Gateway Builder アプリケーション開発ガイド

Gateway Builder アプリケーション開発ガイド

バージョン: 7.1

版数: 第 2 版

リリース: 2008 年 1 月

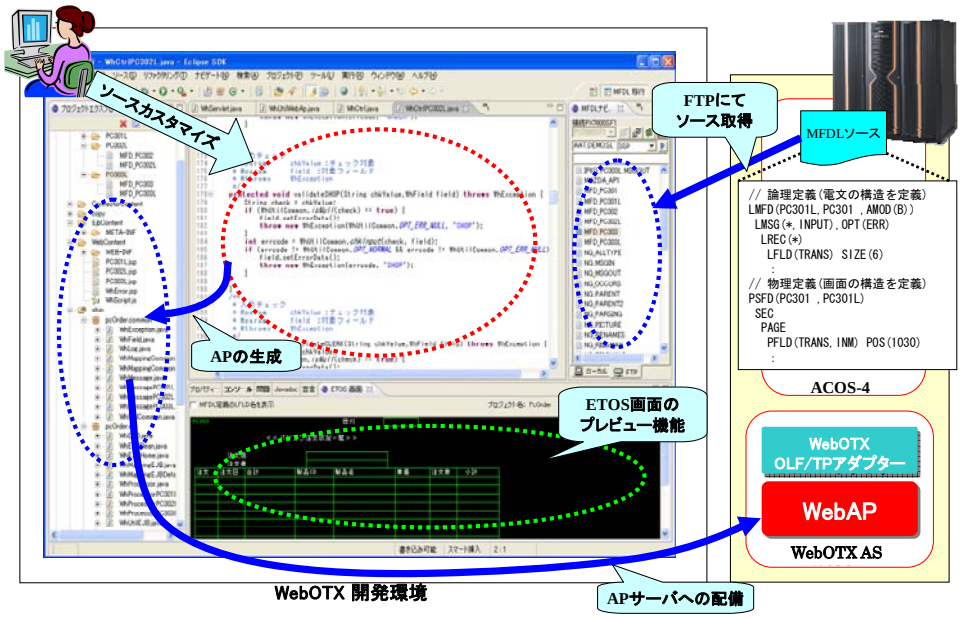
目次

1.Gateway Builder の概要

1.1.Gateway Builder の概要

「WebOTX Gateway Builder」は、「WebOTX Developer’s Studio」のプラグインとして動作し、ACOS ホスト上の ETOS 画面定義（MFDL）を直接変換して Web アプリケーションを自動生成する機能を提供しています。生成した Web アプリケーションは、WebOTX Application Server（以下、WebOTX AS と表記します）に配備して実行します。ETOS 業務の単純な Web 化であれば、プログラミングレスで実現できます。

「WebOTX Gateway Builder」を利用することで、ETOS 業務の Web 環境への移行をスムーズに行うことができ、システム構築コストを削減することができます。



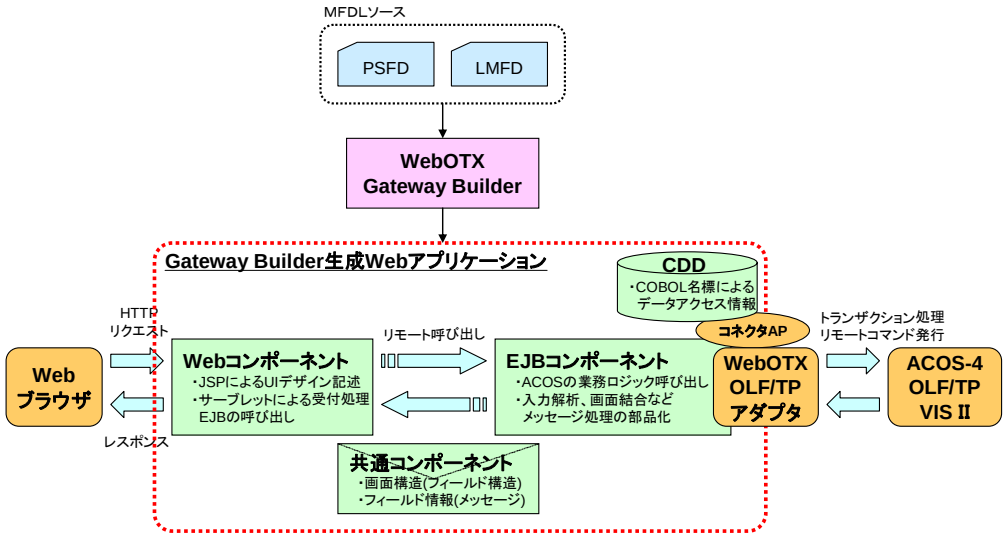
また、「WebOTX Gateway Builder」が生成する Web アプリケーションは、容易にカスタマイズできる構造になっています。ビューの改善や、他システムとの連携といった、Web アプリケーションの強化を自由に行うことができます。

1.2. 生成されるアプリケーションの概要

「WebOTX Gateway Builder」が MFDL ソースから生成する Web アプリケーションは、次のコンポーネントから構成されます。

- Web コンポーネント（JSP、サーブレットでブラウザを用いたユーザインタフェースを実現）
- EJB コンポーネント（ACOS 上の業務ロジックを呼び出す部品）
- 共通コンポーネント（Web コンポーネントと EJB コンポーネント間のインタフェース）

Web アプリケーションのユーザインタフェース(Web コンポーネント)と業務ロジック(EJB コンポーネント、ACOS-4 TPP)を分離することにより、ブラウザに表示する画面デザインに視点を置いたカスタマイズが容易な構造になっています。



Web アプリケーションから ACOS の業務ロジック呼び出しは、OLF/TP Adapter との連携により実現します。Gateway Builder と OLF/TP Adapter の CDD を用いた連携機構により、MFDL の論理情報をコネクタアプリケーションに反映できるため、Web アプリケーションにおいて COBOL 名標を使用したデータアクセスを実現できます。また、OLF/TP Adapter が採用している JCA 技術により、コネクションリソース管理機能を用いた高信頼の通信環境を実現できます。

1.3.Gateway Builder の機能

Gateway Builder では、前述したアプリケーション生成以外にも、さまざまな機能を提供し、アプリケーション生成やカスタマイズを支援します。

1.3.1.GUI で提供する機能

(1) MFDL ナビゲータ

Gateway Builder では、ACOS ホスト上の MFDL ソースを直接指定してアプリケーションを生成することができます。これにより、MFDL ソースを ACOS と PC などの開発環境で冗長に保持する必要がなくなり、MFDL ソースを ACOS 上で一元管理できます。

また、すでにローカル環境にしか MFDL ソースが存在しないような場合でも、ローカル環境の MFDL ソースを指定してアプリケーションを生成することもできます。

ACOS 上の MFDL ソースからアプリケーションを生成するためには、MFDL ナビゲータの FTP 機能を使用します。ローカル環境の MFDL ソースからアプリケーションを生成するためには、MFDL ナビゲータのエクスプローラを使用します。

それぞれの操作方法については、「3.2.2. MFDL の取得」を参照してください。

(2) 打鍵入力必須フィールドのチェック

従来の ETOS 端末では、入力フィールドに打鍵入力しないと入力を受け付けないフィールドがあります（MFDL 定義の PFLD の入出力属性に IN または IOF を指定したフィールド）。ブラウザには打鍵入力を認識する機能はありませんが、Gateway Builder では、以下の 2 種類のチェック方法を提供し、ETOS 端末に近い機能を実現しています。

- 初期値と異なる値が入力された場合に、打鍵入力されたものとして扱う
- null 値以外の値が入力された場合に、打鍵入力されたものとして扱う

打鍵入力のチェック方法は、Developer’s Studio の設定ウィンドウで指定でき、MFDL 移行プロジェクト単位、または、画面単位に変更することも可能です。指定方法については、「3.2.3. プロパティの設定」を参照してください。

(3) 省略不可フィールドのチェック

従来の ETOS 端末では、入力フィールドに値が設定されていないと入力を受け付けないフィールドがあります（MFDL 定義の LFLD に OPT(DESIG)を指定したフィールド）。本フィールドに関しても、(2)と同様のチェック方法を提供します。

省略不可フィールドのチェック方法は、Developer’s Studio の設定ウィンドウで指定でき、MFDL 移行プロジェクト単位、または、画面単位に変更することも可能です。指定方法は、「3.2.3. プロパティの設定」を参照してください。

(4) メッセージの言語

Web アプリケーション動作時のメッセージや、Developer’s Studio のコンソールに出力するメッセージとして、英語または日本語が選択できます。

この設定は、Developer’s Studio の設定ウィンドウで指定でき、MFDL 移行プロジェクト単位に変更することも可能です。指定方法については、「3.2.3. プロパティの設定」を参照してください。

(5) MFDL ソースの文字コードの指定

変換対象となる MFDL ソースの文字コードを指定できます(既定値は UTF-8)。MFDL ナビゲータの FTP の既定値は UTF-8 のため変更不要ですが、ローカル環境に保存している MFDL ソースの文字コードが UTF-8

と異なる場合は、本機能により文字コードを指定します。

この設定は、Developer’s Studio の設定ウィンドウで指定でき、MFDL 移行プロジェクト単位に変更することもできます。指定方法は、「3.2.3. プロパティの設定」を参照してください。

(6) 生成するアプリケーションの文字コード(エンコード)の指定

生成する Web アプリケーションの文字コード(エンコード)を指定することができます(既定値は UTF-8)。ブラウザのエンコードおよび使用する文字の種類に応じて、文字コードを選択します。

この設定は、Developer’s Studio の設定ウィンドウで指定でき、MFDL 移行プロジェクト単位に変更することもできます。指定方法は、「3.2.3. プロパティの設定」を参照してください。

(7) PF キーの作成

従来の ETOS 端末で PF キーに割り当てていた機能を、WEB の画面上のボタンやプルダウンメニューとして実現することができます。

この設定は、MFDL 移行プロジェクト単位、または、画面単位に指定することができます。指定方法は、「3.2.3. プロパティの設定」を参照してください。

(8) カラーの変更

ETOS では背景色が黒ですが、Web 画面では背景色を白にします。このため、ETOS の文字色をそのまま Web 画面で再現すると見づらくなります。

そこで、Gateway Builder では文字について ETOS 定義色と画面の表示色のマッピングを変更できます。

ETOS と Gateway Builder の色設定についてまとめると以下のようになります。

	ETOS	Gateway Builder	
文字(RED)	赤	赤	変更可能
文字(GRE)	緑	緑	変更可能
文字(YEL)	黄	黄	変更可能
文字(BLU)	青	青	変更可能
文字(MAZ)	マゼンタ	マゼンタ	変更可能
文字(CYA)	シアン	シアン	変更可能
文字(WHI)	白	黒	変更可能
文字(色指定なし)	緑	黒	変更可能
罫線	緑	黒	変更可能
背景	黒	白	変更可能

この設定は、Developer’s Studio の設定ウィンドウで指定でき、MFDL 移行プロジェクト単位に変更することもできます。指定方法は、「3.2.3. プロパティの設定」を参照してください。

なお、罫線色や背景色を変更したい場合は、「5.2.4画面項目の文字色と罫線を変更する例」も参考にしてください。

1.3.2.生成したアプリケーションで提供する機能

(1) 入出力フィールドに対応する HTML タグ

ETOS 端末には、フィールドの入出力属性として、以下のものがあります。それぞれ、以下の表に示すように HTML タグが生成されます。

属性	説明	生成する HTML タグ
IN	入力フィールド	<input type=text ～>
INM	入力フィールド(打鍵入力必須)	<input type=text ～>
INP	入力フィールド(入力保護)	<input type=hidden ～><span ～>表示値
IO	入出力フィールド	<input type=text ～>
IOF	入出力フィールド(打鍵入力必須)	<input type=text ～>
IOP	入出力フィールド(入力保護)	<input type=hidden ～><span ～>表示値
OUT	出力フィールド	<input type=hidden ～><span ～>表示値
CNS	固定値フィールド	表示値

(2) HTML タグの動的生成

従来の ETOS 端末では、TPP からの指示によりフィールドの属性を動的に変更できます(MODIFY 指定)。Gateway Builder の生成するアプリケーションでも、フィールドの入出力属性(IO、IOF、IOP)、色、ブリンク、カーソルの位置づけを、TPP からの指示により動的に変更することができます。この動的変更は、JSP 上で以下のメソッドにより実現しています。

```
WhUtilWebAp.getHtmlTag(message.get<フィールド名>)
```

＜例＞

JSP の定義

```
<%= WhUtilWebAp.getHtmlTag(message.getSHOP(),did) %>
```

HTML 展開イメージ

```
<input type=text name="SHOP" size="20" maxlength="20" value="" class=' color_yellow' >
```

HTMLタグの動的生成が不要で、フィールドに格納された値を直接使用したい場合は、JSPをカスタマイズすることで、以下のように直接値を参照することもできます。

```
message.get<フィールド名>.getValue()
```

＜例＞

JSP の定義

```
<%= message.getSHOP().getValue() %>
```

HTML 展開イメージ

```
xx本店
```

(3) ブリンク機能

フィールドのブリンクは、Java スクリプトとブリンク制御用 ID 生成クラス(WhDhtmlId)で提供しています。ブラウザで Java スクリプトを無効としている場合は、ブリンク機能を使用できませんので注意してください。

(4) カーソルの位置づけ機能

カーソルの位置づけは、Java スクリプトで提供しています。ブラウザで Java スクリプトを無効としている場合は、カーソルの位置づけ機能を使用できませんので注意してください。

(5) 画面結合

MFDL 定義で通常ページング(PAGE 指定)として定義している画面は、Web アプリケーションでは1画面として生成します。このため、画面送りなどの操作を行うことなく入力操作などが行えます。

1画面におさまらない表形式での出力などで、繰り返しページングを使用している場合は、Javaソースのカスタマイズで1画面化することができます。カスタマイズ方法については、「5.5.1 ACOSホストからの複数回の出力を結合する例」を参照してください。

(6) ブラウザ操作による重複実行のガード

トランザクション処理要求中のブラウザ操作による重複実行を防ぐための、ダイアログ表示機能を Java スクリプトで提供しています。ブラウザで Java スクリプトを無効としている場合は、ガードされませんので注意してください。

2.準備

2.1.実行環境の設定

本節では、生成する Web アプリケーションを動作させるための環境について記述しています。

2.1.1.OLF/TP Adapter の設定

ACOS ホストへの通信は OLF/TP Adapter を経由して行います。

ここでは、ACOS ホストと通信を行う OLF/TP Adapter について実行環境の設定を行います。

本作業は後で実施しても構いません。本設定は Web アプリケーションを実行する前までに実施してください。

(1) OlfAdapter.ini の設定

ACOS ホスト定義(OlfAdapter.xml)のパス指定、ログ出力設定、コネクション設定などを行います。

ACOS ホスト定義(OlfAdapter.xml)のパス指定は必須です。

[格納場所]
<WebOTX install dir>/Adapter/OLFTP/Run/Conf/OlfAdapter.ini

[定義例]

```
#
# 端末定義ファイル(OlfAdapter.xml)のパスを指定します。
#
# RmtDefinition = /etc/olf/OlfAdapter.xml
RmtDefinition = C:¥¥WebOTX¥¥Adapter¥¥OLFTP¥¥Run¥¥conf¥¥OlfAdapter.xml

#
# OLF/TP-UT プロトコルのコネクションの開設可能総数の上限を指定します。
#
UTConnection = 4

#
# 受信スレッドの最大値を指定します。
#
MaxRecv = 4

#
# 同時開設可能な最大セッション数を指定します。
# 実行するアプリケーションの多重度以上を指定してください。
#
MaxSession = 4

#
# トレースレベルを指定します。
#
TraceLvl = ConDis

#
# ログファイルを出力するディレクトリを指定します。
#
TraceLogDir = C:¥¥WebOTX¥¥domains¥¥domain1¥¥logs¥¥OLFTPAdapter¥¥

#
# ログファイルあたりの最大行数を指定します。
```

```
#
TraceLogFileSize = 5000

#
# 保存するログファイルの最大数を指定します。
#
TraceLogFileCount = 10
```

※ログ出力設定については WebOTX AS のエディションにより設定方法が異なります。特に、

Standard/Enterprise Edition の場合は、書き込み可能なログ出力先を指定してください。

（上記、サンプルの指定が参考になります）

OlfAdapter.ini の他の設定や詳細な内容については、次のマニュアルを参照してください。

```
# WebOTX OLF/TP Adapter 運用ガイド
#  4. WebOTX OLF/TP Adapter 実行環境の運用
#   4.3. 設定
#    4.3.1. 通信環境定義(OLFAdapter.ini)
```

(2) OlfAdapter.xml の設定

通信相手となる ACOS ホストの情報を定義します。この設定は必須です。

[格納場所]

<WebOTX install dir>/Adapter/OLFTP/Run/Conf/OlfAdapter.xml

[定義例]

```
<!--
    ACOS ホストの設定
-->
<RmtDef xmlns="">
  <!--
    RmtName には ACOS ホストのシンボリック名を指定します。
  -->
  <RmtName xmlns="">iPX9000UT</RmtName>

  <!--
    RmtAddr には ACOS ホストのアドレスを指定します。
    ホスト名、あるいは IP アドレス形式（IPv4 形式）で指定します。
  -->
  <RmtAddr xmlns="">192.168.0.1</RmtAddr>

  <!--
    RmtPort には ACOS ホストの着信ポートを指定します。
  -->
  <RmtPort xmlns="">62004</RmtPort>

  <!--
    この ACOS ホストとの最大コネクション数を指定します。
    OlfAdapter.ini の「UTConnection」「UWConnection」に指定した値
    以下を指定してください。
  -->
  <MaxConnection xmlns="">1</MaxConnection>

</RmtDef>
```

OlfAdapter.xml の他の設定や詳細な内容については、次のマニュアルを参照してください。

```
# WebOTX OLF/TP Adapter 運用ガイド
#  4. WebOTX OLF/TP Adapter 実行環境の運用
#   4.3. 設定
#    4.3.2. 端末定義(OLFAdapter.xml)
```

2.1.2.ドメインの設定

ここでは、Web アプリケーションを配備・実行するドメインについての設定を行います。

本作業は後で実施しても構いません。本設定はアプリケーションを配備する前までに実施してください。

(1) アプリケーショングループとプロセスグループの作成

Web アプリケーションを実行するドメインにはアプリケーショングループとプロセスグループを作成する必要があります。

※ WebOTX AS のエディションによっては、本作業は不要です。

アプリケーショングループおよびプロセスグループの作成には次の方法があります。

・統合運用管理ツールによる作成

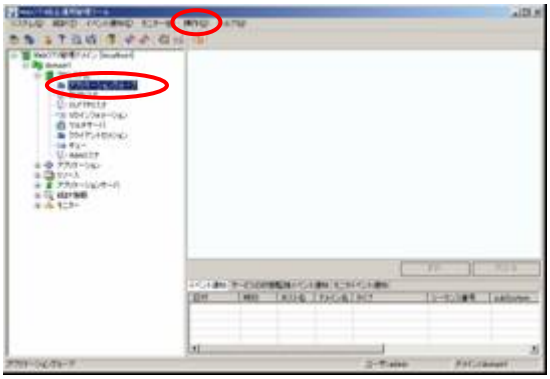
・運用管理コマンドによる作成

ここでは、「統合運用管理ツール」を利用した手順を説明します。

① 統合運用管理ツールを起動し、ドメインへ接続します。

② 左のウィンドウから、「domain1」→「TPシステム」→「アプリケーショングループ」を選択します。

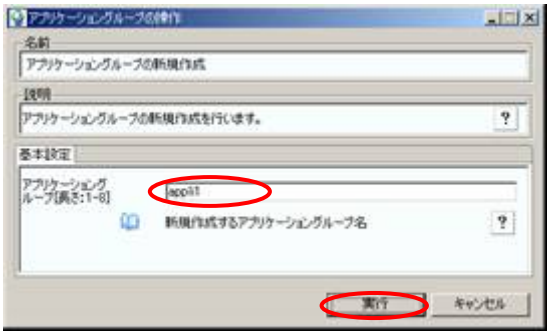
③ メニューバーから、「操作」→「アプリケーショングループの新規作成」を選択します。



④ 次の値を設定します。

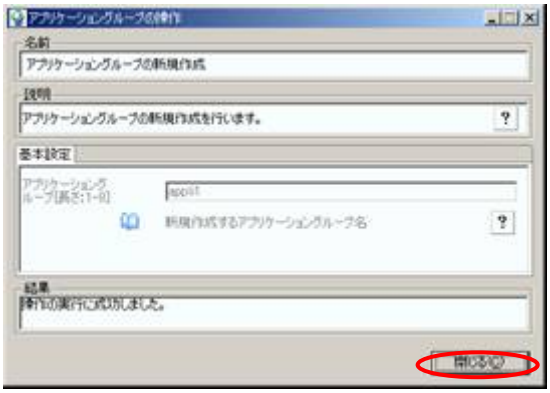
「アプリケーショングループ」:
任意の名称

※ 右の図は、アプリケーショングループ名を”appli1”とした例です。



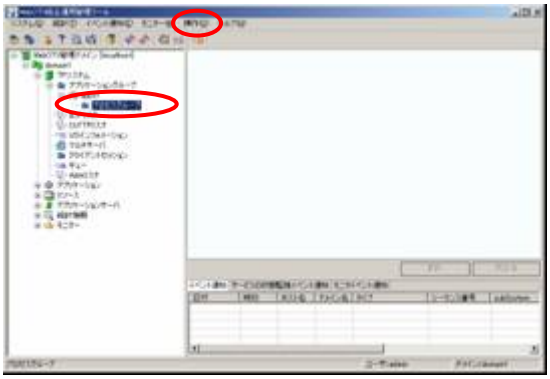
⑤ 「実行」ボタンを押下します。

⑥ 「閉じる」ボタンを押下します。



⑦ 左のウィンドウから、「アプリケーショングループ」→「<④で作成したアプリケーショングループ名>」→「プロセスグループ」を選択します。

⑧ メニューバーから、「操作」→「プロセスグループの新規作成」を選択します。



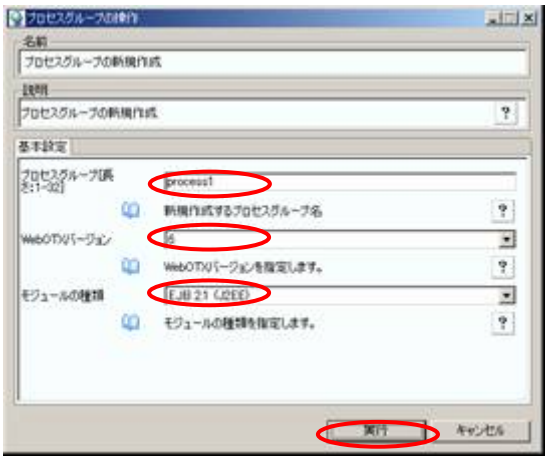
⑨ 次の値を設定します。

「プロセスグループ」:
任意の名称

※ 右の図は、プロセスグループ名を”process1”とした例です。

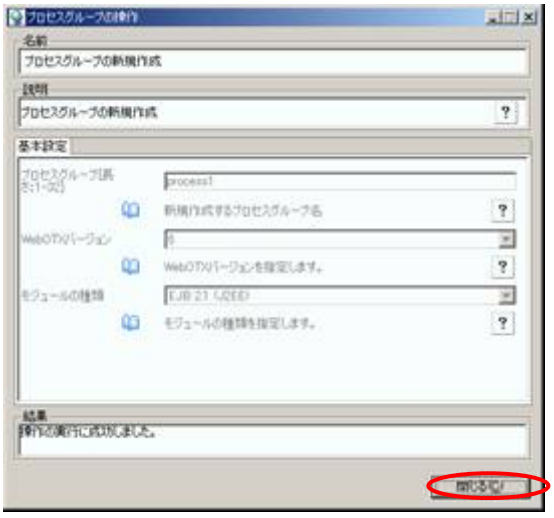
「WebOTX バージョン」:
6

「モジュールの種類」:
EJB 2.1 (J2EE)



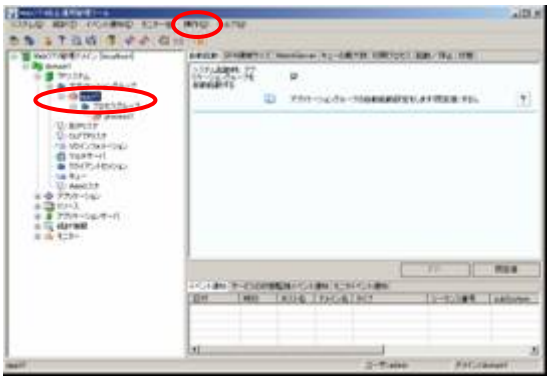
⑩ 「実行」ボタンを押下します。

⑪ 「閉じる」ボタンを押下します。

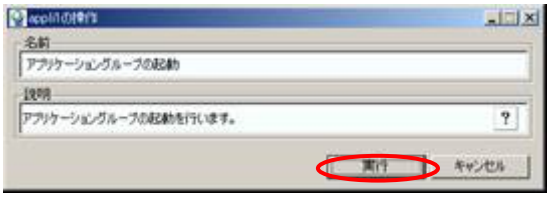


⑫ 左のウィンドウから、「アプリケーショングループ」→「<④で作成したアプリケーショングループ名>」を選択します。

⑬ メニューバーから、「操作」→「アプリケーショングループの起動」を選択します。



⑭「実行」ボタンを押下します。



⑮「閉じる」ボタンを押下します。

以上でアプリケーショングループとプロセスグループの作成および起動が完了しました。

2.2.開発環境の設定

本節では、Gateway Builder アプリケーションの開発環境について記述しています。

2.2.1.Developer's Studio の設定

Developer's Studio の初期設定を行います。

(1) Developer's Studio の設定

ここでは、以下の項目について確認および設定を行います。

- Java インストール済み JRE の確認
- Java ビルドパスの確認
- WebOTX ホームディレクトリの確認
- 配備記述子エディタの確認
- WebOTX ランタイムの設定

設定方法については、次のマニュアルを参照してください。

```
# WebOTX マニュアル  アプリケーション開発ガイド
#   第 2 部  初期設定(pdf 形式)
#      1. Developer's Studio
#      3. WTP
```

その他の項目の設定は必須ではありません。アプリケーション開発ガイドを参照の上、必要に応じて設定してください。

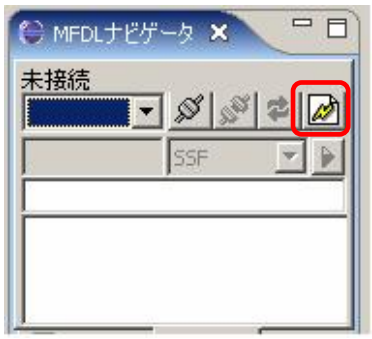
2.2.2.Gateway Builder の設定

(1) FTPの設定

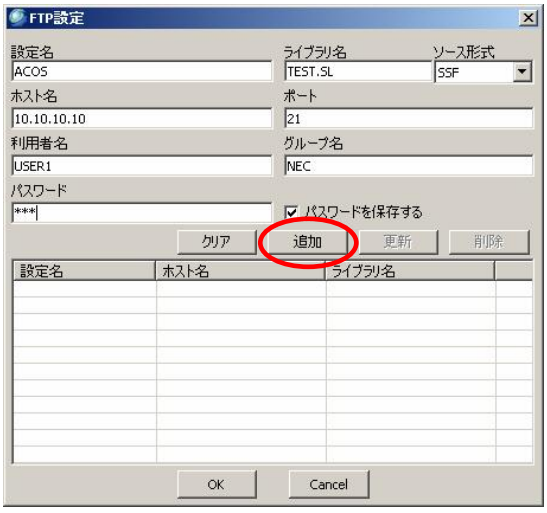
ACOS ホスト上のライブラリから MFDL を取得するための FTP の設定を行ないます。

①メニューバーから、「ウィンドウ」→「パースペクティブを開く」→「MFDL 移行」を選択して、「MFDL 移行」パースペクティブを開きます。

「MFDLナビゲータ」の「FTP接続先を設定」ボタンを押下します。



②「FTP 設定」画面が表示されます。



以下の値を設定します。

「設定名」:　　一覧に表示する名前

「ライブラリ名」:　MFDL を格納しているライブラリ名

「ソース形式」:　SSF または SARF

「ホスト名」:　　IP アドレスまたはホスト名

「ポート」:　　ホストのポート番号（既定値は 21）

「利用者名」:　　ホストに接続するときの利用者名

「グループ名」:　ホストに接続するときのグループ名

「パスワード」:　ホストに接続するときのパスワード

設定が終れば、「追加」ボタンを押下します。

各ボタンの意味を以下に記載します。

「クリア」ボタン:　上部に記述した設定内容をクリアします。

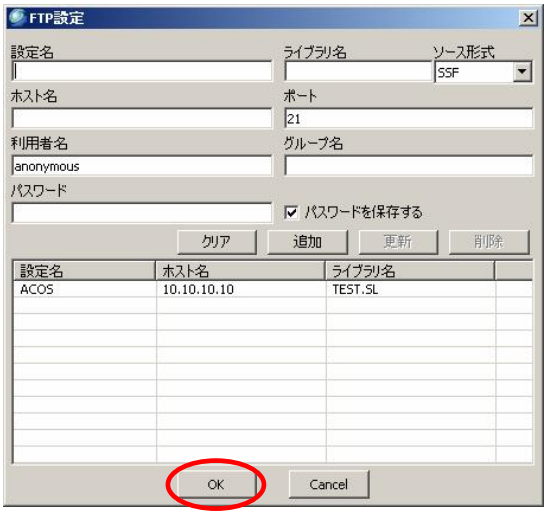
「追加」ボタン:　上部に記述した設定内容を、下部の一覧フィールドに登録します。

「更新」ボタン:　下部の一覧フィールドで選択した設定に対する変更を反映します。

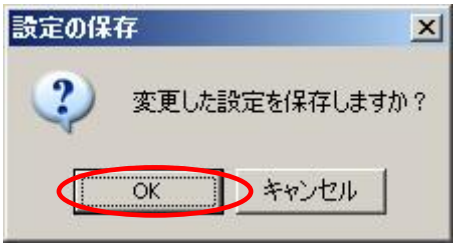
「削除」ボタン:　下部の一覧フィールドで選択した設定を削除します。

③設定した内容が一覧フィールドに表示されます。

「OK」ボタンを押下します。



④「設定の保存」画面が表示されるので、「OK」ボタンを押下します。



⑤「MFDL ナビゲータ」に設定名が表示されます。

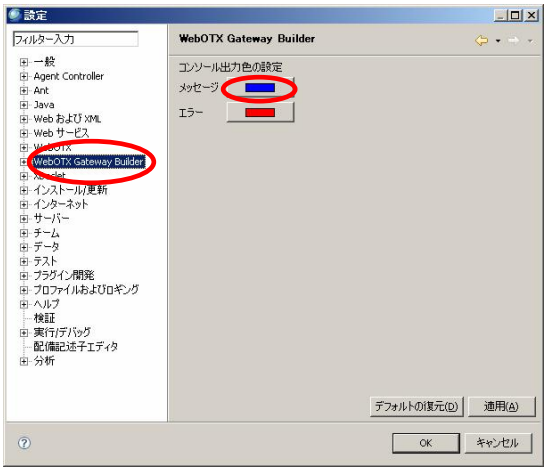
以上で FTP の設定は終りです。

※ここで設定した内容は、以下のファイルに保存されます。
<WebOTX install dir>/Studio/plugins/com.nec.jp.iPX.apd.ui_1.0.0/APDFtpSvrInfo.inf
Gateway Builder をアンインストールする場合は、必要に応じてバックアップしてください。

(2) Develper's Studio の設定

MFDL移行プロジェクト全体の設定を行ないます。ここで設定した内容がプロジェクト作成時の既定値になります。

①メニューバーから、「ウィンドウ」→「設定」を選択して、設定画面を開きます。



「WebOTX Gateway Builder」画面では、コンソールに出力するメッセージの色を設定できます。

「メッセージ」:

コンソールメッセージの色を指定します。

「エラー」:

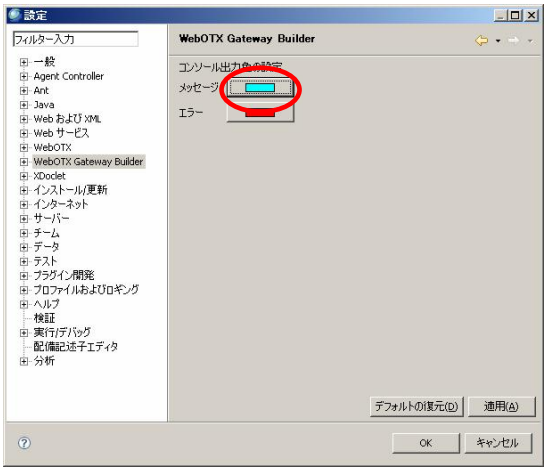
エラーメッセージの色を指定します。

以降に、「メッセージ」ボタンを押下したときの動作例を示します。

②「色の設定」が表示されます。

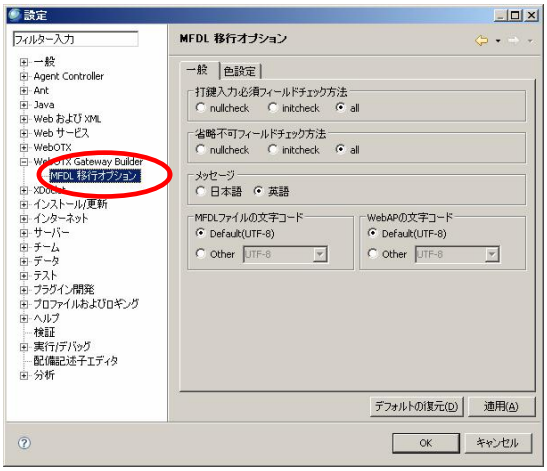
変更する色を選択し、「OK」ボタンを押下します。

③設定が有効になります。



④「MFDL 移行オプション」画面では生成する Web アプリケーションに関する指定を行います。

以下の値を設定します。



「打鍵入力必須フィールドチェック方法」:

ETOS 端末で入力フィールドに打鍵入力しないと入力を受け付けないフィールド（MFDL 定義の PFLD の入出力属性に、IN または IOF を指定したフィールド）のチェック方法を指定します。

「nullcheck 」:

入力フィールドに値が指定されていないときにエラーとします。

「initcheck 」:

入力フィールドの値が初期値と同じ値のときにエラーとします。

「all 」:

上記両方のチェックを行います（既定値）。

「省略不可フィールドチェック方法」:

ETOS 端末で入力フィールドに値が設定されていないと入力を受け付けないフィールド（MFDL 定義の LFLD に OPT(DESIG)を指定したフィールド）のチェック方法を指定します。

各指定値の意味は、「打鍵入力必須フィールドチェック方法」と同じです。

「メッセージ」:

Web アプリケーション動作時のメッセージや Developer’s Studio のコンソールに出力するメッセージとして、英語または日本語を選択できます。

既定値は英語です。

「MFDL ファイルの文字コード」:

MFDL ファイルの文字コードを指定します。既定値は UTF-8 です。

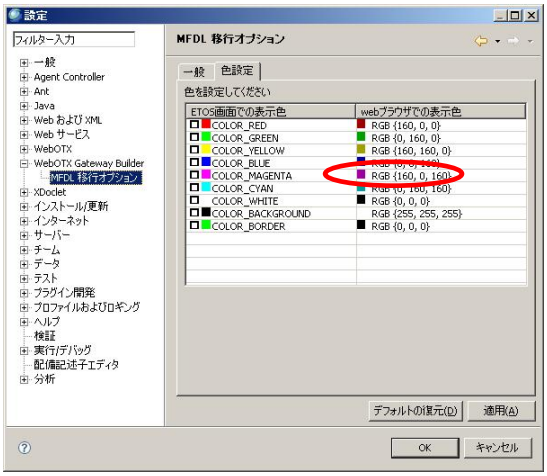
メニュー以外にも Java でサポートされる文字コードを直接指定できます。

「WebAP の文字コード」:

作成する Web アプリケーションのエンコードを指定します。既定値は UTF-8 です。ブラウザのエンコードおよび使用する文字の種類に応じて、文字コードを指定します。

メニュー以外にも Java でサポートされる文字コードを直接指定できます。

⑤「色設定」タブでは文字について ETOS 定義色と画面の表示色のマッピングを変更できます。



「ETOS 画面での表示色」:

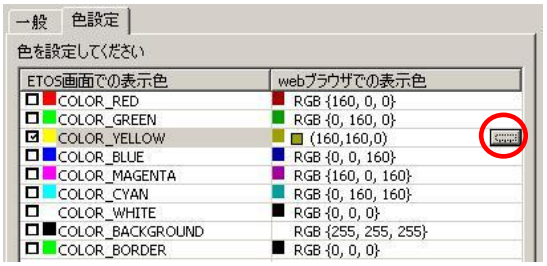
ETOS 画面の表示色。マッピングを変更する場合にチェックします。

「Web ブラウザでの表示色」:

Web アプリケーション画面の表示色。

⑥現在設定されている色が表示されます。

右図の丸で囲った箇所を選択します。

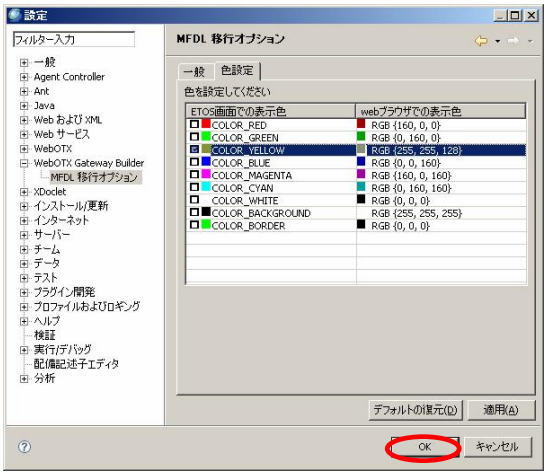


⑦「色の設定」画面が表示されます。

色を選択し、「OK」ボタンを押下します。

⑧指定が反映されます。

「OK」ボタンを押下します。



2.2.3.配備ツールの設定

Web アプリケーションは、実行するサーバに「配備」することで実行可能になります。

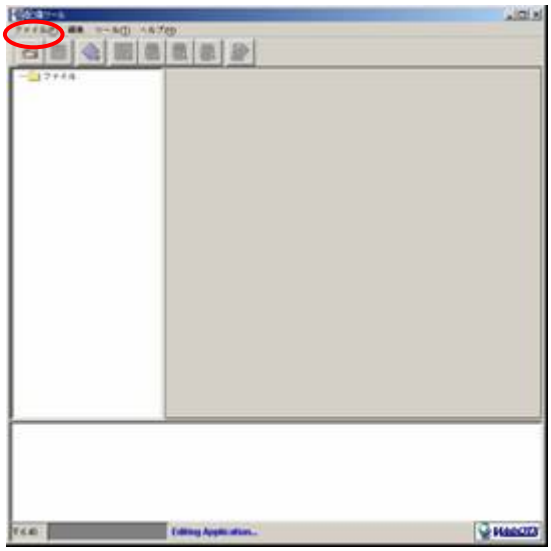
ここでは、「配備」を行うための「配備ツール」についての設定を行います。

(1) 配備ツールの設定

配備ツールを初めて使用するときなど、配備対象のサーバが配備ツールに未登録の場合、サーバの登録を行います。

① 配備ツールを起動します。

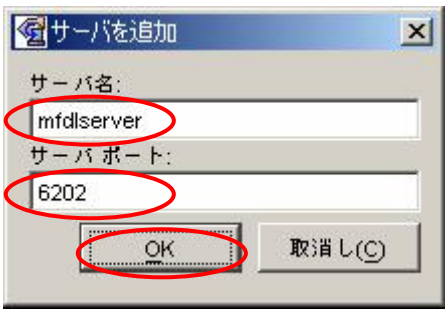
② 配備対象のサーバが未登録の場合、「ファイル」→「サーバを追加」を選択します。



