Sterling Selling and Fulfillment Foundation

# トランザクションの拡張

*バージョン 9.1*

Sterling Selling and Fulfillment Foundation

# トランザクションの拡張

*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、 31 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
Extending Transactions
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 トランザクションの拡張

## トランザクションの拡張について

実装の機能を拡張することや外部システムと統合することで、プログラマチックに Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales を拡張することができます。

このセクションでは、これらのメカニズムを使用して、拡張性をプログラマチックに実現する方法について説明します。アプリケーションにおけるトランザクション処理のメカニズムは、以下の 2 つの基本的なカテゴリーに分類できます。

- 同期 (要求時) または非同期 (メッセージ駆動) サービス
- 時間トリガー・トランザクション

## 時間トリガー・トランザクションについて

時間トリガー・トランザクションまたはエージェントは、さまざまな個々の機能を自動的に、一定の時間間隔で実行するプログラムです。条件、イベントまたはユーザー入力によって起動することはありません。

また、次のように 3 つのタイプがあります。

- ビジネス・プロセス・トランザクション - 日々のトランザクションを実行します。
- モニター - 処理の遅延および例外について監視し、アラートを送信します。
- パージ - 処理完了後に破棄可能なデータを消去します。

以下のメカニズムのいずれかを使用して、アプリケーションによって提供されるトランザクションを拡張できます。

- 外部プログラム
- イベント・ハンドラー

アプリケーションによって提供される時間トリガー・トランザクション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 (Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales: Configuration Guide)」を参照してください。

## 外部プログラムの使用による標準トランザクションの拡張

Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales は、主要なビジネス・アルゴリズムを拡張またはオーバーライドする能力を提供します。これは、アルゴリズムの実行時に呼び出される外部プログラムによって実現されます。

一般的な外部プログラムは、以下のことができます。

- 独自のロジックを提供することでアプリケーション・ロジックをオーバーライドします。
- アプリケーション・アルゴリズムへの入力をより多く提供することで、アプリケーション・ロジックを拡張します。

例えば、オーダーが複数の出荷に分割される場合、出荷ごとに別々にオーダー価格を計算する必要がある可能性があります。 Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales のオーダー価格計算方法を変更するには、特定の計算または repriceOrder 外部プログラムのアルゴリズムをオーバーライドします。

各外部プログラムは、別個の Java インターフェースです。ビジネス・ロジックをオーバーライドまたは拡大したい外部プログラムのみを実装することを選択できます。

**注:** API または時間トリガー・トランザクションがデフォルトのアルゴリズムを実行している場合は、外部プログラムが呼び出されます。関数の実装が例外をスローする場合、現在のトランザクションはロールバックされます。

各外部プログラムの詳細な説明については、以下の Javadocs パッケージを参照してください。

- com.yantra.ycm.japi.ue
- com.yantra.ycp.japi.ue
- com.yantra.ydm.japi.ue
- com.yantra.yfs.japi.ue

## 外部プログラムの実装およびデプロイ

すべての外部プログラム・クラスは、エージェント・アダプター・スクリプトの CLASSPATH、およびアプリケーション・サーバーでデプロイされた smcfs.ear ファイルで使用可能な JAR ファイルとしてデプロイされる必要があります。 外部プログラムの実装について詳しくは、「Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales: *構成ガイド* (Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales: *Configuration Guide*)」を参照してください。

カスタム・クラスの作成および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* (Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales: *Installation Guide*)」を参照してください。

## 外部プログラムのガイドライン

アプリケーション API 内で外部プログラムを使用する場合は、以下のガイドラインに留意する必要があります。

- 外部プログラムは、特定の情報を返すように構造化されています。したがって、それらの使用法はそのような目的のためのみに制限される必要があります。
- 外部プログラムから、データを変更する API を起動することはできません。これは、親トランザクション (外部プログラムを呼び出すトランザクション) で変更さ

れるデータ、および外部プログラムのカスタム・コード内で変更されるデータと同じデータであるため、エラーが発生しないようにするためです。例えば、同じオーダーに関連する情報を取得する外部プログラムから changeOrder() API を呼び出すことはできません。

ただし、データを変更しない API (選択 API など) は、外部プログラム内で起動できます。例えば、外部プログラムから getOrderDetails() API を呼び出すことができます。

## イベント・ハンドラーの使用による標準トランザクションの拡張

アプリケーションは、処理の特定の時期にイベントを呼び出します。それにより、特定のイベントの発生時に実行されるようにアクションを定義できます。

アプリケーション構成では、イベントはアプリケーションのイベント・ハンドラーを呼び出すように定義されます。イベント・ハンドラーは、トランスポート・サービス層にトランスポートされる前に、イベントによって公開されたデータに特別な処理を実行します。

イベント構成の一部として、サービスを呼び出すことができます。このようなサービスが呼び出されると、INVENTORY_CHANGE などの一部のイベントは、他のイベントが XML としてデータを渡すのに対して、マップとしてデータを渡します。マップとして渡されているデータのイベントでは、サービスがこのようなイベントのために構成される場合、データ・マップは以下のように XML に変換されます。

```
<?xml version="1.0" encoding="UTF-8"?>
<SterlingXML>
   <XML AccountNo="" AdjustmentType="ADJUSTMENT"
       EnterpriseCode="DEFAULT" ItemID="ITEM1"
       ItemKey="2005030116023851364" ..... />
</SterlingXML>
```

**注:** デフォルトでは、マップ・データを持つイベントを呼び出すと、データ・マップは、ルート・タグとして「SterlingXML」を持つ XML に変換されます。このルート・タグ値をオーバーライドしたい場合は、*install_dir*/properties/customer_overrides.properties ファイル内で yfs.sci.event.flow.roottag プロパティー値を新しいルート・タグとして設定する必要があります。

### イベントの構成について

リリース 5.0 以降では、アクションをサービスに関連付けて、必要とされる特別な処理を実行できます。例えば、イベント PUBLISH_SHIP_ADVICE は、アプリケーション・イベント・ハンドラーを呼び出して、カスタム Java クラスを呼び出すことができます。Java クラスは、公開出荷通知 XML を拡大することができます。サービス定義フレームワークは、豊富な XML データをトランスポート・サービス層にトランスポートできます。

