Sterling Selling and Fulfillment Foundation

カスタマイズ 基本

バージョン 9.1

Sterling Selling and Fulfillment Foundation

カスタマイズ 基本

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、 57 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。

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
Customization Basics
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 2012.2

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

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 拡張性の概要

アプリケーションの拡張

この章では、Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales で実行可能な拡張のタイプについて説明します。概要、大まかな概念、およびアプリケーションのテクニカル・アーキテクチャー図を提供します。

- 外観の拡張

  アプリケーションではプレゼンテーション・フレームワーク・ツールキットが提供され、これによって情報が機能する方法は変更せずに、そのレンダリングまたは表示方法を変更することができます。

- トランザクションの拡張

  Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales ではサービス定義フレームワークが提供されます。これは、 Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales とサード・パーティー製アプリケーション間のデータの変換と転送を自動化し、そのデータを各システムで読み取り可能な形式に変換するインフラストラクチャーです。サービス定義フレームワークでは、ロギングと例外も処理されます。これにより、アプリケーションの機能を拡張するカスタム・トランザクションを作成できます。

- データベースの拡張

  ユーザー・インターフェースとトランザクションのカスタマイズに加えて、ビジネスに固有の追加の属性を格納するようデータベースを拡張できます。

- 印刷ドキュメントの拡張

  印刷ドキュメントをカスタマイズできます。例えば、バーコードのデフォルトの長さを拡張できます。

  アプリケーションで印刷されたレポートを生成するには、JasperReports を使用できます。また、アプリケーションでレポートを PDF、RTF、またはその他の Jasper 印刷オブジェクトとして生成することもできます。JasperReports は、オープン・ソース Java レポート作成ツールで、別個にダウンロードおよびインストールできます。JasperReports のインストールのためのガイドラインは、*INSTALL_DIR*/xapidocs/code_examples/jasperreports/alert_report_readme.html ファイルで提供されています。sampleAlertReport.pdf という名前のサンプルの JasperReport も、同じディレクトリーから使用できます。

  Jasper 印刷コンポーネントはサービス定義フレームワークで定義され、イベントに基づいてドキュメントを自動的に印刷するために使用できます。これは XML を入力として受け取り、出力 XML として提供する、標準の XML ベースのコンポーネントです。このコンポーネントおよび印刷トランザクションについて詳し

くは、「Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales: *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。

- ブラウザー・ポータルの拡張

  ユーザーがよく使用する検索条件をリストするアプリケーション・データベースにビューをエクスポートできます。

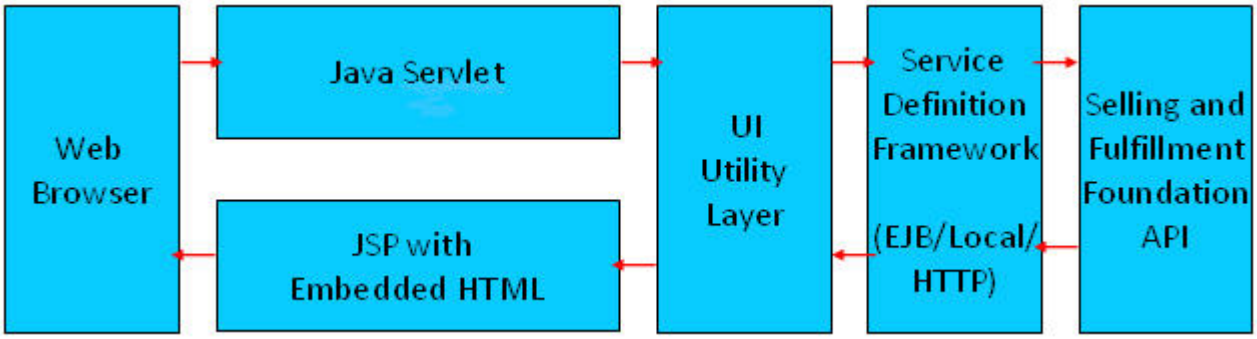
コンソール・ユーザー・インターフェースの拡張

アプリケーション・コンソールは、オーダーや在庫など、日々のトランザクション・ビジネスを実行および追跡するためのユーザー・インターフェースです。

アプリケーション・コンソール UI は、Java Server Pages (JSP) 内で HTML を使用します。ユーザー・インターフェース層は、サービス定義フレームワークで定義されたサービスを介して公開された API にアクセスします。これにより、公開された API のみが確実に使用されるようにできます。

サービス定義フレームワークの UI 層で使用される XML 操作は、最小限に抑えられます。XML 出力の大幅な処理が必要となる個所では、 API に対する変更により、より UI フレンドリーな出力が提供されます。

以下の図は、アプリケーション・コンソール・ユーザー・インターフェースのテクニカル・アーキテクチャーを示しています。



アプリケーション・コンソール・ユーザー・インターフェースの拡張について詳しくは、「Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales: *エンド・ユーザー向けコンソール JSP インターフェース・カスタマイズ・ガイド*」を参照してください。

Applications Manager ユーザー・インターフェースの拡張

Applications Manager は、組織のビジネス・ルールとトランザクションのセットアップを構成するためのユーザー・インターフェースです。それは、Java Swing ページで構成されます。

ユーザー・インターフェースの拡張について詳しくは、「Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales: Swing インターフェース・カスタマイズ・ガイド」を参照してください。

モバイル・ユーザー・インターフェースの拡張

Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales を使用すると、倉庫の操作で使用するモバイル・デバイスに、カスタム・ユーザー・インターフェースを開発および表示できます。

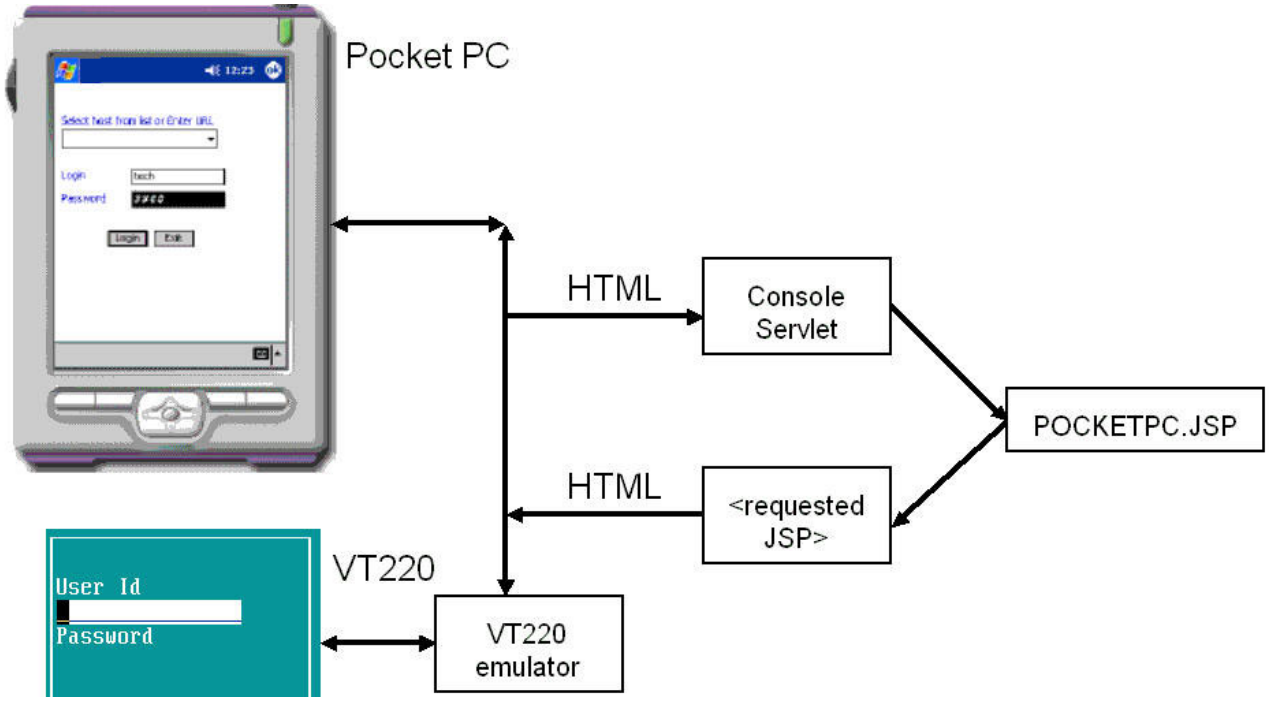
モバイル・アーキテクチャーは、クライアントとサーバーの 2 つのコンポーネントで構成されます。

- モバイル・クライアント — 通常 Pocket PC ベースのハンドヘルド・デバイスですが、 VT220 ターミナル・エミュレーションを指す場合もあります。本書では、モバイル・クライアントはモバイル・デバイスとも呼ばれます。
- アプリケーション・サーバー — アプリケーション・サーバーで稼働中のSterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales。

クライアントとサーバーは、HTTP プロトコルを使用して通信し、要求と応答のモデルに従って HTML を転送します。各クライアント要求のタイプは "/console/*.ppc" です。

アプリケーション・サーバーは any.ppc に対する要求を制御サーブレット (コンソール・サーブレットと呼ばれる) にダイレクトし、サーブレットは要求を pocketpc.jsp ファイルにリダイレクトします。pocketpc.jsp ファイルは、 uientity で指定されているように要求を JSP にリダイレクトします。JSP は HTML 応答をレンダリングし、それはモバイル・クライアントによって表示されます。

以下の図は、このアーキテクチャーを示しています。



モバイル・ユーザー・インターフェースの拡張について詳しくは、「Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales : *モバイル・デバイス向けユーザー・インターフェース・カスタマイズ・ガイド*」を参照してください。

データベースの拡張

データベース拡張により、追加のデータを取り込むための列やテーブルを追加できます。

詳しくは、「Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales: *データベース拡張ガイド*」を参照してください。

トランザクションの拡張

Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Salesは、データの変換と転送時にエラーを処理および解決するためのメカニズムを提供します。このメカニズムは、サービス定義フレームワークと呼ばれ、以下のトランザクション・プロセスにアクセスすることを可能にします。

- アプリケーションにより公開されるシステム API
- アプリケーションにより発行されるデータを、転送サービス層にルーティングするイベント・ハンドラー
- 必要に応じてタスクをモニターおよび実行する、タイム・トリガー・トランザクション

サービス定義フレームワークは、log4j ユーティリティーを介したエラー・チェックを提供します。このユーティリティーは、トレースおよびデバッグ情報の両方をログ・ファイルに書き込みます。

さらに、以下のタイプのカスタム・コードを作成することで、アプリケーションを拡張できます。

- 拡張 (カスタム) API
- デフォルトのビジネス・アルゴリズムを上書きする外部プログラム
- API およびタイム・トリガー・トランザクションによって使用されるビジネス・アルゴリズムを拡張する外部プログラム
- カスタムのタイム・トリガー・トランザクション

トランザクションの拡張について詳しくは、「Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales : *トランザクション拡張ガイド*」を参照してください。

リッチ・クライアント・プラットフォーム・ユーザー・インターフェースの拡張

リッチ・クライアント・プラットフォームは、リモートにデプロイおよび更新可能で管理が容易な、高度に対話的なリッチ・クライアントを提供します。リッチ・クライアントは、その接続先のサーバーに依存せずに大量のデータ操作を処理するクライアントです。ただし、主にデータの保管のためにはサーバーに依存します。リッチ・クライアントは、機能性が高く、オペレーティング・システムのプログラミング機能に完全にアクセスできます。

詳しくは、「Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales : *リッチ・クライアント・プラットフォーム・インターフェース・カスタマイズ・ガイド*」を参照してください。

第 3 章 開発環境のセットアップ

アプリケーション拡張の前提条件

このガイドでは、以下を前提としています。

- アプリケーションが既にインストール済みであること。
- デプロイメント・モードでのエンタープライズ・アーカイブ (EAR) ファイルの作成および実行 (「Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales: *インストール・ガイド*」で説明) に精通していること。
- 標準インストール (出荷時のデフォルト値) に精通していること。

このガイド全体を通して、*INSTALL_DIR* とは、Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales をインストールしたディレクトリーを指します。

開発環境について

必要な開発環境は、実行する作業のタイプによって異なります。

Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales をカスタマイズする場合、変更が意図したとおりに動作するかどうかを確認できるテスト環境が必要です。開発時間を節約するには、テスト環境をカスタマイズして、開発モードでアプリケーションを実行します。

データベースを拡張する場合、yfsdbextn.jar ファイルを yantrashared.jar の前に、すべてのスクリプトの CLASSPATH に含めてください。

任意の Packaged Composite Application (PCA) (例えば、Sterling Call Center および Sterling Store) をインストール済みの場合、アプリケーション・サーバーを始動して開発モードでアプリケーションを実行する前に、yantrautil.jar をアプリケーション・サーバーの CLASSPATH から削除する必要があります。

開発モードは、アプリケーション・サーバーが Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales EAR ファイルから最新バージョンの編集済み JSP ファイルを読み込むのではなく、これを自動的に指定されたのディレクトリーから直接ロードすることを可能にすることによって時間を節約します。このことにより、EAR ファイルを繰り返し作成する必要なく、カスタマイズとテストを繰り返し実行できます。

