ORACLE®

ATG WEB COMMERCE

バージョン 11.1

ストア設定 Core Commerce ガイド

ATG
One Main Street
Cambridge, MA 02142
USA

ストア設定 Core Commerce ガイド

ドキュメントのバージョン

Doc11 COMMSTOREv1 1/15/2011

Contact

ATG • One Main Street, Cambridge, MA 02142, USA • phone 617.386.1000 • fax 617.386.1111 • www.atg.com

目次

1 はじめに

Oracle Commerce Core Commerce アプリケーションはオンライン・ストアの基盤の役割を果たします。このアプリケーションには、製品データベース、価格設定、在庫、フルフィルメント、マーチャンダイジング、ターゲット設定されたプロモーションおよびカスタマ・リレーションシップの管理に必要とされるすべてのものが含まれています。このガイドでは、ストア管理者、ビジネス・ユーザーおよびページ開発者が関心を持つ Core Commerce の概念を説明します。詳細は、『ATG Web Commerce Programming Guide』を参照してください。

この章は次の項から構成されています。

Core Commerce の概要

この項では、Core Commerce の主な特徴を説明します。

- 製品カタログ
- 購買サービスとフルフィルメント・サービス
- ターゲット設定されたプロモーション
- Core Commerce のサービス
- ポータル・ギア
- レポート
- 複数サイトの統合
- リファレンス・アプリケーション

製品カタログ

製品カタログとは、コマース・サイトの組織的なフレームワークとなるリポジトリ項目（カテゴリ、製品、メディアなど）のコレクションです。Core Commerce には、ユーザーが使用したり、必要に応じて拡張できるカタログ実装が含まれています。

ATG Control Center（ACC）を使用して、すべてのリポジトリ項目を作成および編集できます。ATG Control Center では、リポジトリ項目を表示するためのページ・テンプレートも作成できます。閲覧者に合わせて異なるバージョンの製品カタログを表示できます。たとえば、法人顧客が自社の従業員に対して特定の品目の注文しか許可したくない場合、ストア側は、その会社の従業員が表示および注文できる品目をその製品に限定できます。様々な記憶容量のハードディスクを組み込んで購入できるコンピュータのように、製品に含まれる部品を変更できるように商品を構成することもできます。

購買サービスとフルフィルメント・サービス

Core Commerce は、ショッピング・カートへの品目の追加、顧客の希望する方法で品目を出荷するための手配およびクレジット・カード情報の認証など、精算前のオーダー処理タスクを扱うためのツールを備えています。システムは、柔軟性が高く、簡単にカスタマイズできるように設計されています。1 人のユーザーが複数のショッピング・カート、複数の支払方法、複数の送り先住所を指定できるサイトを作成できます。

顧客がオーダーを発行すると、ただちにフルフィルメント・フレームワークが処理を引き継ぎます。このシステムには、オーダー・フルフィルメント・プロセスを調整および実行する標準サービスのコレクションが含まれています。購買プロセスと同様に、フルフィルメント・フレームワークもサイトのニーズに合わせてカスタマイズできます。

Core Commerce には、次の目的に使用できる HTML ベースの Fulfillment Administration ページも含まれています。

- 出荷準備が完了しているオーダーの表示
- オーダーが顧客に出荷されたことのフルフィルメント・システムへの通知
- 出荷グループが変更され、再処理が必要であることのフルフィルメント・システムへの通知
- オーダー情報の印刷

フルフィルメント・フレームワークは次の機能を備えています。

- コスト・センター - コスト・センターを使用すると、顧客は、自社の組織の一部をコスト・センターとして指定することによって内部コストを追跡し、部署別のコストを追跡したり、コスト関連のレポートを実行できます。
- XML 経由でのオーダーのエクスポート - Core Commerce のクラスを使用して、顧客のオーダーを XML でエクスポートし、他のシステムと簡単に統合できます。
- 定期オーダー - 顧客は、新規オーダーまたは既存のオーダーに基づいてテンプレート・オーダーを作成し、顧客の指定した時間枠の間、定期的に同じオーダーを発行するスケジュールを作成できます。たとえば、企業は、翌年の間、特定の用品を毎月購入する定期オーダーを設定し、それを停止して自社のニーズを洗い直し、標準のオーダーを変更できます。
- オーダーの承認 - オーダーを承認または否認する権限を持った人による顧客のオーダーの審査を必須にできます。Core Commerce の承認プロセスでは、承認を必要とする顧客を識別し、オーダー制限を超えたなど、オーダーの承認をトリガーする条件の有無を確認できます。承認者がオーダーを審査した後、オーダーが承認されれば、そのオーダーは精算処理に進みます。
- 請求書の発行 - この機能を利用すれば、顧客は、自分が発行したオーダーに対する請求書を受け取ることができます。
- 購買依頼 - 購買依頼はオーダー承認プロセスと連携して機能し、顧客が購買依頼番号をオーダーに添付してオーダーを発行し、顧客の組織内で承認を得られるようにすることで、顧客が内部アクティビティを追跡できるようになります。

ターゲット設定されたプロモーション

ビジネス・マネージャは、Core Commerce のプロモーションを使用して、顧客に購買を促す手段として製品にスポットライトを当て、値引きを提示できます。プロモーションは、一般的に次のカテゴリに分類されます。

- 特定の製品の定額割引
- オーダー合計額の定額割引
- 特定の製品の定率割引
- オーダー合計額の定率割引

- 属性に基づく製品の定額割引または定率割引
- 無料製品または無料オーダー
- 代替(製品 B の価格で製品 A を購入できる)
- 特定の製品の送料の無料化

このガイドで説明している ACC の簡単なインタフェースを使用して、または Oracle Commerce Merchandising(『ATG Web Commerce Merchandising Administration Guide』を参照)を使用して、プロモーションを作成できます。

Core Commerce のサービス

Core Commerce は、コマース・サイトに様々なマーチャンダイジング機能を実装するためのサービスを提供します。

- ギフト・リスト - ギフト・リストを利用して、顧客は、誕生日や結婚式などのイベントを登録し、他のサイト訪問者が閲覧できる製品のリストを作成できます。顧客はいくつでも自分用のギフト・リストを作成できます。購買プロセスの一部として、送り先住所のセキュリティ、包装、出荷など、ギフトの購入に関する特別な取扱指示を指定できます。
- ウィッシュ・リスト - ウィッシュ・リストを利用すれば、顧客は、実際に自分のショッピング・カートに品目を入れることなく、製品のリストを保存できます。ウィッシュ・リストはギフト・リストに似ていますが、ウィッシュ・リストを作成した人しかアクセスできない点がギフト・リストと異なります。顧客は、いつでも自分のウィッシュ・リストにアクセスし、ウィッシュ・リストに含まれている品目を購入できます。
- 比較リスト - 比較リストを利用すれば、顧客は、複数の製品 SKU を選択し、それらを左右に並べて比較できます。
- 商品券 - 商品券を製品カタログ内の項目として設定できます。顧客が商品券を購入すると、商品券は E メールを介して受信者に配信され、受信者はそれをサイトでの購入の支払いに充てることができます。
- クーポン - クーポンは商品券と似ていますが、特定の顧客に送信されるプロモーションの一種(たとえば、100 ドル以上のオーダーの 20%)である点が異なります。顧客は、精算プロセス時に要求コードを入力することによって商品券やクーポンを換金します。

ACC を使用してギフト・リスト、クーポン、商品券などのリポジトリ項目を管理できます。

ポータル・ギア

Core Commerce のユーザーかつ ATG Portal のユーザーであれば、コマース・サイトのポータル・ページで Core Commerce ギアを使用して、顧客に自分のオーダー情報へのパーソナライズされたゲートウェイを提供できます。

オーダー・ステータス・ギアにより、顧客は現在および過去の自分のオーダー情報にアクセスできるようになります。オーダー承認ギアは、承認者の承認を必要とするオーダーを承認者に表示し、そのオーダーを承認または否認するメカニズムを提供します。これらのポータル・ギア、「ポートレット」の詳細は、ATG ポータル・ギアの使用を参照してください。

レポート

Core Commerce は Oracle Commerce Business Intelligence と完全に統合されており、ストアのパフォーマンスに関する重要な情報を提供できるデフォルトのレポート・セットを備えています。これらのレポートの詳細は、『ATG Web Commerce Reports Guide』を参照してください。

複数サイトの統合

Core Commerce の複数サイト機能を利用すれば、新しいサイトをすばやく構築し、立ち上げて、ブランドや地方の雑貨店、その他の差別化要因を複数のチャネルにわたって効率的に管理できます。この項では、Core Commerce アプリケーションで重要な意味を持つ複数サイトの一部の側面について説明します。

- サイト・コンテキスト - サイト・コンテキストは、ユーザーのセッション内で、どのカタログ、製品または SKU をユーザーが使用できるか、どの価格表がユーザーに適用されるか、どのショッピング・カートをユーザーが使用するかを識別します。
- サイト・メンバーシップ - カタログとカタログの項目が属するサイトを定義します。これらの項目には、カタログ、カテゴリ、製品、SKU、カタログ・フォルダが含まれます。カタログとその他の項目は複数のサイトに属することができます。
- `SiteIdForItemDroplet` および `SiteLinkDroplet` - これらのプラットフォーム・ドロップレット(『ATG Web Commerce Page Developer's Guide』を参照)は Core Commerce 開発者にとって便利です。複数のカタログに表示される項目をまとめて表示できます。
- ショッピング・カート - カートは(顧客が最初の品目を追加したときに)カートが作成されたサイト、個々の品目が追加されたサイト、最新のアクティビティが発生したサイトを追跡します。
- 定期オーダー - 定期オーダーにはオーダーの作成時および価格設定時のサイト情報が含まれています。
- ギフト、購買およびウィッシュ・リスト - これらすべては、これらが作成され、個々の品目が追加されたサイトを追跡します。
- 検索 - 検索フォーム・ハンドラはサイトを認識し、サイトの制約を受けることがあります。
- レポート - すべてのレポートにサイト情報が含まれています。『ATG Web Commerce Reports Guide』を参照してください。

複数サイトでの Core Commerce 機能の使用に関する情報は、このガイドの該当する様々な箇所に記載されています。Oracle Commerce Platform アプリケーションでの複数サイトの実装に関する一般情報は、『ATG Web Commerce Multisite Administration Guide』を参照してください。

リファレンス・アプリケーション

Oracle Commerce Platform には、Web サイトで Core Commerce 機能を使用する方法を説明するリファレンス・アプリケーションが付属しています。

- ATG Commerce 参照ストア・インストレーションおよび構成ガイドおよび関連するドキュメントで説明している、Oracle Commerce 参照ストア
- 『ATG Web Commerce ビジネス・コマース・リファレンス・アプリケーション・ガイド』で説明している Business Commerce リファレンス・アプリケーション(Motorprise)
- 「サンプル・カタログについて」で説明しているサンプル・カタログ

サンプル・カタログについて

Commerce サンプル・カタログは、必要最低限の機能を持ったコマース・サイトを構成するサンプル JSP のセットです。サイトを開発する過程で、サンプル・カタログを参照し、よく使用される次のような Core Commerce 機能の使用方法を示す簡単なコード例を参考にできます。

- 動的な価格設定と在庫の使用
- 製品カタログ内のナビゲーション
- 製品カタログの検索
- ショッピング・カートまたはギフト・リストへの品目の追加

- 単一の出荷グループおよび支払グループを使用したオーダーの精算
- 1 人のセッション内での複数ショッピング・カートの管理

さらに、サンプル・カタログ JSP をユーザー独自の JSP テンプレートの基点として利用できます。

Motorprise リファレンス・アプリケーションを介してサンプル・カタログにアクセスできます。サンプル・カタログにアクセスするには、アプリケーションをアセンブルするときに、DCSSampleCatalog モジュールをモジュールのリストに追加します。

```
MotorpriseJSP DCSSampleCatalog
```

MotorpriseJSP のかわりに独自の Core Commerce アプリケーションを使用できます。

注意: Oracle Commerce Platform アプリケーションのアセンブルの詳細は、『ATG Platform Programming Guide』を参照してください。

DCSSampleCatalog モジュールに含まれている sampleCatalog Web アプリケーションには、サンプル・カタログ JSP が含まれていますが、追加のコードまたは構成プロパティは含まれていません。Web アプリケーションにアクセスするには、まずアプリケーション・サーバー上の<ATG11dir>/DCSSampleCatalog/j2ee-apps/にある sampleCatalog.ear ファイルを配置し、その後、アプリケーションを配置する必要があります。サンプル・カタログ JSP は、よく使用される Core Commerce 機能の簡単な説明を主目的としていることに注意してください。すべての環境で、またはすべてのアプリケーションと連携して動作することは保証されていません。

サンプル・カタログ JSP は<ATG11dir>/DCSSampleCatalog/j2ee-apps/sampleCatalog/web-app/にあります。サンプル・カタログ・モジュールが配置されていれば、ブラウザで http://*hostname*:*port*/sample_catalog/を指定することによってページを表示できます。使用するポートは、アプリケーション・サーバー自体およびアプリケーション・サーバーがどのように構成されているかによって異なります。たとえば、JBoss の場合、デフォルト URL は次のとおりです。

```
http://hostname:8080/sample_catalog/
```

他のアプリケーション・サーバーのデフォルト・ポート情報については、『ATG Web Commerce Installation and Configuration Guide』を参照してください。

ACC の「ページおよびコンポーネント」→「J2EE ページ」領域を介してサンプル・カタログ JSP にアクセスすることもできます。「J2EE ページ」タスク領域を介して ACC の文書エディタで特定の JSP を開き、サンプル・カタログ・アプリケーションが実行されていれば、ブラウザで JSP をプレビューできます。

必要とされる資料の一覧

Core Commerce は、ビジネスの特定のニーズを満たすためにカスタマイズされたコマース Web サイトの作成に必要なツールを備えた包括的な製品です。情報を見つけるための手掛かりを次に示します。

タスク	対象者	参考資料
カタログの作成と、カタログへのカテゴリ、製品、SKU の追加。Core Commerce に標準で付属のフルフィルメント・ツールおよび在庫ツールの構成。	ビジネス・ユーザー	このガイドのカタログ管理および在庫とフルフィルメントの管理

タスク	対象者	参考資料
製品に標準で付属のプロモーション、価格表、シナリオ、放棄されたオーダー、コスト・センター・ツールに関する作業	ビジネス・ユーザー	本マニュアル(ATG Web Commerce ストア設定ガイド)
プロジェクトを使用してユーザーの実行するタスクを管理し、ユーザーの編集する Commerce アセットのバージョンを保守する公開環境でのカタログとカタログのカテゴリ、製品、SKU の開発	ビジネス・ユーザー	ATG Web Commerce マーチャンダイジング・ガイド
Commerce サーブレット Bean を使用する JSP の作成	ページ開発者	本マニュアル(ATG Web Commerce ストア設定ガイド)の複数の章
サブクラスを作成し、リポジトリを変更することによる Oracle Commerce Platform のプログラム拡張	プログラマ	ATG Web Commerce Programming Guide
Core Commerce およびリファレンス・アプリケーション・モジュールが含まれたアプリケーションのアセンブル	サイト管理者	ATG Web Commerce Platform Programming Guide
実稼働環境への Core Commerce データベースのインストール	サイト管理者	ユーザーが Merchandising を持っていない場合は、『ATG Web Commerce Programming Guide』を、Merchandising を持っている場合は、『ATG Web Commerce Merchandising Administration Guide』を参照する必要があります。
Merchandising をサポートするデータベース・テーブルのインストール	サイト管理者	ATG Web Commerce Merchandising Administration Guide
データベース・テーブル、セッション・バックアップ、JMS メッセージおよびレコーダ	サイト管理者	ATG Web Commerce Programming Guide
Motorprise リファレンス・アプリケーションに関する作業	All	ATG Web Commerce ビジネス・コマース・リファレンス・アプリケーション・ガイド
サンプル・カタログに関する作業	ページ開発者、プログラマ	この章のリファレンス・アプリケーションを参照してください。
Core Commerce レポートの表示	ビジネス・ユーザー	『ATG Web Commerce Reports Guide』を参照してください。

2 カタログ管理

製品カタログは、顧客による品目の検索と購入を可能にする、組織とナビゲーションのフレームワークを形成する互いに関連する項目から構成されます。製品カタログは、通常、カテゴリおよび製品の階層ツリーから構築されます。

この章では、ATG Control Center（ACC）を使用してコマース・サイト用のカタログを作成し、変更する方法を説明します。この章では、読者がデフォルトの Oracle Commerce Core Commerce 製品カタログとデフォルトのリポジトリ・エディタを使用していることを前提としています。製品カタログまたはリポジトリ・エディタがサイトでカスタマイズされている場合は、カタログの管理がこの章の説明とは異なる方法で行われている可能性があります。

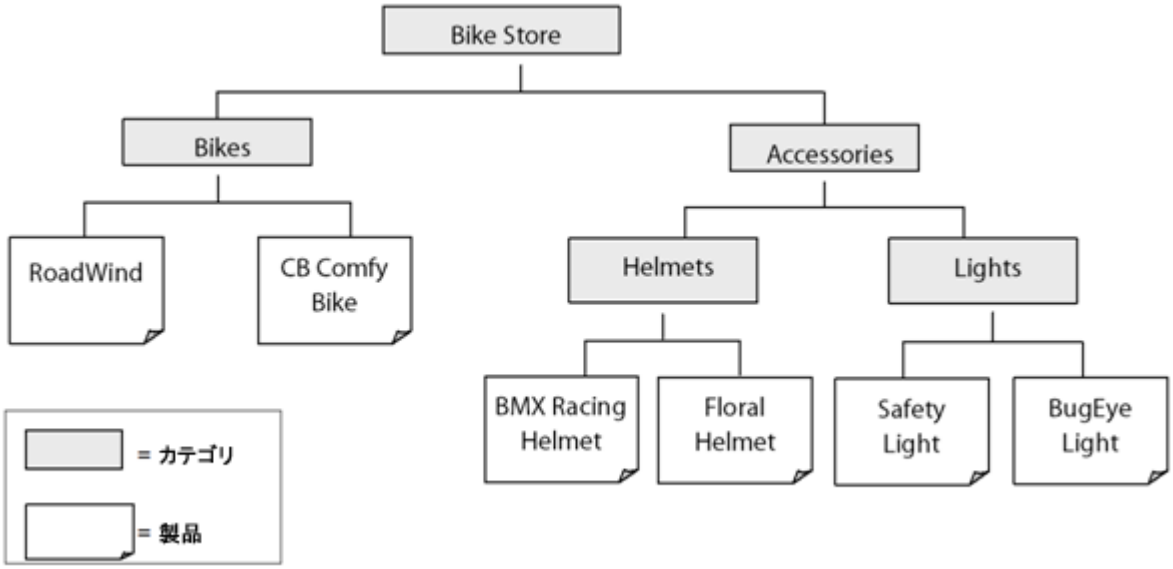
Oracle Commerce Merchandising および ATG Content Administration を持っているユーザーは、ACC ではなく、コンテンツ管理環境でカタログを作成し、カタログに項目を追加するため、そのようなユーザーに関連のあるものはこの章の一部の項のみです。最初の 2 項を読んで、これから使用する Commerce アセットおよびカタログ内で Commerce アセットを編成する方法を学習してください。Merchandising でカタログを使用する方法の詳細は、『ATG Web Commerce マーチャンダイジング・ガイド』を参照してください。

注意: ステージング・サーバーで製品カタログを変更した場合は、使用中のバージョンに変更をコピーする前に、`AncestorGeneratorService` を実行してください。『ATG Commerce Programming Guide』の*カタログの保守サービスの実行*に関する項を参照してください。

この章では、次のトピックを説明します。

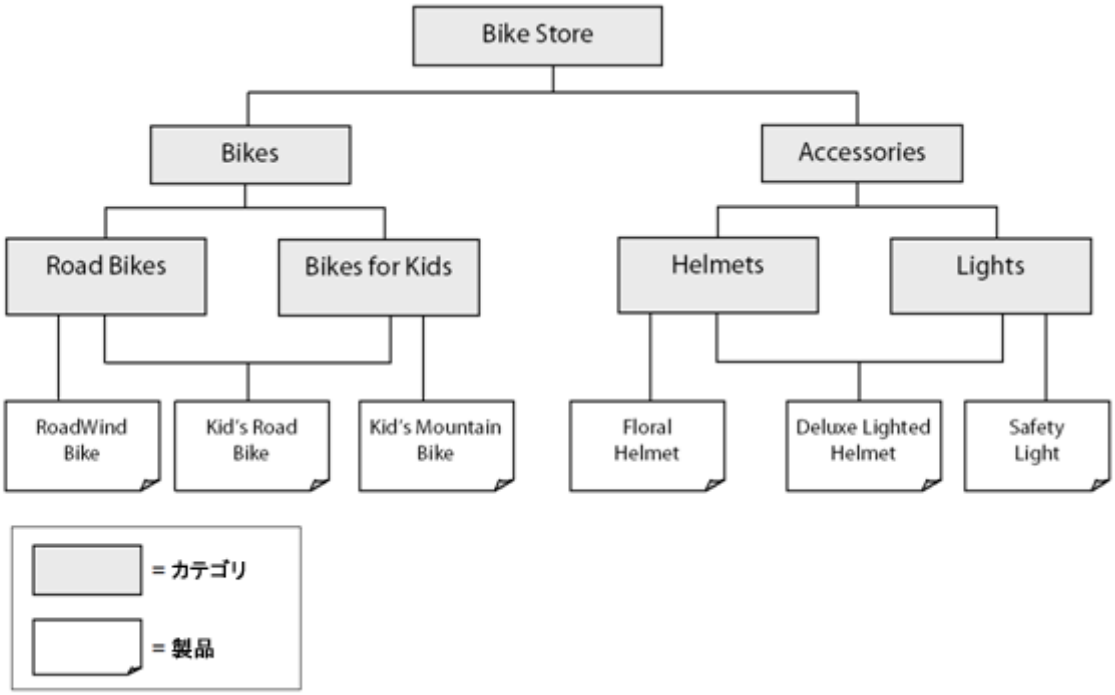
製品カタログの編成

Core Commerce の製品カタログに使用できる編成モデルはいくつかあります。たとえば、項目を単純なツリー状の構造に並べることができます。このモデルでは、次の図に示すように、個々のカテゴリに製品または他のカテゴリが子として含まれ、個々のカテゴリまたは製品は 1 つの親カテゴリを持ちます。

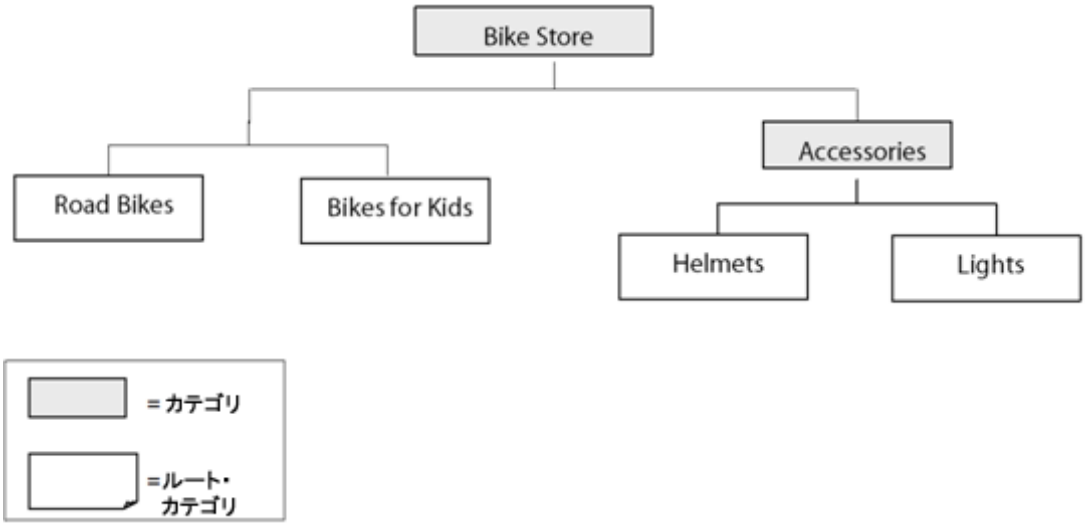


任意の製品に至るナビゲーション・パスが 1 つしかないことに注意してください。たとえば、顧客は、BMX Racing Helmet 製品に至るまでに、**Accessories** を選択し、次に **Helmets** を、その次に **BMX Racing Helmet** を選択する必要があります。

これに代わるモデルとして、特定の製品に至る複数のナビゲーション・パスを顧客に提供できます。次の図は製品が複数の親カテゴリを持つ製品カタログを示しています。



3 番目の選択肢として、ユーザーごとにまったく異なるカタログを提示できます。



Bike Store カタログにアクセスできるユーザーがルート・カテゴリを表示すると、そのユーザーには、Accessories サブカタログ内のすべてのルート・カテゴリおよび Bike Store カタログ自体のルート・カテゴリが見えます。しかし、Accessories カタログにしかアクセスできないユーザーには、Helmets ルート・カテゴリと Lights ルート・カテゴリしか見えません。

Core Commerce のカタログ項目タイプ

Core Commerce の製品カタログはリポジトリであり、カタログの要素（フォルダ、カテゴリ、製品、イメージなど）はリポジトリ項目です。製品カタログを構築するには、新しいリポジトリ項目を追加し、リポジトリ項目どうしの関係を定義します。

カタログの項目タイプは、ユーザーがショッピングするストアの様々なバージョンに相当します。カテゴリはストアの売り場に相当し、製品は販売される個々の製品に相当します。SKU は製品の様々なバージョンに相当し、販売される実際の品目です。たとえば、特定のシャツに相当する製品は、サイズや色の様々な組合せに相当する、製品に関連付けられた様々な SKU を持っています。フォルダはカタログ内の項目を編成するために使用されます。

Core Commerce は、サイトで使用される JSP テンプレート・ページおよびカテゴリ、製品または SKU とともに表示できるイメージに相当するメディア項目タイプも備えています。

カタログの表示

ACC では、グラフィカル・ユーザー・インタフェースを使用して製品カタログを管理できます。ACC を使用して、フォルダ、カタログ、カテゴリおよび製品の作成や変更、イメージのインポート、および項目の検索などのカタログ管理タスクを実行できます。この章では、ACC を使用して製品カタログを管理する方法を説明します。ACC の使用の詳細は、オンライン・ヘルプを参照してください。

階層としてのカタログの表示

個々のカタログの子カテゴリと子製品がカタログの構造を決定します。たとえば、「ベイクド食品」の子カテゴリには、「ケーキ」、「クッキー」、「パン」があり、「パン」の子製品には「ライ麦パン」、「全粒小麦粉パン」、「食パン」があります。カテゴリは、子カテゴリと子製品の両方を持っています。たとえば、「果物」は、「リンゴ」と「梨」を子製品として持ち、「かんきつ類」を子カテゴリとして持ちます。

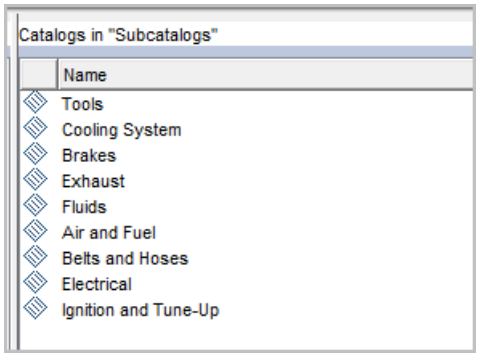
カテゴリまたは製品は複数のカテゴリの子になる可能性があることに注意してください。そのため、カタログはより柔軟性が高くなりますが、ナビゲーションが複雑になる可能性があります。顧客がカタログ階層間の移動によってではなく、検索機能を使用してカテゴリまたは製品にアクセスする場合は、特にその傾向が強くなります。顧客が上の階層に移動する場合、ストア側は移動先の親カテゴリを決定する必要があります。したがって、製品とカテゴリは、個々の項目のデフォルトの親カテゴリを指定するために使用できる親カテゴリ・プロパティを持っています。

カテゴリと製品の個々のプロパティの詳細は、『ATG Web Commerce Programming Guide』の*製品カタログの使用と拡張*を参照してください。

カタログは ACC から表示できます。製品カタログにアクセスするには

1. ATG Control Center のメイン・ナビゲーション・メニューから「カタログ管理」→「カタログ」を選択します。

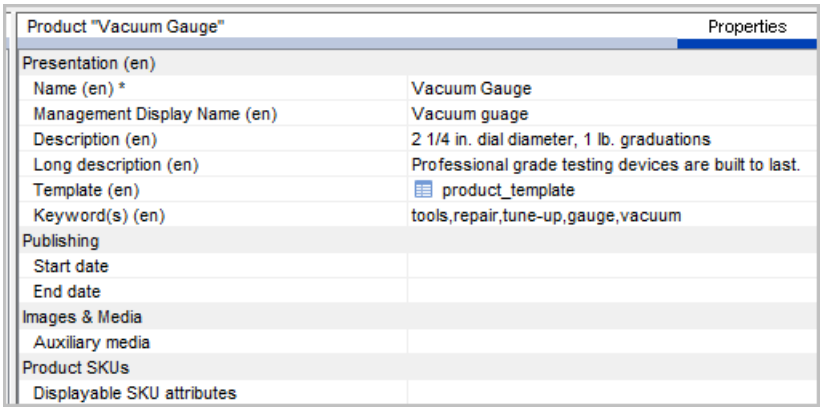
   左パネルに既存のカタログ・フォルダのリストが表示されます。フォルダを選択します。そのフォルダ内の既存のカタログが右パネルに表示されます。



2. カタログ名（Tools など）をクリックして表示します。

カタログがツリー構造として左パネルに表示されます。+記号をクリックすることによって、任意の項目を展開し、その項目の子項目を表示できます。

項目を選択すると、選択されている項目のプロパティの名前と現在の値が右パネルに表示されます。選択した項目タイプによっては、追加の情報が表示されることがあります。たとえば、製品には関連付けられている SKU のセクションと抱合わせ販売情報のセクションがある一方で、カテゴリにはプロパティと、カテゴリに関連付けられているイメージしかありません。次の図は製品のプロパティ・パネルの一部を示しています。

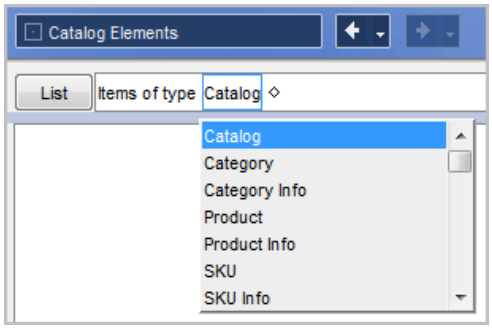


グレー表示されたプロパティは自動的に設定されるもので、読取り専用です。また、表示されるプロパティ名はプロパティの表示名です。リポジトリで使用されているプロパティの名前を表示するには、カーソルを表示名の上に置きます。プロパティ名がツール・チップとして表示されます。JSP でプロパティを参照するときは、実際のプロパティ名を使用する必要があります。

リストとしてのカタログの表示

製品カタログの非階層的なビューにもアクセスできます。

1. ATG Control Center のメイン・ナビゲーション・メニューから「カタログ管理」→「カタログ要素」を選択します。
2. 項目タイプ（Items of Type）のドロップダウン・ボックスから表示する項目タイプを選択し、「リスト」ボタンをクリックします。問合せに条件を追加するには、菱形のドロップダウン・ボックスをクリックします。



カタログ項目の作成

ACC を使用して次のカタログ項目を作成できます。

- フォルダ - 編成グループを形成する他のタイプの項目が含まれています。フォルダは ACC に表示されますが、ユーザーがカタログと相互作用する方法には影響を及ぼしません。
- カタログ - 任意の数の他のカタログまたはカテゴリが含まれています。カタログは、別のカタログの子またはカテゴリの子である可能性があります。

- カテゴリ - 1 つ以上の他のカテゴリの子または*ルート・カテゴリ*のいずれかです。ルート・カテゴリはカタログのナビゲーション構造の基点となります。
- 製品 - カタログのナビゲーションの終点です。ただし、顧客が実際に購入するのは、製品に関連付けられた SKU であり、製品そのものではりません。製品は、様々な種類、サイズ、色に相当する複数の関連付けられた SKU を持っていることがあります。

カタログ・フォルダの作成

カタログ・フォルダは ACC の備える編成ツールです。それらは顧客がカタログと相互作用する方法には影響を及ぼしません。

1. ATG Control Center のメイン・ナビゲーション・メニューから「カタログ管理」→「カタログ」を選択します。
2. 新規カタログ・フォルダを追加するフォルダを選択します
3. 「新規フォルダ」ボタンをクリックします。
4. 新規フォルダの名前を指定します。
5. 「OK」をクリックします。

カタログの作成

カタログに他のカタログまたはカテゴリを含めることができます。カタログをサブカタログとして別のカタログに追加する方法については、このガイドのカタログへのサブカタログの追加を参照してください。

新規カタログを作成するには

1. ATG Control Center のメイン・ナビゲーション・メニューから「カタログ管理」→「カタログ」を選択します。
2. 新規カタログを追加するフォルダを選択するか、「新規フォルダ」ボタンをクリックして、カタログ用の新規フォルダを追加します。
3. ACC の右上隅の「新規カタログ」ボタンをクリックします。「新規項目」ダイアログ・ボックスが表示されます。
4. 新規カタログの名前を入力するか、「OK」をクリックしてデフォルトの名前を受け入れます。
5. 「OK」をクリックします。

ルート・カテゴリの作成

ルート・カテゴリを作成するには

1. ATG Control Center のメイン・ナビゲーション・メニューから「カタログ管理」→「カタログ」を選択します。
2. カテゴリを追加するカタログが存在するフォルダを選択します。
3. カテゴリを追加するカタログをクリックします。カタログ名が左パネルへ移動します。
4. カタログを選択し、カタログ名をハイライトします。
5. 新規カテゴリを作成するには、新規カテゴリボタンをクリックします。既存のカテゴリを現在のカテゴリにルートとして追加するには、カテゴリの追加をクリックします。

6. 新規カテゴリを追加する場合は、名前フィールド（必須）およびオプション・フィールドに入力し、「OK」をクリックします。

カテゴリがカタログの rootCategories プロパティに自動的に追加されます。

子カテゴリの作成

子カテゴリを作成するには

1. ATG Control Center のメイン・ナビゲーション・メニューから「カタログ管理」→「カタログ」を選択します。
2. カテゴリを追加するカタログが存在するフォルダを選択します。
3. カテゴリを追加するカタログをクリックします。カタログ名が左パネルへ移動します。
4. 子カテゴリを追加するカテゴリを選択し、名前をハイライトします。
5. 新規カテゴリを作成するには、新規カテゴリボタンをクリックします。既存のカテゴリを現在のカテゴリに追加するには、カテゴリの追加をクリックします。
6. 新規カテゴリを追加する場合は、名前フィールド（必須）およびオプション・フィールドに入力し、「OK」をクリックします。

カテゴリが親カテゴリの子カテゴリとしてカタログに表示されます。

製品の作成

子製品を作成するには

1. ATG Control Center のメイン・ナビゲーション・メニューから「カタログ管理」→「カタログ」を選択します。
2. 製品を追加するカタログが存在するフォルダを選択します。
3. 製品を追加するカタログをクリックします。カタログ名が左パネルへ移動します。
4. 製品を追加するカテゴリを選択し、名前をハイライトします。
5. 新規製品を作成するには、新規製品ボタンをクリックします。既存の製品を現在のカテゴリに追加するには、製品の追加をクリックします。
6. 新規製品を追加する場合は、名前フィールド（必須）およびオプション・フィールドに入力し、「OK」をクリックします。

製品が親カテゴリの子製品としてカタログに表示されます。

SKU の作成

カタログ内に製品の新規 SKU を作成するには

1. ATG Control Center のメイン・ナビゲーション・メニューから「カタログ管理」→「カタログ」を選択します。
2. カタログ・ツリー上の製品を選択します。
3. ウィンドウ最上部のメニューから SKU を選択します。
4. 「SKU の追加」ボタンをクリックします。
5. ダイアログ・ボックスで、「項目タイプ」メニューから「SKU」を選択します。
6. ダイアログ・ボックスで、「新規項目」をクリックします。

7. 名前フィールド（必須）およびオプション・フィールドに入力します。
8. 「OK」をクリックします。

構成可能な SKU の作成

通常の SKU は、ミディアム・サイズのブルーのシャツなど、1 つの品目に相当しますが、構成可能な SKU は、1 つの品目として販売されてはいるものの、コンピュータ・システムや自動車のように可変コンポーネントを持っているオブジェクトに相当します。構成可能な SKU は通常の SKU と同じ方法で作成されますが、「新規項目」ダイアログ・ボックスで「項目タイプ」ドロップダウン・メニューから構成可能な SKU を選択する必要があります。

構成可能な SKU を作成するには

1. タイプが構成可能な SKU である項目を作成します。例: コンピュータ
2. タイプが「構成可能なプロパティ」である項目を必要な数だけ作成します。構成可能なプロパティとは、コンピュータの例で言えばハードディスク・ドライブや RAM など、構成可能な項目の個別の部分に相当します。
3. 構成可能なプロパティを作成するには、「カタログ管理」→「カタログ要素」へ移動します。項目タイプ（Items of Type）のドロップダウン・リストから「構成可能なプロパティ」を選択し、右上の「新規項目」ボタンをクリックします。
4. タイプが構成可能なオプションである項目を必要な数だけ作成します。構成可能なオプションとは、コンピュータの例で言えば XYZ 社製の 10GB の SCSI ハードディスク・ドライブなど、構成可能なプロパティの個別のオプションに相当します。
5. 構成可能なオプションは既存の SKU にリンクできます。既存の SKU にリンクすると、構成可能なオプションで設定される価格は SKU 価格を上書きします。
6. 構成可能なオプションを作成するには、「カタログ管理」→「カタログ要素」へ移動します。項目タイプ（Items of Type）のドロップダウン・リストから構成可能なオプションを選択し、右上の「新規項目」ボタンをクリックします。
7. 作成した構成可能なプロパティへ戻り、構成可能なオプションをプロパティに追加します。
8. 構成可能な SKU へ戻り、構成可能なプロパティを SKU に追加します。

SKU バンドルの作成

SKU バンドルとは、複数の他の SKU から構成される SKU です。バンドルはオーダーのフルフィルメントでは複数の項目として扱われますが、バンドルを利用することで項目のグループを 1 つの項目として購入できます。バンドルのコンポーネント SKU が常に同じである点で、SKU バンドルは構成可能な SKU と異なります。

SKU バンドルを作成する手順は 3 つの部分に分かれています。

1. SKU バンドルを構成する個別の SKU を作成します。
2. SKU からの SKU リンクを作成します。SKU リンクとは、個別の SKU と数量から構成される項目タイプです。
3. 1 つ以上の SKU リンクから構成されるバンドル・リンク・プロパティを持つ SKU を作成します。

たとえば、6 本の HB 鉛筆と 1 つのペンシルケースから構成される SKU バンドルを作成するとします。

1. その場合は、HB 鉛筆に相当する SKU を作成し、ペンシルケースに相当する別の SKU を作成します。

2. 6 本の HB 鉛筆 SKU に相当する SKU リンクを作成し、1 つのペンシルケース SKU に相当する別の SKU リンクを作成します。
3. これら 2 つの SKU リンクから構成される SKU を作成することによって SKU バンドルを作成します。

SKU リンクの作成

SKU リンクを作成するには

1. ATG Control Center のメイン・ナビゲーション・メニューから「カタログ管理」→「カタログ要素」を選択します。
2. 右上の「新規項目」ボタンをクリックします。「新規項目」ダイアログ・ボックスが表示されます。
3. 「項目タイプ」ドロップダウン・メニューから「SKU リンク」を選択します。
4. 表の項目フィールドを選択し、「...」ボタンをクリックします。SKU を選択するためのダイアログ・ボックスが表示されます。
5. 「リスト」ボタンをクリックして、使用可能な SKU のリストを表示します。リストから SKU を選択し、「OK」をクリックします。
6. `displayName` フィールドと `quantity` フィールド（いずれも必須）に入力します。
7. 必要な任意のオプション・フィールドに入力し、「OK」をクリックします。

SKU リンクからの SKU の作成

1 つ以上の SKU リンクを作成したら、リンクを結合する SKU を作成できます。

1. SKU の作成の説明に従って SKU を作成します。
2. バンドル・リンク・フィールドを選択し、「...」ボタンをクリックします。SKU リンクを指定するためのダイアログ・ボックスが表示されます。
3. 「追加」をクリックします。「新規項目」ダイアログ・ボックスが表示されます。
4. 「リスト」をクリックして、SKU リンクのリストを表示します。
5. リストから SKU リンクを選択します。[Ctrl]を押したまま項目を選択することによって、複数の SKU リンクを選択できます。
6. 「OK」をクリックして「新規項目」ダイアログ・ボックスを閉じた後、「OK」をクリックして「バンドル・リンク」ウィンドウを閉じます。

カタログへのサブカタログの追加

カタログを別のカタログにサブカタログとして追加するには、まずカタログに含めるサブカタログを作成する必要があります（カタログの作成を参照してください）。

既存のカタログにサブカタログを追加するには

1. ATG Control Center のメイン・ナビゲーション・メニューから「カタログ管理」→「カタログ」を選択します。
2. サブカタログを追加するカタログが存在するフォルダを選択します。
3. サブカタログを追加するカタログをクリックします。カタログ名が左パネルへ移動し、カタログのプロパティが右パネルに表示されます。
4. 「カタログの追加」ボタンをクリックして、追加できるカタログの完全なリストを表示します。
5. メイン・カタログのサブカタログとして指定するカタログを選択します。
6. 「OK」をクリックします。

カテゴリへのカタログの追加

カテゴリにはカタログおよび製品を含めることができます。既存のカテゴリにカタログを追加するには

1. ATG Control Center のメイン・ナビゲーション・メニューから「カタログ管理」→「カタログ」を選択します。
2. カタログを追加するカテゴリを選択します。
3. 「カタログの追加」ボタンをクリックして、追加できるカタログの完全なリストを表示します。
4. カテゴリに追加するカタログを選択します。
5. 「OK」をクリックします。

カタログ項目の編集

カタログ・フォルダ、カタログ、カテゴリ、製品または SKU を作成するときは、作成する項目の属性値を「新規項目」ダイアログ・ボックスで指定できます。項目を作成した後で、それらの値を変更できます。

注意: 一部のプロパティでは、値を直接入力できません。たとえば、一部のプロパティはブール値です。その場合、Core Commerce では、True または False のどちらかを選択するためのドロップダウン・メニューを表示します。プロパティが別のリポジトリ項目に相当する場合、またはリポジトリ項目のコレクションである場合も、フィールドに値を直接入力できません。そのかわり、フィールド内をクリックし、「...」ボタンをクリックしてダイアログ・ボックスを表示し、そのダイアログ・ボックスで 1 つ以上の項目を指定できます。

1. ATG Control Center のメイン・ナビゲーション・メニューから「カタログ管理」→「カタログ」を選択します。
2. 編集する項目をカタログ・ツリーから選択します。SKU を編集する場合は、製品を選択した後、ウィンドウ最上部のメニューから「SKU」を選択します。
3. メニューから「プロパティ」を選択します。
4. 属性値を直接編集できる表が ATG Control Center に表示されます。
5. 編集するプロパティの横に値を入力します。
6. カタログ項目を編集した後、「ファイル」→「保存」を選択します。

項目の移動

カテゴリまたは製品を階層上の 1 つの場所から別の場所へ移動するには、項目をカタログ・ツリー上の現在の位置から目的の位置までドラッグします。

1. ATG Control Center のメイン・ナビゲーション・メニューから「カタログ管理」→「カタログ」を選択します。
2. カタログ・ツリー上のカテゴリまたは製品を選択します。
3. カーソルを項目の上に置いて、マウスの左ボタンを押し続けます。
4. 項目を移動先カテゴリの最上部までドラッグします。
5. マウス・ボタンを離します。
6. 「ファイル」→「保存」を選択します。

[Ctrl]を押したまま項目をドラッグすると、その項目は元の親カテゴリとドラッグ先の新しいカテゴリの両方の子項目になります。この操作によって項目の新しいコピーが作成されることはありません。

項目の複製

元の項目と同じプロパティ値を持ちながら、まったく別のリポジトリ項目になる項目の複製を作成できます。それを行うには、次の手順を実行します。

1. 項目を右クリックします。
2. ポップアップ・;メニューから「複製」を選択します。

カテゴリを複製すると、複製は元のカテゴリの子カテゴリとして表示されます。製品は元の項目と同じ階層レベルに表示されます。

新規項目は、名前を含めて、元の項目と同じプロパティ値を持ちます。2 つの項目は同じプロパティを持ちますが、データベース内に別々に格納される別々のリポジトリ項目です。混同を避けるために、新規項目は別名に変更してください。

項目の削除

項目をリポジトリから削除するには

1. ATG Control Center のメイン・ナビゲーション・メニューから「カタログ管理」→「カタログ」を選択します。
2. カタログ・ツリー上の項目を右クリックします。
3. ポップアップ・;メニューから「削除」を選択します。確認のダイアログ・ボックスが表示されます。
4. 「はい」をクリックして削除を確認します。

項目をリポジトリから削除することなく、カタログ構造上の場所から削除するには

1. ATG Control Center のメイン・ナビゲーション・メニューから「カタログ管理」→「カタログ」を選択します。
2. カタログ・ツリー上の項目を右クリックします。
3. ポップアップ・メニューからリンクの削除を選択します。

オンライン専用としての項目のマーク付け

項目をカタログに追加すると、ローカル・ストアで集荷可能または集荷不可能な製品として識別できます。製品または SKU の `onlineOnly` フラグを `true` に設定した場合、ストア内引取ではその項目を利用できません。`OnlineOnlyDroplet` は製品または SKU を受け取り、`CatalogTools.isOnlineOnly` コンポーネントをコールします。詳細は、OnlineOnlyDroplet を参照してください。

カタログへのテンプレートとイメージの追加

テンプレートとイメージはコマース・サイトで使用されるメディア項目です。これらはタイプがメディアであるリポジトリ項目として格納されます。テンプレートはカタログ項目の表示に使用される JSP ページです。イメージは、カテゴリ、製品または SKU の説明に使用されるグラフィック・ファイルです。

イメージまたはテンプレートをカタログに追加すれば、それを 1 つ以上のカテゴリ、製品または SKU のプロパティの値として指定できます。たとえば、特定の複数の製品を表示するテンプレートを追加し、製品 1 つ 1 つのテンプレート・プロパティをそのテンプレートの名前に設定できます。

ACC では、フォルダを使用してテンプレートとイメージを格納および整理します。フォルダは実際のコマース・サイトの一部ではありません。

イメージとテンプレートのフォルダの作成

イメージとテンプレートを格納する新規フォルダを作成するには

1. ATG Control Center のメイン・ナビゲーション・メニューから「カタログ管理」→「カタログ要素」を選択します。
2. 「リスト」項目タイプ（Items of Type）のセレクタの横で、フォルダ・タイプを選択します。
3. 右上の「新規項目」ボタンをクリックします。「新規フォルダ」ダイアログ・ボックスが表示されます。
4. 名前フィールドに入力します。新規フォルダを既存のフォルダの子フォルダにするには、`parentFolder` フィールドを使用して親フォルダを指定します。それ以外の場合は、このフィールドを空白にします。
5. 必要に応じてオプション・フィールドに入力し、「OK」をクリックします。

イメージの追加

イメージはタイプがメディアであるカタログ項目によって表されます。実際のイメージをデータベースにインポートすることによってタイプがメディア - 内部バイナリの項目を作成するか、イメージ・ファイルの URL を参照することによってタイプがメディア - 外部の項目を作成できます。

イメージをデータベースにインポートするには

1. ATG Control Center のメイン・ナビゲーション・メニューから「カタログ管理」→「カタログ要素」を選択します。
2. 右上の「新規項目」ボタンをクリックします。「新規項目」ダイアログ・ボックスが表示されます。
3. 「項目タイプ」ドロップダウン・メニューから「メディア - 内部バイナリ」を選択します。
4. 「...」ボタンをクリックします。ファイル選択ボックスが表示されます。
5. インポートするイメージ・ファイルまで移動し、「OK」をクリックします。ダイアログ・ボックスの最下部にイメージが表示されます。
6. 名前フィールドに入力し、`parentFolder` フィールドを使用して項目を格納するフォルダを指定します（必須）。
7. **注意:** ここで指定する名前は media-internal-binary リポジトリ項目の名前であり、インポートするイメージ・ファイルの名前とは異なります。
8. 必要な任意のオプション・フィールドに入力し、「OK」をクリックします。

外部イメージへの参照を作成するには

1. ATG Control Center のメイン・ナビゲーション・メニューから「カタログ管理」→「カタログ要素」を選択します。
2. 右上の「新規項目」ボタンをクリックします。「新規項目」ダイアログ・ボックスが表示されます。
3. 「項目タイプ」ドロップダウン・メニューから「メディア - 外部」を選択します。
4. 名前フィールドと URL フィールドに入力し、`parentFolder` フィールドを使用して項目を格納するフォルダを指定します（必須）。

5. **注意:** ここで指定する名前は media-external リポジトリ項目の名前であり、URL によって参照されるイメージ・ファイルの名前とは異なります。
6. 必要な任意のオプション・フィールドに入力し、「OK」をクリックします。

テンプレートの追加

イメージと同様に、テンプレートもタイプが「メディア」であるカタログ項目によって表されます。実際のテンプレート・テキストをデータベースに格納することによってタイプが「メディア - 内部 - テキスト」の項目を作成するか、テンプレート・ファイルの URL を参照することによってタイプが「メディア - 外部」の項目を作成できます。

テンプレートをデータベースに格納するには

1. ATG Control Center のメイン・ナビゲーション・メニューから「カタログ管理」→「カタログ要素」を選択します。
2. 右上の「新規項目」ボタンをクリックします。「新規項目」ダイアログ・ボックスが表示されます。
3. 「項目タイプ」ドロップダウン・メニューから「メディア - 内部 - テキスト」を選択します。
4. 名前フィールドに入力し、`parentFolder` フィールドを使用して項目を格納するフォルダを指定します（必須）。
5. ダイアログ・ボックス最下部の大きいテキスト・ボックスに実際のテンプレート・テキストを入力します。たとえば、JSP ページに表示するテキストを入力することも、ペーストすることもできます。
6. 必要な任意のオプション・フィールドに入力し、「OK」をクリックします。

外部テンプレートへの参照を作成するには

1. ATG Control Center のメイン・ナビゲーション・メニューから「カタログ管理」→「カタログ要素」を選択します。
2. 右上の「新規項目」ボタンをクリックします。「新規項目」ダイアログ・ボックスが表示されます。
3. 「項目タイプ」ドロップダウン・メニューから「メディア - 外部」を選択します。
4. 名前フィールドと URL フィールドに入力し、`parentFolder` フィールドを使用して項目を格納するフォルダを指定します（必須）。
5. ここで指定する名前は media-external リポジトリ項目の名前であり、URL によって参照される JSP ファイルの名前とは異なることに注意してください。
6. 必要な任意のオプション・フィールドに入力し、「OK」をクリックします。

イメージおよびテンプレートとカタログ項目の関連付け

テンプレートおよびイメージを ACC に追加したら、それらをカテゴリ、製品または SKU と関連付けることができます。次の例は、小さいイメージをカテゴリまたは製品に追加する方法を示しています。

カタログで小さいイメージを指定するには

1. ATG Control Center のメイン・ナビゲーション・メニューから「カタログ管理」→「カタログ」を選択します。
2. カタログ・ツリー上のカテゴリまたは製品を選択します。
3. メニューから「イメージ」を選択します。

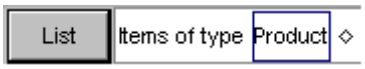
4. ドロップダウン・メニューから「小」を選択します。
5. イメージの選択ボタンをクリックします。「新規項目」ダイアログ・ボックスが表示されます。
6. 「項目タイプ」ドロップダウン・メニューから「メディア - 内部バイナリ」を選択します。
7. 「リスト」ボタンをクリックして、リストからイメージを選択します。「OK」をクリックします。
8. 「ファイル」→「保存」を選択します。

ACC での項目の検索

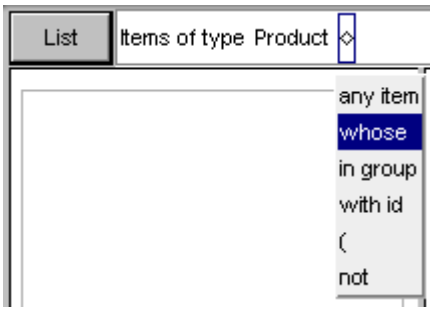
大規模なコマース・サイトのカタログには数千もの項目が含まれていることがあります。大規模なカタログを管理している場合は、特定の項目または特定の基準セットを満たす項目の検索が難しいことがあります。

カタログ内の項目を見つけやすくするために、ACC はカタログの問合せに使用できる強力な検索機能を備えています。たとえば、名前に「shoe」という単語が含まれているすべての製品を検索するための問合せを作成できます。

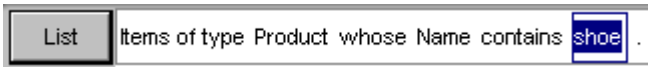
この検索機能にアクセスするには、「カタログ管理」→「カタログ要素」ウィンドウを選択し、項目タイプ（Items of Type）のドロップダウン・メニューから項目タイプを選択します。項目タイプの右の菱形に注目してください。たとえば、Product を選択すると、画面は次のようになります。



この時点で「リスト」ボタンをクリックすると、カタログ内の全製品のリストが表示されます。表示される製品のセットを限定したい場合は、菱形をクリックして、次のドロップダウン・メニューを表示します。



このメニューのオプションは、複雑な問合せの作成を手助けします。メニューからオプションを選択すると、検索基準を指定するための追加のメニューが右側に表示されます。たとえば、次の問合せでは、"Product" の右の個々の単語が別々のドロップダウン・メニューから選択されています。



左で選択する基準によって、その基準の右で選択可能になるオプションが決まります。"shoe"の後のピリオドさえが追加の基準を指定するためのドロップダウン・メニューになります。たとえば、名前に"shoe"が含まれており、作成日が 2009 年 8 月 17 日より前のすべての製品を検索する問合せを作成できます。

問合せを作成したら、「リスト」ボタンをクリックすれば、問合せに一致する項目のリストが表示されます。

バージョン競合の防止

製品カタログ内の項目を編集しているときに、他の管理者が同時にその項目を編集することがあります。ただし、ACC では、管理者どうしが互いの変更を上書きすることを防止します。

注意: ここで説明しているシステムは ATG Content Administration および Oracle Commerce Merchandising のバージョン設定機能とは異なります。その機能については、『ATG Web Commerce Content Administration Programming Guide』を参照してください。

データベース内の現在の値と同期していない変更が発行されると、ACC はそれを検出します。カタログ・リポジトリは、各項目の `version` プロパティを保持しています。このプロパティの値は整数であり、Core Commerce は項目が変更されるたびにこの値を自動的に増分します。

たとえば、新規項目を作成するとします。バージョン・プロパティは 1 です。あるユーザーが、翌日、ACC でこの項目を開き、項目のプロパティを変更し始めます。そのユーザーが加えた変更は項目を保存するまでメモリーにしか存在しません。そのユーザーが項目を編集している間に、管理者の Bob も項目を開きました。そのユーザーがまだ変更を保存していないため、Bob が開いたバージョンはまだバージョン 1 です。

そのユーザーは変更を終了し、項目を保存します。Commerce によってバージョン・プロパティの値が 2 に設定されます。

Bob が自分の変更を保存しようとすると、Core Commerce はデータベース内の現在のバージョンが、Bob が変更中のバージョンと異なることを検出し、Bob の変更を否認します。(最初のユーザーが加えた変更が含まれている) バージョン 2 が Bob のエディタにロードされます。

このシステムでは、項目を最初に開いた人が自分の変更を保存できることは保証されないことに注意してください。最初のユーザーが変更を保存する前に Bob が変更を保存していれば、最初のユーザーの加えた変更は否認されます。

3　在庫とフルフィルメントの管理

この章では、次の Oracle Commerce Core Commerce 管理に関するトピックを説明します。

在庫管理
Inventory Administration ページを使用して在庫を表示および管理したり在庫の変化の通知を送信する方法を説明します。

フルフィルメント管理
Fulfillment Administration ページを使用して出荷グループを管理および処理する方法を説明します。

在庫管理

Inventory Administration ページを使用して、在庫を表示および管理したり、在庫の変化の通知を送信します。

この項では、Inventory Administration に関する次のトピックについて説明します。

Inventory Administration ページへのアクセス

Inventory Administration ページにアクセスするには、次の手順を実行します。

Oracle Commerce Platform、Core Commerce および Dynamo Server Admin が含まれたアプリケーションをアセンブルします。詳細は、『ATG Web Commerce Platform Programming Guide』を参照してください。次に、アプリケーションを配置します。

1. アプリケーション・サーバーにアクセスするための URL をブラウザで指定することによって Core Commerce Administration ページにアクセスします。たとえば、JBoss のユーザーは、デフォルトで次の URL を使用します。

   ```
   http://hostname:8080/dyn/admin/atg/commerce/admin/
   ```

   他のアプリケーション・サーバーのデフォルト・ポート情報については、『ATG Web Commerce Installation and Configuration Guide』を参照してください。
2. Inventory Administration リンクをクリックします。Inventory Administration ページが開きます。

在庫の表示

Inventory Administration ページには、現在の在庫が表形式で表示されます。表には一度に 10 個の項目がアルファベット順に表示されます。「次」リンクと「前」リンクをクリックしてリストの中を移動します。

表の項目数を減らすために項目名でリストをフィルタできます。画面最上部の 2 つのフィールドに文字を入力し、項目の検索を絞り込みます。たとえば、R と S という文字から始まる名前を持つすべての項目を表示するには、最初のフィールドに R、2 番目のフィールドに S を入力し、「表示」ボタンをクリックします。表に R と S から始まる項目が一度に 10 個表示されます。

注意: 在庫のフィルタリングでは、大文字と小文字が区別されます。大文字、小文字が製品カタログ内の表示名エントリと一致している必要があります。

在庫の更新

Inventory Administration ページを使用して、使用可能な商品の在庫構成を更新できます。これらの値の詳細は、『ATG Web Commerce Programming Guide』の *在庫のフレームワーク*を参照してください。

在庫構成を更新するには、次の手順を実行します。

1. Inventory Administration ページにアクセスします。
2. 画面最上部の「在庫の更新」リンクをクリックします。
3. SKU ID フィールドに更新する項目の SKU ID を入力します。
4. 新規値フィールドに更新するプロパティの新しい値を入力します。

   1 列目から Inventory Manager プロパティを選択します。Inventory Manager のプロパティの詳細は、『ATG Web Commerce Programming Guide』の *在庫のフレームワーク*を参照してください。
5. 2 列目から次のいずれかを選択します。
   - 設定: 値フィールドで指定されている値にプロパティを設定します。
   - increases: 値フィールドで指定された値だけ現在の値を増やします。
   - decrease: 値フィールドで指定された値だけ現在の値を減らします。
6. 「更新」ボタンをクリックします。

   選択された在庫プロパティが指定された値に設定されます。

在庫更新通知の送信

Inventory Administration ページを使用して、更新された在庫項目の通知をフルフィルメント・システムに送信できます。通知を送信するには、次の手順を実行します。

1. Inventory Administration ページにアクセスします。
2. ページ最上部の Inventory Update Notification リンクをクリックします。
3. 新規在庫が使用可能になっている項目の SKU ID を入力します。複数の SKU ID があるときはスペースで区切ります。
4. Notify ボタンをクリックします。
5. Java Message Service（JMS）メッセージが在庫更新の通知として送信されます。

フルフィルメント管理

Fulfillment Administration ページを使用して出荷グループを管理および処理します。この項では、Fulfillment Administration に関する次のトピックについて説明します。

Fulfillment Administration ページへのアクセス

Fulfillment Administration ページにアクセスするには、次の手順を実行します。

1. Oracle Commerce Platform、Core Commerce、フルフィルメントおよび Dynamo Server Admin のモジュールが含まれたアプリケーションをアセンブルし、配置します。アプリケーションのアセンブルの詳細は、『ATG Web Commerce Platform Programming Guide』を参照してください。
2. アプリケーション・サーバーにアクセスするための URL をブラウザで指定することによって Core Commerce Administration ページにアクセスします。たとえば、JBoss のユーザーは、デフォルトで次の URL を使用します。

   ```
   http://hostname:8080/dyn/admin/atg/commerce/admin/
   ```

   デフォルトの URL については、『ATG Web Commerce Installation and Configuration Guide』を参照してください。
3. フルフィルメント管理リンクをクリックします。Fulfillment Administration ページが開きます。

フルフィルメント・システムへのオーダー出荷の通知

Fulfillment Administration ページには、出荷準備が整っているリポジトリ内のすべての出荷グループを表示できます。ページ最上部の出荷可能グループセクションには出荷準備が整っているすべての出荷グループを取得するリンクが含まれています。リストを表示した後、この画面を使用して、出荷グループが出荷されたことをフルフィルメント・システムに通知することもできます。

出荷グループのリストを表示し、出荷グループが出荷されたことをフルフィルメント・システムに通知するには、次の手順を実行します。

1. Fulfillment Administration ページにアクセスします。詳細は、Fulfillment Administration ページへのアクセスを参照してください。
2. 「ここをクリックすると、出荷の準備が整っているリポジトリ内のすべての出荷グループのリストを取得できます」という文中の「ここをクリック」リンクをクリックします。

   ステータスが PENDING_SHIPMENT になっているすべての出荷グループのリストが Core Commerce に表示されます。
3. 出荷グループが含まれているオーダーのオーダーID を入力します。
4. 出荷準備が整っている出荷グループの出荷グループ ID を入力します。
5. 「出荷」ボタンをクリックします。

   指定されたグループが出荷されたことをフルフィルメント・システムに通知する JMS メッセージが送信されます。

出荷グループの再処理

このセクションを使用して、特定の出荷グループが変更され、再処理が必要であることをフルフィルメント・システムに通知する `ShippingGroupUpdate` メッセージを送信します。

1. 出荷グループが含まれているオーダーのオーダーID を入力します。
2. 出荷準備が整っている出荷グループの出荷グループ ID を入力します。
3. **「送信」**ボタンをクリックします。

   指定されたグループが変更され、再処理が必要であることをフルフィルメント・システムに通知する JMS メッセージが送信されます。

オーダーの印刷

Fulfillment Administration ページのオーダーの印刷セクションでは、オーダー情報を印刷できます。この機能は、オーダーの出荷グループおよび変更の書面による記録が必要な場合に便利です。

1. 出荷グループが含まれているオーダーのオーダーID を入力します。
2. オーダーの印刷ボタンをクリックします。

   オーダーがブラウザのウィンドウに表示されます。
3. ブラウザでの通常の印刷の手順に従ってブラウザからオーダーを印刷します。

4 価格表の管理

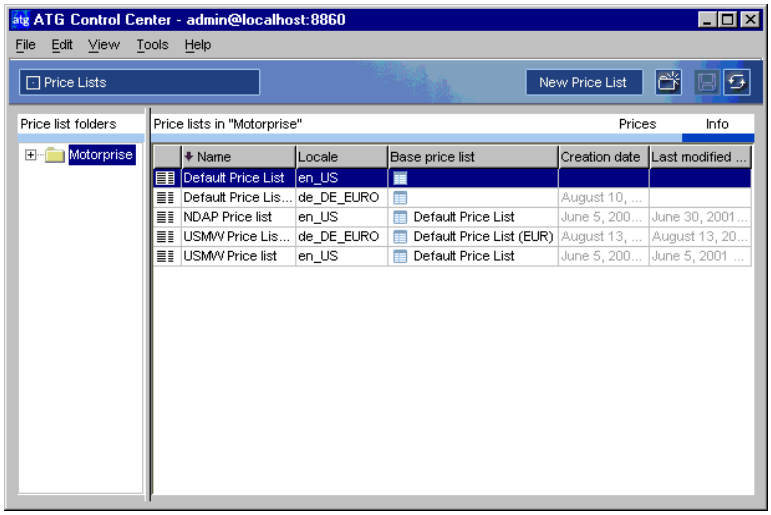
価格表を利用することで、特定の顧客グループに対して特定の価格セットをターゲット設定できます。価格表は ATG Control Center（ACC）の 1 つのインタフェースを使用して管理されます。たとえば、価格表を使用して、契約、見積依頼および事前交渉した価格に基づいて個々の顧客ごとに専用の価格設定を適用する企業間価格設定を実装できます。

ビジネス・ユーザーは ACC を使用して、次の価格表タスクを実行できます。

既存の価格表の表示

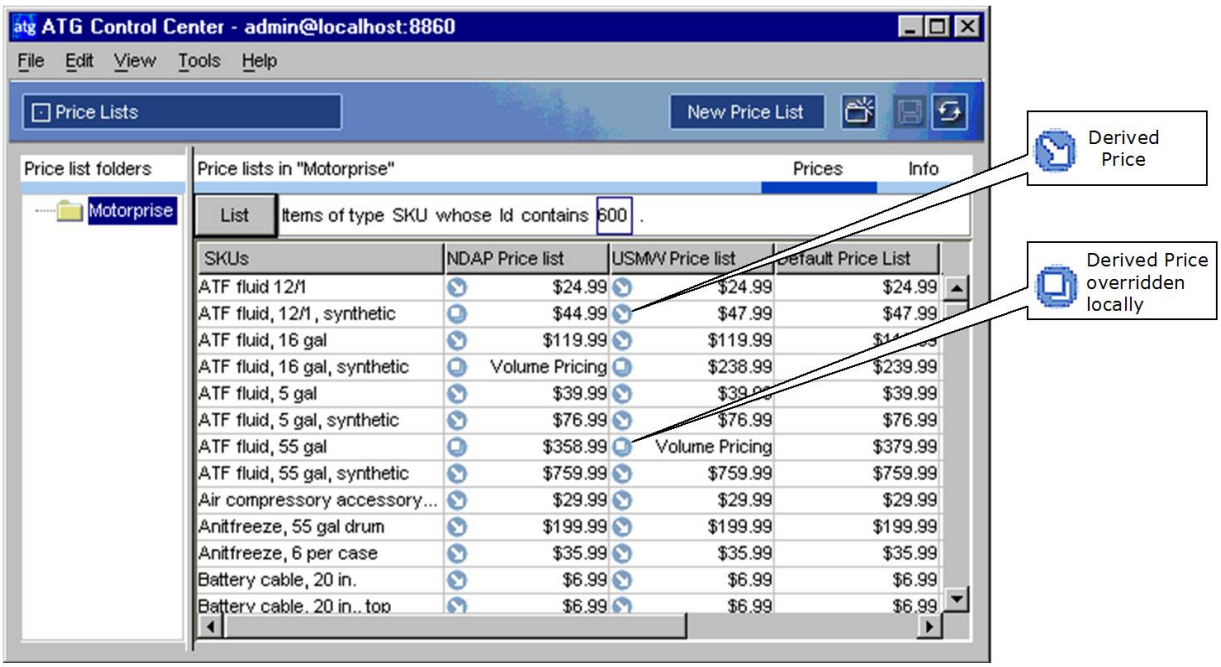
ACC で既存の価格表を表示するには、次の手順を実行します。

1. メイン ACC ナビゲーション・バーから「価格設定」を選択します。
2. 価格設定の選択肢から「価格表」を選択します。
3. 価格表フォルダ・リストからフォルダを選択し、画面右側の「情報」をクリックします。フォルダ内の価格表のリストが ACC のメイン・セクションに表示されます。



注意: 必要な場合は、この画面から価格表の名前またはロケールを変更できます。基本価格表のロケールしか変更できません。親価格表を持つ価格表のロケールは変更できません。親価格表のロケールを変更すると、子価格表のロケールは自動的に変わります。これらの変更を表示するには、「リフレッシュ」をクリックする必要があります。

4. 価格表内の価格のリストを表示するには、「価格」をクリックします。
5. ドロップダウン・メニューをクリックして、検索を「Find Items of type SKU（タイプ SKU の項目を検索）」に変更します。
6. 「検索」をクリックします。価格表内のすべての SKU のリストおよび SKU に関連付けられた価格が表示されます。



新規価格表フォルダの作成

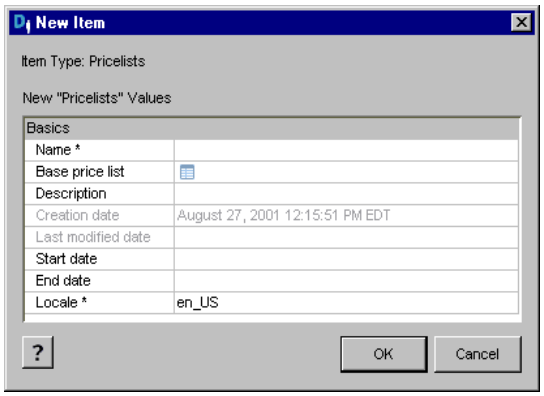
新規価格表フォルダを作成するには、価格表フォルダ・ツリー上の新規フォルダの場所を選択し、ツールバーの「新規フォルダ」ボタンをクリックします。

注意: ルート・レベルにフォルダを作成する場合は、まず[Ctrl]を押したまま現在選択されているフォルダをクリックすることによって、フォルダ・ツリーからフォーカスを外す必要があります。

新規価格表の作成

ACC で新規価格表を作成するには、次の手順を実行します。

1. メイン ACC ナビゲーション・バーから「価格設定」を選択します。
2. 価格設定の選択肢から「価格表」を選択します。
3. 価格表フォルダ・リストから新規価格表の作成先フォルダを選択するか、新規フォルダを作成します。新規フォルダの作成の詳細は、新規価格表フォルダの作成を参照してください。
4. 「新規価格表(&N)」ボタンをクリックします。「新規項目」ボックスが表示されます。



5. 価格表の名前と摘要を入力します。
6. 基本価格表を選択します（オプション）。基本価格表を指定すると、価格表内の項目がデフォルトで基本価格表の価格に設定されます。

   注意: 価格表のみを使用している場合は、すべての製品の価格が価格表に含まれている必要があります。価格表と SKU ベースの価格設定の組合せを使用している場合は、価格表に価格がなければ、カタログ価格がデフォルトで使用されます。この機能を使用する方法については、『ATG Web Commerce Programming Guide』の *SKU をベースとした価格設定と価格表との組合せ*に関する項を参照してください。
7. 必要な場合はロケールを変更します。ロケールを変更するには、ロケール・フィールドをクリックし、ドロップダウン・メニューからロケールを選択します。
8. 「OK」をクリックします。選択されたフォルダに新規価格表が格納されます。

既存の価格表の価格の変更

既存の価格表の価格を変更するには、次の手順を実行します。この項では、価格表の価格を直接編集する方法を説明します。数量価格設定で使用する価格も変更できます。詳細は、数量価格の設定を参照してください。

1. 既存の価格表を開き、価格表に含まれている SKU の価格を表示します。詳細は、既存の価格表の表示を参照してください。
2. 変更する価格を直接ダブルクリックします。複数の価格表が表示されている場合は、適切な価格表の価格が選択されていることを確認してください。数量価格設定を行うこともできます。詳細は、数量価格の設定を参照してください。
3. フィールドに新しい価格を入力し、[Enter]を押します。新しい価格が価格表に表示されます。

   注意: 価格は、価格表に割り当てられているロケールの表記規則に従って入力する必要があります。たとえば、ロケールが欧州通貨の `de_DE_Euro` である場合は、2,79 のようにカンマを使用して価格を入力する必要があります。

価格表間での価格のコピー

価格表間で価格をコピーするには、次の手順を実行します。

注意: 同じロケールの価格表間でのみ価格をコピーできます。

1. コピーする価格フィールドをハイライトします。
2. 価格フィールドを右クリックし、メニューから「コピー」を選択します。
3. ペースト先のフィールドをハイライトします。
4. フィールドを右クリックし、メニューから「ペースト」を選択します。コピーされた価格が選択されたフィールドにペーストされます。

   注意: 価格のコピー元のフィールドと同じ構成のフィールドまたは同じフィールド構成の倍数のフィールドにのみペーストできます。たとえば、ある列から 3 個のフィールドをコピーした場合は、コピー先列内の 3 個のフィールドまたは 6 個のフィールドを選択した場合にのみフィールドをペーストできます。フィールドの構成が一致しない場合は、右クリック・メニューに「ペースト」オプションが表示されません。

一括価格と段階的価格の設定

価格表を使用して様々な価格設定モデルを実装できます。数量価格設定を使用して、購入される品目の個数に基づいて製品の価格を設定できます。一括価格設定と段階的価格設定は数量価格設定の 2 つのスタイルです。

一括価格設定では、オーダーされた総数量に基づいて製品の価格を計算します。次の例では、ハンマーの単価を 1～10 個購入すれば 20ドル、11～20 個購入すれば 17ドル、21 個以上購入すれば 15ドルに設定します。

購入されるハンマーの数	一括購入価格の単価	一括購入価格の合計
ハンマー7 個	ハンマー7 個の単価$20	
ハンマー14 個	ハンマー14 個の単価$17	
ハンマー23 個	ハンマー23 個の単価$15	

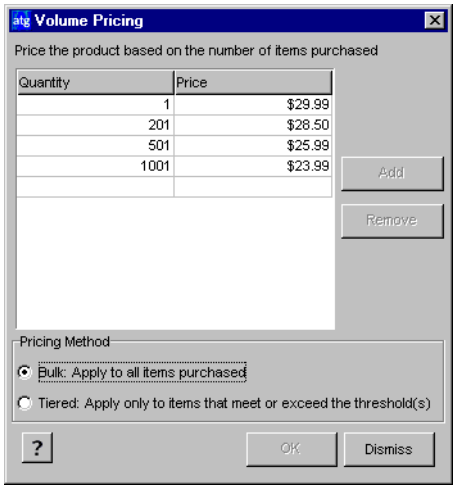
段階的価格設定では、異なる価格設定レベルに対して固定数量または固定重量を使用することによって製品の価格を計算します。次の例では、ハンマーの単価を最初の 10 個に対して 20ドル、次の 10 個に対して 17ドル、それ以上の数量に対して 15ドルに設定します。

購入されるハンマーの数	段階的購入価格の単価	段階的購入価格の合計
ハンマー7 個	ハンマー7 個の単価$20	
ハンマー15 個	ハンマー10 個の単価$20 ハンマー5 個の単価$17	
ハンマー23 個	ハンマー10 個の単価$20 ハンマー10 個の単価$17 ハンマー3 個の単価 $15	

数量価格設定の表示

ACC を使用して価格表内の SKU に対して設定されている数量価格を表示するには、次の手順を実行します。

1. 使用可能な価格表の SKU の価格設定を表示します。詳細は、既存の価格表の表示を参照してください。
2. 数量価格が設定されている価格フィールドを選択します。
3. フィールドを右クリックし、メニューから「数量価格の表示」を選択します。「数量価格設定」ボックスが表示されます。

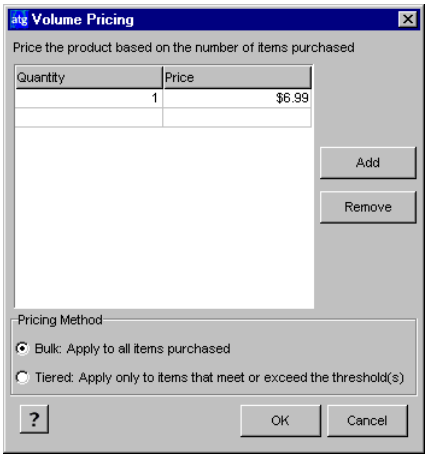


4. 「終了」をクリックして「数量価格設定」ボックスを閉じます。このボックス内の情報を編集する方法については、数量価格の設定を参照してください。

数量価格の設定

ACC を使用して価格表内の SKU の数量価格を設定または編集するには、次の手順を実行します。

1. 使用可能な価格表の SKU の価格設定を表示します。詳細は、この章の既存の価格表の表示を参照してください。
2. 価格表の価格フィールドを選択します。
3. フィールドを右クリックし、メニューから「数量価格の設定/編集」を選択します。「数量価格設定」ボックスが表示されます。



4. ボックス最下部の「価格設定方法」領域から一括または「段階的」を選択します。
5. 表示される列に価格設定方式の数量と価格を入力します。行を追加および削除するには、「追加」ボタンと「削除」ボタンを使用します。
6. 「OK」をクリックします。

価格表の削除

価格表を削除するには、次の手順を実行します。

1. 価格表にアクセスし、画面右側の「情報」タブをクリックして、使用可能な価格表のリストを表示します。詳細は、既存の価格表の表示を参照してください。
2. 削除する価格表が含まれている行を選択します。
3. 「ファイル」→「価格表の削除(&D)」を選択します。処理の確認を求めるプロンプトが表示されます。
4. 「はい」をクリックして価格表を削除します。

ユーザーへの価格表の割当

ACC を使用してユーザーに価格表を割り当てるには、次の手順を実行します。

1. メイン ACC ナビゲーション・バーから「個人および組織」を選択します。
2. 「個人および組織」の選択肢から「プロファイル Repository」を選択します。
3. 項目タイプ（Item of type）でユーザー(User)を選択し、「リスト」をクリックして、すべてのユーザーのリストを表示します。
4. リストからユーザーを選択します。ユーザーの情報が画面のメイン・セクションに表示されます。
5. 「Commerce - 契約」セクションを見つけて、価格表プロパティの横の「…」ボタンをクリックします。
6. 「価格表」ダイアログ・ボックスで、「リスト」をクリックして、すべての価格表を表示します。
7. 価格表の名前をクリックし、「OK」をクリックします。

5 プロモーションの作成と保守

Commerce ベースの Web サイトの作成および保守のプロセスのオプションとして、特定の製品または製品グループの割引を提供するために定期的に使用するプロモーションを設定できます。たとえば、サイトの「ようこそ」ページにイメージを掲載して母の日関連の製品にスポットライトを当て、顧客が特定の期日までにオーダーすれば 10%の割引を適用することによって、母の日向けの様々な製品をプロモーションできます。あるいは、その日に登録すれば 1 つの製品の価格で 2 つの製品を提供するなどして、サイトへの登録を促すために訪問者に割引を提示することもできます。

Oracle Commerce Core Commerce のプロモーションで設定できる割引のタイプの例を次に示します。

- 特定の製品の定額割引。
- オーダー合計額の定額割引。
- 特定の製品の定率割引。
- オーダー合計額の定率割引。
- 製品の属性（たとえば色）に基づく製品の定額割引または定率割引。
- 送料の無料化または割引。

プロモーションが機能する仕組み

プロモーションでは、サイト訪問者がプロモーション対象として適格となる*条件*を定義するオプションを指定します。顧客が適格基準を満たすと、プロモーションが*提供*されます。