アクションをサービスに関連付けて、必要とされる特別な処理を実行できます。例えば、イベント PUBLISH_SHIP_ADVICE は、イベント・ハンドラーを呼び出して、カスタム Java クラスを呼び出すことができます。Java クラスは、公開出荷通知 XML を拡大することができます。サービス定義フレームワークは、豊富な XML データをトランスポート・サービス層にトランスポートできます。

アクションの構成およびイベントへのアクションの関連付け、またはサービスの構築に使用するさまざまなコンポーネントについて詳しくは、「Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales: *構成ガイド* (Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales: *Configuration Guide*)」を参照してください。

アクション (「その他 (Other)」タブで構成される) によって呼び出されるイベント・ハンドラーを以下に説明します。

**注:** 以下のイベント・ハンドラーは、後方互換性の目的のためだけに提供されています。

## E メール・メッセージ・イベント・ハンドラー

アプリケーションは、さまざまなイベントで E メール・メッセージを送信するための標準的な Send E-mail イベント・ハンドラーを提供します。例えば、ラインが出荷されたときに出荷情報を送信できます。SMTP ベースの E メールがこれらのメッセージ用にサポートされています。「アクション」の構成時に「その他 (Other)」タブで構成されている場合は、E メールが同期的に送信されます。メール・サーバーが使用不可の場合は、例外がスローされます。サービスの使用による E メール・メッセージの形式でのイベント通知の送信については、「Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales: *構成ガイド* (Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales: *Configuration Guide*)」を参照してください。

## E メール・テンプレート

E メール・テンプレートを使用すると、E メール・メッセージを書式設定できます。E メール・ノードは、XML データを受け取り、指定した XSL テンプレートとそれを組み合わせます。 Service Builder で E メール・ノードを構成して、E メール・テンプレートを使用できます。

## カスタム E メール・テンプレートの作成

### このタスクについて

カスタム E メール・テンプレートを作成して使用するには、以下の手順を実行します。

### 手順

1. *install_dir*/repository/xapi/template/merged/email/orders_mail.xsl テンプレート・ファイルをコピーします。
2. 必要に応じてファイルを変更し、名前変更し、*install_dir*/extensions/global/template/xsl/*CUSTOM-TEMPLATE-XSL* ディレクトリー内に保存します。別のディレクトリーに保存することもできますが、Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales によって供給される標準のディレクトリー構造を使用すると、整合性を保つために役立ちます。
3. Service Builder から、カスタム E メール・テンプレートを使用するように E メール・ノードを構成します。「本文テンプレート」フィールドで、/template/xsl/*CUSTOM-TEMPLATE-XSL* と指定します。

*install_dir*/repository/xapi/template/merged/email/*file_name*.mlt ファイルと「アクション」を使用して同じ目的を達成することは、リリース 5.0 以降では推奨されないことに注意してください。代わりに、E メール・ノードおよび XSL ファイルを使用してください。

## 例外アラート・イベント・ハンドラー

アプリケーションは、警告コンソールに例外アラートを送信する標準のイベント・ハンドラーをサポートしています。これらの例外は、特定のユーザー、またはユーザーのグループの場合には例外キューにリダイレクトされます。例えば、顧客サービス担当者が直接バイヤーに連絡を取れるように、承認に失敗した場合に、例外アラートを顧客サービス担当者キューに送信できます。警告コンソールについて詳しくは、「Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales : *ユーザー・ガイド* (Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales : *User Guide*)」を参照してください。

## 例外アラート・テンプレート

例外アラート・テンプレートを使用すると、発生したアラートに追加のテキストを供給できます。これにより、エラー・メッセージをより説明的な、理解しやすいものにできます。また、警告コンソールから解決画面へのハイパーリンクを供給する意味もあります。例えば、オーダー、出荷または集合・混載伝票種別用に作成されたどのようなアラートの場合も、ハイパーリンクが作成され、「警告リスト」画面の「作成対象」列に表示されます。例外アラート・テンプレートでは、このハイパーリンクをカスタマイズすることや、他のハイパーリンクを作成することができます。アラート・ノードへの入力データは、指定したテンプレートと組み合わされ、発生したアラートの説明として記入されます。

データ・バッファー形式でデータを公開するイベントは、ECT テンプレートを使用します。これは、タグを含むテキスト・ファイルです。これらのタグは、実行時に実際のデータに置き換えられます。すべてのタグは、|#YFS_*tagname*| 構文を使用します。

例えば、タグ |#YFS_OrderNo| を使用して、その場所に表示される実際のオーダー番号を持ちます (OrderNo がデータ・バッファーの一部として公開される場合)。データ・バッファー内で公開されるデータ要素のいずれも、テンプレート内で使用できます。

例外コンソールには、2 つのテンプレートがあります。1 つのテンプレートは、YFS_INBOX テーブルの DETAIL_DESCRIPTION フィールドを判別するために使用されます。もう 1 つのテンプレートは、YFS_INBOX テーブルの LIST_DESCRIPTION フィールドを判別するために使用されます。テンプレートは (イベントによって) 公開されたデータと組み合わされ、結果のストリングが対応するフィールドに読み込まれます。

## カスタム例外アラート・テンプレートの作成

### このタスクについて

カスタム例外テンプレートを作成して使用するには、以下の手順を実行します。

### 手順

1. *install_dir*/repository/xapi/template/merged/exception_console/example_exception_console.xsl テンプレート・ファイルをコピーします。
2. 必要に応じてファイルを変更し、名前変更し、*install_dir*/extensions/global/template/xsl/*CUSTOM-TEMPLATE-XSL* ディレクトリー内に保存します。別のディレクトリーに保存することもできますが、アプリケーションによって供給される標準のディレクトリー構造を使用すると、整合性を保つために役立ちます。
3. Service Builder から、カスタム例外コンソール・テンプレートを使用するようにアラート・ノードを構成します。「XSL テンプレート (XSL Template)」フィールドで、template/xsl/*CUSTOM-TEMPLATE-XSL* と指定します。

   **注:** *install_dir*/repository/xapi/template/merged/exception_console/*file_name*.ect ファイルと「アクション」を使用して同じ目的を達成することは、リリース 5.0 では推奨されないことに注意してください。代わりに、アラート・ノードおよび XSL ファイルを使用してください。

