BEA Tuxedo

BEA Tuxedo CORBA
ノーティフィケーション・サービス

BEA Tuxedo 8.0
8.0 版
2001 年 10 月 31 日

BEA Tuxedo CORBA ノーティフィケーション・サービス

Document Edition	Date	Software Version
8.0	2001 年 10 月 31 日	BEA Tuxedo 8.0

目次

7. CORBA ノーティフィケーション・サービスの管理

索引

このマニュアルについて

このマニュアルでは、BEA Tuxedo 製品の CORBA ノーティフィケーション・サービスの使い方について説明します。このマニュアルでは、ノーティフィケーション・サービスに関連する概念を定義し、CORBA アプリケーションの開発プロセスについて説明します。また、Notification サンプル・アプリケーションのビルドおよび実行方法、ノーティフィケーション・サービスのアプリケーション・プログラミング・インターフェイス (API)、管理タスク、および管理ツールについても説明します。

このマニュアルでは、以下の内容について説明します。

- 「第 1 章 概要」では、ノーティフィケーション・サービスとそのコンポーネントについて基本的な説明を行います。
- 「第 2 章 CORBA ノーティフィケーション・サービス API のリファレンス」では、ノーティフィケーション・サービス・ソフトウェアでサポートされているアプリケーション・プログラミング・インターフェイスについて説明します。
- 「第 3 章 BEA シンプル・イベント API の使い方」では、BEA シンプル・イベント API と C++ および Java プログラミング言語を使用して、ノーティフィケーション・サービス・アプリケーションを開発する方法について説明します。
- 「第 4 章 CosNotification サービス API の使い方」では、CosNotification API と C++ および Java プログラミング言語を使用して、ノーティフィケーション・サービス・アプリケーションを開発する方法について説明します。
- 「第 5 章 Introductory サンプル・アプリケーションのビルド」では、Introductory サンプル・アプリケーションの概要と、ビルドおよび実行方法について説明します。
- 「第 6 章 Advanced サンプル・アプリケーションのビルド」では、Advanced サンプル・アプリケーションの概要と、ビルドおよび実行方法について説明します。
- 「第 7 章 CORBA ノーティフィケーション・サービスの管理」では、ノーティフィケーション・サービス・ソフトウェアで提供される管理タスクおよび管理ツールについて説明します。

対象読者

このマニュアルは、ノーティフィケーション・サービス・アプリケーションを設計、開発、コンフィギュレーション、および管理するシステム管理者およびプログラマを対象としています。

e-docs Web サイト

BEA Tuxedo 製品のマニュアルは BEA 社の Web サイトで参照することができます。BEA ホーム・ページの [製品のドキュメント] をクリックするか、または http://edocs.beasys.co.jp/e-docs/index.html に直接アクセスしてください。

マニュアルの印刷方法

このマニュアルは、ご使用の Web ブラウザで一度に 1 ファイルずつ印刷できます。Web ブラウザの [ファイル] メニューにある [印刷] オプションを使用してください。

このマニュアルの PDF 版は、Web サイト上にあります。また、マニュアルの CD-ROM にも収められています。BEA Tuxedo この PDF を Adobe Acrobat Reader で開くと、マニュアル全体または一部をブック形式で印刷できます。PDF 形式を利用するには、BEA Tuxedo マニュアルのホーム・ページにある [PDF 版] ボタンをクリックして、印刷するマニュアルを選択します。

Adobe Acrobat Reader をお持ちでない場合は、Adobe 社の Web サイト (http://www.adobe.co.jp/) から無償でダウンロードできます。

関連情報

CORBA、BEA Tuxedo、分散オブジェクト・コンピューティング、トランザクション処理、および C++ プログラミングの詳細については、BEA Tuxedo オンライン・マニュアルの「Bibliography」を参照してください。

サポート情報

皆様の BEA Tuxedo マニュアルに対するフィードバックをお待ちしています。ご意見やご質問がありましたら、電子メールで **docsupport-jp@bea.com** までお送りください。お寄せいただきましたご意見は、BEA Tuxedo マニュアルの作成および改訂を担当する BEA 社のスタッフが直接検討いたします。

電子メール メッセージには、BEA Tuxedo 8.0 リリースのマニュアルを使用していることを明記してください。

BEA Tuxedo に関するご質問、または BEA Tuxedo のインストールや使用に際して問題が発生した場合は、www.bea.com の BEA WebSUPPORT を通して BEA カスタマ・サポートにお問い合わせください。カスタマ・サポートへの問い合わせ方法は、製品パッケージに同梱されているカスタマ・サポート・カードにも記載されています。

カスタマ・サポートへお問い合わせの際には、以下の情報をご用意ください。

- お客様のお名前、電子メール・アドレス、電話番号、Fax 番号
- お客様の会社名と会社の住所
- ご使用のマシンの機種と認証コード
- ご使用の製品名とバージョン
- 問題の説明と関連するエラー・メッセージの内容

表記上の規則

このマニュアルでは、以下の表記規則が使用されています。

規則	項目
太字	用語集に定義されている用語を示します。
Ctrl + Tab	2 つ以上のキーを同時に押す操作を示します。
イタリック体	強調またはマニュアルのタイトルを示します。

等幅テキスト	コード・サンプル、コマンドとオプション、データ構造とメンバ、データ型、ディレクトリ、およびファイル名と拡張子を示します。また、キーボードから入力するテキストも等幅テキストで表示します。
	例 :
	`#include <iostream.h> void main ( ) the pointer psz`
	`chmod u+w *`
	`\tux\data\ap`
	`.doc`
	`tux.doc`
	`BITMAP`
	`float`

太字の等幅テキスト	コード内の重要な語を示します。 例 : `void **commit** ( )`
斜体の等幅テキスト	コード内の変数を示します。 例 : `String *expr*`

規則	項目
大文字のテキスト	デバイス名、環境変数、および論理演算子を示します。 例 : LPT1 SIGNON OR
{ }	構文の行で、選択肢の組み合わせを示します。かっこは入力しません。
[ ]	構文の行で、オプション項目を示します。かっこは入力しません。 例 : `buildobjclient [-v] [-o name ] [-f file-list]...[-l file-list]...`
|	構文の行で、相互に排他的な選択肢の区切りとして使います。記号は入力しません。
...	コマンド・ラインで、以下のいずれかの場合を示します。 ■ コマンド・ラインで、同じ引数を繰り返し使用できることを示します。 ■ 文で、追加のオプション引数が省略されていることを示します。 ■ 追加のパラメータ、値、またはその他の情報を入力できることを示します。 記号は入力しません。 例 : `buildobjclient [-v] [-o name ] [-f file-list]...[-l file-list]...`
. . .	コード例または構文の行で、項目が省略されていることを示します。記号は入力しません。

1　概要

ここでは、次の内容について説明します。

- はじめに
- 機能の概要
- 製品の構成要素

はじめに

ノーティフィケーション・サービスは、BEA Tuxedo CORBA 環境でイベント・サービスを提供します。ノーティフィケーション・サービスはスタンドアロンで機能する製品というよりは、BEA Tuxedo のレイヤード・プロダクトといえます。

ノーティフィケーション・サービスは BEA Tuxedo EventBroker と同様の機能を備えていますが、そのプログラミング・モデルとインターフェイスは CORBA ユーザ向けに設計されています。この手法の副作用は、BEA Tuxedo EventBroker と互換性がないか、EventBroker をはるかに超える機能が提供されるために、CORBA ベースのノーティフィケーション・サービスの大部分がサポートされていないことです。

ノーティフィケーション・サービスは、イベント・ポスト・メッセージを受信してフィルタ処理し、それらのメッセージをサブスクライバに配信する BEA Tuxedo サブシステムです。ポスト元は、対象となるイベントがいつ発生したかを検出し、それをノーティフィケーション・サービスに報告 (ポスト) する BEA Tuxedo CORBA アプリケーションです。サブスクライバは、対象となるイベントがポストされたときに通知アクションの実行を要求する BEA Tuxedo CORBA アプリケーションです。

メッセージを受信して配信する「無名」サービス (ノーティフィケーション・サービス) の概念は、BEA Tuxedo CORBA 環境に新たなクライアント / サーバ通信パラダイムをもたらします。要求側とプロバイダの 1 対 1 の関係ではなく、

任意の数のポスト元が任意の数のサブスクライバに対してメッセージをポストできます。ポスト元は単にイベントをポストするだけで、どこで情報が受信されるのか、何が行われるのかについては関知しません。サブスクライバは、ポスト元を知ることなく、対象となるどのような情報でもノーティフィケーション・サービスから受信できます。また、サブスクライバはさまざまな方法で通知を受けてアクションを実行できます。

通常、ノーティフィケーション・サービス・アプリケーションは例外イベントを処理します。アプリケーション設計者は、アプリケーションでどのイベントを監視するのかを決めなければなりません。たとえば銀行業務アプリケーションでは、非常に高額な払出しについてイベントがポストされるかもしれませんが、すべての払出しでイベントをポストしても特に便利というわけではありません。また、すべてのユーザがそのイベントをサブスクライブする必要はありません。支店長だけに通知すれば十分です。

ノーティフィケーション・サービスのプログラミング・モデルは、CORBA のプログラミング・モデルに基づいています。インターフェイスには 2 つのセットがあります。1 つはこのマニュアルで CosNotification サービス・インターフェイスと呼んでいる CORBA ベースのノーティフィケーション・サービス・インターフェイスのサブセット、もう 1 つは使いやすいように設計された BEA シンプル・イベント・インターフェイス (BEA 独自のインターフェイス) です。どちらのインターフェイスも、CORBA ベースのノーティフィケーション・サービス仕様で定義される標準の構造化されたイベントを渡します。

2 つのインターフェイスはお互いに互換性があります。つまり、CosNotification サービス・インターフェイスを使用してポストされたイベントは、BEA シンプル・イベント・インターフェイスでサブスクライブできます (逆方向も可)。

機能の概要

ノーティフィケーション・サービス・システムには、3 つの基本的な構成要素があります (図 1-1 を参照)。

- イベント・ポスト元 (サプライヤ)

  サプライヤは、イベントを生成します。サプライヤは、イベントを作成してノーティフィケーション・サービスにポストします。

- ノーティフィケーション・サービス (イベント・チャネルとも呼ばれる)

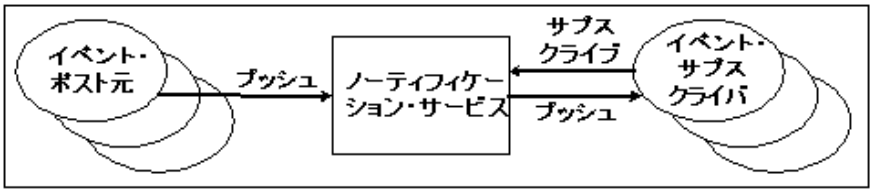
ノーティフィケーション・サービスはイベントを処理します。

- イベント・サブスクライバ (コンシューマ)

  コンシューマは、イベントを受信します。コンシューマは、ノーティフィケーション・サービスに接続してイベントをサブスクライブします。

コンシューマのサブスクリプションに一致するイベントがノーティフィケーション・サービスで受信されると、ノーティフィケーション・サービスではそのイベントをコンシューマに配信しようとします。サプライヤとコンシューマは多数存在することができます。ノーティフィケーション・サービスは複製できますが、論理的にはノーティフィケーション・サービスは 1 つだけです。

図 1-1 ノーティフィケーション・サービス・モデル



CORBA ベースのノーティフィケーション・サービス仕様に従って、イベント・ポスト元では常にプッシュ・モデルを使用します。このため、イベント・ポスト元ではオペレーションを呼び出してノーティフィケーション・サービスにイベントをプッシュします。ノーティフィケーション・サービスでは、イベントのフィルタ処理と配信を行います。イベント・ポスト元とイベント・サブスクライバの間に直接的な関連はありません。イベント・ポスト元およびイベント・サブスクライバの数は、ゼロであったり、1 であったり、または多数であったりします。

また、CORBA ベースのノーティフィケーション・サービス仕様では、サブスクライバはプッシュまたはプルのいずれかのイベント配信モデルを選択できます。このリリースの BEA Tuxedo でサポートされているのはプッシュ・モデルだけです。したがって、ノーティフィケーション・サービスではコンシューマのオペレーションを呼び出してコンシューマにイベントをプッシュします。一致するサブスクリプションのサービスの品質 (QoS) によっては、イベントは永続的に格納され、コンシューマへの配信が保留される場合もあります。

製品の構成要素

BEA Tuxedo CORBA ノーティフィケーション・サービスでは、以下のサポートが提供されます。

- 使いやすさを実現する BEA シンプル・イベント・アプリケーション・プログラミング・インターフェイス (API)
- CosNotification サービス API で定義される最低限のオペレーション・セット
- 一時的および永続的の、サブスクリプションの 2 つのサービスの品質 (QoS)

  一時的なサブスクリプションの場合、ノーティフィケーション・サービスではサブスクライバへのイベント配信が 1 回だけ試行されます。その試行が失敗するとイベントは破棄され、サブスクライバがシャットダウンされているかほかの理由で利用できないとノーティフィケーション・サービスで判断された場合は、サブスクリプションが取り消されます。