プロモーションを定義したら、プロモーション対象としての適格条件を指定します。たとえば、顧客のオーダーがある金額に到達したときに認めるという条件を作成できます。または、特定の製品を購入した個人に必ずプロモーションを提供するという条件を作成することもできます。すべてのプロモーションに条件が必要なわけではありません。たとえば、グループ割引（使用可能な場合）が顧客のカート内のすべての品目に適用されるときは、適格条件が不要になります。

顧客がプロモーション対象に適格となる条件を定義したら、オファーを定義します。オファーを作成するとき、どの品目を割引するか、どのタイプの割引を顧客が受けるかを示す割引ルールを作成します。たとえば、顧客のカート内のすべての品目に適用されて、オーダー全体を 10 パーセント割引するオファーを作成できます。または、特定の製品を 10 ドルという固定価格だけ割り引くオファーを作成することもできます。

Oracle Commerce Merchandising によって、条件とオファーの作成に使用できるテンプレートがいくつか提供されています。これらのテンプレートは、プロモーションのタイプに関連付けられるオファーまたは条件のタイプを選択するために設計されています。たとえば、BOGO テンプレートを使用して Buy One Get One（1 個購入するともう 1 個入手できる）（BOGO）プロモーションを作成できます。このテンプレートには、BOGO プロモーションに適用されるデフォルトの条件とオファーが指定されます。

ただし、テンプレートですべてのプロモーションが対象とされているわけではありません。高度な条件とオファー文を使用して、カスタマイズしたプロモーションを作成できます。このような文を使用すると、高度で複雑なプロモーションを作成できます。

Merchandising で使用可能な条件とオファーのすべてのテンプレート、および高度な条件とオファー文の作成の詳細は、『ATG Web Commerce マーチャンダイジング・ガイド』を参照してください。

プロモーションを設定したら、プロモーション対象としての適格性を決定する必要があります。最も簡単な方法は、すべての顧客に対して自動的に実施されるプロモーションを設定することです。選択性を高める場合は、プロモーション対象として適格な訪問者を指定するパラメータを設定できます。

顧客がログインすると、Core Commerce はその顧客の訪問者プロファイルをチェックし、`activePromotions` 属性から判断して、その顧客が、設定されているいずれかのプロモーションの対象として適格であるかどうかを確認します。それらの割引を使用して、品目がショッピング・カートに追加されたときにその品目の価格を計算します。

顧客にプロモーションを告知する必要はありません。プロモーションは、精算時に調整済の価格が顧客に表示されるように設定できます。顧客にプロモーションを告知する場合は、次のようないくつかのオプションがあります。

- クーポンを作成し、郵便または E メールを使用してクーポン・コードを配布できます。
- サイト・ページにプロモーションについて説明するテキストを掲載できます。
- `GetApplicablePromotions` ドロップレットを使用して、特定の品目に適用されるプロモーションを特定し、その情報を製品ページに表示することもできます。

Business Control Center の Merchandising セクションを使用してプロモーションを作成します。Merchandising コンポーネントの作業の詳細は、『ATG Web Commerce マーチャンダイジング・ガイド』を参照してください。

プロモーションのタイプ

Merchandising UI を使用してプロモーションを作成するとき、使用可能なプロモーションのタイプを指定します。

プロモーションは次の 3 つのタイプのいずれかにできます。

- 品目割引 – これは特定の品目に基づいたプロモーションです。割引する品目を含む特定の製品またはカテゴリを指定できます。
- オーダー割引 – このプロモーションは、個々の製品ではなくオーダー全体に適用されます。このプロモーションでは、オーダー全体に対する割引基準を指定できます。
- 出荷割引 – このプロモーションでは、オーダーの送料に割引を指定できます。

品目割引プロモーション

次のプロモーションは「品目割引」テンプレートを使用して作成されます。これらのプロモーションの詳細は、『ATG Web Commerce マーチャンダイジング・ガイド』を参照してください。

プロモーション	説明
品目割引の取得(製品、カテゴリ、ブランド別など)	これは、特定の品目、カテゴリ、ブランドなど、特定の製品のセットを割り引く基本的なプロモーションです。
BOGO	これは 1 個購入するともう 1 個入手できるプロモーションです。顧客が購入する必要がある製品を設定し、次に 2 つめの品目の割引金額を指定します。たとえば、シャツを 1 枚購入すると、もう 1 枚が 50%オフで購入できます。
品目 X を購入すると品目 Y を入手できる	このプロモーションは、顧客が指定された別の製品を購入するときに、指定された製品のセットを割り引きます。
Y 支払うと品目割引を取得	このプロモーションは、顧客が指定された金額を支払うときに、特定の品目を割り引きます。
X で Y 支払うとオーダー割引を取得	このプロモーションは、顧客のオーダーが製品領域に指定された金額を満たすときに割引を提供します。これは、個別の品目に対するショッピング・カート全体を見ている点で、Y 支払うと品目割引を取得とは異なります。
指定された時間枠内に購入すると品目割引を取得	このプロモーションでは、特定の期間中に購入された品目を割り引くことができます。
グループ割引の取得	このプロモーションでは、割引のために複数の品目をグループ化できます。
段階的価格分岐	このタイプのプロモーションでは、顧客が購入する品目の数に基づいて異なる割引レベルを設定できます。
高度な条件およびオファー	このプロモーションでは、独自の条件およびオファー基準を設計して、カスタマイズしたプロモーションを作成できます。

購入時ギフト(GWP)プロモーションを作成するときにも、「品目割引」テンプレートが使用されます。GWP プロモーションには次が含まれます。これらのプロモーションの詳細は、『ATG Web Commerce マーチャンダイジング・ガイド』を参照してください。

プロモーション	説明
購入時ギフト(GWP)の取得	この基本的なプロモーションでは、顧客が購入を行うとその顧客に無料でギフトを提供できます。
品目 X を購入すると GWP を取得	このプロモーションは、顧客が特定の品目を購入する場合に、その顧客に無料のギフトを提供します。

プロモーション	説明
Y 支払うと GWP を取得	このプロモーションは、顧客のオーダーが指定された金額を満たす場合に、その顧客に無料のギフトを提供します。
X で Y 支払うと GWP を取得	このプロモーションは、顧客が指定された製品領域で指定された金額を支払う場合に、その顧客に無料のギフトを提供します。

オーダー割引プロモーション

次のプロモーションは「オーダー割引」テンプレートを使用して作成されます。これらのプロモーションの詳細は、『ATG Web Commerce マーチャンダイジング・ガイド』を参照してください。

プロモーション	説明
オーダー割引の取得	このプロモーションでは、顧客のオーダー全体に対する割引を設定します。
X を購入するとオーダー割引を取得	このプロモーションは、顧客が特定の品目を購入すると、その顧客のオーダー全体を割り引きます。
Y 支払うとオーダー割引を取得	このプロモーションは、顧客が指定された金額を購入すると、その顧客のオーダー全体を割り引きます。
X で Y 支払うと品目割引を取得	このプロモーションでは、顧客が指定された製品領域で指定された金額を支払うと、オーダーを割り引くことができます。
指定された時間枠内に購入	このプロモーションでは、特定の期間中に行われたオーダーを割り引くことができます。
段階的オーダー割引	このタイプのプロモーションでは、顧客が購入する品目の数に基づいて異なる割引レベルを設定できます。
高度な条件およびオファー	このプロモーションでは、独自の条件およびオファー基準を設計して、カスタマイズしたプロモーションを作成できます。

出荷割引プロモーション

次のプロモーションは「出荷割引」テンプレートを使用して作成されます。これらのプロモーションの詳細は、『ATG Web Commerce マーチャンダイジング・ガイド』を参照してください。

プロモーション	説明
出荷割引の取得	この基本的なプロモーションでは、オーダーの送料に割引を指定できます。
X を購入すると出荷割引を取得	このプロモーションは、顧客が特定の品目を購入したときに、出荷割引を提供します。

プロモーション	説明
Y 支払うと出荷割引を取得	このプロモーションは、顧客のオーダーが指定された金額に達すると、出荷割引を提供します。
X で Y 支払うと出荷割引を取得	このプロモーションは、顧客が指定された製品領域で特定の金額を支払うと、送料を割り引きます。
期間内に購入すると出荷割引を取得	このプロモーションは、顧客が指定された期間にオーダーすると、送料を割り引きます。
高度な条件およびオファー	このプロモーションでは、独自の条件およびオファー基準を設計して、カスタマイズしたプロモーションを作成できます。

プロモーションの作成

この項では、プロモーションを作成するときに実行する手順の概要を示します。プロモーションの作成とテンプレートの作業の詳細は、『ATG Web Commerce マーチャンダイジング・ガイド』にあります。

Web サイトでのプロモーションの宣伝に使用するテキストや画像があれば、プロモーションを作成する前に、ページ開発者はそれらが含まれた HTML ファイルを作成する必要があります。プロモーション用のテキストや画像はメディアといいます。プロモーション・メディアの作業の詳細は、『ATG Web Commerce マーチャンダイジング・ガイド』を参照してください。

Business Control Center でプロモーションを作成するには、次のようにします。

1. 「Commerce アセット」プロジェクトで「Merchandising」に移動します。
2. プロモーションおよびクーポンを選択します。
3. 「プロモーション」を選択します。
4. 「新規項目」ギアから、作成中のプロモーションのタイプに応じて次のいずれかを選択します。
   - 品目割引
   - オーダー割引
   - 出荷割引
5. プロモーション名を指定して、プロモーション・テンプレートを選択します。
6. 必要な場合は条件基準を入力します。
7. オファー基準を入力します。
8. 「可用性およびクーポン」タブを使用して、プロモーション可用性と各顧客の最大使用数を入力します。
9. 「サイト」タブを使用して、このプロモーションを実行するサイトを入力します。
10. 「メディア」タブを使用して、このプロモーションのイメージと表示日を指定します。
11. 「作成」をクリックしてプロモーションを保存します。

新しいプロモーションを作成するときに設定できる多くのオプションの設定があります。これらの設定の詳細は、『ATG Web Commerce マーチャンダイジング・ガイド』を参照してください。

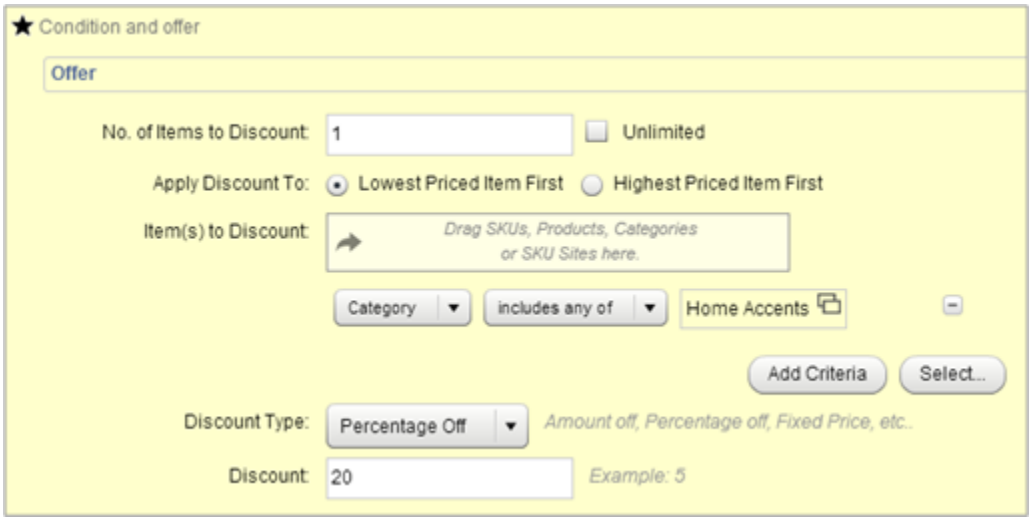
注意: 目的とする日時または任意の日時に顧客が新規プロモーションを利用できるかどうかに影響を及ぼす可能性のある要因は複数あります。これらの要因には、新規プロモーションが追加された日時、顧客のセッションが作成された日時、自動プロモーションの更新スケジュールがあります。開発者は、ビジネス・ユーザーの意図した日時にサイト訪問者がプロモーションの適用を確実に受けられるように措置を講じることができます。詳細は、『ATG Web Commerce Programming Guide』を参照してください。

作成できるいくつかのプロモーションの例を次に示します。

例 1: 特定のカテゴリに属する品目の割引

次の例は、特定カテゴリの品目を割り引くプロモーションの作成方法を示しています。

プロモーション・タイプ	**品目割引**
テンプレート	品目割引の取得(製品、カテゴリ、ブランド別など)
割引タイプ	割引率

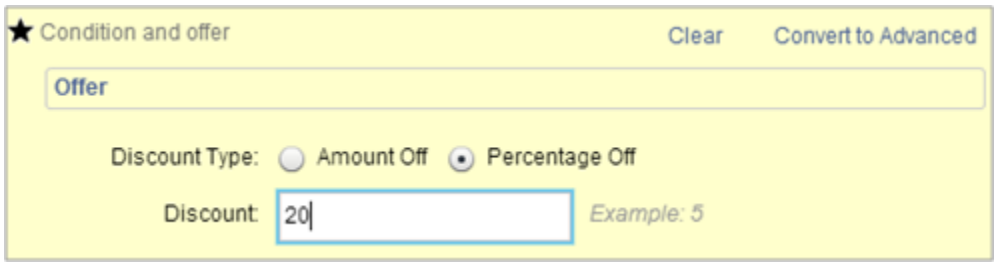


このプロモーションは、顧客がホーム・アクセント・カテゴリから購入した 1 つの品目に対して、20%の割引を提供します。顧客のショッピング・カート内の品目をシステムが識別したときに品目割引が適用されるため、このプロモーションに必要な適格条件はないことに注意してください。

例 2: オーダー合計額からの特定の金額の割引

次の例は、顧客のオーダーを割り引くプロモーションの作成方法を示しています。

プロモーション・タイプ	**オーダー割引**
テンプレート	オーダー割引の取得
割引タイプ	割引率

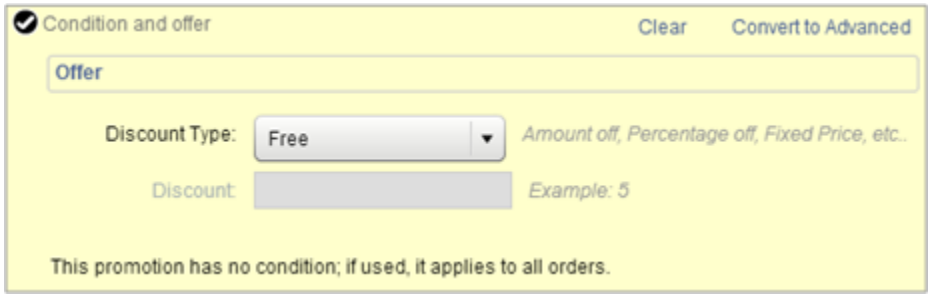


このプロモーションでは、顧客のオーダー合計に対して 20%の割引を適用します。このプロモーションはオーダーを持つ任意の顧客に適用されるため、適格条件はないことに注意してください。

例 3: 送料のプロモーションの提供

次の例は、任意のオーダーの送料を無料にするプロモーションの作成方法を示しています。

プロモーション・タイプ	**出荷割引**
テンプレート	出荷割引の取得
割引タイプ	無料

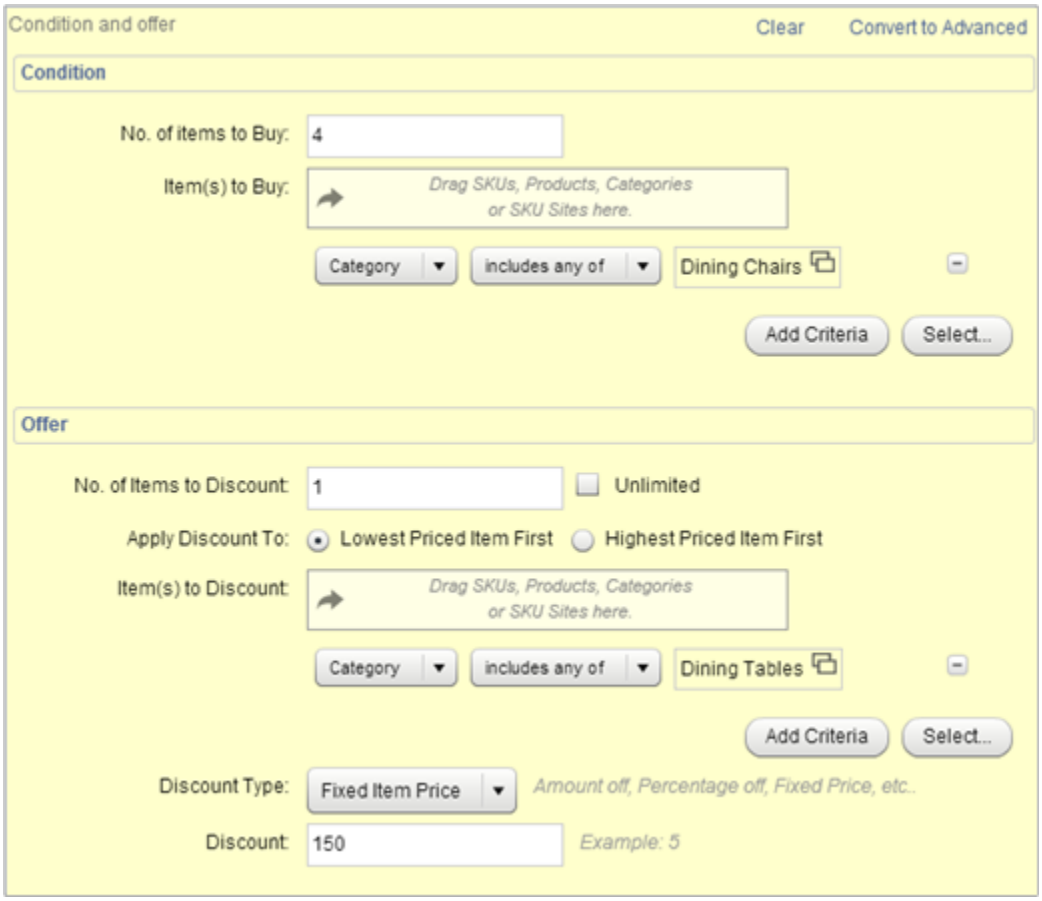


適格条件を指定しないことによって、このプロモーションはすべてのオーダーに適用されます。

例 4:顧客が別の製品を購入すれば 1 つの製品を値引きする割引の提供

次の例は、別の品目を購入するとある品目が固定価格になるプロモーションの作成方法を示しています。

プロモーション・タイプ	**品目割引**
テンプレート	品目 X を購入すると品目 Y を入手できる
割引タイプ	固定価格

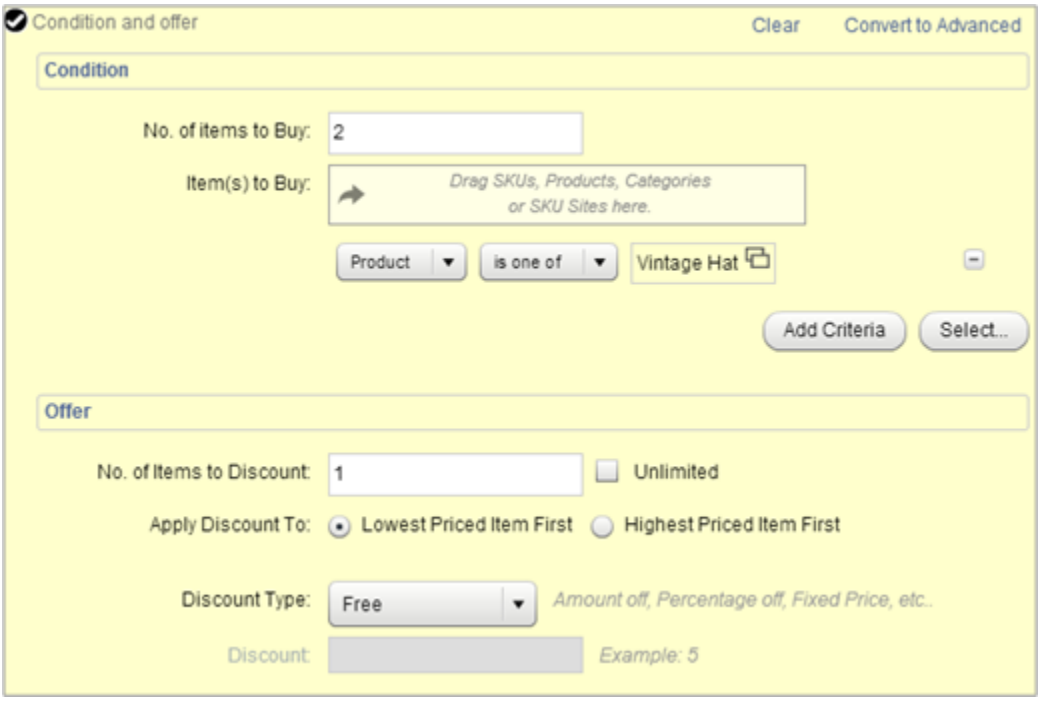


このプロモーションは、顧客がダイニング・チェアー・カテゴリから製品を 4 個購入するとアクティブになり、ダイニング・テーブル・カテゴリの任意のテーブルを固定価格の$150 で提供します。

例 5:1 個の価格で 2 個を入手できる割引の提供

次の例は、1 個の価格で 2 個を入手できる割引を提供するプロモーションの作成方法を示しています。

プロモーション・タイプ	**品目割引**
テンプレート	BOGO
割引タイプ	無料



このプロモーションは Buy One Get One（1 個購入するともう 1 個入手できる）テンプレートを使用します。顧客が古着の帽子を 2 個購入すると、そのうち 1 個が無料になります。

スタック・ルールの使用

スタック・ルールを使用すると、顧客が複数のプロモーションのスタック効果を予定外に利用することを回避できます。（クーポン・バッチは、対象を絞る必要のあるプロモーションを顧客が広めないようにするための、別の方法となります。詳細は、クーポン・プロモーションの設定を参照してください。）

スタック・ルールは Merchandising で作成されます。詳細は、『ATG Web Commerce マーチャンダイジング・ガイド』を参照してください。

プロモーション・アップセルの作成

個々のプロモーションにいくつでもプロモーション・アップセルを持たせることができます。「2500 ドルを超えるオーダーに対して送料を無料にする」プロモーションがあるとします。オーダーの金額が 2400 ドルに達していて、顧客のショッピング・カートにブランド X の自転車が入っているときに、1 つのプロモーション・アップセルが表示されます。このアップセル・メッセージは、たとえば、「あと 100 ドルお買い物をすると送料が無料になります。ブランド X のウィンドブレーカはいかがですか」というようなものになります。ここでは、ブランド X のウィンドブレーカの価格がすべて 100 ドル以上であることが想定されています。同じプロモーションの別のプロモーション・アップセルは、オーダーが 2000 ドルに達している顧客を対象とします。このプロモーション・アップセルでは、「ショッピング・カートに入っている製品の送料は 197.75 ドルです。あと 500 ドル追加すれば、送料が無料になります」と通知します。

プロモーション・アップセルを作成するときは、ルールを作成することによって、プロモーション・アップセルが当てはまる状況を説明します。このルールを作成する方法は、プロモーションの割引ルールを作成する方法と似通っています。プロモーションの適用を受けるために製品を購入しようという意欲をかき立てる画像な

どのメディア項目を指定することもできます。そのような製品は、プロモーション・アップセルに関連付けられたアップセル処理内で指定されます。個々のプロモーション・アップセルは優先度も持っています。

顧客が特定のプロモーションに関する複数のプロモーション・アップセルに該当する可能性があるため、個々のプロモーション・アップセルに優先度が設定されます。Oracle Commerce Platform では、特定のプロモーションについて、まず優先度が最も高いプロモーション・アップセルを評価して、顧客がそのプロモーション・アップセルに該当するかどうかを確認し、顧客に当てはまるプロモーション・アップセルが見つかるまで、そのリストを優先度の高い順に評価します。顧客に当てはまるものが見つかると、そのプロモーション・アップセルが顧客に割り当てられます。

一般に、顧客には同時に 1 つのプロモーションにつき 1 つのプロモーション・アップセルが割り当てられます。プロモーションは複数のプロモーション・アップセルを持っている可能性が高いため、優先度を割り当てることによって、2 つのプロモーション・アップセルに該当する顧客が適切なプロモーション・アップセルを取得することが確実になります。たとえば、2 つのプロモーション・アップセルがある場合、それらを 100 ドル以上のオーダー（優先度 1）と 50 ドル以上のオーダー（優先度 2）として優先順位付けします。そうすることで、両方のプロモーション・アップセルに適格性のある、オーダー額が 100 ドルの顧客には、オーダーの合計額に最も近いプロモーション・アップセルのみが割り当てられます。

ときには、顧客にプロモーションの複数のプロモーション・アップセルを割り当てたいことがあります。前述のように、プロモーション・アップセルの中には、顧客のショッピング・カートに入っている特定の製品に基づいて設定されるものがあります。顧客のショッピング・カートに特定のブランドの製品が入っているときにそのブランドのアップセルを目指す、2 つのブランドを扱う 2 つのプロモーション・アップセルが設定されることがあります。

顧客が両方のプロモーション・アップセルを取得できるように、特定のプロモーションの複数のプロモーション・アップセルに同じ優先度番号を割り当てることができます。Oracle Commerce Platform では、顧客に当てはまる 1 つのプロモーション・アップセルを見つけた場合、操作を終了する前に、同じ優先度を持つ他のすべてのプロモーション・アップセルを評価します。顧客には、同じ優先度を持つ、顧客に当てはまるすべてのプロモーション・アップセルが付与されます。それより優先度の低いその他のプロモーション・アップセルは無視されます。

プロモーション・アップセルの作業の詳細は、『ATG Web Commerce マーチャンダイジング・ガイド』を参照してください。

プロモーションの使用不可化

一般原則として、プロモーションを削除しないでください。そのかわりに、使用可能プロパティを `false` に設定することによって、プロモーションを使用不可にしてください。オーダーで使用中のプロモーションを削除するとオーダー処理中にエラーが発生します。

ただし、プロモーションがオーダーで使用されていないことが確実な場合は、プロモーションを安全に削除できます。

プロモーション・メディアの表示

プロモーション・メディア（テキストまたはイメージ）を作成する場合は、Web サイトを担当しているページ開発者が、コンテンツを表示するための標準的な手法を使用して、適切なサイト・ページにプロモーション・メディアを表示できます。たとえば、ページ開発者は、スロット要素を使用してプロモーション・メディアを表示するシナリオを設定できます。プロモーション・メディアの作業の詳細は、『ATG Web Commerce マーチャンダイジング・ガイド』を参照してください。

クーポン・プロモーションの設定

Core Commerce では、クーポンを一種のプロモーションとして扱います（クーポンは「要求可能なプロモーション」と呼ばれることがあります）。クーポンは新規または既存のプロモーションにリンクできます。

詳細は、『ATG Web Commerce マーチャンダイジング・ガイド』の説明に従ってクーポンを作成します。独立クーポンまたはバッチ・クーポンのいずれかを作成できます。クーポンを作成するとき、クーポン名、クーポンの開始日および終了日、1 つのオーダーに許可されるクーポン数、および顧客がクーポンを要求するために入力するオプションのクーポン・コードなど、クーポンのパラメータを作成します。クーポンを既存のプロモーションまたは新規のプロモーションに関連付けます。これによって、特定の品目が 20%オフになったり、品目を 3 個購入するとオーダー全体が$10 オフになるなどのクーポン基準が指定されます。

ページ開発者またはアプリケーション開発者は、顧客がコードを入力できるフォーム・フィールドを設定します。たとえば、開発者は、精算ページにフィールドを追加し、そのフィールドに「クーポン・コードがあれば、ここに入力してください」というラベルを付けることができます。開発者は、次に、このフィールドを、Core Commerce のクーポンを処理する部分（最初は `CouponFormHandler` コンポーネント）にフックアップします。フォームの作業の詳細は、『ATG Web Commerce Page Developer's Guide』にあります。

ページ開発者はクーポンを使用してほしい顧客に送信する E メール・メッセージ（JSP）を設定します。そのメッセージには、クーポンの要求コードおよびメッセージに含めたい任意の追加のテキストが含まれます。E メール・メッセージを設定する方法については、『ATG Web Commerce パーソナライゼーション・ガイド』の*ターゲット設定された E メールの作業*を参照してください。

メッセージを受け取ってほしい個人（つまり、クーポンを使用してほしい個人）のリストを定義します。それを実行する 1 つの方法は、メッセージを受信する個人のグループを定義する要素と送信するメッセージを指定する E メールの送信要素が含まれたシナリオを作成し、使用可能にすることです。シナリオの作業の詳細は、『ATG Web Commerce パーソナライゼーション・ガイド』を参照してください。

顧客はクーポンを使用するときに、次のことを実行します。

1. 顧客は Web サイトを訪問し、クーポンが適用される製品またはサービスを注文します。注文のプロセス中に、顧客はページ上の適切なフィールドにクーポンの要求コードを入力します。
2. Core Commerce は、要求可能リポジトリおよび該当するプロモーション・リポジトリ内の項目の両方を参照して、たとえば、要求コードの有効性や適用される割引の額やタイプを確認します。

クーポンに使用されるリポジトリの詳細は、『ATG Web Commerce Programming Guide』を参照してください。

E メール経由のプロモーションの配信

シナリオを使用すると、E メール経由でプロモーションの適用を受ける個人を定義できます。シナリオの詳細は、『ATG Web Commerce パーソナライゼーション・ガイド』の*ターゲット設定された E メールの作業*を参照してください。

URL 経由のプロモーションの配信

シナリオやクーポンを経由するなど、様々な方法でプロモーションをサイト訪問者に配信できます。前の各項では、プロモーションを作成し、それらの配信手段を使用する方法を説明しました。この項では、JSP 上の URL 経由という、プロモーションをサイト訪問者に配信する別の方法について説明します。この方法では、提供するプロモーションの ID を URL に埋め込みます。サイト訪問者がその URL をクリックすると、訪問者の `activePromotions` プロファイル・プロパティにプロモーションが追加されます。

URL 経由でプロモーションを配信するには、次の手順を実行します。

1. サーブレットの.properties ファイルで、サーブレットの enabled プロパティを true に設定することによって/atg/dynamo/servlet/dafpipeline/PromotionServlet を使用可能にします。Core Commerce は、デフォルトで、要求処理パイプラインに PromotionServlet を挿入しますが、サーブレットを使用するには、ユーザーがサーブレットを使用可能にする必要があります。
2. 次のようなアンカー・タグを使用して、目的の JSP に URL を含めます。

```
<dsp:a href="../../samplepage.jsp" encode="true"><dsp:param
      value="promo10102" name="PROMO"/>Click here to get a 20% discount
      on shirts.</dsp:a>
```

   ../../samplepage.jsp はリンクするページであり、PROMO 要求パラメータの値は /atg/commerce/pricing/Promotions リポジトリ内のプロモーションの ID です。

   アンカー・タグ内の encode=true が必須であることに注意してください。これによって、PROMO パラメータ・サブタグが query パラメータではなく、URL パラメータとして URL 内にエンコードされます。次に、PromotionServlet が要求の getURLParameter() を呼び出して PROMO パラメータ値を取得します。

   プロモーションの品目記述子が PromotionServlet.promotionItemDescriptorNames 内の品目記述子の 1 つである必要があることにも注意してください。

要求が PromotionServlet に渡されたとき、サーブレットが使用不可になっていると、サーブレットは単に要求をパイプライン内の次のサーブレットに渡します。サーブレットが使用可能になっていれば、サーブレットは要求の URL パラメータ内の PROMO パラメータを探して、関連付けられたプロモーション ID を取得します。次に、サーブレットは、取得した ID を持ち、品目記述子が PromotionServlet.promotionItemDescriptorNames に含まれているプロモーション・リポジトリ内のプロモーションを探します。プロモーションが存在すれば、PromotionServlet は、プロファイルが永続的であるか、またはプロモーションの匿名顧客に付与 giveToAnonymousProfiles プロパティが true に設定されているかを確認します。どちらかの条件が true であれば、プロモーションは顧客の activePromotions プロファイル・プロパティに追加されます。

GWPFormHandler の使用

GWPFormHandler は、顧客が自分のショッピング・カートでギフト品目を選択できるようになるページ・コーダを提供します。フォーム・ハンドラは/atg/commerce/promotion/GWPFormHandler にあり、atg.commerce.promotion.GWPFormHandler クラスに基づいています。

このフォーム・ハンドラは、GWPManager クラスの次のメソッドにアクセスできます。

- makeGiftSelection - 顧客がギフト品目を選択したり、または選択した品目を同じ数量に置換できるようにします。
- handleRemoveSelectableQuantity - 指定した数量のギフト品目（プレースホルダ）をオーダーから削除します。
- handleRemoveAllSelectableQuantity - 選択可能な数量をオーダーからすべて削除します。

GWPManager の詳細は、『ATG Web Commerce Programming Guide』を参照してください。

フォーム・ハンドラは、必要に応じて追加機能を提供するように拡張できます。

フォーム・ハンドラには、次の構成可能なプロパティが含まれています。

- currentSelectedItemId - 既存のギフト選択を置換する品目 ID をここに指定します。
- promotionId - 選択を行う購入時ギフト・プロモーション。
- giftHashCode - 選択を行うプロモーション内のギフト品目。
- productId - 新しい選択の製品 ID。
- skuId - 新しい選択の SKU ID。
- quantity - 新しい選択の金額。
- order - 構成されていない場合は、ショッピング・カートの現在のオーダーが使用されます。デフォルト構成では、現在のオーダーが使用されます。
- shippingGroup - 新しい選択を追加する出荷グループ。指定されていない場合は、オーダーの最初の出荷グループが使用されます。
- commerceItemType - 新しい選択の商品タイプ。
- siteId - 新しい選択のサイト。
- makeGiftSelectionSuccessURL - 選択が成功した場合にリダイレクトされる URL。
- makeGiftSelectionErrorURL - 選択が成功しなかった場合にリダイレクトされる URL。

フォーム・ハンドラには、新しいギフトの選択によって、選択した品目を置換する前に、削除された数量と失敗した数量のいずれを先に置換するかを制御する、2 つのフラグも含まれています。

- replaceRemovedQuantity
- replaceFailedQuantity

デフォルトでは、両方のフラグが false となります。

詳細は、『ATG Web Commerce Platform API Reference』を参照してください。

グループ品目割引の設定

グループ品目割引では、割引または固定価格での製品のグループを提供できます。このタイプのプロモーションは、顧客が品目に対する割引の適用を受ける品目割引プロモーションに対してのみ適用されることに注意してください。

同じプロモーションの一部として割引対象となる品目がグループ化され、グループ ID によって識別されます。この ID は、品目価格詳細情報に作成される価格調整に追加されます。グループ割引カルキュレータによって、グループ化された製品の割引タイプが指定されます。

Merchandising でグループ割引テンプレートがプライシング・モデル記述言語（PMDL）にアクセスし、グループ索引を保守する groupIterator プロパティが生成され、索引はグループごとに一意になります。グループ割引テンプレートは getGroupDiscount.pmdt テンプレート・ファイルを使用して定義されます。PMDL およびプロモーション・テンプレートの詳細は、『ATG Web Commerce Programming Guide』を参照してください。Merchandising UI でのプロモーション・テンプレートの使用の詳細は、『ATG Web Commerce マーチャンダイジング・ガイド』を参照してください。

グループ割引ショッピング・カート・ページの構成

ページ・コーダとして、UnitPriceDetailDroplet および CartModifierFormHandler を使用して、グループ割引品目を表示するショッピング・カート・ページを構成できます。

ショッピング・カート・ページが表示されると、JSP ページが UnitPriceDetailDroplet を呼び出します。このドロップレットは、PricingTool からオーダーの価格設定を要求します。PricingTool は、グループのグループ索引および価格設定モデルへの参照を提供する UnitPriceBean オブジェクトを生成し、カート内の情報を品目またはグループ（あるいはその両方）別にグループ化できるようにします。次に PricingTool は UnitPriceDetailDroplet に単価情報を返し、その情報を JSP ページで使用できるようにします。UnitPriceDetailDroplet は requestedBeans および Order パラメータを使用してグループ割引情報を設定します。

UnitPriceDetailedDroplet の詳細は、UnitPriceDetailDroplet の項および『ATG Web Commerce Platform API Reference』を参照してください。

CartModifierFormHandler を使用すると、catalogIds または commerceItemIds に基づいて商品を変更できます。商品は複数行にわたる品目に分割できるため、グループ割引は複数行にわたって表示できます。

CartModifierFormHandler を使用したショッピング・カートへの品目の追加は、ショッピング・カートへの品目の追加で説明されています。グループ割引プロモーションを処理するための JSP ページの構成の詳細は、ショッピング・カートからの品目の削除の項を参照してください。CartModifierFormHandler の詳細は、『ATG Web Commerce Programming Guide』を参照してください。

6 コスト・センターの管理

コスト・センターは、特定のサイト顧客が、自社の組織の特定の一部をコスト・センターとして指定することによって、自社の内部コストを追跡できるようにする Oracle Commerce Core Commerce 機能です。Core Commerce は、個々のコスト・センターにどの項目とオーダーが所属しているかを追跡します。

たとえば、保険会社の個々の従業員がログオンして事務用品を購入できるように、保険会社が事務用品サイトにアカウントを設定するとします。この保険会社は Web サイト経由で事務用品を注文する個々の部署に対応する複数のコスト・センターを設定できます。保険会社の従業員がログインして事務用品を購入するときは、自分の所属する部署を指定します。そうすれば、保険会社は、部署別のコストを追跡し、コスト関連のレポートを実行できます。

コスト・センター機能には 2 つの主要な部分があります。1 つはコスト・センターをプロファイルに割り当てる部分で、もう 1 つはオーダー内でコストをコスト・センターに割り当てる部分です。コスト・センターをプロファイルに割り当てる方式を使用して、サイトは、特定のユーザーがどのコスト・センターにコストを割り当てられるかを定義できます。ユーザーにはコスト・センターのリストが割り当てられます。コスト・センターにコストを追加することで、ユーザーは、自分のプロファイル内のデータを使用して自分のオーダー内に `CostCenter` オブジェクトを作成できます。さらに、ユーザーは、品目コスト、出荷コストまたは税コストをコスト・センターに割り当てることができます。

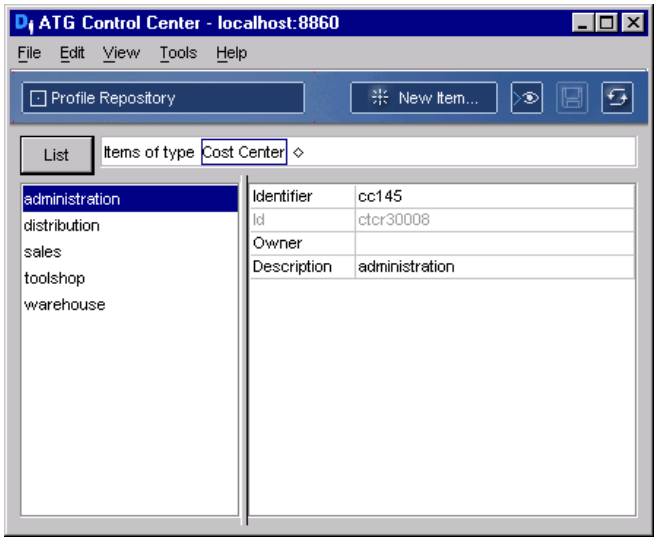
この章は次の項から構成されています。

既存のコスト・センターの表示

ACC で使用可能なコスト・センターを表示するには、次の手順を実行します。

1. メイン・タスク・バーから「個人および組織」を選択します。
2. 「プロファイル Repository」をクリックします。
3. 検索メニューから、項目タイプ（Item of type）でコスト・センター(Cost Center)を選択し、「リスト」をクリックします。

使用可能なすべてのコスト・センターのリストが表示されます。



新規コスト・センターの追加

ACC を使用して新規コスト・センターを追加するには、次の手順を実行します。

1. 既存のコスト・センターをプロファイル・リポジトリ内で検索します。詳細は、既存のコスト・センターの表示を参照してください。
2. 「新規項目」ボタンをクリックします。
3. 「新規項目」をクリックします。
4. 識別子、所有者および摘要を入力します。
5. 「OK」をクリックします。使用可能なコスト・センターのリストに新規コスト・センターが表示されます。

ユーザーへのデフォルト・コスト・センターの割当

コスト・センターをユーザー・プロファイルに割り当てることで、特定のユーザーがどのコスト・センターにコストを割り当てられるかを定義できます。ユーザーに使用可能なコスト・センターのリストを割り当てることができます。ACC を使用してユーザーにコスト・センターを割り当てるには、次の手順を実行します。

1. メイン ACC ナビゲーション・バーから「個人および組織」を選択します。
2. 「個人および組織」の選択肢から「プロファイル Repository」を選択します。
3. 項目タイプ（Item of type）でユーザー(User)を選択し、「リスト」をクリックして、すべてのユーザーのリストを表示します。
4. リストからユーザーを選択します。ユーザーの情報が画面のメイン・セクションに表示されます。
5. プロファイル情報の「請求および出荷」セクション内のデフォルト・コスト・センター・フィールドにコスト・センターを追加します。「…」ボタンをクリックすることによって、使用可能なコスト・センターのリストからコスト・センターを選択できます。
6. 「保存」をクリックして、コスト・センターをプロファイルに保存します。

プロファイル内のコスト・センターの追加、変更、削除

CommerceProfileTools クラスを介してユーザーのプロファイルのコスト・センター・データを追加、変更および削除できます。コスト・センターはユーザーまたは組織（またはサブ組織）のどちらかに割り当てられます。コスト・センターは識別子と摘要から構成されます。

プロファイル内のコスト・センターを追加、編集または削除する方法は 2 つあります。

- 顧客が自分のプロファイル内でコスト・センターを操作できるようにする Web フォームを設定する方法
- サイト管理者が ATG Control Center を使用してコスト・センター情報を保守する方法

CommerceProfileTools のメソッドを使用することで、プロファイルのコスト・センターを追加、変更および削除できます。

顧客が自分のプロファイル内のコスト・センターの変更に使用できる HTML フォームの設定は、CommerceProfileFormHandler クラスを使用して行われます。次の表で CommerceProfileFormHandler クラスのプロパティを説明します。

プロパティ	説明
AddCostCenterIdentifier	リポジトリに追加される新規コスト・センターの識別子、つまり名前を格納します。
AddCostCenterDescription	追加される新規コスト・センターの摘要を格納します。
EditCostCenterIdentifier	変更されるコスト・センターの識別子を格納します。
EditCostCenterDescription	EditCostCenterIdentifier によって識別されるコスト・センターの摘要の新しい値となる摘要を格納します。
RemoveCostCenterIdentifier	削除されるコスト・センターの識別子を格納します。
DefaultCostCenter	特定のコスト・センターをユーザーのデフォルト・コスト・センターにするかどうかを決める情報を格納するブール値。
AddCostCenterSuccessURL	ユーザーがコスト・センターの追加に成功した後、誘導される先の JSP の名前を格納します。
AddCostCenterErrorURL	ユーザーがコスト・センターの追加を試みている間にエラーが発生した後、誘導される先の JSP の名前を格納します。
EditCostCenterSuccessURL	ユーザーがコスト・センターの編集に成功した後、誘導される先の JSP の名前を格納します。
EditCostCenterErrorURL	ユーザーがコスト・センターの編集を試みている間にエラーが発生した後、誘導される先の JSP の名前を格納します。
RemoveCostCenterSuccessURL	ユーザーがコスト・センターの削除に成功した後、誘導される先の JSP の名前を格納します。
RemoveCostCenterErrorURL	ユーザーがコスト・センターの削除を試みている間にエラーが発生した後、誘導される先の JSP の名前を格納します。

コスト・センターの追加、編集および削除は、handleAddCostCenter、handleEditCostCenter および handleRemoveCostCenter を呼び出すことによって実行できます。

次の JSP の例では、顧客がコスト・センターの追加に使用できるフォームを作成します。

```
<h1>Add Cost Center</h1>
<dsp:form action="done.jsp" method="post">
  <dsp:input bean="ProfileFormHandler.AddCostCenterSuccessURL" value="success.jsp"
             type="hidden"/>
  <dsp:input bean="ProfileFormHandler.AddCostCenterErrorURL" value="error.jsp"
             type="hidden"/>
  Name: <dsp:input bean="ProfileFormHandler.AddCostCenterIdentifier"
             type="text"/><P>
  Description: <dsp:input bean="ProfileFormHandler.AddCostCenterDescription"
             type="text"/><P>
  <dsp:input bean="ProfileFormHandler.DefaultCostCenter" value="true"
             type="checkbox"/> Make default<P>
  <dsp:input bean="ProfileFormHandler.AddCostCenter" value="Add Cost Center"
             type="submit"/><P>
</dsp:form>
```

次の JSP の例では、顧客がコスト・センターの編集に使用できるフォームを作成します。

```
<h1>Edit Cost Center</h1>
<dsp:form action="done.jsp" method="post">
  <dsp:input bean="ProfileFormHandler.EditCostCenterSuccessURL"
           value="success.jsp" type="hidden"/>
  <dsp:input bean="ProfileFormHandler.EditCostCenterErrorURL" value="error.jsp"
             type="hidden"/>
  Name: <dsp:input bean="ProfileFormHandler.EditCostCenterIdentifier"
                  type="text"/><P>
  New Description: <dsp:input bean="ProfileFormHandler.EditCostCenterDescription"
                              type="text"/><P>
  <dsp:input bean="ProfileFormHandler.DefaultCostCenter" value="true"
             type="checkbox"/>
  Make default<P>
  <dsp:input bean="ProfileFormHandler.EditCostCenter" value="Edit Cost Center"
             type="submit"/><P>
</dsp:form>
```

次の JSP の例では、顧客がコスト・センターの削除に使用できるフォームを作成します。

```
<h1>Remove Cost Center</h1>
<dsp:form action="done.jsp" method="post">
  <dsp:input bean="ProfileFormHandler.RemoveCostCenterSuccessURL"
           value="success.jsp" type="hidden"/>
  <dsp:input bean="ProfileFormHandler.RemoveCostCenterErrorURL" value="error.jsp"
```

```
            type="hidden"/>
  Name: <dsp:input bean="ProfileFormHandler.RemoveCostCenterIdentifier"
                   type="text"/><P>
  <dsp:input bean="ProfileFormHandler.RemoveCostCenter" value="Remove Cost Center"
            type="submit"/><P>
</dsp:form>
```

オーダーへのコスト・センターの追加

コスト・センターが実装されているサイトでサイト顧客が製品を購入しているとき、顧客はオーダーにコスト・センターを追加できます。CostCenterDroplet サーブレット Bean を使用して、ユーザーが選択できるオプションとしてユーザーに割り当られているすべてのコスト・センターを表示できます。ユーザーに割り当てられているコスト・センターがない場合、リポジトリは、ユーザーの親組織を検索し、親組織のコスト・センターをオプションとして使用するように構成されています。その組織にもコスト・センターがない場合、リポジトリはその組織の親組織、さらにその親組織と検索を続けます。

ユーザーがコスト・センターをオーダーに割り当てるときは、コスト・センター・リポジトリ ID がオーダー・リポジトリ項目の一部として含まれます。

コスト・センターへの品目の追加

1 つのオーダーに複数のコスト・センターが割り当てられている場合は、オーダーに含まれているどの品目がどのコスト・センターに属するかを選択する必要があります。品目は、CostCenterCommerceItemRelationship と呼ばれる一種の関係を介してコスト・センターに割り当てられます。

コストを商品より出荷グループまたはオーダーに関連付けたほうが都合がよい場合は、CostCenterShippingGroupRelationship および CostCenterOrderRelationship を使用できます。そうすることで、出荷コストと税コストの追跡も可能になります。

コスト・センターによるオーダーの追跡

コスト・センターの主目的は組織が自社のコストをより細かく追跡できるようにすることであるため、特定のコスト・センターと関連付けられているオーダーのリストを取得する検索メソッドが用意されています。

CostCenterManager クラスの getOrdersforCostCenter()メソッドを使用して特定のコスト・センターに関連付けられたオーダーを取得します。

コスト・センター・クラス

次の項では、コスト・センターに関連するクラスについて簡単に説明します。これらのクラスの詳細は、『ATG Web Commerce Platform API Reference』を参照してください。

CostCenter インタフェース

CostCenter インタフェースはコスト・センターに含まれているすべての情報を表します。CostCenter インタフェースには次のプロパティが含まれています。

- CostCenterClassType
- Identifier
- Description
- Amount

CostCenterImpl は CostCenter インタフェースのデフォルト実装です。

CostCenterManager

CostCenterManager には、オーダーのコンテキスト内でコスト・センターを操作するためのメソッドが含まれています。このクラスには、CostCenter オブジェクトおよび CostCenterRelationship オブジェクトを追加、削除および編集するためのメソッドと、コスト・センターに基づいてオーダー情報を取得するメソッドが含まれています。

CostCenterContainer

コスト・センターのグループへのアクセスを処理します。CostCenterContainerImpl は CostCenterContainer のデフォルト実装になります。このインタフェースは、オーダー内のコスト・センターのリストを管理するためのメソッドを備えています。

CostCenterRelationship

CostCenterRelationship は、品目とコスト・センターとの関係の一部を表します。CostCenterRelationship インタフェースは 2 つのプロパティから構成されています。

- CostCenter: 該当するコスト・センターを参照します。
- Amount: 特定のオーダー内のそのコスト・センターに属する合計価格を示します。

CostCenterCommerceItemRelationship

CostCenterCommerceItemRelationship は商品と商品の属するコスト・センターとの関係を定義します。このクラスは CostCenterRelationship と CommerceItemRelationship の両方を実装します。この関係には 4 つのタイプがあります。

- Amount: 品目の合計コストのうちどれだけを特定のコスト・センターに属すると見なすべきかを Amount プロパティが指示することを示します。
- AmountRemaining: 別の関係にある異なるコスト・センターに割り当てられていない品目のすべてのコストが特定の関係にあるコスト・センターに属すると見なされることを示します。
- Quantity:特定の商品のうち、いくつをコスト・センターに割り当てるべきかを Quantity プロパティが指示することを示します。
- QuantityRemaining: 別の関係にある異なるコスト・センターに割り当てられていない品目のうちすべての品目が特定の関係にあるコスト・センターに属すると見なされることを示します。

CostCenterShippingGroupRelationship

CostCenterShippingGroupRelationship は、コスト・センターを出荷グループ内の個々の品目にではなく、出荷グループに関連付けるために使用されます。この関係を形成することによって、出荷手数料もコスト・センターに割り当てることができます。このクラスは CostCenterRelationship と ShippingGroupRelationship の両方を実装します。

CostCenterShippingGroupRelationship には 2 つのタイプがあります。

- ShippingAmount
- ShippingAmountRemaining

CostCenterOrderRelationship

CostCenterOrderRelationship は、コスト・センターを出荷グループ内の個々の品目や特定の出荷グループにではなく、オーダーに関連付けるために使用されます。このクラスは CostCenterRelationship と OrderRelationship の両方を実装します。この関係を形成することによって、税と出荷手数料もコスト・センターに割り当てることができます。

この関係には 4 つのタイプがあります。

- TaxAmount
- TaxAmountRemaining
- OrderAmount
- OrderAmountRemaining

CostCenterFormHandler フレームワークの使用

CostCenterFormHandler フレームワークは、顧客が、購買プロセス中に複雑な CostCenter 情報をユーザーから収集することを可能にします。このフレームワークの主な目的は、ユーザーが自社の認可されたコスト・センターをオーダーの様々な CommerceIdentifiers に関連付けられるようにすることです。

この情報の処理を容易にするために、CostCenterFormHandler フレームワークは次のヘルパー・クラスを利用します。

- CommerceIdentifierCostCenter
- CommerceIdentifierCostCenterContainer
- CostCenterMapContainer
- CostCenterContainerService
- CostCenterDroplet
- CostCenterFormHandler

CommerceIdentifierCostCenter

CommerceIdentifierCostCenter オブジェクトには、名前によって参照されるコスト・センターと CommerceIdentifiers を関連付けるのに必要な情報が格納されます。このオブジェクトには次のプロパティが含まれています。

プロパティ名	タイプ
CommerceIdentifier	CommerceIdentifier
CostCenterName	String
RelationshipType	String
Amount	double
SplitAmount	double
Quantity	long
SplitQuantity	long
SplitCostCenterName	String

CommerceIdentifierCostCenterContainer

CommerceIdentifierCostCenterContainer インタフェースは、様々なオーダーの CommerceIdentifiers に関連付けられたすべての CommerceIdentifierCostCenters を追跡します。

CostCenterMapContainer

CostCenterMapContainer インタフェースは、すべてのユーザーのコスト・センターを名前によって追跡し、デフォルト・コスト・センターも追跡します。

CostCenterContainerService

CostCenterContainerService は、両方のコンテナ・インタフェースを実装し、便利なセッション限定コンポーネントから構成される GenericService です。

CostCenterDroplet

CostCenterDroplet サーブレット Bean には、次のタスクを担うサービス・メソッドを備えた要求限定コンポーネントが含まれています。

- CostCenter の初期化 - ユーザーの認可されたコスト・センターが作成され、CostCenterMapContainer に追加されます。
- CommerceIdentifierCostCenter の初期化 - 現在のオーダーに固有の新規 CommerceIdentifierCostCenter インスタンスが作成され、CommerceIdentifierCostCenterContainer に追加されます。

初期化中、CostCenterDroplet は、オプションで、個々の CommerceIdentifier タイプ(たとえば、個々の CommerceItem、ShippingGroup、オーダーおよび税)につき 1 つの CommerceIdentifierCostCenter オブジェクトを作成します。そのオブジェクトは次のデフォルト・プロパティを持ちます。

プロパティ	説明
CommerceIdentifier	CommerceItem、ShippingGroup または Order を参照するように設定されます。

プロパティ	説明
RelationshipType	適切な RelationshipTypes 文字列プロパティ CCAMOUNT_STR、CCQUANTITY_STR、CCSHIPPINGAMOUNT_STR、CCORDERAMOUNT_STR、または CCTAXAMOUNT_STR に設定されます。CostCenterCommerceItemRelationships は、ブール型のドロップレット・パラメータ useAmount が true でないかぎり、デフォルトで CCQUANTITY_STR に設定されます。
CostCenterName	CostCenterMapContainer の DefaultCostCenterName に設定されます。
Amount	PriceInfo が存在すれば、CommerceIdentifier の PriceInfo の Amount プロパティに設定されます。
Quantity	CommerceItem の Quantity プロパティに設定されます。

たとえば、単純なコスト・センター・ページで、1 つのコスト・センターをオーダー全体に割り当てたり、オーダーのコスト全体を複数のコスト・センターに分割することを顧客に許可することがあります。その単純なページでは、ユーザーがコスト・センターを CommerceItems、ShippingGroups または Tax に割り当てることはできません。

より高度な CostCenter ページでは、CostCenter の関連付けを CommerceItem、ShippingGroup および Tax に割り当てて、分割することを顧客に許可します。この動作は、異なる初期化パラメータを持つ同じ CostCenterDroplet サーブレット Bean によって実装されます。CostCenterDroplet の詳細は、付録 A: Core Commerce サーブレット Bean を参照してください。

CostCenterFormHandler

CostCenterFormHandler は PurchaseProcessFormHandler を拡張します。このハンドラは 2 つのハンドラ・メソッドを持つ要求限定コンポーネントです。

- handleSplitCostCenters

  このハンドラは、CommerceIdentifierCostCenter の splitCostCenter プロパティ、splitAmount プロパティおよび splitQuantity プロパティを使用して、余分な CommerceIdentifierCostCenter オブジェクトを数量または金額で分割します。

  ユーザーが、フォームで、元の CommerceIdentifier の金額 100 ドルのうち 50 ドルを別のコスト・センターに分割することがあります。これによって新しい CommerceIdentifierCostCenter オブジェクトが作成され、元の CommerceIdentifier の合計額になるように、新旧両方の CommerceIdentifierCostCenter オブジェクトの金額が調整されます。

- handleApplyCostCenters

  このハンドラの起動がこのフレームワーク全体の目的ですが、このハンドラは、コンテナ内にある情報を取得し、それを現在のオーダーに適用します。ユーザーの作成した CommerceIdentifierCostCenter の関連付けが、まず精査され、CommerceIdentifierCostCenter の RelationshipType に基づいて OrderManager ファミリ内で適切なビジネス・メソッドが呼び出されます。次に、CostCenterMapContainer の任意の DefaultCostCenterName を使用して、残っているオーダーの金額があれば、それがコスト・センターに追加されているかどうかを判断します。この動作があるため、他のコス

ト・センターによって明示的に扱われていない残りのオーダー金額にデフォルト・コスト・センターを適用するアプリケーションを容易に開発できます。

次に、CostCenterFormHandler の使用方法を示すコード例を示します。この結果生成される JSP は、顧客のオーダーに含まれている品目のリストを表示し、個々の品目の横にテキスト・ボックスと利用可能なコスト・センターのドロップダウン・リストを表示します。顧客はこのページを使用して品目の数量を指定し、コスト・センターを品目に関連付けることができます。

```
<dsp:form action="cost_centers_line_item.jsp" method="post">
 <tr>
  <td><br>
  <table border=0 cellpadding=6 cellspacing=0>
   <tr>
    <td></td>
    <td colspan=2><span class="big">Cost Centers</span>
    <dsp:include page="../common/FormError.jsp"></dsp:include></td>
   </tr>

   <tr valign=top>
    <td width=40><dsp:img hspace="20" src="../images/d.gif"/></td>
    <td>

    <table border=0 cellpadding=4 cellspacing=1 width=85%>
     <tr>
      <td colspan=13><span class=help>Assign each line item to a cost center. You
       can also divide a line item between cost centers by entering the number of
       items to assign to the new cost center.
      </span></td>
     </tr>

     <tr valign=bottom bgcolor="#666666">
      <td colspan=2><span class=smallbw>Part #</span></td>
      <td colspan=2><span class=smallbw>Name</span></td>
      <td colspan=3><span class=smallbw>Qty</span></td>
      <td colspan=2><span class=smallbw>Qty. to move</span></td>
      <td colspan=2><span class=smallbw>Price</span></td>
      <td colspan=2><span class=smallbw>Cost Center</span></td>
      <!--<td colspan=2><span class=smallbw>Current Cost Center</span></td>-->
     </tr>
     <dsp:droplet name="ForEach">
      <dsp:param param="order.commerceItems" name="array"/>
      <dsp:oparam name="output">
       <dsp:setvalue paramvalue="element" param="commerceItem"/>
       <dsp:droplet name="BeanProperty">
        <dsp:param param="ciccMap" name="bean"/>
        <dsp:param param="commerceItem.id" name="propertyName"/>
        <dsp:oparam name="output">
         <dsp:setvalue paramvalue="propertyValue" param="ciccList"/>
         <dsp:droplet name="ForEach">
          <dsp:param param="ciccList" name="array"/>
```

```
                <dsp:oparam name="output">
                 <!-- begin line item -->
                 <tr valign=top>
                  <td><dsp:valueof param="commerceItem.catalogRefId"/></td>
                  <td></td>
                  <td><dsp:a href="../catalog/product.jsp?navAction=jump">
                      <dsp:param param="commerceItem.auxiliaryData.productId"
                                 name="id"/>
                      <dsp:valueof
                           param="commerceItem.auxiliaryData.productRef.displayName"/>
                      </dsp:a></td>
                  <td></td>

                  <td> </td>
                  <td align=right><dsp:valueof param="element.quantity"/></td>
                  <td> </td>


                  <td>
                  <dsp:input
                       bean='<%="CostCenterDroplet.CostCenterMapContainer.
                                 CommerceIdentifierCostCenterMap." +
                                 request.getParameter("commerceItem.id")+
                                 "[param:index].splitQuantity"%>' size="4" value="0"
                                  type="text"/></td>
                  <td> </td>

                   <td align=right><dsp:valueof param="element.amount"
                                       converter="currency"/></td>
                  <td> </td>

                  <td>
                   <dsp:select bean='<%="CostCenterDroplet.CostCenterMapContainer.
                                         CommerceIdentifierCostCenterMap." +
                                         request.getParameter("commerceItem.id")+
                                         "[param:index].splitCostCenterName"%>'>
                    <dsp:droplet name="ForEach">
                     <dsp:param param="costCenters" name="array"/>
                     <dsp:oparam name="output">
                      <dsp:getvalueof id="option178" param="element"
                           idtype="java.lang.String">
<dsp:option value="<%=option178%>"/>
</dsp:getvalueof><dsp:valueof param="element"/>
                     </dsp:oparam>
                    </dsp:droplet>
                    </dsp:select>
                  </td>
                  <td> </td>
```

```
        <td> 
        </td>
        <td>
          <dsp:valueof param="element.costCenterName">
            <dsp:valueof bean="CostCenterDroplet.
                               CostCenterMapContainer.defaultCostCenterName"/>
           </dsp:valueof>

        </td>
       </tr>
       <!-- end line item -->
      </dsp:oparam>
     </dsp:droplet>
    </dsp:oparam>
   </dsp:droplet>
 </dsp:oparam>
</dsp:droplet>

<dsp:droplet name="ForEach">
 <dsp:param param="order.shippingGroups" name="array"/>
 <dsp:oparam name="output">
  <dsp:setvalue paramvalue="element" param="shippingGroup"/>
  <dsp:droplet name="BeanProperty">
   <dsp:param param="ciccMap" name="bean"/>
   <dsp:param param="shippingGroup.id" name="propertyName"/>
   <dsp:oparam name="output">
    <dsp:setvalue paramvalue="propertyValue" param="ciccList"/>
    <dsp:droplet name="ForEach">
     <dsp:param param="ciccList" name="array"/>
     <dsp:oparam name="output">
      <!-- begin shipping line item -->
      <tr valign=top>
       <td colspan=7>
       <dsp:valueof param="shippingGroup.description"/></td>
       <td></td>
       <td></td>

       <td align=right><dsp:valueof param="element.amount"
                                converter="currency"/></td>
       <td></td>

       <td>
        <dsp:select bean='<%="CostCenterDroplet.CostCenterMapContainer.
             CommerceIdentifierCostCenterMap." +
             request.getParameter("shippingGroup.id")+
             "[param:index].splitCostCenterName"%>'>
        <dsp:droplet name="ForEach">
         <dsp:param param="costCenters" name="array"/>
         <dsp:oparam name="output">
```

```
                <dsp:getvalueof id="option279" param="element"
                     idtype="java.lang.String">
<dsp:option value="<%=option279%>"/>
</dsp:getvalueof><dsp:valueof param="element"/>
              </dsp:oparam>
             </dsp:droplet>
             </dsp:select>
            </td>
            <td> </td>
            <td> 
            </td>
            <td>
              <dsp:valueof param="element.costCenterName">
               <dsp:valueof bean="CostCenterDroplet.CostCenterMapContainer.
                    defaultCostCenterName"/>
            </dsp:valueof>
            </td>
           </tr>
           <!-- end shipping line item -->
          </dsp:oparam>
         </dsp:droplet>
        </dsp:oparam>
       </dsp:droplet>
      </dsp:oparam>
     </dsp:droplet>

             <dsp:droplet name="BeanProperty">
      <dsp:param param="ciccMap" name="bean"/>
      <dsp:param param="order.id" name="propertyName"/>
      <dsp:oparam name="output">
       <dsp:setvalue paramvalue="propertyValue" param="ciccList"/>
       <dsp:droplet name="ForEach">
        <dsp:param param="ciccList" name="array"/>
        <dsp:oparam name="output">
         <!-- begin tax line item -->


             <dsp:droplet name="Switch">
             <dsp:param param="element.RelationshipType" name="value"/>
             <dsp:oparam name="CCTAXAMOUNT">
   <tr valign=top>
         <td colspan=7>Tax</td>
         <td></td>
         <td></td>

         <td align=right><dsp:valueof param="element.amount"
         converter="currency"/></td>

         <td></td>
```

```
          <td>
           <dsp:select bean='<%="CostCenterDroplet.CostCenterMapContainer.
                                 CommerceIdentifierCostCenterMap." +
                                 request.getParameter("order.id")+
                                 "[param:index].splitCostCenterName"%>'>
           <dsp:droplet name="ForEach">
            <dsp:param param="costCenters" name="array"/>
            <dsp:oparam name="output">
             <dsp:getvalueof id="option376" param="element"
                         idtype="java.lang.String">
<dsp:option value="<%=option376%>"/>
</dsp:getvalueof><dsp:valueof param="element"/>
            </dsp:oparam>
           </dsp:droplet>
           </dsp:select>
          </td>
          <td> </td>
          <td> 
          </td>
          <td>
            <dsp:valueof param="element.costCenterName">
             <dsp:valueof bean="CostCenterDroplet.
                                CostCenterMapContainer.defaultCostCenterName"/>
          </dsp:valueof>
          </td>
          <td> </td>
         </tr>
         <!-- end tax line item -->
          </dsp:oparam>
          </dsp:droplet>

        </dsp:oparam>
       </dsp:droplet>
      </dsp:oparam>
     </dsp:droplet>

     <tr>
      <td colspan=13>
      <table border=0 cellpadding=0 cellspacing=0 width=100%>
       <tr bgcolor="#666666">
        <td><dsp:img src="../images/d.gif"/></td>
       </tr>
      </table>
      </td>
     </tr>
   </table>
   </td>
  </tr>
  <tr>
```

```
   <td></td>
     <dsp:input bean="CostCenterFormHandler.applyCostCentersSuccessURL"
                value="confirmation.jsp" type="hidden"/>
     <dsp:input bean="CostCenterFormHandler.splitCostCentersSuccessURL"
                value="cost_centers_line_item.jsp?init=false" type="hidden"/>
   <td><span class=help>You must save changes before continuing.</span><p>
<!--     <dsp:input bean="CostCenterFormHandler.order" type="hidden"
                beanvalue="ShoppingCart.current"/> -->
 <dsp:input bean="CostCenterFormHandler.splitCostCenters" value="Save changes"
            type="submit"/>
 <dsp:input bean="CostCenterFormHandler.applyCostCenters" value="Continue"
            type="submit"/>

   </td>
  </tr>
 </table>
 </td>
</tr>
</table>
</dsp:form>
```

7 シナリオでの Core Commerce 要素の使用

Oracle Commerce Core Commerce は、コマース・サイト用のシナリオを作成するときに使用できる多数のイベント要素、条件要素、処理要素を備えています。これらの Commerce 要素について説明するこの章は、次の項から構成されています。

シナリオでの Core Commerce イベント要素の使用
Commerce に付属のイベント要素について説明します。

シナリオでの Core Commerce 条件要素の使用
Commerce に付属の条件要素について説明します。

シナリオでの Core Commerce 処理要素の使用
Commerce に付属の処理要素について説明します。

シナリオを使用した製品のクロスセルとアップセル
顧客の現在のショッピング・カートに基づいて製品をクロスセルおよびアップセルするために使用できるシナリオ、シナリオ・テンプレート、シナリオをサポートする要素について説明します。

Motorprise Store に付属の要素については、『ATG Web Commerce ビジネス・コマース・リファレンス・アプリケーション・ガイド』を参照してください。Oracle Commerce Platform に付属の要素の詳細は、『ATG Web Commerce パーソナライゼーション・ガイド』を参照してください。

注意: この章では、読者が『ATG Web Commerce パーソナライゼーション・ガイド』の*シナリオの作成*に記載されている情報を読み、理解していることを前提としています。

シナリオでの Core Commerce イベント要素の使用

イベント要素はシナリオの「what」の部分です。イベント要素は、たとえば、「/shoes フォルダ内のページへの訪問」や「shoes カテゴリ内の製品に属する品目の表示」など、識別して、シナリオ内の次の要素のトリガーとして使用したい訪問者の行動を定義します。

この項では、シナリオで使用される Core Commerce に付属のイベント要素について説明します。これらの Commerce イベント要素は、Scenarios モジュールに付属している要素に追加されたものです。シナリオ、シナリオでのイベント要素の使用方法、非 Commerce イベント要素については、『ATG Web Commerce パーソナライゼーション・ガイド』の*シナリオの作成*を参照してください。

Core Commerce に付属のイベント要素では満たされない要件がある場合は、アプリケーション開発者がカスタム・イベント要素を作成できます。カスタム・イベント要素は、ATG Control Center（ACC）に表示され、それを標準のイベント要素と同様に使用できます。

開発者向けの注意: カスタム・イベント要素の作成の詳細は、『ATG Web Commerce Personalization Programming Guide』の*カスタム・イベント、アクションおよび条件のシナリオへの追加*を参照してください。カスタム Core Commerce 要素で使用するために Scenarios モジュールの構文を拡張する必要がある場合は、同じガイドの「*ATG Expression エディタの構成*」の *Commerce に関連した構文の構成*に関する項を参照してください。

Dynamo Message System メッセージとして送信されたイベントを `ScenarioManager` コンポーネントが受信すると、イベント要素がトリガーされることにも注意してください。この章の個々のイベント要素に関する説明では、要素をトリガーするメッセージおよびメッセージを送信する役割を担うコンポーネントまたはシステムの部分を明らかにします。特定のイベント要素について表示されるオプションのパラメータは、任意の親クラスから継承されたプロパティを含む、そのイベントを表すメッセージ Bean のプロパティに対応しています。Dynamo Message System の詳細は、『ATG Web Commerce Platform Programming Guide』の *Dynamo Message System* を参照してください。

次に、Core Commerce に付属のイベント要素をアルファベット順に説明します。

Approval Complete イベント

オーダーが承認プロセスを完了したとオーダー承認システムが判断するのをシステムが監視します。この要素内でオプションのパラメータを使用して、監視の対象となるオーダーのタイプをさらに細かく定義できます。

技術的な注意事項: この要素は、`ScenarioManager` が `sendApprovalCompleteMessage` プロセッサ（`atg.commerce.approval.processor.ProcSendApprovalCompleteMessage` クラス）によって送信された `ApprovalComplete` メッセージを受信するとトリガーされます。プロセッサは、オーダーが `checkApprovalComplete` パイプライン・チェーンを通過し、オーダー承認プロセスを完了したと判断されると、このメッセージを送信します。詳細は、『ATG Web Commerce Programming Guide』の*オーダー承認プロセスの管理*を参照してください。

Approval Required イベント

オーダーが権限のある承認者による承認を必要としているとオーダー承認システムが判断するのをシステムが監視します。この要素内でオプションのパラメータを使用して、監視の対象となるオーダーのタイプをさらに細かく定義できます。

技術的な注意事項: この要素は、`ScenarioManager` が `sendApprovalRequiredMessage` プロセッサ（`atg.commerce.approval.processor.ProcSendApprovalRequiredMessage` クラス）によって送信された `ApprovalRequired` メッセージを受信するとトリガーされます。プロセッサは、オーダーが `approveOrder` パイプライン・チェーンを通過し、承認を必要とすると判断されると、このメッセージを送信します。詳細は、『ATG Web Commerce Programming Guide』の*オーダー承認プロセスの管理*を参照してください。

Approval Update イベント

承認者によってオーダーが承認または否認されたことをオーダー承認システムが示すのをシステムが監視します。この要素内でオプションのパラメータを使用して、監視の対象となるオーダーのタイプをさらに細かく定義できます。

技術的な注意事項: この要素は、`ScenarioManager` が `ApprovalUpdate` メッセージを受信するとトリガーされます。`ApprovalUpdate` メッセージは、承認者がオーダーを承認したか否認したかに応じて、`orderApproved` パイプライン・チェーン内の `sendApprovalUpdateMessageForApproval` プロセッサ（`atg.commerce.approval.processor.ProcSendApprovalMessage` クラス）または `orderRejected` パイプライン・チェーン内の `sendApprovalUpdateMessageForRejection` プロセッサ

(atg.commerce.approval.processor.ProcSendApprovalMessage クラス)のどちらかによって送信されます。詳細は、『ATG Web Commerce Programming Guide』の*オーダー承認プロセスの管理*を参照してください。

FulfillOrderFragment

オーダー・フルフィルメント・システムが出荷グループ情報を処理する役割を担うシステムの部分(たとえば HardgoodFulfiller)に新規オーダーに関する出荷グループ情報を送信するのをシステムが監視します。この要素内でオプションのパラメータを使用して、監視の対象となる要素のタイプをさらに細かく定義できます。

技術的な注意事項: この要素は、ScenarioManager が OrderFulfiller によって送信された FulfillOrderFragment メッセージを受信するとトリガーされます。詳細は、『ATG Web Commerce Programming Guide』の*オーダー・フルフィルメント・フレームワークの構成*を参照してください。

Gift Purchased

顧客がギフトとして指定したオーダーが処理されるのをシステムが監視します。この要素内でオプションのパラメータを使用して、監視の対象となるギフト購入のタイプをさらに細かく定義できます。

技術的な注意事項: この要素は、ScenarioManager が SendGiftPurchasedMessage プロセッサ(atg.commerce.order.processor.ProcSendGiftPurchasedMessage クラス)によって送信された GiftPurchased メッセージを受信するとトリガーされます。

Inventory Threshold Reached

カタログ内の品目の在庫レベルが特定の値を下回るのをシステムが監視します。

技術的な注意事項: この要素は、ScenarioManager が InventoryManager によって送信された InventoryThresholdReached メッセージを受信するとトリガーされます。このメッセージは、特定の品目の stockLevel 値が stockThreshold 値を下回った(または backorderLevel 値が backorderThreshold を下回った、または preorderLevel が preorderThreshold を下回った)ことを示します。詳細は、『ATG Web Commerce Programming Guide』の*在庫のフレームワーク*を参照してください。

Invoice Is Created

請求書が請求書リポジトリに追加されるのをシステムが監視します。この要素内でオプションのパラメータを使用して、監視の対象となる請求書のタイプをさらに細かく定義できます。

技術的な注意事項: この要素は、ScenarioManager が InvoiceManager の addInvoice()メソッドによって呼び出されるパイプライン・チェーンによって送信された CreateInvoice メッセージを受信するとトリガーされます。詳細は、『ATG Web Commerce Programming Guide』の*請求書の生成*を参照してください。

Invoice Is Removed

請求書が請求書リポジトリから削除されるのをシステムが監視します。この要素内でオプションのパラメータを使用して、監視の対象となる請求書のタイプをさらに細かく定義できます。

技術的な注意事項: この要素は、ScenarioManager が InvoiceManager の removeInvoice()メソッドによって呼び出されるパイプライン・チェーンによって送信された RemoveInvoice メッセージを受信するとトリガーされます。詳細は、『ATG Web Commerce Programming Guide』の*請求書の生成*を参照してください。

Invoice Is Updated

請求書リポジトリ内の請求書が更新されるのをシステムが監視します。この要素内でオプションのパラメータを使用して、監視の対象となる請求書のタイプをさらに細かく定義できます。

技術的な注意事項: この要素は、`ScenarioManager` が `InvoiceManager` の `updateInvoice()`メソッドによって呼び出されるパイプライン・チェーンによって送信された `UpdateInvoice` メッセージを受信するとトリガーされます。詳細は、『ATG Web Commerce Programming Guide』の*請求書の生成*を参照してください。

Item Added to Order

顧客が新規オーダーまたは既存のオーダーに品目を追加するのをシステムが監視します。この要素内でオプションのパラメータを使用して、この要素をトリガーする品目とオーダーの組合せをさらに細かく定義できます。

例:*Item Added to Order and SKU named Silver Helmet* この要素は Silver Helmet というカタログ項目がオーダーに追加されるのを監視します。

技術的な注意事項: この要素は、`ScenarioManager` コンポーネントが `SendScenarioEvent` プロセッサ（`atg.commerce.order.processor.ProcSendScenarioEvent` クラス）によって送信された `ItemAddedToOrder` メッセージを受信するとトリガーされます。

Item Quantity Changed in Order

顧客がオーダー内の既存の品目の数量を変更するのをシステムが監視します。この要素内でオプションのパラメータを使用して、監視の対象となるオーダーまたは品目のタイプをさらに細かく定義できます。

技術的な注意事項: この要素は、`ScenarioManager` が `SendScenarioEvent` プロセッサによって送信された `ItemQuantityChanged` メッセージを受信するとトリガーされます。詳細は、『ATG Web Commerce Programming Guide』の *sendScenarioEvent パイプライン・チェーン*に関する項を参照してください。

Item Removed from Order

顧客が新規オーダーまたは既存のオーダーから品目を削除するのをシステムが監視します。この要素内でオプションのパラメータを使用して、この要素をトリガーする品目とオーダーの組合せをさらに細かく定義できます。

技術的な注意事項: この要素は、`ScenarioManager` が `SendScenarioEvent` プロセッサ（`atg.commerce.order.processor.ProcSendScenarioEvent` クラス）によって送信された `ItemRemovedFromOrder` メッセージを受信するとトリガーされます。

Modify Order

処理を受けるために発行されたオーダーにオーダー・フルフィルメント・システムが任意のタイプの変更を加えるのをシステムが監視します。この要素内でオプションのパラメータを使用して、この要素をトリガーする変更をさらに細かく定義できます。

技術的な注意事項: この要素は、`ScenarioManager` が、オーダーの変更を要求または処理するシステムの任意の部分（たとえば `OrderFulfiller`）によって送信される可能性のある `ModifyOrder` メッセージを受信するとトリガーされます。詳細は、『ATG Web Commerce Programming Guide』の*オーダー・フルフィルメント・フレームワークの構成*を参照してください。

Modify Order Notification

ModifyOrder メッセージ（前述の説明を参照）を受信した結果、または新規オーダーの処理の一部としてオーダーのステータスが変化したことをオーダー・フルフィルメント・システムが示すのをシステムが監視します。この要素内でオプションのパラメータを使用して、この要素をトリガーする通知をさらに細かく定義できます。

技術的な注意事項: この要素は、ScenarioManager が、オーダーを変更するシステムの任意の部分（たとえば HardGoodFulfiller）によって送信される可能性のある ModifyOrderNotification メッセージを受信するとトリガーされます。詳細は、『ATG Web Commerce Programming Guide』の*オーダー・フルフィルメント・フレームワークの構成*を参照してください。

Order Changes

オーダー・フルフィルメント・システムがオーダーのステータスに次のいずれかの変化が起きたことを示すのをシステムが監視します。

- オーダーが完了した。
- 使用できない品目がオーダーに含まれている。
- 顧客がオーダーをキャンセルした。
- たとえば、システムが顧客のクレジット・カード情報を処理できないため、お客様サービス担当者が適切な措置を講じるのを待機する必要があるなど、オーダーが待機状態にあり、オーダー・フルフィルメント・プロセスに関わっている人の処理を必要としている。

この要素内でオプションのパラメータを使用して、この要素をトリガーする変更をさらに細かく定義できます。

例:*Order Changes where Sub type is Order Was Removed* この要素は顧客がオーダーをキャンセルしたことを Core Commerce が示すのを監視します。

技術的な注意事項: この要素は、ScenarioManager が OrderChangeHandler によって送信された OrderModified メッセージを受信するとトリガーされます。詳細は、『ATG Web Commerce Programming Guide』の*オーダー・フルフィルメント・フレームワークの構成*を参照してください。

