SHMIT

Standard Human-Machine Interface Template

マニュアル（基本編）

編集履歴

3 Sep. 2011 ver.1.0 beta　by H.OGAWA

4 Sep. 2012 ver1.1beta_rc1 by H.OGAWA

26 Mar. 2015 ver.2.0 by H.Ogawa

目次

1　SHMIT の特徴

SHMIT は、FA（Factory Automation）や PA（Plant Automation）で使用される HMI（Human-Machine Interface）のプログラミングを支援するアプリケーションです。主に中小規模の PLC 計装システムを対象としています。

SHMIT には、あらかじめ HMI に必要な機能がプログラミングされているため、ユーザは最小限のプログラミングで、簡単に目的の HMI を構築することができます。新しい機能を追加することもできるため、機能的な制限を意識することなく、自由なアプリケーション設計が可能です。

SHMIT を用いた開発の中で特徴的なのが「システム記述ファイル作成」と「フォームデザイン」です。システム記述ファイルとは、PLC の接続設定やアドレス一覧、機器一覧、表示項目一覧など、システム固有の情報が XML 形式で表現されたものです。これらの情報をプログラムから分離することで、全体構成の把握が容易になり、仕様変更が発生した場合でも、プログラムコードの変更を最小限に抑えることができます。システム構成ファイルのフォーマットについては「SHMIT マニュアル(応用編)」(以下、応用編マニュアルと呼ぶ)で詳しく説明します。

フォームデザインについては、SHMIT であらかじめ用意されている画面部品（バルブやポンプなど）を利用すれば、簡単に目的のフォームを作成することができます。さらに、ユーザが新しい部品スタイルを作成し、画面部品に適用することで、より機能的で魅力的な画面を作成することが可能です。新しく作成した部品スタイルはリソースディクショナリと呼ばれるライブラリに追加することで、将来の開発において再利用することが可能となります。

2　インストール・アンインストール

2.1　動作環境

- 開発環境

  Visual Basic2010 以上（Express Edition 可）

  Expression Blend4 以上（なくても構いませんが、利用を強く推奨します)

- ビルドされた実行形式ファイルの動作環境

  .Net Framework 4.0 以上がインストールされていること

2.2　インストール

① 解凍フォルダの、shmit¥shmit¥shmit.sln をダブルクリックしてプロジェクトを開く。

② [ファイル]-> [テンプレートのエクスポート]をクリック。

③ プロジェクトテンプレートが選択されているのを確認して、[次へ]をクリック。

④ [完了をクリック]

次回から、新しいプロジェクトのテンプレートとして shmit が追加されます。

⑤　[ファイル]-> [テンプレートのエクスポート]をクリック。

⑥　項目テンプレートが選択されているのを確認して、[次へ]をクリック。

⑦　Form->SHMITwindow.xaml がチェックされていることを確認して、[次へ]をクリック。

⑧　[次へ]をクリック。

⑨　テンプレート名を SHMITwindow に変更し、[完了をクリック]

次回から、新しい項目のテンプレートとして SHMITwindow が追加されます。

2.3　アンインストール

①　マイドキュメント¥Visual Studio 2010¥My Exported Templates フォルダの shmit.zip と SHMITwindow.zip を削除する。

②　マイドキュメント¥Visual Studio2010¥Templates¥ProjectTtemplates¥shmit.zip を削除する。

③　マイドキュメント¥Visual Studio2010¥Templates¥ItemTemplates¥SHMITwindow.zip を削除する。

3　SHMIT の構成

SHMIT のプログラム構成は、図１のように「基本部」と「ユーザ定義部」から構成されており、最終的にこれらのコンポーネントがコンパイル->リンクされることで目的の

