Sterling Selling and Fulfillment Foundation

エンド・ユーザー用コンソール JSP インターフェースのカスタマイズ

バージョン 9.1

Sterling Selling and Fulfillment Foundation

エンド・ユーザー用コンソール JSP インターフェースのカスタマイズ

バージョン 9.1

お願い

本書および本書で紹介する製品をご使用になる前に、 201 ページの『特記事項』に記載されている情報をお読みください。

本書は、IBM Sterling Selling and Fulfillment Foundation バージョン 9.1、および新しい版で明記されていない限り、以降のすべてのリリースおよびモディフィケーションに適用されます。

お客様の環境によっては、資料中の円記号がバックスラッシュと表示されたり、バックスラッシュが円記号と表示されたりする場合があります。

原典： Sterling Selling and Fulfillment Foundation
Customizing Console JSP Interface for
End-User
Version 9.1

発行： 日本アイ・ビー・エム株式会社

担当： トランスレーション・サービス・センター

第1刷 2012.4

目次

第 1 章 カスタマイズ・プロジェクトのチェックリスト (Checklist for Customization Projects)

カスタマイズ・プロジェクト

Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales をカスタマイズまたは拡張するプロジェクトは、必要な変更のタイプによってさまざまです。ただし、ほとんどのプロジェクトは、特定の順序で最適に実行される、相互接続された変更の連続が関係します。チェックリストは、カスタマイズ・タスクの最も一般的な順序を特定し、文書セットのどのガイドが各ステージの詳細を提供するかを示します。

開発環境の準備

WebLogic、WebSphere®、または JBoss アプリケーション・サーバーにアプリケーションをデプロイするかどうかなど、実稼働環境をミラーリングする開発環境を設定します。これを行うことによって、拡張をリアルタイム環境でテストできるようになります。

実稼働環境にアプリケーションをインストールしてデプロイする手順と同じ手順で開発環境にアプリケーションをインストールしてデプロイします。詳細については、システム要件およびインストール文書を参照してください。

Microsoft COM+ でアプリケーションをカスタマイズするオプションがあります。Microsoft COM+ を使用すると、セキュリティーの向上、パフォーマンスの向上、サーバー・アプリケーションの管理の容易性の向上、混合環境のクライアントのサポートなどの利点を得られます。これを選択する場合、インストールの指示について詳しくは、*カスタマイズ基本ガイド*を参照してください。

カスタマイズの計画

新しいメニュー項目を追加していますか。または、サインイン画面またはロゴをカスタマイズしていますか。または、表示またはウィザードをカスタマイズしていますか。または新しいテーマまたは新しい画面を作成していますか。カスタマイズのそれぞれのタイプの範囲と複雑さはさまざまです。

背景については、実行できる変更のタイプを要約し、ファイル名、キーワード、およびその他の一般的な規則に関するガイドラインを提供している*カスタマイズ基本ガイド*を参照してください。

データベースの拡張

多くのカスタマイズ・プロジェクトにおいて、最初のタスクは、データベースを拡張し、後に行う UI または API の他の変更をデータベースがサポートするようにすることです。この指示については、以下のトピックについて説明しているデータベースの*拡張ガイド*を参照してください。

- データベースで変更できるものおよび変更できないものに関する重要なガイドライン。
- API の変更に関する情報。任意の API が影響を受けるようなデータベース表の変更を行った場合、これらの API のテンプレートを拡張する必要があります。そうしない場合、データベースへのデータの格納またはデータベースからのデータの取り出しを実行できません。この手順は、テーブルの変更が API に影響する場合に必要になります。
- エンティティー・レベルでレコードをトラッキングしてレコード管理を向上させるために監査参照を生成する方法。この手順はオプションです。

API に対するその他の変更の実施

アプリケーションは、標準 API またはカスタム API の呼び出しまたは起動を実行できます。API に関する背景およびサービス・タイプ、動作、ならびにセキュリティーのサービス・アーキテクチャーについては、*API のカスタマイズ・ガイド*を参照してください。このガイドでは、以下のタイプの変更について説明します。

- UI でのデータの表示および UI で行われた変更のデータベースへの保存を行う標準 API の呼び出し。
- 拡張サービス定義およびパイプライン構成でカスタム・ロジックを実行するためのカスタマイズ API の呼び出し。
- API は、入力および出力の XML を使用し、データベースにデータを格納し、またデータベースからデータを取り出します。これらの API 入力および出力の XML ファイルを拡張しない場合、ビジネス・ロジック実行時に UI で必要な結果を取得できない場合があります。
- それぞれの API 入力および出力の XML ファイルには、このファイルに関連付けられた DTD および XSD があります。入力および出力の XML を変更したときには必ず、対応する DTD および XSD を生成し、データ保全性を確保する必要があります。拡張 XML に対して DTD および XSD を生成しないと、不整合データを取得する場合があります。

UI のカスタマイズ

IBM® アプリケーションは、いくつかの UI フレームワークをサポートしています。アプリケーションおよび実行するカスタマイズに応じて、これらのフレームワークの 1 つのみまたはいくつかで作業できます。各フレームワークには、メニュー項目、ロゴ、テーマなどのコンポーネントをカスタマイズする独自のプロセスがあります。

必要なフレームワークに応じて、以下のガイドのいずれかを参照してください。

- *カスタマイズ・ガイド (コンソール JSP インターフェース) (Customizing the Console JSP Interface Guide)*
- *カスタマイズ・ガイド (Swing インターフェース) (Customizing the Swing Interface Guide)*
- *カスタマイズ・ガイド (モバイル・デバイス向けユーザー・インターフェース) (Customizing User Interfaces for Mobile Devices Guide)*

- *カスタマイズ・ガイド (リッチ・クライアント・プラットフォーム) (Customizing the Rich Client Platform Guide)* および *RCP 拡張性ツール使用ガイド (Using the RCP Extensibility Tool Guide)*
- *カスタマイズ・ガイド (Web UI フレームワーク) (Customizing the Web UI Framework Guide)*

必要なフレームワークに応じて、以下のガイドのいずれかを参照してください。

- *カスタマイズ・ガイド (コンソール JSP インターフェース) (Customizing the Console JSP Interface Guide)*
- *カスタマイズ・ガイド (Swing インターフェース) (Customizing the Swing Interface Guide)*
- *カスタマイズ・ガイド (モバイル・デバイス向けユーザー・インターフェース) (Customizing User Interfaces for Mobile Devices Guide)*
- *カスタマイズ・ガイド (リッチ・クライアント・プラットフォーム) (Customizing the Rich Client Platform Guide)* および *RCP 拡張性ツール使用ガイド (Using the RCP Extensibility Tool Guide)*
- *カスタマイズ・ガイド (Web UI フレームワーク) (Customizing the Web UI Framework Guide)*

トランザクションの拡張

条件ビルダーを拡張し、外部システムと統合することによって、アプリケーションの標準機能を拡張できます。トランザクション・タイプの背景、セキュリティー、動的変数、および条件ビルダーの拡張については、*トランザクションの拡張ガイド* および *条件ビルダーの拡張ガイド* を参照してください。これらのガイドでは、以下のタイプの変更について説明します。

- 条件ビルダーを拡張して、カスタム・ビジネス・ロジックを実行し、属性の静的セットを使用するための複雑で動的な条件の定義。
- 変数を定義して、アクション、エージェント、およびサービスの構成に属するプロパティーの動的構成。
- 誰がどのデータにアクセスできるか、どれだけの量を表示できるか、およびデータで何を実行できるかを制御するトランザクション。データ・セキュリティーの設定。
- カスタムの時間トリガー・トランザクションの作成。ご使用のアプリケーションが提供する時間トリガー・トランザクションの呼び出しおよびスケジューリングとほぼ同じ方法でカスタムの時間トリガー・トランザクションの呼び出しおよびスケジューリングを実行できます。
- カスタムの時間トリガー・トランザクションを外部トランザクションと調整し、イベントの起動、外部プログラムの呼び出し、またはカスタム API またはサービスの呼び出しのいずれかによってカスタムの時間トリガー・トランザクションを実行します。

カスタマイズまたは拡張のビルドおよびデプロイ

必要なカスタマイズを実行した後、カスタマイズまたは拡張をビルドしてデプロイする必要があります。

1. カスタマイズまたは拡張を確認できるように、テスト環境でこれらをビルドしてデプロイします。
2. 準備ができたら、同じプロセスを繰り返して、実稼働環境でカスタマイズおよび拡張をビルドしてデプロイします。

このプロセスの指示については、以下のトピックについて説明しているカスタマイズ基本ガイドを参照してください。

- 標準リソース、データベース拡張、およびその他の拡張 (テンプレート、外部プログラム、および Java インターフェースなど) のビルドおよびデプロイ。
- エンタープライズ・レベルの拡張のビルドおよびデプロイ。

第 2 章 JSP コンソールをカスタマイズする前に

コンソール JSP ユーザー・インターフェースについて

プレゼンテーション・フレームワークを使用すると、Application ConsoleConsole 内で、動作を変更することなく、情報がレンダリング (または表示) される方法を変更できます。ユーザー・インターフェースをカスタマイズするには、ユーザー・インターフェースが画面をレンダリングし、データを渡す方法を決定するスクリプトの記述が必要になります。

このセクションでは、ユーザーのビジネス・ニーズに合わせてコンソール・ユーザー・インターフェースをカスタマイズする方法を説明します。各タスクは、Applications ManagerConfigurator のユーザー・インターフェースを通じた構成と修正したい画面の JSP ファイル内の HTML コードの編集の組み合わせによって実行されます。

注: HSDE (Execution Console Framework) 画面は、拡張できません。

Applications ManagerConfigurator の使用について詳しくは、「Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales: *構成ガイド (Configuration Guide)*」を参照してください。JSP ファイルで使用される機能について詳しくは、「コンソール JSP インターフェースの JSP 関数 (JSP Functions for the Console JSP Interface)」を参照してください。

重要: インターフェースをカスタマイズするときには、ご利用のアプリケーションの標準リソースをコピーし、コピーを変更します。ご利用のアプリケーションの標準リソースを変更しないでください。

以下のアプリケーションは、Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales に付属しています。

- Sterling Selling and Fulfillment Suite: Application ConsoleConsole - オーダー (order)、アイテム (item) の在庫 (inventory)、および返品を作成、追跡、および表示するための標準 UI。
- Applications ManagerConfigurator - Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales を構成するための UI。

コンソール JSP ユーザー・インターフェースのカスタマイズに関するガイドライン

ユーザー・インターフェースを拡張する前に、適切な拡張を実現できるように、ここで紹介するガイドラインをよく理解してください。

HSDE の画面は拡張できません。

JSP の比較

Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales の拡張性の一環として、現行リリースの JSP ファイルを拡張できます。

Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales の Service Pack またはメジャー・リリースが公開されて、場合によってはユーザー・インターフェースのホット・フィックスが公開された場合は、JSP の比較と調整を行う必要があります。 JSP 比較プロセス時には、以下のファイルを比較する必要があります。

- お使いのオリジナル製品バージョンに付属しているオリジナル JSP
- 同じオリジナル JSP 用の拡張された JSP
- 新しいリリースに付属している同じ JSP

調整プロセスを簡易化するために、IBM は、3 者間比較を実行できる差異分析ツールの購入を検討することをお勧めします。例えば、高価でないツールとしては Araxis が挙げられます。ただし IBM は、この差異分析ツールの販売も保証もしていません。ここでは、単に高価でないツールの例として紹介しているだけです。3 者間比較に対応しているツールについて独自に調査して、自身のニーズに合ったツールを選択してください。

円滑なアップグレードと簡単な保守のための準備

- 標準のデフォルト構成の一部として提供されているどのリソースのリソース定義も変更しないでください。Applications ManagerApplication Configurator を使用して必ずコピーを作成してから、そのコピーを変更してください。
- 当アプリケーションに付属しているどの JSP ファイル、JavaScript ファイル、およびアイコン JAR ファイルも変更しないでください。これらのファイルを変更した場合は、加えた変更内容がアップグレード時に失われる可能性があります。
- 新しいビューを作成する際は、保守の容易性および作成の容易性に関する問題を考慮してください。新しいビューや内部パネルなどを作成する際は、お使いのアプリケーションで提供されている JSP にリンクすることが可能です。しかし将来のリリースでは、IBM はこれらの JSP にさらなるリソースを追加する可能性があるため、ソフトウェアの変更を監視して、それらの変更に対応するために構成を更新する必要があります。

例えば、新しい内部パネルを作成して、この内部パネルをオーダー詳細ヘッダー JSP (/om/order/detail/order_detail_header.jsp) にリンクした場合は、この JSP ファイルでは、特定の API が呼び出されることが想定されます。具体的には、この JSP ファイルでは、出力テンプレート内の特定のフィールドを使用して getOrderDetails() API が呼び出されることが想定されるとともに、「運送会社とサービス」ドロップダウン・フィールドを対象にして getScacAndServiceList() API が呼び出されることが想定されます。

この内部パネルを使用する場合は、出力テンプレート内の必要なフィールドを使用してこれら 2 つの API を構成する必要があります。その後、IBM が将来のリリースでこの JSP ファイルにフィールドを追加した場合は、これらの変更内容に対応するために必要に応じて構成を変更してください。

使いやすさの考慮

新たに開発するビューはすべて、製品のビューと同じような外観と動作を実現する必要があるため、開発を開始する前に、デフォルト・ビューの動作内容を理解する必要があります。製品の基本的なルック・アンド・フィールについて詳しくは、『Screens in the JSP Console Interface』を参照してください。

開発環境の準備

カスタマイズ・プロセスを開始するには、変更に対応できるように開発環境を準備する必要があります。

参照資料の検索方法

プレゼンテーション・フレームワークは、いくつかの JSP ファイル、JavaScript ファイル、およびクラス・ファイルで構成されています。これらのファイルには、public として宣言されたいくつかの関数が含まれています。ただし、これらの関数のうち一部のみが公開されています。ユーザー・インターフェースの拡張時には、公開された関数のみを使用する必要があります。特定の関数が公開されているかどうかを確認するには、以下の場所を参照してください。

- Java クラス - Javadoc を参照してください。
- JavaScript - 『JavaScript Functions』を参照してください。
- JSP - 『JSP Functions for the Console JSP Interface』を参照してください。
- JSP タグ - 『JSP Tag Library for the Console JSP Interface』を参照してください。

第 3 章 JSP コンソールでの要求の処理とテーマ

JSP コンソールでの要求の処理とテーマについて

HTML フォームが関わるすべての Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales ユーザー・インターフェースにおいて、意図せずにブラウザーによって重複する要求が作成される可能性があります。いくつかのデータを更新する結果となる要求が意図せずに複数回送信された場合に問題が発生する可能性があります。例えば、ユーザーによってブラウザーが手動でリフレッシュされたときに、重複した要求が生成される場合があります。ユーザーが更新を 1 回のみ実行しようとしたとしても、同じ更新に対する複数の要求が生成される場合があります。

重複した要求によって生じた潜在的な問題を回避するために、アプリケーション・インフラストラクチャー・フレームワークはページ・トークンを使用して、オリジナルの要求のみが処理されることを検証します。ユーザーが Application Console にログインすると、開かれた画面にページ・トークンと呼ばれる固有 ID が割り当てられます。このページ・トークンは、ブラウザーによって更新要求が送信されるたびに検証されます。このトークンが以前に画面に割り当てられたものと一致しない場合、その要求は無視されます。この安全機能は、アプリケーション・インフラストラクチャー・フレームワークに組み込まれています。

集中テーマについて

ユーザーが正常にサインインを完了すると、標準アプリケーションの「ホーム・ページ」が開きます。ホーム・ページのルック・アンド・フィールは、ユーザーの ID と関連付けられているテーマによって決定されます。この テーマは、画面、ラベル、およびテーブル・ヘッダーなどのグラフィカル・ユーザー・インターフェース要素のレンダリングに使用されるフォントおよびカラーを決定します。

以下の標準テーマが提供されています。

- サファイア (デフォルト)
- 地球
- ヒスイ

テーマは、どのアプリケーションがそれを使用するかによって、CSS ファイルおよび XML ファイルを介して集中して決定されます。つまり、個々の画面に対して、ユーザー・インターフェース・コントロールのカラーおよびフォントをハードコーディングしてはいけないことを意味します。その代わり、本書で提供されている指示を使用してください。各テーマは、Application Console および Application ManagerConfigurator 全体で使用されます。理想的には、テーマを定義するときには、両方のアプリケーションに対して同じテーマを作成して一貫性のあるユーザー・エクスペリエンスを確実なものとします。

アプリケーション	必要なテーマ・ファイル
Application Console	• CSS - HTML ページ用 • XML - グラフ、チャート、およびマップで使用されるカラーおよびフォントの表示用 • _exui.xml - 実行 UI でのカスタマイズ・テーマの表示用
Applications ManagerConfigurator	• XML - 全体

いずれの標準テーマも変更しないでください。独自の CSS ファイルおよび XML ファイルを作成してください。各スタイルのコントロールを定義するために使用するクラスを決定するには、「コントロールとクラス」を参照してください。

異なるロケール (locale) 用にカスタマイズされたテーマが必要な場合は、各ロケール・コードに対して新しいファイルを作成する必要があります。これは、各テーマに対してユーザーがロケールを切り替えることができるためです。ロケール・コードの定義および使用について詳しくは、「JSP コンソール用サインイン画面のカスタマイズ (Customizing the Sign In Screen for the JSP Console」を参照してください。

新しいテーマの作成

このタスクについて

手順

1. *INSTALL_DIR*/repository/xapi/template/merged/resource/sapphire.xml テーマ・ファイルを *INSTALL_DIR*/extensions/global/template/resource/*theme*.xml にコピーします。

   /extensions/global/template/resource/ ディレクトリー構造が存在しない場合は、必要なディレクトリー構造を作成します。
2. 新しい XML ファイルを編集して、Applications ManagerConfigurator 内と棒グラフおよび円グラフ内で表示したい各カラー名およびフォント名エレメントのカラーおよびフォントで使用する値を定義します。

   注: 新しいテーマ XML 内に定義されるフォントは、使用しているシステム上に存在している必要があります。
3. *INSTALL_DIR*/repository/eardata/platform/war/css/sapphire.css テーマ・スタイル・シートを *INSTALL_DIR*/repository/eardata/platform/war/css/*theme*.css にコピーします。
4. (オプション) スキンを使用可能にしている場合のみ、*INSTALL_DIR*/repository/eardata/platform/war/skins/*skin-name*/css/*skin-name*_sapphire_mb.css テーマ・スタイル・シートを *INSTALL_DIR*/repository/eardata/platform/war/skins/*skin-name*/css/*skin-name_theme*_mb.css にコピーします。
5. 新しいスタイル・ファイルを編集して、新しいテーマ・ファイルに追加された新しいカラーおよびフォントを指定します。

6. *INSTALL_DIR*/repository/xapi/template/merged/resource/sapphire_exui.xml ファイルを *INSTALL_DIR*/extensions/global/template/resource/*theme*_exui.xml にコピーします。
7. 新しい XML ファイルを変更して、Applications ManagerConfigurator 内と棒グラフおよび円グラフ内で表示したい各カラー名およびフォント名エレメントのカラーおよびフォントで使用する値を定義します。

   注: フォントの変更は、フィールドのサイズまたはフォントを使用するその他の UI エレメントなどのその他の変更が必要になる場合があります。フォントの変更について詳しくは、「CSS テーマ・ファイル参照 (CSS Theme File Reference)」を参照してください。
8. Applications ManagerConfigurator のメニューから、テーマを構成します。

   新しいテーマを作成した後に、ユーザーは Application Console のユーザー・インターフェースのドロップダウン・リストから適切なテーマを選択できます。テーマの定義について詳しくは、「Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales: *ユーザー・ガイド (User Guide)*」を参照してください。

ユーザーまたは組織用のテーマの作成

このタスクについて

アプリケーションを使用すると、特定の組織またはユーザー用の UI スタイル (ロゴ、フォント、カラーなど) を定義できます。また、異なるテーマに基づいて、異なるイメージを表示することもできます。

ユーザー固有または組織固有の UI スタイルを定義するには、テーマ固有のイメージ jar を *INSTALL_DIR*/repository/eardata/*application*/war に追加します。このテーマ固有のイメージ jar もローカライズ可能です。例えば、*theme_name_locale*.jar のようになります。

サーバーからイメージを取り込んでいるときに、アプリケーションは最初にテーマ固有のイメージ jar 内のイメージを探します。テーマ固有の jar が存在しない場合、または jar が特定のイメージを持たない場合、フレームワークは、以下の順序でロケール固有の jar 内の特定のイメージを検索します。

注: テーマ固有のイメージ jar は、*INSTALL_DIR*/repository/eardata/*application*/extn/yantraiconsbe.jar を除くその他のイメージ jar をオーバーライドします。

手順

1. *INSTALL_DIR*/repository/eardata/*application*/extn/yantraiconsbe.jar
2. *INSTALL_DIR*/repository/eardata/config/war/yfscommon/yantraiconsbe.jar
3. *INSTALL_DIR*/repository/eardata/*application*/*module_name*/*module-specific_icons_jar*

   注: 既存のイメージの代わりにテーマ固有のイメージを表示するには、テーマ固有イメージ jar に追加した新しいテーマ固有イメージが既存のイメージと同じ名前およびパスを持っていることを確実にしてください。

第 4 章 JSP コンソールでの新規プラグイン・スキンの作成と使用可能化

スキンのサポート

当アプリケーションは、HTML UI でのスキンをサポートしています。スキンを使用すると、UI の見栄えを向上させることができます。スキンを使用するには、デフォルト・ファイルの代わりにスキン関連の UI ファイルをインクルードする必要があります。

新規スキンの作成

すべてのスキン関連ファイルをスキン名と共に *INSTALL_DIR*/repository/eardata/*application_name*/war/skins ディレクトリーに格納する必要があります。スキンは、すべてのテーマの固有のコンテナー、内部パネル、メニュー・バー、アンカー JSP および CSS ファイルを持っている必要があります。

スキンのディレクトリー構造

新しいスキンを作成するときには、以下のディレクトリー構造に従ってください。

- common — このディレクトリーは、スキン固有の menubar_anchor.jsp ファイルおよび menubar_dropdown_multilevel.jsp ファイルを含みます。

  menubar_dropdown_multilevel.jsp ファイルは、指定のメニュー項目に対してサブメニューを表示する場合に必要です。
- console/icons — このディレクトリーは、スキン固有のイメージを含みます。このディレクトリーには、ローカライズしたイメージを保管することもできます。

  注: /console/icons/ ディレクトリーおよび /yfcicons ディレクトリー内に提供されているデフォルトのイメージをオーバーライドする場合は、同じ名前の新しいイメージを /skins/*skin-name*/console/icons ディレクトリーまたは /skins/*skin-name*/yfciconsディレクトリーにコピーします。
- css — このディレクトリーには、ユーザーのスキン固有の CSS ファイルを含む必要があります。これらの CSS ファイルは、ローカライズ可能です。例えば、*skin_name*_sapphire_mb.css、*skin-name*_jade_mb.css、*skin-name*_earth_mb.css など。

- yfc — このディレクトリーには、スキン固有の container_mb.jsp ファイルおよび innerpanel_mb.jsp ファイルを含む必要があります。
- yfcicons — このディレクトリーには、ローカリゼーション・イメージを含む必要があります。

  /yfcicons ディレクトリー構造が存在しない場合は、作成してください。

スキンのプラグイン・ポイント

HTML UI の観点からは、当アプリケーションでは、スキン固有のファイルをインクルードする場所となる以下の 4 つのプラグイン・ポイントが用意されています。

- /yfc/container_mb.jsp

  スキン固有のコンテナー JSP ファイルは、検索リストの位置合わせを処理する必要があり、これにより、リスト JSP ファイル、または検索 JSP ファイル、または詳細 JSP ファイルと、ブレッドクラム JSP ファイルがインクルードされます。例えば、スキン固有のコンテナー JSP ファイルは次のような内容になります。

  ```
  <jsp:include page="/skins/skin-name/yfc/search_mb.jsp flush="true"/>
  <jsp:include page="/skins/skin-name/yfc/list_mb.jsp flush="true"/>
  <jsp:include page="/skins/skin-name/yfc/detail_mb.jsp" flush="true"/>
  ```

  スキン固有の search_mb.jsp ファイル、または list_mb.jsp ファイル、または yfc/detail_mb.jsp ファイルは、スキン関連の UI ファイルとフレームワーク側で用意されている機能ファイルをインクルードする必要があります。例えば、スキン固有の検索 JSP ファイルは次のような内容になります。

  ```
  <jsp:include page="/yfc/search_functional_pre_mb.jsp" flush="true"/>>
  <jsp:include page="/skin-name/search_ui_mb.jsp" flush="true"/>
  <jsp:include page="/yfc/search_functional_post_mb.jsp" flush="true"/>
  ```

  スキンが有効化されている場合は、フレームワーク側で用意されている /yfc/container_mb.jsp ファイルは、スキン固有の /skins/*skin-name*/yfc/container_mb.jsp ファイルをインクルードします。

- /yfc/innerpanel_mb.jsp

  スキン固有の内部パネル JSP ファイルは、スキン関連の UI ファイルと、フレームワーク側で用意されている機能ファイルをインクルードする必要があります。例えば、スキン固有の検索 JSP ファイルは次のような内容になります。

  ```
  <jsp:include page="/yfc/innerpanel_functional_pre_mb.jsp" flush="true"/>
  <jsp:include page="/skins/skin-name/innerpanel_ui_mb.jsp" flush="true"/>
  <jsp:include page="/yfc/innerpanel_functional_post_mb.jsp" flush="true"/>
  ```

  スキンが有効化されている場合は、フレームワーク側で用意されている /yfc/innerpanel_mb.jsp ファイルは、スキン固有の /skins/*skin-name*/yfc/innerpanel_mb.jsp ファイルをインクルードします。

- /common/menubar_mb.jsp

  スキン固有のメニュー・バー JSP ファイルは、メニュー・バーのレイアウトと外観を指定する必要があります。フレームワーク側で用意されているメニュー・バー・ファイルをこのファイルにインクルードすることもできます (必要な場合)。

注: スキン固有の menubar_anchor.jsp ファイルと menubar_dropdown_multilevel.jsp ファイルが使用されるのは、フレームワーク側で用意されている common/menubar_mb.jsp ファイルが使用されており、かつ menubar_anchor.jsp に対して明示的なコンテキスト・パラメーターが指定されていない場合のみです。

スキンが有効化されており、メニュー・バー・アンカー JSP ファイルに対して明示的なコンテキスト・パラメーターが指定されていない場合は、フレームワーク側で用意されている /common/menubar_mb.jsp ファイルは、スキン固有の /skins/*skin-name*/common/menubar_anchor.jsp ファイルをインクルードします。

- CSS ファイル

  スキン固有のテーマ CSS ファイルは、拡張性とローカライズをサポートするために、'../../../console/icons/*img-name*' または '../../../yfcicons/*img-name*' という形式でイメージを参照する必要があります。例:

```
.companyLogo{
float:right;
background: url(../../../console/icons/logo.jpg) no-repeat;
height:42px;
width:151px;
margin-right:0;
cursor:pointer;
  }
```

  スキンが有効化されている場合は、フレームワーク側で用意されている /yfc/container_mb.jsp ファイルは、スキン固有のローカライズ済み CSS ファイルをインクルードします (/skins/*skin-name*/css/*skin-name_theme-name*_mb.css ファイルなど)。

  注: デフォルトでは、元の CSS ファイル (/css/*theme-name*.css など) は、スキン固有の CSS ファイルの前にインクルードされます。

新しいスキンの使用可能化

このタスクについて

新しいスキンを使用可能にするには、'yfc-ui-skin' コンテキスト・パラメーターを *INSTALL_DIR*/repository/eardata/platform/descriptors/weblogic/WAR/WEB-INF/web.xml ファイルに追加する必要があります。新しいスキンの名前は、このパラメーターの値として指定する必要があります。例えば、web.xml ファイル内のエントリーは次のようになります。

```
<context-param>
<param-name>yfc-ui-skin</param-name>
<param-value>skin_name</param-value>
</context-param>
```

第 5 章 JSP コンソール用のサインイン画面のカスタマイズ

「サインイン (Sign In)」画面について

この「サインイン (Sign In)」画面は、アプリケーションを開始したときに最初に表示されるページです。この「サインイン (Sign In)」画面は、以下のファイルで構成されます。

- start.jsp — 新しいブラウザー・ウィンドウで login.jsp を開き、ブラウザー・ツールバーを除去します。
- login.jsp — logindetails.jsp を含みます。
- logindetails.jsp — 拡張可能です。

これらのファイルは、次の図で示されている順序で開きます。

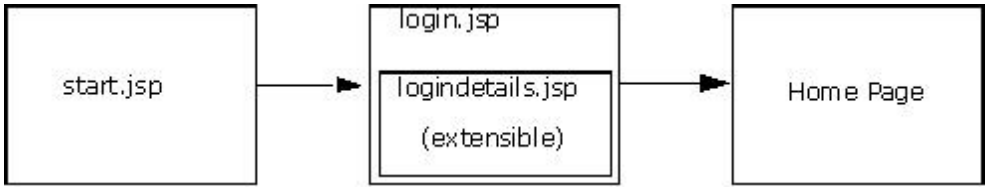


図 1. 「サインイン (Sign In)」画面ロジック

「サインイン (Sign In)」画面の主要な目的は、許可されたユーザーが Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales にログインできるようにし、独自の個人別設定セッションを確立できるようにすることです。ユーザーがサインインすると、以下のプロパティーがセッション用にセットアップされます。

- ロケール設定
- セキュリティー・アクセス制御
- 優先テーマ、組織などのユーザー・プロパティー

login.jsp のカスタマイズ

このタスクについて

login.jsp ページをカスタマイズして、「サインイン (Sign In)」画面内でユーザーからの追加の入力を受け入れるようにすることができます。デフォルトでは、「サインイン (Sign In)」画面は、ユーザー名とパスワードの 2 つのフィールドを受け入れます。必要な追加フィールドを追加した後に、新しく追加したフィールドを検証または認証することができます。

login.jsp をカスタマイズするには、以下の手順を実行します。

手順

1. ログイン中に追加のフィールドを受け入れるには、logininputs.jsp ファイルを編集して、必要な追加フィールドをユーザー名およびパスワード・フィールド定義の後に追加します。
2. 新しく追加したフィールドを検証または認証するには、使用しているカスタム PostAuthentication クラスで、IYFSPostAuthentication インターフェースの doPostAuthenticate() メソッドを実装します。doPostAuthenticate() メソッドは、HttpServletRequest オブジェクトをパラメーターとして使用し、ユーザーはそれを介して logininputs.jsp から追加フィールドにアクセスできます。エラーがある場合は、APIManager.XMLExceptionWrapper 例外にラップされ、このメソッドからスローされます。エラーがない場合は、doPostAuthenticate() メソッドは TRUE を返します。

   デフォルトでは、HTML UI フレームワークはユーザー名とパスワードを認証します ユーザー名またはパスワードが間違っている場合は、「ログインできませんでした (Login Failed)」メッセージが表示されます。同様に、追加フィールドの認証が失敗した場合、「ログインできませんでした (Login Failed)」メッセージが表示されます。ただし、ポスト認証クラスが使用不可である場合、「アプリケーション・エラー、管理者にお問い合わせください (Application error, please contact administrator)」が表示されます。
3. アプリケーションの config.xml ファイルに、<Context-Params> タグを追加し、<Context-Params> タグの下に <Context-Param> タグを追加して、PostAuthentication クラスのパラメーター名を定義します。例:

```
<WebComponents>
     <ContextParams>
          <Context-Param>
            <Param-Name> PostAuthenticationClass</Param-Name>
            <Param-Value>com.yantra.platformdemo.ui.backend.ClientPostAuthClass
            </Param-Value>
          </Context-Param>
      </ContextParams>
    </WebComponents>
```

ロケールのセットアップ

セットアップするロケールに応じて、画面上のリテラルを表示するための言語が決定されます。start.jsp ファイルは、「サインイン」画面をデフォルトの言語で表示します。ユーザーがログインした後に、画面に表示される言語は、現在のロケールで指定された設定に基づいて決定されます。

多国籍組織では、複数言語のローカライズが求められることがあります。例えば、標準のインストール内容はドイツ語で、ユーザーのロケールがフランス語であることが考えられます。このような場合は、start.jsp ファイルと login.jsp ファイルはドイツ語で表示されて、ユーザーのホーム・ページはフランス語で表示されます。login.jsp ファイルに表示される言語は、使用されているリソース・バンドルによって決定されます。

ユーザーのセッション全体は、プロファイルで指定された言語で表示されます。ユーザーが異なるロケールを選択するかセッションが終了した場合は、表示される言語は変更されます。

ユーザーが異なるロケールを選択した場合は、Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales は、その新しいロケールと関連付けられたすべてのリテラルを動的に変更します。

ユーザーがログアウトするか接続がタイムアウトになると、セッションは終了します。セッションが終了すると、「サインイン」画面には標準の言語が表示されます。

特定のロケールに対するログインの構成

このタスクについて

ご利用のユーザー・コミュニティーがマルチリンガルの場合、IBM は、ログイン・ページをすべての言語で表示することを推奨します。これにより、ユーザーが「サインイン (Sign In)」画面から適切な言語を選択できるようになります。

国際化した 「サインイン (Sign In)」画面を表示するには、以下の手順を実行します。

手順

1. 表示する各言語に対応するハイパーリンクを含む企業 HTML ページを作成します。
2. 各ハイパーリンクについて、<a href> タグを挿入し、以下の構文を使用してロケール固有のページにリンクします。

   /smcfs*application_name*/console/start.jsp?LocaleCode=*locale_code*

   locale_code 値は、Applications ManagerConfigurator 内の「ロケール・コード」フィールドに指定されているエントリーと一致している必要があります。「ロケール・コード」フィールドおよびその提案されている構文について詳しくは、「Applications ManagerConfigurator」、「Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales:*構成ガイド (Configuration Guide)*」を参照してください。

   例えば、「サインイン (Sign In)」画面をカナダ・フランス語で開く場合、以下の構文を使用します。

   /smcfs*application_name*/console/start.jsp?LocaleCode=fr_CA

   これにより、指定したロケールで「サインイン (Sign In)」画面が表示されます。ログインした後に、アプリケーションがユーザーのデフォルト・ロケールで開きます。

外部アプリケーションからのユーザー・サインインの構成

外部アプリケーションとSterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales を統合するときには、ユーザーが外部アプリケーションから Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales に自動的にログインできるようにすることが必要です。2 つのアプリケーションを統合するには、Sterling Business CenterSterling Selling and Fulfillment

FoundationSterling Field Sales に自動的に接続する外部アプリケーションのポータルからのリンクを構成する必要があります。

2 つのアプリケーションを統合するときには、以下のシナリオを考慮してください。

- ユーザーのログインおよびユーザーのホーム・ページを開く
- ユーザーのログインおよび特定のビューを開く

外部アプリケーションとSterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales を統合するには、以下の手順を実行します。

要件に基づいて、以下の URL のいずれか 1 つを外部アプリケーションのポータル内に挿入できます。

- Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales を起動して、ユーザーのホーム・ページを開く場合、アプリケーション・ユーザーとしてログインし、以下の構文を使用してユーザーのホーム・ページを開きます。

  ```
  /smcfs application_name /console/start.jsp?UserId= LoginID
  &Password= PassPhrase
  ```

- Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales を起動して、特定のビューを開く場合、アプリケーション・ユーザーとしてログインし、以下の構文を使用してビューを開きます (デフォルトのオーダー検索ビューを開く場合を示します)。

  ```
  /smcfs application_name /console/start.jsp?UserId= LoginID
  &Password= PassPhrase &Redirect=/console/order.search
  ```

いずれの場合も、ログインに失敗した場合は、ブラウザーで Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales の「サインイン (Sign In)」画面が開き、ユーザーにログイン ID とパスワードを入力するようプロンプトが出されます。

外部認証のサポート

ユーザーがドメイン・パスワードを使用して Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales に透過的にログインする必要がある場合は、当アプリケーションはサード・パーティーのシングル・サインオン・アプリケーションをサポートします。現在の環境でシングル・サインオン・アプリケーションがサポートされていない場合は、当アプリケーションは外部認証と内部認証をサポートします (デフォルトで)。

外部認証の使用について詳しくは、「*Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales: インストール・ガイド*」を参照してください。シングル・サインオンのプログラミング情報については、Javadoc を参照してください。

第 6 章 JSP コンソール内の企業ロゴのカスタマイズ

企業ロゴについて

標準アプリケーションは、Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales 商標ロゴをアプリケーション全体に伝達します。画面に表示される商標ロゴを以下の場所で変更できます。

- サインイン (Sign In) 画面
- メニュー・バー
- バージョン情報ボックス (About Box)

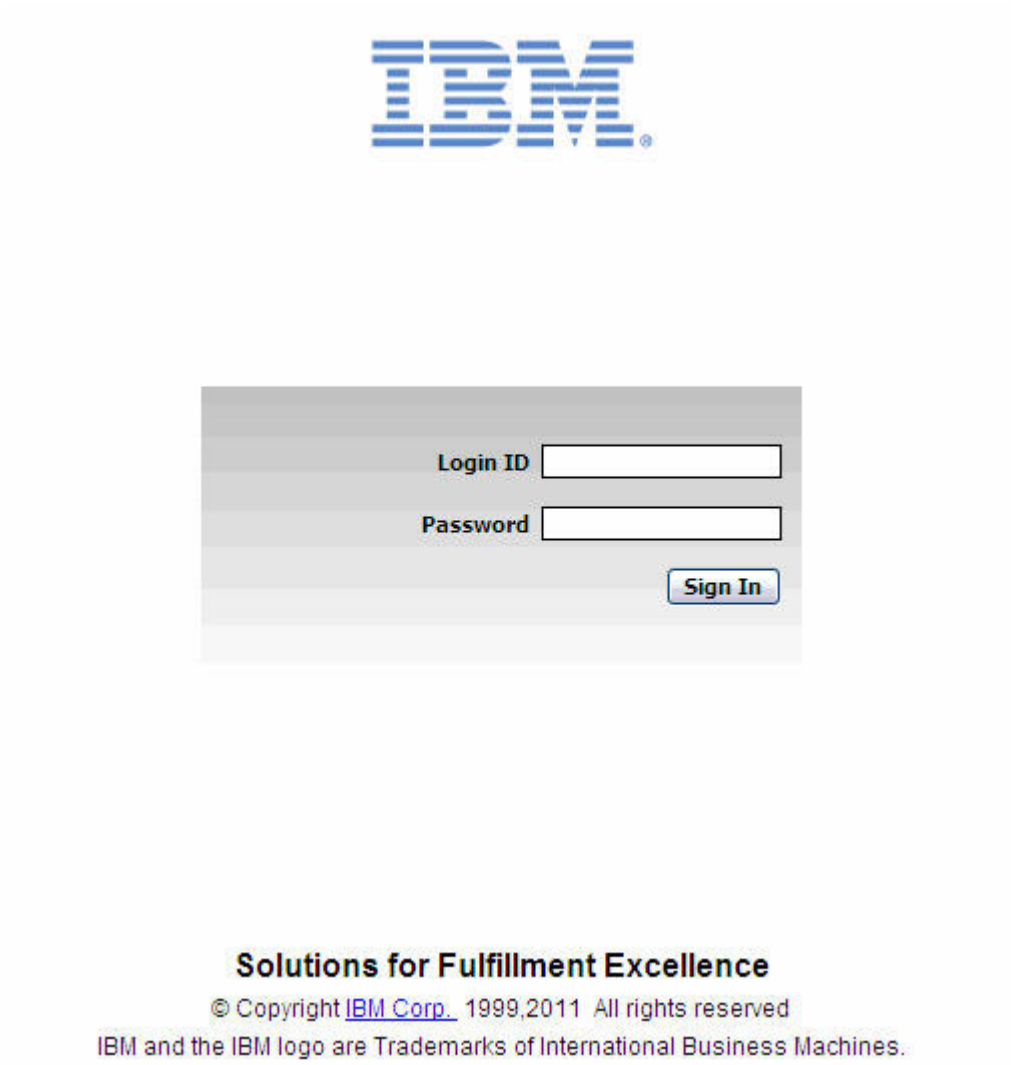
すべて異なるサイズのイメージを使用しているため、各インスタンスで固有のイメージを使用してください。

Application Console の場合は、JAR ファイル内に存在するため、イメージの相対URLをパスを含めて指定する必要があります。例えば、コンソール内で使用されるデフォルトのアイコンは、yantraiconsbe.jar ファイル内にあります。アイコンの多くの JAR ファイル内のパスは console/icons です。したがって、コンソール・リソースに対してリソース階層ツリー内で指定されているイメージは、smcfs*application_name*/console/icons/consoleresource.jpg となります。

「サインイン」画面上のロゴのカスタマイズ

このタスクについて

「サインイン」画面に表示される企業ロゴはカスタマイズできます。



「サインイン」画面上のロゴをカスタマイズするには、次の手順を実行します。

手順

1. *INSTALL_DIR*/repository/eardata/platform/war/console/logindetails.jsp ファイルを *INSTALL_DIR*/extensions/global/webpages/console/logindetails.jsp ファイルにコピーします。

   /global/webpages/console/ ディレクトリー構造が存在しない場合は、必要なディレクトリー構造を作成します。
2. 上記のコピーで得られた新しい logindetails.jsp ファイルを開いて、次の操作を実行します。
   a. YANTRA_LOGIN_RIGHT を検索して、IBM ロゴを参照している <img> タグを見つけます。 <img> タグに変更を加えて、このタグが自社のロゴを指し示すようにします。

      例えば、ロゴ・ファイルが MyLogo.jpg の場合は、次のようにします。

```
<TR>
          <TD align=center valign=bottom>
            <img src="<%=request.getContextPath()%>/extn/console/icons/MyLogo.jpg"/>
          </TD>
        </TR>
```

b. YANTRA_ICON を検索して、IBM アイコンを参照している <link> タグを見つけます。 <link> タグに変更を加えて、このタグが自社のアイコンを指し示すようにします。

例えば、アイコン・ファイルが MyIcon.jpg の場合は、次のようにします。

```
<head>
          <link REL="SHORTCUT ICON"
            HREF="<%=request.getContextPath()%>/extn/console/icons/MyIcon.jpg"/>
          <title><yfc:i18n>Yantra_7x</yfc:i18n></title>
        </head>
```

3. イメージ・ファイルを *INSTALL_DIR*/extensions/global/webpages/console/icons/ ディレクトリーにコピーします。

/global/webpages/console/icons/ ディレクトリー構造が存在しない場合は、必要なディレクトリー構造を作成します。

4. *INSTALL_DIR*/repository/eardata/smcfs*application_name*/extn/web.xml.sample ファイルの名前を、*INSTALL_DIR*/repository/eardata/smcfs*application_name*/extn/web.xml という名前に変更します。手順 2 で指定した両方のイメージ URI のエントリーを追加します。例えば、web.xml のエントリーは次のようになります。

```
<context-param>
   <param-name>bypass.uri.extn1</param-name>
   <param-value>/console/icons/MyLogo.jpg</param-value>
</context-param>
<context-param>
   <param-name>bypass.uri.extn2</param-name>
   <param-value>/console/icons/MyIcon.jpg</param-value>
</context-param>
```

メニュー・バー上のロゴのカスタマイズ

このタスクについて

メニュー・バーは、ポップアップ・ダイアログ・ボックスではないすべての画面に表示されます。

ユーザーが Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales にログインすると、getMenuHierarchyForUser() API が呼び出されて、その出力がセッション・オブジェクトに保管されます。これにより、各画面のメニューを生成するためのオーバーヘッドが低減されて、メニュー・バーをユーザー・レベルで構成できるようになります。

メニュー・バー上のロゴをカスタマイズするための手順は次のとおりです。

スキンをまだ有効にしていない場合は、以下の手順を実行してください。

手順

1. *INSTALL_DIR*/repository/eardata/platform/war/common/menubar.jsp ファイルを *INSTALL_DIR*/extensions/global/webpages/common/menubar.jsp にコピーします。

   /global/webpages/common/ ディレクトリー構造が存在しない場合は、必要なディレクトリー構造を作成します。
2. 上記のコピーで得られた新しい menubar.jsp ファイルを開いて、YANTRA_LOGO を検索して、IBM ロゴを参照している <img> タグを見つけます。<img> タグを編集して、パスを ="../console/icons/ のように指定して、このタグが自社ロゴを指し示すようにします (例えば、ロゴ・ファイルが MyLogo.jpg である場合は ../console/icons/MyLogo.jpg と指定します)。
3. イメージ・ファイルを *INSTALL_DIR*/extensions/global/webpages/icons/console/icons/ ディレクトリーにコピーします。

   /global/webpages/icons/console/icons/ ディレクトリー構造が存在しない場合は、必要なディレクトリー構造を作成します。
4. *INSTALL_DIR*/extensions/global/webpages/icons/ ディレクトリーから、console ディレクトリー全体を yantraiconsbe.jar ファイルとしてアーカイブします。

スキンを使用したメニュー・バー上のロゴのカスタマイズ

このタスクについて

スキンを使用可能にした場合、以下のタスクを実行します。

手順

1. *INSTALL_DIR*/repository/eardata/platform/war/skins/*skin-name*/common/menubar_anchor.jsp ファイルを *INSTALL_DIR*/extensions/global/webpages/skins/*skin-name*/common/menubar_anchor.jsp にコピーします。

   /global/webpages/common/ ディレクトリー構造が存在しない場合は、必要なディレクトリー構造を作成します。
2. 新しい menubar_anchor.jsp ファイルを開いて、以下の手順を実行します。
   a. logo.jpg を検索して、IBM ロゴを参照する <img> タグを検出します。<img> タグを変更して、企業ロゴをポイントするようにします。パスを ="../console/icons/ (例えば、MyLogo.jpg の場合、../console/icons/MyLogo.jpg を使用) として指定します。
   b. 要件に応じて logo.jpg <img> タグに定義されている CSS クラスを変更します。
3. イメージ・ファイルを *INSTALL_DIR*/extensions/global/webpages/icons/console/icons/ ディレクトリーにコピーします。

   /global/webpages/icons/console/icons/ ディレクトリー構造が存在しない場合は、必要なディレクトリー構造を作成します。
4. *INSTALL_DIR*/extensions/global/webpages/icons/ ディレクトリーから、コンソール・ディレクトリー全体を yantraiconsbe.jar ファイルにアーカイブします。

バージョン情報ボックス (About Box) 内のロゴのカスタマイズ

このバージョン情報ボックス (About Box) は、アプリケーションのバージョン番号を示します。Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales は、表示されているメニュー・バーに表示されているロゴをクリックすることによって、ユーザーがアクセスできる標準バージョン情報ボックス (About Box) を出荷しています。. 次の図は、Application Console のバージョン情報ボックス (About Box) を示しています。

バージョン情報ボックス (About Box) を表示するときに、アプリケーションは事前定義されたロジックに基づいてバージョン番号を読み取ります。ロゴをカスタマイズするときには、バージョン番号を表示するロジックを保持することを強くお勧めします。

Packaged Composite Applications (PCA) を IBM から購入した場合、その製品のバージョン番号も、バージョン情報ボックス (About Box) に表示されます。

記号ブランド情報をバージョン情報ボックス (About Box) に追加する場合、表示されるロゴをカスタマイズできます。

スキンが有効化されていない場合の「バージョン情報 (About)」ボックス上のロゴのカスタマイズ

このタスクについて

「バージョン情報 (About)」ボックス上のロゴをカスタマイズするには、次の手順を実行します。

スキンを有効にしていない場合は、次の操作を実行します。

手順

1. *INSTALL_DIR*/repository/eardata/platform/war/console/about.jsp ファイルを *INSTALL_DIR*/extensions/global/webpages/console/about.jsp にコピーします。
2. 上記のコピーで得られた新しい about.jsp ページで独自の HTML を記述します。
3. menubar.jsp ファイルに変更を加えて、<img> タグの onclick イベントによってこの新しい about.jsp ファイルが呼び出されるようにします。

   ヒント: 「バージョン情報 (About)」ボックス (および他のポップアップ・ウィンドウ) の外観と動作を当アプリケーションの他の部分と一貫させるには、ポップアップ・ウィンドウを表示するための window.showModaldialog() 関数を使用します。

スキンが有効化されている場合の「バージョン情報 (About)」ボックス上のロゴのカスタマイズ

このタスクについて

スキンを有効にしている場合は、次の操作を実行します。

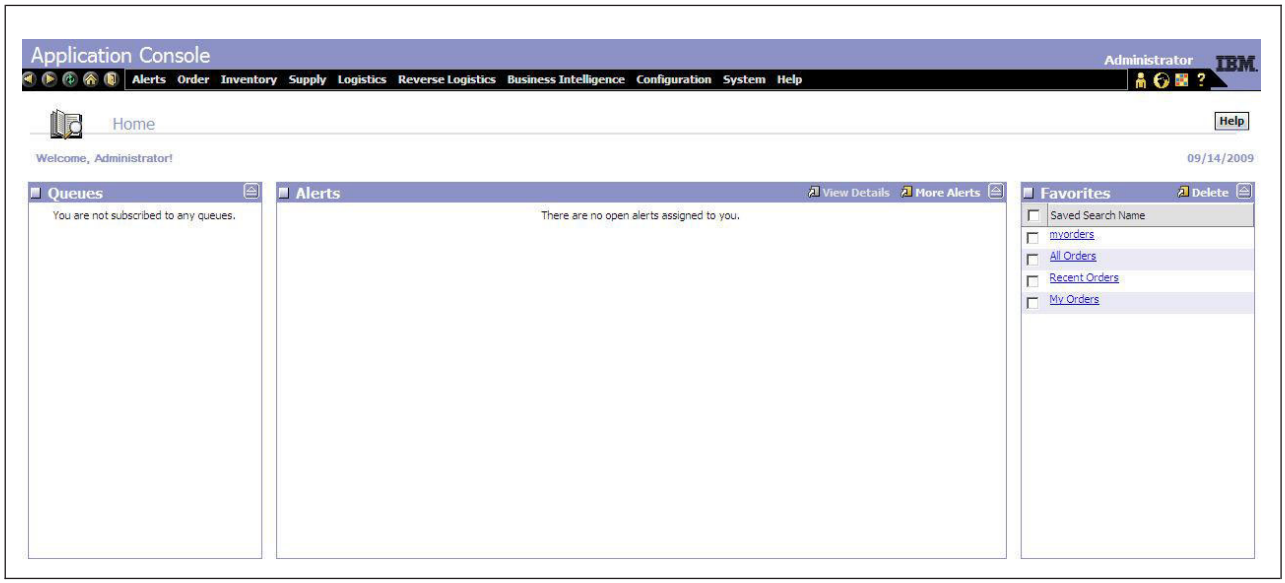
手順

1. *INSTALL_DIR*/repository/eardata/platform/war/console/about_mb.jsp ファイルを *INSTALL_DIR*/extensions/global/webpages/console/about_mb.jsp にコピーします。
2. 上記のコピーで得られた新しい about_mb.jsp ページで独自の HTML を記述します。
3. 「バージョン情報 (About)」ボックスのポップアップ・ウィンドウのサイズと位置を変更するには、menubar_anchor.jsp ファイルに変更を加えて、popupAboutBox_mb() 関数を探します。height、width、left、top などのプロパティーを希望どおりに変更します。

