**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

目次

[1 SHMITの特徴 3](#_Toc302823500)

[2 インストール・アンインストール 3](#_Toc302823501)

[2.1 動作環境 3](#_Toc302823502)

[2.2 インストール 3](#_Toc302823503)

[2.3 アンインストール 4](#_Toc302823504)

[3 SHMITの構成 4](#_Toc302823505)

[4 基本的な開発の流れ 5](#_Toc302823506)

[4.1 プロジェクト作成 6](#_Toc302823507)

[4.2 フォームデザイン 6](#_Toc302823508)

[4.3 システム構成ファイル作成 7](#_Toc302823509)

[4.4 プログラミング 11](#_Toc302823510)

[4.5 プロジェクト完成・ビルド 14](#_Toc302823511)

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「システム構成ファイル作成」と「フォームデザイン」です。システム構成ファイルとは、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。

1. 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がインストールされていること

* 1. 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。

1. [ファイル]-> [テンプレートのエクスポート]をクリック。
2. 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。
3. 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。
4. [次へ]をクリック。
5. 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。

* 1. 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\ProjectTemplates\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。
4. 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「HMIアプリケーション」が完成します。

図1　SHMITのプログラム構成

**ユーザ定義部**

**基本部**

基本機能

拡張機能

メインフォーム

ユーザフォーム用

テンプレート

基本

部品スタイル

ユーザ

部品スタイル

ユーザフォーム

ユーザ拡張機能

HMI

アプリケーション

システム構成

ファイル

参照

参照

参照

読込

コンパイル

リンク

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。

「システム構成ファイル」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。

1. 基本的な開発の流れ

図2　開発の流れ

**SHMIT担当範囲**

要求定義

**システム構成**

**ファイル作成**

**プログラミング**

**フォーム**

**デザイン**

詳細設計

**プロジェクト完成・ビルド**

動作試験

**プロジェクト作成**

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。

* 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。
   1. 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

1. 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 |

1. 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"ValveSimpleBlack

BorderYellow"を適用します。(リソースタブのValve.xamlからスタイル"Valve

SimpleBlackBorderYellow"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 ValveSimpleBlackBorderYellow}"

* 1. システム構成ファイル作成

初期状態のSHMITプロジェクトには最も基本的なシステム構成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。

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

システム構成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" />

<!-- 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>

　　<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>

1. 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>133.75.51.127</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="RR" 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="RR" 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>**

1. 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。

<!-- 実数データ一覧 -->

**<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>**

1. 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。

<!-- 論理データ一覧 -->

<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>

* 1. プログラミング

1. Delegate関数のプログラミング

Delegate.vbファイルに、4.3で指定した以下の3つの関数の処理を入力します。

* 実数データ計算のための関数(CalculationタグのCalcDelegate属性で指定)

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**

* 実数データ表示のための関数(CalculationタグのShowDelegate属性で指定)

関数の形式は、UIElement型、String型、Double型の3つの引数を持つサブルーチンとします。引数には、表示先の画面コントロール、フォーマット、データ値が入ります。

' (Example) 下記ShowValueDefaultを参照

**Public Sub ShowValueDefault(ByVal Control As UIElement, \_**

**ByVal Format As String, ByVal CalData As Double)**

**Kernel.ChangeText(DirectCast(Control, Label), CalData.ToString(Format))**

**End Sub**

* 機器状態変化時に呼び出される関数(StateタグのChangedDelegate属性で指定)

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**

1. 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**

1. 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**

* 1. 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。