MITSUBISHI

三菱マッピングシステム PreSetV WebTcl

WebTcl アプリケーション作成ガイド

2008/04/25 版

作成ガイド

はじめに

本書は、クライアント・サーバ(以下 C/S)形式の PreSerV Tcl I/F(以下 C/S 版)システムを対象に、PreSerV WebTcl(以下 WebTcl)を適用して Web アプリケーションに移植するための手順を示します。

本書は、1 章で移行の概要の説明、2 章~9 章で WebTcl への移行(以下移行)手順を説明します。

この作成ガイドの表記に関して

	非常に重要な注意事項を説明しています。 注意を見逃すとトラブルの原因となる可能性があります。必ずご一読願います。
	作成にあたって、気にとめていただきたいことを説明しています。
	本ガイドの補足説明として、参照していただきたいドキュメント名を示します。

作成ガイド早見表

アプリケーション作成ガイドの早見表です。

関連ドキュメントは  の行に記述します。必要な場合は関連ドキュメントを参照してください。

アプリケーション作成前にお読みください	対象システムの確認する	１章へ
	作成作業のフローを事前学習する	２章へ
	必要な情報を調査する	３章へ

アプリケーション作成時にお読みください	開発環境を作成する  『インストールガイド』  『WebTclSample インストール手順』	４章へ
	設計を行う	５章 ６章へ
	実際に作ってみる  『WebTcl 概要書』  『WebTcl API リファレンス』  『WebTcl Tcl コマンドリファレンス』	７章へ
	サーバ・クライアントで繋いで動かす	８章へ
	実行環境を抽出する	９章へ

アプリケーション作成後に知りたいこと	困った時  『WebTcl FAQ 集』	
	他に資料がないか探している時	10 章へ

目次

1 対象システム

本ガイドは、既存の Tcl 版を利用した業務システムを、Web システムへ効率よく移行することを目的としています。
移行はプレゼンテーション層の移行手順と、ビジネスロジック層に相当するユーザ・アプリケーションの移植に分かれます。
図 1-1に一般的なDBアクセスを伴うC/SシステムをJ2EEベースのWebシステムに移行する場合の構成と、PreSerVのC/SシステムをWebシステムへ移行する場合の比較を図示します。
WebTcl ではプレゼンテーション層にあたる PreSerV マクロが流用できるため、画面遷移の再設計が不要です。

※PSV クライアント関数を使用した、C/S 版のユーザ・アプリケーションを、以後「クライアント AP」と表記します。また WebTcl で Java 言語に移植したアプリケーションも「クライアント AP」と総称します。

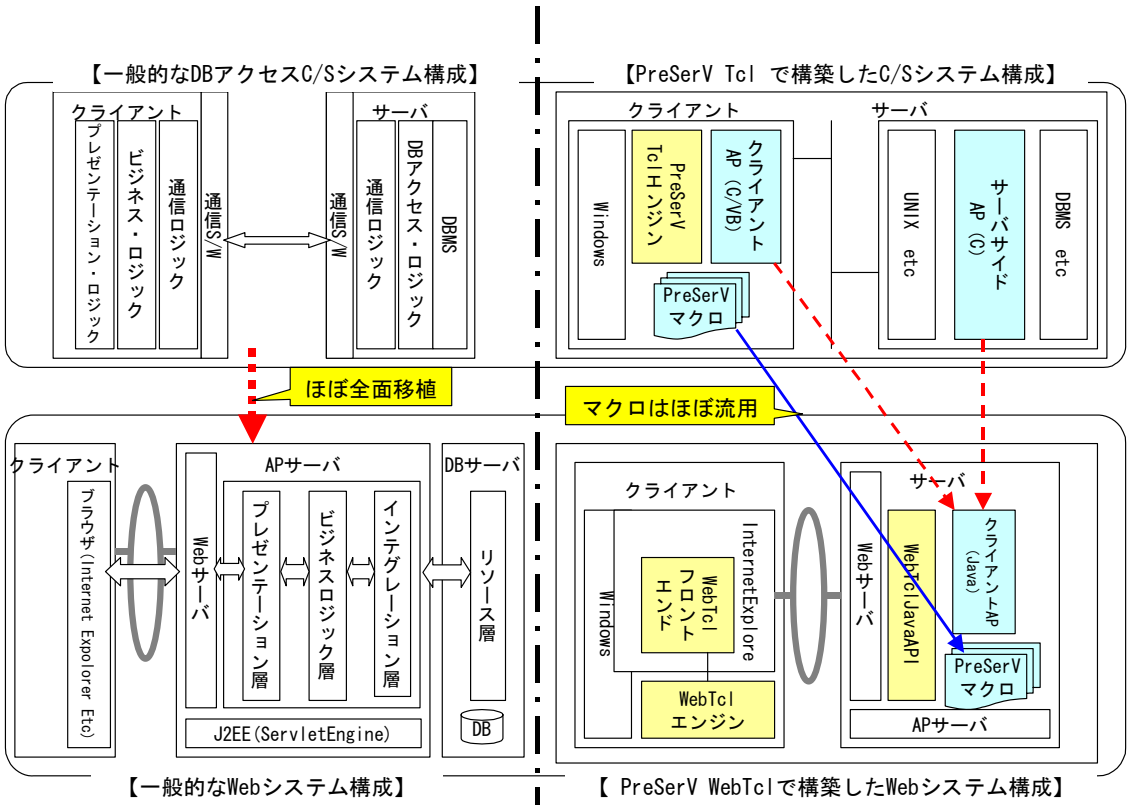


図 1-1 一般的な Web システム移行と WebTcl の比較

2 作業フロー(概要)

PreSerV の C/S 版から WebTcl への移行作業は以下の手順で行います。
新規 WebTcl アプリケーション作成の場合は、４章からお読みください。

3 事前調査作業

移行作業前に以下の作業を行ってください。

3.1 C/S 版の S/W 構成の調査

C/S 版では以下の S/W 構成パターンに大別されます。移行する対象の C/S 版アプリケーションの S/W 構成を確認し、必要な情報の収集を行う必要があります。

「WebTcl 概要書」→「PreServV WebTcl 製品概要」→「処理フローの違い」

以下は通信フローで S/W 構成を分類した例です。

(1) クライアントアプリケーションがないパターン

PSV マクロファイル単独で動作するアプリケーションで、色定義ツールなどがこの分類に相当します。サーバとの通信の必要がない一番単純なアプリケーション形態です。

このS/W構成の場合、起動HTMLの作成(5.4参照)だけで移行が可能です。

(2) クライアントアプリケーションパターン-1

クライアントアプリケーションと Tcl エンジンが 1 対 1 で通信するパターンをパターン-1 と定義します。

起動時の PreServV マクロと、クライアントアプリケーションが e_send と PSV_Send を中心に通信処理を行うことで、処理が構成されます。

本ガイドで移行を検討する範囲は、クライアントアプリケーションパターン-1 です。

(3) クライアントアプリケーションパターン-2

クライアントアプリケーションとTclエンジンがｎ対１で通信する※1パターンをパターン-2 と定義します。

※1 クライアントアプリケーションがｎ個の場合です。

C/S 版は、クライアント関数と E マクロの通信 API は TCP/IP を経由したプロセス間通信を行うため、設計上は Tcl エンジンとクライアントアプリケーションが別の PC に分散してｎ個あるシステムも構築可能です。

パターン-2 で C/S 版のシステムを構築している場合は、ネットワーク概念の移行を検討する必要があります。パターン-2 は、システムごとの検討が必要なため、本書の対象からはずします。

3.2 変更、追加インタフェース概要

WebTcl ではインタフェースに変更、追加が行なわれています。事前に仕様変更点を調査し、影響範囲を把握して製作の計画に作業を織り込んでください。

「WebTcl 概要書」→「PreServV WebTcl 製品概要」→「API 変更概要」

4 開発環境構築

WebTcl を利用したアプリケーションの開発には以下の環境構築が必要です。

4.1 PreSerV V5 基本パックのインストール

WebTcl を利用したアプリケーションではベクトルシンボル生成ツールなどのツール群は C/S 版環境のツールを使用するため C/S 版の開発環境が必要です。WebTcl には対応する C/S 版「PreSerV V5 基本パック」の CD が同封されます。

CDをPCに入れると自動再生でインストールダイアログが表示されます（図 4-1参照）。

インストール手順は添付の「インストールガイド」を参照してください。

図 4-1 「PreSerV V5 基本パック」Ver.5.5 インストールダイアログ

4.2 PreSerV WebTcl Basic Media のインストール

WebTcl の基本的なモジュール、ドキュメントを PC にインストールするには、「PreSerV WebTcl Basic Media」の CD を使用してください。

CDをPCに入れると自動再生でインストールダイアログが表示されます（図 4-2参照）。

インストール手順は添付の「インストールガイド」を参照してください。

図 4-2 「PreSerV WebTcl Basic Media」インストールダイアログ

4.3 DB サーバの環境構築

ビジネスロジック層から DB にアクセスするアプリケーションの場合、JDBC もしくはフレームワークのインテグレーション層経由で DB アクセス可能なように DB サーバの環境を設定してください。

DB サーバの環境に関しては、対象業務システムの設計に依存します。

AP サーバからアクセス可能な DB の場合、WebTcl の制約はありません。

本ドキュメントでは、DB サーバに関する環境構築手順の説明は省略します。

4.4 Web サーバ・AP サーバの環境構築

Web サーバ・AP サーバのマニュアルを参考に J2EE1.4 の開発・動作環境を構築してください。

WebTcl は最小限の機能の Servlet コンテナ上で動作します。

本ドキュメントでは､Web サーバ･AP サーバに関する環境構築手順の説明は省略します。

5 画面(静的コンテンツ)設計

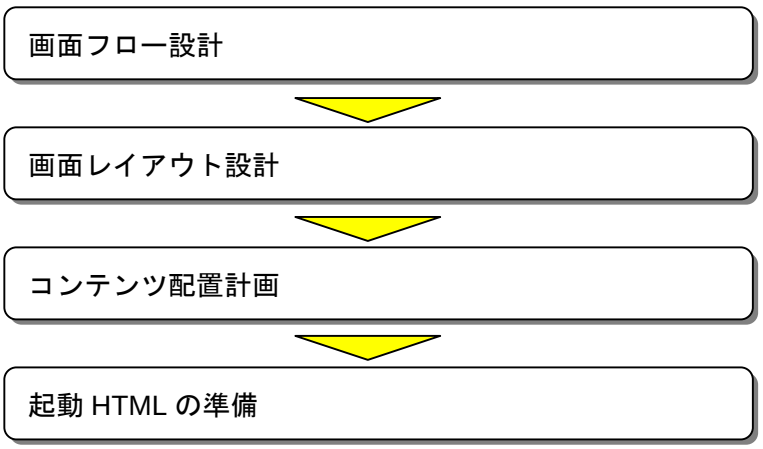
WebTclでは表 5-1のプログラム群を「静的コンテンツ」[注1]と分類します。

