Web emulator2

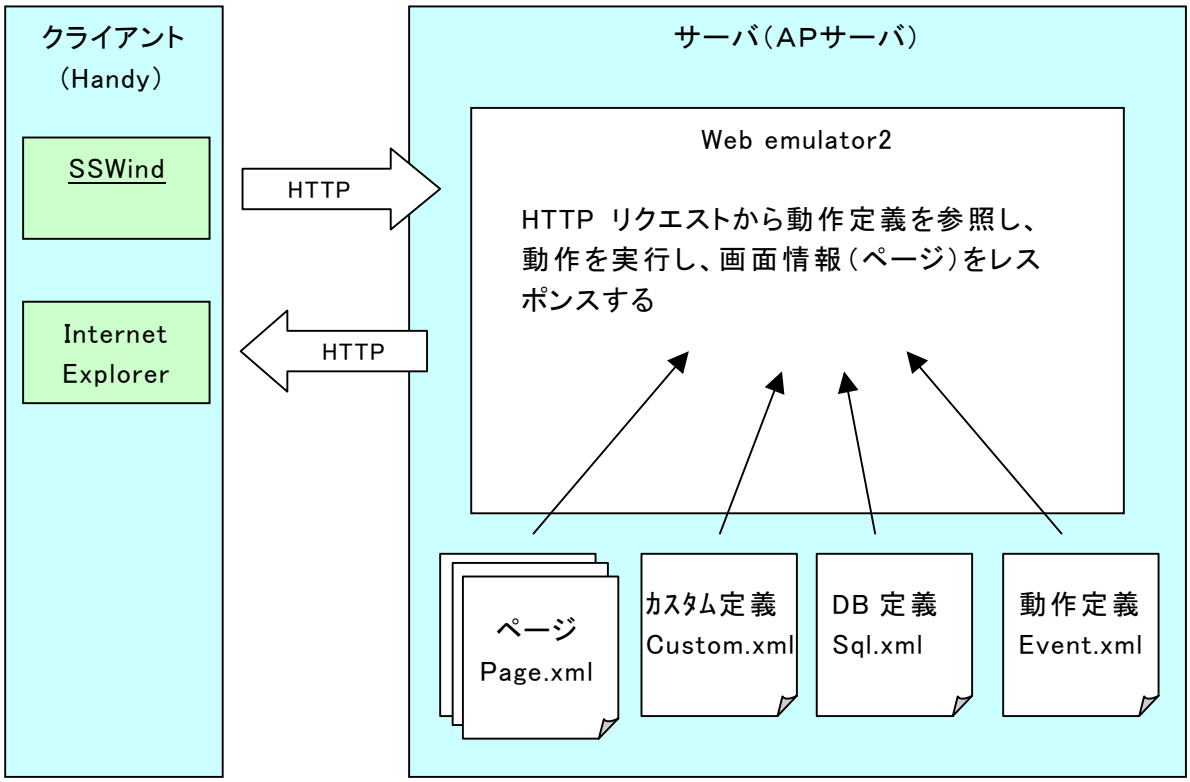
リファレンスマニュアル

2007/5/14 版

STANDARD
Radio Communications

Web emulator2 とは？

Web emulator2 とは、クライアント（ハンディーターミナル）用の画面の構成、画面の動作、データーベースへの問い合わせ等の設定 XML ファイルを記述することで、高度なプログラミング技術が必要なくアプリケーションを作成可能な、RF総合開発パッケージです。



※SSWind とは、弊社のハンディ用リッチクライアントです。

動作環境

OS	Windows2000 以降など、その他 AP サーバが動作する OS
Java	Sun J2SDK Standard Edition 1.4.2 以降 Sun J2SDK Enterprise Edition 1.3.1 以降
JRE	Sun JRE Verion 1.4 以降
APServer	Jakarta Tomcat 4.1 以降など、J2EE準拠のAPサーバ
DB 接続	Oracle JDBC Driver、Microsoft SQL Server 2000 Driver for JDBC その他 JDBC ドライバ

Web emulator2 の基本機能

クライアント（SSWind）の画面表示、制御

- ラベルやエディットの配置
  フォントサイズ（3段階）、カラー表示対応
- 各種ボタンのキー割り当て
- 画面初期表示時やエディット入力時の LED、ブザー、バイブレーションの設定
- エディットの入力制御（文字／数値、最小桁、最大桁）
- 各エディットでのスキャナ読み取りの ON/OFF 設定
- 読み取り可能なバーコードの種類を制御

データベース制御（接続可能なデータベースは1件のみ）

- 参照
- 更新
- ストアドプロシージャ呼出し
- コミット
- ロールバック

サーバ内の処理

- クライアントからのリクエストパラメータや SQL 実行結果をデータシートに保存
- データシートの作成、コピー、削除
- データシートに保存された値のチェック結果による処理分岐
- 画面情報をクライアントへ渡す

インストール方法

ここでは以下の環境で Web emulator2 サーバをインストールする手順を説明します。

OS	WindowsXP
Java	Sun J2SDK Standard Edition 1.4.2_09
APServer	Tomcat 4.1
DB	Oracle9i
インストールディレクトリ	C:¥webemulator2