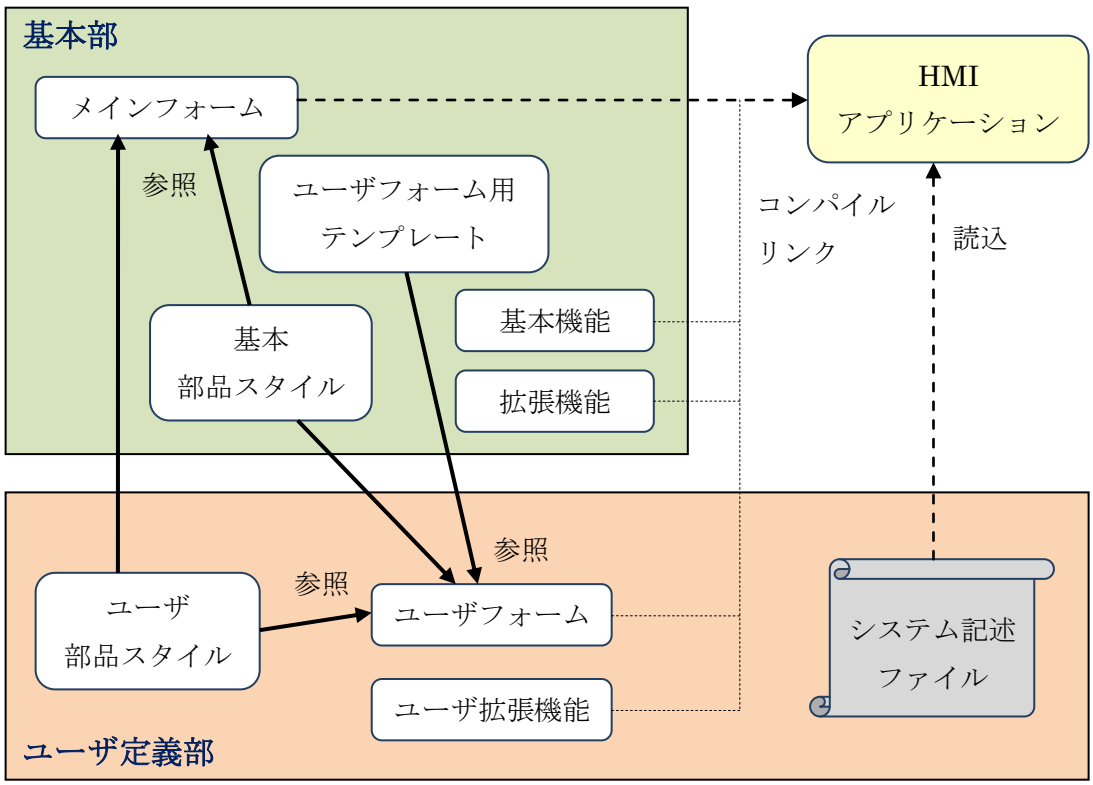

図１　SHMIT のプログラム構成

「HMI アプリケーション」が完成します。

初期状態では「メインフォーム」しか登録されていないため、追加のフォームが必要な場合は「ユーザフォーム用テンプレート」を基に、「ユーザフォーム」を新規作成する必要があります。フォームに配置する部品は、標準の「基本部品スタイル」、およびユーザが作成した「ユーザ部品スタイル」から選択することができます。

SHMIT の基本部には、多くの HMI に必要な「基本機能」があらかじめ組み込まれており、さらにユーザが必要に応じて「拡張機能」を選択・追加することができます。また、基本機能と拡張機能のいずれにも用意されていない機能は、「ユーザ拡張機能」として新たに作成することもできます。SHMIT では、このように必要な機能だけをプログラムに組み込むことで、プログラムサイズの最小化と処理の軽量化を実現しています。

ユーザ部品スタイルの作成方法および拡張機能の追加方法については、応用編マニュアルで説明します。

「システム記述ファイル」は、PLC 担当者と連携を取りながら、プログラムとは独立した形で編集します。出来上がったシステム構成ファイルを HMI アプリケーションが保存されているフォルダと同じフォルダに配置すれば、アプリケーション起動時に自動的に読み込まれ、設定された内容に従って PLC との通信を開始します。基本的なシステム構成ファイル(shmit¥shmit¥System.xml)が SHMIT プロジェクトに含まれていますので、参考にしてください。