  永続的なサブスクリプションの場合、配信の最初の試行が失敗すると、ノーティフィケーション・サービスではそのイベントが保持され、コンフィギュレーション可能な再試行回数の上限に達するまでサブスクリプションの配信が繰り返されます。再試行回数の上限に達すると、イベントはエラー・キューに移動されます。エラー・キューに保持されたイベントは、システム管理者によって処理されます。システム管理者は、エラー・キューからイベントを削除するか (事実上の破棄)、または再び配信できるように保留キューに戻します。
- システム、イベント・キュー、およびサーバ・プロセスの初期コンフィギュレーションのための `UBBCONFIG` ファイルの使用
- BEA Tuxedo 形式の FML フィールド・テーブルの使用。FML フィールド・テーブルを使用することで、ノーティフィケーション・サービスでは次のサポートが可能となります。
  - イベント・ポスト元とイベント・サブスクライバの間のイベント・データのフィルタ処理
  - BEA Tuxedo EventBroker との相互運用性。ノーティフィケーション・サービスでポストされたイベントを Tuxedo EventBroker で消費できます (逆方向も可)。
- 次の BEA Tuxedo ノーティフィケーション・サービス・サーバを使用したイベントの処理

- TMNTS
- TMNTSFWD_P
- TMNTSFWD_T

■ 次の BEA Tuxedo システム・サーバを使用したイベントの処理

- TMSYSEVT
- TMUSREVT
- TMQUEUE
- TMQFORWARD

■ BEA Tuxedo `ntsadmin` 管理ユーティリティを使用したイベント・キューの管理

■ BEA Tuxedo `qmadmin` 管理ユーティリティを使用したイベント・キューのコンフィギュレーションと管理

■ BEA Tuxedo `tmadmin` 管理ユーティリティを使用したトランザクション・ログのコンフィギュレーションと管理

2 CORBA ノーティフィケーション・サービス API のリファレンス

ここでは、次の内容について説明します。

- はじめに
- BEA シンプル・イベント API
- CosNotification サービス API

はじめに

BEA Tuxedo CORBA ノーティフィケーション・サービスでは、2 つのアプリケーション・プログラミング・インターフェイスがサポートされます。1 つは、「CORBAservices: Common Object Services Specification」で定義されている CORBA ベースのノーティフィケーション・サービスに基づいています。このインターフェイスは、このマニュアルでは CosNotification サービス・インターフェイスと呼びます。もう 1 つのインターフェイス (BEA シンプル・イベント・インターフェイス) は、使いやすく設計された BEA 独自のインターフェイスです。

それらのインターフェイスは互いに互換性があり、どちらも CORBA ベースのノーティフィケーション・サービス仕様で定義された構造化イベントを渡します。つまり、CosNotification サービス・インターフェイスを使用してポストされたイベントは BEA シンプル・イベント・インターフェイスでサブスクライブできるということです (逆方向も可)。

ノーティフィケーション・サービス API を使用する前に、次のトピックに目を通してください。

- サービスの品質
- チャネル・ファクトリの取得

- トランザクションの使い方
- 構造化イベントのフィールド、型、およびフィルタ
- イベントの FML フィールド・テーブル・ファイルの作成
- BEA Tuxedo アプリケーションとの相互運用性

サービスの品質

サブスクリプションが永続的かどうか、およびイベント配信の失敗後に配信が再試行されるかどうかを指定するために、サブスクライバではサービスの品質 (QoS) を指定します。サービスの品質には、永続的と一時的の 2 つの設定があります。QoS は、サブスクリプションのプロパティです。

永続的なサブスクリプション

永続的なサブスクリプションでは、イベントの配信とサブスクリプションの永続性が強固に保証されます。ただし、システム・リソース (ディスク領域や CPU サイクルなど) がより多く消費されるとともに、より多くの管理タスク (キューの管理やデッド・サブスクライバの検出など) が必要になります。

永続的なサブスクリプションには以下の特性があります。

- サブスクリプションは、アンサブスクライブが実行されるまで有効です。つまり、サブスクリプションをアクティブな状態のままにして、サブスクライバ・アプリケーションをシャットダウンできます。その場合、イベントはそのサブスクライバ用に保存され、サブスクライバが再起動したときにサブスクリプションを再作成することなくそのサブスクライバに配信されます。
- イベントを配信できない場合、イベント配信は再試行回数の上限に達するまで繰り返されます。イベントの再配信回数の上限を超えると、そのイベントは保留キューからエラー・キューに移動されます。管理者は、イベントをエラー・キューから保留キューに戻すことができます。その保留キューで、イベントの配信が再び試行されます。
- イベントは正常にサブスクライバに配信されるが、何らかの理由で「配信の成功」を通知するメッセージがノーティフィケーション・サービスに届かない場合は、ノーティフィケーション・サービスで同じイベントが複数回配信されることがあります。

一時的なサブスクリプション

一時的なサブスクリプションでは、最小限のオーバーヘッドで最高の性能が実現します。一時的なサブスクリプションには、次の特性があります。

- 一致する各サブスクリプションへのイベントの配信が 1 度試行されます。その試行が失敗した場合、イベントは失われます。

サブスクリプションは、イベント配信の失敗が検出されるまで有効です。配信の失敗が検出された時点で、サブスクリプションは終了します。通常、ノーティフィケーション・サービスでは、性能上の理由のために一時的なサブスクライバに正常にイベントが配信されたかどうかが確認されません。ただし、イベントを一時的なサブスクライバに配信する場合に、ノーティフィケーション・サービスで配信が成功したかどうかが確認されることもあります。イベントが正常に配信されなかった場合で、CORBA::TRANSIENT 例外が返されないときは、ノーティフィケーション・サービスではサブスクリプションが無くなったものと判断し、サブスクリプションを取り消します。配信が失敗して CORBA::TRANSIENT 例外を受信した場合、ノーティフィケーション・サービスではサブスクライバがビジー状態であると判断してイベントを破棄しますが、サブスクリプションは取り消されません。

デッド状態の一時的なサブスクリプションの自動的な取り消しでは、アンサブスクライブされていない一時的なサブスクライバのクリーンアップ・メカニズムが提供されます。ただし、ノーティフィケーション・サービスによる配信の確認はイベントがサブスクライバに最初に送信されたときに行われますが、それから 5 分が経過し、別のイベントが配信されるまでは次の確認は行われません。したがって、確認が実行される間隔は短くて 5 分であり、5 分経過後に配信するイベントがない場合その間隔はさらに長くなります。5 分という最短の間隔は固定であり、変更することはできません。このため、イベント配信の失敗は必ずしも最初の失敗の時点で検出されるわけではありません。イベント配信の失敗は、ノーティフィケーション・サービスで確認が行われたときにのみ検出されます。

チャネル・ファクトリの取得

チャネル・ファクトリは、イベント・チャネルを探すためにイベント・ポスト元のアプリケーションとサブスクライバ・アプリケーションによって使用されます。イベント・チャネルは、イベントのポスト、サブスクリプションのサブスクライブ (作成)、およびサブスクリプションのアンサブスクライブ (取り消し) に使用します。

ノーティフィケーション・サービス・アプリケーションでは、Bootstrap オブジェクトを使用してイベント・チャネル・ファクトリのオブジェクト・リファレンスを取得します。その際には、`Tobj_Bootstrap::resolve_initial_references` オペレーションが使用されます。Bootstrap オブジェクトでは、ノーティフィケーション・サービス・アプリケーションの 2 つのサービス ID (`NotificationService` および `Tobj_SimpleEventsService`) がサポートされています。`NotificationService` オブジェクトは、CosNotification サービス API を使用するアプリケーションで使用します。`Tobj_SimpleEventsService` オブジェクトは、BEA シンプル・イベント API を使用するアプリケーションで使用します。

サービス ID	オブジェクト型
`NotificationService`	`CosNotifyChannelAdmin::EventChannelFactory`
`Tobj_SimpleEventsService`	`Tobj_SimpleEvents::ChannelFactory`

注記 リリース 8.0 の BEA Tuxedo CORBA には、BEA WebLogic Enterprise の旧バージョンで提供されていた BEA クライアント環境オブジェクトが従来のまま含まれており、Tuxedo 8.0 CORBA クライアントで利用することができます。BEA Tuxedo 8.0 クライアントでは、ブートストラップ処理オブジェクト、セキュリティ・オブジェクト、およびトランザクション・オブジェクトの初期リファレンスの解決に引き続きそれらの環境オブジェクトを使用する必要があります。リリース 8.0 の BEA Tuxedo CORBA では、OMG インターオペラブル・ネーミング・サービス (INS) を使用してもブートストラップ処理オブジェクト、セキュリティ・オブジェクト、およびトランザクション・オブジェクトの初期リファレンスを解決できます。INS の詳細については、『BEA Tuxedo CORBA プログラミング・リファレンス』を参照してください。

トランザクションの使い方

トランザクションに関する振る舞いは、BEA シンプル・イベント API と CosNotification サービス API で同じです。トランザクションの振る舞いをサポートする唯一のオペレーションは `push_structured_event` です。このオペレーションは、`CosNotifyChannelAdmin::StructuredProxyPushConsumer` インターフェイスと `Tobj_SimpleEvents::Channel` インターフェイスでサポートさ

れています。ほかのすべてのオペレーションは、トランザクションのコンテキストで使用できますが、トランザクションで実行されても、それ以外で実行されても同じように機能します。

イベントをポストするときの振る舞いは、サブスクリプションの QoS に依存します。サブスクリプションのイベント配信 QoS が永続的である場合に、イベントがトランザクションのコンテキストでポストされると、その配信はトランザクションの結果に左右されます。つまり、トランザクションがコミットされる場合、ノーティフィケーション・サービスでは通常どおりにサブスクライバにイベントが配信されます。トランザクションがロールバックされる場合は、イベントは配信されません。

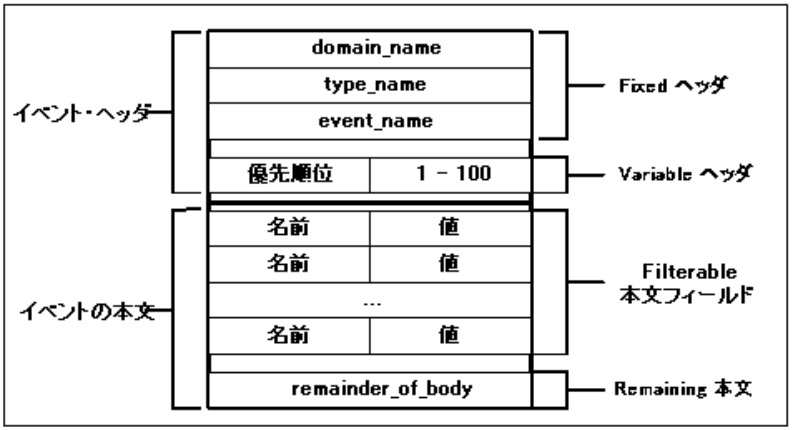
サブスクライバのサブスクリプションのイベント配信 QoS が一時的である場合に、イベントがトランザクションのコンテキストでポストされると、トランザクションの結果に関係なくそのイベントの配信が 1 度試行されます。つまり、トランザクションはイベントが配信されるかどうかに影響を与えず、イベントの配信が 1 度試行されます。

注記 イベントの配信に関連付けられるトランザクション・コンテキストはありません。ただし、永続的なサブスクリプションの場合は、ポスト元のトランザクションがコミットされたときには、イベントがサブスクライバに配信されるか、管理上のアクションを待つためにエラー・キューに入れられることがノーティフィケーション・サービスによって保証されます。

構造化イベントのフィールド、型、およびフィルタ

ポスト元からノーティフィケーション・サービスにプッシュされるか、サブスクライバに配信されるイベントはすべて COS 構造化イベントです。つまり、すべてのイベントは CORBA ベースのノーティフィケーション・サービスで指定されている構造化イベントの定義に適合しています。CORBA ベースのノーティフィケーション・サービスは、CORBAservices イベント・サービスを拡張したサービスです (図 2-1 を参照)。イベントが (ドメインや型ではなく) 内容に基づいてフィルタ処理される場合、またはイベントが BEA Tuxedo アプリケーションによってサブスクライブされる場合は、適用される制限が増えます。それらの制限は、データ型、および内容に基づくフィルタ処理に適用されます。次に、それらの制限について説明します。

図 2-1　構造化イベント



- Fixed ヘッダ・セクションは、構造化イベントを作成するときに使用できる 3 つのフィールドで構成されます。3 つのフィールドは、fixed_header.event_type.domain_name、fixed_header.event_type.type_name、および fixed_header.event_type.event_name です。イベントがポストされるときには、3 つすべてのフィールドがノーティフィケーション・サービスに渡されます。ただし、サブスクリプションが作成されるときには、最初の 2 つのフィールド (domain_name と type_name) のみを使用してイベントがフィルタ処理されます。それらのフィールドは、サブスクリプションで正規表現として定義されます。event_name フィールドは、サブスクリプションでは使用できません。