第 7 章 JSP コンソール内の画面

JSP コンソールの画面について

Application Console では、画面はそれが含む内部パネルを画面が表示する方法を定義するコンテナー・ページで構成されます。Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales 内では、それらと関連付けられているアンカー・ページ、内部パネル、およびロジックは、ビューと呼ばれます。次の図は、標準ホーム・ページを示します。これは検索ビューで使用されるコンポーネントを表示しています。



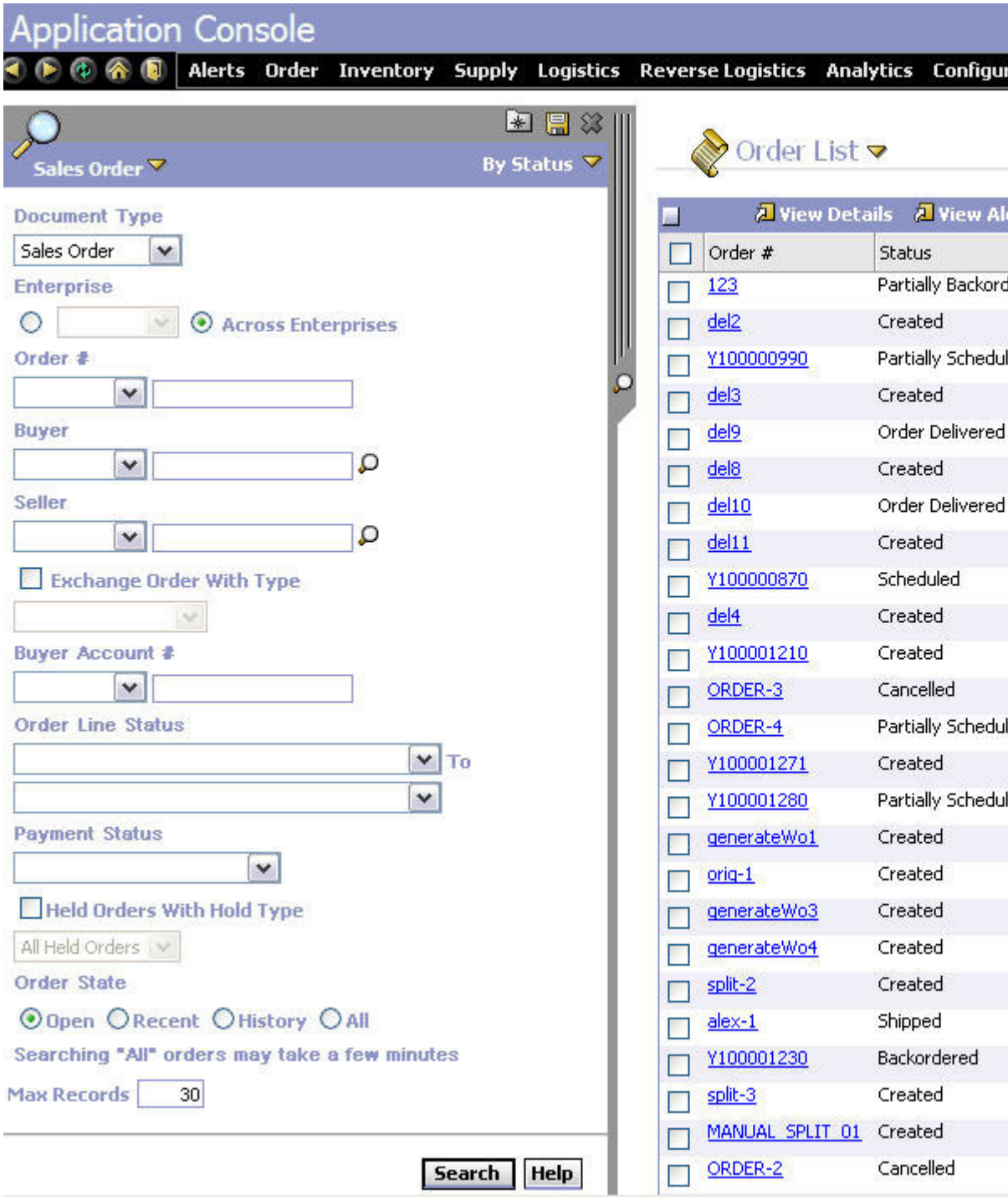
Application Console では、次のタイプのビューが利用可能です。

- 検索ビュー
- リスト・ビュー
- 詳細ビュー

これらのビューの詳細説明については、「拡張可能な画面」を参照してください。

JSP コンソール内の画面レイアウト

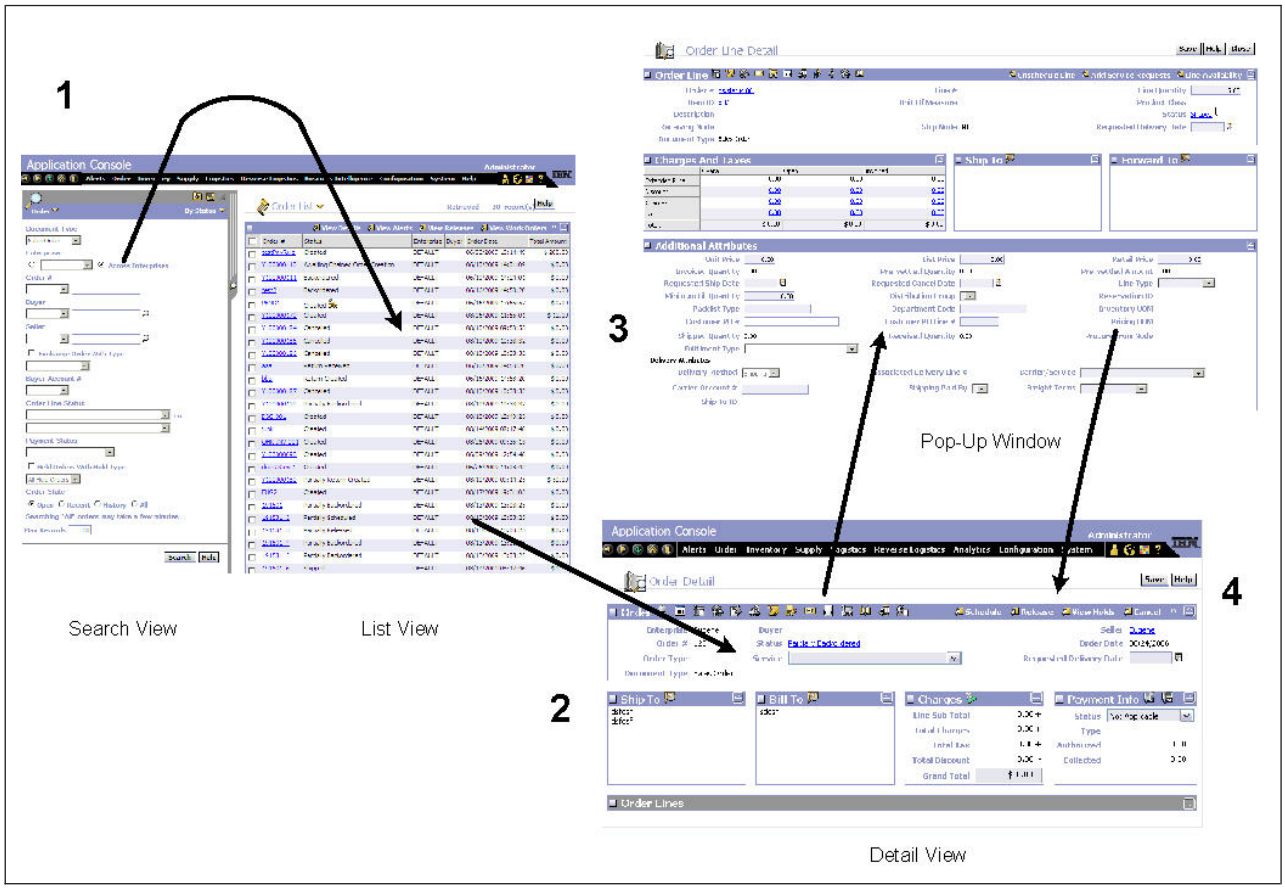
すべての画面は、同じ全体的な構成パターンに従っています。一般に、アプリケーションの画面は、メニュー・バーが一番上に配置されて、その下に検索ビューとリスト・ビューが隣り合わせに配置されるというレイアウトになっています。次の図では、一般的な画面構成を示しています。



注: 標準的な画面レイアウト構成からの逸脱は (メニューを画面の左側に配置するなど)、サポートされていません。

JSP コンソール内の画面ナビゲーション

Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Salesの画面ナビゲーション動作は、一貫した標準パターンに従っています。次の図では、ある内部パネルから次の内部パネルへのナビゲーション・ロジックの流れを示しています。



- 検索ビューは画面の左側に表示されます。
- リスト・ビューは同じ画面の右側に表示されて、検索結果を表示しています。ここから、詳細画面に移動できます。
- 詳細ビューは、検索ビューおよびリスト・ビューと置き換わる形で、画面全体に表示されます。詳細ビュー内のリンクやアイコンをクリックすると、ポップアップ・ウィンドウが開きます。
- そのポップアップ・ウィンドウを閉じると、直前のビューにフォーカスが戻ります。

画面ナビゲーションのカスタマイズ

デフォルトでは、ハイパーリンクを提供するために、プレゼンテーション・フレームワークによって機能が公開されます。これらの機能のリストについては、「コンソール JSP インターフェースの JSP 関数 (JSP Functions for the Console JSP Interface)」を参照してください。一般的に、リンクは十分に構成可能であり、コードを変更することなく、リンクがポイントするビューを変更できます。

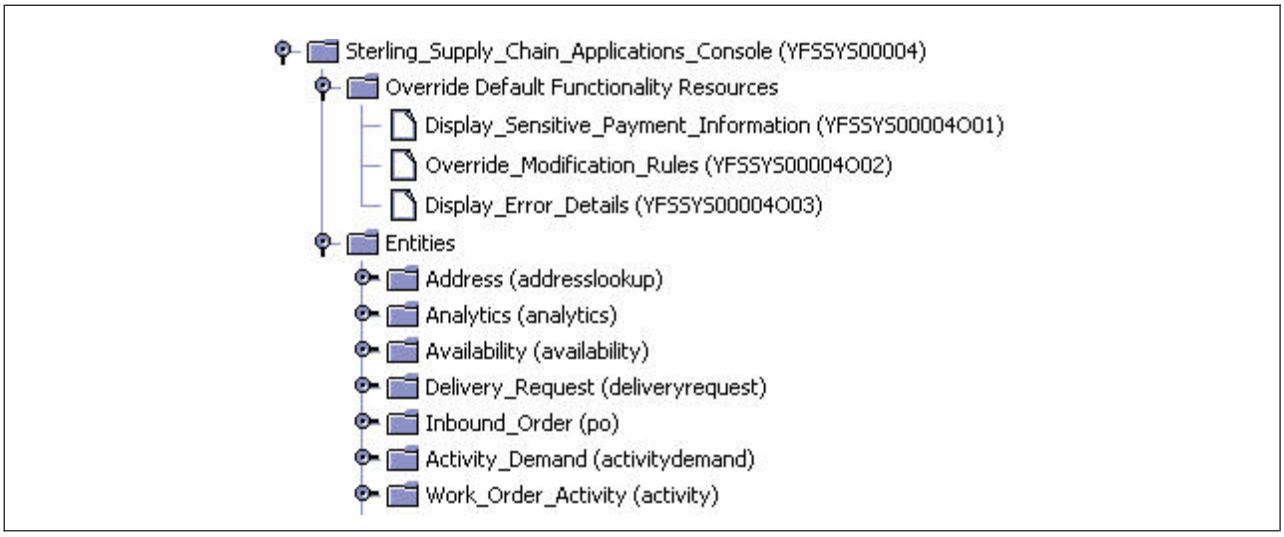
拡張可能な画面

内部パネルに含まれている画面にはすべて、それらに関連付けられているビジネスおよびプログラマチック・ロジックがあります。すべてのタイプのエンティティーの画面を拡張できます。エンティティーは、特定のビジネス・プラクティスに関連する UI 表示とプログラム・ロジックの関連付けられたグループです。一般に、各エンティティーには対応するコンソールがあります。Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales は、ビューを作成することができる、以下のハイレベルなビジネス・エンティティーを含みます。

- アラート・コンソール
- 在庫の調整コンソール
- オーダーの作成コンソール
- ピック・リストの作成コンソール
- 購入オーダーの作成コンソール
- 返品作成コンソール
- テンプレート作成コンソール
- 例外コンソール
- 入荷コンソール
- 在庫コンソール
- 積荷目録コンソール
- オーダー・コンソール
- 出荷コンソール
- 計画コンソール
- 購入オーダー・コンソール
- 返品コンソール
- 「システム管理コンソール」
- テンプレート・コンソール

これらのビジネス関連エンティティーは、より微小な詳細に分類することができます。例えば、オーダー管理は、注文、出荷 (shipment)、およびコンテナー (container) などのより微小なエンティティーに分類できます。独自のエンティティーを作成することもできます。

リソース階層ツリーを使用すると、ビジネス・ニーズに適したエンティティーをカスタマイズおよび作成することができます。次の図は、リソース階層ツリー内でエンティティーがどのように表示されるかを示しています。階層オーダーは、ナビゲーションのデフォルト・オーダーに基づいています。



各エンティティーには、デフォルトの検索ビュー、リスト・ビュー、および詳細ビューがあります。デフォルトのビューは、リソース階層ツリー内のこれらのビューの順序付けによって決定されます。

例えば、オーダー・エンティティーに 4 つの検索ビューがある場合、デフォルトの検索ビューは、4 つの検索ビューの中で最も低いリソース・シーケンス番号を持つものと判別されます。

各エンティティー・リソース下で、1 つの詳細 API と 1 つのリスト API を構成できます。この構成された詳細 API は、エンティティーの詳細ビューが開かれると自動的に呼び出されます。リスト API は、エンティティーのリスト・ビューが開かれると呼び出されます。

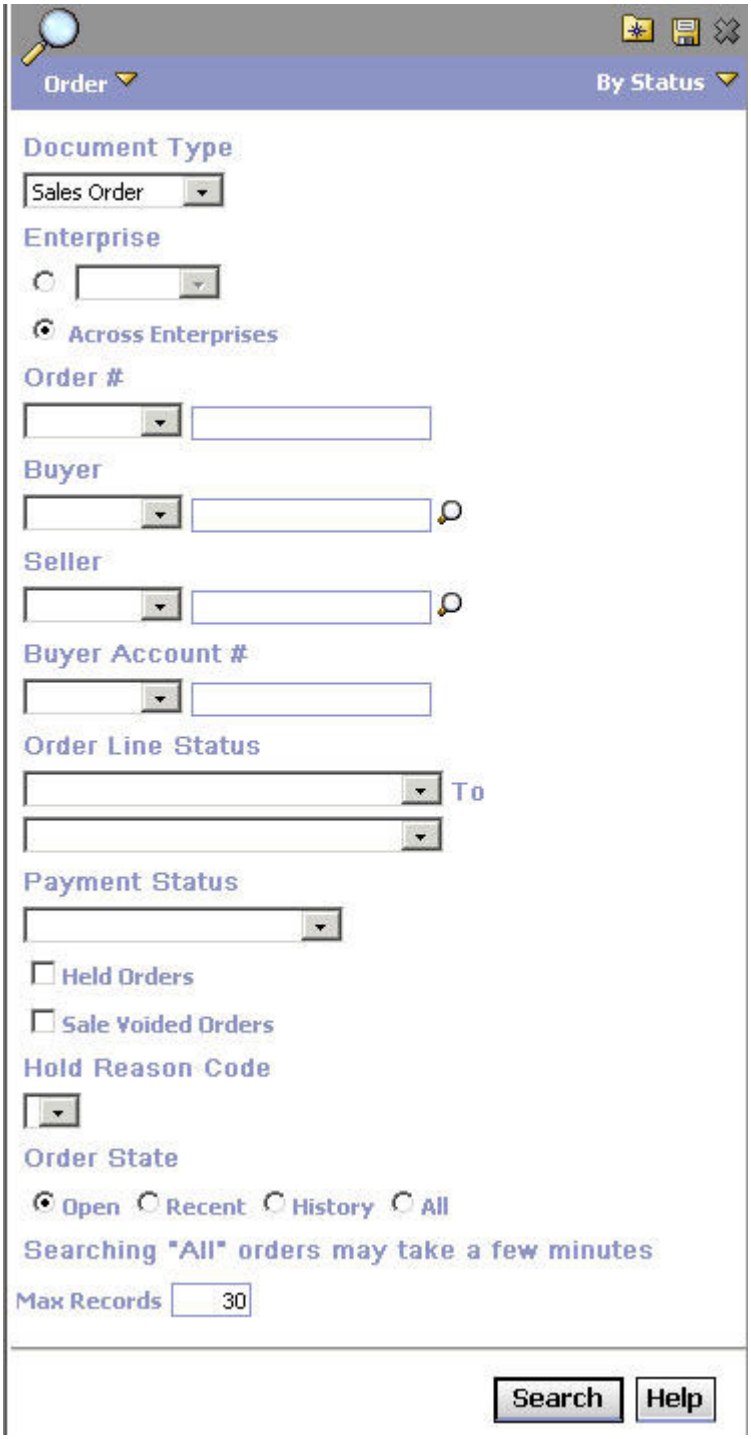
リソース構成画面で「デフォルト API の無視 (Ignore Default API)」パラメーターを選択することで、特定のビューに対してこのデフォルトの API が呼び出されないようにすることができます。

リソース・シーケンスについて詳しくは、「Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales: *構成ガイド (Configuration Guide) アプリケーション・プラットフォーム構成ガイド*」を参照してください。

注: カスタム・ビューには、オンライン・ヘルプは使用できません。カスタム・ビューが作成されると、ヘルプが使用できないことをユーザーに通知するツール・ヒントと共に「ヘルプ」ボタンが表示されます。カスタム画面のヘルプは、画面のリソースを構成しているときに SuppressHelp オプションを選択することによって抑制することができます。

JSP コンソール内の検索ビュー画面

検索ビューには、一連のクエリー条件が表示されています。次の図では、標準の検索ビューを示しています。



検索ビューは、以下の要素のうち 1 つ以上で構成されています。

- JSP ページ - 画面を表示します。JSP について詳しくは、『JSP Functions for the Console JSP Interface』を参照してください。
- API - ドロップダウン・メニューの内容を構成します。各 API について、複数の入力パラメーターと 1 つの出力テンプレートを指定できます。アプリケーショ

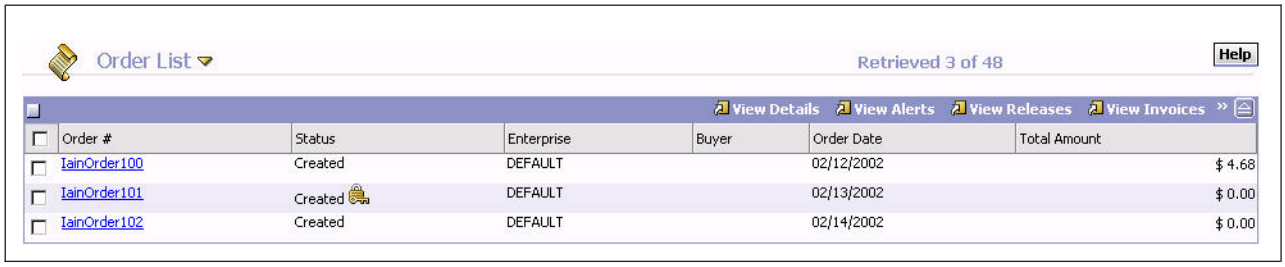
ン・コンソールで使用される API リソースの構成について詳しくは、Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales を参照してください。

JSP コンソール内のリスト・ビュー画面

検索結果は、リスト・ビューに表示されます。リスト・ビューには、標準リスト・ビューと拡張リスト・ビューの 2 種類があります。

標準リスト・ビュー

次の図では、標準リスト・ビューを示しています。



標準リスト・ビューは、見出し領域 (ビューの切り替えを可能にします) と、リスト本体および各種のアクションが含まれたアクション・バーで構成されています。デフォルトでは、リスト本体は画面に 30 件のレコードを表示するように設定されています。ユーザーは、より多くの結果を画面に表示したい場合は、最大で 200 件のレコードを表示できます。すべてのユーザーが 200 件を超えるレコードを表示できるようにするには、*INSTALL_DIR*/properties/customer_overrides.properties ファイルを編集して、新しい最大数を設定できます (手順については『コンソール JSP 内のリスト・ビューの最大レコード数』を参照してください)。

リスト・ビューは、以下の要素のうち 1 つ以上で構成されています。

- JSP ページ - 本体を表示します。これは、プレゼンテーション・フレームワークに属していない唯一のビュー要素です。
- 追加の API - そのリスト・ビューのために呼び出されます。
- アクション - 『Actions from List and Detail Views in the JSP Console』を参照してください。

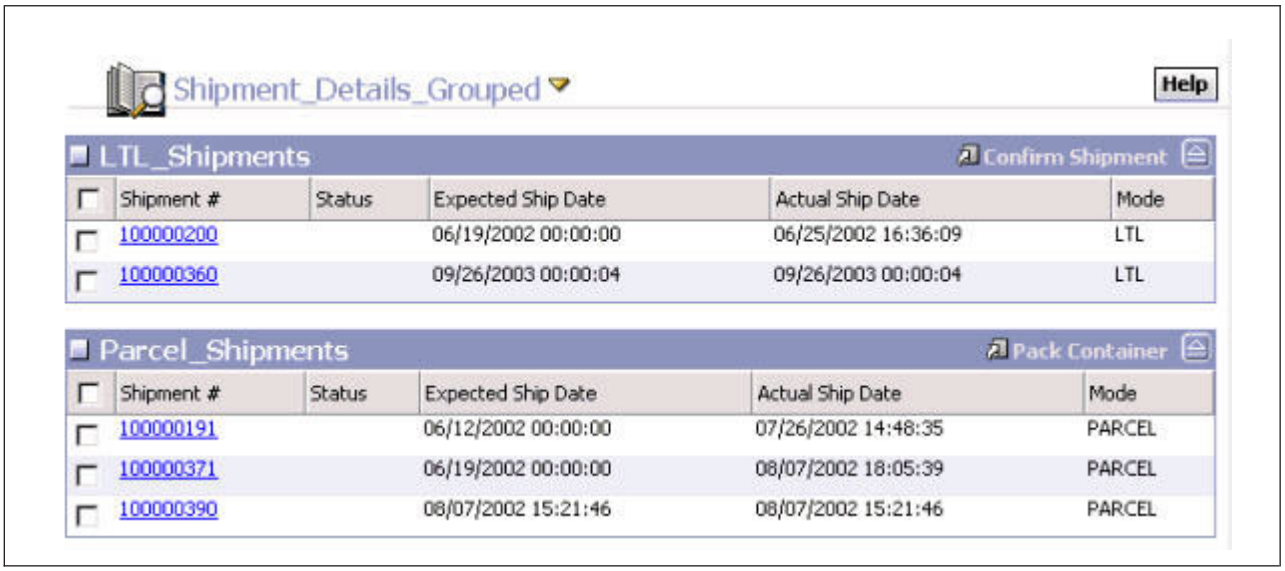
エンティティー内では、すべての従属リスト・ビューではエンティティー・レベルで定義されたのと同じリスト API が使用されます。エンティティー・レベルのリスト API に加えて、それぞれのリスト・ビューに対して追加の API を定義できます。デフォルトのリスト API を呼び出さないリスト・ビューを構成することもできます。このような場合は、そのリスト・ビュー用に構成されたテンプレートは使用されません (リスト API 自体が呼び出されないため)。

拡張リスト・ビュー

追加の情報をリストに表示する必要がある場合は、拡張リスト・ビューを作成できます。ユーザーにとっては、拡張リスト・ビューは標準リスト・ビューとほぼ同じ

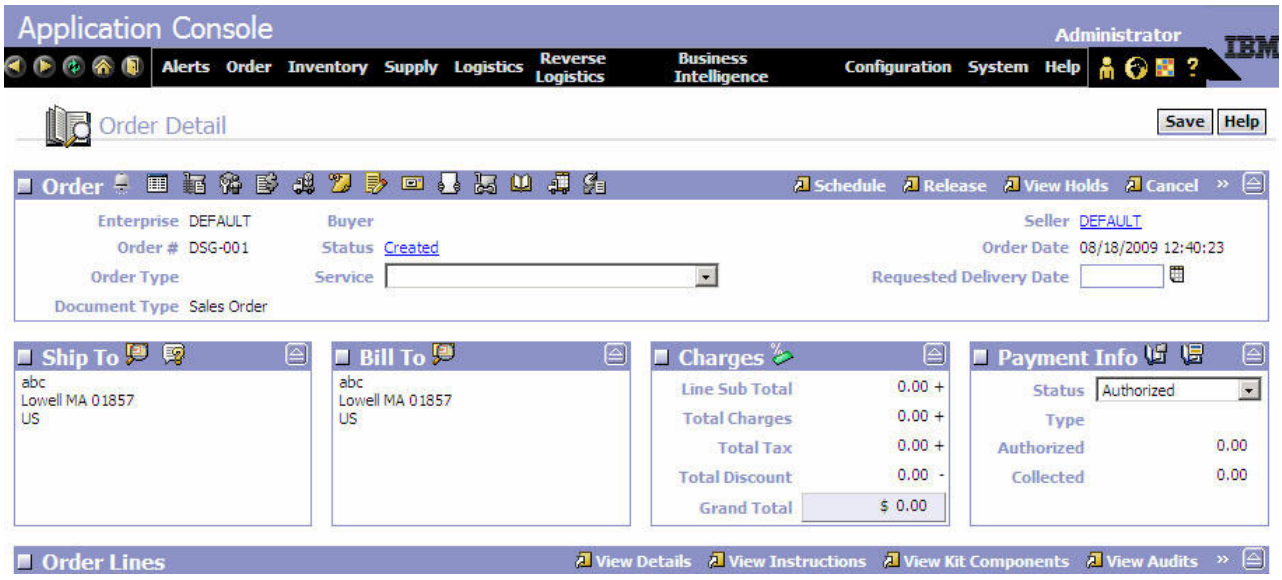
ように見えます。しかし、拡張リスト・ビューは実際には詳細ビュー・リソースとして定義されています。このため、拡張リスト・ビューは標準詳細ビューと同じ機能をすべて持つことが可能になります。

次の図では、拡張リスト・ビューを示しています。ユーザーにとっては、拡張リスト・ビューは標準リスト・ビューとほぼ同じように見えますが、拡張リスト・ビューは実際には詳細ビューであり、複数の内部パネルで構成されており、保存機能または更新機能を備えており、カスタム・ハイパーリンクをサポートしていますが、「1/N を表示中」というコンポーネントはありません。



JSP コンソールの詳細ビュー画面

詳細ビューは、エンティティーに関する、より特定の情報を表示します。次の図は、標準詳細ビューを示しています。



詳細ビューは、1 つ以上の内部パネルで構成されます。アンカー・ページは、これらの内部パネルのレイアウトを定義します。例えば、オーダーの詳細アンカー・ページは、オーダーの詳細に関連するすべての内部パネルを含み、それらを横にレイアウトする方法を定義します。

アンカー・ページはオプションです。アンカー・ページが指定されていない場合、詳細ビュー内の内部パネルは順次自動的に縦にレイアウトされます。

各内部パネルは、複数のフィールドと新しい詳細ウィンドウを開くことを可能にするゼロまたは 1 つ以上の語 (「アクション」)、およびポップアップ・ウィンドウを開くことを可能にするゼロまたは 1つ以上のアイコン (「ビュー」) を含むタイトル・バーで構成されます。このアクションおよびビューを備えた内部パネル・タイトル・バーは、「アクション・バー」と呼ばれます。

詳細ビューは、複数の API を利用します。これらの API は、次のように考慮されます。

1. 各エンティティーに対して 1 つの詳細 API を定義できます。
2. 各詳細ビューは、呼び出す複数の API を指定できます。
3. 各詳細ビューは、内部パネルで構成され、各内部パネルは、呼び出す複数の API を指定できます。

また、詳細ビューがデフォルト詳細 API を呼び出さないように詳細ビューを構成することができます。詳細の構成について詳しくは、「*Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales: 構成ガイド (Configuration Guide)*」を参照してください。

詳細ビューは、1 つ以上の、次の要素で構成されます。

- 内部パネル — 1 つ以上。
- JSP ページ — リソース階層ツリー内のビュー ID によって定義されたアンカー・ページ。
- 保存アクション — 1 つ以上。「JSP コンソール内のリストおよび詳細ビューからのアクション」を参照してください。

JSP コンソールのウィザード

ウィザード・ビューは、マルチスクリーン入力の受け入れに使用されます。ウィザード・ビューは、1 つ以上のウィザード・ページで構成されます。各ウィザード・ページが、詳細ビューとして定義されます。詳細ビューについて詳しくは、『JSP コンソールの詳細ビュー画面 (Detail View Screens in the JSP Console)』を参照してください。

ウィザード・ビューは、アクセス権が制御され、優先順位重視で、ウィザード定義 XML を含みます。ウィザード・ビューは、以下のコンポーネントで構成されます。

- ウィザード定義 — ウィザード定義には、1 つ以上のウィザード・エンティティー (ウィザード・ページまたはウィザード・ルール) 定義と 1 つ以上のウィザード遷移定義が含まれます。ウィザード定義は、ウィザードのフローを定義します。ウィザード定義について詳しくは、『ウィザード定義とは』を参照してください。
- ウィザード・サーブレット — ウィザード・サーブレットは、以下のウィザード関連タスクを処理します。
  - ウィザード・ビューのアクセス権の指定
  - ウィザード定義 XML からのウィザード・ルールの計算
  - 前のウィザード・ページで記入されたデータの処理
  - 現行ウィザード・ページの詳細ビューを指定するための詳細サーブレットの呼び出し、およびその詳細ビューに対応する API の呼び出し
- ウィザード・アンカー・ページ — ウィザード・アンカー・ページは、ブレッドクラムおよびウィザード・アクション・ボタンを組み込むためのプレースホルダーを提供します。ウィザード・アンカー・ページには、現行ウィザード・ページに対応する詳細ビューの JSP も組み込まれます。
- 保存アクション — Save エレメントには、保存アクションで呼び出されるすべての API のリスト、「終了」ボタンの JavaScript ハンドラー、およびウィザードの完了時に表示されるビューのビュー ID が含まれます。
- ウィザード・アクション — 1 つのウィザード・ビューから 4 つのアクションを開始できます。ウィザード・ビューのアクション・タイプについて詳しくは、『ウィザードからのアクション』を参照してください。

ブレッドクラムとは

このブレッドクラムは、ユーザーにウィザード・ビュー内のユーザーのロケーションを示します。ウィザード・ビュー内で現行ページに至るまでにユーザーがトラバースしたパスを示します。ブレッドクラムを使用すると、ユーザーは、現行ページに進むためにナビゲートしてきた前の各ページに戻ることができます。ウィザード定義で各ウィザード・ページに対してブレッドクラム・カテゴリーを定義できます。ブレッドクラム・カテゴリーは、トラバースしたパスの判別に使用され、ブレッドクラムに表示されます。

ウィザード定義とは

ウィザード定義は、ウィザードのフローを定義します。 1 つのウィザード定義は、1 つ以上のウィザード・エンティティー (ウィザード・ページまたはウィザード・ルール) とウィザード遷移で構成されます。ウィザードのフローを制御する新規ウ

ィザード・ルールを定義できます。ウィザードのフローは、ウィザード・ルールの出力値に依存します。ウィザード・ルールの出力は、ウィザード遷移の値と比較されます。ウィザード遷移は、ウィザード・エンティティー間での制御権の移動に使用されます。ウィザード定義には、以下が含まれます。

- ウィザード・エンティティー — 以下の 2 種類のウィザード・エンティティーがあります。
  - ウィザード・ページ — ウィザード・ページでは、ユーザーからの入力を取得するために UI が管理されます。各ウィザード・ページが詳細ビューとして定義され、それにビュー ID が関連付けられます。1 つのウィザード定義に 1 つ以上のウィザード・ページを指定できます。各ウィザード・ページに、次へアクションで呼び出される JavaScript を指定することもできます。各ウィザード・ページにブレッドクラムを指定することもできます。1 つのウィザード定義に 1 つ以上のウィザード・ページを指定できます。
  - ウィザード・ルール — ウィザード・ルールは、さまざまな出力値を評価するために計算を実行する論理ステップです。これらの出力値に基づいて、ウィザードのフローが決定されます。Java ルールと Greex ルールの両方を定義できます。1 つのウィザード定義に 1 つ以上のウィザード・ルールを指定できます。
- ウィザード遷移 — ウィザード遷移は、1 つのウィザード・エンティティー (ウィザード・ページまたはウィザード・ルール) から別のウィザード・エンティティー (ウィザード・ページまたはウィザード・ルール) への制御権の移動に使用されます。ウィザード遷移は、ウィザード・エンティティーのシーケンスを互いに接続します。ウィザード遷移の値はウィザード・ルールの出力と比較され、この比較に基づいて、制御権が次のウィザード・エンティティー (ウィザード・ページまたはウィザード・ルール) に移動されます。1 つのウィザード定義に 1 つ以上のウィザード遷移を指定できます。

ウィザードからのアクション

ウィザード・ビューから、以下のアクションを実行できます。

- 次へ — 次へアクションでは、JavaScript は、すべての検証を実行し、アクションのタイプを示した追加パラメーターとともにデータをポストします。ウィザード・サーブレットは、ルールを実行し、現行ウィザード・ビューの次のウィザード・ページを決定します。詳細サーブレットが呼び出され、現行ウィザード・ビューの次のウィザード・ページが表示されます。前のすべてのウィザード・ページからのデータがマージされ、要求に応じて使用可能になります。
- 前へ (Previous) — 前へ (Previous) アクションでは、ウィザード・サーブレットが、現行ウィザード・ビューの前のウィザード・ページを判別し、そのウィザード・ページでポストされたすべてのパラメーターをセッションから削除します。詳細サーブレットが呼び出され、現行ウィザード・ビューの前のウィザード・ページが表示されます。
- 終了 — 終了アクションでは、ウィザード・サーブレットが、現行ウィザード・ビューに定義されている保存アクションを呼び出します。
- キャンセル — キャンセル・アクションでは、ウィザード・サーブレットが、ウィザードの全データをセッションからパージします。

すべてのウィザード・アクションがデータをサーバーにポストバックし、ウィザード・サーブレットが処理を実行して、ウィザード・フローを決定します。ウィザード・アクションは、アクション・タイプを示した追加パラメーターを要求とともに送信します。

各ウィザード・ページの終了アクションと次へアクションに、リソースの保存に関する JavaScript を定義できます。これらの JavaScript では、追加の検証を実行できます。

JSP コンソール内のリスト・ビューおよび詳細ビューからのアクション

リスト・ビューおよび詳細ビューから、以下のアクションを実行できます。

- 別のビューに移動 — モーダル・ダイアログでビューを開きます
- スクリプトの呼び出し — 指定した JavaScript を呼び出します
- API の呼び出し — 指定した API を呼び出します
- ロールバック専用モードでの API の呼び出し — 指定した API をロールバック専用モードで呼び出します。このアクションは、「仮定」タイプのシナリオで使用できます。例えば、顧客サービス担当者 (customer service representative) は、データベース内でトランザクション (transaction) をコミットすることなく行を追加することによってオーダーの合計価格を確認できます。このオプションは、リソース階層ツリー内で使用可能にすることができます。このオプションの使用可能化について詳しくは、「Applications ManagerConfigurator」、「Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales:*構成ガイド (Configuration Guide)*」を参照してください。

各アクションは、それ自体で使用することができますが、結合するとより役立ちます。例えば、1 つのアクションに対して、1 つの JavaScript と 1 つの API を構成できます。このシナリオでは、指定した API は、指定した JavaScript が TRUE を返した場合にのみ呼び出されます。

別の例では、1 つのアクションに対して 1 つの API と 1 つのビューを構成できます。このシナリオでは、API が呼び出され、それが成功するかどうかにかかわらず、構成されたビューが呼び出されます。このビューは、同じブラウザー・ウィンドウ内で開きます。

注: ポップアップ・ブラウザー・ウィンドウは、JavaScript 呼び出し window.dialogWidth, window.dialogHeight を使用し、必要に応じて幅と高さを指定することによってサイズを変更できます。

API の成功時にのみ開くようにビューを構成

このタスクについて

Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales で、API から成功が返された場合にのみビューが開くようにするには、それを特定して構成する必要があります。

宛先 (To)API の成功時にのみ開くようにビューを構成する手順は、次のとおりです。

手順

1. API を呼び出すアクションを構成します。

   注: アプリケーションのデフォルトのアクション・コードとの不一致を避けるために、一定の命名規則を使用してアクション・コードを定義する必要があります。例えば、カスタム・アクション・コードには、接頭辞 EXTN_ を付けることができます。

2. 指定された属性を API の出力 XML で検出し、その属性を HTML の指定されたカスタム属性に格納するよう JSP を構成します。
3. クライアント・サイドで、指定されたカスタム属性を検索し、showDetailFor() JavaScript 関数を使用して必要な画面に切り替える onLoad 関数を記述します。

第 8 章 JSP コンソール内のビューおよびウィザードのカスタマイズ

JSP コンソール内でのテンプレートを使用しないビューの作成

新しいビューは、独自に作成するか、または既存のビューをテンプレートとして使用して作成できます。テンプレートを使用するとより簡単です。完全に新しいリソースを作成することを選択することもできます。この方式は、既存のリソースから提供される構造およびテンプレートを利用できないため、上級者にのみ推奨します。

重要: 新しく作成しようとする場合は、リソースの構造を完全に理解している必要があります。

新規リソースの作成

このタスクについて

新規リソースを作成するには、以下の手順を実行します。

手順

1. Applications ManagerConfigurator のメニューから、「**アプリケーション・プラットフォーム**」 > 「**プレゼンテーション**」 > 「**リソース**」を選択します。リソース階層ツリーが表示されます。
2. 「**新規作成**」を選択します。
3. 以下のルールに留意して、新規リソースをカスタマイズします。
   - 作成できる最高レベルのリソースは、エンティティー・リソースです。
   - カスタム・エンティティーの下には、ビュー、内部パネルなどで構成されるリソースの完全な階層を作成できます。
   - 一般的に、エンティティー・リソースは、対応するリスト API および詳細 API を含みます。
   - 詳細ビューは、少なくも 1 つの内部パネルを含みます。
   - アクション・リソースは、以下の条件のいずれか 1 つに適合している必要があります。
     - API サブリソースを持つ
     - ビュー ID で構成される
     - 呼び出す JavaScript で構成される

次のタスク

または、より簡単に Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales をカスタマイズする場合には、テンプレートを使用して、Application Console 内で使用可能なエンティティー内で新しいビュー、アクション、またはアイコンを作成できます。

JSP コンソール内でのテンプレートを使用したビューの作成

テンプレートからビューを作成するには、既存のビューをコピーして変更します。ビューに対してマイナーな変更が必要か、または全く新しいビューかにかかわらず、以下の利点が提供されるため、これが最も簡単な方法です。

- 必要な作業量の削減
- 標準のルック・アンド・フィールの保証
- アプリケーション・ロジックの適切な実行の保証

注: ビューをカスタマイズするときには、*INSTALL_DIR*/properties/customer_overrides.properties ファイルを使用して、yfs.uidev.refreshResources プロパティーを「Y」に設定します。先に進んでリフレッシュ・アイコンをクリックする前に、プロパティーを設定していることを確認してください。

注: customer_overrides.properties ファイルを使用したプロパティーのオーバーライドについて詳しくは、「Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales: プロパティー・ガイド (Properties Guide)」を参照してください。

注: 新しいビューを作成するときには、メンテナンスの簡便性と作成の簡便性を考慮してください。詳しくは、「JSP コンソールをカスタマイズする前に」を参照してください。

JSP コンソール内の検索ビューのカスタマイズ

このタスクについて

検索ビューをカスタマイズするには、最小の変更を行います。

検索ビューをカスタマイズするには、以下の手順を実行します。

手順

1. 変更する画面にナビゲートすると、ビュー ID を決定できます。
2. マウス・ポインターを画面のビュー・タイトル上に移動します。ビュー・タイトルのツール・ヒントにビュー ID が示されます。
3. リソース階層ツリーから、ビュー ID にナビゲートして、プロンプトが出されたらサブリソースを含めてコピーします。
4. オリジナル・ビューの JSP ファイルを *INSTALL_DIR*/extensions/global/webpages/*PathOfOriginalJSP*/ ディレクトリーにコピーします。

   注: このビューをカスタマイズするには、その他のファイルを /webpages フォルダーにコピーすることが必要です。これらの必要なファイルのリストは、エンティティーを右クリックして JSP リストを取得することによって参照できます。
5. リソース階層ツリーから、新しいビューにナビゲートします。JSP フィールド内に JSP ファイルへの相対パスを入力します。次に例を示します。

   ```
   /extensions/global/webpages/om/order/search/order_search_bystatus.jsp
   ```

6. 新しい検索ビューをデフォルトのビューにするには、新しいビューを順序付けし直して、シーケンス番号がオリジナルのビューの番号よりも小さい番号になるようにします。
7. 必要に応じて JSP ファイルを編集します。
8. 要求に合う「キャッシュの更新」アイコンを選択します。以下に例を示します。
   - 1 つのエンティティーとその子リソースを更新する場合 - 特定のエンティティーを選択して  「エンティティー・キャッシュの更新」アイコンを選択します。
   - すべてのリソースを更新する場合 -  「キャッシュの更新」アイコンを選択します。
9. 再度 Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales にログインして、変更内容をテストします。

JSP コンソール内の標準リスト・ビューのカスタマイズ

このタスクについて

標準リスト・ビューをカスタマイズするには、最小の変更を行います。表示される要素または表示されるレコードの最大数を変更できます。

標準リスト・ビューをカスタマイズするには、以下の手順を実行します。

手順

1. 変更する画面にナビゲートし、ビュー ID を見つけます。
2. マウス・ポインターを画面のビュー・タイトル上に移動します。ビュー・タイトルのツール・ヒントにビュー ID が示されます。
3. リソース階層ツリーから、ビュー ID にナビゲートして、プロンプトが出されたらサブリソースを含めてコピーします。
4. オリジナル・ビューの JSP ファイルを *INSTALL_DIR*/extensions/global/webpages/*PathOfOriginalJSP*/ ディレクトリーにコピーします。

   注: このビューをカスタマイズするには、その他のファイルを webpages フォルダーにコピーすることが必要です。これらの必要なファイルのリストは、エンティティーを右クリックして JSP リストを取得することによって参照できます。
5. リソース階層ツリーから、新しいビューにナビゲートします。JSP フィールド内に JSP ファイルへの相対パスを入力します。次に例を示します。

   ```
   /extensions/global/webpages/om/order/list/order_list_concise.jsp
   ```
6. 新しい検索ビューをデフォルトのビューにするには、新しいビューを順序付けし直して、シーケンス番号がオリジナルのビューの番号よりも小さい番号になるようにします。
7. 必要に応じて JSP ファイルを編集します。
8. 要求に合う「キャッシュの更新」アイコンを選択します。以下に例を示します。
   - 1 つのエンティティーとその子リソースを更新する場合、特定のエンティティーを選択して  「エンティティー・キャッシュの更新」アイコンを選択します。

- すべてのリソースを更新する場合、  「キャッシュの更新」アイコンを選択します。

9. 再度 Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales にログインして、変更内容をテストします。

JSP コンソール内の拡張リスト・ビューのカスタマイズ

このタスクについて

拡張リスト・ビューのカスタマイズは、カスタム詳細ビューの作成に似ています。拡張リストは、任意の特定の検索ビューに対して表示できます。拡張リスト・ビューを含む検索ビューの場合、ユーザーが検索ボタンを選択すると、標準リスト・ビューの代わりに、そのエンティティーの拡張リスト・ビューが表示されます。その他の検索ビューの場合は、標準リスト・ビューが表示されます。

拡張リスト・ビューをカスタマイズするには、以下の手順を実行します。

手順

1. Applications Manager で、新しい検索ビューを「詳細の表示」フラグを選択した状態で定義します。

   ユーザーが検索を実行すると、(典型的な検索画面のデフォルトのリスト・ビューではなく) そのエンティティーのデフォルト詳細ビューが表示されます。

2. 必要に応じて JSP ファイルを編集します。

   以下のコントロールを使用する場合は、検索ビューの入力フィールドを XML にバインディングするときにアプリケーションの対応する JSP 関数を含めるようにしてください。これにより、入力フィールドが、yfcSearchCriteria 名前空間の拡張リスト画面内で呼び出される API への入力として使用可能になります。

   注: 検索ビューで使用される標準 JSP は、リストされている以外の関数を含む場合があります。拡張リスト・ビューをカスタマイズするときには、コントロールが以下にリストされている関数を使用することを検証します。

コントロール	機能
テキスト・ボックス	getTextOptions
ドロップダウンの選択	getComboOptions
ラジオ・ボタン	getRadioOptions
チェック・ボックス	getCheckBoxOptions
入力の非表示	getTextOptions

3. 「JSP コンソール内の詳細ビューのカスタマイズ」で説明されているように、カスタム詳細 JSP ページを作成します。(これは、拡張リスト・ビューとして使用されます。)
4. 拡張リスト・ビューを表示するために必要なすべての追加 API を定義します。

yfcSearchCriteria 名前空間は、追加 API で必要なすべてのデータを含み、その xml:/SearchData/@MaxRecords 属性は拡張リスト・ビューの「最大レコード数」値を指定します。

5. 拡張リスト・ビューの開き方を決定します。

コンソール JSP 内のリスト・ビューの最大レコード数

デフォルトでは、当アプリケーションでは最大 30 件のレコードがユーザーに対して表示されます。ユーザーは、より多くの結果を画面に表示したい場合は、最大で 200 件のレコードを表示できます。

ユーザーが 200 件を超えるレコードを表示できるようにするには、customer_overrides.properties ファイルを編集することで、新しい最大レコード数を設定できます。 customer_overrides.properties ファイルで新しい上限数を設定したら、このシステム・レベルの設定はすべてのユーザーに適用されます。

デフォルトで 30 件のレコードが表示されるという設定は変更できません。

注: 上限数を 200 よりも大きくすると、パフォーマンスが低下します。また、アプリケーション・サーバーの JVM ヒープを拡大することも必要になります。

最大レコード数の変更

このタスクについて

表示されるレコードの最大数を変更するには、次の操作を実行します。

INSTALL_DIR/properties/customer_overrides.properties ファイル内の yfs.ui.MaxRecords プロパティーを構成します。

JSP コンソール内の詳細ビューのカスタマイズ

このタスクについて

詳細ビューは、より具体的な情報の明細を示します。このビューは、アンカー・ページ内に 1 つ以上の内部パネルで構成されます。各内部パネルは、フィールドとゼロまたは 1 つ以上のアクションおよびゼロまたは 1 つ以上のアイコンで構成されます。詳細ビューは、複数の API を利用できます。

詳細ビューをカスタマイズするには、以下の手順を実行します。

手順

1. 変更する画面にナビゲートすると、ビュー ID を決定できます。
2. マウス・ポインターを画面のビュー・タイトル上に移動します。ビュー・タイトルのツール・ヒントにビュー ID が示されます。
3. リソース階層ツリーから、ビュー ID にナビゲートして、プロンプトが出されたらサブリソースを含めてコピーします。
4. オリジナル・ビューの JSP ファイルを *INSTALL_DIR*/extensions/global/webpages/*PathOfOriginalJSP/* ディレクトリーにコピーします。

注: このビューをカスタマイズするには、その他のファイルを extn フォルダーにコピーすることが必要です。これらの必要なファイルのリストは、エンティティーを右クリックして JSP リストを取得することによって参照できます。

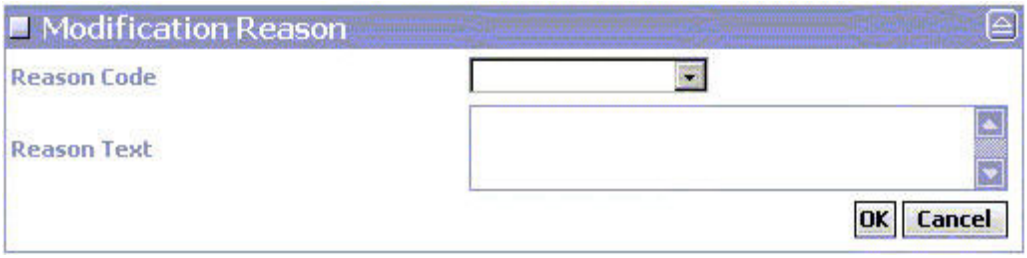
5. オリジナルのビューがアンカー・ページを使用していて、それをカスタマイズする場合は、リソース階層ツリーを使用して新しいビューにナビゲートします。JSP フィールド内に JSP ファイルの相対パスを入力します。
6. 必要に応じてアンカー・ページ JSP ファイルを編集します。

   注: アンカー・ページが、含めるすべての内部パネルのリソース ID を含んでいることを検証します。「*Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales: 構成ガイド (Configuration Guide)*」を参照してください。
7. カスタマイズする必要がある各内部パネルで、以下の手順を実行します。
   a. オリジナルの内部パネルの JSP ファイルを *INSTALL_DIR*/extensions/global/webpages/*PathOfOriginalJSP*/ ディレクトリーにコピーします。
   b. リソース階層ツリーから、新しい内部パネルにナビゲートします。JSP フィールド内に JSP ファイルへの相対パスを入力します。次に例を示します。

      ```
      /extensions/global/webpages/om/order/detail/order_detail_header.jsp
      ```
   c. 必要に応じて内部パネル JSP ファイルを編集します。
8. 新しい詳細ビューをデフォルトのビューにするには、新しいビューを順序付けし直して、シーケンス番号がオリジナルのビューの番号よりも小さい番号になるようにします。
9. すべてのリンクが適切に新しいビューをポイントするようにするには、「カスタマイズしたビューをアプリケーション全体に取り込む (Incorporating Customized Views Across the Application)」の手順に従ってください。
10. 要求に合う「キャッシュの更新」アイコンを選択します。以下に例を示します。
    - 1 つのエンティティーとその子リソースを更新する場合、特定のエンティティーを選択して  「エンティティー・キャッシュの更新」アイコンを選択します。
    - すべてのリソースを更新する場合、  「キャッシュの更新」アイコンを選択します。
11. 再度アプリケーションにログインして、変更をテストします。