開発モードでは、UI カスタマイズをすぐにテストすることも可能です。

開発環境を設定する方法は、使用しているアプリケーション・サーバーによって異なります。

WebLogic での開発環境の準備

このタスクについて

WebLogic で EAR を作成せずに Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales を実行できるようにするには、WebLogic で適切な設定を使用してアプリケーションを定義し、次に開始スクリプトを構成してそのアプリケーションで必要な CLASSPATH を設定します。

アプリケーションのディレクトリー構造を設定すると、WebLogic で EAR ファイルからではなく、ファイルから直接読み取れるようになります。

アプリケーションをエクスプロード・モードで実行するよう WebLogic を構成するには、以下の手順を実行します。

手順

1. Windows の場合は *<WEBLOGIC_DOMAIN>*/bin/startWebLogic.cmd スクリプトを編集し (UNIX の場合は startWebLogic.sh)、「オプション」の以下の引数を Java パラメーターとして設定します。

   ```
   -Dsci.opsproxy.disable=Y -Dvendor=shell
   -DvendorFile=/servers.properties
   ```

2. WebLogic サーバーを始動し、WebLogic システム・コンソールを開きます。システム・コンソールには、次のような URL を使用してアクセスすることができます。

   ```
   http://<hostname or ip-address>:<port number of your
   WebLogic Server>/console
   ```

3. WebLogic サーバーのシステム管理者 ID およびパスワードを使用して、システム・コンソールにログインします。
4. 「ドメイン構成 (Domain Structure)」パネルで、「**デプロイメント (Deployments)**」をクリックします。

   既にアプリケーションがデプロイされている場合は、以下の手順で、アプリケーションを停止し削除します。

   a. 既存のアプリケーションのデプロイメントを停止するには、以下の手順を実行します。
      - 削除する当該デプロイメントのボックスにチェック・マークを付けます。
      - 「**停止**」をクリックし、ポップアップ・メニューから「**今すぐ強制終了 (Force Stop Now)**」を選択します。
      - 「アプリケーション・アシスタントの削除 (Delete Application Assistant)」で、「**はい**」をクリックします。
      - 「メッセージ (Messages)」に、「**選択したデプロイメントは停止するよう要求されました (Selected Deployments have been requested to stop)**」というメッセージが表示されます。

   b. 既存のアプリケーションのデプロイメントを削除するには、以下の手順を実行します。
      - 削除する当該デプロイメントのボックスにチェック・マークを付けます。
      - 「**削除 (Delete)**」をクリックします。

- 「アプリケーション・アシスタントの削除 (Delete Application Assistant)」で、「**はい**」をクリックします。
- 「メッセージ (Messages)」に、「選択したデプロイメントは削除されました。完了後は、必ず「変更のアクティブ化 (Activate Changes)」をクリックしてください。(Selected Deployments were deleted. Remember to click Activate Changes after you are finished.)」というメッセージが表示されます。
- 「センターの変更 (Change Center)」パネルで、「**変更のアクティブ化 (Activate Changes)**」をクリックします。

5. 「ロケーション」で、<*application_name*>.war ファイルが解凍されたディレクトリーを参照し、「**次へ (Next)**」をクリックします。
6. 「**このデプロイメントをアプリケーションとしてインストール (Install this deployment as an application)**」を選択し、「**次へ (Next)**」をクリックします。
7. 「ソースのアクセシビリティー (Source accessibility)」で、「**デプロイメントを次の場所からアクセス可能にする (I will make the deployment accessible from the following location)**」を選択します。

   「ロケーション先 (Location:)」で、<*application_name*>.war ファイルが解凍されたディレクトリーをロケーションが指していることを確認します。
8. weblogic.xml ファイルを *INSTALL_DIR*/repository/eardata/platform/descriptors/weblogic/WAR/WEB-INF ディレクトリーから *INSTALL_DIR*/extensions/smcfs ディレクトリーにコピーします。
9. ycpapibundle.properties ファイルおよび ycpapibundle_<*lang*>_ *country*.properties (該当する場合) を、<*INSTALL_DIR*>/resources ディレクトリーから <*INSTALL_DIR*>/repository/eardata/smcfs/war/yfscommon ディレクトリーにコピーします。
10. yscpapibundle.properties ファイルおよび yscpapibundle_<*lang*>_ <*country*>.properties (該当する場合) を、<*INSTALL_DIR*>/resources ディレクトリーから <*INSTALL_DIR*>/repository/eardata/smcfs/war/yfscommon ディレクトリーにコピーします。
11. extnbundle.properties ファイルおよび extnbundle_<*lang*>_ <*country*>.properties (該当する場合) を、<*INSTALL_DIR*>/resources/extn ディレクトリーから <*INSTALL_DIR*>/repository/eardata/smcfs/war/yfscommon ディレクトリーにコピーします。
12. (オプション) PCA がインストールされている場合は、以下のファイルを <*INSTALL_DIR*>/repository/eardata/smcfs/war/yfscommon ディレクトリーにコピーします。
    - <*INSTALL_DIR*>/resources/com.yantra.yfc.rcp.common_bundle.properties
    - <*INSTALL_DIR*>/resources/com.yantra.yfc.rcp_bundle.properties
    - <*INSTALL_DIR*>/resources/*PCA_Code*bundle.properties. 例えば、Sterling Call Center および Sterling Store アプリケーションの場合は、ycdbundle.properties ファイルをコピーします。
13. <*INSTALL_DIR*>/extensions/global/webpages ディレクトリーに対して行った拡張をコピーします。

例外: 特定のパッケージにカスタマイズされた JSP を含めるには、それを <*INSTALL_DIR*>/extensions/<*package*>/webpages に配置します。例えば、smcfs または sbc war には、それぞれ <*INSTALL_DIR*>/extensions /smcfs/webpages および <*INSTALL_DIR*>/extensions/sbc/webpages を使用します。

すべてのパッケージにカスタマイズされた JSP を含めるには、それを <*INSTALL_DIR*>/extensions /global/webpages に配置します。

14. インストール・プロセス時に行ったのと同様に、EAR ファイルを再ビルドします。
15. 以下の war ファイルを smcfs.ear ファイルから解凍します。
    - smcfs.war
    - sbc.war
    - sma.war

    次に、これらの war ファイルのそれぞれを希望のディレクトリーに解凍します。
16. smcfs.ear ファイルから残りの jar ファイルを解凍し、すべての解凍された jar ファイルを WEB-INF/lib にコピーします。これにより、 WebLogic からこれらの jar ファイルにアクセスできるようになります。これらの jar ファイルを WebLogic CLASSPATH に含める必要はありません。
17. WebLogic 上の各ディレクトリーを Web アプリケーションとしてデプロイします。
18. 以下の WebLogic Hot Deployment Test Mode 標準を使用して、カスタマイズ内容をテストします。

変更の対象	変更を行うファイル	実行する操作
始動パラメーター	プロパティー	WebLogic を再始動する
UI 拡張性	JSP、JavaScript、CSS、テーマ XML	動的にロードする
ローカライズ・リテラル	alertmessages ファイルおよびローカライズ・バンドル・ファイル	WebLogic を再始動する
データベース拡張	エンティティー XML	entities.jar ファイルを再ビルドしてその jar ファイルをクラスパス・ディレクトリーに含めてから、WebSphere を再始動する
API およびその他のテンプレート・ファイル	テンプレート XML	resources.jar ファイルを再ビルドしてその jar ファイルをクラスパス・ディレクトリーに含めてから、WebLogic を再始動する

次のタスク

Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales では、エクスプロード・モードでのコンテキスト・ヘルプのためのドキュメンテーションの拡張をサポートしていません。エクスプロード・モードでコンテキスト・ヘルプを使用する場合は、別の smcfsdocs.ear ファイルをビルドし、デプロイします。

次に、後述するように WebLogic を構成する必要があります。さらに情報が必要な場合は、WebLogic の資料を参照してください。

サーバーで *application_name*.war ファイルが解凍されたディレクトリーから読み取れるように WebLogic を構成する必要があります。開発環境においてエクスプロード (非 EAR) ・モードでアプリケーションを実行できるよう WebLogic を構成するために必要なステップは、以下に示します。

注:

エクスプロード・モードでデプロイされた Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales は、 EAR モードでデプロイされたソリューションと同様に機能します。エクスプロード・モードに固有のパフォーマンス上の影響はありません。

IBM では、実動においては EAR モードによるデプロイメントを推奨します。1 つのアプリケーション・サーバーで複数のアプリケーションをホストする場合、アプリケーション間で jar またはクラスの競合はありません。これは、各アプリケーションが、その他のアプリケーション・パッケージ (EAR) とは分離された単一の EAR ファイルとしてパッケージ化またはデプロイされるためです。ただし、エクスプロード・モードでは、クラスパスに最初に追加されるクラスが常に考慮されます。

WebSphere での開発環境の構築

このタスクについて

WebSphere 使用時に、Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales に対して行った変更をテストすることができます。

注: IBM では、解凍済み EAR のすべての jar ファイルを WEB-INF/lib ディレクトリーへ直接コピーするよう推奨しています。これにより、これらの jar ファイルが WebSphere にアクセスできるようになり、これらの jar ファイルを WebSphere CLASSPATH に組み込む必要がなくなります。

WebSphere を構成して、エクスプロード・モードでアプリケーションを実行するには、以下の手順を実行します。

手順

1. このデプロイメントに対して、以下の JVM 引数を設定します。

   ```
   -Dsci.opsproxy.disable=Y
   -Dvendor=shell -DvendorFile=/servers.properties
   ```

2. IBM により提供された資料を使用して、EAR をデプロイします。デプロイメント中、WebSphere は WAR ファイルおよび EAR ファイルのすべてのコンテンツを <*WAS_HOME*>/AppServer/profiles/<*PROFILE_NAME*>/installedApps/<*CELL_NAME*>/<*APP_NAME*>/ ディレクトリーにコピーします。

   デプロイメント後は、<*WAS_HOME*>/AppServer/profiles/<*PROFILE_NAME*>/installedApps/<*CELL_NAME*>/<*APP_NAME*>/ ディレクトリーへコピーしたファイルを、必要に応じて、変更することができます。例えば、データベース拡張性の

一環として作成されたカスタム・コードを拡張している場合、カスタム・コード・ファイルを、テスト目的で、<*WAS_HOME*>/AppServer/profiles/<*PROFILE_NAME*>/installedApps/<*CELL_NAME*>/<*APP_NAME*>/ ディレクトリー下にある該当ディレクトリーに直接移動することができます。IBM では、必要に応じてファイルを変更および移動するこの機能を「ホット・デプロイメント」と呼びます。

UI 拡張性の一環として作成されたカスタム JSP を、アプリケーション WAR ファイルに直接取り込むことができます。

注: アプリケーションは、エクスプロード・モードで、コンテキスト・ヘルプのドキュメンテーション拡張をサポートしていません。エクスプロード・モードでコンテキスト・ヘルプを使用する場合は、別の smcfsdocs.ear ファイルをビルドし、デプロイします。

3. 拡張をビルドします。
4. アプリケーション・サーバーを停止します。
5. 拡張のビルドおよびデプロイの一環として作成された jar をコピーし、<*WAS_HOME*>/AppServer/profiles/<*PROFILE_NAME*>/installedApps/<*CELL_NAME*>/<*APP_NAME*> の jar に上書きします。

   例えば、以下のようにします。
   - データベースを拡張している場合は、entities.jar をビルドおよびデプロイし、jar を <*WAS_HOME*>/AppServer/profiles/<*PROFILE_NAME*>/installedApps/*CELL_NAME*/<*APP_NAME*> ディレクトリーにコピーします。
   - UI リソースを拡張している場合は、resources.jar をビルドおよびデプロイし、jar を <*WAS_HOME*>/AppServer/profiles/<*PROFILE_NAME*>/installedApps/<*CELL_NAME*>/<*APP_NAME*> ディレクトリーにコピーします。
6. カスタマイズ済みファイル (例えば、ローカライズ・リテラル・ファイル、JSP) を、該当する <*WAS_HOME*>/AppServer/profiles/<*PROFILE_NAME*>/installedApps/<*CELL_NAME*>/*APP_NAME*/smcfs.war/<*Module_Name*> ディレクトリーにコピーします。

   例えば、「カタログ」モジュールでカスタマイズを行う場合は、<*WAS_HOME*>/AppServer/profiles/<*PROFILE_NAME*>/installedApps/<*CELL_NAME*>/<*APP_NAME*>/smcfs.war/ycm ディレクトリーにファイルを追加します。
7. アプリケーション・サーバーを再始動します。
8. 以下の WebSphere ホット・デプロイメントのテスト・モード標準を使用して、カスタマイズのテストを行います。

変更の対象	変更を行うファイル	実行する操作
始動パラメーター	プロパティー	WebSphere を再始動する。
UI 拡張性	JSP、JavaScript、CSS、テーマ XML	動的にロードする。

変更の対象	変更を行うファイル	実行する操作
ローカライズ・リテラル	alertmessages ファイルおよびローカライズ・バンドル・ファイル	WebSphere を再始動する。
データベース拡張	エンティティー XML	entities.jar ファイルを再ビルドする。 jar を以下のディレクトリーに組み込みます。 *WAS_HOME*/AppServer/profiles/*PROFILE_NAME*/installedApps/*CELL_NAME*/*APP_NAME* WebSphere を再始動する。
API およびその他のテンプレート・ファイル	テンプレート XML	resources.jar ファイルを再ビルドする。 jar を以下のディレクトリーに組み込みます。 *WAS_HOME*/AppServer/profiles/*PROFILE_NAME*/installedApps/*CELL_NAME*/*APP_NAME* WebSphere を再始動する。