- Variable ヘッダの構成要素は、単一の名前 / 値 (NV) ペア (優先順位) です。優先順位の値は、1 ～ 100 の範囲で指定します (CORBA ノーティフィケーション・サービス仕様では –32767 ～ 32767 の範囲とされている)。優先順位は、イベント処理の優先順位を付けるためにシステム内部で使用します。最高の優先順位は 100 です。ただし、優先順位の高いイベントが優先順位の低いイベントよりも実際に優先されるとは限りません。Variable ヘッダのサポートは、次の 2 つの点で CORBA ノーティフィケーション・サービス仕様の指定とは異なります。まず、サポートされているフィールドは 1 つ (優先順位) だけですが、仕様では 5 つのフィールドがリストされています。次に、ユーザ定義のフィールドがサポートされていますが、それらのフィールドの内容に応答してアクションが実行されることはありません。ユーザ定義のフィールドは、ただ単に渡されるだけです。

- Filterable 本文は、NV ペアで構成されます。それらのペアの値の型は、any、long、unsigned long、short、unsigned short、octet、char、float、double、string、boolean、void、および null に制限されています。これらのフィールドは、フィルタ式で使用できます。
- Remaining 本文の構成要素は単一の ANY です。値の型は、any、long、unsigned long、short、unsigned short、octet、char、float、double、string、boolean、void、および null に制限されています。このフィールドは、フィルタ式で使用できません。

イベントの設計

イベントの設計は、あらゆるノーティフィケーション・サービスの基本です。イベントの設計は、一致するサブスクリプションに配信される情報の量だけでなく、ノーティフィケーション・サービスの効率と性能にも影響します。したがって、計画を慎重に行って、ノーティフィケーション・サービスが現在のニーズだけでなく将来の規模拡大にも対応できるようにする必要があります。

ノーティフィケーション・サービスでは、(1) ドメイン名、(2) 型名、(3) 優先順位、(4) フィルタ処理可能データ、および (5) 残りの本文の 5 段階のイベント設計がサポートされています。イベントを設計するときには、ドメイン名と型名を指定する必要があります (優先順位とフィルタ処理可能データは任意指定)。ドメイン名は、ビジネスに基づいて選択することができます。たとえば病院は医療ビジネス (health care business) に属しているので、病院のノーティフィケーション・サービス・アプリケーションでは「HEALTHCARE」をドメイン名として選択できます。保険業者の種類に基づいてイベントを分類する必要がある場合は、「HMO」または「UNINSURED」を型名として選択できます。支払い義務のあるエンティティを基準にイベントをさらに細かく定義する必要がある場合は、フィルタ処理可能データを使用して、特定の "HMO_Account" または特定の "Patient_Account" に対する "billing" としてエンティティを識別することも可能です。リスト 2-1 は、このタイプのイベント設計の例を示しています。

リスト 2-1 イベントの設計

```
domain_name = "HEALTHCARE"
type_name = "HMO"
#Filterable data name/value pairs.
filterable_data.name = "billing"
filterable_data.value = 4498
```

```
filterable_data.name = "patient_account"
filterable_data.value = 37621
```

当然、ノーティフィケーション・サービス・アプリケーションでポストおよび受信されるイベントの設計が詳細かつ厳密になればなるほど、ノーティフィケーション・サービスで処理しなければならないイベントの数が少なくなります。このことは、システム・リソースとコンフィギュレーションの要件に直接的に影響します。したがって、イベントを設計するときには十分な考慮が必要です。

イベントの FML フィールド・テーブル・ファイルの作成

イベントのフィールド操作言語 (FML) フィールド・テーブル・ファイルは、次の機能のいずれかが必要な場合のみ作成する必要があります。どの機能も必要でない場合は、FML テーブルは不要です。

- BEA Tuxedo イベントのポスト元とサブスクライバの間の (ドメイン・フィールドと型フィールドに加えた) イベント・データのフィルタ処理
- BEA Tuxedo ノーティフィケーション・サービスと BEA Tuxedo EventBroker の相互運用性

構造化イベントの `filterable_data` フィールドには、名前 / 値 (NV) ペアのリストが格納されます。イベントのデータは通常はこのリストに格納されます。FML フィールド・テーブル・ファイルのフィールド名は、構造化イベント内の名前と一致していなければなりません。フィールドの型には、`carray` を除くすべての FML 型 (`long`、`short`、`double`、`float`、`char`、`string`) を使用できます。構造化イベントの値は、フィールド・テーブルで定義されている型と同じでなければなりません。表 2-1 は、BEA Tuxedo でサポートされている CORBA Any 型を示しています。これらの Any 型を使用して、データのフィルタ処理と BEA Tuxedo の相互運用性を実現できます。

表 2-1 サポートされている CORBA Any 型

CORBA Any 型	データのフィルタ処理と Tuxedo の相互運用性のサポート
`short`	あり

表 2-1 サポートされている CORBA Any 型 (続き)

CORBA Any 型	データのフィルタ処理と Tuxedo の相互運用性のサポート
long	あり
unsigned short	なし
unsigned long	なし
float	あり
double	あり
char	あり
boolean	なし
octet	なし
string	あり
void	なし
null	なし
any	なし

リスト 2-2 は、FML フィールド・テーブル・ファイルの例です。*base 2000 は、フィールドのベース番号です。最初のエントリでは、フィールド名が billing、フィールド番号が 1 (ベース番号に基づく値)、フィールドの型が long となっています。

リスト 2-2 データ・フィルタ処理の FML フィールド・テーブル・ファイル

```
*base 2000

#Field Name        Field #  Field Type    Flags     Comments
#-----------       -------  ----------    ------    --------
billing            1        long           -         -
stock_name         2        string         -         -
price_per_share    3        double         -         -
number_of_shares   5        long           -         -
```

BEA Tuxedo の FML フィールド・テーブル・ファイルには、次のガイドラインと制限が適用されます。

- FML ファイル名の長さは最大で 15 文字です。
- BEA Tuxedo では FML32 が使用されるので、ベース番号とフィールド番号の合計が 101 ～ 33,554,431 の範囲でなければなりません。
- フィールド機能を備えた別のソフトウェアを使用して FML を操作する場合は、フィールド番号に関する別の制約が適用されることがあります。

FML フィールド・テーブル・ファイルの作成とコンフィギュレーションの方法については、『BEA Tuxedo のファイル形式とデータ記述方法』の「field_tables」および『FML を使用した BEA Tuxedo アプリケーションのプログラミング』を参照してください。

BEA Tuxedo アプリケーションとの相互運用性

BEA Tuxedo CORBA ノーティフィケーション・サービスを使用するアプリケーションは、BEA Tuxedo EventBroker を使用する BEA Tuxedo アプリケーションと相互運用できます。BEA Tuxedo ノーティフィケーション・サービスを使用するアプリケーションでは、BEA Tuxedo EventBroker のサブスクライバに配信されるイベントをポストでき、BEA Tuxedo EventBroker によってポストされたイベントを受信できます。

この相互運用性を実現するには、BEA Tuxedo で FML フィールド・テーブルの内容を調整できるように、CosNotification 構造化イベントと BEA Tuxedo FML バッファのマッピングを理解する必要があります。考慮すべき状況は 2 つありま

す。BEA Tuxedo EventBroker を介して BEA Tuxedo アプリケーションによって受信されるイベントのポストと、BEA Tuxedo アプリケーションによってノーティフィケーション・サービスのイベント・チャネルにポストされたイベントの受信です。

イベントのポスト

BEA Tuxedo アプリケーションによってポストされたイベントを BEA Tuxedo アプリケーションでサブスクライブするには、BEA Tuxedo 構造化イベントがポスト時にどのように FML32 およびイベント名にマッピングされるのかを理解する必要があります。マッピングは次のように行われます。

- domain_name と type_name は、イベント名を作成するために domain_name.type_name という形式にまとめられます。これは、tppost オペレーションで使用されるイベント名 (eventname パラメータ) です。
- 構造化イベントの Filterable 本文と Variable ヘッダの各名前 / 値 (NV) ペアは、同じ名前の FML32 フィールドにマッピングされます (そのフィールドが FML でも定義されている場合)。ドメインを "TMEVT" に設定した場合、イベント名は型名と同じになります。

イベントの受信

BEA Tuxedo のシステム・イベントとユーザ・イベントは、BEA Tuxedo アプリケーションで受信できます。システム・イベントは、アプリケーションではなく BEA Tuxedo システムによって生成されます。ユーザ・イベントは、BEA Tuxedo アプリケーションによって生成されます。システム・イベントのリストについては、『BEA Tuxedo のファイル形式とデータ記述方法』の「EVENTS」を参照してください。システム・イベントとユーザ・イベントは、CosNotification 構造化イベントで次のようにマッピングされます。

構造化イベントのフィールド	値
domain_name	常に "TMEVT" に設定します。
type_name	空の文字列
event_name	空の文字列

構造化イベントのフィールド	値
Variable ヘッダ (優先順位)	空のシーケンス
Filterable 本文のフィールド	FML フィールド名と同じです。 **注記** Filterable 本文のフィールドの内容は名前 / 値ペアです。その名前部分は、FML フィールド名と同じです。
残りの本文	常に void に設定します。

BEA Tuxedo システムでは、システムの警告と障害に関連する定義済みの特定のイベントが検出されてポストされます。たとえば、システム生成のイベントでは、コンフィギュレーションの変更、状態の変化、接続の障害、およびマシンの分断が報告されます。

BEA Tuxedo アプリケーションによってポストされたイベントを BEA Tuxedo アプリケーションで受信するには、BEA Tuxedo イベントが格納されている FML バッファをどのように使用して BEA Tuxedo 構造化イベントが作成されるのかを理解する必要があります。また、`domain_name` と `type_name` の BEA Tuxedo イベント名との関連を理解することも必要です。システム・イベントとユーザ・イベントの 2 つのケースを考慮する必要があります。

BEA Tuxedo では、イベント名の先頭にドット (.) を使用してシステム生成のイベントをアプリケーション定義のイベントから区別します。たとえば、システム・イベントには `.SysNetworkDropped` があります。ユーザ・イベントの例としては、`eventsdropped` があります。それらのイベントをサブスクライブするために、ノーティフィケーション・サービスのサブスクライバ・アプリケーションではサブスクリプションを次のように定義する必要があります。

- システム・イベント

  domain_name ="`TMEVT`"
  type_name="`.SysNetworkDropped`"

- ユーザ・イベント

  domain_name ="`TMEVT`"
  type_name="`eventsdropped`"

  イベントが受信されると、ノーティフィケーション・サービスのサブスクライバ・アプリケーションで各イベントが次のように解析されます。

domain_name="TMEVT"
type_name=""
event_name=""
variable_header=empty
Filterable_data=(FML バッファの内容)

サブスクリプションの作成時に使用するパラメータ

サブスクリプションを作成するときには、次のパラメータを指定できます。これらのパラメータでは、BEA のシンプル・イベント API と CosNotification サービス API がサポートされています。

subscription_name

ノーティフィケーション・サービスのサブスクリプションとサブスクライバを識別する名前を指定します。アプリケーションでは、システム管理者にとって意味のある名前を使用しなければなりません。なぜならこれが、管理者がアプリケーションとサブスクリプション、およびそのサブスクリプションを介してサブスクライバに配信されるイベントを関連付ける主要な手段だからです。このパラメータはオプションです (つまり、空の文字列を渡すことができる)。複数のサブスクリプションで同じ名前を使用できます。
subscription_name の長さは最大で 128 文字です。

domain_type

CORBA ベースのノーティフィケーション・サービス仕様で定義されている、構造化イベントの Fixed ヘッダ部分の domain_type フィールドと同じパラメータです。このフィールドは、「Telecommunications」、「Finance」、および「Health Care」など、イベントの型が定義されている特定垂直産業のドメインを識別する文字列です。このパラメータは正規表現なので、フィルタ処理の基準となるドメイン・パターンを設定するために使用することもできます。たとえば、文字 F で始まるすべてのドメインをサブスクライブするには、ドメインを「F.*」に設定します。正規表現を作成する方法については、『BEA Tuxedo CORBA サーバ・アプリケーションの開発方法』で recomp コマンドを参照してください。

type_name

CORBA ベースのノーティフィケーション・サービス仕様で定義されている、構造化イベントの Fixed ヘッダ部分の type_name フィールドと同じパラメータです。これは、Comm_alarm、StockQuote、VitalSigns など、ドメイン内で一意にイベントの型を分類する文字列です。このパラメータは正規表現なので、フィルタ処理の基準となるイベントの型のパターンを設定するために使用することもできます。たとえば、文字 F で始まるすべてのイベントの型をサブスクライブするには、型を「F.*」に設定します。正規表現を作成する方法については、『BEA Tuxedo CORBA サーバ・アプリケーションの開発方法』で recomp コマンドを参照してください。

data_filter

フィルタ処理が行われるフィルタ処理可能データと Variable ヘッダのフィールドの値を指定します。たとえば、ニュース記事のサブスクリプションでは、ドメインが「News」、型が「Sports」、そして data_filter が「Scores > 20」になる場合が考えられます。

このパラメータでは、論理式でサブスクリプションが一致しなければならないデータを定義します。サポートされているデータ型は、short、long、char、float、double、および string です。表 2-2 は、サポートされている論理式の演算子を示しています。

表 2-2 論理式の演算子

式	演算子
単項	+, -, !, ~
倍数	*, /, %
加法	+, -
比較	< , >, <=, >=, ==, !=
等価、一致	==, !=, %%, !%
排他論理和	^
論理積	&&
論理和	||

