーザーにより入力された必要な開始ラベルがシリアル値の読み込みを開始する入力ファイルの行を定義することを意味します。

datafilename[データ・ファイル名] - シリアライゼーション・データを含むデータ・ファイルの名前を定義します。シリアライゼーションが新しいシリアル値を必要とする場合、データ・ファイルは標準のPG4UW "Load file[ロード・ファイル]"の手順で、PG4UWのバッファへロードされます。ファイル形式はバイナリー又は、ヘキサ・ファイル(Intel Hex等)に対応しています。自動認識システムは適切なファイル形式を認識し、そして、正しいファイル形式のファイルのロードを行います。データ・ファイル名はペアレント(playlist)のシリアライゼーション・ファイルと関連しています。

playlist シリアライゼーション・ファイルのサンプル:
;---- 次のファイル・ヘッダーが必要です。----------
FILETYPE=PG4UW SERIALIZATION PLAYLIST FILE

;----- シリアライゼーション・データ・ファイルの参照
[nav1] file1.dat

[nav2] file2.dat
[nav3] file3.dat
...
[label n] filex.dat
;-------- end of file ------------------------
シリアライゼーション・タイプ　From-file playlistのより詳細で完全に機能するサンプル例については、次のようにPG4UWインストレーション・ディレクトリのExamples¥subdirectoryにあるサンプル・ファイルを参照してください:
<PG4UW_inst_dir>\Examples\Serialization\fromfile_playlist_example\
一般的なパスは以下の様になります:
C:\Program Files (x86)\Elnec_sw\Programmer\Examples\Serialization\
fromfile_playlist_example¥

次のステップでシリアライゼーションをテストすることが出来ます:
1. PG4UWを実行
2. ELNECプログラマが接続されて正しくPG4UWで認識されている必要があります。
3. 希望するデバイスを選択、イレース可能なメモリ・デバイスをお薦めします。(OTPメモリではありません)
4. Device | Device Options | Serializationメニューからダイアログを選択
5. パネルFrom-file modeオプションでFrom-file modeをセットしサンプルのシリアライゼーション・ファイルfromfile_playlist.serを選択して下さい。
6. 新しいシリアライゼーションの設定を受け付けるためにOKボタンをクリックします。
7. デバイス操作で "Program[プログラム]"を実行して下さい。

PG4UWのメイン・ウィンドウでシリアライゼーションがラベルを表示し、またデバイスのプログラミング中とプログラミングのリピートを情報プログレス・ウィンドウで見ることが出来ます。

使用されたファイルで追加の操作
このグループ・ボックスには操作の3つのタイプが含まれています。ユーザーは"Playlist From-file mode"で使用されたシリアライゼーション・データ・ファイルの操作の1つを選択することが出来ます。次の操作が利用可能:
· **option Do nothing**
プログラムは使用されたシリアライゼーション・データ・ファイルでいずれの操作も行いません。
· **option Move used file to specified directory**
プログラムは使用されたシリアライゼーション・データ・ファイルをユーザー指定の使用されたシリアライゼーション・ファイルのディレクトリーに移動します。
· **option Delete used file**
プログラムは使用されたシリアライゼーション・データ・ファイルを削除します。

ディレクトリー
このオプションは"playlist From-file"シリアライゼーション･モードでオプション"Move used file to specified directory[指定されたディレクトリーに使用するファイルを移動]"が選択されますと利用出来ます。ユーザーがどのシリアライゼーション・データ・ファイルに移動するかのターゲット・ディレクトリーを指定することが出来ます。

次のエラー表示がPlaylist From-fileシリアライゼーションで使用されます:
- s/n error #3　シリアライゼーション・データ・ファイルは存在しません。
- s/n error #34　使用されたシリアライゼーション・データ・ファイルを削除出来ません(シリアライゼーション・ファイルは書き込みプロテクト・ディスクに置かれているかも知れません)
- s/n error #35　使用されたシリアライゼーション・データ・ファイルを使用されたシリアライゼーション・ファイルのターゲット・ディレクトリーへ移動出来ません(シリアライゼーション・ファイルは書き込みプロテクト・ディスクに置かれているか、又は、ターゲット・ディレクトリーが存在しないかも知れません)

