Cardiff TELE*form*® v8.2

BasicScript™ ガイド

― Ver.1.1 ―

この文書に含まれる情報は、公表された時点での株式会社ハンモックの考え方を表しています。株式会社ハンモックは、この文書で明示もしくは暗黙に示された内容を保証するものではありません。この文書は情報の提供のみを目的としています。記載されている内容は予告なく変更することがあります。

重要事項

- このガイドは、TeleForm の基本と Visual Basic 等のプログラムを理解されていることを前提条件として作成しております。
- BasicScript については、このガイドの提供と弊社ホームページの FAQ に掲載されているサンプルのみのご提供とさせて頂いており、個別のお問い合わせは**サポート範囲外**となります。
  開発を行われる方の責任において、BasicScript の組み込みを行って頂きますようお願い致します。

改版履歴

版	日付	内容
初版	2006 年 6 月 23 日	・ 新規作成
Ver1.1	2007 年 9 月 28 日	・ 重要事項の追記

目次

第1章　BasicScript の概要

この章では以下のことを学びます。

- BasicScript の利便性
- BasicScript の機能
- スクリプト タイプ
- エントリー ポイント
- TELEform オブジェクト モデル

1-1. BasicScript とは

1-1-1. BasicScript について

BasicScript は、TELEform でのフォーム処理に様々な拡張処理を加えるための統合スクリプト言語です。BasicScript を使用することにより、例えば以下のような処理を行うことができます。

- 四則演算
- 数値比較
- フィールド検証
- 外部関数呼び出し

BasicScript 言語は Microsoft Visual Basic 6.0 の言語仕様に似ています。そのため、Microsoft Visual Basic 6.0 を使い馴れている方であれば、容易に習得することが可能です。

また、BasicScript は TELEform と密接に連携して動作します。すでに TELEform の基本動作や操作方法を理解していることが前提となりますので、あらかじめご了承ください。

1-1-2. なぜ BasicScript を使用するのか？

BasicScript は、あらゆるフォーム処理のバックグラウンドで動作するツールとして考えることができます。BasicScript を使用すれば、TELEform にカスタマイズした柔軟な機能を付け加えることが可能です。

例えば、ここにオーダー フォームがあるとします。あなたは Reader や Verifier によってそのフォームが評価されている時に、一連の Price フィールドの和を求めたいとします。フォーム スクリプトの適切なエントリー ポイントにコードを書いておけば、これが可能となるのです。

また、別のエントリー ポイントにコードを入力すると、total フィールドが正しく入力されていない場合にそれを修正することができます。Verifier オペレーターがフォーム モードで修正している時に、total フィールドが不正であることを伝えるメッセージ ボックスを表示し、次のフィールドに移る前に total フィールドに正しい値を入力するよう促すのです。

1-2. BasicScript の機能

BasicScript はフォームの評価、値の確認、エクスポート処理などにおいて様々な機能を付加します。また、それぞれの値に対して拡張された柔軟な操作を行うことができます。BasicScript は次の機能を TELEform に与えます。

1-2-1. フォーム評価時

- フォーム データの検査、修正、拡張
- フォーム再検査の必要性の決定
- フィールドやフォームの評価ステータスの変更

1-2-2. データ修正および入力時

- フィールドの不正データ入力時の強制リトライ
- 入力データに基づいた次フィールド動的選択（スキップ-アンド-フィル ロジック）
- フォーム モードに移行する前の不正チェック
- エクスポート前のデータ整形

1-2-3. エクスポート時

- エクスポート前のデータ整合性の確認
- データ エクスポートの完全な制御
- カスタム操作によるフォーム処理

1-2-4. 全般機能

- TELEform のカスタム メニュー作成
- スクリプトでユーザー インターフェースを提供するカスタム ダイアログ ボックス作成
- TELEform Reader での自動定期インターバル処理

1-2-5. スクリプト タイプ

BasicScript は以下の 7 つのタイプで構成されます。

・ フォーム スクリプト

全ての TELEform フォームにつき一つのフォーム スクリプトが存在します。フォーム スクリプトは、フォーム マージ制御、データ不正確認、Verifier でのデータ コントロール、そしてエクスポート データの修正などに使用されます。フォーム スクリプトの利用頻度は非常に高いため、BasicScript を書く上でフォーム スクリプトを作成する機会が最も多いでしょう。

・ グローバル フォーム スクリプト

グローバル フォーム スクリプトは通常のフォーム スクリプトと同じエントリー ポイントを持ちます。しかし、グローバル フォーム スクリプトのエントリー ポイントは全てのフォームに適用されます。

・ エクスポート スクリプト

エクスポート スクリプトは、データのエクスポートに関する処理を記述します。複数のエクスポート スクリプトを作成することが可能です。

・ システム スクリプト

1 つの TELEform システムにつきシステム スクリプトは 1 つだけ存在します。システム スクリプトはバッチ処理の制御を可能にします。また、Designer、Print Manager、Reader、Scan Station、Remote Capture Station、そして Verifier の起動、終了処理を制御することも可能です。

・ カスタム（メニュー） スクリプト

複数のカスタム スクリプトを作成することが可能です。各カスタム スクリプトは Designer、Print Manager、Reader、Scan Station、Remote Capture Station、そして Verifier のスクリプト メニューにコマンドを作成します。ユーザーがこのコマンドを選択するとカスタム スクリプトが実行されます。

・ ピリオディック スクリプト

1 つの TELEform システムにつきピリオディック スクリプトは 1 つだけ存在します。ピリオディック スクリプトは Reader の定期的なインターバルで処理が実行されます。

・ ライブラリ スクリプト

複数のライブラリ スクリプトを作成することが可能です。ライブラリ スクリプトはそれだけで直接実行することはできません。その代わり、関数を定義しておき、それらを上記のスクリプトから呼び出すことが可能です。例えば、複数のフォーム スクリプトで常に使用する関数がある場合、それをライブラリ スクリプトに格納し、各フォーム スクリプトから呼び出すことができます。

1-2-6. スクリプト エントリー ポイント（サブルーチン）

ライブラリ スクリプトを除く各スクリプトは、それぞれ固有のエントリー ポイントを持っています。これらのエントリー ポイントは、主要な TELEform オペレーションを制御可能にする TELEform 特有のサブルーチンと考えることができます。スクリプト内の適切なエントリー ポイントにコードを入力することで様々な TELEform オペレーションをカスタマイズすることができます。

エントリー ポイントとは、TELEform がスクリプトを実行する場所を指し示します。コードが入力されたエントリー ポイントは、いつ TELEform がそのコードを実行するのかを表しています。言い換えるならば、各エントリー ポイントはフォーム処理サイクルでの一意のポイントを示しています。

エントリー ポイントと TELEform 内のデータ フローとの関連性を理解すると、TELEform が呼び出す適切なエントリー ポイントにコードを書くことができます。

例えば、TELEform Reader でフォーム イメージを評価する時には次の一連のイベントが起こります。

1. TELEform がフォーム評価を開始します。
2. TELEform が Form_Evaluate エントリー ポイントを呼び出します。
3. このエントリー ポイントにスクリプトが存在している場合、スクリプトが実行されます。もし存在しなければ空のエントリー ポイントが実行されます。
4. TELEform はフォーム処理を継続します。

エントリー ポイントは事前に定義されたサブルーチンとみなすことができます。エントリー ポイントにスクリプトが存在していようといまいと、TELEform は対応する処理が実行されるタイミングで必ずエントリー ポイントを呼び出します。

1-2-7. BasicScript 言語の概要

BasicScript 言語は Visual Basic 6.0 と同様に振舞います。なぜならば、これらの言語はオブジェクトを扱うからです。BasicScript は TELEform オブジェクト モデルと呼ばれる特別なオブジェクト群を使用します。スクリプトはこのオブジェクト モデルを使用して TELEform とやり取りをします。

TELEform 固有のクラスやプロパティを理解すると、TELEform が格納する全ての情報を利用することができます。

TELEform オブジェクト モデルは以下のようになっています。

- オブジェクト モデルは複数のクラス オブジェクトで構成されます。各クラスは固有の TELEform 情報を格納します。
- 各 TELEform クラスは固有のプロパティ セットで構成されます。各プロパティは TELEform 情報のサブセットを指しています。プロパティを参照することで TELEform 情報にアクセスすることができます。
- 各プロパティはデータ型を持っています。データ型を確認すると、どんな種類の値が格納されているか知ることができます。

- クラス

TELEform オブジェクト モデルは以下の 8 つのオブジェクト クラスで構成されます。各クラスを通じて、異なる TELEform 情報のセットにアクセスできるようになっています。

オブジェクト クラス	説明
Form	フォームのタイトルやフォーム ID などの、処理されているフォームに関する全般的な情報へのアクセスを提供します。
Export**	現在のエクスポート セッションに関する情報へのアクセスを提供します。
Fields	フォーム上の全てのフィールドへのアクセスを提供します。各フィールドは配列の要素として扱われます。
Field	フォーム上のフィールドのデータやその他フィールド属性へのアクセスを提供します。フィールド クラスはもっとも使用頻度の高いオブジェクト クラスです。
Choices*	選択フィールドの選択肢へのアクセスを提供します。各選択肢は配列の要素として扱われます。
TopChoices*	OCR によって認識された文字のうち、精度の高い順に 3 つ並べた文字へのアクセスを提供します。選択された 3 つの文字は配列の 3 要素として格納されます。
Row*	詳細グループのフィールド (列) へのアクセスを提供します。各詳細グループのフィールドの列は配列の要素として扱われます。
Batch***	バッチ情報へのアクセスを提供します。