③ 配備対象のサーバ名とポート番号を指定します。

④ 「OK」ボタンを押下します。

以上で配備対象のサーバが登録されます。



3.アプリケーションの作成

3.1.開発の流れ

Gateway Builder を利用したアプリケーションの開発は、以下の流れで実施します。

① MFDL 移行プロジェクトによる WebAP ソース生成

ACOS ホストから MFDL ソースを取得し、WebAP ソースを生成します。

② コネクタアプリケーションの作成と配備

①で生成した cdd ファイルを使用して ACOS ホストと接続するためのコネクタアプリケーションを作成し、アプリケーションサーバに配備します。コネクタアプリケーションは、WebOTX OLF/TP Adapter を利用して ACOS ホストに接続します。

③ EJB アプリケーションの作成と配備

①で生成した EJB ソースを使用して EJB アプリケーションを作成し、アプリケーションサーバ(EJB コンテナ)に配備します。EJB アプリケーションでは、ACOS ホストへの入力電文の作成、出力電文の受け取りなどを行っています。

④ Web アプリケーションの作成と配備

①で生成した Servlet ソース、JSP を使用して Web アプリケーションを作成し、アプリケーションサーバ(Web コンテナ)に配備します。Web アプリケーションでは、データの入出力や、入力データのチェック、結果画面のフォワード、HTML 画面の生成などを行っています。

次節以降で詳細に説明します。

3.2.MFDL 移行プロジェクト

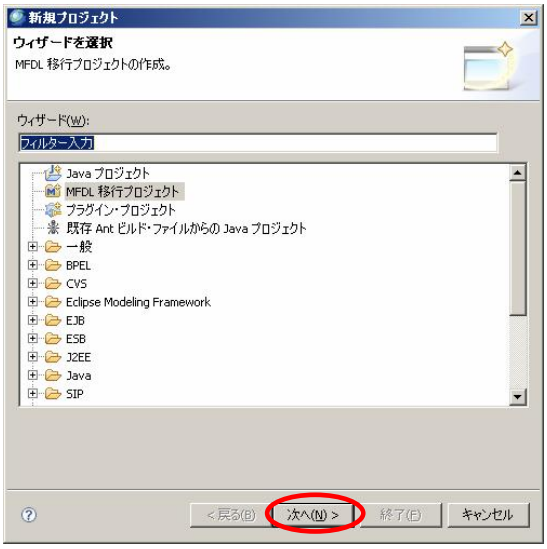
本節では、MFDL 移行プロジェクトの作成について記述しています。以下の手順により、Web アプリケーションの生成を行ないます。

3.2.1.プロジェクトの作成

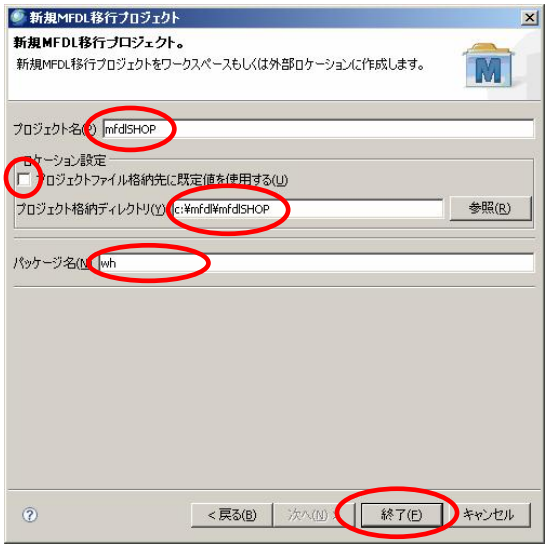
Web アプリケーションを生成するためのプロジェクトを作成します。

①メニューバーから、「ファイル」→「新規」→「プロジェクト」を選択して、「新規プロジェクト」画面を開きます。

「MFDL 移行プロジェクト」を選択して、「次へ」ボタンを押下します。



②「新規 MFDL 移行プロジェクト」画面が表示されます。次の値を指定します。



「プロジェクト名」:

新規プロジェクト名を指定します。

例:mfdlSHOP

「ロケーション設定」:

プロジェクトファイル格納先を指定します。

既定値は、<WebOTX install dir>/Studio/workspace/<project name> 直下に作成します。

既定値以外に格納する場合は、チェックをはずし、プロジェクト格納ディレクトリをフルパスで指定します。

例:c:¥mfdl¥mfdlSHOP

「パッケージ名」:

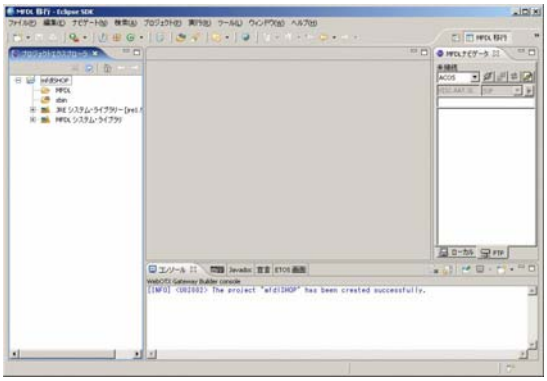
パッケージ名は、実行環境で一意の名前にする必要があります。

例:wh

※「プロジェクト名」、「ロケーション設定」および「パッケージ名」は後から変更できないので注意してください。変更したい場合は、プロジェクトの再作成が必要です。

「終了」ボタンを押下します。

③プロジェクトが作成されます。



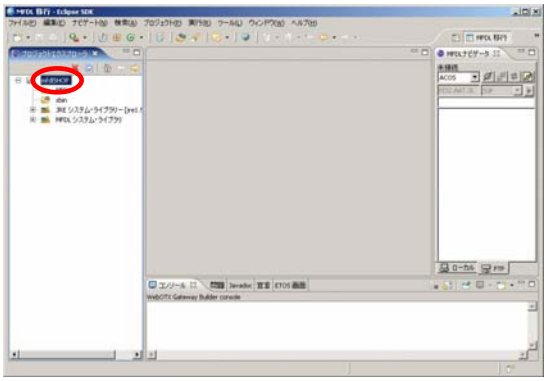
3.2.2.MFDL の取得

Gateway Builder が変換に使用する MFDL ソースは、ACOS ホストから直接取得する方法とローカル環境から取得する方法があります。

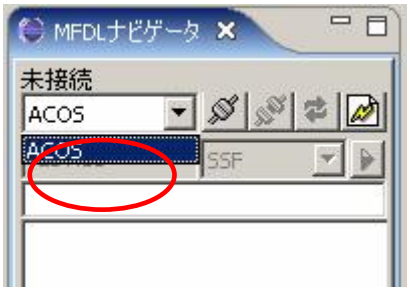
（1）ACOS ホスト上のライブラリから MFDL ソースを取得する場合

①メニューバーから、「ウィンドウ」→「パースペクティブを開く」→「MFDL 移行」を選択して、「MFDL 移行」パースペクティブを開きます。

MFDL ソースを追加するプロジェクトを選択します。



②MFDL ナビゲータから 2.2.2 で設定した設定名を選択します。



③「接続」ボタンを押して ACOS ホストに接続します。

④接続した ACOS ホストのライブラリ上にあるファイルが表示されます。

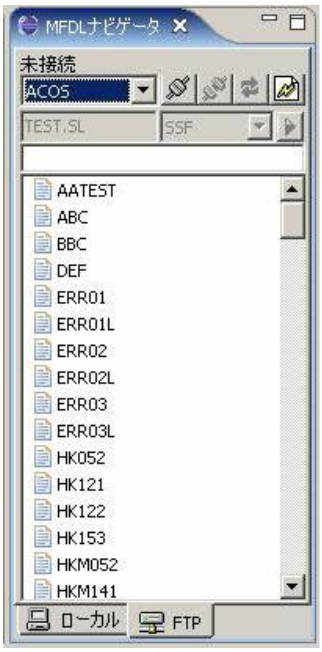
MFDL ナビゲータには他にも以下の機能があります。

 ACOS ホストから切断します。

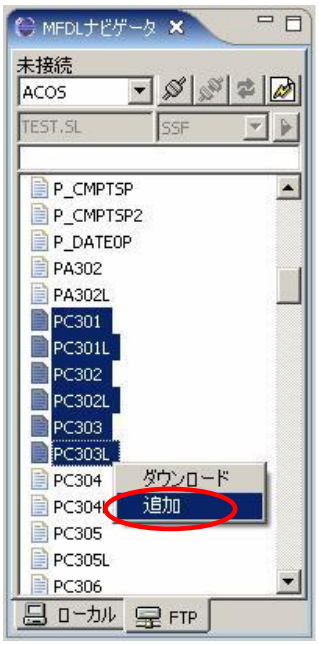
 ファイル一覧を最新の情報に更新します。

 FTP の設定を行ないます。

 ライブラリ名欄（図では TEST.SL が指定されています）で指定したライブラリに変更します。



⑤取得する MFDL 定義ファイルを選択し、右クリックして「追加」を選択します。

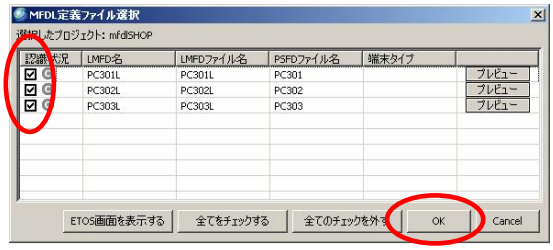


⑥選択した MFDL 定義ファイルから ETOS 画面を定義する LMFD ファイルと PSFD ファイルのペアが抽出、表示されます。

ここで「プレビュー」ボタンを押すと、定義された ETOS 画面を確認することができます。

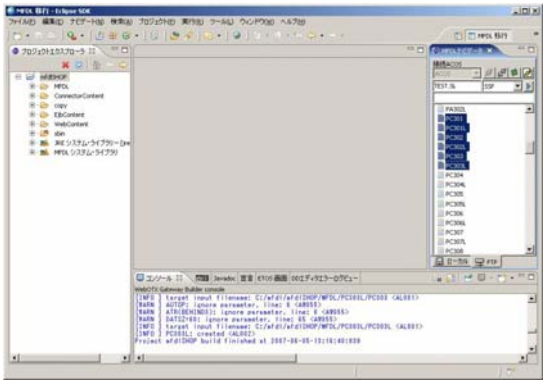
⑦右の図は ETOS 画面の表示例です。

⑧プロジェクトに追加する MFDL にチェックし、「OK」ボタンを押下します。



⑨プロジェクトに MFDL 定義ファイルが追加されます。
「自動的にビルド」の設定が有効になっている場合は、Web アプリケーションのソースの生成とビルドも行なわれます。

※「自動的にビルド」の設定を有効にしていない場合は、「プロジェクト」メニューから「プロジェクトのビルド」を実行すると、Web アプリケーションのソースのビルドを行ないます。

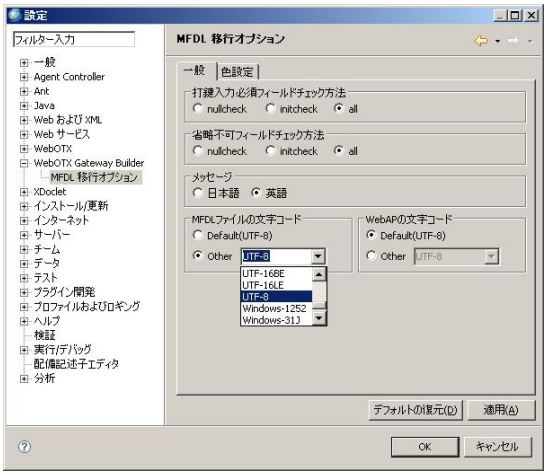


（2）ローカル環境から MFDL ソースを取得する場合

①MFDL 移行プロジェクトのプロパティの「MFDL ファイルの文字コード」にローカル環境にある MFDL ファイルの文字コードを指定します。

指定方法は「3.2.3プロパティの設定」を参照してください。

※ この値は、プロジェクト内で一意になります。プロジェクトで使用する全ての MFDL ソースが、同じ文字コードであることを確認してください。



②MFDL ナビゲータでローカルタブを選択し、ローカルディスク上に作成した MFDL ファイルを選択し、右クリックして「ファイルの追加」を選択します。

③「MFDL 定義ファイル選択」画面が表示されます。

以降は、(1)と同じ操作を行います。

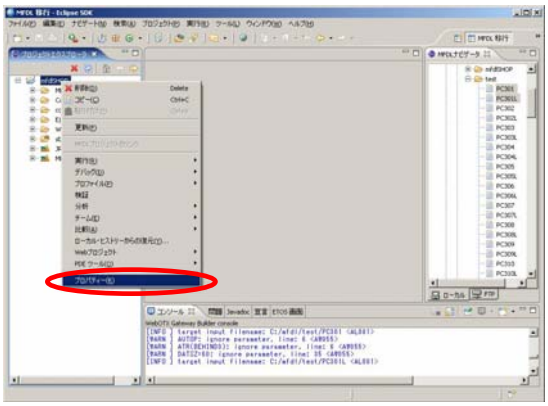
3.2.3.プロパティの設定

Gateway Builder では、MFDL 移行プロジェクト単位、または画面単位にプロパティを設定できます。

（1）MFDL 移行プロジェクトのプロパティ

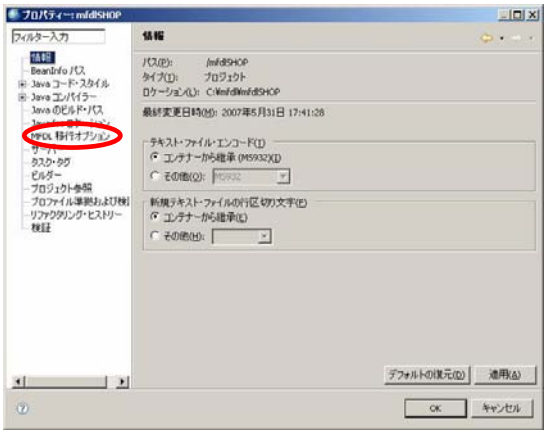
MFDL移行プロジェクト単位で有効になるプロパティを指定します。本設定はDeveloper’s Studioの設定より優先します。

①プロジェクトを選択し、右クリックしてプロパティを選択します。



②プロジェクトのプロパティが表示されます。

MFDL 移行オプションを選択します。



③「MFDL 移行オプション」画面が表示されます。生成する Web アプリケーションに関する指定を行います。

「プロジェクト固有の設定を利用する」をチェックします。

④プロジェクト固有の設定が可能になります。

「一般」タブでは以下の項目が指定できます。

「打鍵入力必須フィールドチェック方法」:

ETOS 端末で入力フィールドに打鍵入力しないと入力を受け付けないフィールド（MFDL 定義の PFLD の入出力属性に、IN または IOF を指定したフィールド）のチェック方法を指定します。

「nullcheck」:

入力フィールドに値が指定されていないときにエラーとします。

「initcheck」:

入力フィールドの値が初期値と同じ値のときにエラーとします。

「all 」:

上記両方のチェックを行います（既定値）。

「省略不可フィールドチェック方法」:

ETOS 端末で入力フィールドに値が設定されていないと入力を受け付けないフィールド（MFDL 定義の LFLD に OPT(DESIG)を指定したフィールド）のチェック方法を指定します。

各指定値の意味は、「打鍵入力必須フィールドチェック方法」と同じです。

「メッセージ」:

Web アプリケーション動作時のメッセージや developer’s Studio のコンソールに出力するメッセージを、英語または日本語から選択できます。

既定値は英語です。

「MFDL ファイルの文字コード」:

MFDL ファイルの文字コードを指定します。既定値は UTF-8 です。

MFDL ナビゲータの FTP 機能により MFDL ファイルを取得した場合は、MFDL ファイルの文字コードは UTF-8 となっているため変更不要ですが、ローカル環境上の MFDL ソースの文字コードが UTF-8 以外になっている場合は、文字コードを指定する必要があります。

シフト JIS で「①」などの機種依存文字を含む場合は、”Windows-31J”を指定してください。

メニュー以外にも Java でサポートされる文字コードを直接指定できます。

「WebAP の文字コード」:

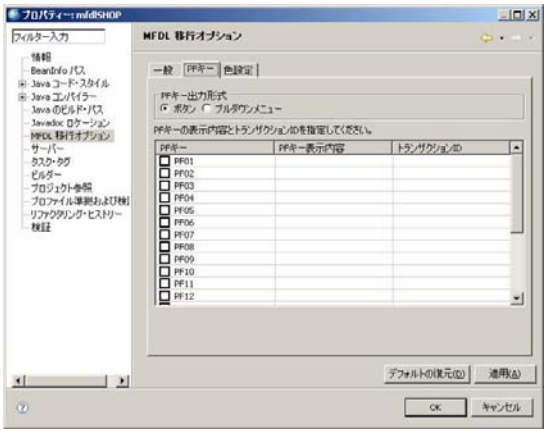
生成する Web アプリケーションのエンコードを指定します。既定値は UTF-8 です。ブラウザのエンコードおよび使用する文字の種類に応じて、文字コードを指定します。

メニュー以外にも Java でサポートされる文字コードを直接指定できます。

「全てのファイルを上書きする」

利用者がカスタマイズしたファイルを含めた全てのファイルを置き換える場合に指定します。

⑤「PF キー」タブでは、ETOS 端末で PF キーに割り当てていた機能を、Web アプリケーションの画面上のボタンやプルダウンメニューとして実現するための指定を行ないます。



「PF キー出力形式」:

PF キー表示の方法を選択します。

「ボタン」または「プルダウンメニュー」が指定できます。既定値は「ボタン」です。

「PF キー」:

Web の画面に表示する PF キーを選択します。

※ 本項目をチェックする場合、「PF キー表示内容」、または、「トランザクションID」の指定が必須です。本項目をチェックしない場合、以下の項目の指定は無視されます。

「PF キー表示内容」:

PF キーに対応するボタンの表示内容を指定します。

「トランザクションID」:

トランザクションを実行する場合に対応するトランザクションIDを指定します。

⑥「色設定」タブでは文字について ETOS 定義色と画面の表示色のマッピングを変更できます。

「プロジェクト固有の設定を利用する」:

既定値以外の設定を行なう場合にチェックします。

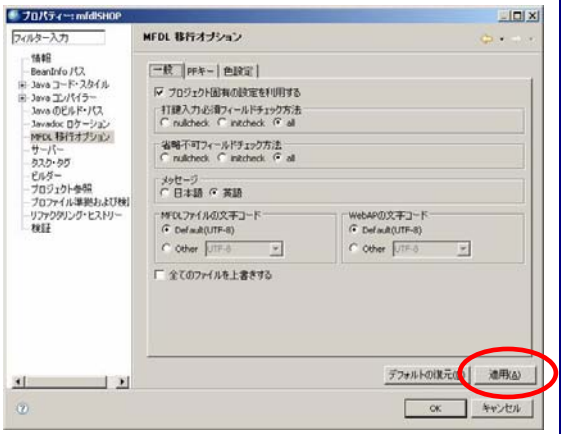
「PSFD 指定色」:

MFDL 定義の PSFD ソースで指定した ETOS 画面の表示色。マッピングを変更する場合にチェックします。

「Web ブラウザでの表示色」:

Web アプリケーション画面の表示色です。

⑦設定が終われば、「適用」ボタンを押下します。



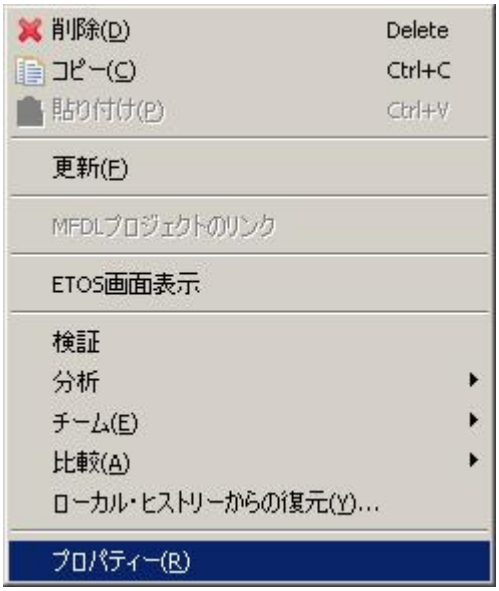
Web アプリケーションのソースの生成とリビルドを行ないます。

設定を既定値に戻す場合は「デフォルトの復元」ボタンを押下します。

（2）画面のプロパティ

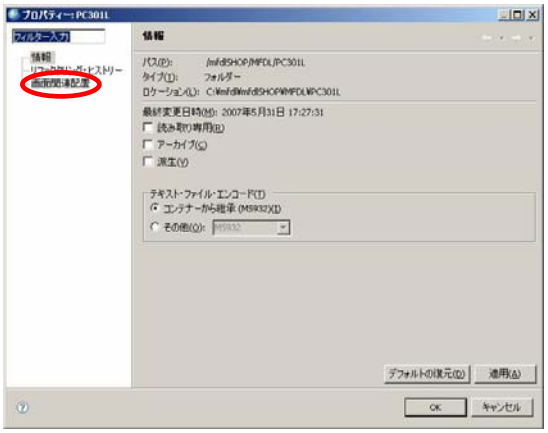
画面単位で有効になるプロパティを設定します。本設定はMFDL移行プロジェクトのプロパティより優先します。

①プロジェクトの「MFDL」フォルダから画面名に対応するフォルダを選択して、右クリックして「プロパティー」を選択します。



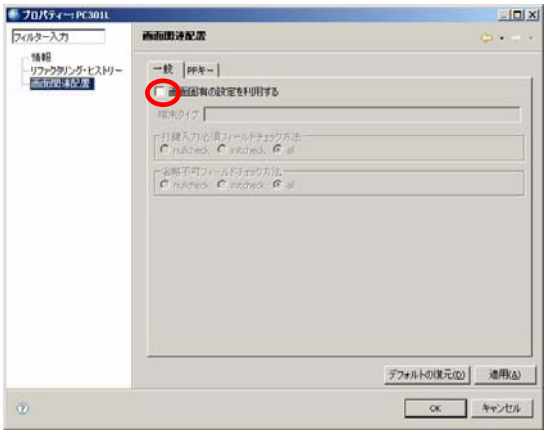
②画面のプロパティーが表示されます。

「画面関連配置」を選択します。

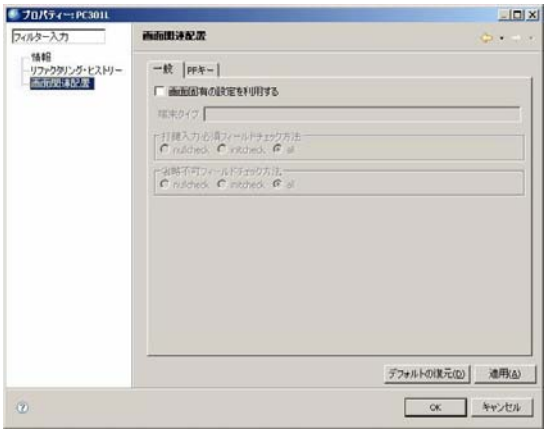


③「画面関連配置」画面が表示されます。

「画面固有の設定を利用する」をチェックします。



④「一般」タブでは以下の項目を指定できます。



「装置タイプ」:

MFDL 定義の PSFD 内に複数の端末タイプ画面があるときに、どの端末タイプの画面を生成するのかを指定します。

既定値は、MFDL ソースの先頭の定義になります。

「打鍵入力必須フィールドチェック方法」:

ETOS 端末で入力フィールドに打鍵入力しないと入力を受け付けないフィールド（MFDL 定義の PFLD の入出力属性に、IN または IOF を指定したフィールド）のチェック方法を指定します。

「nullcheck」:

入力フィールドに値が指定されていないときにエラーとします。

「initcheck」:

入力フィールドの値が初期値と同じ値のときにエラーとします。

「all 」:

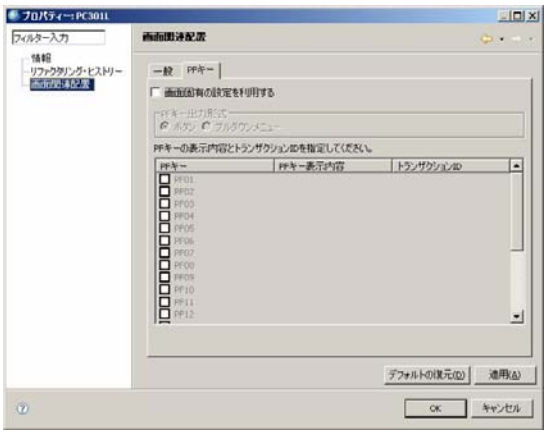
上記両方のチェックを行います（既定値）。

「省略不可フィールドチェック方法」:

ETOS 端末で入力フィールドに値が設定されていないと入力を受け付けないフィールド（MFDL 定義の LFLD に OPT(DESIG)を指定したフィールド）のチェック方法を指定します。

各指定値の意味は、「打鍵入力必須フィールドチェック方法」と同じです。

⑤「PF キー」タブでは、ETOS 端末で PF キーに割り当てていた機能を Web アプリケーションの画面上のボタンやプルダウンメニューとして実現することができます。



「PF キー出力形式」:

PF キー表示の方法を選択します。

「ボタン」または「プルダウンメニュー」が指定できます。既定値は「ボタン」です。

「PF キー」:

Web の画面に表示する PF キーを選択します。

※ 本項目をチェックする場合、「PF キー表示内容」、または、「トランザクションID」の指定が必須になります。本項目をチェックしない場合、以下の項目の指定は無視されます。

「PF キー表示内容」:

PF キーに対応するボタンの表示内容を指定します。

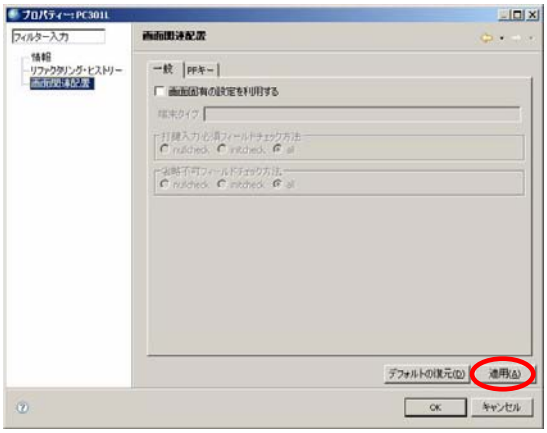
「トランザクションID」:

トランザクションを実行したい場合に対応するトランザクションIDを指定します。

⑥設定が終われば、「適用」ボタンを押下します。

Web アプリケーションのソースの生成とリビルドを行ないます。

設定を既定値に戻す場合は「デフォルトの復元」ボタンを押下します。

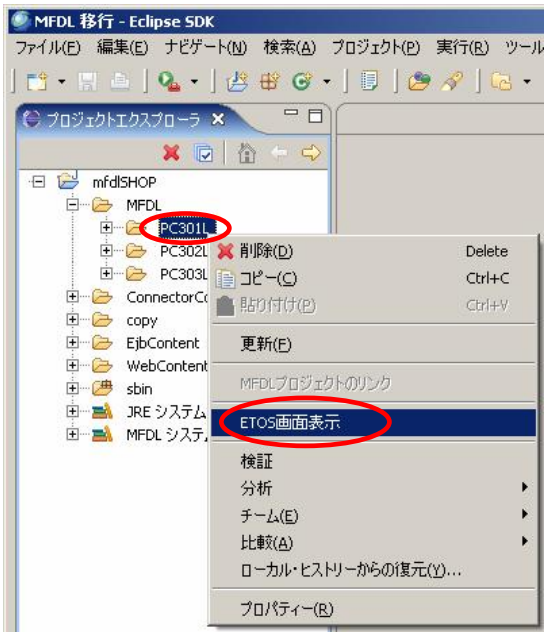


3.2.4.ETOS 画面の表示

MFDL ソースで定義した ETOS 画面を表示する方法を説明します。

①プロジェクトの「MFDL」フォルダから画面名に対応するフォルダを選択します。

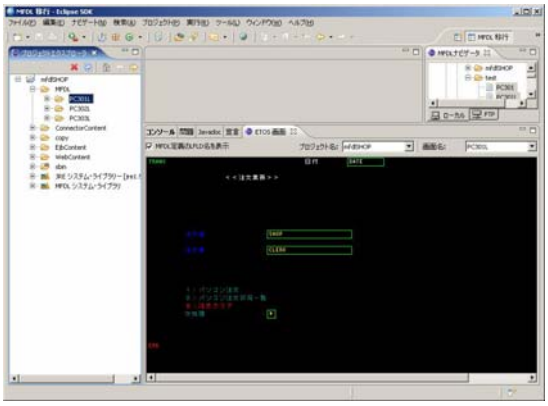
右クリックして「ETOS 画面表示」を選択すると ETOS 画面を表示します。



②ETOS 画面の表示例です。

「MFDL 定義の LFLD 名を表示」:

MFDL ソースの LFLD で定義したフィールド名を表示する場合にチェックします。



3.3.コネクタアプリケーションの作成と配備

本節では、MFDL 移行プロジェクトで作成したファイル(cdd)を使ってコネクタアプリケーションを作成する方法、およびコネクタアプリケーションの配備方法について説明します。

※ ここからの作業は MFDL 移行パースペクティブと J2EE パースペクティブを用いた作業となります。MFDL 移行パースペクティブに切り替えてからの作業をおすすめします。

3.3.1.コネクタアプリケーションの作成

以下の手順によりコネクタアプリケーションを作成します。

(1) コネクター・プロジェクトの作成

(2) MFDL 移行プロジェクトのリンク

(3) 画面のリンク

(4) 確認

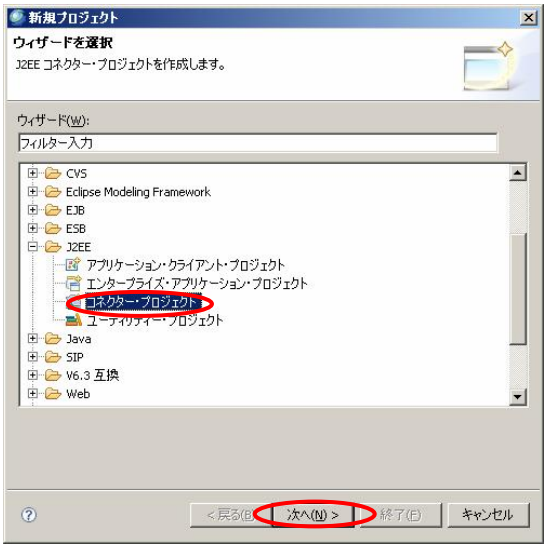
コネクタアプリケーションの作成について詳しく知りたい場合は、次のマニュアルを参照してください。

WebOTX マニュアル アプリケーション開発ガイド
第 4 章 プログラミング・開発(J2EE)
4.6. コネクタアプリケーション(pdf 形式)

(1) コネクター・プロジェクトの作成

① メニューバーから、「ファイル」→「新規」→「プロジェクト」を選択して、「新規プロジェクト」画面を開きます。

② 「J2EE」→「コネクター・プロジェクト」を選択して、「次へ」ボタンを押します。



③ 次の値を指定します。

「プロジェクト名」:
任意の名称

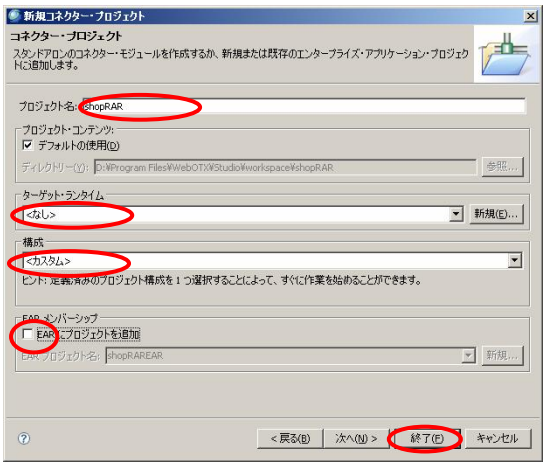
例:shopRAR

「ターゲット・ランタイム」:
<なし>

※ コネクタアプリケーションでは、
Target runtime の指定は不要です。

「構成」:
<カスタム>

※既定値の<カスタム>でよいです。



指定する必要はありません。

④「終了」ボタンを押下します。

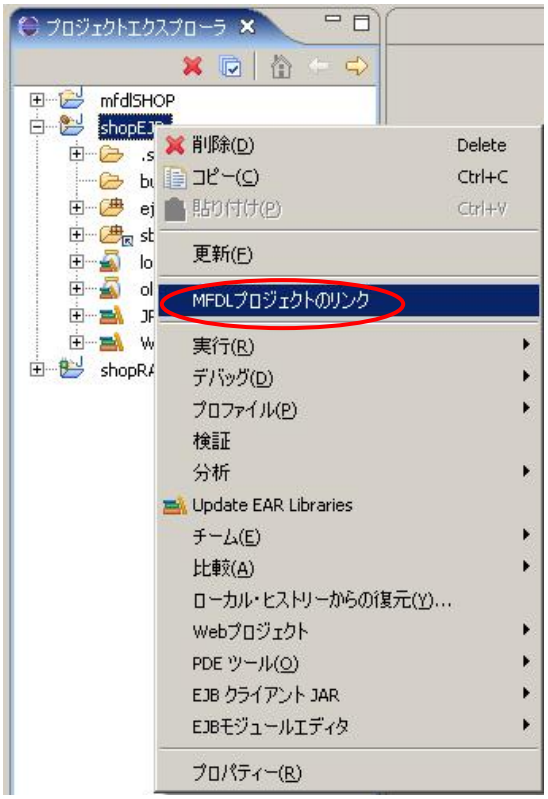
※「次へ」ボタンを押下すると、指定した構成のプロジェクト・ファセットを設定できます。

(2) MFDL 移行プロジェクトのリンク

コネクタアプリケーションの動作に必要な jar ファイルおよび設定ファイルの追加と MFDL 移行プロジェクトのリンクを行ないます。

※MFDL 移行プロジェクトのリンクは MFDL 移行パースペクティブでの作業となります。MFDL 移行パースペクティブに切り替えてから作業してください。

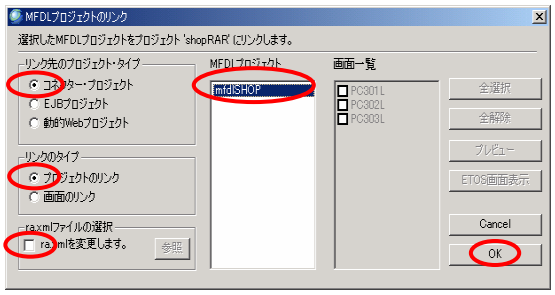
① プロジェクトエクスプローラでコネクタアプリケーションを選択し、右クリックメニューから「MFDL プロジェクトのリンク」を押下します。



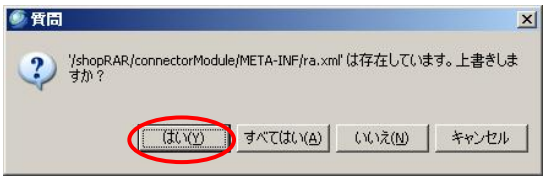
② 「リンク先のプロジェクト・タイプ」にコネクター・プロジェクトを「リンクのタイプ」にプロジェクトのリンクをチェックし、「MFDL プロジェクト」を選択します。

利用する ra.xml ファイルを指定する場合、チェックして ra.xml ファイルを選択します。チェックしないと、OLF/TP Adapter が提供するテンプレートをコピーして使用します。

③ 「OK」ボタンを押下します。



④ ②で選択した ra.xml をプロジェクト内にコピーするため、「上書きしますか？」の質問ダイアログが表示されます。ra.xml をあらかじめ準備していないときは、「はい」を押下します。



設定ファイル(ra.xml)には、コネクタアプリケーションが接続する相手ホストの情報などが含まれます。

「3.3.2 コネクタアプリケーションの設定」で、プロジェクトごとに接続先ホストの情報などを変更できますが、共通の内容については、あらかじめ OLF/TP Adapter が提供するテンプレートの ra.xml を変更しておくことで、作業が簡略化できます。

テンプレートの ra.xml は以下に格納されています。

<WebOTX install dir>/Studio/plugins/com.nec.webotx.olftpdev_X.X.X.X/template/olftp/ra.xml
「X.X.X.X」は OLF/TP Adapter のバージョンです。

あらかじめ変更しておくとよい ra.xml の要素を以下に示します。

```
<config-property-name>ServerName
<config-property-type>java.lang.String
<config-property-value>iPX9000UT
```

↑
接続先のホスト名を指定します。「2.1.1(2) OlfAdapter.xml の設定」で定義した RmtName を指定します。

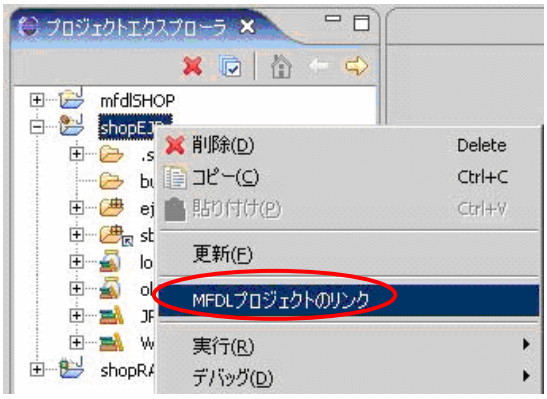
```
<config-property-name>OLFConfigFile
<config-property-type>java.lang.String
<config-property-value>C:/WebOTX/Adapter/OLFTP/Run/conf/OlfAdapter.ini
```

↑
WebOTX のインストール先を変更している場合は、インストール先に合わせて変更します。

(3) 画面のリンク

(2)の代わりにMFDL移行プロジェクトの必要な画面を選択して、リンクを行なうこともできます。

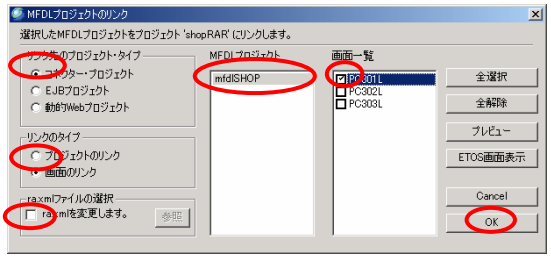
① プロジェクトエクスプローラでコネクタアプリケーションを選択し、右クリックメニューから「MFDL プロジェクトのリンク」を押下します。



② 「リンク先のプロジェクト・タイプ」にコネクター・プロジェクトを「リンクのタイプ」に画面のリンクを選択します。

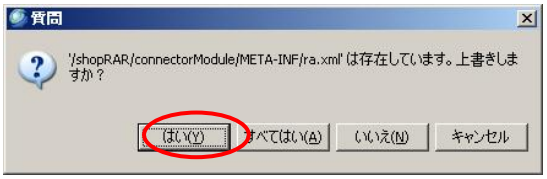
「MFDL プロジェクト」からリンクする MFDL プロジェクトを選択すると「画面一覧」に選択したプロジェクトの画面一覧が表示されます。「画面一覧」からリンクする画面を選択します。

利用する ra.xml ファイルを指定する場合、チェックして ra.xml ファイルを選択します。チェックしないと、OLF/TP Adapter が提供するテンプレートをコピーして使用します。



③ 「OK」ボタンを押下します。

④ ②で選択した ra.xml をプロジェクト内にコピーするため、「上書きしますか？」の質問ダイアログが表示されます。ra.xml をあらかじめ準備していないときは、「はい」を押下します。