Device / Device options / Serialization / Custom generator mode
[デバイス/デバイス・オプション/シリアライゼーション/カスタム・ジェネレーター・モード]
ユーザーが自分でシリアライゼーション・システムを全て持つ場合は、カスタム・ジェネレータ・シリアライゼーション・モードが最もフレキシブルなシリアライゼーション・モードを提供します

シリアライゼーションのCustom generator mode[カスタム･ジェネレータ]モードが選択された時、PG4UW、又は、

PG4UWMCで各デバイスがプログラムされる前にユーザーが作成したプログラムによって"on-the-fly[オンザフライ]"でシリアル番号が生成されます。カスタム･ジェネレーター・モードのシリアライゼーションはユーザーが望むユニークなシリアル番号のシーケンスを生成することが出来ます。シリアル番号はリニア・シーケンス、又は、完全な非リニア・シーケンスとしてインクリメントすることが出来ます。ユーザー作成シリアル番号ジェネレーター・プログラムの詳細は以下の**Custom generator program**セクションで説明します

サンプル:
また、.exeとC/C++のソースファイルも利用可能です。ファイルは次のPG4UWインストレーション・ディレクトリーExamples¥subdirectoryに有ります*:*
<PG4UW_inst_dir>\Examples\Serialization\customgenerator_example
一般的なパスはこのように見えます*:*
C:\Program Files (x86)\Elnec_sw\Programmer\Examples\Serialization\customgenerator_example¥

PG4UWコントロール・ソフトウエアの**Custom generator serialization[カスタム・ジェネレーター・シリアライゼーション]** のための次のオプションがあります:
ダイアログ”Serialization”の**Mode**パネル・オプションでCustom generator modeを選択。次のオプションが表示されます:

Serialization Data File[シリアライゼーション・データ・ファイル]
現在のシリアル番号が含まれるデータ･ファイルのパスと名前を指定します。デバイスをプログラムする時、PG4UWソフトウェアはユーザーが作成したデータ・ファイルを更新するシリアル番号ジェネレーターを呼び出します。データ・ファイルの推奨される拡張子は.datです。弊社のカスタマーの多くがBP Microsystems社のプログラマも使用しているためユーザーは普通同じシリアライゼーション・ソフトウェアを使用することを希望するためです。従って、シリアライゼーション・データ・ファイルはBPマイクロ社のソフトウェアで利用できる"Complex serialization"の.datファイルと互換性が有ります

ノート: データ・ファイルは全て定期的にデバイスのプログラミング中にシリアライゼーションで上書きされます。希望する .datファイルの正しい名前を確実に入力して下さい。例えば：“c:¥serial_files¥serial.dat”です。

Serialization generator[シリアライゼーション・ジェネレーター]
シリアライゼーション・データ・ファイルが生成される実行ファイルのためのパスと名前を指定します。

最初のシリアル番号
このオプションはカスタム･ジェネレーター・シリアライゼーション・プログラムに渡される最初のシリアル番号を指定する必要があります。番号は入力されますと16進数形式で表示されます。

最後のシリアル番号
このオプションは許可されたシリアル番号の最大値を指定します。値がゼロでない場合に、シリアライゼーション・ジェネレーター・プログラムへ渡されます。ジェネレーターは最後のシリアル番号の値を、テストし、そして、現在のシリアル番号が最後のシリアル番号より大きい場合には、そのシリアライゼーション.datファイルに適切なエラー内容を持ったシリアル .datファイルを生成します。最後のシリアル番号の値がゼロの場合、その値はジェネレーター・プログラムに渡されません。
チェック・ボックス **Call generator with -RESULT parameter after device operation completed [デバイスの操作が完了した後にRESULTパラメータと共に、ジェネレータをコール]** この新しいオプションは特別な目的を持っています。特殊なパラメータ -RESULTでカスタム・ジェネレータを呼び出す要件がある場合は、チェックボックスにチェックを入れておく必要があります。それ以外の場合は、チェックを外してオフにしておく必要があります (デフォルトの状態はオフです) チェックした場合、各デバイス操作が完了した後、カスタム･ジェネレーターはデバイス操作の結果がOK、又は、エラーに関係なく PG4UW制御プログラムによって呼び出されます。ジェネレーターのパラメータはPG4UWシリアライゼーション・エンジンによって作成されます。2つのパラメータが使用されます:

-RESULT[*n*]=TRUE | FALSE

*n*はマルチプログラミングが使用されている場合のオプショナルなプログラマ・サイトの番号。TRUEはデバイス操作がOKで終了したことを意味します。FALSEはデバイス操作がエラーで終了したことを意味します。

-N<serial number>

シリアライゼーション・ジェネレーターの通常の呼び出しと同様で、現在のシリアル番号を指定

Custom generator program[カスタム・ジェネレータ・プログラム]
カスタム・ジェネレーター・プログラム、又は、シリアライゼーション・ジェネレータはシリアル番号のユニークなシーケンスを発生しそのシリアル・データをシリアライゼーション.datファイルに書き込むプログラムです。

このプログラムはユーザー側で作成します。シリアライゼーション・プログラムのパスと名前は、カスタム・ジェネレーター・モード・オプションのシリアライゼーション・オプションのダイアログで指定する必要があります。
プログラムは新しいシリアル・データが生成される必要ある度にPG4UWから呼び出されます。これは通常各デバイスのプログラミング操作の前に行われます。PG4UW制御プログラムはシリアライゼーション・プログラムにコマンド・ライン・パラメータを渡し、そして、シリアライゼーション・プログラムはPG4UW制御プログラムによって読み込まれるシリアライゼーション.datファイルを生成します。以下のコマンド・ライン・パラメータが使用されます:

-N<serial number> 現在のシリアル番号を指定
-E<serial number> 最終(又は、最後)のシリアル番号を指定。
パラメータはPG4UWソフトウエアのダイアログ "シリアライゼーション"で最後のシリアル番号の値がゼロで無い時のみ渡されます。シリアライゼーション・プログラムは、もし現在のシリアル番号が最後のシリアル番号よりも大きい場合、シリアライゼーション.datファイルにエラー‥レコードT06を返します。詳細については"シリアライゼーション.datファイル形式"のセクションを見て下さい。

シリアライゼーション .dat ファイル・フォーマット
シリアライゼーション･ジェネレータによって生成されたシリアライゼーション.datファイルは次のテキスト形式でなければいけません。シリアライゼーション.datファイルはレコードとシリアル･データ･セクションで構成されています。

レコードは以下に説明する様にTxxプリフィックスの1つで始まる行です。“xx”の値はレコード･タイプのコードを表します。レコードはPG4UWソフトウェアにシリアライゼーションの状態(現在と最後のシリアル番号、シリアライゼーション・データとデータフォーマット、エラー等)を知らせるために使用されます。必要なレコードはレコードT01, T02, T03とT04です。その他のレコードはオプションです。

T01:<serial number> コマンド・ライン・パラメーター -N<serial number>によってジェネレーターに渡す現在のシリアル番号が含まれています。
T02:<serial number> PG4UWが次のシリアライゼーションで使用する次のシリアル番号値を含んでいます。この値はPG4UWに現在のシリアル番号の次のくシリアル番号を知らせシリアライゼーション・ジェネレーターで生成されます。
T03:<data format code> シリアライゼーション・・データ形式を指定。次のフォーマットがサポートされています:
T03:50 又は、T03:55 ASCIIスペース・データ形式
T03:99 - Intel Hexデータ形式
T04: シリアライゼーション・データが次の行からファイルの最後に続くことを示します。
シリアライゼーション・データは例えばIntel Hex, ASCII Space等々の標準のASCIIデータ・ファイル形式の1つで保存されます。データに使用するフォーマットはレコードT03で指定する必要があります。

サンプル: 典型的なシリアライゼーション・データ・ファイル:
T01:000005
T02:001006
T03:99
T04:
:0300000000096B89
:03000300000005F5
:02000C005A0197
:01003F004F71 :
00000001FF