データのフィルタ処理を利用するには、FML テーブルを設定し、サブスクリプションにフィルタを含めて、データをフィルタ処理し、イベントをポストします。リスト 2-3 は、それらのタスクの例を示しています。

リスト 2-3 データのフィルタ処理の要件

```
//FML テーブルを設定

Field table file.
----------------
*base 2000

*Field Name     Field #    Field Type    Flags     Comments
-----------     -------    ---------     ------    ------
StockName         1         string         -         -
PricePerShare     2         double         -         -
CustomerId        3         long           -         -
CustomerName      4         string         -         -

// サブスクリプション・データのフィルタ処理
1) "NumberOfShares > 100 && NumberOfShares < 1000"
2) "CustomerId == 3241234"
3) "PricePerShare > 125.00"
4) "StockName == 'BEAS'"
5) "CustomerName %% '.*Jones.*'" // CustomerName が「Jones」を格納
6) "StockName == 'BEAS' && PricePerShare > 150.00"

// イベントをポスト
// C++
CosNotification::StructuredEvent ev;
...
ev.filterable_data[0].name    = CORBA::string_dup("StockName");
ev.filterable_data[0].value <<= "BEAS";
ev.filterable_data[1].name    = CORBA::string_dup("PricePerShare");
ev.filterable_data[1].value <<= CORBA::Double(175.00);
ev.filterable_data[2].name    = CORBA::string_dup("CustomerId");
ev.filterable_data[2].value <<= CORBA::Long(1234567);
ev.filterable_data[3].name    = CORBA::string_dup("CustomerName");
ev.filterable_data[3].value <<= "Jane Jones";

// Java
StructuredEvent ev;
...
ev.filterable_data[0].name = "StockName";
ev.filterable_data[0].value.insert_string("BEAS");
ev.filterable_data[1].name = "PricePerShare";
ev.filterable_data[1].value.insert_double(175.00);
ev.filterable_data[2].name = "CustomerId";
ev.filterable_data[2].value.insert_long(1234567);
ev.filterable_data[3].name = "CustomerName";
ev.filterable_data[3].value.insert_string("Jane Jones");
```

フィルタの文法の詳細については、第 2 章の 8 ページ「イベントの FML フィールド・テーブル・ファイルの作成」および『FML を使用した BEA Tuxedo アプリケーションのプログラミング』の「フィールド化バッファの論理式」を参照してください。

`push_consumer`

ノーティフィケーション・サービスで構造化イベントを配信するために使用されるコールバック・オブジェクトを識別します。サブスクライバ・アプリケーションでは、ノーティフィケーション・サービスから呼び出してイベントを配信できるように CosNotifyComm::StructuredPushConsumer インターフェイスをインプリメントする必要があります。

注記　一時的または永続的のいずれかのオブジェクト・リファレンスをコールバック・オブジェクトとして使用できます。どのタイプのオブジェクト・リファレンスを使用するのかを決めるときには、QoS とアプリケーションの実行回数の両方を考慮に入れる必要があります。使用するオブジェクト・リファレンスのタイプを決める際の参考情報については、表 2-3 を参照してください。

表 2-3 共同クライアント / サーバで一時的と永続的のどちらのオブジェクト・リファレンスを使用するのか

サブスクリプションの状況	一時的と永続的のどちらを使用するか
QoS が一時的で、起動とシャットダウンが 1 度行われる	一時的なオブジェクト・リファレンスを使用します。この場合は、システム・リソースを解放するためにサブスクライバ・アプリケーションでシャットダウン時にアンサブスクライブすることをお勧めしますが、必ずしも必要ではありません。
QoS が永続的で、起動とシャットダウンが 1 度行われる	一時的なオブジェクト・リファレンスを使用します。

表 2-3 共同クライアント / サーバで一時的と永続的のどちらのオブジェクト・リファレンスを使用するのか (続き)

サブスクリプションの状況	一時的と永続的のどちらを使用するか
QoS が永続的で、起動とシャットダウンが複数回行われる	永続的なオブジェクト・リファレンスを使用し、サブスクライバのシャットダウンと再起動のたびに同じホストとポートが使用されるようにホストとポートを保存します。この場合は、双方向の IIOP 機能は使用しないことをお勧めします。 **注記** ドメイン内で永続的なオブジェクト・リファレンスはサポートされていないので、共同クライアント / サーバを使用する場合はリモート (BEA Tuxedo ドメインの外側) で使用する必要があります。
QoS が一時的で、起動とシャットダウンが複数回行われる	永続的なオブジェクト・リファレンスを使用できます。ただし、このコンフィギュレーションを使用する場合は、サブスクライバのシャットダウン時にはこのサブスクライバに対するイベントがポストされないようにする必要があります。

qos (サービスの品質)

サブスクリプションのサービスの品質を指定します。一時的または永続的のいずれかを指定できます。

一時的なサブスクリプションの場合、ノーティフィケーション・サービスではサブスクライバへのイベント配信が 1 回だけ試行されます。その試行が失敗した場合は、イベントは破棄されます。ノーティフィケーション・サービスに CORBA::TRANSIENT 例外が届かない場合は、サブスクライバがシャットダウンしているかほかの理由で利用できないと判断され、サブスクリプションが取り消されます。配信が失敗して CORBA::TRANSIENT 例外を受信した場合、ノーティフィケーション・サービスではサブスクライバがビジー状態であると判断してイベントを破棄しますが、サブスクリプションは取り消されません。

永続的なサブスクリプションの場合、配信の最初の試行が失敗すると、ノーティフィケーション・サービスではそのイベントが保留キューに保持され、コンフィギュレーション可能な再試行回数の上限に達するまでサブスクリプションの配信が繰り返されます。再試行回数の上限に達すると、イベントはエラー・キューに移動されます。エラー・キューに保持されたイベントは、システム管理者によって処理されます。システム管理者は、エラー・キューからイベントを削除するか (事実上の破棄)、または再び配信できるように保留キューに戻します。

注記 永続的なサブスクリプションの場合、ノーティフィケーション・サービスでは常にコールバック・オブジェクトを二方向で呼び出してイベントを配信します。共同クライアント / サーバが `orb->run` を呼び出す前にコールバック・オブジェクト (イベントの受信側) をアクティブにしない状況で、ノーティフィケーション・サービスがコールバック・オブジェクトを呼び出した場合、POA に関する限り、コールバック・オブジェクトは存在しません。この場合は、`CORBA::OBJECT_NOT_EXIST` 例外が返されます。

`CORBA::OBJECT_NOT_EXIST` 例外を受信した場合、ノーティフィケーション・サービスではサブスクリプションとイベントを削除します。その例外を受信しない場合は、サブスクリプションは維持され、イベントは再試行されます。

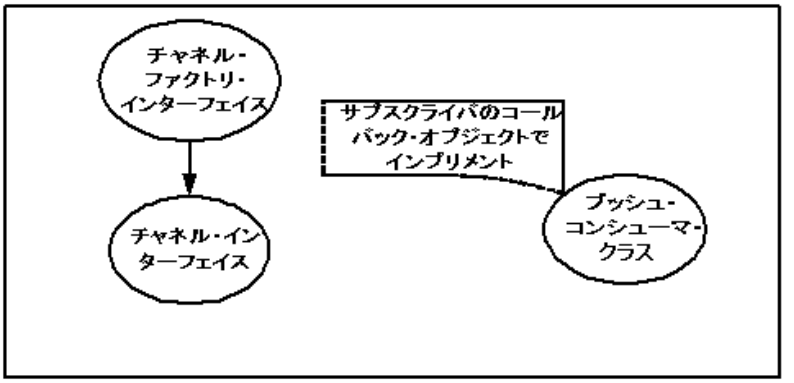
BEA シンプル・イベント API

単純さと使いやすさが、BEA シンプル・イベント・アプリケーション・プログラミング・インターフェイス (API) をの定義特性です。その機能は、BEA Tuxedo EventBroker の機能と似ています。

BEA シンプル・イベント API は、次のインターフェイスで構成されます (図 2-2 を参照)。

- `Tobj_SimpleEvents::Channel`
- `Tobj_SimpleEvents::ChannelFactory`
- `CosNotifyComm::StructuredPushConsumer`

図 2-2 BEA シンプル・イベント・インターフェイス



`Tobj_SimpleEvents::Channel` インターフェイスと `Tobj_SimpleEvents::ChannelFactory` インターフェイスは、ノーティフィケーション・サービスによってインプリメントされます (説明は後述)。

`CosNotifyComm::StructuredPushConsumer` インターフェイスは、サブスクライバによってインプリメントされます。このインターフェイスの説明については、第 2 章の 66 ページ「CosNotifyComm::StructuredPushConsumer::push_structured_event」を参照してください。

注記 この節で言及されている CosNotification サービスのクラスは、`tuxdir\include` ディレクトリにある CosNotification サービス IDL ファイルで完全に記述されています。

注記 サポートされていないクラス・オペレーションを使用すると、`CORBA::NO_IMPLEMENT` 例外が発生します。

TOBJ_SimpleEvents::Channel インターフェイス

Channel インターフェイスは次のように使用されます。

- サブスクライバによって、イベントをサブスクライブおよびアンサブスクライブしたり、サブスクリプションが存在するかどうかを確認したりするために使用されます。

■ ポスト元によって、ノーティフィケーション・サービスにイベントをポストするために使用されます。

このインターフェイスには、次のオペレーションがあります。

- subscribe()
- unsubscribe()
- exists()
- push_structured_event()

このインターフェイスの CORBA IDL は次のとおりです。

```
module Tobj_SimpleEvents
  {
  typedef long   SubscriptionID;
  typedef string RegularExpression;
  typedef string FilterExpression;

  const SubscriptionType TRANSIENT_SUBSCRIPTION = 0;
  const SubscriptionType PERSISTENT_SUBSCRIPTION = 1;

interface Channel
    {
     void push_structured_event(
         in  CosNotification::StructuredEvent   event);

     SubscriptionID subscribe (
       in    string                              subscription_name,
       in    RegularExpression                   domain,
       in    RegularExpression                   type,
       in    FilterExpression                    data_filter,
       in    CosNotification::QoSProperties      qos,
       in    CosNotifyComm::StructuredPushConsumer push_consumer);

     boolean exists( in SubscriptionID id );

     void unsubscribe( in SubscriptionID  id );
    };
  };
```

これらのオペレーションについては、次の節を参照してください。

Channel::subscribe

CORBA IDL

```
SubscriptionID subscribe (
      in    string                              subscription_name,
      in    RegularExpression                    domain,
      in    RegularExpression                    type,
      in    FilterExpression                     data_filter,
      // フィルタ式の長さは 1 で、名前は TRANSIENT_SUBSCRIPTION
      // または PERSISTENT_SUBSCRIPTION でなければならない
      in    CosNotification::QoSProperties      qos,
      in    CosNotifyComm::StructuredPushConsumer push_consumer
);
```

パラメータ このオペレーションでサポートされているパラメータの説明については、第 2 章の 13 ページ「サブスクリプションの作成時に使用するパラメータ」を参照してください。

例外

CORBA::BAD_PARAM

次のいずれかの問題を示します。

Tobj_Events::SUB_INVALID_FILTER_EXPRESSION
Tobj_Events::SUB_UNSUPPORTED_QOS_VALUE

CORBA::IMP_LIMIT

次のいずれかの問題を示します。

Tobj_Events::SUB_DOMAIN_BEGINS_WITH_SYSEV
Tobj_Events::SUB_EMPTY_DOMAIN
Tobj_Events::SUB_EMPTY_TYPE
Tobj_Events::SUB_DOMAIN_AND_TYPE_TOO_LONG
Tobj_Events::SUB_FILTER_TOO_LONG
Tobj_Events::SUB_NAME_TO_LONG
Tobj_Events::TRANSIENT_ONLY_CONFIGURATION

CORBA::INV_OBJREF

次の問題を示します。

Tobj_Events::SUB_NIL_CALLBACK_REF

注記 例外および対応するマイナー・コードの詳細については、第 2 章の 68 ページ「例外のマイナー・コード」を参照してください。

説明 このオペレーションを使用すると、イベントをサブスクライブできます。このオペレーションは、特定のイベントのサブスクリプションを作成するために、ノーティフィケーション・サービスのサブスクライバ・アプリケーションによって呼び出されます。サブスクリプション名、ドメイン名、型名、データ・フィルタ、サービスの品質、およびサブスクライバのコールバック・オブジェクトのオブ

ジェクト・リファレンスが渡されます。コールバック・オブジェクトでは、CosNotifyComm::StructuredPushConsumer IDL インターフェイスがインプリメントされます。

注記 シャットダウンおよび再起動するサブスクライバについては、subscription_id を永続ストレージに書き込む必要があります。

データのフィルタ処理を利用するか、あるいは BEA Tuxedo システム・イベントまたは BEA Tuxedo アプリケーションでポストされたイベントをサブスクライブするには、第 2 章の 8 ページ「イベントの FML フィールド・テーブル・ファイルの作成」および第 2 章の 10 ページ「BEA Tuxedo アプリケーションとの相互運用性」を参照してください。