注１）“静的”とは、これらのファイルがアプリケーション実行時に更新することがなく、あらかじめ決められた命令を実行することを示しています。WebTcl ではアプリケーション作成の分類上「静的コンテンツ」と「動的コンテンツ」と名称をつけています。

表 5-1 WebTcl の静的コンテンツ分類一覧

プラグラム分類	説明
画面生成マクロ	画面を構築する部品を記述した PSV マクロスクリプト。
起動 HTML	WebTcl の起動を行う HTML スクリプト。
PSV マクロや HTML から参照されるデータ	画面に貼り付けるイメージファイルなどのデータファイル。

画面設計ではこれらのコンテンツの設計を以下の項目に分けて説明します。



本章では、この段階までに、既に業務要件が明確になっていることを前提として説明を行います。

5.1 画面フロー設計

画面フローはシステムごとに異なりますので、１つの例を提示しながら説明します。
画面フロー設計では、以下の情報を決定してください。

(1) 画面一覧作成

画面ごとに ID を割り当ててください。この画面 ID は、後で PSV マクロの配列変数名などに使用することで、画面ごとにリソースを分離して管理することができます。

例）画面 ID=GYM001 のトップウィンドウの ID は$GYM001(TOP)に格納するなど。

→新規作成の場合は「画面一覧」を別途設計してください。

(2) 画面遷移図作成

画面のリンク関係をまとめたものを遷移図として例示します。

→新規作成の場合は「画面遷移図」を別途設計してください。

(3) WebTcl 適用範囲決定

「画面一覧」と「画面遷移図」から WebTcl で画面作成する範囲を決定してください。

→「画面一覧」と「画面遷移図」に WebTcl で作成するマークを追記。

(4) イベント一覧

機能、イベント処理をそれぞれ一覧として作成してください。

WebTcl ではイベントは以下の２種類に分けてください。

① サーバ AP にリクエストを送らないイベント処理

画面内の制御だけとなるイベント処理を示します。

静的コンテンツで作成するのはこのイベント処理です。

② サーバ AP にリクエストを送るイベント処理

画面内の制御だけでなく、サーバからリストの一覧を取得するイベント処理などを示します。

このイベント処理は“サーバ AP(動的コンテンツ)の設計”で参照します。

→新規作成の場合は「イベント一覧」を別途設計してください。

5.2 画面レイアウト設計

5.2.1 GUI ビルダーによる部品配置の試作

ここでは新規に画面を作成する場合の例を説明します。

5.1－(3)で決定したWebTclで作成する画面の部品配置を試作する場合、C/S版の「GUI ビルダー」を使用します（図 5-1参照）。

REF. **『PreServ V5 基本パック』→「オンラインマニュアル」→「ユーティリティ説明書」→「１章 GUI ビルダー」**

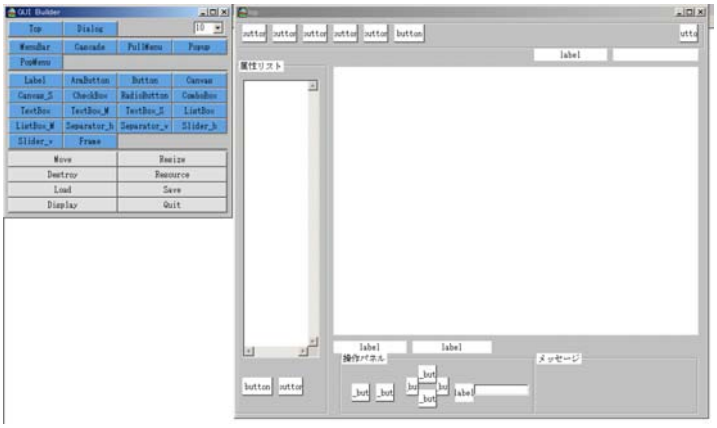


図 5-1 GUI ビルダーで画面部品を配置している例

画面内レイアウトはアプリケーションごとに異なります。ここでは図 5-1の完成形のウィンドウを例として説明します（図 5-2参照）。

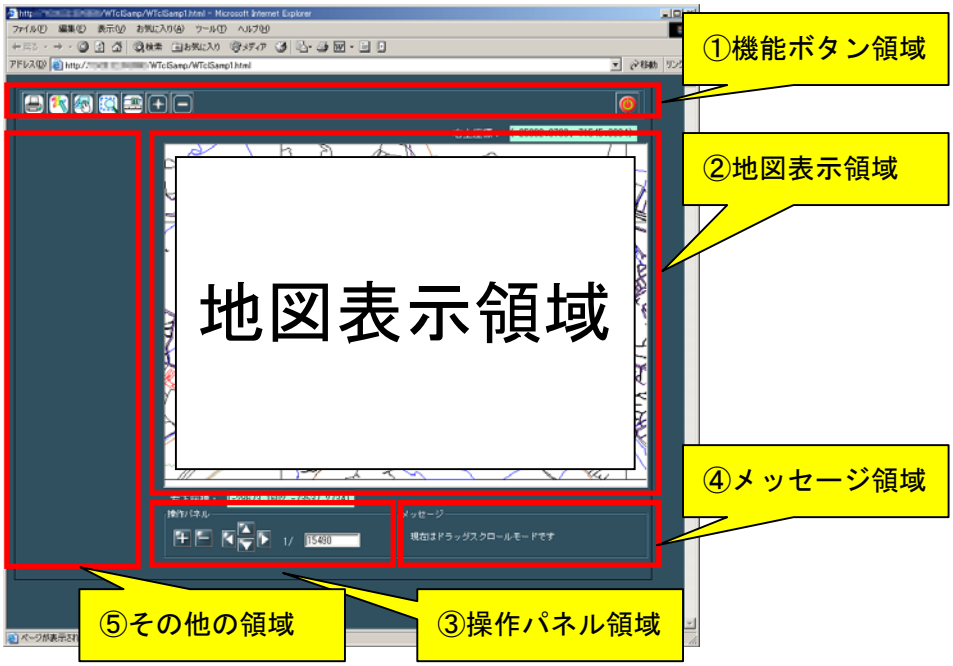


図 5-2 画面レイアウト例

基本的に以下の５項目に分けて画面レイアウトを設計します。

① 機能ボタン領域(w_button)

頻繁に使う機能をボタンとして配列し、操作効率を図ります。基本的に各機能は排他的に動作するものが対象です。

図 5-2の例では「印刷」、「色定義」、「ドラッグスクロール」などの機能ボタンをレイアウトしています。

② 地図表示領域(w_canvas)

地図を表示・操作するキャンバスの領域です。基本的に本領域が画面の大半を占めるようにレイアウトします。

③ 操作パネル領域(w_arm_button)

連続４方向スクロールや、連続固定倍率拡大縮小など画面操作の共通的な機能をまとめた領域です。

④ メッセージ領域(w_label)

現在の操作モードなどユーザの操作ガイダンスとなるメッセージを表示する領域です。

⑤ その他の領域(w_list_box など)

その他の領域としては操作の補助的なものとして、以下の機能があります。

注）これらの画面領域はデータの準備が必要なため、サンプルでは実装を省略しています。

(a) 属性のリストなどインデックス的なもの

例）目標物や設備の一覧などをリスト表示し、ダブルクリックで②の領域がジャンプするなど。

(b) 概要図

②の 10 倍のスケール分母で地図を表示し、処理対象地図領域の全体図を表示します。

例）神奈川県の概要図を表示し、②の地図領域が現在神奈川県のどの辺を表示しているか把握できるようにするなど。

5.2.2 画面レイアウトの完成

「GUIビルダー」で画面を作成後、「Save」ボタンをクリックし、画面を構築するWマクロの命令をPSVマクロとして出力します（リスト 5-1参照）。

リスト 5-1 出力 PSV マクロ例

```
w_def_size 1280 1005 off
w_top w_top1 -x 380 -y 0 -w 774 -h 654 -t top -b &c0c0c0
w_frame w_frame1 @w_top1 -x 0 -y 0 -w 770 -h 50 -f &000000 -b &ffffff -v m7
w_canvas w_canvas1 @w_top1 -x 160 -y 80 -w 600 -h 440 -b &ffffff -s on
w_canvas w_canvas2 @w_top1 -x 620 -y 50 -w 140 -h 20 -b &ffffff
w_frame w_frame2 @w_top1 -x 490 -y 550 -w 270 -h 100 -f &000000 -b &ffffff -l メッセージ -v m7
w_frame w_frame3 @w_top1 -x 170 -y 550 -w 320 -h 100 -f &000000 -b &ffffff -l 操作パネル -v m7
w_frame w_frame4 @w_top1 -x 0 -y 70 -w 150 -h 580 -f &000000 -b &ffffff -l 属性リスト -v m7
w_label w_label2 @w_top1 -x 160 -y 530 -w 120 -h 20 -f &000000 -b &ffffff -l label -v m7 -a c -o 0
w_arm_button w_arm_button1 @w_top1 -x 190 -y 600 -w 30 -h 30 -f &000000 -b &ffffff -l arm_button -v
m7 -s on -i 100
w_arm_button w_arm_button2 @w_top1 -x 230 -y 600 -w 30 -h 30 -f &000000 -b &ffffff -l arm_button -v
m7 -s on -i 100
w_arm_button w_arm_button3 @w_top1 -x 300 -y 570 -w 30 -h 30 -f &000000 -b &ffffff -l arm_button -v
m7 -s on -i 100
w_arm_button w_arm_button4 @w_top1 -x 300 -y 610 -w 30 -h 30 -f &000000 -b &ffffff -l arm_button -v
m7 -s on -i 100
```

GUI ビルダーでは、完全な画面構築スクリプトは作成できないため、出力された PSV マクロに以下の修正を行って、完成させます。

(1) 画面部品のウィンドウ ID の管理

出力された PSV マクロでは画面部品を作るだけで、ウィンドウ ID をイベント処理実装時に参照するなどの処理作りこみができません。

またタグ(リスト中の"@w_top1")も画面作成ごとに連番となるため、複数画面を作成し、同時に実行した場合タグが重複することが想定されます。

そこで PSV マクロを仕上げる時は、既に決定済みの画面 ID を変数名として変数領域を分割してウィンドウ ID を管理することをお奨めします。

また定数値は同様に画面 ID の変数領域に格納して、PSV マクロ内で使用することをお奨めします。

変更前をリスト 5-2、変更後をリスト 5-3に例示します。

リスト 5-2 変更前 PSV マクロ例

```
w_top w_top1 -x 380 -y 0 -w 774 -h 654 -t top -b &c0c0c0
w_frame w_frame1 @w_top1 -x 0 -y 0 -w 770 -h 50 -f &000000 -b &ffffff -v m7
w_canvas w_canvas1 @w_top1 -x 160 -y 80 -w 600 -h 440 -b &ffffff -s on
…
```