JDK1.4 のセットアップ

1）インストール

J2SDK Standard Edition 1.4 を SUN のホームページから入手し、インストールします。

http://java.sun.com

詳しくは J2SE のドキュメントをご覧下さい。

※j2sdk-1_4_2_09-windows-i586-p.exe

2）システム環境変数 JAVA_HOMEを設定します。

例）J2SE を C:¥j2sdk1.4.2_09 にインストールした場合
JAVA_HOME= C:¥j2sdk1.4.2_09

3）システム環境変数PATHにJ2SEのbinディレクトリを追加します。

例）J2SE を C:¥j2sdk1.4.2_09 にインストールした場合
PATH=C:¥j2sdk1.4.2_09¥bin

システム環境変数の設定方法

- スタートメニュー等からマイコンピュータを右クリックしてプロパティを選択し、システムのプロパティウインドウを開きます。（図１）
- 詳細設定タブの環境変数ボタンをクリックして環境変数ウインドウを開きます。（図２）
- システム環境変数の一覧から該当の環境変数名を選択して編集ボタン（環境変数がない場合は新規ボタン）をクリックし、編集ウインドウを開きます。（図 3）
- 環境変数を設定（※）して OK ボタンをクリックし、編集ウインドウを閉じます。
  ※環境変数に設定する項目が複数ある場合は「;」（セミコロン）で区切ります。
- 環境変数ウインドウの OK ボタンをクリックして環境変数ウインドウを閉じます。
- システムのプロパティウインドウの OK ボタンをクリックしてシステムのプロパティウインドウを閉じます。

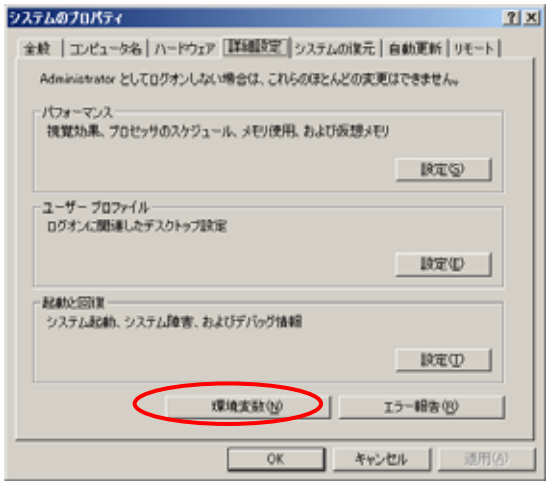


図１

図２

図３

Tomcat のインストール

1）インストール

Tomcat 4 を以下から入手し、インストールします。

http://jakarta.apache.org/site/binindex.cgi

※2005/9/2 現在　Tomcat4 の最新バージョンは 4.1.31 です

Web emulator2 のインストール

ここでは、Web emulator2 を c:¥webemulator2 の配下にインストールする手順を説明します。

1）インストール

添付のCD－ROMの webemulator2 ディレクトリ配下の各ファイルを c:¥webemulator2 ディレクトリへコピーします。

2）Tomcatのserver.xmlへのContext追加

インストールした Tomcat の conf¥server.xml の以下の行の上に Web emulator2 の Context を追加します。

```
<Context path="/examples" docBase="examples" debug="0" reloadable="true" crossContext="true">
```

↓

```
<Context path="/webemulator2"（※1） docBase="c:/webemulator2"（※2）
debug="0" reloadable="true" crossContext="true"/>
<Context path="/examples" docBase="examples" debug="0" reloadable="true" crossContext="true">
```

（※1）「path="/**webemulator2**"」 SSWind の設定にて記述するルートパス名となります。
SSWind では先頭の/を除いて記述します。
（※2）「docBase="**c:/webemulator2**"」 Web emulator2 をコピーしたディレクトリ名を指定します。

3）Web emulator2 のWEB-INF/web.xmlの修正

web.xml には Web emulator2 の設定状況が記述されています。以下の param-name の値を c:/webemulator2/conf に変更して下さい。

eventXml、customXml、sqlXml、pageXml

例）

```
<!-- Event.xml のパス -->
<context-param>
  <param-name>eventXml</param-name>
  <param-value>c:/webemulator2/conf </param-value>
</context-param>
```

※web.xml の詳細は次ページを参照して下さい。

web.xml の context-param タグに記述される設定情報

No	項目 <param-name>	内容 <param-value>
1	eventXml	Event.xml のパスを設定します
2	customXml	Custom.xml のパスを設定します
3	sqlXml	Sql.xml のパスを設定します
4	pageXml	Page.xml が格納されているパスを設定します
5	parseCache	Event.xml、Custom.xml、Sql.xml を AP 起動時にロードするかを設定します false に設定した場合は、クライアントからリクエストごとに設定ファイルを読み込みます
6	pageThreshold	SSWind Mobile 用にページキャッシュさせる Page.xml のサイズの閾値を設定します ※開発段階では、キャッシュしてしまうとページ XML を変更してもクライアントで反映されないため、開発時はキャッシュしないように閾値を大きめに設定して下さい
7	accesslog	Mobile Manager 用のアクセスログディレクトリのパスを設定します ※MobileManager を使用しない場合は定義する必要はありません
8	pageRecycle	イベント発行時に同一ページを表示する場合にコントロールを再利用するかを true、false で設定します
9	commaReplace	SQL 発行時にカンマ記号「'」の置換文字列を指定します
10	doPageXmlParse	Page.xml をサーバ側でパースするかを true、false で指定します
11	dbConnectionSetting	DB 接続の設定を指定します pool : プーリングを行います session : セッション毎に DB コネクションを作成します
12	drivers	接続先 DB の JDBC ドライバを指定します ※詳細は4）データベースの設定を参照
13	logfile	コネクションプーリングのログのパスを設定します
14	haisurf.url	DB 情報を設定します ※詳細は4）データベースの設定を参照
15	haisurf.maxconn	プーリング数を指定します
16	haisurf.user	接続ユーザを指定します
17	haisurf.password	接続ユーザのパスワードを指定します
18	applicationData1	自由な文字列を定義できます 定義した場合は、システムシートに ApplicationData1～3 のフィールド名でデータが格納されます
19	applicationData2	
20	applicationData3	