戻り値 一意のサブスクリプション識別子が返されます。このオペレーションの結果は即時には現れません。このオペレーションからの復帰とイベント配信の実際の開始の間には遅延があります。遅延期間の長さは、コンフィギュレーション次第ではかなり長くなります。この遅延期間に影響する要素の詳細については、第 7 章の 23 ページ「データベースの同期」を参照してください。

注記 1 度だけ起動およびシャットダウンされるノーティフィケーション・サービス・アプリケーションでは、subscription_id を使用してサブスクリプションが自動的にまたはシステム管理者によって取り消されているかどうかを確認できます。

例 **注記** ここで紹介するコード例は全体の一部分です。完全なコード例については、第 3 章の 10 ページ「サブスクリプションの作成」を参照してください。

C++ コード例

```
subscription_id = channel->subscribe(
          subscription_name,
          "News", // ドメイン
          "Sports",   // 型
          "",     // データ・フィルタなし
          qos,
          news_consumer.in()
        );
```

Java コード例

Channel::unsubscribe

CORBA IDL
```
void unsubscribe( in SubscriptionID id );
```

パラメータ

`subscription_id`
サブスクリプション識別子

例外

`CORBA::BAD_PARAM`
次の問題を示します。
`Tobj_Events::INVALID_SUBSCRIPTION_ID`

注記 例外および対応するマイナー・コードの詳細については、第 2 章の 68 ページ「例外のマイナー・コード」を参照してください。

説明 アンサブスクライブに使用します。サブスクライバ・アプリケーションでは、このオペレーションを使用してサブスクリプションを終了します。このオペレーションから復帰すると、それ以降にイベントは配信できません。入力パラメータは、サブスクライブ時に取得した `SubscriptionID` 1 つです。

注記 このオペレーションの結果は即時には現れません。このオペレーションからの復帰後も、サブスクライバでは一定の期間は継続してイベントを受信できます。その期間は、コンフィギュレーション次第ではかなり長くなります。この期間に影響する要素の詳細については、第 7 章の 23 ページ「データベースの同期」を参照してください。

例

C++ コード例

```
channel->unsubscribe(subscription_id);
```

Java コード例

```
channel.unsubscribe(subscription_id);
```

Channel::push_structured_event

CORBA IDL

```
void push_structured_event(
      in   CosNotification::StructuredEvent   notification
);
```

パラメータ `notification`

このパラメータは、CosNotification サービス仕様で定義されている構造化イベントを格納します。

例外 `CORBA_IMP_LIMIT`

サブスクリプションに関する次のいずれかの問題を示します。

```
Tobj_Events::POST_UNSUPPORTED_VALUE_IN_ANY
Tobj_Events::POST_UNSUPPORTED_PRIORITY_VALUE
Tobj_Events::POST_DOMAIN_CONTAINS_SEPARATOR
Tobj_Events::POST_TYPE_CONTAINS_SEPARATOR
Tobj_Events::POST_SYSTEM_EVENTS_UNSUPPORTED
Tobj_Events::POST_EMPTY_DOMAIN
Tobj_Events::POST_EMPTY_TYPE
Tobj_Events::POST_DOMAIN_AND_TYPE_TOO_LONG
```

注記 例外および対応するマイナー・コードの詳細については、第 2 章の 68 ページ「例外のマイナー・コード」を参照してください。

説明 ノーティフィケーション・サービスにイベントをポストするためにポスト元アプリケーションで使用されます。

注記 このオペレーションには、トランザクションのコンテキストで使用される際のトランザクションの振る舞いがあります。詳細については、第 2 章の 4 ページ「トランザクションの使い方」を参照してください。

例 **注記** ここで紹介するコード例は全体の一部分です。完全なコード例については、第 3 章の 4 ページ「イベントの作成とポスト」を参照してください。

C++ コード例

```
channel->push_structured_event(notification);
```

Java コード例

```
channel.push_structured_event(notification);
```

Channel::exists

CORBA IDL
```
boolean exists(in SubscriptionID subscription_id);
```

パラメータ

`subscription_id`
サブスクリプション識別子

例外

`CORBA::BAD_PARAM`
次の問題を示します。
`Tobj_Events::INVALID_SUBSCRIPTION_ID`

`subscription_id` が CosNotification サービス API を使用して作成されたサブスクリプションの識別子ではない場合は、常にこの例外が返されます。

注記 例外および対応するマイナー・コードの詳細については、第 2 章の 68 ページ「例外のマイナー・コード」を参照してください。

説明 サブスクリプションが存在するかどうかを確認するためにサブスクライバ・アプリケーションによって使用されます。システム管理者はサブスクリプションを手動で削除でき、ノーティフィケーション・サービスでは一時的なサブスクリプションを自動的に削除できるので、サブスクライバ・アプリケーションでは必要に応じてサブスクリプションを再作成できるようにこのオペレーションを使用する必要があります。このオペレーションで使用する `subscription_id` は、サブスクライブ時に取得した同じ識別子です。

戻り値 サブスクリプションが存在する場合は true、存在しない場合は false が返されます。

例

C++ コード例

```
if channel->exists (subscription_id) {
    // サブスクリプションは依然として有効
} else {
  // サブスクリプションはもう存在しない

}
```

Java コード例

```
if channel.exists (subscription_id) {
    // サブスクリプションは依然として有効
} else {
  // サブスクリプションはもう存在しない

}
```

TOBJ_SimpleEvents::ChannelFactory インターフェイス

ChannelFactory インターフェイスは、イベント・チャネルの検索に使用します。このインターフェイスには、find_channel という 1 つのオペレーションがあります。

このインターフェイスの CORBA IDL は次のとおりです。

```
module Tobj_SimpleEvents
{
  typedef long   ChannelID;

interface ChannelFactory
{
 Channel find_channel(
   in ChannelID channel_id // DEFAULT_CHANNEL でなければならない
  );
 };
};
```

Channel_Factory::find_channel

CORBA IDL

```
Channel find_channel(
      in  ChannelID  channel_id );
```

パラメータ このリリースの BEA Tuxedo では、イベント・チャネルは 1 つだけです。したがって、`ChannelID` は `Tobj_SimpleEvents::DEFAULT_CHANNEL` (C++) または `Tobj_SimpleEvents.DEFAULT_CHANNEL.Value` (Java) に設定する必要があります。

例外 `CORBA::BAD_PARAM`
次の問題を示します。
`Tobj_Events::INVALID_CHANNEL_ID`

注記 例外および対応するマイナー・コードの詳細については、第 2 章の 68 ページ「例外のマイナー・コード」を参照してください。

説明 ポスト元アプリケーションとサブスクライバ・アプリケーションによって使用されます。このオペレーションを使用すると、イベント・チャネルを見つけて、ポスト元ではイベントをポストし、サブスクライバではイベントをサブスクライブおよびアンサブスクライブできます。

戻り値 デフォルト・イベント・チャネルのオブジェクト・リファレンスが返されます。

例 **注記** ここで紹介するコード例は全体の一部分です。完全なコード例については、第 3 章の 3 ページ「イベント・チャネルの取得」を参照してください。

C++ コード例

```
channel_factory->find_channel(
                Tobj_SimpleEvents::DEFAULT_CHANNEL);
```

Java コード例

```
channel_factory.find_channel(DEFAULT_CHANNEL.value);
```

CosNotification サービス API

この節では、BEA Tuxedo CORBA ノーティフィケーション・サービスによってインプリメントされる CosNotification サービスで定義されるオペレーションについて説明します。それらのオペレーションは、オペレーション・セット全体のサブセットです。このサブセットは、BEA シンプル・イベント API の代わりとして使用できる機能的に完全な API です。

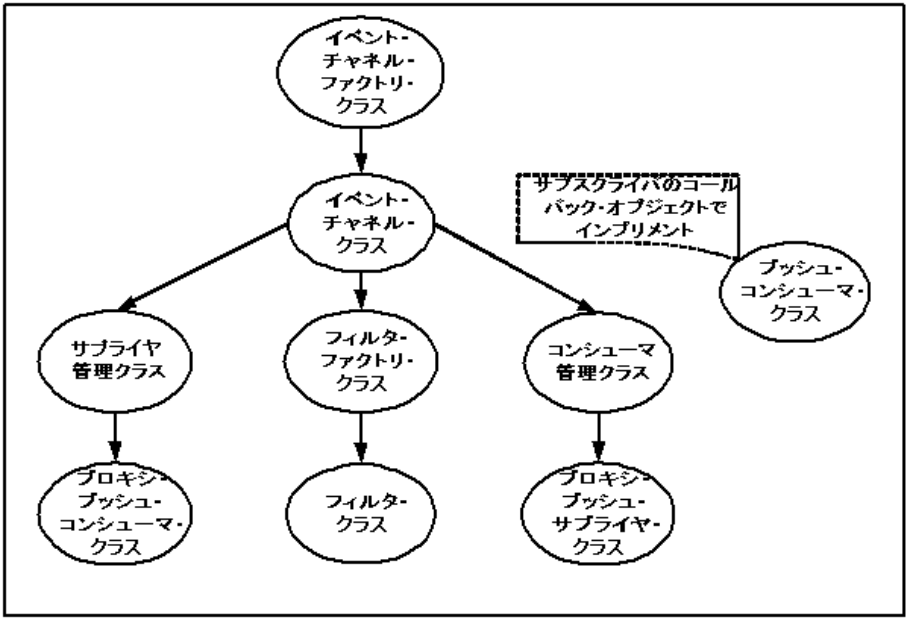
この API は、BEA シンプル・イベント API よりも複雑です。それには、2 つの理由があります。まず、CosNotification サービス API がより複雑になっています。次に、CosNotification サービス API の BEA Tuxedo インプリメンテーションでは、サポートされるオペレーションに制限が追加されます。このように複雑でも、性能または柔軟性の点で利点はないので、できる限りに BEA シンプル・イベント API を使用することをお勧めします。

CosNotification API は、可搬性を実現するため、できる限り標準の API を使用する必要がある場合のために用意されています。この API では、機能の点で、シンプル・イベント API を上回る利点は提供されません。この API を使用して開発されたアプリケーションは、ほぼ可搬性がありますが、完全ではありません。その理由は、完全な可搬性を実現できるほど CosNotification サービス API が十分にサポートされていないことです。たとえば、CORBA ベースのノーティフィケーション・サービスで使用するフィルタ処理の文法は COS Trader 文法に基づいています。BEA Tuxedo ではこの文法がサポートされていず、BEA Tuxedo EventBroker の文法に基づく別の文法がサポートされているので、フィルタ処理を必要とするアプリケーションは可搬になりません。同じことが QoS についても言えます。つまり、CosNotification サービス API では CORBA ベースのノーティフィケーション・サービスの標準のサービス品質がサポートされていず、別のサービス品質がサポートされています。

サポートされている CosNotification サービスのクラスの概要

図 2-3 は、このリリースの BEA Tuxedo で (完全または部分的に) インプリメントされている CosNotification サービスのクラスとそれらの関係を示しています。

図 2-3 インプリメントされている CosNotification サービスのクラス



各クラスでサポートされているオペレーションについては、以下に要約があります。詳細については、第 2 章の 33 ページ「CosNotification サービスのクラスの詳しい説明」を参照してください。

- CosNotifyChannelAdmin::EventChannelFactory クラス

  このクラスは、イベント・ポスト元とサブスクライバ・アプリケーションによって使用されます。このクラスでは、イベントをポスト、サブスクライブ、およびアンサブスクライブするときにチャネル・ファクトリを取得するために使用される `get_channel_factory` オペレーションがサポートされています。

- CosNotifyChannelAdmin::EventChannel クラス

  このクラスは、イベント・ポスト元とサブスクライバ・アプリケーションによって使用されます。このクラスでは、次の 3 つのオペレーションがサポートされています。

  - `default_consumer_admin` ー コンシューマ管理オブジェクトを取得するためにイベント・サブスクライバ・アプリケーションによって使用されます。

- default_supplier_admin ― サプライヤ管理オブジェクトを取得するためにイベント・ポスト元アプリケーションによって使用されます。
- default_filter_factory ― フィルタ・ファクトリ・オブジェクトを取得するためにイベント・サブスクライバ・アプリケーションによって使用されます。

■ CosNotifyChannelAdmin::SupplierAdmin クラス

このクラスは、イベント・ポスト元アプリケーションによって使用されます。このクラスでは、obtain_notification_push_consumer オペレーションがサポートされています。ポスト元アプリケーションでは、このオペレーションを使用して、ノーティフィケーション・サービスにイベントをポストするために使用するプロキシ・プッシュ・コンシューマ・オブジェクトを作成します。

■ CosNotifyChannelAdmin::StructuredProxyPushConsumer クラス

このクラスは、イベント・ポスト元アプリケーションによって使用されます。このクラスでは、次のオペレーションがサポートされています。

- connect_structured_push_supplier ― プロキシ・プッシュ・サプライヤをノーティフィケーション・サービスのイベント・チャネルに接続するためにイベント・ポスト元アプリケーションによって使用されます。
- push_structured_event ― ノーティフィケーション・サービスのイベント・チャネルにイベントをポストするためにイベント・ポスト元アプリケーションによって使用されます。
- disconnect_structured_push_consumer ― ノーティフィケーション・サービスのイベント・チャネルからプロキシ・プッシュ・サプライヤの接続を解除するためにイベント・ポスト元アプリケーションによって使用されます。