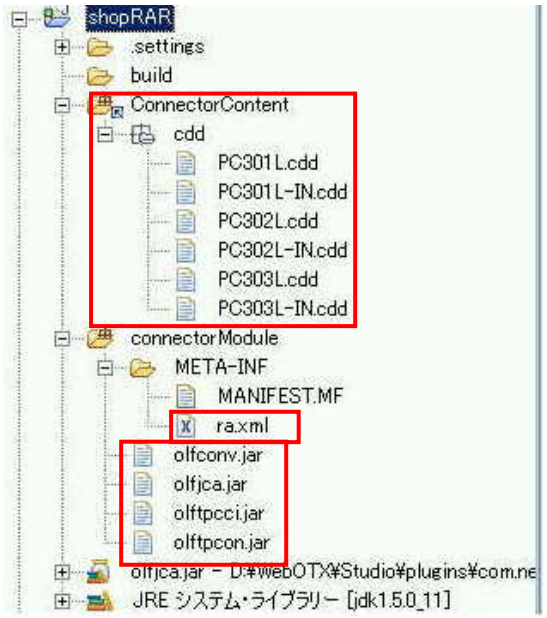
(4) 確認

① 作業が完了したら、右図のようにファイルが格納されていることを確認してください。

・ConnectorContent/cdd/
　<画面名>.cdd、
　<画面名>-IN.cdd

・connectorModule/META-INF/
　ra.xml

・connectorModule/
　olfconv.jar、
　olfjca.jar、
　olftpcci.jar、
　olftpcon.jar



3.3.2.コネクタアプリケーションの設定

ここでは、コネクタアプリケーションの設定を行います。ra.xml という設定ファイルに各種設定を行います。

※コネクタアプリケーションの設定は、J2EE パースペクティブで作業します。

ra.xml の詳細については、次のマニュアルを参照してください。

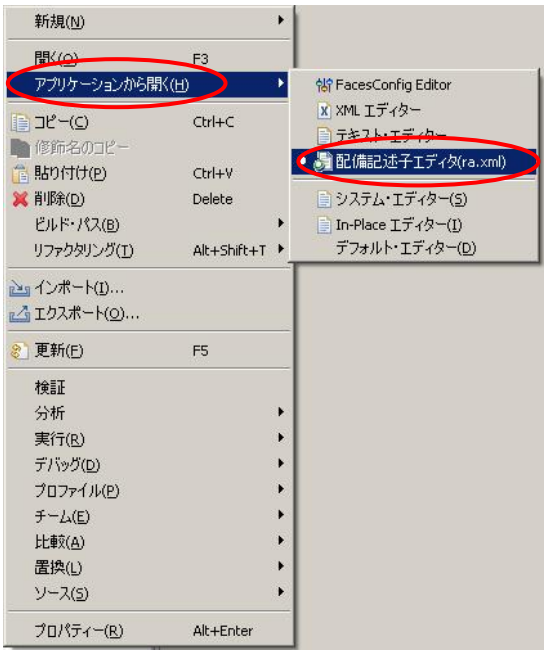
WebOTX　OLF/TP Adapter　運用ガイド
#　4. WebOTX OLF/TP Adapter 実行環境の運用
#　 4.3. 設定
#　　4.3.3. 配備記述子(ra.xml)

(1)　ra.xml の設定

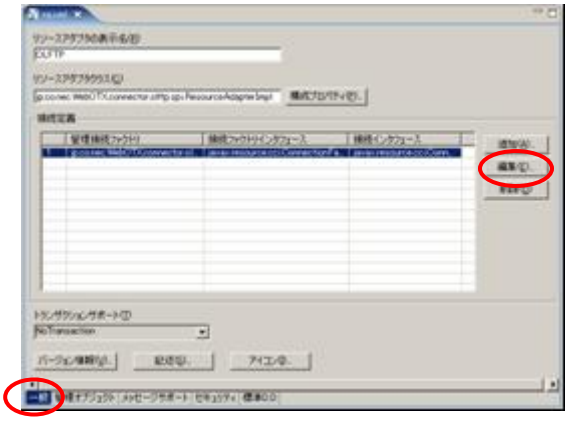
ここでは、通信相手となる ACOS ホストを指定します。

※ なお、3.3.1 の作業で ra.xml を変更済みの場合は、この作業を省くことができます。

① <RAR project dir>/connectorModule/META-INF/ra.xml を右クリックして「アプリケーションから開く」→「配備記述子エディタ」を選択します。



② 「一般」タブの「接続定義」→「編集」ボタンを押下して接続定義の編集画面を開きます。

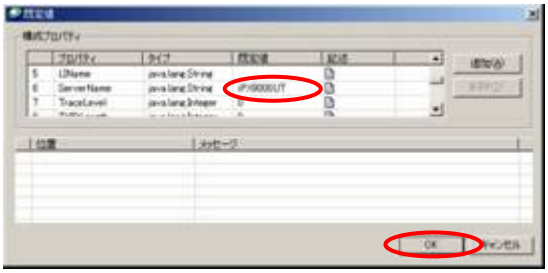


③「接続定義の編集」画面で、「構成プロパティ」ボタンを押下して既定値の設定画面を開きます。

④「構成プロパティ」の「ServerName」に、「2.1.1.(2) OlfAdapter.xml の設定」で RmtName に定義した ACOS ホスト名を指定します。

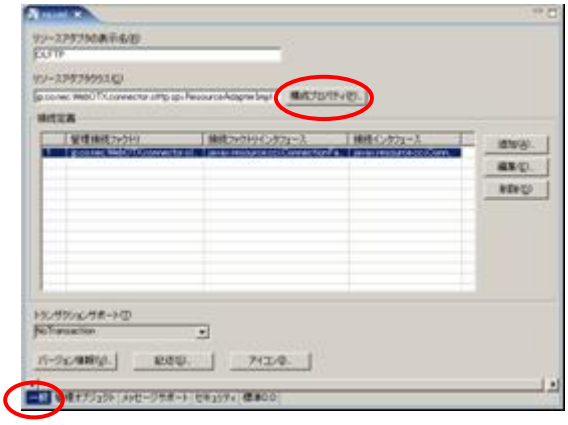
※ この設定で変更される ra.xml の要素を以下に示します。

<config-property-name>**ServerName**
<config-property-type>java.lang.String
<config-property-value>**指定したホスト名**

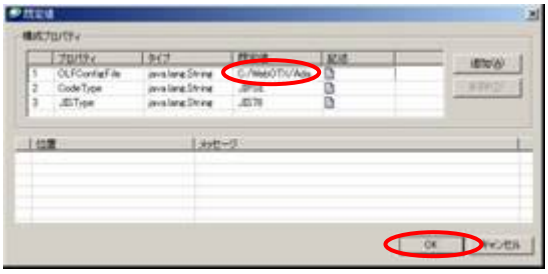


⑤「OK」ボタンを 2 回押下して画面を閉じます。

⑥「一般」タブの「構成プロパティ」ボタンを押下して「既定値」画面を開きます。



⑦「構成プロパティ」の「OLFConfigFile」の設定値を WebOTX インストール環境に合わせます。

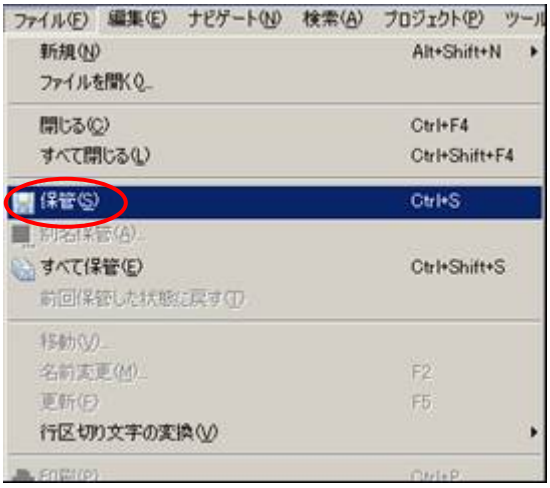


※ この設定で変更される ra.xml の要素を以下に示します。

<config-property-name>**OLFConfigFile**
<config-property-type>java.lang.String
<config-property-value>**C:/WebOTX/**Adapter/OLFTP/Run/conf/OlfAdapter.ini

⑧「OK」ボタンを押下して画面を閉じます。

⑨「ファイル」メニュー-「保管」を選択して ra.xml の内容を更新します。



3.3.3.コネクタアプリケーションの配備

ここでは、コネクタアプリケーションの配備について説明します。

※コネクタアプリケーションの配備は、J2EE パースペクティブで行います。

以下の手順でコネクタアプリケーションの配備を行います。

(1) コネクタアプリケーションのビルド

(2) コネクタアプリケーションのパッケージ

(3) コネクタアプリケーションの配備

(4) コネクションプールの登録

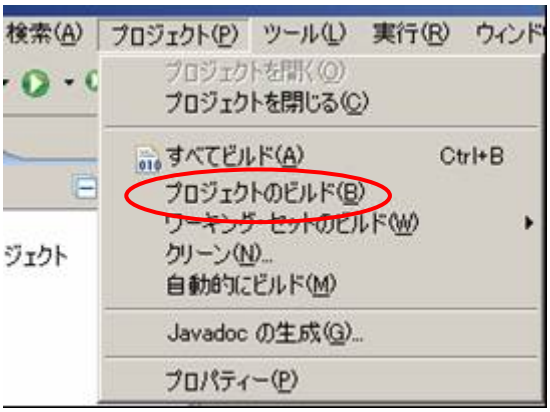
(5) コネクタリソースの登録

(1) コネクタアプリケーションのビルド

配備ツールで配備を行うためには、作成したコネクタアプリケーションをビルドする必要があります。

「自動的にビルド」が設定されている場合は、本作業は不要です。「自動的にビルド」が設定されていない場合は次の手順でビルドを行ってください。

① メニューバーから、「プロジェクト」→「プロジェクトのビルド」を実行します。



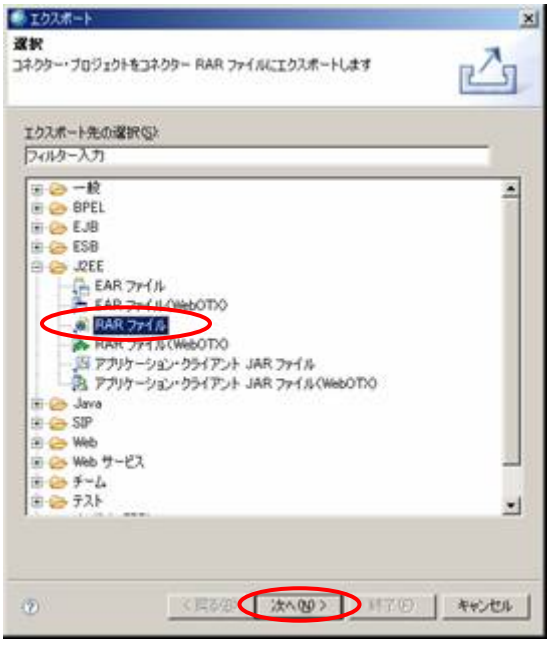
(2) コネクタアプリケーションのパッケージ

配備ツールで配備を行うためには、作成したコネクタアプリケーションをパッケージする必要があります。

① 「ファイル」メニューから「エクスポート」を実行します。

② 「J2EE」→「RAR ファイル」を選択し、「次へ」ボタンを押下します。

※J2EE パースペクティブでは、コネクター・プロジェクトの右クリックメニュー「エクスポート」→「RAR ファイル」からも作業を行うことができます。



③ 次の値を指定します。

「コネクター・モジュール」:
「3.3.1(1)コネクター・プロジェクトの作成」で作成したプロジェクト名

例:shopRAR

「宛先」:
保存するファイル名

例:C:¥mfdl¥package¥shopRAR.rar

※ ファイル名は、実行環境で一意の名前にする必要があるため、プロジェクト名と同じにすることを推奨します。

※ ファイル名にパス指定をしない場合、<WebOTX install dir>/Studio 直下に作成されます。専用フォルダーを用意しフルパス名で指定することを推奨します。

④ 「終了」ボタンを押下して、RAR ファイルを作成します。

(3) コネクタアプリケーションの配備

アプリケーションを配備するためには、以下のようにいくつかの方法があります。

- 配備ツールによる配備
- 運用管理コマンドによる配備
- パッケージ化しないアプリケーションの配備

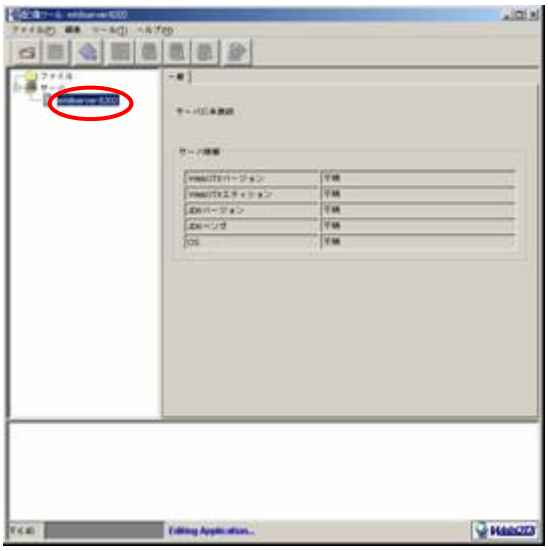
ここでは、「配備ツール」を利用した手順を説明します。

運用管理コマンドによる配備方法については、次のマニュアルを参照してください。

```
# WebOTX マニュアル  運用編
#  アプリケーション配備
#    2.2.4. コマンドによる配備・配備解除
```

① 配備ツールを起動します。

② サーバ名を選択し、「右クリック」-「ドメイン一覧の取得」を選択します。



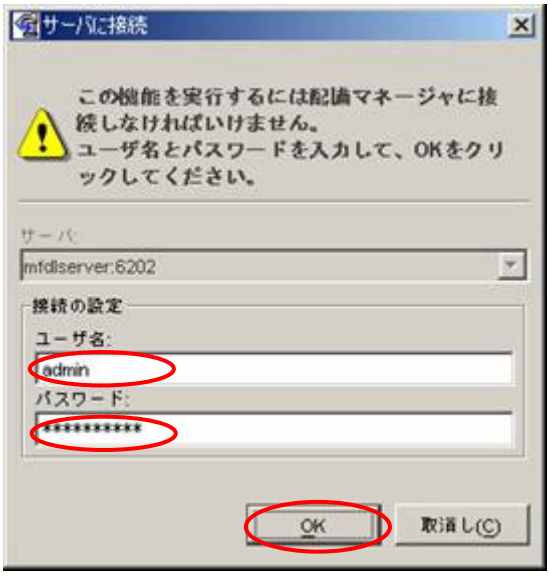
③ ユーザ名とパスワードを入力してサーバに接続(ログイン)を行います。

これで配備可能な状態となります。

※ 接続(ログイン)を行うには WebOTX サービスが起動している必要があります。通常は自動起動していますが、手動に設定している場合は事前に起動してください。
ここで必要な WebOTX サービスは以下です。

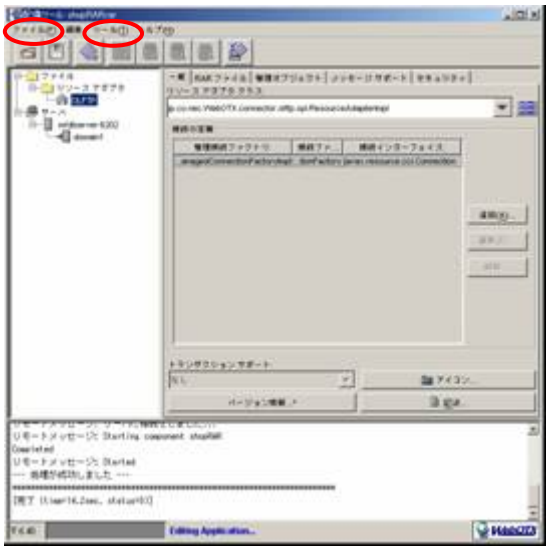
WebOTX AS Agent Service
WebOTX LicServer

WebOTX AS のエディションによっては、存在しないサービスがあります。



④ 「ファイル」-「開く」を選び、パッケージした RAR ファイルを選択します。

⑤ 「ツール」-「配備」を選択して、配備を実行します。



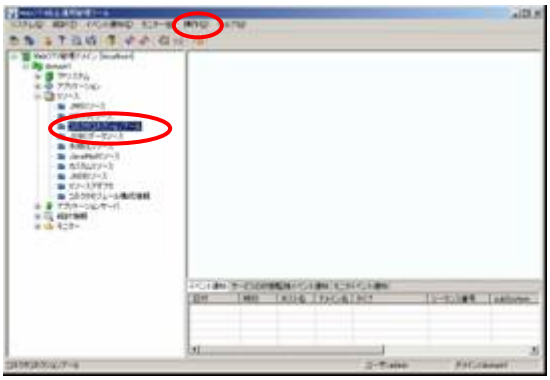
(4) コネクションプールの登録

コネクタアプリケーションを利用するためには、コネクションプールおよびコネクタリソースの作成が必要になります。ここでは、「統合運用管理ツール」を利用した手順を説明します。

① 統合運用管理ツールを起動し、ドメインへ接続します。

② 「domain1」→「リソース」→「コネクタコネクションプール」を選択します。

③ メニューバーから、「操作」→「コネクションプールの登録」を選択します。



④ 次の値を指定します。

「リソースアダプタ名」:
配備した RAR ファイル名の拡張子(.rar)を除いた名前

例:shopRAR

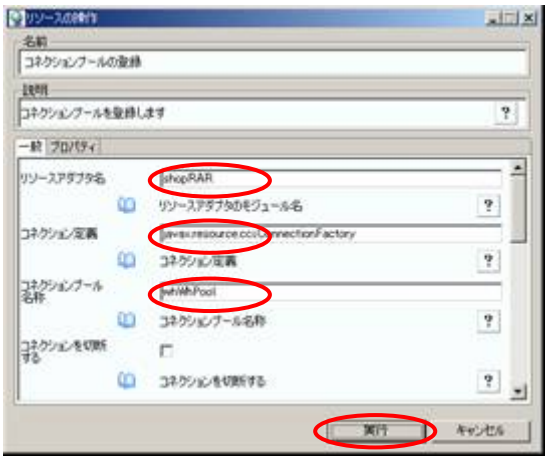
「コネクション定義」:
javax.resource.cci.ConnectionFactory

「コネクションプール名称」:
実行環境で一意の名称

例:whWhPool

※「コネクションプール名称」は、MFDL 移行プロジェクトの
パッケージ名+"WhPool"
とすることを推奨します。

この名称は、(5) コネクタリソースの登録で使用します。



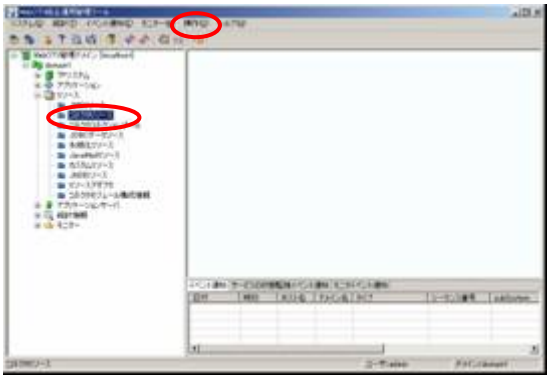
⑤ 「実行」ボタンを押下します。

(5) コネクタリソースの登録

① 統合運用管理ツールを起動し、ドメインへ接続します。

② 「domain1」→「リソース」→「コネクタリソース」を選択します。

③ メニューバーから、「操作」→「コネクタリソースの登録」を選択します。



④ 次の値を指定します。

「プールネーム」:
「(4) コネクションプールの登録」で指定したコネクションプールの名称

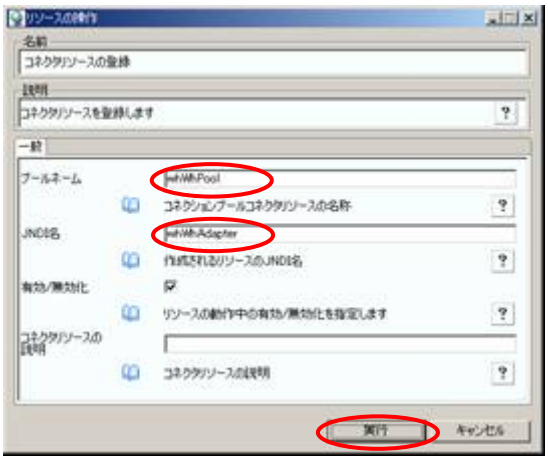
例:whWhPool

「JNDI 名」:
実行環境で一意の名称

例:whWhAdapter

※「JNDI 名」は、MFDL 移行プロジェクトの
パッケージ名＋"WhAdapter"
とすることを推奨します。

この名称は、「3.4.2(1) ejb-jar.xml, nec-ejb-jar.xmlの設定」で使用します。



⑤ 「実行」ボタンを押下します。

3.3.4.コネクタアプリケーションの削除と再配備

MFDL 移行プロジェクトで画面を追加した場合など、コネクタアプリケーションが変更されたときに再配備を行います。

コネクタアプリケーションを再配備するときは、関連するコネクタリソースとコネクションプールも再作成する必要があります。以下の手順で行います。

(1) コネクタリソースの削除

(2) コネクションプールの削除

(3) コネクタアプリケーションの配備解除

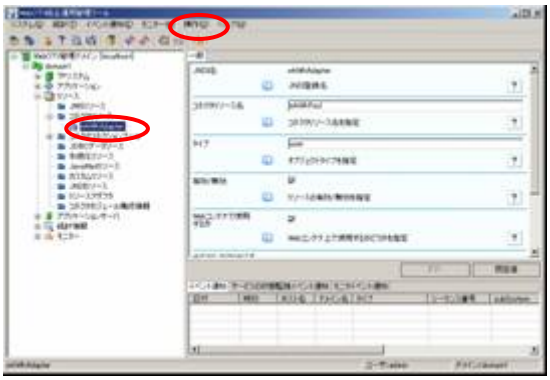
(4) コネクタアプリケーションの再配備

(1) コネクタリソースの削除

① 統合運用管理ツールを起動し、ドメインへ接続します。

② 「domain1」→「リソース」→「コネクタリソース」→「<対象コネクタリソース>」を選択します。

③ メニューバーから、「操作」→「削除」を選択します。



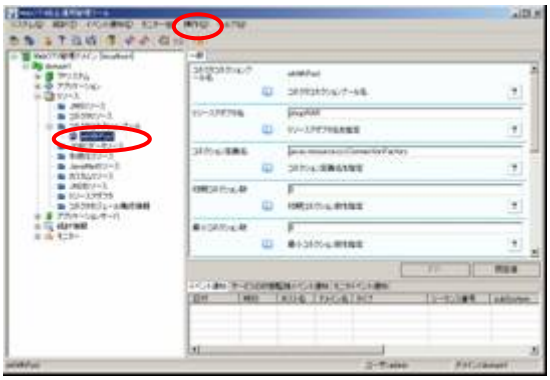
④ リソースの操作画面が表示されますので、「実行」ボタンを押下します。

(2) コネクションプールの削除

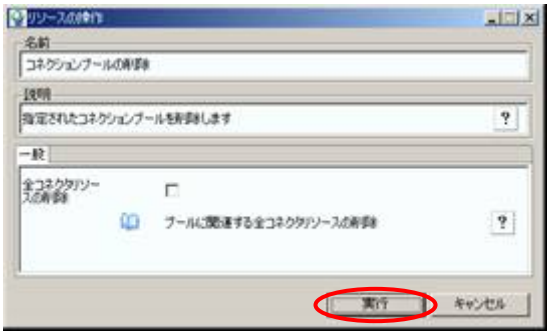
① 統合運用管理ツールを起動し、ドメインへ接続します。

② 「domain1」→「リソース」→「コネクタコネクションプール」→「<対象コネクションプール>」を選択します。

③ メニューバーから、「操作」→「削除」を選択します。



④ リソースの操作画面が表示されますので、「実行」ボタンを押下します。

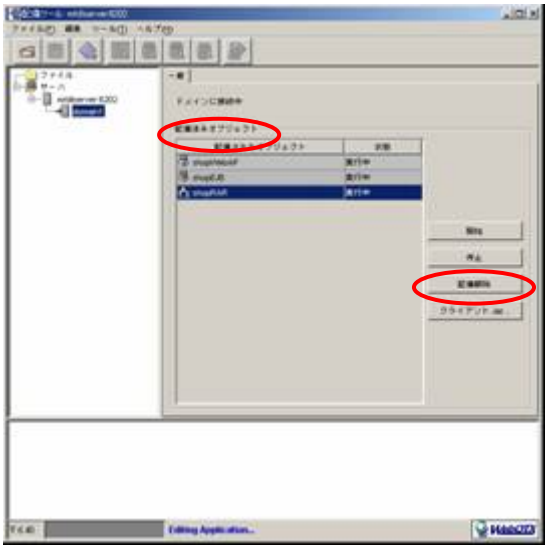


(3) コネクタアプリケーションの配備解除

配備ツールから「配備削除」を選択し、配備解除します。

① 配備ツールを起動し、ドメインへ接続します。

② 配備済みオブジェクトを選択し、「配備解除」ボタンを押下します。



(4) コネクタアプリケーションの再配備

「3.3.3.コネクタアプリケーションの配備」と同様の手順でコネクタアプリケーションを再配備します。

3.4.EJB アプリケーションの作成と配備

本節では、MFDL 移行プロジェクトで作成したファイル(java,xml)を使って EJB アプリケーションを作成する方法、および EJB アプリケーションの配備方法について説明します。

※ ここからの作業は MFDL 移行パースペクティブと J2EE パースペクティブを用いた作業となります。MFDL 移行パースペクティブに切り替えてからの作業をおすすめします。

3.4.1.EJB アプリケーションの作成

以下の手順により EJB アプリケーションを作成します。

(1) EJB プロジェクトの作成

(2) MFDL 移行プロジェクトのリンク

(3) 画面のリンク

(4) 確認

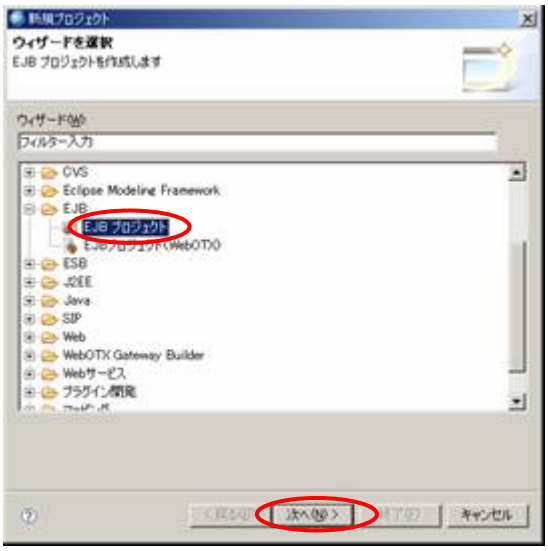
EJB アプリケーションの作成について詳しく知りたい場合、次のマニュアルを参照してください。

WebOTX マニュアル アプリケーション開発ガイド
第 3 章 チュートリアル
3.4. EJB アプリケーション(pdf 形式)

(1) EJB プロジェクトの作成

① メニューバーから、「ファイル」→「新規」→「プロジェクト」を選択して、「新規プロジェクト」画面を開きます。

② 「EJB」→「EJBプロジェクト」を選択して、「次へ」ボタンを押します。



③ 次の値を指定します。

「プロジェクト名」:
任意の名称

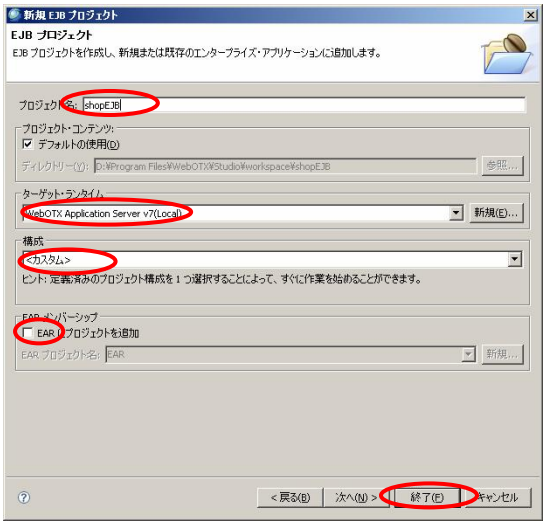
例:shopEJB

「ターゲット・ランタイム」:
WebOTX Application Server v7(Local)

※ EJB アプリケーションの動作に必要な j2ee.jar ランタイムライブラリを指定します。

「構成」:
<カスタム>

※ 既定値の<カスタム>でよいです。



「EAR メンバシップ」:
指定する必要はありません。

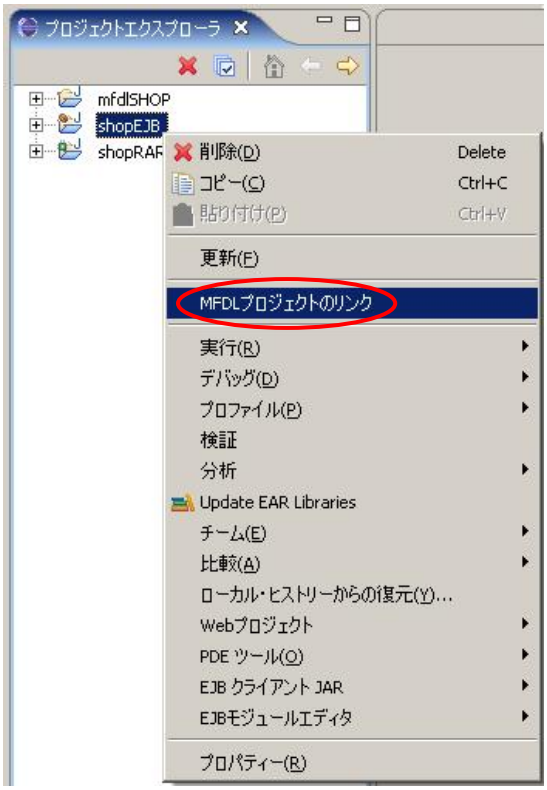
④ 「終了」ボタンを押下します。

※「次へ」ボタンを押下すると、指定した構成のプロジェクト・ファセットを設定できます。

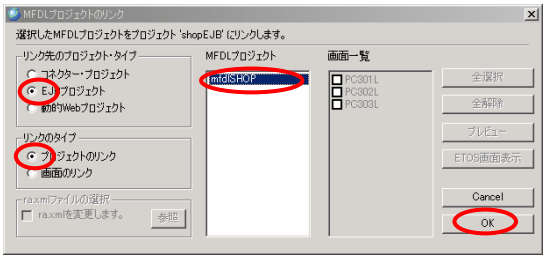
(2) MFDL 移行プロジェクトのリンク

① <プロジェクト名>の右クリックメニューから「MFDL プロジェクトのリンク」を選択します。

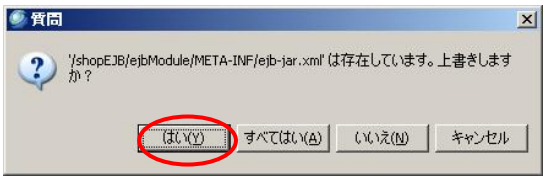
※MFDL 移行プロジェクトのリンクは MFDL 移行パースペクティブでの作業となります。
MFDL 移行パースペクティブに切り替えてから作業してください。



② 「リンク先のプロジェクトタイプ」に EJB プロジェクトを「リンクのタイプ」にプロジェクトのリンクを選択し、「MFDL プロジェクト」にリンクする MFDL プロジェクトを選択して、「OK」ボタンを押下します。



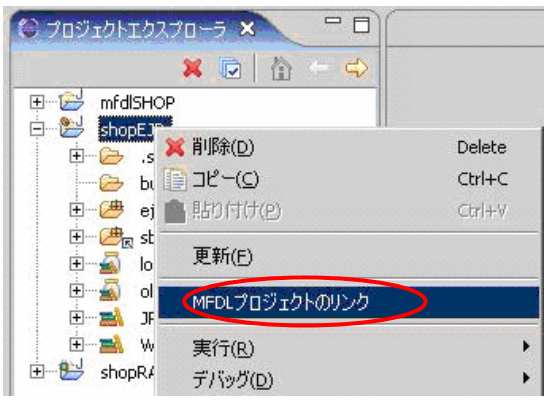
③ 「上書きしますか？」の質問ダイアログが表示されたときは、「はい」を選択します。



(3) 画面のリンク

(2)の代わりにMFDL移行プロジェクトの必要な画面を選択して、リンクを行なうこともできます。

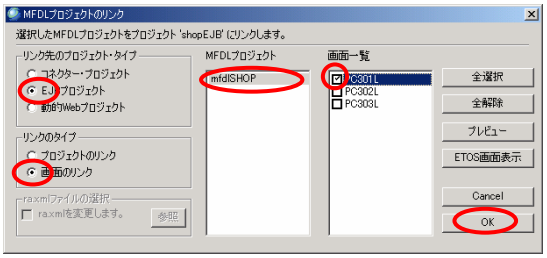
① <プロジェクト名>の右クリックメニューから「MFDL プロジェクトのリンク」を選択します。



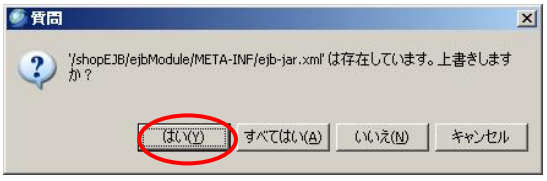
② 「リンク先のプロジェクトタイプ」に EJB プロジェクトを「リンクのタイプ」に画面のリンクを選択します。

「MFDL プロジェクト」からリンクする MFDL プロジェクトを選択すると「画面一覧」に選択したプロジェクトの画面一覧が表示されます。

「画面一覧」からリンクする画面を選択して、「OK」ボタンを押下します。



③ 「上書きしますか？」の質問ダイアログが表示されたときは、「はい」を選択します。



(4) 確認

① 作業が完了したら、右図のようにファイルが格納されていることを確認してください。

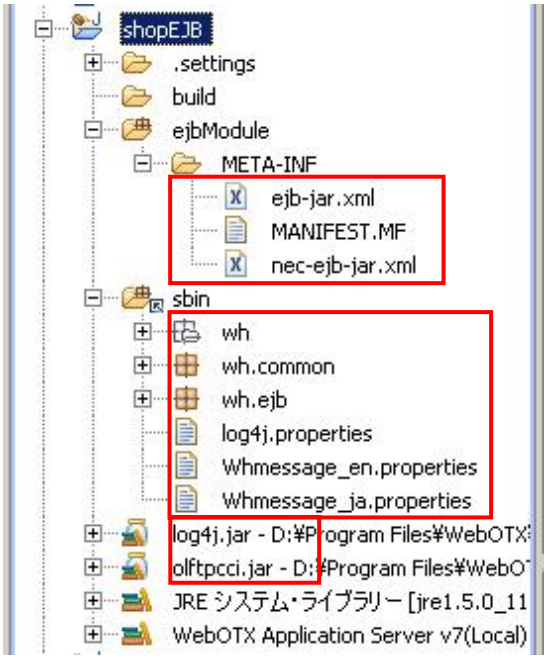
・ejbModule/META-INF/
　ejb-jar.xml
　nec-ejb-jar.xml

・sbin/
　wh.common/
　wh.ejb/
　log4j.properties
　Whmessage_en.properties
　Whmessage_ja.properties

・log4j.jar

・olftpcci.jar

※「wh.common」「wh.ejb」の「wh」は、MFDL移行プロジェクトのパッケージ名を示します。



3.4.2.EJB アプリケーションの設定

ここでは、EJB アプリケーションの設定を行います。ejb-jar.xml、nec-ejb-jar.xml という設定ファイルに各種設定を行います。前記リンク機能を利用したときは、設定が済んでいます。

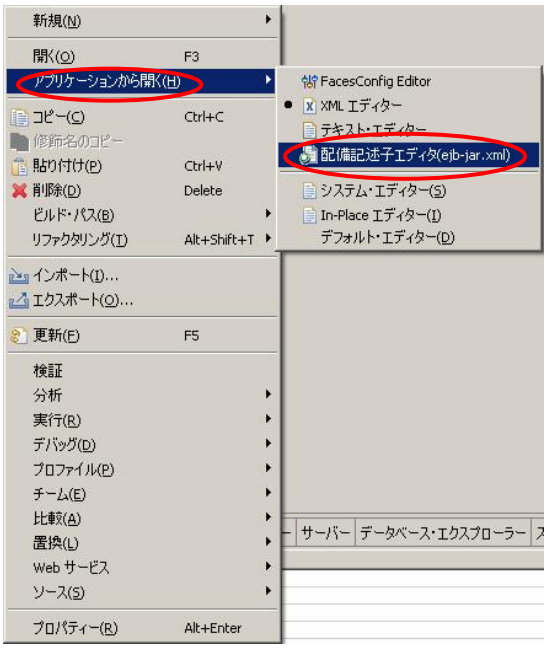
(1) ejb-jar.xml, nec-ejb-jar.xml の設定

ここでは、Web アプリケーションから EJB アプリケーションを呼び出すための情報や、EJB アプリケーションから呼び出すコネクタアプリケーションの情報などを設定します。
以下のような設定項目があります。

- Web アプリケーションから EJB アプリケーションを呼び出すための EJB アプリケーションの名前（EJB アプリケーションの JNDI 名）
- EJB アプリケーションが呼び出すコネクタアプリケーションの名前(コネクタリソースの JNDI 名）

これらの設定は、基本的に変更は不要です。複数の Web アプリケーションで EJB アプリケーションを共用したり、複数の EJB アプリケーションでコネクタアプリケーションを共用するなど、EJB アプリケーションやコネクタアプリケーションの呼び出し関係を変更したい場合に、必要に応じて指定してください。

① <EJB project dir>/ejbModule/META-INF/ejb-jar.xml を右クリックして「アプリケーションから開く」-「配備記述子エディタ」を選択します。

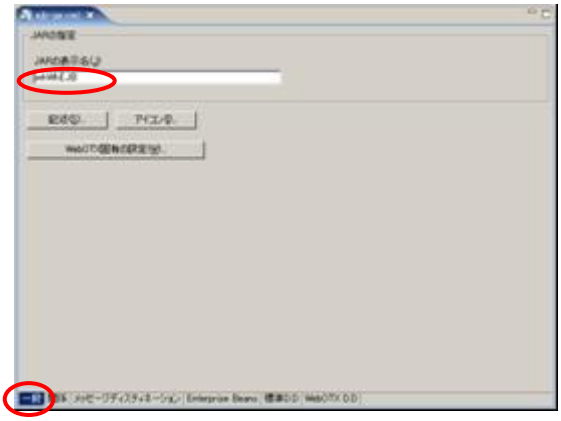


② 「一般」タブの「JAR の表示名」にこのパッケージの名称を指定します。

例：whWhEJB

MFDL 移行プロジェクトで生成した ejb-jar.xml には、以下の値が設定されています。

パッケージ名＋”WhEJB”

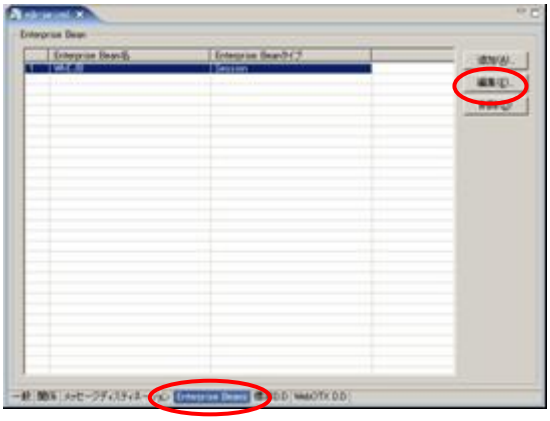


※ この設定で変更される ejb-jar.xml の要素を以下に示します。

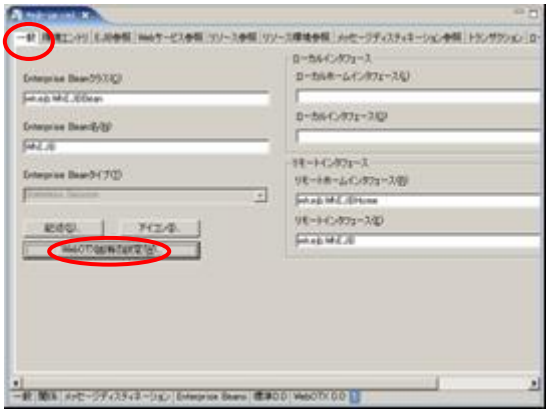
<ejb-jar>

<display-name>**指定した表示名**

③ 「Enterprise Beans」タブの「編集」ボタンを押下して画面を開きます。



④「一般」タブの「WebOTX 固有の設定」ボタンを押下しWebOTX 固有の設定画面を開きます。



⑤「JNDI 名」に Web アプリケーションからEJB アプリケーションを参照するときのEJB アプリケーションの名前を指定します。実行環境で一意の名称にする必要があります。