JBoss での開発環境の構築

このタスクについて

本セクションでは、JBoss 使用時に、Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales に対して行った変更をテストする方法について説明します。

注: IBM では、解凍済み EAR のすべての jar ファイルを WEB-INF/lib ディレクトリーへ直接コピーするよう推奨しています。これにより、これらの jar ファイルが JBoss にアクセスできるようになり、これらの jar ファイルを JBoss CLASSPATH に組み込む必要がなくなります。

JBoss を構成して、エクスプロード・モードでアプリケーションを実行するには、以下の手順を実行します。

手順

1. Windows の <*JBOSS_DOMAIN*>/bin/run.cmd スクリプト (UNIX の場合は run.sh) を編集し、Java パラメーターとして「オプション」に次の引数を設定します。

   ```
   -Dsci.opsproxy.disable=Y -Dvendor=shell
   -DvendorFile=/servers.properties
   ```

2. インストール・プロセス中に実行したように、EAR ファイルを再ビルドします。

   注: アプリケーションは、エクスプロード・モードで、コンテキスト・ヘルプのドキュメンテーション拡張をサポートしていません。エクスプロード・モードでコンテキスト・ヘルプを使用する場合は、別の smcfsdocs.ear ファイルをビルドし、デプロイします。

3. アプリケーション・サーバーを停止し、以下のステップを実行します。

例えば、「カタログ」モジュールでカスタマイズを行う場合は、<*JBOSS_HOME*>/server/<*SERVER_NAME*>/deploy/smcfs.ear/smcfs.war/ycm ディレクトリーにファイルを追加します。

新規ディレクトリーを作成し、smcfs.ear という名前を付けます。

4. 作成した smcfs.ear ディレクトリー内で、EAR を解凍します。
5. smcfs.ear ディレクトリー内には、smcfs.war ファイルが既に存在します。この .war ファイルをリネームするか、または別のディレクトリーにコピーしてください。
6. smcfs.ear ディレクトリー内に、新規サブディレクトリーを作成し、smcfs.war という名前を付けます。
7. 解凍し、smcfs.war ファイルのすべてのファイルを smcfs.ear/smcfs.war サブディレクトリーに抽出します。
8. ステップ 5 でリネームまたはコピーした smcfs.war ファイルを削除します。
9. 拡張のビルドおよびデプロイの一環として作成された jar をコピーし、<*JBOSS_HOME*>/server/<*SERVER_NAME*>/deploy/smcfs.ear ディレクトリーの jar に上書きします。

   例えば、以下のようにします。

   - データベースを拡張している場合は、entities.jar をビルドおよびデプロイし、jar を <*JBOSS_HOME*>/server/<*SERVER_NAME*>/deploy/smcfs.ear ディレクトリーにコピーします。
   - UI リソースを拡張している場合は、resources.jar をビルドおよびデプロイし、jar を <*JBOSS_HOME*>/server/<*SERVER_NAME*>/deploy/smcfs.ear ディレクトリーにコピーします。
10. カスタマイズ済みファイル (例えば、ローカライズ・リテラル・ファイル、JSP) を、該当する <*JBOSS_HOME*>/server/<*SERVER_NAME*>/deploy/smcfs.ear/smcfs.war ディレクトリーにコピーします。
11. アプリケーション・サーバーを再始動します。
12. デプロイ後、<*JBOSS_HOME*>/server/<*SERVER_NAME*>/deploy ディレクトリーにコピーしたファイルを変更することができます。例えば、データベース拡張性の一環として作成されたカスタム・コードを拡張する場合は、テスト目的で、拡張カスタム・コードを <*JBOSS_HOME*>/server/<*SERVER_NAME*>/deploy ディレクトリー下にある該当ディレクトリーに直接移動することができます。JBoss は変更内容を識別し、アプリケーションを再デプロイします (ホット・デプロイメント)。
13. 以下のテーブルに記述された JBoss ホット・デプロイメントのテスト・モード標準を使用して、カスタマイズのテストを行います。

変更の対象	変更を行うファイル	実行する操作
始動パラメーター	プロパティー	JBoss を再始動する。
UI 拡張性	JSP、JavaScript、 CSS、テーマ XML	動的にロードする。
ローカライズ・リテラル	alertmessages ファイルおよびローカライズ・バンドル・ファイル	JBoss を再始動する。

変更の対象	変更を行うファイル	実行する操作
データベース拡張	エンティティー XML	entities.jar ファイルを再ビルドし、*JBOSS_HOME*/server/*SERVER_NAME*/deploysmcfs.ear ディレクトリーに jar を組み込み、JBoss を再始動する。
API およびその他のテンプレート・ファイル	テンプレート XML	resources.jar ファイルを再ビルドし、*JBOSS_HOME*/server/*SERVER_NAME*/deploy/smcfs.ear ディレクトリーに jar を組み込み、JBoss を再始動する。
アプリケーション構成ファイル	web.xml、application.xml、プロパティー・ファイル	JBoss を再始動する。

開発環境での開発とテスト

ここまでで開発環境を設定したので、本書に提供される手順を使用して、Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales のカスタマイズを開始する準備ができました。このセクションでは、アイコンを使用して、UI のカスタマイズを即時テストできるようにする方法について説明します。

注: 以下のファイルに変更を行った場合は、アプリケーション・サーバーを再始動する必要があります。

- datatypes.xml file
- yfsdatatypemap.xml file

UI カスタマイズのテスト

このタスクについて

リソース階層ツリー内の UI リソースを変更後、「キャッシュの更新 (Cache Refresh)」アイコンを使用して、変更をすぐにテストすることができます。

「キャッシュの更新 (cache refresh)」アイコンを使用するには、次の手順を実行します。

手順

1. 必要に応じて、変更を行います。
2. 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。
   - あるエンティティーとその子リソースを更新する場合は、特定のエンティティーを選択し、  「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は、  「キャッシュの*更新*」アイコンを選択します。
3. 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。

UI キャッシュ更新アクションの構成

標準の Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales デプロイメントでは、リソース階層ツリーで構成の変更を行っても、アプリケーション・サーバーが再始動されるまで有効になりません。これは、UI の拡張のテストが時間のかかるアクティビティーとなる可能性があることを意味します。そのため、アプリケーションではリソース階層ツリー内のアクションが用意されています。これらのアクションにより、リソースを更新して、変更が即時テストされるようにすることができます。

これらのアクションは、開発環境でのみ有効にされます。デプロイメント環境では動作しません。このセクションでは、これらのアクションを有効にする方法について説明します。これらのアクションは、デプロイメント環境では無効にする必要があります。

リソース・キャッシュ更新アクションの構成

このタスクについて

リソース・キャッシュ更新アクションを構成するには、次の手順を実行します。

手順

1. <*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。
2. これにより、Applications Manager のリソース階層ツリーで次のアクションが有効になります。
   -  「キャッシュの更新」アイコン - すべてのリソースを最新表示します。
   -  「エンティティーの更新 (Refresh Entity)」アイコン - 選択したエンティティー・リソースとその子リソースを最新表示します。

   「キャッシュの更新」アイコンの使用については、『UI カスタマイズのテスト』を参照してください。

第 4 章 Microsoft COM+ を使用したカスタマイズ

Microsoft COM+ の前提条件

Microsoft COM+ を使用する際は、Windows サーバー上でアプリケーションを作成および構成する必要があります。またクライアント・プロキシーを作成およびインストールする必要もあります。

Windows での COM+ アプリケーションの作成

このタスクについて

Windows オペレーティング・システムを稼働するサーバー上で Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales COM+ アプリケーションを作成するには、以下の手順を実行します。

手順

1. Windows の「スタート (Start)」メニューから、「**管理ツール (Administrative Tools)**」 > 「**コンポーネント・サービス (Component Services)**」にナビゲートします。
2. コンポーネント・サービス・ツリーから、「**コンポーネント・サービス (Component Services)**」 > 「**コンピューター (Computers)**」 > 「**マイ・コンピューター (My Computer)**」 > **COM+ アプリケーション (COM+ Applications)**」にナビゲートし、「**COM+ アプリケーション (COM+Application)**」を右クリックします。「**新規作成 (New)**」 > 「**アプリケーション (Application)**」を選択します。
3. 「COM アプリケーション・インストール・ウィザードへようこそ (Welcome to COM Application Install Wizard)」画面が表示されたら、「**次へ (Next)**」をクリックします。
4. 「**空のアプリケーションを作成する (Create an Empty Application)**」を選択します。
5. 「空のアプリケーションを作成 (Create Empty Application)」ウィンドウで、以下を入力して「**次へ (Next)**」をクリックします。
   - 「アプリケーション名 (Application Name)」に、アプリケーションの名前を入力します。
   - 「アクティブ化の種類 (Activation Type)」で、「**サーバー・アプリケーション (Server Application)**」を選択します。これにより、コンポーネントが専用プロセスとして開始されるようになります。
6. 「アプリケーション ID の設定 (Set Application Identity)」ウィンドウで、「**このユーザー (This User)**」を選択し、適切な Windows ユーザー名とパスワードを入力します。このユーザーとは、アプリケーションを実行する ID です。そのユーザーが Administrators グループに属していることを確認してください。「**次へ (Next)**」をクリックしてから、「**終了**」をクリックします。

新規に作成された COM+ アプリケーションが、コンポーネント・サービス・ツリーの下に表示されます。

タスクの結果

これで、COM+ アプリケーションにコンポーネントを追加できます。

COM+ アプリケーションへのコンポーネントの追加

このタスクについて

コンポーネントを Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales COM+ アプリケーションに追加する手順は、次のとおりです。

手順

1. コンポーネント・サービス・ツリーから、「**コンポーネント・サービス**」 > 「**コンピューター**」 > 「**自分のコンピューター (My Computer)**」 > 「**COM+ アプリケーション (COM+ Applications)**」 > 「**Selling and Fulfillment Foundation**」 > 「**コンポーネント**」にナビゲートし、「**コンポーネント**」を右クリックします。「**新規**」 > 「**コンポーネント**」を選択します。
2. 「COM コンポーネント・インストール・ウィザードへようこそ (Welcome to COM Component Install Wizard)」画面が表示された後、「**次へ (Next)**」をクリックします。
3. 「**新規コンポーネントのインストール (Install New Component)**」を選択します。
4. *INSTALL_DIR*/bin/YIFComApi.dll を参照します。これを選択し、「**開く**」を選択します。「**次へ (Next)**」および「**終了**」をクリックします。
5. システム・パスに、以下の DLL を格納するディレクトリーが含まれていることを確認します。
   - <*INSTALL_DIR*>/bin/YIFJNIApi.dll
   - <*INSTALL_DIR*>/bin/release/msvcrt.dll
   - <*INSTALL_DIR*>/bin/release/msvcp60.dll
   - jvm.dll (通常、クライアント・マシンの <*JAVA_HOME*>/jre/bin/hotspot にあります)
6. <*INSTALL_DIR*>/properties ディレクトリーに yfs.properties ファイルが含まれていて、<*INSTALL_DIR*>/resources ディレクトリーに yifclient.properties ファイルが含まれていることを確認します。

   注: エントリーを *INSTALL_DIR*/properties/customer_overrides.properties ファイルに追加済みの場合、このファイルが <*INSTALL_DIR*>/properties ディレクトリーに含まれていることを確認します。

COM+ サービスの構成

このタスクについて

Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales COM+ サービスを構成するには、以下の手順を実行します。

手順

1. 「コンポーネント・サービス (Component Services)」ツリーで、新規に作成された COM+ アプリケーションを右クリックします。
2. 「**プロパティー**」を選択します。

   「プロパティー」ダイアログ・ボックスが表示されます。
3. 「**拡張**」タブを選択します。
4. 「サーバー・プロセスのシャットダウン (Server Process Shutdown)」パネルで、「**一定時間アイドルした後でシャットダウンする (Minutes Until Idle Shutdown)**」を選択し、プロセスをシャットダウンするまでに経過する時間 (分単位) を入力して、「**OK**」をクリックします。
5. **COM+** アプリケーションをダブルクリックします。
6. 「**コンポーネント**」をダブルクリックします。
7. 「**YIFComApi.YIFComApi.1**」を右クリックし、「**プロパティー**」を選択します。

   「YIFComApi.YIFComApi.1 のプロパティー (YIFComApi.YIFComApi.1 Properties)」ダイアログ・ボックスが表示されます。
8. 「**アクティブ化 (Activation)**」タブを選択します。
9. 「**オブジェクトのプールを有効にする (Enable object pooling)**」を選択します。
10. 「オブジェクト・プーリング (Object pooling)」セクションで、「**最小プール・サイズ (Minimum pool size)**」および「**最大プール・サイズ (Maximum pool size)**」にコンポーネントの使用率に基づいてサイズを入力します。ハードウェア・リソースを最適に利用できるようにプーリングを構成します。利用可能なハードウェア・リソースが変化すると、プールの構成も変化する場合があります。
11. 「**ジャスト・イン・タイム・アクティベーションを有効にする (Enable Just In Time Activation)**」を選択します。