■ CosNotifyFilter::FilterFactory クラス

このクラスは、フィルタ・オブジェクトを作成するためにイベント・サブスクライバ・アプリケーションによって使用されます。このクラスでは、create_filter オペレーションがサポートされています。フィルタ・オブジェクトでは、ドメイン、型、およびフィルタ処理可能データを含めたすべてのデータのフィルタ処理が提供されます。

- CosNotifyFilter::Filter クラス

  このクラスは、イベント・サブスクライバ・アプリケーションによって使用されます。このクラスでは、次のオペレーションがサポートされています。

  - `add_contraints` オペレーション ー フィルタのドメイン、型、およびデータ・フィルタを設定するために使用します。
  - `destroy` オペレーション ー フィルタ・オブジェクトを破棄するために使用します。

- CosNotifyChannelAdmin::ConsumerAdmin クラス

  このクラスは、イベント・サブスクライバ・アプリケーションによって使用されます。このクラスでは、次のオペレーションがサポートされています。

  - `obtain_notification_push_supplier` ー サブスクライバのコールバック・オブジェクトにイベントを配信するために使用されるプロキシ・プッシュ・サプライヤ・オブジェクトを作成するためにイベント・サブスクライバ・アプリケーションによって使用されます。
  - `get_proxy_supplier` ー プロキシ・プッシュ・サプライヤ・オブジェクトのオブジェクト・リファレンスを取得するためにイベント・サブスクライバ・アプリケーションによって使用されます。このオペレーションは、サブスクライバ・アプリケーションがシャットダウンして再起動し、サブスクリプションを取り消す場合にのみ使用されます。その理由は、サブスクライバでは最初の実行のオブジェクト・リファレンスを破棄し、次の実行でオブジェクト・リファレンスを再び取得する必要があるからです。サブスクライバでは、実行をまたがってオブジェクト・リファレンスを再利用することはできません。

- CosNotifyChannelAdmin::StructuredProxyPushSupplier クラス

  このクラスは、イベント・サブスクライバ・アプリケーションによって使用されます。このクラスでは、次のオペレーションがサポートされています。

  - `connect_structured_push_consumer` ー サブスクライバをプロキシ・プッシュ・サプライヤに接続するためにイベント・サブスクライバ・アプリケーションによって使用されます。
  - `set_qos` ー サブスクリプションのサービスの品質を設定するためにイベント・サブスクライバ・アプリケーションによって使用されます。
  - `add_filter` ー フィルタ・オブジェクトをサブスクリプションに追加するためにイベント・サブスクライバ・アプリケーションによって使用されます。

- get_filter ー サブスクリプションと関連付けられているフィルタを取得するためにアンサブスクライブ処理の実行時にイベント・サブスクライバ・アプリケーションによって使用されます。このオペレーションは、サブスクライバ・アプリケーションがシャットダウンして再起動する場合にのみ使用されます。
- disconnect_structured_push_supplier ー アンサブスクライブするためにイベント・サブスクライバ・アプリケーションによって使用されます。

■ CosNotifyComm::StructuredPushConsumer

このインターフェイスは、イベント・サブスクライバ・アプリケーションによってインプリメントされます。このインターフェイスでは、push_structured_event オペレーションがサポートされています。ノーティフィケーション・サービスでは、このオペレーションを呼び出してイベントをサブスクライバに配信します。

CosNotification サービスのクラスの詳しい説明

この節では、このリリースの BEA Tuxedo でインプリメントされる CosNotification サービスのクラスについて説明します。それらのクラスは、tuxdir\include ディレクトリにある CosNotification サービスの IDL ファイルで完全に記述されています。

注記　サポートされていないクラス・オペレーションを使用すると、CORBA::NO_IMPLEMENT 例外が発生します。

CosNotifyFilter::Filter クラス

このクラスは、イベント・サブスクライバ・アプリケーションによって使用されます。このクラスの OMG IDL は次のとおりです。

```
Module CosNotifyFilter
{
interface Filter {
     ConstraintInfoSeq add_constraints (
             in ConstraintExpSeq constraint)
          raises (InvalidConstraint);

      void destroy();
    };
}; //CosNotifyFilter
```

CosNotifyFilter::Filter::add_constraints

概要 フィルタ・オブジェクトでドメイン、型、およびデータ・フィルタの各パラメータを設定します。

OMG IDL
```
ConstraintInfoSeq add_constraints (
            in ConstraintExpSeq constraint)
        raises (InvalidConstraint);
```

例外

CosNotifyFilter::InvalidConstraint
発生することはありません。

CORBA::BAD_PARAM
次の問題を示します。
Tobj_Events::SUB_INVALID_FILTER_EXPRESSION.

CORBA_IMP_LIMIT
次のいずれかの問題を示します。
Tobj_Notification::SUB_ADD_CONS_ON_TIMED_OUT_FILTER
Tobj_Notification::SUB_MULTIPLE_CALLS_TO_ADD_CONS
Tobj_Notification::SUB_MULTIPLE_CONSTRAINTS_IN_LIST
Tobj_Notification::SUB_MULTIPLE_TYPES_IN_CONSTRAINT
Tobj_Notification::SUB_SYSTEM_EVENTS_UNSUPPORTED
Tobj_Events::SUB_DOMAIN_BEGINS_WITH_SYSEV
Tobj_Events::SUB_EMPTY_DOMAIN
Tobj_Events::SUB_EMPTY_TYPE
Tobj_Events::SUB_FILTER_TOO_LONG

注記 例外および対応するマイナー・コードの詳細については、第 2 章の 68 ページ「例外のマイナー・コード」を参照してください。

説明 サブスクライブするときに使用します。このオペレーションは、サブスクライブするイベントの種類を定義するためにサブスクライバ・アプリケーションで使用されます。フィルタ・オブジェクトのドメイン、型、およびデータ・フィルタの各パラメータを設定します。これらのパラメータの説明については、第 2 章の 13 ページ「サブスクリプションの作成時に使用するパラメータ」を参照してください。

注記 add_constraints オペレーションの BEA Tuxedo インプリメンテーションは、呼び出しが 1 回限りであり、プロキシ・オブジェクトにフィルタが追加される前に呼び出す必要があり、イベント型が 1 つの 1 つの制約のみで構成される必要があります。

戻り値 空のリストが返されます。呼び出し側では無視することをお勧めします。

例　**注記**　ここで紹介するコード例は全体の一部分です。完全なコード例については、第 4 章の 12 ページ「サブスクリプションの作成」を参照してください。

C++ コード例

```
// フィルタ処理パラメータを設定
// ( ドメイン = "News"、型、およびデータ・フィルタなし )
CosNotifyFilter::ConstraintExpSeq constraints;
constraints.length(1);
constraints[0].event_types.length(1);
constraints[0].event_types[0].domain_name =
    CORBA::string_dup("News");
constraints[0].event_types[0].type_name =
                CORBA::string_dup ("Sports");
// データ・フィルタなし
constraints[0].constraint_expr = CORBA::string_dup("");
CosNotifyFilter::ConstraintInfoSeq_var
add_constraints_results = // この戻り値は無視
     filter->add_constraints(constraints);
```

Java コード例

CosNotifyFilter::Filter::destroy

概要 フィルタ・オブジェクトを破棄します。

OMG IDL
```
void destroy();
```

例外
```
CORBA::BAD_PARAM
```
次の問題を示します。
```
Tobj_Events::SUB_INVALID_FILTER_EXPRESSION.
```

注記 例外および対応するマイナー・コードの詳細については、第 2 章の 68 ページ「例外のマイナー・コード」を参照してください。

説明 アンサブスクライブするときに使用します。このオペレーションは、フィルタ・オブジェクトを破棄するためにサブスクライバ・アプリケーションで使用されます。

注記 対応するサブスクリプションを取り消す準備ができるまでは、フィルタ・オブジェクトを破棄しないでください。

CosNotifyFilter::FilterFactory クラス

このクラスは、イベント・サブスクライバ・アプリケーションによって使用されます。このクラスの OMG IDL は次のとおりです。

```
Module CosNotifyFilter
{
interface FilterFactory {
   Filter create_filter (
          in string constraint_grammar)
      raises (InvalidGrammar);
   destroy();
  };
}; //CosNotifyFilter
```

CosNotifyFilter::FilterFactory::create_filter

概要 どのイベントがサブスクリプションに配信されるのかを指定します。

OMG IDL

```
Filter create_filter (
          in string constraint_grammar)
      raises (InvalidGrammar);
```

例外 `CosNotifyFilter::InvalidGrammar`
`constraint_grammar` がサポートされていないことを示します。

説明 新しいフィルタ・オブジェクトを作成するためにサブスクライバ・アプリケーションで使用されます。このフィルタは、どのイベントがサブスクリプションに配信されるのかを指定するために使用します。サブスクライバでは、フィルタを設定して 5 分以内にプロキシに追加する必要があります。5 分以内に追加しないと、フィルタは破棄されます。フィルタの文法は、`Tobj_Notification::Constraint_grammar` に設定する必要があります。そうしないと、`InvalidGrammar` 例外が発生します。

戻り値 新しいフィルタのオブジェクト・リファレンスが返されます。

例 **注記** ここで紹介するコード例は全体の一部分です。完全なコード例については、第 4 章の 12 ページ「サブスクリプションの作成」を参照してください。

C++ コード例

```
filter_factory->create_filter(
      Tobj_Notification::CONSTRAINT_GRAMMAR
);
```

Java コード例

CosNotifyChannelAdmin::StructuredProxyPushSupplier クラス

このクラスは、イベント・サブスクライバ・アプリケーションによって使用されます。このクラスの OMG IDL は次のとおりです。

```
Module CosNotifyChannelAdmin
{
 interface StructuredProxyPushSupplier :
       ProxySupplier,
       CosNotifyComm::StructuredPushSupplier {

    void connect_structured_push_consumer (
           in CosNotifyComm::StructuredPushConsumer push_consumer)
         raises(CosEventChannelAdmin::AlreadyConnected,
                CosEventChannelAdmin::TypeError );
    };
    // 次のオペレーションが継承される
    void set_qos(in QoSProperties qos)
         raises (UnsupportedQoS);
    FilterID add_filter (in Filter new_filter );
    Filter get_filter( in FilterID filter )
         raises ( FilterNotFound);
    void disconnect_structured_push_supplier();
    readonly attribute ProxyType     MyType;
  };
}; //CosNotifyChannelAdmin
```

CosNotifyChannelAdmin::StructuredProxyPushSupplier:: connect_structured_push_consumer

概要 サブスクリプションを完了します。

OMG IDL

```
void connect_structured_push_consumer (
        in CosNotifyComm::StructuredPushConsumer push_consumer)
    raises(CosEventChannelAdmin::AlreadyConnected,
           CosEventChannelAdmin::TypeError );
```

例外

`CosEventChannelAdmin::TypeError`
発生することはありません。

`CORBA::INV_OREF`
`Tobj_Events::SUB_NIL_CALLBACK_REF`

`CORBA::IMP_LIMIT`
次のいずれかの問題を示します。
`Tobj_Events::SUB_DOMAIN_AND_TYPE_TOO_LONG`
`Tobj_Events::SUB_NAME_TO_LONG`
`Tobj_Events::TRANSIENT_ONLY_CONFIGURATION`
`Tobj_Notification::SUBSCRIPTION_DOESNT_EXIST.`

`CORBA::OBJECT_NOT_EXIST`
プロキシが存在しません。

`CosEventChannelAdmin::AlreadyConnected`
`connect_structured_push_consumer` オペレーションが既に呼び出されていることを示します。

注記 例外の定義および対応するマイナー・コードについては、第 2 章の 68 ページ「例外のマイナー・コード」を参照してください。

説明 このオペレーションは、サブスクライブするときに使用します。このオペレーションは、イベントをサブスクライブするためにサブスクライバ・アプリケーションで使用されます。`push_consumer` パラメータは、サブスクライバのコールバック・オブジェクトを識別します。

`connect_structured_push_consumer` が呼び出されると、ノーティフィケーション・サービスはコールバック・オブジェクトの `push_structured_event` オペレーションを呼び出してサブスクライバにイベントを送信します。`connect_structured_push_consumer` が既に呼び出されている場合は、`AlreadyConnected` 例外が発生します。

注記 `set_qos` と `add_filter` は、`connect_structured_push_consumer` を呼び出す前に呼び出す必要があります。

例 **注記** ここで紹介するコード例は全体の一部分です。完全なコード例については、第 4 章の 12 ページ「サブスクリプションの作成」を参照してください。

C++ コード例

```
subscription->connect_structured_push_consumer(
    news_consumer.in()
);
```

Java コード例

CosNotifyChannelAdmin::StructuredProxyPushSupplier::set_qos

概要 サブスクリプションの QoS を設定します。

OMG IDL
```
void set_qos(in QoSProperties qos)
        raises (UnsupportedQoS);
```

Exceptions
`UnsupportedQoS`
発生することはありません。

`ORBA::IMP_LIMIT`
次のいずれかの問題を示します。
`Tobj_Notification::SUB_MULTIPLE_CALLS_TO_SET_QOS`
`Tobj_Notification::SUB_CANT_SET_QOS_AFTER_CONNECT`
`Tobj_Notification::SUBSCRIPTION_DOESNT_EXIST`
`Tobj_Notification::SUB_UNSUPPORTED_QOS_VALUE`