リスト 5-3 変数 GYM001 で管理する変更を行った PSV マクロ例

```
global GYM001
…
set GYM001(top) [w_top w_top1 -x 380 -y 0 -w 774 -h 654 -t top -b &c0c0c0]
set GYM001(fr1) [w_frame w_frame1 $GYM001(top) -x 0 -y 0 -w 770 -h 50 -f &000000 -b &ffffff -v m7]
set GYM001(can) [w_canvas w_canvas1 $GYM001(top) -x 160 -y 80 -w 600 -h 440 -b &ffffff -s on]
…
```

(2) 画面解像度を考慮したトップウィンドウのサイズ修正

IE 上に画面を埋め込む場合を考慮して、トップウィンドウを画面解像度より 1 サイズ下のサイズとすることをお奨めします。

画面解像度が 1024x768 とする場合、トップウィンドウは 800x600 程度のサイズを目安としてサイズ修正してください（リスト 5-4修正例）。

リスト 5-4 トップウィンドウサイズ修正例

```
…
set GYM001(top) [w_top w_top1 -x 380 -y 0 -w 800 -h 600 -t top -b &c0c0c0]
…
```

(3) 追加のオプション指定

GUI ビルダーでは指定できないオプションを追記します。

(4) 画面レイアウトの確認

修正したマクロを C/S 版の psvsource.exe コマンドで読み込み、部品の配置などを確認して微調整してください。

5.3 コンテンツ配置計画

以下の方針で PSV マクロを Web サーバ上に配置を計画し、PSV マクロ間の呼出のパス名を決定してください。

(1) 基本的に相対パスで PSV マクロにアクセスするように配置してください。

FE-COM オブジェクトの CodeBase プロパティで指定する Web サーバの URL を、PSV マクロのルートディレクトリとして、全て相対パスでアクセスするように配置してください。

PSV マクロ中の"source"コマンドなどで、絶対パスのアクセス部分は相対パスに修正し、配置を計画してください。

例）A.psv からの B.psv 読み込みが、A.psv 中に"source C:¥pjB¥macros¥B.psv"と記述されている場合

※A.psv は C:¥pjA¥macros¥A.psv に配置しているケースを想定

① B.psv を Web サーバ上に配置する位置を決定

FE-COM.CodeBase プロパティから相対パスで定義可能な位置に決定してください。

② A.psv を修正

"source ../pjB/macros/B.psv"に修正

相対パスの指定が、さかのぼる場合("../")は Web サーバのアクセス制限を考慮してください。1 つ上のレベルに移動不可など Web サーバのセキュリティ面の設定でアクセスが制限される可能性があります。

(2) ローカルファイルの source コマンド実行は避けてください。

C/S 版では Tcl のファイル出力を利用して、動的にマクロファイルを生成し、source コマンドを実行することも可能です。

この source コマンドの使用方法を実施している場合は、以下の対応を行ってください。

① ファイル出力する命令を変数に設定

```
set innerEvalCmd "w_top top1 ..."
```

② source する処理を、変数の"eval"に修正

```
eval $innerEvalCmd
```

この修正で動的なマクロファイルの作成と、source コマンドの実行が可能です。

5.4 起動 HTML の準備

起動 HTML には以下の３つの HTML タグと JavaScript 記述が必須です。

(1) FE-COM のオブジェクト生成記述

表 5-2 に<OBJECT>タグで定義する属性名を示します。

リスト 5-5に示すように、<OBJECT>タグにFE-COMを特定するクラスIDと、基本モジュールのインストール媒体のURL、FE-COMのバージョンを記述します。

表 5-2 <OBJECT>タグの属性名一覧

属性名	説明	定義例
id	要素に固有の ID を付けます。	“WTclFE-COM1”
classid	そのオブジェクトを再生するための実装を具体的に指定します。FE-COM を指定する場合は、”CLSID:”に続けて、グローバル UID を指定します。	″CLSID:2358453A-29C8-4D89-BB1D-63C391A92840″
codebase	classid、data、archive のベース URI を指定します。”#”以降にバージョン文字列を指定して、インストール済みの FE-COM オブジェクトが古いかチェックします。	“http://quinsac:8090/WebTclInst.exe#version=1,2,5,1″
width	オブジェクトの幅をピクセルサイズで指定します。	″600″
height	オブジェクトの高さをピクセルサイズで指定します。	“510”

リスト 5-5 FE-COM オブジェクト生成記述例

```
<OBJECT id="WTclFE-COM1" classid="CLSID:2358453A-29C8-4D89-BB1D-63C391A92840"
codebase="http://quinsac:8080/WebTclInst.exe#version=5,5,0,1" align="baseline" border="1"
width="600" height="510"></OBJECT>
```

リスト 5-5の例ではhttp://quinsac:8080/からWebTclInst.exeのversion5.5.0.1をダウンロードし、600ｘ510 ドットのFE-COMウィンドウを作成する例です。

classid で指定したオブジェクトがインストール済みでない場合は、codebase にアクセスし自動的にインストーラが実行されます。初回インストール時は新規追加モードでインストーラが実行されます。

また、既にインストール済みの FE-COM が version5.5.0.1 未満の時も codebase に

アクセスし自動的にインストーラが実行されます。既にインストールした履歴がある場合は更新モードでインストーラが実行されます。
またJavaScriptからは"id"で指定された名称(WTclFE-COM1)でオブジェクトにアクセスすることができます。

(2) FE-COM のプロパティ定義

FE-COM オブジェクトの動作を定義するプロパティを JavaScript で定義します。
この JavaScript 記述は、起動 HTML のロード時に実行される個所に定義してください。

※ JavaScript の function でプロパティ定義部分を実装し、起動 HTML ロードの後で実行する場合、正常に動作しません。。

表 5-3にFE-COMのプロパティー覧を記載します。リスト 5-6にFE-COMのプロパティ定義例を示します。


最新のプロパティは、FE-COM API リファレンスを参照してください。

表 5-3 FE-COM のプロパティー覧

プロパティ名	説明	定義値説明
InitSource	「初回起動マクロ実行 URL」ー必須 FE-COM 生成時にメニュー画面のマクロが格納される URL 指定し、画面遷移のルートとなる画面を表示します。注）このプロパティは最後に定義してください。	マクロが格納された、Web サーバの URL を指定します。
InitActionURL	「AP サーバ接続 URL」ー必須 e_connect 発行時に、AP サーバ上で実行されるアクションの URL を定義します。	e_connect 時の AP サーバ側アクションの URL を指定します。
EndActionURL	「AP サーバ切断 URL」ー必須 e_close 発行時に、AP サーバ上で実行されるアクションの URL を定義します。	e_close 時の AP サーバ側アクションの URL を指定します。
CodeBase	「コードベース」ー必須 コードベースは、PSV マクロの相対パスの前に結合して URL 生成を行い、ダウンロードで使用します。PSV マクロ格納のルートディレクトリを指定します。	source コマンドの相対パスとあわせて完全な URL となる文字列を指定します。

プロパティ名	説明	定義値説明
AskRequestURL	「サーバ要求問合せ URL」－省略可能 e_recv 発行時に、AP サーバ上でストックしたデータ文字列が存在しないか、ポーリングする URL です。データが存在したときだけ、URL で指定されたアクションが実行されます（※ない時はフィルターが自動応答）。	e_recv でレスポンスを返すアクションの URL を指定します。
TclVersionOption	「互換モードオプション」－省略可能 互換モードの引数を指定します。 “-A”指定時は V5 の A 版互換モード相当の動作を行います。 “-Z”指定時は V5 版で A 版の API を動作可能にする A 版統合モードの動作を行います。	
InitModifyString	「初回 URL 後方修飾文字列」 e_connect の POST 動作時に、アクション URL の後方に追加するクエリー文字列を指定します。先頭の”&”は自動的に付加されます。２つめ以降のパラメータは、”&”で区切って指定してください。	POST 時の URL は InitActionURL + “&” + InitModifyString の形式で加工されます。
ModifyString	「通常 URL 後方修飾文字列」 e_send の POST 動作時に、アクション URL の後方に追加するクエリー文字列を指定します。先頭の”&”は自動的に付加されます。２つめ以降のパラメータは、”&”で区切って指定してください。注）e_send 時のアクション URL は InitActionURL のホスト名までの URL 部分をベースとして、アクション URI で定義される部分を AP サーバ上の相対パスとして結合した URL になります。	POST 時の URL は InitActionURL のホスト名まで +アクション URI + “&” + InitModifyString の形式で加工されます。
EndModifyString	「終了 URL 後方修飾文字列」 e_close の POST 動作時に、アクション URL の後方に追加するクエリー文字列を指定します。先頭の”&”は自動的に付加されます。２つめ以降のパラメータは、”&”で区切って指定してください。	POST 時の URL は EndActionURL + “&” + EndModifyString の形式で加工されます。

プロパティ名	説明	定義値説明
DebugPolicy	「デバッグ・ポリシー」 デバッグログの出力レベルを設定します。 0：ログ出力なし 1:Windows テンポラリディレクトリに WebTclError.log を出力します。 2:ダイアログでエラーメッセージを表示します。 3:1+2 のエラーログ出力・表示を行います。 注）FE-COM の致命的エラーが発生した場合、未設定・設定値に関わらず、設定値:3 の動作を行います。	0～3 の範囲の数値を定義します。 範囲外の数値の場合、デフォルト値 0 が適用されます。
CertRetryCount	「認証リトライ回数指定プロパティ」 認証ダイアログで認証失敗時のリトライ回数を指定します。	正の値が有効。 デフォルト 10
TimeOutExecCmd	「通信タイムアウト実行マクロ指定プロパティ」 TimeOutVal 秒通信処理がなかった場合、実行する Tcl マクロを指定します。 上記 Tcl マクロ実行後、TimeOutVal(TimeOutExecInterval 指定なし時)orTimeOutExecInterval(TimeOutExecInterval 指定あり時)秒後までに通信処理がなかった場合、再度本マクロを実行します。 主に、通信タイムアウトの警告や、データの自動保存目的に使用されます。	
TimeOutExecInterval	「タイムアウト時マクロ間隔指定プロパティ」 TimeOutExecCmd プロパティで指定するマクロの２回目以降の実行間隔を TimeOutVal から変更したいときに指定します。 途中で通信処理が行われた場合、秒数のカウントをリセットします。	
TimeOutVal	「通信タイムアウト時間指定プロパティ」 通信終了後、及び、通信タイムアウト発生後、本プロパティで指定される秒数通信がなかった場合、TimeOutExecCmd プロパティで指定されるマクロを実行します。 途中で通信処理が行われた場合、秒数のカウントをリセットします。	