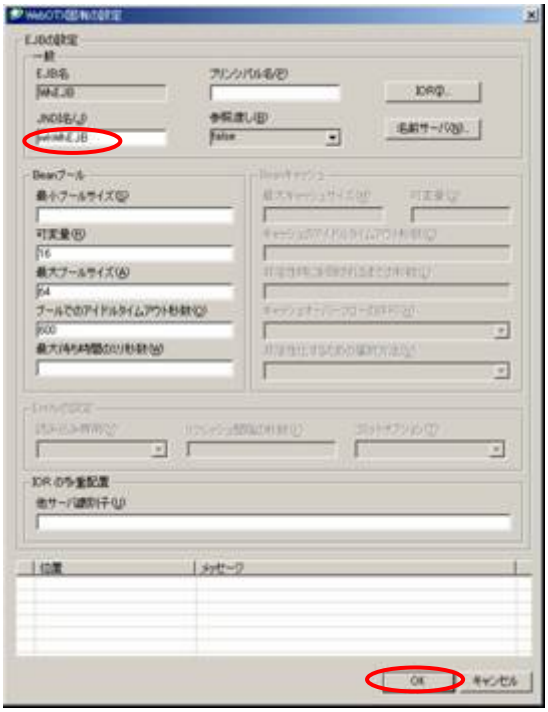
例：whWhEJB

MFDL 移行プロジェクトで生成したnec-ejb-jar.xml には、以下の値が設定されています。

パッケージ名＋”WhEJB”

※ この設定で変更される nec-ejb-jar.xml の要素を示します。

```
<nec-ejb-jar>
 <enterprise-beans>
  <ejb>
   <jndi-name>指定した JNDI 名
```



⑥「OK」ボタンを押下します。

⑦「リソース参照」タブを選択します。

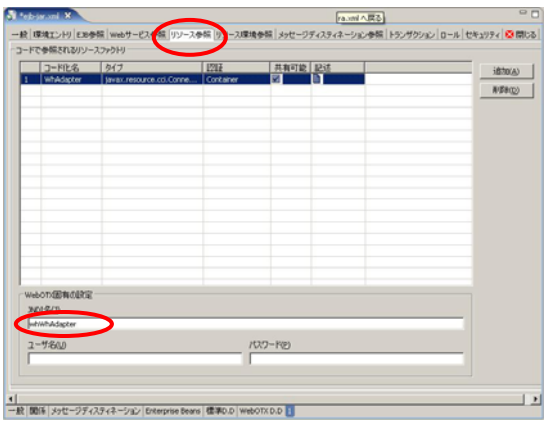
⑧「JNDI 名」には、EJB アプリケーションから呼び出すコネクタアプリケーションの名前を指定します。

例：whWhAdapter

MFDL 移行プロジェクトで生成したnec-ejb-jar.xml には、以下の値が設定されています。

パッケージ名＋”WhAdapter”

この値は、「3.3.3(5) コネクタリソースの登録」で指定したJNDI名に合わせる必要があります。



※ この設定で変更される nec-ejb-jar.xml の要素を示します。

```
<nec-ejb-jar>
 <enterprise-beans>
  <ejb>
   <resource-ref>
    <jndi-name>指定した JNDI 名
```

⑨「閉じる」タブを押下します。

⑩ 「ファイル」メニュー-「保管」を選択して ejb-jar.xml, nec-ejb-jar.xml の内容を更新します。

3.4.3.EJB アプリケーションの配備

ここでは、EJB アプリケーションの配備方法について説明します。以下の手順で行います。

※EJB アプリケーションの配備は、J2EE パースペクティブで行います。

(1) EJBアプリケーションのビルド

(2) EJBアプリケーションのパッケージ

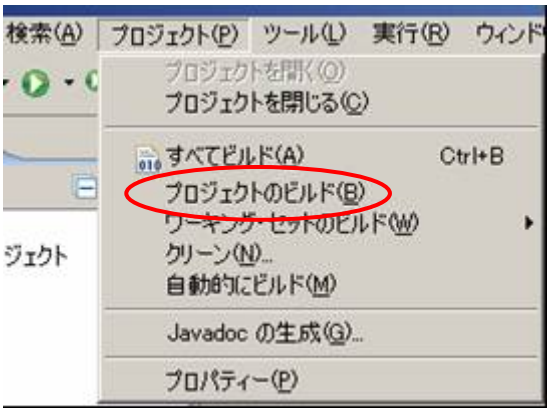
(3) EJBアプリケーションの配備

(1) EJB アプリケーションのビルド

配備ツールで配備を行うためには作成した EJB アプリケーションをビルドする必要があります。

「自動的にビルド」が設定されている場合は、本作業は不要です。自動的にビルドが設定されていない場合は、次の手順でビルドを行ってください。

① メニューバーから、「プロジェクト」→「プロジェクトのビルド」を実行します。



(2) EJB アプリケーションのパッケージ

配備ツールで配備を行うためには作成した EJB アプリケーションをパッケージする必要があります。

① メニューバーから、「ファイル」→「エクスポート」を実行します。

② 「EJB」→「EJB JAR ファイル」を選択し、「次へ」ボタンを押下します。

※J2EE パースペクティブでは、EJB プロジェクトの右クリックメニュー「エクスポート」→「EJB JAR ファイル」からも作業を行うことができます。

③ 次の値を指定します。

「EJBモジュール」:
「3.4.1(1) EJBプロジェクトの作成」で作成したプロジェクト名

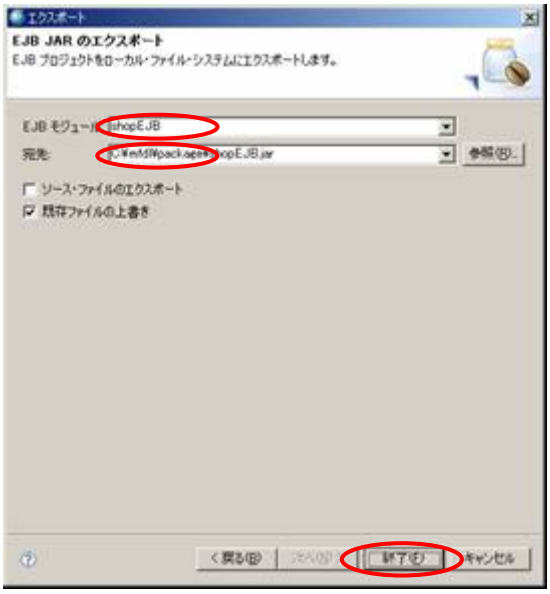
例:shopEJB

「宛先」:
保存するファイル名

例:C:¥mfdl¥package¥shopEJB.jar

※ ファイル名は、実行環境で一意の名前にする必要があるため、プロジェクト名と同じにすることを推奨します。

※ ファイル名にパス指定をしない場合、<WebOTX install dir>/Studio 直下に作成されます。専用フォルダーを用意しフルパス名で指定することを推奨します。



④ 「終了」ボタンを押下して、EJB-JAR ファイルを作成します。

(3) EJB アプリケーションの配備

アプリケーションを配備するためには、以下のようにいくつかの方法があります。

- 配備ツールによる配備
- 運用管理コマンドによる配備
- パッケージ化しないアプリケーションの配備

ここでは、「配備ツール」を利用した手順を記述します。

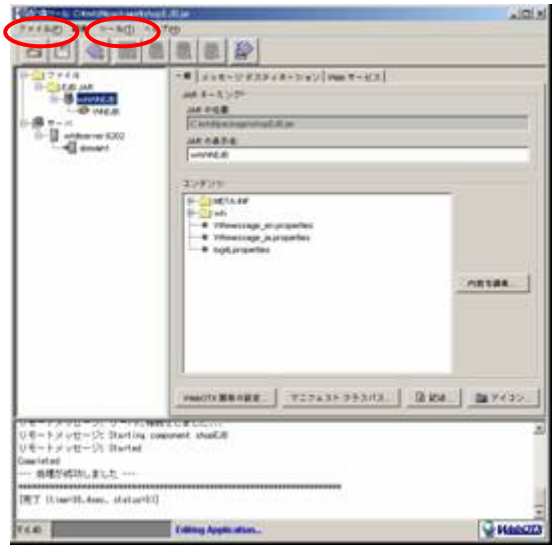
運用管理コマンドによる配備方法については、次のマニュアルを参照してください。

WebOTX マニュアル 運用編
アプリケーション配備
2.2.4. コマンドによる配備・配備解除

① 配備ツールを起動し、ドメインへ接続します。

② メニューバーから、「ファイル」→「開く」を選び、パッケージした EJB-JAR ファイルを選択します。

③ メニューバーから、「ツール」→「配備」を選択して、配備を実行します。



3.4.4.EJB アプリケーションの再配備

MFDL 移行プロジェクトで画面を追加した場合など、EJB アプリケーションが変更されたときに再配備を行います。

ここでは、「配備ツール」を利用した手順を記述します。

(1) EJB アプリケーションのパッケージ

「3.4.3(2) EJBアプリケーションのパッケージ」と同様の手順でEJBアプリケーションをパッケージします。

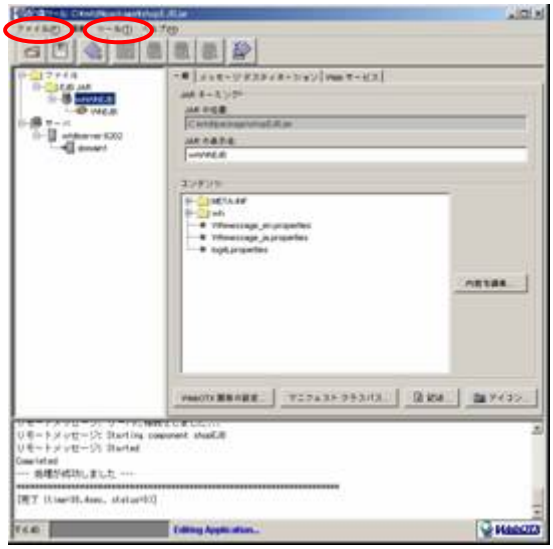
(2) EJB アプリケーションの再配備

配備ツールにて EJB アプリケーションを再配備します。

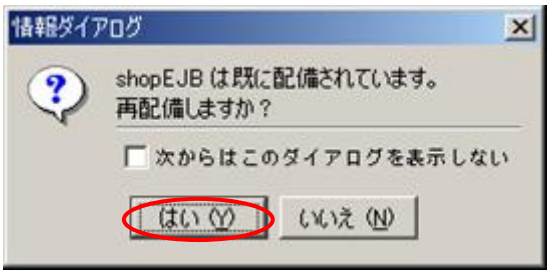
① 配備ツールを起動し、ドメインへ接続します。

② メニューバーから、「ファイル」→「開く」を選び、パッケージした EJB-JAR ファイルを選択します。

③ メニューバーから、「ツール」→「配備」を選択して、配備を実行します。



④ 同名のパッケージが配備済みである確認ダイアログが表示された場合は、「はい」ボタンを押下します。



3.5.Web アプリケーションの作成と配備

本節では、MFDL 移行プロジェクトで作成したファイル(jsp,java,xml)を使って Web アプリケーションを作成する方法、および Web アプリケーションの配備方法について説明します。

※ ここからの作業は MFDL 移行パースペクティブと J2EE パースペクティブを用いた作業となります。MFDL パースペクティブに切り替えてからの作業をおすすめします。

3.5.1.Web アプリケーションの作成

以下の手順により Web アプリケーションを作成します。

(1) 動的 Web プロジェクトの作成

(2) MFDL 移行プロジェクトのリンク

(3) EJB アプリケーションのクライアント JAR ファイルの追加

(4) 確認

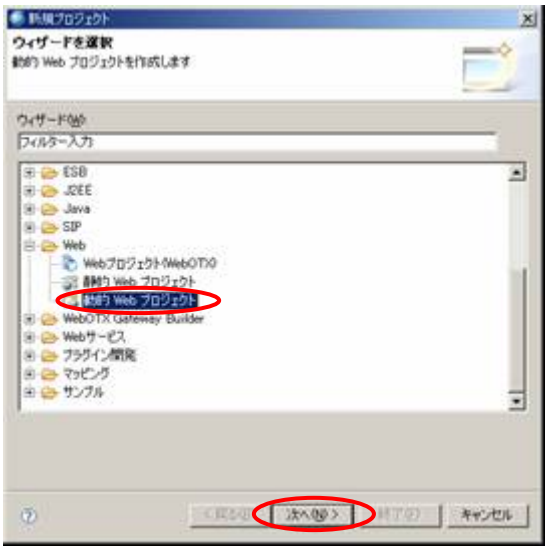
Web アプリケーションの作成について詳しく知りたい場合は、次のマニュアルを参照してください。

WebOTX マニュアル アプリケーション開発ガイド
第 4 章 プログラミング・開発(J2EE)
4.2. Web アプリケーション(pdf 形式)

(1) 動的 Web プロジェクトの作成

① メニューバーから、「ファイル」→「新規」→「プロジェクト」を選択して、「新規プロジェクト」画面を開きます。

② 「Web」→「動的Webプロジェクト」を選択して、「次へ」ボタンを押します。



③ 次の値を指定します。

「プロジェクト名」:
任意の名称

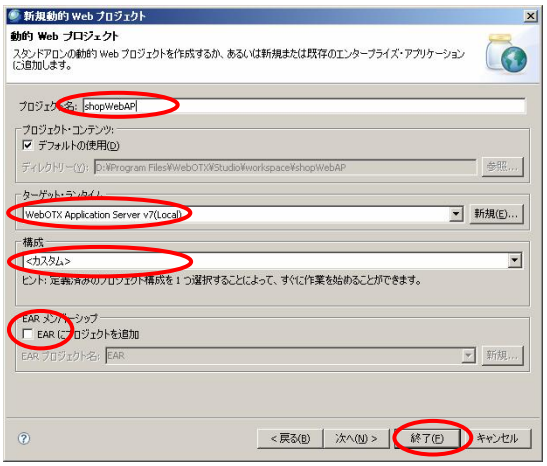
例:shopWebAP

「ターゲット・ランタイム」:
WebOTX Application Server v7(Local)

※ Web アプリケーションの動作に必要な j2ee.jar ランタイムライブラリを指定します。

「構成」:
<カスタム>

※既定値の<カスタム>でよいです。



「EAR メンバシップ」:
　指定する必要はありません。

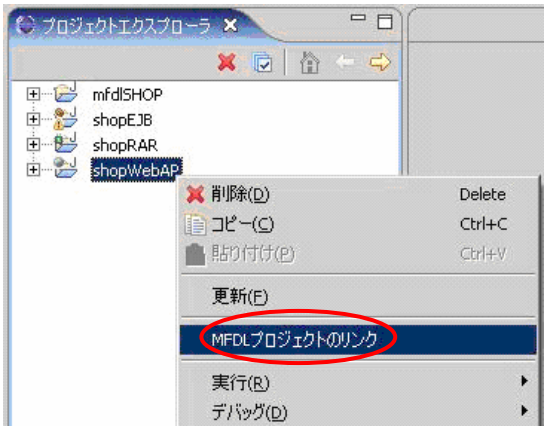
④「終了」ボタンを押下します。

※「次へ」ボタンを押下すると、指定した構成のプロジェクト・ファセットを設定できます。

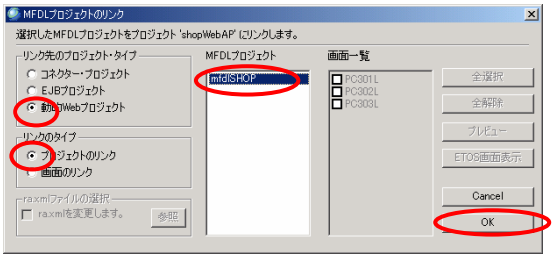
(2) MFDL 移行プロジェクトのリンク

① <プロジェクト名>の右クリックメニューから「MFDL プロジェクトのリンク」を選択します。

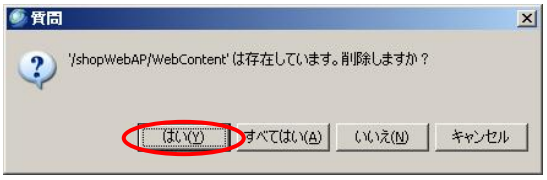
※MFDL 移行プロジェクトのリンクは MFDL 移行パースペクティブでの作業となります。
MFDL 移行パースペクティブに切り替えてから作業してください。



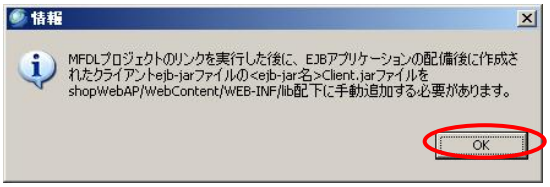
② 「リンク先のプロジェクトタイプ」に動的 Web プロジェクトを「リンクのタイプ」にプロジェクトのリンクを選択し、「MFDL プロジェクト」を選択して、「OK」ボタンを押下します。



③ 「削除しますか？」の質問ダイアログが表示されたときは、「はい」を選択します。



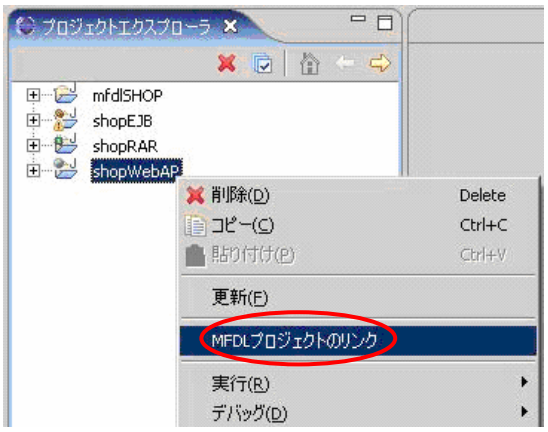
④ メッセージを確認後、「OK」ボタンを押下します。本作業を(4)にて説明します。



(3) 画面のリンク

(2)の代わりに MFDL 移行プロジェクトの必要な画面を選択して、リンクを行なうこともできます。

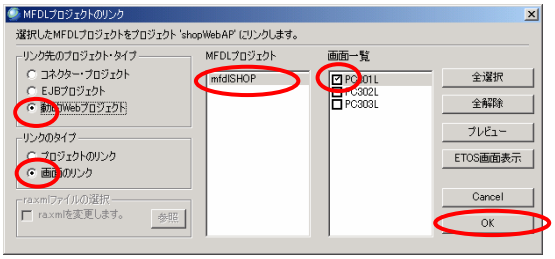
① <プロジェクト名>の右クリックメニューから「MFDL プロジェクトのリンク」を選択します。



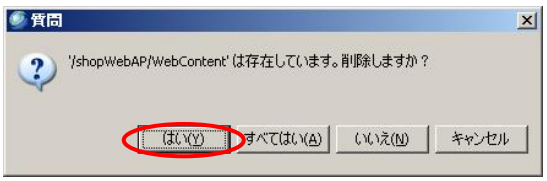
② 「リンク先のプロジェクトタイプ」に動的 Web プロジェクトを「リンクのタイプ」に画面のリンクを選択します。

「MFDL プロジェクト」からリンクする MFDL プロジェクトを選択すると「画面一覧」に選択したプロジェクトの画面一覧が表示されます。

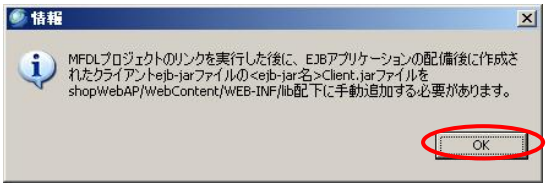
「画面一覧」からリンクする画面を選択して、「OK」ボタンを押下します。



③ 「削除しますか？」の質問ダイアログが表示されたときは、「はい」を選択します。



④ メッセージを確認後、「OK」ボタンを押下します。本作業を次項にて説明します。



画面をリンクするとき、<プロジェクト名>/WebContent/WEB-INF/web.xml はリンクする MFDL プロジェクトからコピーします。web.xml において初期画面として指定した JSP ファイルがリンク先にない場合は、リンク先にある JSP ファイルを初期画面として設定します。

初期画面の変更については、3.5.2(1)を参照してください。

(4) EJB アプリケーションのクライアント JAR ファイル追加

① プロジェクト内の lib フォルダーに、EJB アプリケーションの配備後に作成されたクライアント ejb-jar ファイルを配置します。

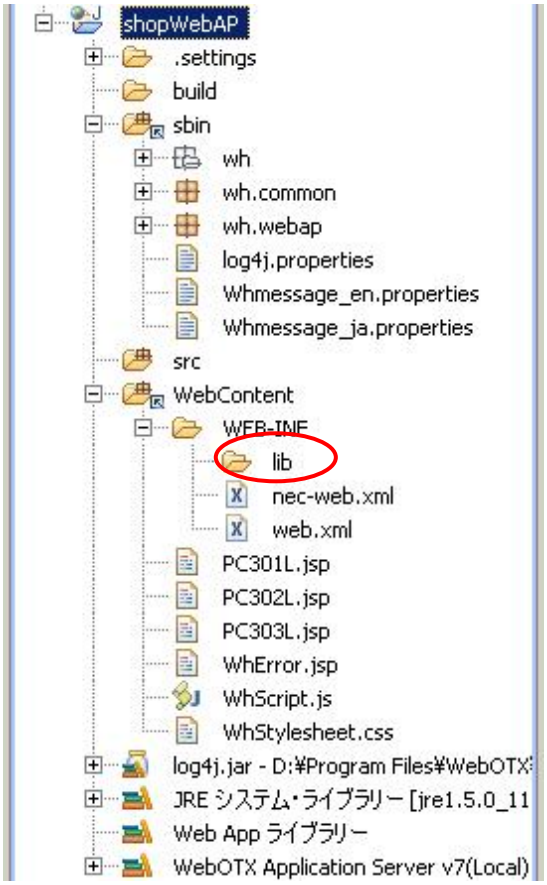
クライアント JAR は、 EJB アプリケーションの配備先サーバ(実行環境)の <WebOTX install dir>/domains/domain1 /applications/j2ee-modules/<ejb-jar 名> フォルダーの<ejb-jar 名>Client.jar になります。

開発環境と実行環境が同じサーバの場合は、「3.3.1(2)MFDL移行プロジェクトの」と同様の方法で、ファイルを追加します。

「ソース・ディレクトリー」:
<WebOTX install dir>/domains/ domain1/applications/j2ee-modules/ <ejb-jar 名>

「宛先フォルダー」:
<プロジェクト名>/WebContent/ WEB-INF/lib

次のファイルをチェック
<ejb-jar 名>Client.jar



開発環境と実行環境が異なるサーバの場合は、実行環境のクライアント JAR を開発環境にコピーしてから、ファイルの追加をして下さい。

(5) 確認

① 作業が完了したら、右図のようにファイルが格納されていることを確認してください。

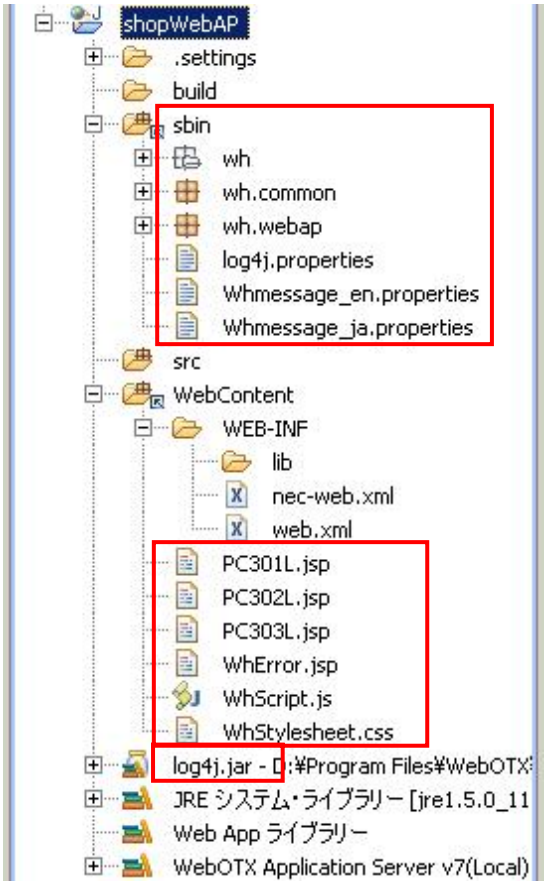
・sbin/
　wh.common/
　wh.webap/
　log4j.properties
　Whmessage_en.properties
　Whmessage_ja.properties

・WebContent/
　<画面名>.jsp
　WhError.jsp
　WhScript.js
　WhStylesheet.css

・WebContent/WEB-INF/
　nec-web.xml
　web.xml

・log4j.jar

「wh.common」「wh.webap」の「wh」は、MFDL移行プロジェクトのパッケージ名を示します。



3.5.2.Web アプリケーションの設定

ここでは、Web アプリケーションの設定を行います。web.xml、nec-web.xml という設定ファイルに各種設定を行います。前記リンク機能を利用したときは、既定の設定が反映された状態になっています。このとき、初期画面の設定を行うことを忘れないように留意してください。

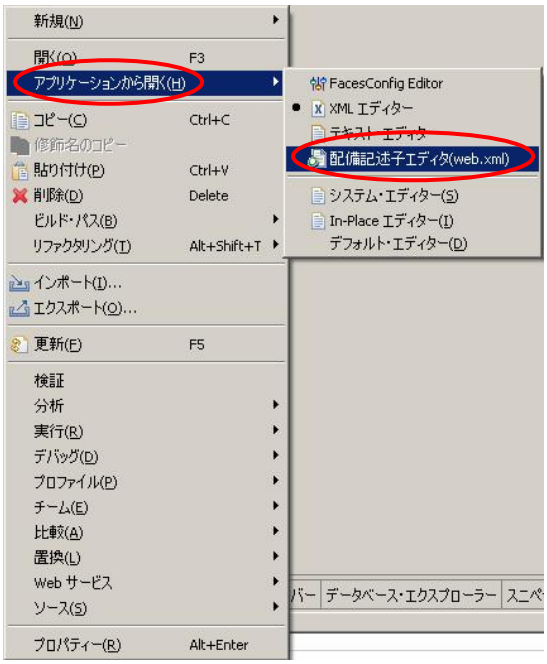
web.xml の詳細については、以下のマニュアルを参照してください。

WebOTX マニュアル アプリケーション開発ガイド
第 4 部 プログラミング・開発
1 章 Developer’s Studio
1.3. 配備記述子エディタ(pdf 形式)

(1) web.xml, nec-web.xml の設定

ここでは、Web アプリケーションの設定ファイル(web.xml,nec-web.xml)を設定します。

① <Web project dir>/WebContent/WEB-INF/web.xml を右クリックして「Open With」-「配備記述子エディタ」を選択します。



② 「一般」タブの「Web アプリケーション情報」-「表示名」にこのパッケージの名称を指定します。

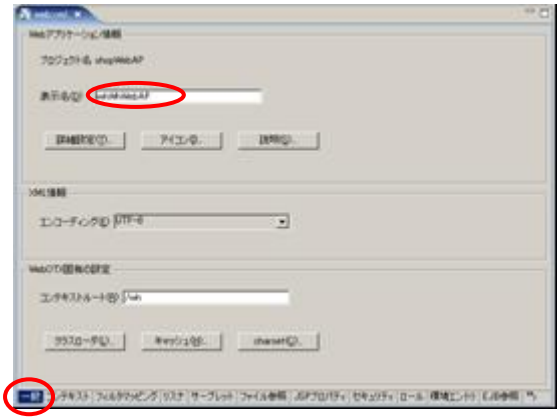
例：whWhWebAP

MFDL 移行プロジェクトで生成した web.xml には、以下の値が設定されています。

パッケージ名＋”WhWebAP”

※ この設定で変更される web.xml の要素を示します。

<web-app>
<display-name>**指定した表示名**

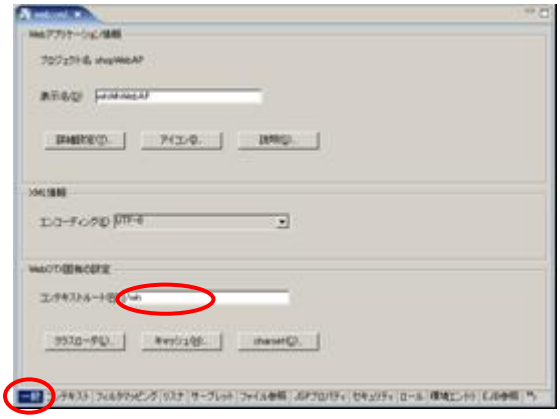


③「一般」タブの「WebOTX 固有の設定」-「コンテキストルート」にこのパッケージのコンテキストルートを指定します。

例：wh

MFDL 移行プロジェクトで生成した web.xml には、以下の値が設定されています。
パッケージ名

同名のコンテキストルートを設定した別アプリケーションを配備済みの場合は、コンテキストルートを変更してください。



コンテキストルートは、ブラウザから Web アプリケーションを呼び出すための名前に使用されます。ブラウザから次の URL にアクセスすることで Web アプリケーションが起動できます。

http://<サーバ名>/<コンテキストルート>/

※ <サーバ名>は、Web アプリケーションを配備したサーバ名です。

この設定で変更される nec-web.xml の要素を示します。

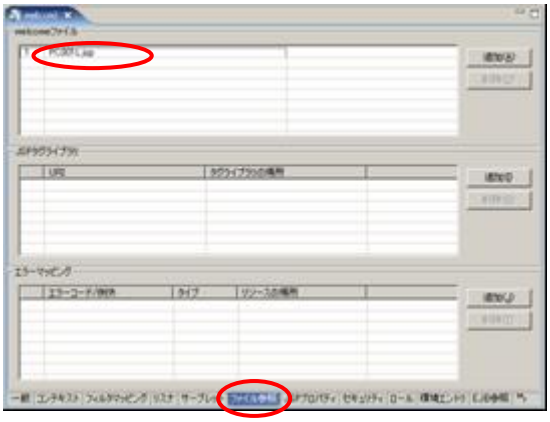
```
<nec-web-app>
 <context-root>指定したコンテキストルート名
```

④「ファイル参照」タブの「welcome ファイル」に、③の URL で Web アプリケーションを起動した際に表示する画面を指定します。

例：PC301L.jsp

MFDL 移行プロジェクトで生成した web.xml には、以下の値が設定されています。
<最初に変換した画面名>＋”.jsp”

表示する画面を変えたい場合に、設定を変更してください。



※ この設定で変更される web.xml の要素を示します。

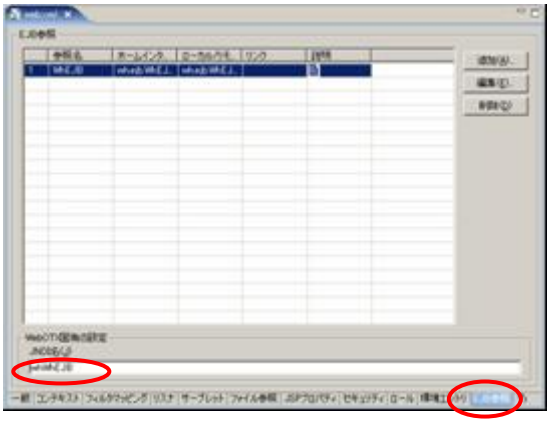
```
<web-app>
 <welcome-file-list>
  <welcome-file>指定した jsp 名
```

⑤「EJB 参照」タブの「WebOTX 固有の設定」→「JNDI 名」には Web アプリケーションから参照する EJB アプリケーションの名前を指定します。

例：whWhEJB

MFDL 移行プロジェクトで生成した nec-web.xml には、以下の値が設定されています。
パッケージ名＋”WhEJB”

この値は「3.4.2EJBアプリケーションの設定」で指定したJNDI名に合わせる必要があります。



※ この設定で変更される nec-web.xml の要素を示します。

```
<nec-web-app>
  <ejb-ref>
    <jndi-name>指定した JNDI 名
```

⑥ 「ファイル」メニュー-「保管」を選択して web.xml, nec-web.xml の内容を更新します。

3.5.3.Web アプリケーションの配備

ここでは、Web アプリケーションの配備方法について説明します。以下の手順で配備を行います。

※Web アプリケーションの配備は、J2EE パースペクティブで行います。

(1) Webアプリケーションのビルド

(2) Webアプリケーションのパッケージ

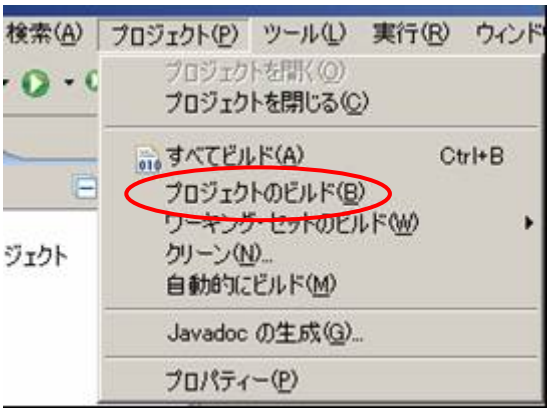
(3) Webアプリケーションの配備

(1) Web アプリケーションのビルド

配備ツールで配備を行うためには、作成した Web アプリケーションをビルドする必要があります。

「自動的にビルド」が設定されている場合は、本作業は不要です。自動的にビルドが設定されていない場合は、次の手順でビルドを行ってください。

① 「プロジェクト」メニューから「プロジェクトのビルド」を実行します。



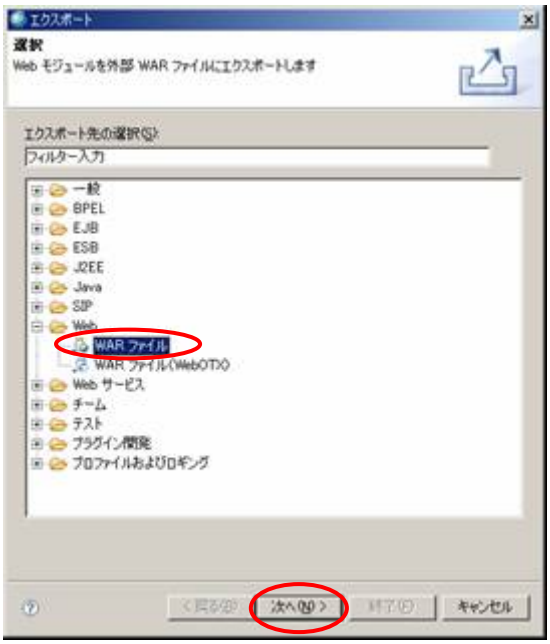
(2) Web アプリケーションのパッケージ

配備ツールで配備を行うためには作成した Web アプリケーションをパッケージする必要があります。

① メニューバーから、「ファイル」→「エクスポート」を実行します。

② 「Web」→「WAR ファイル」を選択し、「次へ」ボタンを押下します。

※J2EE パースペクティブでは、動的 Web プロジェクトの右クリックメニュー「エクスポート」→「WAR ファイル」からも作業を行うことができます。



③ 次の値を指定します。

「Webモジュール」:
「3.5.1(1) 動的Webプロジェクトの作成」で作成したプロジェクト名

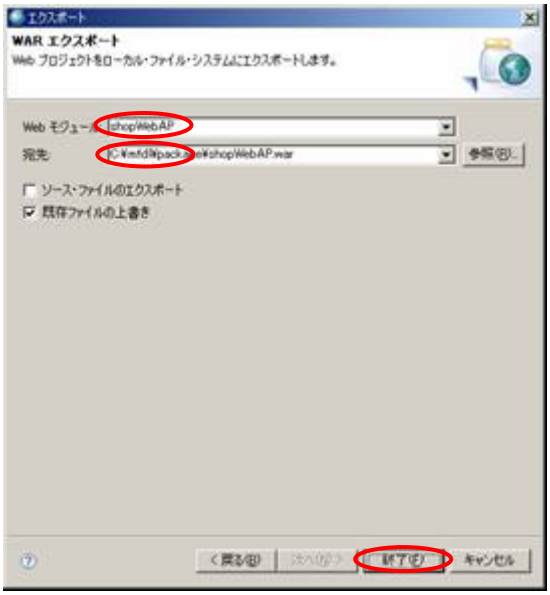
例:shopWebAP

「宛先」:
保存するファイル名

例:C:¥mfdl¥package¥shopWebAP.war

※ ファイル名は、実行環境で一意の名前にする必要があるため、プロジェクト名と同じにすることを推奨します。

※ ファイル名にパス指定をしない場合、<WebOTX install dir>/Studio 直下に作成されます。専用フォルダーを用意しフルパス名で指定することを推奨します。



④ 「終了」ボタンを押下して、WAR ファイルを作成します。

(3) Web アプリケーションの配備

アプリケーションの配備には以下の方法があります。

- 配備ツールによる配備
- 運用管理コマンドによる配備
- パッケージ化しないアプリケーションの配備

ここでは、「配備ツール」を利用した手順を記述します。

運用管理コマンドによる配備について調べたい場合は、次のマニュアルを参照してください。

WebOTX マニュアル 運用編
アプリケーション配備
2.2.4. コマンドによる配備・配備解除

① 配備ツールを起動し、ドメインへ接続します。

② メニューバーから、「ファイル」→「開く」を選び、パッケージした WAR ファイルを選択します。

③ メニューバーから、「ツール」→「配備」を選択して、配備を実行します。

3.5.4.Web アプリケーションの再配備

MFDL 移行プロジェクトで画面を追加した場合など、Web アプリケーションを変更したときに再配備を行います。

ここでは、「配備ツール」を利用した手順を記述します。

(1) Web アプリケーションのパッケージ

「3.5.3(2)Webアプリケーションのパッケージ」と同様の手順でWebアプリケーションをパッケージします。

(2) Web アプリケーションの再配備

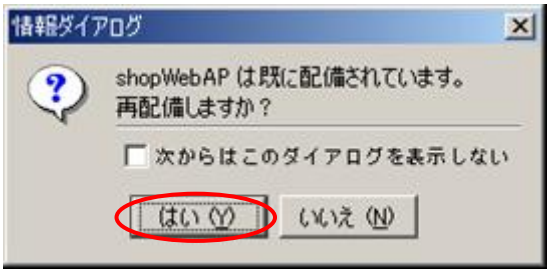
配備ツールにて Web アプリケーションを再配備します。

① 配備ツールを起動し、ドメインへ接続します。

② メニューバーから、「ファイル」→「開く」を選び、パッケージした WAR ファイルを選択します。

③ メニューバーから、「ツール」→「配備」を選択して、配備を実行します。

④ 同名のパッケージが配備済みである確認ダイアログが表示された場合は、「はい」ボタンを押下します。



3.6.Web アプリケーションの実行

本節では、配備した Web アプリケーションの実行方法について記述しています。以下の手順で、Web ブラウザを使用して Web アプリケーションを実行します。

3.6.1.Web アプリケーションの実行

以下の手順により Web アプリケーションを実行します。

(1) Web アプリケーションの実行

① Web ブラウザを起動し、次の URL へアクセスします。

http://<サーバ名>/<コンテキストルート>/

※ <サーバ名>とは Web アプリケーションを配備したサーバ名です。

<コンテキストルート>とは、サーバーで実行している web アプリケーションのルートです。

例：http://localhost/wh/

ここでは、配備したサーバが自サーバ、コンテキストルートが"wh"、最初に表示される画面が PC301L.jsp の場合の例を示します。

② サブミットボタン（実行）を押すことによって、ACOS のトランザクション処理が実行され、その結果画面が表示されます。

③ TPP が実行され、その結果の画面が表示されます。
また、このとき URL には次の値が表示されます。

http://<サーバ名>/
<コンテキストルート>/<url-pattern>

※ <url-pattern>とは Web アプリケーションの設定ファイル web.xml の url-pattern 要素に設定されている値です。
生成された web.xml では"servlet"が設定されています。