4　基本的な開発の流れ

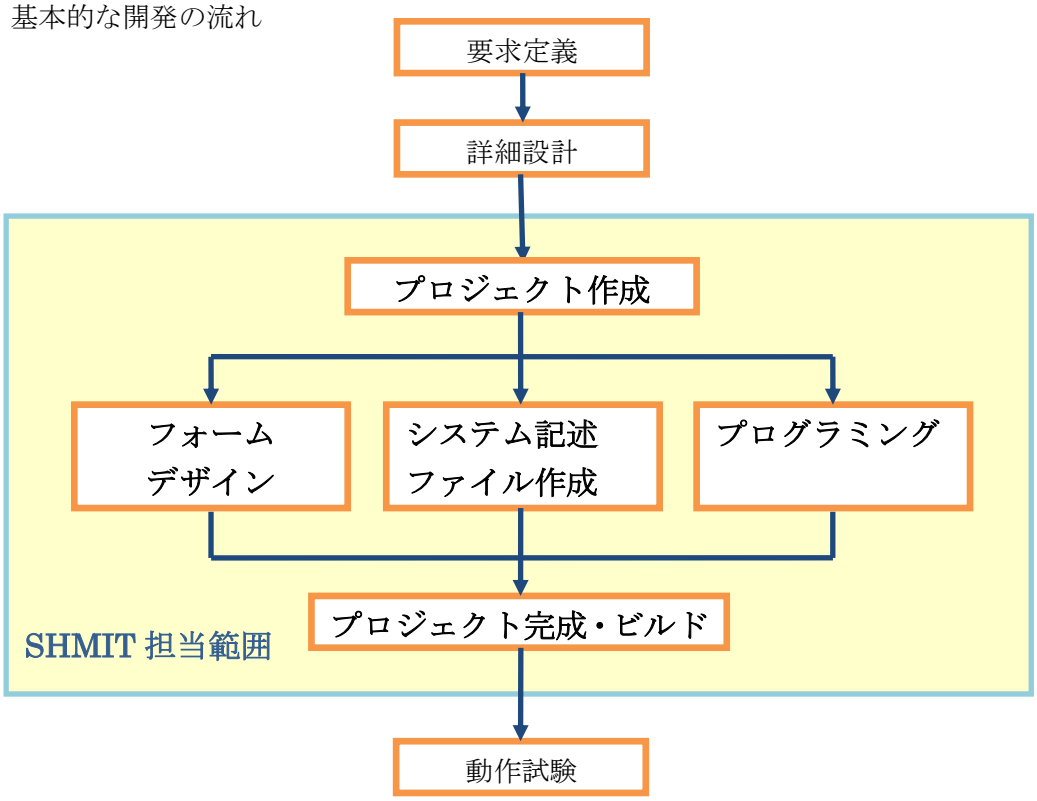

図 2　開発の流れ

この章では、SHMIT を利用した HMI アプリケーション開発の一連の流れについて、具体的な事例（バルブの OPEN/CLOSE やアナログ値の設定・表示など）を交えながら解説を行います。まず、HMI 開発の流れを図 2 に示します。このうち SHMIT が担当するのは、次の 5 つの作業です。

(STEP1)　プロジェクト作成
(STEP2)　フォームデザイン
(STEP3)　システム構成ファイル作成
(STEP4)　プログラミング
(STEP5)　プロジェクト完成・ビルド

これらの作業はすべて一人で行うことが可能ですが、複数人で並行して進めることで、開発期間を大幅に短縮することができます。以下、それぞれの作業の詳細について説明します。

4.1　プロジェクト作成

SHMIT のテンプレートを基に、VisualBasic プロジェクト（WPF アプリケーション）を作成します。

① VisualStudio を起動。
② [ファイル]->[新しいプロジェクト]をクリック。
③ プロジェクトの種類が shmit になっていることを確認。
④ 適当なプロジェクト名を入力し、[OK]をクリック。

4.2　フォームデザイン

以下の説明に従い、図 3 のようなフォームを作成します。

Expression Blend を利用すると、フォームデザインを非常に簡単に行うことができます。Expression Blend の利用を強く推奨します。

① ウィンドウサイズの調整

STEP1 で作成したプロジェクトを Expression Blend または Visual Studio で開き、MainWindow.xaml のウィンドウサイズを width480 x height160 に設定します。キャンバスからはみ出した画面部品はキャンバス内に移動します。