* … フォーム スクリプトでのみ利用可能

** … エクスポート スクリプトでのみ利用可能

*** … システム スクリプトでのみ利用可能

注意:

TELEform オブジェクト モデルのうち Field クラスを除く全てのクラスはユーザーからは見えません。これらのオブジェクト クラスのインスタンスにアクセスするには直接クラス名を参照してアクセスしてください。

- クラス　プロパティ

各オブジェクト クラスは一意のプロパティ セットを持っています。これらのプロパティはオブジェクトの利用可能な特定の情報を表しています。例えば、全てのフィールド オブジェクトはフィールド ID を表す「Name」プロパティを持っています。

各オブジェクトのプロパティに関しては「BasicScript Guide」を参照してください。

あるプロパティはオブジェクト クラス内の別のプロパティを参照することがあります。例えば、フィールド クラスの「HasMask」プロパティは同じフィールド クラスの「Mask」プロパティのステータスを確認します。

- データ型

各プロパティは次のデータ型のいずれかに分類されます。

型	説明
String	文字列型
Long	32 ビット整数型
Integer	16 ビット整数型
Float	単精度浮動小数点数型
Double	倍精度浮動小数点数型
Variant	バリアント型

- 大文字･小文字の区別

全てのオブジェクトやプロパティは大文字･小文字の区別をしません。コードを書く際は、これらを気にする必要はありません。

City.Text = CITY.TEXT = city.text = CiTy.tExT

しかしながら、大文字･小文字の区別は「アルファベット」と「アルファベットと数値」のフィールドの値を評価する際には非常に重要です。TELEform のフィールドは大文字･小文字を解釈し格納するよう設定できるため、フィールドの値をテストするようなスクリプトでは注意が必要です。

例えば、「City」フィールドが ”San Francisco” という値を保持しているとします。この時に以下のようなスクリプトを書いたとします。

IF City.Text = “SAN FRANCISCO”

この評価は失敗します。なぜならば ”San Francisco” は “SAN FRANCISCO” とマッチしないからです。

大文字･小文字の区別を排除した文字列比較を行う場合は、UCase$や LCase$を使用することができます。

IF UCase$(City.Text) = “SAN FRANCISCO”

この評価は真を返します。

1-3. BasicScript ツアー

BasicScript が TELEform とどのように連携するのか、少しわかったところで、BasicScript のコーディングに関してクイック ツアーを行いましょう。このツアーでは BasicScript をどのようにコーディングし実装するのか、についてみていきます。

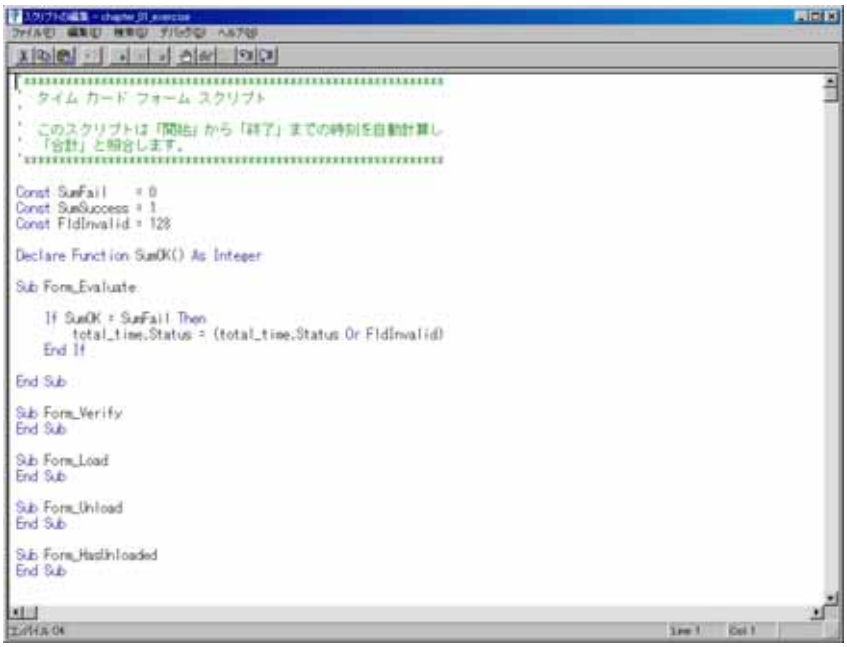
1-3-1. ステップ 1 ： スクリプト タイプを決定する

スクリプトを書く前に、どのタイプのスクリプトを書くかを決めなければいけません。スクリプトに処理させる TELEform のプロセスを明確に見極めておく必要があります。また、エントリー ポイントについても考慮する必要があります。このツアーでは、もっともよく利用されるスクリプト タイプであるフォーム スクリプトを選択します。

1-3-2. ステップ 2 ： TELEform Designer でスクリプトを開く

スクリプトを開くと「スクリプトの編集」ウィンドウが表示されます。「スクリプトの編集」ウィンドウはコードを書いたり、編集したり、コンパイルをするのに使用するアプリケーションです。このツアーでは「Chapter 1 Exercise」のフォーム スクリプトを開きます。

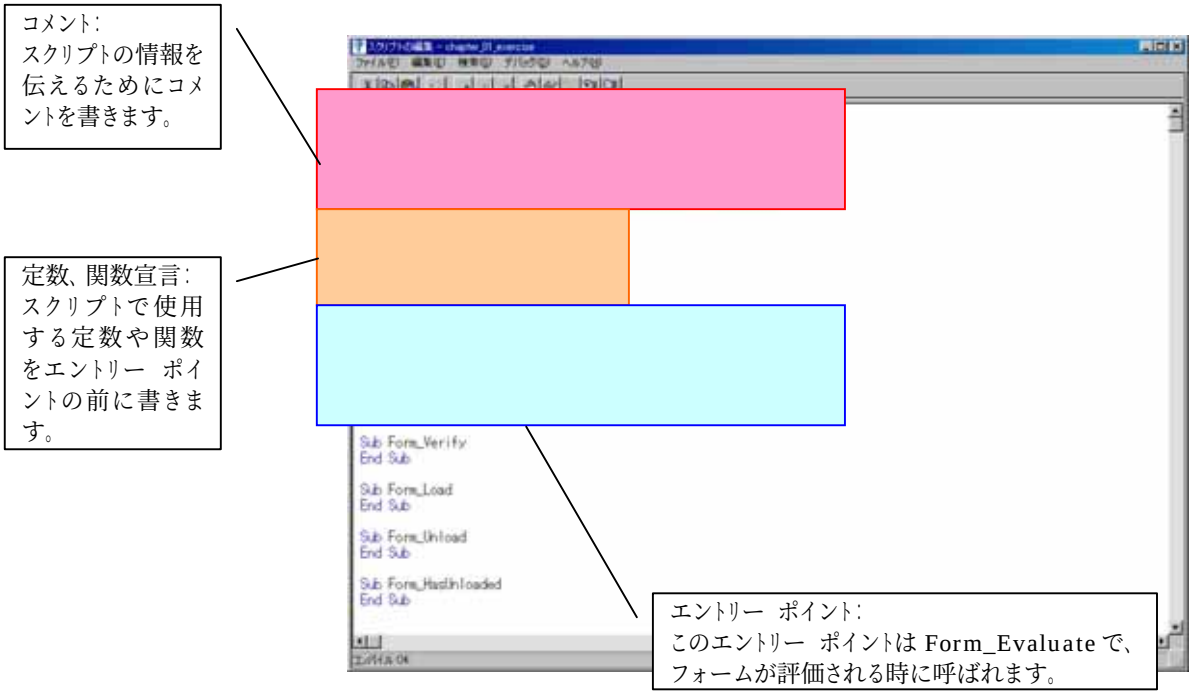
1. TELEform Designer で「Chapter 1 Exercise」を開いてください。
   （「Chapter 1 Exercise」は［02_フォーム］の「tf15597.tfw」です。）
2. 「フォーム」メニューから「スクリプト」を選択します。「スクリプトの編集」ウィンドウが表示されます。



1-3-3. ステップ 3 ： スクリプトを書く

「スクリプトの編集」ウィンドウは、テキスト エディタで見られるような多くの基本機能とスクリプトのステートメントを記述するのに使用される BasicScript 特有の機能を持っています。

スクリプトの構成は以下のようになっています。



何もない状態からスクリプトを作成する場合は、次の慣例に従ってコーディングすることをお奨めします。

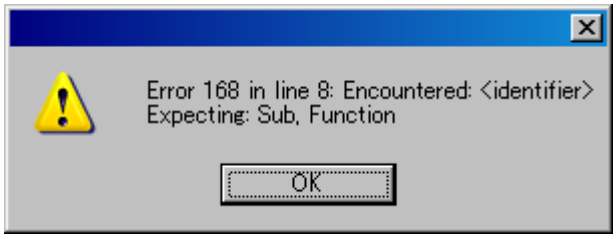
- 他の人にスクリプトの内容を伝えるため、そして自分のためのメモとしてスクリプトの先頭にコメントを書きます。
- 定数や Public 変数、関数を宣言する際は、スクリプトの先頭（エントリー ポイントの前）で行います。
- 論理的に整理できるセクションをグループとして区別するために空白行を使用します。例えばこの開発者は、変数を 1 つのグループに、関数を別のグループにしています。
- 各グループに最低 1 行のコメントを書きます。これにより、他の人がスクリプトを見た時に、コードの内容が明確になります。また自分の理解を助ける手段にもなります。
- フィールド参照に右クリックを使用します。