    JIT アクティベーションにより、オブジェクトのインスタンスに最初の呼び出しが行われる直前にインスタンスがアクティブ化され、その処理が完了するとすぐにインスタンスが非アクティブ化されます。

クライアント・プロキシーの作成

このタスクについて

クライアント・プロキシーを作成するには、以下の手順を実行します。

手順

1. Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales COM+ アプリケーションを右クリックして、「**エクスポート**」を選択します。
2. 「COM アプリケーション・エクスポート・ウィザードへようこそ (Welcome to COM Application Export Wizard)」画面が表示されたら、「**次へ (Next)**」をクリックします。
3. 「アプリケーションのエクスポート (Application Export)」ウィンドウで、エクスポート .MSI ファイルが作成される場所のパスとファイル名を入力します。
4. 「**アプリケーション・プロキシーとしてエクスポート (Export as Application Proxy)**」を選択します。
5. 「**次へ (Next)**」をクリックしてから、「**終了**」をクリックします。

クライアント・プロキシーのインストール

このタスクについて

クライアント・プロキシーをインストールするには、以下の DLL ファイルがシステム・パスに存在することを確認してください。

- <*INSTALL_DIR*>/bin/YIFJNIApi.dll
- <*INSTALL_DIR*>/bin/release/msvcrt.dll
- <*INSTALL_DIR*>/bin/release/msvcp60.dll
- jvm.dll (クライアント・マシンの *JAVA*/jre/bin/hotspot ディレクトリー上に配置されています。*.MSI ファイルをダブルクリックして、クライアント上にコンポーネントをインストールします。)

注: この方法で DLL ファイルを含めることは、リリース 7.7 では非推奨となっています。 Microsoft Windows COM+ 環境から API とサービスを呼び出すための推奨される方法は、Web サービスまたは HTTP を経由する方法です。

第 5 章 ロギング中の機密情報のマスキング

Log4j を使用したロギング中の機密情報のマスク

このタスクについて

log4j ユーティリティーを構成して、クレジット・カード番号やパスワードなどの機密情報がログ・メッセージでログインされることを防ぐことができます。機密情報をマスクするには、アプリケーションで提供されるカスタム log4j レイアウトおよびフィルターを使用し、customer_override.properties ファイルで一連の名前付き正規表現を定義する必要があります。

カスタム log4j レイアウトは、書式設定されたメッセージを取得し、一連の構成可能な正規表現に基づいて結果をフィルター処理します。このカスタム log4j フィルターにより、メッセージを一連の正規表現に対して照合して、一致した場合はメッセージを破棄できます。

ロギング時に機密情報をマスクするには、以下の手順を実行します。

手順

1. カスタム・ロギング構成内でレイアウト・クラス名を SCIFilteredPatternLayout に変更します。 例:

```
<layout
class="com.sterlingcommerce.woodstock.util.frame.logex.SCIFilteredPatternLay
out">
    <param name="ConversionPattern" value="%d:%-7p:%t: %-60m
[%X{AppUserId}]: %-25c{1}%n"/>
    <param name="FilterSet" value="common-filter"/> <!-- Optional -->
</layout>
```

2. カスタム・ロギング構成内でフィルター・クラス名を SCIPatternFilter に変更します。 例:

```
<filter
class="com.sterlingcommerce.woodstock.util.frame.logex.SCIPatternFilter">
     <param name="FilterSet" value="suppress" /> <!-- Optional -->
</filter>
```

3. 以下のプロパティーを使用して、<*INSTALL_DIR*>/properties/customer_overrides.properties ファイル内のメッセージを照合する一連の名前付き正規表現を定義します。

   ```
   filterset.<name>.pattern.<num>=<pattern>
   ```

   次のプロパティーはオプションです。

   ```
   filterset.<name>.replace.<num>=<replace>
   ```

   ここで、<*pattern*> は Java 形式の正規表現で、メッセージ文字列を照合する対象となる正規表現を定義します。`replace` プロパティーはオプションで、式を置き換えるために使用される文字列を定義します。

   以下のプロパティーを設定することにより、デフォルトの FilterSet パラメーターを設定できます。

```
default.filter.filterset=<filter_name>
default.layout.filterset=<layout_name>
```

以下のように、複数のフィルター・セット間で共通の正規表現パターンを定義することもできます。

```
filterset.name.includes=<name1>,<name2>,...
```

<*INSTALL_DIR*>/properties/logfilter.properties.in ファイルを表示して、これらのプロパティーを定義するためのいくつかのサンプル・エントリーを確認できます。

第 6 章 データの妥当性検査

データ検証について

Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales には、要求の入力および出力を検証してサニタイズするデータ検証機能が用意されています。データ検証機能を使用して、入力要求および出力要求において明示的に定義された特性のみを許可し、その他のデータを除去できます。さまざまな要求パラメーターを検証する独自の検証ルールを定義できます。また、ユーザー・インターフェース (UI) にデータを返す前に、データをエンコードすることも可能です。

データ検証またはサニタイズは、パラメーター名、パラメーター値、Cookie 名、Cookie 値などさまざまな種類の入力に対して実行できます。このアプリケーションは、正規表現ベースの検証もサポートしています。

入力バリデーター

入力バリデーターは、特定の入力用に登録されたすべての検証ルールを検出し、検証を実行します。バリデーターは、要求ラッパーによって呼び出され、要求入力を検証します。

デフォルトでは、パラメーター値、パラメーター名などの要求入力を検証するために、入力バリデーターは、アプリケーションによってすぐに使用可能な状態で出荷された正規表現を使用します。すぐに使用可能な状態で出荷された正規表現は、regularexpressions/sc_regularexpressions.xml ファイル (<*INSTALL_DIR*>/jar/platform_afc/5_7/platform_dv.jar 内部にある) で定義されています。

注: 特定の入力に対して検証ルールの一部を緩和する場合、すべてのカスタム検証ルールを入力バリデーターに登録する必要があります。

検証ルール

検証ルールは、入力の検証およびサニタイズを実行します。検証ルールには、検証の実行対象である入力 ID としてプロパティーが含まれています。検証は、対応する入力要求がアクセスされると必ず呼び出されます。検証ルールは、検証する必要がある入力の名前を指定する必要があります。例えば、パラメーターの値を検証するには、検証ルールはそのパラメーターの名前を指定する必要があります。同じ名前を持つ複数の入力が存在する可能性があります。対応する入力を検証するために、すべての検証ルールを入力バリデーターに登録する必要があります。

一部の検証ルールは、アプリケーションによってすぐに使用可能な状態で出荷されます。すぐに使用可能な状態で出荷された検証ルールは、validationrules/sc_validationrules.xml ファイル (<*INSTALL_DIR*>/jar/platform_afc/5_7/platform_dv.jar 内部にある) で定義されています。これらの検証ルールは、同じカテゴリーに属しているすべての入力に対して呼び出されます。例えば、すべての HTTP ヘッダー名は、HTTPHeaderName 正規表現に対して検証されます。

注: 指定された入力に対して定義される検証ルールはありません。以前に指定された検証ルールが入力を検証するために使用されます。

以下のタイプの検証ルールを定義できます。

- 正規表現ベースの検証ルール — このタイプの検証ルールは、正規表現ベースの検証を実行するために設計されています。この検証ルールのタイプは、複数のホワイトリストおよびブラックリストの正規表現をサポートします。
- Java ベースの検証ルール — このタイプの検証ルールは、入力に対して Java ベースの検証およびサニタイズを実行するために設計されています。この検証ルールのタイプは、入力を検証し、次に実装クラスの getValidInput() メソッドを呼び出します。

データ検証の無効化

このタスクについて

デフォルトでは、データ検証はすべての入力要求で有効にされており、これらの入力要求は登録された検証ルールに照らして検証されます。モジュール構成ファイル内にコンテキスト・パラメーターを追加することにより、入力要求でのデータ検証を無効にできます。

データ検証を無効化するには、 EARFILE/WARFILE/WEB-INF フォルダーに配置されている web.xml ファイルに、 context-param 要素のエントリーを以下のように追加します。

```
<context-param>
     <param-name>scui-suppress-request-validation</param-name>
     <param-value>TRUE</param-value>
</context-param>
```

URI のデータ検証のバイパス

デフォルトでは、データ検証はすべての入力要求で有効にされています。ただし、モジュール構成ファイル内にバイパス URI (Universal Resource Indicator) をコンテキスト・パラメーターとして追加することにより、一部の特定の URI の入力要求でのデータ検証をバイパスできます。

データ検証をバイパスするには、EARFILE/WARFILE/WEB-INF フォルダー内の web.xml ファイルで、以下に示すように対象となる各 URI の config-param 要素にエントリーを追加します。

```
<context-param>
   <param-name>request.validation.bypass.uri.1</param-name>
   <param-value>/console/login.jsp</param-value>
</context-param>
<context-param>
   <param-name>request.validation.bypass.uri.2</param-name>
   <param-value>/console/start.jsp</param-value>
</context-param>
<context-param>
   <param-name>request.validation.bypass.uri.endswith.1</param-name>
   <param-value>.js</param-value>
</context-param>
```

```
<context-param>
   <param-name> request.validation.bypass.uri.regex.1</param-name>
   <param-value>^.*test.jsp$</param-value>
</context-param>
```

これらのコンテキスト・パラメーターには、以下のリストで示すように、request.validation.bypass.uri、 request.validation.bypass.uri.endswith、または request.validation.bypass.uri.regex で開始する名前があります。これらのコンテキスト・パラメーターに対して複数のエントリーを定義できます。

- コンテキスト・パラメーターの param-value 要素で指定された値と同じ URI を持つ request.validation.bypass.uri—Any 要求は、バイパスされ検証されません。
- コンテキスト・パラメーターの param-value 要素で指定された値で終了する URI を持つ request.validation.bypass.uri.endswith—Any 要求は、バイパスされ検証されません。
- コンテキスト・パラメーターの param-value 要素で指定されるように、正規表現と一致する request.validation.bypass.uri.regex—Any URI 要求は、バイパスされ検証されません。

データ検証の実装

データ検証を実装するには、以下のタスクを実行します。

- データ・タイプ XML ファイルでの正規表現の定義
- XML ファイルでの正規表現の定義
- 正規表現の登録
- データ・タイプ XML ファイルでの検証ルールの定義
- データ・タイプ XML ファイルで定義された検証ルールの外部化
- XML ファイルでの検証ルールの定義
- 抽象検証ルールの定義
- 抽象検証ルールの拡張
- 検証ルールの登録
- 正規表現の上書き
- 検証ルールの上書き
- エラー・メッセージの定義
- エラー・メッセージのローカライズ
- カスタムの正規表現エラー・メッセージ・プロバイダーの定義
- 検証ルールのローカライズ

データ・タイプ XML ファイルでの正規表現の定義

datatypes.xml ファイル内で、入力の検証のための正規表現を定義できます。これらの正規表現は、一度に複数のデータ・タイプではなく、単一のデータ・タイプに対して使用されている datatypes.xml ファイル内で定義することが推奨されます。そうしないと、複数のデータ・タイプで同じ正規表現を使用する場合に個々のデータ・タイプに対して正規表現を定義すると、そのような正規表現に変更を加えるには、

複数の場所で手動で同様の変更を行う必要が生じ、処理が煩雑になる場合があります。以下の形式で、datatypes.xml ファイル内で入力の検証のための正規表現を定義できます。

```
<DataType Name="Address" Size="70" Type="NVARCHAR">
   <UIType Size="30" UITableSize="30"/>
   <Validation>
      <Regex MaxLength="200" JavaPattern="^[a-zA-Z4-9.@\-\/+=_()
            \{\}\[\],:&amp;apos;&amp;quot;]*$" JSPattern=""/>
   </Validation>
</DataType>
```

注: datatypes.xml ファイル内で定義された正規表現を別個の XML ファイルに書き出して、複数のデータ・タイプにおいてこれらの正規表現の参照を使用することができます。

XML ファイルでの正規表現の定義

入力の検証のための正規表現を個別の XML ファイルで定義して、これらの正規表現への参照を datatypes.xml ファイルで定義される複数のデータ・タイプと、さまざまなルール XML ファイルで定義される複数のルールで使用することができます。このアプローチの主な利点は、1 カ所で正規表現を変更することが容易になり、変更が datatypes.xml ファイルとさまざまなルール XML ファイルの両方で有効になることです。各正規表現には、それに関連付けられた固有の ID があります。正規表現ファイルは、コンテキスト・パラメーターを web.xml ファイルに追加することで、Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales に登録できます。

入力の検証のための正規表現を定義するには、以下の形式で XML ファイルを作成します。

```
<RegularExpressions>
     <RegularExpression id="" javaPattern="" jsPattern="" blacklistErrorMsg=""
       whitelistErrorMsg=""/>
     <RegularExpression id="" javaPattern="" jsPattern=""/>
</RegularExpressions>
```

次の表に、正規表現 XML ファイルの各種要素と属性を示します。

注: 空の属性は、提供されないものと見なされます。例えば、<RegularExpression id="dates" javaPattern="" jsPattern="^[a-zA-Z0-9.,!\-/+=_ :]*$"/> と定義される要素は、<RegularExpression id="dates" jsPattern="^[a-zA-Z0-9.,!\-/+=_ :]*$"/> に相当します。