## 公開イベント・ハンドラー

アプリケーションは、相互運用性のための、イベント・マネージャーによる外部システムへのデータの公開を可能にします。ユーザーは、データを他のシステムに公開するように、任意のイベントを構成できます。例えば、SHIP_CONFIRM トランザクションのイベント ON_SUCCESS で、データを外部の金融システムに公開したい場合があります。

公開イベント・ハンドラーを構成する場合は、セミコロンで区切られたシステム ID のセットを提供する必要があります (例えば、SYSTEM1;System2;TestSystem)。これらは、データを読み取り、処理することが予期される、時間トリガー・トランザクションの ID です。イベント・マネージャーは、トランザクションによって提供されたデータを YFS_EXPORT テーブルに書き込み、構成されたシステム ID ごとに 1 つのレコードを書き込みます。

サービスを使用してデータを公開する場合は、データベース・トランスポート・ノードを使用します。詳しくは、「Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales : *構成ガイド* (Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales : *Configuration Guide*)」を参照してください。

## 実行イベント・ハンドラー

実行イベント・ハンドラーは、指定されたプログラムの起動をイベント・マネージャーに伝えます。アプリケーションは、このプログラムを PATH 内で探します。データは、最初のコマンド・ライン引数としてプログラムに渡されます。公開イベント・ハンドラーのように、実行イベント・ハンドラーは非同期的に呼び出されます。これは、サービス内のプログラムを実行するカスタム API を使用することで実現されます。

## イベント・ハンドラーとしての Java 拡張機能

Java 拡張機能を使用すると、Java クラスとしてイベント・ハンドラーを実装できます。各アクションに、1 つの Java イベント・ハンドラーを関連付けることができます。作成した Java クラスは、com.yantra.yfs.japi.util.YFSEventHandlerEx インターフェースを実装する必要があります。これは、サービス内で実際にプログラムを実行するカスタム API を使用することで実現されます。

## イベント・ハンドラーとしての Oracle ストアード・プロシージャー

Oracle ストアード・プロシージャーとしてイベント・ハンドラーを構成する場合、ストアード・プロシージャーは以下のパラメーターとともに呼び出されます。

```
PROCEDURE DO_ACTION(TRANID IN VARCHAR2,ACTIONCODE IN VARCHAR2,KEY_DATA IN
VARCHAR2,DATA_TYPE IN NUMBER,DATA_BUFFER1 IN VARCHAR2,DATA_BUFFER2 IN
VARCHAR2,DATA_BUFFER3 IN VARCHAR2,DATA_BUFFER4 IN VARCHAR2,DATA_BUFFER5 IN
VARCHAR2,DATA_COMPLETE IN NUMBER,SHIP_NODE IN VARCHAR2,RETURN_VALUE IN OUT NUMBER)
```

PL/SQL では、VARCHAR2 変数の最大サイズは 2,000 バイトに制限されます。そのため、2,000 バイトは、データ・バッファーに渡されるストリングの最大長となります。入力が 10,000 文字より大きい場合、バッファーは切り捨てられ、パラメーター DATA_COMPLETE の値は 0 (ゼロ) となります。バッファーが 10,000 バイトより小さく、ストアード・プロシージャーにすべて渡された場合、DATA_COMPLETE は 1 として渡されます。

**重要:** イベント・ハンドラーに関連付けられたストアード・プロシージャーでは、どのようなときでもストアード・プロシージャーでコミットまたはロールバックを行わないでください。

ストアード・プロシージャーの実行は、サポートされているサービス・コンポーネントではありません。

## イベント・ハンドラーとしての HTTP 拡張機能

HTTP 拡張機能としてイベント・ハンドラーを構成する場合、アプリケーションは post() 関数を使用して、指定された URL にデータを送信します。データは、以下の表の変数を使用してポストされます。

| 変数 | 説明 |
|---|---|
| sTranID | イベントを呼び出すトランザクションの ID |
| iDataType | iDataType が 1 に設定された場合、XML データがイベントとともに公開され、sData には XML ストリング全体が含まれます。それ以外の場合は、iData Type が 0 に設定され、sData には name=value のペアが含まれます。name=value ペアに何が含まれているかを確認するには、Javadocs でイベントに関連付けられた DBD ファイルを確認してください。 |
| sData | iDataType の説明を参照してください。 |
| sShipNode | 使用可能な場合は、出荷ノード ID |

サービスは、HTTP プロトコルを使用してデータをポストするように構成できます。詳しくは、「Sterling Business CenterSterling Selling and Fulfillment

FoundationSterling Field Sales *構成ガイド* (Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales *Configuration Guide*)」を参照してください。

## イベント・ハンドラーとしての COM 拡張機能

COM の使用には、サーバーおよびランタイム・クライアントの設定が必要です。

COM 拡張機能としてイベント・ハンドラーを定義した場合、COM オブジェクトとして指定した COM コンポーネントがアプリケーションによって呼び出されます。作成した COM コンポーネントでは、IDispatch インターフェース、およびそのインターフェースの一部として事前に識別された関数 doAction() を公開する必要があります。doAction() 関数のシグニチャーは、以下のようにする必要があります。

```
[IDL Syntax]
doAction ( [in] BSTR sTranID, [in] BSTR sActionCode, [in] BSTR sKeyData, [in]long
DataType, [in] BSTR sData, [in] BSTR ShipNode, [Out] long *Retval);
```

アプリケーションは IDispatch インターフェースを介してこの関数にアクセスするため、ユーザーは COM コンポーネントを Visual Basic または C++ プログラミング言語のどちらかで記述することができます。Visual Basic を介して公開する必要がある関数シグニチャーは、以下のとおりです。

```
Public Function doAction(ByVal bsTranID As String, ByVal boActionCode As String,
ByVal bsKeyData As String, ByVal DataType As Long, ByVal bsData As String, ByVal
ShipNode As String, retVal as Long) As Long
```

C++ を介して公開する必要がある関数シグニチャーは、以下のとおりです。

```
STDMETHODIMP COMActionImpl::doAction(BSTR sTranID, BSTR sActionCode, BSTR sKeyData,
long DataType, BSTR sData, BSTR Node, long *RetVal)
```