記述方法（例）

```
<context-param>
  <param-name>parseCache</param-name>（項目名を記述）
  <param-value>false</param-value>（内容を記述）
</context-param>
```

<param-name>以外の定義

```
<!-- listener 定義は固定です。以下の定義と同様にして下さい  -->
<listener>
    <listener-class>jp.co.sharedsys.common.util.SessionTimeoutChecker</listener-class>
</listener>

<!-- Servlet 定義 【haisurf】、【haisurfstart】定義は固定です。以下の定義と同様にして下さい  -->
<servlet>
    <servlet-name>haisurf</servlet-name>
    <servlet-class>jp.co.sharedsys.basis.HaisurfServlet</servlet-class>
</servlet>
<servlet>
    <servlet-name>haisurfstart</servlet-name>
    <servlet-class>jp.co.sharedsys.basis.HaisurfServletStart</servlet-class>
</servlet>

<!-- Servlet 定義 【handylist】、【accesslog】定義は MobileManager を使用する場合は、以下の  -->
<!--  定義と同様にして下さい。MobileManager を使用しない場合は定義する必要ありません  -->
<servlet>
    <servlet-name>handylist</servlet-name>
    <servlet-class>jp.co.sharedsys.basis.ClientList</servlet-class>
</servlet>
<servlet>
    <servlet-name>accesslog</servlet-name>
    <servlet-class>jp.co.sharedsys.basis.AccessLog</servlet-class>
</servlet>

<!-- Mapping 定義 【haisurf】、【haisurfstart】定義は固定です。以下の定義と同様にして下さい  -->
<servlet-mapping>
    <servlet-name>haisurf</servlet-name>
    <url-pattern>/haisurf</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>haisurfstart</servlet-name>
    <url-pattern>/haisurfstart</url-pattern>
</servlet-mapping>
```

（次頁へ）

```
<!-- Mapping 定義 【handylist】、【accesslog】定義は MobileManager を使用する場合は、以下の -->
<!-- 定義と同様にして下さい。MobileManager を使用しない場合は定義する必要ありません -->
<servlet-mapping>
    <servlet-name>handylist</servlet-name>
    <url-pattern>/handylist</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>accesslog</servlet-name>
    <url-pattern>/accesslog</url-pattern>
</servlet-mapping>

<!-- session-config 定義は session のタイムアウト時間（秒単位）を指定します -->
<!-- 未定義の場合は APServer のタイムアウト値を使用します -->
<session-config>
    <session-timeout>30</session-timeout>
</session-config>
```

4）データベースの設定

データベースの接続には、使用するデータベースが提供する JDBC ドライバが必要になります。ご使用の環境に合わせ、JDBC ドライバを取得してください。

- Oracleの場合
  http://otn.oracle.co.jp/software/tech/java/jdbc/T より取得してください。（要OTNアカウント）
  使用するOracleのバージョンと、JDKのバージョンに対応するJDBCドライバをダウンロードします。

- Microsoft SQL Server 2000 の場合
  http://www.microsoft.com/japan/msdn/sqlserver/downloads/jdbc/ より setup.exe を取得してインストールしてください。
  インストール後、Microsoft SQL Server 2000 Driver for JDBC ディレクトリーが作成されます。その中にlibディレクトリーが作成され、その中のjar File3つがJDBCドライバです。

取得した JDBC ドライバは Web emulator2 の WEB-INF/lib ディレクトリに配置します。

web.xml のデータベース項目設定（赤字の部分は使用するデータベースの設定を記述）

各データベース共通項目

haisurf.maxconn の設定　:Webemulator2 からデータベースへの同時接続数
haisurf.user の設定　　:データベースへ接続するユーザ名
haisurf.password の設定　:データベースへ接続するユーザのパスワード
logfile の設定　　:データベースの connection ログファイル名のフルパス※
※指定したディレクトリが存在しない等で
ログファイルに書き込みできない場合はサーバエラーとなります。

データベース毎に設定内容が変わる項目

- Oracle の場合
  drivers の設定　　: oracle.jdbc.driver.OracleDriver
  haisurf.url の設定　: jdbc:oracle:thin:@サーバ IP アドレス:ポート:SID

- Microsoft SQL Server 2000 の場合
  drivers の設定　　: com.microsoft.jdbc.sqlserver.SQLServerDriver
  haisurf.url の設定　:
  jdbc:microsoft:sqlserver:// サーバ IP アドレス:ポート; DatabaseName=DBName

5）log4j.xml の修正

Web emulator2 は、ログ出力に Jakarta Project の Log4j を使用しています。Log4j の設定ファイルが xml として/WEB-INF/classes に配置されていますので、設定状況をご確認下さい。
ログの出力場所は、存在するディレクトリが指定されていないと正しくログが出力されませんのでご注意下さい。設定箇所は以下の記述の部分です。

<param name="File" value="c:/webemulator2/log/Haisurf_trace.log"/>

また、出力場所を相対パス指定にする場合は、Tomcat起動バッチの作業フォルダを相対パスの基点となるフォルダに指定する必要があります。

XML ファイル定義

Web emulator2 で使用する XML ファイルには以下の種類があります。

種類	ファイル名	内容
イベント情報定義	Event.xml	画面からのイベント情報ごとに動作を記述します
SQL情報定義	Sql.xml	SQL（ストアド）情報を記述します
カスタム情報定義	Custom.xml	カスタムクラス情報を記述します
ページ情報定義	任意	各画面イメージを記述します