Order Submitted

顧客がオーダーの精算プロセスを完了するのをシステムが監視します。

技術的な注意事項: この要素は、ScenarioManager が SendFulFillmentMessage プロセッサ（atg.commerce.order.processor.ProcSendFulfillmentMessage クラス）によって送信された SubmitOrder メッセージを受信するとトリガーされます。

Orders Merged

匿名ユーザーが登録済ユーザーとしてログインしたときに、匿名ユーザーのショッピング・カートが登録済ユーザーの現在のショッピング・カートにマージされるのをシステムが監視します。この要素内でオプションのパラメータを使用して、監視の対象となるイベントのタイプをさらに細かく定義できます。

技術的な注意事項: この要素は、ScenarioManager が SendScenarioEvent プロセッサによって送信された OrdersMerged メッセージを受信するとトリガーされます。詳細は、『ATG Web Commerce Programming Guide』の *sendScenarioEvent パイプライン・チェーン*に関する項を参照してください。

Payment Group Changes

Core Commerce がオーダー内の支払情報にステータスの変化が起きたことを示すのをシステムが監視します。変化の発生源は顧客サービス・アプリケーションなどの外部システムです。この要素内でオプションのパラメータを使用して、この要素をトリガーするメッセージをさらに細かく定義できます。

技術的な注意事項: この要素は、`ScenarioManager` が `OrderChangeHandler` によって送信された `PaymentGroupModified` メッセージを受信するとトリガーされます。詳細は、*オーダー・フルフィルメント・フレームワークの構成*を参照してください。

Price Changed

オーダー価格がオーダー小計レベルで変化する（つまり、品目数量の変更または品目の追加や削除、あるいはその両方によって品目価格の合計が変化する）のをシステムが監視します。この要素内でオプションのパラメータを使用して、監視の対象となるオーダーのタイプを、新規オーダー価格などのように、さらに細かく定義できます。新規オーダー価格には、オーダーに対して実行された価格設定操作に応じて、オーダー小計（つまり、送料または税なし）またはオーダー合計のどちらかが反映されることに注意してください。

技術的な注意事項: この要素は、`ScenarioManager` が `PricingTools` オブジェクト（`atg.commerce.pricing.PricingTools` クラス）によって送信された `PriceChanged` メッセージを受信するとトリガーされます。`PricingTools` は、`PricingTools.generatePriceChangedEvents` プロパティが true であり、かつ、オーダーが再価格設定されたときにオーダーの小計価格が変化したことが判明すると、このメッセージを送信します。`PriceChanged` メッセージに含まれている `OrderPriceInfo` オブジェクト内の合計金額には、オーダーに対して実行された価格設定操作に応じて、オーダー小計またはオーダー合計のどちらかが反映されることに注意してください。詳細は、『ATG Web Commerce Programming Guide』を参照してください。

`PricingTools.createPriceChangedEvent` がオーダー以外のオブジェクト・タイプ（品目または出荷グループなど）に反応して、`PriceChangeType` を特定の選択肢（`PriceChanged` クラス内の静的定数）の 1 つに設定するように拡張できます。

Promotion Closeness Disqualification

ユーザーが「プロモーション・アップセル」に該当しなくなったことをシステムが検出します。特定の「プロモーション・アップセル」またはすべての「プロモーション・アップセル」を監視するように、この要素を構成できます。特定の「プロモーション・アップセル」を監視するようにこの要素を設定するには、「where `closenessQualifier's repositoryID` is *ID*」を示す必要があります。ここで、ID は「プロモーション・アップセル」の実際の ID です。

Promotion Closeness Qualification

現在プロモーションの適用を受けていないユーザーが、この要素の設定方法に応じて、特定の「プロモーション・アップセル」または任意の「プロモーション・アップセル」を持っているかどうかをシステムが確認します。特定の「プロモーション・アップセル」を監視するようにこの要素を設定するには、「where `closenessQualifier's repositoryID` is *ID*」を示す必要があります。ここで、ID は「プロモーション・アップセル」の実際の ID です。多くの場合、この要素の後に E メールの送信処理または項目へのスロットの追加処理が続きます。

Promotion Offered

シナリオがユーザーにプロモーションを付与するのを、より具体的に言えば、シナリオがプロモーションの付与処理を実行するのをシステムが監視します。この要素内でオプションのパラメータを使用して、監視の対象となるイベントのタイプをさらに細かく定義できます。

プロモーションの付与処理要素の詳細は、この章のシナリオでの Core Commerce 処理要素の使用を参照してください。

技術的な注意事項: この要素は、`ScenarioManager` が `PromotionAction` シナリオ処理によって送信された `PromotionGrantedMessage` を受信するとトリガーされます。

Promotion Revoked

シナリオがユーザーのアクティブなプロモーションを取り消すのを、より具体的に言えば、シナリオがプロモーションの取消し処理を実行するのをシステムが監視します。この要素内でオプションのパラメータを使用して、監視の対象となるイベントのタイプをさらに細かく定義できます。

プロモーションの取消し処理要素の詳細は、この章のシナリオでの Core Commerce 処理要素の使用を参照してください。

技術的な注意事項: この要素は、`ScenarioManager` が `RemovePromotionAction` シナリオ処理によって送信された `PromotionRevokedMessage` を受信するとトリガーされます。

Scenario Added an Item to an Order

シナリオがオーダーに品目を追加するのを、より具体的に言えば、シナリオが品目をオーダーに追加処理を実行するのをシステムが監視します。この要素内でオプションのパラメータを使用して、監視の対象となるオーダーまたは品目のタイプをさらに細かく定義できます。

品目をオーダーに追加処理要素の詳細は、この章のシナリオでの Core Commerce 処理要素の使用を参照してください。

技術的な注意事項: この要素は、`ScenarioManager` が `SendScenarioEvent` プロセッサによって送信された `ScenarioAddedItemToOrder` メッセージを受信するとトリガーされます。このイベントの発生を無効化するには、`PromotionTools.sendEventOnAddItem` プロパティを false に設定します。デフォルト値は true であることに注意してください。

`SendScenarioEvent` プロセッサの詳細は、『ATG Web Commerce Programming Guide』の *sendScenarioEvent パイプライン・チェーン*に関する項を参照してください。

Scheduled Order イベント

システムが、定期オーダーの作成、更新または削除など、あるいは定期オーダーのエラーまたは失敗などの定期オーダー・イベントを監視します。この要素内でオプションのパラメータを使用して、監視の対象となる定期オーダー・イベントのタイプをさらに細かく定義できます。

技術的な注意事項: この要素は、`ScenarioManager` が `ScheduledOrderHandler` フォーム・ハンドラによって送信された `ScheduledOrderMessage` メッセージを受信するとトリガーされます。発生する定期オーダー・イベントのタイプは、メッセージの `action` プロパティの値によって異なります。詳細は、『ATG Web Commerce Programming Guide』の「*購買プロセス・サービスの構成*」の*反復オーダーのスケジューリング*に関する項を参照してください。

Shipping Group Changes

Core Commerce がオーダー内の出荷情報にステータスの変化が起きたことを示すのをシステムが監視します。変化の発生源は顧客サービス・アプリケーションなどの外部システムです。この要素内でオプションのパラメータを使用して、この要素をトリガーするメッセージをさらに細かく定義できます。

技術的な注意事項: この要素は、`ScenarioManager` が `OrderChangeHandler` によって送信された `ShippingGroupModified` メッセージを受信するとトリガーされます。詳細は、『ATG Web Commerce Programming Guide』の*オーダー・フルフィルメント・フレームワークの構成*を参照してください。

Update Inventory

プレ・オーダーされた、バック・オーダーされた、または在庫が切れたカタログ項目の在庫レベルが増加するのをシステムが監視します。この要素内でオプションのパラメータを使用して、監視の対象となるイベントのタイプをさらに細かく定義できます。

技術的な注意事項: この要素は、`ScenarioManager` が `InventoryManager` によって送信された `UpdateInventory` メッセージを受信するとトリガーされます。詳細は、『ATG Web Commerce Programming Guide』の*在庫のフレームワーク*を参照してください。

Uses Promotion

顧客が事前定義された割引を使用して製品を購入するのをシステムが監視します。この要素内でオプションのパラメータを使用して、監視の対象となるプロモーションをさらに細かく定義できます。

例:*Uses promotion named Buy 2 Helmets, Get One Free*

技術的な注意事項: この要素は、`ScenarioManager` が `SendPromotionUsedMessage` プロセッサ（`atg.commerce.order.processor.ProcSendPromotionUsedMessage` クラス）によって送信された `PromotionUsed` メッセージを受信するとトリガーされます。プロセッサは、プロモーションが含まれたオーダーが `processOrder` パイプライン・チェーンを通過したときに、このメッセージを送信します。詳細は、『ATG Web Commerce Programming Guide』の*購買プロセス・サービスの構成*を参照してください。

シナリオでの Core Commerce 条件要素の使用

シナリオ内の条件要素は、イベント要素の後に続き、実質的に"if"文を追加することによってイベント要素をさらに修飾します。条件に対して表示されるオプションは、条件に先行するイベントによって異なります。たとえば、“visits page in folder /shoes”イベント要素の後に、ページを厳密に指定する条件要素の“if page is `/shoes/hikingboots.jsp`.”を配置できます。

この項では、シナリオで使用される Core Commerce に付属の条件要素について説明します。これらの Commerce 条件要素は、Scenarios モジュールに付属している要素に追加されたものです。シナリオ、シナリオで条件要素を使用する方法、非 Commerce 条件要素については、『ATG Web Commerce パーソナライゼーション・ガイド』の*シナリオの作成*を参照してください。

Core Commerce に付属の条件要素では満たされない要件がある場合は、アプリケーション開発者がカスタム条件要素を作成できます。カスタム条件要素は、ATG Control Center に表示され、それを任意の標準の条件要素と同様に使用できます。

開発者向けの注意: カスタム条件要素の作成の詳細は、『ATG Web Commerce Personalization Programming Guide』の*カスタム・イベント、アクションおよび条件のシナリオへの追加*を参照してください。カスタム Core Commerce 要素で使用するために Scenarios モジュールの構文を拡張する必要がある場合は、同じガイドの「*ATG Expression エディタの構成*」の *Commerce に関連した構文の構成*に関する項を参照してください。

次の項以降で、Core Commerce に付属の条件要素について説明します。

Item Where

Core Commerce の品目関連イベントの 1 つをさらに修飾できます。条件は品目ベースのプロモーションの割引ルールと同じ基準を使用します。例:

```
Item where product named Shatterproof helmet
```

詳細は、プロモーションの作成と保守 を参照してください。

Order Where

Core Commerce のオーダー関連イベントの 1 つをさらに修飾できます。条件はオーダー・ベースのプロモーションの割引ルールと同じ基準を使用します。例:

```
Order where order contains at least 1 (product in category named BMXBikes)
```

詳細は、プロモーションの作成と保守 を参照してください。

シナリオでの Core Commerce 処理要素の使用

処理要素はシナリオの「what」の部分を拡張します。イベント要素はサイト訪問者が実行することを定義しますが、処理要素はイベントに反応してシステムが実行することを定義します。たとえば、システムに E メールを送信させたり、ユーザーのプロファイル内の特定の属性を変更させたり、スロット内の特定のコンテンツを表示させたりできます（詳細は、『ATG Web Commerce パーソナライゼーション・ガイド』の*シナリオの作成*を参照してください。）

この項では、シナリオで使用される Core Commerce に付属の処理要素について説明します。これらの Core Commerce 処理要素は、Scenarios モジュールに付属している要素に追加されたものです。シナリオ、シナリオで処理要素を使用する方法、非 Commerce 処理要素については、『ATG Web Commerce パーソナライゼーション・ガイド』の*シナリオの作成*を参照してください。

Core Commerce に付属の処理要素では満たされない要件がある場合は、アプリケーション開発者がカスタム処理要素を作成できます。カスタム処理要素は、ATG Control Center に表示され、それを任意の標準の処理要素と同様に使用できます。

開発者向けの注意: カスタム処理要素の作成の詳細は、『ATG Web Commerce Personalization Programming Guide』の*カスタム・イベント、アクションおよび条件のシナリオへの追加*を参照してください。カスタム Commerce 要素で使用するために Scenarios モジュールの構文を拡張する必要がある場合は、同じガイドの「*ATG Expression エディタの構成*」の *Commerce に関連した構文の構成*に関する項を参照してください。

次に、Commerce に付属の処理要素をアルファベット順に説明します。

Add Item to Order

特定の品目を顧客のショッピング・カートに追加するために使用します。

Fill Related Items to Slot

顧客の現在のショッピング・カート内の製品に関連する製品を特定のスロットに追加するために使用します。使用するスロットとスロットでプロパティ・タイプ（たとえば `relatedProducts`）別に表示する製品を指定します。

ショッピング・カートが空の場合、デフォルトでは、関連する製品は特定のスロットに挿入**されません。**ショッピング・カートが空でも関連する製品をスロットに挿入するには、処理要素の最後に“Add these items if Shopping Cart is empty”を指定します。

この処理要素の詳細は、シナリオを使用した製品のクロスセルとアップセルを参照してください。

Give Promotion

特定のプロモーションをサイト訪問者にとって利用可能にするために使用します。システムはプロモーションを訪問者のプロファイル内の `activePromotions` 属性に追加します。

Revoke Promotion

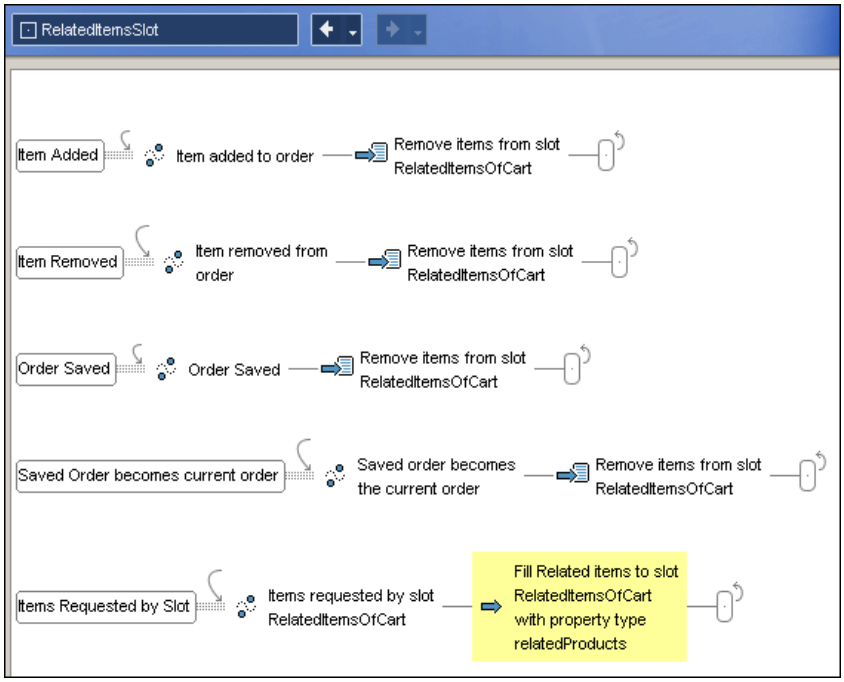
訪問者のプロファイル内の `activePromotions` 属性から特定のプロモーションを削除するために使用します。

シナリオを使用した製品のクロスセルとアップセル

Core Commerce は、顧客の現在のショッピング・カートに入っている製品に関連する製品のクロスセル（またはアップセル）に使用できる `RelatedItemsSlot` という名前の事前定義されたシナリオを備えています。この項では、`RelatedItemsSlot` シナリオが機能する仕組みと、このシナリオをサポートするコンポーネントおよび要素について説明します。

`RelatedItemsSlot` シナリオは、Core Commerce に付属の `RelatedItemsOfCart` という名前の事前定義されたアクティブ・スロットを使用します。デフォルトでは、`RelatedItemsOfCart` スロットはデフォルトの Core Commerce 製品カタログの製品を表示するように構成されています。自社の Commerce アプリケーションで `RelatedItemsSlot` シナリオを使用するとき、ページ開発者は、自社の製品カタログの製品を表示し、顧客のショッピング・カートを表示するアプリケーションの任意のページにその製品を追加するように `RelatedItemsOfCart` スロットを構成します。

`RelatedItemsSlot` シナリオを示す次の図を参照してください（ACC の「シナリオ」→「シナリオ」タスク領域で実際のシナリオにアクセスできることに注意してください）。



RelatedItemsSlot シナリオ

前述の図に見られるように、`RelatedItemsSlot` シナリオでは、顧客が自分の現在のショッピング・カート（オーダー）を変更したことを示す 4 つのイベントのいずれかをシステムが監視します。シナリオの最初の 4 つのセグメントが次のイベントを定義します。

- Item added to order（オーダーに品目が追加されました）
- Item removed from the order（オーダーから品目が削除されました）
- Order saved（オーダーが保存されました）
- Saved order becomes the current order（保存されたオーダーが現在のオーダーになります）

前述のイベントのいずれかが発生すると、システムは、それに反応して、`RelatedItemsOfCart` スロットからすべての品目を削除します。

`RelatedItemsOfCart` スロットのコンテンツを動的に変更するのは、シナリオの最後のセグメントです。このセグメントでは、`RelatedItemsOfCart` スロットが品目を要求するのをシステムが監視します。`RelatedItemsOfCart` はアクティブ・スロットであるため、スロットが空になると必ず品目を要求します。したがって、顧客のショッピング・カートに前述のいずれかの変更が加わると（これらのイベントはいずれもスロットが空になる原因となるため）、このスロットは必ず品目を要求します。

`RelatedItemsOfCart` スロットがコンテンツを要求すると、`Fill Related Items to Slot` 処理（前述の図の黄色でハイライトされた部分）が実行されます。この処理では、システムが顧客の現在のショッピング・カートを調べて、顧客のショッピング・カートに入っている製品に関連する製品を `RelatedItemsOfCart` スロットに挿入します。

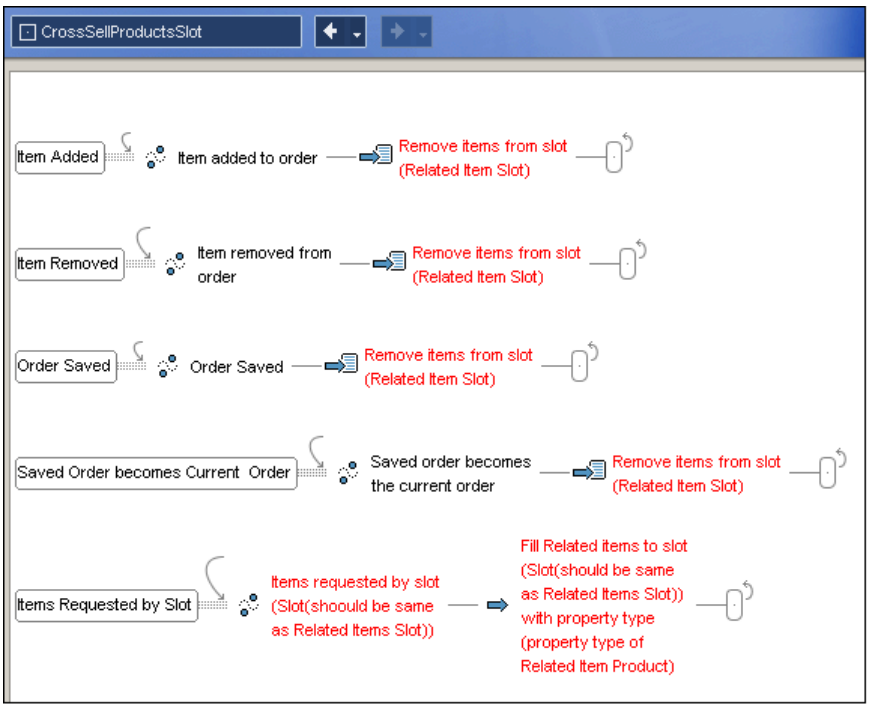
シナリオでは、`Fill Related Items to Slot` 処理は、（関連する）製品をクロスセルするために、デフォルトで `RelatedItemsOfCart` スロットに「プロパティ・タイプ `relatedProducts`」を挿入するように構成されています。ただし、このプロパティ・タイプ値は、製品をアップセルするためにアプリケーション開発者が作成したカスタム・プロパティなど、製品が格納される任意のプロパティに変更できます。

ページ開発者向けの注意: Motorprise リファレンス・アプリケーションでは、`RelatedItemsSlot` シナリオと `RelatedItemsOfCart` スロットを利用して、自社のショッピング・カート・ページで製品をクロスセルします。この実装の詳細は、『ATG Web Commerce ビジネス・コマース・リファレンス・アプリケーション・ガイド』の「*マーチャンダイジング*」の *Commerce シナリオ*に関する項を参照してください。

クロスセルとアップセルを目的とする追加のシナリオの作成

Core Commerce は、アプリケーションのショッピング・カート・ページでの製品のクロスセルおよびアップセルを目的とする追加のシナリオをすばやく作成することを手助けする `CrossSellProductsSlot` という名前のシナリオ・テンプレートを備えています。

`CrossSellProductsSlot` シナリオ・テンプレートを示す次の図を参照してください（ACC の「シナリオ」→「シナリオ」タスク領域で実際のテンプレートにアクセスできることに注意してください）。



CrossSellProductsSlot シナリオ・テンプレート

図を見ればわかるように、`CrossSellProductsSlot` テンプレートは `RelatedItemsSlot` シナリオによく似ています。各セグメントの`(Related Item Slot)`プレースホルダを適切なスロット（`RelatedItemsOfCart` またはアプリケーション開発者が作成したカスタム・スロット）に置き換えて、さらに`(property type of Related Item Product)`プレースホルダを目的のプロパティ値、たとえば、クロスセル対象製品をスロットに挿入する `relatedProducts` プロパティやアップセル対象製品をスロットに挿入する、アプリケーション開発者の作成したカスタム・プロパティなどに置き換えます。

シナリオ・テンプレートの作業に関する一般情報については、『ATG Web Commerce パーソナライゼーション・ガイド』の *シナリオの作成*を参照してください。

8　放棄されたオーダーの管理

放棄されたオーダーまたはショッピング・カートとは、顧客が作成し、品目を追加したものの、精算しないオーダーです。顧客が精算をすることなく Web サイトを離れることで、不完全なオーダーは「放棄」されます。

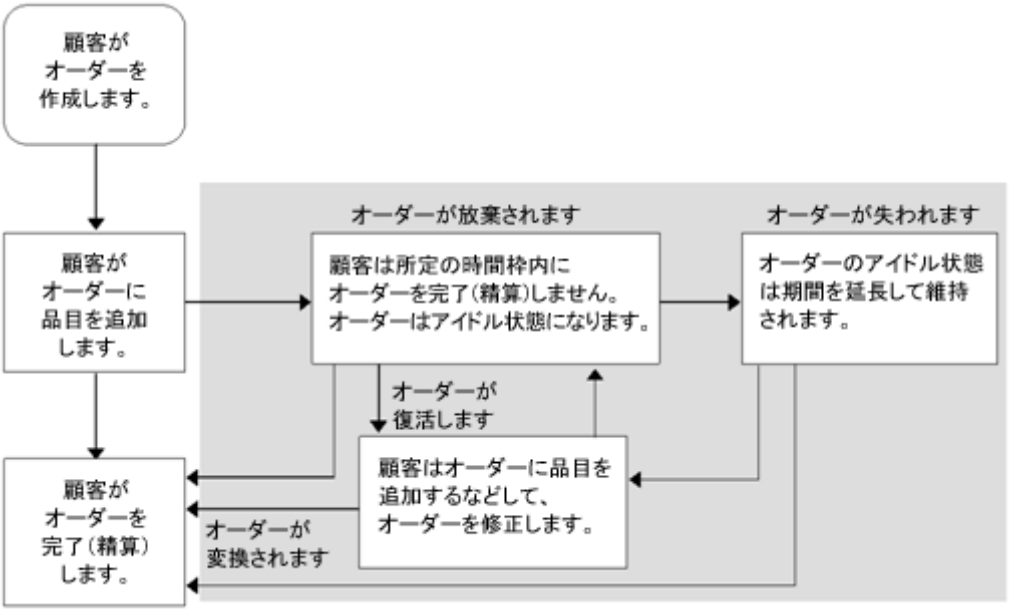
Oracle Commerce Core Commerce に付属の Abandoned Order Services モジュールには、放棄されたオーダーおよびそれに関連するアクティビティを検出し、それらに反応し、それらをレポートするために使用できるサービスやツールのコレクションが含まれています。したがって、このモジュールを利用すれば、顧客がどのようなオーダーを放棄しているのか、どのようなキャンペーンがオーダーを復活させ、完了させるようユーザーに促すのに効果的なのかをよりよく理解できます。その結果、オーダーの変換とオーダーから得られる収益が増大します。

この章では、オーダー放棄アクティビティに反応するキャンペーンの作成を担当する小売業者とビジネス・ユーザーを読者として想定しています。この章には、次の項があります。

重要: モジュールの構成など、開発者によって一般的に実行される関連タスクの詳細は、『ATG Web Commerce Programming Guide』の*放棄されたオーダー・サービスの使用*を参照してください。

オーダー放棄の理解

顧客に作成されたオーダーがたどる様々な道を示す次の処理フロー図を見てください。



この章の冒頭で説明したように、Abandoned Order Services モジュールには、放棄されたオーダーおよびそれに関連するアクティビティ、つまり上の図のグレーの部分に含まれているアクティビティを検出し、それに反応し、それをレポートするのに使用できるサービスやツールのコレクションが含まれています。図が示しているように、この領域に含まれるオーダーのタイプにはいくつかあります。

オーダー・タイプ	説明
放棄されたオーダー	放棄されたオーダーとは、顧客によって精算されておらず、一定の期間にわたってアイドル状態にある不完全なオーダーです。 開発者は、どのようなオーダーを放棄されたと見なすかを定義するためにビジネス・ユーザーが必要とする任意の基準をモジュールが使用するように構成できます。標準で次の基準を使用できます。 -- アイドル状態に置かれている日数 -- 最低金額（オプション） たとえば、10 日間アイドル状態にある不完全なオーダーを検出し、それを放棄されたオーダーと認定するようにシステムを構成できます。あるいは、放棄されたと認定するにはオーダーの価格が最低 25 ドルでなければならないと指定することで、基準をさらに絞り込むことができます。 デフォルト・システムは 1 つのタイプのオーダーおよび前述の基準しかサポートしていませんが、複数のタイプの放棄されたオーダーおよび追加の基準をサポートするようにシステムを構成できることに注意してください。たとえば、高価格の不完全オーダーと低価格の不完全オーダーという 2 つのタイプの放棄されたオーダーをシステムに認定させ、区別させたいことがあります。その場合は、個々のタイプに合わせて調整されたキャンペーン（シナリオ、E メールなど）を作成できます。 放棄されたオーダーに関する要件を特定したら、開発者と相談して、放棄されたオーダーが定期的に検索され、認定されるように、開発者にシステムを構成させる必要があります。
復活したオーダー	復活したオーダーとは、以前放棄されたが、その後、品目の追加や品目数量の変更など、何らかの方法で顧客によって変更されたオーダーです。 参考のために、ユーザーが以前放棄したオーダーを変更するのを監視し、そのオーダーを復活したと認定する、標準で付属しているシナリオを示します。この章のオーダー放棄アクティビティへの反応を参照してください。
変換されたオーダー	変換されたオーダーとは、以前放棄されたが、その後顧客によって精算されたオーダーです。 参考のために、ユーザーが以前放棄したオーダーを精算するのを監視し、そのオーダーを変換されたと認定する、標準で付属しているシナリオを示します。この章のオーダー放棄アクティビティへの反応を参照してください。

オーダー・タイプ	説明
失われた オーダー	失われたオーダーとは、放棄されてから長期間が経過しているため、オーダーの復活が現実的と見なされなくなったオーダーです。デフォルト・システムは、放棄されたオーダーと同じ基準を失われたオーダーに対しても使用します。 -- アイドル状態に置かれている日数 -- 最低金額（オプション） たとえば、25 日間アイドル状態にある不完全なオーダーを検出し、それらを失われたオーダーとして認定するようにシステムを構成できます。 放棄されたオーダーと同様に、前述の基準は標準でサポートされています。ただし、開発者は追加の基準および複数のタイプの失われたオーダーをサポートするようにシステムを構成できます。 失われたオーダーに関する要件を特定したら、開発者と相談して、失われたオーダーが定期的に検索され、認定されるように、開発者にシステムを構成させる必要があります。

最後に、図に描かれたオーダー放棄アクティビティに関する処理フローが常に直線的であるわけではないことに注意してください。たとえば、オーダーはいったん放棄された後、復活し、再び放棄されることがあります。これを覚えておくことは、自分の放棄されたオーダーを復活させ、完了させるようユーザーを誘うキャンペーンを作成するときに特に有用です。この問題については、次のオーダー放棄アクティビティへの反応を参照してください。

オーダー放棄アクティビティへの反応

プロモーションが組み込まれたシナリオとテンプレート化された E メールは、自分の放棄されたオーダーを復活させ、変換するよう顧客を促すための重要なツールです。したがって、この項では、この目的で行われるシナリオの作成とテストに関する重要な情報を説明します。

放棄アクティビティに反応するシナリオの作成
放棄されているオーダーおよび以前に放棄されたオーダーを更新して、ユーザーのアクティビティを反映させる標準のシナリオについて説明します。自分の放棄されたオーダーを復活させ、変換するようユーザーを誘うことを目的とするシナリオの例も提示します。

放棄アクティビティに反応するシナリオのテスト
Commerce Administration ユーザー・インタフェースを使用して放棄関連シナリオをテストする方法を説明します。

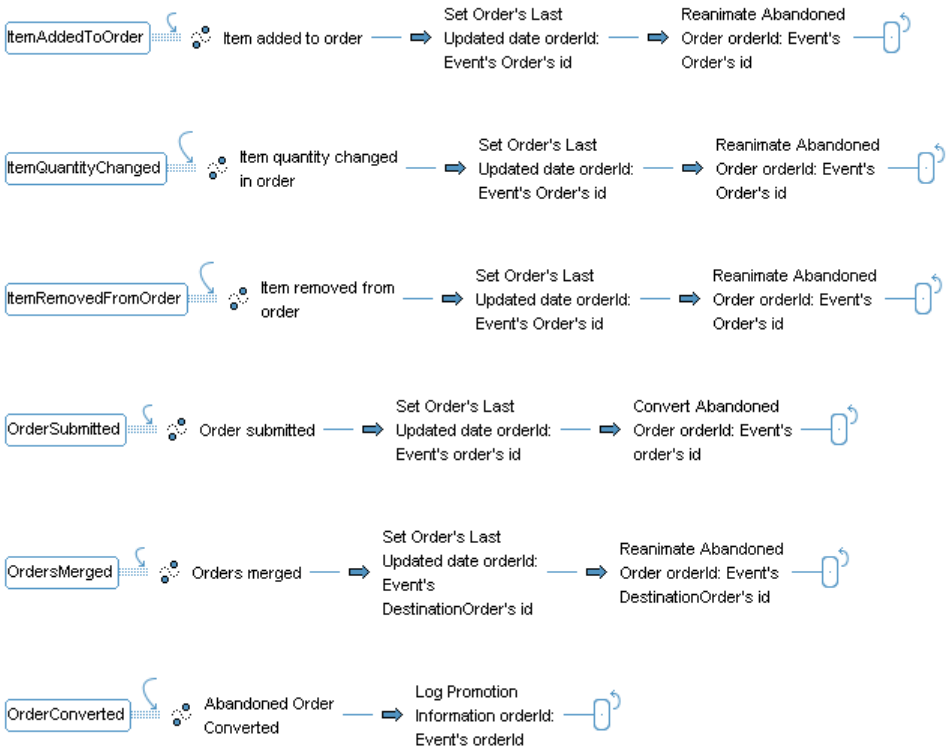
シナリオのイベント要素
放棄関連のシナリオで使用できるイベント要素について説明します。

シナリオの処理要素
放棄関連シナリオで使用できる処理要素について説明します。

放棄アクティビティに反応するシナリオの作成

Web サイトで Abandoned Order Services モジュールを実行している場合、開発者は、すでに、オーダーを定期的に検索し、放棄されたオーダーまたは失われたオーダーとして認定するようにモジュールを構成しています。

小売業者やビジネス・ユーザーのタスクは、オーダー放棄アクティビティに反応するシナリオを作成することです。参考のために、放棄されているオーダーまたは以前に放棄されたオーダーに対するユーザーのアクティビティを監視するシナリオを提示します。ATG Control Center の「シナリオ」→「シナリオ」タスク領域で Abandoned Orders という名前のシナリオを調べることができます。このシナリオは「Abandoned Orders」フォルダにあります。シナリオは次のようなものです。



Abandoned Orders シナリオ

このシナリオでは、顧客が現在のオーダーに変更を加えたことを示すイベントをシステムが監視します。発生する可能性のあるイベントには次のものがあります。

- Item added to order（オーダーに品目が追加されました）
- Item quantity changed（変更された品目数量）
- Item removed from the order（オーダーから品目が削除されました）
- Order submitted（オーダーが発行されました）
- Orders merged（オーダーがマージされました）

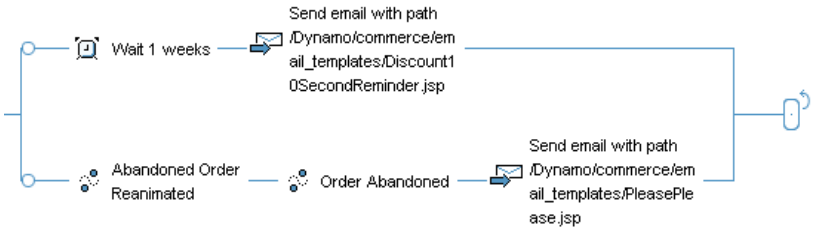
前述のいずれかのイベントが発生すると、システムはオーダーを更新して、最後にオーダーが更新された日時を反映させます。次に、状況に応じて、そのオーダーを、復活したオーダーまたは変換されたオーダー

と認定します。オーダーが変換された場合、システムはレポートを目的としてオーダーのプロモーション関連情報も記録します。

前述のように、Abandoned Orders シナリオは参考のために提示したものです。したがって、小売業者やビジネス・ユーザーに残されたタスクは、放棄されたオーダーおよび失われたオーダーを監視して、それらのオーダーを復活させ、変換するようユーザーを促すシナリオを設計し、作成することです。次に、*仮想のシナリオを例として提示します。*

このシナリオの最初の部分で、システムはオーダーが放棄されたと認定されるのを監視します。このイベントが発生すると、システムは、オーダーが確定すれば 10%の割引を提供するプロモーションを顧客に付与し、`Discount10Reminder.jsp` という E メールを介してそのプロモーションを顧客に通知します。

シナリオの残りの部分は次のようになります。



このシナリオの残りの部分では、2 つの分岐のうち 1 つしか成功しません。

- 放棄されたオーダーが 1 週間アイドル状態に置かれると、顧客にプロモーションを思い出させるための 2 番目の E メール・メッセージ `Discount10SecondReminder.jsp` が送信されます。
- 顧客がオーダーを復活させ、再び放棄すると、別の E メール・メッセージ `PleasePlease.jsp` が送信されます。

1 つの分岐しか成功しないようにシナリオが構成されているため、1 つの放棄されたオーダーにつき 2 つを上回るメッセージは顧客に送信されません（このシナリオは仮想のシナリオであり、インストールには含まれていないことを思い出してください）。

最後に、シナリオで利用できる 2 つの放棄関連プロパティをユーザーのプロファイルが持っていることに注意してください。

- 現在ユーザーに関連付けられている放棄されたオーダーの数が格納される `abandonedOrderCount`
- 現在ユーザーに関連付けられている `abandonedOrder` 項目のリストが格納される `abandonedOrders`。このタイプの項目には放棄されたオーダーの ID が格納されます。

これらのプロパティを利用して、より機能が豊富な放棄関連シナリオを作成できます。たとえば、ユーザーがログインするのを監視し、ユーザーの `abandonmentCount` プロファイル・プロパティが 1 以上の場合に、ユーザーに“Free Shipping on Orders Purchased Today（今日購入すれば送料無料）”プロモーションを付与するシナリオを作成できます。

放棄アクティビティに反応するシナリオのテスト

放棄アクティビティに反応するシナリオを作成したときは、Commerce Administration UI の *Abandoned Order Messages* ページを介してシナリオをテストできます。このインタフェースを利用すれば、1 つ以上のオーダーに特定の放棄状態を手動で割り当てて、システムに適切なシナリオ・イベント（Order Abandoned、Abandoned Order Reanimated など）を生成させることができます。このようにすれば、シナリオの様々なセグメントをオーダーに強制的に通過させて、シナリオをテストすることができます。

1 つ以上のオーダーで放棄シナリオをテストするには、次の手順を実行します。

1. テスト・データとして使用する 1 つ以上のオーダーを作成します。
2. ブラウザでアプリケーション・サーバーに該当するリンクを指定することによって、Dynamo Server Admin にアクセスします。たとえば、JBoss のユーザーは、デフォルトで次の URL を使用します。

   ```
   http://hostname:8080/dyn/admin
   ```

   注意: アプリケーションのアセンブル中、この UI にアクセスするには、Dynamo Server Admin モジュールを指定する必要があります。『ATG Web Commerce Platform Programming Guide』を参照してください。
3. ユーザー名とパスワードを使用して UI にログインします。デフォルトのユーザー名/パスワードは `admin/admin` です。
4. メイン *Dynamo Administration* ページの Commerce Administration リンクをクリックします。
5. *Dynamo Commerce Administration* ページの Abandoned Order Administration リンクをクリックします。

   次の図に示す *Abandoned Order Messages* ページが表示されます。

Commerce Administration UI の Abandoned Order Messages ページ

6. 適切な情報を入力します。
   - オーダーID フィールドに、放棄状態を変更するオーダーの ID を入力します。
   - 「放棄状態」ドロップダウン・リストで放棄状態を選択します。
   - イベントの日付フィールドに日付を入力します。一般的に、ここで使用する日付は現在の日付です。ただし、シナリオが時間要素（たとえば、“wait 2 weeks”）を利用

する場合は、シナリオをその要素より先に進めるために未来の日付を入力できます。

- **Notify** ボタンをクリックします。

シナリオのイベント要素

放棄されたオーダー・アクティビティを監視する必要のあるシナリオで、次のイベント要素を使用できます。

Order Abandoned（オーダーは放棄されました）

Abandoned Order Converted（放棄されたオーダーは変換されました）

Abandoned Order Lost（放棄されたオーダーは失われました）

Abandoned Order Reanimated（放棄されたオーダーは復活しました）

Order Abandoned（オーダーは放棄されました）

オーダーが放棄されたと認定されるのをシステムが監視します。

この要素内でオプションのパラメータを使用して、監視の対象となるオーダーのタイプをさらに細かく定義できますが、一般的にオプションのパラメータは必要とされません。

Abandoned Order Reanimated（放棄されたオーダーは復活しました）

オーダーが復活したと認定されるのをシステムが監視します。

この要素内でオプションのパラメータを使用して、監視の対象となるオーダーのタイプをさらに細かく定義できますが、一般的にオプションのパラメータは必要とされません。

Abandoned Order Converted（放棄されたオーダーは変換されました）

オーダーが変換されたと認定されるのをシステムが監視します。

この要素内でオプションのパラメータを使用して、監視の対象となるオーダーのタイプをさらに細かく定義できますが、一般的にオプションのパラメータは必要とされません。

Abandoned Order Lost（放棄されたオーダーは失われました）

オーダーが失われたと認定されるのをシステムが監視します。

この要素内でオプションのパラメータを使用して、監視の対象となるオーダーのタイプをさらに細かく定義できますが、一般的にオプションのパラメータは必要とされません。

シナリオの処理要素

放棄されたオーダー・アクティビティに反応する必要のあるシナリオで、次の処理要素を使用できます。

Set Order's Last Updated Date

Reanimate Abandoned Order（放棄されたオーダーの復活）

Convert Abandoned Order（放棄されたオーダーの変換）

Log Promotion Information（プロモーション情報のログ記録）

Set Order's Last Updated Date

オーダーを更新して、所有者によってオーダーが最後に変更された日時を反映させます。

この処理要素を使用して、オーダーを更新し、ユーザー・アクティビティが発生したことを反映させます。ユーザー・アクティビティには、品目の追加、品目の削除、品目数量の変更、オーダーのマージおよびオーダーの精算が含まれます。

例:

```
Set Order's Last Updated date orderId:Event's Order's id
Set Order's Last Updated date orderId:Event's DestinationOrder's id
```

2 番目の例では、2 つのオーダーがマージされた結果として処理が呼び出されます。その場合、処理はマージ元オーダーではなく、マージ先オーダーの日付、時刻を更新する必要があります。

Reanimate Abandoned Order（放棄されたオーダーの復活）

放棄されたオーダーを、復活したオーダーとして認定します。

放棄されたオーダーが変更され、したがって、復活させる必要があるときに、この処理要素を使用します。変更には、品目の追加、品目の削除、品目数量の変更、オーダーのマージおよびオーダーの精算が含まれます。

例:

```
Reanimate Abandoned Order orderId:Event's Order's id
Reanimate Abandoned Order orderId:Event's DestinationOrder's id
```

2 番目の例では、2 つのオーダーがマージされた結果として処理が呼び出されます。その場合、処理はマージ元オーダーではなく、マージ先オーダーを復活させる必要があります。

Convert Abandoned Order（放棄されたオーダーの変換）

以前放棄されたオーダーを変換されたと認定します。

以前放棄されたオーダーが所有者によって精算され、したがって、変換されたと認定する必要があるときに、この処理要素を使用します。

例:

```
Convert Abandoned Order orderId:Event's Order's id
```

Log Promotion Information（プロモーション情報のログ記録）

変換されたオーダーに適用されたプロモーションの数と合計値を計算し、記録します。

以前放棄されたオーダーが変換され、そのオーダーのプロモーション関連情報をログに記録するときに、この処理要素を使用します。

例:

```
Log Promotion Information orderId:Event's Order's id
```

9　カタログのナビゲーションと検索

コマース・サイトは、顧客がサイト内をナビゲートし、購入したい製品を見つけるためのメカニズムを提供する必要があります。Oracle Commerce Core Commerce には、サイトにナビゲーション・メカニズムと検索メカニズムを実装するために使用できるサービスが含まれています。

この章は次の項から構成されています。

この章では、`atg.commerce.catalog.SearchFormHandler` に基づいている Core Commerce が本来備えている検索機能に固有のコンポーネントのみを説明します。検索で使用されるコンポーネントに関する一般情報については、『ATG Web Commerce Page Developer's Guide』の*検索フォームの使用*を参照してください。

parentCategory プロパティの使用

カテゴリまたは製品は複数のカテゴリの子になる可能性があります。複数の親カテゴリを指定するとカタログはより柔軟性が高くなりますが、ナビゲーションが複雑になる可能性があります。顧客がカタログ階層間の移動によってではなく、検索機能を使用してカテゴリや製品にアクセスする場合は、特にその傾向が強くなります。顧客が上の階層に移動する場合、ストア側は移動先の親カテゴリを決定する必要があります。その目的でデフォルトの親カテゴリを指定するために、カテゴリ項目および製品項目の `parentCategory` プロパティを使用できます。製品は、製品が表示されるカタログごとに複数の `parentCategories` を持つことができます。

たとえば、カタログ階層の 1 つ上のレベルまで顧客を誘導するリンクが各ページにあるとします。複数の親カテゴリを持つ製品を顧客が表示している場合、サイトは顧客がどの親カテゴリから製品にアクセスしたかを追跡して、その親カテゴリを指すようにリンクを設定できます。しかし、顧客がカタログ階層内をナビゲートしたのではなく、検索によって製品を見つけた場合は、`parentCategory` プロパティによって指定されたカテゴリを指すようにリンクを設定できます。

カタログで `parentCategory` プロパティに異なる名前を使用している場合は、`/atg/commerce/catalog/CatalogTools.properties` ファイル内の `parentCategoryPropertyName` プロパティをプロパティの実際の名前に設定します。たとえば、次のようになります。

```
parentCategoryPropertyName=higherCategory
```

カタログ、カテゴリおよび製品のプロパティおよび製品カタログを拡張する方法の詳細は、『ATG Web Commerce Programming Guide』を参照してください。

カタログ項目の表示

この項で説明するコンポーネントを利用すれば、製品カタログ内の項目を検索し、その項目をユーザーに表示できます。

カタログでの項目の検索

`atg.commerce.catalog.custom.CatalogItemLookupDroplet` クラス（`atg.repository.servlet.ItemLookupDroplet` のサブクラス）を使用して、カタログ関連のリポジトリ内の項目を検索し、表示します。このサーブレット Bean は、リポジトリ、リポジトリ ID、サイト ID、サイト・スコープ、カタログおよび項目タイプを指定する入力パラメータを受け取り、指定された項目をページ上にレンダリングします。

入力パラメータを介してリポジトリと項目タイプを指定するかわりに、サーブレット Bean のプロパティ・ファイルを介してリポジトリと項目タイプを設定できます。Core Commerce には、製品カタログをデフォルトのリポジトリとして使用し、特定の項目タイプを検索するように構成されている複数のルックアップ・コンポーネントが含まれています。次のコンポーネントは、すべて `CatalogItemLookupDroplet` に基づいています。

- `/atg/commerce/catalog/CategoryLookup`
- `/atg/commerce/catalog/ProductLookup`
- `/atg/commerce/catalog/SKULookup`
- `/atg/commerce/promotion/PromotionLookup`
- `/atg/commerce/promotion/PromotionalContentLookup`

サイトに複数のカタログが含まれている場合、またはカタログにない項目タイプをカタログが使用している場合は、追加の `CatalogItemLookupDroplet` コンポーネントを作成します。`CatalogItemLookupDroplet` サーブレット Bean の詳細は、付録 A: Core Commerce サーブレット Bean の CatalogItemLookupDroplet を参照してください。

次の追加のカタログ関連ルックアップ・コンポーネントは、カタログのフィルタ機能を持っていない `atg.repository.servlet.ItemLookupDroplet` クラスに直接基づいています。

- `/atg/commerce/catalog/MediaLookup`
- `/atg/commerce/catalog/custom/CatalogLookup`

`ItemLookupDroplet` については、『ATG Web Commerce Page Developer's Guide』を参照してください。このドロップレットは、ギフト・リスト機能によっても使用されます。『ATG Web Commerce Programming Guide』を参照してください。

次の例は、`ProductLookup` コンポーネントを使用して現在の製品を表示する JSP の部分を示しています。製品のリポジトリ ID は、このページにリンクしているページから（`itemId` パラメータを介して）このページに渡されます。

```
<dsp:droplet name="/atg/commerce/catalog/ProductLookup">
<dsp:param param="itemId" name="id"/>

<dsp:oparam name="output">
<p><b><dsp:valueof param="element.displayName"/></b>
<p><dsp:getvalueof id="img13" param="element.largeImage.url"
        idtype="java.lang.String">
<dsp:img src="<%=img13%>"/>
```

```
</dsp:getvalueof>
<dsp:valueof param="element.longDescription"/>
</dsp:oparam>
</dsp:droplet>
```

ForEachItemInCatalog サーブレット Bean

このサーブレット Bean は、1 つの点を除いて、よく使用される ForEach Bean と同じです。違いは、このサーブレット Bean がユーザーの現在のカタログ内に存在する項目に対してのみ繰り返し処理を行うことです（ForEach サーブレット Bean については、『ATG Web Commerce Page Developer's Guide』を参照してください）。このサーブレット Bean には、追加のオプション・パラメータ profile が含まれています。ユーザーのプロファイルには、そのユーザーが表示できるカタログに関する情報が格納されます。プロファイルを指定しない場合は、そのかわりに、現在のセッション限定プロファイルが使用されます。

relatedProducts、relatedCategories、replacementSkus など、現在のカタログに必ずしも存在しない可能性がある項目のリストに対して、このサーブレット Bean を使用します。

項目が表示されたときのメッセージの送信

顧客がカタログ内の項目を表示したときに、atg.userprofiling.ViewItemEventSender サーブレット Bean を使用して JMS メッセージを送信できます。ViewItemEventSender は atg.userprofiling.dms.ViewItemMessage クラスの JMS オブジェクト・メッセージを送信します。ViewItemMessage オブジェクトは、現在表示されているリポジトリ項目および項目の場所を識別します。これらのメッセージを処理のトリガーに使用できます。たとえば、表示された製品に関する情報を顧客のプロファイルに格納するようにメッセージ・リスナーを構成できます。

ViewItemEventSender は、表示される項目を指定する eventobject と呼ばれる 1 つの入力パラメータを取ります。出力パラメータまたはオープン・パラメータはありません。

Core Commerce には、/atg/commerce/catalog 内の 2 つの ViewItemEventSender コンポーネントである ProductBrowsed と CategoryBrowsed が含まれています。どちらのコンポーネントを使用するかによって、渡される eventobject は、製品またカテゴリのどちらかのリポジトリ項目になります。

次の例は、製品が表示されたときに ProductBrowsed コンポーネントを使用してメッセージを送信する JSP の部分を示しています。製品のリポジトリ ID は、このページにリンクしているページから（itemId パラメータを介して）このページに渡されます。

```
<dsp:droplet name="/atg/commerce/catalog/ProductLookup">
<dsp:param param="itemId" name="id"/>

<dsp:oparam name="output">
  <dsp:droplet name="/atg/commerce/catalog/ProductBrowsed">
    <dsp:param param="element" name="eventobject"/>
  </dsp:droplet>
</dsp:oparam>
</dsp:droplet>
```

カタログのナビゲーション

カタログが階層構造を持っている場合は、顧客がカタログ階層間を移動することによって製品までナビゲートできるようにサイトを設定できます。カタログ階層の構造は、個々のカテゴリの子カテゴリと子製品によって決まります。たとえば、食料品店サイトでは、ユーザーがまずフルーツ・カテゴリを選択し、次にかんきつ類カテゴリ(フルーツの子カテゴリ)を、最後にオレンジ製品(かんきつ類カテゴリの子製品)を選択することで、オレンジと呼ばれる製品にたどりつきます。

ルート・カテゴリの表示

一般的に、カタログ・ホーム・ページにはルート・カテゴリのリストが表示されます。他のカテゴリと異なり、他のカテゴリの childCategories プロパティを介してルート・カテゴリを見つけることはできません。ルート・カテゴリは、ユーザーのカタログの allRootCategories プロパティに表示されるカテゴリです。

次の例では、ForEach サーブレット Bean を allRootCategories プロパティと組み合せて使用して、ルート・カテゴリを見つけています。

```
<HTML> <HEAD>
<TITLE>Home Page</TITLE>
</HEAD>

<BODY BGCOLOR="#FFFFFF">
<H1>Home Page</H1>
 <dsp:droplet name="/atg/dynamo/droplet/ForEach">
   <dsp:param bean="/atg/userprofiling/Profile.catalog.allRootCategories"
        name="array"/>
  <dsp:oparam name="output">
    <tr>
      <td>
      <dsp:getvalueof id="a26" param="element.template.url"
           idtype="java.lang.String">
<dsp:a href="<%=a26%>">
</dsp:getvalueof>
      <dsp:param param="element.repositoryId" name="id"/>
      <dsp:param value="pop" name="navAction"/>
      <dsp:param param="element" name="Item"/>
      <dsp:valueof param="element.displayName"/></dsp:a></td>
    </tr>
  </dsp:oparam>

  <dsp:oparam name="empty">
    <p>No root categories found.
   </dsp:oparam>
 </dsp:droplet>

</BODY> </HTML>
```

子カテゴリと子製品の表示

カタログ・ページでサーブレット Bean を使用して、1 つのカテゴリのすべての子カテゴリおよび子製品のリストを表示できます。たとえば、食料品店サイトのフルーツ・ページには、多数のフルーツ製品（リンゴ、洋梨）と子カテゴリ（かんきつ類）が表示されます。ユーザーがこのページで製品の名前をクリックすると、サイトはその製品を表示します。ユーザーがカテゴリの名前をクリックすると、サイトはそのカテゴリの子カテゴリと子製品のリストを表示します。

次の例は、現在のカテゴリの個々の子カテゴリと子製品の displayName プロパティをレンダリングする JSP の部分を示しています。これらの値は、それぞれ対応する項目へのリンクとしてレンダリングされます。現在のカテゴリのリポジトリ ID は、このページにリンクしているページから（itemId パラメータを介して）このページに渡されます。

```
<dsp:droplet name="/atg/commerce/catalog/CategoryLookup">
<dsp:param param="itemId" name="id"/>

<dsp:oparam name="output">
     <dsp:droplet name="/atg/dynamo/droplet/ForEach">
     <dsp:param param="element.childCategories" name="array"/>
     <dsp:oparam name="outputStart">
     <p><b>Child Categories:</b>
     <ul>
     </dsp:oparam>
     <dsp:oparam name="output">
     <li><dsp:getvalueof id="a24" param="element.template.url"
              idtype="java.lang.String">
<dsp:a href="<%=a24%>">
     <dsp:valueof param="element.displayName"/>
     <dsp:param param="element.repositoryId" name="itemId"/>
     </dsp:a></dsp:getvalueof></li>
     </dsp:oparam>
     <dsp:oparam name="outputEnd">
     </ul>
     </dsp:oparam>
   </dsp:droplet>

   <dsp:droplet name="/atg/commerce/catalog/ForEachItemInCatalog">
     <dsp:param param="element.childProducts" name="array"/>
     <dsp:oparam name="outputStart">
     <p><b>Child Products:</b>
     <ul>
     </dsp:oparam>
     <dsp:oparam name="output">
     <li><dsp:getvalueof id="a61" param="element.template.url"
              idtype="java.lang.String">
<dsp:a href="<%=a61%>">
     <dsp:valueof param="element.displayName"/>
     <dsp:param param="element.repositoryId" name="itemId"/>
     </dsp:a></dsp:getvalueof></li>
     </dsp:oparam>
```

```
      <dsp:oparam name="outputEnd">
      </ul>
      </dsp:oparam>
    </dsp:droplet>
</dsp:oparam>
</dsp:droplet>
```

履歴のナビゲーション

カタログは通常、階層状に構成されていますが、その階層構造は硬直的ではありません。カテゴリまたは製品は、複数のカテゴリの子である可能性があるため、1 つのカタログ項目に至る複数のパスが存在する可能性があります。さらに、ページは、ユーザーが階層のある部分から別の部分へジャンプできるようにする関連製品へのリンクや、検索機能など、他のナビゲーション補助機能を持っていることがあります。

Core Commerce は、顧客がサイト内を移動するときにたどるパスの追跡に使用できるコンポーネントを備えています。履歴ナビゲーションまたは「ブレッドクラム」と呼ばれるこの追跡方法を利用すれば、顧客が以前訪問した項目へ簡単に戻れるように、顧客が訪問した項目のリストを作成し、それらの項目へのリンクを作成および表示できます。

atg.commerce.catalog.CatalogNavHistory クラスは、カタログ内で顧客がたどったパスを追跡します。CatalogNavHistory は、リポジトリ・コンテンツを表示する任意のページのセットに関するナビゲーション履歴の追跡に使用できる atg.repository.servlet.NavHistory のサブクラスです。Core Commerce の /atg/commerce/catalog/CatalogNavHistory には、セッション限定の CatalogNavHistory コンポーネントが含まれています。

CatalogNavHistory は、顧客が表示した実際のリポジトリ項目から構成される顧客が訪問した場所のスタックを保持します。このコンポーネントは、これらの場所（項目）を navHistory プロパティに格納します。

CatalogNavHistory は、すべての可能な処理にわたって顧客のパスを追跡します。

- 顧客がナビゲーション・リンクをクリックすることによって階層を下へナビゲートすると、項目がスタックにプッシュされます。
- 顧客がナビゲーション・リンクまたは「戻る」ボタンをクリックすることによって階層を上にナビゲートすると、適切な項目がスタックから削除され、新しい項目がスタックにプッシュされます。
- 顧客が現在の場所と関係のないサイトの領域へジャンプすると、グローバル・ナビゲーションまたは他のリンクのどちらを経由したかに関係なく、パスはその領域へジャンプします。階層構造のように、新しいページへのデフォルト・ナビゲーション・パスが存在すれば、CatalogNavHistory はそのデフォルト・パスを使用します。

navHistory 配列内で項目を追加および削除するには、atg.repository.servlet.NavHistoryCollector クラスの CatalogNavHistoryCollector サーブレット Bean を使用します。このサーブレット Bean は次の入力パラメータを取ります。

- item - 現在表示されているリポジトリ項目。
- navAction - スタックに対して実行される操作。オプションはプッシュ、ポップおよびジャンプです。空白の navAction はプッシュとして扱われます。
- navCount - 「戻る」ボタンの使用の検出に使用されます。ブレッドクラムを使用するページに至るすべてのリンクとともに navCount パラメータを渡す必要があります。ターゲット・ページが「戻る」ボタンの使用を検出できるように、このパラメータをリンクに埋め込む必要があります。たとえば、次のようになります。

```
<dsp:getvalueof id="templateUrl" idtype="String"
     param="element.template.url">
  <dsp:a page="<%=templateUrl%>">
    <dsp:param name="id" param="element.repositoryId"/>
    <dsp:param name="navAction" value="push"/>
    <dsp:param name="navCount"
         bean="/atg/commerce/catalog/CatalogNavHistory.navCount"/>
    <dsp:valueof param="element.displayName"/>
  </dsp:a>
</dsp:getvalueof>
```

navCount パラメータを使用して、「戻る」ボタンの使用が原因で発生する navHistory 内のエラーを防止できます。navCount パラメータを使用すれば「戻る」ボタンが使用されたことを検出できるため、CatalogNavHistoryCollector はスタックを適切にリセットできます。

たとえば、カテゴリへのリンクを表示するページを開いた場合、個々のカテゴリ・リンクには navCount=1 に設定された navCount パラメータが埋め込まれている必要があります。訪問者がカタログをしばらく閲覧した後、「戻る」ボタンを使用して元のページへ戻ると、ブラウザによってキャッシュされたページが訪問者に渡されます。Core Commerce はそのページを再びレンダリングしません。そのページのカテゴリ・リンクは、まだ navCount=1 を持っています。それらのリンクのいずれかをクリックすると、ターゲット・ページ内の CatalogNavHistoryCollector は、リンクとともに渡された、失効している navCount ページ・パラメータを/atg/commerce/catalog/CatalogNavHistory.navCount の currentvalue と比較します。リンクから渡されたパラメータは失効しているため、これらの値は一致しません。したがって、CatalogNavHistoryCollector は「戻る」ボタンが使用されたことを知り、navHistory を再作成します。

顧客の履歴の収集

navAction プッシュ操作を使用してユーザーのナビゲーション履歴を収集します。次の例の最初の部分では、顧客が CategoryPage.jsp へのリンクをクリックすると、JSP コードが navHistory スタックに項目を追加するため、顧客のたどったパスを追跡できます。

```
<dsp:droplet name="/atg/dynamo/droplet/ForEach">
   <dsp:param bean="/atg/userprofiling/Profile.catalog.allRootCategories"
        name="array"/>
  <dsp:oparam name="output">
  <p>
  <!
    Here we set up the navCount and navAction parameters
    which will indicate to the next page how it should update its
    CatalogNavHistory collector. In particular, to set the navCount,
    we query the CatalogNavHistory component, asking it for a number
    that identifies its current state.
  !>
  %>
  <dsp:a href="CategoryPage.jsp">
   <dsp:valueof param="element.displayName"/>
   <dsp:param param="element.repositoryId" name="itemId"/>
   <dsp:param bean="/atg/commerce/catalog/CatalogNavHistory.navCount"
        name="navCount"/>
```

```
   <dsp:param value="push" name="navAction"/>
  </dsp:a>
 </dsp:oparam>
</dsp:droplet>
```

顧客のパスのレンダリング

この例の後半の部分では、顧客が訪問した場所のリストをレンダリングする方法を示します。まず、このフラグメントが埋め込まれたページに渡された category パラメータ、navAction パラメータおよび navCount パラメータを、CatalogNavHistory を更新する CatalogNavHistoryCollector コンポーネントとともに使用して、CatalogNavHistory を現在の場所に更新します。

「戻る」ボタンが検出されると、CatalogNavHistory はスタックを消去し、カタログのデフォルトの親プロパティを使用して、現在のページに対応する新しいスタックを生成します。

```
<dsp:droplet name="/atg/commerce/catalog/CatalogNavHistoryCollector">
  <dsp:param param="categoryObj" name="item"/>
  <dsp:param param="navAction" name="navAction"/>
  <dsp:param param="navCount" name="navCount"/>
</dsp:droplet>
```

さらに、次のようなページ・フラグメントを使用してユーザーの証跡を表示します。サーブレット Bean は、CatalogNavHistory コンポーネントに格納された個々のカテゴリの名前をリストし、それらのカテゴリ名を表示するカテゴリページへのリンクを作成します。次のパラメータが渡されます。

- itemId - カテゴリの repositoryId が格納されます。
- navAction - pop という値が設定され、CatalogNavHistory が自分のスタックを消去し、デフォルトの親カテゴリ・プロパティを使用してスタックを再作成する必要があることを示します。
- navCount - CatalogNavHistory.navHistory 配列の個々のカテゴリ要素のインデックスが格納されます。

pageType パラメータは、どのタイプのページがサーブレット Bean を呼び出しているかを示します。カテゴリ・ページが呼び出している場合、CatelogNavHistory 内の最後のカテゴリ・エントリはリンクに変換されません。サーブレット Bean が製品ページから呼び出されている場合は、エントリが製品の親カテゴリを表しているため、最後のエントリはリンクに変換されます。

```
<dsp:droplet name="/atg/dynamo/droplet/ForEach">
 <dsp:param bean="/atg/commerce/catalog/CatalogNavHistory.navHistory"
            name="array"/>
 <dsp:oparam name="output">
  <dsp:droplet name="/atg/dynamo/droplet/Switch">
   <dsp:param param="count" name="value"/>
   <dsp:getvalueof id="nameval1" param="size" idtype="java.lang.String">
   <dsp:oparam name="<%=nameval1%>">
       <dsp:droplet name="/atg/dynamo/droplet/Switch">
        <dsp:param param="pageType" name="value"/>
        <dsp:oparam name="product">
         <dsp:a href="CategoryPage.jsp">
          <dsp:param param="element.repositoryId" name="itemId"/>
```

```
          <dsp:param value="pop" name="navAction"/>
          <dsp:param param="index" name="navCount"/>
          <dsp:valueof param="element.displayName"/>
         </dsp:a>
        </dsp:oparam>
        <dsp:oparam name="default">
         <dsp:valueof param="element.displayName"/>
        </dsp:oparam>
       </dsp:droplet>
      </dsp:oparam>
   </dsp:getvalueof>
   <dsp:oparam name="default">
    <dsp:a href="CategoryPage.jsp">
     <dsp:param param="element.repositoryId" name="itemId"/>
     <dsp:param value="pop" name="navAction"/>
     <dsp:param param="index" name="navCount"/>
     <dsp:valueof param="element.displayName"/>
    </dsp:a>
     &gt; 
   </dsp:oparam>
  </dsp:droplet>
 </dsp:oparam>
</dsp:droplet>
```

場所のジャンプの追跡

次の例では、階層型のナビゲーションから逸脱するハードコーディングされたリンクなど、通常の履歴コレクションの外の顧客のナビゲーションを追跡する方法を示します。

```
<a href="param:featuredProduct.template.url">
  <param name="id" value="param:featuredProduct.repositoryId">
  <param name="navAction" value="jump">
  <param name="navCount"
    value="property:/atg/commerce/catalog/CatalogNavHistory.navCount">
     link text
</a>
```

カタログの検索

一定の基準セットを満たす製品を顧客が見つけられるように、検索メカニズムを用意できます。この項には、カタログの検索に関する次の情報が記載されています。

追加の検索情報については、『ATG Web Commerce Page Developer's Guide』の*検索フォームの使用*を参照してください。

カタログ検索は、Oracle Commerce MDEX Engine を使用して構成することもできます。この構成の詳細は、『ATG-Endeca インテグレーション・ガイド』を参照してください。

カタログ検索の概要

Core Commerce は、製品やカテゴリなどの項目のカタログ・リポジトリを検索するための `atg.commerce.catalog.custom.CatalogSearchFormHandler` フォーム・ハンドラ・クラスを備えています。このフォーム・ハンドラを使用して、1 つ以上のカタログ項目タイプの検索を作成し、検索時にどのプロパティを調べるかを指定できます。

たとえば、指定された部分文字列が製品名に含まれている製品、指定されたキーワードでタグ付けされたカテゴリ、または指定されたプロパティ値のセットを持つ製品とカテゴリの両方を検索することができます。

フォーム・ハンドラのプロパティ・ファイルの構成の設定によって、検索の対象となる要素の種類、検索するときに考慮するそれらの要素のプロパティ、および個々の検索のタイプに関する追加の構成の詳細を指定します。一般的に、顧客は JSP 内のフォーム・フィールドを使用して検索するターゲット値を指定します。

`CatalogSearchFormHandler` クラスは、大半の Core Commerce アプリケーションにとって十分な機能と柔軟性を備えています。Core Commerce は、`/atg/commerce/catalog` 内に複数の `CatalogSearchFormHandler` コンポーネントを備えており、そのそれぞれが異なる検索オプションのセット向けに構成されています（事前構成されたカタログ検索コンポーネントを参照してください）。追加の `CatalogSearchFormHandler` コンポーネントを作成し、それらをコンポーネントのプロパティ・ファイルまたは ACC のコンポーネント・エディタを使用して構成できます。ストアがカスタム検索機能を必要とする場合は、`CatalogSearchFormHandler` を拡張するか、別のフォーム・ハンドラを作成できます。

次の 4 つのタイプの検索を実行するように `CatalogSearchFormHandler` を構成できます。

キーワード検索
キーワード検索では、キーワード・プロパティ名と入力検索文字列を使用して、製品とカテゴリのキーワードを検索します。顧客はキーワードのマッチングに使用される値を入力します。"Show me all products and categories with the keyword `shoe`"（キーワード shoe が含まれたすべての製品とカテゴリを表示）はキーワード検索の例です。

テキスト検索
全文検索では、テキスト・プロパティ名と入力検索文字列を使用して、プロパティに対するテキスト・パターン・マッチングを実行します。“Show me all products whose `longDescription` property contains the word `wool`”（longDescription プロパティに wool という単語が含まれたすべての製品を表示）は全文検索の例です。データベースが全文検索をサポートするように適切に構成されている必要があることに注意してください。詳細は、『ATG Web Commerce Installation and Configuration Guide』のデータベースおよびデータベース・アクセスの説明を参照してください。

階層検索
階層検索では、特定のカテゴリから始めて、そのカテゴリの子カテゴリ、その子カテゴリの子カテゴリなどを含むカテゴリのサブセットを検索の対象とします。特定のカテゴリは、`hierarchicalCategoryId` プロパティ内のリポジトリ ID によって示します。階層検索を実行するには、『ATG Web Commerce Programming Guide』の説明に従って、個々の製品項目およびカテゴリ項目の `ancestorCategories` プロパティを生成する必要があります。

高度な検索
高度な検索では、フォーム・ハンドラの `advancedSearchPropertyNames` プロパティで指定された個々のプロパティに対応する検索オプションを使用できます。たとえば、一定の数の値を使用して列挙型がリポジトリで定義されます。高度な検索では、これらの値を定義から取得して、選択ボックスに表示します。高度な問合せは、カタログ検索をさらに細かく定義するために、顧客によって選択されたオプションに基づいて作成されます。たとえば、高度な検索では、摘要、メーカーまたは価格に基づいた検索を顧客が実行できます。“Show me all products with the keyword shoe where price range is expensive”（キーワード shoe が含まれていて、価格範囲が高価であるすべての製品を表示）は高度な検索の例です。

事前構成されたカタログ検索コンポーネント

Core Commerce には、`/atg/commerce/catalog/custom` 内の事前構成された `CatalogSearchFormHandler` のインスタンスが 5 つ含まれています。

- `CatalogSearch` は、キーワード、摘要および表示名を検索し、一致する製品とカテゴリを見つけます。
- `CategorySearch` は、キーワードと摘要を検索し、一致するカテゴリのみを見つけます。
- `ProductSearch` は、キーワードと摘要を検索し、一致する製品のみを見つけます。
- `ProductTextSearch` は、キーワードではなく、摘要フィールドのみを検索し、一致する製品のみを見つけます。
- `AdvProductSearch` は、キーワード検索とテキスト検索を階層型検索および高度な検索と組み合せて、すべての検索基準に一致する製品を見つけます。

これらのコンポーネントがどのように構成されているかを確認するには、ACC コンポーネント・エディタでコンポーネントを開き、コンポーネントのプロパティの設定を表示します。

注意: プレビュー・サーバー上でプレビュー・モードを実行している場合、5 つの事前構成された検索コンポーネントは、`atg/commerce/catalog/custom` 内の `FilteringSearchFormHandler` のインスタンスです。詳細は、プレビュー・モードでのカタログの検索を参照してください。

検索フォーム・ハンドラの構成

`CatalogSearchFormHandler` は、カタログ・リポジトリ内の任意のタイプのリポジトリ項目を検索できます。フォーム・ハンドラのプロパティ・ファイル内の `itemTypes` プロパティをそれぞれが 1 個の項目タイプの名前を示す文字列のリストに設定することによって、検索の対象となる項目タイプを指定します。

項目タイプには、一般的に、category または product が含まれていますが、ユーザーが作成した SKU、カスタム・カテゴリまたは製品サブタイプを検索するように検索フォーム・ハンドラを構成できます。CatalogSearchFormHandler の複数のインスタンスを作成して、様々な種類のオブジェクトを検索するようにそれらのインスタンスを構成できます。たとえば、衣料製品を検索するフォーム・ハンドラと、すべての製品およびカテゴリを検索するフォーム・ハンドラを設定できます。その場合、衣料品の検索ページでは最初のフォーム・ハンドラを使用し、より一般的な検索ページでは 2 番目のフォーム・ハンドラを使用します。

検索の対象となる項目タイプを指定するだけでなく、カテゴリ、製品、SKU、およびその他のカタログ情報へのアクセスを提供する CatalogTools オブジェクトを参照するように catalogTools プロパティも設定する必要があります。独自のカタログ管理システムを実装している場合を除いて、/atg/commerce/catalog/CatalogTools にあるデフォルトの CatalogTools コンポーネントを使用する必要があります。

次の例は、4 つのタイプの検索すべてを実行できる CatalogSearchFormHandler コンポーネント用のプロパティ・ファイルを示しています。

```
$class=atg.commerce.catalog.custom.CatalogSearchFormHandler
$scope=session

doKeywordSearch=true
keywordsPropertyNames=keywords

doTextSearch=true
textSearchPropertyNames=description,displayName

doHierarchicalSearch=true
ancestorCategoriesPropertyName=ancestorCategories

doAdvancedSearch=true
advancedSearchPropertyNames=weightRange,manufacturer,childSKUs

catalog^=/atg/commerce/catalog/custom/CatalogTools.catalog
itemTypes^=/atg/commerce/customCatalogTools.productItemTypes
```

検索オプションには多数の組合せがあるため、複数の CatalogSearchFormHandler コンポーネントをそれぞれ別々の方法で構成すると便利です。Core Commerce には、5 つの異なる CatalogSearchFormHandler コンポーネントが含まれており（事前構成されたカタログ検索コンポーネントを参照）、追加のインスタンスを作成できます。非表示の入力フィールドを設定することによって、ページごとに CatalogSearchFormHandler コンポーネントの動作を変更することもできます。たとえば、キーワード検索専用に構成されている KeywordSearch という名前の CatalogSearchFormHandler コンポーネントが含まれている Core Commerce アプリケーションで、テキスト検索もできるようにしたいページが 1 ページあるとします。その場合は、このコンポーネントを使用して次のタグを含めることによって、そのページだけでテキスト検索を有効にできます。

```
<dsp:input type="hidden" bean="KeywordSearch.doTextSearch" value="true">
```

カタログ検索タイプの構成

前述のように、CatalogSearchFormHandler は、4 つのタイプの検索機能を備えています。

- キーワード検索
- テキスト検索
- 階層検索
- 高度な検索

この項では、SearchFormHandler コンポーネントを、これらの検索のタイプごとに構成する方法を説明します。

キーワード検索

フォーム・ハンドラでキーワード検索を有効にするには、doKeywordSearch プロパティを true に設定します。検索フォーム・ハンドラの keywords プロパティを文字列の配列に設定するか、searchInput プロパティを、スペースで区切られた 1 つ以上の単語が含まれた文字列に設定することによって、検索対象となるターゲット値を指定できます。これらの値は、一般的に、フォーム入力フィールドを介して顧客によって指定されます（フォーム・ハンドラの keywordInputSeparator プロパティを設定することによって、searchInput の解析に使用される区切り文字を変更できます）。

toUpperCaseKeywords プロパティを true に設定することによって、検索フォーム・ハンドラに対して、検索を実行する前に、すべてのキーワード入力を大文字に変換するよう強制できます。同様に、toLowerCaseKeywords プロパティを true に設定することによって、検索フォーム・ハンドラに対して、検索を実行する前に、キーワード入力を小文字に変換するよう強制できます。

デフォルトでは、キーワード検索は個々のカタログ項目の keywords プロパティに注目します。フォーム・ハンドラのプロパティ・ファイル内にあるフォーム・ハンドラの keywordsPropertyNames プロパティを設定することによって、デフォルト動作を上書きできます。キーワード検索で考慮の対象となる、それぞれが単一値または複数値である 1 つ以上のプロパティを指定できます。

キーワード検索では、単一値プロパティと複数値プロパティを別々に扱います。keywordsPropertyNames で指定されているプロパティが単一値（つまり、プロパティが String 型）の場合、キーワード検索アルゴリズムは QueryBuilder.CONTAINS 問合せを使用して個々のターゲット値を調べて、ターゲット値がプロパティの現在の値の内部にあるかどうかを確認します。たとえば、String 型の keywordString プロパティがある場合、キーワード検索を使用して red、green および blue という値を検索すると、結果として生成される問合せは次のようになります。

```
keywordString CONTAINS "red"
OR keywordString CONTAINS "green"
OR keywordString CONTAINS "blue"
```

CONTAINS は部分文字列マッチングを実行するため、この問合せは、値が reduced calorie である keywordString を持っている項目に対して true を返します。その理由は reduced に文字列 red が含まれているからです。

しかし、keywordsPropertyNames で指定されているプロパティが複数値（たとえば、プロパティが String[]型）である場合、キーワード検索アルゴリズムは QueryBuilder.INCLUDESANY 問合せを使用して、プロパティ内の任意の値と検索基準のセット内の任意の値との完全一致を求める単一検索を実行します。たとえば、String[]型の keywordString プロパティがある場合、キーワード検索を使用して red、green および blue という値を検索すると、結果として生成される問合せは次のようになります。

```
keywords INCLUDES ANY ["red","green","blue"]
```

INCLUDES ANY は完全一致を検索するため、この問合せは、diet および reduced calorie というキーワードを持つ項目に対して false を返します。その理由は red が reduced calorie の完全一致ではないからです。

keywordsPropertyNames で複数のプロパティを指定すると、キーワード検索は個々のプロパティに対する問合せを生成し、OR 演算子を使用してそれらの問合せを結合します。つまり、いずれかの問合せが true を返せば、検索操作によってその項目が返されます。

テキスト検索

フォーム・ハンドラでテキスト検索を有効にするには、doTextSearch プロパティを true に設定します。ターゲット検索文字列は、一般的に、顧客がフォーム入力フィールドに値を入力してフォーム・ハンドラの searchInput プロパティを設定することによって指定されます。フォーム・ハンドラの textSearchPropertyNames プロパティを設定することによって検索対象となるプロパティを指定します。このプロパティが設定されていない場合、テキスト検索は、リポジトリによって定義されているデフォルトのプロパティ・セットを使用します。

テキスト検索の実装は RDBMS に固有であり、データベースがテキスト検索機能を備えていれば、それを使用します。データベースが全文検索エンジンをサポートしている場合は、検索エンジンを適切に構成し、リポジトリ・コンポーネントの simulateTextSearchQueries プロパティを false に設定する必要があります。RDBMS が全文検索エンジンをサポートしていないため、または RDBMS がサポートしている全文検索エンジンのライセンスがないため、全文検索エンジンが使用できない場合、SQL リポジトリは、LIKE 演算子を使用して、ターゲット値が検査対象のテキスト・プロパティのいずれかの部分文字列かどうかを判断することによって、全文検索をシミュレートできます。この機能を有効にするには、リポジトリ・コンポーネントの simulateTextSearchQueries プロパティを true に設定します。シミュレートされた全文検索は開発の目的には有用ですが、実稼働環境サーバーに適したパフォーマンスが得られる可能性は低いことに注意してください。

デフォルトでは、製品カタログの simulateTextSearchQueries は true に設定されます。カタログを全文検索用に構成する方法の詳細は、『ATG Web Commerce Programming Guide』を参照してください。

階層検索

フォーム・ハンドラで階層検索を有効にするには、doHierarchicalSearch プロパティを true に設定します。フォーム・ハンドラの hierarchicalCategoryId プロパティを、検索対象を子として持つカテゴリのリポジトリ ID に設定することによって、検索を開始するカテゴリを指定します。このプロパティは、一般的に、非表示の入力タグを介して設定されるか、フォーム入力フィールドを介して顧客によって指定されます。

階層検索では、個々のカテゴリ項目および製品項目が、すべての上位階層カテゴリの完全なリストを値として持つ複数値プロパティを持っている必要があります。階層検索では、hierarchicalCategoryId を介して指定されたカテゴリが上位階層カテゴリに含まれている項目に検索結果が限定されます。

フォーム・ハンドラの ancestorCategoriesPropertyName プロパティを設定することによって上位階層カテゴリ・プロパティを指定します。個々のカテゴリ項目と製品項目が、その項目の上位階層カテゴリのセットを値として持つ ancestorCategories プロパティを持っています。/atg/commerce/catalog/custom/AncestorGeneratorService コンポーネントを使用して、このプロパティの値を自動的に生成できます。AncestorGeneratorService の詳細は、『ATG Web Commerce Programming Guide』の*カタログの保守システムの使用*に関する項を参照してください。

高度な検索

高度な検索を有効にするには、フォーム・ハンドラの doAdvancedSearch プロパティを true に設定します。次に、advancedSearchPropertyNames プロパティを設定することによって、検索対象となるプロパティのセットを指定します。高度な検索は、ここで指定するプロパティのセットに限定されます。

一般的に、フォーム入力フィールドを介してフォーム・ハンドラの propertyValues プロパティに値を追加することによって、1 つ以上のこれらのプロパティに対応するターゲット値を指定します。このプロパティは、検索対象となるプロパティごとに 1 つのキー/値ペアが追加されるディクショナリです。キーはプロパティ名であり、値は検索対象です。たとえば、color プロパティが red に設定されている項目を検索するには、CatalogSearchFormHandler.propertyValues.color を red に設定します。値を空の文字列に設定することによって、検索から値が除外されるため、プロパティが任意の値と一致する動作が指定されます。

propertyValues で指定された個々のプロパティに対して、プロパティが単一値または複数値のどちらであるかに応じて、問合せが生成されます。単一値プロパティの場合は、単純な等価性の検査が使用されます。複数値プロパティの場合は、任意のプロパティの値がターゲット値と一致すれば問合せが成功するように、INCLUDES 検査が生成されます。複数のプロパティを指定した場合は、すべてのプロパティが一致しないとカタログ項目が選択されないように、AND 演算子を使用して問合せが結合されます。