要素/属性	説明
RegularExpressions	必須。正規表現 XML ファイルには、ルート要素として RegularExpressions を指定する必要があります。
RegularExpression	オプション。各正規表現は、1 つの RegularExpression 要素として定義する必要があります。この要素はゼロ個以上出現できます。
ID	必須。正規表現の固有 ID。この ID はさまざまな場所にあるこの正規表現の参照として使用できます。
javaPattern	オプション。サーバー・サイドの検証を実行するための正規表現パターン。指定されない場合、入力の検証は jsPattern 属性に基づいてクライアント・サイドで発生し、サーバー・サイドの検証は実行されません。

要素/属性	説明
jsPattern	オプション。クライアント・サイドの検証を実行するための正規表現パターン。指定されない場合、入力の検証は javaPattern 属性に基づいてサーバー・サイドで発生し、クライアント・サイドの検証は実行されません。
whitelistErrorMsg	オプション。正規表現のホワイトリスト・パターンが失敗した場合に、エラー・メッセージが表示されるためのバンドル・キー。
blacklistErrorMsg	オプション。正規表現のブラックリスト・パターンが失敗した場合に、エラー・メッセージが表示されるためのバンドル・キー。

例えば、正規表現 XML ファイル内で、以下のように dates という ID を持つ正規表現を定義する場合について考えてみましょう。

```
<RegularExpressions>
    <RegularExpression id="dates" javaPattern="^[a-zA-Z0-9.,!\-/+=_ :]*$"
           jsPattern="^[a-zA-Z0-9.,!\-/+=_:]*$" blacklistErrorMsg=""
           whitelistErrorMsg=""/>
</RegularExpressions>
```

これで、この dates 正規表現をルール XML とデータ・タイプ XML の両方で参照できるようになります。

ルール XML での正規表現の参照

ルール XML でルールを定義する際、 RegularExpression 要素内で正規表現 ID の参照を指定できます。例:

```
<ValidationRules>
    <Rule id="" ruleType="Regex" inputType="" inputName="" uri=""
      maxLength="" minLength="" allowNull="" >
     <Whitelist>
        <RegularExpression ref="dates"/>
     </Rule>
</ValidationRules>
```

データ・タイプ XML での正規表現の参照

datatypes.xml ファイルで正規表現を定義する際、Regex 要素内で正規表現 ID の参照を指定できます。例:

```
<DataTypes>
    <DataType Name="Date" PpcSize="12" Size="7" Type="DATE">
        <Validation>
             <Regex Ref="dates" />
        </Validation>
        <UIType Size="8" UITableSize="15"/>
   </DataType>
</DataTypes>
```

正規表現の登録

Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales を使用して正規表現ファイルを登録するには、以下のどちらかの方法で、各正規表現ファイルの新規コンテキスト・パラメーターを web.xml ファイル (EARFILE/WARFILE/WEB-INF ディレクトリー内に配置) に追加します。

```
<context-param>
     <param-name>scui-regex-file-<unique-identifier></param-name>
     <param-value><file-path-inside-webapp>::<load order></param-value>
</context-param>
```

または、

```
<context-param>
     <param-name>scui-regex-file-unique-identifier</param-name>
     <param-value><fully-qualified-name-of-the-file>::<load order></param-value>
</context-param>
```

ここで、<*load order*> は、正規表現ファイルがロードされなければならない順序を定義します。ロード順序の数値が最小のファイルが、最初に読み込まれます。複数のファイルに同じロード順序を付けることができます。ロード順序が定義されていないと、システムは数値をゼロと見なします。

注: 正規表現ファイルを登録するには、500 を超えるロード順序を使用する必要があります。

例:

```
<context-param>
     <param-name>scui-regex-file-fwk-1</param-name>
     <param-value>/WEB-INF/regex1.xml</param-value>
</context-param>
<context-param>
     <param-name>scui-regx-file-fwk-2</param-name>
     <param-value>/WEB-INF/regex2.xml::1</param-value>
</context-param>
<context-param>
     <param-name>scui-regx-file-fwk-3</param-name>
     <param-value>/com/test/regex-used-from-datatypes.xml::2</param-value>
</context-param>
```

注: 正規表現を含む正規表現ファイル (datatypes.xml ファイルから参照される) は JAR 内部に置かれる必要があり、JAR は APP 動的クラスパスで利用可能でなければなりません。その他のルールを含む正規表現ファイル (datatypes.xml ファイルから参照されない) は <*INSTALL_DIR*>/repository ディレクトリーに置かれる必要があります。IBM では、このようなファイルを EARFILE/WARFILE/WEB-INF ディレクトリー内部に置き、HTTP アクセスによりアクセスできなくするよう推奨しています。

データ・タイプ XML ファイルでの検証ルールの定義

datatypes.xml ファイルを使用して、検証ルールを登録できます。この方法による検証ルールの登録は、パラメーター値の入力の場合にのみ使用可能です。パラメーターのデータ・タイプは、データ・タイプ・マップを使用して推定されます。そして、パラメーター値はそのデータ・タイプに対して登録された検証ルールを使用して検証されます。

HTML UI フレームワークに基づいたアプリケーションでは、以下に示すように、datatypes.xml ファイル内に正規表現または Java ベースの検証ルールを登録できます。

```
<DataType Name="Address" Size="70" Type="NVARCHAR">
   <UIType Size="30" UITableSize="30"/>
     <Validation>
        <Regex JavaPattern="<pattern>" JSPattern="<pattern>" allowNull="false"/>
```

```
        <Impl JavaClass="com.sterlingcommerce.test.MyRuleClass"
              JSFunctionName="myJavascriptFunction"/>
     </Validation>
</DataType>
```

デフォルトでは、<Regex> 要素では、検証ルールの最大サイズはデータ・タイプのサイズに設定されています。MaxLength 属性を使用して、検証ルールの最大サイズを上書きできます。また、MinLength 属性を使用して、検証ルールの最小サイズを設定できます。同様に、JSPattern、JavaPattern、および AllowNull などのその他の属性を上書きできます。例:

```
<DataType Size="5" Name="Pincode" Type="NUMBER">
   <Validation>
     <Regex MaxLength="200" MinLength="3" JavaPattern="^[a-zA-Z0-9.,!\-/+=:]*$"
         JSPattern="^[a-zA-Z0-9.,!\-/+=_ :]*$"  allowNull="true"/>
   </Validation>
</DataType>
```

注: Java ベースの検証ルール・クラスには、完全修飾クラス名が必要です。また、このクラスでは ISCValidationRule インターフェースを実装する必要があります。

注: datatypes.xml ファイルでは、クライアント自体で入力を検証できるよう、javascript パターンと関数を定義することもできます。これらのクライアント側の検証はクライアント上でフェッチされ、すべての対応する入力はこれらのクライアント側の検証によって検証されます。

データ・タイプ XML ファイルで定義された検証ルールの外部化

抽象ルールと同様に datatypes.xml ファイルで定義された検証ルールも外部化できます。まず、そのようなルールを抽象ルールとして定義し、その後この抽象ルールを datatypes.xml ファイル内で拡張します。

例えば、datatypes.xml ファイルで以下のような検証ルールを定義したと考えましょう。

```
<DataTypes>
   <DataType Name="Date" PpcSize="12" Size="7" Type="DATE">
      <Validation>
         <Regex MaxLength="200" JavaPattern="^[a-zA-Z0-9.,!\-/+=_:]*$"
             JSPattern="^[a-zA-Z0-9.,!\-/+=_ :]*$"/>
      </Validation>
      <UIType Size="8" UITableSize="15"/>
   </DataType>
</DataTypes>
```

この検証ルールを、以下のようにルール XML ファイル内で抽象ルールとして外部化できます。

```
<ValidationRules>
   <Rule id="abstract1" ruleType="Regex" abstract="true" maxLength="100" minLength="0">
      <Whitelist>
         <RegularExpression ref="ref1"/>
      </Whitelist>
   </Rule>
</ValidationRules>
```

これで、Rule 要素に extends 属性を追加することにより、 datatypes.xml ファイル内のこの抽象検証ルール (abstract1) を参照することができます。例:

```
<DataTypes>
   <DataType Name="Date" PpcSize="12" Size="7" Type="DATE">
      <Validation>
         <Rule Extends="abstract1" />
      </Validation>
      <UIType Size="8" UITableSize="15"/>
   </DataType>
</DataTypes>
```

XML ファイルでの検証ルールの定義

データ・タイプやパラメーター値の検証に関連していない検証ルールは、datatypes.xml ファイルでは定義できません。ただし、新しいルール XML ファイルを作成することにより、そのようなルールを定義できます。ルール XML ファイルは、web.xml ファイル内の各ルール XML ファイルにコンテキスト・パラメーターを追加することで、Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales に登録できます。

入力の検証のルールを定義するには、以下の形式で XML ファイルを作成します。

```
<ValidationRules>
   <Rule id="" ruleType="Regex" inputType="" inputName="" uri="" maxLength=""
         minLength="" allowNull="" >
      <Whitelist>
         <RegularExpression ref="" />
      </Whitelist>
      <Blacklist>
         <RegularExpression ref="" />
      </Blacklist>
   </Rule>
   <Rule id="" ruleType="Java" inputType="" inputName="" uri="" impl="" />
   <Rule id="abstract1" ruleType="Java" abstract="true" impl="" />
   <Rule id="" extends="abstract1" inputType="" inputName="" uri="" />
   <Rule id="abstract2" ruleType="Regex" abstract="true" maxLength=""
         minLength="" allowNull="" >
      <Whitelist>
         <RegularExpression ref="" />
      </Whitelist>
      <Blacklist>
         <RegularExpression ref="" />
      </Blacklist>
   </Rule>
   <Rule id="" extends="abstract2" inputType="" inputName="" uri="" maxLength=""
         minLength="" allowNull="" />
</ValidationRules>
```

次の表に、ルール XML ファイルの各種要素と属性を示します。

注: 空の属性は、提供されないものと見なされます。例えば、<Rule id="dates" ruleType="Regex" inputType="" inputName="" uri="" maxLength="" minLength="" allowNull="" > と定義される要素は、<Rule id="dates" ruleType="Regex" inputName=""> に相当します。この動作の例外は、inputName 属性です。inputName 属性には空の文字列を含めることができ、それは inputName 属性では有効な値となります。

要素/属性	説明
ValidationRules	必須。ルール XML ファイルには、ルート要素として ValidationRules を指定する必要があります。

要素/属性	説明
Rule	オプション。各ルール (java ルールまたは正規表現ルール) は、1 つの Rule 要素として定義する必要があります。この要素はゼロ個以上出現できます。
ID	オプション。ルールの固有 ID。 注: 既存のルールを拡張する場合は、この属性は必須です。指定されない場合、既存のルールは拡張できません。重複した ID が見つかり、それが拡張にならない場合、例外がスローされます。
abstract	オプション。ルールが抽象ルールの場合、この属性の値を true に設定してください。抽象ルールは別のルールによって拡張可能です。
extends	オプション。拡張される抽象ルールの ID。
ruleType	必須。ルールのタイプ。 有効な値: Java、Regex。 **注:** ルールが抽象ルールを拡張する場合、この属性はしてはなりません。
inputType	必須。入力のタイプ。 有効な値: HTTPParameterValue、HTTPParameterName、HTTPCookieValue、 HTTPCookieName、HTTPHeaderValue、HTTPHeaderName、 HTTPScheme、HTTPServerName、 HTTPContextPath、HTTPPath、HTTPQueryString、 HTTPURI、 HTTPURL、HTTPJSESSIONID、 HTTPServletPath、 JavascriptClient。 **注:** JavascriptClient 入力タイプは、データ・タイプが関連付けられていない、入力のクライアント上の検証ルールをフェッチするために使用されます。 **注:** 入力タイプ JavascriptClient には、グローバルまたはデフォルトのルールのみを定義できます。そのため、属性 inputName の有効な値は、 _global_ または _default_ です。属性 uri の値は無視されます。 impl 属性には、javascript メソッドの名前が含まれている必要があります。属性 ref により定義される正規表現には、javascript 正規表現が含まれる必要があります。 **注:** 抽象ルールの場合、この属性は存在してはなりません。
inputName	必須。入力の名前。 **注:** 抽象ルールの場合、この属性は存在してはなりません。
uri	オプション。ルールが有効である必要のある URI。 **注:** 抽象ルールの場合、この属性は存在してはなりません。
ルール・タイプ Java の子要素および属性	
impl	必須。ルールの実装クラスの完全修飾名。このクラスは ISCValidaitonRule クラスを拡張する必要があります。 **注:** ルールが抽象ルールを拡張する場合、この属性はしてはなりません。
ルール・タイプ Regex の子要素および属性	

要素/属性	説明
maxLength	オプション。許可される最大長。 有効な値: 数値、文字列。 **注:** ルールが抽象ルールを拡張する場合、この値は抽象ルールに定義された maxLength を上書きします。
minLength	オプション。許可される最小長。有効な値: 数値、文字列。 **注:** ルールが抽象ルールを拡張する場合、この値は抽象ルールに定義された minLength を上書きします。
allowNull	オプション。入力の値を要求する場合は、この属性の値を false に設定します。 有効な値: true、false。 **注:** ルールが抽象ルールを拡張する場合、この値は抽象ルールに定義された allowNull を上書きします。
Whitelist	オプション。ホワイトリストのパターンのコンテナー要素。この要素はゼロまたは 1 個出現できます。 **注:** ルールが抽象ルールを拡張する場合、この属性はしてはなりません。
Blacklist	オプション。ブラックリストのパターンのコンテナー要素。この要素はゼロまたは 1 個出現できます。 **注:** ルールが抽象ルールを拡張する場合、この属性はしてはなりません。
RegularExpression	オプション。この要素はゼロまたは複数個出現できます。
ref	必須。正規表現の定義の参照。

