Sterling Selling and Fulfillment Foundation

API のカスタマイズ

バージョン 9.1

Sterling Selling and Fulfillment Foundation

API のカスタマイズ

バージョン 9.1

お願い

本書および本書で紹介する製品をご使用になる前に、51 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
Customizing APIs
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 サービスの拡張

サービスの拡張について

Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales の用語では、サービスは、ステートレスのコア・ビジネス・ロジック・コンポーネントで、プレゼンテーション・ロジックを含みません。各サービス (アプリケーションによってすぐに使用可能な状態で提供されるか、または Service Definition Framework を使用してカスタム作成される) は、データの一貫性を失うことなく、1 つのトランザクション境界内で独立して実行できる処理の論理ユニットを表します。Service Definition Framework を使用して、1 つ以上のサービスをより大きい複合サービスに集約して、別のサービスを作成するために使用できます。これは、相互にリンクして複雑なビジネス処理を提供できる小さい再使用可能なコンポーネントを作成する方法を提供します。

Service Definition Framework 内のすべてのサービスは、内部アプリケーション・ビジネス・プロセスまたは外部システムのいずれかを介して、双方向に呼び出すことができます。Service Definition Framework にデプロイされるサービスは、ステートレスで、それぞれに独自のトランザクション・コミットメント境界があります。

サービスは、アクションを使用してイベントに関連付けることによって、アプリケーションで呼び出すことができます。標準の相互運用性イベント・ハンドラーを使用するか、または独自のカスタム・イベント・ハンドラーを実装できます。次に、アプリケーションを構成して、特定のイベントが発生し、条件が満たされたときにイベント・ハンドラーを呼び出すことができます。イベント、条件、およびアクションの構成について詳しくは、Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales: *構成ガイド 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。

アプリケーションは、ビジネス・ロジックを拡張するいくつかの外部プログラムを提供します。トランザクション内から呼び出された外部プログラムは、トランザクションを構成するときにサービスに関連付けることができます。テンプレートは外部プログラムに対してはサポートされていないことに注意してください。外部プログラムの構成について詳しくは、Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales: *構成ガイド 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。

サービスが構成されたら、クライアントによってプログラムでも呼び出すこともできます。

サービス呼び出し構成は、以下のシチュエーションで説明するように Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales のインストール場所に対する、サービスを呼び出すクライアントの場所によって異なります。

- 呼び出すクライアントに Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales がインストールされて**いない**場合 - リモート呼び出しに対して構成します。
- 呼び出すクライアントに Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales がインストールされて**いる**場合 - ローカル呼び出しに対して構成します。

サービスの同期呼び出しまたは非同期呼び出し

呼び出しのモードに応じて、サービスを 2 つの主要なカテゴリーに分類できます。

- 同期呼び出しサービス (オンデマンド) - これらのサービスは、1 回の呼び出しですべての処理を実行し、結果を返すことができます。
- 非同期呼び出しサービス (メッセージ駆動)

同期呼び出しサービス

これらのサービスは、オンデマンドで、1 回の呼び出しですべての処理を実行し、結果を返すことができます。

非同期呼び出しサービス

これらのサービスは、外部システムまたは Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales 内からのメッセージによってトリガーされたときには必ずすべての処理を自動的に実行します。トリガーは、統合のモードに応じて、ファイル、データベース・レコードまたは、メッセージ・キューのメッセージの形式を取ることができます。これらのサービスは、値をまったく返さず、E メールの送信、外部システムからの更新の自動受信、または外部システムへの更新の自動送信などのバックグラウンド・プロセスにのみ使用されます。

一般的に、非同期サービスは、パフォーマンス率に対するコストが同期サービスよりの安価で、これを使用できる場合は優先して使用する必要があります。ただし、非同期サービスは、キューに入り、受信した順にメッセージを処理します。特定のトランザクションがキューに入ってから処理されるまでの時間は、プロセス・サイクルのピークおよびその他の多くの要因によって大きく異なります。したがって、非同期サービスは、トランザクションを指定の時間内に処理する必要があることを SLA (サービス品質保証) によって要求されているような特定にシナリオには不適です。ただし、このようなシナリオは、ほとんどのビジネスおよびビジネス・プロセスではまれであり、非同期処理は、大多数のトランザクションを非常に低コストで実行し、なおかつ高いサービス品質を提供するためには十分に効率的です。

サービスで使用するビジネス関数

通常、サービスは、1 つ以上のメッセージング・コンポーネント (つまり、サービスからまたはサービスへのメッセージの処理方法を定義するコンポーネント)、1 つ以上のユーティリティー・コンポーネント (E メールやアラートのハンドラーなど) および 1 つ以上のビジネス処理コンポーネントから構成されます。アプリケーションで定義されるサービスに使用できるユーティリティー・コンポーネントおよびメッセージング・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 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。このセクションでは、サービスの使用方法、ビジネス処理コンポーネントのカスタマイズならびに拡張方法、およびサービスでこれらを使用可能にする方法について説明します。

アプリケーションには、広範ですぐに使用可能なビジネス関数ライブラリーが付属しています。このライブラリーの各関数は、標準 API と呼ばれます。標準 API それぞれの入力、出力、および動作について詳しくは、Javadocs を参照してください。

独自のビジネス関数を作成し、サービスで使用することも可能です。そのような関数は、拡張 API と呼ばれます。

標準 API を集約および相互にリンクすることによって、より複雑なサービスを形成できますが、ほとんどの場合、API は、ビジネス・トランザクションで必要なすべての機能を提供するため、他のコンポーネントや API とリンクする必要はありません。この最も一般的なシナリオの作業を容易にするために、すべての標準 API は、自動的に同期呼び出しできます。Service Definition Framework を使用して、サービスとしてそれぞれの API をモデル化する必要はありません。これらの API を「自動定義」同期サービスと見なすことができます。ただし、拡張 API およびこれらの API の非同期呼び出しでは、まず Service Definition Framework を使用して、これらをサービスとして明示的にモデル化する必要があります。

非同期サービスのメッセージ・サイズ

データベース表が最大サイズに到達し、send() 関数がテーブルにメッセージを挿入しようとすると、Service Definition Framework は例外をスローします。以下のテーブルに記されているサイズ制限を参照してください。

メカニズム	データ・タイプ	サイズ (Size)
データベース (Oracle)	CLOB	4 GB
データベース (MicrosoftSQL Server)	テキスト	4 GB
JMS Queue	テキスト・メッセージ	4 GB
MSMQ Queue	メッセージ	2 GB

例外処理およびサービス

警告コンソールは、Service Definition Framework によってログに記録されたすべての例外を表示します。また、非同期に構成されたトランザクションで発生した例外を再処理することもできます。データベースまたはキューを使用している場合、呼び出しは非同期です。

Service Definition Framework は、例外情報のロギングに log4j ユーティリティーを使用します。log4j ユーティリティーは、ログ・ファイルにトレース情報とデバッグ情報の両方を書き込みます。ロガーを構成して、さまざまなカテゴリーのメッセージをさまざまな宛先に送信できます。カテゴリーは、階層的に編成され、継承できます。各カテゴリーは、重大度レベルを示す優先順位で構成できます。カテゴリーに優先順位が構成されていない場合、優先順位が割り当てられている最も近い祖先の優先順位を継承します。

API 呼び出し中またはイベント・ハンドラー使用中に発生したすべての例外がログに記録されます。

第 3 章 API について

API について

次の両方を使用できます。アプリケーションによって提供される標準 API とユーザーが作成した拡張 (カスタム) API。Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales は、標準 API を提供して、最も一般的なビジネス・シナリオを処理します。例えば、オーダーの作成、オーダーの割り振り、および出荷確認の報告を行う API があります。標準 API は、直接呼び出すか、またはさらに複雑なサービスに集約できます。

API の動作

各 API は入力として XML 文書を取得し、出力として別の XML 文書を返します。YFS 環境入力パラメーターは、この API が呼び出された実行時の状態を表します。これは、以下のタスクで使用されます。

- セキュリティー監査およびロギング
- トランザクション制御
- 呼び出し固有の API の動作の達成

非同期サービスの場合、Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales は、このオブジェクトのインスタンスを自動的に作成し、これをサービスの各 API 部分に渡します。同期サービスをプログラムで呼び出すには、createEnvironment() API を呼び出して、この環境のインスタンスを作成する必要があります。

注: 一般的に、API への入力には「空の」要素または属性を含めないでください。空の要素は、すべての属性が空の値を持っている要素として定義できます。空の要素が渡されると、API の動作が予測不能になります。

すべての API (標準か拡張かにかかわらず) は、入力パラメーターと戻り値に関して同じシグニチャーを持っています。このシグニチャーの形式は、以下のとおりです。

```
org.w3c.dom.Document APIName(YFSEnvironment env, org.w3c.dom.Document input);
```

カスタム API でカスタム値にアクセスするためには、API は com.yantra.interop.japi.YIFCustomApi インターフェースを実装する必要があります。これらの名前と値のペアは、入力されるとプロパティー・オブジェクトとしてカスタム API に渡されます。com.yantra.interop.japi.YIFCustomApi 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。

API のタイプ

API は、キー属性値に基づいてレコードを処理し、1 次キーからレコードを処理します。1 次キーが見つからない場合、API は次に論理キーを検索し、そのレコードを処理します。例えば、ChangeOrder() API は、まず OrderHeaderKey キー属性を検索し、次に OrderNo と EnterpriseCode のキー属性の組み合わせを検索します。

選択 API

選択 API は、通常、接頭辞が「get」で、エンティティーの 1 つのレコードを返します (例えば、getOrderDetails() API は 1 つのオーダーの詳細を返します)。これらはデータベースを更新しません。

選択 API はレコードを 1 つのみ返すため、入力 XML に渡される固有のキー属性が必要です。固有のキー属性が入力 XML に渡されない場合、API は基準の属性に空白を使用してレコードを選択します。複数の固有キーの組み合わせがある場合もあります。この組み合わせでは、複数の組み合わせのいずれかを渡す必要があります。

例えば、オーダーは、OrderHeaderKey キー属性または OrderNo と EnterpriseCode の属性の組み合わせのいずれかによって固有に識別されます。したがって、getOrderDetails() API を呼び出すとき、OrderHeaderKey 属性または OrderNo、EnterpriseCode および DocumentType のキー属性の組み合わせのいずれかを渡す必要があります。OrderNo のみを渡すと、API は、OrderNo と一致し、空のエンタープライズ・コードを持つオーダーを返します。各 API に対して固有キーの組み合わせを識別するためには、Javadocs を参照してください。

ただし、getOrderDetails() API は、YFS_ORDER_HEADER の更新に選択を使用し、外部プログラム、イベントなどの内部処理がオーダー要素にロックし、その処理をしているスレッドがアクティブになります。こうすることにより、最終コミットまでトランザクション・キャッシュが維持されます。したがって、ネスト化されたトランザクションの使用を回避し、以下を実行してロック機構に対処する必要があります。