たとえば、color が単一の文字列で、availableSizes が文字列の配列である場合に、red を値として持つ color および medium を値として持つ availableSizes を検索すると、結果として次の問合せが生成されます。

```
(color = red) AND (availableSizes INCLUDES medium)
```

CatalogSearchFormHandler は、propertyValuesByType と呼ばれるプロパティを持っています。このプロパティは、タイプが enumerated または RepositoryItem のどちらかである、advancedSearchPropertyValues で指定された個々のプロパティに対応する 1 つのキー/値ペアが含まれたディクショナリです。キーはプロパティの名前であり、値はプロパティが取り得る値のコレクションです。propertyValuesByType プロパティは、たとえば、small、medium、large のような事前定義された値のセットを持つ列挙型のプロパティである項目のサイズを顧客が選択するためのフォームの作成に役立ちます。このプロパティを使用する例を次に示します。

```
<!-- Create a select field to choose size and a default option -->
Size: <dsp:select bean="SearchHandler.propertyValues.size">

<!-- Create a default choice -->
<dsp:option value="" selected="true"/>Any size

<!-- Now create an option for each defined value for size -->
<dsp:droplet name="ForEach">
 <dsp:param value="SearchHandler.propertyValuesByType.size" name="array"/>
 <dsp:oparam name="output">
 <dsp:getvalueof id="option11" param="element" idtype="java.lang.String">

<dsp:option value="<%=option11><dsp:valueof param="element">Unknown
size</dsp:valueof>

 </dsp:oparam>
</dsp:droplet>

</dsp:select>
```

タイプが enumerated または RepositoryItem のプロパティの取り得る値を決定するための別のアプローチは、/atg/commerce/catalog/RepositoryValues にあるサーブレット Bean を使用する方法です。このコンポーネントは atg.repository.servlet.PossibleValues クラスのインスタンスです。このサーブ

レット Bean の詳細は、『ATG Web Commerce Page Developer's Guide』の PossibleValues に関する項を参照してください。

注意: プレビュー・モードで実行している場合は、/atg/commerce/catalog/RepositoryValues コンポーネントが atg.commerce.catalog.FilteringCatalogPossibleValues のインスタンスになります。詳細は、プレビュー・モードでのカタログの検索を参照してください。

RepositoryValues サーブレット Bean は、CatalogSeachFormHandler クラスの propertyValuesByType プロパティと実質的に同じ情報を返します。ただし、重要な違いがいくつかあります。

- RepositoryValues は同時に 1 つの項目タイプだけの取り得る値を決定するのに対して、propertyValuesByType プロパティは同時に複数の項目タイプを対象とします。
- RepositoryValues はリポジトリ項目の任意のプロパティの値を検索できるのに対して、propertyValuesByType は、フォーム・ハンドラの advancedSearchPropertyNames プロパティで指定されたプロパティのみを対象とします。
- RepositoryValues は JSP 内のどこでも機能するのに対して、propertyValuesByType プロパティは、SearchFormHandler クラスを使用して作成する検索フォーム内でしか使用できません。

カタログ検索タイプの組合せ

CatalogSearchFormHandler クラスでは、1 つの要求で複数の検索タイプを指定できます。たとえば、キーワードとテキストの両方を検索したり、高度な検索と階層検索を組み合せて特定のカテゴリ内の項目のみを検索したりできます。実際、検索タイプの任意の組合せを使用できます。

検索タイプを組み合せるときのルールを次に示します。

- テキスト検索とキーワード検索は OR 演算子を使用して結合されるため、どちらの基準セットが一致しても、項目が選択され、検索結果に含まれます。
- 階層検索と高度な検索は AND 演算子を使用して結合されるため、検索の範囲は、指定されたテキスト検索の基準またはキーワード検索の基準の他に、階層検索および高度な検索の要件を満たす項目に限定されます。

問合せの形式は次のとおりです。

```
(KeywordConditions OR TextConditions) AND HierarchicalConditions AND
AdvancedSearchConditions
```

たとえば、映画のカタログがあって、4 つの検索タイプすべてを使用できるように検索フォーム・ハンドラを構成したとします。顧客が次の検索基準を入力します。

```
keywords=comedy
textSearchPropertyNames=description
searchInput=Steve Martin
hierarchicalCategoryId=BudgetMovies
propertyValues.format=DVD
```

この検索では、すべてのコメディ映画プラス摘要に Steve Martin が記述されているすべての映画が見つかりますが、BudgetMovies カテゴリ内にあって、DVD として市販されている、それらの映画のサブセットのみが返されます。

検索の処理

ページに検索を実装するには、JSP に検索サーブレット Bean が含まれていることを確認してください。次の例では、CatalogSearch を使用していますが、他の任意のサーブレット Bean を使用して、範囲を絞った検索を実行できます。

```
<IMPORTBEAN BEAN="/atg/commerce/catalog/CatalogSearch">
```

SearchFormHandler は、handleSearch メソッドが呼び出されたときに、検索問合せを実行します。一般的には、次の例に見られるように、フォーム・ハンドラの search プロパティを発行ボタンに関連付けます。

```
<dsp:input bean="/myCatalog/SearchForm.search" value="Go" type="submit"/>
```

検索を特定の 1 つのカタログまたはカタログのセットに限定できます。CatalogSearchFormHandler および FilteringSearchFormHandler の両方に catalogs プロパティおよび queryByCatalog プロパティが含まれています。catalogs プロパティにはカタログ ID の配列が含まれています。このプロパティに値が入力されていると、指定されたカタログのうちの少なくとも 1 つのメンバーシップを持っている項目のみが返されます。catalogs プロパティは null であるが、queryByCatalog プロパティが true に設定されている場合、問合せはユーザーの現在のカタログに限定されます。catalogs が null で、queryByCatalog が false の場合、問合せはすべてのカタログを検索します。

複数サイト機能を使用している場合は、次のパラメータを使用して、検索を特定の 1 つのサイトまたは複数のサイトに限定することもできます。

- siteIds - 指定されたサイトに属する項目のみが返されます。
- siteScope - 項目を検索するときに使用するサイト・スコープを指定します。次の値を指定できます。
  - current - 現在のサイトのみから一致するリポジトリ項目を返します。これがデフォルト値です。
  - all - すべての一致するリポジトリ項目を返し、サイト・コンテキストに基づくフィルタリングは行いません。
  - any - 任意のサイトに属する一致するリポジトリ項目を返します。
  - none - どのサイトにも属していない一致するリポジトリ項目を返します。
  - *shareable_type_ID* - ShareableType コンポーネント ID によって定義される、現在のサイトと同じ共有グループ内にある任意のサイトに属している一致するリポジトリ項目を返します。たとえば、現在のサイトとショッピング・カートを共有しているサイトに属している項目を返すことができます。

次の例は、siteIds プロパティを使用してサイトによって検索結果をフィルタする製品検索フォームを示しています。

```
<dsp:importbean bean="/atg/dynamo/droplet/ForEach" />
<dsp:importbean bean="/atg/multisite/Site"/>
<dsp:importbean bean="/atg/commerce/catalog/ProductSearch"/>
<dsp:getvalueof id="contextroot" idtype="java.lang.String" bean="/Originating
Request.contextPath"/>

<dsp:form action="${contextroot}/search/searchResults.jsp" method="post" id=
"simpleSearch" formid="simplesearchform">

    <%-- Search input control --%>
```

```
    <dsp:input bean="ProductSearch.searchInput" type="text" value="" />

    <%-- Get the list of sites that share a shopping cart with the
    current site. --%>
    <dsp:droplet name="SharingSitesDroplet">
      <dsp:param name="shareableTypeId" value="atg.ShoppingCart"/>
      <dsp:param name="excludeInputSite" value="true"/>

      <%-- Loop through the sites that share a shopping cart and render labels
      and checkboxes for them. --%>
      <dsp:oparam name="output">
        <dsp:droplet name="ForEach">
          <dsp:param name="array" param="sites"/>
          <dsp:setvalue param="site" paramvalue="element"/>

            <%-- Display a checkbox and name for the current site first. --%>
            <dsp:oparam name="outputStart">
              <dsp:input bean="ProductSearch.siteIds" type="checkbox"
                         beanvalue="Site.id" checked="true" id="currentStore"/>
              <label for="currentStore">
                <dsp:valueof bean="Site.name"/>
              </label>
            </dsp:oparam>

<%-- Display the other sites that share shopping cart with the current
            site. --%>
            <dsp:oparam name="output">
                <dsp:input bean="ProductSearch.siteIds" type="checkbox"
                     paramvalue="site.id" id="otherStore" checked="false"/>
                <label for="otherStore">
                  <dsp:valueof param="site.name"/>
                </label>
            </dsp:oparam>
        </dsp:droplet>
      </dsp:oparam>

      <%-- If there are no shared sites, include the current site only and
      don't display checkboxes. --%>
      <dsp:oparam name="empty">
        <dsp:input bean="ProductSearch.siteIds" type="hidden"
                   beanvalue="Site.id"/>
      </dsp:oparam>
    </dsp:droplet>

    <%-- Display the search form's submit button. --%>
    <dsp:input bean="ProductSearch.search" type="submit"  value="Search"/>

</dsp:form>
```

検索結果の表示

問合せを実行した後、SearchFormHandler は、同じ情報が含まれているが、情報を整理する方法が異なる 2 つの異なるプロパティで検索結果を使用可能にします。

- searchResults プロパティは、検索基準を満たしたすべてのカタログ項目のコレクションです。複数の項目タイプ（カテゴリと製品など）を検索した場合は、項目のタイプに関係なく、検索によって返されたすべての項目がリストに表示されます。
- searchResultsByItemType プロパティは、検索した個々の項目タイプにつき 1 つのキー/値ペアが含まれたハッシュマップです。キーは項目タイプ名（フォーム・ハンドラの itemTypes プロパティで指定された値）であり、値は検索基準を満たしたそのタイプの項目のコレクションです。

たとえば、カタログ・スキーマでカテゴリと製品を検索した場合、searchResultsByItemType プロパティには、一致するカテゴリのコレクションを値に持つ category と呼ばれるキーと、一致する製品のコレクションを値に持つ product と呼ばれる別のキーが含まれます。searchResults プロパティには、一部の項目がカテゴリであり、一部の項目が製品であるコレクションが含まれます。

個々のコレクション内で、項目はソートされませんが、データベースから取得された順に表示されます。サーブレット Bean のソート機能（ForEach など）を使用して、項目が表示される順番を制御できます。

次の例では、ForEach と searchResultsByItemType を組み合せて使用して、検索によって返された製品のみを表示名順にソートして表示します。

```
Your search returned the following products:

<dsp:droplet name="ForEach">
 <dsp:param value="CatalogSearch.searchResultsByItemType.product" name="array"/>
 <dsp:param value="+displayName" name="sortProperties"/>

<dsp:oparam name="outputStart">
  <ul>
</ dsp:oparam>

 <dsp:oparam name="output">
   <li><dsp:valueof param="element.displayName">Unknown product</dsp:valueof></li>
 </dsp:oparam>

<dsp:oparam name="outputEnd">
  </ul>
</dsp:oparam>

 <dsp:oparam name="empty">
   <p>No matching products were found.
 </dsp:oparam>
</dsp:droplet>
```

次の例は、前述の例より長く、より多くの使用可能なオプションが含まれています。

```
<%@ taglib uri="http://www.atg.com/dsp.tld" prefix="dsp" %>
<dsp:page>
```

```
<%--
-------------------------------------------------------------
This JSP bean displays the contents of a search
that potentially returns both category and product repository items.
The one paramater, ResultArray, accepts a HashMap that contains
elements with the keys "category" and "product".  The values of these
keys are collections of category or product repository items found in
the search.
-------------------------------------------------------------
--%>

<dsp:importbean bean="/atg/dynamo/droplet/Switch"/>
<dsp:importbean bean="/atg/dynamo/droplet/IsEmpty"/>
<dsp:importbean bean="/atg/dynamo/droplet/ForEach"/>
<dsp:importbean bean="/atg/dynamo/droplet/RQLQueryForEach"/>

<dsp:droplet name="ForEach">
  <dsp:param param="ResultArray" name="array"/>

<%--Each item in this array is a Collection of Categories or
          Products...--%>
  <dsp:param value="ResultCollection" name="elementName"/>

  <dsp:oparam name="output">
    <dsp:droplet name="Switch">

<%--The key tells us if this is a Collection of Products
              or Categories:--%>
      <dsp:param param="key" name="value"/>

<%--For the list of CATEGORIES: --%>
      <dsp:oparam name="category">

        <blockquote>

        <dsp:droplet name="Switch">
          <dsp:param param="ResultCollection" name="value"/>
          <dsp:oparam name="default">
            <p>

<%--For each Category in the Collection: --%>
            <dsp:droplet name="ForEach">
              <dsp:param param="ResultCollection" name="array"/>
              <dsp:param value="+displayName" name="sortProperties"/>
              <dsp:param value="Category" name="elementName"/>
              <dsp:oparam name="outputStart">
                <b>We found these categories matching your search</b>
                <p>
```

```
              </dsp:oparam>
              <dsp:oparam name="output">


<%-- Display a link to the Category: --%>
                <dsp:getvalueof id="a78" param="Category.template.url"
                                          idtype="java.lang.String">
<dsp:a href="<%=a78%>">
                  <dsp:param param="Category.repositoryId" name="id"/>
                  <dsp:param value="jump" name="navAction"/>
                  <dsp:param param="Category" name="Item"/>
                  <dsp:valueof param="Category.displayName">No
                       name</dsp:valueof></dsp:a></dsp:getvalueof>
                <br>
              </dsp:oparam>
              <dsp:oparam name="empty">
                <b>There are no categories matching your search</b>
                <p>
              </dsp:oparam>
            </dsp:droplet>
          </dsp:oparam>

<%--If NO Categories returned by the search: --%>
          <dsp:oparam name="unset">
            No category items in the catalog could be found that match your query
          </dsp:oparam>
        </dsp:droplet>
<%--ForEach Category--%>

        </blockquote>
        <P>
      </dsp:oparam>

<%--For the list of PRODUCTS: --%>
      <dsp:oparam name="product">
        <blockquote><p>

        <dsp:droplet name="Switch">
          <dsp:param param="ResultCollection" name="value"/>

          <dsp:oparam name="default">


<%--For each Product in the Collection: --%>
            <dsp:droplet name="ForEach">
              <dsp:param param="ResultCollection" name="array"/>
              <dsp:param value="+displayName" name="sortProperties"/>
              <dsp:param value="Product" name="elementName"/>
              <dsp:oparam name="outputStart">
```

```
                <p>
                <b>We found these products matching your search</b>
                <p>
              </dsp:oparam>
              <dsp:oparam name="output">

<%-- Display a link to the Product: --%>
                <dsp:getvalueof id="a173" param="Product.template.url"
                            idtype="java.lang.String">
<dsp:a href="<%=a173%>">
                  <dsp:param param="Product.repositoryId" name="id"/>
                  <dsp:param value="jump" name="navAction"/>
                  <dsp:param param="Product" name="Item"/>
                  <dsp:valueof param="Product.displayName">No name</dsp:valueof>
                    - <dsp:valueof param="Product.description"/>
                </dsp:a></dsp:getvalueof>

                <br>
              </dsp:oparam>
              <dsp:oparam name="empty">
                <b>There are no products matching your search</b>
                <p>
              </dsp:oparam>

            </dsp:droplet>
<%--ForEach Product--%>

          </dsp:oparam>


<%--If NO Products returned by the search:--%>
          <dsp:oparam name="unset">
           No product items in the catalog could be found that match your query<p>
          </dsp:oparam>

        </dsp:droplet>
        </blockquote><P>
      </dsp:oparam>
    </dsp:droplet>

  </dsp:oparam>

</dsp:droplet>
<%--ForEach Item returned by Search --%>

</dsp:droplet>

</dsp:page>
```

プレビュー・モードでのカタログの検索

Content Administration を使用してカタログを開発している場合は、実稼働環境に配置されていない（プレビュー・モードの）カタログに対する検索が配置済のカタログに対する検索と少し異なる方法で実行されることに注意してください。2 つの主な違いは、プレビュー・モードでは、次の現象が起きることです。

- 5 つの事前構成された検索コンポーネントが `atg.commerce.catalog.FilteringSearchFormHandler` のインスタンスになります。
- `/atg/commerce/catalog/RepositoryValues` コンポーネントが `atg.commerce.catalog.FilteringCatalogPossibleValues` のインスタンスになります。

次の例は、同じ結果を取得する方法のモードによる違いを示しています。

- **実稼働環境モードの場合:**`SearchFormHandler` クラスと `CatalogPossibleValues` クラスは、デフォルトで、検索結果をユーザーの現在のカタログに限定します（検索の処理を参照してください）。ユーザーが実稼働環境モードで“red”に対するキーワード検索を実行すると、問合せには“where `keywordString` CONTAINS ‘red’”句が含まれ、`SearchFormHandler` が別の句を追加することによって、問合せを“where `keywordString` CONTAINS ‘red’ and `catalogs` CONTAINS (the user’s current catalog)”にします。したがって、リポジトリはユーザーの現在のカタログ内に存在する項目のみを返します。
- **プレビュー・モードの場合:**カテゴリ、製品および SKU の `catalogs` プロパティは導出プロパティであり、問合せ不能です。したがって、`SearchFormHandler` は、検索の範囲をユーザーの現在のカタログだけに絞るための余分な句を追加できません。そのため、問合せは余分な句なしに実行され、ユーザーの現在のカタログにない項目が返される可能性があります。結果セットを返す前に、`FilteringCatalogSearchFormHandler` が結果に対して繰り返し処理を実行し、個々の結果の `catalogs` プロパティを確認します。`catalogs` にユーザーの現在のカタログが含まれていない項目はリストから削除され、ユーザーの現在のカタログにある項目のみが含まれた「フィルタされた」結果セットが返されます。

国際化されたカタログでの検索フォーム・ハンドラの使用

一部の企業は、製品情報を様々な言語で表示したり、顧客の居住国によって表示する製品のセットを変更できる国際化されたカタログを必要とします。

`CatalogTools` コンポーネントには、`alternateRepositories` と呼ばれるプロパティが含まれています。このプロパティを使用すれば、識別名（*リポジトリ・キー*と呼ばれる）と製品カタログとして使用される代替リポジトリとの間のマッピングを指定できます。そうすれば、顧客のロケールをリポジトリ・キーとして使用して、表示するカタログのバージョンを決定できます。

顧客が国際化されたサイトでカタログを検索する場合は、顧客の言語またはロケールに固有のカタログを検索できるようにします。そうするには、検索フォーム・ハンドラの `repositoryKey` プロパティを、検索対象となるリポジトリを識別する名前に設定します。検索フォーム・ハンドラは `repositoryKey` を使用して `CatalogTools` コンポーネントから適切なカタログを取得します。`repositoryKey` を設定しない場合は、カタログ・リポジトリが使用されます。

次の例に示すように、`repositoryKey` プロパティは、一般的に、検索フォーム内の非表示の入力フィールドを介して設定されます。

```
<dsp:input value='<dsp:valueof bean="Profile.locale"/>' type="hidden"
bean="MySearchFormHandler.repositoryKey">
```

repositoryKey を CatalogTools コンポーネントの alternateRepositories プロパティと組み合せて使用することによって、カタログを検索したときに適切な製品だけが確実に顧客に表示されます。

10 製品比較の実装

コマース・サイトは、多くの場合、サイト・ユーザーが製品カタログ内の品目を比較できる機能を必要とします。単純なサイトでは、1 つの製品比較リストをユーザーに提供し、ユーザーがリスト内で製品を追加および削除したり、リストで製品の特性を比較したりできるようにします。より複雑なサイトでは、タイプの異なる品目を比較するための複数の比較リスト(たとえば、カメラを比較するリストとテレビを比較するリストなど)をユーザーに提供します。Oracle Commerce Core Commerce は、これらの単純な要件と複雑な要件の両方を満たす製品比較システムを備えています。

Core Commerce 製品比較システムのデフォルトの実装を使用すれば、ユーザーは、製品のカテゴリ、製品、SKU および在庫に関する情報を使用して任意の数の製品を比較できます(アプリケーション開発者がシステムを拡張して、追加の情報を含めている可能性があることに注意してください)。さらに、ページ開発者は、製品比較情報を表形式で表示でき、ユーザーはその表を操作して、表示される情報のソート基準を変更できます。

製品比較システムを実装する方法を説明するこの章は、次の項から構成されています。

ProductList コンポーネントの理解

Core Commerce には、Nucleus の `/atg/commerce/catalog/comparison/ProductList` にある `ProductComparisonList` のセッション限定インスタンスがデフォルトで含まれています。ただし、アプリケーション開発者が複数の比較リスト(たとえば、カメラを比較するリスト、テレビを比較するリストなど)を管理するために、`ProductComparisonList` の追加のインスタンスを構成している可能性があります。

`ProductList` コンポーネントの `items` プロパティには、製品比較リスト内の個々の製品を表す `Entry` オブジェクトのリストが格納されます。個々の `Entry` オブジェクトは、カテゴリ、製品、SKU および在庫の情報を 1 つのオブジェクトにまとめて、次の表で説明するプロパティをデフォルトで公開します(開発者が製品比較システムを拡張して、プロパティを追加している可能性があることに注意してください)。

プロパティ名	プロパティのタイプ	説明
`product`	`RepositoryItem`	比較の対象となる製品。
`category`	`RepositoryItem`	比較の対象となる製品のカテゴリ。製品がリストに追加されたときにカテゴリが明示的に設定されていない場合は、製品のデフォルトの親カテゴリが使用されます。製品のデフォルトの親カテゴリが設定されていない場合は、`category` プロパティが null になります。

プロパティ名	プロパティのタイプ	説明
sku	RepositoryItem	製品の SKU。製品がリストに追加されたときに SKU が明示的に設定されていない場合は、製品の childSkus リスト内の最初の SKU が使用されます。製品が子 SKU を持っていない場合は、sku プロパティが null になります。
inventoryInfo	InventoryData	特定の製品および SKU の在庫ステータスを表す InventoryData オブジェクト。sku プロパティが null である場合、または在庫情報が利用できない場合は、inventoryInfo プロパティが null になります。

productLink	String	カタログ内の製品のページにリンクしているアンカー・タグを指定する HTML フラグメント。リンクのデフォルト形式はproduct.displayNameです。ただし、ProductComparisonList.productLinkFormat プロパティを設定することによって形式を変更できます。
		注意: 表形式で製品比較情報を表示する場合は、次の例に示すように、表情報を保持する TableInfo オブジェクトの構成で productLink プロパティを使用できます。
		`columns=\`
		`  Product Name=productLink,\`
		`  Price=sku.listPrice,\`
		`  …`
		あるいは、同様に、表の列に製品リンクを表示する一方で、製品の表示名で列をソートする場合は、次の方法で例を変更できます。
		`columns=\`
		`  Product Name=productLink;`
		`product.displayName,\`
		`  Price=sku.listPrice,\`
		`  …`
		TableInfo コンポーネントの詳細は、『ATG Web Commerce Page Developer's Guide』の*ソート可能な表の実装*を参照してください。

categoryLink	String	カタログ内のカテゴリのページにリンクしているアンカー・タグを指定する HTML フラグメント。リンクのデフォルト形式はcategory.displayNameです。ただし、ProductComparisonList.categoryLinkFormat プロパティを設定することによって、形式を変更できます。 **注意:** productLink プロパティと同様に、categoryLink プロパティも TableInfo コンポーネントの構成で使用できます。詳細は、この表の productLink プロパティの説明を参照してください。

プロパティ名	プロパティのタイプ	説明
id	Int	リスト・エントリを指定する一意の ID。Java コードで ProductComparisonList.getItems(id)を呼び出すことによって、または JSP で<dsp:valueof bean="ProductList.entries[id]"/>を使用することによって、このプロパティを使用して個別のエントリを取得できます。たとえばフォーム・ハンドラでこのプロパティを使用して、特定のエントリを削除することもできます。

次の例に示すように、よく知られた JSP 構文を使用して、製品比較リスト内のエントリのプロパティ(つまり Entry オブジェクト)を参照できます。

```
<dsp:droplet name="ForEach">
  <dsp:param bean="ProductComparisonList.items" name="array"/>

  <dsp:oparam name="output">
    <p>Product Name: <dsp:valueof param="element.product.displayName"/><br>
       Category: <dsp:valueof param="element.category.displayName"/><br>
       Inventory: <dsp:valueof
                  param="element.inventoryInfo.inventoryAvailabilityMsg"/><br>
  </dsp:oparam>

</dsp:droplet>
```

製品比較リストに対する問合せ

カテゴリ、製品および SKU を指定すれば、ProductListContains サーブレット Bean は、製品比較リストにその製品が含まれているかどうかを問い合せます。

Core Commerce には、Nucleus の/atg/commerce/catalog/comparison/ProductListContains にある ProductListContains のグローバルスコープのインスタンスがデフォルトで含まれています。

ProductListContains サーブレット Bean は次の入力パラメータを取ります。

- productList(必須)
  検査の対象となる ProductComparisonList オブジェクト。
- productID(必須)
  productList 内で検索する製品のリポジトリ ID。
- categoryID
  productList 内で検索するカテゴリのリポジトリ ID。

  製品のカテゴリ ID を指定しない場合、ProductListContains は、category プロパティがその製品のデフォルト・カテゴリと一致するリスト・エントリを検索し、その製品のデフォルト・カテゴリがない場合は、category プロパティが null であるリスト・エントリを検索します。

- skuID
  productList 内で検索する SKU のリポジトリ ID。

  製品の SKU を指定しない場合、ProductListContains は、sku プロパティがその製品の最初の子 SKU と一致するリスト・エントリを検索し、その製品の SKU がない場合は、sku プロパティが null であるリスト・エントリを検索します。
- repositoryKey
  検索する品目が含まれている製品カタログ・リポジトリを選択するために CatalogTools に渡すキー。キーからカタログへのマッピングは、CatalogTools で定義されます。このパラメータが設定されていない場合は、デフォルトの製品カタログ・リポジトリが使用されます。

  このオプション・パラメータは、多くの場合、ロケールに合わせて代替製品カタログを使用する必要のあるローカライゼーションで役立ちます。

ProductListContains は、出力パラメータを設定しません。ただし、次のいずれかのオープン・パラメータをレンダリングします。

- true
  製品比較リストに特定の製品、カテゴリおよび SKU が含まれているとレンダリングされます。
- false
  製品比較リストに特定の製品、カテゴリおよび SKU が含まれていないとレンダリングされます。

ProductListContains サーブレット Bean の JSP の例については、この章の製品比較ページの例を参照してください。

製品比較リストの管理

ProductListHandler フォーム・ハンドラは製品比較リストを管理します。

Core Commerce には、Nucleus の/atg/commerce/catalog/comparison/にある ProductListHandler の要求限定のインスタンスがデフォルトで含まれています。ProductListHandler は、/atg/commerce/catalog/comparison/にある ProductList コンポーネントによって管理される製品比較リストを操作するように構成されています。つまり、ProductList コンポーネント（atg.commerce.catalog.comparison.ProductComparisonList クラス）は、ProductListHandler.productList で指定されます。

アプリケーションが ProductComparisonList の複数のインスタンスを使用して複数の製品比較リスト（たとえばカメラを比較するリスト、テレビを比較するリストなど）を管理している場合は、個々のリストのコンテンツを管理するために ProductListHandler の複数のインスタンスを構成することをお薦めします。

アプリケーションが様々なロケール用の代替製品カタログを使用している場合は、製品比較リストを操作するときに使用する製品カタログを指定できます。それには、ProductListHandler.repositoryKey プロパティを CatalogTools に渡すキーに設定します。CatalogTools は、渡されたキーと、キーからカタログへのマッピングを使用して、製品カタログ・リポジトリを選択します。一般的に、フォーム内の非表示の入力フィールド経由で ProductListHandler.repositoryKey プロパティを設定します。repositoryKey プロパティが設定されていない場合は、デフォルトの製品カタログ・リポジトリが使用されます。

次の表では、製品比較リストの管理を目的とする ProductListHandler のハンドル・メソッドを説明します。

ハンドル・メソッド	説明
handleAddProduct	productID によって指定されている製品を製品比較リストに追加します。categoryID および skuID でオプションのカテゴリ情報と SKU 情報が指定されている場合は、それらの情報を適用します。
handleAddProductAllSkus	productID によって指定されている製品のすべての SKU を製品比較リストに追加します。categoryID でオプションのカテゴリ情報が指定されている場合は、その情報を適用します。
handleAddProductList	productIDList によって指定されているすべての製品を製品比較リストに追加します。categoryID でオプションのカテゴリ情報が指定されている場合はその情報を適用し、個々の製品のデフォルト SKU があればそれを適用します。
handleAddProductListAllSkus	productIDList によって指定されているすべての製品のすべての SKU を製品比較リストに追加します。categoryID でオプションのカテゴリ情報が指定されている場合は、その情報を適用します。
handleCancel	productID、categoryID および skuID を null に設定することによって、フォーム・ハンドラをリセットします。
handleClearList	成功時に製品比較リストを消去し、ユーザーを clearListSuccessURL にリダイレクトします。
handleRefreshInventoryData	製品比較リスト内の在庫情報を更新します。 ProductListHandler は refreshInventoryDataSuccessURL と refreshInventoryDataErrorURL という 2 つのオプション・プロパティを持っています。ハンドル・メソッドの成功時または失敗時にユーザーのリダイレクト先となるように、それぞれのプロパティを設定できます。
handleRemoveCategory	categoryID で指定されているカテゴリのすべてのエントリを製品比較リストから削除します。
handleRemoveEntries	entryIds で ID が指定されているリスト・エントリを製品比較リストから削除します。
handleRemoveProduct	productID によって指定されている製品を製品比較リストから削除します。categoryID および skuID でオプションのカテゴリ情報と SKU 情報が指定されている場合は、それらの情報を適用します。
handleRemoveProductAllSkus	productID で指定されている製品のすべてのエントリを製品比較リストから削除します。

ハンドル・メソッド	説明
handleSetProductList	productIDList によって指定されている製品に製品比較リストを設定します。categoryID でオプションのカテゴリ情報が指定されている場合はその情報を適用し、個々の製品のデフォルト SKU があればそれを適用します。
handleSetProductListAllSkus	productIDList によって指定されているすべての製品のすべての SKU が含まれるように製品比較リストを設定します。categoryID でオプションのカテゴリ情報が指定されている場合は、その情報を適用します。

ProductListHander のすべてのハンドル・メソッドがオプションのカテゴリ情報と SKU 情報を同じ方法で管理することに注意してください。categoryID で製品のカテゴリ情報が指定されていない場合、フォーム・ハンドラは製品のデフォルト・カテゴリを検索します。デフォルト値が存在しない場合は、プロパティが null に設定されます。同様に、skuID で製品の SKU 情報が指定されていない場合、フォーム・ハンドラは製品の最初の子 SKU（つまり、デフォルト SKU）を検索します。デフォルト値が存在しない場合は、プロパティが null に設定されます。

ProductListHandler のメソッドの詳細は、『ATG Platform API Reference』を参照してください。ProductListHandler の JSP の例については、次の製品比較ページの例を参照してください。

製品比較ページの例

この項では、次の操作を説明する JSP の例を示します。

- 製品比較表の表示
- 製品比較リストでの製品の追加または削除
- 製品比較リストへの複数の製品の追加
- 製品比較リストからの特定のエントリの削除

製品比較表の表示

この JSP の例では、ProductList.items 内の製品と ProductList.tableInfo（ProductList の参照 TableInfo オブジェクト）内の表情報を使用して、単純な製品比較表を表示する方法を示します。

ProductComparisonList（そのインスタンスが ProductList コンポーネント）は、参照 TableInfo オブジェクトを呼び出す sortProperties や tableColumns のようなコンビニエンス・メソッドを備えていることに注意してください。したがって、式 ProductList.tableColumns と式 ProductList.tableInfo.tableColumns は等価です。

注意: TableInfo コンポーネントの用途と動作の詳細は、『ATG Web Commerce Page Developer's Guide』の*ソート可能な表の実装*を参照してください。TableInfo の追加情報については、このマニュアルを参照してください。

```
<dsp:importbean bean="/atg/dynamo/droplet/ForEach"/>
<dsp:importbean bean="/atg/dynamo/droplet/BeanProperty"/>
```

```
<dsp:importbean bean="/atg/commerce/catalog/comparison/ProductList"/>

<dsp:droplet name="ForEach">
  <dsp:param bean="ProductList.items" name="array"/>
  <dsp:param value="+product.displayName" name="sortProperties"/>

  <!-- Display table headings using TableInfo class -->
  <dsp:oparam name="outputStart">
    <table border="1" cellpadding="5" cellspacing="1">
    <dsp:droplet name="ForEach">
      <dsp:param bean="ProductList.tableColumns" name="array"/>
      <dsp:param value="" name="sortProperties"/>
      <dsp:oparam name="output">
        <dsp:valueof param="element.heading"/>
      </dsp:oparam>
    </dsp:droplet>
  </dsp:oparam>

  <!-- Display one table row for each item -->
  <dsp:oparam name="output">
    <dsp:setvalue paramvalue="element" param="currentProduct"/>
    <tr>
    <dsp:droplet name="ForEach">
      <dsp:param bean="ProductList.tableColumns" name="array"/>
      <dsp:param value="" name="sortProperties"/>
      <dsp:oparam name="output">
        <td>

        <dsp:droplet name="BeanProperty">
          <dsp:param param="currentProduct" name="bean"/>
          <dsp:param param="element.property" name="propertyName"/>
          <dsp:oparam name="output">
            <dsp:valueof valueishtml="<%=true%>" param="propertyValue"/>
          </dsp:oparam>
          </dsp:droplet>

          </td>
        </dsp:oparam>
     </dsp:droplet>
     </tr>
  </dsp:oparam>

  <!-- Close the table -->
  <dsp:oparam name="outputEnd">
    </table>
  </dsp:oparam>
</dsp:droplet>
```

製品比較リストでの製品の追加または削除

この JSP フラグメントは、製品比較リストで 1 つの製品を追加または削除する方法を示しています。この例では、ProductListContains サーブレット Bean が、次のフラグメントを使用して製品表示ページに埋め込まれていることを前提としています。

```
<dsp:include page="example.jsp"><dsp:param name="product"
     value="current product"/></dsp:include>
```

current product は、ページに表示される製品へのアクセスを提供する式です。

指定された製品は productId 入力パラメータを介してサーブレット Bean に渡されます。次に、ProductListContains サーブレット Bean は、その製品が ProductList 内の製品比較リストに格納されているかどうかを確認します。製品が製品比較リスト内にある場合、このサーブレット Bean は製品表示ページ上で true オープン・パラメータをレンダリングし、ユーザーは比較リストからの削除発行ボタンをクリックして、製品をリストから削除できます。製品が製品比較リスト内にない場合、このサーブレット Bean は製品表示ページ上で false オープン・パラメータをレンダリングし、ユーザーは比較リストへの追加発行ボタンをクリックして、製品をリストに追加できます。

```
<dsp:importbean bean="/atg/commerce/catalog/comparison/ProductList"/>
<dsp:importbean bean="/atg/commerce/catalog/comparison/ProductListContains"/>
<dsp:importbean bean="/atg/commerce/catalog/comparison/ProductListHandler"/>

<dsp:form action="product.jsp" method="POST">
<dsp:droplet name="ProductListContains">
  <dsp:param bean="ProductList" name="productList"/>
  <dsp:param param="product.repositoryId" name="productID"/>

  <dsp:oparam name="true">
    <dsp:input bean="ProductListHandler.productID" paramvalue="productID"
         type="hidden"/>
    <dsp:input bean="ProductListHandler.removeProduct" value="Remove from
         comparison list" type="submit"/>
  </dsp:oparam>

  <dsp:oparam name="false">
    <dsp:input bean="ProductListHandler.productID" paramvalue="productID"
         type="hidden"/>
    <dsp:input bean="ProductListHandler.addProduct" value="Add to
      comparison list" type="submit"/>
  </dsp:oparam>

</dsp:droplet>
</dsp:form>
```

製品比較リストへの複数の製品の追加

この JSP の例では、検索結果リストからフォームを作成し、ユーザーがそのフォームを使用して複数の製品を選択し、それらを製品比較リストに追加できるようにする方法を示します。

ユーザーがフォームで選択した個々の製品は、productIdList に追加されます。productIdList には、ProductListHandler の handleAddProductList メソッドまたは handleAddProductListAllSkus メソッドのどちらかを呼び出すときに比較リストに追加する製品のリポジトリ ID のリストが格納されます。この例では、handleAddProductList メソッドが呼び出されます（ProductListHandler のハンドル・メソッドの詳細は、製品比較リストの管理を参照してください）。

```
<dsp:importbean bean="/atg/commerce/catalog/comparison/ProductListHandler"/>
<dsp:importbean bean="/atg/commerce/catalog/SearchFormHandler"/>

<dsp:form action="compare.jsp" method="POST">
<dsp:droplet name="ForEach">
  <dsp:param bean="SearchFormHandler.searchResults" name="array"/>
  <dsp:param value="+displayName" name="sortProperties"/>

  <dsp:oparam name="outputStart">
    <table border=0 cellpadding=0 cellspacing=0>
  </dsp:oparam>

  <dsp:oparam name="output">
    <tr>
    <td>
      <dsp:input bean="ProductListHandler.productIdList"
           paramvalue="element.repositoryId" type="checkbox"/>
      <dsp:valueof param="element.displayName"/> - <dsp:valueof
           param="element.description"/>
    </td>
    </tr>
  </dsp:oparam>

  <dsp:oparam name="outputEnd">
    </table></br>
    <dsp:input bean="ProductListHandler.addProductList" value="Add to list"
         type="submit"/>
  </dsp:oparam>

</dsp:droplet>
</dsp:form>
```

製品比較リストからの特定のエントリの削除

この JSP の例では、ユーザーが製品比較リストから削除する複数のエントリを選択するために、または製品比較リストからすべてのエントリを削除するために使用できるフォームを作成する方法を示します。ProductListHandler は、id プロパティに格納されているエントリの一意の ID によって個々のエントリを識別します。

ユーザーがフォームで選択した個々の製品は、entryIds に追加されます。entryIds には、ProductListHandler の handleRemoveEntries メソッドを呼び出すときに製品比較リストから削除する製品のエントリ ID のリストが格納されます。

ユーザーは、ProductListHandler の handleClearAll メソッドを呼び出すすべて削除発行ボタンをクリックすることによって、製品比較リストからすべてのエントリを削除できます（ProductListHandler のハンドル・メソッドの詳細は、製品比較リストの管理を参照してください）。

```
<dsp:importbean bean="/atg/commerce/catalog/comparison/ProductList"/>
<dsp:importbean bean="/atg/commerce/catalog/comparison/ProductListHandler"/>

<dsp:form action="delete.jsp" method="POST">
<dsp:droplet name="ForEach">
  <dsp:param bean="ProductList.items" name="array"/>
  <dsp:param value="+product.displayName" name="sortProperties"/>

  <dsp:oparam name="empty">
    There are no items in your comparison list.
  </dsp:oparam>

  <dsp:oparam name="outputStart">
    <b>Remove items from comparison list</b>
    <blockquote>
  </dsp:oparam>

  <dsp:oparam name="output">
    <dsp:input bean="ProductListHandler.entryIds" paramvalue="element.id"
               type="checkbox"/>
    <dsp:valueof valueishtml="<%=true%>"
         param="element.product.displayName"/></br>
  </dsp:oparam>

  <dsp:oparam name="outputEnd">
    </blockquote>
    <br>
    <dsp:input bean="ProductListHandler.clearListSuccessURL" value="compare.jsp"
               type="hidden"/>
    <dsp:input bean="ProductListHandler.clearList" value="Remove
      all" type="submit"/>
    <dsp:input bean="ProductListHandler.removeProductSuccessURL"
              value="compare.jsp" type="hidden"/>
    <dsp:input bean="ProductListHandler.removeEntries" value="Remove selected
        items" type="submit"/>
  </dsp:oparam>

</dsp:droplet>
</dsp:form>
```

複数サイト環境での製品比較リストの使用

Oracle Commerce Platform の複数サイト機能を使用している場合は、複数のサイトにまたがって製品を比較する機能をユーザーに提供したいことがあります。この機能を使用するために追加の構成を行う必要はありません。`ProductComparisonList` はデフォルトで共有可能コンポーネントとして登録され、複数サイト環境でも単一サイトと同じように機能します。

注意: 製品比較リストでは、ユーザーが別々のサイトから同じ製品を 1 つのリストに追加することを禁止していません。

`ProductComparisonList` を参照する共有可能な Nucleus コンポーネントは、`/atg/commerce/ShoppingCartShareableType` にあります。`ProductComparisonList` は、デフォルトで、共有可能コンポーネントとして登録されます。

```
id=atg.ShoppingCart
paths=/atg/commerce/ShoppingCart,\
    /atg/commerce/catalog/comparison/ProductList
```

共有可能コンポーネントおよび複数サイト構成で共有グループを使用する方法については、『ATG Web Commerce Multisite Administration Guide』を参照してください。

11 ショッピング・カートの実装

Oracle Commerce Core Commerce のショッピング・カートは、INCOMPLETE 状態にある `Order` です。ショッピング・カートには、特定の顧客が注文したいと考えている品目およびその品目に関連付けられた数量と価格に関する情報が格納されます。さらに、オーダーの出荷情報と支払情報もショッピング・カートに格納されます。

この章では、ページ開発者を対象として、コマース・サイトでショッピング・カートを管理する方法を説明します。この章には、次の項があります。

ShoppingCart コンポーネントの理解
顧客の現在のショッピング・カートおよび保存されているショッピング・カートを格納する `ShoppingCart` コンポーネントに関する情報が記載されています。

ショッピング・カートの管理
ショッピング・カートの取得、作成、変更および保存に関する情報が記載されています。

この章で提示しているコード例の大半は、Commerce サンプル・カタログ内の JSP から引用されています。サンプル・カタログおよびサンプル・カタログを実行する方法の詳細は、サンプル・カタログについてを参照してください。

ショッピング・カートを管理する方法を示す追加の例については、Motorprise ストア・リファレンス・アプリケーションを参照できます。『ATG Web Commerce ビジネス・コマース・リファレンス・アプリケーション・ガイド』を参照してください。

Oracle Commerce Platform の複数サイト機能を使用している場合は、ショッピング・カートが（顧客が最初の品目を追加したときに）ショッピング・カートが作成されたサイト、個々の品目が追加されたサイトおよび最新のアクティビティが発生したサイトを追跡することに注意してください。複数サイトのカート構成は Site Administration を介して実行されます。詳細は、『ATG Web Commerce Multisite Administration Guide』を参照してください。

注意: ショッピング・カートは定義上 INCOMPLETE 状態にある `Order` であるため、この章では、「ショッピング・カート」と「オーダー」という用語が同じ意味で使用されます。

ShoppingCart コンポーネントの理解

`ShoppingCart` コンポーネントは、顧客のショッピング・カートを格納し、管理する役割を担います。このコンポーネントは、購買プロセス中に使用される顧客の現在のショッピング・カートを保守し、その顧客用にデータベースに保持されている他のショッピング・カートがあれば、それを格納します。これらのショッピング・カートは、Core Commerce オブジェクト・モデルの `atg.commerce.order.Order` オブジェクトに相当し、オーダー・リポジトリ内のオーダー品目に相当します。

`/atg/commerce/ShoppingCart` コンポーネントは、デフォルトで、`atg.commerce.order.OrderHolder` のセッション限定インスタンスになります。次の表で、このコンポーネントの重要なプロパティを説明します。

プロパティ名	プロパティのタイプ	説明
current	Order	現在の Order オブジェクト。null の場合は、新規の Order が自動的に作成されます。
currentEmpty	boolean	true であれば、現在の Order が null であるか、現在のオーダーに CommerceItems が含まれていません。
currentExists	boolean	true であれば、現在の Order が存在します。
currentTransient	boolean	true であれば、現在の Order が null であるか、一時オーダーです。
empty	boolean	true であれば、現在のオーダーおよび保存されているオーダーのコレクションの両方が空です。
failoverRecoveryPricing Operation	String	フェイルオーバーのときに実行する操作。デフォルト設定は ORDER_TOTAL です。
handlerOrderId	String	Order を識別します。
last	Order	最後に完了した Order。Order が精算のために発行されると、ShoppingCart.current 内の Order が ShoppingCart.last に移動し、ShoppingCart.current プロパティが再初期化されます。
orderType	String	新規 Order を作成するときに作成されるオーダーのタイプ。 この機能は、デフォルトで /atg/commerce/order/OrderTools. defaultOrderType に設定されています。
persistOrders	boolean	true であれば、Order が保持されています。
persistOrdersForAnonymousUsers	boolean	true であれば、関連プロファイルが一時プロファイルである場合に Order が保持されます。
persistEmptyOrders	boolean	true であれば、現在のオーダーに CommerceItems が含まれていないとしてもオーダーが保持されます。
repriceAfterFailover Recovery	boolean	true であれば、フェイルオーバー・リカバリ後、Order が再価格設定されます。
restorableOrders	RestorableOrders	セッション・バックアップを介して復元できるオーダーのセット。

プロパティ名	プロパティのタイプ	説明
saved	Collection	ユーザーの保存されているショッピング・カートの Collection。 アプリケーションが ShoppingCart.saved から ShoppingCart.current へカートを移動するときは、atg.commerce.order.OrderHolder で提供されているハンドル・メソッドを使用する必要があります。詳細は、『ATG Web Commerce Platform API Reference』を参照してください。
savedEmpty	boolean	true であれば、保存されているショッピング・カートの Collection が null または空です。

空のオーダーの保持の防止

空のオーダーとは、CommerceItems が含まれない Order（空のショッピング・カート）のことです。ShoppingCart の現在の Order が最初にフェッチされたときに、システムでは既存のオーダーの検出を試みます。1 つも見つからないと、空の一時オーダーが新しく作成されます。ShoppingCart コンポーネントの persistOrders プロパティが true に設定されている場合、そのオーダーは保持されます。たとえばサイトで複数の匿名の訪問者を処理しているなど、場合によっては、すべての空のオーダーをデータベースに保持するのは望ましくない場合があります。

空のオーダーが保持されないようにするには、ShoppingCart コンポーネントの persistEmptyOrders プロパティを false に設定します。匿名の訪問者についてのみ空のオーダーを保持しないように設定すること（一般的な要件）や、ログイン済の訪問者についてのみ保持しないように設定すること（あまり一般的ではありません）が可能です。それには、persistOrdersForAnonymousUsers プロパティおよび persistEmptyOrders プロパティを必要に応じて設定します。

保持設定	効果
persistOrders=true persistOrdersForAnonymousUsers=true persistEmptyOrders=true	すべての訪問者について、空のオーダーも空でないオーダーもすべて保持されます。
persistOrders=true persistOrdersForAnonymousUsers=false persistEmptyOrders=false	匿名の訪問者のオーダーは保持されません。 ログイン済の訪問者について、少なくとも 1 つの品目を含むオーダーのみが保持されます。
persistOrders=true persistOrdersForAnonymousUsers=true persistEmptyOrders=false	すべての訪問者について、少なくとも 1 つの品目を含むオーダーのみが保持されます。
persistOrders=true persistOrdersForAnonymousUsers=false persistEmptyOrders=true	匿名の訪問者のオーダーは保持されません。 ログイン済の訪問者について、空のオーダーも空でないオーダーもすべて保持されます。

ショッピング・カートの管理

この項では、次のショッピング・カート関連タスクについて説明します。

- ショッピング・カートの作成と取得
- ショッピング・カートへの品目の追加
- ショッピング・カートへの出荷情報の追加
- ショッピング・カートへの支払情報の追加
- ショッピング・カートの再価格設定
- ショッピング・カートの保存

ショッピング・カートの作成と取得

この章で前述したように、ShoppingCart コンポーネントは、ユーザーの現在のショッピング・カートおよび保存されているショッピング・カートを格納します。ShoppingCart コンポーネントのプロパティとハンドル・メソッドを使用して、新規のショッピング・カートを作成したり、ユーザーの保存されているショッピング・カートの 1 つを取得し、それをユーザーの現在のショッピング・カートに変換したりできます。

次の JSP の例は、ショッピング・カートを作成し、取得する方法を示しています。この例では、ShoppingCart.savedEmpty プロパティを確認して、現在のユーザーが保存されているショッピング・カートを持っているかどうかを判断します。ユーザーが保存されているショッピング・カートを持っていない場合は、ショッピング・カートを作成するという選択肢がユーザーに与えられます。ユーザーが保存されているショッピング・カートを持っている場合は、保存されているショッピング・カートのいずれかを選択して、それを削除するか現在のショッピング・カートに変換する、保存されているすべてのショッピング・カートを削除する、または新規ショッピング・カートを作成するという選択肢がユーザーに与えられます。

```
<dsp:importbean bean="/atg/commerce/ShoppingCart"/>

<dsp:form action="shoppingcart.jsp" method="post">
 <dsp:droplet name="/atg/dynamo/droplet/Switch">
   <dsp:param bean="ShoppingCart.savedEmpty" name="value"/>

   <dsp:oparam name="true">
     <!-- since there are no saved carts, we cannot switch to another so
          we only give them the option to create a new cart -->
     <dsp:input bean="ShoppingCart.create" value="Create" type="submit"/>
another shopping cart
   </dsp:oparam>

   <dsp:oparam name="false">
     <!-- We have other shopping carts, so let them do everything -->
     Shopping Cart <dsp:select bean="ShoppingCart.handlerOrderId">
     <dsp:droplet name="ForEach">
       <dsp:param bean="ShoppingCart.saved" name="array"/>
       <dsp:param value="savedcart" name="elementName"/>
       <dsp:oparam name="output">
         <dsp:getvalueof id="option26" param="savedcart.id"
idtype="java.lang.String">
```

```
<dsp:option value="<%=option26%>"/>
</dsp:getvalueof><dsp:valueof param="savedcart.id"/>
       </dsp:oparam>
     </dsp:droplet>
     </dsp:select>:
  <dsp:input bean="ShoppingCart.switch" value="Switch" type="submit"/> to,
  <dsp:input bean="ShoppingCart.delete" value="Delete" type="submit"/> or
  <dsp:input bean="ShoppingCart.create" value="Create" type="submit"/>
another shopping cart.<BR>
     <dsp:input bean="ShoppingCart.deleteAll"
value="Delete All Shopping Carts" type="submit"/>
   </dsp:oparam>

 </dsp:droplet>
</dsp:form>
```

オーダーの取得の実装

OrderLookup サーブレット Bean を使用して、ユーザーの不完全なオーダー（つまりショッピング・カート）を取得できます。OrderLookup を使用すれば、1 つのオーダー、特定のコスト・センターに割り当てられているすべてのオーダー、特定のユーザーが確定したすべてのオーダーまたは特定のユーザーが確定し、INCOMPLETE など、特定の状態にあるすべてのオーダーを取得できます。

目的のショッピング・カートを ShoppingCart.current へ移動したら、ForEach サーブレット Bean を使用して、カート内の商品に対する繰り返し処理を実行し、商品を表示できます。次の JSP のコード例は、それを実行する方法を示しています。この例では、ユーザーがカートから任意の品目を削除できるように、個々の品目の横にチェック・ボックスがレンダリングされます。同様に、ユーザーが任意の品目の数量を変更できるように、個々の品目の横にテキスト・ボックスがレンダリングされます。

```
<droplet bean="ForEach">
  <param name="array" value="bean:ShoppingCart.current.commerceItems">
  <param name="elementName" value="item">
  <oparam name="output">
<tr valign=top>
<td>
  <input type="checkbox" unchecked
    bean="CartModifierFormHandler.removalCatalogRefIds"
    value="param:item.catalogRefId">
</td>
<td>
  <input size=4 name="param:item.catalogRefId"
value="param:item.quantity">
</td>
<td>
  <droplet src="product_fragment.jsp">
  <param name="childProduct" value="param:item.auxiliaryData.productRef">
  <%@ include file="product_fragment.jsp"%>
  </droplet>
  </oparam>
</droplet>
```

同様の機能の例は、<ATG11dir>\MotorpriseJSP\j2ee-apps\motorprise\web-app\en\catalog\cart.jsp を参照してください。

ショッピング・カートへの品目の追加

この項では、CartModifierFormHandler を使用して現在のショッピング・カートに品目を追加する方法を説明します。この章には、次の項があります。

- 一度に 1 つずつ品目を追加する
- 一度に複数の品目を追加する
- デフォルトの商品タイプの上書き
- カスタム商品プロパティの処理

一度に 1 つずつ品目を追加する

現在のショッピング・カートに品目を追加する最も簡単な方法は、一度に 1 つずつ品目を追加することです。次の JSP のコード例は、それを実行する方法を示しています。

この例では、カートに追加する現在の製品の SKU をユーザーがドロップダウン・リストから選択できます。ForEach サーブレット Bean を使用して、製品の SKU に対する繰り返し処理を実行し、ドロップダウン・リストに SKU を挿入します。このドロップダウン・リストは CartModifierFormHandler の catalogRefIds プロパティおよび commerceId プロパティに関連付けられています。さらに、ユーザーは、テキスト・ボックスで、カートに追加される選択されている SKU の数量を指定できます。この数量は、CartModifierFormHandler の quantity プロパティに関連付けられています。

ユーザーがカートに追加発行ボタンをクリックすると、フォームが処理され、CartModifierFormHandler のプロパティが設定され、CartModifierFormHandler の handleAddItemToOrder メソッドが呼び出されます。handleAddItemToOrder メソッドは、選択されている SKU（CartModifierFormHandler.catalogRefIds で指定されており、リポジトリ ID によって識別される）の数量（CartModifierFormHandler.quantity で指定されている）を現在の Order に追加し、Order を再価格設定します。

```
<dsp:importbean
bean="/atg/commerce/order/purchase/CartModifierFormHandler"/>
<dsp:importbean bean="/atg/dynamo/droplet/ForEach"/>

<%--Create a form for user to select a SKU and add it to his/her cart:--%>
<dsp:getvalueof id="form10" bean="/OriginatingRequest.requestURI"
idtype="java.lang.String">
<dsp:form action="<%=form10%>" method="post">

<%--Store this product's ID in the Form Handler: --%>
<dsp:input bean="CartModifierFormHandler.ProductId"
paramvalue="Product.repositoryId" type="hidden"/>

<%--set id param so that the Navigator won't get messed up in case of an
error that makes us return to this page.--%>
<input value='<dsp:valueof param="Product.repositoryId"/>' type="hidden"
name="id">
```

```
 <table cellpadding=0  cellspacing=0 border=0>
   <tr><td class=box-top-store>Add to Cart</td></tr>
   <tr><td class=box>

<%--Display any errors that have been generated during Cart
        operations:--%>
       <dsp:include
page="../../common/DisplayCartModifierFormHandlerErrors.jsp"></dsp:include>

       Add

<%--Textbox with QTY the user wants to order: --%>
       <dsp:input bean="CartModifierFormHandler.quantity" size="4"
value="1" type="text"/>

<%--Create a dropdown list with all SKUs in the Product.
        Store the selected SKU's id in the form handler: --%>
       <dsp:select bean="CartModifierFormHandler.catalogRefIds">

<%--For each of the SKUs in this Product, add the SKU to the
        dropdown list:--%>
         <dsp:droplet name="ForEach">
           <dsp:param param="Product.childSKUs" name="array"/>
           <dsp:param value="Sku" name="elementName"/>
           <dsp:param value="skuIndex" name="indexName"/>
           <dsp:oparam name="output">

<%--This is the ID to store if this SKU is selected in
            dropdown:--%>
             <dsp:getvalueof id="option73" param="Sku.repositoryID"
idtype="java.lang.String">
<dsp:option value="<%=option73%>"/>
</dsp:getvalueof>

<%--Display the SKU's display name in the dropdown
            list:--%>
             <dsp:valueof param="Sku.displayName"/>
           </dsp:oparam>
         </dsp:droplet>
<%--ForEach SKU droplet--%>
       </dsp:select>
       <br>


<%-- ADD TO CART BUTTON: Adds this SKU to the Order--%>
       <dsp:input bean="CartModifierFormHandler.addItemToOrder"
value="Add to Cart" type="submit"/>

<%-- Go to this URL if NO errors are found during the ADD TO
```

```
CART button processing:--%>
       <dsp:input bean="CartModifierFormHandler.addItemToOrderSuccessURL"
value="/checkout/cart.jsp" type="hidden"/>
     </td>
   </tr>
 </table>
</dsp:form></dsp:getvalueof>
```

CartModifierFormHandler およびそのハンドル・メソッドの詳細は、『ATG Web Commerce Programming Guide』の「*購買プロセス・サービスの構成*」の*オーダーの変更*に関する項を参照してください。

また、Motorprise リファレンス・アプリケーションの AddToCart.jsp でも、同様の JSP のコード例を参照できます。ユーザーが表示されている SKU の数量をユーザーのショッピング・カートに追加できるように、AddToCart.jsp は product.jsp に埋め込まれます。<ATG11dir>\MotorpriseJSP\j2ee-apps\motorprise\web-app\en\catalog\で AddToCart.jsp と product.jsp の両方にアクセスできます。「ページおよびコンポーネント」→「J2EE ページ」タスク領域を介して ACC の文書エディタでこれらのページを開くこともできます。

一度に複数の品目を追加する

ユーザーが 1 回のフォーム発行で複数の品目を現在のショッピング・カートに追加するためのページを作成できます。品目は複数の異なる製品および SKU を参照し、複数の異なる数量を持つことができます。CartModifierFormHandler には、品目ごとにプロパティ値を設定できる items プロパティが含まれています。

次の JSP のコード例は、1 つの製品の複数の SKU に対応する複数の品目を追加する方法を示しています。この例では、ユーザーが、テキスト・ボックスで、カートに追加される SKU ごとに別々の数量を指定できます。製品 ID および SKU 用の非表示の入力フィールドがあります。個々の製品 ID、SKU および数量のテキスト・ボックスは、CartModifierFormHandler.items 配列内の 1 つの要素のサブプロパティに関連付けられています。

ユーザーがカートに追加発行ボタンをクリックすると、フォームが処理され、CartModifierFormHandler のプロパティが設定され、CartModifierFormHandler の handleAddItemToOrder メソッドが呼び出されます。handleAddItemToOrder メソッドは CartModifierFormHandler.items 要素に対して繰り返し処理を実行し、個々の要素に対応する quantity（数量）の品目を、その要素の productId と新規品目の catalogRefId を使用して追加します。

```
<dsp:importbean
bean="/atg/commerce/order/purchase/CartModifierFormHandler"/>
<dsp:importbean bean="/atg/dynamo/droplet/ForEach"/>
<dsp:form action="display_product.jsp" method="post">
<input name="id" type="hidden" value='<dsp:valueof
param="product.repositoryId"/>'>
<dsp:input bean="CartModifierFormHandler.addItemToOrderSuccessURL"
type="hidden" value="shoppingcart.jsp"/>
<table border=1>
<tr>
<td>SKU</td>
<td>Quantity</td>
</tr>
<dsp:droplet name="ForEach">
```

```
  <dsp:param name="array" param="product.childSKUs"/>
  <dsp:param name="elementName" value="sku"/>
  <dsp:param name="indexName" value="skuIndex"/>
  <dsp:oparam name="outputStart">
    <dsp:input bean="CartModifierFormHandler.addItemCount"
paramvalue="size" type="hidden"/>
  </dsp:oparam>
  <dsp:oparam name="output">
    <tr>
    <td><dsp:valueof param="sku.displayName"/></td>
    <td>
      <dsp:input
bean="CartModifierFormHandler.items[param:skuIndex].quantity" size="4"
type="text" value="0"/>
      <dsp:input
bean="CartModifierFormHandler.items[param:skuIndex].catalogRefId"
paramvalue="sku.repositoryId" type="hidden"/>
      <dsp:input
bean="CartModifierFormHandler.items[param:skuIndex].productId"
paramvalue="product.repositoryId" type="hidden"/>
    </td>
    </tr>
  </dsp:oparam>

</dsp:droplet>
</table>
<BR>
<dsp:input bean="CartModifierFormHandler.addItemToOrder" type="submit"
value="Add To Cart"/>
</dsp:form>
```

items 配列に割り当てる要素の数を CartModifierFormHandler に指示する必要があります。これは、CartModifierFormHandler.addItemCount プロパティを設定することによって実行されます。前述の例では、addItemCount が、非表示の入力フィールドで定義されている SKU の数に設定されます。この手法がこの例で機能するのは、すべての CartModifierFormHandler.items 入力フィールドが明示的な value 属性または paramvalue 属性を持っているからです。

次のコード・フラグメントは、CartModifierFormHandler.addItemCount を設定するための、より複雑な手法を示しています。フォーム発行エラーが原因でページが再表示されたときにユーザーの入力を保持したい場合は、この手法が適切です。ページが再表示された場合、dsp:setvalue タグは実行されません。

```
<dsp:droplet name="/atg/dynamo/droplet/Switch">
  <dsp:param name="value" bean="CartModifierFormHandler.addItemCount"/>
  <dsp:oparam name="0">
    <dsp:setvalue bean="CartModifierFormHandler.addItemCount" value="5"/>
  </dsp:oparam>
</dsp:droplet>
<dsp:input bean="CartModifierFormHandler.addItemCount" value="5"
type="hidden"/>
```

デフォルトの商品タイプの上書き

Core Commerce は、デフォルトで 3 つのタイプの商品をサポートしています。1 つは通常の SKU に対応します。他の 2 つは構成可能な SKU およびそのサブプロパティに対応します(*ATG Commerce のカタログ管理*の章の構成可能な SKU の作成を参照してください)。サイトが追加のカスタム商品タイプをサポートしている可能性もあります(『ATG Web Commerce Programming Guide』の「*外部購買プロセスのカスタマイズ*」の*購買プロセスの拡張*に関する項を参照してください)。

`CartModifierFormHandler` の `commerceItemType` プロパティは、`addItemToOrder` によって作成される商品のタイプを決定します。通常、`commerceItemType` は「デフォルト」に設定されます。`CartModifierFormHandler` 構成ファイルまたはフォーム入力フィールドで別の商品タイプを指定できます。1 回のフォーム発行で複数の品目を追加する場合は、`CartModifierFormHandler.items[n].commerceItemType` を設定する非表示の入力フィールドを含めることによって、個別の品目の `CartModifierFormHandler.commerceItemType` の設定を上書きできます。

`commerceItemType` の値は`/atg/commerce/order/OrderTools.commerceItemTypeClassMap` 内のキーと一致している必要があります。

カスタム商品プロパティの処理

`CartModifierFormHandler.addItemToOrder` メソッドには、いくつかのタイプの商品拡張のサポートが組み込まれています。サイトが基本データ型プロパティで商品を拡張している場合は、フォーム・フィールドを `CartModifierFormHandler.value` ディクショナリに関連付けることによって、それらのプロパティの値を指定できます。

たとえば、サイトがモノグラムをサポートするように商品を拡張していて、カスタム・プロパティに `monogram` および `style` という名前が付いているとします。次の JSP のコード例は、`CartModifierFormHandler` による変更を加えることなく、モノグラムのユーザー入力を処理する方法を示しています。

```
<b>Monogram Options</b><br>
Initials / Text: <dsp:input bean="CartModifierFormHandler.value.monogram"
size="20" type="text"/><br>
Style: <dsp:select bean="CartModifierFormHandler.value.style">
    <dsp:option value="Block"/>Block
    <dsp:option value="Diamond"/>Diamond
    <dsp:option value="Panel"/>Panel
    <dsp:option value="Stagger"/>Stagger
    <dsp:option value="Script"/>Script
 </dsp:select><br>
```

1 回のフォーム発行で複数の品目を追加する場合は、個々の `CartModifierFormHandler.items` 配列要素内の `value` ディクショナリを介して、個々の品目に異なるカスタム・プロパティ値を指定できます。前述の例を引き続き使用して、1 つの品目のモノグラム・テキストを次のように指定できます。

```
Initials / Text: <dsp:input
bean="CartModifierFormHandler.items[0].value.monogram" size="20"
type="text"/><br>
```

1 回のフォーム発行で複数の品目を追加する場合、すべてまたは大半の品目に同じカスタム・プロパティ値を指定するには、共通の値に `CartModifierFormHandler.value` を使用し、特別な場合に

CartModifierFormHandler.items[n].value を使用できます。プロパティ名が共通のディクショナリと個別の品目のディクショナリの両方に表示される場合、addItemToOrder は個別の品目の値を使用します。

quantity や catalogRefId などの標準の商品プロパティには value ディクショナリを使用できないことに注意してください。

ショッピング・カートからの品目の削除

CartModifierFormHandler.deleteItems メソッドは、CommerceItemIds が RemovalCommerceIds 配列内に格納されている品目を削除します。削除要求がある明細品目の数量は、ページの商品の合計数量と比較されます。数量が一致した場合、CommerceItemIds はページから削除するために RemovalCatalogRefIds 配列に保持されます。数量が一致しない場合、システムでは数量を適切に調整したうえで RemovalCommerceIds 配列から CommerceItemId を削除します。

deleteItems メソッドでは removalCommerceIds 配列に ID が格納されたすべての品目を削除する前に、削除する ID の有効な配列を返し、その ID の一部を削除するかそれとも完全に削除するかを検証するために prepareLineItemsForRemoval メソッドを実行します。prepareLineItemsForRemoval メソッドは、ID と削減数量のセットが格納された removalLineItem マップを移入します。元の数量から削減数量が減算され、ショッピング・カート内の数量が新しくなります。deleteItems メソッドで必要な要素が削除されると、一部が削除された品目の数量を調整するよう removalLineItems マップが処理されます。

CartMofidiferFormHandler が機能する仕組みの詳細は、『ATG Web Commerce Programming Guide』を参照してください。

複数行に分割された商品の削除

商品を削除する JSP ページを構成する際、グループ割引のプロモーションなどにおいて、UI で商品が複数行にわたって表示されていないかどうか注意する必要があります。UnitPriceDetailsDroplet など、品目を複数行に分割可能なドロップレットを使用している場合、removalCommerceIds 配列に commerceIds を指定し、その後に入力デリミタ、そして明細品目数量と続けます。CartModifierFormHandler では、品目が複数行にわたって表示されることを示す inputDelimiter プロパティが使用されます。たとえば、次のようになります。

```
<dsp:input iclass="atg_store_textButton atg_store_actionDelete" type="submit"
name="remove_ci_${nameSuffix}${inputDelimiter}${currentItemQuantity}"
bean="CartModifierFormHandler.removeItemFromOrder" value="${deleteButtonTitle}"
submitvalue="${currentItemId}${inputDelimiter}${currentItemQuantity}"/>
```

inputDelimiter プロパティのデフォルト値は#ですが、この変数の値は CartModifierFormHandler.properties ファイルで変更できます。

ショッピング・カートへの出荷情報の追加

ショッピング・カートに出荷情報を追加するには、次のサブプロセスが必要です。

- 現在のオーダーで使用される可能性がある出荷グループのリストの作成。ユーザーは、オーダーを精算するときに、これらの出荷グループから選択できます。潜在的出荷グループの作成を参照してください。
- 現在のオーダーで使用する出荷グループの指定。オーダーへの出荷グループの追加を参照してください。

- 「陸路便」や「翌日便」など、オーダーの出荷グループの出荷方法の選択。出荷方法の選択を参照してください。

注意 1: 次の項で説明する ShippingGroupFormHandler フォーム・ハンドラと AvailableShippingMethods サーブレット Bean が特定の Order を再価格設定することは**ない**ことに注意してください。したがって、どちらからのメカニズムを介して顧客がオーダー価格に影響を及ぼすような変更をオーダーに加えられるようにした場合は、オーダーの価格を顧客に表示する前に、RepriceOrderDroplet サーブレット Bean を使用して、その Order を再価格設定する必要があります。RepriceOrderDroplet の詳細は、この章のショッピング・カートの再価格設定を参照してください。

注意 2: オーダーにギフト品目が含まれている場合、ShippingGroupDroplet および ShippingGroupFormHandler は、それらの品目および品目の出荷情報を他の品目とは異なる方法で処理します。詳細は、『ATG Web Commerce Programming Guide』を参照してください。

潜在的出荷グループの作成

次の 2 つの方法のどちらかを使用して、Order で使用する可能性のある出荷グループのリストを作成できます。

- フォームを介してユーザーから収集された情報を利用する方法
- ユーザーのプロファイルに格納されている情報を利用する方法

フォームを介してユーザーから取得された情報に基づいて出荷グループを作成するには、CreateHardgoodShippingGroupFormHandler と CreateElectronicShippingGroupFormHandler を使用します。これらのフォーム・ハンドラは、それぞれハード製品出荷グループと電子機器出荷グループを作成します。さらに、フォーム・ハンドラの addToContainer プロパティが true に設定されていれば（デフォルト設定）、フォーム・ハンドラは ShippingGroupMapContainer に新規の出荷グループを追加し、それをコンテナ内のデフォルトの出荷グループにします。ShippingGroupMapContainer は、現在のオーダーで使用できる出荷グループを格納します。出荷グループが ShippingGroupMapContainer に追加されれば、ユーザーは現在のオーダーを精算するときにその出荷グループを使用できます。オーダーへの出荷グループの追加を参照してください。

サンプル・カタログの hardgood_sg.jsp から引用した次の JSP コードは、CreateHardgoodShippingGroupFormHandler の使用例を示しています。

```
<dsp:importbean
bean="/atg/commerce/order/purchase/CreateHardgoodShippingGroupFormHandler"
/>
<dsp:importbean bean="/atg/userprofiling/Profile"/>

<hr>Enter new shipping address for HardgoodShippingGroup

<dsp:form action="hardgood_sg.jsp" method="post">

ShippingGroup NickName:<dsp:input
bean="CreateHardgoodShippingGroupFormHandler.hardgoodShippingGroupName"
size="30" type="text" value=""/>

<br>First:<dsp:input
bean="CreateHardgoodShippingGroupFormHandler.HardgoodShippingGroup.Shippin
gAddress.firstName" beanvalue="Profile.firstName" size="30" type="text"/>
```

```
Middle:<dsp:input
bean="CreateHardgoodShippingGroupFormHandler.HardgoodShippingGroup.Shippin
gAddress.middleName" beanvalue="Profile.middleName" size="30"
type="text"/>

Last:<dsp:input
bean="CreateHardgoodShippingGroupFormHandler.HardgoodShippingGroup.Shippin
gAddress.lastName" beanvalue="Profile.lastName" size="30" type="text"/>

<br>Address:<dsp:input
bean="CreateHardgoodShippingGroupFormHandler.HardgoodShippingGroup.Shippin
gAddress.address1" beanvalue="Profile.defaultShippingAddress.address1"
size="30" type="text"/>

Address (line 2):<dsp:input
bean="CreateHardgoodShippingGroupFormHandler.HardgoodShippingGroup.Shippin
gAddress.address2" beanvalue="Profile.defaultShippingAddress.address2"
size="30" type="text"/>

<br>City:<dsp:input
bean="CreateHardgoodShippingGroupFormHandler.HardgoodShippingGroup.Shippin
gAddress.city" beanvalue="Profile.defaultShippingAddress.city" size="30"
type="text" required="<%=true%>"/>

State:<dsp:input
bean="CreateHardgoodShippingGroupFormHandler.HardgoodShippingGroup.Shippin
gAddress.state" maxsize="2"
beanvalue="Profile.defaultShippingAddress.state" size="2" type="text"
required="<%=true%>"/>

Postal Code:<dsp:input
bean="CreateHardgoodShippingGroupFormHandler.HardgoodShippingGroup.Shippin
gAddress.postalCode" beanvalue="Profile.defaultShippingAddress.postalCode"
size="10" type="text" required="<%=true%>"/>

Country:<dsp:input
bean="CreateHardgoodShippingGroupFormHandler.HardgoodShippingGroup.Shippin
gAddress.country" beanvalue="Profile.defaultShippingAddress.country"
size="10" type="text"/>

<br>
<dsp:input
bean="CreateHardgoodShippingGroupFormHandler.newHardgoodShippingGroupSucce
ssURL" type="hidden" value="shipping.jsp?init=false"/>

<dsp:input
bean="CreateHardgoodShippingGroupFormHandler.newHardgoodShippingGroupError
URL" type="hidden" value="shipping.jsp?init=false"/>
```

```
<dsp:input
bean="CreateHardgoodShippingGroupFormHandler.newHardgoodShippingGroup"
priority="<%=(int)-10%>" type="submit"
value="Create HardgoodShippingGroup"/>

</dsp:form>
```

サンプル・カタログの electronic_sg.jsp から引用した次の JSP コードは、CreateElectronicShippingGroupFormHandler の使用例を示しています。

```
<dsp:importbean
bean="/atg/commerce/order/purchase/CreateElectronicShippingGroupFormHandle
r"/>
<dsp:importbean bean="/atg/userprofiling/Profile"/>

<hr>Enter new e-mail address for ElectronicShippingGroup

<dsp:form action="electronic_sg.jsp" method="post">

ShippingGroup NickName:<dsp:input
bean="CreateElectronicShippingGroupFormHandler.electronicShippingGroupNam
e" size="30" type="text"/>
<br>E-mail Address:<dsp:input
bean="CreateElectronicShippingGroupFormHandler.emailAddress"
beanvalue="Profile.email" size="30" type="text"/>

<br>
<dsp:input
bean="CreateElectronicShippingGroupFormHandler.newElectronicShippingGroupS
uccessURL" type="hidden" value="shipping.jsp?init=false"/>

<dsp:input
bean="CreateElectronicShippingGroupFormHandler.newElectronicShippingGroupE
rrorURL" type="hidden" value="shipping.jsp?init=false"/>

<dsp:input
bean="CreateElectronicShippingGroupFormHandler.newElectronicShippingGroup"
priority="<%=(int)-10%>" type="submit"
value="Create ElectronicShippingGroup"/>

</dsp:form>
```

サンプル・カタログでは、hardgood_sg.jsp と electronic_sg.jsp の両方が shipping.jsp に埋め込まれており、それ自体がユーザーの現在のオーダーの出荷情報を管理します（electronic_sg.jsp はコメントアウトされていることに注意してください）。

CreateHardgoodShippingGroupFormHandler および CreateElectronicShippingGroupFormHandler の両方の詳細は、『ATG Web Commerce Programming Guide』の「*購買プロセス・サービスの構成*」の*オーダーの精算*に関する項を参照してください。

フォームを介してユーザーから収集された情報に基づいて出荷グループを作成するのとは対照的に、ShippingGroupDroplet サーブレット Bean を使用して、ユーザーのプロファイルに格納されている情報に基づいて出荷グループを作成することもできます。ShippingGroupDroplet は、現在のユーザーのプロファイルから取得された情報を利用して、ShippingGroups を初期化し、それらを ShippingGroupMapContainer に追加します。ShippingGroupDroplet に渡される入力パラメータによって、作成される ShippingGroups のタイプ（ハード製品、電子機器またはその両方）および ShippingGroupMapContainer が作成される前に消去されるかどうかが決まります。HardgoodShippingGroups が作成されると、ShippingGroupDroplet は、ユーザーのプロファイル内のデフォルトの出荷所在地を使用して、コンテナのデフォルトの HardgoodShippingGroup も作成します。出荷グループが ShippingGroupMapContainer に追加されれば、ユーザーは現在の Order を精算するときに、それらの出荷グループの中から選択できます。オーダーへの出荷グループの追加を参照してください。

さらに、ShippingGroupDroplet は、ユーザーの現在の Order 内の情報を利用して、Order 内の CommerceItem オブジェクトに対応する CommerceItemShippingInfo オブジェクトを初期化します。CommerceItemShippingInfo オブジェクトは、CommerceItem と ShippingGroup の間の関係を表すヘルパー・オブジェクトです。このオブジェクトには、CommerceItem 内の合計数量を複数の出荷グループの間で分割するために使用できるプロパティが含まれています。CommerceItemShippingInfoContainer には、現在の Order に対応する CommerceItemShippingInfo オブジェクトが格納されます。

CommerceItemShippingInfo オブジェクトのセットがオーダー用に初期化されれば、次の操作を実行するために使用できるフォームを顧客に提示できます。

- CommerceItemShippingInfo オブジェクトごとに異なる ShippingGroup を指定する操作。
- CommerceItemShippingInfo オブジェクトの SplitQuantity プロパティ値および SplitShippingGroupName プロパティ値を更新して、ShippingGroupFormHandler.splitShippingInfos メソッドを呼び出すことによって変更を発行する操作。この方法で、CommerceItem オブジェクトを、ShippingGroupDroplet による初期化によって提供される出荷グループの他に、追加の ShippingGroups と関連付けることができます。変更は、追加の CommerceItemShippingInfo オブジェクトに格納されます。

顧客は、ShippingGroup と CommerceItem との関連付けに満足すれば、ボタンをクリックして購買プロセスを続行できます。その背後では、このボタンが ShippingGroupFormHandler.applyShippingGroups ハンドラを呼び出します。このハンドラは CommerceItemShippingInfo ヘルパー・オブジェクト内の情報を収集し、対応する ShippingGroupCommerceItemRelationship オブジェクトをオーダーに追加します。ShippingGroupCommerceItemRelationship は CommerceItem と ShippingGroup との関連付けを作成し、ShippingGroup 内の情報を利用して、出荷される CommerceItem 内の品目の数量を表します。

CommerceItemShippingInfo オブジェクトと ShippingGroupCommerceItemRelationship オブジェクトは密接に関連していますが、それぞれの使用目的は少し異なります。CommerceItemShippingInfo オブジェクトはオーダーの外部にあり、商品と出荷グループとの関係を定義する手段を提供します。これらの変更は適用されるまで実際のオーダーに影響を与えないため、ShippingGroupFormHandler.applyShippingGroups ハンドラが呼び出されるまで、オーダーは安定した状態にとどまることができます。CommerceItemShippingInfo オブジェクト内の情報がオーダーに適用されれば、関係は ShippingGroupCommerceItemRelationship オブジェクトとしてオーダーに格納されます。

ShippingGroupDroplet に渡される入力パラメータによって、ドロップレットが個々の CommerceItem に対応する CommerceItemShippingInfo オブジェクトを作成および初期化する方法、および CommerceItemShippingInfoContainer が作成される前に消去されるかどうかが決まります。CommerceItemShippingInfo オブジェクトの作成を制御する 3 つのパラメータは、initShippingInfos、createOneInfoPerUnit および initBasedOnOrder です。これらのパラメータおよび他の ShippingGroupDroplet 入力パラメータについて、次の表で説明します。

注意: CommerceItemShippingInfo オブジェクトの詳細は、『ATG Web Commerce ビジネス・コマース・リファレンス・アプリケーション・ガイド』の「*オーダーの処理*」の *複数の送り先への発送*に関する項を参照してください。ShippingGroupCommerceItemRelationship オブジェクトの詳細は、『ATG Web Commerce Programming Guide』の「*購入プロセス・オブジェクトの操作*」の *関連オブジェクトの使用*に関する項を参照してください。