詳細ビューでの理由コード・ポップアップのブロック

Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales コンソールの詳細画面で行った変更に対する「保存」アクションによって、行った変更に対する理由コードを尋ねるポップアップが表示されます。「理由コード」ポップアップは、カスタム画面でのみ表示をブロックすることができます。



注: ポップアップの編集は、画面がカスタム画面の場合にのみ可能です。デフォルト画面は編集できません。

ポップアップ画面の非表示

このタスクについて

「保存」アクションによって呼び出されたポップアップ画面が表示されないようにするには、次の手順を実行します。

手順

1. Applications ManagerConfigurator のメニューから、「**「アプリケーション・プラットフォーム」 > 「プレゼンテーション」 > 「リソース」**を選択します。これにより、「リソース階層」ツリーが表示されます。
2. **「エンティティー」 > 「詳細ビュー」 > 「アクション」** (編集対象) を選択します。この例では、変更するアクションとして「**保存**」を選択します。
3. ポップアップ画面を呼び出す JavaScript メソッドを削除します。

JSP コンソール内のウィザードのカスタマイズについて

ウィザード・ビューは、マルチ画面入力をサポートします。このビューは、1 つ以上のウィザード・ページで構成されます。各ウィザード・ページは、詳細ビューとして定義され、より具体的な情報を表示します。ウィザード・ビューは、複数の API を利用できます。ウィザード・ビューは、単一のウィザード・ページをカスタマイズするか、またはウィザード定義をカスタマイズすることによって、カスタマイズすることができます。

JSP コンソール内のウィザード・ページのカスタマイズ

このタスクについて

ウィザード・ビュー内のウィザード・ページは、詳細ビューとして定義されています。各ウィザード・ページは、「次へ」、「前へ」、「終了」、および「キャンセル」という 4 つのウィザード・アクションで構成されています。各ウィザード・ページには、ユーザーがたどってきた経路を示すブレッドクラムも表示され、ユーザーはこのブレッドクラムを使用してウィザード・ビュー内の他の任意のページに移動できます。各ウィザード・ページは詳細ビューとして定義されているため、ウィザード・ページをカスタマイズするプロセスは、詳細ビューをカスタマイズするプロセスと似ています。

注: 対象のウィザード・ページがアクセス権に基づいて制御されていないことを確認してください。

ウィザード・ページをカスタマイズするには、次の手順を実行します。

手順

1. ウィザード・ビューで、ビュー ID を確認するために、変更するウィザード・ページに移動します。
2. そのウィザード・ページのビュー・タイトルにマウス・ポインターを合わせます。ビュー・タイトルのツール・ヒントにビュー ID が表示されます。
3. Resource Configurator から、そのビュー ID に移動してビュー ID をコピーします (プロンプトが表示された場合はそのサブリソースも含めてコピーします)。
4. 元のビューの JSP ファイルを *INSTALL_DIR*/repository/eardata/application/war/extn/*PathOfOriginalJSP*/ ディレクトリーにコピーします。

   注: このビューをカスタマイズするには、この extn フォルダーに他のファイルをコピーすることが必要な場合があります。これらの必要なファイルのリストを表示するには、対象のエンティティーを右クリックして JSP リストを取得します。
5. 元のビューでアンカー・ページが使用されている場合に、そのアンカー・ページをカスタマイズするには、Resource Configurator を使用して新しいビューに移動します。お使いの JSP ファイルの相対パスを JSP フィールドに入力します。
6. アンカー・ページの JSP ファイルを必要に応じて編集します。

   注: このアンカー・ページに、組み込みを希望するすべての内部パネルのリソース ID が含まれていることを確認してください。リソース ID について詳しくは、「Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales:構成ガイド」を参照してください。
7. カスタマイズする内部パネルごとに、次の手順を実行します。
   a. 元の内部パネルの JSP ファイルを INSTALL_DIR/repository/eardata/application/war/extn/*PathOfOriginalJSP*/ ディレクトリーにコピーします。
   b. Resource Configurator から、新しい内部パネルに移動します。お使いの JSP ファイルの相対パスを JSP フィールドに入力します。次に例を示します。

      ```
      /extn/om/order/detail/order_detail_header.jsp
      ```
   c. 内部パネルの JSP ファイルを必要に応じて編集します。
8. 新しい詳細ビューをデフォルト・ビューとして使用するには、新しいビューを再シーケンスして、そのシーケンス番号が元のビューのシーケンス番号より小さくなるようにします。
9. すべてのリンクが新しいビューを適切に指し示すようにするには、『アプリケーション全体にわたるカスタマイズ・ビューの導入』に記載された手順に従ってください。
10. 要求に合う「キャッシュの更新」アイコンを選択します。以下に例を示します。

- 1 つのエンティティーとその子リソースを更新するには、そのエンティティーを選択して、  「エンティティー・キャッシュの更新」アイコンをクリックします。
- すべてのリソースを更新するには、  「キャッシュの更新」アイコンをクリックします。

11. 当アプリケーションに再ログインして、変更内容をテストします。

JSP コンソール内のウィザード定義のカスタマイズ

ウィザード定義は、ウィザード規則評価を通じてウィザードのフローを制御します。ウィザード定義は、application.xml ファイル内に定義されます。ここには、個々のウィザード・ページおよび規則実装についての情報が含まれます。ウィザード定義をカスタマイズして、1 つ以上のウィザード・ページを組み込むかまたは除外することができます。また、規則実装を追加または変更することもできます。

既存のウィザード・ビューの変更

このタスクについて

既存のウィザード・ビューを変更するには、その特定のウィザード・ビューのビュー ID を確認する必要があります。ビュー ID を取得する手順は、次のとおりです。

手順

1. ビュー ID を確認するために、変更対象のウィザード・ビューにナビゲートします。
2. ウィザード・ビューのタイトルの上にマウス・ポインターを移動します。ビュー・タイトルのツール・ヒントにビュー ID が示されます。

次のタスク

ウィザード・ビューは、以下を実行することによってカスタマイズできます。

- 新規ウィザード・ページの追加
- 新規ウィザード・ルールの追加
- 新規ウィザード遷移の追加

新規ウィザード・ページの追加

このタスクについて

新しいウィザード・ページを追加するには、次の手順を実行します。

手順

1. application.xml ファイルを開いて、<WizardViews> タグ内で、変更するウィザード・ビューのビュー ID を探します。
2. <WizardEntities> 要素内で、新しい <WizardEntity> 要素を作成して、その属性を設定します。ウィザード・ページを対象にした WizardEntity 要素の属性リストについては、次の表を参照してください。

属性	説明
id	そのウィザード・ページの固有 ID を指定します。
view	そのウィザード・ページのビュー ID を指定します。
start	そのウィザード・ページがウィザード・ビューの最初のページである場合にのみ、この属性を "true" に設定します。
type	PAGE という値を指定します。
javascript	そのウィザード・ページ上で「次へ」アクションが実行されたときに呼び出す JavaScript の名前を指定します。
カテゴリー	ウィザード・ビューにおけるそのウィザード・ページの位置を指定します。この属性の値を使用して、ブレッドクラムが表示されます。

新規ウィザード・ルールの追加

このタスクについて

ウィザード・ルールは、論理計算を実行して、さまざまな出力値を返します。これらの出力値に基づいて、ウィザードのフローが決定されます。Java ルールと Greex ルールの両方を定義できます。

新しいウィザード・ルールを追加するには、次の手順を実行します。

手順

1. application.xml ファイルを開いて、<WizardViews> タグ内で、変更するウィザード・ビューのビュー ID を探します。
2. <WizardEntities> 要素内で、新しい <WizardEntity> 要素を作成して、その属性を設定します。ウィザード・ルールを対象にした WizardEntity 要素の属性リストについては、次の表を参照してください。

属性	説明
id	そのウィザード・ルールの固有 ID を指定します。
type	RULE という値を指定します。
impl	そのウィザード・ルールの実装クラスの完全修飾パスを指定します。 • Java ルールについては、ルール実装クラスの前に "java:" を付加します。例えば、java:com.yantra.yfs.rules.YFSWizardRule1 のように指定します。すべての Java ルールでは、YCPWizard ルール・インターフェースを実装する必要があります。 • Greex ルールについては、ルール実装クラスの前に "greex:" を付加します。例えば、greex:com.yantra.yfs.rules.YFSWizardRule1 のように指定します。Greex ルールは evaluateCondition API を使用して評価されます。

3. <WizardEntity> 要素内で、新しい <Namespace> 要素を作成します。
4. <WizardEntity> 要素の name 属性で、そのウィザード・ルールの入力名前空間を指定します。1 つのルールに複数の入力名前空間を指定する場合は、それらの入力名前空間をコンマで区切ります。

5. <WizardEntity> 要素内で、新しい <Namespace> 要素を作成します。
6. <WizardEntity> 要素の name 属性で、そのウィザード・ルールの入力名前空間を指定します。1 つのルールに複数の入力名前空間を指定する場合は、それらの入力名前空間をコンマで区切ります。
7. <WizardEntity> 要素内で、新しい <Output> 要素を作成します。
8. <Output> 要素の value 属性で、そのウィザード・ルールが返す出力値を指定します。ウィザード・ルールは複数の出力値を返すことができます。それぞれの出力値について、別個の <Output> 要素を作成します。これらの出力値に基づいて、ウィザードのフローが決定されます。

新規ウィザード・トランジションの追加

このタスクについて

ウィザード・トランジションを使用して、ウィザードのフローを制御します。ウィザード・トランジションは、あるウィザード・エンティティー (ウィザード・ページまたはウィザード・ルール) から別のウィザード・エンティティー (ウィザード・ページまたはウィザード・ルール) に制御を移動します。

新しいウィザード・トランジションを追加するには、次の手順を実行します。

手順

1. application.xml ファイルを開いて、<WizardViews> タグ内で、変更するウィザード・ビューのビュー ID を探します。
2. <WizardTransitions> 要素内で、新しい <WizardTransition> 要素を作成して、その属性を設定します。WizardTransition 要素の属性リストについては、次の表を参照してください。

属性	説明
id	そのウィザード・トランジションの固有 ID を指定します。
source	ソースとなるウィザード・エンティティー (ウィザード・ページまたはウィザード・ルール) の ID を指定します。
target	ターゲットとなるウィザード・エンティティー (ウィザード・ページまたはウィザード・ルール) の ID を指定します。

3. <WizardTransition> 要素内で、新しい <Output> 要素を作成します。
4. <Output> 要素の target 属性で、ターゲットとなるウィザード・エンティティー (ウィザード・ページまたはウィザード・ルール) の ID を指定して、<Output> 要素の value 属性で、トランジションを実行する対象となる出力値を指定します。ターゲットと出力値のペアごとに、別個の <Output> 要素を作成します。

第 9 章 JSP コンソール内の JSP ファイルのカスタマイズ

JSP コンソール内の JSP ファイルについて

JSP ファイルは、HTML ページを動的に表示できるようにする構文を含みます。JSP ファイルのタイプは、検索ビュー、リスト・ビュー、および詳細ビューに対応します。各 JSP ファイルの標準は、それが使用する画面のタイプにより異なります。

検索ビューの JSP ファイル

検索ビューの JSP ファイルは、一般的に、ユーザーが検索条件を入力できるようにする入力フィールドを作成するタグを含みます。検索 JSP ファイルは通常、以下のタイプの HTML コントロールを含みます。

- ラベル
- 入力フィールド
- コンボ・ボックス
- ラジオ・ボタン
- チェック・ボックス

リスト・ビューの JSP ファイル

リスト・ビューの JSP ファイルは、一般的に、列ヘッダーとデータ・セルを備えた HTML 表のみを含みます。ほとんどのリスト・ビューは、リスト内のレコード上で実行できるアクションと共に使用するチェック・ボックスも含みます。通常、XML キーは、表上で作成され、チェック・ボックスと関連付けられます。

詳細ビューの JSP ファイル

詳細ビューの JSP ファイルは、一般的に、最も複雑です。詳細ビューは多数の種類のコントロールを必要とするためです。通常、詳細ビューは、検索ビューと同じ種類の HTML コントロールを含みます。また、詳細ビューは以下のコントロールも含みます。

- テキスト領域
- テーブル
- グラフ

要確認: 製品全体で一貫性のある外観を維持するために、カスタマイズするすべての JSP ファイルで同じスタイル・シート (CSS ファイル) を使用してください。

JSP コンソール内のサンプル JSP ファイル

Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales で用意されている JSP ファイルは、開発者が遭遇するすべての一般的な使用事例をカバーしているため、独自の JSP ファイルを開発するためのテンプレートとして役に立ちます。これらの JSP ファイルを調べればわかるように、これらのファイルはモジュール方式であるため、コードの塊を組み合わせることでビューを素早くカスタマイズできます。JSP ファイル・テンプレートに加えて、『*User Interface Style Reference*』というトピックを参照して詳細な技術情報を確認してください。

次の例では、「オーダー」リスト・ビューを表示するために使用できる一般的なオーダー JSP ファイルから抜粋したコードを示しています。

```
<%@taglib  prefix="yfc" uri="/WEB-INF/yfc.tld" %>
<%@include file="/yfsjspcommon/yfsutil.jspf"%>
<%@include file="/console/jsp/currencyutils.jspf" %>
<%@ page import="com.yantra.yfs.ui.backend.*" %>
<%@ include file="/console/jsp/modificationutils.jspf" %>
<script language="javascript" src="/smcfsapplication_name
/console/scripts/modificationreason.js"></script>
<table class="table" editable="false" width="100%" cellspacing="0">
   <thead>
      <tr>
         <td sortable="no" class="checkboxheader">
            <input type="hidden" name="userHasOverridePermissions"
value='<%=userHasOverridePermissions()%>'/>
            <input type="hidden" name="xml:/Order/@Override" value="N"/>
            <input type="checkbox" name="checkbox" value="checkbox"
onclick="doCheckAll(this);"/>
         </td>
         <td class="tablecolumnheader"><yfc:i18n>Order_#</yfc:i18n></td>
         <td class="tablecolumnheader"><yfc:i18n>Status</yfc:i18n></td>
         <td class="tablecolumnheader"><yfc:i18n>Enterprise</yfc:i18n></td>
         <td class="tablecolumnheader"><yfc:i18n>Buyer</yfc:i18n></td>
         <td class="tablecolumnheader"><yfc:i18n>Order_Date</yfc:i18n></td>
         <td class="tablecolumnheader"><yfc:i18n>Total_Amount</yfc:i18n></td>
      </tr>
   </thead>
   <tbody>
       <yfc:loopXML binding="xml:/OrderList/@Order" id="Order">
          <tr>
             <yfc:makeXMLInput name="orderKey">
                 <yfc:makeXMLKey binding="xml:/Order/@OrderHeaderKey"
value="xml:/Order/@OrderHeaderKey" />
             </yfc:makeXMLInput>
             <td class="checkboxcolumn">
                <input type="checkbox" value='<%=getParameter("orderKey")%>'
name="EntityKey"/>
             </td>
             <td class="tablecolumn">
                <a href="javascript:showDetailFor('<%=getParameter("orderKey")%>');">
                <yfc:getXMLValue binding="xml:/Order/@OrderNo"/>
                </a>
              </td>
              <td class="tablecolumn">
                  <% if (isVoid(getValue("Order", "xml:/Order/@Status"))) { %>
                     [<yfc:i18n>Draft</yfc:i18n>]
                  <% } else { %>
                     <yfc:getXMLValue binding="xml:/Order/@Status"/>
                  <% } %>
                  <% if (equals("Y", getValue("Order", "xml:/Order/@HoldFlag")))
                     { %>
                        <img class="icon" onmouseover="this.style.cursor='default'"
```

```
<%=getImageOptions(YFSUIBackendConsts.HELD_ORDER, "This_order_is_held")%>/>
                    <% } %>
                </td>
                <td class="tablecolumn">
                    <yfc:getXMLValue binding="xml:/Order/@EnterpriseCode"/></td>
                <td class="tablecolumn">
                    <yfc:getXMLValue binding="xml:/Order/@BuyerOrganizationCode"/></td>
                <td class="tablecolumn" sortValue="<%=getDateValue
("xml:/Order/@OrderDate")%>"><yfc:getXMLValue binding="xml:/Order/@OrderDate"/></td>
                <td class="numerictablecolumn"
sortValue="<%=getNumericValue("xml:/Order/PriceInfo/@TotalAmount")%>">
                     <%=displayAmount(getValue("Order",
"xml:/Order/PriceInfo/@TotalAmount"), (YFCElement)
request.getAttribute("CurrencyList"), getValue("Order",
"xml:/Order/@RulesetKey"), getValue("Order",
"xml:/Order/PriceInfo/@Currency"))%>
                 </td>
            </tr>
        </yfc:loopXML>
    </tbody>
</table>
```

この例では、先頭にインクルード・ファイルを示しています。プレゼンテーション・フレームワーク内の共通パブリック JSP 関数のほとんどにアクセスするには、ほとんどの JSP ファイルでは、*INSTALL_DIR*/repository/eardata/platform/war/yfsjspcommon/yfsutil.jspf ファイルへの参照のみを必要とします。次の例では、このための include ステートメントを示しています。

```
<%@include file="/yfsjspcommon/yfsutil.jspf"%>
```

yfsGet*Options() JSP 関数にアクセスするには、次の例のように、*INSTALL_DIR*/repository/eardata/platform/war/console/jsp/modificationutils.jspf ファイルを参照する必要があります。

```
<%@ include file="/console/jsp/modificationutils.jspf" %>
```

これらの関数について詳しくは、『JSP Functions for the Console JSP Interface』を参照してください。

JavaScript 関数については、すべてのパブリック関数が独自の JSP ファイルで自動的に使用可能になります。

文書タイプ間の UI ビューの共通 JSP

コンソール画面の特定のセットへのエントリー・ポイントであるコンソール画面 (オーダー検索画面またはオーダー作成画面など) は、ユーザー・インターフェースを文書タイプ機能全体に実装するために使用される共通 JSP を含む必要があります。この機能により、異なる文書タイプの情報を表示するときに既存のコンソール画面を使用することができます。この共通 JSP は、エンティティーのコンソール内のエントリー・ポイント画面が以下のいずれかのフィールドを必要とする場合に使用できます。

- 文書タイプ
- エンタープライズ・コード
- ノード (ノードが主要な重要フィールドである WMS 画面のみ)

common_fields.jsp ファイルの使用

common_fields.jsp (*INSTALL_DIR*/repository/eardata/smcfs/war/yfsjspcommon ディレクトリー *INSTALL_DIR*/repository/eardata/platform/war/yfsjspcommon ディレクトリー内にあります) は、一般的に必要なフィールドを表示するための多数の異なる機能を提供します。common_fields.jsp は、JSP の最上部に含まれます。JSP パラメーターは、JSP の特定の使用のためにどの機能が必要であるかを示すために渡されます。例えば、ノード・フィールドを表示する場合は、「ShowNode」パラメーターを「TRUE」として渡します。

common_fields.jsp に渡すことができるすべての有効なパラメーターを以下に示します。

パラメーター	説明
ShowDocumentType	文書タイプ・フィールドを表示するかどうかを示します。デフォルト : TRUE。
ShowEnterpriseCode	エンタープライズ・コード・フィールドを表示するかどうかを示します。デフォルト : TRUE。
ShowNode	ノード・フィールドを表示するかどうかを示します。デフォルト : FALSE。
DocumentTypeBinding	文書タイプ・フィールドに設定するバインディングを示します。デフォルト : xml:/Order/@DocumentType。
EnterpriseCodeBinding	エンタープライズ・コード・フィールドに設定するバインディングを示します。デフォルト値 : Xml:/Order/@EnterpriseCode。
NodeBinding	ノード・フィールドに設定するバインディングを示します。デフォルト値 : xml:/Order/@ShipNode。
RefreshOnDocumentType	文書タイプが選択されたときに画面全体を最新表示にするかどうかを示します。デフォルト : FALSE。
RefreshOnEnterpriseCode	エンタープライズが選択されたときに画面全体を最新表示にするかどうかを示します。デフォルト : FALSE。
RefreshOnNode	ノードが選択されたときに画面全体を最新表示にするかどうかを示します。デフォルト : FALSE。
ScreenType	この JSP が含まれる画面のタイプを示します。この情報は、JSP 内のフィールドの適切なクラスおよび列レイアウトを設定するために使用されます。有効値は、「search」および「detail」です。デフォルト : search。
ColumnLayout	フィールドを表示するために使用される列数を示します。許可される有効値は、1 および 3 です。デフォルト : ScreenType が「search」の場合は、デフォルトの列レイアウトは 1 です。ScreenType が「detail」の場合は、デフォルトの列レイアウトは 3 です。
NodeLabel	ノード・フィールドの画面ラベルを示します。「ShowNode」が「TRUE」として渡される場合にのみ有効です。デフォルト : Node。

パラメーター	説明
EnterpriseListForNodeField	エンタープライズ・コード・フィールド内に表示される値がノード・フィールド内の選択に基づくかどうかを示します。これは、エンタープライズ・リストがノードの組織内に参加する組織のリストでなければならない WMS 画面に使用されます。「RefreshOnNode」パラメーターが「TRUE」の場合、このパラメーターは「TRUE」として渡し、ノードが選択されたときにエンタープライズ・リストが確実に最新表示になるようにします。「ShowNode」および「ShowEnterpriseCode」が「TRUE」の場合にのみ有効です。デフォルト : FALSE。

注: common_fields.jsp 内のフィールド用にデータを取り込むために呼び出される API は、共通 API 自身によって行われます。これらの API に対して画面内にリソースを定義する必要はありません。例えば、共通 JSP を使用してエンタープライズ・コード・フィールドを表示している場合、画面のリソース内で getOrganizationList() API を定義する必要はありません。

画面の最新表示

伝票種別、エンタープライズ・コード、またはノードに依存しているフィールドについては、共通 JSP の画面最新表示機能を使用してください。例えば、「ステータスに基づいたオーダー検索 (order search by status)」画面では、検索条件として使用できるステータスのリストが表示された「ステータス変更前」フィールドがあります。この有効ステータス・リストは、伝票種別ごとに異なります。したがって、ユーザーが特定の伝票種別を選択した場合は、異なるステータス・リストがこのフィールドに表示される必要があります。このことを実現するには、次の手順を実行する必要があります。

1. common_fields.jsp を JSP の先頭でインクルードします。次のように "RefreshOnDocumentType" パラメーターを "true" という値として渡します。

```
<jsp:include page="/yfsjspcommon/common_fields.jsp" flush="true">
  <jsp:param name="DocumentTypeBinding" value="xml:/OrderRelease/Order/@DocumentType"/>
  <jsp:param name="EnterpriseCodeBinding" value="xml:/OrderRelease/Order/@EnterpriseCode"/>
  <jsp:param name="RefreshOnDocumentType" value="true"/>
</jsp:include>
```

2. 対象のエンティティーのアプリケーション XML 内で、このビュー下で getStatusList() API を定義します。このビューが表示されたときにインフラストラクチャー層によってこの API が呼び出されないような形で、この API を定義します ("LoopAPI" 属性を "Y" に設定します)。次のように DocumentType 属性に対して動的属性を指定することで、画面上のこのフィールドで選択された伝票種別をこの属性が渡すようにしてください。

```
DocumentType="xml:CommonFields:/CommonFields/@DocumentType"
```

3. 対象の画面の JSP では、common_fields.jsp のインクルード箇所の直後で callAPI taglib を使用して getStatusList() API を呼び出します。

注: いずれかの共通フィールド (伝票種別、エンタープライズ・コード、またはノード) を使用して、画面全体を最新表示にすることができます。それぞれの共通フィールドに対して指定する必要のある対応する動的バインディングは、順番に

xml:CommonFields:/CommonFields/@DocumentType、xml:CommonFields:/CommonFields/@EnterpriseCode、および xml:CommonFields:/CommonFields/@Node です。

その他の共通フィールド機能/メモ

共通フィールド JSP (common_fields.jsp) のその他の機能およびメモは次のとおりです。

- common_fields.jsp を使用して特定のフィールドが表示される場合、フィールドの外観はフィールドが表示する必要があるレコードの数により異なります。フィールドに表示するレコードが 1 つのみの場合は、フィールドはユーザーが変更できない保護された入力として表示されます。フィールドに表示するレコードが 2 から 20 の場合は、フィールドはコンボ・ボックスとして表示されます。表示するレコードが 21 より多い場合は、フィールドは保護テキスト・ボックスとして表示され、横にルックアップ・アイコンが表示されます。この場合は、フィールドの値を変更する唯一の方法はルックアップを使用することです。この動作の理由は、コンボ・ボックス内に表示するレコードが多すぎる場合でも「画面最新表示」機能が動作できるようにするためです。
- コンソールにログインしたユーザーがノード・ユーザーの場合、ノード・フィールドは変更できない保護された入力として表示されます。それ以外の場合は、ノード・フィールドは現在ログインしている組織の所有ノード (node) のすべてを表示します。

検索画面用のサンプル common_fields.jsp

次の例では、この共通 JSP を検索画面で使用する方法を示しています。この検索画面にある 2 つのフィールドは、「エンタープライズ・コード」フィールドが変更されるたびに最新表示になる必要があります。

以下のように、すべてのエントリー・ポイント画面の JSP で、common_fields.jsp がインクルードされます。

```
<table class="view">
   <jsp:include page="/yfsjspcommon/common_fields.jsp" flush="true">
      <jsp:param name="DocumentTypeBinding" value="xml:/OrderRelease/Order/@DocumentType"/>
      <jsp:param name="EnterpriseCodeBinding" value="xml:/OrderRelease/Order/@EnterpriseCode"/>
      <jsp:param name="ShowNode" value="true"/>
      <jsp:param name="NodeBinding" value="xml:/OrderRelease/Order/@Node"/>
      <jsp:param name="RefreshOnNode" value="true"/>
      <jsp:param name="RefreshOnEnterprise" value="true"/>
      <jsp:param name="EnterpriseListForNodeField" value="true"/>
   </jsp:include>
   <% // Now call the APIs that are dependent on the common fields (Doc Type, Enterprise
Code, and Node)
      // Product Classes and Unit of Measures are refreshed.
   %>
   <yfc:callAPI apiID="AP2"/>
   <yfc:callAPI apiID="AP3"/>
<tr>
   <td class="searchlabel"><yfc:i18n>field1</yfc:i18n></td>
</tr>
<tr>
   <td nowrap="true" class="searchcriteriacell">
      <select class="combobox" name="xml:/OrderRelease/@Field1QryType">
         <yfc:loopOptions binding="xml:/QueryTypeList/StringQueryTypes/@QueryType"
name="QueryTypeDesc" value="QueryType" selected="xml:/OrderRelease/@Field1QryType "/>
       </select>
```

```
      <input type="text" class="unprotectedinput"
<%=getTextOptions("xml:/OrderRelease/@Field1")%> />
    </td>
</tr>
```

以下のように、API はアプリケーション XML で定義されています。

```
<View ViewGroupID="YOMSXXX" Priority="3" Name="By_Item" ID="YOMSXXX"
 JSP="/om/order/search/wms_by_item.jsp"
OutputNode="Order">
    <APIList>
      <API Name="getQueryTypeList" OutputNode="QueryTypeList">
         <Input>
             <QueryType/>
         </Input>
         <Template>
           <QueryTypeList>
             <StringQueryTypes>
               <QueryType QueryType="" QueryTypeDesc=""/>
             </StringQueryTypes>
           </QueryTypeList>
         </Template>
      </API>
      <API Name="getCommonCodeList" OutputNode="ProductClassList" LoopAPI="Y">
         <Input>
             <CommonCode CodeType="PRODUCT_CLASS"
CallingOrganizationCode="xml:CommonFields:/CommonFields/@EnterpriseCode"/>
         </Input>
         <Template>
             <CommonCodeList>
              <CommonCode CodeValue="" CodeShortDescription=""/>
              </CommonCodeList>
         </Template>
      </API>
      <API Name="getCommonCodeList" OutputNode="UnitOfMeasureList" LoopAPI="Y">
        <Input>
          <CommonCode CodeType="UNIT_OF_MEASURE"
CallingOrganizationCode="xml:CommonFields:/CommonFields/@EnterpriseCode"/>
        </Input>
        <Template>
          <CommonCodeList>
             <CommonCode CodeValue="" CodeShortDescription=""/>
          </CommonCodeList>
        </Template>
      </API>
    </APIList>
</View>
```

詳細ビューの内部パネルの作成

内部パネルについて

各内部パネルは、分離した JSP ファイルから取得され、アンカー・ページ JSP は、これらのファイルをjsp:include JSP タグを介して含めます。詳細ビューに複数の内部パネルを構成する場合、およびそれらを上から下に単純にレイアウトする場合は、プレゼンテーション・フレームワークは、それを行うためのデフォルトのアンカー・ページを提供します。そのような場合は、詳細ビューに対していずれの JSP も構成する必要はありません。

以下は、アンカー・ページに内部パネルを含めるための典型的な構文を示します。

```
<table class="anchor" cellpadding="7px" cellSpacing="0">
<tr>
   <td colspan="2" >
```

```
        <jsp:include page="/yfc/innerpanel.jsp" flush="true" >
            <jsp:param name="CurrentInnerPanelID" value="I01"/>
        </jsp:include>
    </td>
</tr>
<tr>
    <td height="100%" width="75%">
        <jsp:include page="/yfc/innerpanel.jsp" flush="true">
            <jsp:param name="CurrentInnerPanelID" value="I02"/>
        </jsp:include>
    </td>
     <td height="100%" width="25%" addressip="true" >
        <jsp:include page="/yfc/innerpanel.jsp" flush="true">
          <jsp:param name="CurrentInnerPanelID" value="I03"/>
          <jsp:param name="Path" value="xml:/OrderRelease/PersonInfoShipTo"/>
          <jsp:param name="DataXML" value="OrderRelease"/>
          <jsp:param name="AllowedModValue" value='<%=getModificationAllowedValue
             ("ShipToAddress", "xml:/OrderRelease/AllowedModifications")%>'/>
        </jsp:include>
    </td>
</tr>
<tr>
    <td colspan="2" >
        <jsp:include page="/yfc/innerpanel.jsp" flush="true" >
           <jsp:param name="CurrentInnerPanelID" value="I04"/>
        </jsp:include>
    </td>
</tr>
</table>
```

このアプリケーションによって提供される innerpanel.jsp ファイルは、各内部パネル用に表示する必要があるタイトル・バーを含み、各内部パネルのタイトル・バー内で使用可能なアイコンおよびアクション・ボタンを格納します。

JSP:Param JSP タグ・パラメーター

jsp:param JSP タグを使用して、以下のパラメーターを innerpanel.jsp ファイルに指定できます。

- CurrentInnerPanelID - 詳細ビューのリソース ID に対する内部パネル・リソース ID の接尾部。例えば、詳細ビューのリソース ID が YOMD010 の場合、内部パネルのリソース ID は YOMD010I01 です。この例では、I01 接尾部をこの JSP タグに渡します。
- Title - 構成済みの内部パネルのリソース ID に関する説明を置き換えます。
- IPHeight - 内部パネルの高さを固定します。データが、この高さを超えて増えた場合、スクロール・バーが自動的に表示されます。
- IPWidth - 内部パネルの幅を固定します。データが、この幅を超えて増えた場合、スクロール・バーが自動的に表示されます。

これらの属性のほかに、ここで指定するパラメーターは、含められる内部パネルのリソース ID に対して構成された JSP で自動的に使用可能になります。

内部パネルの作成手順

このタスクについて

内部パネルを作成するには、次の手順を実行します。

手順

1. 内部パネルの追加先となるビューをカスタマイズします。
2. アンカー・ページの JSP ファイルを編集して、組み込む他のすべての内部パネルのリソース ID 接尾辞を追加します。

   内部パネルのリソース ID の形式は、<ビュー *ID* のリソース *ID*><*接尾辞*> です。例えば、I01 となります。

   接尾辞のみを参照する必要があります。プレゼンテーション・フレームワークは、完全な内部パネル・リソース ID を生成して、適切な JSP を組み込みます。
3. Applications ManagerApplication Configurator から、新たに作成したビュー下で内部パネル・リソースを作成します。
4. Applications ManagerApplication Configurator から、必要に応じてアクションとリンクを作成します。
5. 標準の詳細内部パネル JSP をテンプレートとして使用して、内部パネルの JSP ファイルを作成します。

JSP パネルを読み取り専用にする

このタスクについて

JSP の特定のパネルを、特定のユーザー・グループに属するユーザーに対して読み取り専用にすることができます。JSP パネルを読み取り専用にすることによって、パネルでユーザーに表示されるデータを変更したり更新したりできないようにすることができます。

特定のユーザーに対して JSP パネルを読み取り専用にするには、*INSTALL_DIR*/database/FactorySetup/XMLS/YCP_YFS_RESOURCE_PERMISSION_CONSOLE.xml ファイルを編集して、特定の JSP 内部パネルのリソース・キーの READ-ONLY-FLAG 属性の値を「True」に設定します。以下に例を示します。

```
<ResourcePermission ActivateFlag="Y" Createprogid="SYSTEM"
   Createuserid="SYSTEM" Modifyprogid="SYSTEM"
   Modifyuserid="SYSTEM" ResourceKey="YEMS012"
   ResourcePermissionKey="SYS_YEMS012" UsergroupKey="SYSTEM"
   READ-ONLY-FLAG="True"/>
```

この例では、YEMS012 が、読み取り専用にする内部パネルのリソース・キーで、SYSTEM が、ユーザーの所属先のユーザー・グループの ID です。

注: アプリケーション・コンフィギュレーターを使用して JSP パネルを読み取り専用にすることもできます。さまざまなユーザー・グループにユーザー・セキュリティーを構成しているときに、特定のユーザー・グループに対して JSP パネルを読み取り専用にできます。ユーザー・セキュリティーの構成について詳しくは、「*Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales 構成ガイド (Configuration Guide)*」を参照してください。

アプリケーション全体にわたるカスタマイズ・ビューの導入

既存のビューをカスタマイズして、当アプリケーション全体にわたってその既存ビューをカスタマイズしたビューに置き換えようとする場合は、その既存ビューを指し示すリンク、アクション、およびアイコンを変更することなく置き換えが可能です。

既存のリンク、アクション、およびアイコンは、同一のビュー・グループ ID を指し示しています。ただし、これらのリンク、アクション、およびアイコンを使用していずれかの画面がアクセスされる場合は、これらは特定のビューを指し示しません。この画面は、そのビュー・グループ ID を持つすべてのビューのうち、最も小さいシーケンス番号が割り当てられたビューとして開かれます。

したがって、当アプリケーション全体にわたって既存のビューを置き換えるには、カスタマイズしたビューには、元のビューと同じグループ ID が割り当てられていること、および元のビューのシーケンス番号より小さいシーケンス番号が割り当てられていることが必要です。

例えば、標準の「オーダーの詳細」ビューをカスタマイズしたビューに置き換えるには、カスタマイズしたビューには YOMD010 というビュー・グループ ID が割り当てられている必要があります。

カスタム・ビューが既存のビューを置き換えない追加の画面である場合は、独自のリンク、アクション、またはアイコンを適切な場所に追加して、これらを使用してその画面にアクセスできるようにする必要があります。

カスタム・ビューには、固有のビュー・グループ ID を割り当てる必要があります。作成するリンク、アクション、またはアイコンは、このビュー・グループ ID を指し示す必要があります。

第 10 章 JSP コンソール内のその他のカスタマイズ

ホーム・ページのカスタマイズ

ホーム・ページは、ホーム・エンティティーのデフォルトの詳細ビューです。標準ホーム・ページは、上部にメニュー・ビューを備え、その下に 3 つのリスト・ビュー (ユーザー指定キュー、警告、およびお気に入り検索) が横に並びます。 標準ホーム・ページを参照するには、「JSP コンソールの画面について」にある図を参照してください。

ホーム・ページをカスタマイズするには、ホーム・エンティティーの詳細ビューをカスタマイズするための手順のいずれかのセットを実行します。

- JSP コンソール内の詳細ビューのカスタマイズ
- 詳細ビューの内部パネルの作成

URL への認証済みアクセスのためのセキュリティー・サーブレット・フィルターのカスタマイズ

コンテキスト・ルート・フィルター用に定義されたサーブレット・フィルターは、ログイン・ページまたは web.xml ファイルに「bypass.uri」として示されているページを除き、Web ルート内に存在するすべてのコンテンツを要求します。サーブレット・フィルターがリソースに対する要求を受け取ると、要求セッション ID を検証します。要求セッション ID が有効な場合は、特定のリソースにリダイレクトします。要求セッション ID が無効な場合は、ログイン・ページにリダイレクトします。サーブレット・フィルター定義が web.xml ファイルに格納され、URL パターン要素がフィルター認証を必要とするすべての URL を含みます。

フィルター認証を適用しない特定の URL を追加する場合は、web.xml ファイル (EARFILE/WARFILE/WEB-INF 内にあります) 内のそのような URL それぞれに config-param 要素を以下のように追加します。

```
<config-param>
   <param-name>bypass.uri.1</param-name>
   <param-value>/console/login.jsp</param-value>
</config-param>
<config-param>
   <param-name>bypass.uri.2</param-name>
   <param-value>/console/start.jsp</param-value>
</config-param>
<config-param>
   <param-name>bypass.uri.3</param-name>
   <param-value>/console/public/screens</param-value>
</config-param>
```

param-value 要素は、フィルター認証を必要としない URI を含みます。

また、特定のフォルダー内のすべてのファイルをフィルター認証からバイパスするには、param-value 要素内にバイパス URI としてコンテキスト・ルートからフォルダー・パスを指定します。

注: param-name 要素の値が「bypass.uri」ストリングから始まることを確認します。また、コンテキスト・ルートがプロパティー・ファイルに指定されていないことも確認します。コンテキスト・ルートは、EAR ファイルを構築しているときに動的に割り当てられます。

ユーザーのセッション・タイムアウトの設定

このタスクについて

ユーザーのセッション・タイムアウト間隔 (分単位) を設定するには、EARFILE/WARFILE/WEB-INF ディレクトリーにある web.xml ファイル内で config-param 要素のエントリーを追加します。以下に例を示します。

```
<session-config>
   <session-timeout>30</session-timeout>
</session-config>
```

注: web.xml ファイルで指定されたセッション・タイムアウト値が有効になるのは、対応するユーザーの YFS_USER テーブル内のセッション・タイムアウト間隔 (秒単位) が 0 以下である場合のみです。デフォルトでは、YFS_USER テーブルでは、各ユーザーのセッション・タイムアウト値 (秒単位) は 6000 に設定されています。

カスタム・ビジネス・エンティティーの作成

このタスクについて

カスタム・エンティティーを作成するには、「リソース階層」ツリーを介して標準エンティティーとそのサブリソースをコピーします。

注: Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales は、複数のエンティティーにまたがって同じビュー・グループ ID を持つリソースをサポートしていません。したがって、画面を拡張するためにエンティティー全体をコピーする場合は、新しいエンティティー用のすべてのビュー、アクション、リンク、およびアイコンのビュー・グループ ID を変更する必要があります。これにより、これらがコピー元のエンティティーと競合することを防止できます。

例えば、「計画済みオーダー」という名前のカスタム・ビジネス・エンティティーを作成する場合は、以下の手順を使用できます。

カスタム・ビジネス・エンティティーを作成するには、次の手順を実行します。

手順

1. 「リソース階層」ツリーから、テンプレートとして使用するエンティティーに移動します。作成するエンティティーとよく似たリソースとサブリソースを持つエンティティーを選択します。
2. 「**名前を付けて保存**」を選択して、エンティティーの新しいリソース・セットを作成します (サブリソースを含む)。

   そのエンティティーを保存する際に、将来のリリースで使用される可能性のあるどのようなリソース ID とも競合しない固有の接頭辞を付与してください。先頭

文字が **Y** ではない接頭辞を選択することをお勧めします (先頭文字が **Y** の接頭辞は当アプリケーション用に予約されています)。

3. 新しいエンティティー・リソースの説明を変更します。すべてのサブリソースの説明を変更します。これらのリソース説明はコンソールに表示されるため、必要に応じてローカライズできます。リテラルをローカライズできることを確認するには、ここで入力したリソース説明を、すべての適切なリソース・バンドルでリソース・バンドル・キーとして使用します。
4. これらの新たに作成されたリソース・キーのエントリーを、*INSTALL_DIR*/extensions/global/resources/extnbundle.properties ファイル内と、各ロケールの対応するプロパティー・ファイル内に作成します。

   注: 次のファイルが存在しないことを確認します。

   INSTALL_DIR/resources/extn/extnbundle.properties

   このファイルが削除される必要があるのは、このファイルはバンドル・エントリーの拡張機能ビルド・プロセスと競合するからです。
5. コピー元エンティティーの関連エンティティーについて、手順 1 から手順 3 を繰り返します。
6. 新たに作成されたエンティティー・リソース下の関連エンティティー・リソースを更新して、新たに作成されたリソースを指し示すようにします。
7. すべてのリンクが新しいビューを適切に指し示すようにするには、『アプリケーション全体にわたるカスタマイズ・ビューの導入』の手順に従ってください。

拡張データベース列の使用

データベースに追加された新規列をすべて取り込むようにビューをカスタマイズできます。

ユーザー・インターフェースにフィールドを追加するには、変更するビュー別に以下の指示に従います。

- 検索ビュー - 『JSP コンソールの検索ビューのカスタマイズ (Customizing a Search View in the JSP Console)』。
- リスト・ビュー - 『JSP コンソールの標準リスト・ビューのカスタマイズ (Customizing a Regular List View in the JSP Console)』および『JSP コンソールの拡張リスト・ビューのカスタマイズ (Customizing an Advanced List View in the JSP Console)』。データベース拡張性ルールを踏襲し、リソース階層ツリーを使用して、API に構成されている出力テンプレートにフィールドを追加します。
- 詳細ビュー - 『JSP コンソールの詳細ビューのカスタマイズ (Customizing Detail Views in the JSP Console)』。

  データベース拡張性ルールを踏襲し、リソース階層ツリーを使用して、API に構成されている出力テンプレートにフィールドを追加します。

エンティティー・キーのオーバーライド (Override Entity Key) 属性の使用

通常、最適の画面レイアウトでは、編集可能なレコード・リストの使用が必然になります。このテーブル形式は通常、画面の更新を処理する API の XML エレメントのリストにマップされます。例えば、「オーダーの詳細」画面のオーダー明細の編集可能リストは、getOrderDetails() API で受け入れられる OrderLine エレメントのリストにマップされます。

デフォルトでは、詳細ビューに表示されるアクションではすべて、アクションに対して呼び出される API への入力として現行エンティティー・キーが使用されます。例えば、「オーダーの詳細」画面の保留アクションでは、デフォルトで現行オーダー・ヘッダー・キーが changeOrder() API に渡されます。アクション・リソースでエンティティー・キーのオーバーライド (Override Entity Key) 属性を使用して、特定のアクションに使用されるエンティティー・キーをオーバーライドできます。JSP に入力 (通常は非表示の入力、またはチェック・ボックス) を構成するには、makeXMLInput JSP タグを使用して構成したエンティティー・キーの値を入力に指定し、その入力の名前をアクションのエンティティー・キー名として指定します。そのアクションがユーザーによって呼び出されると、新規キーが現行エンティティー・キーの代わりに渡されます。

この機能は、詳細ビューに 1 つのエンティティーの *詳細* が表示され、別のエンティティーのレコードの *リスト* を表示する内部パネルも含まれている場合に便利です。例えば、「オーダーの詳細」画面には、オーダー・エンティティーの詳細が表示され、(別のエンティティーである) オーダー明細のリストを示した内部パネルも含まれます。オーダー明細の内部パネルに表示されるアクションによって、オーダー・ヘッダー・キーが API に渡されることはありません。そのアクションによって、選択したオーダー明細のオーダー明細キーが渡されます。

例えば、オーダーのオーダー明細をリストする内部パネルに以下のコードが表示される場合があります。

```
<yfc:makeXMLInput name="orderLineKey">
    <yfc:makeXMLKey binding="xml:/OrderLineDetail/@OrderLineKey"
value="xml:/OrderLine/@OrderLineKey"/>
    <yfc:makeXMLKey binding="xml:/OrderLineDetail/@OrderHeaderKey"
value="xml:/Order/@OrderHeaderKey"/>
</yfc:makeXMLInput>
<td class="checkboxcolumn" >
     <input type="checkbox" value='<%=getParameter("orderLineKey")%>'
name="chkEntityKey"/>
</td>
```

このコードでは、オーダー明細に新規エンティティー・キーが作成され、このキーが chkEntityKey という名前のチェック・ボックスに関連付けられます。オーダー明細の内部パネルに表示されるいずれかのアクション、例えば、詳細の表示 (View Details) アクションのアクション定義で、この名前を指定できます。

この属性は、同じくアクションに定義できる選択キー名属性とともに使用されることが多いです。

編集可能リストのデータのポスト

この方法で要素のリストを受け取る API は、その API に対して要求される機能に基づいて異なる動作を示します。この異なる動作は、ユーザー・インターフェースによって処理される必要があります。

API が要素のリストを処理する方法の 1 つは、その API が呼び出されるたびにそのリスト全体を完全に置換することです。つまり、その API が呼び出されるたびに、ユーザー・インターフェースは各アイテムのすべての属性を渡す必要があります。このことを実現するために、IgnoreChangeNames() JavaScript 関数を使用します。JSP の読み込み時にこの関数を呼び出すことで、画面上のそれぞれの入力すべてが必ずポストされるようになります。

例えば、お使いの JSP ファイルを編集して次のコードを追加してください。

```
<script language="Javascript" >
IgnoreChangeNames();
</script>
```

場合によっては、API では、特定の属性が変更された場合にのみ特定のレコードがその API に渡されると想定されていることがあります。プレゼンテーション・フレームワークは、ユーザーによって変更されていない値を自動的にポストしないため、サービス定義フレームワークによって生成された XML には、特定のレコードのキー属性のみを持つ要素が含まれている可能性は十分にあります。この状況が起こり得る理由は、キー属性は通常は、HTML テーブルの各行に配置された JSP 内の非表示入力オブジェクトであるからです。これらは非表示入力であるため、常に API にポストされます。したがって、ユーザーがある行の特定のレコードのどの属性も変更していない場合は、キー属性のみが API に渡されます。一部の API では、これは無効な入力と見なされます。

プレゼンテーション・フレームワークの JavaScript 関数を使用すると、変更が加えられていないレコードが API にポストされないことを確認できます。該当するページがアンロードされるときに、yfcSpecialChangeNames() 関数が呼び出される必要があります。

例えば、次の JSP コードによってこのことが実現されます。

```
<script language=jscript>
window.document.body.attachEvent("onunload", processSaveRecordsForNotes)
</script>
```

この例で使用している JavaScript 関数は、次のように定義されています。

```
function processSaveRecordsForNotes() {
    yfcSpecialChangeNames("Notes", true);
}
```

この例では、対応する JSP 内の HTML テーブルの ID は、Notes® というリテラルに設定されています。 2 つ目のパラメーターである true が渡される必要があるのは、Notes という ID が新しい空白行からなる場合のみです。既存の行を変更しようとしている場合は、このパラメーターを false に設定する必要があります。

編集可能リスト内の未保存データの保持

一部の画面には、ユーザーが複数行のデータを入力できる編集可能なテーブルがあります。このデータは通常、ユーザーが「保存」ボタンをクリックして、保存 API が呼び出されると保存されます。デフォルトでは、保存 API が例外をスローした場合は、この画面が最新表示になり、編集可能なテーブルに入力されたデータは保持されません。ここでは、API 例外が発生した場合でも、未保存のデータを保持する方法を説明します。

次の例では、この機能を「オーダー指示」画面に実装する方法を説明しています。この画面は、この機能を使用可能にするためにカスタマイズされた次の 2 つの JSP ファイルで構成されています。

- order_detail_instructions_anchor.jsp
- order_detail_instructions.jsp

order_detail_instructions_anchor.jsp

```
<%@include file="/yfsjspcommon/yfsutil.jspf"%>
<%@include file="/console/jsp/order.jspf" %>

<% setHistoryFlags((YFCElement) request.getAttribute("Order")); %>

<table class="anchor" cellpadding="7px"  cellSpacing="0" >
<tr>
    <td >
        <jsp:include page="/yfc/innerpanel.jsp" flush="true" >
            <jsp:param name="CurrentInnerPanelID" value="I02"/>
            <jsp:param name="getRequestDOM" value="Y"/>
        </jsp:include>
    </td>
</tr>
<tr>
    <td >
        <jsp:include page="/yfc/innerpanel.jsp" flush="true" >
            <jsp:param name="CurrentInnerPanelID" value="I01"/>
            <jsp:param name="ChildLoopXMLName" value="Instruction"/>
            <jsp:param name="ChildLoopXMLKeyName"
                value="InstructionDetailKey"/>
        </jsp:include>
    </td>
</tr>
```

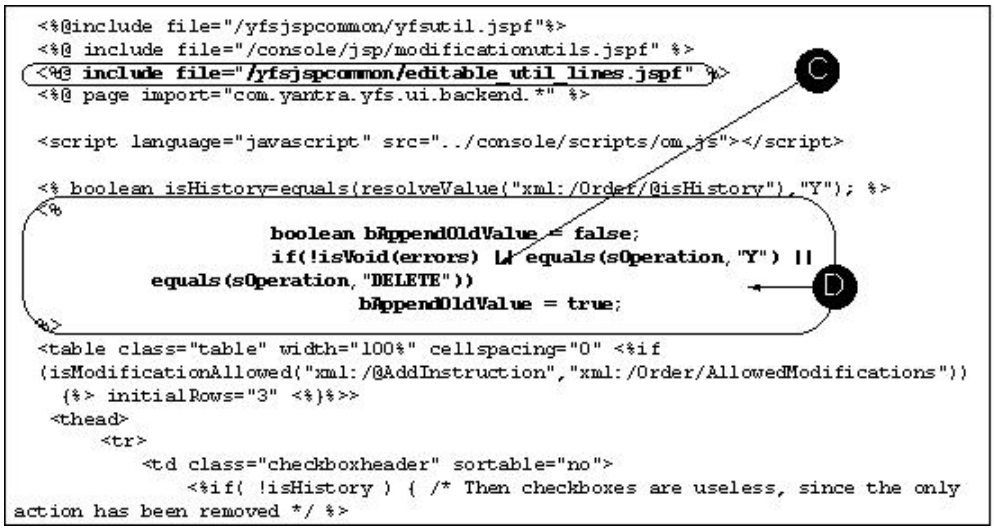
次の表では、図中の丸枠で囲まれた箇所について説明しています。

丸枠箇所	説明
A	getRequestDOM という JSP パラメーターが "Y" という値としてヘッダー JSP (この例では order_detail_header.jsp) に渡されていることに注目してください。すべての未保存行は、XML ファイルとしての要求オブジェクト内に保存されます。このパラメーターを渡すことで、未保存の行が要求オブジェクトから取得されて、画面を最初に読み込むために使用される API 出力とマージされます。最新表示になった画面には、ユーザーが入力した新たに変更された値が表示されます。
B	マージ・ロジックが正しく動作するには、以下のパラメーターを渡して、マージが実行される要素と属性を指定する必要があります。次の表を参照してください。