ファイルは以下の情報で構成されています:
line T01 - 現在のシリアル番号 000005h
line T02 - 最終(最後)のシリアル番号 001006h
line T03 - 行 T04の後のシリアライゼーション・データ形式はIntel Hexです。

line T04 - デバイス・プログラミングの前にPG4UWのバッファにロードされるシリアライゼーション・データ、データはインテルHEXフォーマットで表現されます。

オプショナル・レコードは:

T05:<message> ワーニング、又は、エラー・メッセージ。このレコードはシリアライゼーションが中止されたために起こり、そして、PG4UWソフトウエアでワーニング、又は、エラー・メッセージが表示されます。

T06: 現在のシリアル番号が制限より大きい。このレコードはシリアライゼーションを停止し、PG4UWソフトウェアにより警告、又は、エラー・メッセージが表示される原因となります。シリアライゼーションをオフにする理由は現在のシリアル番号が許可された最大値の最終シリアル番号より大きいためです。このレコードは-Eコマンド・ライン・パラメータが指定されている場合に使用することが出来、それはシリアライゼーション・ダイアログでシリアル値の指定がゼロでないことを意味します。

T11:<message> 余り重要でないワーニング、又は、メッセージ。 シリアライゼーションは停止されません。

カスタム-ジェネレーター・シリアライゼーションでのデバイス・プログラミングのフローチャート

カスタム·ジェネレータのシリアライゼーションが使用される場合、各デバイスのプログラミングが開始される前に、シリアライゼーション・エンジンがシリアル.datファイルを生成するために実行可能なシリアライゼーションを呼び出します。PG4UWのシリアライゼーション・エンジンはシリアライゼーション・ジェネレーターを呼び出すために適切なコマンド・ライン・パラメータを管理します。.datファイルからのデータは直ちに内部プログラマー・バッファに読み出され、そして、プログラミング・デバイス用のデータとして使用されます。 また、次のシリアル番号情報(レコード T02)はPG4UWに記憶されています。

デバイス・プログラミングの典型的なフローチャートは次の通りです:

1. プログラミング・バッチを開始
2. デバイス装着テスト
3. シリアライゼーション・シーケンスは4つのステップからなります:
   - シリアライゼーション .datファイルを発生させるために適切なコマンド・ライン・パラメータでシリアライゼーション・ジェネレーターを呼び出す
   - 利用可能なシリアライゼーション.datファイルを待つ
   - シリアライゼーション .datファイルのデータをプログラマー・バッファへ読み込む(データはプログラミング・デバイスのために使用)
   - データを読み込んだ後シリアライゼーション.datファイルを削除
4. デバイス・プログラミング
5. デバイス・ベリフィケーション
6. 操作結果のチェック

   これは全てPG4UWコントロール・プログラムによって管理されます。シリアライゼーション・ジェネレーターの操作結果はどの操作とも関係がありません。コントロール・プログラムは要求されたコマンド・ライン・パラメータでシリアライゼーション・ジェネレーターを呼び出します。

   OK - PG4UWは次のシリアル番号の要求をします。次のシリアル番号はステップ3で .datファイルから読み込まれています。シリアライゼーション・ジェネレーターの呼び出しにより、コマンド・ラインで指定された次のシリアル番号を持ちます。

   ERROR - PG4UWは新しいシリアル番号の要求をしません。最後のシリアル番号は次のデバイスで使用されます。次のシリアライゼーション・ジェネレーターの呼び出しはコマンド・ラインで指定された最後のシリアル番号を持ちます。
7. 次のデバイスへのプログラミングを繰り返しますか?

   Yes ステップ2へ行く

   No ステップ8を継続
8. プログラミング・バッチの終了

ノート:

エラー・プログラミングの場合、最後のシリアル番号が使用されますが、ジェネレーターはステップ3で呼び出されます。とにかくもし同じ番号が以前にプログラムされたデバイス用として用いた場合であっても、呼び出されます。
もし、シリアライゼーション .datファイルのエラーが検出された場合、プログラムPG4UWはシリアライゼーション・エラーを報告し、即プログラミングのバッチ処理を中止します。