1. オーダーのすべてのイベントに対して 1 回だけコミットまたはロールバックを行います。上記のいずれかが失敗した場合、すべてのイベントがロールバックに設定されることに注意してください。
2. 各イベントおよびプロセスに対してオーダーを選択します。複数のイベントを持つオーダーの経過時間が大きい場合、パフォーマンスに影響する可能性があることに注意してください。

リスト API

リスト API は、通常、接頭辞が「get」で、入力 XML で指定される基準と一致するエンティティーのレコードのリストを返します (例えば、getOrderList() API はオーダーのリストを返します)。入力 XML の任意の属性に空の値が含まれている場合、これは無視されます。リスト API はデータベースを更新しません。getPage API を呼び出し、リスト API を入力として getPage API に渡すことによって、リスト API からページ番号を付けられたデータを取得することもできます。getPage API について詳しくは、Javadocs を参照してください。

更新 API

更新 API は、新しいレコードをデータベースに挿入します。この API は、データベースの既存のレコードの変更または削除も行います。既存のレコードを変更または削除する更新 API は、選択 API と同じ論理を使用して、更新するレコードを識別します。レコードが見つからない場合、更新 API は例外をスローします。

API のセキュリティー

API を呼び出すとき、以下の 2 つのレベルのセキュリティーを通過する必要があります。

1. ユーザー ID または証明書 (あるいはこれら両方) による認証。ログイン API は、他の API が呼び出される前に呼び出されます。
2. アクセスできる API を確認する許可。

このセキュリティー手順は、アプリケーション・サーバー・プロセスを通過するあらゆる API 呼び出し用のものです。デフォルトでは、エージェントおよび統合サーバーには、常に API への全アクセス権限があります。

認証検査をパスすると、許可検査によって、アクセスできる API およびリソースが決定されます。この許可検査は、ユーザー・インターフェース (UI) セキュリティーを補うものです。例えば、UI セキュリティーでは、ユーザーをリストする画面へのアクセスを許可する場合があります。画面でユーザーのリストを生成するには、ユーザーをリストする getUserList API の許可検査もパスする必要があります。

許可検査の他の例としては、以下のものがあります。

- 表示目的で getCommonCodeList API を使用する場合、API の出力から明示的に制限されているユーザー情報を取得できないようにする必要があります。
- アラートを割り当てる前に getUserList API を呼び出す場合、ユーザー・パスワードを取得できないようにする必要があります。
- UserHierarchy API を使用してパスワードを変更する場合、以下のようになります。
  - 自分の IsSuperUser フラグを変更できないようにする必要があります。
  - 別のユーザーの情報を変更できないようにする必要があります。
  - さらなるシステム・アクセス権限を与える他のユーザー・グループに加入できないようにする必要があります。

このセキュリティーは、apisecurity 固有のテンプレート・ファイルを使用して実装されます。この apisecurity のテンプレート・ファイルは、(デフォルトで) すべての API が制限を受けている入力要素および出力要素を文書化する XML ファイルです。これらのファイルは、XAPI デプロイメント中に自動的に生成されます (文書生成がオフになっている場合も同様)。

注: サービスは、apisecurity ファイルを使用しません。

テンプレートは、入力および出力の許可検査に使用されます。これらのテンプレートは、正規のテンプレートより優先されます。

例えば、OrganizationCode=#PROHIBITED# と IsSuperUser=#PROHIBITED# の行がある入力テンプレートは、ユーザーが追加でユーザー・グループに加入し、追加のアクセス権を取得することを防止します。

出力テンプレートは、デフォルトの文書ベースのテンプレートが実行するフィルタリングを補います。要素は、apisecurity ファイルで構成されていないために制限を受けている場合、出力で返されることはありません (文書ベースのテンプレートにある場合も同様)。

注: 入力と出力のある時点で、multiApi や getPage のような API は、任意の要素に対する許可アクセス権を持っています。しかし、これらの API に呼び出されるその他の API は、許可検査を受ける必要があります。

API セキュリティーおよびアクセス権レベルへのアクセスは、yfs.properties ファイルの以下のプロパティーで制御されます。すべての許可失敗は、sci.apisecurity というロギング・カテゴリーに記録されます。

- api.security.enabled
  - Y (デフォルト) — API のセキュリティーを使用可能にします
  - N — API のセキュリティーを使用可能にしません
- api.security.mode
  - STRICT — 任意の検証に失敗すると、例外がスローされます。これは、すべてのアクセス権が正しく構成されている場合、実動システムでは適切です。
  - LAX — 無効な入力をフィルタリングして記録しますが、処理は継続されます。このフィルタリングでは、不正な入力または出力にもかかわらず、システムによるほとんどの処理が許可されます。一方、ログを記録することによって変更する必要がある場所の特定の役に立ちます。

    注: システムは、フィルタリングによって動作が不明確になる場合、例外をスローする場合があります。
  - DEBUG — 不正な入力および出力を記録しますが、何もフィルタリングせず、また例外をスローしません。これは、さまざまなプロセスで必要なアクセス権を特定するために、初期開発段階でのみ適切です。

セキュリティー・モードを指定しない場合、フィルタリングは行われず、例外はスローされず、許可検査は行われません。制限されたロギングがあります。

- api.security.override.*apiName*.mode

  個別の API のアクセス権をオーバーライドするには、この設定を使用します。このプロパティーは、api.security.mode と同じ値を使用します。
- api.security.smc.enabled
  - Y — Applications ManagerConfigurator の API のセキュリティーを使用可能にします
  - N (デフォルト) — Applications ManagerConfigurator の API のセキュリティーを使用可能にしません
- api.security.console.enabled
  - Y — アプリケーション・コンソールの API のセキュリティーを使用可能にします

– N (デフォルト) — アプリケーション・コンソールの API のセキュリティーを使用可能にしません

アップグレードするとき、最初にこの機能を使用不可にし、プロパティーですべてのアクセスを付与する必要があります。アップグレードされたシステムで、アクセス権を定義してテストするように、一度に 1 つの API のセキュリティーを使用可能にして、この機能を段階的に導入できます。使用可能にされると、システム・ユーザー・グループのみが API へのアクセス権を付与されます。他のすべてのカスタム・ユーザー・グループには、適切なアクセス権を付与する必要があります。ユーザー・グループのアクセス権について詳しくは、Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales: *構成ガイド アプリケーション・プラットフォーム構成ガイド*を参照してください。

文書への apisecurity ファイルの組み込み

このタスクについて

apisecurity ファイルを文書に組み込み、EAR (エンタープライズ・アーカイブ) ファイルにパッケージ化するには、以下の手順を実行します。

手順

1. 拡張を *INSTALL_DIR*/xapidocs/extn/input ディレクトリーに配置します。
2. 以下のコマンドを実行して、文書 (apisecurity ファイルを含む) を再ビルドします。

   ```
   deployer.sh -t xapideployer -l info [new target]
   ```

3. EAR ファイルをビルドします。apisecurity テンプレート・ファイルが *INSTALL_DIR*/repository/xapi/template/merged/apisecurity からパッケージ化されます。

API による日時の取り扱い

Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales は、日時と日付の両方に関する値を処理します。日時は日付と時刻のコンポーネントを含む値を表し、日付は日付コンポーネントのみを含む値を表します。

日付値は、エンティティー XML で `Nullable="true"` を指定することによって NULL 可能にできます。こうするとテーブルの日付値が空になります。日付列の予測される動作は、以下の表で Y が記述されるときにマーキングされます。

アクション	説明
挿入	データベース・オブジェクトでフィールドが入力されていない (NULL) 場合、データベース・インフラストラクチャーは、NULL 値をデータベースの列に自動的に挿入します。
更新	アプリケーションは、日付を NULL にした場合、データベースの対応するフィールドを NULL 値に設定します。
選択またはリスト	列が NULL 可能として定義され、データベースの日付が NULL として返された場合、その列は自動的に NULL になります。したがって、対応する get メソッドは NULL を返します。

アクション	説明
日付で検索	これが指定された場合、必要に応じて NULL 値を渡すことができます。

注: リリース 8.5 より前のバージョンで日付値を 01/01/2400 に指定した場合、現在この値は NULL として処理されます。特別な意味のある日付は以下のとおりです。

- NULL 日付 - 01/01/2400
- 将来の日付 - 01/01/2500
- 過去の日付 - 01/01/1900

時間帯の指定

日時は、時間帯と関係があります。時間帯は、協定世界時 (UTC) と相対的です。

例えば、ニューヨークで 2003 年 6 月 15 日の 16:00:00 (米国ニューヨークの時間帯) にシステム上でオーダーが作成された場合、そのオーダーを確認したシカゴのユーザーには、オーダーの作成日時は 2003 年 6 月 15 日の 15:00:00 (米国シカゴの時間帯) として表示されます。

UTC から -5 時間であるボストンから発行される時刻の場合、ストリング・リテラル「-5:00」が API から発行される現在の日時属性に付加されます。入力「2003-04-23T14:15:32-05:00」は、トランザクションの日付、時刻および時間帯リファレンスを提供します。

yfs.properties ファイルの yfs.install.localecode パラメーターによって、Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales の時間帯が決まります。例えば、`yfs.install.localecode=en_US_EST` のようになります。

時間帯を構成するには、yfs.install.localecode プロパティーを *INSTALL_DIR*/properties/customer_overrides.properties ファイルで `en_US_EST` に設定します。

日時構文の使用

日時フィールドを使用するすべての API、外部プログラム、およびイベントには同一の構文 (ISO 8601 仕様の基本フォーマットおよび拡張フォーマットの組み合わせ) があります。この構文は、すべての入力および出力で予期される形式です。

日付のみの構文

YYYY-MM-DD

日時の構文

*YYYY-MM-DD***T***HH*:*MI*:*SS*+*HH*:*MM*

上記の太字の値は、リテラルのプレースホルダーです。例えば、2003 年 3 月 5 日、午後 11:30:59 のフォーマットは、2003-03-05T23:30:59 になります。

注: この構文は、ISO の日時構文で、データベースの構文ではありません。ISO の日時フォーマット以外の構文を使用すると、問題が発生します。例えば、日時構文の時刻要素が見逃されるか、または不正に計算される場合があります。

例えば、日時入力に「2007-05-18-19.10.28.000000」を入力すると、システムは、入力に T 記号がないため、これを「2007-05-18」とだけ解釈する場合があります。

構文パラメーター

パラメーター	説明
YYYY	必須。4 桁の年。日時フィールドと日付フィールドの両方に使用します。
MM	必須。2 桁の月。日時フィールドと日付フィールドの両方に使用します。
DD	必須。2 桁の日。日時フィールドと日付フィールドの両方に使用します。
T	必須。日付と時刻のコンポーネントを分けるリテラル値 T。日時フィールドのみに使用します。
HH	必須。2 桁の時刻。例えば、午後 11 時は 23:00:00 と表示されます。日時フィールドのみに使用します。
MI	必須。2 桁の分。例えば、59 分は、00:59:00 と表示されます。日時フィールドのみに使用します。
SS	必須。2 桁の秒。例えば、21 秒は、00:00:21 と表示されます。日時フィールドのみに使用します。
+HH:MI	オプション。コロン (「:」) で分けられた 2 桁の時分。UTC からの時間を示します。- を使用して UTC より早い時刻を示し、+ を使用して UTC より遅い時刻を示します。入力でこの値が渡されない場合、アプリケーションの時間帯が使用されます。