COMActionImpl は、doAction() 関数を実装するクラス名です。イベント・ハンドラーの構成時に、呼び出す COM コンポーネントのプログラム ID を指定する必要があります。

## イベント・チェーニングおよびイベント・ハンドラー

一部のイベント・ハンドラーは API の呼び出しをサポートしています。これらの API は、呼び出されているイベントに関連のあるデータをより多く取り出すことができます。または、ビジネスの計算やトランザクションを実行する指図を行う場合があります。同様に、トランザクションはその他のイベントを呼び出すことができます。したがって、イベント・チェーニングをもたらします。例えば、ORDER_CREATE イベントのイベント・ハンドラーの 1 つは、オーダーの妥当性を検査して、オーダーを HOLD するための API を呼び出すことができます。これが HOLD イベントのトリガーとなり、それにより、顧客サービス担当者へ E メール・メッセージが送信されます。イベント・チェーニングは、複雑なビジネス処理に必要な拡張性のメカニズムを提供します。

**重要:** ON_FAILURE イベントに呼び出されるように構成されるアクションは、どのようなデータベース更新も実行しないようにしてください。ON_FAILURE イベントに呼び出されたアクションによって実行されたデータベース更新は、ロールバックされます。

Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales によって公開されたデータにアクセスするには、YFSEventHandlerEx() インターフェースを実装する必要があります。このインターフェースは、handleEvent() 関数を提供します。handleEvent() 関数は、以下のように実装します。

```
public boolean handleEvent (YFSEnvironment oEnv, String sTranID, String
 sActionCode, Map sKeyData, int iDataType, Object sData, String sShipNode,
 String [] parms) throws Exception
```

**重要:** アプリケーション内でアクションに関連付けられたイベントを引き起こして同じ API を呼び出す API を実装することは可能ですが、このシチュエーションが生じないよう注意してください。それは無限ループを作成します。

クラス名は、Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales トランザクション構成の「アクション構成 (Action Configuration)」画面上の Java オブジェクトとして構成される必要があります。詳しくは、「Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales *構成ガイド* (Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales *Configuration Guide*)」を参照してください。

handleEvent() 関数のすべてのパラメーターは入力パラメーターであり、アプリケーションによって挿入された値を持ちます。この関数の最後のパラメーターは、parms パラメーターであり、ストリング定数の配列です。これらのストリング定数の値は、アプリケーションでのアクション構成時にクラス名の後に指定されたストリング定数の値です。

**注:** YFSEventHandlerEx インターフェースについて詳しくは、Javadocs を参照してください。

# 第 3 章 構成の変数の使用

## 動的構成のための変数の使用

実装にかかる時間を削減するために、アクション、エージェントおよびサービス構成に属する特定のプロパティーを動的に構成できます。例えば、プロバイダー URL、E メール・サーバーおよび送信者アドレスなどのネットワーク・プロパティーを構成する動的な方法を提供できます。これらのプロパティーは、1 つの場所で変数として構成でき、これらのプロパティーが使用されるときには実行時に解決できます。

ファイル名またはパスをコンフィギュレーターアプリケーション・マネージャーに入力できる場所ならどこでも、ファイルまたはディレクトリー名の代わりに変数を提供できます。この変数置換は、*install_dir*/properties/customer_overrides.properties ファイル内のエントリーに基づいており、実行時に解決されます。

コンフィギュレーターアプリケーション・マネージャーの以下のコンポーネント内で変数名を使用できます。

- サービス定義フレームワーク
  - 「JMS キュー名」、「初期コンテキスト・ファクトリー」、「QCF ルックアップ」および「プロバイダー URL」などのすべてのトランスポート・タイプ。ファイル送信側および受信側、送信側および受信側ディレクトリー用の FTP 送信元および宛先、および HTTP および Web サービス・トランスポート・タイプ。
  - 指定できる E メール・コンポーネント内、E メール・サーバー、件名、リスナー・ポート、および From アドレス。動的構成は、E メール・プロトコルを指定するためにも使用できます。
  - アクション
    - 「HTTP 拡張機能の呼び出し」および「プログラムの実行 (Execute Program)」内。
  - エージェント条件の詳細
    - 「JMS キュー名」、「初期コンテキスト・ファクトリー」、「QCF ルックアップ」および「プロバイダー URL」。
- 「プリンター」、「ドキュメントの印刷」および「コンポーネントの印刷 (Print Components)」
- 「パージ条件」のログ・ファイル・タイプ
- システム管理コンソール内:
  - E メール・サーバー名、サーバー IP アドレス、サーバー・リスナー・ポートおよび E メールのプロトコルなど、インストール・ルールを指定するために変数を使用できます。
  - JMS モニター構成フィールドのために変数を提供できます。それらのフィールドには、WebLogic プロバイダー URL、WebSphere チャネル名、ホスト名、ポート番号およびキュー・マネージャー名が含まれます。

上記のフィールドについては、*install_dir*/properties/customer_overrides.properties ファイル内の yfs.VARIABLE_NAME として値を構成できます。それはそのままデータベース内に保管されます。また、変数が使用されるときは実行時に、値を復号するために customer_overrides.properties ファイル内で検索が実行されます。これらの変数の値は customer_overrides.properties ファイルから取り出されるため、それらは特定の JVM に固有のものとなります。

**注:** この変数の値は、デプロイされる JVM に依存するため、ヘルス・モニターのエージェントの詳細には表示できません。変数の値を表示するには、モニター・エージェントのサーバーの詳細をクリックする必要があります。

例えば、「ファイル入出力受信者」のディレクトリー構造を共通変数 (${ffbase} など) に設定したい場合、入力ディレクトリーは以下のように設定する必要があります。

```
${ffbase}/incoming
```

コンフィギュレーターアプリケーション・マネージャーで定義した変数 ${ffbase} の値は、以下のように yfs の接頭辞を持つ customer_overrides.properties ファイルで定義される必要があります。

```
yfs.ffbase=C:/FileIODir/Receiver
```

この ${ffbase}/入力値はデータベースに保管され、変数は、ファイル・アダプターを処理するときに C:/FileIODir/Receiver/incoming に解決されます。