各 XML の仕様につきましては、別紙「XML 定義仕様書」を参照して下さい。

内部データ(データシート)

Web emulator2 では、システム情報・クライアントからのリクエスト・データベースへの問い合わせ結果等のデータを内部のデータシートという形で保持しています。

■システム情報のシート

システム情報のシートは Web emulator2 が自動的に作成します。シート内のデータは以下の通りです。

【シート名 : SystemSheet】

No	フィールド名	値
1	EventId	クライアントからのイベント ID
2	MACAddress	クライアントの MAC アドレス ※クライアントが SSWind の場合のみ
3	SSWindVer	SSWind クライアントのバージョン
4	OSVer	クライアントの OS バージョン
5	IpAddress	クライアントの IP アドレス
6	WindowSize	クライアントの Windows サイズ ※クライアントが SSWindMobile の場合のみ
7	Browser	クライアントのブラウザ名
8	SessionId	セッション ID
9	LastPage	最後に送信したページ
10	ApplicationData1	Web.xml に定義したデータ1
11	ApplicationData2	Web.xml に定義したデータ2
12	ApplicationData3	Web.xml に定義したデータ3

■http ヘッダシート

http ヘッダシートはクライアントからのリクエスト http のヘッダー情報を Web emulator2 が自動的に作成します。

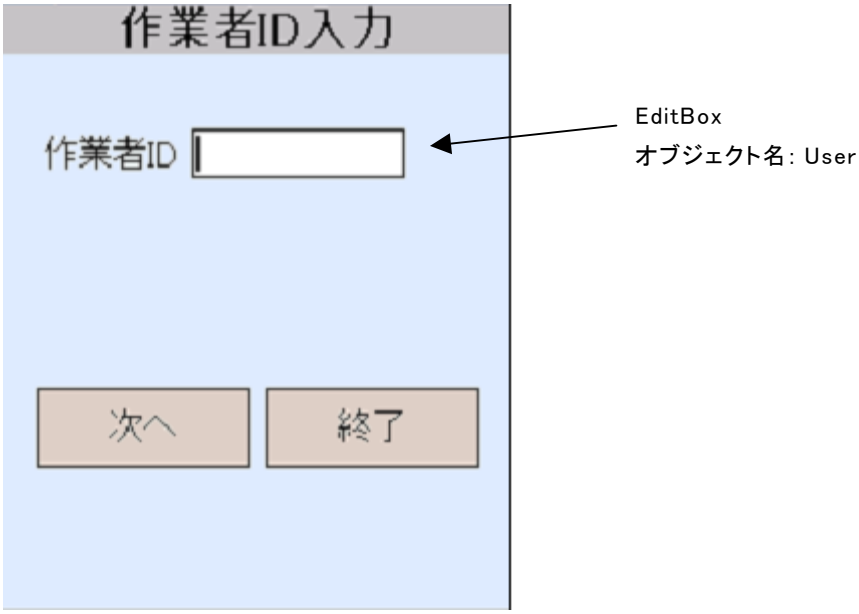
【シート名 : HttpHeaderSheet】

No	フィールド名	値
ヘッダの内容		

■画面からのリクエストシート

画面からのリクエストを受けると、画面ごとのデータシートを作成します。データシートの名前はページ XML ファイルの拡張子を除いた名称になります。

ページ XML:Login.xml



上記の画面からのリクエストを受け取った場合は、以下のデータシートが作成されます。

【シート名：Login】

No	フィールド名	値
1	User	※EditBox で入力された値
2	EditName_CodeType	※ EditBox にてスキャナ読込された場合に、バーコードのコードタイプ固定値 ※ 各コードタイプ固定値は下記【コードタイプ固定値】表を参照。

【コードタイプ固定値】

コードタイプ一覧	
JAN	COOP2OF5
CODE39	RSS
EAN128	QR
CODE128	PDF417
ITF	DataMatrix
NW7	MaxiCode
CODE93	UNKOWN
UNDUSTRIAL2OF5	

■データベースからの問い合わせ結果シート

SELECT の SQL を発行した場合は Sql.xml に記述してある SQLID 名でデータシートが作成されます。

Sql.xml

```
<sql id="USER_CHECK" type="select" ds-set="false">
  SELECT SID, USER_NAME, ADDRESS FROM T_M_USER
</sql>
```

【シート名 : USER_CHECK】

SID	USER_NAME	ADDRESS
80	一郎	東京都新宿区
81	次郎	東京都豊島区
82	三郎	東京都文京区
83	花子	東京都品川区

複数のレコードが取得出来た場合は、複数データのデータシートが作成されます。

INSERT、DELETE の SQL を発行した場合は、SELECT と同様に SQLID 名でデータシートが作成され、"RESULT"フィールドに更新件数が格納されます。

Sql.xml

```
<sql id="INSERT_USER" type="update" ds-set="false">
  INSERT T_M_USER( SID, USER_NAME, ADDRESS ) VALUES ( '60', '四郎', '東京都北区' )
</sql>
```

【シート名 : INSERT_CHECK】

No	フィールド名	値
1	RESULT	更新件数

※RESULT フィールド名は固定

ストアドを実行した場合も、SELECT と同様に SQLID 名でデータシートが作成されます。作成されたデータシートには、戻り値を格納する"RESULT"フィールドと、OUT 引数で定義したフィール名と値が格納されます。

Sql.xml

```
<sql id="SP_EXEC" type="sp" ds-set="false">
  { <param type="out" field="RESULT" attr="int">?</param> = call SP_EXEC(
    <param type="in" ds="Login.User" attr="string">?</param>,
    <param type="out" field="cnt" attr="int">?</param>
  ) }
</sql>
```