第 4 章 API の入力 XML ファイル

API の入力 XML ファイルについて

API は、選択または使用する必要があるレコードを定義する入力 XML ファイルを使用してデータを取り出します。データベースを拡張して追加のフィールドを含める場合、入力 XML も拡張してそのフィールドにデータを入力する必要があります。

注意: 空の要素 (すべての属性が空の値を持っている要素) を API に渡さないでください。また、先頭または末尾にスペースがある属性を渡さないでください。これらを行った場合の結果は予測できません。

以下の例は、入力 XML の変更を示しています。

入力 XML の変更の例

以下の例では、YFS_createOrder() API の入力 XML ファイルを変更します。

```
<Orders AuthenticationKey="">
   <Order EnterpriseCode="DEFAULT" OrderNo="DB04"
OrderName="DB04" OrderDate="20010803" OrderType="Phone" PriorityCode="1"
PriorityNumber="1" ReqDeliveryDate="20010810" ReqCancelDate=""
ReqShipDate="20010810" SCAC="FEDEX" CarrierServiceCode="Express Saver Pak"
CarrierAccountNo="112255" NotifyAfterShipmentFlag="N"
NotificationType="FAX" NotificationReference="" ShipCompleteFlag="N"
EnteredBy="Iain " ChargeActualFreightFlag="Y" AORFlag="Y"
SearchCriteria1="Search" SearchCriteria2="Search Again" >
      <OrderLines>
         <OrderLine PrimeLineNo="1" SubLineNo="1"
OrderedQty="1" ReqDeliveryDate="20010810" ReqCancelDate="20010810"
ReqShipDate="20010810" SCAC="FEDEX" CarrierServiceCode="Express
Saver Pak" PickableFlag="Y" HoldFlag="N" CustomerPONo="11" >
            <Extn ExtnAcmeLineType="Type1"/>
            <Item ItemID="ITEM1" ProductClass="A" ItemWeight="1"
ItemDesc="paintball gun" ItemShortDesc="pball gun"
UnitOfMeasure="EACH" CustomerItem="Spectra Flex" CustomerItemDesc="GEGRG"
SupplierItem="Spectra Flex @ supplier" SupplierItemDesc="Spectra
Flex Desc @ supplier" UnitCost="15.99"  CountryOfOrigin="CA"/>
            <PersonInfoShipTo Title="Mr" FirstName="Quigley"
MiddleName="Al" LastName="Johns" Company="Company" JobTitle="Project
Clert" AddressLine1="Address Line 1 -3 Main Street" AddressLine2="ShipTo
Address line 2" AddressLine3="ShipTo Address line 3" AddressLine4="ShipTo
Address line 4" AddressLine5="ShipTo Address line 5" AddressLine6="ShipTo
Address line 6" City="Acton" State="MA" ZipCode="01720" Country="US"
DayPhone="978-635-9242" EveningPhone="978-635-9252" MobilePhone="978-888-8888"
Beeper="" OtherPhone="other555-5555" DayFaxNo="" EveningFaxNo=""
EMailID="jquigley@maine.com" AlternateEmailID="hfournier@ontario.com"
ShipToID=""/>
         </OrderLine>
         <NumberOfOrderLines/>
      </OrderLines>
      <PersonInfoShipTo Title="MR" FirstName="s"
MiddleName="X" LastName="T" Suffix="T" Department="T" Company="SD"
JobTitle="SS" AddressLine1="SS" AddressLine2="SS" AddressLine3="SS"
AddressLine4="SS" AddressLine5="SS" AddressLine6="SS" City="REDWOOD"
State="CA" ZipCode="01852" Country="USA" DayPhone="3456789234"
EveningPhone="3456789234" MobilePhone=""  EveningFaxNo="SS" />
```

```
      <PersonInfoBillTo Title="mj" FirstName="m"
MiddleName="JJ" LastName="KK" Suffix="lll" Department="l" Company="kj"
JobTitle="k" AddressLine1="HJHKK" AddressLine2="HJKHK" AddressLine3="HKHJ"
AddressLine4="" AddressLine5="" AddressLine6="" City="UUU" State="IUI"
ZipCode="78787" Country="USA"  />
  </Order>
  <NumberOfOrders/>
</Orders>
```

重要: ファクトリー・セットアップ・スクリプトを正しく動作するために、列をデータベース表に追加するとき、列が NULL ではないことおよび列にデフォルト値が入っていることを確認してください。列を NULL 可能にする必要がある場合、デフォルト値が存在してはいけません。

また、XML 名および XML グループを指定している場合、これらの値が有効な Document Object Model (DOM) ストリングである必要があることに注意してください (これらの値には、スペースや DOM 仕様でサポートされていない特殊文字が含まれていてはいけません)。

次の XML ファイルの例では、列を YFS_ORDER_LINE テーブルに追加します。

```
<?xml version="1.0" encoding="UTF-8" ?>
<DBSchema>
 <Entities>
  <Entity TableName="YFS_ORDER_LINE">
   <Attributes>
    <Attribute ColumnName="EXTN_ACME_LINE_TYPE" DecimalDigits=""
Default Value="' '" Size="10" Type="CHAR" XMLGroup="Extn"
XMLName="ExtnAcmeLineType"/>
   </Attributes>
  </Entity>
 </Entities>
</DBSchema>
```

API 入力形成のガイドライン

API 入力パラメーターをコーディングする場合、リテラルの使用および API 入力のフォーマット設定に関して記載されたガイドラインに従います。

空の要素 (すべての属性が空の値を持っている要素) を API に渡さないでください。また、先頭または末尾にスペースがある属性を渡さないでください。これらを行った場合の結果は予測できません。

マップおよび XML でのリテラルの使用

リテラルを使用すると、コンパイラーが *name=value* のペアでの不正な名前の使用を検出するため、ほとんどバグのないコードを書き込むことが可能になります。また、リテラルを使用すると、コードの保守が容易になります。*name* を変更する場合、まず 1 つ以上の *name* インスタンスを編集する代わりに、コードを再コンパイルする必要があるだけになります。

特殊文字の使用

Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales スキーマ (OrganizationCode や OrderNo など) の任意のレコードの論理キーの一部

であるフィールドには、いくつかの制約があります。このようなフィールドでは、アプリケーションは、以下のテーブルにリストされた特殊文字の使用をサポートしていません。

特殊文字	説明
&	アンパーサンド
>	より大
<	より小
%	パーセント
"	引用符
+	正符号
'	一重引用符または アポストロフィ
(	括弧
)	括弧
[	大括弧
]	大括弧

注: 正符号 (+) およびアンパーサンド文字 (&) は、ItemID フィールドでのみ使用できます。

注: スラッシュ (/) は、別の属性のパスの一部を形成する属性では使用しないでください。例えば、カテゴリー ID は、属性カテゴリー・パスを形成するために使用され、カテゴリー・パスはパス分離文字としてスラッシュを使用するため、カテゴリー ID にはスラッシュを含めないでください。

また、サード・パーティー・ベンダーが予約している特殊文字を使用しないことをお勧めします。例えば、ある状況において、Oracle データベースで下線文字 ("_") 付きのデータを使用すると、データベースが下線を単一文字のワイルドカードとして復号するため、クエリーのパフォーマンスが予想外に遅くなる可能性があります。

以下のフィールドには制約事項はなく、すべての文字がサポートされます。

- すべての説明フィールド (例えば、アイテムの説明)
- すべての名前フィールド (例えば、組織名)
- すべてのアドレス・フィールド (例えば、請求先住所)

  注: ただし、UI を使用してアドレス・フィールドを作成すると、引用符 (") の後に入力した情報は切り捨てられ、YFS_PERSON_INFO テーブルで新しいエントリーとして表示されます。この問題を回避するには、引用符の代わりにアポストロフィ (') を使用します。
- すべての指示フィールド (例えば、贈答品の包装材)
- すべてのテキスト・フィールド (例えば、理由やコメント)

注: Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales モバイル・デバイスは、アンパーサンド (&) 文字の使用をサポートしていません。

XML ベース API

以下のテーブルでは、特殊文字および XML で特殊文字の代わりに使用する必要があるエスケープ・シーケンスをリストしています。

特殊文字	特殊文字用のシーケンス
引用符 (")	&quot;
単一引用符 (')	&apos;
より大記号 (>)	&gt;
より小記号 (<)	&lt;
アンパーサンド (&)	&amp;

入力 XML ファイルおよび出力 XML ファイルでの CreateTS および ModifyTS のサポート

CreateTS および ModifyTS は、これらの属性を必要とする入力 XML ファイルまたは出力 XML ファイルのエンティティーに対応するテーブルがある場合、getAPIs (入力または出力) で使用できます。これらの属性は、データベースでいつレコードが作成されたか、またはいつレコードが変更されたかを示します。

リスト API の入力 XML のクエリーの形成

リスト API の入力 XML では、先頭の文字、含む文字、次より大などの条件のクエリーが使用可能になります。以下の例では、特定の重量範囲かつ特定の日付範囲に出荷される特定の出荷ノードでアイテムのリストを返す入力 XML の一部を示しています。

例: クエリー・タイプ値を含む getOrderList API 入力 XML

```
<Order ReqShipDateQryType="DATERANGE" FromReqShipDate="20010113" ToReqShipDate="20030113" /Order>
<OrderLine ShipNode="Atlantic" /OrderLine>
<Item ItemWeightQryType="BETWEEN" FromItemWeight="2" ToItemWeight="20"/>
<OrderRelease CarrierServiceCodeQryType="FLIKE"
CarrierServiceCode="Priority" />
```

注: 入力属性に対して QryType をサポートしていない API もあります。そのような API は以下のとおりです。

- getAssignedPricelistHeaderList
- getCarrierServiceList
- getNodeSCACAccountList
- getOrderLineStatusList
- getPaymentStatusList
- getPriceListForOrdering
- getQueryTypeList
- getReceiptLinesForTransaction
- getRegionList
- getServerList
- getShipmentListForOrder

- getSurroundingNodeList
- getTaskQueueDataList
- getTraceableComponentList
- getTraceList
- getZoneListForDiscount

クエリーの形成

このタスクについて

クエリーを形成する手順は、以下のとおりです。

手順

1. 任意のリスト API のカスタム入力 XML を編集し、クエリー対象の任意の属性に QryType を追加します。リスト API の入力 XML のクエリーの形成に関するトピックの例の ShipNode で示すように、デフォルトのクエリー・タイプ値 EQ を使用して、QryType が付加されていない属性もクエリー対象にできます。
2. QryType が付加された属性の場合、以下のテーブルからクエリー・タイプ値を指定します。これは、大/小文字の区別があります。
3. 検索条件に適用可能な値を指定します。

   QryType 属性の値は、フィールドのデータ・タイプによってさまざまです。以下のテーブルは、各データ・タイプに対してサポートされるクエリー・タイプ値をリストしています。