プロパティ名	説明	定義値説明
TclIndexURL	「インデックスファイル名指定プロパティ」 インデックスファイルの URL を定義します。 インデックスファイルを生成し、本プロパティを設定することにより、アプリケーションは各プロシジャの物理構成を意識することなく利用可能となります。	
ActionCodeBase	「アクション・マップ加工用 URL 指定プロパティ」 アクション URI の登録を行う際に使用するコードベース URL を定義します。 本プロパティを利用しない場合は、InitActionURL のホスト名＋ポート番号をコードベース URL として利用します。	
DuplicationErrorMessage	「重複起動エラーメッセージ指定プロパティ」 重複起動した際に表示されるエラーメッセージを定義します。 本プロパティを利用しない場合は、デフォルトの重複起動エラーメッセージ、"WebTcl の重複起動はできません。"を表示します。	
HideMode	「非表示モード指定プロパティ」 起動 HTML の表示・非表示を定義します。 0:起動 HTML を表示 1:起動 HTML をタスクバーへアイコン化 2:起動 HTML を通知領域にアイコン化。 注）本プロパティに 1 または 2 を指定した場合、画面を起動 HTML に埋め込むことができません。画面フロー設計の際に考慮が必要です。	0～2 の範囲の数値を定義します。 範囲外の数値の場合、デフォルト値 0 が適用されます。

リスト 5-6 FE-COM のプロパティ記述例

```
<script language="JavaScript">
    WTclFE-COM1.InitActionURL = "http://quinsac:8080/PSVJSFront/servlet/InitAction";
    WTclFE-COM1.EndActionURL = "http://quinsac:8080/PSVJSFront/servlet/EndAction";
    WTclFE-COM1.AskRequestURL
                    = "http://quinsac:8080/PSVJSFront/servlet/AutoResponseAction";
    WTclFE-COM1.CodeBase   = "http://quinsac:8080/macros";
    WTclFE-COM1.InitModifyString = "henn=hennda";
    WTclFE-COM1.InitSource = "http://quinsac:8080/macros/GyoumuMenu.psv";
</script>
```

(3) onLoad の function 定義

HTML の<BODY>タグの onLoad 属性に、FE-COM が常駐する IE 画面が自動的に閉じないように function を定義します。この機能は IE の機能を利用しているため、IE の画面遷移を含む現在の画面をアンロードする全ての処理において、確認ダイアログを表示することができます。

定義例をリスト 5-7に示します。

画面アンロード時に実行される処理は"BeforeUnloadProc"です。"return"時に返す文字列がダイアログに表示されます。

"SetBeforeUnloadHandler"では window オブジェクトの onbeforeunload プロパティに function "BeforeUnloadProc"を設定しています。

<BODY>タグの onLoad 属性に"SetBeforeUnloadHandler"を指定することで、HTML 画面表示時に SetBeforeUnloadHandler が実行されます。

リスト 5-7 onLoad の function 記述例

```
function BeforeUnloadProc() {
   return "現在  PreSerV WebTcl 実行中です。データを保存してから終了してください。";
}
function SetBeforeUnloadHandler() {
   window.onbeforeunload = BeforeUnloadProc;
}
…
<BODY bgcolor="#cceeff" onLoad="SetBeforeUnloadHandler()">
…
```

6 サーバ AP(動的コンテンツ)の設計

本章は、APサーバ上のJavaアプリケーションとして配置する、「動的[注1]コンテンツ」の設計に関して記述します。

注１） “動的”とは、AP サーバ上のアプリケーションは WebTcl からの e_send によるリクエストを”POST”メソッドとして受け取り、リクエストに合ったレスポンスを動的に生成して返すことを示しています。

動的コンテンツは WebTcl で説明の分類上つけた名称です。

動的コンテンツと定義する範囲を図 6-1に示します。

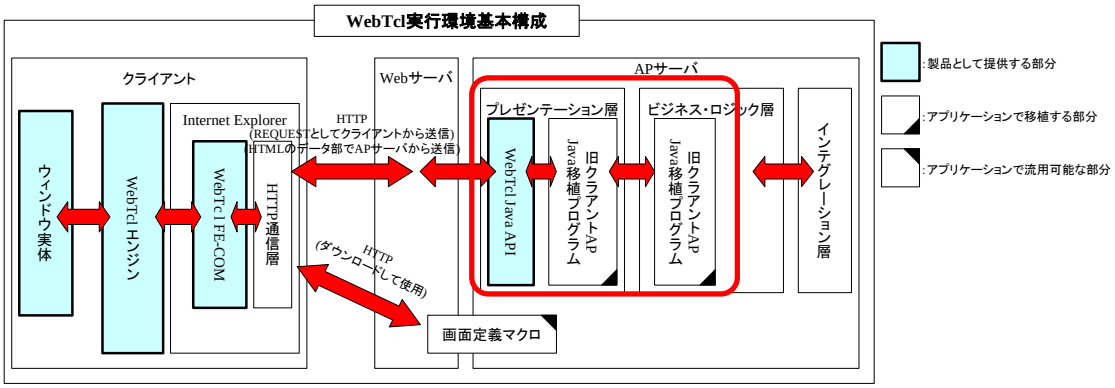


図 6-1 動的コンテンツの範囲

6.1 プレゼンテーション→ビジネスロジック部分

本節では Servlet コンテナ上で Java のクラスとしてサーブレットを構築した時の実装を例として記述します。

フレームワークを使用する場合は、フレームワークの S/W 構造に読み替えて適用してください。

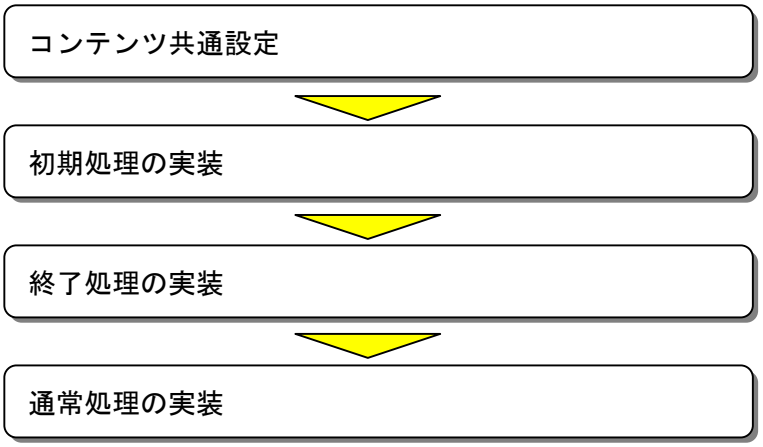
動的コンテンツの処理パターン分類を表 6-1に示します。

この３パターンでほとんどのアプリケーションを作成可能です。

表 6-1 WebTcl の動的コンテンツ分類一覧

処理パターン	説明
初期処理	e_connect リクエスト時の初期処理を示します。
通常処理	e_send リクエスト時の処理を示します。 一般に業務処理はこの処理で行われ、WebTcl クライアントにレスポンスを返します。
終了処理	e_close リクエスト時の終了処理を示します。

画面設計ではこれらのコンテンツの設計を以下の項目に分けて説明します。



TOPICS 終了処理の実装を先に行うのは、先に e_connect、e_close を実装し、途中で動作確認ができるようにするためです。

6.1.1 共通設定

WebTcl ではコンテンツには、以下の共通設定が必須です。

(1) フィルタークラスの設定

AP サーバへのリクエスト時に、セッション管理上必ず WebTcl フィルタークラスを通して前処理を行う必要があります。

フィルタークラス設定はWarファイル中のWEB-INF/web.xml中にリスト 6-1の設定を行うことで可能です。

リスト中の①の設定は、フィルタークラスに WTclFilter という名称を定義しています。

リスト中の②の設定はURLパターンが"/servlet/"から始まるコンテンツ全てにWTclFilterを適用する[注1)]ことを定義しています。

注 1）運用時はアプリケーションのコンテンツ名で絞り込んでください。

フィルターとアプリケーションが同一の WebTcl Java API の Jar ファイルを参照するように、PsvWTclClient.jar ファイルは AP サーバの共通ライブラリのディレクトリに配置してください。

リスト 6-1 WEB-INF/web.xml の設定例

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
      version="2.4">
   <filter>                                                    ①
      <filter-name>WTclFilter</filter-name>
      <filter-class>jp.co.melco.preservcs.PsvWTclFilter</filter-class>
   </filter>
   <filter-mapping>                                            ②
      <filter-name>WTclFilter</filter-name>
      <url-pattern>/servlet/*</url-pattern>
   </filter-mapping>
   <distributable/>
</web-app>
```

(2) アプリケーションのデプロイ

使用する AP サーバにより形式(War/Ear)が異なりますが、最終的に作成したアプリケーションをアーカイブにして、デプロイする必要があります。

「WebTcl 起動サンプルインストール手順書」は AP サーバの設定手順例をドキュメントとして提供しています。

「スタートメニュー」→「PreSerV WebTcl Basic Media」→「追加ﾄﾞｷｭﾒﾝﾄﾃﾞｨﾚｸﾄﾘを開く」から追加ドキュメントのディレクトリを開き、「WebTcl 起動サンプルインストール手順書」を参考に、システムで採用する AP サーバのマニュアルの手順も参考にして、デプロイを行ってください。

6.1.2 初期処理の実装

(1) PSV マクロ側の実装

PSV マクロ側は、以下のシーケンスでアプリケーションとの接続を確立します。

リスト 6-2 e_connect 発行例