抽象検証ルールの定義

抽象検証ルールには、inputType、inputName、および uri 属性がありません。これらの属性は、抽象検証ルールを拡張する検証ルールにより提供されます。抽象属性を true に設定することにより、検証ルールを抽象検証ルールとして定義できます。例:

```
<ValidationRules>
   <Rule id="abstract1" ruleType="Regex" abstract="true" maxLength="10"
         minLength="0" allowNull="false" >
      <Whitelist>
         <RegularExpression ref="regex1" />
      </Whitelist>
   </Rule>
   <Rule id="abstract2" ruleType="Java" impl="com.sterling.validation.testRule"
         abstract="true">
   </Rule>
</ValidationRules>
```

注: Web UI フレームワーク・ベースのアプリケーションの XAPI 入力で使用される JSON パラメーター値の場合、フレームワークでは抽象ルール定義に uifwkimpl-json-xapi-input-param-value という ID を指定します。

JSON 形式であり Web UI フレームワーク・ベースのアプリケーション内の XAPI 呼び出しで使用されているすべての入力には、ルール定義 uifwkimpl-json-xapi-input-param-value を拡張する必要があります。例:

```
<Rule id="sampleRule1" extends="uifwkimpl-json-xapi-input-param-value"
inputType="HTTPParameterValue" inputName="getCategoriesList"/>
```

抽象ルール uifwkimpl-json-xapi-input-param-value は、Java タイプです。

抽象検証ルールの拡張

Rule 要素に extends 属性を追加することにより、抽象ルールを拡張できます。extends 属性の値には、拡張する対象の抽象ルールの ID が含まれている必要があります。例:

```
<ValidationRules>
   <Rule id="impl1" extends="abstract1" inputType="HTTPParameterValue"
         inputName="Test" uri="/console/home.do" maxLength="100">
   </Rule>
   <Rule id="impl2" extends="abstract2" inputType="HTTPParameterValue"
         inputName="Test1" uri="/console/home.do">
   </Rule>
</ValidationRules>
```

この場合、ルール impl1 に定義される maxLength は、ルール abstract1 に定義される maxLength を上書きします。

検証ルールの登録

Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales を使用してルール XML ファイルを登録するには、以下のどちらかの方法で、各ルール XML ファイルの新規コンテキスト・パラメーターを web.xml ファイル (EARFILE/WARFILE/WEB-INF ディレクトリー内に配置) に追加します。

```
<context-param>
   <param-name>scui-validation-rules-file-<unique-identifier></param-name>
   <param-value><file-path-inside-webapp>::<load order></param-value>
</context-param>
```

または、

```
<context-param>
   <param-name>scui-validation-rules-file-unique-identifier</param-name>
   <param-value>fully-qualified-name-of-the-file::load order</param-value>
</context-param>
```

ここで、*load order* は、ルール・ファイルがロードされなければならない順序を定義します。ロード順序の数値が最小のファイルが、最初に読み込まれます。複数のファイルに同じロード順序を付けることができます。ロード順序が定義されていないと、システムは数値をゼロと見なします

注: 検証ルール・ファイルを登録するには、500 を超えるロード順序を使用する必要があります。

例:

```
<context-param>
   <param-name>scui-validation-rules-file-fwk-1</param-name>
   <param-value>/WEB-INF/rules1.xml</param-value>
</context-param>
<context-param>
   <param-name>scui-validation-rules-file-fwk-2</param-name>
   <param-value>/WEB-INF/rules2.xml::1</param-value>
</context-param>
```

```
<context-param>
   <param-name>scui-validation-rules-file-fwk-3</param-name>
   <param-value>/com/test/rules-being-used-from-datatypes.xml::2</param-value>
</context-param>
```

注: 正規表現を含む検証ルール (datatypes.xml ファイルから参照される) は JAR 内部に置かれる必要があり、JAR は APP 動的クラスパスで利用可能でなければなりません。その他のルールを含む正規表現ファイル (datatypes.xml ファイルから参照されない) は *<INSTALL_DIR>*/repository ディレクトリーに置かれる必要があります。IBM では、このようなファイルを EARFILE/WARFILE/WEB-INF ディレクトリー内部に置き、HTTP アクセスによりアクセスできなくするよう推奨しています。

正規表現の上書き

カスタム正規表現ファイル内に同様のエントリーを定義および登録することで、既存の正規表現を上書きすることができます。正規表現を定義する際、正規表現に対して同じ ID を指定する必要があります。より高いロード順序で新しい正規表現 XML ファイルを登録する必要があります。

検証ルールの上書き

カスタムのルール・ファイル内に同様のエントリーを定義および登録することで、既存の検証ルールを上書きできます。ルールを定義する際、以下の属性に対して同じ値を指定する必要があります。

- ID
- inputName
- inputType
- url (提供されている場合)

注: ID 属性は、検証ルール上で類似性チェックを実行するための主要キーです。ID が一致しない場合、または定義されていない場合、検証ルールは拡張されません。また、 ID が一致するが類似性のその他の属性が一致しない場合、例外がスローされます。

注: datatypes.xml ファイルではなく、ルール XML を使用してのみ、登録済みのルールを上書きできます。また、抽象ルールの場合、 ID 属性のみがチェックされます。

より高いロード順序で新しいルール XML ファイルを登録する必要があります。

注: 入力のための検証ルールを登録する際は、以下の inputTypes 定数を使用する必要があります。

- HTTPParameterValue
- HTTPParameterName
- HTTPCookieValue
- HTTPCookieName
- HTTPHeaderValue
- HTTPHeaderName
- HTTPScheme

- HTTPServerName
- HTTPContextPath
- HTTPPath
- HTTPQueryString
- HTTPURI
- HTTPURL
- HTTPJSESSIONID
- HTTPServletPath
- JavascriptClient

検証ルールを検索するためのアダプターの定義

これはオプションのタスクです。アダプターを定義して、検証ルールを検索し、そのルールを使用してパラメーター値を検証することができます。このアダプター・クラスでは、ISCUIInputValidationAdapter インターフェースを実装する必要があり、コンテキスト・パラメーターとしてアプリケーションに登録されていなければなりません。例:

```
<context-param>
     <param-name>scui-param-value-validation-adapter</param-name>
     <param-value>test.MyParamValueValidationAdapter</param-value>
</context-param>
```

ISCUIInputValidationAdapter インターフェースの getValidationRules() メソッドを実装し、name 引数でパラメーター名を渡す必要があります。

パラメーター値を検証する際、システムは登録されたアダプターを呼び出して、パラメーター値の検証に使用するルールを検索します。getValidationRules() メソッドでは、渡されたパラメーター名に登録されたすべてのルールを返すことも、ロジックを指定してその他のルールを検索することもできます。アダプターが何も登録されていない場合、システムは該当するパラメーター名に登録されたすべてのルールを、グローバル・ルールまたはデフォルト・ルールとともに使用して、パラメーター値を検証します。

検証ルールを検索するための URI ベースのアダプターの定義

これはオプションのタスクです。URI ベースのアダプターを定義して、検証ルールを検索するために使用し、その検証ルールを使用してパラメーター値を検証できます。このアダプター・クラスでは、 ISCUIInputValidationAdapter インターフェースを実装する必要があり、以下のようにコンテキスト・パラメーターとしてアプリケーションに登録されていなければなりません。

```
<context-param>
<param-name>scui-param-value-validation-adapter::<uri-without-context-path></param-name>
<param-value><fully-qualified-adapter-class-name><>/param-value>
</context-param>
```

例:

```
<context-param>
<param-name>scui-param-value-validation-adapter::/console/exception.list</param-name>
<param-value>test.Adapter</param-value>
</context-param>
```

ISCUIInputValidationAdapter インターフェースの getValidationRules() メソッドを実装し、name 引数でパラメーター名を渡す必要があります。

パラメーター値を検証する際、システムは登録されたアダプターを呼び出して、パラメーター値の検証に使用する URI ベースのルールを検索します。getValidationRules() メソッドでは、渡されたパラメーター名に登録されたすべての URI ベースのルールを返すことも、ロジックを指定してその他のルールを検索することもできます。アダプターが何も登録されていない場合、システムは該当するパラメーター名に登録されたすべての URI ベースのルールを (グローバル・ルールまたはデフォルト・ルールとともに) 使用して、パラメーター値を検証します。

登録済みの検証ルールの削除

SCValidator クラスの以下のメソッドのいずれかを呼び出すことにより、登録済みの検証ルールを削除できます。

- removeDefaultRules (String inputType)
- removeGlobalRules (String inputType)
- removeRules (String name、String inputType)
- removeRules (ISCValidationRuleKey name、String inputType)

例外の処理

要求の検証時に、無効な入力が見つかると、SCUIRequestValidationException 例外がスローされます。EARFILE/WARFILE/WEB-INF フォルダーに配置された web.xml ファイルに、 TRUE の値を持つ scui-suppress-validation-exception コンテキスト・パラメーターを追加することにより、このデフォルトの動作を上書きできます。例:

```
<context-param>■
    <param-name>scui-suppress-validation-exception</param-name>
    <param-value>TRUE</param-value>
</context-param>
```

このパラメーターの値を TRUE に設定すると、すべての検証の例外は、以下のように ArrayList を使用してアクセス可能なリストに追加されます。

```
ArrayList<SCUIRequestValidationException>
SCUIWebValidationUtils.getValidaionErrorList(HttpServletRequest request)
```

グローバル例外ハンドラーを定義することもできます。検証の例外が検出されず、SCUISafeRequestFilter に戻ると、要求は対応するグローバル・エラー・ハンドラー・サーブレット・コンテナーに送信されます。

このグローバル例外ハンドラーと要求メソッドは、 EARFILE/WARFILE/WEB-INF フォルダー内の web.xml ファイル内にコンテキスト・パラメーターとして定義できます。例:

```
<context-param>
   <param-name>scui-global-validation-exception-handler-path
   </param-name>
   <param-value><path_to_global_exception_handler></param-value>
</context-param>
<context-param>
```

```
   <param-name><scui-global-validation-exception-handler-method>
   </param-name>
   <param-value>FORWARD|INCLUDE|REDIRECT</param-value>
</context-param>
```

Web UI フレームワークに基づいたアプリケーションには、Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales で /jsps/datavalidationerror.jsp がデフォルトの例外ハンドラーとして用意されています。

HTML UI フレームワークに基づいたアプリケーションには、Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales で /common/datavalidationerror.jsp がデフォルトの例外ハンドラーとして用意されています。

また、Web UI フレームワークでは、Struts アクションの結果として「DATAVALIDATIONERROR」が追加されました。これは無効な要求の場合に返されます。これらの Struts アクションには、この結果タイプと対応するパス (say /jsps/datavalidationerror.jsp) を定義できます。

デフォルトでは、グローバル例外ハンドラー・メソッドは FORWARD に設定されます。

エラー・メッセージの定義

各正規表現上でエラー・メッセージを定義して、コンテキストに依存するエラーを表示できます。エラー・メッセージとは、さまざまなバンドル・ファイルで定義される、ローカライズ可能なバンドル・エントリーです。例:

```
<RegularExpressions>
   <RegularExpression id="dates" javaPattern="^[a-zA-Z0-9]*$"
      jsPattern="^[a-zA-Z0-9]*$"  whitelistErrorMsg="only_alphanumeric_chars_allowed"
      blacklistErrorMsg="alphanumeric_chars_not_allowed"/>
</RegularExpressions>
```

bundle.properties ファイルでは、次のエントリーを追加します。

```
only_alphanumeric_chars_allowed={1} must contain only alphanumeric characters.
alphanumeric_chars_not_allowed={1} must not contain any alphanumeric characters.
```

この場合、正規表現の dates が入力の検証で失敗すると、バンドル・キーで定義されたエラー文字列 only_alphanumeric_chars_allowed または alphanumeric_chars_not_allowed がエラー・メッセージとして表示されます。検証される入力の名前は、2 番目のメッセージ・フォーマッターとして使用されます。最初のメッセージ・フォーマッターは正規表現です。

注: whitelistErrorMsg または blacklistErrorMsg 属性が定義されない場合、デフォルトのエラー・メッセージが表示されます。

エラー・メッセージのローカライズ

個々の正規表現に定義されたエラー・メッセージをローカライズできます。

HTML UI ベースのアプリケーションでは、エラー・メッセージのバンドル・キーはサーバー側の検証では Java バンドル・ファイル内に、クライアント側の検証では JavaScript バンドル内に存在する必要があります。

Web UI フレームワーク・ベースのアプリケーションでは、エラー・メッセージのバンドル・キーはサーバー側の検証では Java バンドル・ファイル内に、クライアント側の検証では JavaScript バンドル内に存在する必要があります。

RCP ベースのアプリケーションでは、エラー・メッセージのバンドル・キーはサーバー側とクライアント側のどちらの検証でも、Java バンドル・ファイル内に存在する必要があります。