フィールド・データ・タイプ	サポートされるクエリー・タイプ値
Char/VarChar2	• EQ - 等しい • FLIKE - 次から始まる • LIKE - 含む • GT - より後 • LT - より前
数値	• BETWEEN - 値の範囲 • EQ - 等しい • GE - 以降 • GT - より後 • LE - 以前 • LT - より前 • NE - 等しくない

フィールド・データ・タイプ	サポートされるクエリー・タイプ値
日付	• DATERANGE - 日付範囲 • EQ - 等しい • GE - 以降 • GT - より後 • LE - 以前 • LT - より前 • NE - 等しくない
日時	• BETWEEN - 日付範囲 • EQ - 等しい • GE - 以降 • GT - より後 • LE - 以前 • LT - より前 • NE - 等しくない
NULL	• ISNULL - NULL のレコードを返します。 • NOTNULL - NULL 以外のレコードを返します。 **注:** これら 2 つのクエリー・タイプは、列または属性がエンティティー XML で NULL 可能に設定されている場合に使用されます。

XAPI のクエリー・タイムアウトの設定

このタスクについて

API に対して個別のクエリー・タイムアウトを追加できます。これを行うには、API の入力 XML の API に対して秒単位でクエリー・タイムアウト値を指定します。例:

```
<ApiInput QueryTimeout="10">
...
...
...
</ApiInput>
```

注: QueryTimeout 属性の値は、yfs.properties ファイルの yfs.ui.queryTimeout プロパティーの値より優先されます。ただし、この属性の値は、1 回の API 呼び出しに対してのみ有効です。API の実行が完了した後、このクエリー・タイムアウトは、yfs.properties ファイルの yfs.ui.queryTimeout プロパティーの値に基づいて元の値に設定されます。

リスト API の入力 XML の OrderBy 要素でのソート

リスト API の入力 XML は、OrderBy 要素に基づくソートをサポートしています。

クエリーを形成する手順は、以下のとおりです。

任意のリスト API のカスタム入力 XML を編集し、OrderBy 要素を追加します。Attribute 子要素を追加し、Name 属性で結果のソートに使用するフィールドに基づいてフィールドの名前を指定します。

OrderBy 要素は、属性の並べ替えに昇順と降順の両方をサポートします。デフォルトでは、結果は昇順にソートされます。結果を降順でソートする場合、Desc 属性を Attribute 要素に追加し、これを Y に設定します。

OrderBy 要素を使用して、ネスト化ソートも実行できます。

OrderBy 要素を備えた getOrganizationList API 入力 XML

以下の例では、組織のリストを返し、結果を OrganizationName 属性でソートする入力 XML の一部を示しています。

```
<Organization IgnoreOrdering="N" MaximumRecords="5000">
  <OrderBy>
     <Attribute Name="OrganizationName"/>
   </OrderBy>
</Organization>
```

ネスト化 OrderBy 要素を備えた getOrganizationList API 入力 XML

以下の例では、組織のリストを返し、結果を OrganizationName 属性および LocaleCode 属性でソートする入力 XML の一部を示しています。

```
<Organization IgnoreOrdering="N" MaximumRecords="5000">
   <OrderBy>
     <Attribute Name="OrganizationName"/>
     <Attribute Name="LocaleCode"/>
    </OrderBy>
</Organization>
```

OrderBy 要素および Desc 属性を備えた getOrganizationList API 入力 XML

以下の例では、組織のリストを返し、結果を OrganizationName 属性で降順でソートする入力 XML の一部を示しています。

```
<Organization IgnoreOrdering="N" MaximumRecords="5000">
   <OrderBy>
      <Attribute Name="OrganizationName" Desc="Y"/>
   </OrderBy>
</Organization>
```

第 5 章 API の出力 XML ファイル

API の出力 XML ファイルとテンプレートについて

API は、API で必要な要素および属性を定義する、2 つのタイプの出力 XML ファイルを使用してデータを返します。

- 出力 XML ファイル - API が返すことができるデータの外部制限を定義します。出力 XML ファイルを変更しないでください。
- テンプレート XML ファイル - 入力 XML ファイルで指定されたレコード用の API によって返されるデータを定義し、出力 XML のサブセットに対するデータ量を制限します。このファイルを変更して、出力 XML から属性と要素のサブセットを取り込めます。

出力 XML テンプレート

多くの API は、対応する出力テンプレートを使用します。出力テンプレートは、XML フォーマットで、返す必要がある要素と属性を決定するために API によって読み込まれます。標準出力テンプレートは、あらゆる特定の API のために返される要素と属性を定義します (API が返すことができる値の全範囲を確認するには、Javadocs でその出力 XML を確認します)。標準テンプレートは、Javadocs で出力 XML によって決定されるように、返される値の全範囲のサブセットにできます。

注: 要素と属性を出力テンプレートに追加するとき、Javadocs で文書化されているもののみを使用してください。API はその他の要素と属性を出力できますが、Javadocs で文書化されているもののみがサポートされます。

例えば、getOrderList() API の標準出力テンプレートはオーダーのヘッダー・レベルの情報を返し、getOrderDetails() API の標準出力テンプレートはオーダーに関する詳細情報を返します。

標準出力 XML テンプレートに加えて、API 用にカスタム出力テンプレートを作成して、独自のビジネス要件 (異なる伝票種別に対する異なる出力など) に使用できます。

伝票種別

オーダー、計画済みオーダー、購入オーダー、返品など、さまざまなビジネス関連の伝票種別を使用する場合、それぞれの伝票種別に関する値を API が返すことができるカスタム・テンプレートを使用できます。

例えば、あるテンプレートを getOrderDetails() API で使用して計画済みオーダーに関する情報を返し、別のテンプレートを getOrderDetails() API に対して使用してオーダーに関する別の情報を返すことができます。

標準出力テンプレートの動作

標準出力テンプレートが返す値のセットは、さまざまなビジネス・シナリオに対応します。この範囲は非常に広いため、標準出力テンプレートを使用している API

は、ビジネスの目的に必要なデータよりもはるかに多いデータを返す場合があります (また、その処理に必要以上の時間がかかる場合があります)。

API によって返される情報をカスタマイズする場合、ガイドラインと手順を使用して、カスタム・テンプレートを作成して使用することによって、これを実行できます。

出力 XML テンプレートの拡張

このタスクについて

多くの API は、出力 XML テンプレートを使用して、返されるものを定義します。各 API には、独自の XML テンプレートがあり、これは *INSTALL_DIR*/repository/xapi/template/merged/api/*apiName*.xml ファイルから取得されます。このディレクトリーのファイルは製品の一部です。これを変更しないでください。ただし、これらのテンプレートは、テンプレート拡張を実装することによってオーバーライドできます。

テンプレート・ファイルを拡張する手順は、次のとおりです。

手順

1. テンプレート *INSTALL_DIR*/repository/xapi/template/merged/api/*apiName*.xml ファイルを *INSTALL_DIR*/extensions/global/template/api/ ディレクトリーに同じファイル名を保持してコピーします。

   /global/template/api/ ディレクトリーが存在しない場合、必要なディレクトリー構造を作成します。
2. 必要に応じてコピーしたファイルを変更します。テンプレート・ファイルを拡張するには、Extn タグをエンティティー・タグの下に追加します。例えば、列 EXTN_COLORを YFS_ITEM テーブルに追加した場合、以下のように、タグ Extn を getItemDetails.xml ファイルのタグ Item の下に追加する必要があります。

   ```
   <Item ItemKey=""....>
      <PrimaryInfo MasterCatalogID="" .../>
        ...
       <Extn ExtnColor=""/>
   </Item>
   ```

   注: 出力 XML テンプレートを拡張する場合、拡張したファイルを *INSTALL_DIR*/extensions/global/template/api フォルダーに配置します。ただし、サービス定義中に template.api ファイルの名前を付ける場合、そのパスを /global/template/api/*CUSTOM-TEMPLATE-API* にする必要があります。

カスタム出力 XML テンプレート作成のベスト・プラクティス

API を呼び出すときは必ず、Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales によって提供されるサンプルではなく、独自にカスタマイズしたテンプレートを渡す必要があります。このセクションは、カスタム出力テンプレートの設計方法を計画する意思決定処理の役に立ちます。

一般的に、出力テンプレートをカスタマイズする場合、標準テンプレートのコピーを編集して行います。標準出力テンプレートを変更することはできません。

出力テンプレートをカスタマイズして呼び出す方法は 2 つあります。どちらの機能を選択するかは、API で返すデータ・セットのサイズおよびタイプによって異なります。

API に関する情報の収集

カスタム出力テンプレートは、希望のデータは何でも返す柔軟性を提供するため、API にあまり関係のない情報を返すように出力テンプレートを変更することができることを理解することは重要です。

例えば、getOrderList() API の出力テンプレートをオーダーのヘッダー・レベルの情報だけではなく詳細情報を返すように変更できます。対応する API 固有の特徴を利用するように出力テンプレートを変更する必要があります。各テンプレートをその API 固有に保つことによって、特定のシチュエーションでどの API を使用するかに関するあいまいさがなくなります。

ビジネス要件に関する情報の収集

標準出力テンプレートは、テンプレートの空の要素まで含むすべての属性を返すため、特定のビジネス要件に情報を合わせる必要があります。不要な属性を除外しないと、必要以上のデータを受け取り、余分なデータによって API のパフォーマンスが遅くなります。

例えば、OrderLine 属性のみを返すために getOrderDetails() API を使用しているが、カスタム出力テンプレートに Schedule 属性が含まれている場合、OrderLine と Schedule のすべての属性が返されます。

適切なテンプレート機構の選択

一般的に、任意のテンプレートの形式は、標準テンプレートと同じ構造にする必要があります。この一般的な規則に注意して、標準テンプレートをカスタマイズする方法には、返すデータ量および呼び出し方法によって異なる 2 つの方法があります。

- 静的テンプレート
- 動的テンプレート

静的テンプレートは、新しい要素を追加する機能を提供しますが、どのデフォルトも削除しません。静的テンプレートは、API を呼び出したときに必ずデフォルトでその API によって使用されるため、パーベイシブです。

動的テンプレートは、新しい要素を追加する機能を提供し、任意のデフォルト要素を標準テンプレートから削除します。動的テンプレートは、特定の API 呼び出しに対してのみ使用されるため (ユーザー・インターフェース拡張の間にそのように構成されている場合など)、インスタンスです。

2 つのタイプのテンプレート機構の違いの比較を以下のテーブルに要約します。

テンプレート・タイプ	許可される XML 要素	動作
静的テンプレート	デフォルト・テンプレート要素は削除できません。 新しい要素を追加できます。	パーベイシブ。デフォルトで API によって使用されます。
動的テンプレート	デフォルト・テンプレート要素を削除できます。 新しい要素を追加できます。	インスタンス。ユーザー・インターフェース拡張中に構成された場合のように、特定の API 呼び出しに対して使用されます。

ビジネス要件に最適なメカニズムを選択し、これを使用し続けてください。

動的テンプレートを定義した場合、すべての可能な値が返されることに注意してください。要素に対して返されるデータ量を最小化するために、不要な要素を整理するときに、その親を少なくとも属性の 1 つとともに含める必要があります。