1-3-4. ステップ 4 ： スクリプトをコンパイルする

スクリプトを作成し終えると、BasicScript ステートメントの文法をチェックするため、コンパイルする必要があります。文法に問題なければステップ 5 に進んでください。文法に問題があった場合は、エラー メッセージが表示されます。このエラー メッセージは問題のある行まで移動し、その問題が何なのか知らせてくれます。

重要:
コンパイルが OK だとしても、そのスクリプトの実行が成功するとは限りません。コンパイラーは全ての起こりえるエラーを検出することはできません。

1. 「スクリプトの編集」ウィンドウの「ファイル」メニューから「コンパイル」を選択します。文法に問題がなければ「コンパイル OK」となります。
2. もしスクリプトにエラーがあれば下の図のようなエラー メッセージが表示されます。



3. このエラー メッセージはエラーの行を示しており、そのエラー行が選択されます。またエラーの理由を表示します。全てのエラーを修正したら、再度スクリプトをコンパイルしてください。
4. 「ファイル」メニューから「保存」を選択します。その後「スクリプトの編集」ウィンドウを終了します。
5. TELEform Designer でフォームを保存します。

1-3-5. ステップ 5 ： スクリプトを実行する

スクリプトのコンパイルが OK ならば、それが期待どおりに動作するか実行します。このことはスクリプトを動作させるための環境を用意するということも意味しています。例えばこの例では、サンプル イメージを評価し TELEform Verifier で修正する作業が必要です。(サンプル イメージは total_time フィールドに不正な値が入っています)

1. TELEform Reader を起動します。
2. 「ファイル」メニューから「イメージの評価」を選択します。「ファイルを開く」ダイアログ ボックスが表示されます。
3. 「Image01.tif」を選択し「開く」をクリックします。
   (「Image01.tif」は [03_演習用データ] の [Chapter 1 Exercise] にあります)

4. TELEform Readerはサンプル イメージを評価します。この時、Form_Evaluateエントリー ポイントが呼ばれ、実行されます。
5. TELEform Verifierを起動します。
6. 「フォーム」リストから「Chapter 1 Exercise」を選択します。
7. 「保存されているイメージ」から「Image01.tif」を選択し「修正」ボタンをクリックします。
8. 不正な文字を修正し、フィニッシュ フラグまでたどり着いたら何かキーを押します。
9. フォーム モードが開始されるので、total_time フィールドまで TAB で移動します。
10. 次のフィールドに移るために TAB を押します。この時、下の図のようなエラー メッセージが表示されます。

11. 「OK」をクリックし total_time フィールドに戻ります。
12. total_timeフィールドに正しい時間（08:30）を入力し、TABを押します。今度は、Verifierはこの値を受け入れてくれます。

重要:
フォームに対して何らかの変更を加えた場合は、常にスクリプトをコンパイルしてください。

1-4. 演習

この演習では、「1-3. BasicScript ツアー」を実践します。
以下の手順に従って演習を進めてください。

1. 目の前に次の環境があることを確認してください。
   - TELEform v8.2 がインストールされている Windows マシン
     （以下のアプリケーションがインストールされていることを確認してください）
     - TELEform Designer
     - TELEform Reader
     - TELEform Verifier

2. 「1-3. BasicScript ツアー」のステップ 1～5 を行ってください。

第2章　基本構文

この章では以下のことを学びます。

・ 構文の特徴
・ 構文の各要素
・ よく使用する関数とサブルーチン

注意:
この章で紹介される言語仕様は完全ではありません。詳細は BasicScript のオンライン ヘルプを参照してください。また、BasicScript は Microsoft Visual Basic 6.0 の言語仕様とほぼ同等です。Microsoft Visual Basic 6.0 のヘルプも参考になりますので必要に応じ参照してください。

2-1. 構文の特徴

2-1-1. 全体の流れ

全体的に構文は以下のような流れとなっています。

1. 変数を宣言します。
2. 関数、サブルーチンを宣言します。関数に与える引数の名前と型、および関数の返す値の型の定義を記述します。
3. 関数およびサブルーチンを記述します。

BasicScript では大文字小文字は区別しません。フィールドの値は大文字、小文字を区別します。

また、命令と命令の間にある空白は無視されます。空白をあけないでプログラムを記述することも可能ですが、見やすいように適切に空白を設けます。ブロック（Sub－End Sub、Function－End Function、For－Next、Do－Loop、If－Then－End If）の中は前の段より 4 文字または 2 文字空白を設けて字を下げます。（この字下げのことをインデントと言います）

例:

```
Sub Test
    Dim i As Integer

        For i = 0 To 10
            If i = 5 Then
                MsgBox “i is 5“
            End If
        Next
End Sub
```

2-1-2. コメント

コメントは実行されませんので覚え書きとして使用できます。後でプログラムを読むときにわかるように記述をします。コメントには「'」を使用します。

例:

```
'Comment
Sub Test
```

2-1-3. 値

整数は「.」（ピリオド）を含まない数値です。浮動小数点数は「.」を含む数値です。100 は整数ですが、100.0 は浮動小数点数です。

また、文字列は「"」（ダブルクォート）で囲みます。

2-1-4. 名前

名前とは変数、および関数・サブルーチン名として使用できる文字列です。名前の先頭はアルファベットでなければいけません。それ以降では英数字が使用できます。また「_」（アンダー バー）も使用できます。名前の大文字、小文字は区別されません。つまり変数として Aa を宣言した場合、AA、aA、aa も同じように使用できます。

名前として予約語を使用することはできません。予約語とは、すでに定義済みの名前、BasicScript キーワードのことです。例えば long という名前は使用できません。

以下が予約語の一覧です。

Access	Alias	And	Any	Append	As
Base	Begin	Binary	Boolean	ByRef	ByVal
Call	CancelButton	Case	CDecl	CheckBox	Chr
ChrB	ChrW	Close	ComboBox	Compare	Const
CStrings	Currency	Date	Declare	Default	DefBool
DefCur	DefDate	DefDbl	DefInt	DefLng	DefObj
DefSng	DefStr	DefVar	Dialog	Dim	Do
Double	DropListBox	Else	ElseIf	End	Eqv
Error	Exit	Explicit	For	Function	Get
Global	GoSub	Goto	GroupBox	HelpButton	If
Imp	Inline	Input	InputB	Integer	IsLen
Let	Lib	Like	Line	Listbox	Lock
Long	Loop	LSet	Mid	MidB	Mod
Name	New	Next	Not	Nothing	Object
Off	OKButton	On	Open	Option	Optional
OptionButton	OptionGroup	Or	Output	ParamArray	Pascal
Picture	PictureButton	Preserve	Print	Private	Public
PushButton	Put	Random	Read	ReDim	REM
Resume	Return	RSet	Seek	Select	Set
Shared	Single	Spc	Static	StdCall	Step
Stop	String	Sub	System	Tab	Text
TextBox	Then	Time	To	Type	Unlock
Unit	Variant	WEnd	While	Width	Write
Xor					

2-2. 構文の要素

2-2-1. 変数

変数には値(文字)を入れることができます。変数は使用する前に型を宣言してください。宣言をしなくても変数を使用することはできますが、多くの場合解決の困難なバグの原因となります。変数の宣言は以下のように行います。宣言なしで変数が使用された場合は、使用された時点で変数がバリアント型で作成されます。

Dim（変数名）As（変数の型）

例:

```
Dim a As Integer, b As Long
```

変数の型とはその変数に入れることのできる値の集合のことです。以下の代表的な型があります。

Integer	整数型 (-32768 ～ 32767)
Long	長整数型 (-2^32 ～ 2^32-1)
Double	浮動小数点型
String	文字列型

変数が初期化されたき、変数に入っている値は、文字列変数の場合は空文字(“”)、数値型の場合は 0 です。しかし、これを期待してプログラムを作成することは奨励できません。変数は参照する前に適切な値で初期化してください。

関数･サブルーチンの中で宣言された変数はその中でしか有効でありません。関数 foo で変数 i を宣言して、これに 100 を入れたとします。別の関数で i を見たとき、その値は 100 である保証はありません(見ることができません)。このような変数を局所変数(ローカル変数)と言います。関数･サブルーチンの外で宣言された変数はすべての範囲で使用することができます。これを広域変数(グローバル変数)と言います。(このように変数の有効な範囲のことをスコープと言います)

変数への値の代入は = 演算子を使用します。

例:

```
Dim A As Integer, S As String
A = 1              'A に 1 を代入します。
S = “123“          'S に“123“を代入します。
```

数値型(整数型、浮動小数点型...)同士の代入は自動変換されますが、数値型と文字列型の代入は変換関数が必要です。

例:

```
Dim A As Double, B As Integer, S As String
A = 2.3
B = A           'B には 2 が入ります。(整数変換されます)
S = CStr(A)     'S には"2.3"が入ります。
```

2-2-2. 定数

定数の宣言は以下のように行います。

```
Const (定義名) = (定数値)
```

例:

```
Const konnitiha = "Hello World!!"
Const Pi = 3.14159265358979
```

2-2-3. 構造体型

構造体型とは、複数の変数が集まったものです。例えば構造体 People を定義します。

例:

```
Type people
    sName As String
    iAge As Integer
    sAddress As String
End Type
```