ShippingGroupDroplet は次の入力パラメータを取ります。

パラメータ	説明
Clear	True に設定されていると、ShippingGroupDroplet は、ユーザーの CommerceItemShippingInfoContainer および ShippingGroupMapContainer の両方を消去します。
clearShippingGroups	True に設定されていると、ShippingGroupDroplet は、ユーザーの ShippingGroupMapContainer を消去します。
clearShippingInfos	True に設定されていると、ShippingGroupDroplet は、ユーザーの CommerceItemShippingInfoContainer を消去します。 個々の Order 内の一意の CommerceItem オブジェクトを参照する新規の CommerceItemShippingInfo オブジェクトを作成するために、1 つの Order につき少なくとも 1 回この操作を実行する必要があります。
createOneInfoPerUnit	True に設定されていると、ShippingGroupDroplet は、個々の CommerceItem に含まれている個別の単位に対応する CommerceItemShippingInfo オブジェクトを作成します。たとえば、数量が 5 個の CommerceItem に対しては、5 個の CommerceItemShippingInfo オブジェクトが作成されます。ユーザーが自分のプロファイルにデフォルトの ShippingGroup を持っていれば、個々の CommerceItemShippingInfo オブジェクトはその ShippingGroup で初期化されます。デフォルトで False に設定されます。
initBasedOnOrder	True に設定されていると、ShippingGroupDroplet は、Order 内の個々の ShippingGroupCommerceItemRelationship オブジェクトに対応する CommerceItemShippingInfo オブジェクトを作成します。CommerceItemShippingInfo は、ShippingCommerceItemRelationship 内に存在する ShippingGroup で初期化されます。このオプションは、顧客が精算プロセスを実行中で、オーダーにすでに複数の ShippingGroupCommerceItemRelationship オブジェクトが含まれている状況に対応するために用意されています。デフォルトで False に設定されます。
initShippingGroups	True に設定されていると、shippingGroupTypes 入力パラメータで渡された ShippingGroup タイプが初期化されます。

パラメータ	説明
initShippingInfos	True に設定されていると、ShippingGroupDroplet は、Order 内の個々の CommerceItem に対応する CommerceItemShippingInfo オブジェクトを作成します。ユーザーが自分のプロファイルにデフォルトの ShippingGroup を持っていれば、CommerceItemShippingInfo オブジェクトはその ShippingGroup で初期化されます。デフォルトで True に設定されます。
order	ユーザーのオーダーに含まれているコンポーネントのデフォルト設定を上書きするために使用されます。
shippingGroupTypes	初期化する ShippingGroups のタイプを決定するために使用される hardgoodShippingGroup や electronicShippingGroup などの ShippingGroup タイプのカンマ区切りのリスト。

ShippingGroupDroplet は次の出力パラメータを設定します。

パラメータ	説明
shippingGroups	ShippingGroupMapContainer によって参照される Map。
order	ユーザーのオーダーを表す Order オブジェクト。

ShippingGroupDroplet は、output という名前の 1 つのオープン・パラメータをレンダリングします。

次のコード例は、ShippingGroupDroplet の使用例を示しています。この例では、ユーザーのプロファイル内の出荷所在地情報の可用性に基づいて現在のユーザーの HardgoodShippingGroup オブジェクトを作成します。ShippingGroupDroplet は、ユーザーの現在のオーダー内の品目に対応する CommerceItemShippingInfo オブジェクトも作成します。

```
<dsp:droplet name="ShippingGroupDroplet">
  <dsp:param value="true" name="clear"/>
  <dsp:param value="hardgoodShippingGroup" name="shippingGroupTypes"/>
  <dsp:param value="true" name="initShippingGroups"/>
  <dsp:param value="true" name="initShippingInfos"/>
  <dsp:oparam name="output"> Manipulation of objects here...
  </dsp:output>
</dsp:droplet>
```

サンプル・カタログの shipping.jsp で、ShippingGroupDroplet の使用例を示す別の JSP のコード例を参照できます。

オーダーへの出荷グループの追加

次の 2 つのプロセス(詳細は前の項を参照)を通じて Order の出荷情報が収集されたら、ShippingGroupFormHandler を使用して出荷グループを Order に追加します。

- Order で使用される可能性がある出荷グループが CreateHardgoodShippingGroupFormHandler、CreateElectronicShippingGroupFormHandler および ShippingGroupDroplet、あるいはそのいずれかの組合せを使用して作成されます。出荷グループが ShippingGroupMapContainer に追加されます。
- Order 内の個々の CommerceItem に対応する CommerceItemShippingInfo オブジェクトが ShippingGroupDroplet を使用して作成されます。CommerceItemShippingInfo オブジェクトが CommerceItemShippingInfoContainer に追加されます。

例として、サンプル・カタログ内の complex_shipping.jsp から引用した次のコード・セグメントについて考えてみましょう。

注意: 次のコード・セグメントでは、参照される個々のコンポーネントが dsp:importbean タグを介してページにインポートされたと想定できます。これらのインポート文については、実際の JSP を参照してください。

```
<dsp:droplet name="ShippingGroupDroplet">
  <dsp:param name="clearShippingGroups" value="false"/>
  <dsp:param name="initShippingGroups" value="false"/>
  <dsp:param name="initShippingInfos" param="init"/>
  <dsp:oparam name="output">
  <!-- begin output -->

<table border=0 cellpadding=0 cellspacing=0 width=800>

  <tr>
    <td width=55></td>
    <td valign="top" width=745>
  <table border=0 cellpadding=4 width=80%>
    <tr><td></td></tr>
    <tr><td></td></tr>
    <tr valign=top>
      <td>
<%-- table with multiple rows with eleven cells  --%>
        <table border=0 cellpadding=4 cellspacing=1 width=100%>
          <tr>
            <td colspan=12><span class=help>To ship a line item to another
address, select the address and click the "Save" button. To ship only some
of the items to another address, change the quantity and select the
address. You must save changes individually before continuing.
            </span></td>
          </tr>
          <tr bgcolor="#666666" valign=bottom>
            <td colspan=2><span class=smallbw>Part #</span></td>
            <td colspan=2><span class=smallbw>Name</span></td>
            <td colspan=2 align=middle><span class=smallbw>Qty</span></td>
            <td colspan=2 align=middle><span class=smallbw>Qty to
move</span></td>
            <td colspan=2 align=middle><span class=smallbw>Shipping
address</span></td>
            <td colspan=2><span class=smallbw>Save changes</span></td>
```

```
          </tr>

<%-- get the real shopping cart items  --%>
          <dsp:droplet name="ForEach">
            <dsp:param name="array" param="order.commerceItems"/>
            <dsp:oparam name="output">
              <dsp:setvalue paramvalue="element" param="commerceItem"/>
              <dsp:setvalue bean="ShippingGroupFormHandler.listId"
paramvalue="commerceItem.id"/>
              <dsp:droplet name="ForEach">
                <dsp:param bean="ShippingGroupFormHandler.currentList"
name="array"/>
                 <dsp:oparam name="output">
                   <!-- begin line item -->
                   <dsp:setvalue paramvalue="element" param="cisiItem"/>
                   <dsp:form action="complex_shipping.jsp" method="post">
                   <tr valign=top>
                    <td><nobr><dsp:valueof
param="commerceItem.auxiliaryData.catalogRef.manufacturer_part_number"/>
</nobr></td>
                    <td></td>
                    <td><dsp:valueof
param="commerceItem.auxiliaryData.catalogRef.displayName"/></td>
                    <td></td>
                    <td align=right>
<dsp:valueof param="element.quantity"/></td>
                    <td> </td>
                    <td>
                    <dsp:input
bean="ShippingGroupFormHandler.currentList[param:index].splitQuantity"
paramvalue="element.quantity" size="4" type="text"/></td>
                    <td> </td>
                    <td>
                      <dsp:select
bean="ShippingGroupFormHandler.currentList[param:index].splitShippingGroup
Name">
                      <dsp:droplet name="ForEach">
                        <dsp:param name="array" param="shippingGroups"/>
                        <dsp:oparam name="output">
                          <dsp:droplet name="Switch">
                            <dsp:param name="value" param="key"/>
                            <dsp:getvalueof id="nameval4"
param="cisiItem.shippingGroupName" idtype="java.lang.String">
<dsp:oparam name="<%=nameval4%>">
                              <dsp:getvalueof id="option305" param="key"
idtype="java.lang.String">
<dsp:option selected="<%=true%>" value="<%=option305%>"/>
</dsp:getvalueof><dsp:valueof param="key"/>
                            </dsp:oparam>
```

```
</dsp:getvalueof>
                            <dsp:oparam name="default">
                              <dsp:getvalueof id="option313" param="key"
idtype="java.lang.String">
<dsp:option selected="<%=false%>" value="<%=option313%>"/>
</dsp:getvalueof><dsp:valueof param="key"/>
                            </dsp:oparam>
                          </dsp:droplet>
                        </dsp:oparam>
                      </dsp:droplet>
                      </dsp:select>
                    </td>
                    <td></td>
                    <td>
                      <dsp:input
bean="ShippingGroupFormHandler.splitShippingInfosSuccessURL" type="hidden"
value="complex_shipping.jsp?init=false"/>
                      <dsp:input bean="ShippingGroupFormHandler.ListId"
paramvalue="commerceItem.id" priority="<%=(int)9%>" type="hidden"/>
                      <dsp:input
bean="ShippingGroupFormHandler.splitShippingInfos" type="submit"
value=" Save "/>
                     </td>
                    </tr>
                    </dsp:form>
                    <!-- end line item -->
                  </dsp:oparam>
                </dsp:droplet><!-- end inner ForEach -->
              </dsp:oparam>
            </dsp:droplet><!-- end outer ForEach -->

          <tr>
            <td colspan=12>
<%-- table with one row with one cell  --%>
            <table border=0 cellpadding=0 cellspacing=0 width=100%>
              <tr bgcolor="#666666">
                <td></td>
              </tr>
            </table>
            </td>
          </tr>
        </table>
        </td>
      </tr>
      <tr>
        <td>
          <dsp:form action="complex_shipping.jsp" method="post">
          <dsp:input
bean="ShippingGroupFormHandler.applyShippingGroupsSuccessURL"
```

```
type="hidden" value="billing.jsp?init=true"/>
        <dsp:input bean="ShippingGroupFormHandler.applyShippingGroups"
type="submit" value="Continue"/>
        </dsp:form>
      </td>
    </tr>
 </table>
 </td>
</tr>
</table>

  <!-- end output -->
  </dsp:oparam>
</dsp:droplet><!-- end ShippingGroupDroplet -->
```

complex_shipping.jsp の次の部分に注意してください。

1. ページがレンダリングされるときは、ShippingGroupDroplet を使用して、ユーザーの現在のオーダー内の品目に対応する CommerceItemShippingInfo オブジェクトを初期化し、それらのオブジェクトを CommerceItemShippingInfoContainer に追加します。前の項で説明したように、ShippingGroupDroplet は、デフォルトで、個々の CommerceItemShippingInfo オブジェクトを ShippingGroupMapContainer 内のデフォルトの出荷グループに関連付けます。
2. コードの残りの部分では、ユーザーがオーダー内の品目の数量を別々の出荷グループに割り当てるために使用できるインタフェースをレンダリングします。これは、ネストされた ForEach サーブレット Bean を使用して実行されます。
   - 外側の ForEach サーブレット Bean は Order 内の品目の配列を入力パラメータとして受け取ります。外側のサーブレット Bean は自分の output オープン・パラメータを Order 内の 1 つの CommerceItem につき 1 回レンダリングします。oparam では、ShippingGroupFormHandler.listId プロパティが現在の CommerceItem の ID に設定されます。現在の品目の ID が、ShippingGroupFormHandler.currentList プロパティを介して公開されている CommerceItemShippingInfo オブジェクトの List のキーになります(ページの下にある非表示の入力タグも、それに続く要求に応じてこのプロパティを設定します)。
   - 内側の ForEach サーブレット Bean は、外側の ForEach 繰り返し処理内の現在の CommerceItem に対応する CommerceItemShippingInfo オブジェクトの配列を入力パラメータとして受け取ります。内側のサーブレット Bean は、自分の output oparam を配列内の 1 つの CommerceItemShippingInfo オブジェクトにつき 1 回レンダリングします。レンダリングされた出力は、実質的に、次のものを表示するフォームです。関連付けられている CommerceItem の部品番号と名前、現在の CommerceItemShippingInfo オブジェクト内の数量、CommerceItemShippingInfo オブジェクト内の指定された数量に対応する出荷グループを変更するためのドロップダウン・リスト、選択されている出荷グループに割り当てられる CommerceItemShippingInfo 内の数量を指定するためのテキスト・ボックス、およびこれらの新しい関連付けを作成するための保存発行ボタン。

     出荷グループ・ドロップダウン・リストには、ShippingGroupMapContainer 内の出荷グループが挿入されます。これらの出荷グループは、shippingGroups コンビニエンス・パラメータを介して ShippingGroupDroplet によって公開され、3 番目

のネストされた ForEach サーブレット Bean を使用して出荷グループに対する繰り返し処理が実行され、ドロップダウン・リストに出荷グループが挿入されます。

出荷グループ・ドロップダウン・リストが現在の CommerceItemShippingInfo オブジェクトの splitShippingGroupName プロパティに関連付けられていることに注意してください。同様に、数量テキスト・ボックスは、現在の CommerceItemShippingInfo オブジェクトの splitQuantity プロパティに関連付けられています。

最後に、保存発行ボタンが ShippingGroupFormHandler の handleSplitShippingInfos メソッドを呼び出すことに注意してください。handleSplitShippingInfos メソッドは、quantity 内の値と CommerceItemShippingInfo オブジェクトの splitQuantity プロパティを使用して、オブジェクトを 2 つのオブジェクトに分割するかどうかを決定します。分割の必要があれば、これらのプロパティおよび splitShippingGroupName プロパティで指定された出荷グループを使用して、2 番目の CommerceItemShippingInfo オブジェクトを作成します。このメソッドは、次に、新旧両方のオブジェクトのプロパティを適切に設定し、新規オブジェクトを CommerceItemShippingInfoContainer に追加します。フォームが処理されれば、再びページがレンダリングされ、ユーザーの加えた変更がページに反映されます。

3. ページ最下部の続行発行ボタンを使用して、ユーザーは現在の出荷関連付けをオーダーに適用し、請求情報の指定に進むことができます。この発行ボタンは、ユーザーが選択した出荷グループを Order に追加する、ShippingGroupFormHandler の handleApplyShippingGroups メソッドを呼び出します。それを行うために、このメソッドは、CommerceItemShippingInfoContainer 内の CommerceItemShippingInfo オブジェクトに対する繰り返し処理を実行します。コンテナ内の CommerceItemShippingInfo オブジェクトごとに、関連付けられている出荷グループが ShippingGroupMapContainer から取得されて Order に追加され、関連付けられている CommerceItem の適切な数量がその ShippingGroup に追加されます。

注意: ShippingGroupFormHandler のすべてのハンドル・メソッドの詳細は、『ATG Web Commerce Programming Guide』の「*購買プロセス・サービスの構成*」の *複雑なオーダーを精算する準備*に関する項を参照してください。出荷グループへの品目の追加の詳細は、『ATG Web Commerce Programming Guide』の「*購買プロセス・オブジェクトの操作*」の *出荷グループへの品目の割当*に関する項を参照してください。

ShippingGroupFormHandler の使用方法を示す JSP のコード例については、Motorprise リファレンス・アプリケーションの checkout/shipping.jsp および checkout/ship_to_multiple.jsp を参照してください。これらのページには、<ATG11dir>\MotorpriseJSP\j2ee-apps\motorprise\web-app\en\checkout\でアクセスできます。「ページおよびコンポーネント」→「J2EE ページ」タスク領域を介して ACC の文書エディタでこれらのページを開くこともできます。詳細は、『ATG Web Commerce ビジネス・コマース・リファレンス・アプリケーション・ガイド』の「*オーダーの処理*」の *出荷情報*に関する項を参照してください。

出荷方法の選択

atg/commerce/pricing/AvailableShippingMethods サーブレット Bean を使用して、特定の出荷グループで利用可能な出荷方法(「陸路便」や「翌日便」など)のリストをユーザーに提示できます。出荷グループが指定されると、AvailableShippingMethods は ShippingPricingEngine に対して問合せを行い、指定された出荷グループのタイプで利用可能な出荷方法のリストを返します。

AvailableShippingMethods は、出荷される特定の出荷グループである shippingGroup という名前の 1 つの入力パラメータしか必要としません(AvailableShippingMethods の入力パラメータの完全なリストについては、AvailableShippingMethodsDroplet セクションを参照してください)。このメソッドは、

HardgoodShippingGroup 内の shippingMethod の値の設定に使用できる出荷グループを表す文字列のリストである availableShippingMethods という名前の 1 つの出力パラメータを設定します。さらに、このメソッドは、output という名前の 1 つのオープン・パラメータをレンダリングします。

Motorprise リファレンス・アプリケーションの/checkout/shipping_method.jsp から引用した次の JSP コード・セグメントは、AvailableShippingMethods の使用例を示しています。この例では、AvailableShippingMethods サーブレット Bean を使用して、現在の出荷グループの出荷方法を選択するための選択リストに出荷方法を追加します。

```
<tr valign=top>
 <td align=right width=25%><span class=smallb>Shipping method</span></td>
 <td align=left>
  <%-- The AvailableShippingMethods servlet bean permits the user to
     select a shipping method that is applied to the current
     ShippingGroup. --%>
  <dsp:droplet name="AvailableShippingMethods">
   <dsp:param name="shippingGroup" param="sGroup">
   <dsp:param bean="UserPricingModels.shippingPricingModels"
    name="pricingModels">
   <dsp:param bean="Profile" name="profile">

   <dsp:oparam name="output">
     <dsp:select
bean="ShoppingCart.current.ShippingGroups[param:index].shippingMethod">
      <dsp:droplet name="ForEach">
       <dsp:param name="array" param="availableShippingMethods">
       <dsp:param name="elementName" value="method">
       <dsp:oparam name="output">
        <dsp:getvalueof id="methodname" idtype="String" param="method">
        <dsp:option value="<%=methodname%>"><dsp:valueof
param="method"></dsp:getvalueof>
       </dsp:oparam>
      </dsp:droplet>
     </dsp:select>
    </dsp:oparam>

  </dsp:droplet>
 </td>
</tr>
```

<ATG11dir>\MotorpriseJSP\j2ee-apps\motorprise\web-app\en\checkout\で shipping_method.jsp を参照してください。「ページおよびコンポーネント」→「J2EE ページ」タスク領域を介して ACC の文書エディタでページを開くこともできます。shipping_method.jsp の詳細は、『ATG Web Commerce ビジネス・コマース・リファレンス・アプリケーション・ガイド』の「*オーダーの処理*」の*出荷情報*に関する項を参照してください。

ショッピング・カートへの支払情報の追加

ショッピング・カートに支払情報を追加するには、次のサブプロセスが必要です。

- 現在のオーダーで使用される可能性がある支払グループのリストの作成。ユーザーは、オーダーを精算するときに、これらの支払グループから選択できます。潜在的出荷グループの作成を参照してください。
- 現在のオーダーで使用する支払グループの指定。オーダーへの支払グループの追加を参照してください。

潜在的支払グループの作成

次の 2 つの方法のどちらかを使用して、Order で使用する可能性のある支払グループのリストを作成できます。

- フォームを介してユーザーから収集された情報を利用する方法
- ユーザーのプロファイルに格納されている情報を利用する方法

フォームを介してユーザーから取得された情報に基づいて支払グループを作成するには、CreateCreditCardFormHandler と CreateInvoiceRequestFormHandler を使用します。これらのフォーム・ハンドラは、それぞれ CreditCard 支払グループと InvoiceRequest 支払グループを作成します。さらに、フォーム・ハンドラの addToContainer プロパティが true であれば（デフォルト設定）、フォーム・ハンドラは新規の支払グループを PaymentGroupMapContainer に追加し、それをコンテナ内のデフォルトの支払グループにします。PaymentGroupMapContainer には、現在のオーダーで使用できる支払グループが格納されます。支払グループが PaymentGroupMapContainer に追加されれば、ユーザーは現在のオーダーを精算するときにその支払グループを使用できます。オーダーへの支払グループの追加を参照してください。

サンプル・カタログの credit_card.jsp から引用した次の JSP コードは、CreateCreditCardFormHandler の使用例を示しています。

```
<dsp:importbean
bean="/atg/commerce/order/purchase/CreateCreditCardFormHandler"/>
<dsp:importbean bean="/atg/userprofiling/Profile"/>

<hr><p>Enter new CreditCard information

<dsp:form action="credit_card.jsp" method="post">

<br>CreditCard NickName:<dsp:input
bean="CreateCreditCardFormHandler.creditCardName" size="30" type="text"
value=""/>

<br>CreditCardNumber:<dsp:input
bean="CreateCreditCardFormHandler.creditCard.CreditCardNumber"
maxsize="20" size="20" type="text" value="4111111111111111"/>

<br>CreditCardType:
<dsp:select bean="CreateCreditCardFormHandler.creditCard.creditCardType"
required="<%=true%>">
<dsp:option value="Visa"/>Visa
```

```
<dsp:option value="MasterCard"/>Master Card
<dsp:option value="American Express"/>American Express
</dsp:select>

<br>ExpirationMonth: <dsp:select
bean="CreateCreditCardFormHandler.creditCard.ExpirationMonth">
<dsp:option value="1"/>January
<dsp:option value="2"/>February
<dsp:option value="3"/>March
<dsp:option value="4"/>April
<dsp:option value="5"/>May
<dsp:option value="6"/>June
<dsp:option value="7"/>July
<dsp:option value="8"/>August
<dsp:option value="9"/>September
<dsp:option value="10"/>October
<dsp:option value="11"/>November
<dsp:option value="12"/>December
</dsp:select>

<br>expirationYear:Year: <dsp:select
bean="CreateCreditCardFormHandler.creditCard.expirationYear">
<dsp:option value="2002"/>2002
<dsp:option value="2003"/>2003
<dsp:option value="2004"/>2004
<dsp:option value="2005"/>2005
<dsp:option value="2006"/>2006
<dsp:option value="2007"/>2007
<dsp:option value="2008"/>2008
<dsp:option value="2009"/>2009
<dsp:option value="2010"/>2010
</dsp:select>

<br>FirstName:<dsp:input
bean="CreateCreditCardFormHandler.creditCard.billingAddress.firstName"
beanvalue="Profile.firstName" size="30" type="text"/>
<br>MiddleName:<dsp:input
bean="CreateCreditCardFormHandler.creditCard.billingAddress.middleName"
beanvalue="Profile.middleName" size="30" type="text"/>
<br>LastName:<dsp:input
bean="CreateCreditCardFormHandler.creditCard.billingAddress.lastName"
beanvalue="Profile.lastName" size="30" type="text"/>
<br>EmailAddress:<dsp:input
bean="CreateCreditCardFormHandler.creditCard.billingAddress.email"
beanvalue="Profile.email" size="30" type="text"/>
<br>PhoneNumber:<dsp:input
bean="CreateCreditCardFormHandler.creditCard.billingAddress.phoneNumber"
beanvalue="Profile.daytimeTelephoneNumber" size="30" type="text"/>
<br>Address:<dsp:input
```

```
bean="CreateCreditCardFormHandler.creditCard.billingAddress.address1"
beanvalue="Profile.defaultBillingAddress.address1" size="30" type="text"/>
<br>Address (line 2):<dsp:input
bean="CreateCreditCardFormHandler.creditCard.billingAddress.address2"
beanvalue="Profile.defaultBillingAddress.address2" size="30" type="text"/>
<br>City:<dsp:input
bean="CreateCreditCardFormHandler.creditCard.billingAddress.city"
beanvalue="Profile.defaultBillingAddress.city" size="30" type="text"/>
<br>State:<dsp:input
bean="CreateCreditCardFormHandler.creditCard.billingAddress.state"
beanvalue="Profile.defaultBillingAddress.state" size="30" type="text"/>
<br>PostalCode:<dsp:input
bean="CreateCreditCardFormHandler.creditCard.billingAddress.postalCode"
beanvalue="Profile.defaultBillingAddress.postalCode" size="30"
type="text"/>
<br>Country:<dsp:input
bean="CreateCreditCardFormHandler.creditCard.billingAddress.country"
beanvalue="Profile.defaultBillingAddress.country" size="30" type="text"/>
<dsp:input bean="CreateCreditCardFormHandler.copyToProfile" type="hidden"
value="false"/>

<dsp:input bean="CreateCreditCardFormHandler.newCreditCardSuccessURL"
type="hidden" value="billing.jsp?init=false"/>
<dsp:input bean="CreateCreditCardFormHandler.newCreditCardErrorURL"
type="hidden" value="credit_card.jsp"/>
<dsp:input bean="CreateCreditCardFormHandler.newCreditCard" type="submit"
value="Enter Credit Card"/>

</dsp:form>
```

サンプル・カタログでは、credit_card.jsp が billing.jsp に埋め込まれており、それ自体がユーザーの現在のオーダーの支払情報を管理します。

CreateCreditCardFormHandler を使用するときと同じ方法で CreateInvoiceRequestFormHandler を使用します。ただし、CreateInvoiceRequestFormHandler では、デフォルトで、請求書支払グループの poNumber が指定されている必要があります。CreateCreditCardFormHandler および CreateInvoiceRequestFormHandler の両方の詳細は、『ATG Web Commerce Programming Guide』の「*購買プロセス・サービスの構成*」の*オーダーの精算*に関する項を参照してください。

フォームを介してユーザーから収集された情報に基づいて支払グループを作成するのとは対照的に、PaymentGroupDroplet サーブレット Bean を使用して、ユーザーのプロファイルに格納されている情報に基づいて支払グループを作成することもできます。PaymentGroupDroplet は、現在のユーザーのプロファイルから取得された情報を利用して、支払グループを初期化し、それらの支払グループを PaymentGroupMapContainer に追加します。PaymentGroupDroplet に渡される入力パラメータによって、作成される PaymentGroups のタイプ（クレジット・カード、ストア・クレジット、ストア内引取または商品券、あるいはそれらの組合せ）および PaymentGroupMapContainer が作成される前に消去されるかどうかが決まります。支払グループが PaymentGroupMapContainer に追加されれば、ユーザーは現在の Order を精算するときに、それらの支払グループの中から選択できます。オーダーへの支払グループの追加を参照してください。

さらに、PaymentGroupDroplet は、ユーザーの現在の Order 内の情報を利用して、Order に対応する CommerceIdentifierPaymentInfo を初期化します。CommerceIdentifierPaymentInfo オブジェクトは、Order およびそのコンポーネント(商品、出荷、税)と支払情報の間の関係を表すヘルパー・オブジェクトです。このオブジェクトには、特定の品目のコスト、オーダーの送料および税を複数の支払グループの間で分割するために使用できるプロパティが含まれています。
CommerceIdentifierPaymentInfoContainer には、現在の Order で使用するために作成された CommerceIdentifierPaymentInfo オブジェクトが格納されます。CommerceIdentifierPaymentInfo オブジェクトは、さらに次のタイプに分類されます。

- OrderPaymentInfo オブジェクトには、すべての商品、送料、税を含むオーダー全体の支払情報が格納されます。
- ItemPaymentInfo オブジェクトには、個別の品目の支払情報が格納されます。
- ShippingPaymentInfo オブジェクトには、送料の支払情報が格納されます。
- TaxPaymentInfo オブジェクトには、税の支払情報が格納されます。

PaymentGroupDroplet に渡される入力パラメータによって、4 つの CommerceIdentifierPaymentInfo オブジェクト・タイプのうちどれが Order 用に初期化されるかが決まります。一般に、オーダー全体の OrderPaymentInfo オブジェクトを初期化するか、オーダーのすべてのコンポーネントに対応する ItemPaymentInfo オブジェクト、ShippingPaymentInfo オブジェクト、および TaxPaymentInfo オブジェクトの組合せを初期化します。

CommerceIdentifierPaymentInfo オブジェクトのセットがオーダー用に初期化されれば、次の操作を実行するために使用できるフォームを顧客に提示できます。

- CommerceIdentifierPaymentInfo オブジェクトごとに異なる PaymentGroup を指定する操作。
- CommerceIdentifierPaymentInfo オブジェクトの SplitPaymentMethod プロパティ値、SplitAmount プロパティ値および SplitQuantity プロパティ値を更新して、PaymentGroupFormHandler.handleSplitPaymentInfos メソッドを呼び出すことによって変更を発行する操作。この方法で、CommerceIdentifier オブジェクトを、PaymentGroupDroplet による初期化によって提供される支払グループの他に、追加の PaymentGroups に関連付けることができます。変更は、追加の CommerceIdentifierPaymentInfo オブジェクトに格納されます。

次の図は、ユーザーが複数の PaymentGroups の間で支払金額を分割するのに使用できるサンプルのユーザー・インタフェースを示しています。

顧客は、PaymentGroup と CommerceIdentifier との関連付けに満足すれば、ボタンをクリックして購買プロセスを続行できます。その背後では、このボタンが PaymentGroupFormHandler.handleApplyPaymentGroups メソッドを呼び出します。このメソッドは CommerceIdentifierPaymentInfo ヘルパー・オブジェクト内の情報を収集し、対応する PaymentGroup 関係オブジェクトをオーダーに追加します。PaymentGroup 関係オブジェクトは CommerceIdentifier と PaymentGroup との関連付けを作成し、PaymentGroup 内の情報を利用して、支払われる CommerceIdentifier 内のコストの金額を表します。PaymentGroup 関係オブジェクトには複数のタイプがあります。

- PaymentGroupOrderRelationship は Order と PaymentGroup の間の関係を表します。この関係オブジェクトには、税支払情報も格納されます。
- PaymentGroupCommerceItemRelationship は CommerceItem と PaymentGroup の間の関係を表します。
- PaymentGroupShippingGroupRelationship は ShippingGroup と PaymentGroup の間の関係を表します。

CommerceIdentifierPaymentInfo 関係オブジェクトと PaymentGroup 関係オブジェクトは密接に関連していますが、それぞれの使用目的は少し異なります。CommerceIdentifierPaymentInfo オブジェクトはオーダーの外部にあり、Commerce Identifier と支払グループとの関係を定義する手段を提供します。これらの変更は適用されるまで実際のオーダーに影響を与えないため、PaymentGroupFormHandler.handleApplyPaymentGroups メソッドが呼び出されるまで、オーダーは安定した状態にとどまることができます。CommerceIdentifierPaymentInfo オブジェクト内の情報がオーダーに適用されれば、関係は PaymentGroup 関係オブジェクトとしてオーダーに格納されます。

注意: PaymentGroup 関係オブジェクトの詳細は、『ATG Web Commerce Programming Guide』の「*購入プロセス・オブジェクトの操作*」の*関連オブジェクトの使用*に関する項を参照してください。

PaymentGroupDroplet に渡される入力パラメータによって、ドロップレットがオーダーの CommerceIdentifierPaymentInfo オブジェクトを作成および初期化する方法、および CommerceIdentifierPaymentInfoContainer が作成される前に消去されるかどうかが決まります。CommerceIdentifierPaymentInfo オブジェクトの作成を制御するパラメータは、initOrderPayment、initItemPayment、initShippingPayment、initTaxPayment および initBasedOnOrder です。これらのパラメータおよび他の PaymentGroupDroplet 入力パラメータについて、次の表で説明します。

PaymentGroupDroplet を使用するときは、次のことを考慮する必要があります。

- ユーザーが 1 つ以上の支払グループを使用して Order の合計コストを支払おうとしているのか、**または** 1 つ以上の支払グループを使用して Order のコンポーネントのコスト（商品コスト、送料、税）を支払おうとしているのか。
- ユーザーが Order の支払に使用できる支払グループのタイプ。Core Commerce がデフォルトでサポートする PaymentGroups のタイプは、giftCertificate、storeCredit、creditCard, invoiceRequest、InStore および PaymentGroups です。

これらの要因によって、オーダーでの潜在的使用を目的として、PaymentGroupDroplet を使用して初期化する PaymentGroup オブジェクトの種類が決まります。

PaymentGroupDroplet は次の入力パラメータを取ります。

パラメータ	説明
clear	ブール型。True に設定されていると、PaymentGroupDroplet は、ユーザーの CommerceIdentifierPaymentInfoContainer および PaymentGroupMapContainer の両方を消去します。
cleaPaymentGroups	ブール型。True に設定されていると、PaymentGroupDroplet は、ユーザーの PaymentGroupMapContainer を消去します。
clearPaymentInfos	ブール型。True に設定されていると、PaymentGroupDroplet は、ユーザーの CommerceIdentifierPaymentInfoContainer を消去します。1 つの新規オーダーにつき少なくとも 1 回この操作を実行する必要があります。デフォルト値は false です。
createAllPaymentInfos	ブール型。True に設定されていると、PaymentGroupDroplet は、ユーザーのプロファイルに格納されているすべての可能な支払タイプ（クレジット・カード、ストア・クレジット、商品券または請求書要求）に対応する OrderPaymentInfo を作成します。当初はオーダー全体のコストがデフォルトの PaymentGroup に関連付けられた OrderPaymentInfo に割り当てられ、その他すべての OrderPaymentInfo オブジェクトが 0 に設定されます。このオプションは、前の項で説明しているユーザー・インタフェースとは異なるタイプのユーザー・インタフェースをサポートしています。ユーザーが自分のプロファイル内の個々の支払タイプによって支払われるオーダーの金額を手動で指定するためのユーザー・インタフェースをサポートするのに必要なオブジェクトを createAllPaymentInfos を使用して初期化します。詳細は、createAllPaymentInfos の使用を参照してください。 このオプションはデフォルトで False に設定されます。

パラメータ	説明
initBasedOnOrder	ブール型。True に設定されていると、ShippingGroupDroplet は、Order 内の個々の PaymentGroup 関係オブジェクトに対応する CommerceIdentifierPaymentInfo オブジェクトを作成します。このオプションは、顧客が精算プロセスを実行中で、オーダーにすでに複数の PaymentGroup 関係オブジェクトが含まれている状況に対応するために用意されています。 作成される CommerceIdentifierPaymentInfo オブジェクトのタイプは、PaymentGroup 関係タイプに対応しています。たとえば、Order 内に PaymentGroupCommerceItemRelationship が存在すれば、PaymentGroupDroplet は対応する CommerceItemPaymentInfo オブジェクトを作成し、それを CommerceIdentifierPaymentInfoContainer に追加します。個々の CommerceIdentifierPaymentInfo オブジェクトは、対応する PaymentGroup 関係オブジェクト内に存在する PaymentGroup で初期化されます。 デフォルトで False に設定されます。
initItemPayment	ブール型。True に設定されていると、PaymentGroupDroplet は、オーダー内の個々の CommerceItem に対応する CommerceItemPaymentInfo オブジェクトを作成し、それを CommerceIdentifierPaymentInfoContainer に追加します。ユーザーが自分のプロファイルにデフォルトの PaymentGroup を持っていれば、CommerceItemPaymentInfo オブジェクトはその PaymentGroup で初期化されます。デフォルトで False に設定されます。 **注意:** CommerceItemPaymentInfo オブジェクトは、CommerceIdentifier が CommerceItem である CommerceIdentifierPaymentInfo オブジェクトです。このオブジェクトは CommerceItem 支払情報に使用されます。
initOrderPayment	ブール型。True に設定されていると、PaymentGroupDroplet は、OrderPaymentInfo オブジェクトを作成し、それを CommerceIdentifierPaymentInfoContainer に追加します。ユーザーが自分のプロファイルにデフォルトの PaymentGroup を持っていれば、OrderPaymentInfo オブジェクトはその PaymentGroup で初期化されます。デフォルトで True に設定されます。 **注意:** OrderPaymentInfo オブジェクトは、CommerceIdentifier が Order である CommerceIdentifierPaymentInfo オブジェクトです。このオブジェクトは Order 支払情報に使用されます。
initPaymentGroups	ブール型。True に設定されていると、paymentGroupTypes 入力パラメータで渡された PaymentGroup タイプが初期化されます。

パラメータ	説明
initShippingPayment	ブール型。True に設定されていると、PaymentGroupDroplet は、オーダー内の個々の ShippingGroup に対応する ShippingGroupPaymentInfo オブジェクトを作成し、それを CommerceIdentifierPaymentInfoContainer に追加します。ユーザーが自分のプロファイルにデフォルトの PaymentGroup を持っていれば、ShippingGroupPaymentInfo オブジェクトはその PaymentGroup で初期化されます。デフォルトで False に設定されます。 **注意:** ShippingGroupPaymentInfo オブジェクトは、CommerceIdentifier が ShippingGroup である CommerceIdentifierPaymentInfo オブジェクトです。このオブジェクトは ShippingGroup 支払情報に使用されます。
initTaxPayment	ブール型。True に設定されていると、PaymentGroupDroplet は、TaxPaymentInfo オブジェクトを作成し、それを CommerceIdentifierPaymentInfoContainer に追加します。 TaxPaymentInfo オブジェクトは、CommerceIdentifier が Order である CommerceIdentifierPaymentInfo オブジェクトです。このオブジェクトは税支払情報に使用されます。
order	ユーザーの Order。このパラメータを使用して PaymentGroupDroplet.order のデフォルト設定を上書きできます。
paymentGroupTypes	実行される PaymentGroupInitializer コンポーネントを決定するために使用される PaymentGroup タイプ（「creditCard」、「storeCredit」、「giftCertificate」など）のカンマ区切りのリスト。

PaymentGroupDroplet は次の出力パラメータを設定します。

パラメータ	説明
cipiMap	CommerceIdentifierPaymentInfoContainer によって参照される Map。
order	ユーザーのオーダーを表す Order オブジェクト。
paymentGroups	PaymentGroupMapContainer によって参照される Map。

PaymentGroupDroplet は、output という名前の 1 つのオープン・パラメータをレンダリングします。

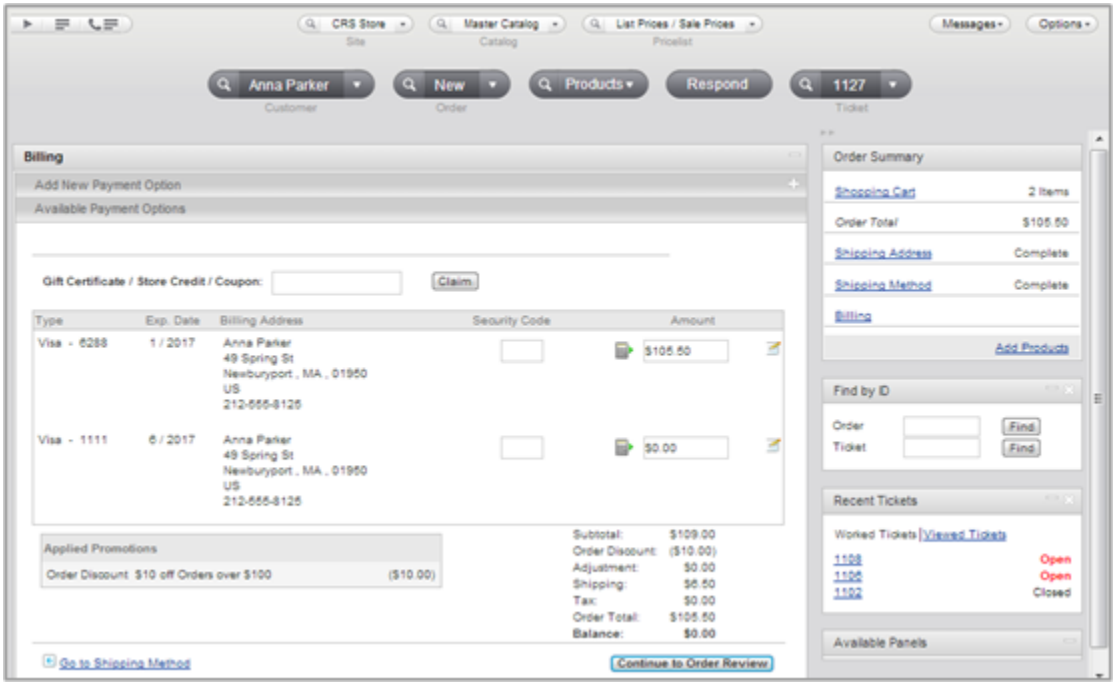
次のコード例は、PaymentGroupDroplet の使用例を示しています。この例では、現在のユーザーにとっての可用性に基づいて、CreditCard、StoreCredit および GiftCertificate の PaymentGroup オブジェクトを作成します。さらに、CommerceItemPaymentInfo オブジェクト、ShippingGroupPaymentInfo オブジェクトおよび TaxPaymentInfo オブジェクトも作成します。この例では、ユーザーが、使用可能な任意の PaymentGroup オブジェクトを使用して、Order 内の CommerceIdentifiers の代金を明細項目レベルで支払うことができます。

```
<dsp:droplet name="PaymentGroupDroplet">
  <dsp:param value="true" name="clear"/>
  <dsp:param value="giftCertificates, storeCredit, creditCard"
       name="paymentGroupTypes"/>
  <dsp:param value="true" name="initPaymentGroups"/>
  <dsp:param value="true" name="initItemPayment"/>
  <dsp:param value="true" name="initTaxPayment"/>
  <dsp:param value="true" name="initShippingPayment"/>
  <dsp:oparam name="output">Manipulation of objects here...
  </output>
</dsp:droplet>
```

サンプル・カタログの `billing.jsp` および `invoice_request.jsp` で、`PaymentGroupDroplet` の使用例を示す別の JSP のコード例を参照できます。

createAllPaymentInfos の使用

ユーザーが自分のプロファイル内の個々の支払タイプによって支払われるオーダーの金額を手動で指定するためのユーザー・インタフェースをサポートするのに必要なオブジェクトを `createAllPaymentInfos` の入力パラメータを使用して初期化します。



`createAllPaymentInfos` パラメータが `True` に設定されていると、`PaymentGroupDroplet` は、ユーザーのプロファイルに格納されているすべての可能な支払タイプ（クレジット・カード、ストア・クレジット、商品券または請求書要求）に対応する `OrderPaymentInfo` を作成します。当初はオーダー全体のコストがデフォルトの `PaymentGroup` に関連付けられた `OrderPaymentInfo` に割り当てられ、その他すべての `OrderPaymentInfo` オブジェクトが 0 に設定されます。たとえば、次のシナリオを考えます。

- オーダーの合計コストは 100 ドルです。

- ユーザーのプロファイルには、Credit Card A と Credit Card B という 2 つのクレジット・カードおよびストア・クレジットが格納されています。
- Credit Card A はユーザーのデフォルトの支払方法です。

このシナリオでは、PaymentGroupDroplet が 2 つのクレジット・カードと 1 つのストア・クレジットに対応する 3 つの OrderPaymentInfo オブジェクトを作成します。Credit Card A に関連付けられている OrderPaymentInfo の amount プロパティは 100 に設定されます。Credit Card B およびストア・クレジットに関連付けられている OrderPaymentInfo オブジェクトの amount プロパティは 0 に設定されます。

このアプリケーションでは、OrderPaymentInfo オブジェクトのリストに対する繰り返し処理によって生成された支払オプションのリストがユーザーに提示されます。ユーザーは個々の支払オプションによって支払われる金額をフォームで直接指定し、個々の PaymentGroup に関連付けられた OrderPaymentInfo オブジェクトの amount プロパティを実質的に設定します。ユーザーが精算プロセス中に別の PaymentGroups を追加した場合は、再度 PaymentGroupDroplet を呼び出して、新規に追加された PaymentGroups に対応する OrderPaymentInfo オブジェクトを作成する必要があります。

オーダーへの支払グループの追加

次の 2 つのプロセス(詳細は前の項を参照)を通じて Order の支払情報が収集されたら、PaymentGroupFormHandler を使用して支払グループを Order に追加します。

- Order で使用される可能性がある支払グループが CreateCreditCardFormHandler、CreateInvoiceRequestFormHandler および PaymentGroupDroplet、あるいはそのいずれかの組合せを使用して作成されます。支払グループが PaymentGroupMapContainer に追加されます。
- Order 内の CommerceIdentifiers に対応する CommerceIdentifierPaymentInfo オブジェクトが PaymentGroupDroplet を使用して作成されます。CommerceIdentifierPaymentInfo オブジェクトが CommerceIdentifierPaymentInfoContainer に追加されます。

例として、サンプル・カタログ内の complex_billing.jsp から引用した次のコード・セグメントについて考えてみましょう。このページでは、ユーザーが Order の合計コストを複数の CreditCard 支払グループ間で分割できます。

注意: 次のコード・セグメントでは、参照される個々のコンポーネントが dsp:importbean タグを介してページにインポートされたと想定できます。これらのインポート文については、実際の JSP を参照してください。

```
<dsp:droplet name="PaymentGroupDroplet">
  <dsp:param name="initOrderPayment" param="init"/>
  <dsp:param name="clearPaymentInfos" param="init"/>
  <dsp:oparam name="output">
  <dsp:setvalue bean="PaymentGroupFormHandler.listId"
paramvalue="order.id"/>
  <!-- begin output -->
     <table border=0 cellpadding=0 cellspacing=0 width=800>
      <tr>
      </tr>
      <tr>
        <td width=55></td>
        <td valign="top" width=745>
        <table border=0 cellpadding=4 width=80%>
          <tr><td></td></tr>
```

```
          <tr>
            <td colspan=2><span class="big">Billing</span></td>
          </tr>
          <tr><td></td></tr>
          <tr>
            <td colspan=2><b>Split payment by order amount</b><br>
            Order total: <dsp:valueof converter="currency"
param="order.priceInfo.total"/><br>
            <span class=help>Enter the amount you wish to move to another
payment method and select the new method. The remaining amount will stay
on the default payment method. <P>You must save changes before
continuing.</span></td>
          </tr>
          <tr valign=top>
            <td>
            <table border=0 cellpadding=4 cellspacing=1>
              <tr valign=top>
                <td colspan=9 align=right>
                </td>
              </tr>
              <tr valign=bottom bgcolor="#666666">
                <td colspan=2><span class=smallbw>Amount</span></td>
                <td colspan=2><span class=smallbw>Amt to move
 </span></td>
                <td colspan=2><span class=smallbw>Payment
method</span></td>
                <td colspan=3><span class=smallbw>Save changes</span></td>
              </tr>
                  <dsp:droplet name="ForEach">
                    <dsp:param bean="PaymentGroupFormHandler.currentList"
name="array"/>
                    <dsp:oparam name="output">
                      <!-- begin order line item -->
                      <dsp:form action="complex_billing.jsp"
method="post">
                      <tr valign=top>
                        <td><dsp:valueof converter="currency"
param="element.amount"/></td>
                        <td> </td>
                        <td>
                        $<dsp:input
bean="PaymentGroupFormHandler.currentList[param:index].splitAmount"
size="6" value="0.00" type="text"/></td>
                        <td> </td>
                        <td>
                          <dsp:select
bean="PaymentGroupFormHandler.currentList[param:index].splitPaymentMe
thod">
                          <dsp:droplet name="ForEach">
```

```
                              <dsp:param name="array"
param="paymentGroups"/>
                              <dsp:oparam name="output">
                                <dsp:droplet name="Switch">
                                  <dsp:param name="value" param="key"/>
                                  <dsp:getvalueof id="nameval3"
param="...element.paymentMethod" idtype="java.lang.String">
<dsp:oparam name="<%=nameval3%>">
                                    <dsp:getvalueof id="option264"
param="key" idtype="java.lang.String">
<dsp:option selected="<%=true%>" value="<%=option264%>"/>
</dsp:getvalueof><dsp:valueof param="key"/>
                                  </dsp:oparam>
</dsp:getvalueof>
                                  <dsp:oparam name="default">
                                    <dsp:getvalueof id="option272"
param="key" idtype="java.lang.String">
<dsp:option selected="<%=false%>" value="<%=option272%>"/>
</dsp:getvalueof><dsp:valueof param="key"/>
                                  </dsp:oparam>
                                </dsp:droplet>
                              </dsp:oparam>
                            </dsp:droplet>
                            </dsp:select>
                          </td>
                          <td> </td>
                          <td> </td>
                          <td> </td>
                          <td>
                            <dsp:input bean="PaymentGroupFormHandler.ListId"
paramvalue="order.id" priority="<%=(int)9%>" type="hidden"/>
                            <dsp:input
bean="PaymentGroupFormHandler.splitPaymentInfosSuccessURL" type="hidden"
value="complex_billing.jsp?init=false"/>
                            <dsp:input
bean="PaymentGroupFormHandler.splitPaymentInfos" type="submit"
value=" Save "/>
                          </td>
                        </tr>
                        </dsp:form>
                        <!-- end order line item -->
                      </dsp:oparam>
                    </dsp:droplet>

      <td colspan=9>
<%-- table with one row with one cell  --%>
      <table border=0 cellpadding=0 cellspacing=0 width=100%>
        <tr bgcolor="#666666">
          <td></td>
```

```
        </tr>
      </table>
      </td>
    </tr>
          </table>
        </td>
      </tr>
      <tr>
        <td><br>
          <dsp:form action="complex_billing.jsp" method="post">
          <dsp:input
bean="PaymentGroupFormHandler.applyPaymentGroupsSuccessURL" type="hidden"
value="order_confirmation.jsp"/>
          <dsp:input bean="PaymentGroupFormHandler.applyPaymentGroups"
type="submit" value="Continue"/>
          </dsp:form>
        </td>
      </tr>
     </table>
     </td>
    </tr>
    </table>
    </div>

  <!-- end output -->
  </dsp:oparam>
</dsp:droplet> <!-- end PaymentGroupDroplet -->
```

complex_billing.jsp の次の部分に注意してください。

1. ページがレンダリングされるときは、PaymentGroupDroplet を使用して、ユーザーの現在のオーダーに対応する OrderPaymentInfo オブジェクトを初期化し、それを CommerceIdentifierPaymentInfoContainer に追加します。前の項で説明したように、PaymentGroupDroplet は、デフォルトで、個々の CommerceIdentifierPaymentInfo オブジェクトを PaymentGroupMapContainer 内のデフォルトの支払グループに関連付けます（前のページの billing.jsp で、使用可能な CreditCard 支払グループが初期化され、PaymentGroupMapContainer に追加されていることに注意してください）。
2. PaymentGroupFormHandler.listId プロパティは、PaymentGroupDroplet の order 出力パラメータで設定された Order オブジェクトの ID に設定されます。オーダーの ID が、PaymentGroupFormHandler.currentList プロパティを介して公開されている CommerceIdentifierPaymentInfo オブジェクトの List のキーになります（ページの下にある非表示の入力タグも、それに続く要求に応じてこのプロパティを設定します）。
3. コードの残りの部分では、ユーザーがオーダー内の合計コストの特定の金額を別々の支払グループに割り当てるために使用できるインタフェースをレンダリングします。これは、ネストされた ForEach サーブレット Bean を使用して実行されます。
   - 外側の ForEach サーブレット Bean は、現在の Order に対応する CommerceIdentifierPaymentInfo オブジェクトの配列を入力パラメータとして

受け取ります。外側のサーブレット Bean は、自分の output oparam を配列内の 1 つの CommerceIdentifierPaymentInfo オブジェクトにつき 1 回レンダリングします。レンダリングされた出力は、実質的に、次のものを表示するフォームです。現在の CommerceIdentifierPaymentInfo オブジェクトに関連付けられている金額、CommerceIdentifierPaymentInfo オブジェクト内の指定された金額に対応する支払グループを変更するためのドロップダウン・リスト、選択されている支払グループに関連付ける CommerceIdentifierPaymentInfo 内の金額を指定するためのテキスト・ボックス、およびこれらの新しい関連付けを作成するための保存発行ボタン。

支払グループ・ドロップダウン・リストには、PaymentGroupMapContainer 内の支払グループが挿入されます。これらの支払グループは、paymentGroups コンビニエンス・パラメータを介して PaymentGroupDroplet によって公開され、2 番目のネストされた ForEach サーブレット Bean を使用して支払グループに対する繰り返し処理が実行され、ドロップダウン・リストに支払グループが挿入されます。

支払グループ・ドロップダウン・リストが現在の CommerceIdentifierPaymentInfo オブジェクトの splitPaymentMethod プロパティに関連付けられていることに注意してください。同様に、金額テキスト・ボックスは、現在の CommerceIdentifierPaymentInfo オブジェクトの splitAmount プロパティに関連付けられています。

最後に、保存発行ボタンが PaymentGroupFormHandler の handleSplitPaymentInfos メソッドを呼び出すことに注意してください。handleSplitPaymentInfos メソッドは、amount 内の値と CommercIdentifierPaymentInfo オブジェクトの splitAmount プロパティを使用して、オブジェクトを 2 つのオブジェクトに分割するかどうかを決定します。分割の必要があれば、これらのプロパティおよび splitPaymentMethod プロパティで指定された支払グループを使用して、2 番目の CommerceIdentifierPaymentInfo オブジェクトを作成します。このメソッドは、次に、新旧両方のオブジェクトのプロパティを適切に設定し、新規オブジェクトを CommerceIdentifierPaymentInfoContainer に追加します。フォームが処理されれば、再びページがレンダリングされ、ユーザーの加えた変更がページに反映されます。

4. ページ最下部の続行発行ボタンを使用して、ユーザーは現在の支払関連付けをオーダーに適用し、オーダーの確認に進むことができます。この発行ボタンは、ユーザーが選択した支払グループを Order に追加する、PaymentGroupFormHandler の handleApplyPaymentGroups メソッドを呼び出します。それを行うために、このメソッドは、CommerceIdentifierPaymentInfoContainer 内の CommerceIdentifierPaymentInfo オブジェクトに対する繰り返し処理を実行します。コンテナ内の CommerceIdentifierPaymentInfo オブジェクトごとに、関連付けられている支払グループが PaymentGroupMapContainer から取得されて Order に追加され、関連付けられている CommerceIdentifier の適切な金額がその PaymentGroup に追加されます。

注意: PaymentGroupFormHandler のすべてのハンドル・メソッドの詳細は、『ATG Web Commerce Programming Guide』の「*購買プロセス・サービスの構成*」の *複雑なオーダーを精算する準備*に関する項を参照してください。支払グループへのコストの追加の詳細は、『ATG Web Commerce Programming Guide』の「*購買プロセス・オブジェクトの操作*」の *支払グループへのコストの割当*てに関する項を参照してください。

PaymentGroupFormHandler の使用方法を示す他の JSP コード例については、Motorprise リファレンス・アプリケーションの checkout/payment_methods.jsp、checkout/SplitPaymentOrderDetails.jsp および checkout/SplitPaymentDetails.jsp を参照してください。特に、/checkout/SplitPaymentDetails.jsp が、オーダー内のコンポーネントのコスト、つまり「明細項目」

（つまり、品目コスト、送料、税）をユーザーが複数の支払グループ間で分割できるようにする方法を示していること注意してください。これらの Motorprise のページには、<ATG11dir>\MotorpriseJSP\j2ee-apps\motorprise\web-app\en\checkout\でアクセスできます。「ページおよびコンポーネント」→「J2EE ページ」タスク領域を介して ACC の文書エディタでこれらのページを開くこともできます。詳細は、『ATG Web Commerce ビジネス・コマース・リファレンス・アプリケーション・ガイド』の「*オーダーの処理*」の *支払情報*に関する項を参照してください。

ショッピング・カートの再価格設定

この章のショッピング・カートへの品目の追加で前述したように、CartModifierFormHandler は、カートへの品目の追加に使用されると、ショッピング・カートを自動的に再価格設定します（カートからの品目の*削除*に使用されたときも、カートを再価格設定することに注意してください）。ただし、ショッピング・カートを再価格設定しない他のフォーム・ハンドラを介して、オーダー価格に影響を及ぼす変更（たとえば、出荷グループを作成および管理するフォーム・ハンドラを介した出荷の変更など）を顧客が加えられる場合、またはシナリオを利用したプロモーションの配信など、オーダー価格に影響を及ぼす他の手段を通じてショッピング・カートが変更される場合は、他のメカニズムを利用してショッピング・カートを再価格設定する必要があります。

ショッピング・カートを再価格設定する必要のあるページがサイトにある一方で、フォーム処理およびそれに対応するハンドル・メソッドを利用してショッピング・カートを再価格設定できない場合は、RepriceOrderDroplet サーブレット Bean を使用します。実際、RepriceOrderDroplet サーブレット Bean を使用して、顧客がショッピング・カート・ページにアクセスするたびに顧客のショッピング・カートを再価格設定できます。そうすることで、顧客は、カートに変更を加えながら、常に正確な価格設定情報を表示できます。

RepriceOrderDroplet サーブレット Bean は 1 つの必須入力パラメータ pricingOp を取ります。このパラメータは、価格設定操作が実行されるように設定する必要があります。可能な価格設定操作は atg.commerce.pricing.PricingConstants で定義されており、そこには次の操作が含まれています。

ORDER_TOTAL
ORDER_SUBTOTAL
ORDER_SUBTOTAL_SHIPPING
ORDER_SUBTOTAL_TAX
ITEMS
SHIPPING
ORDER
TAX
NO_REPRICE

一般的に、pricingOp 入力パラメータは RepriceOrderDroplet を使用するときに指定する必要のある唯一のパラメータです。RepriceOrderDroplet のすべてのパラメータの詳細なリストについては、付録 A: Core Commerce サーブレット Bean の RepriceOrder を参照してください。

サイトのページ上で RepriceOrderDroplet を使用するには、次のインポート文を使用してサーブレット Bean をページにインポートします。

```
<dsp:importbean bean="/atg/commerce/order/purchase/RepriceOrderDroplet">
```

次に、現在のショッピング・カートの価格設定情報を表示する前に、次のコードと同様の JSP コードを追加します。

```
<dsp:droplet name="RepriceOrderDroplet">
  <dsp:param value="ORDER_SUBTOTAL" name="pricingOp"/>
</dsp:droplet>-->
```

RepriceOrderDroplet サーブレット Bean の詳細は、『ATG Web Commerce Programming Guide』の「*購買プロセス・サービスの構成*」の*オーダーの再価格設定*に関する項を参照してください。

ショッピング・カートの保存

SaveOrderFormHandler を使用して、ユーザーの現在のショッピング・カートを保存し、そのショッピング・カートを ShoppingCart.saved 内のユーザーの保存されているカートのリストに追加し、新しい空のカートを ShoppingCart.current 内のユーザーの現在のショッピング・カートとして作成します。

さらに、SaveOrderFormHandler の description プロパティを使用して、Order の description プロパティを設定します。そうすることで、ユーザーは意味のある名前を自分のショッピング・カートに付けることができます。ユーザーがわかりやすい名前を指定しない場合、SaveOrderFormHandler は、ユーザーのロケールと現在の日付、時刻を使用して、自動的にわかりやすい名前を作成します。

次の JSP コードは、SaveOrderFormHandler の使用例を示しています。

```
<dsp:importbean bean="/atg/commerce/order/purchase/SaveOrderFormHandler"/>

Order # <dsp:valueof bean="ShoppingCart.current.id"/>
  <dsp:form action="saved_orders.jsp">
    Enter a name to identify this order:<p>
    <dsp:input bean="SaveOrderFormHandler.description" type="text"/>
    <dsp:input bean="SaveOrderFormHandler.saveOrder" value="Save order"
         type="submit"/>
    <dsp:input bean="SaveOrderFormHandler.saveOrderSuccessURL"
         value="../user/saved_orders.jsp" type="hidden"/>
    <dsp:input bean="SaveOrderFormHandler.saveOrderErrorURL"
         value="../user/save_order.jsp" type="hidden"/>
  </dsp:form>
```

SaveOrderFormHandler のハンドル・メソッドおよびオーダー・リポジトリに Order を保存する方法の詳細は、『ATG Web Commerce Programming Guide』の「*購買プロセス・サービスの構成*」の*オーダーの保存*に関する項を参照してください。SaveOrderFormHandler の使用例を示す JSP のコード例については、『ATG Web Commerce ビジネス・コマース・リファレンス・アプリケーション・ガイド』の「*自分のアカウント*」の*オーダーの保存*に関する項を参照してください。

OrderLookup サーブレット Bean は、渡されている入力パラメータに応じて、1 つ以上の Order オブジェクトを取得します。このサーブレット Bean を使用して次のものを取得できます。

- 1 つのオーダー
- 特定のユーザーによって確定されたすべてのオーダー
- 特定のユーザーによって確定され、特定の状態にあるすべてのオーダー
- 特定のコスト・センターに割り当てられているすべてのオーダー

Core Commerce に付属している OrderLookup サーブレット Bean は、atg.commerce.order.OrderLookup クラスのインスタンスです。詳細は、付録 A: Core Commerce サーブレット Bean を参照してください。

12 オーダー承認プロセスの実装

アプリケーションによっては、オーダーを承認または否認できる権限のある人が顧客のオーダーを審査する必要があります。アプリケーションがオーダー承認プロセスを実装している場合は、特定の承認者の承認を必要とするオーダーをその承認者に表示する必要があり、承認者の決定(承認または否認あるいはその両方)を処理するための何らかの手段を提供する必要があります。ApprovalRequiredDroplet と ApprovalFormHandler は、それを目的に用意されています。

さらに、承認者が自分の承認したオーダーまたは否認したオーダーあるいはその両方の履歴リストの表示を求めることもあります。ApprovedDroplet は、それを目的に用意されています。

この章では、これら 3 つのコンポーネントをアプリケーションの JSP で使用して、オーダー承認プロセスの次の部分をサポートする方法を説明します。

承認を必要とするオーダーの表示

ApprovalRequiredDroplet サーブレット Bean を使用して、特定の承認者による承認を必要とするすべてのオーダーを取得します。ApprovalRequiredDroplet はオーダー・リポジトリに対して問合せを実行し、次の 2 つの基準を満たすすべてのオーダーを返します。

- オーダーの authorizedApproverIds プロパティに承認者の ID が含まれていること。
- オーダーの状態が承認を必要としていること。つまり、状態が ApprovalRequiredDroplet の orderStatesRequiringApproval プロパティで定義されていること。オーダーの状態は、ApprovalRequireDroplet の orderStatePropertyName プロパティで指定されているオーダーのプロパティによって保持されます。デフォルト値は PENDING_APPROVAL です。

詳細は、付録 A: Core Commerce サーブレット Bean の ApprovalRequiredDroplet エントリを参照してください。

承認と否認の処理

ApprovalFormHandler フォーム・ハンドラ(atg.commerce.approval.ApprovalFormHandler クラス)を使用して承認者によるオーダーの承認または否認を処理します。ApprovalFormHandler クラスには、handleApproveOrder と handleRejectOrder という 2 つのハンドル・メソッドが含まれています。次の方法で、これらのハンドル・メソッドを Submit プロパティに関連付けることができます。

```
<dsp:input bean="ApprovalFormHandler.approveOrder" value=" Approve Order"
           type="submit"/>
<dsp:input bean="ApprovalFormHandler.rejectOrder" value=" Reject Order"
           type="submit"/>
```

ApprovalFormHandler.approveOrder に対して handleApproveOrder メソッドが呼び出された場合は、handleApproveOrder メソッドが承認者によるオーダーの*承認*を処理します。それと対照的に、ApprovalFormHandler.rejectOrder に対して handleRejectOrder メソッドが呼び出された場合は、handleRejectOrder メソッドが承認者によるオーダーの*否認*を処理します。

次の JSP の例は、1 つのオーダーに対して ApprovalFormHandler を使用する方法を示しています。この例では、承認者が、オーダーID および承認者によるオーダーの承認または否認に関連付けるコメント(またはメッセージ)を入力します。次に、承認者は、オーダーの承認発行ボタンまたはオーダーの否認発行ボタンをクリックすることによって、オーダーを承認または否認します。

```
<dsp:importbean bean="/atg/commerce/approval/ApprovalFormHandler"/>
<dsp:form action="pendingapprovalOrders.jsp">
   <dsp:input bean="ApprovalFormHandler.ApproveOrderSuccessURL"
         value="pendingapprovalOrders.jsp" type="hidden"/>
   <dsp:input bean="ApprovalFormHandler.ApproveOrderErrorURL"
         value="pendingapprovalOrders.jsp" type="hidden"/>
   <dsp:input bean="ApprovalFormHandler.RejectOrderSuccessURL"
         value="pendingapprovalOrders.jsp" type="hidden"/>
   <dsp:input bean="ApprovalFormHandler.RejectOrderErrorURL"
         value="pendingapprovalOrders.jsp" type="hidden"/>
   Order Id <dsp:input bean="ApprovalFormHandler.orderId" size="10" value=""
             type="text"/>
   Approver Message <dsp:input bean="ApprovalFormHandler.approverMessage"
         size="500" value="" type="text"/>
   <dsp:input bean="ApprovalFormHandler.approveOrder" value="  Approve Order  "
         type="submit"/>
   <dsp:input bean="ApprovalFormHandler.rejectOrder" value="  Reject Order  "
         type="submit"/>
</dsp:form>
```

注意: 実際の実装では、JSP ファイルは Web アプリケーション内にあり、文書ルートの相対パスにはありません。

承認されたオーダーおよび否認されたオーダーの履歴の表示

ApprovedDroplet サーブレット Bean を使用して、承認または否認されたすべてのオーダー、および approverid パラメータ内のプロファイル ID が承認者 ID リストに含まれるすべてのオーダーを取得します。オーダーの現在の状態は考慮されず、承認者がオーダーとやり取りしたかどうかのみが考慮されます。このドロップレットの詳細は、付録 A: Core Commerce サーブレット Bean を参照してください。

13 商品コレクションのフィルタリング

Oracle Commerce Core Commerce では、製品に対して機能するように設計された単独コレクション・フィルタリング・コンポーネントおよびチェーンされたコレクション・フィルタリング・コンポーネントを用意することによって、このプラットフォームのコレクション・フィルタリング機能を拡張しています。

コレクション・フィルタリングについて説明するこの章は、次の項から構成されています。

コレクション・フィルタリングの詳細は、次の資料を参照してください。

- コレクション・フィルタリングの一般的な説明については、『ATG Web Commerce Personalization Programming Guide』の*コレクションのフィルタリング*を参照してください。
- `CollectionFilter` サーブレット Bean のリファレンス情報については、『ATG Web Commerce Page Developer's Guide』の *付録 B: ATG サーブレット Bean* を参照してください。
- API コレクション・フィルタリングに関するドキュメントは、『ATG Web Commerce Platform API Reference』の `atg.service.collections.filter` パッケージおよび `atg.commerce.collections.filter` パッケージを参照してください。

製品コレクション・フィルタリングが機能する仕組み

単独コレクション・フィルタまたはチェーン内のフィルタを使用してコレクションから製品をフィルタできます。この例は、フィルタのチェーンで定義されている条件を満たす製品のコレクションをレンダリングする方法を示しています。レンダリングされたコレクションは将来使用するためにキャッシュされます。

販売中のアクティブな製品のリストを Web ページ上で顧客に表示することを考えてみましょう。それを実行するには、JSP 内でコレクション・フィルタリング・サーブレット Bean (`ProductFilterDroplet`) を使用して、コレクション・フィルタリング・コンポーネント (`ProductFilter`) にアクセスし、そのコンポーネントがフィルタのチェーンを製品のコレクションに適用します。`ProductFilterDroplet` およびその他のコレクション・フィルタリング・サーブレット Bean の詳細は、CollectionFilter を参照してください。

`ProductFilter` の役割は、別々のコレクション・フィルタをチェーン状に連結し、そのチェーンを実行することです。次の例では、`ProductFilter.filters` プロパティが、次の 2 つのコレクション・フィルタを識別しているものと想定しています。`StartEndDateFilter` および `InventoryFilter`。これらのフィルタは集団的に作用して、販売されていない製品 (`StartEndDateFilter`) および在庫がない製品 (`InventoryFilter`) を削除します。

JSP コードは次のようになります。

```
<dspel:droplet name="/atg/commerce/collections/filter/droplet/
      ProductFilterDroplet">
   <dsp:param name="collection" param="item.childproducts/>
   <dsp:param name="collectionIdentifierKey" value="catid-0067-hardscape"/>

   <dspel:oparam name="output">
   Featured Plants:<p>
      <dsp:droplet name="/atg/dynamo/droplet/ForEach">
         <dsp:param name="array" param="filteredCollection"/>

         <dsp:oparam name="output">
            <dspel:valueof param="element"/>
         </dspel:oparam>
      </dsp:droplet>
   </dspel:oparam>

   <dspel:oparam name="empty">
      There are currently no outdoor plants
   </dspel:oparam>
</dspel:droplet>
```

この JSP を実行すると、ProductFilterDroplet は製品を ProductFilter コンポーネントに渡して、チェーンを実行します。

- 最初のコンポーネントは、現在の日付を、個々の製品の startDate プロパティおよび endDate プロパティ内の値と比較する StartEndDateFilter です。アクティブでない製品（「開始されていない」製品またはすでに「終了した」製品）は放棄され、残った製品が InventoryFilter に渡されます。
- InventoryFilter は、在庫マネージャと通信して、スロット内のすべての製品の可用性を調べます。在庫がなくなっている製品はコレクションから削除されます。
- ProductFilter でキャッシュが使用可能になっていれば、FilterCache コンポーネントは、プレフィルタされたコレクションとフィルタされた結果に関する情報を保存します。この JSP のそれ以降のレンダリングでは、キャッシュされているプレフィルタされたコレクションを現在のプレフィルタされたコレクションと比較します。適切な場合は、キャッシュされているフィルタされた結果が使用されます。キャッシュの詳細は、『ATG Web Commerce Personalization Programming Guide』の*コレクションのフィルタリング*を参照してください。

コレクション・フィルタリング・コンポーネントの使用

コレクション・フィルタリング・コンポーネントは、一意の条件セットに基づいてコレクション内のオブジェクトを減らすコンポーネントです。Core Commerce は、3 つのコレクション・フィルタリング・コンポーネントを備えています。

- InventoryFilter の使用
- ExcludeItemsInCartFilter の使用
- ProductFilter の使用
- CartSharingFilter の使用

これらのコンポーネントは、別のリソース(シナリオまたはサーブレット Bean)からコレクションを受け取り、結果として生成されたコレクションを呼び出し元のリソースに返します。シナリオでコレクション・フィルタリング・コンポーネントを使用するには、コレクション・フィルタリング・コンポーネントを使用する `Filter Slot Contents` 処理要素を定義します。このシナリオ処理については、『ATG Web Commerce Personalization Programming Guide』を参照してください。サーブレット Bean を使用してコレクション・フィルタリング・コンポーネントにアクセスするための具体的な手順は、次の各項のコンポーネントの説明に含まれています。

コレクションには任意の種類のオブジェクトを格納できますが、ここで説明するコレクション・フィルタリング・コンポーネントは、タイプが `Product` の `RepositoryItems` に対して作用するように定義されていることを念頭に置いてください。コレクション内のすべての非 `Product` 項目はコンポーネントによって無視され、結果セットに含まれます。

InventoryFilter の使用

`atg.commerce.collections.filter.InventoryFilter` クラスのコンポーネントである `/atg/registry/CollectionFilters/InventoryFilter` コンポーネントは、在庫の可用性が特定の状態にある製品をコレクションから除外するために使用されます。

`InventoryFilter` は、個々の製品の SKU について `InventoryManager` で定義されている在庫ステータスを `InventoryFilter.IncludeInventoryStates` プロパティで指定されているステータスと比較します。製品のいずれかの SKU が `IncludeInventoryStates` プロパティに含まれているステータスを持っていれば、その製品は結果セット・コレクションに残ります。デフォルトで、`InventoryFilter.IncludeInventoryStates` は、`1000`(在庫あり)、`1002`(プレ・オーダー可能)および `1003`(バック・オーダー可能)という値を持っているため、これらの値のいずれかが含まれているすべての製品が返されるコレクションに追加されます。

`InventoryFilterDroplet` サーブレット Bean(直接)および `ProductFilterDroplet`(フィルタ・チェーンの一部として)を経由して、`InventoryFilter` コンポーネントにアクセスできます。コレクション・フィルタのコンテンツをキャッシュすることもできますが、独自のキャッシュ・メカニズムを持っている `CachingInventoryManager` のような在庫マネージャを使用している場合は、`InventoryFilter` のコンテンツをキャッシュしないことをお薦めします。キャッシュの詳細は、『ATG Web Commerce Personalization Programming Guide』の*コレクションのフィルタリング*を参照してください。

ExcludeItemsInCartFilter の使用

`atg.commerce.collections.filter.ExcludeItemsInCartFilter` クラスのコンポーネントである `/atg/registry/CollectionFilters/ExcludeItemsInCartFilter` コンポーネントは、ユーザーのショッピング・カート内にあるすべての製品をコレクションから削除するために使用されます。このコンポーネントは、`shoppingCartPath` プロパティを利用して、現在のユーザーのショッピング・カートを見つけます。デフォルト値は `/atg/commerce/ShoppingCart` です。

`ExcludeItemInCartFilterDroplet` を(直接)経由して `ExcludeItemsInCartFilter` コンポーネントにアクセスできます。コレクション・フィルタのコンテンツをキャッシュすることもできますが、キャッシュされたコンテンツが再利用される可能性が低く、キャッシングがリソースの浪費になるため、`ExcludeItemsInCartFilter` から返される結果をキャッシュしないことをお薦めします。キャッシュの詳細は、『ATG Web Commerce Personalization Programming Guide』の*コレクションのフィルタリング*を参照してください。

ProductFilter の使用

atg.service.collections.filter.ChainedFilter クラスのコンポーネントである /atg/registry/CollectionFilters/ProductFilter コンポーネントは、フィルタの条件を製品のコレクションに適用するコレクション・フィルタのチェーンを作成するために使用されます。

ProductFilter は、ProductFilter.filters プロパティで定義されている個々のフィルタに次々と製品を渡し、個々のコレクション・フィルタによって指定されている条件を満たす最終的なコレクションを生成します。デフォルトで、ProductFilter.filters は、StartEndDateFilter および InventoryFilter に設定されます。

ProductFilterDroplet サーブレット Bean を使用して、JSP で ProductFilter コンポーネントにアクセスできます。ProductFilterDroplet を使用すると、結果として生成されるコンテンツはデフォルトでキャッシュされます(cacheEnabled が true に設定されます)。ChainedFilter クラスおよびキャッシングの詳細は、『ATG Web Commerce Personalization Programming Guide』の*コレクションのフィルタリング*を参照してください。

CartSharingFilter の使用

Oracle Commerce Platform の複数サイト機能を使用している場合、atg.commerce.collections.filter.ItemSiteFilter クラスのコンポーネントである /atg/registry/CollectionFilters/CartSharingFilter コンポーネントは、入力項目コレクションのサイト ID によって入力項目コレクションをフィルタします。

このフィルタは、atg.ShoppingCart 共有可能タイプの構成に基づいて、同じ共有グループ内にある製品のみを返します(『ATG Web Commerce Programming Guide』の*複数の Commerce の構成*に関する項を参照してください)。

2 つの追加プロパティである includeDisabledSites と includeInactiveSites は、どのサイト状態が考慮されるかを決定します(デフォルトで両方が false に設定されます)。共有可能タイプがない場合でも、これらのフィルタは考慮されます。現在のサイトがない場合は、フィルタリングされずにすべての品目が返されます。

JSP で CartSharingFilter コンポーネントにアクセスできます。CollectionFilter を参照してください。

複数サイトのギフト・リストとウィッシュ・リストのフィルタリング

Core Commerce は、ギフト・リストおよびギフト品目のコレクションをフィルタし、顧客のサイト・コンテキストに適したリスト/品目のみを表示する機能を備えています。複数サイト環境では、Profile.giftlists や Profile.wishlist.giftlistItems などのリポジトリ項目のプロパティを参照することによってギフト・リストまたはギフト品目のコレクションを取得するたびに、複数のサイトから取得された品目が含まれている可能性のあるフィルタされていないリストが返されます。これらの状況では、コレクションをフィルタする必要があるかどうかを検討し、フィルタする必要があれば、この項で説明するフィルタ機能を実装する必要があります。

フィルタリングはウィッシュ・リストで特に重要です。顧客は 1 つのウィッシュ・リストしか持てないため、複数のサイトから取得された品目が 1 つのウィッシュ・リスト内に存在する可能性が高くなります。ウィッシュ・リスト品目の表示をサイト・コンテキストに適した品目のみに限定するには、サイト・コンテキスト外のサイトに属するすべての品目を除外する必要があります。

2 つのコンポーネントがギフト・リストとギフト品目のフィルタリングを手助けします。

- /atg/commerce/collections/filter/droplet/ GiftlistSiteFilterDroplet は、GiftlistSiteFilter コンポーネントを呼び出して、指定されたギフト・リスト/ギフト品目のコレクションをフィルタし、フィルタされた結果をレンダリングします。
- /atg/registry/CollectionFilters/GiftlistSiteFilter コンポーネントは、ギフト・リストまたはギフト品目のコレクションをフィルタします。このコンポーネントは、サイト・コンテキストに適したリストまたは品目のみを返します。

GiftlistSiteFilterDroplet は、フィルタされたギフト・リストまたはギフト品目のセットのレンダリング用に設計された atg.service.collections.filter.droplet.CollectionFilter クラスのグローバル・スコープのインスタンスです。GiftlistSiteFilterDroplet は次のプロパティを持っています。

- filter:/atg/registry/CollectionFilters/GiftlistSiteFilter コンポーネントへの参照。
- extraParameterNames:JSP が GiftlistSiteFilterDroplet への入力パラメータとして指定できる追加パラメータ、具体的には siteIds と siteScope を識別するカンマ区切りのリスト。GiftlistSiteFilterDroplet は、これらのパラメータを GiftlistSiteFilter に渡して、サイト・コンテキスに基づいてギフト・リストをフィルタできるようにします。JSP がこれらのパラメータの値を渡さない場合、GiftlistSiteFilter はデフォルトで現在のサイトと GiftlistManager.siteScope の値を使用します。

GiftlistSiteFilter は、atg.commerce.gifts.GiftlistSiteFilter クラスのグローバル・スコープ・コンポーネントです。このクラスは、generateFilteredCollection()メソッドを上書きして、ギフト・リストまたはギフト品目をフィルタするときにサイト・スコープ・パラメータおよびサイト ID パラメータを考慮することによって、汎用のコレクション・フィルタリング・クラス atg.service.collections.filter.CachedCollectionFilter を拡張します。
GiftlistSiteFilter は、サイト・スコープ値とサイト ID 値を次のように解決します。

- サイト・スコープが渡された場合、GiftlistSiteFilter はフィルタリング中にそのスコープを使用します。サイト・スコープが渡されなければ、GiftlistSiteFilter は GiftlistManager を呼び出し、GiftlistManager の siteScope の設定をフィルタリングに使用します。siteScope がデフォルト設定の all になっていると、すべてのギフト・リストまたはギフト品目が常に返され、フィルタリングが行われないため、all 以外の siteScope を使用しているのでないかぎり、ギフト・リスト/ギフト品目フィルタを使用しないでください。
- サイト ID のリストが渡された場合、GiftlistSiteFilter はフィルタリング中にそのリストを使用します。サイト ID が渡されない場合は、現在のサイトの siteId がフィルタリング中に使用されます。