カスタムの正規表現エラー・メッセージ・プロバイダーの定義

これはオプションのタスクです。エラー・メッセージのプロバイダー・クラスを作成することにより、カスタムのエラー・メッセージ・プロバイダーを指定できます。エラー・メッセージのプロバイダー・クラスでは、ISCRegexErrorMessageProvider インスタンスを実装する必要があります。また、エラー・メッセージのプロバイダー・クラスは、以下のように web.xml ファイル (EARFILE/WARFILE/WEB-INF ディレクトリー内に配置されています) に新しいコンテキスト・パラメーターを追加することにより、Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales に登録する必要があります。

```
<context-param>
      <param-name><scui-regular-expression-error-message-provider></param-name>
      <param-value><fully-qualified-class-name></param-value>
</context-param>
```

例:

```
<context-param>
    <param-name>scui-regular-expression-error-message-provider</param-name>
    <param-value>com.text.RegExErrorMsgProvider</param-value>
</context-param>
```

注: エラー・メッセージ・プロバイダー・クラスにより返されるエラー・メッセージは、バンドル・キーである必要があります。エラー・メッセージはローカライズされ、最初のフォーマッターとして渡される正規表現と、2 番目のフォーマッターとして検証される入力により、フォーマットされます。

検証ルールのローカライズ

各ルールは、すべての利用可能なロケールで入力を検証する必要があるため、検証ルールのローカライズは存在しません。それで、ルール・ファイル自体に含まれるすべての利用可能なロケールをサポートするルールを定義する必要があります。

第 7 章 拡張のビルドおよびデプロイ

拡張の作成後の注意事項

Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales に対して行った拡張が満足のいくものであれば (UI とデータベースの拡張やカスタム・コードおよびファイルの作成など)、必ず新しい拡張をビルドおよびデプロイするようにしてください。アプリケーション・エンタープライズ・アーカイブ (EAR) を、作成または変更したすべての Java ファイル、リソース・ファイル、JSP ファイル、カスタム・クラスなどとともに再ビルドおよびデプロイする必要があります。

リソース拡張のビルド

アプリケーション・リソースに拡張をビルドするには、*INSTALL_DIR*/bin ディレクトリーから deployer.sh (または、Windows 上で deployer.cmd) ユーティリティーを実行して、resources.jar ファイルを再ビルドする必要があります。例:

```
./deployer.sh -t resourcejar
```

これは、以下を含むすべてのアプリケーション・リソースに適用されます。

- テーマ、CSS、構成リソース、データ・タイプ・ファイルなど
- 拡張 API、イベント、および XSL テンプレート
- データベース、リソースおよびテンプレートの各ディレクトリーで行われた変更

JSP ファイルまたは JS ファイルの変更を取り込むには、Selling and Fulfillment Foundation EAR を再ビルドします。

任意のリソース XML ファイルを拡張している場合、拡張した *.xml ファイルを <*INSTALL_DIR*>/extensions/global/template/resource フォルダーに配置します。

任意の event.xml ファイルを拡張している場合、拡張した *.xml ファイルを <*INSTALL_DIR*>/extensions/global/template/event フォルダーに配置します。

*.xsl ファイルを拡張している場合、拡張した .xsl ファイルを <*INSTALL_DIR*>/extensions/global/template/xsl フォルダーに配置します。ただし、サービス定義中に template.xsl ファイルの名前を付ける場合、そのパスを /global/template/xsl/<*CUSTOM-TEMPLATE-XSL*> にする必要があります。

すべての拡張 JSP ファイルおよび JS ファイルを <*INSTALL_DIR*>/extensions/global/webpages ディレクトリーに格納します (これらがこのディレクトリーにない場合)。

リソース・バンドルのカスタマイズ

新しいバンドル・エントリーを定義し、すぐに使用可能なバンドル・エントリーをオーバーライドできます。

サーバー・サイドのバンドル・ファイルは、<*INSTALL_DIR*>/resources ディレクトリーにあります。バンドル・ファイルを変更するには、<*INSTALL_DIR*>/extensions/global/resources/extnbundle.properties ファイルの適切なエントリーを追加またはオーバーライドします。

例えば、Detailed_Description=Detailed Description キー値のペアなどの新しいエントリーを以下のように追加できます。

1. キー値のペアを <*INSTALL_DIR*>/extensions/global/resources/extnbundle.properties ファイルに追加します。
2. resources.jar をビルドします。
3. EAR をビルドします。

その他の拡張のビルド

このタスクについて

カスタム・コード拡張 (外部プログラム、拡張 API、提供された Java インターフェースのカスタム実装など) の変更をビルドするには、これらの Java ファイルおよびカスタム・クラスを含む JAR ファイルを生成します。JAR ファイルを作成した後、<*INSTALL_DIR*>/bin ディレクトリーから install3rdParty.sh (または Windows 上で install3rdParty.cmd) ユーティリティーを実行して、新しい JAR ファイルを CLASSPATH 環境関数に含めます。例:

```
./install3rdParty.sh <vendorName> <vendorVersion> <-j | -l | -p | -r | -d >
<filelist> [-targetJVM DCL | EVERY | NOWHERE | APP | AGENT]
```

ここで、<*vendorName*> は、WebLogic、WebSphere、JBoss などのベンダーの名前を表します。<*vendorVersion*> は、ベンダーのバージョンを表します。ファイル・タイプに基づいて、適切な引数を渡します。以下の引数を渡すことができます。

- -j (JAR または圧縮ファイル用)
- -l (共有ライブラリー用)
- -p (プロパティー・ファイル用)
- -r (リソース・プロパティー・ファイル用)
- -d (データベース JAR または圧縮ファイル用)

<*filelist*> は、カスタム・ファイルへのパスを表します。

注: install3rdParty ユーティリティーを実行するときに、このカスタム JAR をアプリケーション・サーバーおよびエージェントで使用できるようにするには、要件に応じて以下のターゲット JVM 引数を渡します。

- DCL — カスタム JAR をメインの動的 classpath.cfg ファイルのみに追加する場合。
- EVERY — カスタム JAR をすべての動的クラスパス・ファイル (例えば、APPDynamicclasspath.cfg、 A GENTDynamicclasspath.cfg、OPSDynamicclasspath.cfg、 ACTIVEMQDynamicclasspath.cfg の各ファイル) に追加する場合。

- NOWHERE — カスタム JAR を <*INSTALL_DIR*>/jar ディレクトリーのみに追加し、動的クラスパス・ファイルのいずれも更新しない場合。
- AGENT — カスタム JAR を AGENTDynamicclasspath.cfg ファイルに追加する場合。
- APP — カスタム JAR を EAR ファイルに追加する場合。

例えば、カスタム JAR を AGENTDynamicclasspath.cfg ファイルに追加する場合、install3rdparty コマンドを以下の引数を指定して実行します。

```
./install3rdParty.sh weblogic 10 -j <Path_to_your_custom_JAR> -targetJVM
AGENT
```

注: Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales が提供するメカニズム (時間トリガー・トランザクション、API、外部プログラムなど) は、改善されたメカニズムに置き換えられることがあります。これらのメカニズムが置き換えられた場合、これらは「非推奨」に指定されます。可能な限り、非推奨のメカニズムではなく新しいメカニズムを使用してください。非推奨のメカニズムを使用する必要がある場合、下位互換性モードで実行する必要があります。また、非推奨メカニズムは 2 つの主要なソフトウェアのバージョンの間サポートされた後、製品から削除されます。

データベース拡張のビルド

このタスクについて

データベースに拡張をビルドするには、<*INSTALL_DIR*>/bin ディレクトリーから deployer.sh (または、Windows 上で deployer.cmd) ユーティリティーを実行して、entities.jar を再ビルドします。例:

```
./deployer.sh -t entitydeployer
```

注: データベース拡張をビルドする前に、すべての拡張ファイルが *INSTALL_DIR*/extensions/global/entities ディレクトリーに格納されていることを確認します。

デフォルトでは、entitydeployer ターゲットを実行するとき、すべてのログ・メッセージが <*INSTALL_DIR*>/logs/entitydeployer.log ファイルに印刷されます。ログ・ファイルおよびコンソールにログ・メッセージを印刷する場合、entitydeployer ターゲットを実行するときに -l info パラメーターを渡します。例:

```
./deployer.sh -t entitydeployer -l info
```

ERD 文書を更新するには、*INSTALL_DIR*/bin ディレクトリーから deployer.sh (または、Windows 上で deployer.cmd) ユーティリティーを実行して、entities.jar を再ビルドします。例:

```
./deployer.sh -t updateERD
```

注: デフォルトでは、entitydeployer ターゲットまたは InstallService スクリプトを実行するとき、dbverify ツールも実行されます。ただし、InstallService スクリプトを実行するときに dbverify ツールに対するこの呼び出しを再び抑制する場合、

<*INSTALL_DIR*>/install/properties/sandbox.cfg ファイルの NO_DBVERIFY プロパティーはオーバーライドされて true に設定されます。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*を参照してください。

データベース拡張のデプロイについては、*拡張のデプロイ*を参照してください。

拡張のデプロイ

必要な Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales 拡張をビルドしたら、それらをデプロイする必要があります。

拡張をデプロイするには、インストール時に行ったように EAR ファイルを再ビルドする必要があります。

IBM では、開発システム上で EAR ファイルを再ビルドおよびデプロイし、まずそこでテストを実施することをお勧めします。次に、拡張を実動システムでデプロイし、再度テストします。

また、実動システムに拡張をデプロイする前に、 <*INSTALL_DIR*>/properties/customer_overrides.properties ファイルが正しく設定されていることを確認してください。例えば、yfs.uidev.refreshResources プロパティーで指定されたキャッシュ更新アイコンが、N に設定されていることを確認してください。customer_overrides.properties ファイルを使用してプロパティーを上書きする方法について詳しくは、「Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales: *プロパティー・ガイド*」を参照してください。

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

テンプレート、データベース拡張、UI リソースなど、Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales によって使用されるリソースを、エンタープライズ・レベルで定義できます。エンタープライズ・レベルのリソースは、デプロイメントのためのサービス・パッケージにバンドルされます。

エンタープライズ・レベルのリソースは、1 つのエンタープライズ・サービス・パッケージとして開発およびパッケージ化できます。このサービス・パッケージにはエンタープライズに搭載する必要のあるすべてのコンポーネントが含まれます。エンタープライズ・レベルのリソースは、固有のリソース ID を使用して識別されます。固有のリソース ID はエンタープライズに属するリソースを特定するために使用されます。固有のリソース ID を使用すると、エンタープライズ・レベルのリソースを簡単にデプロイまたは移動できます。組織を作成する際に、これらのリソース ID を定義します。組織の作成について詳しくは、「Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales: *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。

エンタープライズ・レベルの拡張のビルド

このタスクについて

変更されたエンタープライズ・レベルの拡張 (テンプレート、リソース・ファイル、カスタマイズ Web ページ、エンティティーなど) をビルドする手順は、次のとおりです。

手順

1. descriptor.xml という名前の新しい XML ファイルを作成し、以下のエントリーを追加します。

   ```
   <ExtensionsDescriptor>
      <Package Name="<RESOURCE_IDENTIFIER>"/>
   </ExtensionsDescriptor>
   ```

   ここで、*<RESOURCE_IDENTIFIER>* はエンタープライズに属しているリソースを識別するために定義された固有のリソース ID を表します。

2. エンタープライズ・レベルの拡張ファイルを含むカスタム JAR ファイルを生成します。カスタム JAR ファイルは、作成した拡張に応じて、以下のディレクトリー構造を持っている必要があります。

   注: カスタム JAR を作成する前に、descriptor.xml ファイルを JAR ファイルのルートにコピーします。

   - /template/ *<TEMPLATE_SPECIFIC_FOLDER>* — 拡張テンプレート・ファイル格納用

     ここで、*TEMPLATE_SPECIFIC_FOLDER* は、固有のテンプレートを含むディレクトリーを表します。例:

     - api — API 固有テンプレート格納用
     - email — E メール固有テンプレート格納用
     - event — イベント固有テンプレート格納用
     - userexit — 外部プログラム固有テンプレート格納用

   JAR ファイルのルートに必要なフォルダーを作成することにより、テンプレートの他に、JAR ファイルに JAR ファイル、リソース・ファイル、エンティティー拡張、およびカスタマイズ Web ページを格納できます。例:

   - /jars — 必要な JAR ファイル格納用
   - /uijars — UI 固有 JAR ファイル格納用
   - /entities — 拡張エンティティー XML 格納用
   - /webpages — カスタマイズ Web ページ格納用
   - /resources — 変更されたリソース・ファイル格納用

3. JAR ファイルを作成した後、*INSTALL_DIR*/bin ディレクトリーから InstallExtensions.sh (または Windows 上で InstallExtensions.cmd) ユーティリティーを実行して、新しい JAR ファイルをデプロイします。例:

   ```
   ./InstallExtensions.sh <filename>
   ```

   ここで、*<filename>* は、手順 2 で作成した JAR ファイルへのパスを表します。

次のタスク

テンプレート拡張をビルドした後、resources.jar を再ビルドしたことを確認します。

エンタープライズ・レベルのリソース拡張のビルド

このタスクについて

変更したエンタープライズ・レベルの UI リソース・ファイル (テーマ、CSS、構成リソース、datatypemaps ファイルなど) をビルドするには、*INSTALL_DIR*/bin ディレクトリーから deployer.sh (または、Windows 上で deployer.cmd) ユーティリティーを実行して、resources.jar ファイルを再ビルドします。例:

```
./deployer.sh -t resourcejar
```