要素を空のままにするか親要素に不要な属性を含めると、以下の不適切に整理された動的テンプレートの例に示すように、すべての値が返されます。

不適切に整理された動的テンプレート

```
<!-- getOrderDetails Output XML -->
<Order>
   <OrderLines>
     <!--1 or more order line-->
       <OrderLine>
          <Item CountryOfOrigin="" ItemDesc="" ItemID=""/>
          <Schedules>
             <Schedule Attr1 ..... />
          </Schedules>
       </OrderLine>
   </OrderLines>
<Order>
```

不適切に整理された動的テンプレートは、すべての OrderLine 属性およびいくつかの Item 属性および Schedule 属性を指定しているため、API は以下のような値を返します。

```
<OrderLine AllocationDate="03/28/2002" CarrierAccountNo="112233"
CarrierServiceCode="Next Day Air" Createprogid="CustomTester"
Createts="03/28/2002" Createuserid="CustomTester" CustomerLinePONo="999"
CustomerPONo="111" DeliveryCode="AIR" DepartmentCode="Clothing"
ExtendedFlag="" ExternalReference1="" ExternalReference2=""
ExternalReference3="" ExternalReference4="" ExternalReference5=""
FreightTerms="Buyer" HoldFlag="N" HoldReasonCode="HoldReas"
ImportLicenseExpDate="08/08/2002" ImportLicenseNo="225588"
InternalReference1="" InternalReference2="" InternalReference3=""
InternalReference4="" InternalReference5="" KitCode=""
LineClass="" LineSeqNo="1.1" LineType="Single" Lockid="1" MarkForKey=""
Modifyprogid="CustomTester" Modifyts="03/28/2002"
Modifyuserid="CustomTester" OrderClass="NEW"
OrderHeaderKey="200203281036245174" OrderLineKey="200203281036245175"
OrderedQty="5.00" OrigOrderLineKey="" OriginalOrderedQty="5.00"
OtherCharges="0.00" OtherChargesPerLine="0.00" OtherChargesPerUnit="0.00"
PackListType="Bill" PersonalizeCode="PersCode" PersonalizeFlag=""
PickableFlag="Y" PricingDate="01/01/2500" PrimeLineNo="1"
Purpose="Purpose" ReceivingNode="B1N1" ReqCancelDate="01/01/2500"
ReqDeliveryDate="04/04/2002" ReqShipDate="03/30/2002"
```

```
ReservationID="" ReservationPool="" SCAC="UPS" ShipNode="E1N1" ShipToID=""
ShipToKey="" ShipTogetherNo="Y" SplitQty="0.00" SubLineNo="1"
TotalDiscountAmount="0.00" TotalOtherCharges="0.00">
<Item CountryOfOrigin="" ItemDesc="" ItemID=""/>
<Schedules>
<Schedule ExpectedDeliveryDate="" ExpectedShipmentDate=""
TagNumber="" OrderHeaderKey="" OrderLineKey="" OrderLineScheduleKey=""
ScheduleNo="" ShipByDate="" Quantity=""  PromisedApptStartDate=""
PromisedApptEndDate=""/>
</Schedules>
</OrderLine>
</OrderLines>
</Order>
```

注意深く整理されたカスタム出力テンプレート

この注意深く整理されたカスタム出力テンプレートでは、動的テンプレートが切り取られています。以下のガイドラインに注意してください。

- カスタム出力テンプレートの構造は、標準出力テンプレートの構造をミラーリングしています。
- 余分な要素 (キット、スケジュール、住所などに関して) が整理されています。
- 余分な詳細を抑制するために、親要素に 1 つの属性が入力されています。例えば、Order 属性に OrderNo 属性を指定すると、他のすべての Order 属性が抑制されます。

```
<!-- getOrderDetails Output XML -->
<Order OrderNo="">
   <OrderLines>
      <!--1 or more order line-->
        <OrderLine PrimeLineNo="">
            <Item CountryOfOrigin="" ItemDesc="" ItemID=""/>
     </OrderLine>
    </OrderLines>
</Order>
```

この注意深く整理されたカスタム出力テンプレートは、その親要素に対して少数の Item 属性および 1 つの属性のみを指定するため、getOrderDetails() API は以下の値のみを返します。

```
<?xml version="1.0" encoding="UTF-8" ?>
<Order OrderNo=Y00000765>
   <OrderLines>
      <OrderLine PrimeLineNo="1">
          <Item CountryOfOrigin="IN" Item Description"
ItemDesc="Green Sari" ItemID="GNSARI5LT" />
       </OrderLine>
       <OrderLine PrimeLineNo="3">
          <Item CountryOfOrigin="CA" ItemDesc="Pink Scarf"
ItemID="PKSCARF4LT" />
       </OrderLine>
    </OrderLines>
</Order>
```

テンプレートを整理するこの方法では、オーダー・スケジュールなどの不要な要素へのデータベース・アクセスが行われないため、パフォーマンスが向上します。

便利なテンプレートの作成

INSTALL_DIR/repository/xapi/template/merged/api/ ディレクトリーにある提供されたテンプレートは、サンプル・ガイドです。これらを使用して、独自の出力 XML テン

プレートの作成に役立ててください。独自にカスタマイズされたテンプレートを使用すると、柔軟性がはるかに向上し、パフォーマンスも向上し、より確実に適切なデータ出力を得られます。env を使用するか、または拡張フォルダーに配置することによって、自分のテンプレートを渡すことができます。

パフォーマンス保持の必要性

ビジネス要件に合わせてテンプレートを調整することに加えて、技術的な考慮事項に注意する必要があります。出力の使用に関するパフォーマンス関連の情報については、Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales: *パフォーマンス管理ガイド*を参照してください。

出力 XML の静的テンプレートの定義とデプロイ

このタスクについて

標準出力テンプレートの要素に加えてこれ以外の要素を含むテンプレートを使用する場合、静的出力テンプレートを作成します。この機能を使用すると、標準出力テンプレートのすべての要素と追加した新しい要素を含むテンプレートを作成できます。例えば、追加した任意のデータベース列に UI フィールドを追加する必要がある場合があります。この機能を使用した場合、標準テンプレートに存在する任意の要素を削除できないことに注意してください。

静的テンプレートを定義およびデプロイする手順は、次のとおりです。

手順

1. 変更する API の標準出力テンプレートを *INSTALL_DIR*/repository/xapi/template/merged/api/*FileName*.xml ファイルから *INSTALL_DIR*/extensions/global/template/*FileName*[.*DocType*].xml にコピーします。
   - 新しいテンプレートのファイル名を標準テンプレートの名前と同じままにします。

     出力テンプレートの名前は、これに関連付けられた API またはイベントの名前に対応します。例えば、getOrderDetails() API は、出力テンプレート・ファイル getOrderDetails.xml を取ります。
   - テンプレートが伝票種別を参照する場合、ファイル名に伝票種別コードを含めます。

     例えば、オーダー (0001) 伝票種別の getOrderDetails() API の出力テンプレートを作成するには、テンプレート XML の名前は getOrderDetails.0001.xml になります。
2. 必要に応じて /extensions/global/template/api ディレクトリーでコピーしたテンプレートを変更します。カスタム出力 XML テンプレートの作成のベスト・プラクティスに従います。

   注: 希望の要素を追加できますが、標準出力テンプレートにある任意の要素を削除できません。
3. 標準として API を呼び出すと、カスタム・テンプレートを含むディレクトリーからカスタム出力テンプレートが自動的に取得されます。

出力 XML の動的テンプレートの定義とデプロイ

このタスクについて

標準出力テンプレートの要素のサブセットを含むテンプレートを使用する場合、動的出力テンプレートを作成します。標準テンプレートから一部の要素を削除し、独自の要素を追加する機能が必要な場合、XML データまたはファイル名を YFS 環境オブジェクトに渡すことによってこれを実行できます。

動的テンプレートを定義およびデプロイする手順は、次のとおりです。

手順

1. 変更する API の標準出力テンプレートを *INSTALL_DIR*/repository/xapi/template/merged/api/*FileName*.xml ファイルから *INSTALL_DIR*/extensions/global/template/api/*FileName*.xml にコピーします。

   新しいテンプレート・ファイルに名前を付けるとき、標準テンプレートと同じ名前を付けます。

   出力テンプレートの名前は、これに関連付けられた API またはイベントの名前に対応します。例えば、getOrderDetails() API は、出力テンプレート・ファイル getOrderDetails.xml を取ります。
2. 必要に応じて /extensions/global/template/api ディレクトリーでコピーしたテンプレートを変更します。カスタム出力 XML テンプレート作成のベスト・プラクティスに従います。
3. ユーザー・インターフェースの拡張中、YFS 環境オブジェクトの setApiTemplate() 関数を呼び出します。こうすることによって、以下の関数のいずれかを使用して、API を呼び出す前に出力テンプレートを指定できます。
   - XML データを変数として - 以下の例のようにします。

     ```
     YFSEnvironment env = createEnv();
     Document doc = getTemplateDocument();
     env.setApiTemplate("getOrderDetails", doc);
     private YFSEnvironment createEnv() {
     //create new environment by passing the user id, program id, etc.
     }
     private Document getTemplateDocument() {
     //create a Document object containing the desired template XML.
     }
     ```

   - XML ファイルを変数として - 以下の例のようにします。

     ```
     YFSEnvironment env = createEnv();
     env.clearApiTemplates();
     env.setApiTemplate("getOrderDetails", "myOrderDetails");
     private YFSEnvironment createEnv() {
     //create new environment by passing the user id, program id, etc.
     }
     ```

     次にこの API は、YFS 環境に渡されたテンプレートを使用して、出力 XML 文書を作成します。YFS 環境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。

出力 XML テンプレートの優先順位

Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales では、変更できない標準テンプレートの他に複数のタイプのテンプレートを定義できるため、API およびイベントのテンプレートを読み込むときに Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales が実装する優先順位を理解する必要があります。

API テンプレート

以下の表は、API で使用する出力テンプレートを決定する優先順位を示しています (イベントは、類似の優先順位を使用します)。テンプレートは外部プログラムに対してはサポートされていないことに注意してください。

優先順位	出力テンプレート・パスおよびファイル名
1	setApiTemplate(*file|xmlDocument*) to `YFSEnvironment` ファイルを指定すると、*INSTALL_DIR*/extensions/global/template/api ディレクトリーから取得されます。
2	*INSTALL_DIR*/extensions/global/template/api/apiName.docType.xml
3	*INSTALL_DIR*/extensions/global/template/api/apiName.docType.xml (Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales のサンプル・テンプレート。実動では使用しません。)
4	*INSTALL_DIR*/extensions/global/template/api/apiName.xml
5	*INSTALL_DIR*/extensions/global/template/api/apiName.xml (Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales のサンプル・テンプレート。実動では使用しません。)

イベント・テンプレート

イベント・テンプレートは、イベントの出力 XML に存在する必要がある要素および属性を決定する役に立ちます。発生するイベントについては、Javadocs の関連トランザクションを参照してください。