ギフト・リストをフィルタするときは、GiftlistSiteFilter がサイト・スコープを決定し、次の表に示すように、リポジトリ内のギフト・リスト/ギフト品目の siteId をこのコンポーネントのリスト内の ID と比較します。

互換性テスト	All	Current	ShareableType ID
ギフト・リスト/ギフト品目の siteId がサイト IDリスト内にある	ギフト・リスト/ギフト品目をフィルタされた結果に含める	ギフト・リスト/ギフト品目をフィルタされた結果に含める	ギフト・リスト/ギフト品目をフィルタされた結果に含める

互換性テスト	All	Current	ShareableType ID
ギフト・リスト/ギフト品目の siteId がサイト IDリスト内にない	ギフト・リスト/ギフト品目をフィルタされた結果に含める	ギフト・リスト/ギフト品目をフィルタされた結果に含めない	ギフト・リスト/ギフト品目の siteId が、サイト IDリスト内の任意のサイトが属する指定された共有グループ(たとえばショッピング・カート共有グループ)内にあれば、ギフト・リスト/ギフト品目をフィルタされた結果に含める
ギフト・リスト/ギフト品目の siteId が null である **注意:** ウィッシュ・リストの場合も同じです。	ギフト・リスト/ギフト品目は汎用であり、フィルタされた結果に含める必要がある	ギフト・リスト/ギフト品目は汎用であり、フィルタされた結果に含める必要がある	ギフト・リスト/ギフト品目は汎用であり、フィルタされた結果に含める必要がある

GiftlistSiteFilter コンポーネントのプロパティには次のものがあります。

- giftlistManager:ギフト・リスト・マネージャ・コンポーネント/atg/commerce/gifts/GiftlistManager への参照。
- siteGroupManager:サイト・グループ・マネージャ・コンポーネント/atg/multisite/SiteGroupManager への参照。このコンポーネントは、どのサイトが同じ共有グループの一部であり、ギフト・リストなどのデータを共有できるかを決定します。
- includeDisabledSites:true であれば、フィルタは使用不可のサイトに表示される品目を除外しません。デフォルトは false です。
- includeInactiveSites:true であれば、フィルタは非アクティブ・サイトに表示される品目を除外しません。デフォルトは false です。

GiftlistSearch フォーム・ハンドラと異なり、GiftlistSiteFilter コンポーネントは siteScope プロパティまたは siteIds プロパティを持っていません。そのかわり、後述するように、サイト・スコープとサイト ID のリストが GiftlistSiteFilterDroplet によって GiftlistSiteFilter に渡されます。

この JSP からの抜粋は、GiftlistSiteFilterDroplet を使用してギフト・リストをフィルタする方法の一例を示しています。サイト・スコープが渡されていないため、GiftlistSiteFilter は GiftlistManager コンポーネントの siteScope を使用します。siteScope は、この例の目的に合わせて atg.ShoppingCart 共有可能タイプ・コンポーネントに設定されています。また、サイト ID が指定されていないため、フィルタされるギフト・リストは、現在のサイトおよび現在のサイトとショッピング・カートを共有しているサイトからのみ取得されます。

```
<dsp:droplet
name="/atg/commerce/collections/filter/droplet/GiftlistSiteFilterDroplet">
  <&-- Specify the collection to filter --%>
  <dsp:param name="collection" bean="Profile.giftlists"/>

  <dsp:oparam name="output">

    <%-- Iterate through the collection. --%>
      <dsp:importbean bean="/atg/dynamo/droplet/ForEach"/>
      <dsp:droplet name="ForEach">
```

```
        <dsp:param name="array" param="filteredCollection"/>

        <dsp:oparam name="output">
          <dsp:setvalue param="giftList" paramvalue="element"/>
          <dsp:getvalueof var="eventName" param="giftList.eventName"/>
          <c:out value="${eventName}"/>
        </dsp:oparam>
      </dsp:droplet>

  </dsp:oparam>
</dsp:droplet>
```

この JSP からの抜粋は、ウィッシュ・リスト品目のコレクションをフィルタします。サイト・スコープ値 `current` がサイト ID のセットとともにフィルタに渡されるため、指定されたサイトの品目のコレクションのみが結果として生成されます。

```
<dsp:droplet
name="/atg/commerce/collections/filter/droplet/GiftlistSiteFilterDroplet">
  <&-- Specify the collection to filter and the site scope. --%>
  <dsp:param name="collection" bean="Profile.wishlist.giftlistItems"/>
  <dsp:param name="siteScope" value="current"/>
  <dsp:param name="siteIds" value="siteA,siteB"/>

  <dsp:oparam name="output">

    <%-- Iterate through the collection. --%>
      <dsp:importbean bean="/atg/dynamo/droplet/ForEach"/>
      <dsp:droplet name="ForEach">
        <dsp:param name="array" param="filteredCollection"/>

        <dsp:oparam name="output">
          <dsp:setvalue param="giftItem" paramvalue="element"/>
          <dsp:getvalueof var="displayName" param="giftItem.displayName"/>
          <c:out value="${displayName}"/>
        </dsp:oparam>
      </dsp:droplet>

   </dsp:oparam>
</dsp:droplet>
```

注意:

- `GiftlistSiteFilter` コンポーネントと `GiftlistSiteFilterDroplet` コンポーネントは、`CollectionFilter` クラスに基づく他のフィルタと同様に、フィルタリングのパフォーマンスを向上させるためにキャッシングを使用するように構成できます。『ATG Web Commerce Personalization Programming Guide』の *フィルタリングされたコンテンツのキャッシュ*に関する項を参照してください。
- `GiftlistSiteFilter` コンポーネントは、`GiftlistSiteFilterDroplet` とは独立に使用できます。そうするには、`GiftlistSiteFilter.generateFilteredCollection()`メ

ソッドを呼び出すときに、pExtraParameters マップ内の siteIds 値と siteScope 値を渡す必要があります（ドロップレットを使用するとき、ドロップレットは、GiftlistSiteFilter コンポーネントを呼び出す前に、pExtraParameters マップを作成します）。詳細は、『ATG Web Commerce Platform API Reference』を参照してください。

14 オーダー・マーカーおよび商品マーカーの使用

オーダー・マーカーおよび商品マーカーを使用すると、オーダーまたは商品に任意の情報を追加し、その情報をアプリケーションで使用するために永続化できます。

この目標を別の方法で達成するには、カスタム・プロパティを使用します。ただし、カスタム・プロパティが保持するデータは通常 1 つのみであるため、格納するデータのタイプごとに 1 つのカスタム・プロパティが必要になります。マーカーを使用すると、複数の情報を 1 つのリポジトリ項目に格納し、サポートする条件、処理、イベント、およびマーカーを作成、検索、削除するように設計されたサーブレット Bean を設定できます。このような機能があるため、カスタム・プロパティよりもマーカーをお薦めします。

オーダー・リポジトリの `marker` および `commerceItemMarker` の各リポジトリ項目タイプには同じプロパティがあり、これはすべて文字列です。

- key
- value
- data

さらに、`order` および `commerceItem` 項目タイプのリポジトリ定義には、マーカーの `set` を格納する `markers` プロパティも含まれています。

`key` プロパティによって、1 つの種類のマーカーは別のマーカーと区別されます。`key` はマーカー・タイプと考えることができ、これはマーカー・データを編成する原則として使用されます。`key` プロパティが、`value` のスーパーセットである場合があります。同様に、`data` プロパティは、`value` のサブセットである場合があります。

マーカーが機能する仕組みを次に示します。

1. マーカー・マネージャを構成します。マーカー・マネージャは、マーカーの作成や削除など、マーカーとオーダーの間、またはマーカーと商品の間におけるすべての相互作用を容易にします。
2. いつ、どのようにマーカーを作成し、オーダーまたは商品に添付するかを定義するコードを作成します。
3. アプリケーションでは、特定のマーカーをチェックし、マーカーに基づいて応答するシナリオまたはサーブレット Bean を使用できます。
4. パフォーマンス向上のため、必要でなくなったマーカーは削除します。

詳細は、『ATG Web Commerce Platform API Reference』の `atg.markers` パッケージを参照してください。

マーカー・マネージャの構成

オーダー・マーカーおよび商品マーカーは、次のコンポーネントによって管理されます。

- /atg/commerce/markers/OrderMarkerManager
- /atg/commerce/markers/CommerceItemRepositoryMarkerManager

アプリケーションで、新しいマーカーの追加、既存のマーカーの削除またはオーダー上のマーカーに関する情報の問合せを行うと、マネージャ・コンポーネントは、マークされた項目（order または commerceItem）、マーカー（markers）を保持する項目のプロパティ、およびマーカー（marker）の役割を果たす項目のタイプを識別して、これらの処理を調整します。

マネージャ・コンポーネントには、オーダーと商品のマーカーを検索、追加および削除するためのメソッドが備えられています。また、API を使用してプロパティを追加することもできます（この使用方法は、購入時ギフト・プロモーションで説明します。『ATG Web Commerce Programming Guide』を参照してください）。マネージャ・コンポーネントは、キー、値およびデータの任意の組合せを使用してマーカーを取得します。

マーカー・マネージャは、マーカーを追加すると、使用するように構成されているマーカーの複製ルールと検証ルールを適用します。これらのコンポーネント設定にはデフォルト値があり、サーブレット Bean で逆の値を指定するとコンポーネント設定を上書きできます。

デフォルトでは、単一のマーカー・マネージャがすべてのマーカー・アクティビティを管理します。特定タイプのマーカーに必要な、様々な検証ルールおよび複製ルールを使用するために、マーカー・マネージャ・コンポーネントを追加作成できます。

マーカー複製モードの設定

マーカー・マネージャは、マーカーが追加されると、作成したマーカーに使用するデフォルト複製モードを指定します。マーカー・マネージャが受け入れる defaultDuplicationMode プロパティの 3 つの値は次のとおりです。

- ALLOW_DUPLICATES
- REPLACE_DUPLICATES
- NO_DUPLICATES

マーカー・マネージャが、指定されたオーダーまたは商品の既存のマーカーと同じマーカーを作成することを要求されると、マーカー・マネージャは新しいマーカーを破棄する（NO_DUPLICATES）か、複製によってオリジナルを置換する（REPLACE_DUPLICATES）か、またはオリジナルに追加して新しいマーカーを受け入れる（ALLOW_DUPLICATES）ことができます。マーカーの数が少ない方が、特定のマーカーに対する問合せの応答は早くなります。

NO_DUPLICATES または REPLACE_DUPLICATES を指定した場合、マーカー・マネージャは比較コンポーネントを使用して、新しいマーカーが既存のマーカーの複製であるかどうかを判定します。defaultMarkerDuplicateComparator プロパティは、デフォルトで/atg/markers/CompareByDefaultProperties に設定されており、この場合、key、value および data のプロパティに同じ値が設定されていると、2 つのマーカーは同一と見なされます。Oracle Commerce Core Commerce には、もう 1 つのコンポーネント（/atg/markers/userprofiling/CompareByKeyAndValue）が含まれています。このコンポーネントは、key プロパティと value プロパティの値のみに基づいて等価性を判定します。

マーカーの一意性の定義

指定したマーカーのプロパティまたは定義したビジネス・ロジックに基づいて、一意性を判定する独自のコンポーネントを作成できます。1 つの方法としては、複数の比較コンポーネントを提供し、それぞれのコンポーネントを(マーカーの key によって識別された)特定タイプのマーカーに必要な一意性の条件に合わせて調整します。

マーカー・プロパティの値に基づいて等価性を判定する比較コンポーネントを作成するには

1. atg.markers.CompareByProperties クラスのコンポーネントを作成します。
2. propertiesTocompare プロパティを、マーカー・プロパティまたは比較して等価性を判定するプロパティに設定します。
3. 複数の比較コンポーネントがある場合は、markerDuplicateComparators マップ・プロパティを更新して、各コンポーネントを使用します。キーをマーカーの key 値に設定し、値を比較コンポーネントに設定します。

定義したビジネス・ロジックに基づいて等価性を判定する比較コンポーネントを作成するには

1. atg.markers.MarkerDuplicateComparator のサブクラスを作成し、そのクラスのコンポーネントを作成します。
2. 複数の比較コンポーネントがある場合は、markerDuplicateComparators マップ・プロパティを更新して、各コンポーネントを使用します。キーをマーカーの key 値に設定し、値を比較コンポーネントに設定します。

マーカー検証の構成

アプリケーションでマーカーを作成する場合、指定する必要があるのは key プロパティ値の情報のみです。必要に応じて、すべてのマーカーの key 値を 6 文字以上とするなど、マーカーの準拠基準をより厳格にできます。あるいは、検証要件を、マーカー・プロパティには関係のない、アプリケーション固有のビジネス・ロジックに基づくようにすることもできます。使用するマーカーに適した検証基準を実現する検証クラスを作成してください。

定義する検証基準は、マーカーのタイプによって異なることがあります。key X を含むマーカーには key および value の値が必要ですが、key Y を含むマーカーには、カスタム・プロパティの expirationDate の日付値が必要となるなどの場合です。各検証コンポーネントは key に対して照合する必要があります。特定のマーカー・マネージャによって作成されたすべてのマーカーに対応する、1 つの検証コンポーネントを作成する方法はありません。

新しい検証標準を定義するには、次の手順を実行します。

1. 検証基準のセットごとにクラスを作成します。このクラスは、atg.markers.MarkerValidatorContainer インタフェースを実装する必要があります。各クラスのコンポーネントを作成します。
2. 定義した各検証コンポーネントを使用するように、マーカー・マネージャの markerValidators マップ・プロパティを更新します。キーをマーカーの key 値に設定し、value を検証コンポーネントに設定します。たとえば、次のようになります。

```
markerValidators=X=/atg/KeyXValidator,Y=/atg/KeyYValidator
```

1. 想定される key ごとに検証コンポーネントを定義した場合のみ、マーカー・マネージャの alwaysValidate プロパティを true に設定します。定義していない場合は false 設定のままにします。このとき、検証コンポーネントにマップされたこれらのマーカーは、いずれにしてもこのコンポーネントを使用します。

2. 移入される key、value および data の他にマーカーのプロパティが検証基準に必要な場合は、マーカー・マネージャ・コンポーネントの requiredExtendedProperties プロパティにそれらのプロパティ名を追加します。

オーダーまたは商品のマーク付け

マーカー機能を使用するには、オーダーまたは商品がマーク付けされる状況を定義する必要があります。エラーなしでフォームが発行されたときに表示される、フォームの成功ページでは、AddMarkerToOrderDroplet または AddMarkerToItemDroplet サーブレット Bean を使用できます。

AddMarkerToOrderDroplet または AddMarkerToItemDroplet サーブレット Bean の詳細は、付録 A: Core Commerce サーブレット Bean を参照してください。

マーク付けされたオーダーまたは商品の使用

オーダーまたは商品に付けたマーカーは、様々な方法で使用できます。オーダーまたは商品に特定のマーカーがあるかどうかに応じてテキストを表示したり、処理を開始する JSP を設計できます。あるいは、マーカー・アクティビティが発生した場合にのみシナリオを進めるようにできます。例については、以降のシナリオを参照してください。これらの例はすべて、オーダーおよび商品のマーカーに適用されます。

サーブレット Bean を使用したオーダー・マーカーまたは商品マーカーの検索

特定のオーダーまたは商品にあるマーカーの検索には、複数のサーブレット Bean を使用できます。これらのサーブレット Bean を使用する場合は、操作するオブジェクトを示す入力パラメータを指定します。指定しなかった場合は、アクティブなオーダーまたは商品が使用されます。

HasMarkerDroplet サーブレット Bean（商品の場合）および OrderHasMarkerDroplet サーブレット Bean（オーダーの場合）を使用してマーカーの key、value または data プロパティ値を指定すると、特定のマーカーをチェックできます。

OrderHasLastMarkerDroplet サーブレット Bean は、指定されたオーダーに添付された最終マーカーを特定します。指定された特定の key、value、data または他のマーカー・プロパティ値が最終マーカーにある場合にのみ、その最終マーカーを返すように指定できます。また、このサーブレット Bean を使用すると、マーカーが存在するかどうかの判定の他、マーカー自体にアクセスすることもできます。

最後に、OrderHasLastMarkerWithKeyDroplet を使用すると、特定の key を持つオーダーに追加された最終マーカーに、一致する値とデータが存在するかどうかを判定できます。

ここで説明したサーブレット Bean の詳細は、付録 A: Core Commerce Servlet Beans を参照してください。

オーダーにマーカーがある場合のシナリオの進行

アクティブなオーダーに特定のマーカーが存在する場合にのみシナリオを進める場合は、シナリオでマーカー条件を使用します。オーダー・マーカーに関連する、3 つの条件を使用できます。

- オーダーにオーダー・マーカーが存在する場合にシナリオが応答するようにする場合は、Order has a marker 条件を使用します。

- `Order's last marker` 条件を使用すると、指定した `key`、`value` および `data` の値がオーダーの最終マーカーに存在する場合にのみ、次のシナリオ要素に進むことを指定できます。
- `Order's last marker with key` という名前の 3 番目の条件を使用すると、ここで指定した `value` および `data` の値を持つ特定の `key` を含む最終マーカーがオーダーに存在する場合にのみ、次のシナリオ要素に進むことを指定できます。

ここで説明した処理の詳細は、『ATG Web Commerce パーソナライゼーション・ガイド』の*シナリオの作成*を参照してください。

マーカー・イベントに基づくシナリオの進行

Core Commerce には、次のイベントを待ち受ける 3 つのシナリオ要素が含まれています。

- オーダーへのマーカーの追加（`Marker Added`）
- オーダーからのマーカーの削除（`Marker Removed`）
- 別のマーカーによる、オーダーのマーカーの置換（`Marker Replaced`）

マーカー・イベントを待ち受けるシナリオを作成できます。

ここで説明したイベントの詳細は、『ATG Web Commerce パーソナライゼーション・ガイド』の「*シナリオの作成*」の*シナリオでのイベント要素の使用*に関する項を参照してください。

マーカーの削除

マーカーがアプリケーションに関連しなくなった場合にマーカーを削除するメカニズムを設定することをお薦めします。マーカーはいくつかの動作をモニターするときに使用できますが、必要なデータを収集した後は、ここで説明するいずれかの方法でオーダーからマーカーをフラッシュしてください。

マーカーは、特定のマーカーを個別に削除するか、すべてのマーカーを一度に削除できます。すべてのオーダーまたは商品からマーカーを同時に削除する場合は、`atg.markers.OrderMarkerManager` クラスまたは `CommerceItemMarkerManager` クラスの `deleteMarkers` メソッドを起動します。詳細は、『ATG Web Commerce Platform API Reference』を参照してください。

オーダーからの特定マーカーの削除

`RemoveMarkersFromOrderDroplet` を使用すると、オーダーから特定のマーカーを削除できます。

```
<dsp:droplet
 name="/atg/markers/droplet/RemoveMarkersFromOrderDroplet">
   <dsp: param name="key" value="partner"/>
   <dsp: param name="value" value="site"/>
   <dsp:param name="data" bean="RemoveMarkersFromOrderDroplet.ANY_VALUE"/>

   <dsp:oparam name="output">
      Your output here.
   </dsp:oparam>

</dsp:droplet>
```

サーブレット Bean を使用してマーカーを削除する場合は、マーカーを削除するオーダーを選択します。デフォルトは現在のセッションのショッピング・カートですが、別のショッピング・カートも指定できます。

RemoveMarkersFromOrderDroplet の詳細は、付録 A: Core Commerce サーブレット Bean を参照してください。

オーダーのすべてのマーカーの削除

RemoveAllMarkersFromItemDroplet を使用すると、オーダーのすべてのマーカーを削除できます。

```
<dsp:droplet
 name="/atg/markers/droplet/RemoveAllMarkersFromOrderDroplet">

  <dsp:param name="output">
     There were <dsp:param name="markerCount"/> markers on your
     order.
  </dsp:param>

</dsp:droplet>
```

これらのサーブレット Bean を使用してマーカーを削除すると、シナリオ処理を使用して削除する場合より、削除する項目を詳細に制御できます。削除されたマーカーの数を確認し、その数をユーザーに表示できます。また、order 入力パラメータを使用して、削除するマーカーが含まれるオーダーを指定したり、デフォルト値を使用することもできます。デフォルト値を使用すると、アクティブ・ユーザーのオーダーのマーカーが削除されます。

詳細は、付録 A: Core Commerce サーブレット Bean を参照してください。

15 ATG Portal ギアの使用

ATG Portal ユーザーであれば、コマース・サイトのポータル・ページでギアを使用して、顧客に自分のオーダー情報へのパーソナライズされたゲートウェイを提供できます。たとえば、オーダー・ステータス・ギアを使用することで、顧客は自分が確定した最新の 5 つのオーダーに直接アクセスでき、さらに 1 回クリックするだけで、すべてのオーダーにもアクセスできます。

この章では、Oracle Commerce Core Commerce に付属しているポータル・ギア（「ポートレット」と呼ばれることもある）について説明します。この章には、次の項があります。

この章では、『ATG Web Commerce Portal Development Guide』および『ATG Portal Administration Guide』で説明しているギアの開発と管理の概念に関する知識を読者が持っていることを前提としています。

注意: ATG Portal ギアは、アプリケーション・サーバーに配置する必要のある Web アプリケーションが含まれた個別のモジュールにパッケージ化されています。

オーダー・ステータス・ギア

オーダー・ステータス・ギアは、現在のオーダー情報およびオーダー履歴情報をポータル・エンド・ユーザーが利用できるようにします。

この項では、オーダー・ステータス・ギアを実行し、使用する方法を説明し、オーダー・ステータス・ギアの実装およびデフォルトの構成について解説します。この項では次のトピックを説明します。

オーダー・ステータス・ギアの設定

オーダー・ステータス・ギアを設定するには

1. アプリケーションのアセンブリ中に、オーダー・データが含まれるモジュールを指定します。また、`CommerceGears.orderstatus` モジュール（ATG Portal が必要）も指定します。たとえば、次のモジュールはオーダー・ステータス・ギアと Motorprise を表します。

   ```
   CommerceGears.orderstatus MotorpriseJSP
   ```

モジュール名のリストとアセンブリの手順については、『ATG Web Commerce Platform Programming Guide』を参照してください。

2. アプリケーション・サーバーの文書に記載されている指示に従ってアプリケーションを配置します。
3. オーダー・ステータス・ギアのギア・マニフェスト・ファイルを Portal Administration Framework（PAF）にアップロードすることによって、オーダー・ステータス・ギアを PAF に登録します。ギア・マニフェストは`<ATG11dir>\CommerceGears\orderstatus\orderstatus-manifest.xml` にあります。

   ギアを登録する方法の詳細は、『ATG Web Commerce Portal Administration Guide』の*ポータルの管理*を参照してください。
4. ギアを使用し、コミュニティのページのいずれかにギアを追加するメンバーを持つコミュニティを作成します。

   コミュニティを作成し、そのコミュニティのページのいずれかにギアを追加する方法の詳細は、『ATG Web Commerce Portal Administration Guide』の*コミュニティ管理*を参照してください。
5. オーダー・ステータス・ギアが含まれているコミュニティ・ページの URL（たとえば、`http://host:port/portal/mycommunity/home`）をブラウザで指定します。

   オーダー・ステータス・ギアを表示するには、まずページにログインする必要があります。デフォルト・ポートについては、『ATG Web Commerce Installation and Configuration Guide』を参照してください。
6. ギアを調べるために、次のユーザーとしてページにログインします。
   - Motorprise のバイヤーStuart Lee。Stuart のユーザー名とパスワードは、`stuart:stuart` です。

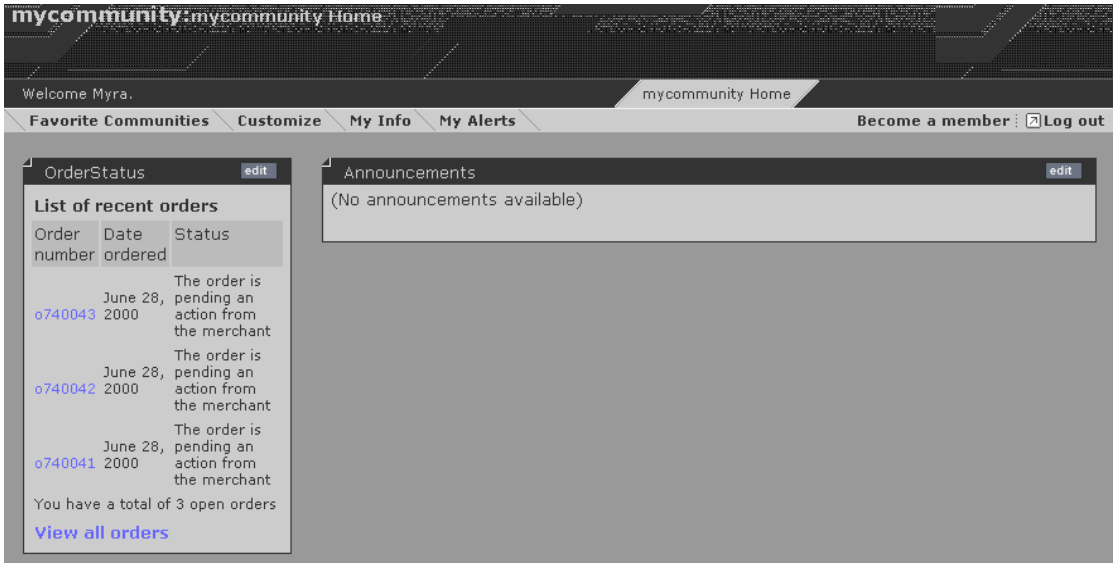
   独自の Commerce アプリケーションでオーダー・ステータス・ギアを実行している場合は、適切なユーザーとしてログインする必要があります。

オーダー・ステータス・ギアの使用

オーダー・ステータス・ギアが含まれているポータル・ページにエンド・ユーザーがログインすると、当初はギアが共有モードで表示されます。デフォルトでは、直前に確定された 5 つのオーダーが、個々のオーダーのオーダー番号、オーダー日および現在のステータスとともにギアに表示されます。さらに、ギアには、ユーザーのオープン・オーダーの合計数も表示されます。このページから、ユーザーは次のいずれかの操作を実行できます。

- 任意のオーダー番号をクリックして、そのオーダーのオーダー詳細ページを表示する操作。
- 「すべてのオーダーの表示」リンクをクリックして、ユーザーのすべてのオーダーを表示する操作。
- ギアの右上隅の「編集」ボタンをクリックして、ギアのユーザー・パラメータを構成する操作。

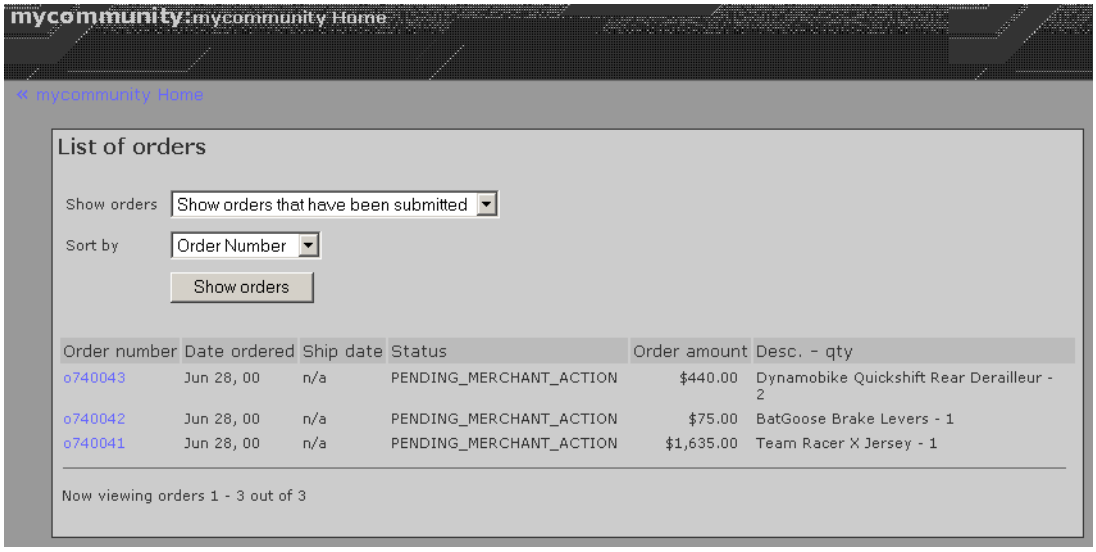
次の図は、共有モードのオーダー・ステータス・ギアを示しています。



オーダー・ステータス・ギアの共有ビュー

ユーザーが「すべてのオーダーの表示」リンクをクリックすると、ポータル・ページが再レンダリングされ、オーダー・ステータス・ギアが全画面モードで表示されます。ギアには、ユーザーのすべてのオーダーが、個々のオーダーのオーダー番号、オーダー日、出荷日（該当する場合）、現在のステータス、金額および摘要とともに表示されます。デフォルトでは、ユーザーのオーダーが 10 個を超える場合は、複数のページにまたがってオーダーが表示されます。

次の図は、全画面モードのオーダー・ステータス・ギアを示しています。



オーダー・ステータス・ギアの全画面ビュー

全画面モードでは、表示するオーダーの状態（たとえば、出荷済や発行済など）をユーザーが指定することによって、オーダーの表示を変更できます。ユーザーは、オーダーのソート基準（オーダー番号、状態、

オーダー日）を指定することもできます。「オーダーの表示」ボタンをクリックすると、指定された基準に従ってオーダーのリストが再表示されます。共有モードと同様に、ユーザーは任意のオーダー番号をクリックして、そのオーダーのオーダー詳細ページを表示できます。

オーダー・ステータス・ギアの構成

オーダー・ステータス・ギアには、ギアの機能と表示をカスタマイズするためのギア構成ページが 3 ページ含まれています。

- ポータル管理用の `installConfig` ページ
- コミュニティ・リーダー用の `instanceConfig` ページ
- ポータル・エンド・ユーザー用の `userConfig` ページ

次の表で、`installConfig` ページで構成できるオーダー・ステータス・ギアのインスタンス・パラメータを説明します。一般的に、これらのパラメータはポータル管理者によって設定されます。

インスタンス・パラメータ	説明	デフォルト値
`OrderPage`	Commerce アプリケーションによって使用されるオーダー詳細ページの URL。このページは、ポータル・エンド・ユーザーがオーダー・ステータス・ギア内のオーダー番号をクリックすると表示されます。 **重要:** デフォルトで、このパラメータは汎用の URL に設定されます。Motorprise ストアでギアを実行している場合は、このパラメータを `/Motorprise/en/user/order.jsp` に変更する必要があります。 独自の Commerce アプリケーションでギアを実行している場合は、このパラメータをアプリケーションに適した JSP に変更する必要があります。JSP が複数ロケールをサポートしていない場合は、必要とされる個々のロケール用に別々のギア・インスタンスを構成する必要があります。	`en/user/order.jsp`

次の表では、`instanceConfig` で構成できるオーダー・ステータス・ギアのインスタンス・パラメータを説明します。一般的に、これらのパラメータはコミュニティ・リーダーによって設定されます。

個々のインスタンス・パラメータには、ポータル・エンド・ユーザーが、全画面モードで、指定されたオーダー状態にあるオーダーのみをフィルタおよび表示できるかどうかを示すブール値が格納されます。パラメータを true に設定すると、関連付けられた状態がフィルタ・ドロップダウン・リストに含まれます。

インスタンス・パラメータ	説明	デフォルト値
`ShowShippedFilterFull`	エンド・ユーザーが出荷済オーダーのみをフィルタおよび表示できるかどうかを示します。	true
`ShowSubmittedFilterFull`	エンド・ユーザーが発行済オーダーのみをフィルタおよび表示できるかどうかを示します。	true

インスタンス・パラメータ	説明	デフォルト値
ShowApprovedFilterFull	エンド・ユーザーが承認済オーダーのみをフィルタおよび表示できるかどうかを示します。	true
ShowRejectedFilterFull	エンド・ユーザーが否認済オーダーのみをフィルタおよび表示できるかどうかを示します。	true
ShowPendingApprovalFilterFull	エンド・ユーザーが承認保留中オーダーのみをフィルタおよび表示できるかどうかを示します。	true
ShowPendingRemoveFull	エンド・ユーザーが削除保留中オーダーのみをフィルタおよび表示できるかどうかを示します。	true
ShowRemovedFull	エンド・ユーザーが削除済オーダーのみをフィルタおよび表示できるかどうかを示します。	true
ShowPendingCustomerActionFull	エンド・ユーザーが顧客処理保留中オーダーのみをフィルタおよび表示できるかどうかを示します。	true
ShowPendingCustomerReturnFull	エンド・ユーザーが顧客返品保留中オーダーのみをフィルタおよび表示できるかどうかを示します。	true

次の表では、ポータル・エンド・ユーザーが userConfig ページで構成できるオーダー・ステータス・ギアのユーザー・パラメータを説明します。

ユーザー・パラメータ	説明	デフォルト値
NumberOfOrdersFull	全画面モードで *1* ページ当たりに表示するオーダーの最大数	10
NumberOfOrdersShared	共有モードで表示されるオーダーの最大数。ゼロに設定すると、最近のオーダー情報は表示されません。	5
ShowOpenOrdersShared	ユーザーが確定したオープン・オーダーの数を共有モードで表示するかどうかを示すブール値。 「オープン」オーダーとは、OrderLookup.openStates で指定されている状態を持つオーダーです（OrderLookup コンポーネントの詳細は、オーダー・ステータス・ギアの実装を参照してください）。	true

オーダー・ステータス・ギアの実装

オーダー・ステータス・ギアは、既存の Portal と Core Commerce の機能を利用します。次の項以降では、オーダー承認ギアの実装の様々な側面を説明します。

ギア・モードと表示モード

次の表は、オーダー・ステータス・ギアで使用されるギア・モードのリスト、およびそれらに対応する表示モード、デバイス出力、ギア・コンテンツおよび構成ページを示しています。

ギア・モード	表示モード	デバイス出力	ページ・フラグメント
content	共有	HTML	OrderStatusShared.jsp
content	全画面	HTML	OrderStatusFull.jsp
installConfig	全画面	HTML	installConfig.jsp
instanceConfig	全画面	HTML	instanceConfig.jsp
userConfig	全画面	HTML	userConfig.jsp

ギア・モード、表示モードおよびデバイス出力に関する一般情報は、『ATG Web Commerce Portal Development Guide』の*ギアの設計*を参照してください。ギア・コンテンツおよび構成ページの作成に関する一般情報については、同じガイドの*ギア・ページ・フラグメントの作成*を参照してください。

コンポーネント

オーダー・ステータス・ギアは次の主なコンポーネントを利用します。

コンポーネント	説明
/atg/portal/gear/GearConfigFormHandler	atg.portal.framework.GearConfigFormHandler クラス。 GearConfigFormHandler は PAF に付属しています。このコンポーネントは、ギアのインスタンスおよびユーザー・パラメータを設定するためのフォームを作成および管理するためにオーダー・ステータス・ギア構成ページで使用されます。 GearConfigFormHandler の詳細は、『ATG Web Commerce Portal Development Guide』の*ギアおよびポータル・アプリケーション・フレームワーク*および*ギア・ページ・フラグメントの作成*を参照してください。
/atg/commerce/states/OrderStates	atg.commerce.states.OrderStates クラス。 OrderStates コンポーネントは、可能性のあるオーダー状態のリストを取得するために instanceConfig.jsp で使用されます。個々の可能性のあるオーダー状態に対して、チェック・ボックスがレンダリングされます。コミュニティ・リーダーがチェック・ボックスを選択すると、ポータル・エンド・ユーザーが全画面モードでオーダーをフィルタするために使用できる状態のリストにチェック・ボックスに関連付けられたオーダー状態が含まれます。 OrderStates の詳細は、『ATG Web Commerce Programming Guide』の「*購買プロセス・オブジェクトの操作*」の *Core Commerce の状態*に関する項を参照してください。

コンポーネント	説明
/atg/commerce/gears/orderstatus/OrderStatusGear	atg.commerce.gears.orderstatus.OrderStatusFormHandler クラス。 OrderStatusGear フォーム・ハンドラは、共有コンテンツ・ページおよび全画面コンテンツ・ページの両方で使用されます。OrderStatusShared.jsp では、このコンポーネントが、コンポーネントの sharedViewStates プロパティで指定されている状態と一致する状態を持つオーダーの表示に使用されます。デフォルトで、OrderStatusGear.sharedViewStates は次の状態に設定されます。 submitted,\ processing,\ pending_merchant_action,\ pending_customer_action,\ no_pending_action OrderStatusFull.jsp では、このコンポーネントが、オーダーを表示するときのオーダー状態とソート基準をエンド・ユーザーが指定するために使用できるフォームの作成と管理に使用されます。
/atg/commerce/order/OrderLookup	atg.commerce.order.OrderLookup クラス。 OrderLookup サーブレット Bean は、ギア・コンテンツ・ページでユーザーのオーダーを取得および表示するために使用されます。 OrderLookup の詳細は、オーダーの取得の実装を参照してください。

タグ・ライブラリ

オーダー・ステータス・ギアは次の標準タグ・ライブラリを使用します。

- コア・タグ・ライブラリ
- DSP タグ・ライブラリ
- PAF タグ・ライブラリ
- Jakarta's i18n タグ・ライブラリ

DSP タグ・ライブラリについては、『ATG Web Commerce Page Developer's Guide』を参照してください。PAF タグ・ライブラリおよび Jakarta's i18n タグ・ライブラリについては、『ATG Web Commerce Portal Development Guide』を参照してください。

このギア用のカスタム・タグ・ライブラリは作成されていません。

オーダー承認ギア

オーダー承認ギアは、現在の承認者の承認を必要とするオーダーを表示し、必要に応じて、そのオーダーを承認するためのメカニズムまたは否認するためのメカニズム、あるいはその両方を提供します。

この項では、オーダー承認ギアを実行し、使用する方法を説明し、オーダー承認ギアの実装およびデフォルトの構成について解説します。この項では次のトピックを説明します。

オーダー承認ギアの設定

オーダー承認ギアにアクセスするには

1. アプリケーションのアセンブリ中に、Motorprise（またはオーダー・データが含まれる独自の Commerce アプリケーション）および `CommerceGears.orderapproval` モジュール（ATG Portal が必要）を指定します。たとえば、次のようになります。

   ```
   CommerceGears.orderapproval MotorpriseJSP
   ```

   モジュールのリストとアセンブリの手順については、『ATG Web Commerce Platform Programming Guide』を参照してください。
2. アプリケーション・サーバーの文書に記載されている指示に従ってアプリケーションを配置します。
3. オーダー承認ギアのギア・マニフェスト・ファイルを Portal Administration Framework（PAF）にアップロードすることによって、オーダー承認ギアを PAF に登録します。ギア・マニフェスト・ファイルは、`<ATG11dir>\CommerceGears\orderapproval\orderapproval-manifest.xml` にあります。

   ギアを登録する方法の詳細は、『ATG Web Commerce Portal Administration Guide』の*ポータルの管理*を参照してください。
4. ギアを使用し、コミュニティのページのいずれかにギアを追加するメンバーを持つコミュニティを作成します。

   コミュニティを作成し、そのコミュニティのページのいずれかにギアを追加する方法の詳細は、『ATG Web Commerce Portal Administration Guide』の*コミュニティ管理*を参照してください。
5. オーダー承認ギアが含まれているコミュニティ・ページの URL（たとえば、`http://hostname:port/portal/mycommunity/home`）をブラウザで指定します。

   オーダー承認ギアを表示するには、まず承認者であるユーザーとしてログインする必要があります。デフォルト・ポートについては、『ATG Web Commerce Installation and Configuration Guide』を参照してください。
6. ギアを調べるために、Motorprise の承認者 Ernesto Hernandez としてページにログインします。Ernesto のユーザー名とパスワードは、`ernesto:ernesto` です。

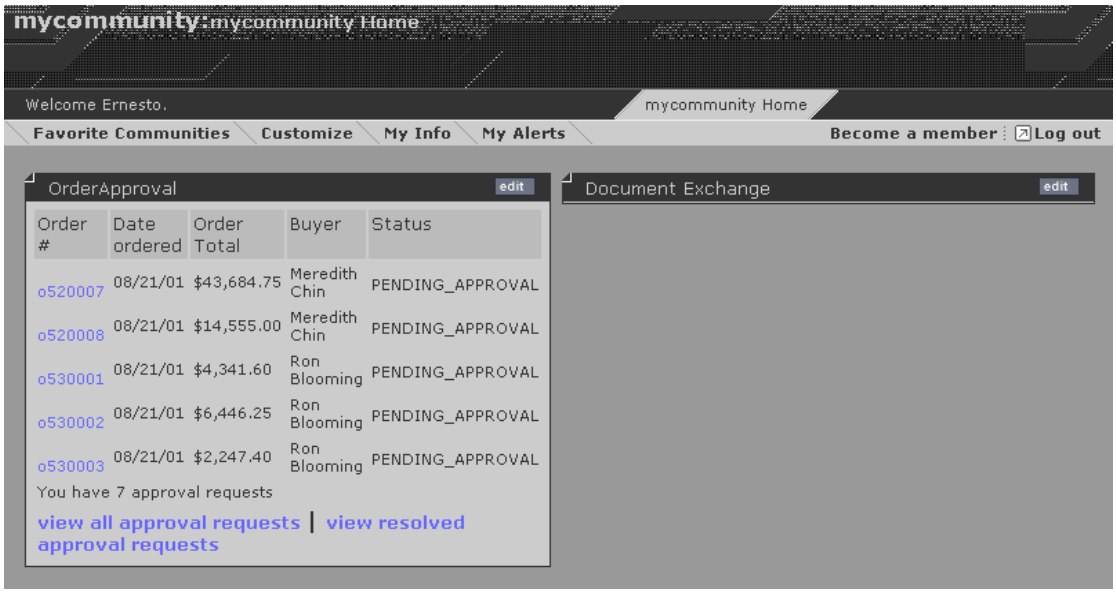
   独自の Commerce アプリケーションでオーダー承認ギアを実行している場合は、適切なユーザーとしてログインする必要があります。

オーダー承認ギアの使用

オーダー承認ギアが含まれているポータル・ページにエンド・ユーザーがログインすると、当初はギアが共有モードで表示されます。デフォルトでは、ユーザーの承認を必要とする、直前に確定された 5 つのオーダーが、個々のオーダーのオーダー番号、オーダー日、合計コスト、バイヤー、現在のステータス（PENDING_APPROVAL）とともにギアに表示されます。さらに、ギアには、ユーザーの承認を必要とするオーダーの合計数も表示されます。このページから、ユーザーは次のいずれかの操作を実行できます。

- 任意のオーダー番号をクリックして、そのオーダーのオーダー詳細ページを表示する操作。
- すべての承認要求の表示(View All Approval Requests)リンクをクリックして、ユーザーの承認を必要とするすべてのオーダーを表示する操作。
- 解決済承認要求の表示(View Resolved Approval Requests)リンクをクリックして、ユーザーがすでに承認または否認しているすべてのオーダーを表示する操作。
- ギアの右上隅の「編集」ボタンをクリックして、ギアのユーザー・パラメータを構成する操作。

次の図は、共有モードのオーダー承認ギアを示しています。



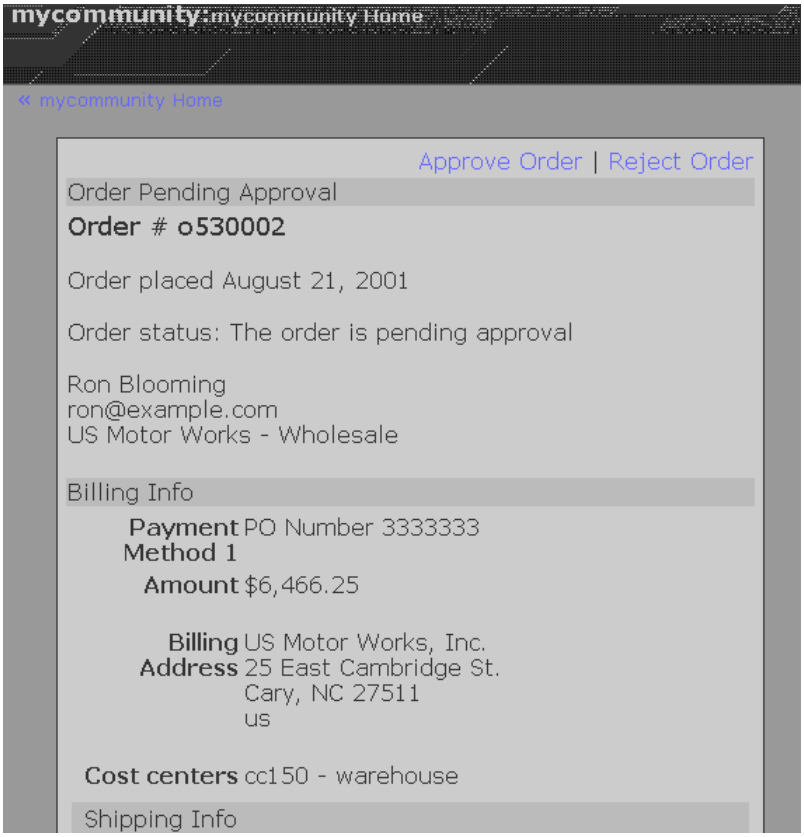
オーダー承認ギアの共有ビュー

ユーザーがすべての承認要求の表示(View All Approval Requests)リンクをクリックすると、ポータル・ページが再レンダリングされ、*オーダー・リスト*・ページが全画面モードで表示されます。デフォルトでは、このページに、ユーザーの承認を必要とするすべてのオーダーが、個々のオーダーのオーダー番号、オーダー日、合計コスト、バイヤー、現在のステータスとともに表示されます。ユーザーの承認を必要とするオーダーが 10 個を超える場合は、複数のページにまたがってオーダーが表示されます。

次の図は*オーダー・リスト*・ページを示しています。

オーダー承認ギアの全画面ビュー(オーダー・リスト・ページが表示されている)

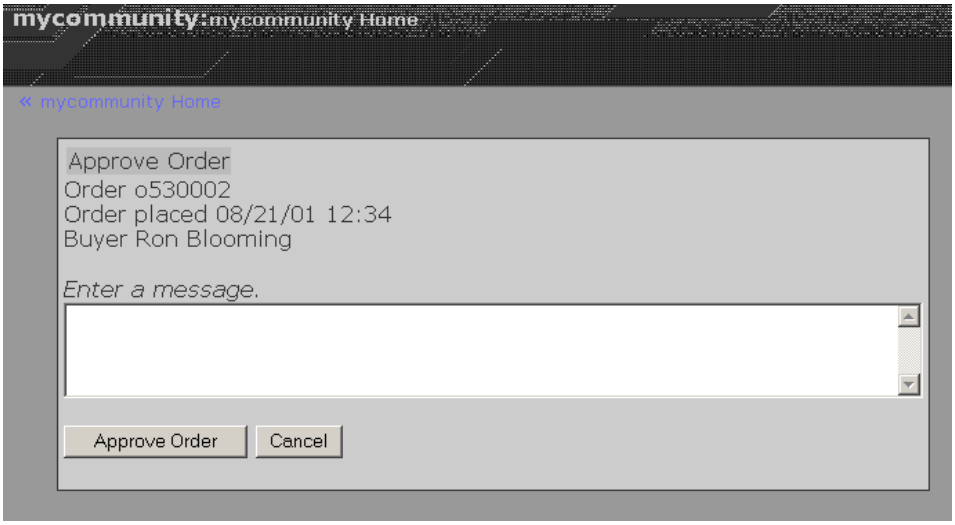
*オーダー・リスト・*ページで、ユーザーは任意のオーダー番号をクリックして、そのオーダーのオーダー詳細ページを表示できます。オーダー承認ギアが実行中の Commerce アプリケーションの承認プロセスを使用するように構成されている場合は、オーダー番号をクリックすると、その Commerce アプリケーションのオーダー詳細ページが表示されます。オーダー承認ギアが独自の承認プロセスを使用するように構成されている場合は、オーダー番号をクリックすると、次の図に示すギアの*オーダー詳細*ページが表示されます。



オーダー承認ギアの全画面ビュー(オーダー詳細ページが表示されている)

デフォルトでは、*オーダー詳細*ページに、基本情報、請求情報および出荷情報が表示されます。承認者は「オーダーの承認」リンクをクリックしてオーダーを承認するか、「オーダーの否認」リンクをクリックしてオーダーを否認できます。

承認者が「オーダーの承認」リンクをクリックすると、ギアには、次の図に示す*オーダーの承認*ページが表示されます。



オーダー承認ギアの全画面ビュー(オーダーの承認ページが表示されている)

*オーダーの承認*ページで、承認者は、承認を決定した理由を入力し、「オーダーの承認」ボタンをクリックすることによって承認を完了できます。

承認者が*オーダー詳細*ページで「オーダーの否認」リンクをクリックすると、*オーダーの否認*ページがレンダリングされることに注意してください。いずれの場合も、承認者がオーダーを承認または否認した後、確認ページで決定を確認することによってプロセスが完了します。

オーダー承認ギアの構成

オーダー承認ギアには、ギアの機能と表示をカスタマイズするためのギア構成ページが 3 ページ含まれています。

- ポータル管理用の `installConfig` ページ
- コミュニティ・リーダー用の `instanceConfig` ページ
- ポータル・エンド・ユーザー用の `userConfig` ページ

次の表で、`installConfig` ページで構成できるオーダー承認ギアのインスタンス・パラメータを説明します。一般的に、これらのパラメータはポータル管理者によって設定されます。

インスタンス・パラメータ	説明	デフォルト値
`UseOrderApprovalOfGear`	ギアのオーダー承認プロセスを使用するかどうかを示します。false に設定されていると、Commerce アプリケーションの承認プロセスが使用されます。	true

インスタンス・パラメータ	説明	デフォルト値
ApprovedOrderPageURL	Commerce アプリケーションが承認されたオーダーおよび否認されたオーダーの表示に使用するオーダー詳細ページの URL。このページは、承認者が *すでに*承認または否認しているオーダーのオーダー番号をクリックすると表示されます。 **重要:** デフォルトで、このパラメータは汎用の URL に設定されます。Motorprise ストアでギアを実行している場合は、このパラメータを /Motorprise/en/user/order.jsp に変更する必要があります。 独自の Commerce アプリケーションでギアを実行している場合は、このパラメータをアプリケーションに適した JSP に変更する必要があります。JSP が複数ロケールをサポートしていない場合は、必要とされる個々のロケール用に別々のギア・インスタンスを構成する必要があります。	checkout/order.jsp
PendingApprovalOrderPageURL	実行中の Commerce アプリケーションが、承認を必要とするオーダーの表示に使用するオーダー詳細ページの URL。 UseOrderApprovalOfGear パラメータが（前述のように）false に設定されていると、承認を必要とするオーダー（つまり、オーダーのステータスが PENDING_APPROVAL）のオーダー番号をポータル・エンド・ユーザーがクリックしたときに、このページが表示されます。 **重要:** デフォルトで、このパラメータは汎用の URL に設定されます。Motorprise ストアでギアを実行している場合は、このパラメータを /Motorprise/en/user/order_pending_approval.jsp に変更する必要があります。 独自の Commerce アプリケーションでギアを実行している場合は、このパラメータをアプリケーションに適した JSP に変更する必要があります。JSP が複数ロケールをサポートしていない場合は、必要とされる個々のロケール用に別々のギア・インスタンスを構成する必要があります。	checkout/order.jsp

次の表で、instanceConfig ページで構成できるオーダー承認ギアのインスタンス・パラメータを説明します。一般的に、これらのパラメータはコミュニティ・リーダーによって設定されます。

これらのインスタンス・パラメータによってギアのオーダー承認プロセスで使用されるコンテンツ・ページの機能と表示が決まることに注意してください。したがって、ギアの承認プロセスが使用されていない場合（つまり、そのかわりに Commerce アプリケーションの承認プロセスが使用されている場合）、これらのパラメータはギアの機能に関連がなく、影響も及ぼしません。

インスタンス・パラメータ	説明	デフォルト値
ShowOrderInfoInDetails	ギアの*オーダー詳細*ページに、オーダーの基本情報を表示するかどうかを示すブール値。	true
ShowBillingInfoInDetails	ギアの*オーダー詳細*ページに、オーダーの請求情報を表示するかどうかを示すブール値。	true
ShowShippingInfoInDetails	ギアの*オーダー詳細*ページに、オーダーの出荷情報を表示するかどうかを示すブール値。	true
ShowOrderInfoInApprove	ギアの*オーダーの承認*ページに、オーダーの基本情報を表示するかどうかを示すブール値。	true
ShowMessageInApprove	ギアの*オーダーの承認*ページにメッセージ・ボックス表示するかどうかを示すブール値。	true
ShowOrderInfoInReject	ギアの*オーダーの否認*ページに、オーダーの基本情報を表示するかどうかを示すブール値。	true
ShowMessageInReject	ギアの*オーダーの否認*ページにメッセージ・ボックス表示するかどうかを示すブール値。	true

次の表では、ポータル・エンド・ユーザーが userConfig ページで構成できるオーダー承認ギアのユーザー・パラメータを説明します。

ユーザー・パラメータ	説明	デフォルト値
NumberOfOrdersShared	共有モードで表示されるオーダーの最大数。ゼロに設定すると、承認を必要とする最近のオーダーは表示されません。	5
ShowPendingApprovalCountShared	承認を必要とするオーダーの合計数を共有モードで表示するかどうかを示すブール値。	true
NumberOfOrdersPerPageFull	全画面モードで *1 ページ当たりに*表示するオーダーの最大数（承認を必要とするオーダーの合計数がこの数を超える場合は、オーダーが複数のページにわたって分割されます）。	10
NumberOfOrdersFull	全画面モードで表示するオーダーの最大合計数。-1 に設定すると、すべてのオーダーが表示されます。	-1

オーダー承認ギアの実装

オーダー承認ギアは、既存の ATG Portal と Core Commerce の機能を利用します。次の項以降では、オーダー承認ギアの実装の様々な側面を説明します。

ギア・モードと表示モード

次の表は、オーダー承認ギアで使用されるギア・モードのリスト、およびそれらに対応する表示モード、デバイス出力および JSP フラグメントを示しています。

ギア・モード	表示モード	デバイス出力	ページ・フラグメント
content	共有	HTML	OrderApprovalShared.jsp
content	全画面	HTML	OrderApprovalFull.jsp
installConfig	全画面	HTML	installConfig.jsp
instanceConfig	全画面	HTML	instanceConfig.jsp
userConfig	全画面	HTML	userConfig.jsp

OrderApprovalFull.jsp には、たとえば orderDetail.jsp や approveOrder.jsp など、オーダー承認ギアに付属の複数の他のページ・フラグメントが含まれていることに注意してください。これらの JSP は <ATG11dir>/CommerceGears/orderapproval/src/orderapproval.war/web/html/ content/にあります。

ギア・モード、表示モードおよびデバイス出力に関する一般情報は、『ATG Web Commerce Portal Development Guide』の*ギアの設計*を参照してください。ギア・コンテンツおよび構成ページの作成に関する一般情報については、同じガイドの*ギア・ページ・フラグメントの作成*を参照してください。

コンポーネント

オーダー承認ギアは次の主なコンポーネントを利用します。

コンポーネント	説明
/atg/portal/gear/GearConfigFormHandler	atg.portal.framework.GearConfigFormHandler クラス。 GearConfigFormHandler は PAF に付属しています。このコンポーネントは、ギアのインスタンスおよびユーザー・パラメータの設定に使用されるフォームを作成するためにオーダー承認ギア構成ページで使用されます。 GearConfigFormHandler の詳細は、『ATG Web Commerce Portal Development Guide』の*ギアおよびポータル・アプリケーション・フレームワーク*および*ギア・ページ・フラグメントの作成*を参照してください。

コンポーネント	説明
/atg/userdirectory/droplet/HasFunction	atg.userdirectory.droplet.HasFunction クラス。 HasFunction サーブレット Bean は、まずログインしたユーザーが承認者がどうかを確認するために、次に、その情報に基づいてページ・コンテンツをレンダリングするために、OrderApprovalShared.jsp と OrderApprovalFull.jsp で使用されます。ユーザーが承認者である場合は、ギア・コンテンツが表示されます。ユーザーが承認者でない場合は、ギアを表示する権限がユーザーにないことを示すメッセージが表示されます。
/atg/commerce/approval/ApprovalRequiredDroplet	atg.commerce.approval.ApprovalRequiredDroplet クラス。 ApprovalRequiredDroplet サーブレット Bean は、現在のユーザーによる承認を必要とするオーダーの取得および表示に使用されます。このサーブレット Bean は、ギアの共有コンテンツ・ページおよび全画面コンテンツ・ページの両方で使用されます。 デフォルトで、ApprovalRequiredDroplet.sortAscending プロパティは true に設定されます。その場合は、承認者の注意を喚起する必要のある最新のオーダーから順に表示されます。最も古いオーダーから順に表示するには、このプロパティを false に設定します。 ApprovalRequiredDroplet の詳細は、オーダー承認プロセスの実装を参照してください。
/atg/commerce/approval/ApprovalFormHandler	atg.commerce.approval.ApprovalFormHandler クラス。 ApprovalFormHandler は、承認者がオーダーを承認または否認するときに使用できるフォームを作成および管理するためにギアの全画面コンテンツ・ページで使用されます。 ApprovalFormHandler の詳細は、オーダー承認プロセスの実装を参照してください。
/atg/commerce/gears/orderapproval/ApprovalResolvedDroplet	atg.commerce.approval.ApprovalRequiredDroplet クラス。 ApprovalResolvedDroplet サーブレット Bean は、現在のユーザーによって承認されたオーダーまたは否認されたオーダー、あるいはその両方の取得および表示に使用されます。このサーブレット Bean は、承認者の「解決済承認要求」を表示する全画面コンテンツ・ページで使用されます。 このオーダー承認ギア・コンポーネントのインスタンス化元である ApprovalRequiredDroplet クラスの詳細は、オーダー承認プロセスの実装を参照してください。
/atg/commerce/order/OrderLookup	atg.commerce.order.OrderLookup クラス。 OrderLookup サーブレット Bean は、特定のオーダーを取得および表示するためにギアの全画面コンテンツ・ページで使用されます。 OrderLookup の詳細は、オーダーの取得の実装を参照してください。

コンポーネント	説明
/atg/commerce/catalog/ProductLookup	atg.commerce.catalog.custom.CatalogItemLookupDroplet クラス。 ProductLookup サーブレット Bean は、承認者のオーダーに対応する製品情報の取得および表示に使用されます。このサーブレット Bean は、*オーダー・リスト・ギア・コンテンツ・ページ*で使用されます。 ProductLookup の詳細は、付録 A: Core Commerce サーブレット Bean の CatalogItemLookupDroplet 参照エントリを参照してください。

タグ・ライブラリ

オーダー承認ギアは次の標準タグ・ライブラリを使用します。

- コア・タグ・ライブラリ
- DSP タグ・ライブラリ
- PAF タグ・ライブラリ
- Jakarta's i18n タグ・ライブラリ

DSP タグ・ライブラリについては、『ATG Web Commerce Page Developer's Guide』を参照してください。PAF タグ・ライブラリおよび Jakarta's i18n タグ・ライブラリについては、『ATG Web Commerce Portal Development Guide』を参照してください。

このギア用のカスタム・タグ・ライブラリは作成されていません。

付録 A: Core Commerce サーブレット Bean

この付録には、Oracle Commerce Core Commerce に付属しているサーブレット Bean の参照エントリが記載されています。サーブレット Bean が機能領域別に分類されているので、互いに関連しているサーブレット Bean を簡単に見つけることができます。次に、詳細な参照エントリを示します。同じクラスから複数のサーブレット Bean がインスタンス化されることがあるため、参照エントリはインスタンス化元のクラスごとに整理され、アルファベット順に記載されていることに注意してください。

サーブレット Bean および JSP でサーブレット Bean を使用する方法に関する一般情報については、『ATG Web Commerce Page Developer's Guide』の *ATG サーブレット Bean の使用*を参照してください。独自のカスタム・サーブレット Bean を作成する方法については、『ATG Web Commerce Platform Programming Guide』の *ATG サーブレット Bean の作成と使用*を参照してください。

AddItemToCartServlet

クラス名	`atg.commerce.order.AddItemToCartServlet`
コンポーネント	`/atg/commerce/order/AddItemToCartServlet`

`AddItemToCartServlet` サーブレット Bean は URL 経由でショッピング・カートに品目を追加します。この機能を利用して、品目の SKU や数量などの製品情報を URL の一部としてサードパーティ・アプリケーションから Core Commerce に渡すことができます。

`AddItemToCartServlet` は、要求処理パイプライン内のサーブレット `CommerceCommandServlet`（`atg.commerce.order.CommerceCommandServlet` クラス）によって呼び出されます。`CommerceCommandServlet` は `addItemToCart` の `dcs_action` 入力パラメータを受け取ると、`AddItemToCartServlet` を呼び出します。

`AddItemToCartServlet` の必須パラメータしか渡されていない場合、このサーブレット Bean は `atg.commerce.order.CommerceItemImpl` クラスの商品を作成し、それをカートに追加します。追加パラメータが渡されている場合、`AddItemToCartServlet` は、渡されたオプション・パラメータに応じて異なる機能を果たします。

`CommerceCommandServlet` および `AddItemToCartServlet` の実装と動作の詳細は、『ATG Web Commerce Programming Guide』の「*購買プロセス・オブジェクトの操作*」の *品目の URL を経由したオーダーへの追加*に関する項を参照してください。

入力パラメータ

url_catalog_ref_id（必須）
カートに追加される品目の SKU。

url_product_id（必須）
カートに追加される品目の製品 ID。

url_quantity（必須）
カートに追加される品目の数量。

url_shipping_group_id
品目の追加先となる出荷グループの ID。

url_item_type
商品の作成に使用される商品タイプ。

url_commerce_item_id
オーダーから削除される商品の ID。カート内の品目を別の品目に置き換える場合に、このパラメータを使用します。

dcs_ci_*
`CommerceItem` プロパティを設定するための識別子。

このプリフィクスが付いたパラメータが渡されると、パラメータが参照している `CommerceItem` プロパティは、商品内の指定された値に設定されます。たとえば、`dcs_ci_catalogKey` パラメータが指定値 `en_US` とともに渡された場合、`catalogKey` プロパティは `en_US` に設定されます。

dcs_conf_<n>
`ConfigurableCommerceItem` の構成可能プロパティを設定するための識別子。

メモリーとハードディスク・ドライブは、コンピュータ品目の構成可能プロパティの例です。つまり、顧客は 1 台のコンピュータとともに、様々な容量のメモリーと 1 台以上のハードディスク・ドライブを注文できます。メモリーとハードディスク・ドライブの両方がコンピュータ品目という基本 SKU のサブ SKU として表されます。

このプリフィクスが付いたパラメータが渡されると、パラメータが参照している構成可能プロパティが指定された値に設定されます。このパラメータの形式を次に示します。

```
dcs_conf_<n>=<sku id, product id, individual quantity>
```

<n>は、特定の `ConfigurableCommerceItem` の構成可能プロパティのリスト内の構成可能プロパティを表す整数です。“sku id”、“product id”、“individual quantity”は、特定の `ConfigurableCommerceItem` の構成可能プロパティを表す `SubSkuCommerceItem` の作成に使用されるデータです。“sku id”は、特定のサブ SKU の SKU ID です。“product id”は、特定のサブ SKU の製品 ID です。“individual quantity”は、*1* つの `ConfigurableCommerceItem` に追加される特定のサブ SKU の数量です。たとえば、1,000 台のコンピュータのそれぞれに 2 台のハードディスク・ドライブを追加する場合、ハードディスク・ドライブ・サブ SKU の individual quantity は 2 です。

dcs_subsku
サブ SKU の識別子。構成可能 SKU およびそのサブ SKU をカートに追加するときに、このパラメータを指定します。このパラメータの値は次の形式に従う必要があります。

```
SKU_ID,product_ID, quantity,
```

複数のサブ SKU を指定するには、個々のサブ SKU に関する情報をカンマで区切った文字列を作成します。たとえば、次のようになります。

```
79054,12159,1,79303,11900,4,90931,20133,2
```

出力パラメータ

なし。AddItemToCartServlet は、要求処理パイプライン内のサーブレット CommerceCommandServlet によって呼び出されます。

オープン・パラメータ

なし。AddItemToCartServlet は、要求処理パイプライン内のサーブレット CommerceCommandServlet によって呼び出されます。

例

JSP の例はありません。AddItemToCartServlet は、要求処理パイプライン内のサーブレット CommerceCommandServlet によって呼び出されます。

AddBusinessProcessStage

クラス名	atg.markers.bp.droplet.AddBusinessProcessStage
コンポーネント	/atg/commerce/bp/droplet/AddShoppingProcessStageDroplet

このサーブレット Bean は、ショッピング・プロセスの新しいステージに達するとオーダーをマークします。このサーブレット Bean は、businessProcessName プロパティが ShoppingProcess に設定されている AddBusinessProcessStage のインスタンスです。

入力パラメータ

businessProcessName
ビジネス・プロセスの名前。このパラメータが指定されていない場合は、デフォルトで ShoppingProcess になるサーブレット Bean の defaultBusinessProcessName プロパティの値が使用されます。

businessProcessStage（必須）
ビジネス・プロセス内のステージ。

出力パラメータ

errorMsg
失敗を記述するエラー・メッセージ。

オープン・パラメータ

output
正常終了するとレンダリングされます。

error
エラーが発生するとレンダリングされます。

例

```
<dsp:droplet name="AddShoppingProcessStageDroplet">
  <dsp:param name="businessProcessStage" value="ShippingPriceViewed"/>
</dsp:droplet>
```

AddMarkerToOrder

マーカー・マネージャと通信してマーカーを作成し、指定したマーカー情報とオーダー情報に基づいて、適切なオーダーにマーカーを添付します。

クラス名	atg.commerce.markers.AddMarkerToOrder
コンポーネント	/atg/commerce/markers/droplet/AddMarkerToOrderDroplet

入力パラメータ

key（必須）
マーカーの key プロパティに保存する値。key プロパティには、グループ化されたマーカーの類似性を表す文字列が保持されます。たとえば、key によって、マーカーのタイプ、またはマーカーが割り当てられる状況が表される場合があります。

order
マーカーの割当先となるオーダー。省略した場合は、アクティブなオーダーが使用されます。

value
マーカーの value プロパティに保存される値。value プロパティには、key プロパティに関連する文字列が保持されます。

data
マーカーの data プロパティに保存される値。data プロパティには、key プロパティおよび value プロパティに関連する文字列が保持されます。

extendedProperties
新しいマーカーに保存されるマーカー・プロパティ（key、value および data 以外）のプロパティ値を指定するマップ。

duplicationMode
マーカー・マネージャによって使用される複製ポリシー。オプションには、ALLOW_DUPLICATES、REPLACE_DUPLICATES および NO_DUPLICATES があります。マーカーが付けられているオーダーに、マーカー・マネージャが同じマーカーを追加しようとした場合、REPLACE_DUPLACES モードでは、既存のマーカーが新しいマーカーで置換され、NO_DUPLICATES モードでは、既存のマーカーが維持され、新しいマーカーが破棄されます。このパラメータを省略した場合、サーブレット Bean はその defaultDuplicationMode プロパティに指定されているデフォルト・モードを使用します。

markerPropertyName
このサーブレット Bean によって作成されたマーカーを格納するオーダーのプロパティ。省略した場合、サーブレット Bean はその defaultMarkerPropertyName プロパティに指定されているデフォルトを使用します。

markerItemType
マーカーが使用する RepositoryItem のタイプ。省略した場合、サーブレット Bean はその defaultMarkerItemType プロパティに指定されているデフォルトを使用します。

markedItemType
マーカーを受け入れる RepositoryItem のタイプ。省略した場合、サーブレット Bean はその defaultMarkedItemType プロパティに指定されているデフォルトを使用します。

出力パラメータ

marker
サーブレット Bean によって作成されたオーダー・マーカーが格納されます。

errorMsg
サーブレット Bean の実行時に生成されたエラー・メッセージが格納されます。

オープン・パラメータ

output
マーカーがオーダーに追加された場合にレンダリングされます。

error
マーカーを作成してオーダーに添付する必要があるが、いずれかまたは両方の処理が正常に完了していない場合にレンダリングされます。

ApprovalRequiredDroplet

クラス名	atg.commerce.approval.ApprovalRequiredDroplet
コンポーネント	/atg/commerce/approval/ApprovalRequiredDroplet /atg/commerce/gears/orderapproval/ApprovalResolvedDroplet (オーダー承認ポータル・ギアのみ)

ApprovalRequiredDroplet サーブレット Bean は、特定の承認者による承認を必要とするすべてのオーダーを取得することによって、オーダー承認プロセスをサポートします。このサーブレット Bean は、オーダー・リポジトリに対して問合せを実行し、次の基準を満たすすべてのオーダーを返します。

- オーダーの authorizedApproverIds プロパティに承認者の ID が含まれていること。
- オーダーの状態が承認を必要としていること。つまり、状態が ApprovalRequiredDroplet の orderStatesRequiringApproval プロパティで定義されていること。オーダーの状態は、ApprovalRequireDroplet の orderStatePropertyName プロパティで指定されているオーダーのプロパティによって保持されます。デフォルト値は PENDING_APPROVAL です。
- オーダーのサイトが siteID と一致するか、指定された siteScope 内にあること。

ApprovalRequiredDroplet サーブレット Bean は、承認者である現在のユーザーに、自分がオーダーを承認する権限を持っている顧客のオーダーしか表示されないようにするセキュリティ機能を備えています。この機能はデフォルトで使用可能になっています。この機能を使用不可にするには、enableSecurity プロパティを false に設定します。

入力パラメータ

approverid（必須）
現在のユーザー、つまり承認者のユーザー・プロファイルの ID。

numOrders
問合せに対して返すオーダーの数。このパラメータはオプションであり、一般的に、大規模な結果セットを管理しやすい単位に分割するために使用されます。

siteIds

指定されたサイトに関連付けられたオーダーに問合せを限定するために使用されるサイト ID のコレクション。`siteIds` が指定されていれば、`siteScope` は無視されます。

siteScope

Oracle Commerce Platform の複数サイト機能を使用している場合は、オーダーをサイトでフィルタできます。`siteScope` を `siteIds` のかわりに使用し、次のいずれかのスコープを指定します。

- `null` または `all`: すべてのサイトのオーダーを検索します。
- `current`:サイト・コンテキストによって決まる現在のサイトのオーダーを検索します。
- `shareableTypeId`: `atg.ShoppingCart` など、共有可能タイプの ID を渡して、そのタイプの共有グループ内のすべてのサイトのオーダーを検索します。

デフォルトの `siteScope` は、コンポーネントの構成可能プロパティを介して設定できます。デフォルト値は `all` です。

startIndex
返す最初のオーダーの索引。`startIndex` が null の場合、このパラメータはデフォルトで 0 になります。このパラメータはオプションであり、一般的に、大規模な結果セットを管理しやすい単位に分割するために使用されます。

出力パラメータ

result
`Order` オブジェクトの配列。

count
`result` 出力パラメータ内の `Order` オブジェクトの数。

totalCount
基準を満たした `Order` オブジェクトの合計数。

nextIndex
次の結果セット内の最初のオーダーの索引。`startIndex` または `numOrders` が null だった場合は、このパラメータも null になります。

previousIndex
前の結果セット内の最初のオーダーの索引。`startIndex` または `numOrders` が null だった場合は、このパラメータも null になります。

`nextIndex` と `previousIndex` を使用すると、ユーザーは結果セット間を行き来できます。

startRange
結果セット内の最初の `Order` の 1 ベースの索引。

endRange
結果セット内の最後の `Order` の 1 ベースの索引。

errorMessage
エラーが発生したときにユーザーに表示するエラー・メッセージ。

オープン・パラメータ

output
このオープン・パラメータは、結果出力パラメータで設定されているオーダー・オブジェクトの配列をレンダリングします。

empty
このパラメータは、現在のユーザーによる承認を必要とするオーダーがない場合にレンダリングされます。

error
このパラメータは、エラーが発生するとレンダリングされます。

例

次の例は、承認者である現在のユーザーによる承認を必要とするオーダーをリポジトリから取得し、個々のオーダーのリポジトリ ID をページに表示します。

```
<dsp:droplet name="ApprovalRequiredDroplet">
 <dsp:param bean="/atg/userprofiling/Profile.repositoryId"
name="approverid"/>
 <dsp:param value="0" name="startIndex"/>
 <dsp:param value="10" name="numOrders"/>
 <dsp:oparam name="output">
<dsp:droplet name="ForEach">
   <dsp:param param="result" name="array"/>
   <dsp:setvalue param="order" paramvalue="element"/>
       <dsp:oparam name="output">
          <dsp:valueof param="order.repositoryId"/><br>
       </dsp:oparam>
       <dsp:oparam name="error">
          <dsp:valueof param="errorMsg"/><br>
       </dsp:oparam>
    </dsp:droplet>
 </dsp:oparam>
</dsp:droplet>
```

ApprovedDroplet

クラス名	atg.commerce.approval.ApprovedDroplet
コンポーネント	/atg/commerce/approval/ApprovedDroplet

ApprovedDroplet サーブレット Bean は、承認または否認されたすべてのオーダー、および approverid パラメータ内のプロファイル ID が承認者 IDリストに含まれるすべてのオーダーを取得することによって、

オーダー承認プロセスをサポートします。オーダーの現在の状態は考慮されず、承認者がオーダーとやり取りしたかどうかのみが考慮されます。

ApprovedDroplet サーブレット Bean は、承認者である現在のユーザーに、自分がオーダーを承認する権限を持っている顧客のオーダーしか表示されないようにするセキュリティ機能を備えていることに注意してください。この機能はデフォルトで使用可能になっています。この機能を使用不可にするには、enableSecurity プロパティを false に設定します。

入力パラメータ

approverid（必須）
現在のユーザー、つまり承認者のユーザー・プロファイルの ID。

numOrders
問合せに対して返すオーダーの数。このパラメータはオプションであり、一般的に、大規模な結果セットを管理しやすい単位に分割するために使用されます。

startIndex
返す最初のオーダーの索引。startIndex が null の場合、このパラメータはデフォルトで 0 になります。このパラメータはオプションであり、一般的に、大規模な結果セットを管理しやすい単位に分割するために使用されます。

出力パラメータ

result
Order オブジェクトの配列。

count
result 出力パラメータ内の Order オブジェクトの数。

totalCount
基準を満たした Order オブジェクトの合計数。

nextIndex
次の結果セット内の最初のオーダーの索引。startIndex または numOrders が null だった場合は、このパラメータも null になります。

previousIndex
前の結果セット内の最初のオーダーの索引。startIndex または numOrders が null だった場合は、このパラメータも null になります。

nextIndex と previousIndex を使用すると、ユーザーは結果セット間を行き来できます。

startRange
結果セット内の最初の Order の 1 ベースの索引。

endRange
結果セット内の最後の Order の 1 ベースの索引。

errorMessage
エラーが発生したときにユーザーに表示するエラー・メッセージ。

オープン・パラメータ

output
このオープン・パラメータは、result 出力パラメータで設定されている Order オブジェクトの配列をレンダリングします。

empty
現在のユーザーによって承認されたオーダーまたは否認されたオーダー、あるいはその両方がない場合にレンダリングされるオープン・パラメータ。

error
エラーが発生するとレンダリングされるオープン・パラメータ。

例

次の例は、承認者である現在のユーザーによって承認されたオーダーまたは否認されたオーダーあるいはその両方をリポジトリから取得し、個々のオーダーのリポジトリ ID をページに表示します。

```
<dsp:droplet name="ApprovedDroplet">
 <dsp:param bean="/atg/userprofiling/Profile.repositoryId" name="approverid"/>
 <dsp:param value="0" name="startIndex"/>
 <dsp:param value="10" name="numOrders"/>
 <dsp:oparam name="output">
  <dsp:droplet name="ForEach">
    <dsp:param param="result" name="array"/>
    <dsp:param value="order" name="elementName"/>
    <dsp:oparam name="output">
     <dsp:valueof param="order.repositoryId"/><br>
    </dsp:oparam>
    <dsp:oparam name="error">
     <dsp:valueof param="errorMsg"/><br>
    </dsp:oparam>
  </dsp:droplet>
 </dsp:oparam>
</dsp:droplet>
```

AvailableShippingMethodsDroplet

クラス名	atg.commerce.pricing.AvailableShippingMethodsDroplet
コンポーネント	/atg/commerce/pricing/AvailableShippingMethods

AvailableShippingMethods サーブレット Bean は、特定の出荷グループの利用可能な出荷方法を表示します。このクラスのサービス・メソッドは、ShippingPricingEngine の getAvailableMethods メソッドを呼び出して、出荷方法を表す文字列のリストを返します。それらの出荷方法は、HardgoodShippingGroup オーダー・クラスの shippingMethod プロパティに対応しています。

入力パラメータ

shippingGroup（必須）
出荷される ShippingGroup。

pricingModels
出荷価格設定モデルのコレクション。このパラメータが null である場合は、セッション限定の PricingModelHolder が解決され、コレクションが取得されます。PricingModelHolder コンポーネントへのパスは、AvailableShippingMethodsDroplet の userPricingModelsPath プロパティを介して構成されます。

profile
出荷方法を要求している顧客を表す RepositoryItem。このパラメータが null である場合は、セッション限定の Profile が解決されます。Profile コンポーネントへのパスは、AvailableShippingMethodsDroplet の profilePath プロパティを介して構成されます。

locale
出荷方法を要求している顧客のロケール。このパラメータは、java.util.Locale オブジェクトまたはロケールを表す文字列のどちらかです。このパラメータが見つからない場合は、デフォルトで、要求からロケールが取得されます。このロケールを特定できない場合は、このコンポーネントのデフォルトのロケールが使用されます。

出力パラメータ

availableShippingMethods
HardgoodShippingGroup 内の shippingMethod 値の設定に使用できる出荷方法を表す文字列のリスト。

オープン・パラメータ

output
availableShippingMethods パラメータが含まれている oparam。

例

次の例では、AvailableShippingMethods サーブレット Bean を使用して、最初の出荷グループの shippingMethod プロパティにバインドされている利用可能な出荷方法の選択ボックスを表示します。

```
<dsp:droplet name="/atg/commerce/pricing/AvailableShippingMethods">
<dsp:param bean="ShoppingCartModifier.shippingGroup" name="shippingGroup"/>
<dsp:oparam name="output">
 <dsp:select bean="ShoppingCartModifier.shippingGroup.shippingMethod">
 <dsp:droplet name="ForEach">
 <dsp:param param="availableShippingMethods" name="array"/>
 <dsp:param value="method" name="elementName"/>
 <dsp:oparam name="output">
   <dsp:getvalueof id="option16" param="method" idtype="java.lang.String">
<dsp:option value="<%=option16%>"/>
</dsp:getvalueof><dsp:valueof param="method"/>
 </dsp:oparam>
 </dsp:droplet>
 </dsp:select>
</dsp:oparam>
</dsp:droplet>
```