例：http://localhost/wh/servlet

3.7.その他

本節では、エンタープライズ・アプリケーションの作成方法と運用管理コマンドによる配備方法について説明します。

「3.4.3(2) EJBアプリケーションのパッケージ」で作成したファイルと「3.5.3(2) Webアプリケーションのパッケージ」で作成したファイルを使って、エンタープライズ・アプリケーションのプロジェクトを作成し、EARファイルを作成します。

3.7.1.エンタープライズ・アプリケーションの作成

エンタープライズ・アプリケーションの作成については、次のマニュアルを参照してください。

```
# WebOTX マニュアル  アプリケーション開発ガイド
#  第 4 章  プログラミング・開発（J2EE）
#    4.5. J2EE アプリケーション(pdf 形式)
```

エンタープライズ・アプリケーションの作成の手順における「EAR に追加する J2EE モジュール」は、

「3.4.3(2) EJBアプリケーションのパッケージ」、「3.5.3(2) Webアプリケーションのパッケージ」にて作成済みのWebアプリケーションとEJBアプリケーションを選択してください。

3.7.2.エンタープライズ・アプリケーションの配備

アプリケーションの配備には以下の方法があります。

・配備ツールによる配備

・運用管理コマンドによる配備

・パッケージ化しないアプリケーションの配備

ここでは、「運用管理コマンドによる配備」を利用した手順を記述します。

運用管理コマンドによる配備方法については、次のマニュアルを参照してください。

```
# WebOTX マニュアル  運用編
#  アプリケーション配備
#    2.2.4. コマンドによる配備・配備解除
```

(1) エンタープライズ・アプリケーションのパッケージ

運用管理コマンドで配備を行うためには、作成したエンタープライズ・アプリケーションをパッケージする必要があります。

「3.4.3(2) EJBアプリケーションのパッケージ」、「3.5.3(2) Webアプリケーションのパッケージ」と同様に「エクスポート」を使用し、EARファイルを作成します。

(2) エンタープライズ・アプリケーションの配備

通常、運用管理コマンドは、1つのコマンドを実行しますが、複数のコマンドをまとめて実行することもできます。

複数のコマンドをまとめて実行する場合、スクリプトファイルを作成して、運用管理コマンドを起動し、コマンド(multimode)を実行します。

下記にコマンドと配備を行うスクリプトファイル例を示します。

```
otxadmin> multimode --file "C:¥mfdl¥script¥deploy_shop.txt"
```

[スクリプトファイル例:C¥mfdl¥script¥deploy_shop.txt]

```
login --user <admin> --password <adminadmin> --host <localhost> --port <6212>
deploy "<C:¥mfdl¥package¥shopRAR.rar>"
create-connector-connection-pool  --steadypoolsize <poolsize> --raname <shopRAR>
    --connectiondefinition javax.resource.cci.ConnectionFactory <whWhPool>
create-connector-resource --poolname <whWhPool> <whWhAdapter>
deploy "<C:¥mfdl¥package¥shopEAR.ear>"
logout
exit
```

[説明]
login :
ドメインアプリケーションサーバの管理ユーザ名、パスワード、マシン名、ポート番号を指定して、ログインします。
deploy :
コネクタアプリケーションを配備します。
create-connector-connection-pool :
リソースアダプタ名、コネクションプール名を指定してコネクションプールを作成します。
<poolsize> には、ACOS とのコネクション数を指定します。
create-connector-resource :
コネクションプール名を指定してコネクタリソースを登録します。
deploy :
「(1)エンタープライズ・アプリケーションのパッケージ」で作成したearファイルを配備します。

下記にコマンドと配備解除を行うスクリプトファイル例を示します。

```
otxadmin> multimode --file "C:¥mfdl¥script¥undeploy_shop.txt"
```

[スクリプトファイル例:C:¥mfdl¥script¥undeploy_shop.txt]

```
login --user <admin> --password <adminadmin> --host <localhost> --port <6212>
delete-connector-resource <whWhAdapter>
delete-connector-connection-pool <whWhPool>
undeploy <shopRAR>
undeploy <shopEAR>
logout
exit
```

[説明]
delete-connector-resource :
コネクタリソースを削除します。
delete-connector-connection-pool :
コネクションプールを削除します。
undeploy :
コネクタアプリケーションを削除します。
undeploy :
エンタープライズアプリケーションを削除します。

4.生成するアプリケーションの構成

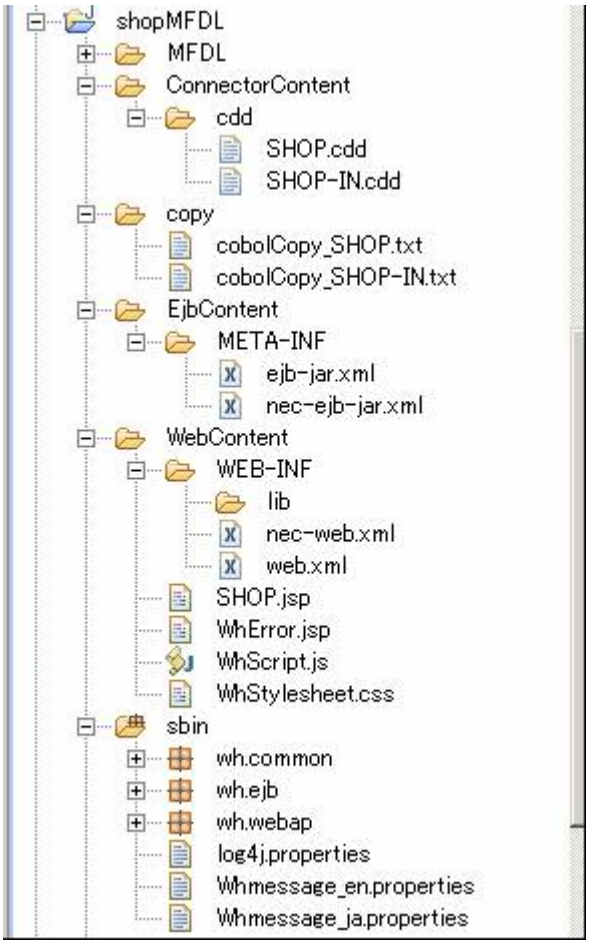
4.1.生成するファイルの構造

本節では、MFDL 移行プロジェクトが生成するファイルについて記述しています。

4.1.1.生成するファイルの構造

MFDL 移行プロジェクトが、生成するファイルのフォルダー構造と説明を下記に示します。

※ここで 〈パッケージ名〉は「wh」、MFDL ソースの LMFD 名（画面名）は「SHOP」を使用した例を示します。



フォルダー	生成するファイル	説明
ConnectorContent/cdd	〈画面名〉.cdd 〈画面名〉-IN.cdd	コネクタアプリケーションで使用する COBOL データ定義ファイル(cdd ファイル)です。
copy	cobolCopy_〈画面名〉.txt cobolCopy_〈画面名〉-IN.txt	cdd ファイル作成のために使用する COBOL COPY 句ファイルです。
EjbContent/META-INF	ejb-jar.xml nec-ejb-jar.xml	EJB アプリケーションの設定ファイルです。
WebContent/WEB-INF	web.xml nec-web.xml	Web アプリケーションの設定ファイルです。

WebContent	<画面名>.jsp	Web ブラウザに表示する画面ファイルです。
	WhError.jsp	Web アプリケーション実行時に発生する例外を Web ブラウザに表示するエラー画面ファイルです。
	WhScript.js	Web ブラウザ上でブリンク制御などを行うためのスクリプトが格納された JavaScript ファイルです。
	WhStylesheet.css	Web ブラウザに表示する画面の表示属性(色・センタリング)定義ファイルです。
sbin	<パッケージ名>.common <パッケージ名>.ejb <パッケージ名>.webap	Java アプリケーションソースです。common、ejb、webapの 3 つのコンポーネントから構成されます。詳細は「4.2 コンポーネント構造」で説明します。
	Whmessage_en.properties Whmessage_ja.properties	メッセージ出力用のプロパティファイルです。
	log4j.properties	ログ出力設定用のプロパティファイルです。

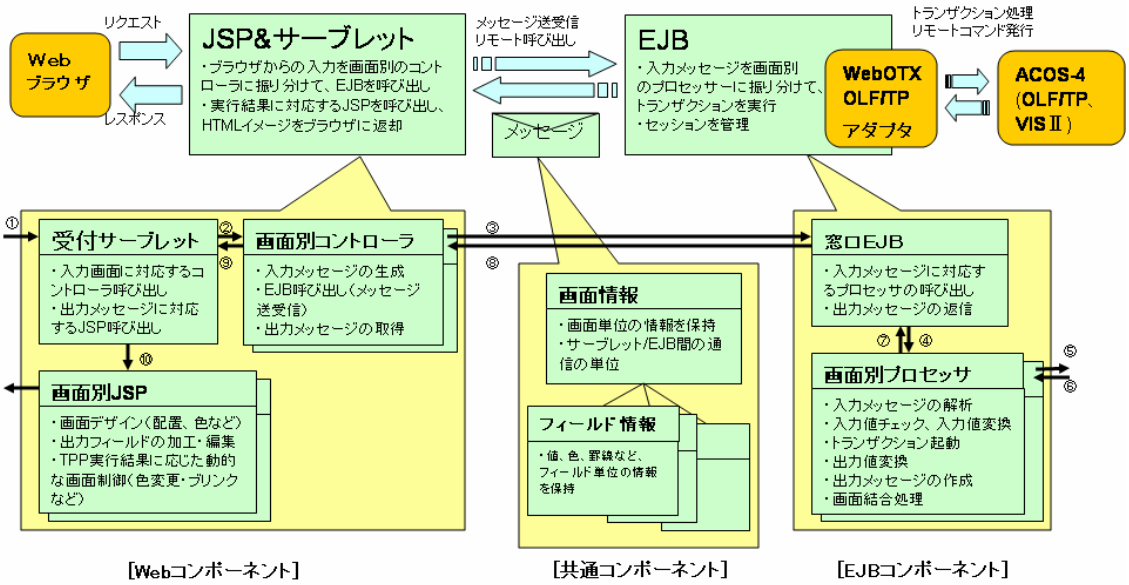
4.2.コンポーネント構造

本節では、MFDL 移行プロジェクトが生成する Web アプリケーションについて記述しています。

4.2.1.コンポーネント構造

コンポーネント構造は、下記のようになります。

- 受付サーブレット、画面別コントローラ、画面別 JSP は Web コンポーネントです。
- 窓口 EJB、画面別プロセッサは EJB コンポーネントです。
- 画面情報、フィールド情報は共通コンポーネントです。



4.3.Web コンポーネント

本節では、生成した Web アプリケーションの Web コンポーネントについて記述しています。

MFDL 移行プロジェクトで生成した「sbin/<パッケージ名>/webap」フォルダに作成されます。

4.3.1.Web コンポーネント

Web コンポーネントには以下のクラスがあります。

クラスの概要	
WhCtrl	画面ごとのコントローラである WhCtrl <画面名>クラスのスーパークラスです。 JNDI(Web コンポーネントから EJB コンポーネントを呼び出すための名称)を使って EJB コンポーネントのリモートインタフェースを取得します。実際には WhEJBBean インスタンスを取得し、セッションへ登録します。
WhCtrl <画面名>	画面ごとのコントローラである WhCtrl <画面名>クラスです。 トランザクション処理、入力メッセージ作成処理、入力チェック処理、入力値設定処理、入力値復元処理を行います。
WhDhtmlId	Html タグの ID を管理するクラスです。ブリンク制御で使われます。
WhMappingWebApDefault	画面名に関連付けられたクラス名や JSP 名を取得するクラスです。 WhCtrl <画面名>クラス名、JSP 名、エラーJSP 名を取得します。
WhMappingWebAp	WhCtrl <画面名>クラスや JSP をカスタマイズしたときに、画面名との関連付けを変更するために使用します。詳細は「5. アプリケーションのカスタマイズ」を参照してください。
WhServlet	リクエストの受け付けおよび処理の振り分けを行うクラスです。
WhUtilWebAp	Web コンポーネントで使用するユーティリティクラスです。

(1) WhCtrl クラス

メソッドの概要	
getEJB	JNDI を使って WhEJBBean クラスのインスタンスを取得します。
getInstance	WhCtrl <画面名>クラスのインスタンスを取得します。

(2) WhCtrl <画面名> クラス

メソッドの概要	
buildMessage	ブラウザから入力された値を元に、EJB コンポーネントに渡す入力メッセージを構築します。 入力フィールドごとに generate<フィールド名>メソッドを呼び出します。
clearBlink	ブリンクしている項目のブリンクを解除します。
generate<フィールド名>	ブラウザから入力された値を、入力メッセージ(WhMessage<画面名>オブジェクト)に設定します。
getInMessage	入力メッセージのインスタンスを作成します。
request	EJB コンポーネントを呼び出します。
setInMessage	入力メッセージを設定します。

setInputData	入力チェックでエラーになった時に入力画面を再表示するために使用します。再表示のための入力値の復元を行います。
validate<フィールド名>	ブラウザから入力された値の検証を行います。

(3) WhDhtmlIdクラス

メソッドの概要	
getElementId	ブリンク処理のための Html タグを取得します。
getSeqNo	ブリンク処理のための順次番号を取得します。

(4) WhMappingWebApDefault クラス

メソッドの概要	
getCtrlClassName	画面名に対応する WhCtrl <画面名>クラス名を取得します。
getErrJspName	エラー画面 JSP 名称を取得します。
getJspName	画面名に対応する JSP 名を取得します。

(5) WhServletクラス

メソッドの概要	
doPost	リクエスト受付を行います。
doPost0	リクエスト受付時のトランザクション起動、画面遷移を行います。
exceptionProcessor	Exception 発生時にログ出力を行い、エラー画面へ遷移します。
init	Servlet の初期化を行います。

(6) WhUtilWebAp クラス

メソッドの概要	
convXsite	Xsite スクリプティング対応として、 Html 特殊文字(“<”、”>”、” ”)を変換します。
getHtmlTag	TPP から返却された情報(値、modify 情報など)を元に、Html タグを生成します。
getJndi	JNDI 名(Web コンポーネントから EJB コンポーネントを呼び出すための名称)を取得します。

4.4.EJB コンポーネント

本節では、生成した Web アプリケーションの EJB コンポーネントについて記述しています。

MFDL 移行プロジェクトで生成した「sbin/<パッケージ名>/ejb」フォルダに作成されます。

4.4.1.EJB コンポーネント

EJB コンポーネントには以下のインタフェースやクラスがあります。

インタフェースの概要	
WhEJB	WhEJBBean のメソッドを定義するリモートインタフェースクラスです。 トランザクションの実行メソッドの定義です。
WhEJBHome	WhEJBBean 取得時に必要なインタフェースクラスです。 WhEJBBean を生成・取得するメソッドの定義です。

クラスの概要	
WhEJBBean	EJB コンポーネントのリモートインターフェースとしてトランザクションを実行するクラスです。 アダプタ JNDI(EJB コンポーネントからリソースアダプタを呼び出すための名称)を使ってリソースアダプタ用のレコード作成、トランザクション実行、セッションコンテキスト設定を行います。
WhMappingEJBDefault	画面名に関連付けられたクラス名を取得するクラスです。 画面名に対応する WhProcessor<画面名>クラス名を取得します。
WhMappingEJB	WhProcessor<画面名>クラスをカスタマイズしたときに、画面名との関連付けを変更するために使用します。詳細は「5. アプリケーションのカスタマイズ」を参照してください。
WhProcessor	ACOS ホストと通信を行う WhProcesser<画面名>クラスのスーパークラスです。
WhProcessor<画面名>	Web コンポーネントから渡された入力メッセージを元にトランザクションを実行し、実行結果から出力メッセージ(WhMessage<画面名>オブジェクト)を生成するクラスです。
WhUtilEJB	EJB コンポーネントで使用するユーティリティクラスです。

(1) インタフェース WhEJB

メソッドの概要	
execute	WhEJBBean クラスの execute メソッドのインターフェース定義です。

(2) インタフェース WhEJBHome

メソッドの概要	
create	WhEJBBean インスタンスの生成、取得を行います。

(3) WhEJBBean クラス

メソッドの概要	
execute	EJB コンポーネントのメインメソッドです。画面別の WhProcessor<画面名>クラスを呼び出し、画面に対応したトランザクションを実行します。また、トランザクションの実行

	結果である出力メッセージを Web コンポーネントに返却します。
getTermID	会話型トランザクション用端末識別子を取得します。 通常は未使用で、カスタマイズする際に使用します。
releaseTermID	会話型トランザクション用端末識別子を解放します。 通常は未使用で、カスタマイズする際に使用します。
setSessionContext	セッションコンテキストを設定します。 通常は未使用で、カスタマイズする際に使用します。

(4) WhMappingEJBDefault クラス

メソッドの概要	
getProcessorClassName	WhProcessor<画面名>クラス名を取得します。

(5) WhProcessor クラス

メソッドの概要	
continueMultiMessage	トランザクション実行を繰り返すメソッドです。 複数の出力画面を結合するために、トランザクションを複数回実行したい場合に使用します。 通常は未使用で、カスタマイズする際に使用します。
getConnection	ACOS ホストへのコネクションを取得します。
getInstance	WhProcessor<画面名>クラスのインスタンスを取得します。
getRecordFactory	リソースアダプタ用の RecordFactory オブジェクトを取得します。
launchTransaction	トランザクションを実行します。
setConnection	ACOS ホストへのコネクションを設定します。
setRecordFactory	リソースアダプタ用の RecordFactory オブジェクトを設定します。

(6) WhProcessor<画面名>クラス

メソッドの概要	
buildContinueRecord	複数の出力画面を結合する場合に、次のトランザクションを実行するための入力電文を作成します。 通常は未使用で、カスタマイズする際に使用します。
buildInteractionSpec	トランザクション ID と画面名を設定します。
buildMessage	トランザクションの出力電文から出力メッセージを作成します。
buildOutRecord	リソースアダプタ用の出力レコードを作成します。
buildRecord	Web コンポーネントから渡された入力メッセージを元に、入力電文を作成します。 入力フィールドごとに generate<フィールド名>メソッドを呼び出します。
clearBlink	ACOS ホストからの出力電文で、ブリンクしている項目のブリンクを解除します。
clearField	項目の初期化を行います。
convert<フィールド名>	入力値からフィル文字を削除します。

generate<フィールド名>	入力メッセージの該当フィールドから入力値を取り出し、入力電文に値を設定します。
generateOut<フィールド名>	出力電文から値を取り出し、出力メッセージの該当フィールドに値を設定します。
getInMessage	入力メッセージを取得します。
getOutMessage	出力メッセージを取得します。
isMultiMessage	複数の出力画面を結合する場合に true を返却します。既定値では false を返却します。 カスタマイズする際に使用します。
isLastMessage	複数の出力画面を結合する場合に、最後の画面であるかどうかを返します。 通常は未使用で、カスタマイズする際に使用します。
makeOutMessageInstance	出力メッセージのインスタンスを取得します。
setInMessage	入力メッセージを設定します。
setOutMessage	出力メッセージを設定します。
validate<フィールド名>	入力フィールドの値の検証を行います。

(7) WhUtilEJB クラス

メソッドの概要	
getAdapterJndi	アダプタ JNDI 名(EJB コンポーネントからリソースアダプタを呼び出すための名称)を取得します。
getHost	ACOS ホスト名を取得します。
getTerm	端末名を取得します。
getTimeout	タイムアウト時間を取得します。

4.5.共通コンポーネント

本節では、生成した Web アプリケーションの共通コンポーネントについて記述しています。

MFDL 移行プロジェクトで生成した「sbin/<パッケージ名>/common」フォルダに作成されます。

4.5.1.共通コンポーネント

共通コンポーネントには以下のクラスがあります。

クラスの概要	
WhField	画面上のフィールドをあらわすクラスです。フィールドの属性(色、罫線等)や値を保持するクラスです。 各種属性値設定、項目属性の設定、値の設定、複製インスタンス取得を行うメソッドを持ちます。
WhLog	ログ出力を行うクラスです。 log4j.properties に出力先、出力レベル、フォーマット等を指定します。
WhMappingCommonDefault	画面名に関連付けられたクラス名を取得するクラスです。 画面名に対応する WhMessage<画面名>クラス名を取得します。
WhMappingCommon	WhMessage<画面名>クラスをカスタマイズしたときに、画面名との関連付けを変更するために使用します。詳細は「5. アプリケーションのカスタマイズ」を参照してください。
WhMessage	Web コンポーネントと EJB コンポーネント間での通信媒体となる WhMessage<画面名>クラスのスーパークラスです。
WhMessage<画面名>	Web コンポーネントと EJB コンポーネント間の通信媒体クラスです。 画面のフィールドに対応する WhField クラスのインスタンスを保持し、画面名取得、次画面名取得,、フィールド取得、PF キー取得,、TrnsPFKey 用のトランザクション ID 取得などを行います。
WhUtilCommon	共通コンポーネントで使用するユーティリティクラスです。
WhException	Exception 処理を行うクラスです。

(1) WhField クラス

フィールドの概要	
COLOR_DEFAULT	規定色(0)を表し、既定の表示色は黒色です。
COLOR_RED	赤色(1)を表し、規定の表示色は赤色です。
COLOR_GREEN	緑色(2)を表し、規定の表示色は緑色です。
COLOR_YELLOW	黄色(3)を表し、規定の表示色は黄色です。
COLOR_BLUE	青色(4)を表し、規定の表示色は青色です。
COLOR_MAGENTA	紫色(5)を表し、規定の表示色は紫色です。
COLOR_CYAN	水色(6)を表し、規定の表示色は水色です。
COLOR_WHITE	白色(7)を表し、規定の表示色は黒色です。
BORDER_NO	罫線無し(0)を表します。
BORDER_TOP	上罫線(1)を表します。
BORDER_RIGHT	右罫線(2)を表します。
BORDER_BOTTOM	下罫線(4)を表します。
BORDER_LEFT	左罫線(8)を表します。

BORDER_ALL	上下左右の全ての罫線である枠線(15)を表します。
ENCODE_TEXT	表示テキスト(0) を表します。
ENCODE_INPUT_TEXT	入力テキスト(1) を表します。
ENCODE_INPUT_HIDDEN	入力操作ができない入力テキスト(2) を表します。
ENCODE_PASSWORD	パスワードテキスト(3) を表します。
EFFECT_NO	文字飾り無(0) を表します。
EFFECT_REVERSE	文字色と背景色の逆転(1) を表します。
EFFECT_BLINK	ブリンク(2) を表します。
fieldname	項目名を表します。
fieldsize	画面上の項目サイズを表します。
lfldsize	コピー句の項目サイズを表します。
value	項目の値を表します。
メソッドの概要	
analyzeAttribute	項目属性情報を設定します。
getColorClass	色のクラス属性定義名を取得します。
getColorNo	色番号を取得します。
getCompareCondition	Compare チェック条件を取得します。
getCompareValue	Compare チェック値を取得します。
getCopyField	複製インスタンスを取得します。
getDpnt	小数点桁数を取得します。
getEncodeStyle	テキスト表示種別を取得します。
getError	エラーの有無を取得します。
getFieldName	フィールド名称を取得します。
getFieldSize	項目のサイズを取得します。
getFil	フィル文字を取得します。
getInitValue	初期値を取得します。
getLfldSize	COBOL COPY 句での項目サイズを取得します。
getRangeConditionFrom	Range チェックの最小値を取得します。
getRangeConditionTo	Range チェックの最大値を取得します。
getRegularColorClass	色番号を元に色のクラス属性定義名を取得します。
getValue	設定されている値を取得します。
getValuesCondition	Values チェック項目を取得します。
isBitBorder	罫線指定有無を取得します。
isBlink	ブリンク表示有無を取得します。
isBold	フォントボールド表示有無を取得します。
isDesigInitCheck	入力値変更チェック有無を取得します。
isEquCheck	EQU チェック有無を取得します。
isIndispensableCheck	必須入力チェック有無を取得します。
isIniofInitCheck	MFDL 定義の PFLD の入出力属性が IN または IOF の時、入力値変更チェック有無を取得します。
isKanjiCheck	漢字チェック有無を取得します。

isNumCheck	数値チェック有無を取得します。
isReverse	文字色と背景色の反転有無を取得します。
isSetValue	値の設定有無を取得します。
isSignCheck	符号チェック有無を取得します。
resetValue	値を空にします。
setBlink	ブリンク表示有無を設定します。WhUtilWebAp クラスの Html タグ生成で参照します。
setBorder	罫線を設定します。WhUtilWebAp クラスの Html タグ生成で参照します。
setColor	色を設定します。WhUtilWebAp クラスの Html タグ生成で参照します。
setColorClass	色のクラス属性定義名を設定します。WhUtilWebAp クラスの Html タグ生成で参照します。
setColorNo	色番号を設定します。
setCompareCondition	Compare チェック項目を設定します。WhUtilCommon クラスの入力チェックで参照します。
setDesigInitCheck	入力値変更チェック有無を設定します。WhUtilCommon クラスの入力チェックで参照します。
setDpnt	小数点桁数を設定します。WhUtilCommon クラスの入力チェックで参照します。
setEncodeStyle	テキスト表示種別を設定します。WhUtilWebAp クラスの Html タグ生成で参照します。
setEquCheck	EQU チェック有無を設定します。WhUtilCommon クラスの入力チェックで参照します。
setErrorData	ブリンク表示や!!!表示のエラー設定を行います。
setFieldName	フィールド名称を設定します。WhUtilCommon クラスの入力チェックで参照します。
setFieldSize	サイズを設定します。WhUtilWebAp クラスの Html タグ生成で参照します。
setFil	フィル文字を設定します。WhUtilCommon クラスの入力チェックで参照します。
setIndispensableCheck	必須入力チェック有無を設定します。WhUtilCommon クラスの入力チェックで参照します。
setIniofInitCheck	MFDL 定義の PFLD の入出力属性が IN または IOF の時、入力値変更チェック有無を設定します。WhUtilCommon クラスの入力チェックで参照します。
setInitValue	初期値を設定します。WhUtilCommon クラスの入力チェックで参照します。
setKanjiCheck	漢字チェック有無を設定します。WhUtilCommon クラスの入力チェックで参照します。
setLfldSize	コピー句サイズを設定します。WhUtilCommon クラスの入力チェックで参照します。
setNumCheck	数値チェック有無を設定します。WhUtilCommon クラスの入力チェックで参照します。
setRangeCondition	Range チェック項目を設定します。WhUtilCommon クラスの入力チェックで参照します。
setSignCheck	符号チェック有無を設定します。WhUtilCommon クラスの入力チェックで参照します。

setValue	値を設定します。WhUtilWebAp クラスの Html タグ生成で参照します。
setValuesCondition	Values チェック項目を設定します。WhUtilCommon クラスの入力チェックで参照します。

(2) WhLog クラス

メソッドの概要	
initProperty	プロパティファイルの読み込みと設定を行います。
putlog	ログ出力を行います。

(3) WhMappingCommonDefault クラス

メソッドの概要	
getMessageClassName	WhMessage<画面名>クラス名を取得します。

(4) WhMessage クラス

メソッドの概要	
getInstance	WhMessage<画面名>クラスのインスタンスを取得します。
getSendReceive	送受信モードを取得します。
getTermID	端末識別子を取得します。 通常は未使用で、カスタマイズする際に使用します。
getTermName	外部端末識別名を取得します。 通常は未使用で、カスタマイズする際に使用します。
isGetTerm	端末識別子取得処理の有無を取得します。
isReleaseTerm	端末識別子解放処理の有無を取得します。
isTransaction	トランザクション実行の有無を取得します。
setGetTerm	端末識別子取得処理の有無を設定します。 通常は未使用で、カスタマイズする際に使用します。
setReleaseTerm	端末識別子解放処理の有無を設定します。 通常は未使用で、カスタマイズする際に使用します。
setSendReceive	送受信モードを設定します。 通常は未使用で、カスタマイズする際に使用します。
setTermID	端末識別子を設定します。 通常は未使用で、カスタマイズする際に使用します。
setTermName	外部端末識別名を設定します。 通常は未使用で、カスタマイズする際に使用します。
setTransaction	トランザクション実行の有無を設定します。 通常は未使用で、カスタマイズする際に使用します。
getArraySize	出力メッセージ数を取得します。 通常は未使用で、カスタマイズする際に使用します。
getArrayOutMsg	出力メッセージを取得します。 通常は未使用で、カスタマイズする際に使用します。

(5) WhMessage<画面名>クラス

メソッドの概要	
WhMessage<画面名>	コンストラクタです。全てのフィールドの情報を初期化します。
clearFocus	フォーカスを設定するフィールド名を削除します。
get<フィールド名>	フィールド名に対応する WhField オブジェクトを取得します。
getCopyInstance	複製インスタンスを取得します。
getERRORMSG	エラーメッセージ項目情報を取得します。
getFocus	フォーカスを設定するフィールド名を取得します。
getMessageName	画面名を取得します。
getNextMessageName	次画面名を取得します。
getPFKeyID	PF キーID を取得します。
getPFKeyName	PF キー名称を取得します。
getTrnsPFKey	TrnsPFKey 用のトランザクション ID を取得します。
setFocus	フォーカスを設定するフィールド名を設定します。
setNextMessageName	次画面名を設定します。
setPFKeyID	PF キーID を設定します。
setPFKeyName	PF キー名称を設定します。
setTrnsPFKey	TrnsPFKey 用のトランザクション ID を設定します。

(6) WhUtilCommon クラス

メソッドの概要	
chgDecData	数値データから不要文字を削除します。
chgDecDataSign	数値データから＋－を含めて不要文字を削除します。
chkInput	入力チェック処理を行います。 以下の isXX メソッドを呼び出します。
convertANK	ANK 文字用に全角文字を半角文字へ変換します。 通常は未使用で、カスタマイズする際に使用します。
delFillChar	フィル文字を削除します。
generateMessage	メッセージ本文へキーワードを設定します。
getDateS	和暦を取得します。
getDateTime	指定フォーマットで現在の日付時刻を取得します。
getPFKeyName	PF キーID の文字列を取得します。
isAlpha	半角アルファベット文字チェックを行います。 通常は未使用で、カスタマイズで使用できます。
isBit	ビット演算を行います。
isCompare	Compare チェックを行います。
isDecData	数値チェックを行います。
isDecDataSp	数値及び特殊文字チェックを行います。
isDecimal	小数点チェックを行います。
isDecSignData	数値項目符号チェックを行います。
isDpnt	数値項目桁落ちチェックを行います。

isDpntHigh	数値項目整数部桁落ちチェックを行います。
isDpntLow	数値項目小数部桁落ちチェックを行います。
isFil	Fil 文字のみかどうかのチェックを行います。
isIndispensable	必須入力チェックを行います。
isKanji	全角文字チェックを行います。
isLength	文字列長チェックを行います。
isNull	Null 文字チェックを行います。
isNum	数値属性チェックを行います。
isSpace	空白チェックを行います。
isValues	Values チェックを行います。
loadProperties	プロパティファイルを読込みます。

(7) WhException クラス

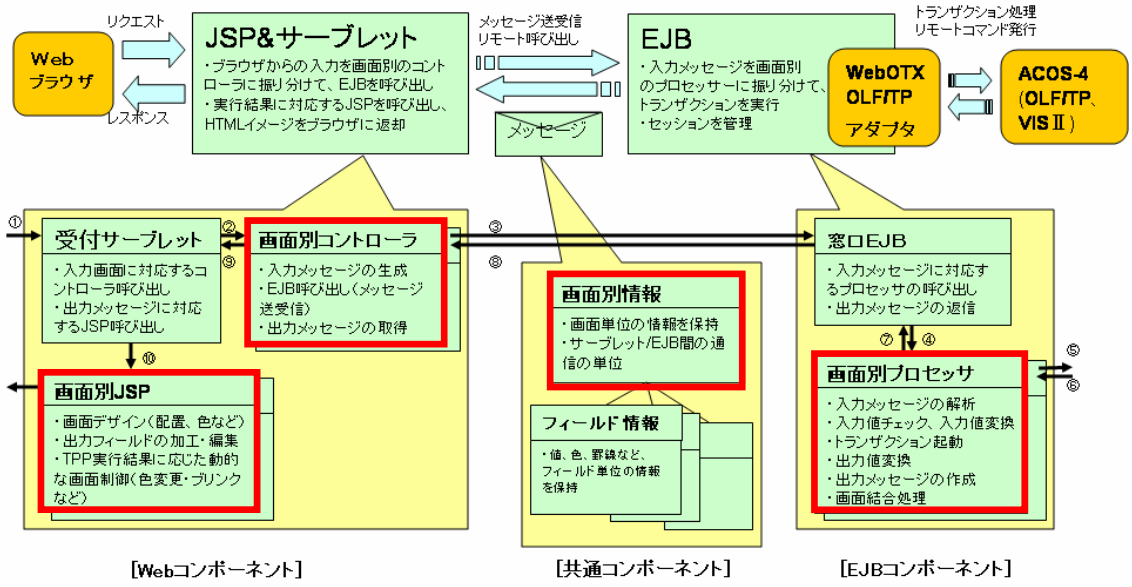
メソッドの概要	
getCauseString	発生した例外の原因のメッセージを取得します。
getClassName	エラーメッセージを設定したクラス名を取得します。 通常は未使用で、カスタマイズする際に使用します。
getError	エラーメッセージを取得します。
getErrorDate	エラーが発生したタイムスタンプを取得します。 通常は未使用で、カスタマイズで使用できます。
getExceptionName	発生した例外名を取得します。
getMessageID	メッセージ ID を取得します。
getMessageString	発生した例外のエラーメッセージを取得します。
getStackTraceString	スタックトレースを取得します。
setCauseString	発生した例外の原因のメッセージを設定します。
setClassName	エラーメッセージを設定したクラス名を設定します。
setError	エラーメッセージを設定します。
setExceptionName	発生した例外名を設定します。
setMessageString	発生した例外のエラーメッセージを設定します。
setPropertyMessage	エラーメッセージを組み立てます。
setStackTraceString	スタックトレースの組立てを行います。

5.アプリケーションのカスタマイズ

5.1.カスタマイズ例の概要

本節では、各コンポーネントやエラーメッセージに対するカスタマイズ例について記述しています。また、アプリケーションの高度なカスタマイズ例についても記述しています。

5.1.1.カスタマイズ例の概要



上記の各コンポーネントに対するカスタマイズ例は、以下に記述します。

- 画面別 JSP ・・・・・・ 画面デザイン(配置、色、プルダウンやラジオボタン形式)を変更します。
  5.2.2画面の入力項目をプルダウン形式に変更する例
  5.2.3画面の入力項目をラジオボタン形式に変更する例
  5.2.4画面項目の文字色と罫線を変更する例
- 画面別コントローラ ・・・・・・ 入力チェックやエラーメッセージの変更をします。
  5.3.2画面の入力値に対してチェックを追加する例
- 画面別情報 ・・・・・・ フィールド情報として保持している入力チェック内容の変更をします。
  5.3.3画面の入力値に対して全桁数が入力されたかをチェックする例
- 画面別プロセッサ ・・・・・・ ACOS との通信を行う際の入力電文、出力電文の変更をします。

エラーメッセージのカスタマイズ例として、以下を記述します。

- 5.4.2エラーメッセージを変更する例

その他の高度なカスタマイズ例として、以下を記述します。

- 5.5.1ACOSホストからの複数回の出力を結合する例
- 5.5.2会話型トランザクションを使用する例
- 5.5.3入出力項目を追加・削除する例

5.2.画面(JSP)のカスタマイズ例

本節では、画面(JSP)のカスタマイズ例を記述しています。

5.2.1.画面(JSP)のカスタマイズ方法

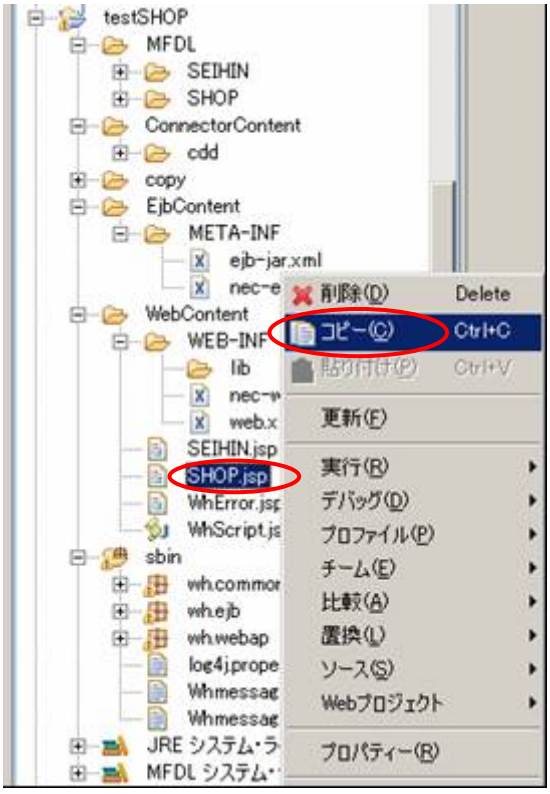
画面(JSP)のカスタマイズでは、<画面名>.jsp を別名ファイルにコピーしてカスタマイズします。

(1) JSP のカスタマイズ

MFDL 移行プロジェクトで作成した<画面名>.jsp を「Developer's Studio」の「プロジェクトエクスプローラ」で開き、別名ファイルにコピーしてからカスタマイズします。

※ ここでは、カスタマイズする<画面名>.jsp が"SHOP.jsp"の例を示します。

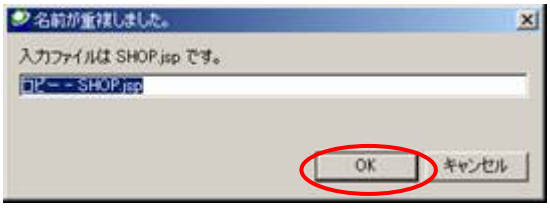
① <プロジェクト名>/WebContent/SHOP.jsp を選択して、右クリックメニューから「コピー」を選択します。



② 再度 SHOP.jsp を選択して、右クリックメニューから「貼り付け」を選択します。

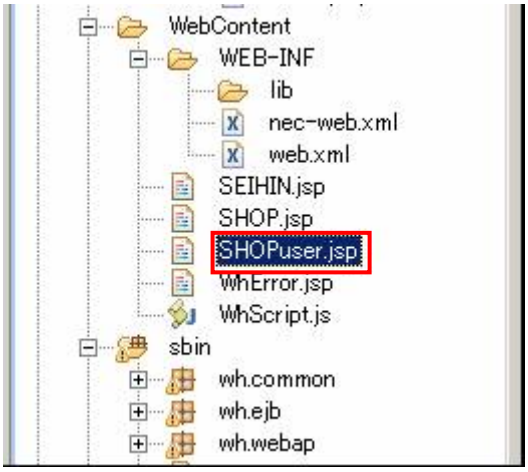
③ 「名前が重複しました」画面が表示されるので、コピー後のファイル名を入力し、「OK」ボタンを押下します。

※ ここでは、ファイル名に"SHOPuser.jsp"とする例を示します。



④ 入力したファイル名で別名ファイルが作成されます。

以上で別名ファイルが作成されますので、ダブルクリックでファイルを開き、必要なカスタマイズを行ってください。



(2) WhMappingWebAp クラスの変更

MFDL 移行プロジェクトが生成するアプリケーションは、「画面名」に関連付けられた<画面名>.jsp を使用して画面を表示します。カスタマイズした JSP を利用するためには、WhMappingWebAp クラスでこの関連付けを変更する必要があります。

MFDL移行プロジェクトで作成した<パッケージ名>.webap配下のWhMappingWebAp.javaを「プロジェクトエクスプローラ」で開き、「(1)JSPのカスタマイズ」で作成したJSPを「画面名」に関連付けます。

MFDL 移行プロジェクトで作成した SHOP.jsp をコピーして、SHOPuser.jsp をカスタマイズした場合の WhMappingWebAp クラスの変更例を以下に示します。

```
public class WhMappingWebAp extends WhMappingWebApDefault {

        /**
         * コンストラクタ
         */
        public WhMappingWebAp() {}

        /**
         * カスタマイズ画面呼び出し
         */
        public static String getJspName(String msgName){          ・・・①
                if (msgName.equals("SHOP") == true) {
                        return "SHOPuser.jsp";                    ・・・②
                } else {
                        return msgName + ".jsp";
                }
        }
}
```