出力テンプレートを取るイベントを確認するには、*INSTALL_DIR*/repository/xapi/template/merged/event/ ディレクトリーのファイルを参照してください。これらのテンプレートは、*INSTALL_DIR*/extensions/global/template/ ディレクトリーに配置したファイルによってオーバーライドできます。

テンプレートの命名規則は、BaseTxnName.eventName.xml です。例えば、createOrder() API の on_success イベントは、ORDER_CREATE.ON_SUCCESS.xml イベント・テンプレートを使用します。

注: テンプレートは外部プログラムに対してはサポートされていません。

第 6 章 DTD、XSD、および複合クエリー

DTD および XSD ジェネレーター

どの Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales API も標準の入力 XML 、出力 XML、およびエラー XML を使用します。これらの XML は、関連する伝票種別定義 (DTD) に準拠しています。例えば、以下の XML について考えてみます。

```
<?xml version="1.0" encoding="UTF-8">
<Order EnterpriseCode="DEFAULT" OrderNo="S100" />
```

この XML に対応する DTD は以下のとおりです。

```
<!ELEMENT Order>
<!ATTLIST Order OrderNo CDATA #IMPLIED>
<!ATTLIST Order EnterpriseCode CDATA #REQUIRED>
```

拡張 Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales XML にこのような DTD を作成するために、xsdGenerator.xml というツールが *INSTALL_DIR*/bin ディレクトリーに用意されています。このツールは、特別にフォーマットされた XML ファイルを DTD および XML スキーマ定義 (XSD) に変換します。このツールを実行するコマンドは以下のとおりです。

```
sci_ant.sh -f xsdGenerator.xml generate
```

また、以下のプロパティーをコマンド・ライン引数として渡すこともできます。

- `xsdgen.use.targetnamespace`
- `xsdgen.use.datatypeimport`

例:

```
sci_ant.sh  -Dxsdgen.use.targetnamespace=N
-Dxsdgen.use.datatypeimport=N -f xsdGenerator.xml generate
```

以下のテーブルでは、XSD ジェネレーターのプロパティーについて説明します。

フィールド	説明
xsdgen.use.targetnamespace	オプション。デフォルト値は Y です。Y に設定すると、XSD ファイルが定義済みのターゲット名前空間とともに生成されます。
xsdgen.use.datatypeimport	オプション。デフォルト値は Y です。Y に設定すると、すべての XSD ファイルがすべての共通データ・タイプ定義を含む 1 つの共通の XSD ファイルを参照します。N に設定すると、各 XSD ファイルが、ファイル内に組み込まれたデータベース定義のコピーとともに作成されます。

入力 XML ファイルは、*INSTALL_DIR*/xapidocs/extn/input ディレクトリーに配置される必要があります。結果として得られる DTD ファイルおよび XSD ファイルは、それぞれ *INSTALL_DIR*/xapidocs/extn/output/dtd ディレクトリーと *INSTALL_DIR*/xapidocs/extn/output/xsd ディレクトリーに配置されます。

注: xsdgen.use.datatypeimport は、「Y」に設定されると、データ・タイプ拡張を含むマージされた datatypes.xml に基づいて、更新された datatypes.xsd ファイルを <INSTALL_DIR>/xapidocs/extn/output/xsd ディレクトリーに生成します。

入力ディレクトリーに配置され、XSD および DTD に変換できる以下のサンプル XML について考えてみます。

```
<Item yfc:DTDOccurrence="REQUIRED" ItemKey="" ItemID="REQUIRED"
OrganizationCode="REQUIRED" UnitOfMeasure="">
   <PrimaryInformation Description="" ItemType="" />
   <AdditionalAttributeList>
        <AdditionalAttribute Name="" Value=""/>
    </AdditionalAttributeList>
    <Extn ExtnAttr1="" ExtnRefId="">
       <CSTItemDataList yfc:DTDOccurrence="ZeroOrOne">
         <CSTItemData yfc:DTDOccurrence="ZeroOrMany" ItemDataKey=""
Description="">
           <CSTItemExtraData yfc:DTDOccurrence="ZeroOrOne" CodeType=""
DataType="" />
           <YFSCommonCode yfc:DTDOccurrence="REQUIRED" CodeName=""
CodeType="" CodeValue="" />
         </CSTItemData>
       </CSTItemDataList>
    </Extn>
</Item>
```

以下のテーブルには、XML の特別な属性の説明が含まれています。

フィールド	説明
yfc:QryTypeSupported	この属性は、クエリー・タイプの機能がこの要素の属性でサポートされるかどうかを決定します。Y に設定すると、すべての要素に対して有効になります。
yfc:ComplexQuerySupported	この属性は、複雑なクエリー・タイプをサポートするかどうかを指定します。この属性は、ルート要素にのみ存在します。
yfc:XSDType	ルート要素スキーマ定義に使用するタイプの名前。
yfc:DTDOccurrence	この属性には、以下の値のいずれかを含むことができます。 • REQUIRED - この要素は、親要素が存在する場合、存在する必要があります。 • ZeroOrOne - この要素はオプションですが、1 回だけ使用できます。 • ZeroOrMany - この要素はオプションですが、複数回使用できます。 • OneOrMany - この要素は必須で、複数回使用できます。
yfc:UseEntityOrdering	この属性は、要素の第 1 レベルの子すべてがエンティティー XML での順序で並べられるかどうかを決定します。この属性には、以下の値のいずれかを含むことができます。 • true - 要素の第 1 レベルの子すべてがエンティティー XML での順序で並べられます。 • false - 要素の第 1 レベルの子はエンティティー XML での順序で並べられません。
xmlns	出力 XSD でターゲット名前空間に使用する名前空間。この属性は、ルート要素にある場合のみ有効になります。

値が REQUIRED である属性は、DTD および XSD で必須の属性として生成されます。ただし、既存の必須の属性をオプションとしてマークを付けることはできません。

属性値を指定して、追加の制約を与えることもできます。オプションのリストは、垂直バー (|) によって区切られます。属性の値は、指定のオプションのいずれかである必要があります。これは、ストリングに基づくデータ・タイプに対してのみサポートされます。この値は、値自体がすべてスペースの場合、空白文字を取り除きます。この場合、列挙されたオプションは変更されないままになります。

例えば、SomeAttr="A | B | C | |" は、「A」、「B」、「C」、「 」、および「」の有効なオプションになります。

注: DTD は、空白文字のみ含む列挙値をサポートしません。したがって、このタイプの制約事項は、DTD で表すことはできません。

カスタム XML のベースとして使用できるデフォルトの入力 XML および出力 XML は、*INSTALL_DIR*/xapidocs/xmlstruct/ ディレクトリーにあります。また、標準テーブルに提供される DTDOccurrence および REQUIRED データは、xmlstruct ディレクトリーのベース・ファイルから推測され、提供される必要はありません。これらが提供されている場合、既存の情報は、カスタム XML にある任意の新しい情報によってオーバーライドされます。任意の必須データ・タイプおよび関連情報は、エンティティー XML から取得されます。

注: カスタム XML を xmlstruct ディレクトリーに配置しないでください。

したがって、ツールが実行されると、これらのベース XML ファイルは、カスタム XML ファイルのデフォルトとして動作し、カスタム XML ファイルには拡張要素や属性など、実行された変更を含む必要があるだけになります。このことによって、今後のアップグレードで xmlstruct ディレクトリーの XML ファイルを安全に変更できます。XSD 生成ツールを再実行すると、これらの更新が自動的に取得されます。

カスタム XML に関連付けられた xmlstruct ディレクトリーの適切な XML ファイルは、ファイル名で識別されます。カスタム XML は、オプションの接頭辞で始まり、下線が続き、次にベース・ファイル名が続きます。例えば、Custom_File_YFS_getOrderDetails_input.xml というカスタム XML ファイルは、xmlstruct ディレクトリーの YFS_getOrderDetails_input.xml ファイルを表します。

ただし、この命名規則はオプションです。例えば、カスタム XML sampleCustomApi.xml も命名できますが、ベース・ファイルを使用していません。この場合、ツールは、ベース XML がないことを示す情報メッセージを出力します。

注: 変換にベース XML ファイルを使用する場合、カスタム XML の命名規則に適切に接尾辞を付ける必要があります。例えば、Custom_File_YFS_getOrderDetails_input.xml は、YFS_getOrderDetails_input.xml というベース・ファイルを使用します。

生成された XSD は、以下に示すようにターゲット名前空間を指定します。

```
<xsd:schema attributeFormDefault="unqualified"
elementFormDefault="qualified"
targetNamespace="http://www.sterlingcommerce.com/documentation"
xmlns:xsd=http://www.w3.org/2001/XMLSchema
xmlns:yfc="http://www.sterlingcommerce.com/documentation">
```

この名前空間は、入力 XML のルート要素の xmlns 属性から取得され、デフォルトで http://www.sterlingcommerce.com/documentation になります。

XSD ファイルおよび DTD ファイルには、入力 XML のルート要素で QryTypeSupported="Y" が設定された場合、リスト API で使用されるクエリー・タイプ属性が含まれます。同様に、getItemList() API と getOrganizationList() API に対して定義された複雑なクエリー・タイプは、ComplexQuerySupported="Y" が設定された場合、XSD ファイルと DTD ファイルで表されます。

ただし、API では、これらの制約を純粋な DTD または XSD (あるいは、これら両方) で表すことができないため、以下の例外が DTD に表示されます。

- XML に複数の Extn 属性が含まれている場合、生成される DTD のみ (生成される XSD は含まれない) がすべての可能な Extn 要素の和集合として表示される 1 つの Extn 要素を定義します。
- 条件付き必須属性。例えば、属性のグループまたは属性の別のグループ (OrderHeaderKey や EnterpriseCode/OrderNo など) を指定する必要があります。
- ノードの必須条件は、一部の属性値によって異なります。例えば、createOrder() API では、DraftOrderFlag="N" の場合、OrderLine ノードは必須です。

複雑なクエリーの定義

複雑なクエリーは、API からの出力として取得される詳細リストを絞るのに役立ちます。希望の出力を生成するために、API の入力 XML に And 演算子または Or 演算子を使用して、クエリーを渡すことができます。

例えば、複雑なクエリー演算子 (And または Or) を使用して、計測単位、アイテム・グループ・コードなどの API に用意されているパラメーターに基づいて、getItemList API をクエリーできます。

複雑なクエリーは以下の API でサポートされています。

- deletePricelistAssignmentList
- deletePricingRuleAssignmentList
- getAttributeAllowedValueList
- getClassificationPurposeList
- getCustomerContactList
- getInventoryReservationList
- getItemList
- getOrderLineList
- getOrderList
- getOrganizationList
- getSearchIndexTriggerList
- getShipmentList

注: アイテム、組織、オーダー、オーダー明細、出荷、荷の明細の各エンティティーのみが複雑なクエリーの実行に対してサポートされています。複雑なクエリーの属性は、これらのエンティティーの有効なデータベース列に直接マッピングする必要があり、また、同じ XML 要素内にある必要があります。

これらの API について詳しくは、Javadocs を参照してください。有効なデータベース列について詳しくは、Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales ERD を参照してください。