AvailableStoreCredits

クラス名	atg.commerce.claimable.AvailableStoreCredits
コンポーネント	/atg/commerce/claimable/AvailableStoreCredits

AvailableStoreCredits サーブレット Bean を使用して、特定のユーザーのプロファイルに関連付けられたストア・クレジットにアクセスできます。その情報をページ・コードで利用できます。

入力パラメータ

profile（必須）
ストア・クレジット情報を表示したい顧客を表す RepositoryItem。

出力パラメータ

storeCredits
提供されているユーザー・プロファイルに関連付けられたストア・クレジットのリスト。

overallAvailableAmount
利用可能なストア・クレジット金額の合計。

オープン・パラメータ

output
常に表示されます。

empty
現在のユーザーにストア・クレジットがないとレンダリングされるオープン・パラメータ。

例

次の例では、AvailableStoreCredits サーブレット Bean を使用して、ユーザーのストア・クレジットに関する情報を提供します。

```
<dsp:droplet name="AvailableStoreCredits">
   <dsp:param name="profile" bean="Profile"/>
   <dsp:oparam name="output">

     <dsp:getvalueof var="onlineCredits" vartype="java.lang.Object"
param="storeCredits"/>
     <c:if test="${not empty onlineCredits}">
       <div id="atg_store_onlineCredits">
         <h3><fmt:message key="myaccount_onlineCredits.savedOnlineCredits"/></h3>
         <c:forEach var="onlineCredit" items="${onlineCredits}"
varStatus="onlineCreditStatus">
           <dsp:setvalue param="storeCredit" value="${onlineCredit}"/>
           <dsp:getvalueof var="storeCredit" param="storeCredit"/>
```

```
          <c:if test="${not empty storeCredit}">
            <div class="atg_store_onlineCreditsDetails">
              <dsp:getvalueof var="count" vartype="java.lang.Double"
value="${onlineCreditStatus.count}"/>
              <h4>
                <fmt:message key="common.credit"/><fmt:message
key="common.numberSymbol"/>
                <fmt:formatNumber value="${count}" type="number"/>
              </h4>
              <div>
                <fmt:message
key="myaccount_onlineCredits.remainingCredit"/><fmt:message
key="common.labelSeparator"/>
              </div>
              <dsp:getvalueof var="amountRemaining" vartype="java.lang.Double"
param="storeCredit.amountAvailable"/>
              <dsp:getvalueof var="currencyCode" vartype="java.lang.String"
param="currencyCode"/>
              <div class="atg_store_onlineCreditTotal">
                <fmt:formatNumber value="${amountRemaining}" type="currency"
currencyCode="${currencyCode}" />
              </div>
            </div>
          </c:if>
        </c:forEach>
      </div>
    </c:if>
  </dsp:oparam>
</dsp:droplet>
```

CatalogItemLookupDroplet

クラス名	atg.commerce.catalog.custom.CatalogItemLookupDroplet
コンポーネント	/atg/commerce/catalog/CategoryLookup /atg/commerce/catalog/ProductLookup /atg/commerce/catalog/SKULookup /atg/commerce/promotion/PromotionLookup /atg/commerce/promotion/PromotionalContentLookup

CatalogItemLookupDroplet クラスからインスタンス化されるサーブレット Bean は、RepositoryItem の ID を使用してリポジトリ内の項目を検索します。項目が見つかると、サーブレット Bean は、その項目がユーザーの現在のプロファイル内のユーザーのカタログに属しているかどうかを確認します。カタログが catalog 入力パラメータを介して指定されている場合、サーブレット Bean は、項目が指定されたカタログに

属しているかどうかを確認します。CatalogItemLookupDroplet ではまた、その項目が現在のサイト・コンテキストに属しているかどうかも確認します。両方の条件が満たされている場合、output オープン・パラメータがレンダリングされます。項目がカタログにない場合、wrongCatalog オープン・パラメータがレンダリングされます。項目が現在のサイト・コンテキストにない場合、wrongSite オープン・パラメータがレンダリングされます。

項目が現在のサイト・コンテキストに属しているかどうかを判断するロジックは、CatalogItemLookupDroplet の親クラス atg.repository.servlet.ItemLookupDroplet に含まれています。このクラスの詳細は、『ATG Web Commerce Page Developer's Guide』の *付録 B: ATG サーブレット Bean* の ItemLookupDroplet エントリを参照してください。

プロパティを介して、特定の項目を検索するときに使用するリポジトリと項目記述子タイプを構成したり、キーから代替リポジトリ・セットへのマッピングを定義したりできます。たとえば、次のプロパティを持つ ProductLookup サーブレット Bean があるとします。

```
$class=atg.commerce.catalog.custom.CatalogItemLookupDroplet
repository=/atg/commerce/catalog/ProductCatalog
itemDescriptor=product
alternateRepositories=\
      fr_FR=/atg/commerce/catalog/FrenchProductCatalog
      ja_JP=/atg/commerce/catalog/JapaneseProductCatalog
      de_DE=/atg/commerce/catalog/GermanProductCatalog
```

この場合、useParams プロパティが true であれば、入力パラメータを介してリポジトリと項目記述子を解決できます。

入力パラメータ

id（必須）
検索の対象となる項目の ID。

catalog
検索の対象となる項目が属するカタログ。カタログは RepositoryItem である必要があります。

このパラメータは、通常、必要とされないことに注意してください。このパラメータが設定されていない場合、サーブレット Bean は、ユーザーのプロファイル内のカタログで項目の有無を確認します。ただし、ユーザーの現在のカタログ以外のカタログで項目の有無を確認する場合、または現在のセッションがない場合（たとえば、E メール・テンプレートのコンテキスト内にある場合）は、このパラメータを介して明示的にカタログを指定できます。

elementName
指定されると、この名前が output オープン・パラメータ内で設定されるパラメータに使用されます。

itemDescriptor 項目をロードするために使用する項目記述子の名前。
（**注意:** このパラメータを使用することはお薦めできません。サーブレット Bean の itemDescriptor プロパティを介して項目記述子を指定する方法をお薦めします）。

repository 検索の対象となる項目が属するリポジトリ。
（**注意:** このパラメータを使用することはお薦めできません。使用するリポジトリのセットごとに異なる CatalogItemLookupDroplet のインスタンスを定義する方法をお薦めします）。

repositoryKey
指定されていると、このパラメータは、セカンダリ・リポジトリ・セットへのマッピングのキーとして使用されます。

出力パラメータ

element
指定されている ID に対応する RepositoryItem（elementName 入力パラメータを設定することによって、このパラメータ名を変更できます）。

error
項目の検索中にエラーが発生するとレンダリングされるオープン・パラメータ。

オープン・パラメータ

output
項目がリポジトリ内で見つかり、現在のユーザーのカタログ（指定されている場合は、catalog 入力パラメータを介して渡されたカタログ）に属していて、かつ現在のサイト・コンテキストに属しているとレンダリングされるオープン・パラメータ。

wrongCatalog
項目がリポジトリ内で見つかったものの、現在のユーザーのカタログ（指定されている場合は、catalog 入力パラメータを介して渡されたカタログ）に属していないとレンダリングされるオープン・パラメータ。このパラメータを利用すれば、必要な場合に項目にアクセスできます。

wrongSite
項目がリポジトリ内で見つかったものの、現在のサイト・コンテキストに属していないとレンダリングされるオープン・パラメータ。このパラメータを利用すれば、必要な場合に項目にアクセスできます。

empty
項目がリポジトリで見つからない場合やユーザーが必須パラメータを指定しなかった場合など、他のすべての場合にレンダリングされるオープン・パラメータ。

noCatalog
ドロップレットがユーザーに関連付けられたカタログを特定できないと、このオープン・パラメータがレンダリングされます。明示的なカタログ・パラメータなしにドロップレットを呼び出した場合で、かつユーザー・プロファイル内にカタログがなかったときに、その状態になります。

例

```
<dsp:droplet name="ProductLookup">
  <dsp:param param="productId" name="id"/>
  <dsp:param bean="/OriginatingRequest.requestLocale.localeString"
             name="repositoryKey"/>
  <dsp:param value="product" name="elementName"/>
  <dsp:oparam name="output">
   <dsp:getvalueof id="a10" param="product.template.url"
               idtype="java.lang.String">
<dsp:a href="<%=a10%>">
     <dsp:param param="product.repositoryId" name="prod_id"/>
     <dsp:valueof param="product.displayName"/>
   </dsp:a></dsp:getvalueof>
  </dsp:oparam>
</dsp:droplet>
```

渡されているキーのマッピングが見つかると、そのリポジトリが使用されます。それ以外の場合、システムはデフォルトのリポジトリを使用して項目を検索します。

検索対象リポジトリ（たとえばフランス語の製品カタログ）内で項目が見つからない場合、サーブレット Bean は再びデフォルトのリポジトリを使用し、同じ ID を使用して項目の検索を試みます。この方式は、項目が個々のリポジトリで同じ ID を持っている場合にのみ役立ちます。たとえば、フランス語のサイトが表示されている状態で、製品 1234 の表示を試みているとします。フランス語の製品カタログで ID が定義されていれば、その製品のフランス語バージョンが表示されます。しかし、1234 がフランス語の製品カタログで定義されていない場合は、製品 1234 のデフォルトの英語バージョンが表示されます。

useDefaultRepository プロパティと getDefaultItem プロパティを使用することで、この動作を変更できます。useDefaultRepository が false で、代替リポジトリが見つからない場合、リポジトリは検索されず、empty オープン・パラメータがレンダリングされます。同様に、代替リポジトリは選択されているが、項目が見つからない場合、かつ、getDefaultItem が false の場合は、empty オープン・パラメータがレンダリングされます。

CatalogPossibleValues

クラス名	atg.commerce.catalog.custom.CatalogPossibleValues
コンポーネント	/atg/commerce/catalog/RepositoryValues

CatalogPossibleValues サーブレット Bean は、顧客が自分の表示できるカタログから返される製品しか検索できないようにします。

入力パラメータ

itemDescriptorName（必須）
リポジトリ XML ファイル内の項目記述子の名前。

catalog
使用された場合にカタログ ID を指定するオプション値。ここで指定されたカタログ内に存在する項目のみが検索によって返されます。カタログを指定しない場合は、ユーザーのプロファイルで指定されているカタログが使用されます。

propertyName
使用された場合に、リンクしているプロパティを参照する必要のあるオプション値。このパラメータが指定されていると、リンク・タイプのリポジトリ項目が返されます。

repository
このパラメータは検索対象となるリポジトリを定義します。プロパティ・ファイルを介してこのパラメータを設定することもできます。

sortProperties
リポジトリ項目のリストをソートする方法を指定する文字列。このパラメータは、カンマで区切られたプロパティ名のリストとして指定されます。最初の名前が最優先のソート基準、2 番目の名前が 2 番目に優先するソート基準、それ以降も同様に、名前の順番にソート基準の優先順位が決まります。キーワードの最初の文字が「-」である場合、そのソートは降順で実行されます。キーワードの最初の文字が「+」であるか、「-」以外の文字である場合、リストは昇順でソートされます。**注意:** このパラメータはリポジトリ項目に対してのみ有効です。列挙データ型に対しては機能しません。

出力パラメータ

values
出力オープン・パラメータの本体内で、このパラメータは、指定された項目記述子およびプロパティの可能な値のリストに設定されます。値は、タグのリスト（列挙型プロパティの場合）またはリポジトリ項目の配列のどちらかになります。

オープン・パラメータ

output
このパラメータは、検索結果とともに 1 回レンダリングされます。

例

次の例で、RepositoryValues は CatalogPossibleValues のインスタンスです。

```
<dsp:droplet name="RepositoryValues">
 <dsp:param value="category" name="itemDescriptorName"/>
  <dsp:oparam name="output">
   <dsp:droplet name="ForEach">
    <dsp:param param="values" name="array"/>
    <dsp:param value="+displayName" name="sortProperties"/>
    <dsp:oparam name="output">
    <dsp:getvalueof id="option14" param="element.repositoryId"
                idtype="java.lang.String">
<dsp:option value="<%=option14%>"/>
</dsp:getvalueof>
    <dsp:valueof param="element.displayName"/>
   </dsp:oparam>
   </dsp:droplet>
  </dsp:oparam>
</dsp:droplet>
```

ClosenessQualifierDroplet

クラス名	atg.commerce.promotion.ClosenessQualifierDroplet
コンポーネント	atg/commerce/promotion/ClosenessQualifierDroplet

ClosenessQualifierDroplet は、特定のオーダーに関連付けられた closenessQualifier 項目のリストをレンダリングします。タイプ、品目、オーダー、出荷および税を指定することによって、返される closenessQualifiers のタイプを限定できます。

入力パラメータ

elementName
返された closenessQualifiers が格納されている output パラメータを指定します。ここで値が指定されていない場合は、closenessQualifers 出力パラメータが使用されます。

order
アクセスする closenessQualifiers に対応するオーダーID。ここでオーダーが指定されていない場合は、アクティブなショッピング・カートからオーダーID が取得されます。ショッピング・カートは、ClosenessQualiferDroplet.promotionUpsellTools プロパティから /atg/commerce/promotion/PromotionUpsellTools.shoppingCartPath プロパティへの参照を介して取得されます。

type
アクセスする closenessQualifier のタイプ。次のオプションがあります。item、order、shipping、tax および all。order の値は、オーダーのすべての closenessQualifiers を返す必要があることではなく、特定のオーダー項目タイプに対して指定された closenessQualifiers を返す必要があることを示していることに注意してください。このパラメータを省略すると、すべてのタイプのプロモーション・アップセルが返されます。

出力パラメータ

closenessQualifiers
指定されたオーダーに対して返されるプロモーション・アップセルのリスト。このパラメータは、elementName 入力パラメータによって代替パラメータが指定されていない場合にのみ適用可能です。

errorMsg
エラーが発生したときにユーザーに表示するエラー・メッセージ。

オープン・パラメータ

empty
このパラメータは、プロモーション・アップセルが返されない場合にレンダリングされます。

error
このパラメータは、処理中にエラーが発生するとレンダリングされます。

output
このパラメータは、プロモーション・アップセルが返される場合にレンダリングされます。

例

次の例では、アクティブなユーザーが出荷関連のプロモーション・アップセルを取得したかどうかを ClosenessQualifierDroplet が調べます。オーダー・パラメータを除外することによって、Core Commerce は、現在のユーザーのショッピング・カート内の項目に基づいてこの確認を行います。ユーザーがプロモーション・アップセルに適格である場合は、関連付けられたメディア項目によってユーザーにメッセージが表示されます。

```
<dsp:droplet name="/atg/commerce/promotion/ClosenessQualifierDroplet">
  <dsp:param name="type" value="shipping"/>
  <dsp:param name="elementName" value="closenessQualifiers"/>
  <dsp:oparam name="output">

     <dsp:droplet name="/atg/dynamo/droplet/ForEach">
```

```
        <dsp:param name="array" param="closenessQualifiers"/>
        <dsp:param name="elementName" value="closenessQualifier"/>
        <dsp:oparam name="output">

         <dsp:getvalueof id="media_url"
          param="closenessQualifier.upsellMedia" idtype="String">
          <dsp:include page="<%=media_url%>"/>
         </dsp:getvalueof>

        </dsp:oparam>
      </dsp:droplet>

  </dsp:oparam>
</dsp:droplet>
```

CollectionFilter

クラス名	atg.service.collections.filter.droplet.CollectionFilter
コンポーネント	/atg/commerce/collections/filter/droplet /InventoryFilterDroplet /atg/commerce/collections/filter/droplet /ProductFilterDroplet /atg/commerce/collections/filter/droplet /ExcludeItemInCartFilterDroplet /atg/commerce/collections/filter/droplet /CartSharingFilterDroplet /atg/commerce/collections/filter/droplet /GiftListSiteFilterDroplet

CollectionFilter サーブレット Bean は、コレクション・フィルタリング・コンポーネントを使用してコレクション内のオブジェクトを減らします。サーブレット Bean ごとに使用するコレクション・フィルタリング・コンポーネントが異なります。

- 製品のコレクション用の InventoryFilterDroplet は、在庫マネージャによる実行時の判断に従って、どの SKU が使用可能かを判断します。このサーブレット Bean は、フィルタリング処理を扱う InventoryFilter にアクセスします。
- ProductFilterDroplet は、InventoryFilterDroplet と StartEndDateFilterDroplet の両方を実行して、両方の要件セットを満たす品目のみを返します。たとえば、在庫があり、かつアクティブな startDate を持っている製品は返されますが、それと同じ startDate を持っているが、在庫のない製品は返されません。このサーブレット Bean は、フィルタリング処理を扱う ProductFilter にアクセスします。
- ExcludeItemInCartFilterDroplet は、ユーザーのショッピング・カートにない製品を返します。このサーブレット Bean は ExcludeItemsInCartFilter を利用してショッピング・カート内にある品目を調べて、コレクションからそれらの品目を除外します。

- CartSharingFilterDroplet は CartSharingFilter を使用して、カート共有グループ内のサイトの製品のみを返します。
- GiftlistSiteFilterDroplet は、指定されたギフト・リストまたはギフト品目のコレクションをフィルタします。

このクラスに関する主な解説は、『ATG Web Commerce Page Developer's Guide』の *付録 B: ATG サーブレット Bean* に記載されています。コレクション・フィルタリング・コンポーネントおよびキャッシングの詳細は、『ATG Web Commerce Personalization Programming Guide』の *コレクションのフィルタリング* を参照してください。

ドロップレットごとに filter 入力パラメータが持っている CollectionFilter クラスのデフォルト値が異なることに注意してください。

- InventoryFilterDroplet は /atg/registry/CollectionFilters/InventoryFilter を使用します。
- ExcludeItemInCartFilterDroplet は /atg/registry/CollectionFilters/ExcludeItemInCartFilter を使用します。
- ProductFilterDroplet は/atg/registry/CollectionFilters/ProductFilter を使用します。
- CartSharingFilterDroplet は /atg/registry/COllectionFilters/CartSharingFilter を使用します。

ProductFilterDroplet の例

次の例では、ProductFilterDroplet を使用することで、ProductFilter で指定されているフィルタ（InventoryFilter と StartEndDateFiler）がそれぞれのフィルタリング・メカニズムを製品のコレクションに適用します。結果として、在庫があり、かつアクティブな製品のコレクションが表示されます。

```
<dspel:droplet name="/atg/commerce/catalog/CategoryLookup">
   <dspel:param name="Id" param="catId"/>
   <dspel:oparam name="output">

   <%
    String collIdentifierKey = request.getParameter("catId") + "-childprd";
   %>
      <dspel:droplet name="/atg/commerce/collections/filter/droplet/
             ProductFilterDroplet">
         <dsp:param name="collection" param="item.childproducts/>
         <dsp:param name="collectionIdentifierKey" value="<%=collIdentifierKey
          %>"/>

         <dspel:oparam name="output">
             Featured Plants:
             <p><dsp:droplet name="/atg/dynamo/droplet/ForEach">
                   <dsp:param name="array" param="filteredCollection"/>

                   <dsp:oparam name="output">
                      <dspel:valueof param="element"/>
                   </dspel:oparam>
                </dsp:droplet>
```

```
        </dspel:oparam>

        <dspel:oparam name="empty">
           There are currently no outdoor plants
        </dspel:oparam>
      </dspel:droplet>
   </dspel:oparam>
</dspel:droplet>
```

GiftListSiteFilterDroplet の例

この JSP からの抜粋は、GiftlistSiteFilterDroplet を使用してギフト・リストをフィルタする方法の一例を示しています。サイト・スコープが渡されていないため、GiftlistSiteFilter は GiftlistManager コンポーネントの siteScope を使用します。siteScope は、この例の目的に合わせて atg.ShoppingCart 共有可能タイプ・コンポーネントに設定されています。また、サイト ID が指定されていないため、フィルタされるギフト・リストは、現在のサイトおよび現在のサイトとショッピング・カートを共有しているサイトからのみ取得されます。

```
<dsp:droplet
name="/atg/commerce/collections/filter/droplet/GiftlistSiteFilterDroplet">
  <&-- Specify the collection to filter --%>
  <dsp:param name="collection" bean="Profile.giftlists"/>

  <dsp:oparam name="output">

    <%-- Iterate through the collection. --%>
      <dsp:droplet name="="/atg/dynamo/droplet/ForEach">
        <dsp:param name="array" param="filteredCollection"/>

        <dsp:oparam name="output">
          <dsp:setvalue param="giftList" paramvalue="element"/>
          <dsp:getvalueof var="eventName" param="giftList.eventName"/>
          <c:out value="${eventName}"/>
        </dsp:oparam>
      </dsp:droplet>

  </dsp:oparam>
</dsp:droplet>
```

この JSP からの抜粋は、ウィッシュ・リスト品目のコレクションをフィルタします。サイト・スコープ値 current がフィルタに渡されますが、サイト ID が渡されないため、現在のサイトから取得された品目のコレクションのみが結果として生成されます。

```
<dsp:droplet
name="/atg/commerce/collections/filter/droplet/GiftlistSiteFilterDroplet">
  <&-- Specify the collection to filter and the site scope. --%>
  <dsp:param name="collection" bean="Profile.wishlist.giftlistItems"/>
  <dsp:param name="siteScope" value="current"/>
```

```
  <dsp:oparam name="output">

    <%-- Iterate through the collection. --%>
      <dsp:droplet name="="/atg/dynamo/droplet/ForEach">
        <dsp:param name="array" param="filteredCollection"/>

        <dsp:oparam name="output">
          <dsp:setvalue param="giftItem" paramvalue="element"/>
          <dsp:getvalueof var="displayName" param="giftItem.displayName"/>
          <c:out value="${displayName}"/>
        </dsp:oparam>
      </dsp:droplet>

   </dsp:oparam>
</dsp:droplet>
```

注意:

- GiftlistSiteFilter コンポーネントと GiftlistSiteFilterDroplet コンポーネントは、CollectionFilter クラスに基づく他のフィルタと同様に、フィルタリングのパフォーマンスを向上させるためにキャッシングを使用するように構成できます。『ATG Web Commerce Personalization Programming Guide』の *フィルタリングされたコンテンツのキャッシュ*に関する項を参照してください。
- サイト ID を GiftlistFilterDroplet に渡すには、カンマで区切られたリストを使用します。

ComplexPriceDroplet

クラス名	atg.commerce.pricing.priceLists.ComplexPriceDroplet
コンポーネント	/atg/commerce/pricing/priceLists/ComplexPriceDroplet

ComplexPriceDroplet サーブレット Bean は複合価格を受け取り、複合価格に含まれているレベルを返します。

入力パラメータ

complexPrice（必須）
複合価格の ID。

出力パラメータ

levelMinimums
各レベルに適用される最小数量。このパラメータは常に 1 から始まります。

levelMaximums
各レベルに適用される最大数量。最後のレベルは最大値を持っていないため、このパラメータは常に levelMinimums より 1 品目少なくなります。

prices
各レベルの価格のリスト。この配列の最後の価格は常に defaultPrice です。

numLevels
quantities 配列および prices 配列の長さ。

出力パラメータ

error
エラーが発生するとレンダリングされるパラメータ。

オープン・パラメータ

output
複合価格が正常に処理されるとレンダリングされる oparam。

例

次の例は、ComplexPriceDroplet の JSP コードを示しています。

```
<dsp:droplet name="ComplexPriceDroplet">
 <dsp:param param="complexPrice" name="complexPrice"/>
 <dsp:oparam name="output">
  <table border=1>
   <dsp:droplet name="For">
    <dsp:param param="numLevels" name="howMany"/>
    <dsp:param value="index" name="indexName"/>
    <dsp:oparam name="output">
        <tr>
      <td>
       <dsp:valueof param="leveMinimums[param:index]"/> -
       <dsp:valueof param="levelMaximums[param:index]">?</dsp:valueof>
      </td>
      <td>
        <dsp:valueof param="prices[param:index]"/>
      </td>
    </tr>
 </dsp:oparam>
</dsp:droplet>
```

ConvertAbandonedOrderDroplet

クラス名	atg.commerce.order.abandoned.ConvertAbandonedOrderDroplet
コンポーネント	/atg/commerce/order/abandoned/ConvertAbandonedOrderDroplet （Abandoned Order Services モジュールのみ）

ConvertAbandonedOrderDroplet サーブレット Bean は、特定のオーダーに対する放棄されたオーダー、復活したオーダーまたは失われたオーダーとしての指定を、変換されたオーダーとしての指定に置き換えます。より具体的に説明すれば、このサーブレット Bean は次のことを実行します。

1. オーダーが放棄されており、失われたり復活したりしていない場合に、ユーザーの abandonedOrders プロファイル・プロパティ内の放棄されたオーダーのリストからそのオーダーを削除します。
2. オーダーの abandonmentInfo 項目を次のように変更します。
   - state プロパティを CONVERTED に設定します。
   - conversionDate プロパティを現在の日付、時刻に設定します。
3. AbandonedOrderTools.sendOrderConvertedMessage プロパティが true に設定されていれば、AbandonedOrderConverted メッセージを生成します。

オーダーの abandonmentInfo 項目の state プロパティが null である場合は、オーダーが放棄されておらず、この処理によって何も実行されないことに注意してください。

Abandoned Order Services モジュールの詳細は、『ATG Web Commerce Programming Guide』の*放棄されたオーダー・サービスの使用*を参照してください。

入力パラメータ

orderId（必須）
現在のオーダーの ID。

出力パラメータ

なし。

オープン・パラメータ

output
このパラメータは、オーダーが変換されるとレンダリングされます。

error
このパラメータは、エラーが発生するとレンダリングされます。

例

```
<dsp:droplet name="ConvertAbandonedOrderDroplet">
 <dsp:param name="orderId" bean="/atg/commerce/ShoppingCart.current.id"/>…
</dsp:droplet>
```

CostCenterDroplet

クラス名	atg.commerce.order.purchase.CostCenterDroplet
コンポーネント	/atg/commerce/order/purchase/CostCenterDroplet

CostCenterDroplet サーブレット Bean は、次のタスクを実行するサービス・メソッドを備えた要求限定コンポーネントです。

- CostCenter の初期化 - ユーザーの認可されたコスト・センターを表す CostCenter オブジェクトが作成され、CostCenterMapContainer に追加されます。
- CommerceIdentifierCostCenter の初期化 - 現在の Order に固有の新しい CommerceIdentifierCostCenter のインスタンスが作成され、CommerceIdentifierCostCenterContainer に追加されます。

これらのタスクによって、ユーザーは自社の認可されたコスト・センターをオーダーの様々な CommerceIdentifier オブジェクトに関連付けることができます。初期化中、CostCenterDroplet は、オプションで、1 つの CommerceIdentifier タイプ(たとえば、個々の CommerceItem、個々の ShippingGroup、Order および Tax)のオブジェクトごとに 1 つの CommerceIdentifierCostCenter オブジェクトを作成します。

コスト・センターのクラスとフレームワークの詳細は、コスト・センターの管理を参照してください。

入力パラメータ

clear
このパラメータが true に設定されていると、CostCenterDroplet は CommerceIdentifierCostCenterContainer および CostCenterMapContainer の両方を消去します。

clearCostCenterContainer
このパラメータが true に設定されていると、CostCenterDroplet は CommerceIdentifierCostCenterContainer 内の CommerceIdentifierCostCenters を消去します。

clearCostCenterMap
このパラメータが true に設定されていると、CostCenterDroplet は CostCenterMapContainer 内の CostCenters を消去します。

initCostCenters
このパラメータが true に設定されていると、CostCenterDroplet は、ユーザーの認可されたコスト・センターを表す CostCenter オブジェクトを CostCenterMapContainer に追加します。

initItemCostCenters
このパラメータが true に設定されていると、CostCenterDroplet は、オーダー内の個々の CommerceItem に対応する CommerceIdentifierCostCenter オブジェクトを作成し、それらを CommerceIdentifierCostCenterContainer に追加します。

initShippingCostCenters
このパラメータが true に設定されていると、CostCenterDroplet は、オーダー内の個々の ShippingGroup に対応する CommerceIdentifierCostCenter オブジェクトを作成し、それらを CommerceIdentifierCostCenterContainer に追加します。

initTaxCostCenters
このパラメータが true に設定されていると、CostCenterDroplet は、税に対応する CommerceIdentifierCostCenter オブジェクトを作成し、それを CommerceIdentifierCostCenterContainer に追加します。

initOrderCostCenters
このパラメータが true に設定されていると、CostCenterDroplet は、オーダーに対応する CommerceIdentifierCostCenter オブジェクトを作成し、それを CommerceIdentifierCostCenterContainer に追加します。

useAmount
useAmount パラメータは、商品とコスト・センターとの間に存在する関係のタイプを記述します。この関係は、開発者が、数量または金額のどちらを基準にしてコスト・センター間で商品を分割することをユーザーに許可したかよって異なります。このパラメータが true に設定されていると、コスト・センター関係の決定に金額が使用されます。値が false に設定されていると、コスト・センター関係の決定に数量が使用されます。デフォルト値は false です。

order
コスト・センターの割当先となるユーザーのオーダー。

出力パラメータ

costCenters
ユーザーの有効なコスト・センターのリスト。

ciccMap
Commerce Identifier をキーとして持ち、その Commerce Identifier に関連付けられた CommerceIdentifierCostCenters のリストを値として持つマップ。

order
渡されたユーザーのオーダー。

オープン・パラメータ

output
操作が成功するとレンダリングされるオープン・パラメータ。

例

CostCenterDroplet の用途として一般的なのは、(1)サーブレット Bean によって設定された出力を表示し、(2)ユーザーが品目およびその他のコストを他の有効なコスト・センターに再割当できるようにするフォームをサーブレット Bean の output パラメータ内に作成することです。次の JSP の例で、clear パラメータ、clearCostCenterMap パラメータおよび clearCostCenterContainer パラメータがすべて false に設定されているのは、その用途があるからです。発行後にリロードされるフォームをサーブレット Bean で使用すれば、これらのパラメータを false に設定することによって、ユーザーが加え、発行した変更が消去されることを防止できます。

```
<dsp:droplet name="CostCenterDroplet">
  <dsp:param bean="ShoppingCartModifier.order" name="order"/>
  <dsp:param value="false" name="clear"/>
  <dsp:param value="false" name="clearCostCenterMap"/>
  <dsp:param value="false" name="clearCostCenterContainer"/>
  <dsp:param value="true" name="initCostCenters"/>
```

```
  <dsp:param value="true" name="initItemCostCenters"/>
  <dsp:param value="true" name="initShippingCostCenters"/>
  <dsp:param value="true" name="initTaxCostCenters"/>
  <dsp:param value="false" name="useAmount"/>
  <dsp:oparam name="output">some form
  </dsp:oparam>
</dsp:droplet>
```

CouponDroplet

クラス名	atg.commerce.promotion.CouponDroplet
コンポーネント	/atg/commerce/promotion/CouponDroplet

CouponDroplet サーブレット Bean は、プロモーション・オブジェクトまたはプロモーション ID のどちらかを受け取り、そのプロモーションのクーポンを要求可能リポジトリ内で生成します。ターゲット設定された E メール JSP に CouponDroplet を含めることによって、顧客に送信できる状態にあるクーポンを作成できます。

入力パラメータ

promoId（このパラメータまたは promotions を使用する必要があります）
クーポンの作成に使用されるプロモーションの ID。

promotion
現在は使用されていません。

promotions（このパラメータまたは promoId を使用する必要があります）
クーポンの作成に使用される 1 つ以上のプロモーション・オブジェクト。

displayName（オプション）
クーポンの表示名。値が指定されていない場合は、関連付けられた最初のプロモーションの名前がクーポンの表示名として使用されます。デフォルトは null です。

usePromotionSiteConstraint（オプション）
true に設定されていると、このフラグは、作成されているクーポンがクーポンに関連付けられているプロモーションと同じサイト制約を使用することを示します。クーポンの作成に複数のプロモーションが使用されている場合は、すべてのプロモーションが同じサイト制約を持つ必要はありません。デフォルトは false です。

出力パラメータ

coupon
クーポン・オブジェクト。coupon.id を使用することによって、このクーポンの要求コードを取得できます。クーポンの要求コードでは大文字と小文字が区別されることに注意してください。たとえば、COUP100 と coup100 は 2 つの別々の要求コードです。

オープン・パラメータ

output
クーポン・オブジェクトが正常に作成されるとレンダリングされるオープン・パラメータ。

error
クーポンの作成中にエラーが発生するとレンダリングされるオープン・パラメータ。

例

次の例は、CouponDroplet の JSP コードを示しています。

```
<dsp:importbean bean="/atg/commerce/promotion/CouponDroplet"/>
<h2>here is a coupon that was created: </h2>
<dsp:droplet name="CouponDroplet">
<dsp:param value="promo60001" name="promoId"/>
<dsp:oparam name="output">
   <dsp:valueof param="coupon.id">no value</dsp:valueof>
</dsp:oparam>
<dsp:oparam name="error">
</dsp:oparam>
</dsp:droplet>
```

CurrencyCodeDroplet

クラス名	atg.commerce.pricing.CurrencyCodeDroplet
コンポーネント	/atg/commerce/pricing/CurrencyCodeDroplet

CurrencyCodeDroplet サーブレット Bean は、ロケールを入力として受け取り、そのロケールの通貨コードを返します。

入力パラメータ

locale（必須）
現在のロケール。

出力パラメータ

currencyCode
ロケールの ISO 4217 通貨コード。

オープン・パラメータ

output
このオープン・パラメータは常にレンダリングされます。

例

次の例は、CurrencyCodeDroplet サーブレット Bean の JSP コードを示しています。この例では、ユーザーのプロファイルからロケールが取得され、商品券の金額の書式設定に使用されます。

```
<dsp:droplet name="CurrencyCodeDroplet">
  <dsp:param name="locale" bean="Profile.PriceList.locale"/>
  <dsp:oparam name="output">
    <dsp:getvalueof var="currencyCode" vartype="java.lang.String"
param="currencyCode"/>
    <dsp:getvalueof var="amount" vartype="java.lang.Double"
param="giftCertificate.amount"/>
  </dsp:oparam>
</dsp:droplet>
<%-- The format of message to display is:
A gift certificate has been purchased for you in the amount of {0} by {1} {2} --%>
<fmt:message key="emailtemplates_giftCertificate.purchasedInfo">
  <fmt:param>
    <fmt:formatNumber value="${amount}" type="currency"
currencyCode="${currencyCode}"/>
  </fmt:param>
….
```

DisplaySkuProperties

クラス名	atg.commerce.catalog.DisplaySkuProperties
コンポーネント	/atg/commerce/catalog/DisplaySkuProperties

DisplaySkuProperties サーブレット Bean は、SKU 項目を入力として受け取り、指定されたプロパティのセットを連結文字列としてレンダリングします。たとえば、DisplaySkuProperties を使用して、SKU の displayName プロパティ、price プロパティおよび description プロパティを表示できます。

入力パラメータ

sku（必須）
SKU 項目のロケール。

delimiter
連結文字列内の異なるプロパティ値の間のセパレータとして使用する文字。このパラメータを省略すると、スペースがデリミタとして使用されます。

propertyList または product
propertyList パラメータを使用して、カンマ区切りのリストとして表示される SKU プロパティのリストを指定

するか、product パラメータを使用して SKU の親製品を指定できます。後者の場合、プロパティのリストは製品の displayableSkuAttributes プロパティから取得されます。

displayElementName
output オープン・パラメータ内で設定されるパラメータとして使用する名前。

出力パラメータ

displayElement
プロパティ値が含まれた連結テキスト文字列（オプションの displayElementName 入力パラメータを介してこのパラメータに異なる名前を指定できます）。

オープン・パラメータ

output
出力テキスト文字列が空でない場合にレンダリングされるオープン・パラメータ。

empty
出力テキスト文字列が空の場合にレンダリングされるオープン・パラメータ。

例

次の例は、DisplaySkuProperties サーブレット Bean の JSP コードを示しています。この例では、DisplaySkuProperties を内部にネストしている別のサーブレット Bean によって SKU が DisplaySkuProperties に渡されます。

```
<dsp:droplet name="/atg/commerce/catalog/DisplaySkuProperties">
   <dsp:param value=" | " name="delimiter"/>
   <dsp:param param="element" name="sku"/>
   <dsp:param value="displayName,listPrice,description" name="propertyList"/>
   <dsp:oparam name="output">
   <p><dsp:valueof param="displayElement"/>
   </dsp:oparam>
   <dsp:oparam name="empty">
   <p>There is no information available about this item.
   </dsp:oparam>
</dsp:droplet>
```

ExcludeItemsInCartFilterDroplet

クラス名	atg.service.collections.filter.droplet.CollectionFilter
コンポーネント	/atg/commerce/collections/filter/droplet/ExcludeItemsInCartFilterDroplet

ExcludeItemsInCartFilterDroplet サーブレット Bean を使用すると、すでに顧客のカート内にある品目をリストから除外できます。

入力パラメータ

collectionIdentifierKey
この値はフィルタされていないコレクションを識別します。この値が指定されていない場合は、consultCache パラメータと updateCache パラメータが強制的に false に設定されます。

filter
任意のコレクション・フィルタ・コンポーネント。

profile
任意のプロファイル・リポジトリ項目が値になります。

collection（必須）
フィルタされていないコレクション。

consultCache
“true”または“false”である文字列値。

updateCache
“true”または“false”である文字列値。

出力パラメータ

filteredCollection
フィルタされたコレクション。

errorMsg
処理エラーが発生したときのエラー・メッセージ。

オープン・パラメータ

output
このタグはフィルタが正常終了すると 1 回レンダリングされます。

empty
このタグは、フィルタされたコレクションが null であるか、フィルタされたコレクションにオブジェクトが含まれていないとレンダリングされます。

error
このタグは、エラーが発生するとレンダリングされます。

例

次の JSP の例では、`filteredCollection` パラメータを `ExcludeItemsInCart` で使用して、フィルタされたアップセル製品のリストを表示します。

```
<dsp:droplet name="/atg/commerce/collections/filter/droplet/ExcludeItemsInCart
FilterDroplet">
   <dsp:param name="collection" param="category.upsellProducts"/>
   <dsp:oparam name="output">
      You may also like these<p>
      <dsp:droplet name="/atg/droplet/ForEach">
         <dsp:param name="array" param="filteredCollection" />
         <dsp:oparam name="output">
            Product <dsp:valueof param="element.repositoryId"/><p>
```

```
                <dsp:valueof param="element.description"/><p>
            </dsp:oparam>
        </dsp:droplet>
    </dsp:oparam>
</dsp:droplet>
```

FindMatchingOrderMarkers

key、value、data および extended のプロパティに基づいて、一致する項目からすべてのマーカーを返します。

クラス名	atg.commerce.markers.droplet.FindMatchingOrderMarkers
コンポーネント	/atg/commerce/markers/droplet/FindMatchingOrderMarkers

入力パラメータ

order（必須）
マーカーの有無が確認されるオーダー。

key
マーカーの key プロパティの値。マーカーの key 値に任意の値を指定できることを示す場合は、このパラメータを ANY_VALUE に設定します。たとえば、次のようになります。

```
<dsp:param name="key" bean="HasLastMarkerDroplet.ANY_VALUE"/>
```

value
マーカーの value プロパティの値。マーカーの value プロパティを null にする必要がある場合は、このパラメータを NULL に設定するか、パラメータを省略します。

マーカーの value 値に任意の値を指定できることを示す場合は、このパラメータを ANY_VALUE に設定します。たとえば、次のようになります。

```
<dsp:param name="value" bean="HasMarkerDroplet.ANY_VALUE"/>
```

data
マーカーの data プロパティの値。マーカーの data プロパティを null にする必要がある場合は、このパラメータを NULL に設定するか、パラメータを省略します。

マーカーの data 値に任意の値を指定できることを示す場合は、このパラメータを ANY_VALUE に設定します。たとえば、次のようになります。

```
<dsp:param name="data" bean="HasMarkerDroplet.ANY_VALUE"/>
```

extendedProperties
マーカーに存在する必要のある追加のマーカー・プロパティ（マップの key に設定されている）およびプロパティ値（マップの value に設定されている）を指定するマップ。

markerPropertyName
マーカーの有無が確認されるオーダーのプロパティ。省略した場合、サーブレット Bean はその defaultMarkerPropertyName プロパティに指定されているデフォルト値を使用します。

出力パラメータ

marker
サーブレット Bean によって特定されたマーカーが格納されます。

オープン・パラメータ

error
サーブレット Bean の実行時にエラーが発生した場合にレンダリングされます。

true
指定した基準に一致するマーカーがオーダーに付けられている場合にレンダリングされます。

false
指定した基準に一致するマーカーがオーダーに付けられていない場合にレンダリングされます。

ForEachItemInCatalog

クラス名	`atg.commerce.catalog.custom.ForEachItemInCatalog`
コンポーネント	`/atg/commerce/catalog/ForEachItemInCatalog`

`ForEachItemInCatalog` サーブレット Bean は、ユーザーの現在のカタログ内に存在する `array` 入力パラメータ内の要素ごとに、サーブレット Bean の `output` オープン・パラメータを 1 回ずつレンダリングします。

このサーブレット Bean は、`atg.droplet.ForEach` を拡張する `atg.commerce.catalog.custom.ForEachItemInCatalog` のインスタンスです。`ForEachItemInCatalog` パラメータは、ユーザーの現在のカタログを（`Profile.catalog` プロパティから）取得するために使用される 1 つの追加パラメータ `profile` を持っています。

入力パラメータ

array（必須）
出力する品目のリストを定義するパラメータ。このパラメータは、Collection（Vector、List または Set）、Enumeration、Iterator、Map、Dictionary または配列である可能性があります。

profile
現在のユーザーのプロファイル。

sortProperties
出力配列内の品目のリストをソートする方法を指定する文字列。JavaBeans または Dynamic Beans のプロパティ、あるいは Dates、Numbers または Strings に対してソートを実行できます。

JavaBeans をソートするには、`sortProperties` の値をプロパティ名のカンマ区切りのリストとして指定します。最初の名前が最優先のソート基準、2 番目の名前が 2 番目に優先するソート基準、それ以降も同様に、名前の順番にソート基準の優先順位が決まります。キーワードの最初の文字が「+」である場合、そのソートは昇順で実行されます。最初の文字が「-」である場合、ソートは降順で実行されます。

JavaBeans の出力配列を、最初にタイトル・プロパティを基準にアルファベット順にソートし、次にサイズ・プロパティを基準に降順にソートする例

```
<param name="sortProperties" value="+title,-size">
```

Dates、Numbers または Strings をソートするには、sortProperties の値を、昇順または降順を示す 1 つの“+”または 1 つの“-”とともに指定します。

例:Strings の出力配列をアルファベット順にソートするには

```
<param name="sortProperties" value="+">
```

reverseOrder
配列内の走査順が後ろから前または前から後ろのどちらであるかを指定するブール値。後ろから前にソートするには、このパラメータを true に設定します。前から後ろにソートするには、このパラメータを false に設定します。このパラメータは、sortProperties 入力パラメータが設定されていない場合にのみ効力を持つことに注意してください。

出力パラメータ

index
このパラメータは、出力パラメータがレンダリングされるたびに配列の現在の要素のゼロベースの索引に設定されます。最初の繰り返し処理の索引の値は 0 です。

count
このパラメータは、出力パラメータがレンダリングされるたびに配列の現在の要素の 1 ベースの索引に設定されます。最初の繰り返し処理の count の値は 1 です。

key
配列パラメータが Map または Dictionary である場合、このパラメータは Map または Dictionary のキーの値に設定されます。

element
このパラメータは、索引が増分され、出力パラメータがレンダリングされるたびに配列の現在の要素に設定されます。

size
該当する場合、このパラメータは配列のサイズに設定されます。配列が Enumeration または Iterator の場合、サイズは-1 に設定されます。

オープン・パラメータ

output
このパラメータは、配列内の 1 つの要素につき 1 回レンダリングされます。

outputStart
配列が空でなければ、このパラメータは他のどの出力要素より前にレンダリングされます。たとえば表の見出しのレンダリングにこのパラメータを使用できます。

outputEnd
配列が空でなければ、このパラメータは他のすべての出力要素の後でレンダリングされます。たとえば表に続くテキストのレンダリングにこのパラメータを使用できます。

empty
このオプション・パラメータは、配列に要素が含まれていない場合にレンダリングされます。

例

次の JSP の例では、ForEachItemInCatalog を使用して、特定の製品の関連製品のリストを製品表示ページに表示します。個々の関連製品が、選択された関連製品を反映させて製品表示ページを再表示するリンクです。

```
<dsp:droplet name="/atg/commerce/catalog/ForEachItemInCatalog">
 <dsp:param param="element.relatedProducts" name="array"/>

  <dsp:oparam name="outputStart">
   <table border=0 cellpadding=1 width=100%>
   <tr><td><span class=smallbw> Related Items</span></td></tr>
   <tr><td></td></tr>
  </dsp:oparam>

   <dsp:oparam name="output">
    <tr><td>
    <dsp:getvalueof id="a28" param="element.template.url"
                idtype="java.lang.String">
<dsp:a href="<%=a28%>">
    <dsp:param param="element.repositoryId" name="id"/>
    <dsp:param value="jump" name="navAction"/>
    <dsp:param param="element" name="Item"/>
    <dsp:valueof param="element.displayName">
         No name</dsp:valueof></dsp:a></dsp:getvalueof><br>
    </td></tr>
   </dsp:oparam>

  <dsp:oparam name="outputEnd">
   </table>
  </dsp:oparam>
</dsp:droplet>
```

GetApplicablePromotions

クラス名	atg.commerce.pricing.GetApplicableItemPromotions
コンポーネント	/atg/commerce/pricing/GetApplicablePromotions

GetApplicablePromotions サーブレット Bean は、特定の製品または SKU に適用できる品目プロモーションを調べるために使用されます。

GetApplicablePromotions サーブレット Bean は、atg.commerce.pricing.GetApplicableItemPromotions からインスタンス化されます。

入力パラメータ

item（必須）
直接評価できる CommerceItem。

pricingModels
品目の価格設定に使用される価格設定モデル（プロモーション）のコレクション。この値が指定されていないと、デフォルトで、ユーザーの PricingModelHolder コンポーネントに格納されている価格設定モデルのコレクションが使用されます。このコンポーネントは、userPricingModelsPath プロパティを介して解決されます。

product
評価される品目の製品定義を表すオブジェクト。一般に、品目は SKU です。その場合、このオブジェクトは特定の SKU を持つ製品です。

elementName
output オープン・パラメータ内で設定されるパラメータとして使用する名前。

quantity
入力された製品の Long 型の数量。このパラメータは、提供された情報に基づいて CommerceItem を作成するときに使用されます。

出力パラメータ

element
評価された CommerceItem。このパラメータ名は elementName 入力パラメータを設定することによって変更できます。

promotions
この CommerceItem を対象としたプロモーションのコレクション。

オープン・パラメータ

output
CommerceItem が正常に価格設定されるとレンダリングされるオープン・パラメータ。

例

次の例では、プロモーション、ロケールおよびプロファイルがパラメータとして指定されていないため、要求から抽出されます。

```
<dsp:droplet name="/atg/commerce/pricing/GetApplicablePromotions">
  <dsp:param name="product" param="product"/>
  <dsp:param name="item" param="product.childSkus[0]"/>
  <dsp:oparam name="output">
     <dsp:droplet name="ForEach">
            <dsp:param name="array" param="promotions"/>
          <dsp:oparam name="output">
               Promotion: <dsp:valueof param="element.displayName"/><br>
          </dsp:oparam>
     </dsp:droplet>
   </dsp:droplet>
```

GeoLocatorDroplet

クラス名	atg.commerce.locations.GeoLocatorDroplet
コンポーネント	/atg/commerce/locations/StoreLocatorDroplet

GeoLocatorDroplet を使用すると、顧客が指定したポイントから一定の距離内にあるストアの場所を特定できます。このポイントの指定には、次のいずれかを使用できます。

- 市区町村および都道府県/州
- Coordinate リポジトリ項目
- 自由形式の住所
- 緯度および経度

このドロップレットを使用すると、顧客がオンライン購入のストア内引取の場所を指定したり、最も近い実際の小売店の場所を検索するのに役立てることができます。

CoordinateManager は検索する項目を決定し、実際の検索を実行します。geoLocatorService は、郵便番号、市区町村と都道府県/州、または任意の住所に基づいて場所を検索するように構成できます。geoLocatorService が構成されていない場合、ドロップレットは緯度と経度による検索のみをサポートします。

注意: 示される座標値はローカル・エリアのものであり、単なる目安です。

場所の指定方法を複数使用する場合の優先順位は、次のようになります。

1. Coordinate 項目
2. 自由形式の住所
3. 郵便番号
4. 市区町村および都道府県/州
5. 緯度および経度

入力パラメータ

coordinateItem（オプション）
検索元となる場所を提供する Coordinate リポジトリ項目。Coordinate タイプのプロパティ名構造は、このドロップレットによって使用される座標マネージャのものと同じである必要があります。

address（オプション）
自由形式の住所。

postalCode（オプション）
検索の中心位置として使用される郵便番号。

city（オプション）
検索する市区町村。

state（オプション）
検索する都道府県または州。市区町村も指定されている場合にのみ使用します。

latitude（オプション）
検索の中心位置として使用される緯度。指定した場合は、経度も指定する必要があります。

longitude（オプション）
検索の中心位置として使用される経度。指定した場合は、緯度も指定する必要があります。

distance（オプション）
検索対象の半径（メートル単位）。中心位置からこの距離内にあるすべての項目が返されます。指定しなかった場合は、このコンポーネントに構成されている距離が使用されます。

itemType（オプション）
検索する項目タイプ。有効なタイプが場所リポジトリに定義されている必要があります。このパラメータを使用すると、ストアの場所や一般的な場所など、様々な場所タイプを検索できます。

siteIds（オプション）
場所はサイトに所属させることができます。この場合は、このパラメータを使用するとサイトで場所をフィルタできます。`siteIds` パラメータと `siteScope` パラメータを両方使用した場合は、`siteIds` が `siteScope` より優先されます。

siteScope（オプション）
検索するサイト・スコープを指定します。これには、all、current または共有項目 ID のいずれかを指定できます。このパラメータを渡さなかった場合は、デフォルトのサイト・スコープが使用されます。

country（オプション）
ロケールの一部として使用する国。

language（オプション）
ロケールの一部として使用する言語。

elementName（オプション）
結果要素の名前。

オープン・パラメータ

output
このパラメータは、中心位置からの範囲内で項目が見つかった場合にレンダリングされます。

empty
このパラメータは、中心位置付近に項目が見つからなかった場合にレンダリングされます。

error
このパラメータは、このドロップレットの実行時にエラーが発生した場合にレンダリングされます。

GetRelatedReturnRequest

クラス	`atg.commerce.csr.returns.GetRelatedReturnRequests`
コンポーネント	`/atg/commerce/custsvc/returns/` `GetRelatedReturnRequests.properties`

GetRelatedReturnRequests サーブレット Bean は、orderId パラメータを使用して指定されたオーダーに関連する ReturnRequests を提供します。関連する ReturnRequests は、オーダーまたは replacementOrder プロパティを使用して、指定したオーダーに対して追跡できます。

入力パラメータ

orderId
関連する ReturnRequests を持つオーダーの ID。

オープン・パラメータ

output
関連する返品がある場合にレンダリングされます。

empty
関連する返品がない場合にレンダリングされます。

出力パラメータ

relatedReturnRequests
関連する ReturnRequests の配列が得られます。

GiftCertificateAmountAvailable

クラス名	atg.commerce.claimable.GiftCertificateAmountAvailable
コンポーネント	/atg/commerce/claimable/GiftCertificateAmountAvailable

GiftCertificateAmountAvailable サーブレット Bean を使用すれば、現在の購買の後に残る商品券の残額がわかります。

入力パラメータ

usedAmount（必須）
使用される商品券の金額。

giftCertificateNumber（必須）
商品券の要求コード。

出力

amountAvailable
usedAmount を差し引いた後に残る商品券の残額。

オープン・パラメータ

output
このオープン・パラメータは常にレンダリングされます。

例

次のコード例は、GiftCertificateAmountAvailableドロップレットを使用してユーザーの商品券の残額を表示する方法を示しています。

```
<dsp:droplet name="GiftCertificateAmountAvailable">
  <dsp:param name="giftCertificateNumber"
param="paymentGroup.giftCertificateNumber"/>
  <dsp:oparam name="output">
    <dt>
      <fmt:message key="common.remainingAmount"/><fmt:message
key="common.labelSeparator"/>
    </dt>
    <dd>
      <dsp:getvalueof var="amountRemaining" vartype="java.lang.Double"
param="amountAvailable"/>
      <fmt:formatNumber value="${amountRemaining}" type="currency"
currencyCode="${currencyCodeVar}"/>
    </dd>
  </dsp:oparam>
</dsp:droplet>
```

GiftitemDroplet

クラス名	atg.commerce.gifts.GiftitemDroplet
コンポーネント	/atg/commerce/gifts/BuyItemFromGiftlist /atg/commerce/gifts/RemoveItemFromGiftlist

GiftitemDroplet クラスからインスタンス化されたサーブレット Bean を使用すると、顧客は自分の個人的なギフト・リストから品目を購入または削除できます。BuyItemFromGiftlist と RemoveItemFromGiftlist という 2 つの Core Commerce サーブレット Bean が GiftitemDroplet からインスタンス化されています。

入力パラメータ

giftId（必須）
ギフトの ID。

giftlistId（必須）
ギフト・リストの ID。

出力

なし。

オープン・パラメータ

output
品目がリストから正常に購入または削除されるとレンダリングされるオープン・パラメータ。

error
処理中にエラーが発生するとレンダリングされるオープン・パラメータ。

例

次のコード例は、RemoveItemFromGiftlist コンポーネントを使用して顧客の個人的なギフト・リストから品目を削除する方法を示しています。

```
<dsp:droplet name="/atg/dynamo/droplet/IsEmpty">
<dsp:param param="giftId" name="value"/>
<dsp:oparam name="false">
   <dsp:droplet name="/atg/commerce/gifts/RemoveItemFromGiftlist">
     <dsp:param param="giftlistId" name="giftlistId"/>
     <dsp:param param="giftId" name="giftId"/>
   </dsp:droplet>
</dsp:oparam>
</dsp:droplet>
```

GiftlistDroplet

クラス名	atg.commerce.gifts.GiftlistDroplet
コンポーネント	/atg/commerce/gifts/GiftlistDroplet

GiftlistDroplet サーブレット Bean では、他の顧客のギフト・リストを顧客のプロファイルに追加したり、顧客のプロファイルから削除したりします。

入力パラメータ

action（必須）
ギフト・リストに対して実行する処理。

giftlistId（必須）
ギフト・リストの ID。

profile
現在の顧客のプロファイル。このパラメータを渡さない場合、プロファイルは Nucleus によって解決されます。

出力パラメータ

なし。

オープン・パラメータ

output
ギフト・リストがプロファイルに正常に追加されるか、プロファイルから正常に削除されるとレンダリングされるオープン・パラメータ。

error
ギフト・リストの追加中または削除中にエラーが発生するとレンダリングされるオープン・パラメータ。

例

次のコード例は、GiftlistDroplet を使用して、取得されたギフト・リストを顧客のプロファイルに追加する方法を示しています。

```
<dsp:droplet name="/atg/dynamo/droplet/IsEmpty">
<dsp:param param="giftlistId" name="value"/>
<dsp:oparam name="false">
 <dsp:droplet name="/atg/commerce/gifts/GiftlistDroplet">
   <dsp:param param="giftlistId" name="giftlistId"/>
   <dsp:param value="add" name="action"/>
   <dsp:param bean="/atg/userprofiling/Profile" name="profile"/>
 </dsp:droplet>
</dsp:oparam>
</dsp:droplet>
```

GiftShippingGroupDroplet

クラス名	atg.commerce.gifts.GiftShippingGroupDroplet
コンポーネント	/atg/commerce/gifts/IsGiftShippingGroup

IsGiftShippingGroup サーブレット Bean は、特定のオーダー内の出荷グループにギフトが含まれているかどうかを確認します。

入力パラメータ

sg（必須）
ギフトの有無が確認される出荷グループ。

出力パラメータ

giftlistId
このパラメータには、出荷グループの一部であるギフトのギフト・リスト ID が格納されます。

オープン・パラメータ

true
出荷グループに 1 つ以上のギフトが含まれているとレンダリングされるオープン・パラメータ。

false
出荷グループにギフトが含まれていないとレンダリングされるオープン・パラメータ。

例

次の例は、IsGiftShippingGroup サーブレット Bean の JSP コードを示しています。

```
<dsp:droplet name="/atg/commerce/gifts/IsGiftShippingGroup">
<dsp:param param="sg" name="sg"/>
<dsp:oparam name="true">
   Gift in shipping group
</dsp:oparam>
<dsp:oparam name="false">
    No gift in shipping group
</dsp:oparam>
<dsp:oparam name="error">
    Error
</dsp:oparam>
</dsp:droplet>
```

GiftShippingGroupsDroplet

クラス名	atg.commerce.gifts.GiftShippingGroupsDroplet
コンポーネント	/atg/commerce/gifts/GiftShippingGroups

GiftShippingGroups サーブレット Bean は、特定のオーダー内にギフトがあるかどうかを確認します。オーダー内にギフトが存在すれば、このサーブレット Bean はギフトが含まれた出荷グループのコレクションを作成します。

入力パラメータ

order（必須）
ギフトの有無が確認されるオーダー。

出力パラメータ

giftsg
1 つ以上のギフトが含まれているオーダー内の出荷グループのコレクション。

allgifts
このパラメータはブール型です。オーダー内のすべての品目がギフトであれば、このパラメータの値は true になります。オーダー内の一部の品目だけがギフトであれば、このパラメータの値は false になります。

オープン・パラメータ

true
オーダーにギフトが含まれているとレンダリングされるオープン・パラメータ。

false
オーダーにギフトが含まれていないとレンダリングされるオープン・パラメータ。

例

次の JSP の例は、GiftShippingGroups サーブレット Bean を呼び出すために使用できるパラメータを示しています。

```
<dsp:droplet name="/atg/commerce/gifts/GiftShippingGroups">
<dsp:param param="order" name="order"/>
<dsp:oparam name="true">
   Gifts in order
</dsp:oparam>
<dsp:oparam name="false">
   No gifts
</dsp:oparam>
<dsp:oparam name="error">
   Error
</dsp:oparam>
</dsp:droplet>
```

GiftWithPurchaseSelectionsDroplet

クラス名	atg.commerce.promotion.GiftWithPurchaseSelectionDroplet
コンポーネント	/atg/commerce/promotion/GiftWithPurchaseSelection

ストアで購入時ギフト・プロモーションを使用する場合は(『ATG Web Commerce マーチャンダイジング・ガイド』を参照)、このドロップレットを使用すると、顧客がまだ決定していないギフト選択を確認できます。

選択は、GiftWithPurchaseSelection Bean のコレクションとして出力されます。この出力を GiftWithPurchaseSelectionChoicesDroplet に渡すと、製品および SKU の選択項目を使用可能にできます。

入力に商品が含まれている場合は、次の記述が適用されます。

- このドロップレットは、その商品の購入時ギフト選択に関する詳細を返します。
- それぞれの数量は、指定された商品にのみ適用されます。
- これはオーダーが入力の場合にのみ適用されるため、quantity および quantityAvailableForSelection はゼロです。

商品が入力されていない場合でも、quantityAvailableForSelection は数量として顧客が選択でき、今後の選択のためのプレースホルダの役割を果たします。通常、アプリケーションは giftType および giftDetail の参照を実行して表示用のセレクタを提供し、GiftWithPurchaseSelectionChoicesDroplet を使用して使用可能な製品選択および SKU 選択を取得します。

入力パラメータ

order
これは、使用するオーダーを渡し、使用可能な購入時ギフト選択を取得するためのオプション・パラメータです。指定されていない場合は、セッションの現在のオーダーが使用されます。

item
これは、商品を渡し、その商品の現在の GWP 選択を取得するためのオプション・パラメータです。指定されていない場合は、そのオーダーに対して行われる選択がかわりに出力されます。

出力パラメータ

selections
これは、`GiftWithPurchaseSelection` Bean のコレクションです。

errorMsg
これは、エラーが発生した場合のエラー・メッセージです。

オープン・パラメータ

output
selections 出力パラメータが空でない場合にレンダリングされます。

empty
selections 出力パラメータが空の場合にレンダリングされます。

error
エラーが発生した場合にレンダリングされます。

例

例として、顧客が 2 つの購入時ギフト・プロモーションに該当していることを前提とします。

- 含まれる無料ギフト: 金のブレスレット SKU 1 点およびノベルティ・カテゴリの品目 2 点
- 含まれる無料ギフト: GWP コンテンツ・グループの品目 3 点

カートには、すでにノベルティのペンが 1 点入れられており、これは自動的に無料となります。金のブレスレットは自動的にカートに追加されます。顧客はすでに GWP コンテンツ項目の 1 つ（商品券）を選択済です。さらに顧客はノベルティ・カテゴリの品目を 1 点、GWP コンテンツ・グループの品目を 2 点選択する必要があります。

商品の入力がない場合、`GiftWithPurchaseSelectionsDroplet` は 2 つの `GiftWithPurchaseSelection` Bean を含むコレクションを出力します。

```
giftType = 'category', giftDetail = 'Novelty', quantity = 3,
automaticQuantity = 1, selectedQuantity = 0, targetedQuantity = 1,
quantityAvailableForSelection = 1

giftType = 'contentGroup', giftDetail = 'GWP', automaticQuantity = 0,
selectedQuantity = 1, targetedQuantity = 0, quantityAvailableForSelection
= 2
```

入力の商品として商品券がある場合、`GiftWithPurchaseSelectionsDroplet` は 1 つの `GiftWithPurchaseSelection` Bean を含むコレクションを出力します。

```
giftType = 'contentGroup', giftDetail = 'GWP', quantity = 0,
automaticQuantity = 0, selectedQuantity = 1, targetedQuantity = 0,
quantityAvailableForSelection = 0
```

GiftWithPurchaseSelectionChoicesDroplet

クラス名	atg.commerce.promotion.GiftWithPurchaseSelectionChoice
コンポーネント	/atg/commerce/promotion/GiftWithPurchaseSelectionChoice

このドロップレットを使用すると、顧客が特定のギフト選択に選択可能な製品選択および SKU 選択を指定できます。

入力には、GiftWithPurchaseSelection Bean またはそのかわりのギフト選択の giftType および giftDetail を指定できます。

選択は、GiftWithPurchaseSelectionChoice Bean のコレクションとして出力されます。

入力パラメータ

selection
giftType および giftDetail を指定するオプションの GiftWithPurchaseSelection Bean パラメータ。指定しなかった場合は、giftType および giftDetail の入力パラメータをかわりに指定する必要があります。

giftType
giftType に渡すオプションの文字列パラメータ。指定しなかった場合は、selection 入力パラメータをかわりに指定する必要があります。

giftDetail
giftDetail に渡すオプションの文字列パラメータ。指定しなかった場合は、selection 入力パラメータをかわりに指定する必要があります。

出力パラメータ

choices
GiftWithPurchaseSelectionChoice Bean のコレクション。

errorMessage
エラーが発生した場合のエラー・メッセージ。

オープン・パラメータ

output
choices 出力パラメータが空ではない場合にレンダリングされます。

empty
choices 出力パラメータが空の場合にレンダリングされます。

error
エラーが発生した場合にレンダリングされます。

HasBusinessProcessStage

クラス名	atg.markers.bp.droplet.HasBusinessProcessStage
コンポーネント	/atg/commerce/bp/droplet/HasShoppingProcessStageDroplet

このサーブレット Bean は、ショッピング・プロセス内の指定されたステージにオーダーが到達したかどうかを確認します。このサーブレット Bean は、businessProcessName プロパティが ShoppingProcess に設定されている HasBusinessProcessStage のインスタンスです。

入力パラメータ

businessProcessName
ビジネス・プロセスの名前。このパラメータが指定されていない場合は、デフォルトで ShoppingProcess になるサーブレット Bean の defaultBusinessProcessName プロパティの値が使用されます。

businessProcessStage（必須）
ビジネス・プロセス内のステージ。トークン値!_anyvalue_!を使用した場合、サーブレット Bean は、任意のビジネス・プロセス・ステージに到達したときに true オープン・パラメータをレンダリングします。

出力パラメータ

errorMsg
失敗を記述するエラー・メッセージ。

オープン・パラメータ

true
ステージに到達するとレンダリングされます。

false
ステージに到達していないとレンダリングされます。

error
エラーが発生するとレンダリングされます。

例

```
<dsp:droplet name="HasShoppingProcessStageDroplet">
  <dsp:param name="businessProcessStage" value="ShippingPriceViewed"/>
  <dsp:oparam name="true">
...
  </dsp:oparam>
  <dsp:oparam name="false">
...
  </dsp:oparam>
</dsp:droplet>
```

InventoryDroplet

クラス名	atg.commerce.inventory.InventoryDroplet
コンポーネント	/atg/commerce/inventory/InventoryLookup

InventoryLookup サーブレット Bean は、指定された品目の在庫情報を返します。

入力パラメータ

itemId（必須）
在庫情報の取得先となる製品カタログ SKU の catalogRefId。

locationId
在庫検索に使用される場所の ID。

useCache
true に設定されていると、キャッシュされた在庫データが取得されます。在庫情報を更新する方法によっては、キャッシュされたデータが古いことがあります。パフォーマンスが重視される一般的なストアの閲覧では、useCache を true に設定する必要があります。取得された在庫情報がリポジトリ内の最新情報と一致している必要がある場合は、useCache を false に設定します。

出力パラメータ

inventoryInfo
取得された在庫情報が格納される情報オブジェクト。inventoryInfo オブジェクトは次のプロパティを持っています。

- availabilityStatus:数値的な可用性ステータス。
- availabilityStatusMsg:次のように数値的な availabilityStatus にマップされる文字列。
- INSTOCK
- OUTOFSTOCK
- PREORDERABLE
- BACKORDERABLE
- availabilityDate:品目が使用可能になる日付。
- stockLevel:現在の在庫の合計数。
- preorderLevel:プレ・オーダー可能な合計数。
- backorderLevel:バック・オーダー可能な合計数。
- stockThreshold:在庫レベルのしきい値。
- preorderThreshold:プレ・オーダー・レベルのしきい値。
- backorderThreshold:バック・オーダー・レベルのしきい値。

error
在庫情報の検索中に発生した可能性がある任意の例外。

オープン・パラメータ

output
在庫情報が正常に取得されるとレンダリングされるオープン・パラメータ。

例

次のコードは、InventoryLookup サーブレット Bean の使用例です。

```
<dsp:droplet name="/atg/commerce/inventory/InventoryLookup"> </td>
   <dsp:param param="link.item.repositoryId" name="itemId"/>
   <dsp:param value="true" name="useCache"/>
   <dsp:oparam name="output">
     This item is
     <dsp:valueof param="inventoryInfo.availabilityStatusMsg"/><br>
     There are
     <dsp:valueof param="inventoryInfo.stockLevel"/>
     left in the inventory.<br>
   </dsp:oparam>
</dsp:droplet>
```

IsHardGoodsDroplet

クラス名	atg.commerce.order.IsHardGoodsDroplet
コンポーネント	/atg/commerce/order/IsHardGoods

IsHardGoods サーブレット Bean は、オーダーを受け取り、ハード製品出荷グループ経由で出荷される品目がオーダーに含まれているかどうかを確認します。

入力パラメータ

order（必須）
商品が含まれている HardgoodShippingGroup の有無が調べられるオーダー・オブジェクト。

出力パラメータ

なし。

オープン・パラメータ

true
オーダーの HardGoodShippingGroup に品目が含まれているとレンダリングされるオープン・パラメータ。

false
オーダーの HardGoodShippingGroup に品目が含まれていないとレンダリングされるオープン・パラメータ。

例

次の例は、IsHardGoods サーブレット Bean の JSP コードを示しています。

```
<dsp:droplet name="/atg/commerce/order/IsHardsGoods">
   <dsp:param param="someorder" name="order"/>
    <dsp:oparam name="true">
       Order contains items in a Hardgood shipping group
    </dsp:oparam>
    <dsp:oparam name="false">
       No items in a Hardgood shipping group
    </dsp:oparam>
</dsp:droplet>
```

IsReturnActiveDroplet

クラス	atg.commerce.csr.returns.IsReturnActive
コンポーネント	/atg/commerce/custsvc/returns/IsReturnActive.properties

IsReturnActive サーブレット Bean は、現在進行中の返品があるかどうかを確認します。このドロップレットを使用すると、サイト・ページで条件に左右されるナビゲーションなどの処理を実行できます。

オープン・パラメータ

true
進行中の返品がある場合にレンダリングされます。

false
進行中の返品がない場合にレンダリングされます。

出力パラメータ

ReturnRequest
oparam が true の場合、このパラメータは現在の ReturnRequest への参照を提供します。

IsReturnableDroplet

クラス	atg.commerce.csr.returns.IsReturnable
コンポーネント	/atg/commerce/custsvc/returns/IsReturnable.properties

IsReturnable サーブレット Bean はオーダーID または商品 ID を受け取り、その商品が返品可能かどうかを判定します。このドロップレットは、オーダーまたは商品の返品可能状態、およびローカライズ済の返品可能状態に関する説明を返します。

入力パラメータ

item
CommerceItem オブジェクトまたはリポジトリ項目。

order
order オブジェクトまたはリポジトリ項目。

オープン・パラメータ

true
商品が返品可能な場合にレンダリングされます。

false
商品が返品可能ではない場合にレンダリングされます。

出力パラメータ

returnableState
商品またはオーダーの返品可能状態の値。

returnableDescription
状態のリソース記述。

ItemLookupDroplet

クラス名	atg.repository.servlet.ItemLookupDroplet
コンポーネント	/atg/content/droplet/ArticleLookup /atg/content/droplet/MediaContentLookup /atg/commerce/catalog/custom/CatalogLookup /atg/commerce/catalog/MediaLookup /atg/commerce/custsvc/returns/DispositionLookup /atg/commerce/custsvc/returns/ReturnReasonLookupDroplet /atg/commerce/gifts/GiftitemLookupDroplet /atg/commerce/gifts/GiftlistLookupDroplet /atg/commerce/location/StoreLookupDroplet /atg/dynamo/service/preview/UserLookupDroplet

ItemLookupDroplet クラスからインスタンス化されたサーブレット Bean は、品目の ID を使用してリポジトリ内で品目を検索し、品目をレンダリングします。そのように構成されているリポジトリについて、ItemLookupDroplet クラスでは、その品目をレンダリングする前にそれが現在のサイト・コンテキストに適し

ているかどうかを判断します。様々な入力パラメータ、出力パラメータおよびオープン・パラメータの詳細は、『ATG Web Commerce Page Developer's Guide』の *付録 B: ATG サーブレット Bean* の ItemLookupDroplet エントリを参照してください。

例

次のコード例では、GiftlistLookupDroplet を使用してリポジトリ内でギフト・リストを検索し、ギフト・リストを表示する前に、そのギフト・リストの所有者 ID が現在のプロファイルの ID と同じであることを確認します。

```
<dsp:droplet name="/atg/commerce/gifts/GiftlistLookupDroplet">
 <dsp:param param="giftlistId" name="id"/>
 <dsp:oparam name="output">
   <dsp:droplet name="IsEmpty">
     <dsp:param param="element" name="value"/>
     <dsp:oparam name="false">
        <dsp:setvalue paramvalue="element" param="giftlist"/>
        <dsp:droplet name="/atg/dynamo/droplet/Switch">
        <dsp:param bean="Profile.id" name="value"/>
        <dsp:getvalueof id="nameval2" param="giftlist.owner.id"
                    idtype="java.lang.String">
<dsp:oparam name="<%=nameval2%>">
        </dsp:oparam>
</dsp:getvalueof>
        </dsp:droplet>
     </dsp:oparam>
   </dsp:droplet>
 </dsp:oparam>
</dsp:droplet>
```

ItemPricingDroplet

クラス名	atg.commerce.pricing.ItemPricingDroplet
コンポーネント	なし。

ItemPricingDroplet は、品目を価格設定し、結果を顧客に表示するための基本クラスとして使用される抽象クラスです。PriceEachItemDroplet と PriceItemDroplet の両方がこのクラスを拡張します。

このクラスを拡張する場合は、performPricing メソッドを上書きして、価格設定された 1 つ以上の CommerceItem を返す必要があります。これらの品目は、その後、デフォルト名 element で output オープン・パラメータにバインドされます。

入力パラメータ

次の *オプション・パラメータ*が許可されています。

pricingModels
品目の価格設定に使用する必要がある価格設定モデル(プロモーション)のコレクション。この値が指定されていないと、デフォルトで、ユーザーの `PricingModelHolder` コンポーネントに格納されている価格設定モデルのコレクションが使用されます。このコンポーネントは、`userPricingModelsPath` プロパティを介して解決されます。

locale
価格設定が行われる場所のロケール。

profile
価格設定が実行されるユーザー。この値が指定されていない場合、プロファイルは `profilePath` プロパティを介して解決されます。

product
価格設定される品目の製品定義を表すオブジェクト。一般に、価格設定される品目は SKU です。その場合、このオブジェクトはすべての SKU が含まれている製品です。

elementName
`output` オープン・パラメータ内で設定されるパラメータとして使用する名前。

出力パラメータ

element
価格設定された 1 つの `CommerceItem` または価格設定された `CommerceItems` のコレクション。このパラメータ名は `elementName` 入力パラメータを設定することによって変更できます。

オープン・パラメータ

output
1 つ以上の `CommerceItem` が正常に価格設定されるとレンダリングされるオープン・パラメータ。

例

なし。`ItemPricingDroplet` は、サブクラス化されるように設計された抽象クラスです。JSP の例については、PriceEachItemDroplet および PriceItemDroplet を参照してください。

MostRecentBusinessProcessStage

クラス名	`atg.markers.bp.droplet.MostRecentBusinessProcessStage`
コンポーネント	`/atg/commerce/bp/droplet/MostRecentShoppingProcessStageDroplet`

このサーブレット Bean は、オーダーが直前に到達したショッピング・プロセス・ステージが指定されたステージと一致するかどうかを確認します。このサーブレット Bean は、`businessProcessName` プロパティが `ShoppingProcess` に設定されている `MostRecentBusinessProcessStage` のインスタンスです。

入力パラメータ

businessProcessName
ビジネス・プロセスの名前。このパラメータが指定されていない場合は、デフォルトで ShoppingProcess になるサーブレット Bean の defaultBusinessProcessName プロパティの値が使用されます。

businessProcessStage（必須）
ビジネス・プロセス内のステージ。

出力パラメータ

errorMsg
失敗を記述するエラー・メッセージ。

marker
指定されたステージと一致する到達したステージ（そのステージが見つかった場合）。

オープン・パラメータ

true
指定されたショッピング・プロセス・ステージに到達しているとレンダリングされます。

false
指定されたショッピング・プロセス・ステージに到達していないとレンダリングされます。

error
エラーが発生するとレンダリングされます。

例

```
<dsp:droplet name="MostRecentShoppingProcessStageDroplet">
  <dsp:param name="businessProcessStage" value="ShippingPriceViewed"/>
  <dsp:oparam name="true">
...
  </dsp:oparam>
  <dsp:oparam name="false">
...
  </dsp:oparam>
</dsp:droplet>
```

NavHistoryCollector

クラス名	atg.repository.servlet.NavHistoryCollector
コンポーネント	/atg/commerce/catalog/CatalogNavHistoryCollector

CatalogNavHistoryCollector サーブレット Bean は、「ブレッドクラム証跡」の作成に使用できます。つまり、このサーブレット Bean は、顧客が現在のページに到達するまでに訪問した品目のリストを作成し、それ

らの品目へのリンクを作成し、表示します。この証跡があれば、顧客は前に訪問した品目に簡単に戻ることができます。

CatalogNavHistoryCollector サーブレット Bean は、顧客が表示したリポジトリ項目を、リポジトリ項目のスタックが格納されている CatalogNavHistory.navHistory プロパティに追加することによって、またはこのプロパティから削除することによって、顧客のナビゲーション・パスを管理します。

入力パラメータ

item（このパラメータまたは itemName を使用する必要があります）
現在表示されているリポジトリ項目。この品目が CatalogNavHistory.navHistory に追加されます。

itemName（このパラメータまたは item を使用する必要があります）
現在のページの名前。このページがブレッドクラムとして表示されている場合は、この名前がページへ戻るリンク内のアンカー・テキストとして使用されます。

navAction
navHistory のスタックに対して実行される操作。プッシュ、ポップおよびジャンプを選択できます。未設定の navAction はプッシュとして扱われます。

navCount（必須）
ナビゲーション・パスをリセットするために「戻る」ボタンの使用を検出する目的に使用されます。この機能を使用するには、各ページの最上部で、navCount という名前のページ・パラメータを CatalogNavHistory.navCount プロパティの値に設定します。たとえば、次のようになります。

```
<param name="navCount" value="bean:CatalogNavHistory.navCount">
```

次に、CatalogNavHistoryCollector サーブレット Bean を呼び出すときに、navCount 入力パラメータの値を CatalogNavHistory.navCount プロパティの現在の値に設定します。
CatalogNavHistoryCollector は、その値を navCount ページ・パラメータの値と比較します。2 つの値が一致すれば、ページに到達するために「戻る」ボタンが使用されています。

出力パラメータ

なし。

オープン・パラメータ

output
なし。

例

次の例は、CatalogNavHistoryCollector を使用してナビゲーション履歴を*収集する*方法を示しています。品目を navHistory スタックに追加する個々のページでこの JSP フラグメントを呼び出す必要があります。

```
<dsp:droplet name="/atg/commerce/catalog/CategoryLookup">
<dsp:param param="itemId" name="id"/>

<dsp:oparam name="output">

   <dsp:droplet name="/atg/commerce/catalog/CatalogNavHistoryCollector">
   <dsp:param param="element" name="item"/>
```

```
   <dsp:param value="push" name="navAction"/>
   <dsp:param bean="/atg/commerce/catalog/CatalogNavHistory.navCount"
              name="navCount"/>
   </dsp:droplet>

</dsp:oparam>
</dsp:droplet>
```

2 番目の例は、顧客が訪問した場所のリストを*レンダリングする*方法を示しています。これらの場所は、「果物」→「かんきつ類」→「オレンジ」などのようにパスとして表示されます。パス内の個々のカテゴリまたは製品名が、対応する品目に戻るためのリンクとして機能します。これらのリンクをいずれかをクリックすると、階層内でその品目の下にある品目がスタックからポップオフされます。

```
<dsp:droplet name="/atg/dynamo/droplet/ForEach">
 <dsp:param bean="CatalogNavHistory.navHistory" name="array"/>
 <dsp:oparam name="output">
    <dsp:getvalueof id="a6" param="element.template.url"
                idtype="java.lang.String">
<dsp:a href="<%=a6%>">
    <dsp:param param="element.repositoryId" name="itemId"/>
    <dsp:param value="pop" name="navAction"/>
    <dsp:param bean="CatalogNavHistory.navCount" name="navCount"/>
    <dsp:valueof param="element.displayName"/>
    </dsp:a></dsp:getvalueof>
 </dsp:oparam>
</dsp:droplet>
```