渡される必要のあるパラメーターは次のとおりです。

パラメーター名	デフォルト値	コメント
RootNodeName	オーダー	新たに入力された値のマージ先である出力 XML のルート・ノード。
ChildLoopXMLName	OrderLine	編集可能リスト内の繰り返し要素名を表す XML 要素。
ChildLoopXMLKeyName	OrderLineKey	繰り返し XML 要素を一意的に識別するキー。マージ・ロジックはこのキーを使用して、指定された要素のデータがユーザーによって変更されたかどうかを判別します。

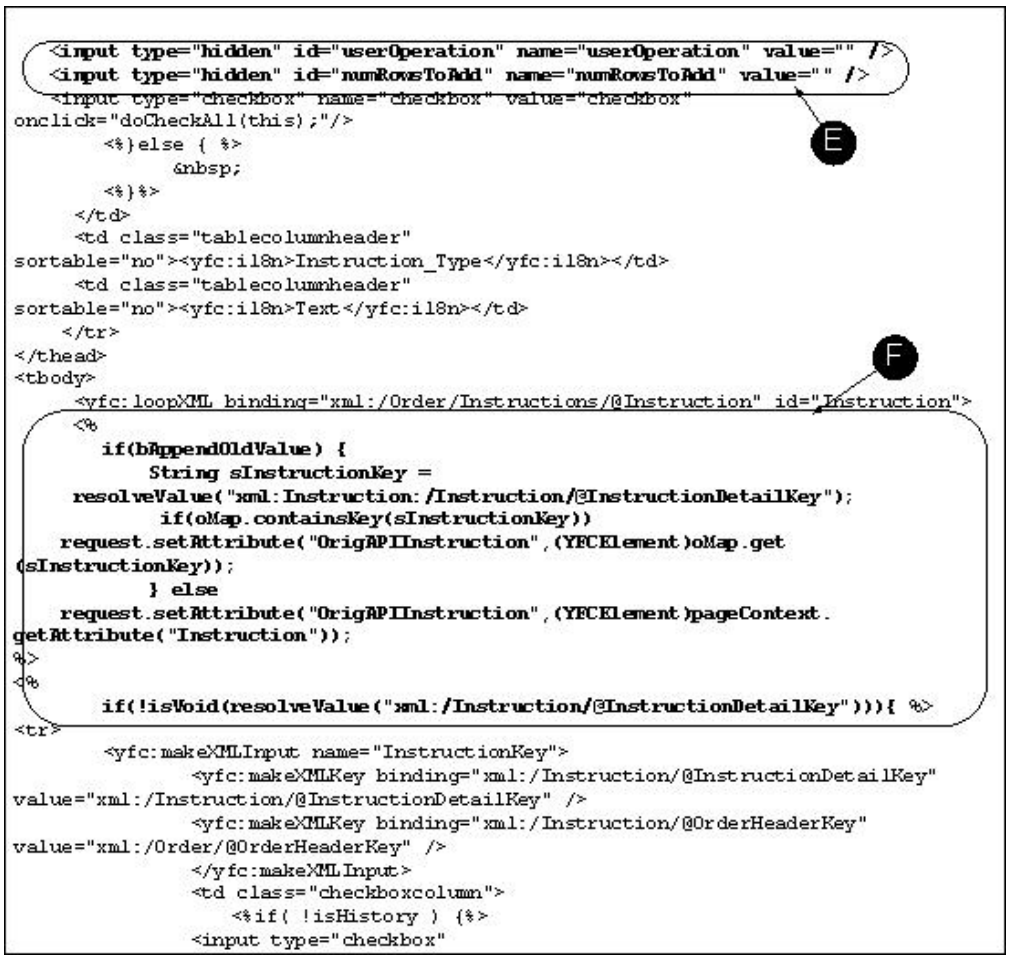
次の図で強調表示されたコードは、API 例外が発生した後もデータが保持されることを保証する変更箇所を示しています。




```
<%@include file="/yfsjspcommon/yfsutil.jspf"%>
<%@ include file="/console/jsp/modificationutils.jspf" %>
<%@ include file="/yfsjspcommon/editable_util_lines.jspf" %>
<%@ page import="com.yantra.yfs.ui.backend.*" %>

<script language="javascript" src="../console/scripts/om.js"></script>

<% boolean isHistory=equals(resolveValue("xml:/Order/@isHistory"),"Y"); %>
<%
                    boolean bAppendOldValue = false;
                    if(!isVoid(errors) || equals(sOperation,"Y") ||
          equals(sOperation,"DELETE"))
                        bAppendOldValue = true;
%>
<table class="table" width="100%" cellspacing="0" <%if
(isModificationAllowed("xml:/@AddInstruction","xml:/Order/AllowedModifications"))
  {%> initialRows="3" <%}%>>
 <thead>
     <tr>
         <td class="checkboxheader" sortable="no">
             <%if( !isHistory ) { /* Then checkboxes are useless, since the only
action has been removed */ %>
```




```
    <input type="hidden" id="userOperation" name="userOperation" value="" />
    <input type="hidden" id="numRowsToAdd" name="numRowsToAdd" value="" />
    <input type="checkbox" name="checkbox" value="checkbox"
onclick="doCheckAll(this);"/>
        <%}else { %>
               
        <%}%>
      </td>
      <td class="tablecolumnheader"
sortable="no"><yfc:i18n>Instruction_Type</yfc:i18n></td>
      <td class="tablecolumnheader"
sortable="no"><yfc:i18n>Text</yfc:i18n></td>
    </tr>
</thead>
<tbody>
      <yfc:loopXML binding="xml:/Order/Instructions/@Instruction" id="Instruction">
      <%
        if(bAppendOldValue) {
            String sInstructionKey =
      resolveValue("xml:Instruction:/Instruction/@InstructionDetailKey");
             if(oMap.containsKey(sInstructionKey))
     request.setAttribute("OrigAPIInstruction",(YFCElement)oMap.get
(sInstructionKey));
            } else
     request.setAttribute("OrigAPIInstruction",(YFCElement)pageContext.
getAttribute("Instruction"));
%>
<%
        if(!isVoid(resolveValue("xml:/Instruction/@InstructionDetailKey"))){ %>
<tr>
        <yfc:makeXMLInput name="InstructionKey">
                <yfc:makeXMLKey binding="xml:/Instruction/@InstructionDetailKey"
value="xml:/Instruction/@InstructionDetailKey" />
                <yfc:makeXMLKey binding="xml:/Instruction/@OrderHeaderKey"
value="xml:/Order/@OrderHeaderKey" />
                </yfc:makeXMLInput>
                <td class="checkboxcolumn">
                   <%if( !isHistory ) {%>
                <input type="checkbox"
```

```
 value='<%=getParameter("InstructionKey")%>' name="chkEntityKey"/>
        <%}else { %>
                 
        <%}%>
  </td>
  <td class="tablecolumn">
        <% String instTargetBinding =
"xml:/Order/Instructions/Instruction_" + InstructionCounter +
"/@InstructionType"; %>
        <select <%if(bAppendOldValue) {
%>OldValue="<%=resolveValue("xml:OrigAPIInstruction:/Instruction/
@InstructionType")%>" <%}%>
<%=yfsGetComboOptions(instTargetBinding, "xml:/Instruction/@InstructionType",
"xml:/Order/AllowedModifications")%>>
                        <yfc:loopOptions
binding="xml:InstructionTypeList:/CommonCodeList/@CommonCode"
name="CodeShortDescription" value="CodeValue"
selected="xml:/Instruction/@InstructionType" isLocalized="Y"
targetBinding="<%=instTargetBinding%>"/>
        </select>
                </td>
                <td class="tablecolumn">
                   <table class="view" cellspacing="0" cellpadding="0">
   <tr>
                <td>
                        <textarea rows="3" cols="100
         <%if(bAppendOldValue) {
%>OldValue="<%=resolveValue("xml:OrigAPIInstruction:/Instruction/
@InstructionText")%>"   <%}%>
<%=yfsGetTextAreaOptions("xml:/Order/Instructions/Instruction_" +
InstructionCounter + "/@InstructionText","xml:/Instruction/@InstructionText",
"xml:/Order/AllowedModifications")%>><yfc:getXMLValue binding="xml:/Instruction/
@InstructionText"/></textarea>
                </td>
     </tr>
     <tr>
                <td>
                        <img align="absmiddle"
<%=getImageOptions(YFSUIBackendConsts.INSTRUCTION_URL, "Instruction_URL")%>/>
                        <input type="text" <%if(bAppendOldValue) {
%>OldValue="<%=resolveValue("xml:OrigAPIInstruction:/Instruction/
@InstructionURL")%>"   <%}%>
```

```
<%=yfsGetTextOptions("xml:/Order/Instructions/Instruction_" +
InstructionCounter + "/@InstructionURL",
"xml:/Instruction/@InstructionURL","xml:/Order/AllowedModifications")%>/>
                        <input type="button" class="button" value="GO"
onclick="javascript:goToURL('xml:/Order/Instructions/
Instruction_<%=InstructionCounter%>/@InstructionURL');"/>
                </td>

                <td>
                        <input type="hidden"
<%=getTextOptions("xml:/Order/Instructions/Instruction_" + InstructionCounter
+ "/@InstructionDetailKey", "xml:/Instruction/@InstructionDetailKey")%>/>
                </td>
        </tr>
        <tr>
                <td> </td>
                <td> </td>
                <td> </td>
                <td> </td>
        </tr>
     </table>
   </td>
</tr>
```

```
<%      } else { %>
                <tr DeleteRowIndex="<%=InstructionCounter%>">
                        <td class="checkboxcolumn">
                                <img class="icon"
onclick="setDeleteOperationForRow(this,'xml:/Order/Instructions/Instruction')"
<%=getImageOptions(YFSUIBackendConsts.DELETE_ICON, "Remove_Row")%>/>
                        </td>
                        <td class="tablecolumn">
                                <input type="hidden" OldValue=""
<%=getTextOptions("xml:/Order/Instructions/Instruction_
'+InstructionCounter+"/@Action", "xml:/Order/Instructions/Instruction_
'+InstructionCounter+"/@Action", "CREATE")%> />
                                <input type="hidden"
<%=getTextOptions("xml:/Order/Instructions/Instruction_
'+InstructionCounter+"/@DeleteRow",  "")%> />
                                <select OldValue=""
<%=yfsGetTemplateRowOptions("xml:/Order/Instructions/Instruction_
'+InstructionCounter+"/@InstructionType", "xml:/Instruction/@InstructionType",
'xml:/Order/AllowedModifications", "ADD_INSTRUCTION", "combo")%>>
                                <yfc:loopOptions
binding="xml:InstructionTypeList:/CommonCodeList/@CommonCode"
name="CodeShortDescription" value="CodeValue" isLocalized="Y"
selected="xml:/Instruction/@InstructionType"/>
                                </select>
                        </td>
                        <td class="tablecolumn">
                                <table class="view" cellspacing="0" cellpadding="0">
                                        <td>
                                        <textarea rows="3" cols="100" OldValue=""
<%=yfsGetTemplateRowOptions("xml:/Order/Instructions/Instruction_
"+InstructionCounter+"/@InstructionText", "xml:/Instruction/@InstructionText",
"xml:/Order/AllowedModifications", "ADD_INSTRUCTION",
"textarea")%>></textarea>
                        </td>
                        <tr>
                                <td>
                                 <img align="absmiddle"
<%=getImageOptions(YFSUIBackendConsts.INSTRUCTION_URL, "Instruction_URL")%>/>
<input type="text" OldValue=""
<%=yfsGetTemplateRowOptions("xml:/Order/Instructions/Instruction_
"+InstructionCounter+"/@InstructionURL", "xml:/Instruction/@InstructionURL",
"xml:/Order/AllowedModifications", "ADD_INSTRUCTION", "text")%>/>
```

H

```
                        </td>
                                </tr>
                                <tr>
                                        <td> </td>
                                        <td> </td>
                                        <td> </td>
                                        <td> </td>
                                </tr>
                        </table>
                </td>
        </tr>
   <%   } %>
   </yfc:loopXML>
</tbody>
<tfoot>
        <%if (isModificationAllowed("xml:/@AddInstruction","xml:/Order/
AllowedModifications")) {%>
        <tr>
                <td nowrap="true" colspan="3">
                        <jsp:include page="/common/editabletbl.jsp" flush="true">
                        <jsp:param name="ReloadOnAddLine" value="Y"/>
                        </jsp:include>
                </td>
        </tr>
        <%}%>
</tfoot>
</table>
```

I

次の表では、図中の丸枠で囲まれた箇所について説明しています。

丸枠箇所	説明
C	/webpages/yfsjspcommon/editable_util_lines.jspf ファイルは、リスト innerpanel.jsp にインクルードされました。 このファイルは、ユーザーによって変更された値と元の API 出力をマージします。
D	行の削除が適切に処理されることを保証するために JSP コードが追加されました。
E	ユーザー操作と追加行数について、非表示入力が追加されました。
F	元の繰り返し XML 要素の固有キーのマップが追加されました。
G	それぞれの編集可能フィールドの OldValue 属性が、その前に作成されたマップを使用して元の API出力の値に設定されます。これにより、ユーザーによって変更された値が確実に保存 API にポストされるようになります。
H	固有のキー属性が含まれていない繰り返し XML 要素ごとに、1 つの新しい行が編集可能テーブル内に作成されます。これは、その行がデータベースにまだ保存されていないこと、および API 例外の発生前にユーザーによって入力されたことを示します。
I	ユーザーがテーブルの下のプラス・アイコンをクリックしたときに行を動的に追加するために、/common/editabletbl.jsp ファイルが次にインクルードされます。ReloadOnaddLine 属性の値として "Y" をこの JSP に渡します。新しい行が追加されるたびに、画面が最新表示になります。これらの変更を加える前のコードでは、新しい行が動的に追加されても画面は最新表示になりません。

ルックアップの追加

ルックアップを使用すると、ユーザーはデータに入力するのではなくオプションのリストを選択できます。 次の図は、使用可能なルックアップ・アイコンとそれに関連付けられているフィールドを示しています。

図 2. ルックアップ・アイコン

プレゼンテーション・フレームワークは、以下のタイプのルックアップをサポートします。

- 単一フィールド・ルックアップ - ユーザーが特定の値のエンティティーを検索し、その値を選択し、それを適切な単一入力フィールドに挿入できるようにします。callLookup() JavaScript 関数を使用します。「callLookup」を参照してください。
- カレンダー・ルックアップ - ユーザーがポップアップ・カレンダーから日付を選択できるようにします。invokeCalendar() JavaScript 関数を使用します。「invokeCalendar」を参照してください。

複数フィールド・ルックアップが必要な場合、yfcShowSearchPopup() JavaScript 関数を使用できます。この例では、アイテム・ルックアップから yfcShowSearchPopup() JavaScript 関数を使用して入力する製品クラス (product class)、アイテム ID、および計測単位を示します。

```
//this function should be called from "onclick" event of the icon next to
//item id field.
function callItemLookup(sItemID,sProductClass,sUOM,entityname)
{
    var oItemID = document.all(sItemID);
    var oProductClass = document.all(sProductClass);
    var oUOM = document.all(sUOM);
    showItemLookupPopup(oItemID, oProductClass, oUOM, entityname);
}
function showItemLookupPopup(itemIDInput, pcInput, uomInput, entityname)
{
    var oObj = new Object();
    oObj.field1 = itemIDInput;
    oObj.field2 = pcInput;
    oObj.field3 = uomInput;
    yfcShowSearchPopup('','itemlookup',900,550,oObj,entityname);
}
```

ルックアップ・リスト・ビューで、ポップアップが呼び出されたフィールドに入力する以下の関数を呼び出します。

```
function setItemLookupValue(sItemID,sProductClass,sUOM)
{
    var Obj = window.dialogArguments
    if(Obj != null)
    {
        Obj.field1.value = sItemID;
        Obj.field2.value = sProductClass;
        Obj.field3.value = sUOM;
    }
    window.close();
}
```

「ルックアップ」アイコンは、変更可能フィールドとのみ使用します。特定のタイプのルックアップをフィールドに取り込む場合は、適切なアイコンをルックアップの右側に直接配置します。

ユーザーによってソート可能なテーブルの作成

このタスクについて

どのテーブルでも、ユーザーは列見出しをクリックすることで結果をソートできます。ユーザーが同じ列見出しをもう一度クリックすると、結果は逆順でソートされます。

テーブルのソートを行うと、新しい API 呼び出しが生成されることなく、選択された列に表示されているデータがソートされます。

ユーザーによってソート可能なテーブルを作成するには、次の手順を実行します。

手順

1. 対象のテーブルの style 属性で table.htc を使用します。Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales のデフォルト CSS ファイルを使用している場合は、<table> タグで class="table" を使用できます。
2. そのテーブルでは、<td> タグが含まれた <tbody> タグと <thead> タグが使用されている必要があります。 <thead> タグ内のいずれかの <td> タグで sortable="no" を指定した場合は、その列はソート可能にはなりません。
3. 日付と数値については、実際の <tbody> タグ内で別個の sortValue="<nonlocalized_value>" を指定して、そのデータが適切にソートされるようにします。

   この例では、ユーザーによってソート可能なテーブルを作成するために必要なタグを示しています。

```
<table class="table" editable="false" width="100%" cellspacing="0">
 <thead>
  <tr>
   <td sortable="no" class="checkboxheader">
     <input type="hidden" name="userHasOverridePermissions"
value='<%=userHasOverridePermissions()%>'/>
     <input type="hidden" name="xml:/Order/@Override" value="N"/>
     <input type="checkbox" name="checkbox" value="checkbox"
onclick="doCheckAll(this);"/>
   </td>
   <td class="tablecolumnheader"><yfc:i18n>Order_#</yfc:i18n></td>
   <td class="tablecolumnheader"><yfc:i18n>Enterprise</yfc:i18n></td>
   <td class="tablecolumnheader"><yfc:i18n>Buyer</yfc:i18n></td>
   <td class="tablecolumnheader"><yfc:i18n>Order_Date</yfc:i18n></td>
  </tr>
 </thead>
 <tbody>
     <yfc:loopXML binding="xml:/OrderList/@Order" id="Order">
        <tr>
          <yfc:makeXMLInput name="orderKey">
               <yfc:makeXMLKey binding="xml:/Order/@OrderHeaderKey"
value="xml:/Order/@OrderHeaderKey" />
          </yfc:makeXMLInput>
          <td class="checkboxcolumn">
             <input type="checkbox"
value='<%=getParameter("orderKey")%>'           name="EntityKey"/>
          </td>
          <td class="tablecolumn">
             <a href="javascript:showDetailFor
('<%=getParameter("orderKey")%>');">
             <yfc:getXMLValue binding="xml:/Order/@OrderNo"/></a>
           </td>
           <td class="tablecolumn">
             <yfc:getXMLValue binding="xml:/Order/@EnterpriseCode"/>
           </td>
           <td class="tablecolumn">
             <yfc:getXMLValue binding="xml:/Order/@BuyerOrganizationCode"/>
            </td>
            <td class="tablecolumn"
SortValue="<%=getDateValue("xml:/Order/@OrderDate")%>">
              <yfc:getXMLValue binding="xml:/Order/@OrderDate"/>
            </td>
        </tr>
     </yfc:loopXML>
 </tbody>
</table>
```

グラフおよび円グラフの追加

グラフおよび円グラフを使用すると、ユーザーがデータのグラフィカル表現を表示できます。グラフおよび図表は、テーマからルック・アンド・フィール (カラーおよびフォントの表示) を派生させます。テーマおよび変更方法の詳細手順については、「集中テーマについて」を参照してください。

データのグラフを表示するため、プレゼンテーション・フレームワークは Visual JChart から jbchartx.jar ファイルを使用します。このツールは将来のバージョンで変更される可能性があるため、どのグラフ作成ツールを使用するかは、独自の評価を実行してください。プレゼンテーション・フレームワークは、FusionCharts と統合してグラフを生成します。

FusionCharts を使用する理由

グラフを生成するために FusionCharts と統合することには様々な利点があります。いくつかの利点は、以下のとおりです。

- 従来のグラフ作成コンポーネントによってレンダリングされる静的イメージと異なり、アニメーション作成および対話式グラフをレンダリングします。
- ブラウザー間およびクロスプラットフォーム互換を可能にする XML データ・インターフェースを使用します。これは任意のスクリプト言語およびデータベースで使用できます。
- 狭い帯域幅の接続でも、動的および対話式グラフ作成に適した小さいサイズです。
- シン・クライアントの動作を高める Macromedia Flash Player を利用します。
- クライアント上でサーバー要求を呼び出すことなく、グラフのタイプおよびデータを動的に変更できるようにします。

メニュー構成のカスタマイズ

カスタマイズ画面を作成するときには、ユーザーがメニュー構成またはナビゲーションのいずれかを使用してアクセスできることを検証してください。

メニューのカスタマイズでは、最初に Applications ManagerConfigurator のグラフィカル・ユーザー・インターフェース (graphical user interface) を使用してメニューの構造をレイアウトしてから、リソース・バンドル内にリテラルを指定することが必要です。リソース・バンドルは、スクリーン内のリテラルおよびメッセージのすべてを含むファイルです。Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales は、以下の標準リソース・バンドルです。

- ycpapibundle.properties - 標準メニューおよびスクリーン内のメッセージで使用されるリテラルを含みます。これは変更できません。
- extnbundle.properties - カスタム・メニューで使用されるリテラルを含みます。これは必要に応じてカスタマイズおよびローカライズできます。

カスタム・メニューに行った変更を含むカスタム・リソース・バンドルを作成することができます。

注: メニューをカスタマイズするときには、メニュー説明が Java 標準で定義されているように有効なリソース・バンドル・キーになることができないスペースまたはその他の文字を含まないことを検証してください。

カスタム・メニューの作成

このタスクについて

カスタム・メニューを作成するには、次の手順を実行します。

手順

1. 「Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales:*構成ガイド*」で説明されているグラフィカル・ユーザー・インターフェースを使用して、新しいメニューを作成します。
2. *INSTALL_DIR*/extensions/global/resources/extnbundle.properties ファイルを開いて、等号 (=) を使用した「キー=説明」マッピングを追加して、手順 1 で追加した説明をキーと関連付けます。例えば、`Special_Tasks=Special Tasks` のようにします。当アプリケーションは、画面に表示する内容を決定するためにこのキーを読み取ります。

   注: 次のファイルが存在しないことを確認します。

   INSTALL_DIR`/resources/extn/extnbundle.properties`

   このファイルが削除される必要があるのは、このファイルはバンドル・エントリーの拡張機能ビルド・プロセスと競合するからです。

メニュー構造のローカライズ

このタスクについて

単一インストールで複数言語がサポートされるようにリソース・バンドルをカスタマイズできます。アプリケーション・コンソールをローカライズするときに、リソース・バンドルに追加するメニュー・キーの前に `MENU_` を付加します。ローカライズについて詳しくは、「*ローカライズ・ガイド*」を参照してください。

注: メニューをローカライズする際には、Java 標準で定義されているように、有効なリソース・バンドルとして指定できないスペースなどの文字がメニューの説明に含まれていないことを確認してください。

画面ナビゲーションのカスタマイズ

リンクまたはアクションのリソースを構成することによって、ユーザーがエンティティーからエンティティーにナビゲートする方法をカスタマイズすることができます。リンクおよびアクションは、任意のエンティティーの任意の詳細ビューをポイントすることができます。ナビゲートしているエンティティーが異なるエンティティー・キーを要求する場合があります。

例えば、「オーダーの詳細」画面には、オーダー明細のリストがあります。ユーザーは、「オーダー明細番号」ハイパーリンクをクリックするか、またはオーダー明細を選択し、オーダー明細の内部パネルにある「詳細の表示」アクションをクリックすることによって、オーダー明細の詳細画面にナビゲートできます。オーダー明

細の詳細画面では、「オーダーの詳細」画面とは異なるエンティティー・キーが必要であるため、オーダー明細の詳細画面で使用される別のエンティティー・キーをオーダー明細の JSP 内に作成する必要があります。

例 10 から 3 は、画面ナビゲーション動作を指定するためのサンプル・コードを示しています。

```
 <yfc:makeXMLInput name="orderLineKey">
     <yfc:makeXMLKey binding="xml:/OrderLineDetail/@OrderLineKey"
value="xml:/OrderLine/@OrderLineKey"/>
     <yfc:makeXMLKey binding="xml:/OrderLineDetail/@OrderHeaderKey"
value="xml:/Order/@OrderHeaderKey"/>
     </yfc:makeXMLInput>
```

このエンティティー・キーがオーダー明細の詳細画面に渡される方法は、ハイパーリンクおよびアクション経路とは異なります。ハイパーリンクの場合、このキーは、以下のように getDetailHrefOptions() 関数に渡されます。

```
 <a <%=getDetailHrefOptions("L03", getParameter("orderLineKey"), "")%>>
    <yfc:getXMLValue binding="xml:/OrderLine/@PrimeLineNo"/></a>
```

いくつかの入力パラメーターに基づいて、異なる動作を詳細ビューにさせたい場合があります。例えば、詳細ビューを呼び出しているパラメーターに基づいて詳細ビュー内のフィールドを表示または非表示にする場合があります。

ターゲット詳細ビューに渡される追加パラメーターを使用して、getDetailHrefOptions() JSP 関数を呼び出すことができます。

これらの追加パラメーターを渡すために必要な形式には、アンパーサンド「&」記号で区切った名前と値のペアが必要です。これは、URL 内でパラメーターを渡すための標準形式です。

詳細の表示アクションでは、「選択キー名」を、JSP 内で作成されたチェック・ボックスの名前に設定する必要があります。例えば、この JSP では、チェック・ボックスは次のように作成できます。

```
<input type="checkbox" value='<%=getParameter("orderLineKey")%>'
name="chkEntityKey"/>
```

チェック・ボックスの名前は chkEntityKey です。リソース階層ツリー内でアクションを構成するときには、「選択キー名」をこれに設定して、正しいキーがオーダー明細の詳細画面に渡されることを確実にします。

詳細画面へのダイレクト・ナビゲーションの無効化

このタスクについて

アプリケーションによって表示される「オーダー」、「購入オーダー」、または「返品の検索」画面で、検索結果が 1 つのみのレコードの場合、ユーザーはその 1 つのレコードのデフォルトの詳細ビューにダイレクトされます。これによって、特定のレコードの詳細の表示に必要なクリック数が軽減されます。

例えば、アプリケーションによって表示される「オーダーの検索」画面からは、固有のオーダー番号を入力して「検索」を押すと、「オーダーの詳細」画面に直接ナビゲートされます。

このタイプのナビゲーションは、リソース階層ツリーのビュー定義で制御されます。特定のユーザー・インターフェース・エンティティーのみが、このタイプのナビゲーションをサポートします。このため、エンティティー・リソースの「リソースの詳細」には、特定のエンティティーがこの機能をサポートするかどうかを示す「1 つのレコードが返された場合のリストから詳細へのダイレクト・ナビゲーションをサポートする (Support Direct List To Detail Navigation with One Record Returned)」チェック・ボックスがあります。この機能をサポートするエンティティーの場合は、そのエンティティーに定義されているリスト・ビューで、「1 つのレコードが返された場合のリストから詳細へのダイレクト・ナビゲーションをサポートする (Support Direct List To Detail Navigation with One Record Returned)」チェック・ボックスを有効または無効にすることによって、このナビゲーションのオンとオフを切り替えることができます。

この機能を無効にするには、既存のリスト・ビューのコピーを作成し、このチェック・ボックスのチェック・マークを外して、リソース・シーケンスが既存のリスト・ビューより下位になっていることを確認してください。

第 11 章 JSP コンソール内のイベント・ハンドラーのカスタマイズ

JSP コンソール内のイベント・ハンドラーについて

以下のイベントのユーザー・インターフェース・コントロールに対してクライアント・サイド検証を作成してプラグインすることができます。

- そのコントロールまたは画面に対して Internet Explorer によって発生したイベント。例えば、入力に対する onblur イベントまたはページに対する onunload イベント。
- ページに対してサービス定義フレームワークによって発生したイベント。例えば、詳細ビューにデータを保存する前、または検索ビューで検索を呼び出す前。

制御レベル・イベント・ハンドラー

各テキスト・ボックスには、onblur イベント (失われたフォーカス) に検証メソッドを動的に付加する動作クラスが関連付けられています。

各 XML 属性は、データ型に結合されている必要があり、インフラストラクチャーは、データ要素がバインドされる XML 属性に基づいてデータ型を決定します。このデータ型は、データ検証に使用されます。例えば、数値フィールドは数値エントリーのみを受け入れます。

クライアント・サイド・フィールド・レベル検証を最小にすることを推奨します。onblur="myValFun();" を HTML ページ内で直接使用して、カスタム検証を実行することができます。ただし、関数がプレゼンテーション・フレームワーク関数より前に呼び出されることは保証されません。したがって、数値フィールドまたは日付 (date) フィールドを使用している場合は、関数は無効なデータを返す場合があります。検証を行う前に、プレゼンテーション・フレームワーク JavaScript ユーティリティー関数を呼び出して、最初に日付および数値を検証する必要があります。

画面レベルのイベント・ハンドラー

プレゼンテーション・フレームワークは、document.body で onload イベントを使用します。他のすべてのイベントは、開発者が使用できます。プレゼンテーション・フレームワークは、attachEvent() 関数を使用して、イベント・ハンドラーを他のイベント (入力の onblur など) に動的に関連付けます。したがって、作成する JSP コードでは、他の関数を使用できます。独自の onload 関数を呼び出すことを希望する場合でも、次のように JSP 内の script タグ内で attachEvent() 関数を使用できます。

```
<script language=jscript src="/extensions/global/webpages/scripts/om.js">
</script>
<script language=jscript>
window.document.body.attachEvent("onload", myFunc)
</script>
```

これにより、この HTML の読み込み時に myFunc() 関数が実行されるようになります。なお、myFunc 関数の本体は *INSTALL_DIR*/extensions/global/webpages/scripts/om.js ファイル内に存在する必要があります。

プレゼンテーション・フレームワークは、「保存」アイコンがクリックされると、詳細ビューの「保存」アクションを呼び出します。このイベント用の独自のカスタム・イベント・ハンドラーを組み込むことを希望する場合は、このアクションによって独自に作成した JavaScript 関数が呼び出されるようにこのアクションを構成します。この関数は、*INSTALL_DIR*/extensions/global/webpages/scripts/extn.js ファイル内に存在する必要があります。

プレゼンテーション・フレームワークは、検索ビューで「検索」アイコンがクリックされると、リスト・ビューを呼び出します。このイベント用の独自のカスタム・イベント・ハンドラーを組み込むことを希望する場合は、そのページの読み込み時に yfcGetSearchHandle() JavaScript API によって返されるオブジェクトの onclick イベントに、独自に作成した JavaScript 関数を関連付けます。次の例では、このことを実現する方法を示しています。

```
<script language=jscript src="/extensions/global/webpages/scripts/om.js">
</script>
<script language=jscript>
//Get the handle to search button.
 var oObj = yfcGetSearchHandle();
//The setParentKey function is defined inside om.js.
var sVal = oObj.attachEvent("onclick",setParentKey);
</script>
```

フィールド・レベルの検証の作成

このタスクについて

フィールド・レベルの検証を作成するには、次の手順を実行します。

手順

1. ビューをカスタマイズします。
2. カスタマイズした JSP に変更を加えて、対応するコントロールの `onblur` イベントのイベント・ハンドラーを組み込みます。
3. JavaScript 関数の本体を別個の JS ファイルに格納して、その JS ファイルを JSP にインクルードします。
4. 要求に合う「キャッシュの更新」アイコンを選択します。以下に例を示します。
   - 1 つのエンティティーとその子リソースを更新するには、そのエンティティーを選択して、  「エンティティー・キャッシュの更新」アイコンをクリックします。
   - すべてのリソースを更新するには、  「キャッシュの更新」アイコンをクリックします。
5. 当アプリケーションに再ログインして、変更内容をテストします。

画面レベルの検証の作成

このタスクについて

画面レベルの検証を作成するには、次の手順を実行します。

手順

1. 検証の組み込み先となるビューをカスタマイズします。
2. カスタマイズされた JSP に変更を加えて、attachEvent を組み込みます。
3. 要求に合う「キャッシュの更新」アイコンを選択します。以下に例を示します。
   - 1 つのエンティティーとその子リソースを更新するには、そのエンティティーを選択して、  「エンティティー・キャッシュの更新」アイコンをクリックします。
   - すべてのリソースを更新するには、  「キャッシュの更新」アイコンをクリックします。
4. 当アプリケーションに再ログインして、変更内容をテストします。

第 12 章 伝票種別および要求レコードの操作

伝票種別の処理

デフォルトのオーダー・コンソールでは、オーダー伝票種別 (0001) が使用されます。新規伝票種別を作成する場合、ビューを伴う新規エンティティーを作成する必要もあります。リソース・タイプ・エンティティーのリソースの場合、伝票種別を指定できます。

I18N タグ内のリテラルは、特定の順序で解決されます。最初に、キーに伝票種別の接頭辞が付けられ、キーがリソース・バンドル内で検索されます。一致が見つからないと、キーはそのまま、つまり接頭辞なしで検索されます。

例えば、Order_# という I18N リテラルがあり、現行伝票種別がオーダー (0001) である場合、アプリケーションではまずリソース・バンドルで 0001_Order_# のエントリーが検索されて、リソース・キーの解決が試行されます。見つからないと、Order_# が検索されます。このスキームによって、特定の JSP を再使用できるようになる一方で、引き続き、リテラルが伝票種別に固有の場合に、画面に表示されるそれらのリテラルを変更できます。

新しい伝票種別の新しい画面セットの作成

このタスクについて

新しい伝票種別の新しい画面セットを作成するには、次の手順を実行します。

手順

1. 「リソース階層」ツリーから、オーダー・エンティティー (リソース ID オーダー) に移動します。
2. 「**名前を付けて保存**」を選択して、オーダー・エンティティー (そのサブリソースも含む) から新しいリソース・セットを作成します。

   そのエンティティーを保存する際に、将来のリリースで使用される可能性のあるどのようなリソース ID とも競合しない固有の接頭辞を付与してください。先頭文字が Y ではない接頭辞を選択することをお勧めします (先頭文字が Y の接頭辞は Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales 用に予約されています)。

   新しい伝票種別のリソースをコピーする際は、既存の伝票種別のすべてのエンティティー (サブリソースを含む) をコピーする必要があります。

   a. 次の SQL スクリプトを使用して、コピーするリソースを特定します。

      ```
      select resource_id from yfs_resource
      where resource_type='ENTITY' and document_type='0001'
      ```

   b. ビューを呼び出す JavaScript 関数、リンク、アクション、およびアイコンのすべてのビュー・グループ ID を変更します。ビュー・グループ ID は、新しいエンティティー接頭辞と既存のビュー・グループ ID を組み合わせたものに変更する必要があります。

3. 新しいエンティティー・リソースの説明を、新しい伝票種別の説明に変更します。すべてのサブリソースについても、それらの説明に適切な変更を加えて、これらの新たに作成されたリソース・キーのエントリーを *INSTALL_DIR*/extensions/global/resources/extnbundle.properties ファイル内に作成するとともに、各ロケールの対応するプロパティー・ファイル内にも作成します。

   注: 次のファイルが存在しないことを確認します。

   INSTALL_DIR/resources/extn/extnbundle.properties

   このファイルが削除される必要があるのは、このファイルはバンドル・エントリーの拡張機能ビルド・プロセスと競合するからです。
4. リソース・タイプ・エンティティーの新しいリソースの伝票種別を、新しい伝票種別に変更します。
5. 新たに作成されたエンティティー・リソース下の関連エンティティー・リソースを更新して、新たに作成されたリソースを指し示すようにします。
6. すべてのリンクが新しいビューを適切に指し示すようにするには、『アプリケーション全体にわたるカスタマイズ・ビューの導入』の手順に従ってください。

警告の詳細ビューのカスタマイズ

このタスクについて

警告コンソールでは、オーダーに対する警告の発生時に、警告の詳細の表示からそのオーダーをクリックして、詳細を表示できます。そのリンクが伝票種別のメイン詳細ビューもポイントするように指定するには、警告の詳細ビューをカスタマイズする必要があります。

警告の詳細ビューをカスタマイズする手順は、次のとおりです。

手順

1. 「警告」の新規詳細ビューの下で新規リンク ID を作成します。
2. この新規リンク ID が、新規伝票種別の新規作成したビューをポイントするようにします。
3. 新規伝票種別のリンク ID パラメーターを指定して getDetailHrefOptions() JSP ユーティリティー関数を呼び出すように新規警告の詳細ビューの JSP をカスタマイズします。
4. 新規警告の詳細ビューがデフォルトの警告の詳細ビューになるように、新規警告の詳細ビューのシーケンスを変更します。詳しくは、「*構成ガイド*」を参照してください。
5. すべてのリンクが新規ビューを適切にポイントするように設定するには、『カスタマイズしたビューのアプリケーション全体への取り込み (Incorporating Customized Views Across the Application)』のステップに従ってください。
6. 特定のオーダー番号の伝票種別を取り出すには、オーダー詳細を取り出す別の API (例外詳細 API の後) を呼び出すよう警告の詳細ビューを構成します。この API を、例外を無視するよう構成します。この API は、オーダー番号がパラメーターとして指定された形で、すべての例外に対して呼び出されますが、API はオーダー番号が無効の場合はエラーをスローします。この API は例外を無視するよう構成されるため、ユーザーには報告されません。

需要レコードの処理

伝票種別によっては、作成済みの需要を表示できるビューが関連付けられているものがあります。伝票種別は、供給の作成には関連付けられません。

Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales には、在庫コンソールの販売オーダー、返品、および購入オーダー (伝票種別はそれぞれ 0001、0003、および 0005) の詳細ビュー伝票種別から作成される需要を表示するためのコンソールが用意されています。「需要の詳細」画面には、指定した需要を作成する伝票へのハイパーリンクが含まれています。需要を作成できる新規伝票種別 (および該当する伝票種別 UI) の作成時には、「需要リスト」画面を拡張して、新規伝票種別の詳細画面にユーザーがナビゲートできるようにするリンク・リソースを追加する必要があります。

需要レコードを発生させる可能性があるカスタムの伝票種別を表示するには、需要の詳細が適切に表示されるように「需要の詳細」ビューを拡張する必要があります。「需要の詳細」画面には、新規伝票種別用の追加のリンク・リソースが必要です。伝票種別の新規 UI を作成してから、ビューを拡張します。

伝票種別の需要リスト・ビューの拡張

このタスクについて

伝票種別の需要リスト・ビューを拡張する手順は、次のとおりです。

手順

1. 『JSP コンソールの詳細ビューのカスタマイズ (Customizing Detail Views in the JSP Console)』の指示に従って、詳細ビュー (YIMD215) を拡張します。
2. YIMD215I01 内部パネルからコピーしたカスタム内部パネルの下にリンク・リソースを追加します。このリンク・リソースには、接尾辞の値として新規伝票種別の伝票種別コードが含まれている必要があり、また、このリンク・リソースは、新規伝票種別の詳細ビューのビュー ID をポイントする必要があります。
3. カスタマイズしたビューが、Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales で定義されているビューの前になるように、リソース ID シーケンスを設定します。

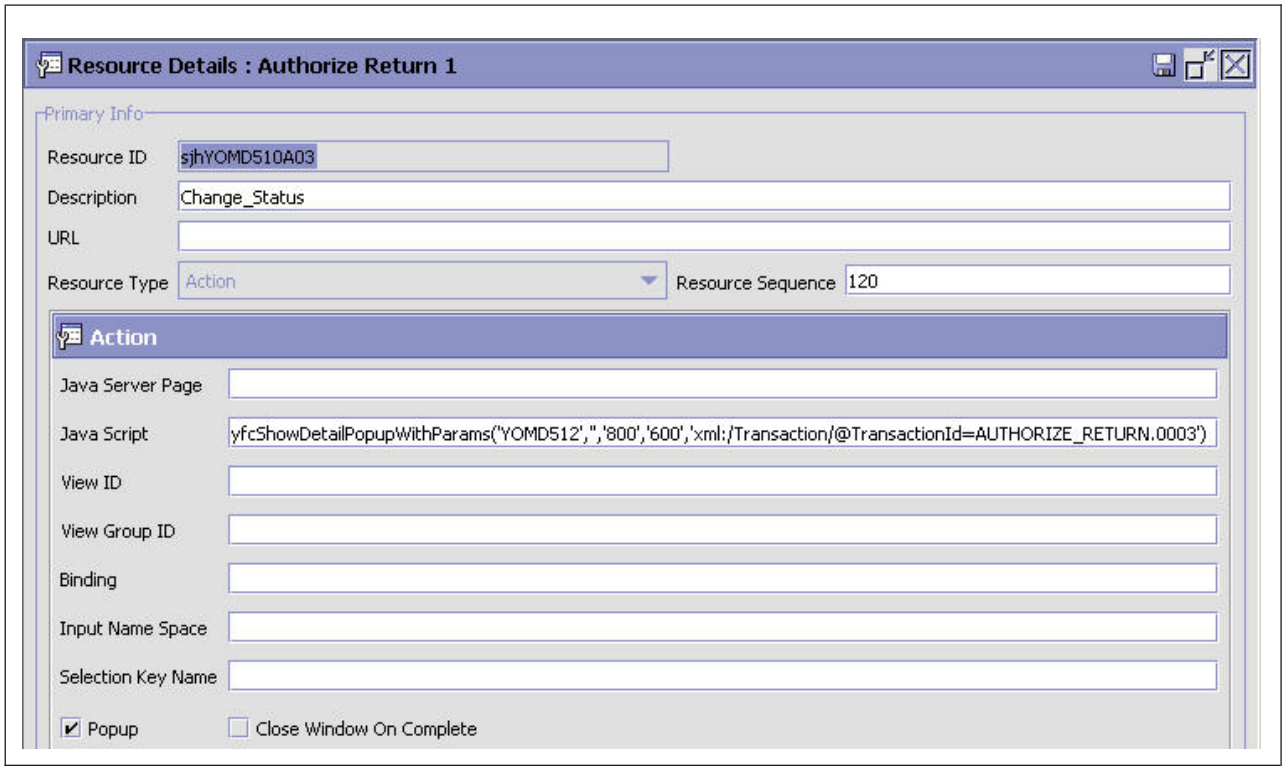
第 13 章 アクション、XML バインディング、API、ダイナミック名前空間、およびクレジット・カード番号

アクションの構成とカスタム・トランザクションの使用可能化

ご利用のビジネス・プロセスに応じて、パイプライン (pipeline) がエンティティーのステータス変更となる追加手順を必要とする場合があります。例えば、オーダーのライフサイクルにセキュリティー検査を追加して、顧客サービス担当者が、出荷する前にオーダーを手動で許可できるようにしたい場合があります。

Applications Manager のプロセス・モデリング内で構成されたカスタム・ステータス変更トランザクションは、エンティティーの詳細ビュー内でアクションとして構成できます。アクションを構成して、以下に説明するように、カスタム・トランザクションを直接呼び出すか、または個別のトランザクションを手動で確認することをユーザーに許可する新しいカスタム・ビューを使用してカスタム・トランザクションを呼び出すことができます。

この図は、「返品の承認」画面のステータス変更アクションを示しています。



内部パネルからのアクションの作成

このタスクについて

内部パネルからのアクションを作成するには、次の手順を実行します。

手順

1. アプリケーション・ルールのサイド・パネルのツリーから、「プロセス・モデリング」を選択します。各自の要件を満たすピックアップ・ステータスとドロップ・ステータス (drop status) とともに、パイプライン内で新しいトランザクションを構成します。
2. カスタム詳細ビューの内部パネルで、yfcShowDetailPopupWithParams() JavaScript 関数を呼び出すアクション・リソースを作成します。詳細ビューへパラメーターを渡す方法について詳しくは、『yfcShowDetailPopupWithParams』を参照してください。
3. 新しいトランザクション ID (手順 1 で作成) とビュー・リソース ID が、この JavaScript 関数への入力として渡されることを確認します。

対象エンティティー	使用するリソース ID
出荷	YOMD770
集合・混載	YDMD280
オーダー	YOMD390
購入オーダー	YOMD3390
返品	YOMD512

XML バインディング

API の入出力は、XML 文書形式です。 XML 文書を使用すると、API に渡してその出力を画面に読み込む必要がある入力を開発者が形成できるようにする XML バインディング・メカニズムをプレゼンテーション・フレームワークで提供できるようになります。

バインディング・ロジックは、入力フィールドの name 属性を使用して XML 属性にマップする処理に基づいています。形成する必要のある実際の HTML ストリングは、これを目的に提供されている、HTML タグ固有のさまざまな JSP 関数および JSP タグによって返されます。レンダリングされる HTML には、該当する XML パスに設定されている name 属性が含まれます。データがサーバーにポストされると、サービス定義フレームワークによって提供されるサーブレットが要求を取り込み、name 属性に含まれている入力および XML パスのデータから XML 文書を形成します。次に、この XML 文書は入力として、実行されるアクションに構成されている API に渡されます。

例えば、アイテムの検索ビューの getItemList() API の ShortDescription 属性に入力ボックスをバインドする場合があります。

```
<tr>
    <td class="searchlabel" ><yfc:i18n>Short_Description</yfc:i18n></td>
</tr>
<tr>
   <td nowrap="true" class="searchcriteriacell" >
```

```
       <input type="text" class="unprotectedinput"
<%=getTextOptions("xml:/Item/PrimaryInformation/@ShortDescription")%> />
    </td>
</tr>
```

JSP 関数および JSP タグが解決されると、HTML が以下のように形成されます。

```
<tr>
    <td class="searchlabel" >Short Description</td>
</tr>
<tr>
   <td nowrap="true" class="searchcriteriacell" >
       <input type="text" class="unprotectedinput"
name="xml:/Item/PrimaryInformation/@ShortDescription" value=""/>
   </td>
</tr>
```

注: この例は、JSP 関数によって返されるすべてのカスタム属性を示しているわけではありません。この例は、このトピックに関連するカスタム属性のみを示しています。

別の例では、ユーザーが入力ボックスに Telephone と入力します。データがサーバーにポストされると、プレゼンテーション・フレームワークは、入力ボックスの名前と値に基づいて以下の XML 文書を形成します。

```
<Item>
  <PrimaryInformation ShortDescription="Telephone"/>
</Item>
```

プレゼンテーション・フレームワークは、XML を形成するためにバインディング・ストリングを解析するので、バインディング・ストリングは、以下の構文に従う必要があります。

XML データ・バインディング構文

画面でバインドされる入力 XML に伴って API が呼び出されます。その XML バインディングは、API の出力に一致する必要があります。

- XML バインディング構文

  xml:NameSpace:/Root/Child@Attribute

  以下の例は、正しく構造化された構文を示しています。

  ```
  xml:/Order/PersonInfoShipTo/@Name
  xml:Order:/Order/PersonInfoShipTo/@Name
  ```

  以下の例は、不適切に構造化された構文を示しています。

  ```
  xml:/@Name
  xml:/Order
  xml:Order:/Order
  ```

- XML バインディング・パラメーター

  使用されるさまざまな XML バインディング・パラメーターは以下のとおりです。

  – **xml:** — この文字どおりに使用されます。

– **Namespace** — このバインディングの適用先の XML を含む名前空間。指定されていない場合、次に続くパスのルート・ノードと見なされます。Namespace は、共通コードに対するバインディング時にのみ含める必要があります。
– **:/Root/Child@Attribute** — 属性の XML パス。属性がルート・ノード自体の場合は、構文を /Root@Attribute として指定します。Root は、バインディングの適用先の XML のルート・ノード名を表します。Child は、子エレメント・ノード名を表します。このルールは、あらゆる深さレベルに適用されます。

この関数は、バインディング・ストリングを解析し、「@」文字を検索して、「@」文字の次に続くストリングを返します。

XML バインディングに関する特別な考慮事項

画面上の入力フィールドの XML バインディングによって、任意のフィールドを XML ドキュメント内の単一の属性に一意的にバインドできます。XML バインディングは、HTML 入力オブジェクトの名前として使用されます。バインディングは、ターゲット属性の完全 XML パスと属性名で構成されます。このため、XML リスト内に存在する XML 属性に入力ボックスをバインドする方法は直ちに明らかにはなりません。例えば、次の XML について考えてみてください。

```
<Order>
  <OrderLines>
    <OrderLine OrderLineKey="1000001" ShipNode="ShipNode1"/>
    <OrderLine OrderLineKey="1000002" ShipNode="ShipNode2"/>
  </OrderLines>
</Order>
```

複数のエレメント名の場合の XML バインディング

XML バインディングの一般ルールは、絶対パスと属性名の使用から成ります。ただし、これによってJSP に同じ名前の入力オブジェクトが複数生じることになる場合があります。同じ名前の複数の入力オブジェクトは、オブジェクトのアレイとしてポストされ、API にはポストされません。

各入力を特定の XML エレメントの一部として一意的に識別するには、アンダースコアー (「_」) とカウンターを含む接尾辞を繰り返しエレメント名の後に追加します。

例えば、オーダー明細のリストの各出荷ノード・フィールドのバインディングは、次のようになります。

```
xml:/Order/OrderLines/OrderLine/@ShipNode
```

各明細に編集可能な出荷ノードが記載されたオーダー明細のリストが含まれる画面が必要な場合、特別な XML バインディング規則を使用して、このシナリオを処理できます。

繰り返しエレメントは OrderLine です。出荷ノード (ship node) の入力オブジェクトごとに、特別な接尾辞が各明細に追加されます。結果として 2 つの固有の XML バインディングができます。このデータがポストされるとき、同じ特別な接尾辞を持つすべての XML バインディングが、API 入力で同一の XML エレメントに結合されます。

```
xml:/Order/OrderLines/OrderLine_1/@ShipNode
```

から (and)

```
xml:/Order/OrderLines/OrderLine_2/@ShipNode
```

この特別な接尾辞の XML バインディングの使用をより簡単にするために、loopXML JSP タグでは、個々のループに固有のカウンターが含まれる JSP 変数を使用できます。この JSP 変数は、loopXML JSP タグ内で使用でき、loopXML タグに指定された ID 属性にリテラルの Counter を付けたものです。例えば、JSP で以下の loopXML を使用します。

```
<yfc:loopXML name="Order" binding="xml:/Order/OrderLines/@OrderLine"
id="OrderLine">
```