以下の条件は、この変数の使用法のための前提となります。

- 参照されるときは、すべての変数は以下の形式である必要があります。
  - ${variable_name}
- すべての変数は正しく形成される必要があります。変数が見つからない場合の代替はありません。
- 変数には、'}' 文字は含まないでください。
- 変数の先頭または末尾に空白文字を使用しないでください。
- ファイル名のための変数は常にクラスパス内で解決されるため、テンプレートはそれらをサポートしていません。

# 第 4 章 カスタム・トリガー時刻トランザクションの作成

## カスタム時間トリガー・トランザクションについて

アプリケーションには、独自の時間トリガー・トランザクションを記述できるインフラストラクチャーが用意されています。ユーザーは、標準の時間トリガー・トランザクションを呼び出してスケジュールに入れるのとほぼ同じ方法で、これらの時間トリガー・トランザクションを呼び出してスケジュールに入れます。

標準の時間トリガー・トランザクションの構成方法について詳しくは、「Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales *構成ガイド* (Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales *Configuration Guide*)」を参照してください。

処理するタスクのリスト (それらの作業負荷) を時間トリガー・トランザクションが判断する方法によって、時間トリガー・トランザクションを以下のカテゴリーのいずれかに分類できます。

- 非タスクベース (汎用) - これらの時間トリガー・トランザクションは、カスタム・ロジックを使用して、実行する必要がある作業を決定します。一元的なタスク・キューを使用する場合も使用しない場合もあります。
- タスクベース (固有) - これらの時間トリガー・トランザクションは、タスク・キューを使用して、それらの作業を決定します。

アプリケーションには、両方のタイプのカスタム時間トリガー・トランザクションを作成するためのインフラストラクチャーが用意されています。

非タスクベースの時間トリガー・トランザクションを記述する能力は、com.yantra.ycp.japi.util.YCPBaseAgent クラスを使用して提供されます。このクラスは、タスク・キューが使用されているかどうかを問わず、汎用インフラストラクチャーを提供します。したがって、どのような時間トリガー・トランザクションの記述にも使用できます。

タスクベースの時間トリガー・トランザクションは、com.yantra.ycp.japi.util.YCPBaseTaskAgent クラスをサブクラス化することでプログラムできます。

トランザクションがタスク・キュー・ベースの場合は、タスクベースのトランザクションを記述するように特別に提供されたインフラストラクチャーを使用することをお勧めします。このインフラストラクチャーは、タスク・キューからカスタム・エージェントのための作業を自動的に決定します。したがって、トランザクションのために必要な設計および開発の量が削減されます。

アプリケーション仕様書に記述されたすべてのカスタム・エージェントは、2 つの抽象関数がある com.yantra.ycp.japi.util.YCPBaseAgent クラスからサブクラス化されます。これらの関数を実装すると、それらが時間トリガー・トランザクションの処理機能を提供します。

## getJobs() 抽象関数

getJobs() 抽象関数は以下を使用します。

- Env。API に渡すことができる、事前に作成済みの、YFSEnvironment オブジェクトのインスタンス
- inXML。入力 XML を含む org.w3c.dom.Document オブジェクト

この関数の実装は、実行する必要があるジョブを取得し (データベースから)、org.w3c.dom.Document オブジェクトのリストを構成し、そのリストを返す必要があります。以下のシグニチャーを参照してください。

```
public List getJobs(YFSEnvironment Env, Document inXML)
```

## executeJobs() 抽象関数

getJobs() 関数によって返されたリスト内の各ドキュメントは、実行のためにこの関数に渡されます。この関数が例外をスローする場合、例外がログに記録され、トランザクションがロールバックされます。それ以外の場合は、トランザクションはコミットされます。以下のシグニチャーを参照してください。

```
public Document executeJob(YFSEnvironment Env, Document inXML)
```

executeJob() 関数によってすべてのドキュメントが処理されたら、executeJob() 関数が処理する必要がある次の一連のタスクを取得するために、アプリケーションが再度 getJobs() 関数を呼び出します。これは、getJobs() 関数によって返されるジョブがなくなるまで繰り返されます。

入力 XML の例としては、Javadocs で YCPBaseTaskAgent クラスを参照してください。

また、com.yantra.ycp.japi.util.YCPBaseAgent も、以下のシグニチャーを持つトレース・ロギングおよびタイマー情報のためのユーティリティー関数を提供します。

- public void log(String classMethodName, String message);
- public startTimer(String timerName);
- public endTimer(String timerName);

# 非タスクベースのカスタム時間トリガー・トランザクションの作成

### このタスクについて

非タスクベースのカスタム時間トリガー・トランザクションは、com.yantra.ycp.japi.util.YCPBaseAgent クラスのサブクラスとして記述される必要があります。カスタム時間トリガー・トランザクションの例として、Javadocs で com.yantra.ycp.japi.util.YCPBaseAgent クラスを参照してください。

非タスクベースのカスタム時間トリガー・トランザクションを記述するには、以下の手順を実行してください。

### 手順

1. com.yantra.ycp.japi.util.YCPBaseAgent をサブクラス化します。
2. このクラスに executeJob() 関数および getJobs() 関数を実装します。

3. アプリケーション・マネージャーから時間トリガー・トランザクションを構成し、それにエージェント・サーバーを割り当てます。
4. 「Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales: *インストール・ガイド* (Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales: *Installation Guide*)」の説明に従って、カスタム時間トリガー・トランザクションをスケジュールに入れ、実行します。

## タスクベースのカスタム時間トリガー・トランザクションの作成について

タスクベースのカスタム時間トリガー・トランザクションは、com.yantra.ycp.japi.util.YCPBaseTaskAgent クラスのサブクラスとして記述されます。com.yantra.ycp.japi.util.YCPBaseTaskAgent クラスは、getJobs() 関数および executeJob() 関数が既に実装されている com.yantra.ycp.japi.util.YCPBaseAgent のサブクラスです。このクラスをサブクラス化することによるタスクベースのカスタム時間トリガー・トランザクションの作成には、入力として渡される 1 つのタスク・キュー・レコードを処理するための、executeTask() 関数の実装が含まれます。