【シート名 : SP_EXEC】

No	フィールド名	値
1	RESULT	ストアドからの戻り値データ
2	cnt	第二引数の戻り値

※RESULT フィールド名は固定

データベース問い合わせ時のデータシートは Sql.xml の<sql>タグの ds-set 属性を"true"に設定しないかぎり、クライアントへレスポンス返信した段階で削除さます。

ds-set="true"を指定した場合は保持されますが、データベースからの複数レコードの大量データを保持するとメモリが圧迫されます。

■Event.xml 内で作成するカスタムシート

Event.xml で<append>、<remove>、<copy>タグを使用してデータシートの作成・削除が行えます。

Event.xml

```
<event id="login>
  <append ds="LoginBuffer.Message" value="認証エラーです"/>
  <append ds="LoginBuffer.Wave" value="alarm1.wav"/>
        :
        :
        :
```

【シート名 : LoginBuffer】

No	フィールド名	値
1	Message	"認証エラーです"
2	Wave	"alarm1.wav"

■カスタムクラス内で作成するカスタムシート

カスタムクラス内で自由にデータシートの作成・削除が行います。詳細につきましては「カスタムクラス」の章を参照して下さい。

■データシートの参照方法

Event.xml、Sql.xml からデータシートを参照する方法は、各タグの ds 属性を使用します。

```
ds="XXXXXX.YYYY"

    XXXXXX       . YYYY
[データシート名].[項目]
```

[データシート名]
　データシート名を指定します。必須です。

[項目]
　データシート内の項目を指定します。
　項目を指定せず、以下のシステム定義も使用することができます。

$ROWCOUNT	シート内の行数を取得
$COLCOUNT	シート内の項目数を取得

<remove>や<copy>タグの場合はデータシートのみの指定です。

Page.xml からデータシートを参照する方法は、

```
<label name"title">
  <x>10</x>
  <y>10</y>
  <w>120</w>
  <h>12</h>
  <caption>%XXXXX.YYY%</cation>
</label>
```

ds 属性と同様に[データシート名].[項目]を"%"記号でくくります。

カスタムクラス

Custom.xml に Java で作成されたクラスを登録することで Web emulator2 から呼び出すことが出来ます。カスタムクラスのクラス名、メソッド名は自由な名称を付けることができますが、メソッドの引数には次の二通りのケースがあります。

1)カスタムクラス内でデータベース操作を行わないケース

```
public void testCustomClass( DataSheetFactory factory ){
        :
        :
}
```

第一引数に DataSheetFactory を定義します。

2)カスタムクラス内でデータベース操作を行うケース

```
public void testCustomClass( DataSheetFactory factory, DBProcessor processor ){
        :
        :
}
```

第一引数に DataSheetFactory を定義します。
第二引数に DBProcessor を定義します。

DataSheetFactory はデータシートの取得・設定など、データシートに関連するメソッドを提供します。DBProcessor はデータベースへの問い合わせ・更新等のメソッドを提供します。

【DataSheetFactory クラス】

■ *DataSheet getDataSheet(String sheetName)*

sheetName で指定されたデータシートを取得します。

引数:
　String sheetName - 取得するデータシート名

戻り値:
　一致するデータシートが見つかった場合はデータシートを返します。
　一致するデータシートがない場合は null を返します。

■ *void setDataSheet(DataSheet sheet)*

指定されたデータシートを保存します。シート名は sheet.getSheetName()の名前になります。

引数:
　DataSheet sheet - データシート

戻り値:
　なし

■ *void setDataSheet(String name, DataSheet sheet)*

シート名を指定してデータシートを保存します。

引数:
　String name - シート名
　DataSheet sheet - データシート

戻り値:
　なし

■ *void remove(String sheetName)*

指定されたデータシートを削除します。

引数:
　String sheetName - 削除するデータシート名

戻り値:
　なし

■ *DataSheet createDataSheet(String sheetName)*

新規のデータシートを作成します。

引数:

String sheetName - 作成するデータシート名

戻り値:

なし

【DataSheet クラス】

■ *String getSheetName( )*

シート名を取得します。

引数:
　なし

戻り値:
　シート名

■ *void setField(String field, Object value)*

指定の値と指定されたフィールド名をデータシートの１行目に関連付けます。

引数:
　String field - フィールド名
　Object value - データ

戻り値:
　なし

■ *void setField(int index, String field, Object value)*

指定の値と指定されたフィールド名をデータシートの指定行目に関連付けます。

引数:
　int index - 行番号
　String field - フィールド名
　Object value - データ

戻り値:
　なし

■ *Object getField(String field)*

指定されたフィールド名の1行目にセットされている値を取得します。

引数:
　String field - フィールド名

戻り値:
　値を返します。指定のフィールドが存在しない場合は null を返します。

■ *Object getField(int index, String field)*

指定されたフィールド名の指定行にセットされている値を取得します。

引数:
　int index - 指定行
　String field - フィールド名

戻り値:
　値を返します。指定のフィールドが存在しない場合は null を返します。

■ *int getCount( )*

データシートの行数を取得します。

引数:
　なし

戻り値:
　行数を返します。

■ *List getColums( )*

データシートのカラム名リストを取得します。

引数:
　なし

戻り値:
　List 型のカラム名を返します。

■ *void setDeleteFlg(boolean flg)*

データシートに削除フラグを設定します。削除フラグを true に設定すると、クライアントへレスポンスを返信した後にデータシートが削除されます。

引数:
　boolean flg – 削除フラグ

戻り値:
　なし

■ boolean isDeleteFlg( )