JSP ファイルまたは JS ファイルの変更を取り込むには、Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales EAR を再ビルドします。

注: すべての拡張 JSP ファイルおよび JS ファイルが *INSTALL_DIR*/repository/eardata/smcfs/war ディレクトリーに格納されていることを確認します。

エンタープライズ・レベルのデータベース拡張のビルド

このタスクについて

変更したエンタープライズ・レベルのデータベースをビルドするには、*INSTALL_DIR*/bin ディレクトリーから deployer.sh (または、Windows 上で deployer.cmd) ユーティリティーを実行して、entities.jar ファイルを再ビルドします。例:

```
./deployer.sh -t entitydeployer
```

注: データベース拡張をビルドする前に、すべての拡張ファイルが *INSTALL_DIR*/repository/entity/extensions ディレクトリーに格納されていることを確認します。

注: デフォルトでは、entitydeployer ターゲットまたは InstallService スクリプトを実行するとき、dbverify ツールも実行されます。ただし、InstallService スクリプトを実行するときに dbverify ツールに対するこの呼び出しを再び抑制する場合、*INSTALL_DIR*/install/properties/sandbox.cfg ファイルの NO_DBVERIFY プロパティーはオーバーライドされて true に設定されます。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*を参照してください。

デフォルトでは、entitydeployer ターゲットを実行するとき、すべてのログ・メッセージが *INSTALL_DIR*/logs/entitydeployer.log ファイルに印刷されます。ログ・ファイルおよびコンソールにログ・メッセージを印刷する場合、entitydeployer ターゲットを実行するときに -l info パラメーターを渡します。例:

```
./deployer.sh -t entitydeployer -l info
```

エンタープライズ・レベルのテンプレート拡張

InstallExtensions.sh スクリプトの実行後に *INSTALL_DIR*/extensions ディレクトリーで作成されるテンプレート・フォルダーから、拡張されたテンプレートがシステムで自動的に読み取られます。

エンタープライズ・レベルの拡張のデプロイ

このタスクについて

必要なエンタープライズ・レベルの拡張をビルドしたら、それらの拡張をデプロイする必要があります。

エンタープライズ・レベルの拡張をデプロイするには、Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales エンタープライズ・アーカイブ (EAR) を再ビルドおよびデプロイします。EAR をビルドする方法について詳しくは、「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。

注: IBM では、開発システム上で EAR ファイルを再ビルドおよびデプロイし、まずそこでテストを実施することをお勧めします。次に、拡張を実動システムでデプロイし、再度テストします。Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales のデプロイについて詳しくは、「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。

また、実動システムに拡張をデプロイする前に、*INSTALL_DIR*/properties/customer_overrides.properties ファイルが正しく設定されていることを確認してください。例えば、yfs.uidev.refreshResources property プロパティーで指定されたキャッシュ更新アイコンが、N に設定されていることを確認してください。customer_overrides.properties ファイルを使用してプロパティーを上書きする方法について詳しくは、「Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales: *プロパティー・ガイド*」を参照してください。

web.xml のカスタマイズ

複数のアプリケーションに対する web.xml のカスタマイズ

Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales では、 web.xml に以下の拡張を提供しています。

拡張パッケージには「.extn」接尾辞が必要であり、「package_name」属性を使用してパッケージを指定する必要があります。例:

```
<WebComponents Package = "package_name.extn">
```

アプリケーションには、一部の共通の構成を抑制する機能があります。 Suppress 要素を使用することにより、アプリケーションでこれらの構成を抑制できます。抑制条件に一致する要素を削除することで、抑制が実行されます。例:

```
<WebComponents Package = "package_name.extn">
<Suppress>
<servlet><servlet-name>JasperPDFReport</servlet-name></servlet>
```

```
servlet-mapping><servlet-name>JasperPDFReport</servlet-name>
</servlet-mapping>
</Suppress>
<web-app>
<!-- All the web.xml pieces needed, in standard web.xml format. -->
</web-app>
</WebComponents>
```

この例では、サーブレット要素に任意の名前を持つ子サーブレット名が含まれる、すべての構成が抑制されます。

セッション・タイムアウトのための web.xml のカスタマイズ

デフォルトのセッション・タイムアウト値は 6000 秒で、YFS_USER テーブルの SessionTimeout 値から設定されます。別のセッション・タイムアウトを設定するには、 web.xml ファイル内の以下の 2 つのパラメーターを構成します。

- タイムアウト値をファイルから設定可能にするコンテキスト・パラメーター

  注: この値が設定されない場合、セッション・タイムアウト・パラメーターは無視されます。
- 数値を設定するためのセッション・タイムアウト・パラメーター

セッション・タイムアウトをカスタマイズするには、以下の手順を実行します。

1. EARFILE/WARFILE/WEB-INF/web.xml ファイルを編集して、コンテキスト・パラメーター scui-suppress-user-level-sessiontimeout-override を追加します。以下に示すように値を y に設定します。

   ```
   <context-param>
      <param-name>scui-suppress-user-level-sessiontimeout-override</param-name>
      <param-value>y</param-value>
   </context-param>
   ```

   これにより、セッション・タイムアウトがタイムアウト値から設定できるようになります。
2. 以下に示すように web.xml にエントリーを追加して、セッション・タイムアウト構成パラメーターを分単位で設定できるようにします。

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

エンタープライズ・アーカイブ・パッケージのデプロイ

このタスクについて

Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales を使用できるようにするには、アプリケーション EAR ファイルを作成してデプロイする必要があります。

手順

1. アプリケーションをデプロイするために、アプリケーション・サーバーを適切に設定します。アプリケーション・サーバーの設定について詳しくは、インストール文書を参照してください。
2. アプリケーション・サーバーの EAR パッケージを作成します。

- 単一の WAR デプロイのためのアプリケーション EAR ファイルを作成するには、次のコマンドを *<INSTALL_DIR>*/bin ディレクトリーから実行します:
  UNIX

  ```
  ./buildear.sh (.cmd for Windows) -Dappserver=<application server>
  -Dwarfiles=<war file> -Dearfile=<ear file>
  ```

- 複数の WAR デプロイのためのアプリケーション EAR ファイルを作成するには、次のコマンドを *<INSTALL_DIR>*/bin ディレクトリーから実行します。

  ```
  ./buildear.sh (.cmd for Windows) -Dsupport.multi.war=true
  -Dappserver=<application server> -Dwarfiles=<war file,<comma-separated packages>
  -Dearfile=<ear file>
  ```

  追加の WAR ファイルを作成するには、適切なパッケージをコンマで区切って、`-Dwarfiles` 引数の値に追加します。例えば、3 つの WAR ファイルを作成するには、手順 2 のコマンドで `-Dwarfiles` 引数を次のように設定します。

  ```
  -Dwarfiles=<war file 1>,<war file 2>,<war file 3>
  ```

  この手順でコマンドを実行すると、EAR ファイルが *<INSTALL_DIR>*/external_deployments ディレクトリーに作成されます。また、複数の WAR ファイルが EAR ファイルに配置されます。

3. EAR ファイルをアプリケーション・サーバーにデプロイします。EAR ファイルの作成およびデプロイについて詳しくは、インストール文書を参照してください。

1 台のアプリケーション・サーバー上での複数の EAR のデプロイ

Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales では、 1 台のアプリケーション・サーバー上での複数の EAR (エンタープライズ・アーカイブ) のデプロイメントのサポートを提供します。同じアプリケーション・サーバー上で、以下の操作のどちらかを実行できます。

- 同じまたは異なるバージョンのアプリケーションの異なるカスタマイズをデプロイする。
- 同じアプリケーションの異なるバージョンをデプロイする。

1 台のアプリケーション・サーバー上でデプロイ可能な異なる EAR の数は、アプリケーション・サーバーで利用可能なリソースによって異なります。この複数の EAR のデプロイメントをサポートするには、同じアプリケーションの異なるバージョンまたはカスタマイズは、エクスプロード・モードでではなく、 EAR としてデプロイされている必要があります。

注: Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales では、各 EAR ファイルが異なる *<INSTALL_DIR>* ディレクトリーから生成されたと見なします。

複数の EAR をデプロイするには、以下を実行する必要があります。

- JNDI (Java Naming and Directory Interface) コンテキスト名前空間の定義
- コンテキスト・ルート項目の定義

JNDI コンテキスト名前空間の定義

このタスクについて

JNDI (Java Naming and Directory Interface) の競合を回避するには、JNDI コンテキスト名前空間プロパティーを定義する必要があります。JNDI エントリーを定義するには、以下の手順を実行します。

手順

1. <*INSTALL_DIR*>/properties/sandbox.cfg ファイルを編集します。
2. YFS_CONTEXT_NAMESPACE プロパティーを追加して、それに名前を割り当てます。
3. <*INSTALL_DIR*>/bin ディレクトリーから以下のスクリプトを実行して、このプロパティーの値を yifclient.properties ファイルに Munge 処理します。
   - setupfiles.sh (UNIX/Linux の場合)
   - setupfiles.cmd (Windows の場合)

コンテキスト・ルート項目の定義

このタスクについて

build.properties.in ファイルでコンテキスト・ルートを定義するには、WAR ファイル・マッピングを追加する必要があります。これにより、単一のアプリケーション・サーバーにインストールされた Web アプリケーションに固有のコンテキスト・ルートが指定されるようになります。

注: 複数の EAR またはコンテキスト・ルートを使用すると、アプリケーション・サーバー JVM で余分なメモリーが必要になります。テストによると、2 つ目の EAR ファイルをデプロイすると、単一の EAR の場合よりも 2.5 から 3.5 倍のメモリーが必要になります。2 つのデプロイメントをサポートするには、最大 2.5 GB のヒープ領域と 1.2 GB の永続領域が必要になる場合があります。

インストール時に、 JVM 固有の引数を使用して、メモリー不足エラーを回避できます。詳しくは、「Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales: *プロパティー・ガイド*」の ADDITIONAL_ANT_JAVA_TASK_ARGS および ADDITIONAL_ANT_COMPILER_TASK_ARGS の説明を参照してください。

これらの項目を追加するには、以下の手順を実行します。

手順

1. <*INSTALL_DIR*>/install/bin/build.properties.in ファイルを編集します。
2. 任意の Web アプリケーションのコンテキスト・ルート・パスに WAR ファイル・マッピングを追加します。キーはアプリケーションの WAR ファイルの名前である必要があります。例:

```
platformdemo.war=/myplatformdemo
yantrawebservices.war=/yantrawebservices
platformdemodocs.war=/myplatformdemodocs
```

同じ WAR ファイルを 2 回デプロイする場合は、以下に示す例のように WAR ファイル・マッピングを使用してください。

```
platformdemo1.war=platformdemo,platform
platformdemo2.war=platformdemo,platform
```

次に、EAR ビルド時に、次の引数を渡します。

```
-Dwarfiles=platformdemo1,platformdemo2
```

3. *INSTALL_DIR*/bin ディレクトリーから以下のスクリプトを実行して、コンテキスト・ルート WAR ファイル・マッピングを build.properties ファイルに Munge 処理します。
   - setupfiles.sh (UNIX/Linux の場合)
   - setupfiles.cmd (Windows の場合)

タスクの結果

注: yfs.context.namespace プロパティーを指定しており、build.properties.in ファイル内で WAR ファイルのマッピングを指定していない場合、buildEAR.cmd (UNIX/Linux では buildear.sh) スクリプトはエラーをスローし、ユーザーに WAR ファイル・マッピングを指定するよう強制します。また、 build.properties.in ファイルで WAR ファイルのマッピングを指定しており、 yfs.context.namespace プロパティーを指定していない場合は、WAR ファイルのマッピングは無視されます。

さらに、yfs.context.namespace プロパティーの値と WAR ファイル・マッピングの名前は、同じアプリケーション・サーバー上にデプロイする EAR ごとに異なるものにする必要があります。

第 8 章 ファイル名、キーワード、およびその他の規則

予約された特殊文字とキーワードの概要

Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales では、内部的にのみ使用されるキーワードと特殊文字が予約されています。特殊文字の使用方法について詳しくは、「Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales: *API カスタマイズ・ガイド*」を参照してください。

ファイルの名前付け

ファイルを名前付けする際は、IBM では、 A から Z および 0 (ゼロ) から 9 などの、標準のアルファベット文字セットから文字を選択することをお勧めします。そうするなら、アプリケーションを英語以外の言語にローカライズする必要が生じた場合に、ファイルの名前を変更する必要がありません。

予約済みキーワード

一部のキーワードは、Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales で使用するために予約されており、API を使用したプログラミング時およびエラー・コードの作成時には注意が必要です。以下で始まるファイル名およびエラー・コードを作成しないでください。

- DCS
- INV
- OMD
- OMP
- OMR
- OMS
- PLT
- SYS
- WMS
- YCM
- YCP
- YCS
- YDM
- YFC
- YFE
- YFS
- YFX
- YIC

- YIF
- YPM
- YRET

マルチバイト文字の使用

マルチバイト文字を使用する場合は、マルチバイト文字をサポートするようデータベースを構成する必要があります。マルチバイト文字とローカライズについて詳しくは、「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。

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 2011. 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. 2011.

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:

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