**注:** executeTask() 関数が例外をスローする場合、例外がログに記録され、トランザクションがロールバックされます。それ以外の場合は、トランザクションはコミットされます。

使用可能なロギングおよびタイミング・ユーティリティー関数は、com.yantra.ycp.japi.util.YCPBaseAgent クラスによって提供される関数と似ています。executeTask() 関数のシグニチャーは public Document executeTask(YFSEnvironment oEnv, Document inXML) です。Env は、事前に作成済みの、API に渡すことができる YFSEnvironment オブジェクトのインスタンスであり、InXML は、カスタム・タスク XML が含まれる org.w3c.dom.Document オブジェクトです。カスタム・タスク XML には、TransactionFilters Node も含まれます。TransactionFilters Node には、タスクベースのカスタム時間トリガー・トランザクションに渡されるすべてのパラメーターが含まれます。このノードは、入力 XML のルート・ノードの下にあります。例えば、以下のタスクベースのカスタム時間トリガー・トランザクションの例を参照してください。

```
<?xml version="1.0" encoding="UTF-8"?>
<TaskQueue  TaskQKey="" TransactionKey="" DataKey="" DataType="" AvailableDate=""
     Lockid="" Createts="" Createprogid="" Createuserid=""
     Modifyts=""Modifyprogid="" Modifyuserid="" >
                 <TransactionFilters   AgentName=""  TransactionKey=""
               CurrentThread="" NumRecordsToBuffer="" TotalThreads=""/>
</TaskQueue>
```

タスクベースのカスタム時間トリガー・トランザクションの例として、Javadocs で com.yantra.ycp.japi.util.YCPBaseTaskAgent クラスを参照してください。

## タスクベースのカスタム時間トリガー・トランザクションの作成

### このタスクについて

タスクベースのカスタム時間トリガー・トランザクションを記述するには、以下の手順を実行してください。

## 手順

1. com.yantra.ycp.japi.util.YCPBaseTaskAgent をサブクラス化します。
2. このクラスに executeTask() 関数を実装します。
3. アプリケーション・マネージャーから時間トリガー・トランザクションを構成し、それにエージェント・サーバーを割り当てます。
4. 「Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales: *インストール・ガイド* (Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales: *Installation Guide*)」の説明に従って、カスタム時間トリガー・トランザクションをスケジュールに入れ、実行します。

# 第 5 章 外部トランザクション

## 外部トランザクションとの調整

アプリケーションは、イベントの呼び出し、外部プログラムの呼び出し、またはカスタム API やサービスの呼び出しによって、外部コードを実行する機能を提供します。これらの呼び出しの間に、外部システムはトランザクションを開始できます。外部トランザクションは Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales トランザクションの一部ではないため、Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales トランザクションがロールバックされたが外部トランザクションはされなかった場合に、データの不整合を引き起こす可能性があります。

外部トランザクションの調整によって、外部システムは、それらのトランザクションを Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales に登録します。 Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales がそのトランザクションをコミットする準備ができると、YFSTxnCoordinatorUE 外部プログラムが呼び出されます。その外部プログラムによって、アプリケーションが、外部トランザクションのコミットおよびロールバックを処理します。

## 外部トランザクション調整の実装

### このタスクについて

外部トランザクション調整を実装するには、以下の手順を実行します。

### 手順

1. 外部トランザクション処理を実装したいカスタム API を作成しますが、外部トランザクションはコミットしないでください。代わりに、(YFSEnvironment) env.setTxnObject(String ID, Object txnObj) を呼び出すことで、YFSEnvironment とともに外部トランザクション・オブジェクトを登録します。

   String ID は、カスタム・トランザクション・オブジェクトを識別するために外部プログラム実装クラスによって使用される固有の名前です。

   以下の例は、単純なカスタム API を示しています。

```
public class doSomethingAPI
{
private YFCLogCategory cat = YFCLogCategory.instance("DoSomethingAPI");
public doSomethingAPI()  // constructor is empty
{
}
public void writeToDB (String key, YFSEnvironment oEnv)
{
try
{
Driver aDriver =
(Driver)Class.forName("oracle.jdbc.OracleDriver").newInstance();
        String url = "jdbc:oracle:thin:@127.0.0.1:1521:qotree2";
```

```
        Connection conn = DriverManager.getConnection(url, "Scott", "Tiger");
conn.setAutoCommit(false);
       String sql = "insert into TxnTest (key) values ('" + key + "')";
       Statement stmt = conn.createStatement();
       stmt.executeUpdate(sql);
       oEnv.setTxnObject("YDBconn", conn);
}
catch (Exception e)
{
System.out.println ("Caught Exception :\n" + e);
}
   }
public Document doSomething(YFSEnvironment env, Document doc) throws Exception
{
      System.out.println("Executing doSomething method..........");
writeToDB ("doSomething", env);
return doc;
   }
}
```

2. com.yantra.yfs.japi.ue.YFSTTxnCoordinatorUE 外部プログラム・インターフェースを実装し、Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales が外部トランザクションのコミットを実行する前または後に、それをコミットします。Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales トランザクションがロールバックを起動すると afterYantraTxnRollback(YFSEnvironment oEnv) メソッドが呼び出されて外部トランザクションをロールバックするように、ロールバック・メソッドを実装します。これを実現するには、以下のメソッドを実装します。

   - beforeYantraTxnCommit(YFSEnvironment oEnv)
   - afterYantraTxnCommit(YFSEnvironment oEnv)
   - afterYantraTxnRollback(YFSEnvironment oEnv)

   **注:** 統合要件に応じて、コミットを同期化するために beforeYantraTxnCommit または afterYantraTxnCommit 外部プログラムのどちらかを使用できます。

   (YFSEnvironment)env.getTxnObject(ID) を呼び出すと、これらのメソッドが、以前に (YFSEnvironment)env.setTxnObject(String ID, Object txnObj) によって登録された外部トランザクション・オブジェクトへのハンドルを取得できます。getTxnObject 呼び出しおよび setTxnObject 呼び出しの両方で、ID が同じであることに注意してください。