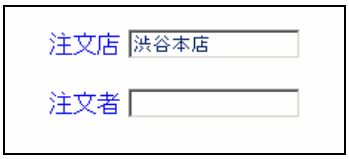
WhMappingWebApDefault クラスの getJspName メソッドは、「画面名」に関連付けられた<画面名>.jsp を返却するメソッドです(①)。継承した WhMappingWebAp クラスでこのメソッドをカスタマイズし、SHOPuser.jsp を返却するようにします。上記の例では、画面名が"SHOP"の場合に、"SHOPuser.jsp"を返却しています(②)。

5.2.2.画面の入力項目をプルダウン形式に変更する例

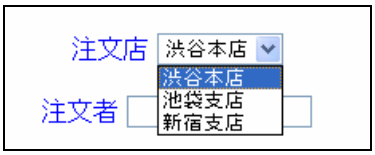
通常、Web アプリケーション画面の入力は打鍵入力しますが、入力ミスを防ぐためプルダウン形式に変更できます。ここでは、入力項目をプルダウン形式に変更する例を示します。

画面イメージ

【カスタマイズ前】



【カスタマイズ後】



(1) JSP ファイルのカスタマイズ

JSP ファイルをカスタマイズする場合、対象ファイルを直接編集せずに別名ファイルへコピーして、別名ファイルをカスタマイズしてください。

なお、以下の例ではポイントとなるタグや属性を示しています。

【MFDL移行プロジェクトが生成したJSPファイル】

例：SHOP.jsp

```
<td class='color_blue'>
  注文店
</td>
<td>
  <%= WhUtilWebAp.getHtmlTag(message.getSHOP(),did) %>
</td>
```

WhUtilWebAp.getHtmlTag メソッドは、フィールドの属性に応じた HTML タグを自動生成するメソッドです。上記の SHOP フィールドの例では、以下のような HTML タグが自動生成されます。

<input type=text name="SHOP" size="20" maxlength="20" value="" class='color_blue'>

【カスタマイズした JSP ファイル】

例：SHOPuser.jsp

```
<td class='color_blue'>
  注文店
</td>
<td class='color_yellow'>
  <select name="SHOP">                                          ・・・①
    <option value="渋谷本店" selected>渋谷本店</option>
    <option value="池袋支店">池袋支店</option>                    ・・・②
    <option value="新宿支店">新宿支店</option>
  </select>
</td>
```

上記の例では、SHOP フィールドの HTML タグを<input>から<select>へ変更し、プルダウン形式にしています(①)。 入力項目の選択肢として“渋谷本店”、“池袋支店”、“新宿支店”を設定しています(②)。

(2) マッピングファイルの変更

JSPをカスタマイズした場合は、マッピングファイル(WhMappingWebApクラス)を変更します。変更方法については、「5.2.1(2)WhMappingWebApクラスの変更」を参照してください。

(3) アプリケーションの配備

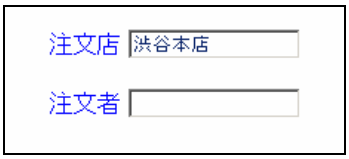
Webアプリケーションのパッケージ及び配備を行います。「3.5.3Webアプリケーションの配備」を参照してください。

5.2.3.画面の入力項目をラジオボタン形式に変更する例

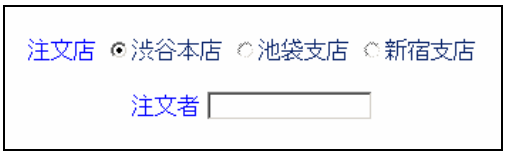
通常、Web アプリケーション画面の入力は打鍵入力しますが、入力ミスを防ぐためラジオボタン形式に変更できます。ここでは、入力項目をラジオボタン形式に変更する例を示します。

画面イメージ

【カスタマイズ前】



【カスタマイズ後】



(1) JSP ファイルのカスタマイズ

JSP ファイルをカスタマイズする場合、対象ファイルを直接編集せずに別名ファイルへコピーして、別名ファイルをカスタマイズしてください。

なお、以下の例ではポイントとなるタグや属性を示しています。

【MFDL移行プロジェクトが生成したJSPファイル】

例：SHOP.jsp

```
<td class='color_blue'>
  注文店
</td>
<td>
  <%= WhUtilWebAp.getHtmlTag(message.getSHOP(),did) %>
<td>
```

WhUtilWebAp.getHtmlTag メソッドは、フィールドの属性に応じた HTML タグを自動生成するメソッドです。上記の SHOP フィールドの例では、以下のような HTML タグが自動生成されます。

<input type=text name="SHOP" size="20" maxlength="20" value="" class='color_blue'>

【カスタマイズした JSP ファイル】

例：SHOPuser.jsp

```
<td class='color_blue'>
  注文店
</td>
<td class='color_yellow'>
  <input type=radio name="SHOP" value="渋谷本店" checked>渋谷本店
  <input type=radio name="SHOP" value="池袋支店">池袋支店
  <input type=radio name="SHOP" value="新宿支店">新宿支店
<td>
```

・・・①

上記の例では、SHOP フィールドの HTML タグを<input type=radio>へ変更し、ラジオボタン形式にしています(①)。 入力項目の選択肢として“渋谷本店”、“池袋支店”、“新宿支店”を設定しています。

(2) マッピングファイルの変更

JSPをカスタマイズした場合は、マッピングファイル(WhMappingWebApクラス)を変更します。変更方法については、「5.2.1(2)WhMappingWebApクラスの変更」を参照してください。

(3) アプリケーションの配備

Webアプリケーションのパッケージおよび配備を行います。配備の方法については、「3.5.3Webアプリケーションの配備」を参照してください。

5.2.4.画面項目の文字色と罫線を変更する例

Web アプリケーション画面はデザインを整えるために文字色や罫線を設定します。

ここでは、項目の文字色と罫線を変更する例を示します。

画面イメージ

【カスタマイズ前】

【カスタマイズ後】

(1) JSP ファイルのカスタマイズ

JSP ファイルをカスタマイズする場合、対象ファイルを直接編集せずに別名ファイルへコピーして、別名ファイルをカスタマイズしてください。

なお、以下の例ではポイントとなるタグや属性を示しています。

【MFDL移行プロジェクトが生成したJSPファイル】

例:SHOP.jsp

```
<td class='color_green'>                                          ・・・①
      注文番号
</td>
<td style='border-width: 1 0 1 1;>                                ・・・②
      <%= WhUtilWebAp.getHtmlTag(message.getORDER(),did) %>
</td>
```

文字色は、td タグの class 属性で指定します。罫線は、td タグの style 属性 border-width で指定します。上記の例では、「注文番号」の文字色は緑色(①)、ORDER フィールドの罫線は上下左の罫線(②)を表示します。なお、入出力属性フィールドの文字色は AP(Java ソース)の値が表示されるため、JSP ファイルでの設定は不要です。

【カスタマイズした JSP ファイル】

例:SHOPuser.jsp

```
<td class='color_blue'>                                           ・・・①
      注文番号
</td>
<td style='border-width: 0 0 1 0;>                                ・・・②
      <%= WhUtilWebAp.getHtmlTag(message.getORDER(),did) %>
</td>
```

上記の例では、文字色を青色に (①)、罫線を下罫線のみに設定しています(②)。

なお、class 属性で指定する各文字色は、スタイルシート(WhStylesheet.css)で値を設定しますが、Web 画面の背景色が白色のため ETOS 定義色の color_white は、既定値での表示色を #000000(黒色)としているので留意してください。

(2) WhMessage<画面名>ファイルのカスタマイズ

カスタマイズするフィールドが入出力項目の場合は、文字色や罫線の値を AP(Java ソース)でも設定しているため、その値を変更する必要があります。カスタマイズ対象は WhMessage<画面名>のコンストラクタになります。

以下にカスタマイズ例を示します。実際のカスタマイズ手順などについては「5.3.1AP(Javaソース)のカスタマイズ方法」を参照してください。

【MFDL移行プロジェクトが生成したAP(Javaソース)ファイル】

例:WhMessageSHOP.java

```
        /**
         * クラス内に保持する全ての WhField インスタンスを初期化する
         */
        public WhMessageSHOP() {
```

```
                fieldname = "ORDER";                                    ・・・①
                initvalue = " ";                                   ・・・②
                fieldsize = 5;                                          ・・・③
                lfldsize = 5;                                           ・・・④
                ORDER = new WhField(fieldname, initvalue, fieldsize, lfldsize); ・・⑤
                ORDER.setBorder(WhField.BORDER_ALL);                    ・・・⑥
                ORDER.setColor(WhField.COLOR_GREEN);                    ・・・⑦
                ORDER.setEncodeStyle(WhField.ENCODE_TEXT);              ・・・⑧
                ORDER.setFil("");                                       ・・・⑨
        }
```

WhMessage<画面名>のコンストラクタは、入出力フィールドの入出力値を保持する WhField クラスを初期化します。上記の ORDER フィールドの例では、項目名に"ORDER"(①)、初期値に空白(②)、項目長に 5(③)、コピー句のデータ長に 5(④)を設定しています(⑤)。 また、罫線に BORDER_ALL(上下左右罫線)(⑥)を設定し、文字色に COLOR_GREEN(緑色)(⑦)を、JSP 表示時の INPUT タグ TYPE 属性に ENCODE_TEXT を(⑧)、フィル文字に""(⑨)を設定しています。

文字色や罫線で使用している定数については、「4.5.1(1)WhFieldクラス」を参照してください。

【カスタマイズした AP(Java ソース)ファイル】

例として WhMessageSHOPuser.java

```
package wh.common;

public class WhMessageSHOPuser extends WhMessageSHOP {

        /**
         * コンストラクタ
         */
        public void WhMessageSHOPuser() {

                super();                                        ・・・①
                ORDER.setColor(WhField.COLOR_RED);              ・・・②
                ORDER.setBorder(WhField.BORDER_BOTTOM);         ・・・③
        }
}
```

上記の例では、親クラスのコンストラクタを実行することでメッセージクラスの ORDER フィールドに初期の設定を行っています(①)。 その後、文字色に COLOR_RED(赤色)を設定し(②)、罫線に BORDER_BOTTOM(下罫線)を設定しています(③)。ただし、この罫線の設定は、表示のために使用されません。JSP ファイルで設定した罫線表示になります。

文字色や罫線で使用している定数については「4.5.1(1)WhFieldクラス」を参照してください。

(3) マッピングファイルの変更

JSPをカスタマイズした場合は、マッピングファイル(WhMappingWebApクラス)を変更します。変更方法については、「5.2.1(2)WhMappingWebApクラスの変更」を参照してください。

(4) アプリケーションの配備

EJBアプリケーションとWebアプリケーションのパッケージ及び配備を行います。配備の方法については、「3.4.3EJBアプリケーションの配備」、および「3.5.3Webアプリケーションの配備」を参照してください。

5.2.5.エラー画面を変更する例

Web アプリケーション実行時に例外が発生すると、その内容をエラー画面で表示します。エラー画面の表示内容は変更することができます。ここでは、Web アプリケーションの例外をシステム担当に連絡する内容に変更する例を示します。

画面イメージ

【カスタマイズ前】

【カスタマイズ後】

(1) WhError.jsp ファイルのカスタマイズ

WhError.jsp ファイルをカスタマイズする場合、対象ファイルを直接編集せずに別名ファイルへコピーして、別名ファイルをカスタマイズしてください。

なお、以下の例ではポイントとなるタグや属性を示しています。

【MFDL移行プロジェクトが生成したJSPファイル】

例:WhError.jsp

```
    :
<form>
<span style='color:Red'>例外が発生しました。</span>              ・・・①
<pre>
<table>
    :
  <tr>
    <td colspan=2>例外のメッセージ：</td>
  </tr>
  <tr>
    <td> </td>
    <td><%= ex.getMessageString() %></td>                       ・・・②
  </tr>
    :
  <tr>
    <td colspan=2>例外の原因：</td>
  </tr>
  <tr>
    <td> </td>
    <td><%= ex.getCauseString() %></td>                         ・・・③
  </tr>
    :
  <tr>
    <td colspan=2>スタックトレース：</td>
  </tr>
</table>
    <%= ex.getStackTraceString() %>                             ・・・④
</pre>
    :
```

例外が発生したことを通知する固定のメッセージ(①)です。

発生した例外(exception)の情報は、WhException クラスに保持しています。このクラスから例外のメッセージを取得するために getMessageString メソッドを呼び出しています(②)。同様に例外の原因を取得するために getCauseString メソッドを(③)、スタックトレースを取得するために getStackTraceString メソッドを(④)呼び出しています。

【カスタマイズした JSP ファイル】

例:WhErrorUser.jsp

```
    :
<form>
<span style='color:Red'>
Web アプリケーションエラー！<br>
<br>
次の内容をシステム担当へ連絡して下さい。<br>
・エラー発生時刻<br>
・画面名<br>
・エラーメッセージ<br>
・操作内容
</span>
<pre>
<table>
    :
  <tr>
    <td colspan=2>エラーメッセージ：</td>
  </tr>
  <tr>
    <td> </td>
    <td><%= ex.getMessageString() %></td>
  </tr>
    :
~~  <tr>~~
~~    <td colspan=2>例外の原因：</td>~~
~~  </tr>~~
~~  <tr>~~
~~    <td> </td>~~
~~    <td><%= ex.getCauseString() %></td>~~
~~  </tr>~~
~~    :~~
~~  <tr>~~
~~    <td colspan=2>スタックトレース：</td>~~
~~  </tr>~~
</table>
~~    <%= ex.getStackTraceString() %>~~
</pre>
    :
```

・・・①

・・・②

・・・③

上記の例では、固定のメッセージを変更し(①)、例外のメッセージのタイトルを変更しています(②)。

また、例外の原因とスタックトレースを削除しています(③)。

なお、WhException クラスから取得した情報を加工したり、独自の情報を表示するなどのカスタマイズが可能です。

(2) マッピングファイルの変更

カスタマイズした JSP を利用するためには、WhMappingWebAp クラスでこの画面の関連付けを変更する必要があります。

MFDL移行プロジェクトで作成した<パッケージ名>.webap配下のWhMappingWebAp.javaを「プロジェクトエクスプローラ」で開き、「(1)WhError.jspファイルのカスタマイズ」で作成した別名ファイルを関連付けます。

MFDL 移行プロジェクトで作成した WhError.jsp をコピーして、WhErrorUser.jsp をカスタマイズした場合の WhMappingWebAp クラスの変更例を以下に示します。

```
public class WhMappingWebAp extends WhMappingWebApDefault {

        /**
         * コンストラクタ
         */
```

```
        public WhMappingWebAp()  {}

        /**
         * エラー画面呼び出し
         */
        public static String getErrJspName(){                    ・・・①
                return "WhErrorUser.jsp";                        ・・・②
        }
}
```

WhMappingWebApDefault クラスの getErrJspName メソッドは、エラー画面として関連付けられた WhError.jsp を返却するメソッドです(①)。上記の例では、継承した WhMappingWebAp クラスでこのメソッドをカスタマイズし、WhErrorUser.jsp を返却するようにしています(②)。

(3) アプリケーションの配備

Webアプリケーションのパッケージ及び配備を行います。「3.5.3Webアプリケーションの配備」を参照してください。

5.3.AP(Java ソース)のカスタマイズ例

本節では、AP(Java ソース)のカスタマイズ例を記述しています。

5.3.1.AP(Java ソース)のカスタマイズ方法

AP(Java ソース)のカスタマイズでは、Java の継承を利用して別名ファイルをカスタマイズします。

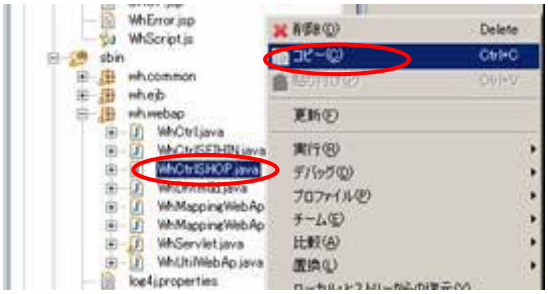
(1) WhCtrl<画面名>、WhProcessor<画面名>、WhMessage<画面名>クラスのカスタマイズ

MFDL 移行プロジェクトで作成した AP(Java ソース)を「プロジェクトエクスプローラ」で開き、別名ファイルにコピーし、java の継承を利用してカスタマイズします。

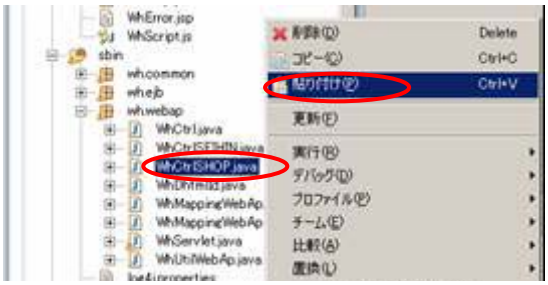
カスタマイズ可能なファイルは、以下のファイルです。
WhCtrl<画面名>java、WhProcessor<画面名>java、WhMessage<画面名>java

※ ここでは、カスタマイズする AP(Java ソース)が"WhCtrlSHOP.java"の例を示します。

① <プロジェクト名>/sbin/<パッケージ名>/webap/WhCtrlSHOP.java を選択して、右クリックメニューから「コピー」を選択します。

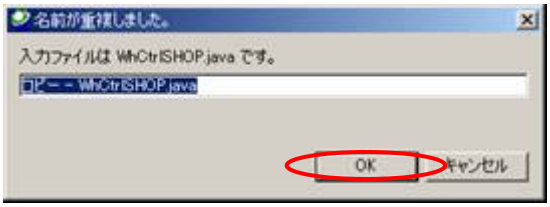


② 再度 WhCtrlSHOP.java を選択して、右クリックメニューから「貼り付け」を選択します。

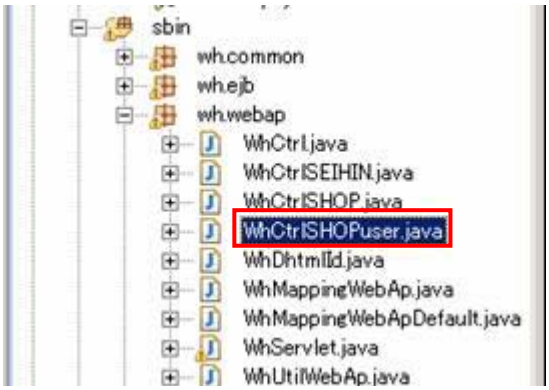


③ 「名前が重複しました」画面が表示されるので、コピー後のファイル名を入力し、「OK」ボタンを押下します。

※ ここでは、ファイル名に"WhCtrlSHOPuser.jsp"とする例を示します。



④ 入力したファイル名で別名ファイルが作成されます。



以上で別名ファイルが作成されますので、ダブルクリックでファイルを開き、必要なカスタマイズを行ってください。

オーバーライドするメソッドのみを残して、他は削除してください。

(2) WhMapping<コンポーネント名>クラスの変更

MFDL 移行プロジェクトが生成するアプリケーションは、「画面名」に対応する WhCtrl<画面名>.java、WhMessage<画面名>.java、WhProcessor<画面名>.java を関連付けています。カスタマイズしたソースを利用するためには、WhMapping<コンポーネント名>クラスで、この関連付けを変更する必要があります。

各 AP(Java ソース)の関連付けは以下の WhMapping<コンポーネント名>クラスで変更します。

【カスタマイズするソース と 関連付けする WhMapping<コンポーネント名>クラスの対応】

WhCtrl<画面名>.java	・・・・・・・・・	WhMappingWebAP.java
WhProcessor<画面名>.java	・・・・・・・・・	WhMappingEJB.java
WhMessage<画面名>.java	・・・・・・・・・	WhMappingCommon.java

MFDL 移行プロジェクトで作成したパッケージ配下の WhMapping<コンポーネント名>.java を「プロジェクトエクスプローラ」で開き、(1)で作成した別名ファイルを「画面名」に関連付けます。

MFDL 移行プロジェクトで作成した WhProcessorSHOP.java をコピーして、WhProcessorSHOPuser.java をカスタマイズした場合の WhMappingEJB クラスの変更例を以下に示します。

```
public class WhMappingEJB extends WhMappingEJBDefault {

        /**
         * コンストラクタ
         */
        public WhMappingEJB() {}

        /**
         * カスタマイズ画面呼び出し
         */
        public static String getProcessorClassName(String msgName){      ・・・①
                if (msgName.equals("SHOP") == true) {
                        return "wh.ejb.WhProcessorSHOPuser";              ・・・②
                } else {
                        return "wh.ejb.WhProcessor" + msgName;
                }
        }
}
```

WhMappingEJB クラスの getProcessorClassName メソッドは、画面名に対応する WhProcessor<画面名>のクラス名を返却するメソッドです(①)。 上記の例では、画面名が"SHOP"の場合に"wh.ejb.WhProcessorSHOPuser"を返却するように変更しています(②)。

5.3.2.画面の入力値に対してチェックを追加する例

Web アプリケーション画面の入力値に対して、そのチェック処理を Web アプリケーション上で行いますが、それとは別のチェック処理を追加することが出来ます。

ここでは、「注文店」項目に「本店」と入力された場合をエラーとする変更例を示します。

(1) WhCtrl<画面名>ファイルのカスタマイズ

入力値のチェック処理に関するカスタマイズは、WhCtrl<画面名>クラスの validate<フィールド名>メソッドで行います。

WhCtrl<画面名>クラスをカスタマイズする場合、対象ファイルを直接編集せず、カスタマイズ元のクラスを継承した別名ファイルを新規に作成してください。別名ファイルにはカスタマイズするメソッドのみをオーバーライドで記述します。

【MFDL移行プロジェクトが生成したAP(Javaソース)ファイル】

例:WhCtrlSHOP.java

```
        protected void validateSHOP(String chkValue,WhField field) throws WhException {
                String check = chkValue;
                if (WhUtilCommon.isNull(check) == true) {                ・・・①
                        field.setErrorData();
                        throw new WhException(WhUtilCommon.OPT_ERR_NULL, "SHOP");  ②
                }
                int errcode = WhUtilCommon.chkInput(check, field);       ・・・③
                if (errcode != WhUtilCommon.OPT_NORMAL &&                ・・・④
                    errcode != WhUtilCommon.OPT_ERR_NULL) {
                        field.setErrorData();
                        throw new WhException(errcode, "SHOP");          ・・・⑤
                }
```

validate<フィールド名>メソッドは、入力フィールドの入力値に対するチェック処理を行います。

上記の SHOP フィールドの例では、まず入力項目が存在するかどうかを判断して(①)、存在しない場合は例外をスローしています(②)。 続いて WhUtilCommon.chkInput メソッドによる入力値についての入力チェック処理を呼び出しています(③)。 入力チェック処理でエラーと判断された場合(④)は例外をスローして呼び出し元へ返しています(⑤)。

【カスタマイズした AP(Java ソース)ファイル】

例:WhCtrlSHOPuser.java

```
public class WhCtrlSHOPuser extends WhCtrlSHOP {
        protected void validateSHOP(String chkValue,WhField field)       ・・・①
                                                    throws WhException {
                super.validateSHOP(chkValue, field);                     ・・・②

                //入力値が"本店"の場合エラーとする
                if (chkValue.equals("本店") == true) {                    ・・・③
                        field.setErrorData();
                        throw new WhException
                                ("SHOP 項目に「本店」は入力できません");
                }
        }
}
```

上記の例では、WhCtrlSHOP クラスの validateSHOP メソッドをオーバーライドしています(①)。親クラスの validateSHOP メソッドを実行して既存の入力チェック処理を行い(②)、入力値が"本店"である場合は、エラーメッセージを設定した例外をスローしています(③)。

(2) マッピングファイルの変更

AP(Javaソース)をカスタマイズした場合は、マッピングファイル(WhMappingWebApクラス)を変更します。変更方法については、「5.3.1(2)WhMapping<コンポーネント名>クラスの変更」を参照してください。

(3) アプリケーションの配備

Webアプリケーションのパッケージおよび配備を行います。配備の方法については、「3.5.3Webアプリケーションの配備」を参照してください。

5.3.3.画面の入力値に対して全桁数が入力されたかをチェックする例

Webアプリケーション画面の入力値に対して、そのチェック処理をWebアプリケーション上で行いますが、チェック処理の追加／変更をすることが出来ます。

ここでは、「注文店」項目に全桁数が入力されたか(EQU)をチェックする変更例を示します。

(1) WhMessage<画面名>ファイルのカスタマイズ

入力値のチェック機能に関するカスタマイズは、WhMessage<画面名>ファイルで行います。

WhMessage<画面名>ファイルをカスタマイズする場合、対象ファイルを直接編集せず、カスタマイズ元のクラスを継承した別名ファイルを新規に作成してください。別名ファイルにはカスタマイズするメソッドのみをオーバーライドで記述します。

【MFDL移行プロジェクトが生成したAP(Javaソース)ファイル】

例:WhMessageSHOP.java

```
        /**
         * クラス内に保持する全ての WhField インスタンスを初期化する
         */
        public WhMessageSHOP() {

                fieldname = "SHOP";                    ・・・①
                initvalue = "";                        ・・・②
                fieldsize = 20;                        ・・・③
                lfldsize = 20;
                SHOP = new WhField(fieldname, initvalue, fieldsize, lfldsize);
                SHOP.setBorder(WhField.BORDER_ALL);    ・・・④
                SHOP.setColor(WhField.COLOR_BLUE);     ・・・⑤
                SHOP.setEncodeStyle(WhField.ENCODE_INPUT_TEXT);
                SHOP.setFil(" ");
                SHOP.setKanjiCheck(true);              ・・・⑥

        }
```

WhMessage<画面名>のコンストラクタでは、入出力フィールドの入出力値を保持するWhFieldクラスを初期化します。上記のSHOPフィールドの例では、項目名=SHOP(①)、初期値=""(②)、項目長=20(③)、罫線をBORDER_ALL(上下左右罫線)(④) 、文字色をCOLOR_BLUE(青色)(⑤)に設定しています。最終行では入力値に対する漢字属性チェックを行う設定をしています(⑥)。

文字色や罫線で使用している定数については、「4.5.1(1)WhFieldクラス」を参照してください。

【カスタマイズしたAP(Javaソース)ファイル】

例:WhMessageSHOPuser.java

```
public class WhMessageSHOPuser extends WhMessageSHOP {

        /**
         * コンストラクタ
         */
        public void WhMessageSHOPuser() {

                super();                               ・・・①
                SHOP. setEquCheck(true);               ・・・②
        }
}
```

上記の例では、親クラスのコンストラクタを実行することでメッセージクラスのSHOPフィールドに初期値の設定を行っています(①)。入力値が全桁数入力されているかのEQUチェックを行う設定を追加しています(②)。

(2) マッピングファイルの変更

AP(Javaソース)をカスタマイズした場合は、マッピングファイル(WhMappingCommonクラス)を変更します。変更方法については、「5.3.1(2)WhMapping<コンポーネント名>クラスの変更」を参照してください。

(3) アプリケーションの配備

EJBアプリケーションとWebアプリケーションのパッケージおよび配備を行います。配備の方法については、「3.4.3EJBアプリケーションの配備」、および「3.5.3Webアプリケーションの配備」を参照してください。

5.4.設定ファイルのカスタマイズ例

本節では、設定ファイル(プロパティファイル、スタイルシート)のカスタマイズ例を記述しています。

5.4.1.メッセージプロパティファイルのカスタマイズ方法

メッセージプロパティファイルのカスタマイズは、プロパティファイルの内容をエディタで直接カスタマイズします。

(1) エラーメッセージの変更

MFDL 移行プロジェクトで作成した Whmessage_en(ja).properties を「Developer's Studio」の「プロジェクトエクスプローラ」で開き、直接　エラーメッセージを変更または追加します。

なお、日本語のエラーメッセージ(Whmessage_ja.properties)を変更するためには、PropertiesEditor プラグイン等を利用してください。

5.4.2.エラーメッセージを変更する例

Web アプリケーション画面の入力値に対して、MFDL ソースに指定されているチェックを Web アプリケーション上で行います。入力値が不適切と判断された場合にエラーメッセージを表示しますが、そのメッセージを変更したり、追加することが出来ます。

ここでは、Web アプリケーション画面で「SHOP is Not Input」と表示していたメッセージを「Input is Nothing : SHOP」へ変更する例を示します。

また、エラーメッセージ「SHOP Number is mistaken」を追加しています。

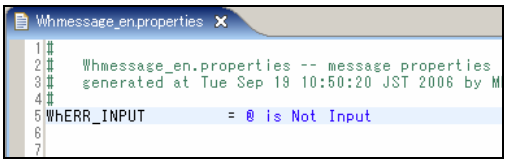
(1) エラーメッセージの変更及び追加

メッセージ ID「WhERR_INPUT」のメッセージ内容を変更し、メッセージ ID「ERR_NUMERR」のメッセージを追加しています。

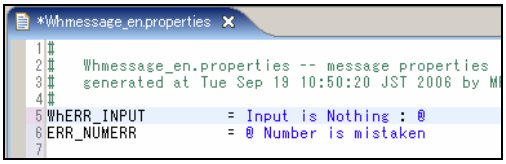
【カスタマイズファイル】

WhMessage_en.properties

【カスタマイズ前】



【カスタマイズ後】



(2) アプリケーションの配備

EJBアプリケーションとWebアプリケーションのパッケージおよび配備を行います。配備の方法については、「3.4.3EJBアプリケーションの配備」、および「3.5.3Webアプリケーションの配備」を参照してください。

5.5.高度なカスタマイズ例

本節では、生成した Web アプリケーションの高度なカスタマイズ例を記述しています。

5.5.1.ACOS ホストからの複数回の出力を結合する例

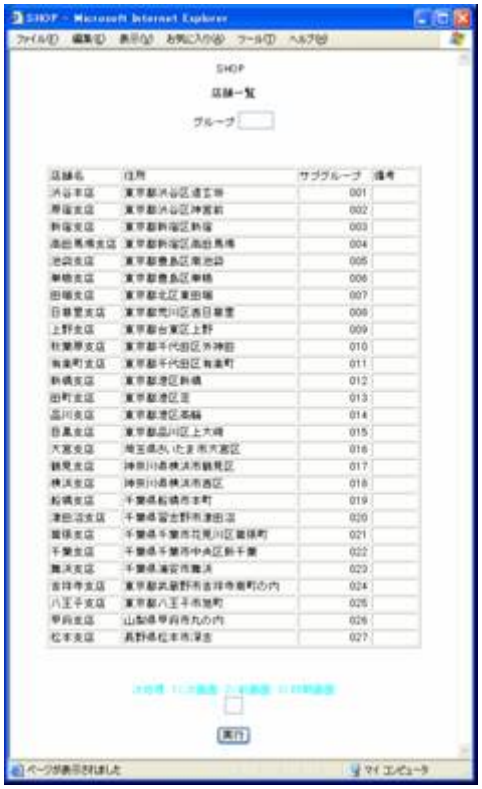
一覧画面を表示する場合、ETOS 画面では画面の大きさの制限により複数回の画面表示が必要でしたが、Web 画面では 1 画面に全ての一覧を表示することができます。

ここでは、店舗一覧画面の複数回の表示を 1 画面に結合する例を示します。

画面イメージ

【カスタマイズ前】

【カスタマイズ後】



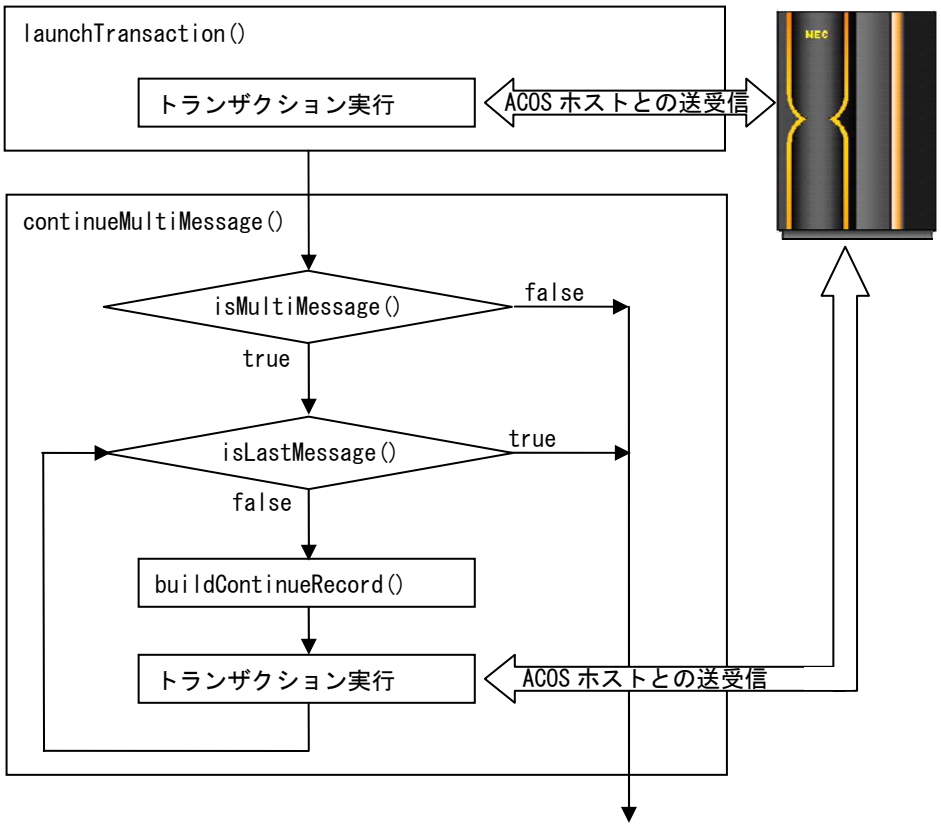
ACOS ホストとの複数回送受信の仕組み

ACOS ホストとの複数回送受信の処理について説明します。

ACOS ホストとの送受信に関係するメソッドには以下があります。

クラス	メソッド	処理概要
WhProcessor	launchTransaction()	1 回目のトランザクションを実行する。
	continueMultiMessage()	継続メッセージがある場合、2 回目以降のトランザクションを実行する。
WhProcessor<画面名>	isMultiMessage()	継続メッセージの有無を判断する。
	isLastMessage()	継続メッセージの終了を判断する。
	buildContinueRecord()	直前の出力電文から、次の入力電文を作成する。

上記メソッドの関連を表した処理フローとその処理内容を説明します。



launchTransaction メソッドでは、1回目のトランザクションを実行します。

continueMultiMessage メソッドでは、isMultiMessage メソッドで継続メッセージの有無を判断します。継続メッセージが無い(false)場合は、continueMultiMessage メソッドを終了します。継続メッセージがある(true)場合は、継続メッセージ終了判断およびトランザクション実行を繰り返します。

isMultiMessage メソッドの戻り値は、既定値では継続メッセージ無し(false) なので、継続メッセージあり(true) を返却するようにカスタマイズします。

isLastMessage メソッドは受信データが最後であるかを判断します。最後のメッセージである(true)場合は continueMultiMessage メソッドを終了します。継続する(false)場合は、再度トランザクションを実行します。isLastMessage メソッドの継続メッセージの終了条件については、業務によって条件が変わるため、カスタマイズして判断条件を追加します。

buildContinueRecord メソッドは、継続メッセージがある場合、次の入力電文を作成する処理です。直前の出力電文を元に入力電文を作成します。キー操作により入力していたフィールドがある場合、カスタマイズして値を設定します。

この例では以下のファイルをカスタマイズします。

ファイル	処理	処理概要
SHOPuser.jsp	for 文の追加	全ての出力メッセージを取得する。
	for 文の終了条件を追加	有効データのみ表示する条件を設定する。
WhProcessorSHOPuser.java	isMultiMessage()	継続メッセージ有無を設定する。
	isLastMessage ()	継続メッセージの終了を判断する。
	buildContinueRecord()	2 回目以降の入力電文を作成する。

(1) JSP ファイルのカスタマイズ

JSP ファイルをカスタマイズする場合、対象ファイルを直接編集せずに別名ファイルへコピーして、別名ファイルをカスタマイズしてください。

店舗一覧画面(SHOP.jsp)を別名ファイル(SHOPuser.jsp)としてコピーし、出力データの表示行数を動的に変更できるようカスタマイズします。

【MFDL移行プロジェクトが生成したJSPファイル】

例:SHOP.jsp

```
      :
<% for (int i = 0; i < 15; i++) { %>                    ・・・①
      :
    <td>
      <%= WhUtilWebAp.getHtmlTag(message.getSHOP(i),did) %>
    </td>
      :
<% } %>
      :
```

上記の例では、for ループにより 1 画面 15 行分のデータを表示しています(①)。

【カスタマイズした JSP ファイル】

例:SHOPuser.jsp

```
      :
<!-- 全ての出力メッセージ(arraylist 要素)を表示する -->
<% for (int j = 0; j < message.getArraySize(); j++) { %>                    ・・・①
  <!-- 1回分の出力メッセージを取得 -->
  <% WhMessageSHOP outMsg = (WhMessageSHOP) message.getArrayOutMsg(j); %>    ・・・②

  <% for (int i = 0; i < 15; i++) { %>
    <!-- 「注文店」が空白になったら表示を終了する -->
    <% if (outMsg.getSHOP(i).getValue().trim().equals("") == true) break; %>  ・・③
      :
    <td>
      <%= WhUtilWebAp.getHtmlTag(outMsg.getSHOP(i),did) %>                   ・・・④
    </td>
      :
  <% } %>
<% } %>
      :
```

上記の例では、for ループを追加して複数画面分のデータを表示します(①)。 複数画面のデータは配列(arraylist)で保持しているため、この配列から 1 画面分ずつデータを取得します(②)。getArraySize メソッドは画面数を返し、getArrayOutMsg(j)メソッドは j 番目の画面の出力メッセージを返します。

この例では、SHOP フィールドが空白となった場合、表示を終了します(③)。

配列(arraylist)から取得したデータを表示するために、②の outMsg を使用するように変更しています(④)。

(2) WhProcessor<画面名>ファイルのカスタマイズ

ACOS ホストからの出力値に関するカスタマイズは、WhProcessor<画面名>クラスで行います。

WhProcessor<画面名>クラスをカスタマイズする場合、対象ファイルを直接編集せず、カスタマイズ元のクラスを継承した別名ファイルを新規に作成してください。別名ファイルには、カスタマイズするメソッドのみをオーバーライドで記述します。

ACOS ホストから複数回の出力電文がある場合(継続メッセージがある場合)、継続メッセージを受信するためのカスタマイズを行います。

【MFDL移行プロジェクトが生成したAP(Javaソース)ファイル】

例:WhProcessorSHOP.java

```
        /**
         * 継続メッセージの有無を設定
         * @return                  boolean :true=継続あり、false=継続なし
         */
        protected boolean isMultiMessage() {
                return false;                                   ・・・①
        }

        /**
         * 継続メッセージの最後を判定する
         * @param          curMsg :出力メッセージ
         * @return                  boolean :true=最後である、false=継続する
         */
        protected boolean isLastMessage(WhMessage curMsg) {
                //以下はカスタマイズで使用
                //WhMessageSHOP outMsg = (WhMessageSHOP)curMsg;
                return true;                                    ・・・②
        }
        /**
         * 継続用送信レコードの生成
         * @param          curMsg :メッセージ
         * @return                  Record :レコード
         * @throws                  ResourceException
         */
        protected Record buildContinueRecord(WhMessage curMsg)
                                                        throws ResourceException {
                return buildRecord(curMsg);                     ・・・③
        }
```

isMultiMessage メソッドは、継続メッセージの有無を設定します。 既定値は false(継続メッセージ無し)です(①)。

isLastMessage メソッドは、継続メッセージの終わりを設定します。既定値は true(継続メッセージの終了)です(②)。

buildContinueRecord メソッドは、ACOS ホストへの入力電文を作成します。buildRecord メソッドを使用し、出力メッセージを元に ACOS ホストへの入力電文を作成しています(③)。