これによって、OrderLineCounter JSP 変数を入力 XML バインディング内で使用できるようになります。以下に例を示します。

```
<input type="text" <%=yfsGetTextOptions("xml:/Order/OrderLines/OrderLine_"
+ OrderLineCounter + "/@ShipNode", "xml:/OrderLine/@ShipNode",
"xml:/OrderLine/AllowedModifications")%>/>
```

API へのデータ渡し

ユーザー・インターフェースをカスタマイズする際は、使用している API に正しい入力データが渡されることを確認する必要があります。ユーザー・インターフェースでデータを取得したり更新を実行したりするための主な方法は、API を使用することです。これらの API に正しい入力データが渡されることを確認することは、ユーザー・インターフェースをカスタマイズする上で重要な作業です。

当アプリケーションでは、ユーザー・インターフェースを介して入力データを API に渡すためのさまざまなメカニズムが用意されています。どのメカニズムをどのような組み合わせで使用すべきなのかは、画面のタイプと呼び出される API のタイプに応じて決まります。

ここでは、API にデータを渡すための以下のメカニズムの特長と利点を説明します。

- 入力名前空間
- エンティティー・キー
- 動的属性

入力名前空間

多くの場合、UI 上のフィールドのデータは API に直接渡される必要があります。簡単な例としては、編集可能な入力フィールドがある詳細画面が挙げられます。これらのフィールドは、当アプリケーションのデータベース内の入力済みデータを更新するために、保存 API に渡される必要があります。この保存 API は、特定の XML 構造を入力として受け取ります。詳細画面上のフィールドは、その入力をこの API と対応付ける XML バインディングを保有している必要があります。

「リソース階層」ツリーでは、アクション・リソース内の「入力名前空間」フィールドが適切に設定されて、コンソールから正しいデータが保存 API に渡されることが確認される必要があります。「Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales: *構成ガイド アプリケーション・プラットフォーム 構成ガイド*」を参照してください。

画面上のすべての入力フィールドは、その入力から同じ API への XML バインディングを保有しているため、これらのフィールドはすべてバインディング内で同じ XML ルート要素名を持っています。XML バインディングのこのルート要素名は、その入力フィールドの*名前空間*と呼ばれます。したがって、保存アクションがユーザー・インターフェース上で実行されると、「リソース階層」ツリーでそのアクションに対して指定された名前空間が含まれたすべての入力フィールドから XML が生成されます。この XML はその後、その保存アクション下で構成された API に渡されます。詳しくは、『XML Binding』を参照してください。

入力名前空間がよく使用されるもう 1 つの場所は、検索条件が入力されるユーザー・インターフェース画面です。検索条件はリスト API に渡されて、入力された検索条件に基づいてデータが取得されます。すべての検索条件フィールドは、その入力をリスト API と対応付ける XML バインディングを保有している必要があります。この XML のルート要素名は、ユーザーが検索を実行したときに表示されるリスト・ビューに対して指定されている必要のある名前空間です。

エンティティー・キー

頻繁に、詳細画面は、入力として渡される現在のエンティティーの 1 次キーのみを必要とする API 呼び出しアクションを備えています。ユーザー・インターフェースで詳細ビューが呼び出されるとき、少なくとも 1 つのエンティティー・キーが詳細ビューに渡されます。このエンティティー・キーは、そのエンティティーを一意的に識別する属性を含む XML 構造で構成されます。例えば、オーダー・エンティティーのエンティティー・キーは、常にオーダー・ヘッダー・キーの属性を含みます。このエンティティー・キーは、そのエンティティーの詳細 API に自動的に渡されます。このタイプのアクションが詳細ビューで構成された場合、正しい入力が API に渡されたことを確認するためのその他の手順は必要ありません。

動的属性

API に渡される予定の入力が、入力名前空間またはエンティティー・キーから使用できない場合があります。これらの場合、動的属性の使用を適用できる場合があります。リソース階層ツリー内で構成されたすべての API リソースには、入力フィールドがあります。この目的は、動的属性を指定する機能を提供することです。このフィールドの値は、要素および属性を使用した有効な XML 構造です。ここで指定される XML 構造は、呼び出された API によって受諾される正確な入力 XML 構造と一致している必要があります。

動的属性を使用した最も一般的な例の 1 つは、特定の内部パネルが表示する必要があるすべてのデータを取得するために複数の API を呼び出す必要がある場合です。例えば、オーダー・エンティティーの詳細ビューの内部パネルは、getOrderDetails() (標準詳細 API)、および画面上のいくつかのコンボ・ボックス用のデータを取得するために getCommonCodeList() API を呼び出すことが必要な場合があります。共通コード (common codes) はルール・セット・レベルで保管されるため、そのオーダーの正しいルール・セットが getCommonCodeList() API が入力として渡されることを確認することが必要です。getOrderDetails() API は、正しい出力テンプレートを指定されると、そのオーダーのルール・セット・キーを返します。このルール・セット・キーは、getOrderDetails() API の出力への参照によって getCommonCodeList() API の入力 XML 内の動的属性として、getCommonCodeList() API に渡すことができます。

例えば、getOrderDetails() API は、以下のように返します。

```
<Order OrderHeaderKey="..." OrderNo="..." RulesetKey="..."/>
```

getCommonCodeList() の API リソース定義の入力フィールドは、次のとおりです。

```
<CommonCode CodeType="ORDER_TYPE" RulesetKey="xml:/Order/@RulesetKey"/>
```

RulesetKey 属性の値は、getOrderDetails() 関数の出力を参照する XML バインディングに設定されることに注意してください。入力を JSP の内部にバインディングするときに適用される XML バインディングの正確に同じルールが、動的属性の使用時にも適用されます。この例は、動的属性を使用できる別の方法も示します。CodeType 属性も API リソース定義内の入力フィールドに指定されます。ここで、属性の値は、単純に、静的値「ORDER_TYPE」に設定されます。この属性と値は、呼び出されると *常に* この API に渡されます。非変更入力値は、この方法で指定できます。

詳細ビューに定義される API の動的属性に使用できる別の値は、現在のエンティティー・キー XML の任意の属性です。これは、API へ渡される入力が詳細画面で形成されたエンティティー・キー XML の同じ XML 構造を持っていない場合に役立ちます。現在のエンティティーの XML は、SelectionKeyName と呼ばれる特別な名前空間で利用可能です。

API 入力を指定する異なる機構を結合することができます。例えば、詳細ビュー・アクションで構成された API にエンティティー・キーおよび動的属性を渡す必要がある場合があります。エンティティー・キーの引き渡しは自動的に行われるため、正しい XML 構造を使用して、API の下に入力を指定することができます。キーの XML 構造は、入力フィールドの XML 構造と一致していなければならないことに注意してください。他にも動的属性内で使用できる特別な名前空間があります。詳しくは、「使用可能な動的属性名前空間」を参照してください。

使用可能な動的属性名前空間

Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales には、動的属性内で使用可能な以下の特別な名前空間があります。

- **CommonFields** - この名前空間は、common_fields JSP を使用しているときのみ使用可能です。利用可能な属性は、JSP の使用方法に依存します。
- **CurrentUser** - この名前空間は、getUserDetails() API を使用する現在のログイン・ユーザーについての詳細を含みます。使用可能な XML は、以下のとおりです。

  ```
  <User Activateflag="" BillingaddressKey="" BusinessKey=""
  ContactaddressKey="" Createprogid="" Createts="" Createuserid=""
  CreatorOrganizationKey="" Imagefile="" Localecode="" Loginid=""
  Longdesc="" MenuId="" Modifyprogid="" Modifyts="" Modifyuserid=""
  NoteKey="" OrganizationKey="" ParentUserKey="" Password="" PreferenceKey=""
  Pwdlastchangedon="" Theme="" UserKey="" UsergroupKey="" Username=""
  Usertype=""/>
  ```

- CurrentEnterprise - 現在のユーザーがエンタープライズ (enterprise) である組織 (organization) に属する場合、この名前空間は、そのエンタープライズに関する詳細を含みます。現在のユーザーがエンタープライズではない組織に属する場合、

この名前空間は、現在の組織の主要エンタープライズに関する詳細を含みます。組織の詳細は、getOrganizationHierarchy() API から取得されます。使用可能な XML は、以下のとおりです。

```
<Organization AccountWithHub="" AuthorityType=""
BillingAddressKey="" CatalogOrganizationCode="" CollectExternalThroughAr=""
ContactAddressKey="" CorporateAddressKey="" Createprogid="" Createts=""
Createuserid="" CreatorOrganizationKey="" DefaultDistributionRuleId=""
DefaultPaymentRuleId="" DunsNumber="" InterfaceTime="" InventoryKeptExternally=""
InventoryOrganizationCode="" InventoryPublished="" IsHubOrganization=""
IsSourcingKept="" IssuingAuthority="" ItemXrefRule="" LocaleCode=""
MerchantId="" Modifyprogid="" Modifyts="" Modifyuserid="" OrganizationCode=""
OrganizationKey="" OrganizationName=""
ParentOrganizationCode="" PaymentProcessingReqd=""
PrimaryEnterpriseKey="" PrimarySicCode="" PrimaryUrl="" RequiresChainedOrder=""
RequiresChangeRequest="" RulesetKey="" TaxExemptFlag="" TaxExemptionCertificate=""
TaxJurisdiction="" TaxpayerId="" XrefAliasType="" XrefOrganizationCode="">
```

- **CurrentOrganization** - この名前空間は、 getOrganizationHierarchy() API を使用する現在のログイン・ユーザーの組織についての詳細を含みます。使用可能な XML は、CurrentEnterprise 名前空間で使用可能な XML と同じです。
- **SelectionKeyName** - この名前空間は、現在のエンティティーの現在アクティブなキーにバインドされている XML を含みます。一般的に、リスト画面は XML キーを形成し、そのキーを、詳細画面にナビゲートするために使用されるチェック・ボックス (またはハイパーリンク) に関連付けます。このキーは、現在選択されているエンティティー・キーになります。詳細ビューは、このキーを使用して、そのエンティティーの詳細 API を呼び出します。この名前空間について詳しくは、「API へのデータの引き渡し (Passing Data to APIs)」を参照してください。

API へのデータのポスト

デフォルトでは、画面の編集可能なコンポーネントで検出されたデータは、それらの入力フィールドが何らかの XML にバインドされている場合でも、保存 API に自動的に送られません。このデータがポストされるのは、ユーザーが画面上でそのデータを変更した場合のみです。

ただし場合によっては、編集可能なコンポーネント内のデータが変更されていない場合でも、そのデータの一部または全体を渡すことが必要になります。例えば、入力ボックスに JSP 内の特定のデフォルト値が表示されるように設定されている画面では、すべてのデータがポストされる必要があります。

ユーザーがその入力ボックス内のデータを変更しない場合でも、その値が API に渡されることが望まれます。このため、ユーザー・インターフェースで実際に変更された内容にかかわらず、編集可能なコンポーネント内のすべてのデータが JSP 全体の API に渡されることを確認するメカニズムが用意されています。以下のコード例では、このことを実現する方法を示しています。

```
<script language="Javascript">
 IgnoreChangeNames();
</script>
```

通常はこのコードは、JSP の先頭に配置されている include ステートメントの直後に配置されます。

データ型

HTML ページの入力ボックスは、フィールド・サイズおよびデータ型に関する特定の制約に準拠している必要があります。例えば、入力ボックスのサイズは、収集されるデータのタイプに対応する必要があります。同様に、各入力ボックスは、フィールドが収集するように設計されているデータのタイプ (数値、日付、ストリングなど) に対応していることを検証することが必要です。例えば、ストリング・フィールドは、特定の長さより長いデータをユーザーが入力するのを防ぐ必要があります。テキスト・ボックスを API 出力にバインドするために使用される属性も、データ型および関連プロパティー (サイズ、10 進数など) を解決します。

プレゼンテーション・フレームワークには、2 つの API、getTextOptions() および yfsGetTextOptions() を備え、各フィールドにこれらの属性を明示的に設定する必要がなくなるようにします。これらの API を入力ボックスに使用すると、データ型関連の属性および検証が自動的に行われます。

これらの属性定義およびマッピングは、次の 2 つのファイルに含まれます。

- yfsdatatypemap.xml ファイル — 抽象データ型を XML 属性名にマップします。
- datatypes.xml ファイル — データ型を定義します。

抽象データ型マッピング

XML 属性は、抽象データ型にマップされます。このマッピングは、*INSTALL_DIR*/repository/xapi/template/merged/resource/yfsdatatypemap.xml ファイルに含まれます。

例えば、Name 属性が XML 属性 TaxBreakupKey を含み、DataType 属性が抽象データ型キーを含む場合、これは次のように yfsdatatypemap.xml ファイルに定義されます。

```
<Attribute Name="TaxBreakupKey" DataType="Key" />
```

抽象データ型定義

抽象データ型は、データベース・データ型 (DATE、VARCHAR2 など)、データベース・サイズなどを定義します。抽象データ型は、*INSTALL_DIR*/repository/datatypes/datatypes.xml ファイルに定義されます。

このファイルからのサンプル・エントリーを示します。

```
<DataType Name='Address' Type='VARCHAR2' Size='70'>
   <UIType Size="30" UITableSize="30"/>
</DataType>
<DataType Name='Count' Type='NUMBER' Size='5'
NegativeAllowed="false" ZeroAllowed="true">
   <UIType Size="5" />
</DataType>
<DataType Name='TimeStamp' Type='DATETIME' Size='17'/>
<DataType Name='Date' Type='DATE' Size='7'>
   <UIType Size="12" UITableSize="15"/>
</DataType>
<DataType Name='Quantity' Type='NUMBER' Size='10' NegativeAllowed="false" ZeroAllowed="true">
   <XMLType Type="QUANTITY"/>
</DataType>
```

標準抽象データ型の定義については、*INSTALL_DIR*/repository/datatypes/datatypes.xml ファイルを参照してください。

属性がサポートされた datatypes.xml のリストについては、「データ型参照 (Data Type Reference)」を参照してください。

データ型の決定

Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales は、yfsdatatypemap.xml ファイルをデータ型の決定に使用します。

最初にアプリケーションは、バインディングのために指定されたストリングを検索するためにファイルを検索します (xml: 接頭部を含む)。バインディング・ストリング全体が見つからない場合、アプリケーションは、バインディング・ストリングの属性部分のみをファイルで検索します。

属性は、これらの 2つのフォーマットのうちの 1 つで検出できなければなりません。

次に、datatypes.xml ファイル内の定義を使用して、プレゼンテーション・フレームワーク API は、クライアント・サイドで使用されるカスタム属性を備えた適切な HTML ストリングを作成します。

データ型検証

クライアント・サイドの JavaScript 関数は、ページがロードされると実行し、入力ボックス内のカスタム属性およびサイズを適切に読み取ります。これらの入力ボックスに、window.attachEvent() 関数を介して検証機能が付加されます。検証機能は、カスタム属性を使用して、データ型検証も実行します。

クレジット・カード番号の表示

クレジット・カード番号は、それらを見る権限を持つユーザーにのみ表示されなければなりません。したがって、クレジット・カード番号を表示するカスタム画面を作成するときには、以下のルールを使用して、確実にこのセキュリティーが維持されるようにします。

- CurrentUser 名前空間は、ユーザー・ノードの下に属性 ShowCreditCardInfo を含みます。この属性は、現在のログイン・ユーザーがクレジット・カード番号を見る権限を持っている場合は TRUE、現在のログイン・ユーザーが必要な権限を持っていない場合には FALSE になります。
- クレジット・カード番号を返す API は、通常、暗号化されたクレジット・カード番号を返します。これらの API は、クレジット・カード番号の最後の 4 桁を含む DisplayCreditCardNo 属性も返します。
- DisplayCreditCardNo 属性を showEncryptedCreditCardNo() JSP 関数と共に使用して、クレジット・カード番号の最初をアスタリスク (*) で表示し、その後に最後の4 桁を表示します。
- ログイン・ユーザーが暗号化解除されたクレジット・カード番号を見る権限を持っている場合のみ表示するハイパーリンクをクレジット・カード番号の上に形成します。以下に例を示します。

```
 <% if (userHasDecryptedCreditCardPermissions()){%>
    <yfc:makeXMLInput name="encryptedCCNoKey">
       <yfc:makeXMLKey
binding="xml:/GetDecryptedCreditCardNumber/@EncryptedCCNo"
```

```
value="xml:/PaymentMethod/@CreditCardNo"/>
       </yfc:makeXMLInput>
       <td class="protectedtext">
          <a <%=getDetailHrefOptions(decryptedCreditCardLink,
getParameter("encryptedCCNoKey"),"")%>>
             <%=showEncryptedCreditCardNo(resolveValue("xml:/PaymentMethod/
@DisplayCreditCardNo"))%>
            </a>
        </td>
 <% } else { %>
      <td class="protectedtext">

       <%=showEncryptedCreditCardNo(resolveValue("xml:/PaymentMethod/
@DisplayCreditCardNo"))%> 
           <yfc:getXMLValue binding="xml:/PaymentMethod/
@DisplayCreditCardNo"/> 
       </td>
 <% } %>
```

- 次に、ハイパーリンクがクリックされたときに開くポップアップ・ウィンドウを作成します。
- このポップアップ・ウィンドウで getDecryptedCreditCardNumber() を呼び出して、クレジット・カード番号を暗号化解除し、現在のログイン・ユーザーが権限を持っている場合は TRUE、現在のログイン・ユーザーが権限を持っていない場合は FALSE として、DisplayFlag 属性を渡します。
- getDecryptedCreditCardNumber() の出力を使用して、暗号化解除されたクレジット・カード番号を画面上に表示します。

画面用に getDecryptedCreditCardNumber() API を、Applications ManagerConfigurator を介して構成するときには、動的入力を指定して、DisplayFlag 属性が現在のユーザーの許可に基づいて API に渡されるようにする必要があります。「入力」フィールドの指定方法の例を示します。

```
<GetDecryptedCreditCardNumber
DisplayFlag="xml:CurrentUser:/User/@ShowCreditCardInfo"
EncryptedCCNo="xml:/Order/PaymentMethods/PaymentMethod/@CreditCardNo"/>
```

以下の例に従って、「テンプレート」フィールドを指定します。

```
<GetDecryptedCreditCardNumber DecryptedCCNo=""/>
```

複数のクレジット・カード番号の表示

リストにクレジット・カード番号を表示する場合、DisplayCreditCardNo 属性を表示することを選択する場合があります。この属性は、CreditCardNo を出力する API によって返されます。

API によって返されたクレジット・カード番号にアスタリスクを付加するには、DisplayCreditCardNo 属性および showEncryptedCreditCardNo() メソッドを使用します。

リストに暗号化解除されたクレジット・カード番号のリストを表示するには、各行に対してループで getDecryptedCreditCardNumber() を呼び出します。これは費用のかかる操作であるため、暗号化されたクレジット・カード番号 (**********1234 と表示) のリストを DisplayCreditCardNo 属性を使用して表示したい場合があります。CreditCardNo を出力するすべての API は、この属性を返します。次に、暗号化されたクレジット・カード番号を、指定したクレジット・カード番号を暗号化解除され

た形式で表示するポップアップ・ウィンドウにリンクします。

第 14 章 ユーザー・インターフェース・スタイル・リファレンス

コントロールとクラス

このセクションは、最も一般的なタイプの HTML コントロールおよびそれに対応する CSS クラス・タグのクイック・リファレンス・リストを示します。JSP ファイルで使用される典型的なコントロールおよび対応する CSS クラスを以下の表にリストしています。

コントロールおよびタグ	使用可能なクラス	説明
ボタン	button	すべてのボタン用。
チェック・ボックス	checkboxcolumn	テーブルの列内に表示されるチェック・ボックス用。
	checkboxheader	テーブルのヘッダーに表示されるチェック・ボックス用。このクラスは、<td> セル・レベルで使用し、<input> タグ・レベルでは使用しないでください。
コンボ・ボックス/選択	combobox	すべてのコンボ・ボックス用。
アイコン	columnicon	テーブル列内のアイコン用。
	icon	テーブル列内にないアイコン用。
	lookupicon	特定の編集可能入力テキストの右に表示されるルックアップ・アイコン用。
入力ボックス	dateinput	編集可能日付入力用。
	numericprotectedinput	編集不可能数値入力用。右寄せ。囲み <td> タグには、class="protectednumber" が必要です。
	numericunprotectedinput	編集可能数値入力用。右寄せ。
	protectedinput	編集不可能入力用。囲み <td> タグには、class="protectedtext" が必要です。
	unprotectedinput	編集可能入力用。
ラベル	該当なし	これが常駐するテーブル・クラスによって指定されるものと同じフォントを使用します。詳細ビューには詳細ラベルを、検索ビューには検索ラベルを使用します。
ラジオ・ボタン	radiobutton	すべてのラジオ・ボタン用。
テーブル (表形式ではないデータ)	表示	詳細ビュー用。このクラスは、HTML ページがロードされたときに動的に列幅を設定する view.htc に関連付けられてます。

コントロールおよびタグ	使用可能なクラス	説明
テーブル (表形式データ)	テーブル	表形式データ型のすべてのテーブル用。テーブル・エレメントに以下の属性を指定します。 • editable="true" • deleteAllowed="true" • addAllowed="true" ヘッダー行は、<thead> タグに含まれている必要があります。ヘッダー・セルは、<td> タグに含まれている必要があります。<th> タグではありません。本体行は、<tbody> タグに含まれている必要があります。テーブルが追加を許可する編集可能リストである場合は、<tfoot> タグ内にテンプレート行を指定します。テンプレート行 <tr> タグに以下の属性を指定します。 • "style="display:none" • TemplateRow="true" • ByPassRowColoring="true"
テキスト	numerictablecolumn	テーブル列内でのテキストの表示用。右寄せ。
	protectednumber	数値データの表示用。右寄せ。
	protectedtext	テキストの表示用。
	searchcriteriacell	各検索条件の下部 <td> タグ用。 **注:** クラスを searchcriteriacell として使用しているすべての <td> タグは、nowrap="true" を指定する必要があります。これにより、テキスト・ボックス、入力ボックス、および照会コンボ・ボックスと一緒に使用したときに、ルックアップ・アイコンが次の行に折り返すのを防ぎます。
	tablecolumn	テーブル列内での編集不可能テキストの表示用。
	tablecolumnheader	テーブル列・ヘッダー内でのテキストの表示用。
テキスト領域	textarea	テキスト領域を一貫性のあるものに保持するため。getTextAreaOptions() も参照してください。
合計フィールド	totaltext	合計フィールドを識別するためにテキストを異なるカラーで表示します。

ページ・レイアウト

次の表では、アプリケーション・コンソールの画面をカスタマイズする際に従うべきガイドラインを示しています。

対象	ガイドライン
アンカー・ページ	画面に組み込まれる内部パネルのレイアウトを決定します。例えば、「オーダーの詳細」アンカー・ページは、オーダーの詳細に関連するすべての内部パネルを組み込んで、これらの内部パネルのレイアウトを決定します。各エンティティーは、独自のアンカー・ページを持っている必要があります。 アンカー・ページの作成時には以下のガイドラインに従ってください。 テーブル・タグ - 一番外側 (つまりコンテナーの) <table> タグには、cellspacing="0" 属性と cellpadding="0" 属性が含まれている必要があります。間隔設定は、内部パネル自体用に使用されているクラスを通じて行われるからです。これらの属性が "0" に設定されていない場合は、間隔とパディング (外枠と内容の間の余白) の大きさにばらつきが生じる可能性があります。 セル・タグ - 横方向に並んだセルの高さを統一するために、同じ <tr> タグ内の <td> タグには height="100%" が含まれている必要があります。
内部パネルのタイトル・バー	タイトル・バーの左側に配置されたアイコンを通じて許可される、使用可能なポップアップ・ウィンドウへのアクセス。 タイトル・バーの右側に配置されたドロップダウンに含まれている実行可能なアクション。 上記のどちらも、サービス定義フレームワークを通じて実現されます。
インセット	対称的なレイアウトで配置される必要があり、他の画面と一貫している必要があります。 相互の間隔およびメイン・ページの端部との間隔が 5 ピクセルである必要があります。この間隔を内部パネルに適用してください (最初に一番上の内部パネルに適用してから、下および右の順に適用します)。それぞれの個別の内部パネルについて、上辺、左辺、右辺、および底辺の順に間隔を適用します。必要以上の間隔を追加しないように注意してください。例えば、横方向に並べられた 2 つの内部パネルがある場合に、左側のパネルの右辺に 5 ピクセルの間隔が適用されている場合は、右側のパネルの左辺には 5 ピクセルの間隔を適用する必要はありません。後者の間隔を適用すると合計の間隔が 10 ピクセルになるからです。
ラベルと入力	画面上のラベルの個別の名前は、Applications Manager Configurator で使用されている名前と同じである必要があります。Applications ManagerConfigurator で使用されていないラベルの名前は、旧リリースのコンソールと同じである必要があります。 相互の上下間隔は 5 ピクセルである必要があります。パディングや間隔のデフォルト値はありません。
リスト・チェック・ボックス	ユーザーが個別の行を選択できるようにするためにリストに表示されるチェック・ボックス。これらは、各行の左側に表示される必要があり、これらを表示する JSP 内にコーディングされています。

対象	ガイドライン
リスト	数値を表示するすべての列は右寄せ表示に設定される必要があります。
メニュー・バー	ポップアップ・ウィンドウではないすべてのページの最上部に表示される必要があります。
ページのタイトル・バー	現在のページを説明する語句が表示されて、使用可能なビューのリストが含まれている必要があります(複数のビューを使用できる場合)。
検索条件	すべての使用可能な検索条件テキスト入力の前にはコンボ・ボックスが配置される必要があります。テキスト検索フィールドについては、「*完全一致 (is)*」、「*前方一致 (starts with)*」、および「*部分一致 (contains)*」のコンボ・ボックス・オプションを配置してください。数値検索フィールドについては、「*次より小*」、「*次より大*」、および「*等しい*」のコンボ・ボックス・オプションを配置してください。 ルックアップは、テキスト入力の右側に配置する必要があります。

ハイパーテキスト・リンク

画面上に、別のエンティティーへの論理参照であるフィールドがある場合は、そのフィールドのデータをその参照先であるエンティティーにハイパーリンクします。例えば、「オーダー明細の詳細」画面の「オーダー番号」フィールドを「オーダー」画面にハイパーリンクします。

第 15 章 JSP コンソール・インターフェースのプログラミング標準

整形式 JSP ファイルを作成するための標準手法

HTML コードは Java Server Pages に埋め込まれていますが、読みやすい JSP コードを記述することを目指してください。API に組み込むことができない特別な XML 操作を必要とする場合は、別個の JSP ファイルをインクルードすることで、HTML タグと Java コードが混在することを回避してください。

JSP ファイルを記述する際は、次の標準手法に従ってください。

- タブ間隔 - エディターのタブ間隔を 4 に設定します。
- JavaScript ファイル - JSP ファイルには JavaScript を一切記述しないでください。すべての JavaScript を別個の JS ファイルに格納してください。
- HTML タグ - すべての HTML タグと HTML 属性を小文字で入力します。
- HTML 属性 - すべての HTML 要素属性値を二重引用符で囲みます。単一引用符や引用符なしでも正常に機能する可能性がありますが、二重引用符を使用するのが標準の書式です。
- HTML テーブル - HTML ページ内のテーブル数をできるだけ少なくします。特に、ネストされたテーブル (テーブル内の別のテーブル) の数を減らします。
- タグ - すべてのタグを閉じます (必要かどうかにかかわらず)。
- 制御要素 - 制御要素ごとに、その制御要素の最後の属性として get...Options 属性を追加します。
- コメント - すべてのコメントを <%/*.....*/%> という形式で囲みます。

ヒント: フォームのコーディングが完了したら、そのフォームをビジュアル HTML エディターで開いて、HTML が整形式であることを確認してください。

有効な HTML タグおよび属性

HTML 属性をアプリケーションで使用する際には、使用の助けとなるこの HTML 参照資料に従ってください。

注: 新たに独自の標準セットを構築する限りは、その他の HTML 属性を使用することもできます。

次の表は、推奨される標準 HTML タグとその属性を示しています。各 HTML タグについて、「タグ属性」列に示されている属性のみを使用してください。

HTML タグ	タグ属性
<% %>	可能な限り JSP の先頭に保持します。
<a>	href
<imp>	alt
	border

HTML タグ	タグ属性
	name
	src
	style
<input>	class
	maxlength
	name
	onblur
	style
	値
<option>	binding
	selected
	type
	値
<select>	class
	name
<table>	editable
	cellPadding
	cellspacing
	class
	style
<tbody>	該当なし
<td>	class
	colspan
	nowrap
	onclick
	rowspan
	sortable
	sortValue
	style
<tfoot>	該当なし
<thead>	該当なし
<tr>	style
	templateRow (true/false)

JSP ファイルおよびディレクトリーの命名規則

ディレクトリーに JSP ファイルを取り込む場合、一貫性のある階層ディレクトリー構造および一貫性のあるファイル命名規則に従ってください。

JSP ファイルの名前を選択するときには、以下の規則を使用してください。

- 大文字は使用しないでください。

- ワードの区切りにはハイフンではなくアンダースコアーを使用してください。
- JSP ファイルがアンカー・ページの場合は、名前に *anchor* というワードを含めます。

ディレクトリーおよびファイル名構文

module/*entity_screen_type_viewdesc*|anchor.jsp

例

```
om/orderline/search/orderline_search_bydate.jsp
```

module は、2 文字のモジュール・コードを表します。次に例を示します。

- cm - カタログ管理
- em - アラート管理
- im - 在庫管理
- om - オーダー管理
- pm - 参加者管理

entity は、エンティティーのリソース ID を表します。次に例を示します。

- order - オーダー・エンティティー
- orderline - オーダー明細エンティティー
- orderrelease - オーダー・リリース・エンティティー

screen_type は、ビューのリソース・タイプを表します。次に例を示します。

- list - リスト・ビュー
- detail - 詳細ビュー
- search - 検索ビュー

viewdesc は、ビューまたは内部パネルの簡略説明を表します。次に例を示します。

- primaryinfo - 主要情報
- paymentinfo - 支払い情報
- collectiondtl - 集金の詳細

コントロールの命名規則

JSP 内の XML バインディング入力コントロールにプレゼンテーション・フレームワーク関数を使用する場合は、そのコントロールに名前属性を設定しないでください。つまり、各コントロールの命名を回避します。その代わり、HTML オブジェクトまたは DOM 階層を使用してコントロールにアクセスします。コントロールに名前を付ける場合は、それぞれのページで名前が固有のものであることを確認してください。

国際化対応

プレゼンテーション・フレームワークは、多言語に対応したアプリケーションを作成することを可能にします。このことを実現するために、プレゼンテーション・フレームワークは、ロケールに合わせてカスタマイズできる以下の機能を提供しています。

- リテラル用の i18n JSP タグ
- グラフィックスとイメージ
- クライアント側のエラー・メッセージ
- 日付と数値の検証

HTML ファイルおよび CCS ファイルの検証

HTML ファイルと CSS ファイルの両方を検証できます。市販のソフトウェア・パッケージや以下の World Wide Web Consortium (W3C) 検証サービスなどの無償のオンライン・アプリケーションを使用できます。

- W3C CSS 検証サービス (http://jigsaw.w3.org/css-validator/)
- W3C HTML 検証サービス (http://validator.w3.org/)

別の方法として、フォームのコーディングの完了後に、そのフォームを任意のビジュアル HTML エディターで開き、HTML のフォームが適切であることを検証できます。

第 16 章 CSS テーマ・ファイル参照

JSP コンソールの CSS テーマ

標準 Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales テーマでは、CSS ファイル内に指定されているように Tahoma フォントが使用されます。異なるサイズのフォントを使用すると、ドロップダウン・リスト項目の切り捨てなどの、表示の問題が発生する場合があります。そのような状態の場合、CSS ファイルを編集して、画面を正しく表示できるようにするプロパティーを指定することができます。次の表に説明されているクラスおよびプロパティーを使用します。

クラス (Class)	説明
favouritespopuprowhighlight	マウスオーバー・アクション中に強調表示する「お気に入り」フォルダー・アイコンの下のドロップダウン・リスト項目。以下のプロパティーがあります。 • charheight - 文字の垂直方向のサイズ。favouritespopuprownormal クラスで指定したものと同じ値を使用します。 • charwidth - 文字の水平方向のサイズ。favouritespopuprownormal クラスで指定したものと同じ値を使用します。
favouritespopuprownormal	「お気に入り」フォルダー・アイコンの下のドロップダウン・リスト。以下のプロパティーがあります。 • charheight - 文字の垂直方向のサイズ。favouritespopuprowhighlight クラスで指定したものと同じ値を使用します。 • charwidth - 文字の水平方向のサイズ。favouritespopuprowhighlight クラスで指定したものと同じ値を使用します。
ipactionspopuprowhighlight	マウスオーバー・アクション中に強調表示する詳細ビューのドロップダウン・リスト項目。ヘッダーおよび行情報の両方を含みます。以下のプロパティーがあります。 • charheight - 文字の垂直方向のサイズ。ipactionspopuprownormal クラスで指定したものと同じ値を使用します。 • charwidth - 文字の水平方向のサイズ。ipactionspopuprownormal クラスで指定したものと同じ値を使用します。

クラス (Class)	説明
ipactionspopuprownormal	詳細ビューのドロップダウン・リスト項目。ヘッダーおよび行情報の両方を含みます。以下のプロパティーがあります。 • charheight - 文字の垂直方向のサイズ。ipactionspopuprowhighlight クラスで指定したものと同じ値を使用します。 • charwidth - 文字の水平方向のサイズ。ipactionspopuprowhighlight クラスで指定したものと同じ値を使用します。
listactionspopuphighlight	マウスオーバー・アクション中に強調表示するリスト・ビューのドロップダウン・リスト項目。リスト・ビュー上。以下のプロパティーがあります。 • charheight - 文字の垂直方向のサイズ。listactionspopupnormal クラスで指定したものと同じ値を使用します。 • charwidth - 文字の水平方向のサイズ。listactionspopupnormal クラスで指定したものと同じ値を使用します。
listactionspopupnormal	リスト・ビューのドロップダウン・リスト項目。以下のプロパティーがあります。 • charheight - 文字の垂直方向のサイズ。listactionspopuphighlight クラスで指定したものと同じ値を使用します。 • charwidth - 文字の水平方向のサイズ。listactionspopuphighlight クラスで指定したものと同じ値を使用します。
menuitempopuprowhighlight	マウスオーバー・アクション中に強調表示するメニュー・バーのドロップダウン・リスト項目。以下のプロパティーがあります。 • charheight - 文字の垂直方向のサイズ。menuitempopuprownormal クラスで指定したものと同じ値を使用します。 • charwidth - 文字の水平方向のサイズ。menuitempopuprownormal クラスで指定したものと同じ値を使用します。
menuitempopuprownormal	メニュー・バーのドロップダウン・リスト項目。以下のプロパティーがあります。 • charheight - 文字の垂直方向のサイズ。menuitempopuprowhighlight クラスで指定したものと同じ値を使用します。 • charwidth - 文字の水平方向のサイズ。menuitempopuprowhighlight クラスで指定したものと同じ値を使用します。
menulevel1hl	マウスオーバー・アクション中に強調表示するメニュー・バー項目。以下のプロパティーがあります。 • height - 背景の垂直方向のサイズ。

クラス (Class)	説明
menulevel1norm	メニュー・バー項目。例えば、オーダー、供給、システム管理など。以下のプロパティーがあります。 • height - 背景の垂直方向のサイズ。
searchentitiespopuprowhighlight	マウスオーバー・アクション中に強調表示する検索ビューのドロップダウン・リスト項目 (左側)。以下のプロパティーがあります。 • charheight - 文字の垂直方向のサイズ。 searchentitiespopuprownormal クラスで指定したものと同じ値を使用します。 • charwidth - 文字の水平方向のサイズ。 searchentitiespopuprownormal クラスで指定したものと同じ値を使用します。
searchentitiespopuprownormal	検索ビューのドロップダウン・リスト項目 (左側)。以下のプロパティーがあります。 • charheight - 文字の垂直方向のサイズ。 searchentitiespopuprowhighlight クラスで指定したものと同じ値を使用します。 • charwidth - 文字の水平方向のサイズ。 searchentitiespopuprowhighlight クラスで指定したものと同じ値を使用します。
searchviewspopuprowhighlight	マウスオーバー・アクション中に強調表示する検索ビューのドロップダウン・リスト項目 (右側)。以下のプロパティーがあります。 • charheight - 文字の垂直方向のサイズ。 searchviewspopuprownormal クラスで指定したものと同じ値を使用します。 • charwidth - 文字の水平方向のサイズ。 searchviewspopuprownormal クラスで指定したものと同じ値を使用します。
searchviewspopuprownormal	検索ビューのドロップダウン・リスト項目 (右側)。以下のプロパティーがあります。 • charheight - 文字の垂直方向のサイズ。 searchviewspopuprowhighlight クラスで指定したものと同じ値を使用します。 • charwidth - 文字の水平方向のサイズ。 searchviewspopuprowhighlight クラスで指定したものと同じ値を使用します。

第 17 章 コンソール JSP インターフェースの JSP 関数

changeSessionLocale

説明

ロケールはユーザー・レベルで構成されますが、changeSessionLocale JSP 関数を使用して、特定のロケールに動的に切り替えることもできます。

構文

void changeSessionLocale(String localecode)

入力パラメーター

localecode - 切り替え先のロケール。

equals

説明

equals JSP 関数は、ヌル、ゼロまたは 1 つ以上の空白を含む、オブジェクトを処理する Java の equals 関数をカバーするものです。そのような状態では、2 つのオブジェクトは等しいと考えられます。

構文

booleanequals(Object obj1, Object obj2)

入力パラメーター

obj1、obj2 - 必須。比較すべき 2 つのオブジェクト。

例

この例では、この関数がストリング比較を行う方法を示します。

```
<% String sAvailable="";
if(equals(resolveValue("xml:/InventoryInformation/Item/@TrackedEverywhere"),
"N"))
   sAvailable=getI18N("Available") + ": " + getI18N("INFINITE");
else
   sAvailable=getI18N("Available")
+ ": " +resolveValue("xml:/InventoryInformation/Item/@AvailableToSell");
%>
```

getCheckBoxOptions

説明

この JSP 関数は、変更ルールを考慮する必要がない場合の XML バインド・チェック・ボックスに対する標準関数です。

構文

String getCheckBoxOptions(String name)

String getCheckBoxOptions(String name,String a_checked, String a_value)

入力パラメーター

name - 必須。チェック・ボックス入力の名前属性の値。バインディングまたはリテラルに設定できます。

checked - 必須。値属性の値がこの値と等しい場合、CHECKED 属性は TRUE に設定します。

value - 必須。チェック・ボックス入力用の値属性の値。バインディングまたはリテラルに設定できます。

注: 名前パラメーターのみが渡される場合、値は名前パラメーターに渡された同じ値にデフォルト設定されます。

JSP 使用法

```
<input type="checkbox"
<%=getCheckBoxOptions("xml:Order:/Order/Addlinfo/@Country" ,"IND",
"xml:Order:/Order/Addlinfo/@Country" )%></yfc:i18n>India</yfc:i18n></input>
```

結果 HTML

```
<input type="checkbox" name="xml:Order:/OrderAddlinfo/@Country"
value="IND">India</input>
```

getColor

説明

この JSP 関数は、特定のカラー・オブジェクトに基づいて、HTML カラーを 16 進コードで返します。

構文

public String getColor(java.awt.Color color)

入力パラメーター

color - 必須。カラー・オブジェクト。

getComboOptions

説明

この JSP 関数は、変更ルールを考慮する必要がない場合に XML バインド・コンボ・ボックスに対して標準関数を提供します。

構文

String getComboOptions(String name)

String getComboOptions(String name, String value)

入力パラメーター

name - 必須。フォームがポストされたときに入力テキスト内の値が送信されるターゲット XML内のパス。プレゼンテーション・フレームワークを介して、ターゲット XML は適切な API に渡されます。

value - 必須。コンボ・ボックス内で選択される値を指定します。バインディングまたはリテラルに設定できます。

JSP 使用法

この例は、「オーダー・エントリー」画面でエンタープライズ・コード・コンボ・ボックスをレンダリングする方法を示します。この API は、loopOptions JSP タグと共に使用されます。

```
   <select class="combobox" onChange="updateCurrentView()"
<%=getComboOptions("xml:/Order/@EnterpriseCode",enterpriseCode)%>>
      <yfc:loopOptions binding="xml:/OrganizationList/@Organization"
name="OrganizationCode" value="OrganizationCode"
selected="xml:/Order/@EnterpriseCode"/>
   </select>
```

getComboText

説明

この JSP 関数は、値のリストから説明を取得するための標準機能を提供します。

構文

String getComboText(String binding, String name, String value, String selected)

String getComboText(String binding, String name, String value, String selected,boolean localized)

入力パラメーター

binding - 必須。API 出力内の繰り返し要素をポイントするバインディング・ストリング。繰り返し要素は、at 文字 (「@」) を使用して固定されるものである必要があります。

name - 必須。フォームがポストされたときに入力テキスト内の値が送信されるターゲット XML内のパス。プレゼンテーション・フレームワークを介して、ターゲット XML は適切な API に渡されます。

value - 必須。コンボ・ボックス内で選択される値を指定します。バインディングまたはリテラルに設定できます。

selected - オプション。評価され、デフォルトの選択値として設定される必要があるバインディング・ストリング。これは、値属性と一致し、説明属性とは一致しません。デフォルトは、ブランク、例えば、スペース (「 」) です。

localized - オプション。TRUE として渡されると、表示するローカライズした説明を取り出します。

JSP 使用法

この例は、「オーダー・エントリー」画面でエンタープライズ・コード・コンボ・ボックスをレンダリングする方法を示します。この API は、 loopOptions JSP タグと共に使用されます。

```
   <%=getComboText("xml:TaxNameList:/CommonCodeList/@CommonCode",
"CodeShortDescription","CodeValue","xml:/HeaderTax/@TaxName",true)%>
```

getDateOrTimePart

説明

この JSP 関数は、タイム・スタンプ値の日付部分または時間部分を表すストリングを返します。

構文

String getDateOrTimePart(String type, String value);

入力パラメーター

type - 必須。日付または時間を表示するかどうかを指定します。YFCDATE を渡してタイム・スタンプの日付部分を返します。YFCTIME を渡してタイム・スタンプの時間部分を返します。

value - 必須。現在のログイン・ユーザーのロケールのタイム・スタンプ・フォーマット内のタイム・スタンプ値を含むストリング。

出力パラメーター

タイム・スタンプ属性の日付部分または時間部分を表すストリング。

例

この例では、getDateOrTimePart() 関数を使用して、xml:/OrderRelease/@HasDerivedParent バインディング内で参照する属性の日付部分を返します。

```
getDateOrTimePart("YFCDATE",
resolveValue("xml:DeliveryPlan:/DeliveryPlan/@DeliveryPlanDate));
```

getDateValue

説明

この JSP 関数は、XML 形式で XML から日付を取り出します。現在のロケールの形式ではありません。これは、一般的に表のソート用に列内のカスタム sortValue 属性をソートするために使用されます。

構文

String getDateValue(String bindingStr);

入力パラメーター

bindingStr - 必須。日付ストリング内に解決する必要があるバインディング・ストリング。

出力パラメーター

以下の値を持つ日付ストリング (YYYYMMDDHH24MISS 構造)。

YYYY - 必須。年の 4 桁表示 (例: 2002)。

MM - 必須。月の 2 桁表示 (例: 5 月は 05)。

DD - 必須。日付の 2 桁表示 (例: 5 日は 05)。

HH24 - 必須。24 時間スケールの時間の 2 桁表示 (例: 午後 4 時は 16)。

MI - 必須。分の 2 桁表示。

SS - 必須。秒の 2 桁表示。

例

この例は、オーダーに対するアラートのリストに、getDateValue() 関数が、ユーザーによる後続のクライアント・サイド・ソートのためにこのフォーマットで日付を保管する方法を示します。

```
<table class="table" editable="false" width="100%" cellspacing="0">
  <thead>
     <tr>
       <td class="tablecolumnheader"><yfc:i18n>Alert_ID</yfc:i18n></td>
       <td class="tablecolumnheader"><yfc:i18n>Raised_On</yfc:i18n></td>
     </tr>
  </thead>
  <tbody>
     <yfc:loopXML binding="xml:/InboxList/@Inbox" id="Inbox">
        <tr>
          <td class="tablecolumn">
              <yfc:getXMLValue binding="xml:/Inbox/@InboxKey"/>
          </td>
          <td class="tablecolumn" sortValue=
"<%=getDateValue("xml:Inbox:/Inbox/@GeneratedOn")%>">
               <yfc:getXMLValue binding="xml:/Inbox/@GeneratedOn"/>
          </td>
```

```
        </tr>
      </yfc:loopXML>
  </tbody>
</table>
```

getDBString

説明

この JSP 関数は、入力ストリングのローカライズ済みバージョンを表すストリングを返します。このメソッドに対する入力は、YFS_LOCALIZED_STRINGS テーブル内で翻訳済みであるデータ・ストリングである必要があります。

構文

String getDBString(String inString)

入力パラメーター

inString - 必須。翻訳対象のストリング。このストリングは、YFS_LOCALIZED_STRINGS テーブル内で翻訳される必要があります。

例

この例では、singleDocType 要素の Description 属性の翻訳済みバージョンを表示する方法を示しています。

```
<%=getDBString(singleDocType.getAttribute("Description"))%>
```

getDetailHrefOptions

説明

この JSP 関数は通常、別の詳細ビューを開く内部パネル内のリンクを生成するために使用されます。リンクは、Link タイプのリソースとしてモデル化されます。このリソースは他の任意の詳細ビューを指し示すことができ、これは「リソース階層」ツリーを通じて構成できます。この関数は、<a> タグ内で使用してください。この関数は, 内部パネル内でのみ (すなわち詳細ビュー内でのみ) 使用できます。

構文

String getDetailHrefOptions(String linkIdSuffix, String entityKey, String extraParams)

入力パラメーター

linkIdSuffix - 必須。リンク ID の接尾辞。 Link タイプのリソースの名前は、*<現在の内部パネルのリソース ID><接頭辞>* という形式で指定されます。例えば、現在の内部パネルのリソース ID が YOMD010I01 である場合は、リンク ID は YOMD010I01L01 であり、接頭辞は L01 です。接頭辞の L01 のみを渡してください。

entityKey - 必須。このリンクを選択することによって呼び出されるビューに渡される必要のあるキー (makeXMLInput という JSP タグを介して生成されます)。

extraParams - 必須。<a href> タグ用に生成される URL の末尾に付加される追加パラメーターを格納しているストリング。このストリングは、アンパーサンド ("&") で始まる必要があり、*name=value* という形式の「名前/値」ペアを含んでいる必要があります。URL で渡すことができるデータにはサイズ制限があるため、このパラメーターは、どうしても必要な場合に限って使用してください。一般に、それぞれのビューは前述のキーのみを受け取って、そのキーに基づいて API から他の詳細情報を取得する必要があります。

出力パラメーター

href="" 属性と onclick="" 属性が含まれたストリングは、HTML 内の <a> タグ内に組み込まれる必要があります。

Link タイプのリソースは、アクセス権に基づいて制御されません。ただし、リンクが指し示すビューはアクセス権に基づいて制御されます。それでも、この関数は <a> タグ内で呼び出されるため、リンクが指し示すビューに対するアクセス権をユーザーが持っているかどうかにかかわらず、リンクは生成されます。ユーザーがこのリンクを選択した場合は、表示されるビューには「アクセスが拒否されました」というメッセージが表示されます。

例

この例では、getDetailHrefOptions() 関数が、オーダーの警告のリストから「警告の詳細」ビューへのハイパーリンクを生成する方法を示しています。

```
<table class="table" editable="false" width="100%" cellspacing="0">
 <thead>
   <tr>
     <td sortable="no" class="checkboxheader">
        <input type="checkbox" name="checkbox" value="checkbox"
onclick="doCheckAll(this);"/>
      </td>
      <td class="tablecolumnheader"><yfc:i18n>Alert_ID</yfc:i18n>
      </td>
   </tr>
 </thead>
 <tbody>
   <yfc:loopXML binding="xml:/InboxList/@Inbox" id="Inbox">
   <tr>
      <yfc:makeXMLInput name="inboxKey">
      <yfc:makeXMLKey binding="xml:/Inbox/@InboxKey"
value="xml:/Inbox/@InboxKey"/>
      </yfc:makeXMLInput>
      <td>
         <input type="checkbox" value='<%=
getParameter("inboxKey")%>' name="EntityKey"/>
      </td>
      <td class="tablecolumn">
          <a <%=getDetailHrefOptions("L01", getParameter("inboxKey"),"")%>
              <yfc:getXMLValue binding="xml:/Inbox/@InboxKey"/>
          </a>
       </td>
   </tr>
   </yfc:loopXML>
 </tbody>
</table>
```

getDetailHrefOptions (追加パラメーターあり)

説明

この JSP 関数は、getDetailHrefOptions() とほぼ同じですが、追加のパラメーターを受け取るという点のみが異なります。この追加のパラメーターを使用することで、同じハイパーリンクを条件に応じて異なるビューにリンクさせることができます。まず、条件付き接頭辞を除いた部分のリンク ID が同じである複数のリンク・リソースを同じ内部パネル下で構成します。例えば、あるビューを指し示す YOMD010I01L010001 という ID を持つ 1 つのリンクを構成して、異なるビューを指し示す YOMD010I01L010002 という ID を持つもう 1 つのリンクを構成します。次に JSP 内で、この関数を <a> タグ内で使用して、conditionalLinkId パラメーターとして異なる値を渡すことで、条件に応じて異なるビューにリンクできます。

構文

getDetailHrefOptions(String linkIdSuffix, String conditionalLinkId, String entityKey, String extraParams)

入力パラメーター

linkIdSuffix - 必須。リンク ID の接尾辞。 Link タイプのリソースの名前は、*<現在の内部パネルのリソース ID><接頭辞>* という形式で指定されます。例えば、現在の内部パネルのリソース ID が YOMD010I01 である場合は、リンク ID は YOMD010I01L01 であり、接頭辞は L01 です。接頭辞の L01 のみを渡してください。

conditionalLinkId - 必須。リンク ID の接頭辞の部分。条件に応じて異なるビューにリンクするために使用されます。

entityKey - 必須。このリンクを選択することによって呼び出されるビューに渡される必要のあるキー (makeXMLInput という JSP タグを介して生成されます)。

extraParams - 必須。<a href> タグ用に生成される URL の末尾に付加される追加パラメーターを格納しているストリング。このストリングは、アンパーサンド (&) で始まる必要があり、*name=value* という形式の「名前/値」ペアを含んでいる必要があります。URL で渡すことができるデータにはサイズ制限があるため、このパラメーターは、どうしても必要な場合にのみ使用してください。一般に、それぞれのビューは前述のキーのみを受け取って、そのキーに基づいて API から他の詳細情報を取得する必要があります。