   以下は、YFSTTxnCoordinatorUE 外部プログラム・インターフェースの実装の例です。

```
public class afterTxnCommit implements YFSTxnCoordinatorUE
{

public void beforeYantraTxnCommit (YFSEnvironment oEnv) throws YFSUserExitException
{
// before method is not implemented because after method is implemented.
}
public void afterYantraTxnCommit (YFSEnvironment oEnv) throws YFSUserExitException
{
System.out.println ("Entering method afterYantraTxnCommit.........");
try
{
   Connection ydbConn = (Connection)oEnv.getTxnObject("YDBconn");
   ydbConn.commit();
}
catch (Exception e)
{
```

```
System.out.println ("Caught Exception :\n" + e);
}
}
public void afterYantraTxnRollback (YFSEnvironment oEnv) throws
YFSUserExitException
{
System.out.println ("Entering method afterYantraTxnRollback.........");
try
{
   Connection ydbConn = (Connection)oEnv.getTxnObject("YDBconn");
   ydbConn.rollback();
}
catch (Exception e)
{
System.out.println ("Caught Exception :\n" + e);
}
}
```

3. コンフィギュレーターアプリケーション・マネージャーを起動して、「**システム管理**」>「**外部プログラム管理**」に移動し、YFSTxnCoordinatorUE 実装クラスを構成します。

   YFSTxnCoordinatorUE 外部プログラム・インターフェース定義について詳しくは、Javadocs を参照してください。

# 第 6 章 トランザクション・データ・セキュリティー

## トランザクション・データを保護する方法

セキュリティー問題は、どのユーザーがどのデータへのアクセス権限を持つか、それらのユーザーがどのくらい表示できるか、およびそれらのユーザーがそのデータで何をできるかの制御を伴います。アプリケーションは、以下のセキュリティー問題に対処するメカニズムを提供します。

- ユーザー・アクセス制御
- シングル・サインオン
- データ暗号化

### ユーザー・アクセス制御

ユーザー・グループ・アクセス権、ログイン、およびオーダー変更アクセス権のアクセス制御メカニズムによって、アプリケーションは、顧客サービスおよび管理組織の複数レベルのためのセキュリティー手段を提供します。

API は、システム機能の別の領域へのアクセス権限を制御します。呼び出す側のユーザーが呼び出される機能に対するアクセス権限を持つことを確認することは、呼び出し元の責任です。アプリケーションは、この取り組みを支援するためにセキュリティー・マネージャー API を提供します。Javadocs の com.yantra.api.ycp.security パッケージを参照してください。

### シングル・サインオン

アプリケーション内でシングル・サインオンのメカニズムが必要な場合は、インターフェース com.yantra.ycp.japi.util.YCPSSOManager を実装します。このインターフェースには、String(UserId) を返す getUserData() 関数があります。詳しくは、Javadocs を参照してください。

### データ暗号化

暗号化は、許可されていないユーザーによって機密データが表示されないようにします。アプリケーションは、ユーザー名、パスワードおよびクレジット・カード番号などのデータを暗号化できるようにする API を提供します。

また、暗号化および暗号化解除は、アプリケーション・マネージャー内で指定された後にのみ適用されます。例えば、クレジット・カード情報を渡された外部プログラムのみが、暗号化解除されたクレジット・カード番号にアクセスできます。

## 暗号化ロジック

アプリケーションは、暗号化ロジックを処理するために com.yantra.ycp.japi.util.YCPEncrypter インターフェースを公開します。すべてのアプリケーション暗号化および暗号化解除は、このインターフェースを実装するエンクリプター・クラスによって処理されます。このクラスは、*install_dir*/properties/customer_overrides.properties ファイル内の以下のプロパティーを構成することで指定されます。

- yfs.encrypter.class
- yfs.propertyencrypter.class properties

両方のクラスは、com.yantra.ycp.japi.util.YCPEncrypter インターフェースを実装する必要があります。

com.yantra.ycp.japi.util.YCPEncrypter インターフェースには、以下の 2 つの関数があります。

- public java.lang.String encrypt(java.lang.String sData) - sData は、暗号化のために実装するクラスに、アプリケーションによって渡されるデータです。戻り値は暗号化されたストリングです。
- public java.lang.String decrypt(java.lang.String sData) - sData は、暗号化解除に必要なデータです。

独自のプロパティー・エンクリプター・クラスの記述について詳しくは、Javadocs で YCPEncrypter インターフェースを参照してください。

このインターフェース内の暗号化関数および暗号化解除関数は、Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales によって複数回呼び出されます。アプリケーションは、平文と暗号化された情報を区別しません。そのため、暗号化関数は、以前に暗号化されたデータとともに呼び出される可能性があります。二重暗号化を防ぐために、暗号化関数には、平文と以前に暗号化された情報を区別できることが重要です。以前に暗号化された情報が関数に渡された場合、この関数の実装は、その情報を再度暗号化することなく、関数に渡されたままの情報を返す必要があります。

また、暗号化解除関数も、平文と以前に暗号化されたテキストを区別できる必要があります。

### 暗号化および暗号化解除の無効化

暗号化 (または暗号化解除) を無効にするには、どのような処理も行うことなく、入力として渡された値と同じ値を返すように、暗号化 (または暗号化解除) 関数を実装します。

## 暗号化および暗号化解除戦略の選択

暗号化戦略の選択時には、複数のデプロイメント・オプションがあります。最も一般的なオプションは、次のとおりです。

- 暗号化なしまたは暗号化解除なし
- 外部トークン化

- 暗号化および暗号化解除の両方
- 暗号化解除なし

**注:** 暗号化戦略として外部トークン化を使用することをお勧めします。

意思決定処理をガイドするために以下の説明を使用します。

## 暗号化および暗号化解除を使用しない

データベース内に平文のクレジット・カード番号を保存しないでください。ただし、次の場合は、Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales 内で暗号化を構成できる可能性があります。

- ビジネスでクレジット・カード番号またはその他の機密情報の受け入れ、処理、または保管がない。
- すべての暗号化および暗号化解除が外部で処理される。アプリケーションは、外部で暗号化されたクレジット・カード番号を他のシステムに渡します。暗号化を有効にした場合、暗号化されたストリングは再度暗号化されます。

**注:** 暗号化および暗号化解除が実装されていない場合であっても、暗号化された PAN の保管により、アプリケーションが PCI DSS および PA-DSS 監査スコープに置かれます。

## 外部トークン化

Sterling Sensitive Data Capture Server アプリケーションは、クレジット・カード番号およびストア・バリュー・カード番号を収集し、トークン化します。 PCI DSS および PA-DSS の要件を満たすためのアプローチとして、「Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales: セキュア・デプロイメント・ガイド (Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales: *Secure Deployment Guide*)」をレビューすることをお勧めします。