```
#接続
proc samp_connect {samp} {
global DEMO
        if { $DEMO(sess_id) == "-1" } {
                set DEMO(sess_id) [eval "e_connect quinsac on 20005" ]
        }
}
```

リスト 6-2ではC/S版との互換性のため、接続先ホスト名“quinsac”や、接続ポート番号”20005”の指定はそのまま残しています。

接続先ホストの設定は無効となり、FE-COM オブジェクトの InitActionURL プロパティで指定されるホストと接続します。

接続ポート番号は、仮想ポート番号として WebTcl のセッションの特定のキーの１つとして参照されます。

WebTcl では、e_connect 時は WebTcl エンジンから FE-COM へ内部コマンド発行としてプロセス間通信を行います。

FE-COM からは、e_connect の内部コマンドを、InitActionURL プロパティで指定されるアクションへの POST リクエストとして加工して、送信します。

FE-COMでは、表 6-2に示すようにe_connectの通信内容をHTTPプロトコル形式に加工します。

表 6-2 e_connect 時のリクエスト内容

定義部分	情報名	説明	HTTP 標準か？
ヘッダ	Content-Type	送信するコンテンツの型を指定します。 WebTcl のリクエストでは " application/psv-wtcl-request"固定です。	標準
	command	送信する WebTcl の処理を指定します。 e_connect 時は "connect-request"固定です。	WebTcl 拡張
	session_id	本来は WebTcl セッション ID を格納します。 e_connect 時は未取得を表す"-1"を指定し、 WebTcl フィルタークラスの採番のトリガーとなります。	WebTcl 拡張
	process_id	IE のプロセス ID を指定します。	WebTcl 拡張
	port	e_connect で指定する TCP/IP のポート番号を指定します。現在は仮想ポート番号として、WebTcl の通信の連続性を特定するキーの１つに使用されています。	WebTcl 拡張
	client_id	クライアント PC の ID です。	WebTcl 拡張
ボディ	ー	ー	ー

(2) AP サーバ側の実装

AP サーバ側は、サーブレット用のプロジェクトを作成して、javax.servlet.http.HttpServlet を継承したクラス InitAction を、新規クラスとして追加してください。このクラスを AP サーバ上で動作させた時に参照する URL が FE-COM.InitActionURL です。

InitActionクラスにはリスト 6-3に示すdoPostメソッドを実装してください。

リスト 6-3 e_connect 時の AP サーバ側の実装例

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    int sessionID = -1;

    /*** 本来インプット・アクションで行う処理 ***/
    // WebTcl セッション管理クラスのインスタンス取得
    PsvWTclClient wtclInst = PsvWTclClient.getInstance();
    // ヘッダー情報の取得
    PsvWTclHeaderInf headInf = wtclInst.getHeaderInformation(request);

    // 対象クライアントとの WebTcl セッション ID の採番
    sessionID = wtclInst.getSessionID(headInf);                                    ①

    /*** ここはビジネス・ロジックで何か初期処理を実装してください ***/
    /*****/                                                                        ②

    /*** 本来アウトプット・アクションで行う処理 ***/
    // アクション URI の登録
    wtclInst.registerActionURI(sessionID, "Start", "/PSVJSFront/servlet/StartAction");
    …
    wtclInst.registerActionURI(sessionID, "", "/PSVJSFront/servlet/DefaultAction");  ③

    // レスポンスの作成依頼
    wtclInst.createResponseData(sessionID, response,
                    (Object)null, (short)PsvWTclClient.CREATE_CONNECT_RESPONSE);   ④
        /*****/
}
```

以下にリスト 6-3の処理を記述します。

① WebTcl セッション ID の取得

WebTcl セッション ID を取得するには、WebTcl セッション管理クラスのインスタンスと、リクエストのヘッダ情報を格納したヘッダ情報格納クラスのインスタンスが必要です。

WebTcl セッション管理クラスのインスタンスは PsvWtclClient#getInstance()メソッドで常に取得可能です。

取得した WebTcl セッション管理クラスから getHeaderInforamtion()メソッドを実行すると、ヘッダ情報を解析してヘッダ情報格納クラスに格納して返します。

さらに getSessionID()メソッドをコールすると、e_connect に対応するユニークな WebTcl セッション ID を採番して返します。

以後の API ではこの WebTcl セッション ID を指定することで、WebTcl の通信を特定します。

② ビジネスロジック部分

アプリケーションで DB 接続の準備など、初期処理が必要な時は、この部分にアプリケーションの処理を追加してください。

本ガイドでは DB 接続の処理は省略しますので、この部分の処理は実装していません。

③ アクション URI の登録

アクション URI は、“通常処理”のリクエストに対応した処理を実行するために登録します。キーワードでリクエスト時のサーブレット URL の一部を切り替えが可能です。

URLは、WebTclセッション管理クラスのregisterActionURI()メソッドをコールし、アクション・マップ※を設定することで登録します。

※ アクション・マップとは、アクション・キーワードとアクション URI の対応をマッピングした情報です。

これは、C/S 版の PSV_AddCallback() API で定義していたイベントハンドラに相当します。

アクション・マップは、5.1－(4)－②で作成した「イベント一覧」から設定するキーワードとサーブレット URL のリストになります。

④ レスポンスの作成依頼

WebTcl セッション管理クラスの createResponseData()メソッドを PsvWTclClient.CREATE_CONNECT_RESPONSE モードでコールします。

これにより③で登録したアクション・マップを HttpServletResponse に書き込みます。

アクション･マップはこのレスポンスを受信する FE-COM 側でメモリ上に展開され、以後の e_send 時やエラー発生時の通信時に参照されます。

6.1.3 終了処理の実装

(1) PSV マクロ側の実装

PSV マクロ側は、以下のシーケンスでアプリケーションとの接続を終了します。

リスト 6-4 e_close 発行例

```
#終了処理
proc samp_end {} {
global DEMO
        if { $DEMO(sess_id) != "-1" } {
                e_close -s $DEMO(sess_id)
                exit
        } else {
                exit
        }
```

リスト 6-4ではWebTclセッションIDが存在する時にe_closeを発行しています。WebTclでは、e_close 時はWebTclエンジンからFE-COMへ内部コマンド発行としてプロセス間通信を行います。

FE-COM からは、e_close の内部コマンドを、EndActionURL プロパティで指定されるアクションへの POST リクエストとして加工して、送信します。

FE-COMでは、表 6-3に示すようにe_closeの通信内容をHTTPプロトコル形式に加工します。

表 6-3 e_connect 時のリクエスト内容

定義部分	情報名	説明	HTTP 標準か？
ヘッダ	Content-Type	送信するコンテンツの型を指定します。 WebTcl のリクエストでは ”application/psv-wtcl-request”固定です。	標準
	command	送信する WebTcl の処理を指定します。 e_close 時は“close-request”固定です。	WebTcl 拡張
	session_id	WebTcl セッション ID を格納します。	WebTcl 拡張
	process_id	IE のプロセス ID を指定します。	WebTcl 拡張
	port	e_connect で指定する TCP/IP のポート番号を指定します。現在は仮想ポート番号として、WebTcl の通信の連続性を特定するキーの１つに使用されています。	WebTcl 拡張
	client_id	クライアント PC の ID です。	WebTcl 拡張
ボディ	―	―	―

(2) AP サーバ側の実装

AP サーバ側は、サーブレット用のプロジェクトを作成して、javax.servlet.http.HttpServletを継承したクラスEndActionを、新規クラスとして追加してください。このクラスをAPサーバ上で動作させた際に参照するURLがFE-COM.EndActionURLになります。EndActionクラスには、リスト 6-5に示すdoPostメソッドを実装してください。

リスト 6-5 e_close 時の AP サーバ側の実装例

```
protected void doPost(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
    int sessionID = -1;
    /*** 本来インプット・アクションで行う処理 ***/
    // WebTcl セッション管理クラスのインスタンス取得
    PsvWTclClient wtclInst = PsvWTclClient.getInstance();
    // ヘッダー情報の取得
    PsvWTclHeaderInf headInf = wtclInst.getHeaderInformation(request);
    // WebTcl セッション ID の取得(InitAction 時と違う API なので注意)
    sessionID = headInf.getSessionID();
    /*****/                                                          ①
    /*** ここはビジネス・ロジックで何か終了処理を実装してください ***/
    String EndCmd = ";";
    /*****/                                                          ②

    /*** 本来アウトプット・アクションで行う処理 ***/
    // レスポンスの作成依頼
    wtclInst.createResponseData(sessionID, response, EndCmd,
            (short) PsvWTclClient.CREATE_CLOSE_RESPONSE);
    /*****/                                                          ③
    }
```

注）e_connect時に偽接続モードでレスポンス作成した場合、直後にFE-COM#EndActionURLを実行して接続終了を行うため、リスト 6-5の処理が実行されます。偽接続モード時に不要な処理を実行しないように、WebTcl Java APIのPsvWTclClient#getPseudoStatus()メソッドを使用して、偽接続モードを検出してください。

以下にリスト 6-5の処理を記述します。

① WebTcl セッション ID の取得

PsvWtclClient#getInstance()メソッドをコールして、WebTcl セッション管理クラスのインスタンスを取得します。

取得した WebTcl セッション管理クラスから getHeaderInforamtion()メソッドを実行すると、ヘッダ情報を解析してヘッダ情報格納クラスに格納して返します。さらに getSessionID()メソッドをコールすると、ヘッダ情報から WebTcl セッション ID を取得します。

② ビジネスロジック部分

アプリケーションで DB 接続の終了など、終了処理が必要な時は、この部分にアプリケーションの処理を追加してください。サンプルでは DB 接続の処理は省略します。

③ レスポンスの作成依頼

WebTcl セッション管理クラスの createResponseData()メソッドを PsvWTclClient.CREATE_CLOSE_RESPONSE モードでコールします。

これにより、WebTcl セッション ID に関連した情報を開放し、終了用のダミーのレスポンスを返します。このダミーのレスポンスを受信後 e_close コマンドの処理が完全に終了します。

6.1.4 通常処理の実装

(1) PSV マクロ側の実装

PSV マクロ側は、以下のシーケンスでアプリケーション側に処理のリクエストを行います。

以下「設備の属性検索」時の e_send を例としています。

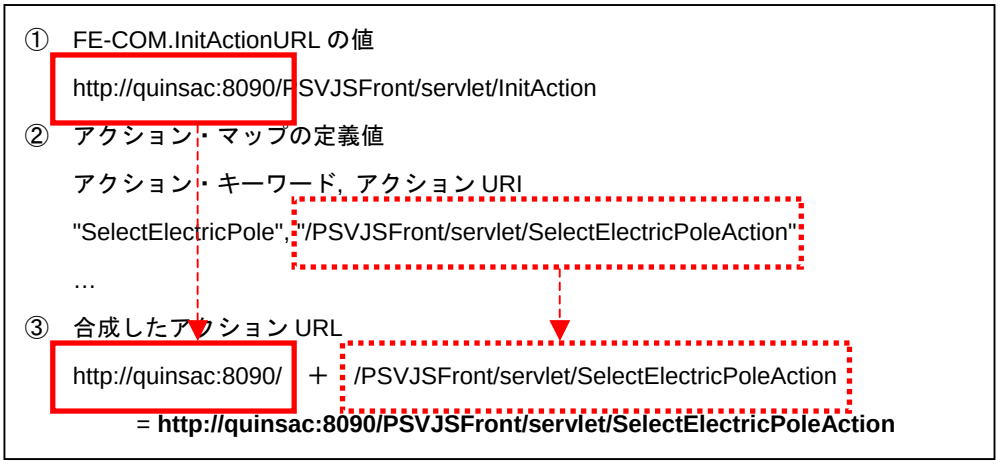
リスト 6-6 e_send 発行例

```
…
        #ユーザ定義データ設定
        set map(usrkey) [g_get_usrdata $map(objz) $map(usr_data)]

        #属性情報取得のためにキー送信
        e_send "SelectElectricPole $map(usrkey)"
…
```