【カスタマイズした AP(Java ソース)ファイル】

例:WhProcessorSHOPuser.java

```
package wh.ejb;
import wh.common.*;
import javax.resource.cci.*;
import javax.resource.ResourceException;

public class WhProcessorSHOPuser extends WhProcessorSHOP {

        /**
         * 継続メッセージの有無を設定
         * @return                  boolean :true=継続あり、false=継続なし
         */
        protected boolean isMultiMessage() {
                return true;                                            ・・・①
        }
```

```
        /**
         * 継続メッセージの最後を判定する
         * @param          curMsg :出力メッセージ
         * @return                 boolean :true=最後である、false=継続する
         */
        protected boolean isLastMessage(WhMessage curMsg) {
                //以下はカスタマイズで使用
                WhMessageSHOP outMsg = (WhMessageSHOP)curMsg;
                //出力データ行数分確認
                for(int i = 0; i < outMsg.getSHOP().length; i++) {
                        //SHOP 値が空白なら終了
                        String value = outMsg.getSHOP(i).getValue();
                        if (value.trim().equals("") == true) {          ・・・②
                                return true;
                        }
                }
                return false;                                           ・・・③
        }

        /**
         * 継続用送信レコードの生成
         * @param          curMsg :メッセージ
         * @return                 Record :レコード
         * @throws                 ResourceException
         */
        protected Record buildContinueRecord(WhMessage curMsg)
                                                throws ResourceException {
                ((WhMessageSHOP)curMsg).getNEXT().setValue("1");        ・・・④
                return super.buildContinueRecord(curMsg);               ・・・⑤
        }
}
```

上記の例では、WhProcessorSHOP クラスの isMultiMessage メソッドをオーバーライドしています。複数回の出力電文処理を行うため、true(継続メッセージあり)を設定します(①)。

isLastMessage メソッドは、最終メッセージか否かの判定を行います。WhMessageSHOP クラスに型変換しているコメントを外し判定処理を追加します。この例では、出力メッセージの SHOP フィールドが空白となっている行が現れると、最終メッセージと判断し true(継続メッセージの終了)を返却しています(②)。また、全ての SHOP フィールドが空白でない場合は、最終メッセージでないと判断し false(継続する)を返却しています(③)。

buildContinueRecord メソッドは、直前の出力メッセージから次のトランザクション用の入力電文を作成します。この例では、次画面を要求するための指示が必要なため、NEXT フィールドを"1"(次画面表示)に変更しています(④)。 その後、親クラスの buildContinueRecord メソッドを呼び、入力電文を作成します(⑤)。

(3) マッピングファイルの変更

JSPとAP(Javaソース)をカスタマイズした場合は、マッピングファイル(WhMappingWebApクラス)を変更します。変更方法については、「5.2.1(2)WhMappingWebApクラスの変更」、および「5.3.1(2)WhMapping<コンポーネント名>クラスの変更」を参照してください。

(4) アプリケーションの配備

EJBアプリケーションとWebアプリケーションのパッケージおよび配備を行います。配備の方法については、「3.4.3EJBアプリケーションの配備」、および「3.5.3Webアプリケーションの配備」を参照してください。

5.5.2.会話型トランザクションを使用する例

通常、Web アプリケーションと ACOS ホストとの通信はステートレスセッションで行われるため、複数のトランザクション要求が同一ブラウザ画面からのものとは保障されません。MFDL 移行プロジェクトが、生成した Web アプリケーションは問合せ型トランザクションです。

会話型トランザクションを使用した場合、ACOS ホストは同一ブラウザ画面からの通信であることを端末識別子により確認できます。

ここでは、会話型トランザクションにカスタマイズする例を示します。

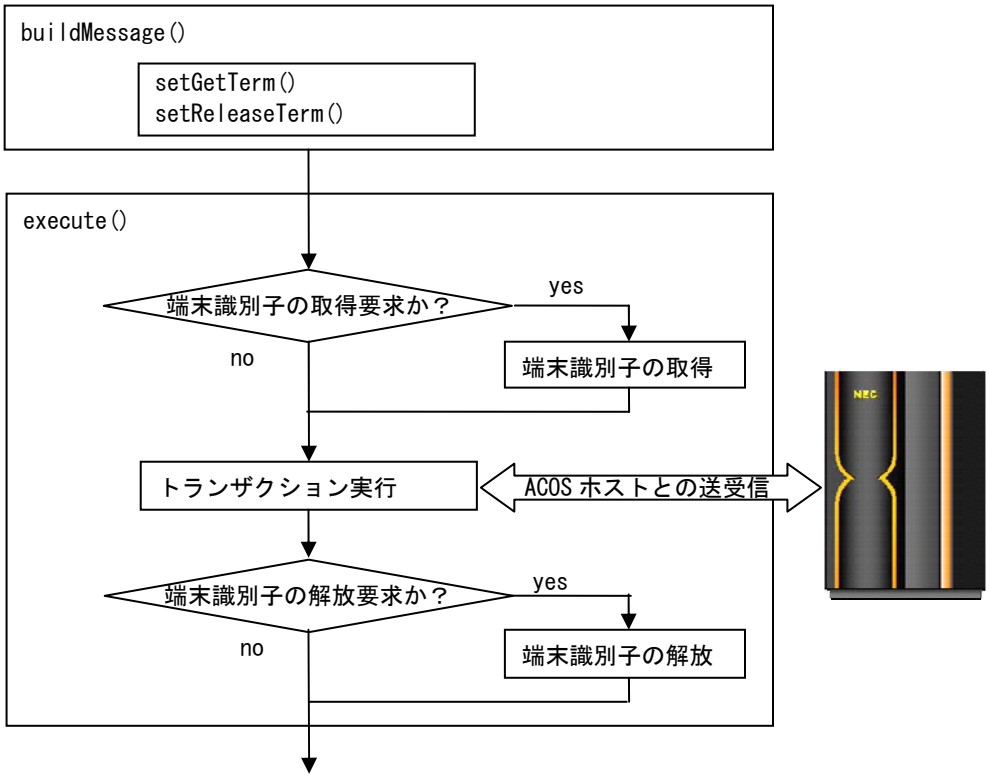
ACOS ホストとの会話型トランザクションの仕組み

ACOS ホストとの会話型トランザクションの処理について説明します。

ACOS ホストとの会話型トランザクションに関係するメソッドは、以下があります。

クラス	メソッド	処理概要
WhCtrl<画面名>	buildMessage()	端末識別子の取得・開放要求を指示する。
WhMessage	setGetTerm()	端末識別子の取得要求を設定する。
	setReleaseTerm()	端末識別子の開放要求を設定する。
WhEJBBean	execute()	端末識別子を設定したトランザクションを実行する。

上記メソッドの関連を表した処理フローとその処理内容を説明します。



buildMessage メソッドにて、setGetTerm メソッドを呼び出して端末識別子を取得するようにカスタマイズすることで、会話型トランザクション（端末識別子を設定したトランザクション）を開始します。また、setReleaseTerm メソッドを呼び出し、端末識別子を解放するような終了条件にカスタマイズすることで会話型トランザクションを終了します。

execute メソッドにて、端末識別子の取得要求を判断します。取得要求があれば、端末識別子を取得してトランザクションを実行します。取得済みの場合はその端末識別子を設定してトランザクションを実行します。また、会話型トランザクションの終了条件となった場合、端末識別子を解放して終了します。

(1) WhCtrl<画面名>ファイルのカスタマイズ

会話型トランザクションを使用するために必要なメソッドは用意されています。そのメソッドを WhCtrl<画面名>クラスから呼び出し、端末識別子の取得・解放有無を設定することにより会話型トランザクションを使

用できます。

WhCtrl<画面名>ファイルをカスタマイズする場合、対象ファイルを直接編集せず、カスタマイズ元のクラスを継承した別名ファイルを新規に作成してください。別名ファイルにはカスタマイズするメソッドのみをオーバーライドで記述します。

【MFDL移行プロジェクトが生成したAP(Javaソース)ファイル】

例:WhCtrlSHOP.java

```
        /**
         * 入力メッセージ作成処理
         */
        protected void buildMessage(HttpServletRequest httpReq) ～ {
                getInMessage();                                    ・・・①
                generateTRANS(httpReq);                            ・・・②
                generateSHOP(httpReq);                            }
                generateCLERK(httpReq);                            ・・・③
                generateNEXT(httpReq);                            }
                //not Null の時はエラー発生
                if (ctrlException != null) {                       ・・・④
                        throw ctrlException;
                }
        }
```

buildMessage メソッドは、generate<フィールド名>メソッドにより、入力フィールドの値を入力メッセージに格納しています。

上記の例では、まず入力メッセージのインスタンスを取得しています(①)。 次に generateTRANS メソッドを呼び出し、TRANS フィールドの入力値チェックと入力メッセージへの格納を行っています(②)。 その他のフィールドも同様です(③)。 各フィールドの入力値チェックでエラーが発生した場合は、その内容を設定した例外オブジェクトを作成し、例外を呼び出し元へスローしています(④)。

【カスタマイズした AP(Java ソース)ファイル】

例:WhCtrlSHOPuser.java

```
package wh.webap;
import wh.common.*;

public class WhCtrlSHOPuser extends WhCtrlSHOP {

        protected void buildMessage(HttpServletRequest httpReq) ～ {      ・・・①
                super.buildMessage(httpReq);                              ・・・②

                //端末識別子取得の設定
                WhMessageSHOP msg = getInMessage();                       ・・・③
                msg.setGetTerm(true);                                     ・・・④

                //端末識別子解放の設定
                if (msg.getNEXT().getValue().equals("END") == true) {
                        msg.setReleaseTerm(true);                         ・・・⑤
                }
        }
}
```

上記の例では、WhCtrlSHOP クラスの buildMessage メソッドをオーバーライドしています(①)。 まず、親クラスの buildMessage メソッドを実行し、入力メッセージを作成しています(②)。 その後、入力メッセージのインスタンスを取得して(③)、会話型トランザクションを実行するために、端末識別子の取得を行うように指示します(④)。 また、この例では、NEXT フィールドの入力値が“END”となった場合、端末識別子の解放を行い、会話型トランザクションを終了するように指示しています(⑤)。

(2) マッピングファイルの変更

AP(Javaソース)をカスタマイズした場合は、マッピングファイル(WhMappingWebApクラス)を変更します。変更方法については、「5.3.1(2)WhMapping<コンポーネント名>クラスの変更」を参照してください。

(3) アプリケーションの配備

Webアプリケーションのパッケージおよび配備を行います。配備の方法については、「3.5.3Webアプリケーションの配備」を参照してください。

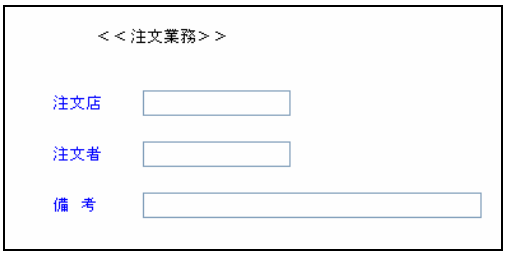
5.5.3.入出力項目を追加・削除する例

入出力項目の追加や削除を行う例を示します。

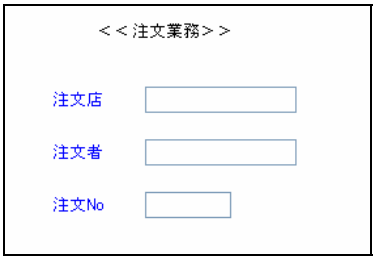
ここでは、注文業務画面に注文 No(ORDER(入力項目))を追加し、備考(MEMO(出力項目))を削除する例を示します。

画面イメージ

【カスタマイズ前】



【カスタマイズ後】



(1) JSP ファイルのカスタマイズ

JSP ファイルをカスタマイズする場合、対象ファイルを直接編集せずに別名ファイルへコピーして、別名ファイルをカスタマイズしてください。

【MFDL移行プロジェクトが生成したJSPファイル】

例:SHOP.jsp

```
        :
<td class='color_blue'>
  注文店
</td>
<td>
  <%= WhUtilWebAp.getHtmlTag(message.getSHOP(),did) %>
</td>
        :
<td class='color_blue'>
  注文者
</td>
<td>
  <%= WhUtilWebAp.getHtmlTag(message.getCLERK(),did) %>
</td>
        :
<td class='color_blue'>
  備　考
</td>
<td>
  <%= WhUtilWebAp.getHtmlTag(message.getMEMO(),did) %>
</td>
        :
```

WhUtilWebAp.getHtmlTag メソッドは、フィールドの属性に応じた HTML タグを自動生成するメソッドです。上記の MEMO フィールドの例では、以下のような HTML タグが自動生成されます。

<input type=hidden name="MEMO" value=" ">

【カスタマイズした JSP ファイル】

例:SHOPuser.jsp

```
        :
<td class='color_blue'>
  注文店
</td>
```

```
  <td>
    <%= WhUtilWebAp.getHtmlTag(message.getSHOP(),did) %>
  </td>
        :
  <td class='color_blue'>
    注文者
  </td>
  <td>
    <%= WhUtilWebAp.getHtmlTag(message.getCLERK(),did) %>
  </td>
        :
  <td class='color_blue'>
    注文 No                                                    ・・・①
  </td>
  <td>
    <%= WhUtilWebAp.getHtmlTag(message.getORDER(),did) %>
  </td>
        :
~~  <td class='color_blue'>~~
~~    備　考~~
~~  </td>~~
~~  <td>~~
~~    <%= WhUtilWebAp.getHtmlTag(message.getMEMO(),did) %>~~          ・・・②
~~  </td>~~
        :
```

上記の例では、入力項目 ORDER を追加しています(①)。 また出力項目 MEMO を削除しています(②)。

(2) WhMessage<画面名>ファイルのカスタマイズ

メッセージクラスでは、各フィールド名に対応する WhField オブジェクトに関する処理が用意されています。入出力項目の追加や削除に合わせて、これらの処理も追加・削除します。既存のメソッドを参考にしてコンストラクタや複製インスタンス取得処理への追加、項目情報取得処理の追加を行います。

WhMessage<画面名>ファイルをカスタマイズする場合、対象ファイルを直接編集せず、カスタマイズ元のクラスを継承した別名ファイルを作成してください。別名ファイルにはカスタマイズするメソッドのみをオーバーライドで記述します。

【MFDL移行プロジェクトが生成したAP(Javaソース)ファイル】

例:WhMessageSHOP.java

```
        private WhField    SHOP;
        private WhField    CLERK;
        private WhField    MEMO;                                ・・・①

        public WhMessageSHOP() {
                  :
                fieldname = "SHOP";
                  :
                fieldname = "CLERK";
                  :
                fieldname = "MEMO";                             ・・・②
                initvalue = " ";
                fieldsize = 60;
                lfldsize = 60;
                initColor = WhField.COLOR_RED;
                MEMO = new WhField(fieldname, initvalue, fieldsize, lfldsize);
                MEMO.setColor(initColor);
                MEMO.setEncodeStyle(WhField.ENCODE_TEXT);
                MEMO.setFil(" ");
        }
                  :
        public WhField getSHOP() {
                  :
        public WhField getCLERK() {
                  :
        public WhField getMEMO() {                              ・・・③
                return MEMO;
        }

        public WhMessage getCopyInstance() {
```

```
                    WhMessageSHOP msg = new WhMessageSHOP();
                    msg.SHOP = this.SHOP.getCopyField();
                    msg.CLERK = this.CLERK.getCopyField();
                    msg.MEMO = this.MEMO.getCopyField();            ・・・④
                    msg.setNextMessageName(this.nextMessageName);
                    return msg;
          }
```

WhMessage<画面名>クラスのコンストラクタでは、各フィールドの情報を保持する WhField クラスを初期化します。上記の例では、MEMO フィールドを用意し(①)、コンストラクタで初期化しています(②)。 また、この MEMO フィールドを取得する getMEMO メソッドが用意されており(③)、複製インスタンスを作成する処理でも、MEMO フィールドが使用されています(④)。

【カスタマイズした AP(Java ソース)ファイル】

例:WhMessageSHOPuser.java

```
package wh.common;

public class WhMessageSHOPuser extends WhMessageSHOP {

          private WhField    SHOP;
          private WhField    CLERK;
          private WhField    MEMO;                                ・・・①
          private WhField    ORDER;                               ・・・②

          public WhMessageSHOP() {
                    :
                    fieldname = "SHOP";
                    :
                    fieldname = "CLERK";
                    :
                    fieldname = "MEMO";
                    initvalue = " ";
                    fieldsize = 60;
                    lfldsize = 60;
                    initColor = WhField.COLOR_RED;                                    ・・・③
                    MEMO = new WhField(fieldname, initvalue, fieldsize, lfldsize);
                    MEMO.setColor(initColor);
                    MEMO.setEncodeStyle(WhField.ENCODE_TEXT);
                    MEMO.setFil(" ");

                    fieldname = "ORDER";
                    initvalue = "";
                    fieldsize = 10;
                    lfldsize = 10;
                    initColor = WhField.COLOR_BLUE;                                   ・・・④
                    ORDER = new WhField(fieldname, initvalue, fieldsize, lfldsize);
                    ORDER.setColor(initColor);
                    ORDER.setEncodeStyle(WhField.ENCODE_INPUT_TEXT);
                    ORDER.setFil(" ");
          }

                    :
          public WhField getSHOP() {
                    :
          public WhField getCLERK() {
                    :

          public WhField getMEMO() {
                    return MEMO;                                ・・・⑤
          }

          public WhField getORDER() {                                         ・・・⑥
                    return ORDER;
          }

          public WhMessage getCopyInstance() {
                    WhMessageSHOP msg = new WhMessageSHOP();
                    msg.SHOP = this.SHOP.getCopyField();
                    msg.CLERK = this.CLERK.getCopyField();
                    msg.MEMO = this.MEMO.getCopyField();            ・・・⑦
```

```
                msg.ORDER = this.ORDER.getCopyField();          ・・・⑧
                msg.setNextMessageName(this.nextMessageName);
                return msg;
        }
}
```

上記の例では、削除した出力項目の MEMO フィールドに関する処理を削除しています(①③⑤⑦)。また、追加した入力項目の ORDER フィールドに関する処理を追加しています(②④⑥⑧)。コンストラクタでは、ORDERフィールドの項目名をORDER、初期値を""、項目長を10、コピー句のデータ長を10、文字色をCOLOR_BLUE(青)、JSP 表示時の INPUT タグ TYPE 属性を ENCODE_INPUT_TEXT(type=text)、フィル文字を" "に初期化しています(④)。

文字色や罫線で使用している定数については「4.5.1(1)WhFieldクラス」を参照してください。

(3) WhCtrl<画面名>ファイルのカスタマイズ

入力値のチェックを WhCtrl<画面名>クラスで行っているため、追加した入力項目は、既存のメソッドを参考にして入力メッセージ作成処理や入力チェック処理、入力データ設定処理を追加します。

削除する出力項目については不要な処理を削除します。

WhCtrl<画面名>ファイルをカスタマイズする場合、対象ファイルを直接編集せず、カスタマイズ元のクラスを継承した別名ファイルを作成してください。別名ファイルにはカスタマイズするメソッドをオーバーライドで記述するか、新規メソッドを記述します。

【MFDL移行プロジェクトが生成したAP(Javaソース)ファイル】

例:WhCtrlSHOP.java

```
        protected void buildMessage(HttpServletRequest httpReq) ～ {
                getInMessage();
                generateSHOP(httpReq);
                generateCLERK(httpReq);
                //not Null の時はエラー発生
                if (ctrlException != null) {
                        throw ctrlException;
                }
        }
                        :
        protected void validateSHOP(String chkValue,WhField field) throws WhException {
                        :
        protected void validateCLERK(String chkValue,WhField field) throws WhException {
                        :
        protected void generateSHOP(HttpServletRequest httpReq) {
                        :
        protected void generateCLERK(HttpServletRequest httpReq) {
                        :
        protected void clearBlink() {
                inMsg.getSHOP().setBlink(WhField.EFFECT_NO);
                inMsg.getCLERK().setBlink(WhField.EFFECT_NO);
                inMsg.getMEMO().setBlink(WhField.EFFECT_NO);            ・・・①
        }

        protected void setInputData(HttpServletRequest httpReq) {
                String value;
                value = httpReq.getParameter("SHOP");
                if (inMsg.getSHOP().getError() == false)
                        inMsg.getSHOP().setValue(value);
                value = httpReq.getParameter("CLERK");
                if (inMsg.getCLERK().getError() == false)
                        inMsg.getCLERK().setValue(value);
                value = httpReq.getParameter("MEMO");                   ・・・②
                if (inMsg.getMEMO().getError() == false)
                        inMsg.getMEMO().setValue(value);
        }
```

上記の MEMO フィールドの例では、ブリンク設定をクリアする処理(①)を行っています。また、入力値のチェックでエラーとなった場合の画面再表示のために、入出力値を再設定してしています(②)。

【カスタマイズした AP(Java ソース)ファイル】

例：WhCtrlSHOPuser.java

```
package wh.webap;
import wh.common.*;
import javax.servlet.http.*;

public class WhCtrlSHOPuser extends WhCtrlSHOP {

        protected void buildMessage(HttpServletRequest httpReq) ～ {
                getInMessage();
                generateSHOP(httpReq);
                generateCLERK(httpReq);
                generateORDER(httpReq);                                    ・・・①
                //not Null の時はエラー発生
                if (ctrlException != null) {
                        throw ctrlException;
                }
        }
         :
        protected void validateORDER(String chkValue,WhField field)
                                        throws WhException {
                String check = chkValue;
                if (WhUtilCommon.isNull(check) == true) {
                        field.setErrorData();
                        throw new WhException(
                                WhUtilCommon.OPT_ERR_NULL, "ORDER");       ・・・②
                }
                int errcode = WhUtilCommon.chkInput(check, field);
                if (errcode != WhUtilCommon.OPT_NORMAL &&
                    errcode != WhUtilCommon.OPT_ERR_NULL) {
                        field.setErrorData();
                        throw new WhException(errcode, "ORDER");
                }
        }

        protected void generateORDER(HttpServletRequest httpReq) {
                String value;
                value = httpReq.getParameter("ORDER");
                try {
                        validateORDER(value, inMsg.getORDER());            ・・・③
                        inMsg.getORDER().setValue(value);
                } catch(WhException we) {
                        inMsg.setFocus("ORDER");
                        if (ctrlException == null) ctrlException = we;
                }
        }

        protected void clearBlink() {
                inMsg.getSHOP().setBlink(WhField.EFFECT_NO);
                inMsg.getCLERK().setBlink(WhField.EFFECT_NO);
                inMsg.getMEMO().setBlink(WhField.EFFECT_NO);               ・・・④
                inMsg.getORDER().setBlink(WhField.EFFECT_NO);              ・・・⑤
        }

        protected void setInputData(HttpServletRequest httpReq) {
                String value;
                value = httpReq.getParameter("SHOP");
                if (inMsg.getSHOP().getError() == false)
                        inMsg.getSHOP().setValue(value);
                value = httpReq.getParameter("CLERK");
                if (inMsg.getCLERK().getError() == false)
                        inMsg.getCLERK().setValue(value);
                value = httpReq.getParameter("MEMO");
                if (inMsg.getMEMO().getError() == false)                   ・・・⑥
                        inMsg.getMEMO().setValue(value);
                value = httpReq.getParameter("ORDER");
                if (inMsg.getORDER().getError() == false)                  ・・・⑦
                        inMsg.getORDER().setValue(value);
        }
}
```

上記の例では WhCtrlSHOP クラスの buildMessage メソッドをオーバーライドして、ORDER フィールドの処理を追加します(①)。さらに、他の入力項目のメソッドを参考に、validateORDER メソッド(②)、generateORDER メソッド(③)、ブリンククリア処理(⑤)、入出力データ復元処理(⑦)を追加します。また、MEMO フィールドに関する処理を削除しています(④⑥)。

(4) WhProcessor<画面名>ファイルのカスタマイズ

ACOS ホストへの入力値の設定や ACOS ホストからの出力値の取得を WhProcessor<画面名>クラスで行っているため、追加した入出力項目は同様の設定をします。既存のメソッドを参考にしてレコード作成処理や入力データ書込処理、入力チェック処理、データ変換処理、フィールド初期化処理、ブリンククリア処理を追加します。

WhProcessor<画面名>ファイルをカスタマイズする場合、対象ファイルを直接編集せず、カスタマイズ元のクラスを継承した別名ファイルを作成してください。別名ファイルにはカスタマイズするメソッドをオーバーライドで記述するか、新規メソッドを記述します。

【MFDL移行プロジェクトが生成したAP(Javaソース)ファイル】

例:WhProcessorSHOP.java

```
protected Record buildRecord(WhMessage inMsg) throws ResourceException {
        setInMessage(inMsg);
        MappedRecord inRec = rf.createMappedRecord("SHOP-IN");
        generateSHOP(inRec);
        generateCLERK(inRec);
        return inRec;
}
                :
protected void generateSHOP(MappedRecord inRec) {
                :
protected int validateSHOP(String value, WhField field) {
                :
protected String convertSHOP(String value, WhField field, int opterr) {
                :
protected void generateCLERK(MappedRecord inRec) {
                :
protected int validateCLERK(String value, WhField field) {
                :
protected String convertCLERK(String value, WhField field, int opterr) {
                :
public WhMessage buildMessage(Record outRec, String inMsgName) ～ {
                :
        generateOutMEMO(rec);                                    ・・・①
                :
        return outMsg;
}

public void generateOutMEMO(MappedRecord outRec) {               ・・・②
        WhField field = outMsg.getMEMO();
        String wk     = (String)outRec.get("SHOP-OUT-MEMO-M");
        int modify    = Integer.parseInt(wk);
        String value  = (String)outRec.get("SHOP-OUT-MEMO");
        //Modify 情報は通常送信、部分送信関係なく設定
        if (field.analyzeAttribute(mcastat, modify) == true) {
                outMsg.setFocus("MEMO");
        }

        //値は送信内容は判別して設定
        if (field.isSetValue(modify, noData, mcastat, inMsgName, outMsgName)) {
                field.setValue(WhUtilCommon.delNullChar(value));
        }
}

protected void clearField() {
        if (outMsg.getSHOP().getEncodeStyle()
                == WhField.ENCODE_INPUT_TEXT) {
                outMsg.getSHOP().resetValue();
        }
        if (outMsg.getCLERK().getEncodeStyle()
                == WhField.ENCODE_INPUT_TEXT) {
                outMsg.getCLERK().resetValue();
```

```
                }
                if (outMsg.getMEMO().getEncodeStyle()                    ・・・③
                        == WhField.ENCODE_INPUT_TEXT) {
                        outMsg.getMEMO().resetValue();
                }
        }

        protected void clearBlink() {
                outMsg.getSHOP().setBlink(WhField.EFFECT_NO);
                outMsg.getCLERK().setBlink(WhField.EFFECT_NO);
                outMsg.getMEMO().setBlink(WhField.EFFECT_NO);            ・・・④
        }
```

上記の MEMO フィールドの例では、ACOS ホストからの出力値を取得する処理を行っています(①②)。また、フィールド初期化を行う clearField メソッドや、ブリンククリアを行う clearBlink メソッドで、MEMO フィールドを参照・設定しています(③④)。

【カスタマイズした AP(Java ソース)ファイル】

例:WhProcessorSHOPuser.java

```
package wh.webap;
import wh.common.*;
import javax.resource.cci.*;
import javax.resource.ResourceException;

public class WhProcessorSHOPuser extends WhProcessorSHOP {

        protected Record buildRecord(WhMessage inMsg)
                                        throws ResourceException {
                setInMessage(inMsg);
                MappedRecord inRec = rf.createMappedRecord("SHOP-IN");
                generateSHOP(inRec);
                generateCLERK(inRec);
                generateORDER(inRec);                                    ・・・①
                return inRec;
        }
                        :
        protected void generateORDER(MappedRecord inRec) {
                int ret = 0;
                String value;
                value = inMsg.getORDER().getValue();
                ret = validateORDER(value, inMsg.getORDER());            ・・・②
                value = convertORDER(value, inMsg.getORDER(), ret);
                inRec.put("SHOP-IN-ORDER", value);
                inRec.put("SHOP-IN-ORDER-E", String.valueOf(ret));
        }

        protected int validateORDER(String value, WhField field) {
                int ret = WhUtilCommon.chkInput(value, field);           ・・・③
                return ret;
        }

        protected String convertORDER(String value, WhField field, int opterr) {
                String conv = WhUtilCommon.delFillChar(value, field);    ・・・④
                return conv;
        }

        public WhMessage buildMessage(Record outRec, String inMsgName) ～ {
                        :
                generateOutMEMO(rec);                                    ・・・⑤
                        :
                return outMsg;
        }

        protected void clearField() {
                if (outMsg.getSHOP().getEncodeStyle()
                        == WhField.ENCODE_INPUT_TEXT) {
                        outMsg.getSHOP().resetValue();
                }
                if (outMsg.getCLERK().getEncodeStyle()
                        == WhField.ENCODE_INPUT_TEXT) {
```

```
                        outMsg.getCLERK().resetValue();
                }



                if (outMsg.getMEMO().getEncodeStyle()                    ・・・⑥
                        == WhField.ENCODE_INPUT_TEXT) {
                        outMsg.getMEMO().resetValue();
                }
                if (outMsg.getORDER().getEncodeStyle()                   ・・・⑦
                        == WhField.ENCODE_INPUT_TEXT) {
                        outMsg.getORDER().resetValue();
                }
        }

        protected void clearBlink() {
                outMsg.getSHOP().setBlink(WhField.EFFECT_NO);
                outMsg.getCLERK().setBlink(WhField.EFFECT_NO);
                outMsg.getMEMO().setBlink(WhField.EFFECT_NO);            ・・・⑧
                outMsg.getORDER().setBlink(WhField.EFFECT_NO);           ・・・⑨
        }

}
```

上記の例では、WhProcessorSHOP クラスの各メソッドをオーバーライドしています。buildRecord メソッドは、追加した入力フィールド ORDER の入力値を設定しています(①)。さらに、入力値を入力電文に設定する generateORDER メソッド(②)、入力値のチェックを行う validateORDER メソッド(③)、入力値のデータ変換を行う convertORDER メソッド(④)を追加しています。

また、ACOS ホストからの出力値を取得する buildMessage メソッドから出力項目 MEMO の処理を削除しています(⑤)。

出力表示を行うためのフィールド初期化処理や、ブリンククリア処理において MEMO フィールドの削除、ORDER フィールドの追加をそれぞれ行っています(⑥⑦⑧⑨)。

(5) cdd ファイルのカスタマイズ

入出力項目を追加・削除する場合、cdd ファイルを再作成する必要があります。

cdd ファイルは、MFDL 移行プロジェクトが作成したコピー句ファイルを別名ファイルへコピーして、別名ファイルに項目の追加/削除を行い、COBOL Data Composer（WebOTX OLF/TP Adapter　運用ガイド）を使って再作成します。

なお、cdd ファイルは入力メッセージ用 cdd ファイル(<画面名>-IN.cdd)と出力メッセージ用 cdd ファイル(<画面名>.cdd)があります。

以下にコピー句ファイルをカスタマイズする例を示します。

【MFDL移行プロジェクトが生成したコピー句ファイル】

例：cobolCopy_SHOP-IN.txt

```
          :
       07 SHOP-IN-SHOP-E              PIC 99.
       07 SHOP-IN-SHOP                PIC N(10) .          ・・・①
       07 SHOP-IN-CLERK-E             PIC 99.
       07 SHOP-IN-CLERK               PIC N(10) .
          :
```

例：cobolCopy_SHOP.txt

```
          :
       07 SHOP-OUT-MEMO-M              USAGE COMP-1.        ・・・②
       07 SHOP-OUT-MEMO                PIC X(60) .
          :
```

上記の例では、入力メッセージ用コピー句ファイルに入力項目 SHOP、CLERK が記述されています(①)。出力メッセージ用コピー句ファイルに出力項目 MEMO が記述されています(②)。

【カスタマイズしたコピー句ファイル】

例：cobolCopy_SHOP-INuser.txt

```
          :
       07 SHOP-IN-SHOP-E              PIC 99.
       07 SHOP-IN-SHOP                PIC N(10) .
       07 SHOP-IN-CLERK-E             PIC 99.
       07 SHOP-IN-CLERK               PIC N(10) .
       07 SHOP-IN-ORDER-E             PIC 99.          }・・・①
       07 SHOP-IN-ORDER               PIC N(10) .
          :
```

例：cobolCopy_SHOPuser.txt

```
          :
       07 SHOP-OUT-MEMO-M                USAGE COMP-1.     }・・・②
       07 SHOP-OUT-MEMO                  PIC X(60) .
          :
```

上記の例では、入力項目 ORDER を入力メッセージ用コピー句ファイルに追加しています(①)。また、出力項目 MEMO を出力メッセージ用コピー句ファイルから削除しています(②)。

cdd ファイルは、COBOL Data Composer（WebOTX OLF/TP Adapter 運用ガイド）を使って <MFDL project dir>/ConnectorContent/cdd 配下に再作成します。

※<MFDL project dir>は MFDL 移行プロジェクトで作成したディレクトリーで、既定値は <WebOTX install dir>/Studio/workspace/<project name> です。

cdd ファイルは、以下の方法で再作成します。

(1) コマンドプロンプトを起動する。

(2) COBOL Data Composer を実行する。

コマンド形式 （以下のコマンドは、一行で入力してください）

```
>java -classpath <COBOL Data Composer のクラスパス> -create <cdd ファイル>
  -dir <コピー句ファイル>
```

例） 以下は画面名が SHOP の場合の例です。

```
>java -classpath Composer.jar:olfjca.jar com.nec.jp.iPX.tool.CobolDataComposer
 -create C:¥WebOTX/Studio/workspace/shopMFDL/ConnectorContent/cdd/SHOP-IN.cdd
 -dir C:¥WebOTX/Studio/workspace/shopMFDL/copy/cobolCopy_ SHOP-INuser.txt
```

(6) マッピングファイルの変更

JSPとAP(Javaソース)をカスタマイズした場合は、マッピングファイル(WhMappingWebApクラス)を変更します。変更方法については、「5.2.1(2)WhMappingWebApクラスの変更」及び「5.3.1(2)WhMapping<コンポーネント名>クラスの変更」を参照してください。

(7) カスタマイズファイルの配備

EJBアプリケーションとWebアプリケーションのパッケージおよび配備を行います。配備の方法については、「3.4.3EJBアプリケーションの配備」、および「3.5.3Webアプリケーションの配備」を参照してください。

5.6.カスタマイズ可能なファイル、クラス、メソッドの説明

本節では、Web アプリケーションとして生成したファイルのうち、カスタマイズ可能なファイルや、Web アプリケーションの各コンポーネントにおけるクラス・メソッドのカスタマイズ内容について記述しています。

5.6.1.カスタマイズ可能なファイル

WebAP が生成したファイルのうち、カスタマイズ可能なファイルは以下です。

ファイルの概要	
<画面名>.jsp	Web ブラウザに表示する画面ファイルです。 罫線や文字色、フィールドの配置など画面デザインを変更します。 別名ファイルにコピーしてカスタマイズします。この別名ファイルは WhMappingWebAp クラスで画面名に関連付けます。
WhError.jsp	Web アプリケーション実行時に発生する例外を Web ブラウザに表示するエラー画面ファイルです。 別名ファイルにコピーしてカスタマイズします。この別名ファイルは WhMappingWebAp クラスでエラー画面として関連付けます。
WhScript.js	Web ブラウザでブリンク制御などを行うためのスクリプトが格納された JavaScript ファイルです。 ブリンク制御や、ブラウザ操作による重複実行ガードで表示するメッセージを変更します。 別名ファイルにコピーしてカスタマイズします。この別名ファイルを<画面名>.jsp から参照します。
WhStylesheet.css	Web ブラウザに表示する画面の表示属性(色・センタリング)を設定したファイルです。 文字色や背景色、画面の表示位置(左寄せ・センタリング・右寄せ)を設定します。 別名ファイルにコピーしてカスタマイズします。この別名ファイルは<画面名>.jsp で参照します。 色設定を変更し Web アプリケーションを再作成すると、別名ファイルにはその色設定が反映されないため、その色設定を反映させる必要があります。
WhCtrl<画面名>.java WhProcessor<画面名>.java WhMessage<画面名>.java	Javaアプリケーションソースです。詳細は「5.6.2Webコンポーネント」以降で説明します。 別名ファイルにコピーし、Java の継承を利用してカスタマイズします。この別名ファイルは WhMapping<コンポーネント名>クラスで画面名に関連付けます。
Whmessage_en.properties Whmessage_ja.properties	メッセージ出力用のプロパティファイルです。 メッセージを追加、変更します。 エディタで直接カスタマイズします。 Web アプリケーションを再作成すると、カスタマイズしたプロパティファイルは上書きされますので、事前に別名ファイルに保存しておく必要があります。
log4j.properties	ログ出力設定用のプロパティファイルです。 ログファイル名、ログファイルサイズ、ログ出力レベル、ログ出力フォーマットなどを変更します。 エディタで直接カスタマイズします。 Web アプリケーションを再作成すると、カスタマイズしたプロパティファイルは上書きされますので、事前に別名ファイルに保存しておく必要があります。

5.6.2.Web コンポーネント

Web コンポーネントのカスタマイズ処理の内容を記載します。

クラスの概要	
WhCtrl <画面名>	画面ごとのコントローラクラスです。 トランザクション処理、入力メッセージ作成処理、入力チェック処理、入力値設定処理がカスタマイズできます。
WhMappingWebAp	画面名に関連付けられたクラス名や JSP 名を取得します。

(1) WhCtrl <画面名>クラス

メソッドの概要	
buildMessage	複数フィールドの入力値を組合せてチェックします。
clearBlink	特定フィールドのブリンク解除の有無を変更します。
generate<フィールド名>	画面の入力値を加工します。 特定フィールドの入力値を他のフィールドの入力値に設定します。 入力チェックを行わず入力メッセージに設定します。
getInMessage	入力メッセージを再作成します。
request	複数のトランザクション処理を実行します。
setInMessage	入力メッセージの特定フィールドにフォーカスを設定します。 入力メッセージの PF キーID に特定値を設定します。
validate<フィールド名>	ブラウザからの入力値について入力チェックの有無を変更します。 入力チェックのエラーコードに対しての処理を追加します。 入力チェック処理を追加します。

(2) WhMappingWebAp クラス

メソッドの概要	
getCtrlClassName	WhCtrl<画面名>クラスの別名を指定します。
getErrJspName	エラー画面 JSP の別名を指定します。
getJspName	画面 JSP の別名を指定します。

5.6.3.EJB コンポーネント

EJB コンポーネントのカスタマイズ処理の内容を記載します。

クラスの概要	
WhProcessor<画面名>	ACOS ホストと通信を行うときの入力電文作成や出力電文展開を行うクラスです。 入力電文作成処理、出力メッセージ作成処理、エラー値設定処理がカスタマイズできます。
WhMappingEJB	カスタマイズした時、画面名に関連付く WhProcessor<画面名>クラス名称を指定します。

(1) WhProcessor<画面名>クラス

メソッドの概要	
buildContinueRecord	継続処理を行うための入力メッセージを設定します。
buildInteractionSpec	トランザクション ID や画面名を変更します。
buildMessage	ブリンク解除やフィールド初期化を変更します。
buildOutRecord	リソースアダプタ用の出力レコードとして特定の cdd ファイルを指定します。
buildRecord	画面の入力値について入力電文への設定を変更します。
clearBlink	特定フィールドのブリンク解除を変更します。
clearField	特定フィールドの初期化を変更します。
generate<フィールド名>	入力フィールドの値について入力電文への設定を変更します。
generateOut<フィールド名>	出力電文から取出した出力フィールドの値を変更します。
isMultiMessage	継続メッセージの有無を変更します。
isLastMessage	継続メッセージの最後を判断します。
makeOutMessageInstance	特定画面の出力メッセージのインスタンスを作成します。
validate<フィールド名>	入力フィールドの値についてエラーの値を変更します。