出力パラメーター

HTML 内の <a> タグ内に組み込まれる必要がある href="" 属性と onclick="" 属性を格納しているストリング。

Link タイプのリソースは、アクセス権に基づいて制御されません。ただし、リンクが指し示すビューはアクセス権に基づいて制御されます。それでも、この関数は <a> タグ内で呼び出されるため、リンクが指し示すビューに対するアクセス権をユーザーが持っているかどうかにかかわらず、リンクは生成されます。ユーザーがこ

のリンクを選択した場合は、表示されるビューには「アクセスが拒否されました」というメッセージが表示されます。

例

この関数が役に立つのは、画面上の特定のハイパーリンクを複数の伝票種別にまたがってリンクさせる必要がある場合です。例えば、「配送計画」画面上の出荷のリストは、複数の異なる伝票種別の出荷である可能性があります (オーダーおよび購入オーダー (purchase order) の出荷)。これら 2 つのタイプの出荷について表示される必要がある詳細ビューはそれぞれ異なります。出荷の伝票種別は、conditionalLinkId として使用できます。

```
<a <%=getDetailHrefOptions("L01", getValue("Shipment",
"xml:/Shipment/@DocumentType"), getParameter("shipmentKey"), "")%>>
  <yfc:getXMLValue binding="xml:/Shipment/@ShipmentNo"/>
</a>
```

この例では、getValue() 関数の呼び出しによって、conditionalLinkId として使用されている出荷の伝票種別が返されることを示しています。この例が機能するには、この JSP を使用している内部パネルが、次のプロパティーを使用して定義されているリソースをリンクする必要があります。

```
Link 1: ID="YDMD100I02L010001" View ID="YOMD330"
Link 2: ID="YDMD100I02L010005" View ID="YOMD7330"
getDoubleFromLocalizedString -
```

getDoubleFromLocalizedString

説明

この JSP 関数は、特定ロケール用の形式で数値を格納しているストリングで表現される倍精度値を返します。

この関数は, getLocalizedStringFromDouble() 関数の逆の処理を実行します。

構文

double getDoubleFromLocalizedString(YFCLocale aLocale, String sVal)

入力パラメーター

aLocale - 必須。数値を特定ロケール用の形式に変換することを希望する対象の YFCLocale オブジェクト。

sVal - 必須。形式変換された数値表現を格納しているストリング。

出力パラメーター

形式変換されていない数値が格納されている倍精度値。

JSP の使用法

この例では、形式変換された倍精度値が格納された sTotalInternalUnassignedDemand という名前のストリング変数を、まず倍精度値に変換することによって、ゼロと比較する方法を示しています。

```
<% if ((getDoubleFromLocalizedString(getLocale(),
sTotalInternalUnassignedDemand)) > 0) {%>
  <table  border="1" style="border-color:Black" cellspacing="2" cellpadding="2"
    bgcolor="<yfc:getXMLValue
    binding="xml:/InventoryInformation/Item/InventoryTotals/Demands/@Total
    InternalUnassignedDemand" />">
    <tr>
      <td style="height:10px;width:15px"></td>
    </tr>
  </table>
<%}%>
```

getElement

説明

この JSP 関数は、指定された名前空間内にある YFCElement オブジェクトを取得します。この関数を使用すると、YFCElement のハンドルを取得して、その後に YFCElement オブジェクト内で XML を操作できます。

YFCElement は、Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales DOM ユーティリティー・パッケージに含まれています。このパッケージに含まれている API を表示するには、Javadoc を参照してください。

構文

YFCElement getElement(String nameSpace)

入力パラメーター

nameSpace - 必須。取得する必要のある YFCElement が含まれている名前空間。

出力パラメーター

YFCElement - 必須。指定された名前空間内にある YFCElement オブジェクト。

例

この例では、スケジュール操作のアクティブ状態または非アクティブ状態のどちらによってこの関数が使用されるのかを「返品の詳細」ビューが制御する方法を示しています。

スケジュール操作は、ドラフト・オーダー (draft order) に対しては無効です。

getOrderDetail() API は、XML 内の DraftOrderFlag 属性を返します。

このフラグは、オーダーがドラフト・オーダーの場合は Y であり、そうでない場合は N です。

これは、反対の意味を持つ別のフラグに変換される必要があります。その結果として、ConfirmedFlag という属性を使用します。この属性は、オーダーがドラフト・オーダーのときは N であり、オーダーがドラフト・オーダーでなくなると Y になります。

```
<%
    YFCElement elem=getElement("Order");
    if (elem != null) {
     //Flip the draft order flag into confirmed flag.
     elem.setAttribute("ConfirmedFlag", !isTrue("xml:/Order/@DraftOrderFlag"));
                        }
%>
```

getImageOptions

説明

この JSP 関数は、HTML 内でイメージ・タグを構成するために使用されます。

Java 定数ファイルによって、そのイメージのパスとアイコンが一箇所にまとめて保持されます。このパスが /smcfs*application_name*/console/icons で始まる場合は、そのイメージ・ファイルはまず /extensions/global/webpages/icons/yantraiconsbe.jar (ローカライズ済みアイコンの JAR ファイル) 内で検索されてから、次に /webpages/yfscommon/ yantraiconsbe.jar (ローカライズ済みアイコンの JAR ファイル) 内で検索されます。指定するパスは、JAR ファイル内のイメージ・ファイルのパスです。

このパスが /smcfs*application_name*/console/icons で始まらない場合は、EAR ファイルで指定された場所からイメージ・ファイルが取得されます。イメージは、/console/icons/ ディレクトリーにあるカスタム・アイコンの JAR ファイル (yantraiconsbe.jar) 内に配置することを強くお勧めします。

指定するパスは、JAR ファイル内のイメージ・ファイルのパスです。

変更ルールの考慮事項に基づいて非表示になる可能性のあるイメージを使用することを希望する場合は、yfsgetImageOptions() 関数を使用します。

構文

getImageOptions(imgfilewithpath, alt)

入力パラメーター

imgfilewithpath - 必須。イメージのファイルの絶対パス。

alt - 必須。イメージの alt 属性として使用するストリング。このストリングは、イメージを画面に表示できない場合に表示されます。

JSP の使用法

```
<img class="lookupicon" name="search"
<%=getImageOptions ("smcfsapplication_name/console/icons/shipnode.jpg"
"Search_for_Organization") %> />
```

getLocale

説明

この JSP 関数は、Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales にログインしているユーザーのロケールを表す YFCLocale オブジェクトを返します。

構文

YFCLocale getLocale()

入力パラメーター

なし。

出力パラメーター

ログインしているユーザーのロケールを表す YFCLocale オブジェクト。

JSP の使用法

この例では、getLocale 関数を getDoubleFromLocalizedString 関数と組み合わせて使用する方法を示しています。

```
<%
if ((getDoubleFromLocalizedString(getLocale(),
sTotalInternalUnassignedDemand)) > 0) {%>
 <table  border="1" style="border-color:Black" cellspacing="2" cellpadding="2"
  bgcolor="<yfc:getXMLValue
  binding="xml:/InventoryInformation/Item/InventoryTotals/Demands/
@TotalInternalUnassignedDemand" />">
   <tr>
     <td style="height:10px;width:15px"></td>
   </tr>
 </table>
<%}%>
```

getLocalizedStringFromDouble

説明

この JSP 関数は、ストリング値を返して、特定ロケール用の正しい形式でその値を表示します。

Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales は常に、ログインしているユーザーのロケールに対応した形式で数値データを表示します。ユーザーに表示する必要がある 10 進値が、まだどのロケールの形式にも変換されていない場合は、この関数を使用して、正しい形式に変換された 10 進値表現を表すストリングを取得してください。

この関数は, getDoubleFromLocalizedString() 関数の逆の処理を実行します。

構文

String getLocalizedStringFromDouble(YFCLocale aLocale, double aDblVal)

入力パラメーター

aLocale - 必須。数値を特定ロケール用の形式に変換することを希望する対象の YFCLocale オブジェクト。

aDblVal - 必須。特定ロケール用の形式に変換することを希望する数値 (小数を含む)。

出力パラメーター

正しい形式に変換された数値表現を格納しているストリング。

JSP の使用法

この例では、2500.75 という数値のローカライズ済み形式を取得する方法を示しています。使用されているロケールが en_US の場合は、sBalance 変数は 25,00.75 となります。

```
String sBalance = getLocalizedStringFromDouble(locale, 2500.75);
```

getLocalizedStringFromInt

説明

この JSP 関数は、整数値を返して、特定ロケール用の正しい形式でその値を表示します。

Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales は常に、ログインしているユーザーのロケールに対応した形式で数値データを表示します。ユーザーに表示する必要がある 10 進値が、まだどのロケールの形式にも変換されていない場合は、この関数を使用して、正しい形式に変換された整数値表現を表すストリングを取得してください。

構文

String getLocalizedStringFromInt(YFCLocale aLocale, int intVal)

入力パラメーター

aLocale - 必須。数値を特定ロケール用の形式に変換することを希望する対象の YFCLocale オブジェクト。

intVal - 必須。特定ロケール用の形式に変換することを希望する整数。

出力パラメーター

正しい形式に変換された数値表現を格納しているストリング。

JSP の使用法

この例では、quantity という名前の整数変数を <td> タグ内に表示する方法を示しています。

```
<td class="protectednumber">
  <%=getLocalizedStringFromInt(getLocale(), quantity)%>
</td>
```

getLoopingElementList

説明

この JSP 関数は、loopXML JSP タグの代替として使用できます。

お使いのアプリケーション・サーバーが JSP 仕様バージョン 1.1 までしかサポートしていない場合に、ループ内に別の JSP をインクルードする必要がある場合は (jsp:include を使用して)、この関数を使用してください。

構文

ArrayList getLoopingElementList(String binding)

入力パラメーター

binding - 必須。XML 内の反復対象の要素への XML バインディング。

出力パラメーター

後でループ内で使用できる要素のリストが格納された ArrayList。

JSP の使用法

この例では、xml:PromiseList:/Promise/Options/@Option 要素についてループします。このループが反復されるたびに、/om/lineschedule/list/lineschedule_list_option.jsp という JSP ファイルがインクルードされます。

ループ要素は、pageContext の属性内に設定されていることに注目してください。これにより、インクルードされた JSP 内でループ要素を使用可能になります。

```
<td colspan="6" style="border:1px ridge black">
   <%  ArrayList optList = getLoopingElementList("xml:PromiseList:/Promise/Options/@Option");
for (int OptionCounter = 0; OptionCounter < optList.size(); OptionCounter++) {
   YFCElement singleOpt = (YFCElement) optList.get(OptionCounter);
   pageContext.setAttribute("Option", singleOpt); %>
 <% request.setAttribute("Option",
(YFCElement)pageContext.getAttribute("Option")); %>
<jsp:include page="/om/lineschedule/list/lineschedule_list_option.jsp"
flush="true">
</jsp:include>
   <% } %>
</td>
```

getNumericValue

説明

この JSP 関数は、現在のロケールの形式ではなく、元の XML 形式の XML 出力から数値を取得します。この JSP 関数は通常、テーブルのソートのためにカスタム sortValue 属性を列に格納するために使用されます。

構文

String getNumericValue(String bindingStr)

入力パラメーター

bindingStr - 必須。解決されて数値ストリングに変換される必要があるバインディング・ストリング。

出力パラメーター

Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales の XML 形式の数値ストリング (小数部分を含む)。

例

この例では、getNumericValue() 関数を使用して、優先順位を XML 形式で保管する方法を示しています。これにより、後でクライアント側でユーザーがオーダーの警告のリストをソートできるようになります。

```
<table class="table" editable="false" width="100%" cellspacing="0">
  <thead>
    <tr>
      <td sortable="no" class="checkboxheader">
         <input type="checkbox" name="checkbox" value="checkbox"
onclick="doCheckAll(this);"/></td>
      <td class="tablecolumnheader"><yfc:i18n>Alert_ID</yfc:i18n></td>
      <td class="tablecolumnheader"><yfc:i18n>Priority</yfc:i18n></td>
    </tr>
  </thead>
  <tbody>
    <yfc:loopXML binding="xml:/InboxList/@Inbox" id="Inbox">
    <tr>
       <yfc:makeXMLInput name="inboxKey">
           <yfc:makeXMLKey binding="xml:/Inbox/@InboxKey"
value="xml:/Inbox/@InboxKey"/>
       </yfc:makeXMLInput>
       <td>
         <input type="checkbox" value='<%=getParameter("inboxKey")%>'
name="EntityKey"/>
        </td>
        <td class="tablecolumn">
           <a <%=getDetailHrefOptions("L01", getParameter("inboxKey"),"")%>>
                <yfc:getXMLValue binding="xml:/Inbox/@InboxKey"/>
           </a>
        </td>
        <td class="tablecolumn"
sortValue="<%=getNumericValue("xml:Inbox:/Inbox/@Priority")%>"><yfc:
getXMLValue binding="xml:/Inbox/@Priority"/>
        </td>
    </tr>
    </yfc:loopXML>
  </tbody>
</table>
```

getParameter

説明

この JSP 関数は、指定されたパラメーターの値を pageContext() 関数から取得して、次の順序で要求します。

pageContext.getAttribute() **-> 見つからない場合 ->** request.getAttribute() **-> 見つからない場合 ->** request.getParameter()

この関数は通常、yfc:makeXMLKey や yfc:loopXML などのさまざまなプレゼンテーション・フレームワーク JSP タグを使用しているときに、指定されたパラメーターを抽出するために使用されます。

構文

String getParameter(String paramName);

入力パラメーター

paramName - 必須。値を取得するパラメーターの名前。

例

この例では、デフォルトの詳細ビューを開くハイパーリンクされたオーダー番号が「オーダー・リスト」ビューで表示される方法を示しています。yfc:makeXMLInput という JSP タグは、指定されたキーを使用して XML を作成および保管します。XML は、getParameter() 関数を使用して抽出できます。

```
<table class="table" editable="false" width="100%" cellspacing="0">
 <thead>
    <tr>
      <td sortable="no" class="checkboxheader">
          <input type="checkbox" name="checkbox" value="checkbox"
onclick="doCheckAll(this);"/>
      </td>
      <td class="tablecolumnheader"><yfc:i18n>Order_#</yfc:i18n></td>
      <td class="tablecolumnheader"><yfc:i18n>Enterprise</yfc:i18n></td>
    </tr>
 </thead>
 <tbody>
   <yfc:loopXML binding="xml:/OrderList/@Order" id="Order">
     <tr>
       <yfc:makeXMLInput name="orderKey">
           <yfc:makeXMLKey binding="xml:/Order/@OrderHeaderKey"
value="xml:/Order/@OrderHeaderKey" />
        </yfc:makeXMLInput>
        <td class="checkboxcolumn">
           <input type="checkbox" value='<%=getParameter("orderKey")%>'
name="EntityKey"/>
         </td>
         <td class="tablecolumn"><a
href="javascript:showDetailFor('<%=getParameter("orderKey")%>');">
             <yfc:getXMLValue binding="xml:/Order/@OrderNo"/></a>
          </td>
          <td class="tablecolumn"><yfc:getXMLValue
binding="xml:/Order/@EnterpriseCode"/>
          </td>
     </tr>
  </yfc:loopXML>
 </tbody>
</table>
```

getRadioOptions

説明

この JSP 関数は、変更ルールを考慮する必要がない場合に、ラジオ・ボタンを XML バインドします。

構文

String getRadioOptions(String name)

String getRadioOptions(String name, String checked)

String getRadioOptions(String name, String a_checked, String a_value)

入力パラメーター

name - 必須。ラジオ入力の name 属性の値。バインディングまたはリテラルを指定できます。

checked - 必須。一致する ID を持つ要素の checked 属性を設定します。リテラルを指定する必要があります。

value - 必須。ラジオ入力の value 属性の値。バインディングまたはリテラルを指定できます。

注: name パラメーターのみが渡された場合は、value パラメーターの値はデフォルトで、name パラメーターで指定されたのと同じ値になります。

JSP の使用法

```
<input type="radio" <%=getRadioOptions("xml:Order:/OrderAddlinfo/@Country",
"US", "xml:Order:/OrderAddlinfo/@Country" )%>>United States</input>
```

生成される HTML

```
<input type="radio" name="US" value="US" CHECKED>United States</input>
```

getRequestDOM

説明

この JSP 関数は、この関数の呼び出し時に使用可能な要求パラメーターから作成できたすべての XML 要素が含まれた YFCElement を生成します。特定の画面が「ポスト」されると、その画面上の入力フィールドのうち XML 値にバインドされたフィールドに、この関数を介してアクセスできます。生成される YFCElement には、次の構造が含まれています。

```
<root>
   <namespace1 ... />
   <namespace2 .../>
   ...
</root>
```

この関数が役に立つのは、何らかの JSP 処理のために適切な XML 構造内のポストされた XML 値にアクセスする必要がある場合です。

構文

String getRequestDOM()

入力パラメーター

なし。

出力パラメーター

この関数が呼び出されたときに使用可能であった要求パラメーターから作成できたすべての XML が含まれた YFCElement。

JSP の使用法

次の例では、以下の入力フィールドがポストされた場合の getRequestDOM の出力を示しています。

```
<input type="hidden" name="xml:/Order/@OrderNo" value="Order0001"/>
<input type="hidden" name="xml:/Order/@OrderType" value="Customer"/>
<input type="hidden" name="xml:/User/@UserName" value="user01"/>
```

getRequestDOM の出力:

```
<root>
  <Order OrderNo="Order0001" OrderType="Customer"/>
  <User UserName="user01"/>
</root>
```

getSearchCriteriaValueWithDefaulting

説明

この JSP 関数は、検索条件フィールドでデフォルト値を使用することを希望する状況に対処します。この特別な関数が必要な理由は、ユーザーがある画面に初めてアクセスした場合と、この属性が特別に「ブランク」として保存されている保存した検索 (saved search) が読み込まれた場合を区別する必要があるからです。このような保存した検索が検索ビューに読み込まれた場合は、デフォルト値を表示してはいけません。

構文

String getSearchCriteriaValueWithDefaulting(String binding, String defaultBinding)

入力パラメーター

binding - 必須。状況に応じてデフォルト値が表示される必要がある検索条件フィールドのターゲット XML バインディング。

defaultBinding - 必須。ユーザーがその検索画面に初めてアクセスする場合にデフォルト値の取得元となる XML バインディング (または静的値)。

出力パラメーター

その検索条件フィールドに表示される値を格納しているストリング。

JSP の使用法

この例では、「オーダーの検索」画面の「エンタープライズ・コード」コンボ・ボックスに、ログインしているユーザーの組織の主要エンタープライズ・コード (デフォルト値) を表示する方法を示しています。

```
<td class="<%=inputTdClass%>" nowrap="true">
   <select class="combobox" <%=getComboOptions(enterpriseCodeBinding)%>>
   <yfc:loopOptions
binding="xml:CommonEnterpriseList:/OrganizationList/@Organization"
name="OrganizationCode" value="OrganizationCode"
selected='<%=getSearchCriteriaValueWithDefaulting("xml:/Order/@EnterpriseCode",
"xml:CurrentOrganization:/Organization/@PrimaryEnterpriseKey")%>'/>
    </select>
</td>
```

getTextAreaOptions

説明

この JSP 関数は、変更ルールを考慮する必要がない場合に、テキスト領域を XML バインドします。

構文

getTextAreaOptions(String name)

入力パラメーター

name - 必須。テキスト領域ボックス入力の name 属性の値。バインディングまたはリテラルを指定できます。

JSP の使用法

```
<=%getTexAreaOptions("xml:/Order/Instructions/Instruction/@InstructionText")%>
```

getTextOptions

説明

この JSP 関数は、変更ルールを考慮する必要がない場合に、テキスト入力フィールドを XML バインドします。

構文

String getTextOptions(String name)

String getTextOptions(String name, String value)

String getTextOptions(String name, String value, String defaultValue)

この関数を挿入すると、テキスト入力の実際に表示される値を設定したり、フォームがポストされたときに値の送り先となる XML 内のパスと属性を設定したりできます。

値とデフォルト値が指定されない場合は、デフォルト値は blanks になり、値はデフォルトである name になります。

入力パラメーター

name - 必須。フォームがポストされたときに入力テキスト内の値の送り先となるターゲット XML 内のパス。その後、ターゲット XML は、サービス定義フレームワークを通じて適切な API に渡されます。

value - 必須。入力テキストとして表示する内容を指定します。バインディングまたはリテラルを指定できます。デフォルト値は name です。

defaultValue - 必須。このパラメーターでは、値のバインディングが何も返さない場合のデフォルト値として使用されるリテラルまたはバインディングを指定できます。デフォルト値は blanks です。

例

この例では、バインドされた属性の値を表示する入力テキストが生成されて、フォームがポストされたときにバインドされた属性の値を設定します。以下はバインディングによって参照される XML です。

```
<Order>
   <addlInfo Country="US"></addlInfo>
</Order>
```

JSP の使用法

```
<input type="text" <%=getTextOptions("xml:Order:/OrderAddlinfo/@Country",
"xml:Order:/OrderAddlinfo/@Country", "USA")%> />
```

HTML の結果

値のバインディングが見つかった場合:

```
<input type="text" name=" xml:Order:/OrderAddlinfo/@Country " value="US"
Datatype='' size="10" Decimal='' OldValue="United States"/>
```

名前のバインディングが見つからなかった場合:

```
<input type="text" name="US" value="USA" Datatype='' size="10" Decimal=''
OldValue="United States"/>
```

編集可能なリスト内での getTextOptions の使用:

- アンダースコアー ("_") とカウンターを getTextOptions の最初のパラメーターの後ろに付加する必要があります。
- このカウンターの名前は loopXML タグで指定された ID 属性の値です。この ID 属性の値は、ループ場所である子ノードの名前と同じになるように設定してください。

例

テキスト・ボックスの挿入。

```
<input type="text" class="unprotectedinput"
<%=getTextOptions("xml:/Order/OrderLines/OrderLine_" + OrderLineCounter +
"/@ShipNode", "xml:/OrderLine/@ShipNode")%>/>
```

getUITableSize

説明

この JSP 関数は、入力として渡された属性の UI 幅を返します。これを使用して、画面内のテーブルの列幅を設定することで、当アプリケーション全体にわたって列サイズを統一できます。使用される幅は、属性のデータ・タイプ定義から取得されます。拡張データ・タイプを参照してください。

構文

getUITableSize(String binding)

パラメーター

XML 内の現在の属性へのパス。

注: これは、リスト・テーブルのすべての <thead> タグ内のすべての <td> タグの style 属性で使用される必要があります。

JSP の使用法

```
style="width:<%=getUITableSize("xml:/Order/@OrderDate")%>"
```

getValue

説明

この JSP 関数は、指定されたバインディング・ストリングの値を指定された XML 名前空間から取得します。

構文

String getValue(String xmlName,String binding)

入力パラメーター

xmlName - 必須。XML の名前空間。この名前空間がバインディング・ストリングに含まれている場合でも、このパラメーターを指定する必要があります。

binding - 必須。バインディング・ストリング。

例

この例では、「在庫調整 (Inventory Adjustment)」画面で API の出力から供給タイプが抽出される方法を説明しています。

```
<%
   String
supplyType=getValue("Item","xml:/Item/Supplies/InventorySupply/@SupplyType");
%>
```

goToDetailView

説明

この JSP 関数を使用すると、JSP ページで指定されたロジックに基づいて、条件に応じて異なる詳細ビューを表示できます。この関数と組み合わせて使用できるのは、「リダイレクター・ビュー」として定義された詳細ビューのみです。この関数が役に立つのは、何らかのロジックに基づいて (場合によっては API 呼び出しの出力に基づいて)、条件に応じて異なる詳細ビューにアクセスする必要がある場合です。リダイレクター・ビューの JSP アンカー・ページでは、この関数を使用して、ユーザーに表示される詳細ビューに最終的にアクセスします。

構文

void goToDetailView(HttpServletResponse response, String viewGroupId)

入力パラメーター

response - 必須。応答オブジェクト。リダイレクター JSP から得られたのと同じ状態で「応答」オブジェクトを渡します。

viewGroupId - 必須。エンド・ユーザーに表示されるビュー・グループ ID。

出力パラメーター

なし。

JSP の使用法

この例では、「出荷の詳細」リダイレクター・ビューのアンカー・ページである完全な JSP を示しています。表示されている出荷が提供サービス (provided service) の出荷である場合は、異なる詳細ビューが表示されます。getShipmentDetails() API の出力に基づいて、表示されるビューが決定されることに注目してください。

```
<%@include file="/yfsjspcommon/yfsutil.jspf"%>
<%
   String sViewGrp = "YOMD710";
   if (isTrue("xml:/Shipment/@IsProvidedService")) {
        sViewGrp = "YOMD333";
   }
   goToDetailView(response, sViewGrp);
%>
```

isModificationAllowed

説明

この JSP 関数を使用して、現在のエンティティーの特定の属性について変更が許可されているかどうかを確認します。

構文

boolean isModificationAllowed(String name, String allowModBinding)

入力パラメーター

name - 必須。ターゲット XML 属性内のパス。現在のエンティティーのステータスの場合に、この属性が変更可能である場合は、この関数は `true` を返します。変更可能でない場合は、この関数は `false` を返します。

allowModBinding - 必須。現在のステータスに対して許可されている変更タイプが含まれた一連の要素を指し示すバインディング・ストリング。

JSP の使用法

この例では、現在のオーダーについて行の追加が許可されているかどうかに基づいて、行の動的追加機能を備えたテーブル・フッターをページに組み込む方法を示しています。

```
   <%if (isModificationAllowed("xml:/@AddInstruction",
"xml:/Order/AllowedModifications"))
{%>
   <tr>
     <td nowrap="true" colspan="3">
       <jsp:include page="/common/editabletbl.jsp" >
       </jsp:include>
     </td>
    </tr>
 <%}%>
```

isPopupWindow

説明

この JSP 関数は、現在のウィンドウがポップアップ・ウィンドウで表示されているかどうかを判別します。この関数を使用するのは、画面内のロジックがポップアップ・ウィンドウで表示されている場合は、そのロジックを通常とは異なるものにする必要がある場合です。

構文

isPopupWindow()

入力パラメーター

なし。

出力パラメーター

現在のウィンドウがポップアップ・ウィンドウで表示されているかどうかを示すブール値。

JSP の使用法

この例では、コンボ・ボックスに表示される選択済みの値は、この画面がポップアップ・ウィンドウで表示されているかどうかに応じて異なります。

```
<select name="xml:/Shipment/@EnterpriseCode" class="combobox">
   <% if (isPopupWindow()) { %>
     <yfc:loopOptions
     binding="xml:EnterpriseList:/OrganizationList/@Organization"
```

```
    name="OrganizationCode"
    value="OrganizationCode" selected="xml:/Shipment/@EnterpriseCode" />
  <% } else { %>
    <yfc:loopOptions
    binding="xml:EnterpriseList:/OrganizationList/@Organization"
    name="OrganizationCode"
    value="OrganizationCode"
    selected='<%=getSelectedValue("xml:/Shipment/@EnterpriseCode")%>' />
  <% } %>
</select>
```

isTrue

説明

この JSP 関数は、入力パラメーターで指定された属性の値が Y または true の場合は、true を返します。そうでない場合は、false を返します。この関数では大文字と小文字は区別されません。

構文

boolean isTrue(String bindingStr);

入力パラメーター

bindingStr - 必須。評価対象の属性を指定するバインディング・ストリング。

出力パラメーター

評価対象の属性の値が Y または true であるかどうかを示すブール値。

例

この例では、isTrue() 関数を使用して、xml:/OrderRelease/@HasDerivedParent バインディングで参照されている属性の値を調べます。

```
boolean isAgainstOrder=isTrue("xml:/OrderRelease/@HasDerivedParent");
```

isVoid

説明

この JSP 関数は、渡されたオブジェクトがヌルであるのかまたは空白のみを含んでいるのかを判別します。

構文

boolean isVoid(Object obj)

入力パラメーター

obj - 必須。ヌルであるのかまたは空白のみを含んでいるのかを確認する対象となるオブジェクト。

例

この例では、この関数を使用して特定の属性が空なのかどうかを確認する方法を示しています。

```
<% if (!isVoid(getParameter("ShowShipNode"))) {%>
<tr>
   <td class="detaillabel" ><yfc:i18n>Ship_Node</yfc:i18n></td>
   <td class="protectedtext"><yfc:getXMLValue
binding="xml:/InventoryInformation/Item/@ShipNode"
name="InventoryInformation"></yfc:getXMLValue></td>
</tr>
<%}%>
```

resolveValue

説明

この JSP 関数は、指定されたバインディング・ストリングの値を指定された YFCElement から取得します。

構文

String resolveValue(String binding)

入力パラメーター

binding - 必須。バインディング・ストリング。バインディング・ストリングには名前空間を格納できます。

例

この例では、この関数を使用して、バインディング・ストリングが指し示す値を解決する方法を示しています。

```
<%
 String reqshipdate=resolveValue("xml:OrderEntry:/Order/@ReqShipDate");
%>
```

showEncryptedCreditCardNo

説明

この JSP 関数は、暗号化されたクレジット・カード番号を表す値をディスプレイに返します。

構文

showEncryptedCreditCardNo(String CreditCardNo)

入力パラメーター

CreditCardNo - 必須。クレジット・カード番号の下 4 桁を格納しているストリング。

例

```
<%=showEncryptedCreditCardNo(resolveValue
("xml:/PaymentMethod/@DisplayCreditCardNo"))%>
```

userHasOverridePermissions

説明

この JSP 関数は、現行ログイン・ユーザーに、変更ルール構成のオーバーライド権限があるかどうかを指定します。

構文

boolean userHasOverridePermissions()

yfsGetCheckBoxOptions

説明

この JSP 関数は、変更ルールを考慮する必要がある場合にチェック・ボックスに XML バインディングを実行します。

構文

String yfsGetCheckBoxOptions(String name,String a_checked, String a_value, String allowModBinding)

入力パラメーター

name - 必須。フォームをポストするときの入力テキスト内の値の送信先であるターゲット XML 内のパス。このターゲット XML は、サービス定義フレームワークを通して、該当する API に渡されます。

checked - 必須。value 属性の値がこの値と等しい場合、checked 属性は `true` に設定されます。

value - 必須。チェック・ボックス入力の value 属性の値。バインディングまたはリテラルを指定できます。

allowModBinding - 必須。現行ステータスに対して許可される変更タイプを含むエレメントのセットをポイントするバインディング・ストリング。

JSP での使用法

```
<input class="checkbox" type="checkbox"
<%=yfsGetCheckBoxOptions("xml:/Order/@ChargeActualFreightFlag",
"xml:/Order/@ChargeActualFreightFlag","Y","xml:/Order/AllowedModifications")
%>/>
```

yfsGetComboOptions

説明

この JSP 関数は、変更ルールを考慮する必要がある場合にコンボ・ボックスに XML バインディングを実行します。

構文

String yfsGetComboOptions(String name, String allowModBinding)

String yfsGetComboOptions(String name, String value, String allowModBinding)

入力パラメーター

name - 必須。フォームをポストするときの入力テキスト内の値の送信先であるターゲット XML 内のパス。このターゲット XML は、サービス定義フレームワークを通して、該当する API に渡されます。

value - 必須。入力テキストとして表示するものを指定します。バインディングまたはリテラルを指定できます。

allowModBinding - 必須。現行ステータスに対して許可される変更タイプを含むエレメントのセットをポイントするバインディング・ストリング。

JSP での使用法

```
<select <% if (isVoid(modifyView)) {%> <%=getProtectedComboOptions()%> <%}%>
<%=yfsGetComboOptions("xml:/Order/@ScacAndServiceKey",
"xml:/Order/AllowedModifications")%>>
   <yfc:loopOptions binding="xml:/ScacAndServiceList/@ScacAndService"
name="ScacAndServiceDesc" value="ScacAndServiceKey"
selected="xml:/Order/@ScacAndServiceKey"/>
</select>
```

yfsGetImageOptions

説明

この JSP 関数は、HTML にイメージ・タグを作成します。getImageOptions() 関数とは異なり、パラメーターとして渡される XML 属性の変更が許可されているかどうかに基づいて、イメージが非表示になる場合があります。

Java 定数ファイルで、イメージのパスとアイコンが一元的に保持されます。イメージ・パスが /smcfs*application_name*/console/icons で始まっている場合、そのイメージ・ファイルはまず /extensions/global/webpages/icons/yantraiconsbe.jar (またはローカライズされたアイコンの JAR ファイル) 内で検索されてから、/webpages/yfscommon/yantraiconsbe.jar (またはローカライズされたアイコンの JAR ファイル) 内で検索されます。指定されるパスは、JAR ファイル内のイメージ・ファイルのパスです。

パスが /smcfs*application_name*/console/icons で始まっていない場合は、EAR ファイル内のロケーションからファイルが選択されます。/console/icons の下のカスタム・アイコンの JAR ファイル (yantraiconsbe.jar) にイメージを配置することを強くお勧めします。

指定されるパスは、JAR ファイル内のイメージ・ファイルのパスです。

構文

String yfsGetImageOptions(String src, String alt, String name, String allowModBinding)

パラメーター

src - 必須。アイコンの JAR ファイル内の、パスも含めたイメージ・ファイル名。

alt - 必須。イメージに使用するツール・ヒント。

name - 必須。ターゲット XML 属性内のパス。この関数は、現行エンティティーのステータスに基づいて、この属性の変更が許可されている場合にのみイメージを表示します。

allowModBinding - 必須。現行オーダー・ステータスに対して許可される変更タイプを含むエレメントのセットをポイントするバインディング・ストリング。

JSP での使用法

```
<img class="lookupicon" name="search" onclick="invokeCalendar(this);
return false"
<%=yfsGetImageOptions(YFSUIBackendConsts.DATE_LOOKUP_ICON, "Calendar",
"xml:/Order/@ReqShipDate", "xml:/Order/AllowedModifications")%>/>
```

yfsGetTemplateRowOptions

説明

この JSP 関数は、入力フィールドが編集可能テーブルのテンプレート行内に表示される場合にその入力フィールドに XML バインディングを実行します。テンプレート行は、プラス・アイコン (「+」) が編集可能テーブルで選択された場合に表示されます。

構文

String yfsGetTemplateRowOptions(String name, String allowModBinding, String modType, String controlType)

入力パラメーター

name - 必須。入力の name 属性の値。バインディングまたはリテラルを指定できます。

allowModBinding - 必須。エンティティーの現行ステータスに対して許可されるすべての変更タイプを含むエレメントのリストに解決されるバインディング・ストリング。

modType - 必須。現行コントロールに関連付けられた変更タイプ。

controlType - 必須。コントロールのタイプ。テキスト・ボックス、チェック・ボックス、またはテキスト領域を指定できます。

JSP での使用法

```
<input type="text"
<%=yfsGetTemplateRowOptions("xml:/Order/OrderLines/OrderLine_/Item/@ItemID",
"xml:/Order/AllowedModifications", "ADD_LINE", "text")%>/>
```

例

次の例は、「オーダーの詳細」ビューで 1 つのオーダーのオーダー明細リストにテンプレート行を保管するためにこの関数を使用する方法を示しています。

```
<tfoot>
   <tr style='display:none' TemplateRow="true">
    <td class="checkboxcolumn">
      <input type="hidden"
<%=getTextOptions("xml:/Order/OrderLines/OrderLine_/@Action", "",
"CREATE")%> />
    </td>
    <td class="tablecolumn"> </td>
    <td class="tablecolumn"> </td>
    <td class="tablecolumn" nowrap="true">
      <input type="text"
<%=yfsGetTemplateRowOptions("xml:/Order/OrderLines/OrderLine_/Item/@ItemID",
"xml:/Order/AllowedModifications", "ADD_LINE", "text")%>/>
      <img class="lookupicon"
onclick="templateRowCallItemLookup(this,'ItemID','ProductClass','UnitOfMeasure',
'item')" <%=getImageOptions(YFSUIBackendConsts.LOOKUP_ICON,
"Search_for_Item")%>/>
    </td>
    <td class="tablecolumn">
        <select
<%=yfsGetTemplateRowOptions("xml:/Order/OrderLines/OrderLine_/Item/
@ProductClass", "xml:/Order/AllowedModifications", "ADD_LINE", "combo")%>>
          <yfc:loopOptions
binding="xml:ProductClassList:/CommonCodeList/@CommonCode" name="CodeValue"
value="CodeValue" selected="xml:/Order/OrderLine/Item/@ProductClass"/>
        </select>
    </td>
    <td class="tablecolumn">
        <select
<%=yfsGetTemplateRowOptions("xml:/Order/OrderLines/OrderLine_/Item/
@UnitOfMeasure", "xml:/Order/AllowedModifications", "ADD_LINE", "combo")%>>
         <yfc:loopOptions
binding="xml:UnitOfMeasureList:/CommonCodeList/@CommonCode"  name="CodeValue"
value="CodeValue" selected="xml:/Order/OrderLine/Item/@UnitOfMeasure"/>
        </select>
        </td>
        <td class="tablecolumn"> </td>
        <td class="tablecolumn" nowrap="true">
          <input type="text"
<%=yfsGetTemplateRowOptions("xml:/Order/OrderLines/OrderLine_/@ReceivingNode",
 "xml:/Order/AllowedModifications", "ADD_LINE", "text")%>/>
         <img class="lookupicon" onclick="callLookup(this,'shipnode')"
 <%=getImageOptions(YFSUIBackendConsts.LOOKUP_ICON,
"Search_for_Recieving_Node")%>/>
        </td>
        <td class="tablecolumn" nowrap="true">
           <input type="text"
<%=yfsGetTemplateRowOptions("xml:/Order/OrderLines/OrderLine_/@ShipNode",
"xml:/Order/AllowedModifications", "ADD_LINE", "text")%>/>
```

```
              <img class="lookupicon" onclick="callLookup(this,'shipnode')"
<%=getImageOptions(YFSUIBackendConsts.LOOKUP_ICON, "Search_for_Ship_Node")%>/>
        </td>
        <td class="tablecolumn" nowrap="true">
           <input type="text"
<%=yfsGetTemplateRowOptions("xml:/Order/OrderLines/OrderLine_/@ReqShipDate",
"xml:/Order/AllowedModifications", "ADD_LINE", "text")%>/>
              <img class="lookupicon" onclick="invokeCalendar(this)"
<%=getImageOptions(YFSUIBackendConsts.DATE_LOOKUP_ICON, "Calendar")%>/>
        </td>
        <td class="numerictablecolumn">
          <input type="text"
<%=yfsGetTemplateRowOptions("xml:/Order/OrderLines/OrderLine_/@OrderedQty",
"xml:/Order/AllowedModifications", "ADD_LINE", "text")%>>
        </td>
        <td class="tablecolumn"> </td>
        <td class="tablecolumn"> </td>
     </tr>
<%if (isModificationAllowed("xml:/@AddLine","xml:/Order/AllowedModifications"))
{ %>
     <tr>
        <td nowrap="true" colspan="13">
          <jsp:include page="/common/editabletbl.jsp" >
          </jsp:include>
        </td>
     </tr>
     <%}%>
</tfoot>
```

---

yfsGetTextAreaOptions

説明

この JSP 関数は、変更ルールを考慮する必要がある場合にテキスト領域に XML バインディングを実行します。

構文

String yfsGetTextAreaOptions(String name, String a_value, String allowModBinding)

String yfsGetTextAreaOptions(String name, String allowModBinding)

パラメーター

name - 必須。フォームをポストするときの入力テキスト内の値の送信先であるターゲット XML 内のパス。このターゲット XML は、サービス定義フレームワークを通して、該当する API に渡されます。

value - 必須。入力テキストとして表示するものを指定します。バインディングまたはリテラルを指定できます。

allowModBinding - 必須。現行ステータスに対して許可される変更タイプを含むエレメントのセットをポイントするバインディング・ストリング。

JSP での使用法

```
<textarea class="unprotectedtextareainput" rows="3" cols="100"
<%=yfsGetTextAreaOptions("xml:/Order/Instructions/Instruction_" +
InstructionCounter +  "/@InstructionText","xml:/Instruction/@InstructionText",
"xml:/Order/AllowedModifications")%>><yfc:getXMLValue
binding="xml:/Instruction/@InstructionText"/></textarea>
```

yfsGetTextOptions

説明

この JSP 関数は、変更ルールを考慮する必要がある場合にテキスト入力フィールドに XML バインディングを実行します。

構文

String yfsGetTextOptions(String name, String allowModBinding)

String yfsGetTextOptions(String name, String value, String allowModBinding)

String yfsGetTextOptions(String name, String value, String defaultValue, String allowModBinding)

入力パラメーター

name - 必須。フォームをポストするときの入力テキスト内の値の送信先であるターゲット XML 内のパス。このターゲット XML は、サービス定義フレームワークを通して、該当する API に渡されます。

value - 必須。入力テキストとして表示するものを指定します。バインディングまたはリテラルを指定できます。

defaultValue - 必須。これにはバインディングまたはリテラルを指定でき、値バインディングが何も返さない場合はデフォルトに設定されます。

allowModBinding - 必須。これは、現行ステータスに対して許可される変更タイプを含むエレメントのセットをポイントするバインディング・ストリングです。

JSP での使用法

```
<input type="text" <%=yfsGetTextOptions("xml:/Order/@ReqShipDate",
"xml:/Order/AllowedModifications")%>/>
```

第 18 章 コンソール JSP インターフェースの JSP タグ・ライブラリー

callApi

説明

callApi JSP タグは、JSP ファイル内から API を呼び出します。ほとんどの場合、JSP ファイル内部から API 呼び出しを行う必要はありません。ただし、ほかのオプションがない場合があります。例えば、API がループ内で複数回呼び出される必要がある場合は、callApi JSP タグを使用します。

この JSP タグをビューで使用するときに、呼び出そうとしている API リソースに対してリソース構成画面の「自動実行のスキップ」チェック・ボックスを使用可能にすることができます。これにより、ビューが最初に開かれたときに、API が呼び出されるのを防ぎます。このオプションは、エンティティー・リソースの直下に作成された API リソースには使用できません。

属性

apiID - 必須。呼び出される API のリソース ID の接尾部。API リソースがリソース階層ツリーから構成される場合、リソース ID に接尾部を指定する必要があります。これは、使用する必要がある接尾部値です。

本文

なし。

例

この例では、 ID に API を含む API リソース内に定義されている getItemDetails() API を使用して、callAPI がアイテムに関する追加属性を取り出すために使用されています。API 入力またはテンプレートが JSP 内のどこにも指定されていないことに注意してください。これは、その他すべての API と同様に API リソース定義内に構成されます。

```
<yfc:loopXML binding="xml:/OrderLineStatusList/@OrderStatus" id="OrderStatus">
   <tr>
      <yfc:makeXMLInput name="orderLineKey">
       <yfc:makeXMLKey binding="xml:/OrderLineDetail/@OrderLineKey"
value="xml:/OrderStatus/OrderLine/@OrderLineKey"/>
       <yfc:callAPI apiID='AP1'/>
  <... >
   </tr>
```

callApi JSP タグが JSP 内で使用された後で、対応する出力名前空間で出力が利用可能になります。

callAPI (代替メソッド)

説明

callAPI JSP タグも、リソース階層ツリー内に API を定義することなく JSP 内で API を呼び出す方法をサポートします。この方法で呼び出される場合、タグに入力として異なる属性を渡すことが必要です。この代替メソッドは、API 呼び出しの入力またはテンプレートが JSP 内のいくつかの条件に基づいて動的になる必要がある場合に使用します。また、この代替メソッドは、API への入力が複雑で従来の技法を使用して形成できない場合に使用されます。

属性

apiName - オプション。呼び出される API の名前。この callAPI の代替メソッドを使用するときには、apiName または serviceName のいずれかが必要です。

suppressInputDecode - オプション。inputElement のすべての属性およびすべての子要素の再帰的 HTML デコードを抑制する場合には、この属性の値を TRUE に設定します。

serviceName - オプション。呼び出されるサービスの名前 (サービス定義フレームワークから)。サービスの呼び出しは、テンプレートの引き渡しをサポートしません。この callAPI の代替メソッドを使用するときには、serviceName または apiName のいずれかが必要です。

inputElement - この代替メソッドを使用するときには必須です。API に渡される入力要素を表す YFCElement。

templateElement - 条件付きで必須。API に予期されている出力テンプレートを表す YFCElement。この代替メソッドを使用するときに、apiName 属性が使用されている場合は、templateElement が必須です。serviceName が使用される場合は、templateElement は無視されます。

outputNamespace - オプション。 API の出力が配置される名前空間。

API の出力は、この名前空間に保存されます。名前空間はオプションですが、指定されない場合、考慮中の XML のルート・ノード名がデフォルト設定されます。したがって、API の出力を参照する際は、ここに名前空間が指定されない場合でも、名前空間は出力のルート・ノード名と同じであると仮定できます。

名前空間は、特定の XML の識別に使用できるタグです。プレゼンテーション・フレームワークを使用して複数の API を呼び出し、それらの出力を異なる名前空間に格納することができます。JSP 内または API への入力内で、その時点で使用可能な任意の名前空間の値を参照できます。

inputNamespace - オプション。入力名前空間は、API への追加入力を動的に解決するために使用されます。入力名前空間について詳しくは、「API へのデータの引渡し (Passing Data to APIs)」を参照してください。

本文

なし。

例

以下の例は、リソース階層ツリー内で API リソースを定義せずに JSP 内から getOrderDetails() API を呼び出す方法を示しています。callAPI タグが使用される前に、入力およびテンプレート要素が JSP 内で形成される方法に注意してください。callAPI タグ呼び出しの後で、JSP 内で後から使用できる RelatedFromOrderDetails 名前空間で getOrderDetails() API の出力が利用可能になります。

```
<%
  YFCDocument inputDoc = YFCDocument.parse("<Order
OrderHeaderKey=\"xml:/Document/@RelatedFromOrderHeaderKey\"/>");
YFCDocument templateDoc = YFCDocument.parse("<Order EnterpriseCode=\"\"
OrderHeaderKey=\"\"
OrderNo=\"\"
   Status=\"\" BuyerOrganizationCode=\"\" SellerOrganizationCode=\"\"
OrderDate=\"\" RulesetKey=\"\" HoldFlag=\"\" DocumentType=\"\"
isHistory=\"\"/>");
%>
<yfc:callAPI apiName='getOrderDetails'
inputElement='<%=inputDoc.getDocumentElement()%>'
templateElement='<%=templateDoc.getDocumentElement()%>'
outputNamespace='RelatedFromOrderDetails'/>
```

getXMLValue

説明

getXMLValue JSP タグは、XML バインディングに固有の XML 属性の値を返します。

属性

name - オプション。値の取得元となる XML の名前空間を格納しているストリング。このパラメーターを指定しない場合は、値はバインディングから取得されます。例えば、バインディングとして xml:/Menu/@MenuDescription を指定した場合は、name の値はデフォルトで Menu となります。または、もう 1 つの例として、バインディングとして xml:/mymenu:/Menu/@MenuDescription を指定した場合は、name の値はデフォルトで *mymenu* となります。

binding - 必須。目的の値の属性を指し示す XML パスを格納しているストリング。

本文

なし。

例

```
<td
class="protectedtext"><yfc:getXMLValue binding="xml:/Category/@CategoryID"
name="Category" /></td>
```

getXMLValueI18NDB

説明

getXMLValueI18NDB JSP タグは、ユーザーのロケールに基づいて XML バインディングに固有の XML 属性のローカライズ済み値を返します。

属性

name - オプション。値の取得元となる XML の名前空間を格納しているストリング。このパラメーターを指定しない場合は、値はバインディングから取得されます。例えば、バインディングとして `xml:/Menu/@MenuDescription` を指定した場合は、name の値はデフォルトで *Menu* となります。または、もう 1 つの例として、バインディングとして `xml:/mymenu:/Menu/@MenuDescription` を指定した場合は、name の値はデフォルトで *mymenu* となります。

binding - 必須。目的の値の属性を指し示す XML パスを格納しているストリング。

本文

なし。

例

```
<td
class="protectedtext"><yfc:getXMLValueI18NDB
binding="xml:/Category/@Description" name="Category" /></td>
```

hasXMLNode

説明

hasXMLNode JSP タグを使用して、特定の XML 要素または XML 属性が API によって返されるかどうかを確認します。

属性

binding - 必須。目的の要素または属性の XML パスを格納しているストリング。バインディング・ストリングに属性が含まれている場合に、その属性が空の場合は、該当する要素が存在する場合であっても、このタグの本体を処理することは許可されません。

本文

hasXMLNode の評価結果が true の場合にのみ書き込まれる HTML を格納できます。

例

この例では、キット (kit) が含まれたオーダー・リリースに属する明細についてキット・アイコンが表示される方法を示しています。

```
<td class="tablecolumn" nowrap="true">
    <yfc:hasXMLNode binding="xml:/OrderLine/KitLines/KitLine">
       <a <%=getDetailHrefOptions("L03", getParameter("orderLineKey"), "")%>>
           <img class="columnicon"
<%=getImageOptions(YFSUIBackendConsts.KIT_COMPONENTS_COLUMN, "
Kit_Components")%>>
        </a>
    </yfc:hasXMLNode>
</td>
```

この例では、親キット明細を持つ明細について親キット明細アイコンが表示される方法を示しています。

```
<yfc:hasXMLNode binding="xml:/OrderLine/@OrigOrderLineKey">
  <a <%=getDetailHrefOptions("L05", getParameter("origOrderLineKey"), "")%>>
      <img class="columnicon" <%=getImageOptions
(YFSUIBackendConsts.DERIVED_ORDERLINES_COLUMN, "Kit_Parent_Line")%>>
  </a>
</yfc:hasXMLNode>
```

i18n

説明

i18n JSP タグは、リソース・バンドルからローカライズされたキーの説明を取得します。このタグを HTML 内のすべてのリテラルに使用してください。

属性

なし。

本文

ローカライズされたストリングに変換される必要のあるキー。システムでロケール固有のリソース・バンドルが使用される方法について詳しくは、「Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales: *ローカライズ・ガイド*」を参照してください。

例

この例では、クエリー・タイプ (「次から始まる」や「次を含む」) が API の出力の select タグにどのように表示されるのかを示しています。

```
<tr>
   <td class="searchlabel" ><yfc:i18n>Product_Class</yfc:i18n></td>
</tr>
```

i18ndb

説明

i18ndb JSP タグは、ユーザーのロケールに基づいて、YFS_LOCALIZED_STRING テーブルからローカライズされた値の説明を取得します。このタグを使用して、HTML 内のローカライズされたデータベースの説明を取得してください。

属性

なし。

本文

ローカライズされたストリングに変換される必要のあるキー。システムでロケール固有のリソース・バンドルが使用される方法について詳しくは、「Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales: *ローカライズ・ガイド*」を参照してください。