データシートの削除フラグを取得します。

引数:
　なし

戻り値:
　削除フラグを返します。

【DBProcessor クラス】

■ *DataSheet executeQuery(String sql)*

SELECT の SQL を実行します。

引数:
　String sql – SQL 文

戻り値:
　実行結果の DBResultSet(データシート)を返します。

■ *DataSheet executeUpdate(String sql)*

更新系の SQL を実行します。

引数:
　String sql – SQL 文

戻り値:
　実行結果の DBResultSet(データシート)を返します。

■ *DataSheet executeStored(String sql)*

ストアドプロシージャのコールを行います。

引数:
　String sql – ストアドプロシージャ

戻り値:
　実行結果の DBResultSet(データシート)を返します。

Eclipse を使用したカスタムクラスの開発例

このカスタムクラスの開発例ではEclipse(http://www.eclipse.org/)を使用します。Eclipseとは、オープンソースでフリーの統合開発環境です。詳細、実行ファイルの取得は上記アドレスをご参照下さい。ここでは、Eclipseが既にインストールされているものとして解説いたします。

1. プロジェクトの作成

パッケージエクスプローラ上で右クリックメニューを開き、「新規(W)」から「プロジェクト(R)」を選択します。

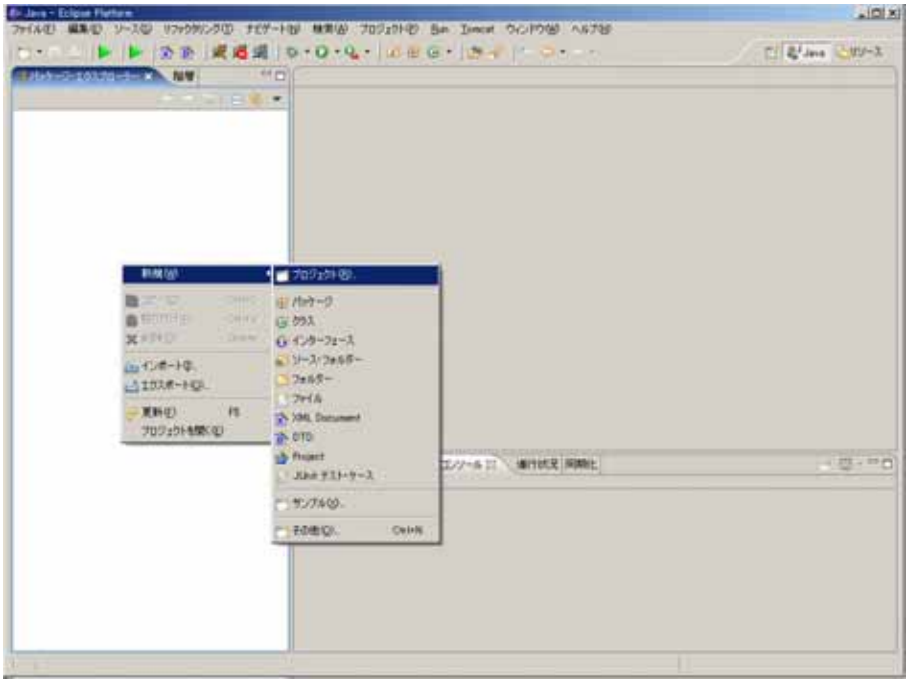
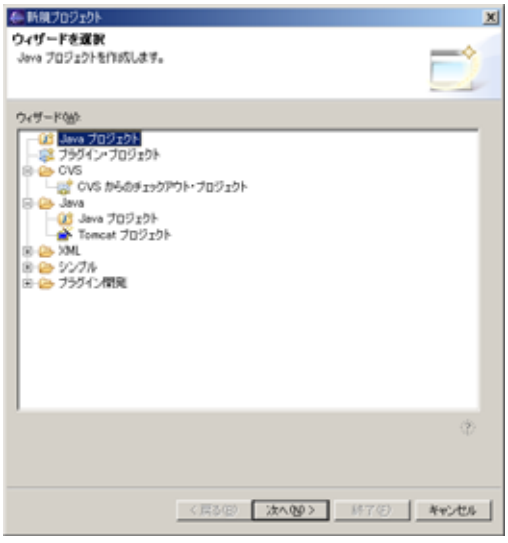


図 1

ウィザードが開くので、Java プロジェクトを選択し「次へ(N)」をクリックします。
次にプロジェクト名を入力する画面になるので、プロジェクト名を入力してください。ここでは"Web emulator2_CustomClass"とします。プロジェクト名を決めましたら、「次へ(N)」をクリックしてください。

図 2



Java 設定の画面が開くので、「ライブラリー(L)」のタブをクリックします。「外部 JAR の追加(X)」をクリックし、Web emulator2 をインストールしたフォルダの/WEB-INF/lib 配下にある"webemulator2.jar"を指定します。図 4 の状態になりましたら「終了(F)」をクリックします。