## 暗号化および暗号化解除の両方の使用

アプリケーションは、必要に応じて自動でクレジット・カード番号を暗号化および暗号化解除します。ただし、この戦略は推奨される暗号化戦略ではありません。

## 暗号化を使用するが暗号化解除はなし

ビジネスで Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales にクレジット・カード番号を保管させたいが、どのような事情であっても Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales に自動でそれらを暗号化解除させたくない場合、暗号化関数のみを使用可能にし、暗号化解除関数を使用不可にしたいことがあります。

このように、Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales は平文として渡されたクレジット・カード番号を暗号化するが、それらを変換して返すことは決してありません。Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales が情報を暗号化すると、すべてのカスタム拡張機能は、暗号化されたクレジット・カード番号として渡されます。また、外部で暗号化解除を処理する必要があります。Sterling

Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales 内の少数の外部プログラム (例えば、YFSbeforeCreateOrderUE) はクレジット・カード番号が暗号化される前に呼び出されるため、それらの外部プログラムが平文の番号にまだアクセスできることに注意することが重要です。

## 暗号化をサポートする方法

アプリケーションは、以下の場所のための暗号化をサポートしています。

- yfs.properties、yifclient.properties および management.properties ファイル内で指定されたプロパティー
- クレジット・カード番号

### yfs.properties による暗号化

JDBC URL、データベース・ユーザー ID およびパスワードなどのプロパティーは、暗号化して、customer_overrides.properties ファイルに保管できます。アプリケーションはデータベースへの接続にこの情報が必要なため、これらの値は、アプリケーションによって暗号化解除される必要があります。アプリケーションに絶対にデータを暗号化解除させたくない場合、これらのプロパティーは、暗号化および保管できません。

**注:** yfs.properties ファイル内で指定されたプロパティーのいずれかを設定したい場合、その特定のプロパティーのためのエントリーを *install_dir*/properties/customer_overrides.properties ファイルに追加します。

### クレジット・カード番号のための暗号化

アプリケーションは、データベース内に保管する前にクレジット・カード番号を暗号化できます。yfs.properties ファイル内で指定されたプロパティーと異なり、デフォルト処理では、暗号化解除されたクレジット・カード番号がアプリケーションによって必要とされることはありません。ただし、課金またはユーザー・プリファレンスの保管のために、暗号化解除されたクレジット・カード番号を必要とする外部プログラムを実装することで、アプリケーションを拡張する場合があります。アプリケーションに自動で情報を暗号化解除させたくない場合は、外部プログラムの実装において、これらのクレジット・カード番号を暗号化解除する必要があります。

## クレジット・カード番号の暗号化

暗号化を実装している場合、アプリケーションは以下のシチュエーションでクレジット・カード番号を暗号化します。

- API によって返されたとき
- イベント・データの一部として公開されたとき
- データベース内のどこかに保管されたとき
- ユーザー・インターフェース上に表示されたとき (しかし、ユーザー・インターフェースには、ユーザー・アクセスに基づくこの動作をオーバーライドするオプションがある場合がある)

### API によるクレジット・カード番号の暗号化

**注:** システムに入力する支払い情報は、暗号化されるのではなく既にトークン化されていることが推奨されます。

Encrypter クラスによって暗号化アルゴリズムおよび暗号化解除アルゴリズムの両方が実装されていると仮定すると、アプリケーションは、クレジット・カード番号を暗号化および暗号化解除できるようにする以下の API を提供します。

- getEncryptedString() - 渡されるストリングを受け入れ、暗号化されたストリングを返す
- getDecryptedString() - 暗号化されたストリングを受け入れ、暗号化解除されたストリングを返す
- getEncryptedCreditCardNumber() - 暗号化されたクレジット・カード番号を返す

  **注:** getEncryptedCreditCardNumber() は、リリース 9.0 では推奨されていません。getEncryptedString() に置き換えられました。
- getDecryptedCreditCardNumber() - getEncryptedCreditCardNumber() API を使用して暗号化されたクレジット・カード番号を返す

**注:** getDecryptedCreditCardNumber() は、リリース 9.0 では推奨されていません。getDecryptedString() に置き換えられました。

### 外部プログラムによるクレジット・カード番号の暗号化

クレジット・カード情報を渡された外部プログラムのみが、暗号化解除されたクレジット・カード番号にアクセスできます。アプリケーションは、クレジット・カード・データを渡すための以下の外部プログラムを提供します。

- YFSCollectionCreditCardUE
- YFSCollectionCustomerAccountUE
- YFSCollectionOthersUE
- YFSCollectionStoredValueCardUE
- YFSValidateInvokedCollectionUE

**注:** YFSBeforeCreateOrder および YFSBeforeChangeOrder などのいくつかの外部プログラムは、システム内で暗号化される前のクレジット・カード番号にアクセスできる場合があります。

これらの外部プログラムについて詳しくは、Javadocs を参照してください。

## プロパティー・ファイルによる暗号化

### このタスクについて

いくつかのプロパティーは、ユーザーが暗号化したい可能性があるユーザー ID およびパスワードなどの機密データをリレーします。以下のファイル内で示されるどのプロパティー (yfs.properties ファイル内の yfs.propertyencrypter.class プロパティー以外) も、*install_dir*/properties/customer_overrides.properties ファイルを使用して暗号化できます。

- *install_dir*/properties/yfs.properties

- *install_dir*/resources/yifclient.properties ファイル

プロパティーを暗号化するには、以下の手順を実行します。

## 手順

1. 暗号化したいプロパティー値に接頭辞として `encrypted:` を付けます。例:

   ```
   yfs.dblogin.datasource.name=encrypted:encrypted value
   ```

2. security.propertyencrypter.class プロパティーに CLASSPATH 環境変数からアクセス可能なことを確認します。
3. YCPEncrypter インターフェースを実装します。このインターフェースについて詳しくは、Javadocs を参照してください。

   `encrypted:` が先頭に付くこれらのプロパティーは、実行時に自動的に暗号化解除されます。

# 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: 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