(2) WhMappingEJB クラス

メソッドの概要	
getProcessorClassName	WhProcessor<画面名>クラスの別名を指定します。

5.6.4.共通コンポーネント

共通コンポーネントのカスタマイズ処理の内容を記載します。

クラスの概要	
WhMessage<画面名>	Web コンポーネントと EJB コンポーネント間での通信媒体クラスです。 フィールド情報の設定値、画面名称取得、フォーカス設定フィールド名取得がカスタマイズできます。
WhMappingCommon	カスタマイズした時、画面名に関連付く WhMessage<画面名>クラス名称を指定します。

(1) WhMessage<画面名>クラス

メソッドの概要	
WhMessage<画面名>	特定フィールドの追加や削除をします。 特定フィールドの初期値、色、罫線や入力チェックの有無を変更します。
getFocus	特定フィールドにフォーカスを当てます。
getMessageName	画面名を変更します。

(2) WhMappingCommon クラス

メソッドの概要	
getMessageClassName	WhMessage<画面名>クラスの別名を指定します。

6.エラーメッセージ

6.1.エラーメッセージ

本節では、GatewayBuilder と生成された WebAP のエラーメッセージの説明、対処方法を示します。

エラーメッセージのプロパティファイルは次の通りです。

・mfdcmessage_en.properties（GatewayBuilder の英文メッセージ）

・mfdcmessage_ja.properties（GatewayBuilder の和文メッセージ）

・Whmessage_en.properties（WebAP の英文メッセージ）

・Whmessage_ja.properties（WebAP の和文メッセージ）

6.1.1.GatewayBuilder のエラーメッセージ

GatewayBuilder のエラーメッセージについて、英文メッセージ、和文メッセージ、説明、対処方法を示します。

なお、エラーメッセージのレベルは、次のようになります。

INFO	情報	単純なメッセージを表示し、WebAP を生成します。
WARN	警告	警告エラーメッセージを表示し、WebAP を生成します。
ERROR	異常	異常エラーメッセージを表示し、WebAP は生成されません。
FATAL	致命的	致命的エラーメッセージを表示し、WebAP は生成されません。 また、致命的エラーメッセージを表示した後　WebAP 生成処理は終了します。

メッセージ ID	内容・説明・対処	
INFO	INFO レベルのメッセージ内容	
AL001	英文	target input filename: @
	和文	MFDLソースファイルは(@)である
	説明	処理する MFDL ソースファイルを示します。
	対処	なし。
AL002	英文	@: created
	和文	画面(@)が生成された
	説明	生成した画面名を示します。
	対処	なし。
AL065	英文	charset(@): going to change
	和文	文字コード(@)を変更しようとしています
	説明	文字コードを変更しようとしています。 @は既存の文字コードを示します。処理は続行します。
	対処	パッケージ内の全てのアプリケーションの文字コードを同一にして下さい。
WARN	WARN レベルのメッセージ内容	
AW007	英文	@: illegal DEVATTR keyword, line: @
	和文	@ は無意味な DEVATTR である　@行

	説明	DEVATTR の端末タイプとして、この値は無意味です。処理は続行します。@は指定された端末タイプを示します。
	対処	正しい端末タイプを指定して、再実行して下さい。
AW018	英文	@: invalid keyword, line: @
	和文	キーワード(@)は不正である @行
	説明	不正なキーワードが@行に指定されています。 @は指定された不正なキーワードを示します。処理は続行します。
	対処	正しいキーワードを指定して、再実行して下さい。
AW019	英文	@: invalid parser keyword, line: @
	和文	キーワード(@)は不正である @行
	説明	不正なキーワード(()=等)が@行に指定されています。 @は指定された不正なキーワードを示します。処理は続行します。
	対処	正しいキーワードを指定して、再実行して下さい。
AW023	英文	@: invalid, line: @
	和文	@ が不正である @行
	説明	不正な文字が@行に指定されています。 @は指定された不正な文字を示します。処理は続行します。
	対処	不正な文字を修正して、再実行して下さい。
AW025	英文	@: invalid decimal, line: @
	和文	@ は不正な十進数である @行
	説明	不正な十進数が@行に指定されています。 @は指定された不正な十進数を示します。処理は続行します。
	対処	10 進数を修正して、再実行して下さい。
AW026	英文	@: invalid LMFD keyword, line: @
	和文	LMFD キーワード(@)は不正である @行
	説明	不正な LMFD キーワードが@行に指定されています。 @は指定された不正な LMFD キーワードを示します。処理は続行します。
	対処	不正な文字を修正して、再実行して下さい。
AW027	英文	@: invalid COR keyword, line: @
	和文	COR キーワード(@)は不正である @行
	説明	不正な COR キーワードが@行に指定されています。 @は指定された不正な COR キーワードを示します。処理は続行します。
	対処	不正な文字を修正して、再実行して下さい。
AW028	英文	@: invalid OPT keyword, line: @
	和文	OPT キーワード(@)は不正である @行
	説明	不正な OPT キーワードが@行に指定されています。 @は指定された不正な OPT キーワードを示します。処理は続行します。
	対処	不正な文字を修正して、再実行して下さい。
AW029	英文	@: invalid VALUES keyword, line: @
	和文	VALUES キーワード(@)は不正である @行
	説明	不正な VALUES キーワードが@行に指定されています。 @は指定された不正な VALUES キーワードを示します。処理は続行します。
	対処	不正な文字を修正して、再実行して下さい。
AW030	英文	@: invalid EDIT keyword, line: @

	和文	EDIT キーワード(@)は不正である　@行
	説明	不正な EDIT キーワードが@行に指定されています。 @は指定された不正な EDIT キーワードを示します。処理は続行します。
	対処	不正な文字を修正して、再実行して下さい。
AW031	英文	@: invalid LFLD keyword, line: @
	和文	LFLD キーワード(@)は不正である　@行
	説明	不正な LFLD キーワードが@行に指定されています。 @は指定された不正な LFLD キーワードを示します。処理は続行します。
	対処	不正な文字を修正して、再実行して下さい。
AW032	英文	@: invalid NUM keyword, line: @
	和文	NUM キーワード(@)は不正である　@行
	説明	不正な NUM キーワードが@行に指定されています。 @は指定された不正な NUM キーワードを示します。処理は続行します。
	対処	不正な文字を修正して、再実行して下さい。
AW033	英文	@: invalid SIGN keyword, line: @
	和文	SIGN キーワード(@)は不正である　@行
	説明	不正な SIGN キーワードが@行に指定されています。 @は指定された不正な SIGN キーワードを示します。処理は続行します。
	対処	不正な文字を修正して、再実行して下さい。
AW035	英文	@: invalid COMPARE keyword, line: @
	和文	COMPARE キーワード(@)は不正である　@行
	説明	不正な COMPARE キーワードが@行に指定されています。 @は指定された不正な COMPARE キーワードを示します。処理は続行します。
	対処	不正な文字を修正して、再実行して下さい。
AW037	英文	@: invalid PSFD keyword, line: @
	和文	PSFD キーワード(@)は不正である　@行
	説明	不正な PSFD キーワードが@行に指定されています。 @は指定された不正な PSFD キーワードを示します。処理は続行します。
	対処	不正な文字を修正して、再実行して下さい。
AW038	英文	@: invalid DEVATTR keyword, line: @
	和文	DEVATTR キーワード(@)は不正である　@行
	説明	不正な DEVATTR キーワードが@行に指定されています。 @は指定された不正な DEVATTR キーワードを示します。処理は続行します。
	対処	不正な文字を修正して、再実行して下さい。
AW040	英文	@: invalid PAGE keyword, line: @
	和文	PAGE キーワード(@)は不正である　@行
	説明	不正な PAGE キーワードが@行に指定されています。 @は指定された不正な PAGE キーワードを示します。処理は続行します。
	対処	不正な文字を修正して、再実行して下さい。
AW043	英文	@: not supported POS keyword, line: @
	和文	POS キーワード(@)は利用できない　@行
	説明	利用できない POS キーワードが@行に指定されています。 @は利用できない POS キーワードを示します。処理は続行します。
	対処	不正な文字を修正して、再実行して下さい。

AW046	英文	@: invalid type VALUEQ keyword, line: @
	和文	VALUEQ キーワード(@)のタイプが不正である @行
	説明	VALUEQ キーワードにて、C,X 以外のタイプが@行に指定されています。 @は指定された不正なタイプを示します。処理は続行します。
	対処	不正な文字を修正して、再実行して下さい。
AW048	英文	@: invalid decimal VALUEQ keyword, line: @
	和文	VALUEQ キーワード(@)は不正な数値である @行
	説明	VALUEQ キーワードにて、不正な数値が@行に指定されています。 @は指定された不正な数値を示します。処理は続行します。
	対処	不正な数値を修正して、再実行して下さい。
AW050	英文	@: invalid FMT keyword, line: @
	和文	FMT キーワード(@)は不正である @行
	説明	不正な FMT キーワードが@行に指定されています。 @は指定された不正な FMT キーワードを示します。処理は続行します。
	対処	不正な文字を修正して、再実行して下さい。
AW051	英文	@: invalid ATR@ keyword, line: @
	和文	ATR@キーワード(@)は不正である @行
	説明	不正な ATR キーワード(ATR,ATR3,ATR4,ATR5)が@行に指定されています。 @は指定された不正な ATR キーワードを示します。処理は続行します。
	対処	不正な文字を修正して、再実行して下さい。
AW053	英文	@: not supported parameter, line: @
	和文	パラメータ(@)は利用できない @行
	説明	利用できないパラメータが@行に指定されています。 @は利用できないパラメータを示します。処理は続行します。
	対処	文字を修正して、再実行して下さい。
AW054	英文	@: invalid parameter
	和文	パラメータ(@)は不正である
	説明	不正なパラメータが指定されています。 @は不正なパラメータを示します。処理は続行します。
	対処	不正な文字を修正して、再実行して下さい。
AW055	英文	@: ignore parameter, line: @
	和文	パラメータ(@)は無意味である @行
	説明	無意味なパラメータが@行に指定されています。 @は無意味なパラメータを示します。処理は続行します。
	対処	不正な文字を修正して、再実行して下さい。
AW056	英文	@: invalid type VALUEX keyword, line: @
	和文	VALUEX キーワード(@)は不正である @行
	説明	不正な VALUEX キーワードが@行に指定されています。 @は指定された不正な VALUEX キーワードを示します。処理は続行します。
	対処	不正な文字を修正して、再実行して下さい。
AW059	英文	@: invalid parameter in Color Property
	和文	Color プロパティの内容(@)が誤っている
	説明	カラープロパティの記述に誤りがあります。 @は不正な記述を示します。処理は続行します。

	対処	カラープロパティファイルの記述を修正して、再実行してください。
AW061	英文	@: invalid charset, make JSP default charset
	和文	不正な文字コード(@)のため、規定値の文字コードで JSP を生成した
	説明	文字コード指定が不正の為、JSP 生成は既定値の"UTF-8"で生成されました。 @は指定された不正な文字コードを示します。処理は続行します。
	対処	charset オプションを修正して、再実行して下さい。
AW062	英文	@: invalid VALUEC keyword, line: @
	和文	VALUEC キーワード(@)は不正である @行
	説明	不正な VALUEC キーワードが@行に指定されています。 @は指定された不正な VALUEC キーワードを示します。処理は続行します。
	対処	不正な文字を修正して、再実行して下さい。
AW063	英文	@: invalid VALUENC keyword, line: @
	和文	VALUENC キーワード(@)は不正である @行
	説明	不正な VALUENC キーワードが@行に指定されています。 @は指定された不正な VALUENC キーワードを示します。処理は続行します。
	対処	不正な文字を修正して、再実行して下さい。
AW064	英文	@: invalid Hexadecimal Data, line: @
	和文	(@)は不正な 16 進データである @行
	説明	不正な 16 進データが@行に指定されています。 @は指定された不正な 16 進データを示します。処理は続行します。
	対処	不正な文字を修正して、再実行して下さい。
ERROR	ERROR レベルのメッセージ内容	
AE011	英文	duplicate tableData, POSH: @
	和文	POS(@)にてテーブルデータが生成できない
	説明	テーブルデータが作成できませんでした。 @は PFLD の POS を示します。 処理中の画面処理は中断され、次の画面処理を続行します。
	対処	同一 POS(@)の PFLD は、行数や桁数を変更したりして、再実行して下さい。
AE012	英文	not found @ in LMFD
	和文	LMFD 定義に LFLD 名(@) が存在しない
	説明	PSFD 定義と LMFD 定義の LFLD 名が不一致です。 @は LFLD 名を示します。 処理中の画面処理は中断され、次の画面処理を続行します。
	対処	PSFD 定義と LMFD 定義の LFLD 名を一致させて、再実行して下さい。
AE013	英文	error PFLD(@): @
	和文	PFLD 名(@) が解析エラーである @行
	説明	PFLD(@) が構文解析エラーになりました。 セパレータ(,()"等)を確認して下さい。
	対処	PFLD 定義を修正して、再実行して下さい。
AE041	英文	@: invalid LINREP(nn,m)/FLDREP(nn*kk,m[,LEN=l]) keyword, line: @
	和文	LINREP(nn,m)/FLDREP(nn*kk,m[,LEN=l])キーワード(@)は不正である @行
	説明	LINREP や FLDREP の不正な記述が、@行に指定されています。 @は不正な記述を示します。 処理中の画面処理は中断され、次の画面処理を続行します。

	対処	不正な記述を修正して、再実行して下さい。
AE042	英文	@: not supported LINREP(kk)/LINREP(nn*kk,m) keyword, line: @
	和文	LINREP(kk)/LINREP(nn*kk,m)キーワード(@)は利用できない　@行
	説明	LEN 指定なしの LINREP が、@行に指定されています。 @は指定された LINREP キーワードを示します。 処理中の画面処理は中断され、次の画面処理を続行します。
	対処	LINREP キーワードに LEN を指定して、再実行して下さい。
AE049	英文	invalid OVERLAY POS[V=@, H=@]
	和文	OVERLAY キーワードの POS[V=@, H=@]が不正である
	説明	OVERLAY の POS 指定が誤っています。 @は OVERLAY の POS 指定を示します。 処理中の画面処理は中断され、次の画面処理を続行します。
	対処	OVERLAY の POS 指定を修正して、再実行して下さい。
AE057	英文	not MFDL file
	和文	MFDL ソースファイルではない
	説明	MFDLソースファイルの内容が不正です。 処理中の画面処理は中断され、次の画面処理を続行します。
	対処	正しいMFDLソースファイルを指定して、再実行して下さい。
AE058	英文	@: already existed
	和文	画面(@)は既に作成済みである
	説明	既に作成済みの画面名と同じ画面名が指定されました。 処理中の画面処理は中断され、次の画面処理を続行します。
	対処	正しいMFDLソースファイルを指定して、再実行して下さい。
AE060	英文	PSFD is not Screen
	和文	PSFD 定義は画面ではない
	説明	PSFD 定義が画面定義ではありません。 処理中の画面処理は中断され、次の画面処理を続行します。
	対処	画面定義のMFDLソースファイルを指定して、再実行して下さい。
FATAL	FATAL レベルのメッセージ内容	
AF001	英文	@: invalid parameter
	和文	@ は不正なパラメータである
	説明	パラメータが誤っています。 @は指定された不正なパラメータを示します。処理は終了します。
	対処	パラメータを修正し、再実行して下さい。
AF002	英文	@: invalid option parameter
	和文	@ は不正なオプションパラメータである
	説明	オプションパラメータが誤っています。 @は指定された不正なオプションパラメータを示します。処理は終了します。
	対処	オプションパラメータを修正し、再実行して下さい。
AF003	英文	@: invalid path
	和文	@ は不正なパスである
	説明	パスが誤っています。 @は指定された不正なパスを示します。処理は終了します。
	対処	パスを修正し、再実行して下さい。

AF005	英文	@: invalid directory
	和文	デイレクトリ（@）が不正である
	説明	デイレクトリが誤っています。 @は指定された不正なデイレクトリを示します。処理は終了します。
	対処	デイレクトリを修正し、再実行して下さい。
AF066	英文	package(@): going to change
	和文	パッケージ名(@)を変更しようとしています
	説明	パッケージ名は変更できません。 @は既存のパッケージ名を示します。処理は終了します。
	対処	既存のパッケージ名を指定し、再実行して下さい。
AF090	英文	system error
	和文	システムエラーとなった
	説明	予期しないエラーが発生しました。 終了します。
	対処	PSFD 定義や LMFD 定義の構文を確認/修正し、再実行して下さい。
AF091	英文	syntax analyze error
	和文	構文解析エラーとなった
	説明	MFDL ファイル解析中に予期しないエラーが発生しました。 終了します。
	対処	PSFD 定義や LMFD 定義の構文を確認/修正し、再実行して下さい。

6.1.2.WebAP のエラーメッセージ

WebAP のエラーメッセージについて、英文メッセージ、和文メッセージ、説明、対処方法を示します。

メッセージ ID	内容・説明・対処	
WhERR_INPUT	英文	@ is Not Input
	和文	項目(@)は未入力である
	説明	LFLD 名(@)で示した項目は未入力です。なお、FIL 文字は入力値と見なしません。
	対処	FIL 文字以外の値を入力して下さい。
WhERR_CHANGE	英文	@ is Not Change
	和文	項目(@)は入力値が変更されていない
	説明	LFLD 名(@)で示した項目の入力値が変更されていません。
	対処	入力値を変更してください。
WhERR_NUM	英文	@ is Not Number
	和文	項目(@)は数字が入力されていない
	説明	LFLD 名(@)で示した項目の入力値が数字ではありません。
	対処	入力値に0～9、$や,(カンマ)を入力して下さい。
WhERR_NULL	英文	@ is Null
	和文	項目(@)は存在しない
	説明	LFLD 名(@)で示した項目が存在しません。
	対処	対象項目が JSP 上から削除されていないか確認して下さい。

WhERR_KANJI	英文	@ is Not Kanji
	和文	項目(@)は漢字が入力されていない
	説明	LFLD 名(@)で示した項目の入力値が全角文字ではありません。
	対処	入力値を全角文字で入力して下さい。
WhERR_RANGCOMP	英文	@ RNG/VALUES/COMPARE error detected
	和文	項目(@)は RNG/VALUES/COMPARE 等の入力エラーとなった
	説明	LFLD 名(@)で示した項目の入力値が RNG、VALUES、COMPARE の何れかのチェックでエラーになりました。
	対処	RNG、VALUES、COMPARE の何れかのエラーにならない入力値を指定して下さい。
WhERR_DPNTH	英文	@ DPNT overflow error detected
	和文	項目(@)は上位桁が桁落となった
	説明	LFLD 名(@)で示した項目の整数部が指定された桁数より、多く入力されています。
	対処	整数部を指定された桁数内で入力して下さい。
WhERR_DPNTL	英文	@ DPNT underflow error detected
	和文	項目(@)は下位桁が桁落となった
	説明	LFLD 名(@)で示した項目の小数部が指定された桁数より、多く入力されています。
	対処	少数部を指定された桁数内で入力して下さい。
WhERR_DPNTHL	英文	@ DPNT overflow and underflow error detected
	和文	項目(@)は上位桁、下位桁ともに桁落となった
	説明	LFLD 名(@)で示した項目の整数、小数部共に指定された桁数より、多く入力されています。
	対処	整数部、少数部を指定された桁数内で入力して下さい。
WhERR_DPNT	英文	@ DPNT error detected
	和文	項目(@)は小数点(.)が入力された
	説明	LFLD 名(@)で示した項目の入力値に、小数点が入力されました。
	対処	入力値から小数点を削除して下さい。
WhERR_DESIG	英文	@ DESIG error detected
	和文	項目(@)は入力が省略された
	説明	LFLD 名(@)で示した省略不可能項目に入力値が存在しません。
	対処	値を入力して下さい。
WhERR_EQU	英文	@ EQU error detected
	和文	項目(@)は全桁が入力されていない
	説明	LFLD 名(@)で示した項目に全桁が入力されていません。
	対処	LFLD のサイズ分の値を入力してください。
WhERR_DECIMAL	英文	@ is Illegal Decimal data
	和文	項目(@)は不正な十進数が入力された
	説明	LFLD 名(@)で示した項目に不正な 10 進数が入力されました。
	対処	正しい 10 進数を入力して下さい。
WhERR_OVERFLOW	英文	@ Overflow error detected
	和文	項目(@)は桁あふれとなった
	説明	LFLD 名(@)で示した項目の入力桁数が、LFLD のサイズより大きいため、桁あふ

		れとなりました。
	対処	入力の桁数は LFLD のサイズより少ない桁を入力して下さい。
WhERR_SIGN	英文	@ SIGN error detected
	和文	項目(@)は不正な符号が入力された
	説明	LFLD 名(@)で示した符号なし数値項目に符号が入力されたか、複数の符号が入力されました。
	対処	符号なし項目には符号を指定しないで下さい。 符号あり項目には正しい符号を指定して下さい。
WhERR_SPECIAL	英文	@ is Not Decimal
	和文	項目(@)は十進数が入力されていない
	説明	LFLD 名(@)で示した項目の入力値が特殊文字のみです。
	対処	入力値に0～9、$や,(カンマ)を入力して下さい。
WhERR_AADATA	英文	AAData initialization failure @ not found, invalid
	和文	初期化ファイル(@)が不正なため、AAData の初期化に失敗した
	説明	初期化ファイルが不正なため、AAData ライブラリの初期化に失敗しました。
	対処	初期化ファイルの内容を確認して下さい。
WhERR_CDD	英文	cdd initialization failure, /cdd not found
	和文	/cdd が検出できないなため、cdd の初期化に失敗した
	説明	cdd フォルダや cdd ファイルが存在しないため、初期化に失敗しました。
	対処	cdd フォルダと cdd ファイルを配置して下さい。

7.MFDL 機能のサポート範囲

7.1.MFDL 機能のサポート範囲

本節では、MFDL 機能のサポート範囲について記述しています。

7.1.1.LMFD 機能のサポート範囲

対応状況の記号の意味は、以下のとおりです。
○：機能組み込み　　△：一部機能組み込み　　－：サポート対象外パラメータ

キーワード	パラメータ	設定値	機能内容	対応状況
LMFD	第一定位置パラメータ	lmfd 名	メッセージ形式定義名（画面名）	△
	第二定位置パラメータ	psfd 名	対応するセクション形式定義名	○
	AMOD		メッセージの入出力属性を指定する。 使用する端末種別に対応する。	
		B	入出力メッセージ形態(入出力あり) (PSFD 定義では CRT 端末などに対応)	○
		I	入出力メッセージ形態(入力のみ) (PSFD 定義ではキーボード端末などに対応)	○
		O	入出力メッセージ形態(出力のみ) (PSFD 定義ではプリンタ端末などに対応)	○
LMSG	第一定位置パラメータ	lmsg 名	入出力メッセージ名	○
		*	入出力メッセージに名前が不要な場合に指定する。	○
	メッセージの入出力属性	INPUT	入力処理用メッセージ定義	○
		OUTPUT	出力処理用メッセージ定義	○
	MCA		TPP からのメッセージ出力時、MCA 領域内の下記のフィールドの内容によって、表示方法を制御する。	
		PARTIAL	一部のデータを更新して表示する場合にオンにする。 オン：部分送信 オフ：通常送信（全送信）	○
		VARIABLE	可変長画面、可変長ページング画面にて使用する。 オン：可変長送信要求 オフ：通常送信要求	－
		PROTECT	可変長送信時、以降を入力不可項目に変更する。	－
		CLEAR	可変長送信時、指定された行以降を null クリアする。	－
		NODATA	部分送信時、MODIFY を指定したフィールドはデータは送信せず、ブリンク、色変更等を行う。	○
		SOPT	部分送信（PARTIAL）時、日付、時刻の処理を行う	○
		LOCAL	予め画面フォーマットデータをクライアント PC に保存しておき、そのローカル画面(FMR 命令)を表示する。 通常は ACOS から画面フォーマットデータと表示データをおくる。	－
		BCLR	ブリンクフィールドを通常フィールドに変更する	○
		NCLR	全ての非保護フィールド（打鍵入力可能なフィールド）を NULL クリアする。	○

		MOFF	全ての非保護フィールドの MDT ビットをオフにする(入力省略不可とする)。 (INM, IO の属性を IN,IOF へ変更)	△
		CPMDFY	下記の SCLR,LOCK を有効とする。 SCLR:現画面をクリア後表示 LOCK:キーボード入力不可	ー
	MODIFY	CURSOR	カーソルを位置づける	○
		SEND	当該フィールドデータを送信する	○
		ATR	当該フィールドの保護(更新不可)/非保護、送信/非送信(入力省略不可)の属性を変更する	△
		BLINK	ブリンク	○
		COLOR	色変更	○
		SLIT	罫線表示変更	ー
	IND	nn:mm	標識に関する指定を行う。	ー
	INDRED		標識に関するコピー項目を変更する。	ー
	OPT	ERR	エラー用の通知フィールドを付加する。 NOCOR 指定時 エラー値を TPP へ返却する。	○
		COR(NOCOR / BLINK)	入力編集誤りの処理方法 NOCOR:エラー値を TPP へ渡す BLINK:ブリンク表示する 無指定:!表示を行う	○
		BS	入力専用端末で有効な機能	ー
		DEL	入力専用端末で有効な機能	ー
		CAN(FLD/MSG)	入力専用端末で有効な機能	ー
		FTAB(DMY/SKP)	入力専用端末で有効な機能	ー
LREC	lrec 名		lrec 名 メッセージを分割する場合使用する	ー
LFLD	lfld 名		フィールド名 1 バイトカタカナ文字や 2 バイト文字もありえる	△
	FLDREP	nn / nn*mm	フィールド繰り返し	○
	SETREP	nn / nn*mm	フィールド群(行)繰り返し開始項目	○
	SETEND		フィールド群(行)繰り返し終端項目	○
	MODIFY		LMSG の MODIFY と同一内容。 LMSG での指定は全ての LFLD に対して機能する。	○
	PKEY		プログラムキー	ー
	PASSWORD		パスワード格納領域	ー
	ACOUNT		課金単位格納領域	ー
	USER		利用者名格納領域	ー
	OPT	DESIG	入力省略不可フィールド (打鍵入力しないと入力とみなさない)	△
		EQU	物理フィールド桁数に等しい桁数の入力データが必要	○
		SER(VSN1/VSN2/TSN1)	帳票にて出力通番を指定する	ー
		DATEJ	YY-DDD 形式の日付(西暦)	○
		DATEM	MM-DD-YY 形式の日付(西暦)	○
		DATED	DD-MM-YY 形式の日付(西暦)	○
		DATEY	YY-MM-DD 形式の日付(西暦)	○
		DATES	YY.MM.DD 形式の日付(和暦)	○
		TIME	HH:MM:SS 形式の時刻	○
		CRTID	ファンクションキーに対応する識別コード通知フィールド	○
		CURSOR	カーソル位置を格納するフィールドを定義する。 出力時:出力値で指定された位置にカーソルを位置づける 入力時:現在のカーソル位置を TPP に渡す	△

	SIZE		フィールド長	○
	EDIT	ALPHA	英数字項目	○
		KANJI	漢字項目	○
		MIX	1 バイト文字と 2 バイト文字の混在項目	○
		NUM	数字項目	○
		DPNT	数値の小数点桁数を指定する コピー項目例 ：08　xxxx　　9(x)V9(x). 入力時、小数点の桁あわせを行う。	○
		SIGN	数値の符号編集	○
		RNG	数値の範囲指定	○
		PCK	数値のアンパック→パック変換（特定端末）	－
		UNP	数値のパック→アンパック変換（特定端末）	－
		COMPARE	数値の入力値比較	○
		OFRM	編集表示するための編集用文字	○
	LABEL		COBOL の項目名	－
	VALUES		入力値チェック	○
INDLABEL			標識のコピー項目名指定	－
/COPY			コピー項目の内容を変更する	－
ENDLMFD			LMFD 定義の終わりを示す	○

7.1.2.PSFD 機能のサポート範囲

対応状況の記号の意味は、以下のとおりです。
　　○：機能組み込み　　△：一部機能組み込み　　－：サポート対象外パラメータ

キーワード	パラメータ	設定値	機能内容	対応状況
PSFD	第一定位置パラメータ	psfd 名	セクション形式定義名	○
	第二定位置パラメータ	lmfd 名	対応するメッセージ形式定義名（画面名）	○
	OUTSCR		入出力フィールドを持たない表示のみの定義	－
	INTERMED		中間罫線 （フィールド中に縦罫線を指定する）	－
	PAGEREP		ページリピート 出力のみの画面で、TPP の複数送信により、複数画面が表示される。	－
DEVATTR	TYPE		端末タイプ N 型番を指定する。	△
	AMOD		端末の入出力属性	
		READ	入力端末（キーボード）	－
		WRITE	出力端末（プリンター）	－
		RD_WR	入出力端末（ディスプレイ）	○
	OPT	MSET	帳票の 1 頁の大きさ	－
		ATAB	帳票の TAB 位置指定	－
		SINGLE	帳票にて単票指定	－
		HINST	帳票にて単票指定	－
		HTAB/VTAB	帳票の水平 TAB、垂直 TAB	－
		MONO/COLOR	モノクロ・カラー端末指定	－
SEC	sec 名		sec 名	○
PAGE	page 名		ページ名 複数の PAGE 画面は 1 画面として作成する。	○
	CLEAR		前画面消去を行う。 端末のバッファをクリアして送信画面を表示する。	○
	CP		前画面消去、入力不可・入力可にする。	－
	AUTO		非保護項目の終端指定 入力項目の終わりを作成する。	○
	AUTOP		入力項目の終端指定 保護・非保護にかかわらず、入力項目の終わりを作成する。	○
	SECRET		ハードコピー時にシークレット項目は印字しない。	○
	ATR	BEHIND/BEHIND2/OCCUPY	画面上制御文字を占有するか、占有しないかの指定	－
	PARTIAL		標識機能（CLEAR,CP 機能を有効とする）	－
	SCREEN		ローカル画面番号 端末に画面イメージデータを保存し、利用する。	－
	DISPLAY		1 画面の表示行数を指定する。	－
	FOMFEED	HEAD/BOTTOM/	帳票にて改ページ指定	－
	LINK		TOOLS-N プログラムのリンク番号を指定する。 （TOOLS-N プログラムによる入出力処理を行う）	－
	CURSOR	rrcc	指定位置に対するカーソル位置付け	△
		UNPROT	非保護第一項目にカーソル位置付け	○
PFLD	第一定位置パラメータ	lfld 名	フィールド名 1 バイトカタカナ文字や 2 バイト文字もありえる	△
	第二定位置パラメータ		入出力属性を指定する	

	IN	入力フィールド(入力省略不可) 打鍵入力しないと入力されない。	△
	INM	入力フィールド(入力省略可) 打鍵入力しない場合、表示している文字が入力される。	○
	INP	入力フィールド(値の変更不可) 打鍵入力不可、表示している文字を入力する。	○
	OUT	出力フィールド TPP からのデータを表示する。	○
	IO	入出力フィールド 打鍵入力可能、打鍵入力しなければ、表示文字が入力される。 TPP からのデータを表示する。	○
	IOP	入出力フィールド (値の変更不可) 打鍵入力不可能な 入出力フィールド	○
	IOF	入出力フィールド (入力省略不可) 打鍵入力しないと入力されない 入出力フィールド	△
	CNS	固定値フィールド MFDL で指定された表示データを固定表示する。	○
	*IN	LFLD に CRTID,PKEY パラメータを指定した時 必須となる。	△
	*OT	CURSOR パラメータを指定した時 必須となる。	△
	*EDT	COMPARE/VALUES/RNG/OFRM を指定した時 必須となる。	○
FLDREP	kk / nn,m / nn*kk,m LEN	フィールド繰り返し kk 横方向の繰り返し回数 nn 縦方向の繰り返し回数 n 改行数 nn*kk 縦繰り返し回数*横繰り返し回数 LEN 繰り返しの間隔	○
LINREP	kk / nn,m / nn*kk,m LEN	フィールド群(行)繰り返しの開始項目を指定する。 各値は FLDREP と同様。 kk(横方向)を指定する場合、LEN を指定する。	○
LINEND		フィールド群(行)繰り返しの終端項目	○
RESEND		再送時のみ表示するフィールド 回線障害等による VIS の再送信機能で表示する。	―
TEST		テストモード時のみ表示するフィールド	―
FIL	C / X / NC	フィル文字の指定 入力値からフィル文字で指定された文字を削除する。	○
SCRSZ		混在(MIX)フィールド時の項目表示サイズ	○
DATASZ		混在(MIX)フィールド時の物理データ長 混在(MIX)フィールド以外の物理データ長は VALUE で示した初期値の大きさとなる。	―
INCREMENT		行繰り返し定数データの増分値を指定する。	○
POS		フィールド位置(rrcc)を示す。	○
ATR	ALPHA	フィールド属性定義(半角文字のみ入力可能)	○
	NUM	フィールド属性定義(数値(0～9)と特殊文字 (+−.¥$,)のみ入力可能。 符号、少数点の位置変更等を行う。	○
	KANJI	フィールド属性定義(2 バイト文字のみ入力可能)	○
	MIX	フィールド属性定義(すべての文字が入力可能)	○
	NORM	通常輝度で表示する。	○
	HIGH	高輝度で表示する。	―
	BLINK	ブリンク	○
	NODISP	非表示	○
	NMPEN	通常輝度のライトペン入力	―
	HIPEN	高輝度のライトペン入力	―
CNT 色		文字色を指定する。	○

	CNT 罫線		罫線表示を指定する。	○
	ATR3	JUSTIFY	フィールド属性定義(右寄せ) 入力データを右詰で表示する。	○
		REVERSE	反転	○
		ALPHA	半角英数字のみ入力可能	○
		KANA	半角カタカナのみ入力可能	－
		OVERFLOW	項目のオーバーフローチェック 自動的に次項目にフォーカスを移動させない	－
		VERIFY	入力値の 2 重チェック 2 回打鍵入力することによって次項目の処理ができる。	－
		TR1	入力操作を TR1(入力装置)で行う。	－
		TR2	入力操作を TR2(入力装置)で行う。	－
	ATR4		TOOLS-N プログラムとのリンク番号 (特定のカーソル位置で TOOLS-N プログラムを起動する)	－
	ATR5	MAT	入力操作(MAT モード)を示す。	－
		IDCR	入力操作(IDCR モード)を示す。	－
		KYBD	入力操作(キーボードモード)を示す。	－
		KBMAT	入力操作(MAT モードとキーボードモード)を示す。	－
	VALUEC		初期値(半角文字表記)を示す。	○
	VALUEX		初期値(16 進表記)を示す。	○
	VALUENC		初期値(全角文字表記)を示す。	○
	VALUEQ	size/C/NC/X	初期値を持たない領域を示す。	○
	OVERLAY		繰返し項目に対して個々に再定義する。	○
	CNT1		初期値の直前の CNT を変更する。	○
	CNT2		初期値の直後 の CNT を変更する。	○
	VALUEC/VALUENC		初期値を変更する。	○
CHANGE			標識機能を示す。	－
ENDPSFD			PSFD 定義の終わりを示す。	○

その他機能	パラメータ	設定値	機能内容	対応状況
可変長画面表示			1 画面の途中までを表示する。以降の項目は定数項目、罫線等を含めて表示しない。	－
出力画面合成			入力属性がなく、出力属性のみの画面を表示している画面に追加表示する。	－
可変長ページング			定義された複数ページの一部のページまでを表示する。	－
入出力属性の IN,IOF 属性			ACOS 上 打鍵入力することによって入力データとして扱うが、ブラウザではそのチェックができないため、 null が入力された時と画面上の表示値と同じ値が入力された時、入力データ無とみなし、エラーとする。	△
VISUCA の TRNS-PFKEY 項目			VISUCA TRNS-PFKEY に PFKey 情報を設定する。	○
トランザクション起動 PFKEY			PFKey を使って、任意のトランザクションを起動する。	○

＜文字色について＞

①ETOS 画面では背景色が黒色で、文字色の既定値は緑ですが、Web 画面の背景色は白色のため、文字色の既定値は黒色にしています。
また、白色が指定された場合、黒色表示にしています。

8.注意事項

8.1.注意事項

本節では、生成されるjava ソースや jsp における注意事項を示します。

8.1.1.LMFD 名、LFLD 名と Java 識別子の関係

GatewayBuilder が生成する Web アプリケーションのクラス名、メソッド名、変数名には、MFDL ソースの LMFD 名や LFLD 名を使用しています。

なお、下記に示すように、MFDL ソースの LMFD 名、LFLD 名を ERR-2L、FLD-01 と指定すると、「-」は Java 識別子の規約により使用できないため、LMFD 名と LFLD 名の「-」を「_」として Web アプリケーションを生成します。

<MFDL ソース例>

```
LMFD(ERR-2L,ERR02 ,AMOD(B))
  LMSG(*,INPUT)
    LFLD(FLD-01)        SIZE(7)
```

<生成された java ソース例>

```
WhMessageERR_2L.java
private WhField   FLD_01;
  public WhField getFLD_01 () {
```

また、Java 識別子は以下の規約があるため、MFDL ソースの LMFD 名や LFLD 名は以下の規約に沿っていなければなりません。

<Java 識別子の規約>

① Java の予約語は使用できない。

② 最初の一文字目として、「$」や「_」以外の記号、数字は使用できない。

③ 二文字目以降として、「$」や「_」以外の記号は使用できない。

なお、Java(J2SE1.4)の予約語は以下の内容です。

abstract boolean break byte case catch char class const continue

default do double extends final finally float for goto

if implements import instanceof int interface long

native new package private protected public return

short static super switch synchronized this

throw throws transient try void volatile while

8.1.2.表示桁位置の変更

GatewayBuilder は、MFDL ソースの ATR、CNT や縦線を非占有扱いで JSP を作成します。
そのため、上下行で異なる PFLD 指定があれば、上下行の表示文字の桁位置がずれることがあります。

<MFDL ソース例>

```
PFLD ...  POS(0501)   FMT(ATR, CNT(ULV), VALUE)   VALUEC"xxx"
PFLD ...  POS(0602)   FMT(ATR, VALUE)   VALUEC"yyy"
```

<生成された jsp 例>

下記例は、表示値「xxx」と次行の表示値「yyy」の桁位置を 01 桁にします。
修正内容は下記　箱枠部分を削除することにより、表示値「yyy」の桁位置を 01 桁に修正します。

```
<tr>                                  #行の開始を示す。
    <td colspan=3>                    #行内の先頭 td タグであるため、01 桁、PFLD サイズが  3 である。
      xxx                             #表示値「xxx」を示す。
    </td>
      :
<tr>                                  #行の開始を示す。
    <td colspan=1>                    #行内の先頭 td タグであるため、01 桁、PFLD サイズが  1 である。
                                 #表示値は、半角スペースを示す。
    </td>
    <td colspan=3>                    #行内の先頭 td タグからの colspan の合計値と先頭桁 01 を加え 02
                                       桁となる。 PFLD サイズは  3 である。
      yyy                             #表示値「yyy」を示す。
    </td>
      :
```

< tr タグ、td タグの説明>

・PSFD 定義の行桁位置(POS)は、tr タグで行を示し、td タグの colspan を使って桁位置と PFLD のサイズを表します。

・任意の td タグで示した PFLD の桁位置は、その td タグの直前までの colspan の合計値と先頭桁 01 を加えることにより求められます。

8.1.3.PFLD の上書き

GatewayBuilder は、MFDL ソースの PFLD の上書きが発生する指定が行われた場合、次のように動作します。

ここでは、LINREP の表示属性 PFLD に単一の表示属性 PFLD を上書きした例を示します。

表示属性 PFLD(0601)の"yy"に表示属性 PFLD(0602)の"x"を上書きして "yx"と表示します。

<PSFD 定義例>

```
PFLD ... LINREP(3,1)
PFLD ... POS(0502)   FMT(ATR,VALUE)   VALUEC"x"
PFLD ... LINEND
PFLD ... POS(0601)   FMT(ATR,VALUE)   VALUEC"yy"
```

<表示例>　　yx

なお、入力系属性と出力属性の PFLD の場合、以下の動作となります。

1.入力系属性(IN INM INP IO IOF IOP)PFLD に対する上書き指定は許可しないため、エラーとなります。

2.出力属性(OUT)PFLD に対する上書きの場合、部分表示になります。

POS(0601)の"yy"の後に POS(0602)の"x"を生成するため、先頭の"y"を表示します。