図 3

これでプロジェクトの作成は終了です。パッケージエクスプローラに作成されたプロジェクトが追加されます。

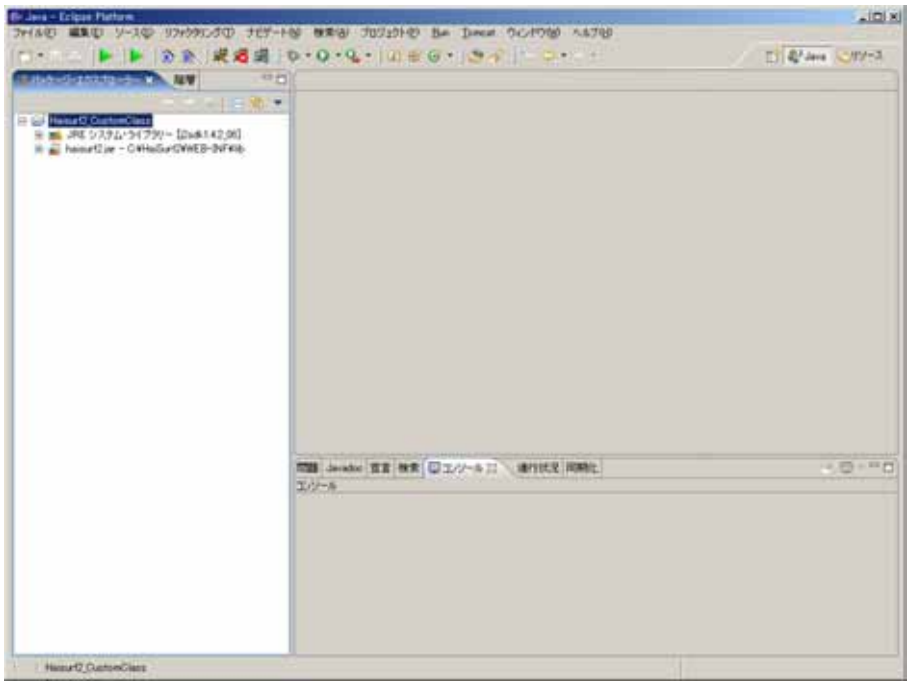


図 4

2. カスタムクラスの作成

1. で作成されたプロジェクト上で右クリックし、「新規(w)」から「クラス」を選択します

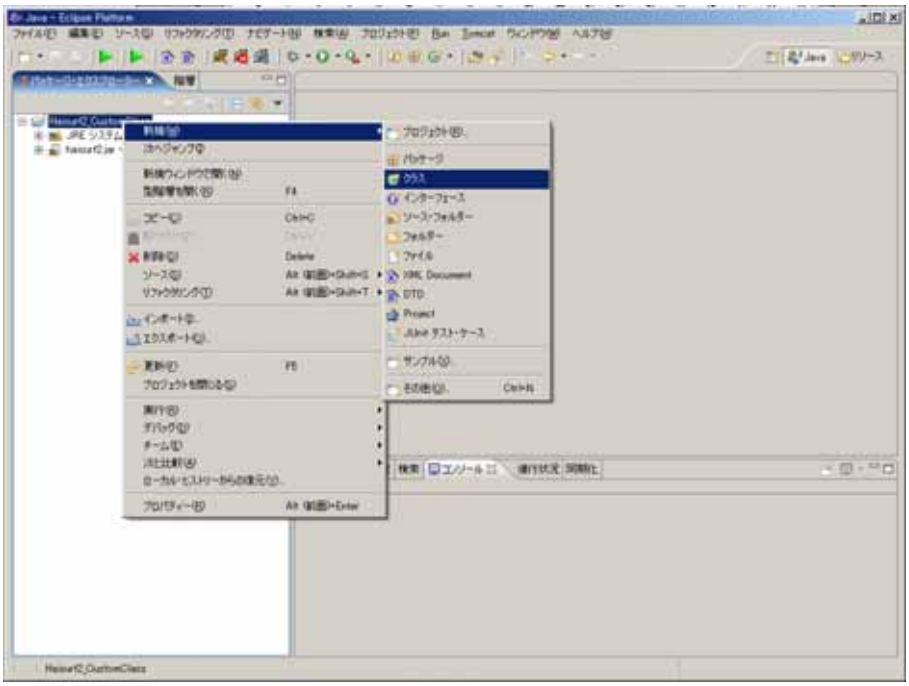


図 5

クラス作成のウィザードが開きますので、パッケージ、クラス名などを決めてください。ここでは、パッケージを" jp.co.sharedsys.webemulator2.customclass"、クラス名を"Custom"とします。

スーパークラス、インターフェースには特に制約はありません。自由に設定して頂いて構いませんが、Web emulator2 で使用するメソッドは、カスタムクラスで示しました引数定義にして頂かないと正常に動作しませんのでご注意下さい。

「終了(F)」をクリックすると、図 8 のようにプロジェクト配下にクラス及びパッケージが作成されます。
※ 作成されたクラスのコメント部分は、Eclipse の設定により異なります。

3. メソッドの作成

2. で作成したクラスに、Web emulator2 で使用するカスタムクラス用のメソッドを作成します。メソッドの引数には、前述(カスタムクラス)で示しました

public void testCustomClass(**DataSheetFactory factory**)
(データベース操作を行わない場合)

または

public void testCustomClass(**DataSheetFactory factory, DBProcessor processor**)
(データベース操作を行う場合)

の、2通りから選択します。ここでは、データベース操作を行わないこととし、 **DataSheetFactory** のみを引数としたメソッドを作成します。

※ 引数の指定を上記以外のものにすると、Web emulator2 では動作しませんのでご注意下さい。

クラス"Custom"にメソッドを作成します。ここでは、メソッド名を"setList"とし、引数は DataSheetFactory のみとします。

```
package jp.co.sharedsys.webemulator2.customclass;

import jp.co.sharedsys.basis.helper.DataSheetFactory;

public class Custom {
    public void setList(DataSheetFactory factory){

    }
}
```