例: getItemList API への複雑なクエリーの追加

getItemList API に複雑なクエリーを追加する以下のシナリオについて考えてみます。

getItemList API は、入力 XML で指定する選択条件 (アイテムの属性、エイリアス、カテゴリーなど) に基づいてアイテムのリストを返します。以下の例に示すように getItemList 入力 XML で複雑なクエリーを作成できます。

```
<Item OrganizationCode="DEFAULT" ItemGroupCode="PS" >
  <PrimaryInformation PricingQuantityStrategy="IQTY">
    <ComplexQuery Operator="OR">
       <And>
          <Or>
            <Exp Name="UnitOfMeasure" QryType="ISNULL"/>
            <Exp Name="UnitOfMeasure" Value="HR" QryType="FLIKE"/>
          </Or>
        <And>
             <Exp Name="ManufacturerName" Value="XYZ"/>
        </And>
        </And>
    </ComplexQuery>
  </PrimaryInformation>
</Item>
```

この例では、OrganizationCode および ItemGroupCode は、<Item> 要素の 2 つの属性で、PricingQuantityStrategy は、<PrimaryInformation> 要素の属性です。ただし、getItemList API に任意またはすべての属性を含めることができます。この API のすべての属性は、複雑なクエリー演算子とともに暗黙的な And として解釈されます。

複雑なクエリーを含める場合、以下のルールを適用します。

- 1 つの要素に対して定義できる ComplexQuery は 1 つのみです。例えば、Item 要素に 2 つの ComplexQuery 演算子を使用することはできません。
- 2 つの異なるテーブルに 1 つの複雑なクエリーを追加できません。例えば、getShipmentList API で、同じクエリーに ChainedFromOrderHeaderKey と ShipmentLineNo を使用できません。これは、前者が YFS_ORDER_LINE テーブルに属していて、後者が YFS_SHIPMENT_LINE テーブルの属性であるためです。
- 値のない属性は、複雑なクエリーでは考慮されません (Attribute="" など)。
- QryType が付加された属性の場合、以下のテーブルからクエリー・タイプ値を指定します。これは、大/小文字の区別があります。

QryType 属性の値は、フィールドのデータ・タイプによってさまざまです。以下のテーブルは、各データ・タイプに対してリスト API が使用する、サポートされるクエリー・タイプ値をリストしています。

フィールド・データ・タイプ	サポートされるクエリー・タイプ値
Char/VarChar2	• EQ - 等しい • FLIKE - 次から始まる • LIKE - 含む • GT - より後 • LT - より前
数値	• BETWEEN - 値の範囲 • EQ - 等しい • GE - 以降 • GT - より後 • LE - 以前 • LT - より前 • NE - 等しくない
日付	• DATERANGE - 日付範囲 • EQ - 等しい • GE - 以降 • GT - より後 • LE - 以前 • LT - より前 • NE - 等しくない
日時	• BETWEEN - 日付範囲 • EQ - 等しい • GE - 以降 • GT - より後 • LE - 以前 • LT - より前 • NE - 等しくない
NULL	• ISNULL - NULL のレコードを返します。 • NOTNULL - NULL 以外のレコードを返します。 **注:** これら 2 つのクエリー・タイプは、列または属性がエンティティー XML で NULL 可能に設定されている場合に使用されます。

- ComplexQuery つまり `And` または `Or` には 1 つの要素のみを使用できます。
- `And` 要素または `Or` 要素は、必要に応じて 1 つまたは多数の子要素を持つことができます。
- `And` 要素または `Or` 要素は、子要素として他の `And` 式要素または `Or` 式要素を持つことができます。

この例は、以下の論理式として解釈できます。

```
(OrganizationCode="DEFAULT" AND ItemGroupCode="PS") AND
((PricingQuantityStrategy="IQTY") OR ( ( UnitOfMeasure = "EACH"
OR UnitOfMeasure="HR" ) AND ( ManufacturerName = "XYZ") ))
```

上記の例に従うことによって、複雑なクエリーを含めて、上述の API を使用して、データベースから希望の結果を取得できます。

第 7 章 拡張 API の作成

拡張 API の呼び出し

このタスクについて

拡張 API は、ユーザーが提供する API で、カスタム API と呼ばれることもあります。拡張 API を使用して、Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales API またはサード・パーティー API を呼び出したり、Service Definition Framework でカスタム処理を実行したりすることができます。

拡張 API を呼び出す手順は、次のとおりです。

手順

1. クラスをコード化します。
2. YFSEnvironment と Document の 2 つのタイプのパラメーターのみを持っている関数をコード化し、この関数が文書を返すことを確認します。

   ```
   public Document <method-name> (YFSEnvironment env, Document doc)
   ```

3. API ノードを含むサービスを構成します。API ノードを構成するとき、以下のテーブルで説明されているプロパティーを使用します。

プロパティー	説明
「一般」タブ	
「拡張 API」	カスタム API を呼び出す場合、このオプションを選択します。
「API 名」	呼び出される API を選択または入力します。 **注:** このフィールドは、統合の目的にのみ使用されます。
「クラス名」	最初の手順でコード化したクラスを指定します。
「メソッド名」	前の手順でコード化したときに呼び出される関数を指定します。
「引数」タブ	
「引数名」	「引数」タブに値を入力することで、名前と値のペアを API に渡すことができます。 カスタム API がカスタム値にアクセスするには、カスタム API は com.yantra.interop.japi.YIFCustomApi インターフェースを実装する必要があります。 これらの名前と値のペアは、入力されるとプロパティー・オブジェクトとしてカスタム API に渡されます。
「引数値」	引数値を入力します。
「テンプレート」タブ	

プロパティー	説明
「XML テンプレート」	このラジオ・ボタンを選択して、API 出力に使用される XML を構成します。テンプレートのルート要素名を入力し、「OK」をクリックします。これで、XML を構成できます。
「ファイル名」	このラジオ・ボタンを選択して、API 出力テンプレートとして使用される XML ファイルのファイル名を入力します。このファイルは、ご使用の CLASSPATH にも存在している必要があります。
「ファクト」タブ 「ファクト」タブを使用して、データベースまたはカスタム API の「ファクト検索 (Fact Lookup)」を構成できます。ファクト検索の名前と値のペアを定義できます。値は XML パスにできます。	
「ファクト名」	ファクト名を入力します。
「ファクト値」	ファクト値を入力します。

サービス内でノードを接続する場合、次の API ノード接続プロパティー・テーブルにリストされているような API ノード接続プロパティーに注意してください。

接続	ノード接続のルール
開始ノード後の最初のノードにすることが可能	同期呼び出しされるサービス用のみ
次のノードの前に配置可能	• FTP またはファイル入出力を除く任意のトランスポート・ノード • その他のコンポーネント・ノード
次のノードの後に配置可能	• 開始ノード • FTP またはファイル入出力を除く任意のトランスポート・ノード • その他のコンポーネント・ノード
データを未変更で受け渡し	あり

4. クラスが Service Definition Framework の CLASSPATH にあることを確認します。
5. 2 つのパラメーター `YFSEnvironment` と `Document` のみを取るシグニチャーを備えたメソッドをクラスが実装することを確認します。

   以下の例は、クラスの実装方法を示します。

```
import com.yantra.yfs.japi.YFSEnvironment;
import org.w3c.dom.Document;
public class Bar {
  public Bar () {
    }
  public Document foo(YFSEnvironment env, Document doc)
    {
    //write your implementation code here
    }
}
```

6. 作成した拡張 API にアクセスするには、拡張 API を含むサービスを呼び出します。

   カスタム API を構成したときに指定したアクセス・プロパティーにアクセスする方法の詳細およびサンプル・コードについては、Javadocs の YIFCustomAPI インターフェースを参照してください。

エラー・シーケンス外部プログラムの実装

Service Definition Framework を構成して、API が属している例外グループの以前のエラーをチェックする外部プログラムを呼び出すことができます。この外部プログラムは、メッセージの処理を開始する前に呼び出されます。Java インターフェースは、この実装のために提供されます。このインターフェース定義は、com.yantra.interop.japi.YIFErrorSequenceUE クラスにあります。この外部プログラムは、ユーザー定義のカスタム・コードに基づいてメッセージ・キーを計算します。

YIFErrorSequenceUE は 2 つの関数を定義します。この関数の定義は以下のとおりです。

```
1) public Document getExceptionGroupReference(Document document, String apiName)
throws Exception
2) public void setExceptionGroupFinder (YIFExceptionGroupFinder finder)
```

getExceptionGroupReference() 関数は 2 つのパラメーターを取ります。

- Document - Integration Adapter によって取り出される入力 XML 文書
- String - Integration Adapter が XML を取り出すための API

setExceptionGroupFinder() 関数は、YIFExceptionGroupFinder() インターフェースを設定します。以前のエラーが存在する場合、このインターフェースの実装を使用して、exceptionGroupId を取り出します。

この関数の実装例は、以下のとおりです。

```
public void setExceptionGroupFinder (YIFExceptionGroupFinder finder){
   this.finder = finder;
}
```

YIFExceptionGroupFinder インターフェースの実装

このインターフェースは、入力パラメーターとして Document で取得される findExistingError() 関数を定義します。

例えば、外部プログラムが findExistingError() 関数に渡す入力 XML 文書には以下のコードが含まれています。

```
<?xml version="1.0"?>
<ExceptionGroupReference messageKey="xyz"/>
```

拡張 API の例外処理

クライアントは、常に、例外発生時にこれを処理する代わりに Service Definition Framework に例外をスローするというオプションを使用できます。構成に応じて、Service Definition Framework は、例外を警告コンソールに送信するか、または例外をログに記録します。

拡張 API のレコードのロック

Service Definition Framework (SDF) でカスタム・エンティティー API を実行するときに、カスタム・テーブルのレコードをロックできます。レコードをロックするには、SelectMethod 属性を入力 XML の一部としてカスタム・エンティティー API に渡す必要があります。ロックは、カスタム API 呼び出しのトランザクション境界内で発生します。

SelectMethod 属性の値によって、使用されるロックのタイプ (ある場合) が決まります。SelectMethod 属性として、以下の値のいずれかを渡すことができます。

- WAIT — レコードは SELECT FOR UPDATE 命令に対してロックされます。
- NO_WAIT — レコードは、SELECT FOR UPDATE NOWAIT 命令に対してロックされます。
- NONE — ロック機構は使用されません。

注: SelectMethod 属性に上記以外の値を渡すと、「SelectMethod」属性の値が有効ではないことを示すエラーがスローされます。

注: SelectMethod 属性が存在しないか、または入力 XML で NONE に設定されている場合、ロック機構は使用されません。

第 8 章 API およびサービスの起動

クライアント環境からの API の呼び出し

クライアントから標準 API を呼び出すためには、クライアント環境が正しく設定されている必要があります。クライアント環境には、基本構成に関するこのセクションの説明のように、適切な CLASSPATH 設定および JAR ファイルがある必要があります。SSL、JNDI などのセキュリティー要件は、構成に影響する場合があります。