リスト 6-6では$map(objz)で特定される設備図形(電柱を想定)のユーザデータを取得し、これをアクション・キーワード“SelectElectricPole”に続けて送信しています（ワード間はスペースで区切られています。）。
WebTcl セッション ID は省略しているので、初回 e_connect 時の WebTcl セッション ID が適用されています。
また e_connect 同様に、e_send 時は WebTcl エンジンから FE-COM へ内部コマンド発行としてプロセス間通信を行います。
FE-COMからは、e_sendの内部コマンドを、InitActionURLプロパティのホスト名までのURLと、e_connect時に返信されたアクション・マップのアクションURIと結合し、e_send用のアクションURLを合成します（リスト 6-7参照）。
注)FE-COM.ActionCodeBase が定義されている場合は、FE-COM.ActionCodeBase で定義された URL と e_connect 時に返信されたアクション・マップのアクション URI と結合し、e_send 用のアクション URL を合成します。

リスト 6-7 e_send 時のアクション URL の合成



通信内容はこのアクション URL への POST リクエストとして加工して､送信します。
FE-COMでは、表 6-4に示すようにe_sendの通信内容をHTTPプロトコル形式に加工します。e_connect時の情報もFE-COMが補完して送信します。

表 6-4 e_send 時のリクエスト内容

定義部分	情報名	説明	HTTP 標準か？
ヘッダ	Content-Type	送信するコンテンツの型を指定します。 WebTcl のリクエストでは " application/psv-wtcl-request"固定です。	標準
	command	送信する WebTcl の処理を指定します。 e_send 時は"send-request"固定です。	WebTcl 拡張
	session_id	WebTcl セッション ID を格納します。WebTcl フィルタークラスはこの WebTcl セッション ID が e_connect 時に採番されたものかをチェックし、不正な場合は自動的に 400 のレスポンスコードでエラーを返します。	WebTcl 拡張
	process_id	IE のプロセス ID を指定します。	WebTcl 拡張
	port	e_connect 時に指定した TCP/IP のポート番号を指定します。現在は仮想ポート番号として、WebTcl の通信の連続性を特定するキーの１つに使用されています。	WebTcl 拡張
	client_id	クライアント PC の ID です。	WebTcl 拡張
ボディ	e_send で送信した文字列	e_send で送信した文字列が格納されています。	標準

(2) AP サーバ側の実装

AP サーバ側は、サーブレット用のプロジェクトを作成して、javax.servlet.http.HttpServletを継承したクラスSelectElectricPoleActionを、新規クラスとして追加してください。このクラスをAPサーバ上で動作させた際に参照するURLがリスト 6-7の③で合成するアクションURLになります。
SelectElectricPoleActionクラスにはリスト 6-8に示すdoPostメソッドを実装してください。

リスト 6-8 e_send 時の AP サーバ側の実装例

```
protected void doPost(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
        int sessionID = -1;

        /*** 本来インプット・アクションで行う処理 ***/
        // WebTcl セッション管理クラスのインスタンス取得
        PsvWTclClient wtclInst = PsvWTclClient.getInstance();
        // ヘッダー情報の取得                                                    ①
        PsvWTclHeaderInf headInf = wtclInst.getHeaderInformation(request);

        // ここはフィルターで実行するセッション ID の取得(InitAction 時は必要)
        sessionID = headInf.getSessionID();

        // 内容の解析。書式が通常の POST 形式なら簡易な API で取れる
        BufferedReader reader = request.getReader();
        String loopStr = null; String poi = null; String key = null; String body = null;
        int index = 0;
        loopStr = reader.readLine();                                          ②
        while (loopStr != null) {
            poi = loopStr;
            loopStr = reader.readLine();
            index = poi.indexOf(' ', 0);
            key = poi.substring(0, index);
            body = poi.substring(index + 1, poi.length());
        }

        /*** ここはビジネス・ロジックで何か検索処理を実装してください ***/
        String EPProcStr1 =
        "makeEPDialog EP194529 ４５-２９ -3097462.020 -3071278.5781 101 F50 be";  ③
        ….
        String selectCmd = EPProcStr1;
        /*** 本来アウトプット・アクションで行う処理 ***/
        wtclInst.createResponseData(sessionID, response, selectCmd,           ④
                (short) PsvWTclClient.CREATE_SEND_RESPONSE);
}
```

以下にリスト 6-8の処理を説明します。

① WebTcl セッション ID の取得

e_connect 時と同様に WebTcl セッション管理クラスのインスタンス取得と、ヘッダ情報の取得を行います。

e_send 時は、WebTcl セッション ID はヘッダ部の情報に格納されているので、ヘッダ情報格納クラスの getSessionID()メソッドを実行すると WebTcl セッション ID が取得可能です。

② 受信内容解析部分

HTTP リクエストのボディ部を直接読み込み文字列を分割しています。送信する内容がクエリー文字列形式で定義されている場合は、HttpServletRequest#getParameter()で取得可能です。今回は C/S 版との互換性のため、単純な文字列の送信時の受け取り方を実装しています。

想定した「電柱の設備属性検索」では、ここで電柱の設備キーを取り出したものとします。

③ ビジネスロジック部分

C/S 版のクライアント AP で DB 検索処理などがある場合は、この部分にアプリケーションの実装を追加してください。サンプルでは DB 検索などの処理は省略します。

④ レスポンスの作成依頼

WebTcl セッション管理クラスの createResponseData() メソッドを PsvWTclClient.CREATE_SEND_RESPONSE モードでコールします。

これにより selectCmd の内容をクライアント PC に送信し WebTcl エンジンで実行します。

6.1.5 アプリケーションによる接続拒否の実装

本項では、アプリケーション側で認証による接続拒否の実装を、PreSerV WebTcl Java API の「偽接続」を利用して実現する方式を説明します。

(1) PSV マクロ側の実装

PSV マクロ側の実装は、基本的に通常の e_connect と同じですが、戻り値に偽接続を表す-2 もエラーとして判断する実装に変更してください。

リスト 6-9 偽接続時の e_connect 例

```
#接続
proc samp_connect {samp} {
global DEMO
        if { $DEMO(sess_id) == "-1" } {
                set DEMO(sess_id) [eval "e_connect quinsac" ]
        }
        if { $DEMO(sess_id) > 0 } {
                #エラーで無い場合の処理がこの部分に実装される
                ...
        }
}
```

リスト 6-9ではe_connectの戻り値が、通常のエラー値 0、-1 に加え偽接続の-2 が返ってきても良いように、正の値の場合接続成功時の処理を実装する条件文を記述しています。

POSTリクエストの内容や動作までは、e_connectと同じです。6.1.2項を参照してください。

(2) AP サーバ側の実装その 1

APサーバ側は、6.1.2項のリスト 6-3の④の実装に関して、リスト 6-10の実装に置き換えます。

リスト 6-10 偽接続時の AP サーバ側の実装例その 1

```
String dispMsg = "w_top errWin -x 200 -y 100 -t エラー -w 500 -h 150 -b &ECE9D8;"
 ...
dispMsg += "w_button btn1 @errWin -x 210 -y 120 -w 80 -h 25 -b &ECE9D8 -l 閉じる"
dispMsg += " -c {exit}";
wtcllnst.createResponseData(sessionID, response, dispMsg,
                        PsvWTclClient.CREATE_PSEUDO_CONNECT_RESPONSE);
```

④

注）偽接続は仮の接続を保持してコマンドを実行する機能のため、偽接続で実行するコマンドに通信に関する処理(e_send など)は使用しないでください。

以下にリスト 6-10の処理を記述します。

○ レスポンスの作成依頼(リスト 6-10の④の処理)

WebTcl セッション管理クラスの createResponseData() メソッドを PsvWTclClient.CREATE_PSEUDO_CONNECT_RESPONSE モードでコールします。

これにより dsipMsg で定義する、エラーダイアログ表示の処理を実行し、接続状態を偽接続とします。

(3) AP サーバ側の実装その２

偽接続はエラーダイアログ表示などコマンドの実行を行った直後に、仮の接続状態を解消します。

この場合、6.1.3項のe_close時のAPサーバ側の実装で、アプリケーションの後処理が実行されないように、偽接続を識別する必要があります。

この実装は、6.1.3項のリスト 6-5の②の実装に関して、リスト 6-11の実装を参考に偽接続の識別処理(WebTcl Java APIのPsvWTclClient#getPseudoStatus()メソッド)を追加してください。

リスト 6-11 偽接続時の AP サーバ側の実装例その２

```
//偽接続状態の確認
Logger logInst = Logger.getLogger(this.getClass());                    ②
if (wtclInst.getPseudoStatus(sessionID) == 1) {
    logInst.debug("偽接続状態の e_close を検出しました。<" + sessionID + ">");
} else if (wtclInst.getPseudoStatus(sessionID) == -1) {
    logInst.debug("不正なセッション ID で偽接続の状態取得を行いました。<"
            + sessionID + ">");
} else {
   /*** ここはビジネス・ロジックで何か終了処理を実装してください ***/
}
```

以下にリスト 6-11の処理を記述します。

○　ビジネス・ロジックの修正例(リスト 6-11の②の処理)

偽接続状態は PsvWTclClient#getPseudoStatus()メソッドの戻り値で判断できます。

デフォルトは 0 で、これは通常の接続状態を示します。戻り値 1 の場合は、偽接続の接続状態を示します。また戻り値-1 は引数の WebTcl セッション ID が不正な場合です。

提示した例では、偽接続状態時にLoggerがdebugのログを出力します(リスト 6-12 参照)。

リスト 6-12 偽接続時のログ出力例

```
...
09:56:29,728 DEBUG - [0294160454],IN：getAppAttribute()コール<294160454><CMD>
09:56:29,728 DEBUG - [0294160454],IN：isPseudoConnect()コール<294160454>
09:56:29,728 DEBUG - [0294160454],偽接続状態の e_close を検出しました。<294160454>
...
```

6.1.6 データ取得処理の実装

本項では、ストックしたデータをe_recvによって取り出す処理の実装を説明します。e_recvの利用は、下記の表 6-5のパターンで可能です。

表 6-5 e_recv を利用可能な処理パターン

psv マクロ側の処理 e_send と e_recv のコールタイミング	AP サーバ側の処理 e_send のアクションでレスポンス作成する際のモード	FE-COM. AskRequestURL で定義した e_recv のアクションの実装
同一プロシジャー内でコール	NOP_STOCK_RESPONSE	必要
	CREATE_SEND_RESPONSE	必要
	CREATE_SEND_DATA_RESPONSE	不要
別のプロシジャーでコール	NOP_STOCK_RESPONSE(*推奨)	必要
	CREATE_SEND_RESPONSE	必要
	CREATE_SEND_DATA_RESPONSE	不要