この構造体は sName、sAddress という文字列変数、iAge という整数変数をもつ構造体です。この構造体の使用例を以下に示します。

例:

```
Dim Tarou As people, Jirou As people

Tarou.Name = "Tarou"
Tarou.Age = 10
Jirou.Name = "Jirou"
```

2-2-4. 演算子

BasicScript では以下の演算子が使用できます。

変数への代入	=
四則演算	+ - * /
べき乗演算	^
整数除算	¥ (例: 5 ¥ 2 の結果は 2 です)
剰余算	Mod (例: 5 Mod 2 の結果は 1 です)
文字列の結合	+ & (例: "AB" + "C" の結果は "ABC" です)
比較演算	= <> > < <= >=
論理演算･ビット演算	And Or Not Xor

演算子の優先順位は一般の数字と同じですが、式が複雑な場合には()を用いて明示的に示すようにしましょう。

例:

```
If ((A = 1) And (B = 2)) Or ((A = 2) And (B = 1)) Then
```

2-2-5. 関数とサブルーチン

関数は呼び出されると仕事(処理)を行い、何かの値を返す処理の単位です。サブルーチンは関数と違い値を返しません。

関数は以下のように定義します。(ここでは foo という関数を定義します)

例:

```
Function foo(a As Integer, b As Integer) As Integer '関数の定義を開始
    If a > b Then       'a が b より大きいなら
        foo = a         'この関数は a を返します
    Else                'そうでないなら…
        foo = b         'この関数の戻り値を b とします
    End If              '条件文の終わり
End Function            '関数の定義の終わり
```

この関数は a、b の 2 つの整数を受け取り大きい方の整数を返します。関数の返す値を設定するには、その関数の名前に値を代入することで行います。

サブルーチンは以下のように定義します。

例:

```
Sub bar(A As String)
    (処理内容)
End Sub
```

関数とサブルーチンは使用する前に、その型を宣言しておくことを奨励します。関数型の宣言は Declare 文で行い、通常スクリプトの先頭で行います。

例:

```
Declare Function foo(a As Integer, b As integer) As Integer
Declare Sub bar(A As String)
```

関数やサブルーチンの途中で、それらを終了したい場合があります。その時は Exit Function または Exit Sub を使用します。次の例は、再起呼び出しという手法を用いて階乗(!)を計算します。5!は 5*4*3*2*1 = 120 です。

例:

```
Function kaijou(n As Long) As Long
    If n = 1 Then
        Kaijyou = 1
        Exit Function
    End If
    Kaijyou = n * Kaijyou(n - 1)
End Function
```

これは次のようにも記述できます。

例:

```
Function kaijou(n As Long) As Long
    If n = 1 Then
        Kaijyou = 1
    Else
        Kaijyou = n * Kaijyou(n - 1)
    End If
End Function
```

関数やサブルーチンを呼び出すには、次のようにします。

例:

```
i = Kaijyou(5)
bar “Hello!!“
```

2-2-6. 制御構造

If 文は条件によってある処理を行わせたい場合に使用します。今 A が 1 で B が 2 だとします。次の条件を実行すると A に 2 が入ります。

例:

```
If A < B Then B = A Else A = 0
```

もし A が-1 だと A には 0 が入ります。

行わせたい処理が複数行ある場合は、

例:

```
If A < B Then
    B = A
    A = 0
Else
    A = B
    B = 0
End If
```

のように記述します。

For 文は繰り返し文です。次の例では 1 + 2 +… + 9 + 10 を計算します。

例:

```
sum = 0
For i = 1 To 10
    sum = sum + i
Next
```

For－Next の間の文は必ず 1 度は実行されます。次の例では sum に 1 が入ります。

例:

```
sum = 0
For i = 1 To 0
    sum = sum + i
Next
```

For 文の他に繰り返しを行う命令として、Do－Loop 文があります。

例:

```
sum = 0
i = 1
Do
    sum = sum + i
    i = i + 1
Loop While i < 11
```

例:

```
sum = 0
i = 1
Do
    sum = sum + i
    i = i + 1
Loop until i > 10
```

Do－Loop 内の文を実行する前に条件判定を行うか、実行した後に条件判定を行うかの区別は重要です。次の文では内部の命令は実行されるので x に-1 が入ります。

例:

```
i = 0
x = 0
Do
    x = x - 1
    i = i + 1
Loop While i < 0
```

しかし、次の例では x は 0 のままです。

例:

```
i = 0
x = 0
Do While i < 0
    x = x - 1
    i = i + 1
Loop
```

2-2-7. よく使用する関数とサブルーチンの説明
(□で囲まれた引数は省略可能な引数です)

〈数値操作〉

関数　ABS(数値)

数値の絶対値を返します。つまり ABS(-1)は 1、ABS(1)も 1 です。

関数　INT(数値)

数値の小数点を取り除いた、整数部分を返します。INT(1.5)は 1 です。

〈文字列の操作〉

関数　CSTR(数値)

数値を文字列にして返します。数値には整数、小数ともに使用できます。

例:

```
CSTR(100)　→ "100"
CSTR(-100)　→ "-100"
```

この関数は、数値を表示したい場合 MsgBox 文とともによく使用します。

例:

```
MsgBox "Value of X is" + CStr(X)
```

変数 X の値を表示するメッセージボックスを表示します。

関数　　VAL(文字列)

文字列で表現された数値を数値型に変換します。変数 X が整数型で文字列 “123“ で表現された数値を X にしたい場合、A = “123“ ではエラーとなりますので、A = VAL(123) とします。

関数　　LEFT$(文字列、数値)　RIGHT$ (文字列、数値)

それぞれ、文字列の左から数値で指定された文字数分、右から指定された文字数分を取り出します。

例:

LEFT$(“1234“, 2)　　→ “12“

RIGHT$(“1234“, 2)　→ “34“

関数　　STRING$(数値、文字列)

文字列を指定された数値回繰り返した文字列を返します。

例:

STRING$(3, “A“)　→ “AAA“

関数　　MID$(文字列、先頭位置、[文字数])

文字列の先頭から指定された文字数を取り出します。文字数が省略されたときは先頭から後ろの文字列を返します。

例:

MID$(“1234“, 2, 2)　→ “23“

MID$(“1234“, 2)　　→ “234“

サブルーチン　　MID$ (文字列変数、先頭位置、文字数) → 置換文字列

文字列変数の先頭から文字数で指定された文字数を置換文字列で置き換えます。

例:

A$ = “1234“

MID$(A$, 2, 2) = “56“　→ A$には “1564“ が入ります。

関数　　WORD$(文字列、数値)

文字列から指定された位置の語(スペースで区切られた文字)を返します。これは文字列の前後にある余分な空白を取り除くことに使用できます。

例:

WORD$(“This is a pen.“, 2)　→ “is“

WORD$(“ABC“, 1)　→ “ABC“

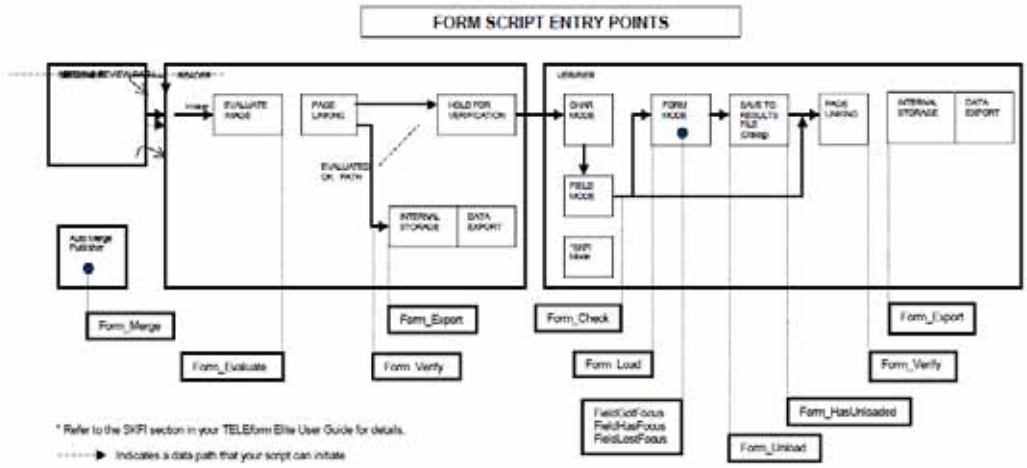
第3章　フォーム スクリプト

この章では以下のことを学びます。

- エントリー ポイント
- Form クラス
- Fields コレクション
- Field クラス

3-1. エントリー ポイント

次の図は、各フォーム スクリプトのエントリー ポイントが TELEform のデータ フロー上、いつ呼ばれるかを示しています。



このようにエントリー ポイントはフォーム処理の中で順に処理されます。各エントリー ポイントの詳細は次のページから紹介しています。

エントリー ポイント	説明
Sub Form_Merge : End Sub	このエントリー ポイントは TELEform Auto Merge Publisher でのフォーム マージ処理の一番初めに呼ばれます。フォーム上のマージされるデータはスクリプト内のフィールドと一致します。このエントリー ポイントでユーザーはマージ前にデータを修正できます。 注意: マージはフォーム スクリプトで中断できません。Form_Merge エントリー ポイントはマージ処理で使用されるデータを修正するためだけに使用されます。
Sub Form_Evaluate : End Sub	このエントリー ポイントは TELEform Reader に評価された後、すぐに呼ばれます。 マルチページ フォームの場合: ・ マルチページ フォームのリンクされたグループ毎に実行されます。 ・ ページ リンク フィールドがいくつかのページ結合に失敗するとフォームは複数回処理されます。 このエントリー ポイントは、フォーム データの評価や再評価用にフィールドにマークをしたりするのに頻繁に使用されます。 このエントリー ポイントは TELEform Internet Server で受け取った全ての HTML フォームで呼ばれます。
Sub Form_Check : End Sub	TELEform Verifier でイメージが修正されている時、文字モードとフィールド モードが終了した後、フォーム モードに入る前にこのエントリー ポイントが呼ばれます。このエントリー ポイントを使用してフォーム モードへ入る前に、修正データを保存したり評価を実行したりできます。 TELEform Verifier で同時に複数のイメージを修正している場合は次の事柄が適用されます: ・ フォームの全フィールドが文字モードやフィールド モードで評価される場合、Sub Form_Check はそのフォームが文字モードやフィールド モードで評価され続ける間、バックグラウンド プロセスとして初期化されます。 ・ フォーム モードがどのフォームでも必要な場合、Sub Form_Check はフォーム モード発生前に初期化されます。 ・ フォームが SKFI 領域を含んでいた場合、Form_Check は文字モード、フィールド モード、SKFI モードが終了した後に呼ばれます。 注意: Form_Check での評価が 1 秒または 2 秒以上かかる場合、Verifier オペレーターは、Verifier がフォーム モードへデータを流す準備をしている間、「Waiting for validation」というメッセージを受け取るかもしれません。
Sub Form_Load : End Sub	このエントリー ポイントは TELEform Verifier でフォーム モードに入るとすぐに呼ばれます。

エントリー ポイント	説明
Sub FieldGotFocus : End Sub	このエントリー ポイントは TELEform Verifier のフォーム モードでフィールドがフォーカスを受け取る直前に呼ばれます。
Sub FieldHasFocus : End Sub	このエントリー ポイントは TELEform Verifier のフォーム モードでフィールドがフォーカスを受け取った時 (フィールドが反転した時) に呼ばれます。Sub FieldHasFocus はそのフィールドに関してオペレーターから情報を取得するのに使用されます。
Sub FieldLostFocus : End Sub	このエントリー ポイントは TELEform Verifier のフォーム モードでフィールドがフォーカスを失った時 (フィールドからタブ アウトした時) に呼ばれます。Sub FieldLostFocus は修正されたデータをすぐにチェックし Verifier オペレーターによる不正なデータ投入を防ぐために使用されます。
Sub Form_Unload : End Sub	このエントリー ポイントは TELEform Verifier でフォームが閉じられる前に呼ばれます。正確には変更を保存するダイアログが表示される直前か、ユーザーが手動でフォームを閉じた時に呼ばれます。これはデータのダブルチェックや再度修正をさせるためにフォームのステータスを強制的に変更するのに使用されます。 このエントリー ポイントは SKFI ストリーミング モードの SKFI 領域から出た直後にも呼ばれます。
Sub Form_HasUnloaded : End Sub	このエントリー ポイントは Sub Form_Unload の直後に呼び出されます。変更を保存するダイアログは Sub Form_HasUnloaded が呼び出される前に表示されます。これは Form_Load で開いたファイルを閉じるのに使用されます。
Sub Form_Verify : End Sub	このエントリー ポイントは Verifier オペレーターが全てのフォーム イメージを修正した後で呼び出されます。このエントリー ポイントを使用して完全なクロス バリデーションを行うことができます。評価が正しくなければ再度評価をさせることができます。 このエントリー ポイントは Verifier による評価プロセスを通っていない場合でも呼び出されます。(TELEform Reader で評価 OK となったフォームのこと) TELEform Reader や Verifier で Form_Verify に入る前に全てのページ リンク処理は終了しています。 注意: Form_Verify は、メイン (ユーザー) スレッドで実行される他のエントリー ポイントとは異なる (バックグラウンド) スレッドで実行されます。Control Center はメイン スレッドの動作をログに書き込み、バックグラウンド スレッドの動作は書き込みません。そのため Form_Verify での動作はログに書き込まれません。
Sub Form_Export : End Sub	このエントリー ポイントはデータ ファイルへデータがエクスポートされる直前に呼ばれます。Sub Form_Export はエクスポート前にデータを変更することができます。そのため、エクスポート スクリプトで定義されたデータ フォーマットに変更したりすることが可能です。

- フィールド固有のエントリー ポイント

フォーム評価時に、あるフィールドのためだけにスクリプトが動作すると便利なことがあります。これを利用すると、修正されたデータをすぐにチェックしたり、Verifier オペレーターからの不正データ入力を防いだりすることができます。この機能を有効にするために、BasicScript は各フィールド用に 3 つのエントリー ポイントを用意しています。

TELEform Verifier のフォーム モード時にフィールドがフォーカスを受け取ったり失ったりすると、次のエントリー ポイントが実行されます。

エントリー ポイント	説明
Sub *FieldName*_GotFocus ： End Sub	このエントリー ポイントは TELEform Verifier のフォーム モードで該当のフィールドがフォーカスを受け取る直前（フィールドへタブ インした時）に呼ばれます。
Sub *FieldName*_HasFocus ： End Sub	このエントリー ポイントは TELEform Verifier のフォーム モードでフィールドがフォーカスを受け取った時（フィールドが反転した時）に呼ばれます。
Sub *FieldName*_LostFocus ： End Sub	このエントリー ポイントは TELEform Verifier のフォーム モードでフィールドがフォーカスを失った時（フィールドからタブ アウトした時）に呼ばれます。

スクリプトが通常のエントリー ポイント（FieldGotFocus や FieldLostFocus など）とフィールド固有のエントリー ポイントを定義していた場合、通常のエントリー ポイントがフィールド固有のエントリー ポイントより先に呼ばれます。

今 total というフィールドがあると仮定した場合、フィールド固有エントリー ポイントは以下のように定義されます。

- total_GotFocus
- total_HasFocus
- total_LostFocus

3-2. フォーム スクリプト クラスとプロパティ

TELEform には 8 つのオブジェクト クラスが存在します。各クラスは固有のプロパティ セットを持っています。これらのプロパティを使用するとあらゆる TELEform 情報にアクセスすることが可能です。

3-2-1. Form クラス

Form クラスのプロパティは現在フォーム スクリプトで処理されているフォームの情報を保持しています。Form クラスの情報を参照するには以下のようにします。

Form.*FormPropertyName*

FormPropertyName は Form クラスの適切なプロパティを指しています。例えば以下のようになります。

```
id = Form.FormId          'フォーム ID を id に代入
name$ = Form.Title        'フォームのタイトルを name$に代入
```

Form クラスのプロパティは以下のように定義されています。

プロパティ	型	アクセス	説明
Title	String	Read Only	フォームの名前。
FormID	Long	Read Only	フォームにアサインされた ID。
Mode	Integer	Read Only	現在の処理モード。
Image	String	Read Only	フォーム イメージのパス。
Status	Integer	Read Only	フォームの状態を格納。フォーム評価時は再評価が必要か、評価 OK のどちらかが指し示されています。Verifier 時は、修正されたデータをどのようにコントロールするか決定することができます。「Form.Status プロパティ値」で更に詳細を説明します。
CurField	String	Read Only	GotFocus、HasFocus、LostFocus エントリー ポイントのみで定義されます。このプロパティは現在のフィールド名を保持します。
CurGroup	String	Read Only	GotFocus、HasFocus、LostFocus エントリー ポイントのみで定義されます。このプロパティは現在のフィールドが属する詳細グループ名を保持します。現在のフィールドが詳細グループに属さない場合はこのプロパティは空です。

- Form.Mode プロパティ値

値	値	処理ステータス
Evaluation	2	Reader：フォーム評価。
FormFill	4	Verifier：新規フォーム入力。 Reader：HTML またはフォーム処理。
Suspense	4	Verifier：フォーム モードでの修正、またはフォーム モードで SKFI データ エントリーを処理。
Exporting	16	Reader、Verifier、Designer：データのエクスポート。
FinalValidate	128	Reader、Verifier：Form_Verify の評価または修正終了。
FormCheck	512	Verifier：文字モード、フィールド モード、SKFI ストリーミング モードからフォーム モードへの遷移。
FormMerge	1024	Auto Merge Publisher：Form_Merge のフォームのマージ。この値はマージ前のフォームの全フィールドへの読み書きアクセスを許可します。Remote_Fax や Remote_Phn のようなフォーム上に印刷されない仮想マージ フィールドはこれに含まれません。
SKFI	2048	Verifier：SKFI ストリーミング モードでの SKFI データ入力。
DataReview	8192	Verifier：データ再評価モードでのクオリティ コントロールの実行。

- Form.Status プロパティ値

次の表は TELEform Reader と Verifier がエントリー ポイントでセットできる Status プロパティ値を示しています。他のエントリー ポイントでは Form.Status を設定することはできません。

値	Form_Evaluate From_Verify	Form_Unload	Form_Verify	Index 値
Accept	評価 OK	修正をファイルに保存	修正をファイルに保存	0
Suspend	要再検査	フォーム モードに戻る	後の評価のために変更保存	1
Cancel	設定できない	後の評価のために変更を保存せずフォーム モードを終了	設定できない	2
SaveAndExit	設定できない	現在の状態でフォームを保存しフォーム モードを終了	設定できない	3

3-2-2. Fields コレクション (配列)