OnlineOnlyDroplet

このドロップレットを使用して、製品または SKU のストア内引取が可能か、またはオンラインでのみ使用可能かどうかを判定します。

クラス名	atg.commerce.catalog.OnlineOnlyDroplet
コンポーネント	/atg/commerce/catalog/OnlineOnlyDroplet

入力パラメータ

product（sku が指定されていない場合は必須）
商品がオンラインのみで使用可能かどうかを判定する製品リポジトリ項目。

sku（product が指定されていない場合は必須）
商品がオンラインのみで使用可能かどうかを判定する SKU リポジトリ項目。

items
複数の製品または SKU を渡す場合は、このパラメータを使用します。すべての商品がオンラインで使用可能な場合にのみ結果が `true` となります。

出力パラメータ

なし。

オープン・パラメータ

true
商品がオンラインのみで使用可能な場合にレンダリングされます。

false
商品がストアおよびオンラインで使用可能な場合にレンダリングされます。

error
サーブレット Bean の実行時にエラーが発生した場合にレンダリングされます。

OrderHasLastMarker

`OrderHasLastMarker` は、マーカー・マネージャを使用して、特定のオーダーに追加された最終マーカーに指定したパラメータがあるかどうかを判定します。たとえば、`key`、`value`、`data` またはその他のプロパティの特定の値が最終マーカーに含まれているかどうかを確認できます。指定されたパラメータが最終マーカーに含まれている場合は、`true` オープン・パラメータがレンダリングされ、`marker` 出力パラメータを通してマーカーにアクセスできます。

クラス名	`atg.commerce.markers.droplet.OrderHasLastMarker`
コンポーネント	`/atg/commerce/markers/droplet/OrderHasLastMarkerDroplet`

入力パラメータ

key（必須）
マーカーの `key` プロパティの値。最終マーカーには、ここで指定する `key` プロパティの値が設定されている必要があります。

マーカーの `key` 値が使用目的に関係ないことを示す場合は、このパラメータを ANY_VALUE に設定します。たとえば、次のようになります。

```
<dsp:param name="key" bean="OrderHasLastMarkerDroplet.ANY_VALUE"/>
```

order
アクセスする最終マーカーを持つオーダー。省略した場合は、アクティブなオーダーが使用されます。

value
マーカーの `value` プロパティの値。最終マーカーには、ここで指定する `value` プロパティの値が設定されている必要があります。マーカーの `value` プロパティが null の場合にのみマーカーを返すには、このパラメータを `NULL` に設定するか、パラメータを省略します。

マーカーの value が使用目的に関係ないことを示す場合は、このパラメータを ANY_VALUE に設定します。たとえば、次のようになります。

```
<dsp:param name="value" bean="OrderHasLastMarkerDroplet.ANY_VALUE"/>
```

data
マーカーの data プロパティの値。最終マーカーには、ここで指定する data プロパティの値が設定されている必要があります。マーカーの data プロパティが null の場合にのみマーカーを返すには、このパラメータを NULL に設定するか、パラメータを省略します。

マーカーの data 値が使用目的に関係ないことを示す場合は、このパラメータを ANY_VALUE に設定します。たとえば、次のようになります。

```
<dsp:param name="data" bean="OrderHasLastMarkerDroplet.ANY_VALUE"/>
```

extendedProperties
最終オーダー・マーカーを取得するため、最終オーダー・マーカーに存在する必要のある追加のマーカー・プロパティ（マップの key に設定されている）およびプロパティ値（マップの value に設定されている）を指定するマップ。

markerPropertyName
マーカーの有無が確認される項目のプロパティ。省略した場合、サーブレット Bean はその defaultMarkerPropertyName プロパティに指定されているデフォルト値を使用します。

markedItemType
処理するマーカーが含まれる RepositoryItem のタイプ。省略した場合、サーブレット Bean はその項目の defaultMarkedItemType プロパティを使用します。

出力パラメータ

marker
サーブレット Bean によって特定されたマーカーが格納されます。

errorMsg
サーブレット Bean の実行時に生成されたエラー・メッセージが格納されます。

オープン・パラメータ

error
サーブレット Bean の実行時にエラーが発生した場合にレンダリングされます。

true
オーダーに追加された最終マーカーに、指定されたプロパティ値がある場合にレンダリングされます。

false
オーダーに追加された最終マーカーに、指定されたプロパティ値がない場合にレンダリングされます。

OrderHasLastMarkerWithKey

OrderHasLastMarkerWithKey は、特定のキーを持つオーダーに追加された最終マーカーが、指定した値に一致するかどうかを判定します。これらの追加パラメータには、value、data または指定した他のマーカー・プロパティに固有の値を指定できます。特定の key を持つ最終マーカーが、指定した他のパラメータに一致する場合は、true オープン・パラメータがレンダリングされ、marker 出力パラメータを通してページからマーカーを使用できるようになります。

クラス名	atg.commerce.markers.droplet.OrderHasLastMarkerWithKey
コンポーネント	/atg/markers/droplet/OrderHasLastMarkerWithKey Droplet

入力パラメータ

key（必須）
マーカーの key プロパティの値。マーカーの key 値が使用目的に関係ないことを示す場合は、このパラメータを ANY_VALUE に設定します。たとえば、次のようになります。

```
<dsp:param name="key"
bean="OrderHasLastMarkerWithKeyDroplet.ANY_VALUE"/>
```

order
アクセスする最終マーカーを持つオーダー。省略した場合は、アクティブなオーダーが使用されます。

value
マーカーの value プロパティの値。マーカーの value プロパティが null の場合にのみマーカーを返すには、このパラメータを NULL に設定するか、パラメータを省略します。

マーカーの value が使用目的に関係ないことを示す場合は、このパラメータを ANY_VALUE に設定します。たとえば、次のようになります。

```
<dsp:param name="value"
bean="OrderHasLastMarkerWithKeyDroplet.ANY_VALUE"/>
```

data
マーカーの data プロパティの値。マーカーの data プロパティが null の場合にのみマーカーを返すには、このパラメータを NULL に設定するか、パラメータを省略します。

マーカーの data 値が使用目的に関係ないことを示す場合は、このパラメータを ANY_VALUE に設定します。たとえば、次のようになります。

```
<dsp:param name="data"
bean="OrderHasLastMarkerWithKeyDroplet.ANY_VALUE"/>
```

extendedProperties
取得するマーカーに存在する必要のある追加のマーカー・プロパティ（マップの key に設定されている）およびプロパティ値（マップの value に設定されている）を指定するマップ。

markerPropertyName
マーカーの有無が確認されるオーダーのプロパティ。省略した場合、サーブレット Bean はその defaultMarkerPropertyName プロパティに指定されているデフォルト値を使用します。

markedItemType
処理するマーカーが含まれる RepositoryItem のタイプ。省略した場合、サーブレット Bean はその defaultMarkedItemType プロパティに指定されているデフォルト値を使用します。

出力パラメータ

marker
サーブレット Bean によって特定されたマーカーが格納されます。

errorMsg
サーブレット Bean の実行時に生成されたエラー・メッセージが格納されます。

オープン・パラメータ

error
サーブレット Bean の実行時にエラーが発生した場合にレンダリングされます。

true
特定の key を持つ追加された最終マーカーが指定の基準に一致する場合にレンダリングされます。

false
特定の key を持つ追加された最終マーカーが指定の基準に一致しない場合にレンダリングされます。

OrderHasMarker

OrderHasMarker は、マーカー・マネージャを使用して、指定された key、value および data のプロパティ値を持つマーカーがオーダーにあるかどうかを判定します。オーダーにマーカーがあるかどうかをチェックするには、3 つのパラメータをすべて ANY_VALUE に設定します。

クラス名	atg.commerce.markers.droplet.OrderHasMarker
コンポーネント	/atg/commerce/markers/droplet/OrderHasMarkerDroplet

入力パラメータ

key（必須）
マーカーの key プロパティの値。マーカーの key 値が使用目的に関係ないことを示す場合は、このパラメータを ANY_VALUE に設定します。たとえば、次のようになります。

```
<dsp:param name="key" bean="OrderHasMarkerDroplet.ANY_VALUE"/>
```

order
マーカーの有無が確認されるオーダー。省略した場合は、アクティブなオーダーが使用されます。

value
マーカーの value プロパティの値。マーカーの value プロパティを null にする必要がある場合は、このパラメータを NULL に設定するか、パラメータを省略します。

マーカーの value が使用目的に関係ないことを示す場合は、このパラメータを ANY_VALUE に設定します。たとえば、次のようになります。

```
<dsp:param name="value" bean="OrderHasMarkerDroplet.ANY_VALUE"/>
```

data
マーカーの data プロパティの値。マーカーの data プロパティを null にする必要がある場合は、このパラメータを NULL に設定するか、パラメータを省略します。

マーカーの data 値が使用目的に関係ないことを示す場合は、このパラメータを ANY_VALUE に設定します。たとえば、次のようになります。

```
<dsp:param name="data" bean="OrderHasMarkerDroplet.ANY_VALUE"/>
```

extendedProperties
マーカーに存在する必要のある追加のマーカー・プロパティ(マップの key に設定されている)およびプロパティ値(マップの value に設定されている)を指定するマップ。

markerPropertyName
マーカーの有無が確認されるオーダーのプロパティ。省略した場合、サーブレット Bean はその defaultMarkerPropertyName プロパティに指定されているデフォルト値を使用します。

markedItemType
処理するマーカーが含まれる RepositoryItem のタイプ。省略した場合、サーブレット Bean はその defaultMarkedItemType プロパティに指定されているデフォルト値を使用します。

出力パラメータ

errorMsg
サーブレット Bean の実行時に生成されたエラー・メッセージが格納されます。

オープン・パラメータ

error
サーブレット Bean の実行時にエラーが発生した場合にレンダリングされます。

true
指定した基準に一致するマーカーがオーダーに付けられている場合にレンダリングされます。

false
指定した基準に一致するマーカーがオーダーに付けられていない場合にレンダリングされます。

OrderLookup

クラス名	atg.commerce.order.OrderLookup
コンポーネント	/atg/commerce/order/AdminOrderLookup /atg/commerce/order/OrderLookup

OrderLookup サーブレット Bean は、渡されている入力パラメータに応じて、1 つ以上の Order オブジェクトを取得します。このサーブレット Bean を使用すれば、1 つのオーダー、特定のユーザーが確定したすべてのオーダー、または特定のユーザーが確定し、特定の状態にあるすべてのオーダーを取得できます。OrderLookup サーブレット Bean の詳細は、オーダーの取得の実装を参照してください。

OrderLookup は、現在のユーザーが自分のオーダーしか表示できないようにするセキュリティ機能を備えています。デフォルトでは、この機能が/atg/commerce/order/OrderLookup で使用可能になっています。この機能を使用不可にするには、enableSecurity プロパティを false に設定します。

入力パラメータ

orderId(このパラメータまたは userId のどちらかが必須です)
取得されるオーダーの ID。

userId(このパラメータまたは orderId のどちらかが必須です)
取得されるオーダーを持っているユーザー・プロファイルの ID。

quoteOrderHistory
オーダーの見積履歴を取得するにはこのフラグを true に設定する必要があります。見積履歴を返すためにはこのフラグとオーダーID が必要です。

costCenterId
取得されるオーダーを持つコスト・センターの ID。

state
取得されるオーダーの望ましい状態。

このパラメータは userId と組み合せて使用できます。次のいずれかを指定できます。

- atg.commerce.states.OrderStates で定義されている状態のいずれか。
- open
- closed

「open」を指定すると、OrderLookup コンポーネントの openStates プロパティで指定されている状態にあるすべてのオーダーが返されます。デフォルトで、この状態リストは次の値に設定されます。

```
submitted
processing
pending_merchant_action
pending_customer_action
```

「closed」を指定すると、OrderLookup コンポーネントの closedStates プロパティで指定されている状態にあるすべてのオーダーが返されます。デフォルトで、この状態リストは次の値に設定されます。

```
no_pending_action
```

オプションの openStates 入力パラメータまたは closedStates 入力パラメータを使用して、どちらの状態リストも上書きできます。

openStates
「open」状態に対応する状態のカンマ区切りのリスト。

state 入力パラメータが「open」に設定されている場合は、このパラメータを state 入力パラメータと組み合せて使用できます。OrderLookup コンポーネントの openStates プロパティ内の構成されている状態リストを上書きするときに、このオプション・パラメータを使用します。

closedStates
「closed」状態に対応する状態のカンマ区切りのリスト。

state 入力パラメータが「closed」に設定されている場合は、このパラメータを state 入力パラメータと組み合せて使用できます。OrderLookup コンポーネントの closedStates プロパティ内の構成されている状態リストを上書きするときに、このオプション・パラメータを使用します。

sortBy
オーダーをソートするときの基準となる Order プロパティを指定する文字列。

このパラメータは userId と組み合せて使用できます。このパラメータを使用するときは、id、state または submittedDate など、任意のオーダー・リポジトリ・プロパティの名前（つまり、orderrepository.xml で定義されている任意のプロパティの名前）を指定できます。

sortAscending
true または false。このパラメータは sortBy 入力パラメータと組み合せて使用します。true に設定されていると、結果として生成される配列内の Order オブジェクトが、sortBy 入力パラメータで指定されているプロパティを基準にして昇順にソートされます。デフォルト値は false です。

numOrders
特定の問合せに対して返すオーダーの数。

startIndex
結果セット内の最初のオーダーの索引。このパラメータは大量のオーダーの中を循環するときに便利です。

queryTotal
取得されたオーダーの数に基づいて `totalCount` 出力パラメータおよび `total_count` 出力パラメータを介してアクセスできる合計数を計算するかどうかを示します。このプロパティを `false` に設定すると、`queryTotal` プロパティで指定されている値に関係なく、合計数は生成されません。このパラメータを省略すると、デフォルト値である `true` が使用されます。必要なときにだけデータベースに対する問合せが実行されるようにするには、このパラメータを使用します。

queryTotalOnly
オーダーの合計数およびオーダー自体がサーブレット Bean から生成されるかどうかを示します。このパラメータを `true` に設定すると、取得されたオーダーの合計数が `totalCount` 出力パラメータおよび `total_count` 出力パラメータを介して使用可能になります。オーダー自体は取得されず、オーダー自体にもアクセスできません。必要なときにだけデータベースに対する問合せが実行されるようにするには、このパラメータを使用します。

このパラメータを省略することはパラメータを `false` に設定することと同じですが、その場合は、オーダー・オブジェクトのリストが `output` オープン・パラメータに保存されるだけでなく、オーダーの合計数も `totalCount` 出力パラメータと `total_count` 出力パラメータに保存されます。

`queryTotal=false`（オーダーあり、合計なし）かつ `queryTotalOnly=true`（合計あり、オーダーなし）の場合、合計は、`queryTotalOnly` パラメータで指定されたとおりに生成されます。

siteIds
指定されたサイトに関連付けられたオーダーに問合せを限定するために使用されるサイト ID のコレクション。

organizationIds
特定のオーダーに関連付けられた組織の ID。

siteScope
Oracle Commerce Platform の複数サイト機能を使用している場合は、オーダーをサイトでフィルタできます。`siteId` が指定されていない場合は、`siteScope` を使用して次のいずれかのスコープを指定します。

- `null` または `all`: すべてのサイトのオーダーを検索します。
- `current`:サイト・コンテキストによって決まる現在のサイトのオーダーを検索します。
- `shareableTypeId`: `atg.ShoppingCart` など、共有可能タイプの ID を渡して、そのタイプの共有グループ内のすべてのサイトのオーダーを検索します。

デフォルトの `siteScope` は、コンポーネントの構成可能プロパティを介して設定できます。デフォルト値は null です。

出力パラメータ

result
`Order` オブジェクトの配列。`orderId` 入力パラメータが使用された場合、このパラメータには 1 つの `Order` オブジェクトが含まれます。

errorMsg
エラーが発生したときにユーザーに表示される詳細なエラー・メッセージ。

count
`Order` オブジェクトの配列のサイズ。

totalCount
queryTotal プロパティが true に設定されていると、このパラメータは、オーダー・ルックアップの基準を満たすオーダーの合計数を示します。

total_count
（前述の）totalCount 出力パラメータと同じ。

startRange
オーダーの範囲の先頭をマークする索引番号。たとえば、特定の OrderLookup 問合せから 5 個のオーダーが返された場合、startRange は 1 に設定されます。

endRange
オーダーの範囲の最後をマークする索引番号。たとえば、startRange が 6 に設定されている特定の OrderLookup 問合せから 5 個のオーダーが返された場合、endRange は 10 に設定されます。

nextIndex
次の結果セット内の最初のオーダーの索引。startIndex 入力パラメータまたは numOrders 入力パラメータが null だった場合、このパラメータは null になります。

previousIndex
前の結果セット内の最初のオーダーの索引。startIndex 入力パラメータまたは numOrders 入力パラメータが null だった場合、このパラメータは null になります。

オープン・パラメータ

output
オーダーが正常に取得されるとレンダリングされるオープン・パラメータ。

empty
返すオーダーがないとレンダリングされるオープン・パラメータ。

error
エラーが発生するとレンダリングされるオープン・パラメータ。

例

次の例は、OrderLookup サーブレット Bean を使用して現在のユーザーのすべてのオープン・オーダーを取得し、それらの ID を表示する方法を示しています。

```
<dsp:droplet name="/atg/commerce/order/OrderLookup">
 <dsp:param bean="/atg/userprofiling/Profile.repositoryId" name="userId"/>
 <dsp:param value="open" name="state"/>
 <dsp:oparam name="output">
    <dsp:droplet name="/atg/dynamo/droplet/ForEach">
       <dsp:param param="result" name="array"/>
       <dsp:oparam name="outputStart">
          <OL>
       </dsp:oparam>
       <dsp:oparam name="output">
          <LI> <dsp:valueof param="element.id">no order number</dsp:valueof>
       </dsp:oparam>
       <dsp:oparam name="outputEnd">
          </OL>
       </dsp:oparam>
```

```
      <dsp:oparam name="empty">
         No open orders.
      </dsp:oparam>
   </dsp:droplet>
 </dsp:oparam>
 <dsp:oparam name="error">
   <span class=profilebig>ERROR:
      <dsp:valueof param="errorMsg">no error message</dsp:valueof>
   </span>
 </dsp:oparam>
</dsp:droplet>
```

次の例は、OrderLookup サーブレット Bean を使用して、ID が 123 の特定のオーダーに関する情報を表示する方法を示しています。

```
<dsp:droplet name="/atg/commerce/order/OrderLookup">
  <dsp:param value="123" name="orderId"/>
  <dsp:oparam name="error">
    <p>
    ERROR:
    <dsp:valueof param="errorMsg">no error message</dsp:valueof>
    <p>
  </dsp:oparam>
  <dsp:oparam name="output">
    <p>
    order #<dsp:valueof param="result.id">no order id</dsp:valueof>
    <p>
This order is in state:
<dsp:valueof param="result.stateAsString"/>
    <p>
    This order was placed on
      <dsp:valueof date="MMMMM d, yyyy" param="result.submittedDate"/>.
    <p>
  </dsp:oparam>
</dsp:droplet>
```

PaymentGroupDroplet

クラス名	atg.commerce.order.purchase.PaymentGroupDroplet
コンポーネント	/atg/commerce/order/purchase/PaymentGroupDroplet

PaymentGroupDroplet サーブレット Bean は、PaymentGroupFormHandler が使用できるように、ユーザーの PaymentGroups オブジェクトおよび CommerceIdentifierPaymentInfo オブジェクトを初期化するために使用されます。PaymentGroupDroplet サーブレット Bean は、

atg.commerce.order.purchase.PaymentGroupDroplet からインスタンス化されます。PaymentGroupDroplet クラスは次のコンテナから構成されています。

- PaymentGroupMapContainer - ユーザーの名前付き PaymentGroup オブジェクト用のコンテナ。
- CommerceIdentifierPaymentInfoContainer - ユーザーの Order 内の CommerceIdentifier オブジェクトに対応する CommerceIdentifierPaymentInfo オブジェクト用のコンテナ。

これらのコンテナ、PaymentGroupDroplet サーブレット Bean および PaymentGroupFormHandler フォーム・ハンドラの詳細は、『ATG Web Commerce Programming Guide』の「*購買プロセス・サービスの構成*」の *複雑なオーダーを精算する準備*に関する項を参照してください。

入力パラメータ

clear
このパラメータが true に設定されていると、PaymentGroupDroplet はユーザーの CommerceIdentifierPaymentInfoContainer および PaymentGroupMapContainer の両方を消去します。

clearPaymentGroups
このパラメータが true に設定されていると、PaymentGroupDroplet はユーザーの PaymentGroupMapContainer を消去します。Order が確定された後、PaymentGroup オブジェクトが変更される可能性がある（PaymentGroup オブジェクトが ClaimableRepository にあることが原因である可能性が高い）場合は、1 つの Order につき少なくとも 1 回これを実行する必要があります。

clearPaymentInfos
このパラメータが true に設定されていると、PaymentGroupDroplet はユーザーの CommerceIdentifierPaymentInfoContainer を消去します。1 つの Order につき少なくとも 1 回これを実行して、個々の Order の一意の CommerceIdentifier オブジェクトを参照する新規の CommerceIdentifierPaymentInfo オブジェクトを作成する必要があります。

createAllPaymentInfos
このパラメータが true に設定されていると、PaymentGroupDroplet は、ユーザーのプロファイル内のすべての PaymentGroups に対応する OrderPaymentInfo を作成します。このオプションは、潜在的出荷グループの作成で説明しているユーザー・インタフェースとは異なるタイプのユーザー・インタフェースをサポートしています。この UI では、PaymentGroups のリストが含まれたフォームがユーザーに提示されます。ユーザーは個々の PaymentGroup によって支払われる金額をフォームで直接指定し、個々の OrderPaymentInfo オブジェクトの amount プロパティを実質的に設定します。ユーザーが精算プロセス中に別の PaymentGroups を追加した場合は、再度 PaymentGroupDroplet を呼び出して、新規に追加された PaymentGroups に対応する OrderPaymentInfo オブジェクトを作成する必要があります。

このオプションはデフォルトで False に設定されます。

initBasedonOrder
このパラメータが true に設定されていると、PaymentGroupDroplet は、Order 内の個々の PaymentGroup 関係オブジェクトに対応する CommerceIdentifierPaymentInfo オブジェクトを作成します。作成される CommerceIdentifierPaymentInfo オブジェクトのタイプは、PaymentGroup 関係タイプに対応しています。たとえば、Order 内に PaymentGroupCommerceItemRelationship が存在すれば、PaymentGroupDroplet は対応する CommerceItemPaymentInfo オブジェクトを作成し、それを CommerceIdentifierPaymentInfoContainer に追加します。個々の CommerceIdentifierPaymentInfo オブジェクトは、対応する PaymentGroup 関係オブジェクト内に存在する PaymentGroup で初期化されます。

このオプションは、顧客が精算プロセスを実行中で、オーダーにすでに複数の PaymentGroup 関係オブジェクトが含まれている状況に対応するために用意されています。デフォルトで False に設定されます。

initItemPayment
このパラメータが true に設定されていると、PaymentGroupDroplet は、オーダー内の個々の CommerceItem に対応する CommerceItemPaymentInfo オブジェクトを作成し、それらを CommerceIdentifierPaymentInfoContainer に追加します。ユーザーが自分のプロファイルにデフォルトの PaymentGroup を持っていれば、CommerceItemPaymentInfo オブジェクトはその PaymentGroup で初期化されます。デフォルトで False に設定されます。

注意: CommerceItemPaymentInfo オブジェクトは、CommerceIdentifier が CommerceItem である CommerceIdentifierPaymentInfo オブジェクトです。このオブジェクトは CommerceItem 支払情報に使用されます。

initOrderPayment
このパラメータが true に設定されていると、PaymentGroupDroplet は OrderPaymentInfo オブジェクトを作成し、それを CommerceIdentifierPaymentInfoContainer に追加します。ユーザーが自分のプロファイルにデフォルトの PaymentGroup を持っていれば、OrderPaymentInfo オブジェクトはその PaymentGroup で初期化されます。デフォルトで True に設定されます。

注意: OrderPaymentInfo オブジェクトは、CommerceIdentifier が Order である CommerceIdentifierPaymentInfo オブジェクトです。このオブジェクトは Order 支払情報に使用されます。

initPaymentGroups
このパラメータが true に設定されていると、paymentGroupTypes 入力パラメータで指定されている PaymentGroup タイプが初期化されます。

initShippingPayment
このパラメータが true に設定されていると、PaymentGroupDroplet は、オーダー内の個々の ShippingGroup に対応する ShippingGroupPaymentInfo オブジェクトを作成し、それらを CommerceIdentifierPaymentInfoContainer に追加します。ユーザーが自分のプロファイルにデフォルトの PaymentGroup を持っていれば、ShippingGroupPaymentInfo オブジェクトはその PaymentGroup で初期化されます。デフォルトで False に設定されます。

注意: ShippingGroupPaymentInfo オブジェクトは、CommerceIdentifier が ShippingGroup である CommerceIdentifierPaymentInfo オブジェクトです。このオブジェクトは ShippingGroup 支払情報に使用されます。

initTaxPayment
このパラメータが true に設定されていると、PaymentGroupDroplet は TaxPaymentInfo オブジェクトを作成し、それを CommerceIdentifierPaymentInfoContainer に追加します。

注意: TaxPaymentInfo オブジェクトは、CommerceIdentifier が Order である CommerceIdentifierPaymentInfo オブジェクトです。このオブジェクトは税支払情報に使用されます。

order
ユーザーのオーダー。このパラメータを使用して PaymentGroupDroplet.order のデフォルト設定を上書きできます。

paymentGroupTypes
実行される PaymentGroupInitializer コンポーネントを決定するために使用される creditCard、storeCredit、giftCertificate などの PaymentGroup タイプのカンマ区切りのリスト。

PaymentGroupInitializer コンポーネントは、適切な PaymentGroup オブジェクトを作成および初期化し、それらを PaymentGroupMapContainer に追加する役割を担います。使用できる個々の

PaymentGroup タイプは、PaymentGroupInitializer ServiceMap 内の PaymentGroupInitializer コンポーネントを参照するように構成されます。マップへのキーは、この paymentGroupTypes 入力パラメータで指定される Strings です。値は、その PaymentGroup タイプ向けに初期化を実行する、基礎となる PaymentGroupInitializer コンポーネントです。

デフォルトで、PaymentGroupDroplet サーブレット Bean は次の PaymentGroupInitializer コンポーネントで構成されます。

```
## ServiceMap of paymentGroupTypes to PaymentGroupInitializer Nucleus components

paymentGroupInitializers=\
giftCertificate=/atg/commerce/order/purchase/GiftCertificateInitializer,\
storeCredit=/atg/commerce/order/purchase/StoreCreditInitializer,\
creditCard=/atg/commerce/order/purchase/CreditCardInitializer
inStorePayment=/atg/commerce/order/purchase/InStorePaymentInitializer
```

出力パラメータ

paymentInfo
CommerceIdentifierPaymentInfoContainer によって参照されるマップ。

order
ユーザーのオーダーを表す Order オブジェクト。

paymentGroups
PaymentGroupMapContainer によって参照されるマップ。

オープン・パラメータ

output
常にレンダリングされるオープン・パラメータ。

例

この例では、現在のユーザーにとっての可用性に基づいて、CreditCard、StoreCredit および GiftCertificate の PaymentGroup オブジェクトを作成します。さらに、CommerceItemPaymentInfo オブジェクト、ShippingGroupPaymentInfo オブジェクトおよび TaxPaymentInfo オブジェクトも作成します。この例では、ユーザーが、使用可能な任意の PaymentGroup オブジェクトを使用して CommerceIdentifiers の代金を明細項目レベルで支払うことができます。

```
<dsp:droplet name="PaymentGroupDroplet">
  <dsp:param value="true" name="clear"/>
  <dsp:param value="giftCertificates, storeCredit, creditCard"
             name="paymentGroupTypes"/>
  <dsp:param value="true" name="initPaymentGroups"/>
  <dsp:param value="true" name="initItemPayment"/>
  <dsp:param value="true" name="initTaxPayment"/>
  <dsp:param value="true" name="initShippingPayment"/>
  <dsp:oparam name="output">Manipulation of objects here…
  </dsp:output>
</dsp:droplet>
```

PossibleValues

クラス名	atg.repository.servlet.PossibleValues
コンポーネント	/atg/commerce/catalog/RepositoryValues

RepositoryValues サーブレット Bean は、リポジトリに対して問合せを実行し、特定のリポジトリ項目タイプに使用できる値の配列を返します。

RepositoryValues サーブレット Bean は、atg.repository.servlet.PossibleValues クラスからインスタンス化されます。同じクラスからインスタンス化される PossibleValues サーブレット Bean は、Oracle Commerce Platform に付属しています。このサーブレット Bean の様々なパラメータの詳細および JSP 例は、『ATG Web Commerce Page Developer's Guide』の *付録 B: ATG サーブレット Bean* を参照してください。

PriceDroplet

クラス名	atg.commerce.pricing.priceLists.PriceDroplet
コンポーネント	/atg/commerce/pricing/priceLists/PriceDroplet

PriceDroplet サーブレット Bean は、特定の製品または SKU の価格を返します。PricingEngine を使用して実際に 1 つの品目の価格を計算する PriceItemDroplet と PriceDroplet を混同しないでください。

入力パラメータ

product（このパラメータまたは sku を使用する必要があります）
価格が求められている製品。

sku（このパラメータまたは product を使用する必要があります）
価格が求められている SKU。

parentSku
価格設定される SKU が構成可能な SKU のオプションである場合は、このパラメータが基本 SKU 品目であり、そのコンテキスト内で構成可能なオプションを価格設定する必要があります。

priceList
価格の取得先 priceList。

出力パラメータ

price
価格リポジトリ項目。

error
価格を取得するときに発生したエラー。

オープン・パラメータ

output
製品または SKU の価格が正常に取得されるとレンダリングされます。

empty
特定の製品または SKU に価格がない場合にレンダリングされます。価格表と SKU ベースの価格設定を組み合せて使用している場合は、品目に価格表の価格がないときに、このパラメータを使用して SKU ベースの価格をレンダリングできます(この機能を構成する方法については、『ATG Web Commerce Programming Guide』の *SKU をベースとした価格設定と価格表との組合せ*に関する項を参照してください)。

例

次の例は、PriceDroplet サーブレット Bean の JSP コードを示しています。

```
<dsp:droplet name="/atg/commerce/pricing/priceLists/PriceDroplet">
   <dsp:param name="sku" value="sku" />
   <dsp:param name="product" value="product" />
   <dsp:oparam name="output">
      <dsp:droplet name="/atg/dynamo/droplet/Switch">
         <dsp:param name="value" param="price.pricingScheme" />
         <dsp:oparam name="listPrice">
            <dsp:valueof param="price.listPrice" converter="currency">no
             price</dsp:valueof>
         </dsp:oparam>
      </dsp:droplet>
   </dsp:oparam>
</dsp:droplet>
```

PriceEachItemDroplet

クラス名	atg.commerce.pricing.PriceEachItemDroplet
コンポーネント	/atg/commerce/pricing/PriceEachItem

PriceEachItem サーブレット Bean は、プロモーションを考慮に入れることによって品目のコレクションを動的に価格設定することを目的とします。静的な価格しか表示する必要がない場合は、SKU オブジェクトから直接定価または販売価格を取得できます。

PriceEachItem サーブレット Bean は、atg.commerce.pricing.ItemPricingDroplet を拡張する atg.commerce.pricing.PriceEachItemDroplet からインスタンス化されます。

品目の*コレクション*の動的な価格設定に使用できるサーブレット Bean については、PriceItemDroplet を参照してください。

入力パラメータ

items（必須）
価格設定される品目を表す RepositoryItems のコレクション、または直接価格設定できる CommerceItems のコレクションのどちらかです。渡された品目が RepositoryItems の場合は、新規の CommerceItems のコレクションが作成されます。

pricingModels
品目の価格設定に使用される価格設定モデル（プロモーション）のコレクション。この値が指定されていないと、デフォルトで、顧客の PricingModelHolder コンポーネントに格納されている価格設定モデルのコレクションが使用されます。このコンポーネントは、userPricingModelsPath プロパティを介して解決されます。

locale
価格設定が行われる場所のロケール。

profile
価格設定が実行されるユーザー。この値が指定されていない場合、プロファイルは profilePath プロパティを介して解決されます。

product
価格設定される品目の製品定義を表すオブジェクト。通常、価格設定される品目は SKU です。その場合、このオブジェクトはすべての SKU が含まれている製品です。

elementName
output オープン・パラメータ内で設定されるパラメータとして使用する名前。

出力パラメータ

element
価格設定された CommerceItems のコレクション。このパラメータ名は elementName 入力パラメータを設定することによって変更できます。

オープン・パラメータ

output
CommerceItems が正常に価格設定されるとレンダリングされるオープン・パラメータ。

例

次の例では、プロモーション、ロケールおよびプロファイルがパラメータとして指定されていないため、要求から抽出されます。

```
<dsp:droplet name="/atg/commerce/pricing/PriceEachItem">
 <dsp:param param="product.childSKUs" name="items"/>
 <!-- the product param is already defined in this scope so we do not
  need to set it -->
 <dsp:oparam name="output">
 <!-- Now iterate over each of the CommerceItems to display the prices -->
  <dsp:droplet name="/atg/dynamo/droplet/ForEach">
   <dsp:param param="element" name="array"/>
   <dsp:param value="pricedItem" name="elementName"/>
   <dsp:oparam name="output">
    <dsp:valueof param="pricedItem.auxiliaryData.catalogRef.displayName"/> -
 <!-- Toggle a different display depending if the item is on sale or not -->
```

```
    <dsp:droplet name="Switch">
    <dsp:param param="pricedItem.priceInfo.onSale" name="value"/>
    <dsp:oparam name="false">
     <dsp:valueof param="pricedItem.priceInfo.amount" converter="currency">
                        no price</dsp:valueof>
    </dsp:oparam>
    <dsp:oparam name="true">
     List price for <dsp:valueof param="pricedItem.priceInfo.listPrice"
                              converter="currency">no price
            </dsp:valueof>
                 on sale for <dsp:valueof param="pricedItem.priceInfo.salePrice"
                                      converter="currency"/>!
    </dsp:oparam>
    </dsp:droplet><BR>
   </dsp:oparam>
  </dsp:droplet>
 </dsp:oparam>
</dsp:droplet>
```

PriceItemDroplet

クラス名	atg.commerce.pricing.PriceItemDroplet
コンポーネント	/atg/commerce/pricing/PriceItem

PriceItem サーブレット Bean は、プロモーションを考慮に入れることによって 1 つの品目を動的に価格設定するために使用されます。このサーブレット Bean は、オーダー依存のプロモーションではなく、品目レベルで適用されるプロモーションのみを適用することに注意してください。静的な価格しか表示する必要がない場合は、SKU オブジェクトから直接定価または販売価格を取得できます。

PriceItem サーブレット Bean は、atg.commerce.pricing.ItemPricingDroplet を拡張する atg.commerce.pricing.PriceItemDroplet からインスタンス化されます。

品目の*コレクション*の動的な価格設定に使用できるサーブレット Bean については、PriceEachItemDroplet を参照してください。

入力パラメータ

item（必須）
価格設定される品目を表す RepositoryItem または直接価格設定できる CommerceItem のどちらかです。渡された品目が RepositoryItem の場合は、価格設定用として新規の CommerceItem が作成されます。

pricingModels
品目の価格設定に使用される価格設定モデル（プロモーション）のコレクション。この値が指定されていないと、デフォルトで、ユーザーの PricingModelHolder コンポーネントに格納されている価格設定モデルのコレクションが使用されます。このコンポーネントは、userPricingModelsPath プロパティを介して解決されます。

locale
価格設定が行われる場所のロケール。

profile
価格設定が実行されるユーザー。この値が指定されていない場合、プロファイルは profilePath プロパティを介して解決されます。

product
価格設定される品目の製品定義を表すオブジェクト。一般に、価格設定される品目は SKU です。その場合、このオブジェクトは特定の SKU を持つ製品です。

elementName
output オープン・パラメータ内で設定されるパラメータとして使用する名前。

quantity
価格設定される入力された製品の Long 型の数量。このパラメータは、提供された情報に基づいて CommerceItem を作成するときに使用されます。

出力パラメータ

element
価格設定された CommerceItem。このパラメータ名は elementName 入力パラメータを設定することによって変更できます。

オープン・パラメータ

output
CommerceItem が正常に価格設定されるとレンダリングされるオープン・パラメータ。

例

次の例では、プロモーション、ロケールおよびプロファイルがパラメータとして指定されていないため、要求から抽出されます。

```
<dsp:droplet name="/atg/commerce/pricing/PriceItem">
<dsp:param param="sku" name="item"/>
<dsp:param param="product" name="product"/>
<dsp:param value="pricedItem" name="elementName"/>
<dsp:oparam name="output">
    <dsp:valueof param="pricedItem.priceInfo.amount" converter="currency">
                        no price</dsp:valueof>
</dsp:oparam>
</dsp:droplet>
```

PriceRangeDroplet

クラス名	atg.commerce.pricing.PriceRangeDroplet
コンポーネント	/atg/commerce/pricing/PriceRangeDroplet

製品を指定されると、このドロップレットは、その製品に関連付けられた SKU の範囲内の最高価格と最低価格を確定します。

入力パラメータ

productId（必須）
価格設定する必要のある製品リポジトリ項目の ID。

pricelist（オプション）
価格設定に使用される価格表。このパラメータが設定されていない場合は、プロファイルの割当済価格表が使用されます。

salePriceList（オプション）
価格設定に使用される販売価格表。このパラメータが設定されていない場合は、プロファイルの割当済販売価格表が使用されます。

出力パラメータ

lowestPrice
見つかった最低価格を表す Double 型の値。

highestPrice
見つかった最高価格を表す Double 型の値。

オープン・パラメータ

output
- 常にサービスされます。

例

次の例は、PriceRangeDroplet の使用方法を示しています。

```
<dsp:droplet name="/atg/commerce/custsvc/pricing/PriceRangeDroplet">
   <dsp:param name="productId" value="productId">
   <dsp:param name="priceList" bean="priceList">
   <dsp:oparam name="output">
      <dsp:valueof param="lowestPrice">no price no price
```

ProductListContains

クラス名	atg.commerce.catalog.comparison.ProductListContains
コンポーネント	/atg/commerce/catalog/comparison/ProductListContains

カテゴリ、製品および SKU を指定すれば、ProductListContains サーブレット Bean は、製品比較リストにその製品が含まれているかどうかを問い合せます。

入力パラメータ

productList（必須）
検査の対象となる ProductComparisonList オブジェクト。

productID（必須）
productList 内で検索する製品のリポジトリ ID。

categoryID
productList 内で検索するカテゴリのリポジトリ ID。

カテゴリ ID を指定しない場合、ProductListContains は、category プロパティが指定された製品のデフォルト・カテゴリと一致するリスト・エントリを検索し、指定された製品のデフォルト・カテゴリがない場合は、category プロパティが null であるリスト・エントリを検索します。

skuID
productList 内で検索する SKU のリポジトリ ID。

SKU を指定しない場合、ProductListContains は、sku プロパティが指定された製品の最初の子 SKU と一致するリスト・エントリを検索し、指定された製品の SKU がない場合は、sku プロパティが null であるリスト・エントリを検索します。

repositoryKey
検索する品目が含まれている製品カタログ・リポジトリを選択するために CatalogTools に渡すキー。キーからカタログへのマッピングは、CatalogTools で定義されます。このパラメータが設定されていない場合は、デフォルトの製品カタログ・リポジトリが使用されます。

このオプション・パラメータは、多くの場合、ロケールに合わせて代替製品カタログを使用する必要のあるローカライゼーションで役立ちます。

出力パラメータ

なし。

オープン・パラメータ

true
製品比較リストに特定の製品、カテゴリおよび SKU が含まれているとレンダリングされます。

false
製品比較リストに特定の製品、カテゴリおよび SKU が含まれていないとレンダリングされます。

例

この JSP 例は、製品比較リストで 1 つの製品を追加または削除する方法を示しています。この例では、ProductListContains サーブレット Bean が、次のフラグメントを使用して製品表示ページに埋め込まれていることを前提としています。

```
<dsp:include page="example.jsp"><dsp:param name="product"
            value="current product"/></dsp:include>
```

current product は、ページに表示される製品へのアクセスを提供する式です。

指定された製品は productId 入力パラメータを介してサーブレット Bean に渡されます。次に、ProductListContains サーブレット Bean は、その製品が ProductList 内の製品比較リストに格納されているかどうかを確認します。

製品比較リストに製品がある場合、このサーブレット Bean は output オープン・パラメータを製品表示ページ上にレンダリングします。その場合、ユーザーは、比較リストからの削除発行ボタンをクリックして、製品をリストから削除できます。製品比較リストに製品がなければ、このサーブレット Bean は false オープン・パラメータを製品表示ページ上にレンダリングします。その場合、ユーザーは、比較リストへの追加発行ボタンをクリックして、製品をリストに追加できます。

```
<dsp:importbean bean="/atg/commerce/catalog/comparison/ProductList"/>
<dsp:importbean bean="/atg/commerce/catalog/comparison/ProductListContains"/>
<dsp:importbean bean="/atg/commerce/catalog/comparison/ProductListHandler"/>

<dsp:form action="product.jsp" method="POST">
<dsp:droplet name="ProductListContains">
 <dsp:param bean="ProductList" name="productList"/>
 <dsp:param param="product.repositoryId" name="productID"/>

 <dsp:oparam name="true">
  <dsp:input bean="ProductListHandler.productID" paramvalue="productID"
             type="hidden"/>
  <dsp:input bean="ProductListHandler.removeProduct" value="Remove from comparison
       list" type="submit"/>
 </dsp:oparam>

 <dsp:oparam name="false">
  <dsp:input bean="ProductListHandler.productID" paramvalue="productID"
             type="hidden"/>
  <dsp:input bean="ProductListHandler.addProduct" value="Add to
   comparison list" type="submit"/>
 </dsp:oparam>

</dsp:droplet>
</dsp:form>
```

PromotionDroplet

クラス名	atg.commerce.promotion.PromotionDroplet
コンポーネント	/atg/commerce/promotion/PromotionDroplet

PromotionDroplet サーブレット Bean は、プロモーションをユーザー・プロファイルに関連付けます。プロモーションは、ユーザー・プロファイルの activePromotions プロパティ内のプロモーション・リストに追加されます。

入力パラメータ

promotion（必須）
プロファイルに関連付けられるプロモーション。このパラメータの値は、RepositoryItem タイプである必要があります。

profile
プロモーションに関連付けられるユーザー・プロファイル。このパラメータが指定されていないか、Profile のインスタンスでない場合、パラメータは Nucleus から解決されます。

出力パラメータ

error
プロモーションをユーザー・プロファイルに関連付けている最中に発生した例外。

オープン・パラメータ

error
このパラメータは、エラーが発生するとレンダリングされます。

例

```
<dsp:droplet name="/atg/commerce/promotion/PromotionDroplet">
  <dsp:param param="someIncomingPromotion" name="promotion"/>
</dsp:droplet>
```

ReanimateAbandonedOrderDroplet

クラス名	atg.commerce.order.abandoned.ReanimateAbandonedOrderDroplet
コンポーネント	/atg/commerce/order/abandoned/ReanimateAbandonedOrderDroplet （Abandoned Order Services モジュールのみ）

ReanimateAbandonedOrderDroplet サーブレット Bean は、放棄されたオーダーまたは失われたオーダーを復活させます。より具体的に説明すれば、このサーブレット Bean は次のことを実行します。

1. オーダーが放棄されており、失われていない場合に、ユーザーの abandonedOrders プロファイル・プロパティ内の放棄されたオーダーのリストからそのオーダーを削除します。
2. オーダーの abandonmentInfo 項目を次のように変更します。
   - state プロパティを REANIMATED に設定します。
   - reanimationDate プロパティを現在の日付、時刻に設定します。
3. AbandonedOrderTools.sendOrderReanimatedMessage プロパティが true に設定されていれば、AbandonedOrderReanimated メッセージを生成します。

指定されたオーダーが放棄されていない場合または失われていない場合、この処理は何も実行しないことに注意してください。

Abandoned Order Services モジュールの詳細は、『ATG Web Commerce Programming Guide』の*放棄されたオーダー・サービスの使用*を参照してください。

入力パラメータ

orderId（必須）
現在のオーダーの ID。

出力パラメータ

なし。

オープン・パラメータ

output
このパラメータは、オーダーが復活するとレンダリングされます。

error
このパラメータは、エラーが発生するとレンダリングされます。

例

```
<dsp:droplet name="ReanimateAbandonedOrderDroplet">
  <dsp:param name="orderId"
   bean="/atg/commerce/ShoppingCart.current.id"/>…
</dsp:droplet>
```

RemoveMarkersFromItem

このドロップレットは、リポジトリ項目からマーカーを削除します。マーカーは、一致する key、value、data および extendedProperties によって識別されます。指定されたすべての値が、削除対象のマーカーに一致する必要があります。拡張プロパティ・マップは、ドロップレットの入力パラメータまたは getExtendedProperties メソッドの拡張によって指定できます。

クラス名	atg.markers.droplet.RemoveMarkersFromItem
コンポーネント	/atg/markers/droplet/RemoveMarkersFromItem Droplet

入力パラメータ

itemId（必須）
削除対象のマーカーを持つ項目 ID。省略した場合は、item 入力パラメータに指定された値が使用されます。

item
削除対象のマーカーを持つ項目。省略した場合は、アクティブな項目が使用されます。

key（必須）
マーカーの key プロパティの値。マーカーの key 値が使用目的に関係ないことを示す場合は、このパラメータを ANY_VALUE に設定します。たとえば、次のようになります。

```
<dsp:param name="key" bean="OrderHasMarkerDroplet.ANY_VALUE"/>
```

value
マーカーの value プロパティの値。マーカーの value プロパティを null にする必要がある場合は、このパラメータを NULL に設定するか、パラメータを省略します。

マーカーの value 値が使用目的に関係ないことを示す場合は、このパラメータを ANY_VALUE に設定します。たとえば、次のようになります。

```
<dsp:param name="value" bean="OrderHasMarkerDroplet.ANY_VALUE"/>
```

data
マーカーの data プロパティの値。マーカーの data プロパティを null にする必要がある場合は、このパラメータを NULL に設定するか、パラメータを省略します。

マーカーの data 値が使用目的に関係ないことを示す場合は、このパラメータを ANY_VALUE に設定します。たとえば、次のようになります。

```
<dsp:param name="data" bean="OrderHasMarkerDroplet.ANY_VALUE"/>
```

extendedProperties
取得するマーカーに存在する必要のある追加のマーカー・プロパティ（マップの key に設定されている）およびプロパティ値（マップの value に設定されている）を指定するマップ。

markerManager
使用するマーカー・マネージャ・コンポーネント。省略した場合は、サーブレット Bean の repositoryMarkerManager プロパティで指定されたマーカー・マネージャが使用されます。

markerPropertyName
削除対象のマーカーを持つプロファイルのプロパティ。省略した場合、サーブレット Bean はその defaultMarkerPropertyName プロパティに指定されているデフォルト値を使用します。

markedItemType
処理するマーカーが含まれる RepositoryItem のタイプ。省略した場合、サーブレット Bean はその defaultMarkedItemType プロパティに指定されているデフォルト値を使用します。

出力パラメータ

markerCount
このサーブレット Bean の実行によって削除されたマーカーの数が格納されます。

errorMsg
サーブレット Bean の実行時に生成されたエラー・メッセージが格納されます。

オープン・パラメータ

output
すべてのマーカーが項目から削除された場合にレンダリングされます。

error
サーブレット Bean の実行時にエラーが発生した場合にレンダリングされます。

RemoveMarkersFromOrder

このドロップレットは、オーダーからマーカー項目を削除します。拡張プロパティ・マップは、ドロップレットの入力パラメータまたは `getExtendedProperties` メソッドの拡張によって指定できます。

クラス名	`atg.commerce.markers.droplet.RemoveMarkersFromOrder`
コンポーネント	`/atg/commerce/markers/droplet/RemoveMarkersFromOrder Droplet`

入力パラメータ

order
削除対象のマーカーを持つオーダー。省略した場合は、アクティブなオーダーが使用されます。

key（必須）
マーカーの `key` プロパティの値。マーカーの `key` 値が使用目的に関係ないことを示す場合は、このパラメータを ANY_VALUE に設定します。たとえば、次のようになります。

```
<dsp:param name="key" bean="OrderHasMarkerDroplet.ANY_VALUE"/>
```

value
マーカーの `value` プロパティの値。マーカーの `value` プロパティを null にする必要がある場合は、このパラメータを `NULL` に設定するか、パラメータを省略します。

マーカーの `value` 値が使用目的に関係ないことを示す場合は、このパラメータを ANY_VALUE に設定します。たとえば、次のようになります。

```
<dsp:param name="value" bean="OrderHasMarkerDroplet.ANY_VALUE"/>
```

data
マーカーの `data` プロパティの値。マーカーの `data` プロパティを null にする必要がある場合は、このパラメータを `NULL` に設定するか、パラメータを省略します。

マーカーの `data` 値が使用目的に関係ないことを示す場合は、このパラメータを ANY_VALUE に設定します。たとえば、次のようになります。

```
<dsp:param name="data" bean="OrderHasMarkerDroplet.ANY_VALUE"/>
```

extendedProperties
取得するマーカーに存在する必要のある追加のマーカー・プロパティ(マップの key に設定されている)およびプロパティ値(マップの value に設定されている)を指定するマップ。

markerPropertyName
削除対象のマーカーを持つオーダーのプロパティ。省略した場合、サーブレット Bean はその defaultMarkerPropertyName プロパティに指定されているデフォルト値を使用します。

出力パラメータ

markerCount
このサーブレット Bean の実行によって削除されたマーカーの数が格納されます。

errorMsg
サーブレット Bean の実行時に生成されたエラー・メッセージが格納されます。

オープン・パラメータ

output
すべてのマーカーが項目から削除された場合にレンダリングされます。

error
サーブレット Bean の実行時にエラーが発生した場合にレンダリングされます。

RemoveAllMarkersFromItem

RemoveAllMarkersFromItem は、マーカー・マネージャと連携して、特定の項目からすべてのマーカーを削除します。

クラス名	atg.markers.droplet.RemoveAllMarkersFromItem
コンポーネント	/atg/markers/userprofiling/droplet/RemoveAllMarkersFromItem Droplet

入力パラメータ

itemId
削除対象のマーカーを持つ項目 ID。省略した場合は、item 入力パラメータに指定された値が使用されます。

item
削除対象のマーカーを持つ項目。省略した場合は、アクティブな項目が使用されます。

markerManager
使用するマーカー・マネージャ・コンポーネント。省略した場合は、サーブレット Bean の repositoryMarkerManager プロパティで指定されたマーカー・マネージャが使用されます。

markerPropertyName
削除対象のマーカーを持つプロファイルのプロパティ。省略した場合、サーブレット Bean はその defaultMarkerPropertyName プロパティに指定されているデフォルト値を使用します。

出力パラメータ

markerCount
このサーブレット Bean の実行によって削除されたマーカーの数が格納されます。

errorMsg
サーブレット Bean の実行時に生成されたエラー・メッセージが格納されます。

オープン・パラメータ

output
すべてのマーカーが項目から削除された場合にレンダリングされます。

error
サーブレット Bean の実行時にエラーが発生した場合にレンダリングされます。

例

次の例では、現在の項目からすべてのマーカーが削除されます。マーカーが削除されると、削除されたマーカーの数を示すメッセージがユーザーに表示されます。

```
<dsp:droplet
 name="/atg/markers/droplet/RemoveAllMarkersFromItemDroplet">

   <dsp:param name="output">
      There were <dsp:param name="markerCount"/> markers removed from your
      item.
   </dsp:param>

</dsp:droplet>
```

RemoveAllMarkersFromOrder

RemoveAllMarkersFromOrder は、マーカー・マネージャと連携して特定のオーダーからすべてのマーカーを削除します。

クラス名	atg.commerce.markers.droplet.RemoveAllMarkersFromOrder
コンポーネント	/atg/commerce/markers/droplet /RemoveAllMarkersFromOrder Droplet

入力パラメータ

order
削除対象のマーカーを持つオーダー。省略した場合は、アクティブなオーダーが使用されます。

markerPropertyName
削除対象のマーカーを持つオーダーのプロパティ。省略した場合、サーブレット Bean はその defaultMarkerPropertyName プロパティに指定されているデフォルト値を使用します。

出力パラメータ

markerCount
このサーブレット Bean の実行によって削除されたマーカーの数が格納されます。

errorMsg
サーブレット Bean の実行時に生成されたエラー・メッセージが格納されます。

オープン・パラメータ

output
すべてのマーカーがオーダーから削除された場合にレンダリングされます。

error
サーブレット Bean の実行時にエラーが発生した場合にレンダリングされます。

RemoveBusinessProcessStage

クラス名	atg.markers.bp.RemoveBusinessProcessStage
コンポーネント	/atg/commerce/bp/droplet/RemoveShoppingProcessStageDroplet

このサーブレット Bean は、指定されたステージと一致する既存のショッピング・プロセス・ステージ・マーカーを削除します。このサーブレット Bean は、businessProcessName プロパティが ShoppingProcess に設定されている RemoveBusinessProcessStage のインスタンスです。

入力パラメータ

businessProcessName
ビジネス・プロセスの名前。このパラメータが指定されていない場合は、デフォルトで ShoppingProcess になるサーブレット Bean の defaultBusinessProcessName プロパティの値が使用されます。

businessProcessStage（必須）
ビジネス・プロセス内のステージ。値!_anyvalue_!を使用すると、すべてのビジネス・プロセス・ステージ・マーカーが削除されます。

出力パラメータ

errorMsg
失敗を記述するエラー・メッセージ。

markerCount
削除されたステージ到達マーカーの数。

オープン・パラメータ

output
このパラメータは正常終了するとレンダリングされます。

error
このパラメータはエラーが発生するとレンダリングされます。

例

次の例では、ShippingPriceViewed ステージを削除します。

```
<dsp:droplet name="RemoveShoppingProcessStageDroplet">
  <dsp:param name="businessProcessStage" value="ShippingPriceViewed"/>
</dsp:droplet>
```

次の例では、見つかったすべてのショッピング・プロセス・ステージ・マーカーを削除します。

```
<dsp:droplet name="RemoveShoppingProcessStageDroplet">
  <dsp:param name="businessProcessStage" value="!_anyvalue_!"/>
</dsp:droplet>
```

RepriceOrder

クラス名	atg.commerce.order.purchase.RepriceOrder
コンポーネント	/atg/commerce/order/purchase/RepriceOrderDroplet

RepriceOrderDroplet サーブレット Bean は、atg.service.pipeline.servlet.PipelineChainInvocation を拡張する atg.commerce.order.purchase.RepriceOrder のインスタンスです。RepriceOrder クラスは、再価格設定パイプライン・チェーンの実行に必要なオブジェクトを便利なプロパティとして備えています。一般に、再価格設定パイプライン・チェーンの実行には、Order、Profile、OrderManager およびユーザーの PricingModelHolder が必要です。PipelineChainInvocation クラスは、これらの要件を処理するのに十分な柔軟性（および必要に応じてマップでプロパティを構成する機能）を備えており、RepriceOrder は、それらのオブジェクトを参照するようにうまく構成されています。つまり、ページ開発者は、RepriceOrderDroplet が呼び出されるたびに、これらのオブジェクトを入力パラメータとして渡す必要がありません。

デフォルトで、RepriceOrderDroplet は、repriceOrder パイプライン・チェーンを呼び出してオーダーを再価格設定するように構成されています。そのように、このサーブレット Bean は、顧客がショッピング・カート・ページにアクセスするたびにオーダーの価格を更新するメカニズムを備えています。運営しているサイトで、標準のハイパーリンクなど、フォーム以外の処理を介して顧客が自分のショッピング・カートにアクセスできる場合は、この機能が便利です。価格設定が動的に行われるため、顧客がハイパーリンクを介してショッピング・カートにアクセスすると、自分のショッピング・カートに不正確な価格が表示される可能性があります。

オーダーの再価格設定の詳細は、『ATG Web Commerce Programming Guide』の「*購買プロセス・サービスの構成*」の*オーダーの再価格設定*および*オーダーの精算*に関する項を参照してください。「*Implementing Shopping Carts*」のショッピング・カートの再価格設定に関する項も参照してください。repriceOrder パイプライン・チェーンの詳細は、『ATG Web Commerce Programming Guide』を参照してください。

入力パラメータ

pricingOp（必須） 実行される価格設定操作。実行可能な価格設定操作は atg.commerce.pricing.PricingConstants インタフェースで定義されています。次の表は、それらの操作の一覧です。

価格設定操作	価格設定定数
ORDER_TOTAL	PricingConstants.OP_REPRICE_ORDER_TOTAL
ORDER_SUBTOTAL	PricingConstants.OP_REPRICE_ORDER_SUBTOTAL
ORDER_SUBTOTAL_SHIPPING	PricingConstants.OP_REPRICE_ORDER_SUBTOTAL_SHIPPING
ORDER_SUBTOTAL_TAX	PricingConstants.OP_REPRICE_ORDER_SUBTOTAL_TAX
ITEMS	PricingConstants.OP_REPRICE_ITEMS
SHIPPING	PricingConstants.OP_REPRICE_SHIPPING
ORDER	PricingConstants.OP_REPRICE_ORDER
TAX	PricingConstants.OP_REPRICE_TAX
NO_REPRICE	PricingConstants.OP_NO_REPRICE

priceList
使用される価格表の ID。このパラメータは指定しないが、価格表を使用してオーダーの価格を決定する場合は、アクティブ・ユーザーに対して指定された価格表が使用されます。このパラメータを利用すれば、代替の価格表を使用できます。

chainId
実行されるパイプライン・チェーンの ID。このパラメータが設定されていない場合、このサーブレット Bean は、defaultChainId プロパティ内の構成されているパイプライン・チェーン ID を検索します。デフォルトで、RepriceOrderDroplet.defaultChainId は repriceOrder に設定されます。

paramObject
パイプライン・チェーンの実行中に PipelineManager の runProcess メソッドへの引数として使用されるパラメータ・オブジェクト。このパラメータが設定されていない場合、このサーブレット Bean は、extraParametersMap プロパティ内の構成されている値を検索します。この構成されている値を使用して、パイプライン・チェーンの実行中に PipelineManager への引数として使用するハッシュマップが構築されます。extraParametersMap は、新しいハッシュマップ値にバインドされている要求パラメータへの新しいハッシュマップ・キーのマッピングを表します。デフォルトで、RepriceOrderDroplet.extraParametersMap は空です。

pipelineManager
パイプライン・チェーンの実行に使用する PipelineManager のインスタンス。このパラメータが設定されていない場合、このサーブレット Bean は、defaultPipelineManager プロパティ内の構成されている PipelineManager を検索します。デフォルトで、RepriceOrderDroplet.defaultPipelineManager は /atg/commerce/PipelineManager に設定されます。

出力パラメータ

exception
再価格設定処理中に発生した任意の例外。

pipelineResult
パイプライン・チェーンが正常に実行された後、PipelineManager によって返された PipelineResult のインスタンス。

オープン・パラメータ

success
パイプライン・チェーンが正常に実行された後、レンダリングされるパラメータ。

successWithErrors
このパラメータは、パイプライン・チェーンの正常な実行後、PipelineResult オブジェクトにエラー・メッセージが含まれているとレンダリングされます。

failure
パイプライン・チェーンを実行しようとして失敗した後、レンダリングされるパラメータ。

例

```
<dsp:droplet name="RepriceOrderDroplet">
  <dsp:param value="ORDER_SUBTOTAL" name="pricingOp"/>
</dsp:droplet>
```

SetLastUpdatedDroplet

クラス名	atg.commerce.order.abandoned.SetLastUpdatedDroplet
コンポーネント	/atg/commerce/order/abandoned/SetLastUpdatedDroplet （Abandoned Order Services モジュールのみ）

SetLastUpdatedDroplet サーブレット Bean は、指定されたオーダーが abandonmentInfo 項目を持っているかどうかを確認し、持っていない場合に、それを作成し、オーダーに関連付けます。このサーブレット Bean は、さらにオーダーの abandonmentInfo 項目の orderLastUpdated プロパティを現在の日付、時刻で更新します。

Abandoned Order Services モジュールの詳細は、『ATG Web Commerce Programming Guide』の*放棄されたオーダー・サービスの使用*を参照してください。

入力パラメータ

orderId（必須）
現在のオーダーの ID。

出力パラメータ

なし。

オープン・パラメータ

output
このパラメータは、オーダーに abandonmentInfo 項目が含まれている場合、または、このサーブレット Bean によってオーダーの abandonmentInfo 項目が作成された場合にレンダリングされます。

error
このパラメータは、エラーが発生するとレンダリングされます。

例

```
<dsp:droplet name="SetLastUpdatedDroplet">
  <dsp:param name="orderId"
   bean="/atg/commerce/ShoppingCart.current.id"/>
</dsp:droplet>
```

ShipItemRelPrice

クラス名	atg.commerce.pricing.ShipItemRelPriceDroplet
コンポーネント	/atg/commerce/pricing/ShipItemRelPrice

ShipItemRelPrice サーブレット Bean は、ShippingGroupCommerceItemRelationship の価格を返します。このサーブレット Bean は、ShippingGroupCommerceItemRelationship の範囲を調べて、その範囲に適用される DetailedItemPriceInfo オブジェクトの金額の合計を返します。

ShippingGroupCommerceItemRelationship オブジェクトおよび Range オブジェクトの詳細は、『ATG Web Commerce Programming Guide』の*購買プロセス・オブジェクトの操作*を参照してください。
DetailedItemPriceInfo 価格保持クラスの詳細は、『ATG Web Commerce Programming Guide』の「*価格設定サービス*」の*使用と拡張*に関する項を参照してください。

入力パラメータ

shipItemRel（必須）
価格が求められている ShippingGroupCommerceItemRelationship。

propertyName
ShippingGroupCommerceItemRelationship の価格の計算に使用される個別の DetailedItemPriceInfo オブジェクトのプロパティ。デフォルトのプロパティは amount です。

出力パラメータ

price
ShippingGroupCommerceItemRelationship の価格。

error
ShippingGroupCommerceItemRelationship の価格の処理中に発生した可能性がある任意の例外。

オープン・パラメータ

output
価格が正常に処理されるとレンダリングされるオープン・パラメータ。

error
エラーが発生するとレンダリングされるオープン・パラメータ。

例

```
<dsp:droplet name="/atg/commerce/pricing/ShipItemRelPrice">
<dsp:param param="shipItemRel" name="shipItemRel"/>
<dsp:oparam name="output">Price = <dsp:valueof param="price"
       converter="currency">no price</dsp:valueof>
</dsp:oparam>
</dsp:droplet>
```

ShippableGroupsDroplet

クラス名	atg.commerce.fulfillment.ShippableGroupsDroplet
コンポーネント	/atg/commerce/fulfillment/droplet/ShippableGroupsDroplet

ShippableGroupsDroplet サーブレット Bean は、「出荷可能な」出荷グループが含まれているすべてのオーダー、つまり PENDING_SHIPMENT 状態にある出荷グループが含まれているすべてのオーダーを表示します。

入力パラメータ

なし。

出力パラメータ

orders
1 つ以上の出荷可能な出荷グループが含まれているオーダーのオーダーID の配列。

shippingGroups
出荷可能な出荷グループの出荷グループ ID の配列。

count
出荷可能な出荷グループの数。

error
出荷グループの取得中に発生した可能性がある任意の例外。

オープン・パラメータ

shipSchedule
現在の時刻、最後の実行時刻およびスケジュールを含む HardgoodShipper のスケジュールされたサービスに関連する情報をレンダリングします。

output
出荷可能な出荷グループが含まれたオーダーがあるとレンダリングされるオープン・パラメータ。

empty
出荷可能な出荷グループが含まれたオーダーがないとレンダリングされるオープン・パラメータ。

例

```
<dsp:droplet name="/atg/commerce/fulfillment/droplet/ShippableGroupsDroplet">
  <dsp:oparam name="shipSchedule">
    <table border="1">
      <tr>
        <td colspan=2>Shipper</td>
      </tr>
      <tr>
        <td>Current Time</td><td><dsp:valueof
            param="currentTime">NA</dsp:valueof></td>
      </tr>
      <tr>
        <td>Last Run</td><td><dsp:valueof param="lastRun">NA</dsp:valueof></td>
      </tr>
      <tr>
        <td>Schedule</td><td><dsp:valueof param="schedule">NA</dsp:valueof></td>
      </tr>
    </table>
  </dsp:oparam>

  <dsp:oparam name="output">
    <table border="1">
      <tr>
        <td><b>Order ID</b></td><td><b>Shipping Group Id</b></td>
      </tr>
    <dsp:droplet name="/atg/dynamo/droplet/For">
      <dsp:param param="count" name="howMany"/>
      <dsp:oparam name="output">
         <tr>
           <td><dsp:valueof param="orders[param:index]">
                           no orderId</dsp:valueof></td>
           <td><dsp:valueof param="shippingGroups[param:index]">no
                shippingGroupId</dsp:valueof></td>
         </tr>
      </dsp:oparam>
```

```
    </dsp:droplet>
    </table>
    <br>
  </dsp:oparam>

  <dsp:oparam name="empty">
    <table border="1">
      <tr>
        <td><b>Order ID</b></td><td><b>Shipping Group Id</b></td>
      </tr>
      <tr>
        <td colspan=2>There are no shippable groups.</td>
      </tr>
    </table>
  </dsp:oparam>

</dsp:droplet>
```

ShippingDroplet

クラス名	atg.commerce.fulfillment.ShippingDroplet
コンポーネント	/atg/commerce/fulfillment/droplet/ShippingDroplet

ShippingDroplet サーブレット Bean は、特定の出荷グループが出荷されたことをフルフィルメント・システムに通知します。

入力パラメータ

orderId（必須）
出荷された出荷グループが含まれたオーダーの ID。

shippingGroupId（必須）
出荷された出荷グループの ID。

出力パラメータ

status
このパラメータは、次のいずれかの値に設定されます。ShipCallSucceeded または ShipCallFailed。

オープン・パラメータ

output
このオープン・パラメータは、出荷グループが出荷されたことがフルフィルメント・システムに通知されるとレンダリングされます。

例

```
<dsp:droplet name="ShippingDroplet">
  <dsp:param param="orderId" name="orderId"/>
  <dsp:param param="shippingGroupId" name="shippingGroupId"/>
  <dsp:oparam name="output">
    <dsp:valueof param="status">no status</dsp:valueof>
  </dsp:oparam>
</dsp:droplet>
```

ShippingGroupDroplet

クラス名	atg.commerce.order.purchase.ShippingGroupDroplet
コンポーネント	/atg/commerce/order/purchase/ShippingGroupDroplet

ShippingGroupDroplet は、ShippingGroupFormHandler フォーム・ハンドラが使用できるように、ShippingGroup オブジェクトと CommerceItemShippingInfo オブジェクトを初期化するサーブレット Bean です。ShippingGroupDroplet クラスは次のコンテナから構成されています。

- ShippingGroupMapContainer - ユーザーの認可された出荷グループを表す ShippingGroup オブジェクト用のコンテナ。
- CommerceItemShippingInfoContainer - ユーザーのオーダーに含まれている商品に対応する CommerceItemShippingInfo オブジェクト用のコンテナ。

これらのコンテナ、ShippingGroupDroplet サーブレット Bean および ShippingGroupFormHandler フォーム・ハンドラの詳細は、『ATG Web Commerce Programming Guide』の「*購買プロセス・サービスの構成*」の *複雑なオーダーを精算する準備*に関する項を参照してください。

入力パラメータ

clear
このパラメータが True に設定されていると、ShippingGroupDroplet はユーザーの CommerceItemShippingInfoContainer および ShippingGroupMapContainer の両方を消去します。

clearShippingGroups
このパラメータが True に設定されていると、ShippingGroupDroplet はユーザーの ShippingGroupMapContainer を消去します。

clearShippingInfos
このパラメータが True に設定されていると、ShippingGroupDroplet はユーザーの CommerceItemShippingInfoContainer を消去します。個々の Order 内の一意の CommerceItem オブジェクトを参照する新規の CommerceItemShippingInfo オブジェクトを作成するために、1 つの Order につき少なくとも 1 回この操作を実行する必要があります。

createOneInfoPerUnit
True に設定されていると、ShippingGroupDroplet は、個々の CommerceItem に含まれている個別の単位に対応する CommerceItemShippingInfo オブジェクトを作成します。たとえば、数量が 5 個の

CommerceItem に対しては、5 個の CommerceItemShippingInfo オブジェクトが作成されます。ユーザーが自分のプロファイルにデフォルトの ShippingGroup を持っていれば、各 CommerceItemShippingInfo オブジェクトはその ShippingGroup で初期化されます。デフォルトで False に設定されます。

initBasedOnOrder
True に設定されていると、ShippingGroupDroplet は、Order 内の個々の ShippingGroupCommerceItemRelationship オブジェクトに対応する CommerceItemShippingInfo オブジェクトを作成します。CommerceItemShippingInfo は、ShippingCommerceItemRelationship 内に存在する ShippingGroup で初期化されます。このオプションは、顧客が精算プロセスを実行中で、オーダーにすでに複数の ShippingGroupCommerceItemRelationship オブジェクトが含まれている状況に対応するために用意されています。デフォルトで False に設定されます。

initShippingGroups
このパラメータが True に設定されていると、shippingGroupTypes 入力パラメータで指定されている ShippingGroup タイプが初期化されます。

initShippingInfos
True に設定されていると、ShippingGroupDroplet は、Order 内の個々の CommerceItem に対応する CommerceItemShippingInfo オブジェクトを作成します。ユーザーが自分のプロファイルにデフォルトの ShippingGroup を持っていれば、CommerceItemShippingInfo オブジェクトはその ShippingGroup で初期化されます。デフォルトで True に設定されます。

order
ユーザーのオーダーに対応するコンポーネントのデフォルト設定を上書きするためにこのパラメータを使用できます。

shippingGroupTypes
実行される PaymentGroupInitializer コンポーネントを決定するために使用される hardgoodShippingGroup や electronicShippingGroup などの ShippingGroup タイプのカンマ区切りのリスト。

ShippingGroupInitializer コンポーネントは、適切な ShippingGroup オブジェクトを作成および初期化し、それらを ShippingGroupMap コンテナに追加する役割を担います。使用できる個々の ShippingGroup タイプは、ShippingGroupInitializers ServiceMap 内の ShippingGroupInitializer コンポーネントを参照するように構成されます。マップへのキーは、この shippingGroupTypes 入力パラメータで指定される Strings です。値は、その ShippingGroup タイプ向けに初期化を実行する、基礎となる ShippingGroupInitializer コンポーネントです。

デフォルトで、ShippingGroupDroplet サーブレット Bean は次の ShippingGroupInitializer コンポーネントで構成されます。

```
## ServiceMap of shippingGroupTypes to ShippingGroupInitializer Nucleus components

shippingGroupInitializers=\
hardgoodShippingGroup=/atg/commerce/order/purchase/
                      HardgoodShippingGroupInitializer,\
electronicShippingGroup=/atg/commerce/order/purchase/
                        ElectronicShippingGroupInitializer
InstorePickupShippingGroup=\
                        InStorePickupShippingGroupInitializer
```

出力パラメータ

shippingGroups
ShippingGroupMapContainer によって参照されるマップ。

order
ユーザーのオーダーを表す Order オブジェクト。

オープン・パラメータ

output
常にレンダリングされるオープン・パラメータ。

例

この例では、現在のユーザーにとっての可用性に基づいて、HardgoodShippingGroup オブジェクトを作成します。さらに、Order 内の任意の CommerceItems とユーザーの任意の HardgoodShippingGroup オブジェクトとの間の関連付けを容易にする CommerceItemShippingInfo オブジェクトを作成します。

```
<dsp:droplet name="ShippingGroupDroplet">
  <dsp:param value="true" name="clear"/>
  <dsp:param value="hardgoodShippingGroup" name="shippingGroupTypes"/>
  <dsp:param value="true" name="initShippingGroups"/>
  <dsp:param value="true" name="initShippingInfos"/>
  <dsp:oparam name="output"> Manipulation of objects here…
  </dsp:output>
</dsp:droplet>
```

SiteIdForCatalogItem

クラス名	atg.droplet.multisite.SiteIdForItemDroplet
コンポーネント	/atg/commerce/multisite/SiteIdForCatalogItem

注意: このドロップレットは Oracle Commerce Platform の複数サイト機能で使用することを目的にしています。『ATG Web Commerce Multisite Administration Guide』を参照してください。

SiteIdForItemDroplet に関する説明は、『ATG Web Commerce Page Developer's Guide』に記載されています。情報については、このドキュメントを参照してください。SiteIdForCatalogItem の実装は、デフォルトで、shareableTypeId を atg.ShoppingCart に設定します。

UnitPriceDetailDroplet

クラス名	atg.commerce.pricing.UnitPriceDetailDroplet
コンポーネント	/atg/commerce/pricing/UnitPriceDetailDroplet

UnitPriceDetailDroplet は、特定の明細項目内またはグループの単価に関する詳細な情報を割引情報とともに提供できます。この機能は、“3 @ $2.00”や“2 @ $10.00, 1 @ $0.00（2 個購入して 1 個を無料で入手）”などの情報を表示するときに便利です。

入力パラメータ

item
詳細情報が表示される明細項目。このパラメータを使用すると、適切なサーブレット Bean が要求されます。requestedBeans パラメータを使用した場合でも、item パラメータではグループ情報は何も表示されないことに注意してください。

order
価格サーブレット Bean を要求するオーダー。グループ割引のプロモーションの場合、商品を複数のグループに分割できます。そのため、サーブレット Bean は商品に対してではなくオーダーに対して生成されます。order を渡す場合、requestedBeans を渡す必要があります。

requestedBeans
要求するサーブレット Bean を識別します。このパラメータは、order パラメータとともに使用されます。item パラメータを渡す場合、要求するサーブレット Bean を指定する必要はありません。次の変数があります。

- allByGroup: サーブレット Bean を、グループの一部ではないものも含め要求します。
- groupOnly: 割引グループに含まれているサーブレット Bean を要求します。

注意: 3 つの入力パラメータすべてを渡した場合、item パラメータは無視され、order パラメータおよび requestedBeans パラメータに基づいて価格情報が返されます。

出力パラメータ

UnitPriceBeans
このドロップレット出力は、商品およびグループの順に並んだ UnitPriceBean オブジェクトのリストです。

オープン・パラメータ

output
このオープン・パラメータは常にレンダリングされます。

例

次の例は、UnitPriceDetailDroplet サーブレット Bean を使用してオーダー内の明細項目に関する詳細情報を表示する JSP の部分を示しています。

```
<dsp:droplet name="UnitPriceDetailDroplet">
  <dsp:param name="item" param="currentItem"/>
  <dsp:oparam name="output">
```

```
  <dsp:getvalueof var="unitPriceBeans" vartype="java.lang.Object"
      param="unitPriceBeans"/>
    <c:forEach var="unitPriceBean" items="${unitPriceBeans}">

      <dsp:param name="unitPriceBean" value="${unitPriceBean}"/>
      <dsp:getvalueof var="quantity" vartype="java.lang.Double"
          param="unitPriceBean.quantity"/>
        <p class="price">
          <fmt:formatNumber value="${quantity}" type="number"/>
          <fmt:message key="common.atRateOf"/>
          <dsp:getvalueof var="unitPrice" vartype="java.lang.Double"
              param="unitPriceBean.unitPrice"/>
          <fmt:formatNumber value="${unitPrice}" type="currency"
              currencyCode="${currencyCode}"/>
        </p>

      <dsp:getvalueof var="pricingModels" vartype="java.lang.Object"
          param="unitPriceBean.pricingModels"/>
        <c:choose>
          <c:when test="${not empty pricingModels}">
            <c:forEach var="pricingModel" items="${pricingModels}">
              <dsp:param name="pricingModel" value="${pricingModel}"/>
                <p class="note">
                 (<dsp:valueof param="pricingModel.description">
                    <fmt:message key="common.promotionDescriptionDefault"/>
                  </dsp:valueof>)
                </p>
            </c:forEach><%-- End for each promotion used to create the unit
                 price --%>
          </c:when>

          <c:otherwise>
            <dsp:getvalueof var="currentItemOnSale"
                param="currentItem.priceInfo.onSale"/>
              <c:if test='${currentItemOnSale == "true"}'>
                <p><fmt:message key="cart_detailedItemPrice.salePriceB"/></p>
              </c:if>
          </c:otherwise>
        </c:choose>
      </dsp:oparam>
    </dsp:droplet>
  <%-- Render each price bean in the shopping cart. We supply an order a
          request for all beans sorted by group (allByGroup). Currently the only
          other supported option is 'groupedOnly' which will skip any bean that
          isn't part of a group --%>
      <dsp:droplet name="UnitPriceDetailDroplet">
        <dsp:param name="requestedBeans" value="allByGroup" />
```

```
        <dsp:param name="order" bean="ShoppingCart.current" />
    </dsp:droplet>
```

ViewItemEventSender

クラス名	atg.userprofiling.ViewItemEventSender
コンポーネント	/atg/commerce/catalog/CategoryBrowsed /atg/commerce/catalog/ProductBrowsed

CategoryBrowsed と ProductBrowsed は、atg.userprofiling.ViewItemEventSender クラスからインスタンス化される 2 つのサーブレット Bean です。これらのサーブレット Bean は、顧客がカタログ内の品目を表示したときにメッセージング・システムに JMS メッセージを送信します。

入力パラメータ

eventobject
イベント・メッセージで送信される顧客が表示したリポジトリ項目。このパラメータの値は、使用するサーブレット Bean（CategoryBrowsed または ProductBrowsed）に応じて、製品リポジトリ項目またはカテゴリ・リポジトリ項目のどちらかになります。

出力パラメータ

なし。

オープン・パラメータ

error
メッセージング・システムにメッセージを送信中にエラーが発生するとレンダリングされるオープン・パラメータ。

例

次の例は、ProductBrowsed サーブレット Bean を使用して、製品が表示されたときにメッセージを送信する JSP の部分を示しています。製品のリポジトリ ID は、このページにリンクしているページから（ItemId パラメータを介して）このページに渡されます。

```
<dsp:droplet name="/atg/commerce/catalog/ProductLookup">
<dsp:param param="ItemId" name="id"/>

<dsp:oparam name="output">
   <dsp:droplet name="/atg/commerce/catalog/ProductBrowsed">
     <dsp:param param="element" name="eventobject"/>
   </dsp:droplet>
</dsp:oparam>
</dsp:droplet>
```

索引

V

い

お

か

き

く

け

こ

さ

し

す

せ

た

て

と

は

ひ

ふ

ほ

ま

る

わ