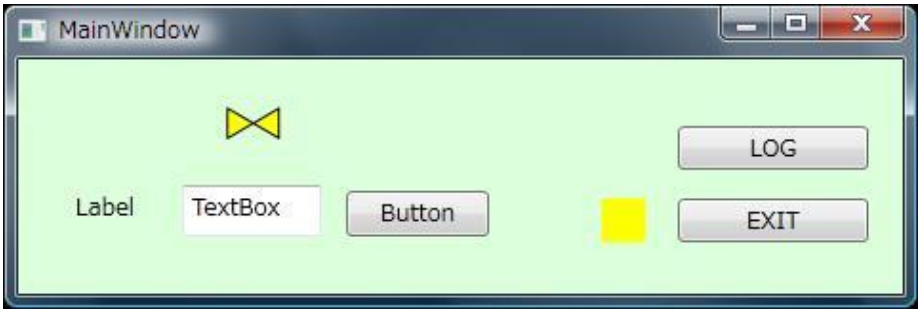

図 3　サンプルフォーム

② 部品の配置

図 3 を参考に、以下の 4 つの部品を画面上に配置して下さい。

内容	種類	Name プロパティ
バルブ	コマンドボタン	valve001
流量設定ボタン	コマンドボタン	flow001_execute
流量現在設定値表示	ラベル	flow001_set
流量入力枠	テキストボックス	flo1001_input

③ スタイルの適用

Expression Blend を使用している場合は、valve001 右クリック-> テンプレートの編集-> リソースの適用を選択し、 valve001 にスタイル "ValveSimple1.0_Yellow_BlackBorder"を適用します。(リソースタブの Valve.xaml からスタイル"ValveSimple1.0_Yellow_BlackBorder"をドラッグすることで、部品の配置とスタイルの適用を同時に行うことも可能です。)

Visual Studio を使用している場合は、MainWindow.xaml を XAML ビューで開き、valve001 の Button タグに以下の Style 属性を追加します。

```
Style="{DynamicResource ValveSimple1.0_Yellow_BlackBorder}"
```

4.3 システム記述ファイル作成

初期状態の SHMIT プロジェクトには最も基本的なシステム記述ファイルが登録されています。以降の説明では、このファイルのルート要素**<SystemRoot>**の中に各種情報を入力します。

システム記述ファイルの各要素の詳細については、「システム記述ファイル項目リファレンス」を参考にしてください。

システム記述ファイルの編集は非常に骨の折れる作業です。現在、編集ツールの開発を検討中です。次期バージョンに期待してください。

① パラメータ情報の入力

システム全体に共通する各種パラメータを入力します。

<DataLogFolder>タグおよび<OperationLogFolder>タグに設定されているフォルダ名が存在しない場合は、実行時にエラーとなりますので、あらかじめ作成しておいて下さい。

```
<ParameterArray>
    <!-- プログラム内部で使用する内部クロック  100msec (変更不要) -->
    <TicksMilliSecond Value="100" />
    <!-- データ収集周期  3 秒  -->
    <AcquisitionCycleMilliSecond Value="3000" />
    <!-- ログデータの保存周期  10 秒  -->
    <ArchiveCycleMilliSecond Value="10000" />

    <LogSetting>
        <!-- 収集データログを保存するフォルダ(複数指定可) -->
        <DataLogFolderArray >
            <DataLogFolder Value="%MyDocuments%¥Log¥" />
        </DataLogFolderArray>
        <!-- 操作・警報履歴を保存するフォルダ(複数指定可) -->
        <OperationLogFolderArray >
            <OperationLogFolder Value="%MyDocuments%¥Log¥" />
        </OperationLogFolderArray>
    </LogSetting>

    <WindowArray>
        <!--プログラム起動時に MainWindow を可視状態でオープン(変更禁止) -->
        <Window Tag="MainWindow" ClassName="MainWindow" _
                    LoadTiming="Default" OpenMode="Visible"/>
        <!--プログラム起動時に LOGMSG を不可視状態でオープン-->
        <Window Tag="LOGMSG" ClassName="LOGMSG" _
                    LoadTiming="Initial" OpenMode="Hide"/>
    </WindowArray>
</ParameterArray>
```