注記 例外および対応するマイナー・コードの詳細については、第 2 章の 68 ページ「例外のマイナー・コード」を参照してください。

説明 サブスクライブするときに使用します。このオペレーションは、サブスクリプションの QoS を設定するためにサブスクライバ・アプリケーションで使用されます。このオペレーションは、サブスクライバが要求しているサービス品質プロパティ設定をカプセル化する名前と値のペアのシーケンスを入力パラメータとしてとります。

QoS は、サブスクリプションの種類とサブスクリプションの名前という 2 つ要素で構成されます。サブスクリプションの種類は、名前と値のペアを作成して設定します。その名前は `Tobj_Notification::SUBSCRIPTION_TYPE` で、値は `Tobj_Notification::PERSISTENT_SUBSCRIPTION` または `Tobj_Notification::TRANSIENT_SUBSCRIPTION` です。詳細については、第 2 章の 2 ページ「サービスの品質」を参照してください。

サブスクリプションの名前は、名前と値のペアを作成して設定します。その名前は `Tobj_Notification::SUBSCRIPTION_NAME` で、値はユーザ定義の文字列です。

このパラメータの詳細については、第 2 章の 13 ページ「サブスクリプションの作成時に使用するパラメータ」を参照してください。

例 **注記** ここで紹介するコード例は全体の一部分です。完全なコード例については、第 4 章の 12 ページ「サブスクリプションの作成」を参照してください。

C++ コード例

```
CosNotification::QoSProperties qos;
qos.length(2);
qos[0].name =
    CORBA::string_dup(Tobj_Notification::SUBSCRIPTION_NAME);
qos[0].value <<= "MySubsription";
qos[1].name =
    CORBA::string_dup(Tobj_Notification::SUBSCRIPTION_TYPE);
qos[1].value <<=
    Tobj_Notification::TRANSIENT_SUBSCRIPTION;

subscription->set_qos(qos);
```

Java コード例

CosNotifyChannelAdmin::StructuredProxyPushSupplier::add_filter

概要 サブスクライバのコールバック・オブジェクトでフィルタ・オブジェクトを設定します。

OMG IDL
```
add_filter(
      in Filter new_filter
);
```

例外

`CORBA::IMP_LIMIT`
次のいずれかの問題を示します。
`Tobj_Notification::SUB_MULTIPLE_CALLS_TO_SET_FILTER`
`Tobj_Notification::SUB_ADD_FILTER_AFTER_CONNECT`
`Tobj_Notification::SUB_NIL_FILTER_REF`
`Tobj_Notification::SUB_NO_CUSTOM_FILTERS`

`CORBA::OBJECT_NOT_EXIST`
サブスクリプションが存在しないことを示します。

注記 例外および対応するマイナー・コードの詳細については、第 2 章の 68 ページ「例外のマイナー・コード」を参照してください。

説明 サブスクライブするときに使用します。このオペレーションは、サブスクライバのコールバック・オブジェクトのフィルタ・オブジェクトを設定するためにサブスクライバ・アプリケーションで使用されます。このオペレーションを使用するアプリケーションがシャットダウンされて再起動される場合は、`filter_id` を永続ストレージに書き込む必要があります。

注記 このオペレーションは、サブスクライバのコールバック・オブジェクトが接続された後に呼び出すことはできません (上記の `connect_structured_push_consumer` を参照)。また、複数回呼び出すこともできません。さらに、このオペレーションを呼び出すときには、フィルタの制約式が既にフィルタに存在していなくてはなりません (`CosNotifyFilter::Filter add_constraints` を参照)。

注記 イベント・チャネルのデフォルト・フィルタ・ファクトリで作成されたフィルタのみ追加することができます。

戻り値 `filter_id` が返されます。

例 **注記** ここで紹介するコード例は全体の一部分です。完全なコード例については、第 4 章の 12 ページ「サブスクリプションの作成」を参照してください。

C++ コード例

```
CosNotifyFilter::FilterID filter_id =
      subscription->add_filter(filter.in());
```

Java コード例

CosNotifyChannelAdmin::StructuredProxyPushSupplier::get_filter

概要 サブスクライバのコールバック・オブジェクトに現在関連付けられているフィルタのオブジェクト・リファレンスを取得します。

OMG IDL

```
Filter get_filter( in FilterID filter )
      raises ( FilterNotFound);
```

例外 `CosNotifyChannelAdmin::FilterNotFound`
フィルタを見つけることができなかったことを示します。

説明 再起動可能なサブスクライバでアンサブスクライブする必要があるときに使用します。このオペレーションは、サブスクライバのコールバック・オブジェクトと現在関連付けられているフィルタのオブジェクト・リファレンスを取得するためにサブスクライバ・アプリケーションで使用されます。渡される `FilterID` は、サブスクライバの StructuredProxyPushSupplier オブジェクトで有効でなければなりません。`FilterID` がイベント・チャネルと関連付けられているプロキシ・オブジェクトで有効ではない場合は、`FilterNotFound` 例外がスローされます。このオペレーションは、シャットダウンして再起動されるサブスクライバでのみ使用されます。

制限 このオペレーションでは、次の制限とガイドラインが適用されます。

a. このオペレーションから返されるフィルタのオブジェクト・リファレンスは、比較操作では使用できません。

b. このオペレーションから返されるフィルタのオブジェクト・リファレンスは、`CosNotifyFilter::Filter::destroy` オペレーションで使用できますが、修正したりプロキシ・オブジェクトに追加したりできないのでほとんど役に立ちません。

戻り値 サブスクライバのコールバック・オブジェクトに現在関連付けられているフィルタのオブジェクト・リファレンスが返されます。

例 **C++ コード例**

```
CosNotify::Filter_var filter =
     subscription->get_filter( filter_id() );
```

Java コード例

CosNotifyChannelAdmin::StructuredProxyPushSupplier:: disconnect_structured_push_supplier

概要　アンサブスクライブに使用します。

OMG IDL　`void disconnect_structured_push_supplier();`

例外　`CORBA::OBJECT_NOT_EXIST`
接続を解除するサブスクリプションが存在しないことを示します。

注記　例外および対応するマイナー・コードの詳細については、第 2 章の 68 ページ「例外のマイナー・コード」を参照してください。

説明　アンサブスクライブするときにサブスクライバ・アプリケーションによって使用されます。このオペレーションは、ノーティフィケーション・サービスとサブスクライバのコールバック・オブジェクトの接続を終了するためにサブスクライバ・アプリケーションで使用されます。

注記　このオペレーションで、イベントの配信が即座に停止することはありません。このオペレーションからの復帰後も、サブスクライバでは一定の期間は継続してイベントを受信できます。

例　**C++ コード例**

```
subscription->disconnect_structured_push_supplier();
```

Java コード例

CosNotifyChannelAdmin::StructuredProxyPushSupplier::MyType

概要 常に CosNotifyChannelAdmin::PUSH_STRUCTURED プロキシを返します。

OMG IDL `readonly attribute ProxyType MyType`

説明 常に CosNotifyChannelAdmin::PUSH_STRUCTURED プロキシを返します。

CosNotifyChannelAdmin::StructuredProxyPushConsumer クラス

このクラスは、イベント・ポスト元アプリケーションによって使用されます。このクラスの OMG IDL は次のとおりです。

```
Module CosNotifyChannelAdmin
{
 interface StructuredProxyPushConsumer :
       ProxyConsumer,
       CosNotifyComm::StructuredPushConsumer {

    void connect_structured_push_supplier (
          in CosNotifyComm::StructuredPushSupplier push_supplier)
       raises(CosEventChannelAdmin::AlreadyConnected);
    // 次のオペレーションが継承される
    readonly attribute MyType;
    void push_structured_event(
          in CosNotification::StructuredEvent notification )
       raises( CosEventComm::Disconnected );
    void disconnect_structured_push_consumer();
   };
}; \\StructuredProxyPushConsumer
```

CosNotifyChannelAdmin::StructuredProxyPushConsumer:: connect_structured_push_supplier

概要 イベントを受信するようにノーティフィケーション・サービスを準備をします。

OMG IDL
```
void connect_structured_push_supplier (
        in CosNotifyComm::StructuredPushSupplier push_supplier)
    raises(CosEventChannelAdmin::AlreadyConnected);
```

例外 `CosEventChannelAdmin::AlreadyConnected`
発生することはありません。

説明 イベントのポスト時にポスト元アプリケーションによって使用されます。このオペレーションを呼び出さないと、ノーティフィケーション・サービスではイベント受信の準備ができません。また、このオペレーションを使用するときには NIL を渡す必要があります。次の順序で使用します。

1. プロキシを作成します。
2. このオペレーションを使用してノーティフィケーション・サービスに接続し、NIL を渡します。
3. イベントをポストします。
4. ポスト元プログラムを終了する前に、接続を解除します。

例 **注記** ここで紹介するコード例は全体の一部分です。完全なコード例については、第 4 章の 4 ページ「イベントの作成とポスト」を参照してください。

C++ コード例

```
proxy_push_consumer->connect_structured_push_supplier(
      CosNotifyComm::StructuredPushSupplier::_nil()
    );
```

Java コード例

```
proxy_push_consumer.connect_structured_push_supplier(null);
```

CosNotifyChannelAdmin::StructuredProxyPushConsumer:: push_structured_event

概要 イベント・チャネルにイベントをポストします。

OMG IDL

```
void push_structured_event(
        in CosNotification::StructuredEvent notification )
    raises( CosEventComm::Disconnected );
```

例外

`CosEventComm::Disconnected`
発生することはありません。

`CORBA::IMP_LIMIT`
次のいずれかの問題を示します。
`Tobj_Events::POST_UNSUPPORTED_VALUE_IN_ANY`
`Tobj_Events::POST_UNSUPPORTED_PRIORITY_VALUE`
`Tobj_Events::POST_DOMAIN_CONTAINS_SEPARATOR`
`Tobj_Events::POST_TYPE_CONTAINS_SEPARATOR`
`Tobj_Events::POST_SYSTEM_EVENTS_UNSUPPORTED`
`Tobj_Events::POST_EMPTY_DOMAIN`
`Tobj_Events::POST_EMPTY_TYPE`
`Tobj_Events::POST_DOMAIN_AND_TYPE_TOO_LONG`

注記 例外および対応するマイナー・コードの詳細については、第 2 章の 68 ページ「例外のマイナー・コード」を参照してください。

説明 イベントをポストするときに使用します。このオペレーションは、イベント・チャネルにイベントをポストするためにポスト元アプリケーションで使用されます。

注記 このオペレーションは、次の点で標準の CORBA 定義とは異なります。

a. イベントの Variable ヘッダの優先順位は、(指定する場合は) 1 ～ 100 の範囲の `short` 値でなければなりません。

b. (ドメインと型のみのフィルタ処理ではなく) イベントのフィルタ処理可能データのフィルタ処理が必要な場合、またはイベントが BEA Tuxedo サブスクライバによって受信される場合は、制限が追加されます。第 2 章の 5 ページ「構造化イベントのフィールド、型、およびフィルタ」および第 2 章の 10 ページ「BEA Tuxedo アプリケーションとの相互運用性」を参照してください。

注記　このオペレーションには、トランザクションのコンテキストで使用される際のトランザクションの振る舞いがあります。詳細については、第 2 章の 4 ページ「トランザクションの使い方」を参照してください。

例　**注記**　ここで紹介するコード例は全体の一部分です。完全なコード例については、第 4 章の 4 ページ「イベントの作成とポスト」を参照してください。

C++ コード例

```
proxy_push_consumer->push_structured_event(notification);
```

Java コード例

```
proxy_push_consumer.push_structured_event(notification);
```

CosNotifyChannelAdmin::StructuredProxyPushConsumer:: disconnect_structured_push_consumer

概要 イベントのポストを停止します。

OMG IDL `void` **`disconnect_structured_push_consumer`**`();`

説明 イベントをポストするときに使用します。このオペレーションは、イベントのポストを停止するためにポスト元アプリケーションによって使用されます。入力パラメータは不要で、値は返されません。次の順序で使用することをお勧めします。

1. プロキシを作成します。
2. ポスト元アプリケーションの実行ごとに接続し接続を解除します。

例 **注記** ここで紹介するコード例は全体の一部分です。完全なコード例については、第 4 章の 4 ページ「イベントの作成とポスト」を参照してください。

C++ コード例

```
proxy_push_consumer->disconnect_structured_push_consumer();
```

Java コード例

```
proxy_push_consumer.disconnect_structured_push_consumer();
```

CosNotifyChannelAdmin::StructuredProxyPushConsumer::MyType

概要 常に CosNotifyChannelAmdmin::PUSH_STRUCTURED プロキシを返します。

OMG IDL `readonly attribute ProxyType MyType`

説明 常に CosNotifyChannelAmdmin::PUSH_STRUCTURED プロキシを返します。

CosNotifyChannelAdmin::ConsumerAdmin クラス

このクラスは、イベント・サブスクライバ・アプリケーションによって使用されます。このクラスの OMG IDL は次のとおりです。

```
Module CosNotifyChannelAdmin
{
 interface ConsumerAdmin :

       CosNotification::QoSAdmin,
       CosNotifyComm::NotifySubscribe,
       CosNotifyFilter::FilterAdmin,
       CosEventChannelAdmin::ConsumerAdmin {

    ProxySupplier obtain_notification_push_supplier (
          in ClientType ctype,
          out ProxyID proxy_id)
        raises ( AdminLimitExceeded )

    ProxySupplier get_proxy_supplier (
          in ProxyID proxy_id )
       raises ( ProxyNotFound );

  };
}; //CosNotifyChannelAdmin
```