Fields コレクションはフォーム スクリプトとエクスポート スクリプトで使用できます。Fields は、処理中のフォームの全フィールドへのアクセスを提供するコレクション オブジェクトです。他のコレクション オブジェクトと同様に、Fields コレクションにはフォームのフィールド数を示す Count プロパティがあります。これは詳細グループの中のフィールドは含めません。トップ レベル フィールドの名前つきフィールド数を示します。SKFI 領域、詳細グループ、仮想フィールドを含むアドレス グループのメンバー フィールドは、このコレクションに含まれています。

注意:
詳細グループは各列に Fields コレクションがあります。

- Fields コレクション情報の参照

Fields コレクションはフォームのフィールドのセット、または詳細グループの列のフィールドを表しています。これらのコレクションは各アイテムへアクセスするのに配列構造を利用します。0 は配列の最初の要素を示します。各 Fields の Field クラスのプロパティを参照するには以下のようにします。

Fields(i).*PropertyName* または Fields(*FieldName*).*PropertyName*

i	0 から Fields.Count - 1 の間の整数
FieldName	フォームのフィールドのフィールド ID
PropertyName	フィールド クラスの適切なプロパティ

例:

```
Dim i As integer
For i = 0 To Fields.Count - 1
    DispMsg “The “ & i & “th field is“ & Fields(i).Name
Next i

If Not Fields(“Name“) Is Nothing Then
    DispMsg “The Name field contains the value “ & Fields(“Name“).Text
End If
```

次のプロパティが Fields コレクションで定義されています。

プロパティ	型	アクセス	説明
Count	Integer	Read Only	フォームのトップ レベルのフィールド数。エクスポート時はエクスポートされたフィールド数となります。
( )	Field	Read Only	フォームのトップ レベルのフィールド コレクション。括弧の中の値は次のようになります: ・ 0 から Fields.Count - 1 の間の整数 ・ フィールド ID 注意: このプロパティにバリアントは使用できません。 文字列が Fields コレクションの中の適切なフィールドを示していない場合、Nothing が返ります。整数が 0 から Fields.Count - 1 の間でなかった場合、ランタイム エラーが発生します。

3-2-3. Field クラス

最もよく使用されるオブジェクト クラスが Field クラスです。Field クラスの各オブジェクトはフォーム上の一つのフィールドのデータを含んでいます。Field クラスはフォーム スクリプト、エクスポート スクリプト、ライブラリ スクリプトで使用することができます。エクスポート スクリプトでは Fields コレクションを通じてのみ Field オブジェクトにアクセスすることができます。

- Field クラス情報の参照

Field クラス情報を参照するには以下のようにします。

FieldName.FieldPropertyName

FieldName	フォームのフィールドのフィールド ID
FieldPropertyName	フィールド クラスの適切なプロパティ

例:

City:

S	A	N		F	R	A	N	C	I	S	C	O		

State:


```
If City.Text = "SAN FRANCISCO" Then State.Text = "CA"
```

各 Field オブジェクトは次のプロパティを持っています。

プロパティ	型	アクセス	説明
Name	String	Read Only	フィールドのフィールド ID。
Type	Integer	Read Only	データ フィールド形式を表した事前に定義された値。
Text	String	Read-Write	16KB までのフィールドの実際の値。空白部はスクリプトの実行前に削除されます。 注意: 全ての TELEform オブジェクトのプロパティ同様、Text プロパティは直接代入することでのみ値を変更することができます。関数内で引数として渡された場合には、値を変更することができません。

プロパティ	型	アクセス	説明
Value	Double	Read-Write	フィールドの数値を保持しています。数値フィールドでよく使用されます。
Status	Long	Read-Write	フィールド ステータスを表す。フィールドが評価されたり修正されたりすると、このプロパティはフィールドがOKか、要再検査か指し示します。
Missing	Integer	Read Only	フィールドが属しているフォームのページの欠落ステータス。このプロパティはForm_Evaluate でのみ利用できます。 True：フォーム ページは欠落している False：フォーム ページは欠落していない マルチページ フォームの各ページは個別に評価されるため、フィールド評価の前にこのプロパティを確認する必要があるかもしれません。例えば SSN.Missing が True の場合、現在評価中のページセットに SSN フィールドが含まれていないことを意味しています。この行方のわからないフィールドは、後続のスクリプト呼び出しで処理されることになります。
Mask	String	Read-Write	Text プロパティの各文字の状態を表した 0 から 9 までの文字。0 は認識に成功したことを表しており、その他の値は次のように認識に失敗したことを表しています： 0：認識成功 1：他のエラー 2：予約 3：未入力 4：複数マーク 5：不正文字 6：あいまいなマーク 7：マーク（辞書にない単語） 8：マーク（最高精度の文字） 9：マーク（最高精度が見つからない） 重要： このプロパティは常に利用可能ではありません。（HasMask プロパティを参照） Choice フィールドは選択肢ごとに 1 つの Mask を持っています。

プロパティ	型	アクセス	説明
HasMask	Integer	Read Only	フィールドの Mask ステータス: True: Mask プロパティ有効 False: Mask プロパティ利用不可 重要: Field.HasMask が False の時に、Field.Mask にアクセスするとランタイムエラーが発生し、スクリプトの実行が終了します。
Length	Integer	Read Only	Text プロパティの最大文字数。
TabIndex	Integer	Read Only	TELEform Verifier のフォーム モードでの、フィールドのタブ オーダー。 この値は 0 から Fields.Count - 1 の間になります。 新しいタブ インデックスが設定されると、そのインデックス以降のフィールドのオーダーがずれます。 Form_Evaluate でこのプロパティにアクセスするとエラーになります。 注意: BasicScript でタブ インデックスを設定する場合は、昇順に割り当ててください。降順に割り当てると矛盾が生じる可能性があります。
TabStop	Integer	Read-Write	TELEform Verifier のフォーム モードでの、フィールドのタブ ストップのステータス。 True: Field.Status がゼロでない時 False: Field.Status がゼロの時 注意: TabStop は前バージョンとの互換性のために提供されています。極力 Field.Status を使用してください。

プロパティ	型	アクセス	説明
TopChoices	TopChoice	Read Only	TopChoice オブジェクトのコレクションです。各オブジェクトは Text プロパティの文字に相当します。 このプロパティは HasChoices プロパティが True の時に利用できます。 重要: このプロパティは常に利用可能ではありません。(HasChoices プロパティを参照)
HasChoices	Integer	Read Only	TopChoices プロパティのステータス。 True: TopChoices プロパティが定義されている False: TopChoices プロパティが定義されていない 重要: Field.HasChoices が False の時に、Field.TopChoices にアクセスするとランタイム エラーが発生し、スクリプトの実行が終了します。
SetFocus	Method		TELEform Verifier でフィールド選択に用います。 マウス操作でフィールド選択をしている場合、SetFocus は無視されます。ただし、ユーザーがフォーカスを変更しようとしているフィールドに SetFocus がされている場合は別です。これはフィールドに正しい値を入力させることを保証するのに利用できます。 SetFocus は LostFocus や GotFocus エントリー ポイントで使用できます。 注意: SKFI ストリーミング モードでは、現在の SKFI 領域外のフィールドへの SetFocus は無視されます。
Choices	Choice	Read Only	Choice オブジェクトのコレクション。 Choice オブジェクトでない場合は常に Nothing になります。 Choices はコレクションなので Count プロパティを持っています。

プロパティ	型	アクセス	説明
Count	Integer	Read Only	詳細グループの列の数。詳細グループでない場合は、常にゼロです。
( )	Row	Read Only	列のコレクション。このコレクションは詳細グループのみに適用されます。この値は 0 から DetailField.Count - 1 の間になります。 注意: 列自身もコレクションなので Count プロパティを持ちます。 インデックスを指定すると詳細グループの列を返します。GotFocus、HasFocus、LostFocus では、-1 を指定すると現在の列が返ります。
CurRow	Integer	Read Only	現在処理されている列。この値は GotFocus、HasFocus、LostFocus のみで利用可能で、詳細グループのメンバーであるフィールドに適用されます。
ImagePage-Number	Integer	Read Only	現在処理されているフォーム イメージのページ ナンバー。イメージの最初のページはページ 0 です。
DoubleKey	Integer	Read Only	フィールドがダブル キーとマークされている場合、Field.DoubleKey は 1 となり、それ以外では 0 となります。 注意: Field.DoubleKey はデータ再評価モードとフォーム モードのみで定義されます。フォーム モード修正中は 0 となります。
Image-Orientation	Integer	Read Only	現在処理されているフォーム イメージの方向を格納。このプロパティは次の値がセットされます: ImageRotate90 – ページは 90 度回転です ImageRotate180 – ページは 180 度回転です 全ての回転は、Left、Right、Top、Bottom 座標を適用する前に左回転で適用されます。つまり、座標適用前にどのくらいイメージを回転させるかを決めるプロパティです。

プロパティ	型	アクセス	説明
Lefr、Right Top、Bottom			重要: 各フィールド座標は Image-Orientation プロパティの回転後のピクセル値で表現されます。
Left	Long	Read Only	フィールドの左端の X 座標。
Right	Long	Read Only	フィールドの右端の X 座標。
Top	Long	Read Only	フィールドの上端の Y 座標。
Bottom	Long	Read Only	フィールドの下端の Y 座標。

- *FieldName*.Type プロパティ値

Field.Type プロパティは次の表の値のいずれかとなります。

Field.Type	値	説明
NumberType	1	数字
StringType	2	文字列
TextFileType	3	テキスト ファイルのファイル名
ImageFileType	4	イメージのファイル名
DetailType	5	詳細グループ レコード
ChoiceType	6	選択フィールド
KFIType	9	SKFI 領域