② PLC 情報

PLC への接続情報、読み書きするアドレス情報、問い合わせコマンドについての情報などを記述します。

以下のコードは、オムロンの PLC と FinsTCP プロトコルで通信する場合のコードです。SHMIT は現在、様々なプロトコルへの対応を進めているところです。詳細については、応用編マニュアルをご参照ください。

```
<DeviceArray>
    <!-- デバイスを登録し有効化する  -->
    <Device Name="TEST_PLC" Type="PLC" ActiveFlag="1">
        <!-- 通信プロトコルを設定する  -->
        <Protocol Name="FinsTcp" Type="Tcp">
            <ServerIp>127.0.0.1</ServerIp>
            <ServerPort>9600</ServerPort>
        </Protocol>
```

```
        <!--  読み込む word データのアドレス情報  -->
        <WordGetArray>
            <!-- DM の ch100 -->
            <WordGet Type="DM" Address="100" Name="流量"    _
                                                Tag="FLOW001_RAW" />
        </WordGetArray>
        <!-- word データの問合せコマンドについての情報  -->
        <WordGetQuerryArray>
            <!-- DM の ch100 から 1ch 分の情報を取得する-->
            <WordGetQuerry Type="DM" StartAddress="100" Count="1">
                <!--  取得したデータの 0 番目の項目(ch100 の値)を取得-->
                <GetItem Num="0" />
            </WordGetQuerry>
        </WordGetQuerryArray>
        <!--  書き込む word データのアドレス情報  -->
        <WordSetArray>
            <WordSet Name="流量" Tag="FLOW001_SET" Type="DM"    _
                        Address="100" Format="##0.##" Unit="L/min" />
        </WordSetArray>

        <!--  読み込む bit アドレスの情報を登録する-->
        <SixteenBitsGetArray>
            <!-- CIO の 1000ch -->
            <SixteenBitsGet Type="IR" Address="1000">
                <!-- bit0 -->
                <BitGet BitNum="0" Name="Valve001 _Status" _
                                                Tag="VALVE001_STATUS" />
            </SixteenBitsGet>
        </SixteenBitsGetArray>
        <!-- bit データの問合せコマンドについての情報  -->
        <SixteenBitsGetQuerryArray>
            <!-- ch1000 から 1ch 分の情報を取得する  -->
            <SixteenBitsGetQuerry Type="IR" StartAddress="1000" _
                                                        Count="1">
                <!--  取得したデータの 0 番目の項目(ch1000 の値)を取得-->
                <GetItem Num="0" />
            </SixteenBitsGetQuerry>
        </SixteenBitsGetQuerryArray>
    </Device>
</DeviceArray>
```

※<ServerIp>タグのアドレスは環境に合わせて変更してください。

※<WordGet>,<SixteenBitsGet,<BitGet>に設定する番地は環境に合わせて変更してください。

③　実数データ情報

実数データの計算方法や画面表示についての情報を入力します。

通常、Word データから実数値（工学値）への計算方法や、実数値の画面上への表示方法は項目ごとに異なるため、SHMIT では Delegate という仕組みを利用して、項目ごとに指定された関数へ処理を振り分けるようにしています。なお、これらの関数のプログラミングは、4.4 で行います。

```
<RealDataArray>
    <!-- FLOW001 の情報  -->
    <RealData Name="flow meter" Tag="FLOW001" _
            Unit= "L/min" UnknownValue="-99999" GraphItemNum="1">
        <!-- fncCalcFlow で指定される関数に WordGet の FLOW001_RAW を引数 _
                                    として渡し、FLOW001 を計算する  -->
        <Calculation CalcDelegate="fnc0_4000to0_100">
            <Argument Type="WordGet" Tag="FLOW001_RAW" />
        </Calculation>
        <!--"ShowValueDefault"で指定された関数で、MainWindow 上の  _
            部品 flow001_set にフォーマット"##0.0"で FLOW001 を表示する  -->
        <DisplayItemArray>
            <DisplayItem WindowName="MainWindow" ItemName= _
       "flow001_set" ShowDelegate="ShowValueDefault" Format="##0.0" />
        </DisplayItemArray>
    </RealData>
</RealDataArray>
```