注）e_recv のアクションの実装が不要なパターンのものでも、FE-COM. AskRequestURL を定義する必要があります。

以下の(1)～(4)では、別のプロシジャーで e_send と e_recv をコールし、レスポンス作成を NOP_STOCK_RESPONSE モードで行う推奨パターンに付いて説明します。

(1) PSV マクロ側の実装その１

PSV マクロで e_send を発行し、処理のリクエストを行います。

リスト 6-13 e_send 発行例

```
proc stock_data {} {
        global SAMP
        if { $SAMP(sess_id) != "-1" } {
                e_send -s $SAMP(sess_id) "StockData samp1"
        } else {
                w_text_str $SAMP(txt1) "接続できていません。"
        }
}
```

e_sendで呼ばれるアクションURLの合成については、リスト 6-7を参照してください。

(2) AP サーバ側の処理その 1

AP サーバ側では、javax.servlet.http.HttpServlet を継承したクラスを、新規クラスとして追加してください。

リスト 6-14　データをストックする AP サーバ側の実装例

```
protected void doPost(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException {
    int sessionID = -1;
    // WebTcl セッション管理クラスのインスタンス取得
    PsvWTclClient wtclInst = PsvWTclClient.getInstance();
    // ヘッダー情報の取得
    PsvWTclHeaderInf headInf =  wtclInst.getHeaderInformation(request);
    // WebTcl セッション ID の取得
    sessionID = headInf.getSessionID();

    ...

    // ストックするデータ
    String str = "A";
    // データをストック
    wtclInst.stockResponseData(sessionID, str);

    ...

    // レスポンス作成依頼
    wtclInst.createResponseData(sessionID, response, "",
          PsvWTclClient.NOP_STOCK_SEND_RESPONSE);
}
```

① ②

以下にリスト 6-14の処理を記述します。

① データのストック処理

セッション ID とストックするデータを指定し、stockResponseData()メソッドをコールします。この例では”A”をストックしています。

② レスポンスの作成依頼

WebTcl セッション管理クラスの createResponseData()メソッドを PsvWTclClient.NOP_STOCK_SEND_RESPONSE モードでコールします。

(3) PSV マクロ側の実装その２

e_send を発行したマクロとは別のマクロで e_recv を発行して、処理のリクエストを発行します。

リスト 6-15　e_recv 発行例

```
proc data_recv {} {
        global SAMP
        set SAMP(res) [e_recv $SAMP(sess_id) 5]
  ...
}
```

リスト 6-15の例では、変数SAMP(res)にe_recvで取得したデータをセットします。e_recvで呼ばれるアクションURLは、FE-COMのAskRequestURLで指定されるURLです。

(4) AP サーバ側の処理その２

AP サーバ側では、javax.servlet.http.HttpServlet を継承したクラスを、新規クラスとして追加してください。このクラスを AP サーバ上で動作させた際に参照する URL が FE-COM. AskRequestURL になります。