- *FieldName*.Status プロパティ値

次の表は Field.Status プロパティの値のリストです。これらの定数は事前に定義されていません。そのため、コードの中で使用する場合はスクリプトの上段(エントリー ポイントの前)でこれらを定義する必要があります。FldOK を除くステータスの値は Or オペレーターで結合させることができます。

例:

```
Const FldOK = 0
FieldName.Status = FldOK
```

例:

```
Const FldInvalid = 128
If Not SumsAddOK Then
    total.Status = (total.Status Or FldInvalid)
End If
```

Or を使用すると、すでに total フィールドが持っている FldReview のようなステータスを保持することが可能です。

定数	値	説明
Const FldOK	0	評価 OK
Const FldNotFilled	1	入力なし
Const FldThreshold	2	あいまいなマーク
Const FldRange	4	フィールドのプロパティ画面で定義される数値範囲外のデータ
Const FldTooMany	8	選択フィールドでの複数選択
Const FldBadInterp	16	低精度の文字認識
Const FldIOError	32	ファイル書き込み失敗 (イメージ領域)
Const FldReview	64	フィールドのプロパティ画面で「常に閲覧」がチェックされている
Const FldInvalid	128	フィールド評価失敗
Const FldLookup	256	データベース ルックアップ エラー、不正な値
Const WordNotFound	512	辞書にない単語
Const FldIllegalChar	1024	不正文字
Const FldMissingPg	2048	ページ欠落
Const FldBlankZone	4096	イメージ領域に入力なし
Const FldLengthErr	8192	長さが不正 (バー コード)
Const FldKeyNotFound #	16384	フィールドのキーが見当たらない
Const FldWordChg	32768	辞書による単語変換
Const FldInvalidDate #	268435456	不正な日付
Const FldBestGuessChar #	536870912	最高精度の文字
Const IndefiniteLocation #	1073741824	不正なフィールド オブジェクト位置

3-3. 演習

- 準備

演習を始める前にまずは環境を準備します。以下の手順を行ってください。

1. TELEform Designer で「Chapter 3 Exercise」を開いてください。
   （「Chapter 3 Exercise」は［02_フォーム］の「tf34411.tfw」です。）
2. 「フォーム」メニューから「スクリプト」を選択してください。

- 演習

「Chapter 3 Exercise」は、「Chapter 1 Exercise」と同一の単純なタイムカードです。ユーザーは勤務の開始時間と終了時間を記入し、総勤務時間を合計に記入します。（お昼などの休憩は考慮しません）

作成するスクリプトは、開始時間と終了時間を使って合計が正しいかどうかを確認し、正しくない場合は Verifier オペレーターにメッセージ ボックスを表示し、修正させます。

フォームは、以下の 3 つの区切り付きフィールドで構成されています。

- start_time（開始時間）
- end_time （終了時間）
- total_time （合計）

1. まずはスクリプトの先頭にコメントを書きます。以下のコメントを書いてください。

```
'*****************************************************************
'   タイム カード フォーム スクリプト
'
'   このスクリプトは「開始」から「終了」までの時刻を自動計算し
'   「合計」と照合します。
'*****************************************************************

Sub Form_Evaluate
```

2. 次にスクリプトで使用する定数と関数宣言を書きます。コメントと Sub Form_Evaluate の間に以下のように書いてください。

```
'*************************************************************

Const SumFail    = 0
Const SumSuccess = 1
Const FldInvalid = 128

Declare Function SumOK() As Integer

Sub Form_Evaluate
```

3. 2 で宣言した SumOK() は合計が正しいかを確認する関数です。スクリプトの一番下にその関数の実体を定義します。

```
Sub Form_Export
End Sub

Function SumOK() As Integer

        Dim start_min As Integer
        Dim end_min   As Integer
        Dim total_min As Integer

        start_min = Val(Left(start_time.Text, 2)) * 60 + Val(Mid(start_time.Text, 3, 2))
        end_min   = Val(Left(end_time.Text, 2)) * 60 + Val(Mid(end_time.Text, 3, 2))
        total_min = Val(Left(total_time.Text, 2)) * 60 + Val(Mid(total_time.Text, 3, 2))

        If total_min = end_min - start_min Then
                SumOK = SumSuccess
        Else
                SumOK = SumFail
        End If

End Function
```

4. フォーム評価の時に SumOK() を実行し、合計が正しくない場合、total_time.Status を FldInvalid にします。

```
Sub Form_Evaluate

        If SumOK = SumFail Then
                total_time.Status = (total_time.Status Or FldInvalid)
        End If

End Sub
```

5. ここで一度スクリプトをコンパイルし保存します。コンパイルでエラーが発生する場合はその内容を確認しスクリプトを修正してください。

6. スクリプトが保存できたら TELEform Designer でフォームを保存します。その後、TELEform Reader、Verifier を起動し、「Image01.tif」と「Image02.tif」を使ってスクリプトを実行してみてください。
   (各 tif ファイルは C:¥Basic Script Education¥Basic¥Chapter 3 Exercise の下にあります)

「Image02.tif」を評価すると、Verifier では、total_time フィールドにどんな値を入力しても不正となってしまいます。これはフォーム評価時に total_time.Status に FldInvalid が入ってしまったためです。

- 問題

スクリプトを完全なものとするため、total_time フィールドからフォーカスが外れる時に値を再度確認し、問題なければ Status を FldOK、問題があればメッセージを表示し再度入力を促す、ということをしたいとします。どのエントリー ポイントを使用し、どのようなロジックを実装すべきか考えてみてください。

(回答は [03_演習用データ] ― [Chapter 3 Exercise] の Answer.txt にあります。)

※ スクリプトの完成版は [03_演習用データ] ― [Chapter 3 Exercise] の Script.txt です。

第4章　エクスポート スクリプト

この章では以下のことを学びます。

- エントリー ポイント
- Export クラス
- エクスポート スクリプトの実行

4-1. エクスポート スクリプトの開き方

エクスポート スクリプトを作成、編集、コンパイルをするには BasicScript エディターを使用しなければいけません。このスクリプト エディターは TELEform Designer で「スクリプトの編集」ウィンドウを開くことで初期化されます。

新規エクスポート スクリプトを作成するには
1. TELEform Designer を起動します。
2. 「ユーティリティ」メニューから「エクスポート/システムスクリプト」をクリックします。
3. 「スクリプトの編集」ウィンドウが新規のエクスポート スクリプトを表示します。

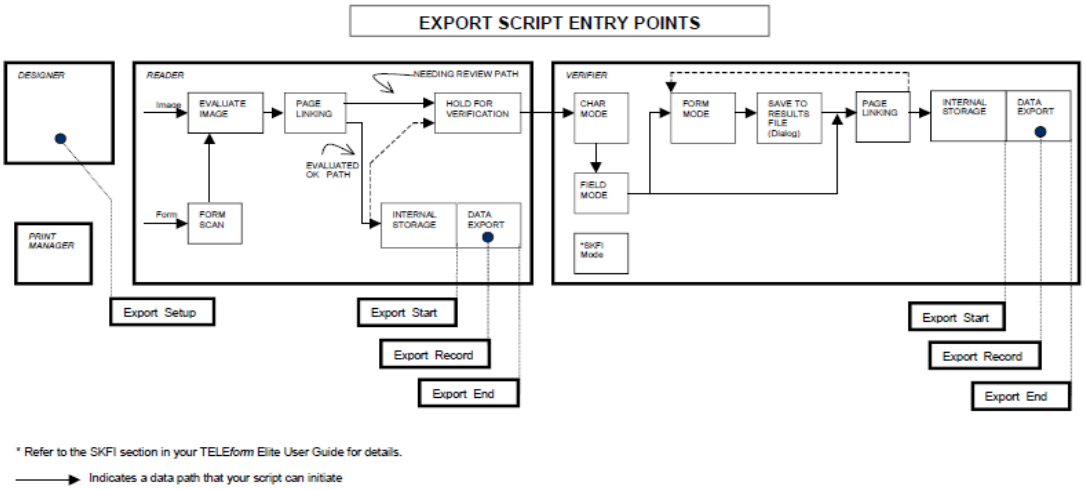
既存のエクスポート スクリプトを開くには
1. TELEform Designer を起動します。
2. 「ユーティリティ」メニューから「エクスポート/システムスクリプト」をクリックします。
「スクリプトの編集」ウィンドウが開きます。
3. 「ファイル」メニューから「開く」をクリックします。
4. スクリプトの保存に関するメッセージが表示されたら、「いいえ」を選択し Untitled スクリプトの保存をキャンセルします。
「オープンスクリプト」ダイアログが表示されます。
5. リストの中のエクスポート スクリプトを選択し、「OK」をクリックします。

注意:
エクスポート スクリプトの編集は「スクリプトの編集」ウィンドウからのみ行ってください。別のアプリケーションでスクリプトを編集した場合、コンパイルや保存が行えません。