例

この例では、クエリー・タイプ (「次から始まる」や「次を含む」) が API の出力の select タグにどのように表示されるのかを示しています。

```
<tr>
  <td>
      <yfc:i18ndb><%=resolveValue("xml:/Shipment/Status/@StatusName")%>
      </yfc:i18ndb>
   </td>
</tr>
```

loopOptions

説明

loopOptions JSP タグは、HTML の select タグに属しているオプションを構成します。

属性

binding - 必須。API 出力内の繰り返し要素を指し示すバインディング・ストリング。この繰り返し要素は、アットマーク (@) を使用して固定されているものである必要があります。

name - オプション。option タグ内のユーザーに表示される説明用に使用されるバインディング要素内の属性名。このパラメーターが渡されない場合は、デフォルト値である `name` が使用されて、その結果として Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales は name という属性を検索します。

value - オプション。option タグの value 属性用に使用されるバインディング要素内の属性名。このパラメーターが渡されない場合は、デフォルト値である `value` が使用されて、その結果として当アプリケーションは value という属性を検索します。

selected - オプション。デフォルトの選択済み値として評価および設定される必要があるバインディング・ストリング。これは、description 属性ではなく value 属性と対応付けられます。デフォルト値はスペース (" ") などのブランクです。

isLocalized - オプション。このパラメーターが "Y" という値として渡された場合は、ユーザーのロケールに基づいて表示されるローカライズ済みの説明が YFS_LOCALIZED_STRINGS テーブルから取得されます。

targetBinding - オプション。 select のターゲット・バインディングがソース・バインディングとは異なる場合は、loopOptions の使用時にターゲット・バインディングを入力として指定する必要があります。これにより、API で例外が発生した場合でも、エンド・ユーザーによって入力されたデータが失われることが防止されます。

本文

なし。

例

この例では、クエリー・タイプ (「次から始まる」や「次を含む」) が API の出力の select タグにどのように表示されるのかを示しています。

```
<td nowrap="true" class="searchcriteriacell" >
   <select name="xml:/Item/@ItemIDQryType" class="combobox" >
      <yfc:loopOptions
binding="xml:/QueryTypeList/StringQueryTypes/@QueryType"
name="QueryTypeDesc" value="QueryType" selected="xml:/Item/@ItemIDQryType"/>
   </select>
   <input type="text" class="unprotectedinput"
<%=getTextOptions("xml:/Item/@ItemID") %> />
</td>
```

この例では、編集可能なリスト内でコンボ・ボックスを使用しています。

- 下線文字 ("_") とカウンターを select 要素の name 属性の後ろに付加する必要があります。
- このカウンターの名前は loopXML タグで指定された ID 属性の値です。この ID 属性の値は常に、ループ場所である子ノード名と同じになるように設定される必要があります。

```
<select name="xml:/Order/Instructions/Instruction_
<%=InstructionCounter%>/@InstructionType" class="combobox">
   <yfc:loopOptions binding="xml:InstructionTypeList:/CommonCodeList/
@CommonCode" name="CodeShortDescription" value="CodeValue"
selected="xml:/Instruction/@InstructionType"/>
</select>
```

loopXML

説明

loopXML JSP タグは、ソース XML 内の特定の繰り返し要素をループ処理します。

注: お使いのアプリケーション・サーバーが JSP 仕様バージョン 1.1 までしかサポートしていない場合は、そのアプリケーション・サーバーでは、body タグが含まれたカスタム JSP タグ内で jsp:include を使用することはサポートされません。loopXML タグを使用すると、ランタイム JSP エラーが発生して、「カスタム・タグ内でのフラッシュは許可されていません (Illegal to flush within a custom tag)」というメッセージが表示されます。

このランタイム・エラーを回避するには、loopXML タグの代わりに getLoopingElementList() 関数を使用してください。『JSP Functions for the Console JSP Interface』を参照してください。

属性

binding - 必須。ソース XML 内でループ処理する対象となる要素のパス。この繰り返し要素は、アットマーク (@) を使用して固定されているものである必要があります。

name - オプション。ソース XML の名前。このパラメーターを指定しない場合は、値はバインディングから取得されます。例えば、バインディングとして `xml:/Menu/@MenuDescription` を指定した場合は、name の値はデフォルトで `Menu` となります。または、もう 1 つの例として、バインディングとして `xml:/mymenu:/Menu/@MenuDescription` を指定した場合は、name の値はデフォルトで `mymenu` となります。

id - オプション。バインディングから解決された要素を保持している作成された YFCElement の名前。このパラメーターが指定されない場合は、binding パラメーターが指し示す要素ノード名が使用されます。例えば、binding の値が `xml:/ItemList/@Item` であり、id パラメーターが渡されない場合は、id の値はデフォルトで `Item` となります。

本文

ループの反復ごとに書き込まれる HTML を格納できます。

例

この例では、loopXML JSP タグを使用してアイテム・ルックアップでアイテムのリストを表示する方法を示しています。

```
<tbody>
   <yfc:loopXML name="ItemList" binding="xml:/ItemList/@Item" id="item">
   <tr>
      <td class="tablecolumn">
          <img class="icon"
onclick="setItemLookupValue('<%=resolveValue("xml:item:/Item/@ItemID")%>',
'<%=resolveValue("xml:item:/Item/PrimaryInformation/@DefaultProductClass")
%>','<%=resolveValue("xml:item:/Item/@UnitOfMeasure")%>')"
value="<%=resolveValue("xml:item:/Item/@ItemID")%>"
<%=getImageOptions(YFSUIBackendConsts.GO_ICON,"Click_to_select")%> />
      </td>
      <td class="tablecolumn"><yfc:getXMLValue name="item"
binding="xml:/Item/@ItemID"/></td>
      <td class="tablecolumn"><yfc:getXMLValue name="item"
binding="xml:/Item/PrimaryInformation/@DefaultProductClass"/></td>
      <td class="tablecolumn"><yfc:getXMLValue name="item"
binding="xml:/Item/@UnitOfMeasure"/></td>
      <td class="tablecolumn"><yfc:getXMLValue name="item"
binding="xml:/Item/PrimaryInformation/@ShortDescription"/></td>
      <td class="tablecolumn"><yfc:getXMLValue name="item"
 binding="xml:/Item/PrimaryInformation/@MasterCatalogID"/></td>
      <td class="tablecolumn"><yfc:getXMLValue name="item"
binding="xml:/Item/@OrganizationCode"/></td>
   </tr>
   </yfc:loopXML>
</tbody>
```

makeXMLInput

説明

makeXMLInput JSP タグを makeXMLKey と組み合わせて使用して、リストから詳細画面にデータを渡すために使用される非表示キーを生成します。

属性

name - 必須。この JSP タグの結果として生成される非表示の入力 HTML タグの名前。

本文

複数の makeXMLKey JSP タグを格納できます。これらの makeXMLKey JSP タグの出力が連結されて、単一の非表示入力が生成されます。

例

この例では、この JSP タグを makeXMLKey JSP タグと組み合わせて使用して、「在庫」リスト・ビューから「在庫」詳細ビューに在庫キー・データを渡すための非表示入力を生成する方法を示しています。

<indexterm>makeXMLInput JSP タグ;JSP タグ・ライブラリー:makeXMLInput</indexterm>

```
<tbody>
  <yfc:loopXML name="InventoryList"
binding="xml:/InventoryList/@InventoryItem" id="InventoryItem"
keyName="InventoryItemKey" >
  <tr>
    <yfc:makeXMLInput name="inventoryItemKey">
      <yfc:makeXMLKey binding="xml:/InventoryItem/@ItemID"
value="xml:/InventoryItem/@ItemID" />
      <yfc:makeXMLKey binding="xml:/InventoryItem/@UnitOfMeasure"
 value="xml:/InventoryItem/@UnitOfMeasure" />
      <yfc:makeXMLKey binding="xml:/InventoryItem/@ProductClass"
value="xml:/InventoryItem/@ProductClass"  />
      <yfc:makeXMLKey binding="xml:/InventoryItem/@OrganizationCode"
 value="xml:InventoryList:/InventoryList/@OrganizationCode" />
    </yfc:makeXMLInput>
    <td class="checkboxcolumn">
       <input type="checkbox"
value='<%=getParameter("inventoryItemKey")%>' name="EntityKey"/>
    </td>
    <td class="tablecolumn">
      <a href="javascript:showDetailFor('<%=getParameter("inventoryItemKey")
%>');"><yfc:getXMLValue
name="InventoryItem" binding="xml:/InventoryItem/@ItemID"/></a>
    </td>
   <td class="tablecolumn"><yfc:getXMLValue
name="InventoryItem"  binding="xml:/InventoryItem/@ProductClass"/></td>
   <td class="tablecolumn"><yfc:getXMLValue name="InventoryItem"
 binding="xml:/InventoryItem/@UnitOfMeasure"/></td>
   <td class="tablecolumn"><yfc:getXMLValue name="InventoryItem"
 binding="xml:/InventoryItem/Item/PrimaryInformation/@Description"/></td>
  </tr>
  </yfc:loopXML>
</tbody>
```

makeXMLKey

説明

makeXMLKey JSP タグを makeXMLInput と組み合わせて使用して、リストから詳細画面にデータを渡すために使用される非表示キーを生成します。

属性

binding - 必須。解決されてから、詳細画面に渡される非表示入力内に保管される必要のあるバインディング・ストリング。

本文

なし。

例

『makeXMLInput』に記載されている例を参照してください。

第 19 章 JavaScript 関数

コンソール JSP インターフェースの JavaScript 関数について

Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales UI は、JavaScript 関数を使用して、ポップアップ・ウィンドウを開く、ビューの切り替え、ユーザー入力の検証などのクライアント・サイド操作を実行します。この UI で使用される JavaScripts は、UI インフラストラクチャー層によって提供されます。 UI 拡張を実行しながら同じ関数を使用することができます。このセクションでは、アプリケーションの UI 層によって提供される JavaScript 関数について説明します。

このアプリケーションは、UI インフラストラクチャーによって提供されない JavaScript 関数も使用することに注意してください。これらの関数は、通常、特定の画面に対する特定のアクションを実行し、UI 拡張時には使用する必要はありません、

また、UI インフラストラクチャーが JavaScript 関数を提供しない画面に追加ロジックが必要な場合は、必要に応じて独自に作成して使用できます。

ルックアップ

callLookup。[GET] を使用

invokeCalendar。[GET] を使用

yfcShowSearchPopup。[GET] を使用

コントロール名

ignoreChangeNames

yfcDoNotPromptForChanges

yfcDoNotPromptForChangesForActions

yfcHasControlChanged

yfcSetControlAsUnchanged

yfcSpecialChangeNames

イベント・ハンドラー

validateControlValues

yfcBodyOnLoad

yfcGetSaveSearchHandle

yfcGetSearchHandle

yfcValidateMandatoryNodes

詳細の表示

showDetailFor。[GET] を使用

showPopupDetailFor。[GET] を使用

yfcChangeDetailView。[POST] を使用

yfcShowDefaultDetailPopupForEntity。[GET] を使用

yfcShowDetailPopupWithDynamicKey

yfcShowDetailPopupWithKeys。[GET] を使用

yfcShowDetailPopupWithParams

リスト・ポップアップの表示

yfcShowListPopupWithParams。[GET] を使用

その他

doCheckAll

doCheckFirstLevel

expandCollapseDetails

getAttributeNameFromBinding

getCurrentSearchViewId

getCurrentViewId

getObjectByAttrName

goToURL

showHelp。[GET] を使用

yfcAllowSingleSelection

yfcDisplayOnlySelectedLines

setRetrievedRecordCount

callLookup

説明

この JavaScript 関数は、ユーザーが現在の画面で使用するためのレコードを検索および選択できるようにする検索画面を表示します。例えば、オーダー・エントリー画面の組織ルックアップは、ユーザーがバイヤー組織を選択できるようにします。一般的に、この関数を JSP ページ内のイメージの onclick イベントに付加します。

構文

callLookup(obj,entityname,extraParams)

入力パラメーター

entityname - オプション。検索画面内で検索するエンティティー。渡されない場合は、デフォルトで、現在のエンティティーの名前に設定されます。

obj - 必須。選択されるイメージのハンドル。

extraParams - オプション。検索画面に追加パラメーターを渡します。このパラメーターのフォーマットは、URL フォーマットの名前と値のペアです。渡されると、パラメーターは検索画面に渡されます。

出力パラメーター

なし。

例

この例は、検索のバイヤー役割フィールド内にセラー役割 (role) の表示をデフォルト設定する組織ルックアップを表示する方法を示します。

```
<img class="lookupicon"
onclick="callLookup(this,'organization',
'xml:/Organization/OrgRoleList/OrgRole/@RoleKey=BUYER')" name="search"
<%=getImageOptions(YFSUIBackendConsts.LOOKUP_ICON, "Search_for_Buyer") %> />
```

doCheckAll

説明

この JavaScript 関数は、以下の想定を使用して、テーブル内のすべてのチェック・ボックスの状態を切り替えます。

- テーブルには、分離したヘッドおよび本体セクションがある必要があります。
- 本体セクション内のチェック・ボックスには、指定したチェック・ボックス・オブジェクトと同じ列インデックスがある必要があります。複数のチェック・ボックスを含むセルは、すべて切り替えられます。

構文

doCheckAll(obj)

入力パラメーター

obj - オプション。テーブル・ヘッダー上のチェック・ボックス・オブジェクト (HTML オブジェクト階層内) へのハンドル。このオブジェクトが渡されない場合は、関数が返されます。

戻り値

なし。

例

この例は、オーダー番号およびエンタープライズを表示するオーダー・リスト・ビューがテーブル・ヘッダー行内の「すべて選択」および「すべてを選択解除」オプションを処理する方法を示します。

```
<table class="table" editable="false" width="100%" cellspacing="0">
 <thead>
  <tr>
   <td sortable="no" class="checkboxheader">
    <input type="checkbox" name="checkbox" value="checkbox"
onclick="doCheckAll(this);"/>
   </td>
    <td class="tablecolumnheader"><yfc:i18n>Order_#</yfc:i18n></td>
    <td class="tablecolumnheader"><yfc:i18n>Enterprise</yfc:i18n></td>
   </tr>
 </thead>
 <tbody>
   <yfc:loopXML binding="xml:/OrderList/@Order" id="Order">
    <tr>
      <yfc:makeXMLInput name="orderKey">
        <yfc:makeXMLKey binding="xml:/Order/@OrderHeaderKey"
value="xml:/Order/@OrderHeaderKey" />
      </yfc:makeXMLInput>
      <td class="checkboxcolumn">
        <input type="checkbox" value='<%=getParameter("orderKey")%>'
name="EntityKey"/>
      </td>
      <td class="tablecolumn">
        <a href="javascript:showDetailFor('<%=getParameter("orderKey")%>');">
           <yfc:getXMLValue binding="xml:/Order/@OrderNo"/>
         </a>
      </td>
      <td class="tablecolumn"><yfc:getXMLValue
binding="xml:/Order/@EnterpriseCode"/>
       </td>
    </tr>
   </yfc:loopXML>
 </tbody>
</table>
```

doCheckFirstLevel

説明

この JavaScript 関数は、テーブル列ヘッダー内のチェック・ボックスの onclick イベントで使用されます。この関数は、テーブル内のチェック・ボックスの**第 1** レベル内のすべてのチェック・ボックスを選択または選択解除します。この関数は、

doCheckAll JavaScript 関数と非常に似ていますが、doCheckAll は、指定した HTML テーブル内の**すべての**チェック・ボックスを選択または選択解除する点が異なります。

この関数は、この「すべてを選択 (解除)」機能を必要とし、選択チェック・ボックスに影響されないテーブル内に 1 つ以上の関連していないチェック・ボックスも持っている HTML テーブルに使用します。

構文

doCheckFirstLevel(obj)

入力パラメーター

obj - オプション。テーブル・ヘッダー上のチェック・ボックス・オブジェクト (HTML オブジェクト階層内) へのハンドル。このオブジェクトが渡されない場合は、関数は何もしません。

出力パラメーター

なし。

例

この例は、ユーザーがテーブル内で 1 つ以上の項目を選択できるチェック・ボックスを含む項目のリストに対するテーブル・ヘッダー定義を示します。テーブルのヘッダー行は、チェック・ボックスを含みます。このチェック・ボックスが選択されると、HTML テーブル内のすべての第 1 レベルのチェック・ボックスが選択または選択解除されます。

```
<table class="table" cellspacing="0" width="100%">
   <thead>
     <tr>
       <td class="checkboxheader" sortable="no" style="width:10px">
         <input type="checkbox" value="checkbox" name="checkbox"
onclick="doCheckFirstLevel(this);"/>
        </td>
         <td class="tablecolumnheader"
style="width:30px"><yfc:i18n>Options</yfc:i18n></td>
         <td class="tablecolumnheader"
style="width:<%=getUITableSize("xml:AdditionalServiceItem:/OrderLine/
AdditionalServiceItems/Item/@ItemID")%>"><yfc:i18n>Item_ID</yfc:i18n></td>
         <td class="tablecolumnheader"
style="width:<%=getUITableSize("xml:AdditionalServiceItem:/OrderLine/
AdditionalServiceItems/Item/@UnitOfMeasure")%>"><yfc:i18n>UOM</yfc:i18n></td>
         <td class="tablecolumnheader"
style="width:<%=getUITableSize("xml:AdditionalServiceItem:/OrderLine/
AdditionalServiceItems/Item/PrimaryInformation/@Description")%>">
<yfc:i18n>Item_Description</yfc:i18n></td>
         <td class="tablecolumnheader"
style="width:<%=getUITableSize("xml:AdditionalServiceItem:/OrderLine/
AdditionalServiceItems/Item/@Price")%>"><yfc:i18n>Price</yfc:i18n></td>
     </tr>
   </thead>
```

expandCollapseDetails

説明

この JavaScript 関数は、ビューを展開または省略する指定済みタグの表示状態を切り替えます。

構文

expandCollapseDetails(div_id, expandAlt, collapseAlt, expandgif, collapsegif)

入力パラメーター

div_id - 必須。展開または省略を行うオブジェクトの ID。

expandAlt - 必須。選択を展開するために表示するツール・ヒント。このツール・ヒントは、オブジェクトが省略状態にあるときに表示されます。

collapseAlt - 必須。選択を省略するために表示するツール・ヒント。オブジェクトが展開状態にあるときに使用可能です。

expandgif - 必須。選択が省略状態にあるときに表示されるイメージ。

collapsegif - 必須。選択が展開状態にあるときに表示されるイメージ。

戻り値

なし。

例

この例は、ユーザーが行レベルで特別なアイコンを選択することによって取得できるいくつかの拡張情報を非表示にするために expandCollapseDetails() 関数をテーブル内で使用する方法を示します。この例は、クレジット・カード番号などの支払集金の詳細をプラス (+) アイコンを選択することによって表示する方法を示します。また、この例では、情報を非表示にするかまたは表示するかどうかを指定できるようにする div も示します。デフォルトでは、div は非表示 (display:none) です。

```
<tbody>
 <yfc:loopXML
binding="xml:/Order/ChargeTransactionDetails/@ChargeTransactionDetail"
id="ChargeTransactionDetail">
   <%request.setAttribute("ChargeTransactionDetail",
YFCElement)pageContext.getAttribute("ChargeTransactionDetail"));%>
    <yfc:makeXMLInput name="InvoiceKey">
      <yfc:makeXMLKey binding="xml:/GetOrderInvoiceDetails/@InvoiceKey"
value="xml:/ChargeTransactionDetail/@OrderInvoiceKey" />
     </yfc:makeXMLInput>
     <tr>
        <td class="tablecolumn"
sortValue="<%=getDateValue("xml:ChargeTransactionDetail:/
ChargeTransactionDetail/@Createts")%>">
       <yfc:getXMLValue
binding="xml:/ChargeTransactionDetail/@Createts"/>
        </td>
        <td class="tablecolumn">
        <yfc:getXMLValue
binding="xml:/ChargeTransactionDetail/@ChargeType"/>
        <% if
```

```
(equals("AUTHORIZATION",getValue("ChargeTransactionDetail",
"xml:/ChargeTransactionDetail/@ChargeType")) ||
equals("CHARGE",getValue("ChargeTransactionDetail",
"xml:/ChargeTransactionDetail/@ChargeType"))) {%>
                <% String divToDisplay="yfsPaymentInfo_" +
ChargeTransactionDetailCounter; %>
                <img onclick="expandCollapseDetails('<%=divToDisplay%>',
'<%=getI18N("Click_To_See_Payment_Info")%>',
'<%=getI18N("Click_To_Hide_Payment_Info")%>',
'<%=YFSUIBackendConsts.FOLDER_COLLAPSE%>',
'<%=YFSUIBackendConsts.FOLDER_EXPAND%>')" style="cursor:hand"
<%=getImageOptions(YFSUIBackendConsts.FOLDER,"Click_To_See_Payment_Info")%>/>
                  <div id=<%=divToDisplay%>
style="display:none;padding-top:5px">
                     <table width="100%" class="view">
                        <tr>
                           <td height="100%">
                              <jsp:include page="/om/Orderdetail/
order_detail_paymenttype_collections.jsp">
                                 <jsp:param name="PrePathId"
value="ChargeTransactionDetail"/>
                                 <jsp:param name="ShowAdditionalParams"
value="Y"/>
                                 <jsp:param name="DecryptedCreditCardLink"
value="L02"/>

                              </jsp:include>
                           </td>
                        </tr>
                     </table>
                  </div>
                <%}%>
            </td>
             <td class="numerictablecolumn"
sortValue="<%=getNumericValue("xml:ChargeTransactionDetail:/
ChargeTransactionDetail/@CreditAmount")%>">
                <yfc:getXMLValue binding="xml:/ChargeTransactionDetail/
@CreditAmount"/>
            </td>
         </tr>
   </yfc:loopXML>
</tbody>
```

getAttributeNameFromBinding

説明

この JavaScript 関数は、入力として渡されるバインディング・ストリングを解析し、ストリングから属性を返します。

構文

getAttributeNameFromBinding(str)

入力パラメーター

str - オプション。バインディング・ストリングを含むストリング。渡されない場合、関数はヌルを返します。

戻り値

バインディング・ストリングの属性部分。

getCurrentSearchViewId

説明

この JavaScript 関数は、現在の検索ビューのリソース ID を取り出します。この関数は、検索ビューでのみ使用できます。現在の詳細ビューのリソース IDを 取得するには、詳細ビュー JSP ページで getCurrentViewId JavaScript 関数を使用します。

構文

getCurrentSearchViewId()

入力パラメーター

なし。

戻り値

現在の検索ビューのリソース ID。

例

この例では、現在のビュー ID を取得することによって、値がコンボ・ボックスから選択されたときに現在の検索ビューを最新表示する方法を示します。

```
<select class="combobox" onChange="changeSearchView(getCurrentSearchViewId())"
<%=getComboOptions(documentTypeBinding)%>>
   <yfc:loopOptions
binding="xml:CommonDocumentTypeList:/DocumentParamsList/@DocumentParams"
name="Description" value="DocumentType" selected="<%=selectedDocumentType%>"/>
</select>
```

getCurrentViewId

説明

この JavaScript 関数は、現在の詳細ビューのリソース ID を取り出します。

構文

getCurrentViewId()

入力パラメーター

なし。

戻り値

現在の詳細ビューのリソース ID。

例

この例では、現在のビュー ID を取得することによって、現在のビューを最新表示する方法を示します。

```
<td class="detaillabel" ><yfc:i18n>Horizon_End_Date</yfc:i18n></td>
<td class="protectedtext" nowrap="true">
  <input type="text" class="dateinput" onkeydown="return checkKeyPress(event)"
<%=getTextOptions("xml:/InventoryInformation/Item/@EndDate","xml:/
InventoryInformation/Item/@EndDate","")%> />
  <img class="lookupicon" onclick="invokeCalendar(this);return false"
<%=getImageOptions(YFSUIBackendConsts.DATE_LOOKUP_ICON,"View_Calendar")%>/>
  <input type="button" class="button" value="GO"
onclick="if(validateControlValues())changeDetailView(getCurrentViewId())"/>
</td>
```

getObjectByAttrName

説明

この JavaScript 関数は、指定された属性にバインドされているオブジェクトを返します。

バインディングを実現するためには、getTextOptions や getComboOptions などの JSP 関数が使用されます。JSP 関数のバインディングについて詳しくは、『yfsGetTextOptions』または『yfsGetComboOptions』を参照してください。

フィールドがバインドされると、そのフィールドの name 属性にはバインディング XML パスが格納されます。この JavaScript 関数は、指定された HTML タグ内のすべての入力ボックス、コンボ・ボックス、およびテキスト領域を検索して、name 属性の属性部分を照合します。最初の一致対象が返されます。

属性部分は、アットマーク (@) によって name 属性の残り部分と区切られています。例えば、名前が xml:/Order/@ChargeNameKey の場合は、属性部分は *ChargeNameKey* です。

構文

getObjectByAttrName(obj, attributeName)

入力パラメーター

obj - 必須。検索が実行される HTML オブジェクトのハンドル。

attributeName - オプション。指定されたオブジェクト下で検索が実行される対象となる属性名。このパラメーターが渡されない場合は、この関数はヌルを返します。

戻り値

指定された属性にバインドされているオブジェクトのハンドル。該当するオブジェクトが見つからない場合は、ヌルが返されます。

例

この例では、チェック・ボックスの選択状態に基づいて明細税の「料金名」フィールドを有効または無効にする方法を示しています。

```
function setAsPriceCharge(thisCheckbox) {
  var checkboxName=thisCheckbox.name
  var trNode=getParentObject(thisCheckbox, "TR");
  var sel=getObjectByAttrName(trNode, "ChargeNameKey");
```

```
 if (sel != null) {
   if (thisCheckbox.checked){
     sel.disabled=true;
     sel.value="";
    } else {
      sel.disabled=false;
    }
  }
}
```

getParentObject

説明

この JavaScript 関数は、渡されたオブジェクトより上位の HTML 階層内で最初に出現する指定タグを取得します。

構文

getParentObject(obj, tag)

入力パラメーター

obj - 必須。HTML オブジェクト階層内のオブジェクトのハンドル。

tag - オプション。検索で使用される上位ノードの名前を格納しているストリング。このパラメーターが渡されない場合は、この関数はヌルを返します。

戻り値

渡されたオブジェクトより上位の HTML 階層内で最初に出現する指定タグ。

例

この例では、ユーザーがテーブルの行で「削除」アイコンを選択したときに実行するクライアント側の削除処理をコーディングする方法を示しています。この例では、element とはユーザーが選択するオブジェクトを指します。

```
function deleteRow(element) {
 var row=getParentObject(element, "TR");
 oTable=getParentObject(row, "TABLE");
 row.parentNode.deleteRow(row.rowIndex - 1);
 fireRowsChanged(oTable);
 return false;
}
```

goToURL

説明

この JavaScript 関数は、指定された URL を新しいウィンドウで開きます。

構文

goToURL(URLInput)

入力パラメーター

URLInputObj - オプション。ユーザーによって指定された URL が含まれた input タグの名前。このパラメーターが渡されない場合は、この関数は何の処理も実行せずに完了します。

戻り値

なし。

例

この例では、goToUrl() によって「オーダー指示」画面が新しいウィンドウで開かれる方法を示しています。

```
<td>
   <input type="text"
<%=yfsGetTextOptions("xml:/Order/Instructions/Instruction_"
+ InstructionCounter + "/@InstructionURL", "xml:/Instruction/@InstructionURL",
"xml:/Order/AllowedModifications")%>/>
   <input type="button" class="button" value="GO"
onclick="javascript:goToURL('xml:/Order/Instructions/
Instruction_<%=InstructionCounter%>/@InstructionURL');"/>
</td>
```

ignoreChangeNames

説明

いずれかの詳細ビューがポストされるたびに、プレゼンテーション・フレームワークは、画面上のコントロールを通じて変更されたデータがないかチェックします。変更が加えられていないコントロールについては、name 属性は「old」+ [現在の名前] に変更されます。こうすることで、これらのコントロール内のデータは API に渡されなくなります。この結果として、未変更のデータを更新する必要がなくなるため、パフォーマンスが向上します。ただし、一部の API は置換モードで動作するように設計されています。これらの API は、情報 (未変更の部分を含む) の完全なスナップショットを生成して、データベース内でそのスナップショット全体を置換します。このような API の場合は、画面上のすべてのデータは入力として渡される必要があります。

このことを実現するために、この関数を onload イベントで呼び出すことができます。この関数は, ウィンドウ・オブジェクト内のカスタム・プロパティーを設定します。画面がポストされると、プレゼンテーション・フレームワークはこのカスタム・プロパティーの有無をチェックします。このプロパティーが設定されている場合は、自動的な名前変更は行われません。

プレゼンテーション・フレームワークは、ユーザーが入力したデータを忘れずに保存することを支援します。ユーザーが何らかのデータを変更した後、ページから離れようとすると、プレゼンテーション・フレームワークはその変更されたデータを検知して、ユーザーに作業内容を保存することを求めるプロンプトを表示します。この関数は、この機能の動作を変更することは一切ありません。この関数は, 変更が加えられていないコントロールの name プロパティーが確実に保持されるようにするだけです。この点で、この関数は yfcDoNotPromptForChanges() と異なります。

構文

ignoreChangeNames()

入力パラメーター

なし。

戻り値

なし。

例

この例では、この関数を onload イベントに関連付けています。

```
<script language="javascript">
window.attachEvent("onload", IgnoreChangeNames);
</script>
```

invokeCalendar

説明

この JavaScript 関数は、カレンダー・ルックアップを呼び出します。この関数では、渡されたオブジェクトの 1 つ前のオブジェクト (HTML の DOM 階層内) は、ルックアップで選択された日付が格納される必要のあるオブジェクトであると想定しています。

構文

invokeCalendar(obj)

入力パラメーター

obj - 必須。カレンダーを呼び出すために選択されたイメージ・オブジェクトのハンドル。

出力パラメーター

なし。

例

この例では、「在庫」詳細ビューの「期間の終了日」フィールドからカレンダー・ルックアップが呼び出される方法を示しています。

```
 <td class="detaillabel" ><yfc:i18n>Horizon_End_Date</yfc:i18n></td>
 <td class="protectedtext" nowrap="true">
       <input type="text" class="dateinput" onkeydown="return
checkKeyPress(event)"
<%=getTextOptions("xml:/InventoryInformation/Item/@EndDate","xml:
/InventoryInformation/Item/@EndDate","")%> />
    <img class="lookupicon"  onclick="invokeCalendar(this);return false"
<%=getImageOptions(YFSUIBackendConsts.DATE_LOOKUP_ICON,"View_Calendar")%>/>
    <input type="button" class="button" value="GO"
onclick="if(validateControlValues())changeDetailView(getCurrentViewId())"/>
 </td>
```

invokeTimeLookup

説明

この JavaScript 関数は、時間ルックアップを呼び出します。この関数では, 渡されたオブジェクトの 1 つ前のオブジェクト (HTML の DOM 階層内) は、ルックアップで選択された日付が格納される必要のあるオブジェクトであると想定しています。

構文

invokeTimeLookup(obj)

入力パラメーター

obj - カレンダーを呼び出すためにクリックされたイメージ・オブジェクトのハンドル。

出力パラメーター

なし。

例

この例では、日付と時刻の検索条件フィールドで時間ルックアップが使用される方法を示しています。

```
<tr>
  <td nowrap="true">
     <input class="dateinput" type="text"
<%=getTextOptions("xml:/Shipment/@FromExpectedShipmentDate_YFCDATE")%>/>
     <img class="lookupicon" name="search" onclick="invokeCalendar(this);
return false"
<%=getImageOptions(YFSUIBackendConsts.DATE_LOOKUP_ICON, "Calendar") %> />
     <input class="dateinput" type="text"
<%=getTextOptions("xml:/Shipment/@FromExpectedShipmentDate_YFCTIME")%>/>
     <img class="lookupicon" name="search"
onclick="invokeTimeLookup(this);return false"
<%=getImageOptions(YFSUIBackendConsts.TIME_LOOKUP_ICON,"Time_Lookup") %> />
     <yfc:i18n>To</yfc:i18n>
  </td>
</tr>
<tr>
  <td>
    <input class="dateinput" type="text"
<%=getTextOptions("xml:/Shipment/@ToExpectedShipmentDate_YFCDATE")%>/>
     <img class="lookupicon" name="search" onclick="invokeCalendar(this);
return false"
<%=getImageOptions(YFSUIBackendConsts.DATE_LOOKUP_ICON, "Calendar") %> />
     <input class="dateinput" type="text"
<%=getTextOptions("xml:/Shipment/@ToExpectedShipmentDate_YFCTIME")%>/>
     <img class="lookupicon" name="search"
onclick="invokeTimeLookup(this);return false"
       <%=getImageOptions(YFSUIBackendConsts.TIME_LOOKUP_ICON,
"Time_Lookup") %>/>
   </td>
</tr>
```

showDetailFor

説明

この JavaScript 関数は、現在のページを変更して、現在のエンティティーのデフォルト・ビューを表示します。その結果として表示される画面は、新しいウィンドウではなく同じブラウザー・ウィンドウに表示されます。通常は showDetailFor() 関数を使用して、リスト・ビューから詳細ビューに移動します。詳細ビューが開かれた後は、この関数は使用されません。その理由は、後続のビューは通常はポップアップ・ウィンドウとして開かれますが、この関数はそのような処理を実行しないからです。

この関数を詳細画面で使用する場合は、以下の動作に留意する必要があります。この関数は, [post] (ポスト) ではなく [get] (取得) を実行します。したがって、画面に「次へ」アイコンまたは「前へ」アイコンが表示されている場合に、この関数を使用して詳細ビューに切り替えた場合は、これらのアイコンは失われます。これらのアイコンが消える理由は、この関数が [get] を実行すると、「次へ」ビューや「前へ」ビューに関する情報が含まれた現行ページ内の非表示入力が失われるからです。

リスト画面では、この関数は yfc:makeXMLInput JSP タグと組み合わせて使用されます。makeXMLInput JSP タグは、キー属性が含まれた XML を作成します。この XML は、デフォルトの詳細ビューに渡される必要があります。

構文

showDetailFor(entityKey)

入力パラメーター

entityKey - 必須。詳細ビューで必要なキー属性が含まれた URL エンコード済み XML を格納しているストリング。

出力パラメーター

なし。

例

この例では、「オーダー番号」と「エンタープライズ・コード」という 2 つの列が含まれた「オーダー」リスト・ビューを表示します。「オーダー番号」は、「オーダー」のデフォルト詳細ビューを開くためのハイパーリンクになっています。yfc:makeXMLInput によって作成される XML は、その後で getParameter() JSP 関数を使用することで showDetailFor() 関数の入力パラメーターとして使用されることに注目してください。

```
<table class="table" editable="false" width="100%" cellspacing="0">
  <thead>
    <tr>
       <td sortable="no" class="checkboxheader">
         <input type="checkbox" name="checkbox" value="checkbox"
onclick="doCheckAll(this);"/>
        </td>
        <td class="tablecolumnheader"><yfc:i18n>Order_#</yfc:i18n></td>
        <td class="tablecolumnheader"><yfc:i18n>Enterprise</yfc:i18n></td>
```

```
    </tr>
  </thead>
  <tbody>
    <yfc:loopXML binding="xml:/OrderList/@Order" id="Order">
      <tr>
         <yfc:makeXMLInput name="orderKey">
               <yfc:makeXMLKey binding="xml:/Order/@OrderHeaderKey"
value="xml:/Order/@OrderHeaderKey" />
         </yfc:makeXMLInput>
         <td class="checkboxcolumn">
             <input type="checkbox" value='<%=getParameter("orderKey")%>'
name="EntityKey"/>
         </td>
         <td class="tablecolumn"><a
href="javascript:showDetailFor('<%=getParameter("orderKey")%>');">
             <yfc:getXMLValue binding="xml:/Order/@OrderNo"/></a>
          </td>
          <td class="tablecolumn"><yfc:getXMLValue
binding="xml:/Order/@EnterpriseCode"/></td>
      </tr>
    </yfc:loopXML>
  </tbody>
</table>
```

showDetailForViewGroupId

説明

この JavaScript 関数は、現在のページを変更して、指定されたビュー・グループ ID のデフォルト・ビューを表示します (最も小さいリソース・シーケンス番号が割り当てられたビュー ID が特定のビュー・グループ ID のデフォルト・ビューです)。その結果として表示される画面は、同じブラウザー・ウィンドウに表示されます。showDetailForViewGroupId() 関数を使用して、リスト・ビューから詳細ビューに移動します。詳細ビューが開かれるときには、この関数は使用されません。その理由は、後続のビューはポップアップ・ウィンドウとして開かれますが、この関数はそのような処理を実行しないからです。

リスト画面では、この関数は yfc:makeXMLInput JSP タグと組み合わせて使用されます。makeXMLInput JSP タグは、キー属性が含まれた XML を作成します。この XML は、デフォルトの詳細ビューに渡される必要があります。

構文

showDetailForViewGroupId (entityname, viewGroupId, entityKey, extraParameters)

入力パラメーター

entityName - 必須。詳細画面で検索場所となるエンティティー。

viewGroupId - 必須。ユーザーに表示されるビュー・グループ ID。

entityKey - 必須。詳細ビューで必要なキー属性が含まれた URL エンコード済み XML を格納しているストリング。

出力パラメーター

なし。

例

```
<td class="tablecolumn">
   <a href = "javascript:showDetailForViewGroupId
('load','YDMD200','<%=getParameter("loadKey")%>');">
<yfc:getXMLValue binding="xml:/Load/@LoadNo"/>
   </a>
</td>
```

showHelp

説明

この JavaScript 関数は、オンライン・ヘルプを新しいウィンドウで開きます。オンライン・ヘルプは多言語に対応させることができます。

構文

showHelp()

入力パラメーター

なし。

戻り値

なし。

例

次の例では、メニュー・バーでヘルプ・アイコンがクリックされたときにオンライン・ヘルプが開かれる方法を示しており、このオンライン・ヘルプでは最初に目次が表示されます。

```
<img alt="<%=getI18N("Help")%>" src="<%=YFSUIBackendConsts.YANTRA_HELP%>"
onclick='showHelp();'/>
```

注: 画面レベルのヘルプを使用できるのは、システム側で定義された検索ビュー、リスト・ビュー、および詳細ビューに対してのみです。カスタム・ビュー用の機能は、異なる内部 JavaScript 関数を通じて提供されます。この showHelp 関数が公開されている主な目的は、カスタマイズ可能であるメニュー・バーから使用されることです。メニューからは通常、全体的なシステム・ヘルプのみが必要とされ、画面レベルのコンテキスト・ヘルプは必要とされません。

showPopupDetailFor

説明

この JavaScript 関数は、現在のエンティティーのデフォルト・ビューをポップアップ・ウィンドウ (モーダル・ダイアログ) に表示します。これはブロック化呼び出しです。この関数は、モーダル・ダイアログが閉じられるまで復帰しません。

構文

showPopupDetailFor(key, name, width, height, argument)

入力パラメーター

key - 必須。詳細ビューで必要なエンティティー・キー。このパラメーターが渡されない場合は、現在のエンティティーのキーが自動的にポップアップ・ウィンドウに渡されます。

name - 必須。ブランク・スペース (" ") として渡します。これは使用されません。

width - 必須。ポップアップ・ウィンドウの水平方向のサイズ。単位はピクセルです。0 を指定した場合は、特定のデフォルトの幅が使用されます。

height - 必須。ポップアップ・ウィンドウの垂直方向のサイズ。単位はピクセルです。0 を指定した場合は、特定のデフォルトの高さが使用されます。

argument - 必須。このフィールドで渡された任意の値は、window.dialogArguments 属性を通じてモーダル・ダイアログで使用可能になります。

戻り値

なし。

例

この例では、「在庫監査」リスト画面から在庫監査の詳細が呼び出される方法を示しています。

同じリスト画面が、リスト・ビューと詳細ポップアップ・ウィンドウで使用されます。現在の画面がポップアップ・ウィンドウである場合に、トランザクション日を選択すると、別のポップアップ・ウィンドウが開いて監査の詳細が表示されます。現在のビューがリスト・ビューである場合は、「監査の詳細」画面は同じウィンドウに表示されます。

```
<tbody>
  <yfc:loopXML name="InventoryAudits"
binding="xml:/InventoryAudits/@InventoryAudit" id="InventoryAudit">
    <tr>
      <yfc:makeXMLInput name="inventoryAuditKey">
        <yfc:makeXMLKey binding="xml:/InventoryAudit/@InventoryAuditKey"
value="xml:/InventoryAudit/@InventoryAuditKey" />
        <yfc:makeXMLKey binding="xml:/InventoryAudit/@OrganizationCode"
value="xml:/InventoryAudit/@InventoryOrganizationCode" />
      </yfc:makeXMLInput>
      <td class="checkboxcolumn">
         <input type="checkbox" value='<%=getParameter("inventoryAuditKey")%>' name="EntityKey"/>
      </td>
      <td class="tablecolumn"
sortValue="<%=getDateValue("xml:/InventoryAudit/@Modifyts")%>">
        <%if ( "Y".equals(request.getParameter
(YFCUIBackendConsts.YFC_IN_POPUP)) ) {%>
            <a href=""
onClick="showPopupDetailFor('<%=getParameter("inventoryAuditKey")%>',
'','900','550',window.dialogArguments);return false;" >
              <yfc:getXMLValue name="InventoryAudit"
binding="xml:/InventoryAudit/@Modifyts"/>
            </a>
         <%} else {%>
            <a
href="javascript:showDetailFor('<%=getParameter("inventoryAuditKey")%>');">
              <yfc:getXMLValue name="InventoryAudit"
binding="xml:/InventoryAudit/@Modifyts"/>
            </a>
         <%}%>
       </td>
       <td class="tablecolumn">
           <yfc:getXMLValue name="InventoryAudit"
```

```
binding="xml:/InventoryAudit/@ItemID"/>

       </td>
       <td class="tablecolumn">
           <yfc:getXMLValue
name="InventoryAudit" binding="xml:/InventoryAudit/@ProductClass"/>
       </td>
       <td class="tablecolumn">
           <yfc:getXMLValue
name="InventoryAudit" binding="xml:/InventoryAudit/@UnitOfMeasure"/>
       </td>
       <td class="tablecolumn">
           <yfc:getXMLValue
name="InventoryAudit" binding="xml:/InventoryAudit/@TransactionType"/>
       </td>
        <td class="tablecolumn">
           <yfc:getXMLValue
name="InventoryAudit" binding="xml:/InventoryAudit/@ShipNode"/>
       </td>
    </tr>
   </yfc:loopXML>
</tbody>
```

validateControlValues

説明

この JavaScript 関数は、クライアント・サイドの検証エラーがないかどうかをチェックします。ユーザーが入力フィールドに無効なデータを入力すると、プレゼンテーション・フレームワークによって、エラーであることを示すフラグがそのフィールドに立てられます。ユーザーがそのページ内のデータを送信するときに、この関数が呼び出され、無効なデータがポストされないことが確認されます。

構文

validateControlValues()

入力パラメーター

なし。

戻り値

true - エラーは検出されませんでした。

false - 1 つ以上のエラーが検出されました。

例

次の例は、現行ページを送信する前にエラーがチェックされる方法を示しています。

```
<td class="detaillabel" ><yfc:i18n>Horizon_End_Date</yfc:i18n></td>
<td class="protectedtext" nowrap="true">
  <input type="text" class="dateinput" onkeydown="return checkKeyPress(event)"
<%=getTextOptions("xml:/InventoryInformation/Item/@EndDate",
"xml:/InventoryInformation/Item/@EndDate","")%> />
  <img class="lookupicon"  onclick="invokeCalendar(this);return false"
<%=getImageOptions(YFSUIBackendConsts.DATE_LOOKUP_ICON,"View_Calendar")%> />
  <input type="button" class="button" value="GO"
onclick="if(validateControlValues())changeDetailView(getCurrentViewId())"/>
</td>
```

yfcAllowSingleSelection

説明

操作によっては、一度に 1 つのレコードに対してのみ実行できるものがあります。ただし、ユーザー・インターフェースでは通常、操作を選択する前に複数のオプションの選択が許可されます。このため、複数の選択をサポートしない操作では、処理対象のレコードが複数選択されていないことを自身で検証する必要があります。この関数が、その検証を行います。

構文

yfcAllowSingleSelection(chkName)

入力パラメーター

chkName - オプション。チェック・ボックス・コントロール・セットの名前で、いずれかのチェック・ボックスが操作の実行前に選択されている必要があります。値が渡されない、または空白の場合、デフォルトの EntityKey に設定されます。

出力パラメーター

true - ゼロまたは 1 つのレコードが選択されました。

false - 複数のレコードが選択されました。

例

転送中の更新の受入は、一度に 1 カ所で実行できます。このため、転送中の更新の受入操作は、最初に JavaScript 関数 yfcAllowSingleSelection() を呼び出し、次に receiveIntransitUpdates() API を呼び出すように構成されます。

次の例では、sKeyName 変数で渡される値に設定されている名前を持つチェック・ボックスについて 1 つのみが選択された場合にのみ、アクションが実行されます。

```
function goToOrderLineSchedules(sSearchViewID, sKeyName, bPopup)
{
  if(yfcAllowSingleSelection(sKeyName))
  {
     ...
   }
 }
```

yfcBodyOnLoad

説明

この JavaScript 関数は、ページがロードされるときは常に呼び出されます。通常、これはページがロードされるときに、自動的に呼び出されます。ただし、onload イベントで何か特別な操作の実行がページで必要な場合、まずこの関数を呼び出してから、独自の window.onload() 関数を呼び出すことができます。

構文

yfcBodyOnLoad()

入力パラメーター

なし。

戻り値

なし。

例

次の例は、プレゼンテーション・フレームワークではなくカスタムの JSP によって、onload イベントを処理する方法を示しています。

```
function window.onload()     {
    if (!yfcBodyOnLoad()
&& (!document.all('YFCDetailError'))) {
     return;
    }
//Do your special processing here
}
```

yfcChangeDetailView

説明

この JavaScript 関数は、特定の詳細ビューに切り替えます。この関数では、ビューの切り替えに POST 関数が使用されます。

構文

yfcChangeDetailView(viewID)

入力パラメーター

viewID - 必須。切り替え先の詳細ビューのリソース ID。

戻り値

なし。

例

次の例は、オーダー料金と税のサマリーで、コンボ・ボックス値の変更時に現行ビューにページをリフレッシュするために yfcChangeDetailView() 関数を使用する方法を示しています。

```
<select name="chargeType" class="combobox"
onchange="yfcChangeDetailView(getCurrentViewId());">
  <option value="Overall" <%if (equals(chargeType,"Overall")) {%> selected
<%}%>><yfc:i18n>Ordered</yfc:i18n></option>
  <option value="Remaining" <%if (equals(chargeType,"Remaining")) {%> selected
<%}%>><yfc:i18n>Open</yfc:i18n></option>
  <option value="Invoiced" <%if (equals(chargeType,"Invoiced")) {%> selected
<%}%>><yfc:i18n>Invoiced</yfc:i18n></option>
</select>
```

yfcChangeListView

説明

この JavaScript 関数は、現行ビューをリスト・ビューに切り替えます。この関数は追加のフィルター条件は受け入れないため、リスト・ビューには、事前指定のフィルター条件が設定されていることが必要です。

構文

function yfcChangeListView(entity, searchViewId,maxrecords)

入力パラメーター

entity - 必須。 searchViewId が属するエンティティー。

searchViewId - 必須。切り替え先の検索ビューの ID。

maxrecords - オプション。リスト・ビューに表示できる最大レコード数。パフォーマンスを強化するには、このパラメーターを使用してください。これが渡されない場合は、yfs.properties ファイルに指定されている値がデフォルトとなります。

注: yfs.ui.MaxRecords プロパティーを設定するには、*INSTALL_DIR*/properties/customer_overrides.properties ファイルを使用します。

出力パラメーター

なし。

例

ホーム・ページに、表示の最大数として設定されている一定の数を上限とした警告のリストを表示します。すべての警告の完全なリストを表示するには、ユーザーは「その他の警告」操作を選択します。この操作は、yfcChangeListView() JavaScript 関数を呼び出すよう構成されています。

yfcDisplayOnlySelectedLines

説明

この JavaScript 関数は、ユーザーが画面 A のリストから複数のレコードを選択する必要があり、それらのレコードを画面 B に渡す必要があるシチュエーションで使用されます。画面 B では、選択されたレコードが表示され、各レコードの追加情報も表示される場合があります。このような場合、ロジックでは、画面 A の構築に使用したものと同じ API セットを呼び出して画面 B も構築し、クライアント・サイドでは、フィルター処理によって表示が画面 A で選択されたもののみに限定されるようにします。

この関数では、対象のテーブル内の各行に、URL エンコードされた XML (yfc:makeXMLInput JSP タグを使用して形成) に設定されている yfcSelectionKey という属性が必要です。

構文

yfcDisplayOnlySelectedLines(tableId)

入力パラメーター

tableId - 必須。コンテンツを前の画面で選択されたものに制限する必要のあるテーブルの ID 属性。

出力パラメーター

なし。

例

次の例は、オーダー明細の依存関係の作成画面が、オーダー明細リストの結果を、オーダーの詳細画面で選択された特定の明細に制限する方法を示しています。まず、この関数を onload イベントで呼び出す必要があります。

```
<script language="javascript">
   function window.onload() {
     if (!yfcBodyOnLoad() && (!document.all('YFCDetailError'))) {
          return;
     }
     yfcDisplayOnlySelectedLines("DependentLines");
   }
</script>
```

次に、各 <tr> タグに、yfcSelectionKey 属性を含める必要があります。

```
<tbody>
    <yfc:loopXML name="Order"
binding="xml:/Order/OrderLines/@OrderLine" id="OrderLine">
      <yfc:makeXMLInput name="orderLineKey">
      <yfc:makeXMLKey binding="xml:/OrderLineDetail/@OrderLineKey"
value="xml:/OrderLine/@OrderLineKey"/>
      <yfc:makeXMLKey binding="xml:/OrderLineDetail/@OrderHeaderKey"
value="xml:/Order/@OrderHeaderKey"/>
     </yfc:makeXMLInput>
     <tr yfcSelectionKey="<%=getParameter("orderLineKey")%>">
          <td class="tablecolumn"><yfc:getXMLValue
binding="xml:/OrderLine/Item/@ItemID"/></td>
          <td class="tablecolumn"><yfc:getXMLValue
binding="xml:/OrderLine/Item/@ProductClass"/></td>
          <td class="tablecolumn"><yfc:getXMLValue
binding="xml:/OrderLine/Item/@UnitOfMeasure"/></td>
          <td class="tablecolumn"><yfc:getXMLValue
binding="xml:/OrderLine/Item/@ItemDesc"/></td>
     </tr>
     </yfc:loopXML>
 </tbody>
```

yfcDoNotPromptForChanges

説明

この JavaScript 関数は、データに加えた変更の保存をユーザーに再認識させる自動プロンプトをオフにします。いずれの画面でもデフォルトで、ユーザーがデータを