④　論理データ情報

1 つまたは複数の bit データの状態を組み合わせて、論理的な機器の状態を定義します。機器の状態が変化した場合は、ChangeDelegate 要素に記述された関数を呼び出します。

```
<LogicalItemArray>
    <LogicalItem>
        <!--  論理条件に BitGet の VALVE001_STATUS を加える  -->
        <BitItemArray>
            <BitItem id="0" Type="BitGet" Tag="VALVE001_STATUS" />
        </BitItemArray>

        <StateArray>
            <!-- id=0 の入力が 1(ON)の時に VALVE001 open とし、  _
                状態変化時に関数 Handler_VALVE001_OPEN を呼び出す  -->
            <State Name="VALVE001 open" Tag="VALVE001_OPEN" _
                        ChangedDelegate="Handler_VALVE001_OPEN">
                <BitCondition id="0" Value="1" />
            </State>
```

```
            <!-- id=0 の入力が 0(OFF)の時に VALVE001 close とし、 _
              状態変化時に関数 Handler_VALVE001_CLOSE を呼び出す  -->
            <State Name="VALVE001 close"    Tag="VALVE001_CLOSE"    _
                        ChangedDelegate="Handler_VALVE001_CLOSE">
                <BitCondition id="0" Value="0" />
            </State>
            <!-- id=0 の入力が 1 でも 0 でもない時は VALVE001 unknow _
                とし、状態変化時に関数 Handler_VALVE001_UNKNOWN _
                                                    を呼び出す  -->
            <State Name="Valve001 Unknown"    _
                    Tag="VALVE001_UNKNOWN"    _
                    ChangedDelegate="Handler_VALVE001_UNKNOWN" />
        </StateArray>
    </LogicalItem>
</LogicalItemArray>
```

4.4　プログラミング

①　Delegate 関数のプログラミング

- 実数データ計算のための関数定義

　ファイル CalcDelegate.vb に、上記 4.3 節③で<Calculation>タグの CalcDelegate 属性に指定した関数「fnc0_4000to0_100()」の処理を追加します。関数の形式は、Double 型の配列を引数とし、戻り値は Double 型とします。引数には、<Argument>タグの Tag 属性で指定される Word データ配列の現在値が入ります。

```
' (Example)  デジタル値 0-4000 を電圧値 0-100L/min に変換する
Function fnc0_4000to0_100(ByVal ValueArray() As Double) As Double
    Return 0 + (100 - 0) / (4000 - 0) * ValueArray(0)
End Function
```

- 機器状態変化時に呼び出される関数

　ファイル ChangedDelegate.vb に、上記 4.3 節④で<State>タグの ChangedDelegate 属性に指定した 3 つの関数、Handler_VALVE001_OPEN(), Handler_VALVE001_CLOSE()、Handler_VALVE001_UNKNOWN()の処理を追加します。関数の形式は、LogicalItem 型の変数を引数に持つサブルーチンとします。引数には、状態変化した論理データ(LogicalItem)が入ります。

```
' (Example)典型的なバルブの状態変化
Public Sub Handler_VALVE001_OPEN(ByVal LogicalItem As LogicalItem)
    CommonAction(LogicalItem, "VALVE001_OPEN", _
      New Control() {GetWindow(Of MainWindow)("MainWindow").Valve001},  _
      New String() {"ValveSimpleBlackBorderRed"})
End Sub

Public Sub Handler_VALVE001_CLOSE(ByVal LogicalItem As LogicalItem)
    CommonAction(LogicalItem, "VALVE001_CLOSE", _
      New Control() {GetWindow(Of MainWindow)("MainWindow").Valve001}, _
      New String() {"ValveSimpleBlackBorderGreen"})
End Sub

Public Sub Handler_VALVE001_UNKNOWN(ByVal LogicalItem As LogicalItem)
    CommonAction(LogicalItem, "VALVE001_UNKNOWN", _
      New Control() {GetWindow(Of MainWindow)("MainWindow").Valve001}, _
      New String() {"ValveSimpleBlackBorderYellow"})
End Sub
```