注: サーバーの初期化の前に、アプリケーション JVM でローカル API を呼び出さないことをお勧めします。また、ローカル API 呼び出しを作成している場合、YIFClientFactoryImpl.getLocalApi および api.invoke を含むブロックのローカル API 呼び出しの後に以下のコードを追加する必要があります。

```
YFCRemoteManager.setIsLocalInvocation(false);
```

INSTALL_DIR/resources/ ディレクトリーに yifclient.properties ファイルを含める必要があります。

ローカル・モードで呼び出している場合、クライアント CLASSPATH に、*INSTALL_DIR*/properties/dynamicclasspath.cfg ファイルで参照されるすべての JAR ファイルを含める必要があります。

EJB または HTTP を介して API を呼び出す場合、クライアント CLASSPATH に、*WAS_HOME*/AppClient/properties ディレクトリーの以下のファイルを含める必要があります。

- xapi.jar
- log4j-1.2.15.jar
- platform_afc.jar
- xercesImpl.jar
- xml-apis.jar
- ejbstubs.jar (または EJB スタブを含む同等の .jar ファイル)

クライアント CLASSPATH に、*INSTALL_DIR*/jar/: の以下のファイルも含める必要があります。

- install_foundation.jar
- smcfs/9.1/smcfsshared.jar
- platform_afc_demo.jar (デモンストレーション・アプリケーション用)

アプリケーション・サーバー固有のファイル

上記のファイルの他に、以下のファイルも必要です。

アプリケーション・サーバー	必須ファイル
IBM WebSphere	WebSphere の場合、EAR ファイルにある *ejb.jar を使用して、ejbstubs.jar を取り出します。 • j2ee.jar • com.ibm.ws.ejb.thinclient_*version*.jar • com.ibm.ws.orb_7.0.0.jar • com.ibm.ws.sib.client.thin.jms_*version*.jar • com.ibm.ws.wcom.jar
Oracle WebLogic	• wlfullclient.jar • その他の CLASSPATH 要件については、アプリケーション・サーバーの文書を参照してください。
JBoss	CLASSPATH には、以下の jar ファイルのエントリーを含める必要があります。 • log4j-1.2.15.jar • jboss-j2ee.jar • jnpserver.jar • jboss-common-client.jar • concurrent.jar • jboss.jar • jboss-serialization.jar • jboss-remoting.jar • jbosssx.jar • jboss-transaction.jar

サービスおよび標準 API のプログラムでの呼び出し

Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales は、アプリケーションの標準 API およびサービスをプログラムで呼び出す方法を例示するサンプル・コードが用意されています。*INSTALL_DIR*/xapidocs/code_examples/ ディレクトリーにあるサンプル・ファイルを参照してください。

注: YIFApi インターフェースの executeFlow() メソッドを使用して、Service Definition Framework 内で定義されたサービスを実行します。

アプリケーションから発信される API およびサービス・トランザクションは、Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales: *構成ガイド アプリケーション・プラットフォーム構成ガイド*の説明に従って、Service Builder を使用して構成できます。

Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales に到着する API およびサービス・トランザクションは、以下のプロトコルを使用して呼び出すことができます。

- EJB
- HTTP および HTTPS
- LOCAL
- Web サービス
- COM+

EJB

コードのサーバー・サイドでの実行に EJB を使用します。Java で呼び出します。Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales のすべてのメソッドは、YFS 環境および文書を取り、文書を返します。EJB は、リモートに呼び出されるように設計されているため、これらの文書はそれぞれ一方の端でシリアル番号付けされ、他方の端でシリアル番号が解除されます。ただし、アプリケーションは EJB を使用し、そこで各 API は 2 つのストリング・パラメーターを取り、ストリングを返します。こうすることによって、標準の適切に定義されたインターフェースを使用して、任意の文書実装を強制的にシリアライズおよびシリアライズ解除します。

例えば、新しい EJB は、以下のようなメソッド・シグニチャーを使用して作成されます。

```
String createOrder(String env, String inputXML) throws YFSException, RemoteException;
```

ここで、env は、createEnvironment 変数への有効な入力である必要がある XML です。戻り値は出力 XML です。

YIFClientFactory.getInstance().getApi("EJB") を使用して API を呼び出すとき、この呼び出しは、このストリング・ベースの EJB を使用して行われます。このタイプの呼び出しでは、YFS 環境および文書を渡すことができ、文書が返されます。Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales のコードは、変換を透過的に実行します。

注: DOM ベースの EJB は非推奨になりました。したがって、サーバー・サイドの実行にはストリング・ベースの EJB を使用する必要があります。

HTTP

コードのサーバー・サイドでの実行に HTTP を使用します。Java で呼び出します。

LOCAL

コードのクライアント・サイドでの実行にローカルを使用します。COM または Java で呼び出します。

Web サービス

コードのクライアント・サイドでの実行に Web サービスを使用します。COM または Java で呼び出します。

COM+

VB または C++ コードのクライアント・サイドでの実行に COM を使用します。COM または Java で呼び出します。

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。

注: EJB、COM、または HTTP トランスポート・プロトコルを使用した同期 API 呼び出しの作成が再処理のキューに入れられていない場合、例外が発生します。

サービス呼び出しの構成

このタスクについて

サービス呼び出しを構成する手順は、以下のとおりです。

手順

1. *INSTALL_DIR*/resources/yifclient.properties.in ファイルの名前を *INSTALL_DIR*/resources/yifclient.properties に変更します。
2. CLASSPATH に以下のファイルが含まれていることを確認します。
   - log4j-1.2.15.jar
   - xercesImpl.jar
   - install_foundation.jar
   - platform_afc.jar
   - resources.jar
   - entities.jar
   - xapi.jar
   - ご使用のアプリケーション・サーバーで必要な JAR
   - 外部プログラムおよびカスタム API で必要な JAR
3. Java コマンド・ライン・プロパティーを次のように設定します。

   ```
   -Dlog4j.configuration=resources/log4jconfig.xml
   ```

4. *INSTALL_DIR* ディレクトリーがご使用の CLASSPATH にあることを確認します。
5. log4jconfig.xml ファイルの log4j プロパティーをご使用の環境に適した値に設定します。これらのプロパティーが正しく指定されていない場合、Service Definition Framework は正しく初期化しません。
   - EJB プロトコルおよび Oracle WebLogic を使用している場合、`weblogic.jar` がご使用の CLASSPATH 環境変数に含まれていることを確認します。また、`xercesImpl.jar` および `xalan.jar` は、ご使用の CLASSPATH で `weblogic.jar` より前にある必要があります。
   - EJB プロトコルおよび JBoss を使用している場合、*JBOSS_HOME*/client/jbossall-client.jar がご使用の CLASSPATH 環境変数に含まれていることを確認します。
   - EJB プロトコルおよび IBM WebSphere を使用している場合、CLASSPATH 環境変数に必要な JAR ファイルが含まれていることを確認します。

WebSphere JAR ファイルについて詳しくは、IBM の文書を参照してください。CLASSPATH 環境変数に適切なプロパティー・ディレクトリーが含まれていることを確認します。

注: サーバーが実行されているマシンからサービスまたは API を呼び出している場合、CLASSPATH 環境変数に *WAS_HOME*/AppServer/properties/ ディレクトリーが含まれていることを確認します。

異なるマシンからサービスまたは API を呼び出している場合、CLASSPATH 環境変数に *WAS_HOME*/AppClient/properties/ ディレクトリーが含まれていることを確認します。

- COM+ プロトコル呼び出しを構成している場合、必要な以下の COM シグニチャーのいずれかを使用します。

```
createEnvironment(VARIANT *lEnvHandle, BSTR sProgID,
BSTR sUserID, int *iRetval)
```

標準 API を呼び出すためのシグニチャーは次のとおりです。

```
 <SterlingAPI>(VARIANT *lEnvHandle, BSTR inXML,
VARIANT *outXML, VARIANT *errXML, int *retval)
```

サービスを呼び出すためのシグニチャーは次のとおりです。

```
 executeFlow(VARIANT *lEnvHandle, BSTR flowName,
BSTR flowMsg, VARIANT *outXML, VARIANT *errXML, int *retval)
```

VB コードの例については、*INSTALL_DIR*/xapidocs/code_examples/complus ディレクトリーにあるサンプルを参照してください。

API 呼び出しの特定のサーバーへの指定

このタスクについて

このアプリケーションには、カスタム API をリモートまたはローカルのいずれかで呼び出したときに、これらの API 呼び出しを特定のサーバーまたはサーバーのグループに経路指定する機能が用意されています。

この機能を使用可能にするには、サーバーおよびプロトコルを yifclient.properties ファイルおよび必要な API で指定する必要があります。エンドポイントは、API 呼び出しを経路指定するために使用される構成済みのサーバーまたはプロトコルです。

XAPI クライアントが複数の URL に対して構成されている場合、接続の試行はグループ内の各サーバーに対して順番に行われます。接続できたが別の何かが失敗した場合、他のサーバーへの接続試行は行われず、例外がスローされます。

API 呼び出しを特定のサーバーまたはサーバーのグループに指定するには、以下の手順を実行します。

手順

1. ディレクトリー *INSTALL_DIR*/resources/ の yifclient.properties ファイルで `endpoint` 属性を指定します。以下の形式でのエンドポイントの宣言および使用法を含むように yifclient.properties ファイルを変更します。

```
endpoint.Server_Name.apifactory.protocol=HTTP
endpoint.Server_Name.httpapi.url=http://server:port/context_root/
interop/InteropHttpServlet
```

複数の URL に対して XAPI クライアントを構成するには、グループをコンマ区切りリストとして定義します。

```
endpoint.Server_Name.httpapi.url=http://server1:port1/context_root1/
interop/InteropHttpServlet,http://server2:port2/context_root2/
interop/InteropHttpServlet...
```

式 endpoint.*Server_Name* によって、名前 *Server_Name* のサーバーが指定されます。例えば、endpoint.INBOXSERVER は、名前 INBOXSERVER のエンドポイントを作成します。プロパティーをエンドポイント名に割り当てて、他のすべての割り当て済みプロパティーより優先することができます。

エンドポイントに構成されたプロトコルは、次の行で指定されているように HTTP です。endpoint.*Server_Name*.apifactory.protocol=HTTP

2 行目で指定された設定 endpoint.*Server_Name*.httpapi.url は、エンドポイントで指定されたサーバーへの接続に使用されます。

2. 指定のサーバーへの接続に使用されるプロトコルを構成します。
HTTP、HTTPS、EJB、LOCAL、AUTO などのプロトコルは、エンドポイント名として予約されています。これらのいずれかをエンドポイントに構成すると、システムは、(規定どおりに) API 呼び出しの経路指定にデフォルトの接続設定を使用します。
3. 各 API に対して、使用するエンドポイントを指定します。

```
yfs.api.apiname.endpoint=ENDPOINT
```

API は、エンドポイント属性で指定されたサーバーを呼び出します。

注: エンドポイントが API に対して構成されていない場合、API は、デフォルト (ローカル) サーバーまたはすべての API に対して構成されている一般的なサーバーを使用します。API レベルで設定されたプロパティーは、他の一般的なプロパティーより優先されます。

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