リスト 6-16 e_recv の実装例

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    int sessionID = -1;
    /*** 本来インプット・アクションでやる処理 ***/
    // WebTcl セッション管理クラスのインスタンス取得
    PsvWTclClient wtclInst = PsvWTclClient.getInstance();
    // ヘッダー情報の取得
    PsvWTclHeaderInf headInf = wtclInst.getHeaderInformation(request);
    // WebTcl セッション ID の取得
    sessionID = headInf.getSessionID();
    // レスポンスの作成依頼
    wtclInst.createResponseData
        (sessionID, response, null, PsvWTclClient.CREATE_SEND_DATA_RESPONSE);
}
```

WebTcl セッション管理クラスの createResponseData() メソッドを PsvWTclClient. CREATE_SEND_DATA_RESPONSE モードでコールします。

6.2 その他ビジネスロジック→インテグレーション部分に関して

この層における実装に関しては WebTcl では、特に制約はありません。

ただし、以下の点に注意してアプリケーション構築を行ってください。

① 単純なマクロ生成部は共通化する

プレゼンテーション層のマクロを生成するロジックは、単純な処理の繰り返しとなりやすいため、できるだけ共通化し構造化することをお奨めします。

② 文字列の処理が多いため、StringBuffer の使用を考慮する

文字列の結合などが多い場合は、String 型より StringBuffer 型の処理が効率的です。

7 製作

製作は、主に以下の実装作業があります。

① イベント処理実装

５章で作成した画面表示用の PSV マクロに対して、W マクロを利用して、画面部品間のリンク・制御処理を実装します。W マクロの実装は C/S 版と同じです。

『PreSerV V5 基本パック』→「オンラインマニュアル」→「PSV 製品概要」→「2.2 節 ウィンドウの操作」

② 地図表示・制御処理実装

５章で作成した画面表示用の PSV マクロに対して、G マクロを利用して、地図表示処理の実装と、各種制御処理を実装します。G マクロの実装は C/S 版と同じです。

『PreSerV V5 基本パック』→「オンラインマニュアル」→「PSV 製品概要」→「2.3 節 グラフィックスの操作」

8 結合試験

結合試験では、以下の機能を使用して、効率よく実施可能です。

① FE-COM オブジェクトの DebugPolicy 設定

FE-COM ではエラーを逐次ウィンドウで表示するデバッグモードと、ファイル出力するデバッグモードがあります。

結合試験時にエラーが発生した場合は、DebugPolicy を 1~3 に設定することで、エラー個所を効率よく特定できます。

② デバッグコンソールの組み込み

デバッグツール・プラグインという形態で、マクロコマンドの実行が可能なデバッグコンソールを提供しています（図 8-1参照）。

ログなどで原因特定が不可能な場合、事前にデバッグコンソールを実行し、タスクトレイに常駐させておくことで、エラーに関連する変数の内容などを取得してより詳細な情報の取得が可能です。

REF. **「WebTcl 概要書」→「エラーメッセージ及びデバッグツール」→「デバッグツールによるエラー情報の取得」**

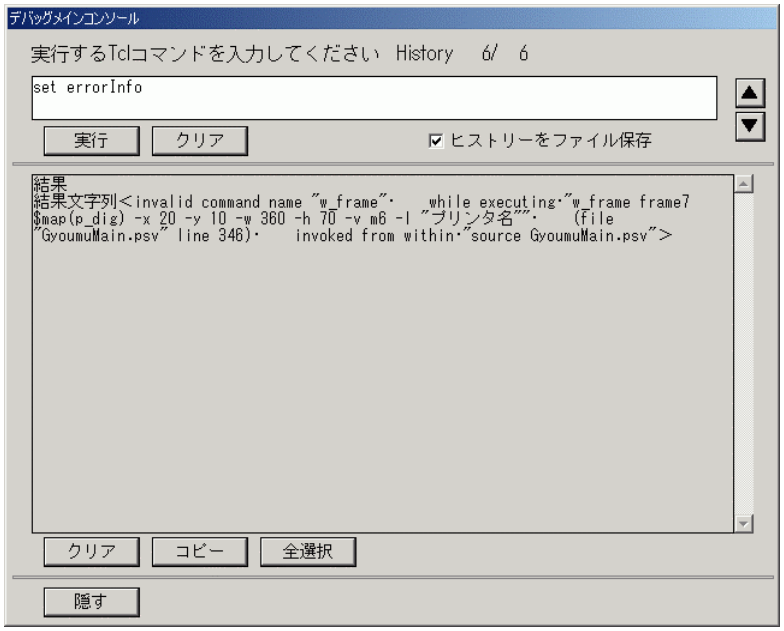


図 8-1 デバックコンソール画面

9 ランタイム環境構築

9.1 標準の配布メディアを使用する場合

ランタイム環境は配布メディアを標準で使う場合は
「スタートメニュー」→「PreSerV WebTcl Basic Media」→「配布ﾒﾃﾞｨｱﾃﾞｨﾚｸﾄﾘを開く」を実行して、エクスプローラで表示されるディレクトリ下のファイルを使用します。表 9-1に配布メディアの種別を示します。

表 9-1 配布メディア種別

ファイル名	使用目的	説明
WebTclInst.exe	一般の運用時	一般の運用時に新規追加、更新のみ操作可能なインストールプログラムです。 削除などの不要な動作を選択できないようになっています。
WebTclInstFull.exe	メンテナンス時	新規追加、更新、削除、構成の変更など、インストーラのフル機能が使用可能です。 メンテナンス時にアンインストール等に使用してください。

9.2 配布メディアをカスタマイズする場合

ランタイム環境をカスタマイズして配布する場合、以下の手順を参考に、基本モジュールを配布するインストール処理を実装し、独自の配布用カスタマイズモジュールに組み込んでください。

① 「PreSerV WebTcl Basic Media」 CD から基本モジュールのファイルを取り出す。
InstallImg¥BasicModuleContents¥の下にモジュールを配置しています。

② 独自の配布の仕掛けを実装する
一般的にインストーラや CAB ファイル形式で配布メディアを作成しますが、以下の処理を実装してください。

(a) 配布メディアには電子署名が必要
一般的なIEの設定では電子署名がないEXEやCABファイルはダウンロードできません。PreSerVではベリサイン Class3 の電子署名で、配布メディアの作成元を証明しています（図 9-1参照）。実現方法はシステムにより異なると思いますが、インストール作業なしで、配布する場合IEにデフォルトでインストールされているCAに対応した電子署名を取得してください。

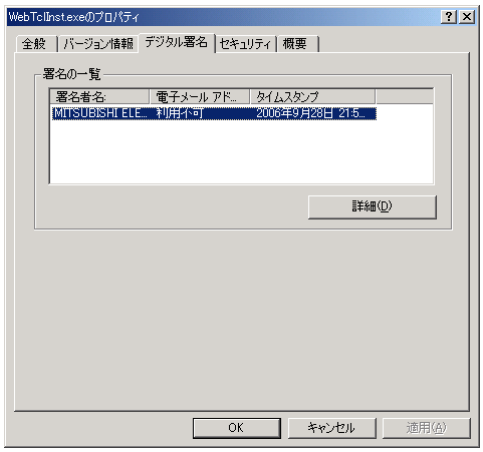


図 9-1 WebTclInst.exe の電子署名

(b) WebTclServer.exe は System32 ディレクトリ下に配置

(c) その他のモジュールは環境変数 PATH に設定があるディレクトリに配置

(d) WebTclFELib.dll は登録が必要

WebTclFELib.dll は自動登録機能を持つ DLL で、FE-COM オブジェクトの実体です。登録することで HTML 画面に貼り付け可能なオブジェクトとして認識されます。

したがって、regsrv32.exe(VC++付属のツール。Microsoft 社のダウンロードサイトから入手可能)など登録機能をもつモジュールを介して、DLL 自体の自動登録機能を呼び出し実行してください。

10 資料

10.1 V5 版と過去バージョンの仕様差分

V5 版の Tcl 仕様と、過去バージョン(A 版、Ver.3 版、Ver.4 版)の仕様差分を、A 版と Ver.4 版、Ver.4 版と V5 版に分けて記述します。
また V5 版がサポートする A 版互換モードと A 版との動作仕様の差異も記述します。

(1) A 版と Ver.4 版間の仕様差異

A 版から Ver.4 版間は G マクロと W マクロの機能追加を主に実施しています。詳細な修正箇所に関しては V5 版のオンラインマニュアルで、API の更新履歴を参照してください。

表 10-1 Ver.4.x 版までに追加した機能一覧

項番	機能分類	機能名称(マクロ名)	追加バージョン
1	W マクロ	マウスカーソル形状の変更が可能	Ver.3.0.0
2	〃	ウィンドウ部品の３D 陰影表示	Ver.3.0.0
3	〃	リストボックス機能拡張	Ver.3.0.0 Ver.4.5.6
4	〃	テキストボックスの機能拡張	Ver.3.0.0
5	〃	ダブルクリックのイベントマクロが指定可能	Ver.3.0.0 Ver.4.0.0
6	G マクロ	テキストオブジェクトで複数行のテキストサポート	Ver.3.0.0
7	〃	TrueType フォント描画が可能	Ver.4.0.0
8	〃	ビットマップシンボルオブジェクトのサポート	Ver.4.0.0
9	〃	イメージファイルによるラスター地図サポート	Ver.3.0.0 Ver.4.0.0
10	〃	ＧＤＩ印刷 (Windows プリンタへの出力)	Ver.3.0.0
11	〃	クリップボード転送	Ver.3.0.0
12	標準部品	標準部品のオペレーション対象レイヤ制御	Ver.4.5.6
13	E マクロ	時間関連 API の追加	Ver.3.0.0
14	〃	プラグインによる機能追加	Ver.3.0.0[注１]

注１）各プラグインは随時追加、バージョンアップしています。

(2) Ver.4 版と V5 版の仕様差異

V5 版ではTcl以外のI/FとしてVB/IFが追加され、共通機能部分と各言語I/F部分に分けて機能追加しました。表 10-2に共通機能部分とTcl機能部分の機能追加内容を記述します。

表 10-2 V5 版で追加された機能一覧

項番	機能分類	機能名称
1	共通機能	イメージ展開性能の向上
2	〃	イメージ回転時の表示品質改善
3	〃	ビットマップシンボルの白地透過可能
4	〃	カラーシンボル
5	〃	最新 OS への対応
6	〃	インストーラの自動化
7	Tcl 機能	W マクロ追加
8	〃	Tcl インタプリターのバージョンアップ
9	〃	デバッグトレース機能強化
10	〃	A 版との共存
11	〃	複数プロセスが実行可能
12	〃	その他

(3) A 版互換モードと A 版との動作仕様の差異

表 10-3にWebTclのA版互換モードとA版の動作仕様の差異を示します。

表 10-3 A 版互換モードと A 版の動作仕様差異一覧

項番	動作概要	動作仕様の差異
1	w_dialog モーダル時の親ウィンドウの挙動	A 版互換動作モードでは、モーダル時に操作禁止中の親ウィンドウをクリックすると、ウィンドウが点滅します。
2	フォーカス	w_combo_box などで、親ウィンドウがアクティブでなくなった時、A 版ではフォーカスがなくなるが、A 版互換時ではフォーカスが残ります。
3	w_combo_box のテキストボックスの表示位置	A 版では、w_combo_box のテキストボックスの表示位置が１ドット程度左上にずれているが、A 版互換動作モードでは正しい位置に表示されます。
4	w_list_box のスクロールバーの挙動	w_list_box の項目がある場合、A 版では表示範囲内でもスクロールバーが表示され、クリックすると消去されます。A 版互換動作モード時は最初から表示されません。
5	w_slider の背景色	A 版では w_slider の背景色を w_set_value で変更できませんが、A 版互換動作モードでは変更可能です。
6	色 ID256 が使用不可	V4 版以降は 8bit で色 ID を指定するようになったため、A 版互換動作モード時も色 ID256 は使用不可です。
7	太さ２以上の線種が有効	図形オブジェクト描画時に A 版互換動作モード時は WindowsNT 系の OS では線幅２以上の線描画の線種が有効になっています。
8	図形描画イメージのずれ	A 版互換動作モード時は、V4 版以降の WC 系座標の DC 系座標に対する投影処理の精度改善や、描画時に最大描画領域を越える図形描画の描画領域改善が行われているため、１ドット単位で描画イメージが異なる。

項番	動作概要	動作仕様の差異
9	ミラー文字のサポートに関して	文字幅など文字オブジェクトに対するパラメータを負値に指定した場合、A 版ではミラー文字として描画していましたが、A 版互換動作モード時は内部的に TrueType フォント描画をサポートすることから、負値指定は使用できません。
10	縦横比率不一致時の回転文字列描画	グラフィックウィンドウの DC サイズ(ピクセル単位のサイズ)の縦横比率と、WC サイズの縦横比率が異なる場合、文字列回転を行った複数文字の文字オブジェクトを A 版では１列に描画するが、A 版互換動作モード時は１文字ずれて表示します。
11	フィルター制御点の色適用の差異	フィルターを使用した点変更モード時に表示する制御点の色の適用方法が、A 版では今アクティブな点が図形オブジェクトの色でそれ以外の制御点が g_open_filter で指定した色、A 版互換動作モード時はアクティブな点が g_open_filter で指定した色、それ以外の制御点が図形オブジェクトの色となっています。
12	パラメータ指定時の区切り文字の差異	パラメータ指定時のスペースを全角スペースにすると、A 版互換ではエラーになります。
13	矩形範囲指定描画時の図形クリッピング	g_draw_obj で、矩形範囲 DC 指定で描画した場合、A 版は矩形からはみ出た図形も表示しますが、A 版互換動作モード時は矩形内しか表示しません。
14	矩形検索時の円の判定領域	A 版の円の最大描画領域は矩形で判定していますが、A 版互換動作モードでは円に近い形で判定します。
15	文字オブジェクトの線種が有効	A 版互換動作モード時は、文字オブジェクトの線種指定が有効です。
16	スプラインの線タイプの設定が無効	A 版互換動作モード時は、スプラインの線種指定が有効になりません。
17	オプションパラメータ指定無し時	A 版ではメモリ不正アクセスとしてエラー処理を実行しますが、A 版互換動作モードではデフォルト値で正常動作します。
18	w_text_box のモード不正時の動作	A 版ではモードを不正指定した場合、２行のマルチラインテキストとなりますが、A 版互換動作モードではシングルラインテキストとなります。

項番	動作概要	動作仕様の差異
19	ビットマップ指定不正時の動作	w_button ・ w_arm_button ・ w_check_box ・ w_radio_button で、不正なビットマップファイル名を指定すると、A 版では「def」と書かれたビットマップを表示するが、A 版互換動作モードではキャプション文字列のデフォルト表示となります。
20	w_combo_box・w_list_box の幅指定不正時	w_combo_box・w_list_box で不正な幅を指定すると、A 版ではウィンドウが描画されないが、A 版互換動作モード時は 2･3 ドットの幅のウィンドウが表示されます。
21	w_combo_box 高さ指定不正時	w_combo_box・w_list_box で不正な高さを指定すると、A 版ではウィンドウが描画されませんが、A 版互換動作モード時は１文字分の高さのウィンドウが表示されます。
22	w_message_box の親ウィンドウ	A 版では、ダイアログを親ウィンドウに指定するとエラーになりますが、A 版互換動作モードでは指定可能です。
23	w_combo_add で項目設定時の表示位置調整動作	w_combo_box のリストボックスを開いた時に、A 版互換動作モードでは、反転している項目がトップに近くなるようにスクロールしています。A 版では、スクロールしてから開いています。
24	w_text_ins_pos でテキストカーソル位置移動の動作	1 行目の改行位置にテキストカーソルあった場合、w_text_ins_pos の実行後テキストカーソルが 2 行目の先頭に移動することがある。A 版互換では、移動しない。
25	w_message_box のメッセージ省略時の動作	メッセージボックスのメッセージ文字列指定をなしにすると、A 版では NULL となりますが、A 版互換動作モードは、messege と表示されます。
26	w_get_value で w_combo_box の-m オプションの動作	get_value に-m を指定した場合、A 版では、マニュアルにない w_combo_box の値が取得されます。A 版互換動作モードでは取得されません。
27	w_combo_del のアイテム指定不正時の動作	w_combo_del で、アイテム指定なし・存在しないアイテムを指定した場合、A 版では最終項目が削除されますが、A 版互換動作モードでは処理無効となります。

項番	動作概要	動作仕様の差異
28	w_list_del でアイテムの指定不正時の動作	w_list_del で、アイテム指定なし・存在しないアイテムを指定した場合、A 版では最終項目が削除されますが、A 版互換動作モードでは処理無効となります。
29	w_list_sel ・ w_list_desel でアイテム指定不正時の動作	w_list_sel・w_list_desel で、アイテム指定なし・存在しないアイテムを指定した場合、A 版では最終項目が選択・選択解除されますが、A 版互換動作モードでは処理無効となります。
30	図形オブジェクト作成時のモード指定(-m)の動作	A 版では-m オプションで、スプライン指定の１と塗りつぶしの２の論理和である３を指定しても、有効であったが、A 版互換動作モード時ではスプラインの時に同時に塗りつぶしを指定すると線が描画されない。
31	廃棄済みオブジェクトの再廃棄	g_close_context などオブジェクトの廃棄命令を同一のオブジェクトに対して２回コールした場合、A 版ではエラーになりませんが、A 版互換動作モード時はエラーになります。
32	変形時の制御点の動作差異	A 版ではパイの半径の線を伸ばしても半径より描画されませんが、A 版互換動作モード時は伸ばした分も描画されます。
33	テキストの負回転角度指定時の動作	回転角度に負値を指定すると、A 版では描画されませんが、A 版互換動作モードでは絶対値で描画されます。

注) A 版互換動作モードでは、以下の機能項目に対して V5 版の TCL/IF に対して機能制限を加え、A 版と内部処理の動作仕様を調整しています。

① A 版互換モードでは、スレッドを生成しません
スレッドの生成がないため、図形の描画・図形のメモリ開放時に処理性能が劣るケースがあります。 A 版ではシングルスレッドのプログラムであり、A 版互換では互換性維持のためシングルスレッドで動作します。

② 前景色・背景色を指定した塗りつぶしはできません

③ A 版以降追加された TCL/IF のマクロは使用できません
A 版互換動作モード時の動作は V5 版の機能を抑制しているため、A 版以降追加した TCL/IF のマクロコマンドを削除して動作しています。