② ビットデータ操作、ダイアログ追加

図 4 のようなダイアログを作成し、バルブ操作(ビット書込み)を行う処理を追加します。

図 4　バルブ操作確認用ダイアログ

valve001(コマンドボタン)の Click イベントハンドラを以下のように登録します。
ビット操作の処理は、ダイアログクラスの中でプログラミングされています。

```
' (Example) ビット操作(書込みビットが一つで、ダイアログで確認する場合)
Private Sub Valve001_Click(ByVal sender As System.Object, _
            ByVal e As System.Windows.RoutedEventArgs) Handles Valve001.Click
    ' バルブ操作を行うダイアログを作成する
    ' ボタンのキャプションを"OPEN","CLOSE"とし、ボタン操作によって
    ' BitGet の"VALVE001_SET" を ON/OFF する
    Dim TwoButtons As TwoButtons _
        = New TwoButtons(Kernel, "バルブ操作", "valve001", _
                                    "OPEN", "CLOSE", "VALVE001_SET", False)
    ' ダイアログを開く
    TwoButtons.Show()
End Sub
```

ダイアログを表示せず、"OPEN"と"CLOSE"のコマンドボタンを配置して、それぞれの Click イベントハンドラにビット操作の処理を書いても同じことが実現できます。その場合は、以下のようにプログラミングします。(OPEN と CLOSE のコマンドボタン名をそれぞれ Valve001_open、Valve001_close としています。)

```
' (Example) ビット ON 操作
Private Sub Valve001_open_Click(ByVal sender As System.Object, _
            ByVal e As System.Windows.RoutedEventArgs) _
                Handles Valve001_open.Click
    Kernel.SetBit(New String() {"VALVE001_SET"}, 1, True)
End Sub
' (Example) ビット OFF 操作
Private Sub Valve001_close_Click(ByVal sender As System.Object, _
            ByVal e As System.Windows.RoutedEventArgs) _
                Valve001_open.Click
    Kernel.SetBit(New String() {"VALVE001_SET"}, 0, True)
End Sub
```

③ アナログデータの設定

アナログデータ設定(Word 書込み)を行う処理を追加します。

flow001_execute (コマンドボタン)の Click イベントハンドラを以下のように登録します。

```
' (Example) アナログデータ設定
Private Sub flow001_execute_Click(ByVal sender As System.Object, _
        ByVal e As System.Windows.RoutedEventArgs) _
            Handles flow001_execute.Click
    Dim CheckedValue As Double

    ' テキストボックスの入力値を Double 型に変換する
    ' 同時に範囲チェック(0-100)とフォーマットチェックを行う
    If SharedFunctions.CheckValue(Of Double)( "FLOW_SET", _
            flow001_input.Text, 0, 100, CheckedValue) = 0 Then
        ' PLC に Word データを書込む
        Kernel.SetWord("FLOW001_SET", _
            CUShort(CheckedValue / 100 * 4000), True)
    End If
End Sub
```

4.5　プロジェクト完成・ビルド

ビルド -> shmit のビルドを選択するとプログラムのビルドが開始し、エラーがなければ、実行形式ファイルが shmit¥shmit¥bin¥Release に作成されます。

今後、実行ファイルとシステム構成ファイルを同じフォルダに配置することで、別の PC でもプログラムを実行できます。(実行する PC には.Net Framework4.0 以上がインストールされている必要があります。)

今回のサンプルでは、フォームデザイン、システム構成ファイルおよびプログラミングをすべて同じ担当者が行うという想定の下、説明を行いましたが、先に説明したように shmit を利用した開発では複数の担当者で並行して作業を進めることも可能です。その場合は、ビルドを行う前に、外部で開発したフォームデザインとシステム構成ファイルをプロジェクトに取り込む作業が必要になります。並行作業の詳細については応用編マニュアルで説明します。