入力して、そのデータを保存せずに別のところにナビゲートし始めると、プレゼンテーション・フレームワークがこれを検出して、ユーザーにデータを保存するよう警告を出します。

この関数を呼び出すと、ウィンドウ・オブジェクトにパラメーターが設定されます。このパラメーターは、onunload イベント時に確認され、このパラメーターがこの関数を通して設定されている場合は、ユーザーに警告は表示されません。

この関数を呼び出して、プロンプトの表示をオフにすると、保存時に画面内のすべてのデータが API に渡されます。

この JavaScript 関数は、内部パネル・アクションの実行時に、データに加えた変更の保存をユーザーに再認識させるプロンプトをオフにすることはありません。

内部パネルで API を呼び出し、変更の保存のプロンプトをユーザーに表示しない場合は、yfcSetControlAsUnchanged または yfcDoNotPromptForChangesForActions 関数も使用する必要があります。

構文

yfcDoNotPromptForChanges(value)

入力パラメーター

value - 必須。ユーザーが入力して、まだ保存していない新規データの保存を求めるプロンプトをユーザーに表示するかどうかを指定します。有効な値は true および false です。true と指定すると、ユーザーに保存を求めるプロンプトは表示されません。false と指定すると、ユーザーにデータの保存を求めるプロンプトが表示されます。

戻り値

なし。

例

次の例は、この関数が積荷目録 (manifest) の詳細画面の自動プロンプトをオフにする方法を示しています。

```
<script language="javascript">
yfcDoNotPromptForChanges(true);
</script>
```

yfcDoNotPromptForChangesForActions

説明

この JavaScript 関数は、ユーザーが内部パネル上のアクションをクリックしたときに実行される「現在の画面のデータに加えた変更は失われます」の検証をスキップするときに使用できます。通常、アプリケーション・コンソールの内部パネル・アクションは、画面上の編集可能フィールドとともには使用されません。このため、ユーザーが入力フィールドを変更して、アクションをクリックすると、デフォルトで警告メッセージが表示されます。この検証を回避するには、この JavaScript メソ

ッドを呼び出します。このメソッドをスクリプト・タグの JSP で呼び出すことによって、ビュー上のすべてのアクションに対してこのメソッドを呼び出すことができます。または、アクション・リソースの JavaScript プロパティーの一部としてこのメソッドを呼び出すことによって、特定のアクションに対してこのメソッドを呼び出すこともできます。

構文

yfcDoNotPromptForChangesForActions(value)

入力パラメーター

value - 必須。加えた変更の検証をスキップするには、`true` を渡します。検証をオンにするには、`false` を渡します。デフォルトで、検証はオンになっています。

戻り値

なし。

例

次の例は、JSP から yfcDoNotPromptForChangesForActions 関数を呼び出して、ビュー上のすべてのアクションに対して、加えた変更の検証をオフにする方法を示しています。

```
<script language="Javascript" >
      yfcDoNotPromptForChangesForActions(true);
</script>
```

yfcGetCurrentStyleSheet

説明

この JavaScript 関数は、現行ウィンドウのスタイル・シートの名前を取得します。

構文

yfcGetCurrentStyleSheet()

入力パラメーター

なし。

出力パラメーター

currentStyleSheet - 現行ウィンドウのスタイル・シートの名前。スタイル・シートのフルネームが、ファイル拡張子も含めて返されます (例えば、sapphire.css)。

例

次の例は、ウィンドウの現行スタイル・シートを取得する方法を示しています。

```
var currentStyleSheet = yfcGetCurrentStyleSheet();
```

yfcGetSaveSearchHandle

説明

この JavaScript 関数は、検索ビューの「検索の保存 (Save Search)」アイコンのハンドルを提供します。このハンドルをイベントの関連付けに使用して、カスタム動作を実現できます。「検索」アイコンに関連付けられている動作を変更する場合は、『yfcGetSearchHandle』を参照してください。

構文

var oObj=yfcGetSaveSearchHandle();

入力パラメーター

なし。

出力パラメーター

var - 検索ビューの「検索の保存 (Save Search)」アイコンのハンドル。

例

次の例は、ユーザーが「検索の保存 (Save Search)」アイコンを選択したときにアプリケーションでカスタム処理が実行されるようにする方法を示しています。

```
<script language="javascript">
 function attachBehaviorFn()
 {
       ...
  var oObj1=yfcGetSaveSearchHandle();
  var sVal1=oObj1.attachEvent("onclick",fixDerivedFromReturnSearch);
 }
   window.attachEvent("onload",attachBehaviourFn);
...
```

yfcGetSearchHandle

説明

この JavaScript 関数は、検索ビューの「検索」アイコンのハンドルを提供します。このハンドルをイベントの関連付けに使用して、カスタム動作を実現できます。「検索の保存 (Save Search)」アイコンに関連付けられている動作を変更する場合は、『yfcGetSaveSearchHandle』を参照してください。

構文

var oObj=yfcGetSearchHandle();

入力パラメーター

なし。

出力パラメーター

var - 検索ビューの「検索」アイコンのハンドル。

例

次の例は、ユーザーが「検索」アイコンを選択したときにアプリケーションでカスタム処理が実行されるようにする方法を示しています。

```
<script language="javascript">
 function attachBehaviourFn()
 {
  var
oObj=yfcGetSearchHandle();
  var sVal=oObj.attachEvent("onclick",fixDerivedFromReturnSearch);
       ...
  }
    window.attachEvent("onload",attachBehaviourFn);
...
```

yfcHasControlChanged

説明

この JavaScript 関数は、特定のコントロールのコンテンツが、ページがロードされてからユーザーによって変更されているかどうかを判別します。

これは、特定のコントロールの現在値を、ページがロードされたときにそのコントロールに格納されたカスタム属性の OldValue と比較することによって実行されます。

チェック・ボックスおよびラジオ・ボタンの場合、このカスタム属性は oldchecked です。

構文

yfcHasControlChanged(ctrl)

入力パラメーター

ctrl - 必須。HTML オブジェクト階層内のオブジェクト。

戻り値

true - 指定されたコントロールの値は、ページが最初にロードされたときから変更されています。

false - 指定されたコントロールの値は、ページが最初にロードされたときと同じです。

例

次の例は、「オーダー変更理由」ポップアップ・ウィンドウで、この関数を使用して隠しフィールドにオーバーライド・フラグを設定する方法を示しています。

隠しフィールドは、特別なオーバーライド権限があるユーザーによってのみ変更が許可される特定のフィールド (例えば、出荷指定日) がそのようなユーザーによって変更された場合にのみ、changeOrder() API に渡されます。この関数は、画面のいずれかの入力が変更されていないかどうかを検出します。

```
function setOverrideFlag(overrideFlagBinding) {
    var overrideFlagInput=document.all(overrideFlagBinding);
    var docInputs=document.getElementsByTagName("input");
    for (var i=0;i<docInputs.length;i++) {
       var docInput=docInputs.item(i);
       if (docInput.getAttribute("yfsoverride") == "true") {
           if (yfcHasControlChanged(docInput)) {
               overrideFlagInput.value="Y";
               return;
            }
       }
    }
    var docSelects=document.getElementsByTagName("select");
    for (var i=0;i<docSelects.length;i++) {
       var docSelect=docSelects.item(i);
       if (docSelect.getAttribute("yfsoverride") == "true") {
           if (yfcHasControlChanged(docSelect)) {
               overrideFlagInput.value="Y";
               return;
            }
       }
    }
}
```

yfcMultiSelectToSingleAPIOnAction

説明

この JavaScript 関数は、非推奨の yfcMultiSelectToSingleAPI() JavaScript 関数に置き換わるものです。

この関数は、リストでの複数の選択を使用して単一 API 呼び出しを行います。デフォルトでは、ユーザーによって複数の選択が行われているときに、リスト画面で API を呼び出すアクションが実行されると、その API は選択されたレコードごとに 1 回ずつ実行されます。この関数を使用すると、選択されたレコードすべてに対して 1 回のみ実行される API を呼び出すアクションを構成できます。

この関数をアクション・リソースに関連付けます。この関数は、ユーザーが選択するレコードごとに、リスト画面に非表示入力を作成します。アクションの入力名前空間が適切に定義されていれば、API が一度呼び出されて、選択されたすべてのレコードが渡されます。

構文

yfcMultiSelectToSingleAPIOnAction(checkBoxName, counterAttrName, valueAttrName, keyAttributeName, parentNodePrefix, parentNodePostfix)

入力パラメーター

checkBoxName - 必須。アクションが定義されている、JSP 内のチェック・ボックス・オブジェクトの名前。

counterAttrName - 必須。テーブル内の対象行を一意的に識別するカウンター値を含む、チェック・ボックス・オブジェクトの HTML 属性の名前。このパラメーターには常にストリング yfcMultiSelectCounter を使用することをお勧めします。

valueAttrName - 必須。keyAttributeName パラメーターで渡される属性に設定する必要のある値を含む、チェック・ボックス・オブジェクトの HTML 属性の名前。この値に、yfcMultiSelectValue ストリングの接尾辞を付けることをお勧めします。詳しくは、例を参照してください。

keyAttributeName - 必須。API に渡す必要のある属性の名前。

parentNodePrefix - 必須。API に対する入力として渡される、繰り返し XML エレメントまで (繰り返し XML エレメントを含む) の XML バインディングの部分。

parentNodePostfix - オプション。keyAttributeName パラメーターに指定された属性が渡されるときに格納される、API 入力の繰り返し XML エレメントから最終要素まで (最終要素を含む) の XML バインディングの部分。

出力パラメーター

なし。

例

次の例は、オーダー・リリース・リスト・ビューに定義されている出荷の作成アクションを示しています。これは、JSP 内でチェック・ボックス・オブジェクトが作成される方法を示しています。

```
<td class="checkboxcolumn"><input type="checkbox"
value='<%=getParameter("orderReleaseKey")%>' name="EntityKey"
yfcMultiSelectCounter='<%=OrderReleaseCounter%>'
yfcMultiSelectValue1='<%=getValue("OrderRelease",
"xml:/OrderRelease/@OrderReleaseKey")%>'
yfcMultiSelectValue2='Add'/>
</td>
```

さらに、出荷の作成アクションによって、JavaScript フィールドが以下のように設定されます。

```
yfcMultiSelectToSingleAPIOnAction('EntityKey', 'yfcMultiSelectCounter',
'yfcMultiSelectValue1', 'OrderReleaseKey',
'xml:/Shipment/OrderReleases/OrderRelease',
null);yfcMultiSelectToSingleAPIOnAction('EntityKey', 'yfcMultiSelectCounter',
'yfcMultiSelectValue2', 'AssociationAction',
'xml:/Shipment/OrderReleases/OrderRelease', null);
```

API に必要な 2 つの非表示入力を作成するために、yfcMultiSelectToSingleAPIOnAction 関数が 2 回呼び出されることに注意してください。

yfcSetControlAsUnchanged

説明

この JavaScript 関数は、コントロールが内部パネルに配置されたときに、ユーザーにデータの保存を求めるプロンプトが表示されないようにします。これは、コントロールを「変更なし」に設定することによって実現します。何らかの関数によって、プロンプト「現在の画面のデータに加えた変更は失われます」の表示が設定されます。ユーザーのコントロールに対する変更について詳しくは、『ignoreChangeNames』を参照してください。

ページ上のすべてのコントロールをこの関数を使用するように構成したら、アクションを呼び出す前に、ページ上のコントロールごとにこの関数を呼び出します。

内部パネルで Action が使用され、その Action に必要な入力を取得する変更可能なコントロールがその内部パネルに設定されている場合、この関数を使用して、「現在の画面のデータに加えた変更は失われます」というメッセージが表示されないように設定できます。

この関数を使用するときは、Action が含まれる JSP で yfcDoNotPromptForChanges() 関数も呼び出す必要があります。ユーザー・プロンプトについて詳しくは、『yfcDoNotPromptForChanges』を参照してください。

構文

yfcSetControlAsUnchanged (control)

入力パラメーター

control - 必須。HTML オブジェクト階層内のオブジェクト。

戻り値

なし。

例

次の例は、Action から CallSetControl() 関数を呼び出す方法を示しています。

```
<script language="javascript"> yfcDoNotPromptForChanges(true) </script>
<script language="javascript">
function CallSetControl() {
 var myControl=document.all("xml:/InventoryItem/SKU/@OldSKU");
 var myControl_1=document.all("xml:/InventoryItem/SKU/@NewSKU");
 var myControl_2=document.all("xml:/InventoryItem/@EMailID");
 yfcSetControlAsUnchanged(myControl);
 yfcSetControlAsUnchanged(myControl_1);
 yfcSetControlAsUnchanged(myControl_2);
 return(true);
 }
</script>
```

yfcShowDefaultDetailPopupForEntity

説明

この JavaScript 関数は、ポップアップ・ウィンドウ (形式指定ダイアログ) に、エンティティーのデフォルトの詳細ビューを表示します。ビューを表示するエンティティーを、名前が入力として渡されるチェック・ボックス・オブジェクトの yfsTargetEntity 属性に指定する必要があります。これは、ブロック呼び出しです。形式指定ダイアログがクローズされるまで値が返されることはありません。

構文

yfcShowDefaultDetailPopupForEntity(checkBoxName)

入力パラメーター

checkBoxName – 必須。デフォルトの詳細ビューを表示するエンティティーの ID が含まれた yfsTargetEntity 属性が設定されている 1 つ以上のチェック・ボックス・オブジェクトの名前。

戻り値

なし。

例

次の例は、オーダー・リスト画面の詳細の表示アクションでこの関数を使用して、オーダー・エンティティーのデフォルトの詳細ビューを表示する方法を示しています。

チェック・ボックスの JSP コードは、次のとおりです。

```
<td class="checkboxcolumn">
<input type="checkbox" value='<%=getParameter("orderKey")%>'
name="chkRelatedKey" yfsTargetEntity="order"/>
</td>
```

詳細の表示アクションは、次のプロパティーで定義する必要があります。

```
ID="<Some ID>"
Name="View_Details"
Javascript="showDefaultDetailPopupForEntity('chkRelatedKey')"
Selection Key Name="chkRelatedKey"
```

yfcShowDetailPopup

説明

この JavaScript 関数は、形式指定であるポップアップ・ウィンドウに特定のビュー ID を表示します。これはブロック呼び出しです。形式指定ダイアログ・ボックスがクローズされるまで値が返されることはありません。

構文

yfcShowDetailPopup(viewID, name, width, height, argument, entity, key)

入力パラメーター

viewID - 必須。ポップアップ・ウィンドウとして表示される詳細ビューのリソース ID。空ストリングとして渡された場合、ポップアップ・ウィンドウには、entity パラメーターに指定されたエンティティーのデフォルトの詳細ビューが表示されます。

name - 必須。空白スペース (" ") として渡します。未使用。

width - 必須。ポップアップ・ウィンドウの水平方向のサイズ。ピクセル単位で測定されます。0 として渡された場合、所定のデフォルトの幅が使用されます。

height - 必須。ポップアップ・ウィンドウの垂直方向のサイズ。ピクセル単位で測定されます。0 として渡された場合、所定のデフォルトの高さが使用されます。

argument - 必須。このフィールドで渡される値はすべて、window.dialogArguments 属性を通して形式指定ダイアログで使用可能です。

entity - オプション。開かれる詳細ビューのエンティティー。この値が渡されない場合、現在表示されているビューのものと同じエンティティーにデフォルト設定されます。

key - オプション。詳細ビューに必要なエンティティー・キー。この値が渡されない場合、現行エンティティーのキーがポップアップ・ウィンドウに渡されます。

戻り値

なし。

例

次の例は、「オーダーの詳細」画面で「保存」が選択されたときに「変更理由コード」ポップアップ・ウィンドウが表示される方法を示しています。

```
function enterActionModificationReason(modReasonViewID, modReasonCodeBinding,
modReasonTextBinding) {
    var myObject=new Object();
    myObject.currentWindow=window;
    myObject.reasonCodeInput=document.all(modReasonCodeBinding);
    myObject.reasonTextInput=document.all(modReasonTextBinding);
    // If the current screen has a hidden input for draft order flag
    // and the value of the input is "Y", don't show the modification
    // reason window.
    var draftOrderInput=document.all("hiddenDraftOrderFlag");
    if (draftOrderInput != null) {
        if ("Y" == draftOrderInput.value) {
           return (true);
        }
   }
    yfcShowDetailPopup(modReasonViewID, "", "550", "255", myObject);

    if (getOKClickedAttribute() == "true") {
       window.document.documentElement.setAttribute("OKClicked", "false");
       return (true);
    }
    else {
       window.document.documentElement.setAttribute("OKClicked", "false");
       return (false);
    }
}
```

yfcShowDetailPopupWithDynamicKey

説明

(HTML オブジェクト階層内の) 特定のオブジェクトとともに呼び出された場合、この JavaScript 関数は、そのオブジェクトの下に (再帰的に) すべての値を含んでいる URL エンコードされた XML を準備します。ポストされる値のみが考慮されます。次に XML がパラメーターとしてポップアップ・ウィンドウに渡され、指定されたビューが表示されます。

構文

yfcShowDetailPopupWithDynamicKey(obj, view, entity, inputNodeName, winObj)

入力パラメーター

obj - 必須。キーが動的に準備されるベースとなるオブジェクトのハンドル。特定のオブジェクトについて、この関数は、HTML 階層を全探索して、最も近い <table> タグを見つけます。その <table> タグから、この関数はすべての入力とチェック・ボックスを検索します。これらのコントロールのバインディングに基づいて、URL エンコードされた XML が形成され、エンティティー・キーとして、呼び出される詳細ビューに渡されます。

viewID - 必須。ポップアップ・ウィンドウとして表示される詳細ビューのリソース ID。空ストリングとして渡された場合、指定されたエンティティーのデフォルトの詳細ビューがポップアップ・ウィンドウに表示されます。

entity - オプション。表示される詳細ビューのエンティティーのリソース ID。これが渡されない場合は、現行エンティティーにデフォルト設定されます。

inputNodeName - 必須。詳細ビューに受け渡す準備をする XML のルート・ノード名。

winObj - オプション。このフィールドで渡される値はすべて、window.dialogArguments 属性を通して形式指定ダイアログで使用可能になります。このパラメーターが渡されない場合、空のオブジェクトがポップアップ・ウィンドウに渡されます。

戻り値

なし。

例

次の例は、この関数を使用して、返品に追加できるオーダー明細のリストを呼び出す方法を示しています。このオーダー明細のリストには、オーダー番号を指定する必要がありますが、この番号はユーザーによって編集可能です。このため、入力をサーバー・サイドで makeXMLInput JSP タグを使用して形成することができません。したがって、入力は、この関数を使用してクライアント・サイドで準備されます。次の例には、続行アイコンを doClick(this); として選択した場合に呼び出されるように構成する必要がある doClick() 関数が含まれています。このように、ボタン・オブジェクト自体がパラメーターとして doClick() 関数に渡されます。これが機能するように、オーダー番号入力ボックスを含むものと同じ <table> タグにボタン・オブジェクトを格納する必要があります。

```
function okClick(obj) {
  yfcShowDetailPopupWithDynamicKey(obj, 'YOMD2002', 'return',
'Order',new Object());
}
```

yfcShowDetailPopupWithKeys

説明

この JavaScript 関数は、ポップアップ・ウィンドウ (形式指定ダイアログ) に特定のビュー ID を表示します。これは、ブロック呼び出しです。形式指定ダイアログがクローズされるまで値が返されることはありません。

詳細ビューに渡すためにプレゼンテーション・フレームワークによって生成されたデフォルトのキーが、呼び出された詳細ビューで受け入れられないシチュエーションでこの関数を使用します。

構文

yfcShowDetailPopupWithKeys(viewID, name, width, height, argument, keyName, entity, selectionKeyName)

入力パラメーター

viewID - 必須。ポップアップ・ウィンドウとして表示される詳細ビューのリソース ID。空ストリングとして渡された場合、指定されたエンティティーのデフォルトの詳細ビューが表示されます。

name - 必須。空白スペース (" ") として渡します。未使用。

width - 必須。ポップアップ・ウィンドウの水平方向のサイズ。ピクセル単位で測定されます。0 として渡された場合、所定のデフォルトの幅が使用されます。

height - 必須。ポップアップ・ウィンドウの垂直方向のサイズ。ピクセル単位で測定されます。0 として渡された場合、所定のデフォルトの高さが使用されます。

argument - オプション。ポップアップ・ウィンドウの表示に使用される showModalDialog() 関数に引数パラメーターとして渡されます。この値は、window.dialogArguments 属性を通して形式指定ダイアログで使用可能になります。これが渡されない場合は、新規オブジェクトが作成されて、ポップアップ・ウィンドウに渡されます。

keyName - 必須。詳細ビューに必要なエンティティー・キーを含むコントロールの名前属性。これが渡されない場合は、デフォルト値 EntityKey に設定されます。

entity - オプション。表示される詳細ビューのリソース ID。これが渡されない場合は、現行エンティティーにデフォルト設定されます。

selectionKeyName - オプション。ポップアップ・ウィンドウを呼び出す前に、ユーザーによって選択される必要のあるチェック・ボックス・コントロールの名前。この名前が渡されない (または NULL として渡された) 場合、チェック・ボックス・コントロールの選択は実行されず、ポップアップ・ウィンドウがすぐに呼び出されます。

戻り値

なし。

例

次の例は、独自のエンティティー・キーを指定した住所の変更ダイアログを内部パネルから呼び出す方法を示しています。

```
function doModifyAddressDialogWithKeys(source, viewID, entityKeyName) {
   var myObject=new Object();
   myObject.currentwindow=window;
   myObject.currentsource=source;
```

```
   if(viewID == null) {
      viewID="YADD001";
   }
    if (entityKeyName == null) {
        entityKeyName="EntityKey";
    }
    yfcShowDetailPopupWithKeys(viewID, "", "600", "425", myObject,
entityKeyName);
}
```

yfcShowDetailPopupWithKeysAndParams

説明

この JavaScript 関数は、指定した詳細ビューを形式指定ダイアログ内に呼び出します。name1=value1&name2=value2 の形式でストリングを形成し、このストリングをこの関数にパラメーターとして渡すことによって、詳細ビューにパラメーターを渡すことができます。

この関数は、ビューの呼び出しに使用される URL に、渡されたストリングを追加します。このようにして、渡されたパラメーターは、呼び出されたビューの要求オブジェクトで使用可能になります。

詳細ビューに渡すためにプレゼンテーション・フレームワークによって生成されたデフォルトのキーが、呼び出された詳細ビューで受け入れられないシチュエーションでこの関数を使用します。

構文

yfcShowDetailPopupWithKeysAndParams(viewID, name, width, height, argument,keyName, entity, selectionKeyName, params)

入力パラメーター

viewID - 必須。ポップアップ・ウィンドウとして表示される詳細ビューのリソース ID。空ストリングとして渡された場合、指定されたエンティティーのデフォルトの詳細ビューが表示されます。

name - 必須。空白スペース (" ") として渡します。未使用。

width - 必須。ポップアップ・ウィンドウの水平方向のサイズ。ピクセル単位で測定されます。0 として渡された場合、所定のデフォルトの幅が使用されます。

height - 必須。ポップアップ・ウィンドウの垂直方向のサイズ。ピクセル単位で測定されます。0 として渡された場合、所定のデフォルトの高さが使用されます。

argument - オプション。ポップアップ・ウィンドウの表示に使用される showModalDialog() 関数に引数パラメーターとして渡されます。この値は、window.dialogArguments 属性を通して形式指定ダイアログで使用可能になります。これが渡されない場合は、新規オブジェクトが作成されて、ポップアップ・ウィンドウに渡されます。

keyName - 必須。詳細ビューに必要なエンティティー・キーを含むコントロールの名前属性。これが渡されない場合は、デフォルト値 EntityKey に設定されます。

entity - オプション。表示される詳細ビューのリソース ID。これが渡されない場合は、現行エンティティーにデフォルト設定されます。

selectionKeyName - オプション。ポップアップ・ウィンドウを呼び出す前に、ユーザーによって選択される必要のあるチェック・ボックス・コントロールの名前。この名前が渡されない (または NULL として渡された) 場合、チェック・ボックス・コントロールの選択は実行されず、ポップアップ・ウィンドウがすぐに呼び出されます。

params - 必須。呼び出される詳細ビューに渡されるパラメーターを含んだストリング。構文 `name1=value1&name2=value2` を使用してください。詳細ビューを呼び出す URL にこのストリングが追加され、要求オブジェクトの詳細ビューでパラメーターが使用できるようになります。

戻り値

なし。

例

次の例では、カスタム詳細ページを開き、いくつかのカスタム・パラメーターをその詳細ページに渡します。

```
yfcShowDetailPopupWithKeysAndParams('CSTOrder012','',800,600,new
Object(),'EntityKey','order','EntityKey',
'CustParam1=xml:/Order&CustParam2=process')
```

yfcShowDetailPopupWithParams

説明

この JavaScript 関数は、指定した詳細ビューを形式指定ダイアログ内に呼び出します。`name1=value1&name2=value2` の形式でストリングを形成し、このストリングをこの関数にパラメーターとして渡すことによって、詳細ビューにパラメーターを渡すことができます。

この関数は、ビューの呼び出しに使用される URL に、渡されたストリングを追加します。このようにして、渡されたパラメーターは、呼び出されたビューの要求オブジェクトで使用可能になります。

構文

yfcShowDetailPopupWithParams(viewID,name,width,height,params,entity,key, argument)

入力パラメーター

viewID - 必須。ポップアップ・ウィンドウとして表示される詳細ビューのリソース ID。空ストリングとして渡された場合、指定されたエンティティーのデフォルトの詳細ビューが表示されます。

name - 必須。未使用。ただし、空ストリングを渡す必要があります。

width - 必須。ポップアップ・ウィンドウの水平方向のサイズ。ピクセル単位で測定されます。0 として渡された場合、所定のデフォルトの幅が使用されます。

height - 必須。ポップアップ・ウィンドウの垂直方向のサイズ。ピクセル単位で測定されます。0 として渡された場合、所定のデフォルトの幅が使用されます。

params - 必須。呼び出される詳細ビューに渡されるパラメーターを含んだストリング。構文 `name1=value1&name2=value2` を使用してください。詳細ビューを呼び出す URL にこのストリングが追加され、要求オブジェクトの詳細ビューでパラメーターが使用できるようになります。

entity - オプション。詳細ビューに対応するリソース ID。これが渡されない場合は、現行エンティティーにデフォルト設定されます。

key - オプション。詳細ビューにパラメーターとして渡されるキーの値。これが渡されない場合は、現行ビューのキーが、呼び出される詳細ビューに渡されます。

argument - オプション。ポップアップ・ウィンドウの表示に使用される showModalDialog() 関数に引数パラメーターとして渡されます。この値は、window.dialogArguments 属性を通して形式指定ダイアログで使用可能になります。これが渡されない場合、空のオブジェクトが形式指定ダイアログに渡されます。

戻り値

なし。

例

次の例は、この関数を使用してメモ・ポップアップ・ウィンドウが表示される方法を示しています。メモ・ポップアップ・ウィンドウ詳細ビューには、いくつかのパラメーターを渡す必要があります。例えば、現行オーダー状況についてメモが編集可能かどうかを制御する属性をポイントする XML バインディングが必要です。これを実現するために、次の例では、これらのパラメーターを含むストリングを形成して、この JavaScript 関数を呼び出します。

```
var
extraParams="allowedBinding=xml:/Order/AllowedModification&getBinding=xml:
/Order&saveBinding=xml:/Order";
yfcShowDetailPopupWithParams('YOMD020', '', "800", "600", extraParams);
```

yfcShowListPopupWithParams

説明

この JavaScript 関数は、ポップアップ・ウィンドウ (形式指定ダイアログ) に、指定されたリスト・ビューを表示します。これは、ブロック呼び出しです。ウィンドウがクローズされるまでこの関数から値が返されることはありません。

構文

yfcShowListPopupWithParams(viewID, name, width, height, argument, entity, params)

入力パラメーター

viewID - 必須。ポップアップ・ウィンドウとして表示されるリスト・ビューのリソース ID。空ストリングとして渡された場合、指定されたエンティティーのデフォルトのリスト・ビューが表示されます。

name - 必須。空白スペース (" ") として渡します。未使用。

width - 必須。ポップアップ・ウィンドウの水平方向のサイズ。ピクセル単位で測定されます。0 として渡された場合、所定のデフォルトの幅が使用されます。

height - 必須。ポップアップ・ウィンドウの垂直方向のサイズ。ピクセル単位で測定されます。0 として渡された場合、所定のデフォルトの高さが使用されます。

argument - 必須。ポップアップ・ウィンドウの表示に使用される showModalDialog() 関数に引数パラメーターとして渡される値。この値は、window.dialogArguments 属性を通して形式指定ダイアログで使用可能になります。

entity - オプション。表示される詳細ビューのリソース ID。これが渡されない場合は、現行エンティティーにデフォルト設定されます。

params - オプション。先頭がアンパーサンド (&) で、検索実行のベースとなる追加パラメーターを含むストリング。渡されたパラメーターは、要求パラメーターとして、呼び出されるリスト・ビューで使用可能になります。

戻り値

なし。

例

次の例は、指定された組織、アイテム、単位、および製品クラスについて、「在庫サマリー」画面から在庫監査リストが直接呼び出される方法を示しています。

```
function showInvAuditSearch(sViewID,sItemID,sUOM,sProductClass,sOrgCode)
{
      var ItemID=document.all(sItemID).value;
      var UOM=document.all(sUOM).value;
      var PC=document.all(sProductClass).value;
      var Org=document.all(sOrgCode).value;
      var entity="inventoryaudit";
      var sAddnParams="&xml:/InventoryAudit/@ItemID="+ItemID+"&xml:
/InventoryAudit/@UnitOfMeasure="+UOM;
      sAddnParams=sAddnParams + "&xml:/InventoryAudit/@ProductClass="+PC+"&xml:
/InventoryAudit/@OrganizationCode="+Org;

       yfcShowListPopupWithParams(sViewID,"",'900', '500','',entity,
sAddnParams);
}
```

yfcShowSearchPopup

説明

この関数は、ポップアップ・ウィンドウに、指定された検索ビューを呼び出します。この関数を検索結果の表示に使用できます。

構文

yfcShowSearchPopup(viewID, name, width, height, argument, entity)

入力パラメーター

viewID - 必須。ポップアップ・ウィンドウとして表示される検索ビューのリソース ID。空ストリングとして渡された場合、指定されたエンティティーのデフォルトの詳細ビューが表示されます。

name - 必須。空白スペース (" ") として渡します。未使用。

width - 必須。ポップアップ・ウィンドウの水平方向のサイズ。ピクセル単位で測定されます。0 として渡された場合、所定のデフォルトの幅が使用されます。

height - 必須。ポップアップ・ウィンドウの垂直方向のサイズ。ピクセル単位で測定されます。0 として渡された場合、所定のデフォルトの高さが使用されます。

argument - 必須。ポップアップ・ウィンドウの表示に使用される showModalDialog() 関数に引数パラメーターとして渡される値。この値は、window.dialogArguments 属性を通して形式指定ダイアログで使用可能になります。

entity - オプション。検索対象のエンティティーに対応するリソース ID。これが渡されない場合は、現行エンティティーの名前にデフォルト設定されます。

戻り値

なし。

例

次の例は、単一フィールド検索を呼び出す方法を示しています。callLookup() 関数によって、検索ポップアップ・ウィンドウが呼び出されます。

検索ポップアップ・ウィンドウから、ユーザーが行を選択したときに、選択された値がパラメーターとして指定されて setLookupValue() 関数が呼び出されます。

setLookupValue() 関数によって、テキスト・フィールドに値が取り込まれ、検索ウィンドウがクローズされます。

```
function setLookupValue(sVal)
{
   var Obj=window.dialogArguments
   if(Obj != null)
      Obj.field1.value=sVal;
   window.close();
}
//obj is to be passed as "this",
// which would be the icon that was selected for lookup.
//This function assumes that the lookup icon is placed
// immediately after the text field on which lookup is requested.
//entityname is the entity name of the search view
// that needs to be shown in the lookup.
function callLookup(obj,entityname)
{
  var oObj=new Object();
  var oField=obj.previousSibling;
  while(oField != null && oField.type != "text" && oField.type != "TEXT")
  {
   oField=oField.previousSibling;
```

```
  }
  oObj.field1=oField;
  yfcShowSearchPopup('','lookup',900,550,oObj,entityname);
}
```

yfcSpecialChangeNames

説明

この JavaScript 関数は、キーが渡されるときに行全体が渡されることが API で必要な場合に、呼び出す必要があります。

構文

yfcSpecialChangeNames(id, checkOnlyBlankRow)

入力パラメーター

id - 必須。その下で名前変更を実行する必要がある HTML タグの ID。

checkOnlyBlankRow - オプション。これが `true` として渡された場合は、新規空白行 (すべての入力と選択が空になっている) のみが、名前変更の対象になります。これが `false` として渡された場合は、ID が渡されたオブジェクトの下のすべての既存行が対象となります。これが渡されない場合、値はデフォルトの `false` に設定されます。

戻り値

なし。

yfcSplitLine

説明

この JavaScript 関数は、特定の行を 2 つの行に分割します。

構文

yfcSplitLine(imageNode)

入力パラメーター

imageNode - 必須。この関数が呼び出されたときの応答として選択されるイメージへのオブジェクト・ポインター。

出力パラメーター

なし。

次の表は、新規作成された行の動作を指定するときに、各セル・レベルで使用する属性を示しています。

属性	動作
ShouldCopy	子セルを含めて、セルのコンテンツをコピーするかどうかを指定します。true と指定すると、コンテンツがコピーされます。false と指定すると、コンテンツはコピーされず、空のセルが作成されます。デフォルトは false です。
NewName	新規セルで、自動生成される名前を獲得するかどうかを指定します。生成される名前は、コピー対象の現行オブジェクトの名前と行番号から派生されます。true と指定すると、新規セルで新規名が獲得されます。false と指定すると、名前は生成されません。デフォルトは false です。名前生成ロジックでは、元の名前に「*_integer*」を含める必要があり、この場所に行番号を挿入する必要があります。例えば、元の名前が `xml:/InspectOrder/ReceiptLines/FromReceiptLine_1/@ReceiptLineNo` である場合、新規行には、`xml:/InspectOrder/ReceiptLines/FromReceiptLine_2/@ReceiptLineNo` という名前のオブジェクトが格納されます。
NewClassName	オブジェクトの新規コピーのクラス。例えば、元のオブジェクトのクラスは `unprotectedinput` だが、新規コピーを保護したい場合は、新規クラスを protectedinput と指定します。この属性を使用しない場合は、コピーで元のオブジェクトのクラスが使用されます。
NewDisabled	コントロールを使用不可の状態で作成するかどうかを指定します。一部の HTML コントロール (<img> タグなど) については、これによってコントロール上のすべてのアクションを使用不可にすることになります。true と指定すると、`disabled` という名前のプロパティーが true に設定されます。false と指定すると、disabled プロパティーは設定されません。デフォルトは false です。
NewResetValue	新規オブジェクトの value 属性の状態を指定します。これを true に設定すると、新規オブジェクトの value 属性は無効になります。この結果、ほとんどの HTML コントロールについて、コントロールのコンテンツが無効になります。false と指定すると、元のコントロールの値がコピーで設定されます。デフォルトは false です。
NewContentEditable	オプション。新規オブジェクトで、現行オブジェクトの ContentEditable プロパティーを継承するかどうかを指定します。これを指定すると、現行オブジェクトの ContentEditable プロパティーも新規オブジェクトの ContentEditable プロパティーになります。一部の HTML コントロール (テキスト・ボックスなど) については、このプロパティーは、コントロールが編集可能かどうかを制御します。指定する値が、コピーの ContentEditable 属性に設定される値になります。これを指定しないと、新規オブジェクトは現行オブジェクトの ContentEditable プロパティーを継承します。
NewTRClassName	オプション。明細分割後に形成される新規行のクラスを指定します。この属性が渡されないと、デフォルトの動作となります。例えば、<tr> に `classname="oddrow"` がクラスとして指定されており、明細分割後に新規行で同じクラスを保持したい場合、JSP には、`<tr classname="oddrow">` ではなく、`<tr classname="oddrow" NewTRClassName = "oddrow">.` を含める必要があります。

例

次の例は、返品検査の際に、クライアント・サイドで明細を分割して、特定の受入明細を複数の在庫処置で使用できるようにする方法を示しています。

```
<yfc:loopXML binding="xml:/ReceiptLines/@ReceiptLine" id="ReceiptLine">
<tr>
   <yfc:makeXMLInput name="receiptLineKey">
       <yfc:makeXMLKey binding="xml:/ReceiptLine/@ReceiptLineKey"
value="xml:/ReceiptLine/@ReceiptLineKey"/>
   </yfc:makeXMLInput>

   <td class="checkboxcolumn" ShouldCopy="false" nowrap="true">
       <input type="checkbox" value='<%=getParameter("receiptLineKey")%>'
name="chkEntityKey"/>
   </td>
   <td class="checkboxcolumn" nowrap="true" ShouldCopy="false" >
       <img class="columnicon" <%=getImageOptions
(YFSUIBackendConsts.RECEIPT_LINE_HISTORY,"Disposition_History")%>></a>
    </td>
  <td class="tablecolumn" nowrap="true" ShouldCopy="false">
<yfc:getXMLValue binding="xml:/ReceiptLine/@SerialNo"/></td>
  <td class="tablecolumn" nowrap="true" ShouldCopy="false">
<yfc:getXMLValue binding="xml:/ReceiptLine/@LotNumber"/></td>
  <td class="tablecolumn" nowrap="true" ShouldCopy="false">
<yfc:getXMLValue binding="xml:/ReceiptLine/@ShipByDate"/></td>
  <td class="numerictablecolumn" nowrap="true" ShouldCopy="false">
<yfc:getXMLValue binding="xml:/ReceiptLine/@AvailableForTranQuantity"/></td>
  <td class="tablecolumn" nowrap="true" ShouldCopy="false">
<yfc:getXMLValue binding="xml:/ReceiptLine/@DispositionCode"/></td></td>
  <td class="tablecolumn" ShouldCopy="false" >
      <yfc:getXMLValue binding="xml:/ReceiptLine/@InspectionComments"/>
  </td>
    <td class="tablecolumn" nowrap="true" ShouldCopy="true" >
       <img IconName="addSplitLine" src="../console/icons/add.jpg"
          <% if
(getNumericValue("xml:/ReceiptLine/@AvailableForTranQuantity") > 1) { %>
             class="lookupicon" onclick="yfcSplitLine(this)"
          <%} else {%>
style="filter:progid:DXImageTransform.Microsoft.BasicImage(grayScale=1)"
          <%}%>
          />
          <input type="hidden"
<%=getTextOptions("xml:/InspectOrder/ReceiptLines/FromReceiptLine_" +
ReceiptLineCounter + "/@ReceiptHeaderKey",
"xml:/ReceiptLine/@ReceiptHeaderKey")%>/>
          <input type="hidden"
<%=getTextOptions("xml:/InspectOrder/ReceiptLines/FromReceiptLine_" +
ReceiptLineCounter + "/@ReceiptLineNo",
"xml:/ReceiptLine/@ReceiptLineNo")%>/>
          <select NewName="true" NewClassName="unprotectedinput"
NewContentEditable="true" NewResetValue="true" class="combobox"
<%=getComboOptions("xml:/InspectOrder/ReceiptLines/FromReceiptLine_" +
ReceiptLineCounter + "/ToReceiptLines/ToReceiptLine_1/@DispositionCode")%>>
        <yfc:loopOptions binding="xml:/ReturnDispositionList/
@ReturnDisposition" name="Description" value="DispositionCode"
selected="xml:/ReceiptLine/@DispositionCode"/>
          </select>
    </td>
    <td class="tablecolumn" nowrap="true" ShouldCopy="true" >
      <input type="text" NewName="true" ewClassName="numericunprotectedinput"
NewContentEditable="true" NewResetValue="true"
class="numericunprotectedinput"
<%=getTextOptions("xml:/InspectOrder/ReceiptLines/FromReceiptLine_" +
ReceiptLineCounter + "/ToReceiptLines/ToReceiptLine_1/@Quantity",  "")%>/>
    </td>
    <td class="tablecolumn" nowrap="true" ShouldCopy="true" >
```

```
      <input type="text" NewName="true" NewClassName="unprotectedinput"
NewContentEditable="true" NewResetValue="true" class="unprotectedinput"
<%=getTextOptions("xml:/InspectOrder/ReceiptLines/FromReceiptLine_" +
ReceiptLineCounter + "/ToReceiptLines/ToReceiptLine_1/@InspectionComments",
"")%>/>
    </td>
  </tr>
 </yfc:loopXML>
</tbody>
```

yfcValidateMandatoryNodes

説明

この JavaScript 関数は、画面の必須セクションを検証します。この関数は、ページ内のすべての TABLE エレメントを解析し、テーブルごとに yfcMandatoryMessage 属性があるかどうか検索します。この属性が見つかると、この関数は、そのテーブルのコンテンツを確認します。コンテンツが変更されていない場合、この関数は、対応する <table> タグの yfcMandatoryMessage 属性内に設定されているメッセージを発行します。

構文

yfcValidateMandatoryNodes()

入力パラメーター

なし。

出力パラメーター

なし。

例

次の例は、返品受入の場合に保存操作を行う前に必須の検証が実行される方法を示しています。まず、yfcValidateMandatoryNodes() 関数を呼び出すように保存操作が構成されます。

次に、内部パネルの JSP で、検証を必要とするテーブルに以下の属性が設定されます。

```
<table class="table" ID="ReceiveLines" width="100%" editable="true"
 yfcMandatoryMessage="<yfc:i18n>Receipt_information_must_be_entered</yfc:i18n>">
```

yfcFindErrorsOnPage

説明

この JavaScript 関数は、JSP 内で使用できます。この関数は、ページ上のエラーの検出に使用されます。エラーを検出すると、この関数は該当する警告メッセージを出します。

構文

yfcFindErrorsOnPage()

入力パラメーター

なし。

戻り値

なし。

例

次の例は、JSP から yfcFindErrorsOnPage() 関数を呼び出す方法を示しています。JSP 内に動的行を追加する一方で、この関数はその JSP にページ上のエラーがないかどうかを確認します。エラーが検出されると、該当する警告メッセージが表示されます。

```
function addRows(element)
 {
    if(yfcFindErrorsOnPage())
       return;
}
```

setRetrievedRecordCount

説明

この JavaScript 関数は、リスト・ビュー用に作成された JSP 内で使用できます。すべてのリスト・ビュー画面のタイトルの横には通常、取得されたレコードの数を示すメッセージが表示されます。例えば、リスト・ビューに 2 件のレコードが表示されている場合は、「2 件のレコードを取得 (Retrieved 2 record(s)」というメッセージが表示されます。この JavaScript を使用すると、このメッセージに表示されるレコード数を動的に設定できます。通常は、UI インフラストラクチャーは、このリスト・ビューのエンティティー・リソース下で定義された "リスト" API の出力に基づいて、正しいメッセージを自動的に表示します。

ただし場合によっては、"リスト" API を特定のリスト・ビューに対して使用できないことがあります。これらのケースでは、そのリスト・ビューはデフォルトのリスト API を無視するように設定されており、代わりにそのリスト・ビューは独自の API を呼び出します。独自の API を呼び出すために、callAPI taglib を使用して独自の JSP 内で、またはリスト・ビュー・リソース下で、異なる API を定義しています。この場合は、UI インフラストラクチャーは正しいメッセージを自動的に表示できません。

構文

setRetrievedRecordCount (recordCount)

入力パラメーター

recordCount - 必須。「X 件のメッセージを取得 (Retrieved X record(s))」というメッセージに表示する正しいレコード数。

戻り値

なし。

例

この例では、リスト・ビュー用に定義された JSP から setRetrievedRecordCount() 関数を呼び出す方法を示しています。正しい数が JSP コードとして計算されます。次に、この結果が script タグ内で呼び出される setRetrievedRecordCount メソッドに渡されます。

```
<%
   YFCElement root = (YFCElement)request.getAttribute("OrganizationList");
   int countElem = countChildElements(root);
%>
<script language="javascript">
    setRetrievedRecordCount(<%=countElem%>);
</script>
```

第 20 章 データ型参照

コンソール JSP インターフェースのデータ型参照

DataType ノードは、UIType ノードおよび XMLType ノードを含みます。プレゼンテーション・フレームワークでは、UIType に指定されている属性は XMLType に指定されている属性をオーバーライドし、XMLType に指定されている属性は DataType に指定されている属性をオーバーライドします。

以下の表は、どのノードが特定のデータ型属性によってサポートされるかをリストしています。

属性	説明	サポートされるノード
Name	抽象データ型の固有 ID。	DataType 型
型	値 NUMBER、VARCHAR2、DATE、DATETIME、QUANTITY を指定できます。 型が QUANTITY として選択された場合、yfs.properties ファイル内の yfs.install.displaydoublequantity プロパティー内に指定されているデフォルト値に基づいて、10 進数を使用するかまたは使用しない場合があります。 **注:** このプロパティーを変更するには、*INSTALL_DIR*/properties/customer_overrides.properties ファイル内にエントリーを追加します。プロパティーおよび customer_overrides.properties ファイルの変更について詳しくは、「Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales: *プロパティー・ガイド (Properties Guide)*」を参照してください。	DataType XMLType UIType
サイズ (Size)	DataType ノード内に指定された場合、入力ボックス内に入力可能な文字の最大数として扱われます。 UIType ノード内に指定された場合、5 で乗算され、その結果は入力ボックスの長さとして使用されるピクセル数として扱われます。	DataType XMLType UIType

属性	説明	サポートされるノード
PpcSize	DataType ノード内に指定された場合、RF UI 画面内の入力ボックス内に入力可能な文字の最大数として扱われます。 UIType ノード内に指定された場合、5 で乗算され、その結果は RF UI 画面内の入力ボックスの長さとして使用されるピクセル数として扱われます。 PpcSize 属性が指定されていないインスタンスでは、Size 属性が考慮されます。	DataType XMLType UIType
ZeroAllowed	数値フィールドでのみ使用されます。	DataType XMLType UIType
UITableSize	ここで指定される値は、5 で乗算され、その結果はピクセル単位の幅として使用されます。この特定の属性は、特別な要求によってのみ利用可能です。 getUITableSize() JSP 関数を使用して、 この値を eve します (eve this value.)。	UIType
NegativeAllowed	数値フィールドでのみ使用されます。	DataType XMLType UIType

特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものです。

本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒103-8510

東京都中央区日本橋箱崎町19番21号

日本アイ・ビー・エム株式会社

法務・知的財産

知的財産権ライセンス渉外

以下の保証は、国または地域の法律に沿わない場合は、適用されません。IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。 IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

IBM Corporation

J46A/G4

555 Bailey Avenue

San Jose, CA 95141-1003

U.S.A.

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、IBM より提供されます。

この文書に含まれるいかなるパフォーマンス・データも、管理環境下で決定されたものです。そのため、他の操作環境で得られた結果は、異なる可能性があります。一部の測定が、開発レベルのシステムで行われた可能性がありますが、その測定値が、一般に利用可能なシステムのものと同じである保証はありません。さらに、一部の測定値が、推定値である可能性があります。実際の結果は、異なる可能性があります。お客様は、お客様の特定の環境に適したデータを確かめる必要があります。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。IBM は、それらの製品のテストは行っておりません。したがって、 他社製品に関する実行性、互換性、またはその他の要求については確証できません。IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

IBM の将来の方向または意向に関する記述については、予告なしに変更または撤回される場合があり、単に目標を示しているものです。

表示されている IBM の価格は IBM が小売り価格として提示しているもので、現行価格であり、通知なしに変更されるものです。卸価格は、異なる場合があります。

本書はプランニング目的としてのみ記述されています。記述内容は製品が使用可能になる前に変更になる場合があります。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、名称や住所が類似する企業が実在しているとしても、それは偶然にすぎません。

著作権使用許諾:

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・ プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。これらのサンプル・プログラムは特定物として現存するままの状態で提供されるものであり、いかなる保証も提供されません。IBM は、お客様の当該サンプル・プログラムの使用から生ずるいかなる損害に対しても一切の責任を負いません。

それぞれの複製物、サンプル・プログラムのいかなる部分、またはすべての派生的創作物にも、次のように、著作権表示を入れていただく必要があります。

© IBM 2012. このコードの一部は、IBM Corp. のサンプル・プログラムの派生物です。© Copyright IBM Corp. 2012.

この情報をソフトコピーでご覧になっている場合は、写真やカラーの図表は表示されない場合があります。

商標

IBM、IBM ロゴおよび ibm.com® は、世界の多くの国で登録された International Business Machines Corp. の商標です。他の製品名およびサービス名等は、それぞれ IBM または各社の商標である場合があります。現時点での IBM の商標リストについては、http://www.ibm.com/legal/copytrade.shtml をご覧ください。

Adobe、Adobe ロゴ、PostScript、および PostScript ロゴは、Adobe Systems Incorporated の米国およびその他の国における登録商標または商標です。

IT Infrastructure Library は、英国 Office of Government Commerce の一部である the Central Computer and Telecommunications Agency の登録商標です。

Intel、Intel (ロゴ)、Intel Inside、Intel Inside (ロゴ)、 Intel Centrino、Intel Centrino (ロゴ)、Celeron、Intel Xeon、Intel SpeedStep、Itanium、Pentium は、Intel Corporation または子会社の米国およびその他の国における商標または登録商標です。

Linux は、Linus Torvalds の米国およびその他の国における商標です。

Microsoft、Windows、Windows NT および Windows ロゴは、Microsoft Corporation の米国およびその他の国における商標です。

ITIL は英国 Office of Government Commerce の登録商標および共同体登録商標であって、米国特許商標庁にて登録されています。

UNIX は、The Open Group の米国およびその他の国における登録商標です。

Java およびすべての Java 関連の商標およびロゴは Oracle やその関連会社の米国およびその他の国における商標または登録商標です。

Cell Broadband Engine, Cell/B.E は、米国およびその他の国における Sony Computer Entertainment, Inc. の商標であり、同社の許諾を受けて使用しています。

Linear Tape-Open、LTO、LTO ロゴ、Ultrium および Ultrium ロゴは、米国およびその他の国における HP、IBM Corp. および Quantum の商標です。

Connect Control Center®、Connect:Direct®、Connect:Enterprise™、Gentran®、Gentran®:Basic®、Gentran:Control®、Gentran:Director®、Gentran:Plus®、Gentran:Realtime®、Gentran:Server®、Gentran:Viewpoint®、Sterling Commerce™、Sterling Information Broker®、および Sterling Integrator® は、Sterling Commerce™, Inc.、IBM Company の商標です。

プログラム番号: xxxx-xxx

Printed in Japan

日本アイ・ビー・エム株式会社
〒103-8510 東京都中央区日本橋箱崎町19-21