4-2. エントリー ポイント

エクスポート スクリプトのエントリー ポイントは、TELEform がエクスポート スクリプトをどこで実行するかを示しています。

次の図は、各エクスポート スクリプトのエントリー ポイントが TELEform のデータ フロー上、いつ呼ばれるかを示しています。



各エントリー ポイントは以下のようになっています。

エントリー ポイント	説明
Sub Export_Setup : End Sub	このエントリー ポイントは TELEform Designer の「自動エクスポート設定」で「保存」を選択した際に呼ばれます。また、このエントリー ポイントはエクスポート スクリプトを保存する際や「編集機能」において、「構成ダイアログのサポート」が選択されている場合のみ呼ばれます。
Sub Export_Start : End Sub	このエントリー ポイントはエクスポート処理の開始時に呼ばれます。データ ファイルの作成などに使用されます。
Sub Export_Record : End Sub	このエントリー ポイントは、各エクスポート レコードごとに一度呼ばれます。フォーム上の全てのエクスポート フィールドにアクセスすることができます。通常はこのエントリーで各フィールドのデータ ファイル書き込みなどを行います。
Sub Export_End : End Sub	このエントリー ポイントはエクスポート処理の終了時に呼ばれます。データ ファイルの終了処理などに使用されます。

4-3. エクスポート スクリプト クラスとプロパティ

4-3-1. Export クラス

唯一エクスポート スクリプトだけが Export クラスを使用することができます。このオブジェクト クラスはエクスポート セッションに関する情報を保持しています。

- Export クラス情報の参照

Export クラス情報を参照するには以下のようにします。

```
Export.ExportPropertyName
```

ExportPropertyName は Export クラスの適切なプロパティを指しています。例えば以下のようになります。

```
Open Export.Path For Output As #FileNum
```

Export クラスのプロパティは以下のように定義されています。

プロパティ	型	アクセス	説明
Capabilities	Integer	Read Only	エクスポート スクリプトの機能を表した数値を格納。 この値は「スクリプトの編集」ウィンドウの「編集」メニューの「機能」から変更することができます。  このプロパティの値は次の定数の合計になります。これらは事前に定義された値ではありませんが、以下のようにスクリプト内で定義し使用できます: Const AppendSupport = 04 - 追加モードのサポート Const HeaderSupport = 08 - ヘッダを含める Const AllowNull = 32 - NULL 値のサポート Const SetupSupport = 02 - 構成ダイアログのサポート Const NoImageTab = 524288 - イメージコピーのサポート 例えば、ヘッダと追加モードが有効の場合は Export.Capabilities = 12 となります。 各機能は以下のように And 演算子を使用して確認することができます。 If (Export.Capabilities And _ AllowNull) Then…
Path	String	Read-Write (Export_Setup では Write のみ)	データ ファイルのフル パス。これはスクリプトがフィールド データを書き込むファイルです。
MaxFields	Integer	Read Only	一度に 1 つのフォームからエクスポートされるフィールドの最大値。この値は「編集機能」で変更することができます。 範囲: 1-4096

プロパティ	型	アクセス	説明
MaxWidth	Integer	Read Only	データ エントリー フィールドに格納できるフィールド幅の最大値。 この値は「編集機能」で変更することができます。 範囲：1-16384
MaxNameLen	Integer	Read Only	データ エントリー フィールドのフィールド ID の最大長。 この値は「編集機能」で変更することができます。 範囲：1-29
Format	String	Read Only	「オープンスクリプト」ダイアログ ボックスに表示されるエクスポート スクリプトの名前。
Count	Integer	Read Only	現在処理中のエクスポート セッションの残りのフォーム数。Export_Start ではこの値はフォームのトータル数であるが、Export_Record が呼ばれるたびに 1 つずつ減っていく。
Append	Integer	Read Only	エクスポートの追加モードのステータス。 True：ファイルはすでに存在し追加中 False：新規ファイル
Header	Integer	Read Only	エクスポートのヘッダのステータス。 True：「ヘッダを含める」を選択 False：「ヘッダを含める」を未選択
Result	Long	Read-Write	レコードのエクスポートの結果を格納。この値は、各 Export_Record の最後で TELEform に返ります。 エクスポートに失敗した場合はゼロでない値（通常-1）を、成功した場合はゼロ（0）を格納します。
Master	Export	Read Only	処理中のエクスポート処理が詳細グループのネストされたエクスポート処理である場合、マスターが返ります。詳細グループの各列は、エクスポート処理時は別々のレコードとして扱われます。 このプロパティの値は： ・ フォーム レベルのエクスポートの場合は Nothing ・ 詳細グループの場合は値

- 詳細グループのエクスポート

エクスポート スクリプトを作成する際、通常詳細グループのための特別な処理は必要ありません。エクスポート スクリプトはマスター フォーム レコードの時と同様、詳細グループの各列に対して呼ばれます。

TELEform はマスター レコードよりも詳細グループを先に処理するため、固定されたファイル ハンドルは使用すべきではありません。この場合、FeeFile() 関数を使用し、利用可能なファイル ハンドルを取得してください。

もし詳細グループに対して特別な処理をしたい場合、Export.Master がマスター エクスポート オブジェクトを指し示していますので、これを利用してください。マスター フォーム レコードのエクスポート時は Export.Master は Nothing となっています。

4-3-2. Field クラス

エクスポート スクリプトは Fields コレクションを通じて Field クラスへアクセスします。*FieldName* を使用することはできませんので注意してください。

4-4. エクスポート スクリプトの実行

エクスポート スクリプトを実行するには、まずはTELEform Designerの「スクリプトの編集」ウィンドウでスクリプトをコンパイルし、保存をする必要があります。コンパイル エラーが発生した場合は、スクリプトの実行を行う前に、そのエラーを解決しなければいけません。

エクスポート スクリプトを実行するには以下の手順に従ってください。

1. 新規エクスポート スクリプトを作成し、TELEform Designer を終了します。

重要:
「自動エクスポート設定」の形式リストにこの新しいエクスポート形式を登録するには TELEform Desinger を再起動する必要があります。

2. TELEform Designer を起動します。
3. フォームを開きます。
4. 「フォーム」メニューから「自動エクスポート設定」をクリックし、ダイアログ ボックスを表示します。
5. 「新規」ボタンをクリックします。
6. 「メイン」タブで「形式」から作成したエクスポート スクリプトを選択し「保存」をクリックします。
7. Export_Setup エントリー ポイントで指定したパスが表示されることを確認します。

注意:
「保存」ボタンをクリックしても何も起こらない場合、「構成ダイアログのサポート」をチェックしているにも関わらずExport_Setupエントリー ポイントにコードがありません。この場合は「構成ダイアログのサポート」のチェックを外してください。

8. 「有効」をチェックし「OK」ボタンをクリックください。
9. 「自動エクスポート設定」ダイアログ ボックスで、エクスポート パスと有効のチェックを確認してください。問題がなければ「OK」をクリックします。
10. フォームを保存します。
11. TELEform Reader、Verifier を再起動します。
12. TELEform Reader でフォーム イメージを評価します。
13. TELEform Verifier でフォーム イメージを修正します。エクスポート処理は自動で実行されます。

※ 「ユーティリティ」メニューの「内部データエクスポート」を使用して手動エクスポートを実行することもできます。

4-5. 演習

- 準備

演習を始める前にまずは環境を準備します。以下の手順を行ってください。

1. TELEform Designer で「Chapter 4 Exercise」を開いてください。
   (「Chapter 4 Exercise」は[02_フォーム]の「tf48673.tfw」です。)
2. 「ユーティリティ」メニューから「エクスポート/システムスクリプト」を選択して「スクリプトの編集」ウィンドウを開いてください。
3. 「ファイル」メニューから「新規」 → 「エクスポートスクリプト」を選んでください。

- 演習

「Chapter 4 Exercise」はあるお店の注文書です。顧客は欲しい商品の数量を書き込んで送付します。

今回の演習では、入力されたデータをファイルに出力するスクリプトを作成します。

1. Export_Setup エントリー ポイントに出力するデータ ファイルを指定します。以下のコードを入力してください。

```
Sub Export_Setup

        Export.Path = "C:¥Export.log"

End Sub
```

2. 複数のエントリー ポイント サブルーチンで使用する出力ファイルのファイル番号の変数を宣言します。

```
Dim fileNum As Integer

Sub Export_Setup
```

3. ファイル出力は以下のように行うものとし、その機能を実装します。
・ Export_Start エントリー ポイントでファイルをオープン
・ Export_Record エントリー ポイントでファイルにデータを入力
・ Export_End エントリー ポイントでファイルをクローズ

```
Sub Export_Start

        fileNum = FreeFile()

        If FileExists(Export.Path) Then
                Open Export.Path For Append Access Read Write Lock Write As #fileNum
        Else
                Open Export.Path For Output Access Write Lock Write As #fileNum
        End If

End Sub

Sub Export_Record

        Print #fileNum, "エクスポートしました。"

End Sub

Sub Export_End

        Close #fileNum

End Sub
```

4. 「ファイル」メニューから「名前を付けて保存」をクリックします。ダイアログで「エクスポート形式名」を「Chapter 4 Exercise」とし、「構成ダイアログのサポート」にチェックをして「OK」を押します。

「4.4 エクスポート スクリプトの実行」を参照して、エクスポート スクリプトを実行してください。イメージ ファイルは［03_演習用データ］－［Chapter 4 Exercise］の「Image01.tif」を使用してください。

「C:¥Export.log」が作成され、「エクスポートしました。」と書き込まれていることを確認します。

- 問題

まず、「自動エクスポート設定」でエクスポートするフィールドを su_1、su_2、su_3 に限定します。このとき、これらのフィールドの値をファイルに出力するにはどうしたらよいでしょうか。

（回答は［03_演習用データ］－［Chapter 4 Exercise］の「Answer.txt」にあります。）

※ スクリプトの完成版は［03_演習用データ］－［Chapter 4 Exercise］の「Script.txt」です。

Cardiff TELE*form*® v8.2

BasicScript™ ガイド

著者･発行者

株式会社ハンモック