メソッド内の処理は、他のデータシートからデータを取得、Page.xml の Combobox タグ items 子要素の形式に加工を行い新規作成したデータシートへ加工した結果を格納する、と言う内容です。

```
package jp.co.sharedsys.webemulator2.customclass;

import jp.co.sharedsys.basis.helper.DataSheetFactory;
import jp.co.sharedsys.basis.helper.DataSheet;
import jp.co.sharedsys.basis.HaisurfFatalException;

public class Custom {
    public void setList(DataSheetFactory factory) throws HaisurfFatalException{
        DataSheet sheetBasho = factory.getDataSheet("T_M_BASHO");
        DataSheet sheetBashoList = factory.createDataSheet("CUSTOM_BASHO_LIST");
        String bashoList = "";
        for(int i = 0; i < sheetBasho.lineCount(); i++){
            if (i != 0){
                bashoList += "¥t";
            }
            Object cd = sheetBasho.getField(i,"CD");
            Object name = sheetBasho.getField(i,"NAME");
            if ( cd !=null && name != null){
                bashoList += (String)cd + "¥t" + (String)name;
            }
        }
        sheetBashoList.setField("BASHO_NAME", bashoList);
        factory.setDataSheet(sheetBashoList);
    }
}
```

①

②

③

④

① DataSheetFactory から既にセットされている "T_M_BASHO" と言うシート名の DataSheet を取得します。

② DataSheetFactory.createDataSheet()を使い、 "CUSTOM_BASHO_LIST" と言うシート名の DataSheet を新規作成します。
注：DataSheetFactory.createDataSheet()は HaisurfFatalException をスローします。Exception が発生するタイミングは、**シート名に英数字及び"-"(ハイフン)、"_"(アンダーバー)**以外の文字が入っている場合です。

③ ①で取得したシートを、シートにあるデータの件数分ループさせ、"CD" 、"NAME" のフィールド名を持つデータを取得します。取得したデータをタブ区切りで連結させていきます。

④ ③で作ったデータを、②で作成した DataSheet に、フィールド名 "BASHO_NAME" 格納します。DataSheetFactory.setDataSheet()を使用し、 このクラスで作成した DataSheet を Web emulator2 へセットします。

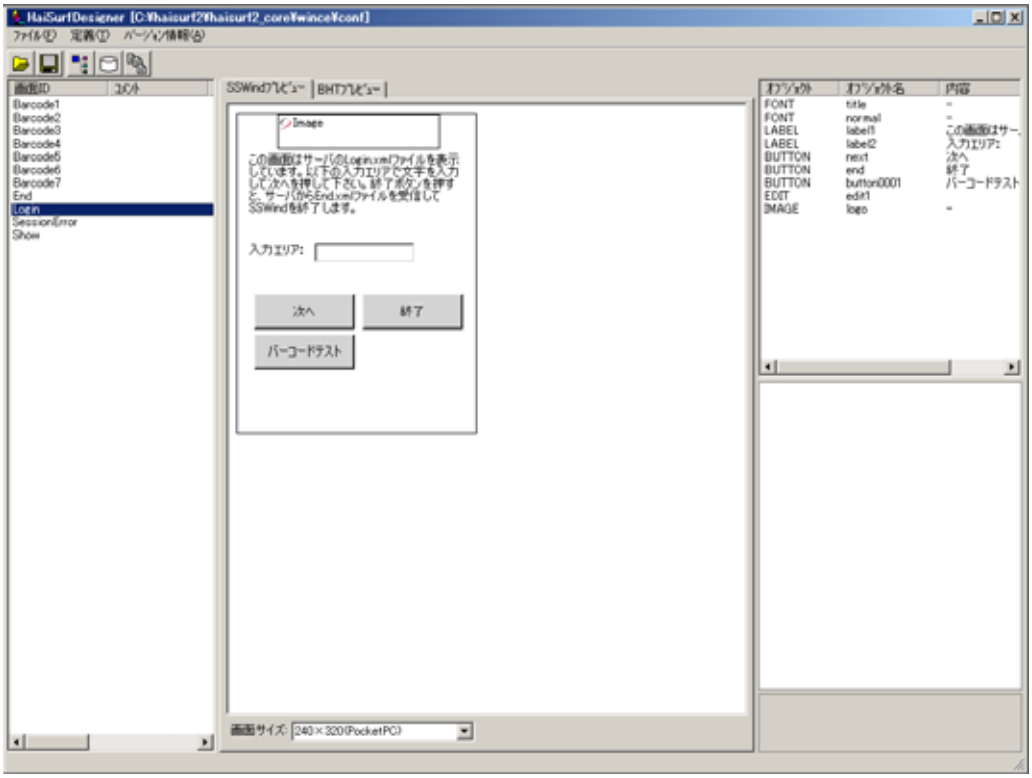
以上でカスタムクラスの作成は終了です。作成された class ファイルを AP サーバの適切な場所へ配置してください。

<その他製品> Web emulator2 Designer

Web emulator2 Designer とは Web emulator2 で使用される Event.xml、Sql.xml、Custom.xml、そして画面表示用 XML ファイルの作成を支援するアプリケーションです。GUI 操作による設定により、簡単に Web emulator2 の開発を行うことができ、煩わしい xml ファイルの編集をする必要がありません。

※ 詳細は製品添付の" Web emulator2 Designer-操作マニュアル"を参照してください。

【Web emulator2 Designer 画面例】



<その他製品> MobileManager

MobileManager とはハンディターミナルの動態監視・管理を行うことができるアプリケーションです。また、Web emulator2 を使用することで、アクセスログ等を取得することが出来ます。
※ 詳細は製品添付の"MobleManager-操作マニュアル"を参照して下さい。

【MoblieManager 画面例】