CosNotifyChannelAdmin::ConsumerAdmin:: obtain_notification_push_supplier

概要 プロキシ・プッシュ・サプライヤ・オブジェクトを作成します。

OMG IDL
```
ProxySupplier obtain_notification_push_supplier (
            in ClientType ctype,
            out ProxyID proxy_id)
          raises ( AdminLimitExceeded )
```

例外

`CosNotifyChannelAdmin::AdminLimitExceeded`
発生することはありません。

`CORBA::IMP_LIMIT`
次の問題を示します。
`Tobj_Notification::SUB_UNSUPPORTED_CLIENT_TYPE`

説明 サブスクライブするときに使用します。このオペレーションは、プロキシ・プッシュ・サプライヤ・オブジェクトを作成するためにサブスクライバ・アプリケーションで使用されます。構造化イベントのみがサポートされています。つまり、`ANY_EVENT` と `SEQUENCE_EVENT ClientTypes` はサポートされていません。したがって、`ClientType` 入力パラメータは `CosNotifyComm::STRUCTURED_EVENT` に設定する必要があります。サブスクライバをシャットダウンして再起動する場合で、サブスクリプションがプログラムの複数の実行にまたがって存続する場合は、このオペレーションで返される `ProxyID` を永続的に保存する必要があります。サブスクライバでは、プロキシ・サプライヤを `CosNotifyChannelAdmin::StructuredProxyPushSupplier` にナロー変換する必要があります。必要なすべてのオペレーションは 5 分間で完了する必要があります。

注記 1 度だけ起動およびシャットダウンされるノーティフィケーション・サービス・アプリケーションでは、`proxy_id` を使用してサブスクリプションが自動的にまたはシステム管理者によって取り消されているかどうかを確認できます。

戻り値 このオペレーションでは、新しいプロキシのオブジェクト・リファレンスが返されます。新しい `proxy_id` も、`proxy_id` 出力パラメータを通じて返されます。

例 **注記** ここで紹介するコード例は全体の一部分です。完全なコード例については、第 4 章の 12 ページ「サブスクリプションの作成」を参照してください。

C++ コード例

```
CosNotifyChannelAdmin::ProxySupplier_var generic_proxy =
       consumer_admin->obtain_notification_push_supplier(
          CosNotifyChannelAdmin::STRUCTURED_EVENT,
          proxy_id
        );

CosNotifyChannelAdmin::StructuredProxyPushSupplier_var proxy =
      CosNotifyChannelAdmin::StructuredProxyPushSupplier::_narrow(
            generic_proxy.in ()
          );
```

Java コード例

CosNotifyChannelAdmin::ConsumerAdmin::get_proxy_supplier

概要 コンシューマ管理オブジェクトの obtain_notification_push_supplier オペレーションを使用して作成されたプロキシ・プッシュ・サプライヤ・オブジェクトを返します。

OMG IDL

```
ProxySupplier get_proxy_supplier (
        in ProxyID proxy_id )
    raises ( ProxyNotFound );
```

例外 CosNotifyChannelAdmin::ProxyNotFound
ProxyID を見つけることができなかったことを示します。

説明 アンサブスクライブするときに使用します。このオペレーションは、コンシューマ管理オブジェクトの obtain_notification_push_supplier オペレーションを使用して作成されたプロキシ・プッシュ・サプライヤ・オブジェクトを返すためにサブスクライバ・アプリケーションで使用されます。ProxyID 入力パラメータは、プロキシ・オブジェクトを一意に識別します。呼び出し側では、一時的なサブスクリプションの配信エラーまたは ntsadmin 管理コマンドによってプロキシ・オブジェクトが破棄される可能性があることを認識していなければなりません。プロキシ・オブジェクトが破棄されると、関連付けられている ProxyID が無効になります。ProxyID が無効な場合は、ProxyNotFound 例外が発生します。サブスクライバでは、プロキシ・サプライヤを CosNotifyChannelAdmin::StructuredProxyPushSupplier にナロー変換する必要があります。

戻り値 既存のプロキシのオブジェクト・リファレンスが返されます。

例 **C++ コード例**

```
CosNotifyChannelAdmin::ProxySupplier_var generic_proxy =
        m_consumer_admin->get_proxy_supplier(
          m_subscription_info.news_proxy_id()
        );

CosNotifyChannelAdmin::StructuredProxyPushSupplier_var proxy =
      CosNotifyChannelAdmin::StructuredProxyPushSupplier::_narrow(
          generic_proxy.in()
      );
```

Java コード例

CosNotifyChannelAdmin::SupplierAdmin クラス

このクラスは、イベント・ポスト元アプリケーションによって使用されます。このクラスの OMG IDL は次のとおりです。

```
Module CosNotifyChannelAdmin
{
 interface SupplierAdmin :
        CosNotification::QoSAdmin,
        CosNotifyComm::NotifyPublish,
        CosNotifyFilter::FilterAdmin,
        CosEventChannelAdmin::SupplierAdmin {

     ProxyConsumer obtain_notification_push_consumer (
               in ClientType ctype,
               out ProxyID proxy_id)
          raises ( AdminLimitExceeded );
  };
}; //SupplierAdmin
```

CosNotifyChannelAdmin::SupplierAdmin:: obtain_notification_push_consumer

概要 プロキシ・プッシュ・コンシューマ・オブジェクトを作成します。

OMG IDL

```
ProxyConsumer obtain_notification_push_consumer (
            in ClientType ctype,
            out ProxyID proxy_id)
        raises ( AdminLimitExceeded );
```

例外

`CosNotifyChannelAdmin::AdminLimitExceeded`
発生することはありません。

`CORBA::IMP_LIMIT`
次の問題を示します。
`Tobj_Notification::SUB_UNSUPPORTED_CLIENT_TYPE`

説明 イベントをポストするときに使用します。このオペレーションは、プロキシ・プッシュ・コンシューマ・オブジェクトを作成するためにポスト元アプリケーションで使用されます。`ClientType` は、`"CosNotifyChannelAdmin::STRUCTURED_EVENT"` に設定する必要があります。返される `ProxyID` は無視してください。プロキシ・コンシューマは、`CosNotifyChannelAdmin::StructuredProxyPushConsumer` にナロー変換する必要があります。

注記 1 度だけ起動およびシャットダウンされるノーティフィケーション・サービス・アプリケーションでは、`proxy_id` を使用してサブスクリプションが自動的にまたはシステム管理者によって取り消されているかどうかを確認できます。

戻り値 このオペレーションでは、新しいプロキシのオブジェクト・リファレンスが返されます。新しい `proxy_id` も、`proxy_id` 出力パラメータを通じて返されます。

例 **注記** ここで紹介するコード例は全体の一部分です。完全なコード例については、第 4 章の 4 ページ「イベントの作成とポスト」を参照してください。

C++ コード例

```
CosNotifyChannelAdmin::ProxyConsumer_var generic_proxy_consumer =
      supplier_admin->obtain_notification_push_consumer(
        CosNotifyChannelAdmin::STRUCTURED_EVENT,
        proxy_id
      );

CosNotifyChannelAdmin::StructuredProxyPushConsumer_var
   proxy_push_consumer =
      CosNotifyChannelAdmin::StructuredProxyPushConsumer::_narrow(
```

```
        generic_proxy_consumer
      );
```

Java コード例

```
supplier_admin.obtain_notification_push_consumer(
     ClientType.STRUCTURED_EVENT, proxy_id );
```

CosNotifyChannelAdmin::EventChannel クラス

このクラスは、イベント・ポスト元アプリケーションによって使用されます。このクラスの OMG IDL は次のとおりです。

```
Module CosNotifyChannelAdmin
{
 interface EventChannel :
       CosNotification::QoSAdmin,
       CosNotification::AdminPropertiesAdmin,
       CosEventChannelAdmin::EventChannel {

     readonly attribute ConsumerAdmin default_consumer_admin;
     readonly attribute SupplierAdmin default_supplier_admin;
     readonly attribute CosNotifyFilter::FilterFactory
                                 default_filter_factory;
  };
}; //CosNotifyChannelAdmin
```

CosNotifyChannelAdmin::EventChannel:: ConsumerAdmin default_consumer_admin

概要 ConsumerAdmin オブジェクトを取得します。

OMG IDL

```
readonly attribute ConsumerAdmin default_consumer_admin;
```

説明 サブスクライブおよびアンサブスクライブするときに使用します。このオペレーションは、ConsumerAdmin オブジェクトを取得するためにサブスクライバ・アプリケーションで使用されます。

戻り値 ConsumerAdmin オブジェクトのオブジェクト・リファレンスが返されます。

例

注記 ここで紹介するコード例は全体の一部分です。完全なコード例については、第 4 章の 10 ページ「イベント・チャネル、ConsumerAdmin オブジェクト、およびフィルタ・ファクトリ・オブジェクトの取得」を参照してください。

C++ コード例

```
channel->default_consumer_admin();
```

Java コード例

注記

CosNotifyChannelAdmin::EventChannel:: ConsumerAdmin default_supplier_admin

概要 SupplierAdmin オブジェクトを取得します。

OMG IDL

```
readonly attribute SupplierAdmin default_supplier_admin;
```

説明 イベントをポストするときに使用します。このオペレーションは、SupplierAdmin オブジェクトを取得するためにイベント・ポスト元アプリケーションで使用されます。

戻り値 SupplierAdmin のオブジェクト・リファレンスが返されます。

例 **注記** ここで紹介するコード例は全体の一部分です。完全なコード例については、第 4 章の 4 ページ「イベントの作成とポスト」を参照してください。

C++ コード例

```
channel->default_supplier_admin();
```

Java コード例

```
channel.default_supplier_admin();
```

CosNotifyChannelAdmin::EventChannel::default_filter_factory

概要 デフォルトの FilterFactory オブジェクトを取得します。

OMG IDL

```
readonly attribute CosNotifyFilter::FilterFactory
                   default_filter_factory;
```

説明 サブスクライブするときに使用します。このオペレーションは、デフォルトの FilterFactory オブジェクトを取得するためにサブスクライバ・アプリケーションで使用されます。

戻り値 デフォルトの FilterFactory オブジェクトのオブジェクト・リファレンスが返されます。

例 **注記** ここで紹介するコード例は全体の一部分です。完全なコード例については、第 4 章の 10 ページ「イベント・チャネル、ConsumerAdmin オブジェクト、およびフィルタ・ファクトリ・オブジェクトの取得」を参照してください。

C++ コード例

```
channel->default_filter_factory();
```

Java コード例

CosNotifyChannelAdmin::EventChannelFactory クラス

このクラスは、イベント・ポスト元アプリケーションによって使用されます。このクラスの OMG IDL は次のとおりです。

```
Module CosNotifyChannelAdmin
{
 interface EventChannelFactory {
     EventChannel get_event_channel ( in ChannelID id )
     raises (ChannelNotFound);
   };
}; //CosNotifyChannelAdmin
```

CosNotifyChannelAdmin::EventChannelFactory::get_event_channel

概要 EventChannel オブジェクトを取得します。

OMG IDL

```
EventChannel get_event_channel ( in ChannelID id )
     raises (ChannelNotFound);
```

例外 CosNotifyChannelAdmin::ChannelNotFound
チャネルを見つけることができないことを示します。

説明 イベントをサブスクライブ、アンサブスクライブ、およびポストするときに使用します。このオペレーションは、EventChannel オブジェクトを取得するためにアプリケーションで使用されます。サブスクライブのときには、EventChannel オブジェクトを使用してフィルタ・ファクトリ・オブジェクトと ConsumerAdmin オブジェクトを取得します。アンサブスクライブのときには、EventChannel オブジェクトを使用して ConsumerAdmin オブジェクトを取得します。イベントをポストするときには、EventChannel オブジェクトを使用して SupplierAdmin オブジェクトを取得します。ChannelID パラメータは、Tobj_Notification::DEFAULT_CHANNEL に設定しなければなりません。そのように設定しないと、ChannelNotFound 例外が発生します。

戻り値 デフォルト・イベント・チャネルのオブジェクト・リファレンスが返されます。

例 **注記** ここで紹介するコード例は全体の一部分です。完全なコード例については、第 4 章の 3 ページ「イベント・チャネルの取得」および第 4 章の 10 ページ「イベント・チャネル、ConsumerAdmin オブジェクト、およびフィルタ・ファクトリ・オブジェクトの取得」を参照してください。

C++ コード例

```
channel_factory->get_event_channel(
    Tobj_Notification::DEFAULT_CHANNEL );
```

Java コード例

```
channel_factory.get_event_channel(DEFAULT_CHANNEL.value);
```

CosNotifyComm::StructuredPushConsumer インターフェイス

このインターフェイスは、イベントの配信を目的としてイベント・サブスクライバ・アプリケーションによって使用されます。このインターフェイスをインプリメントしないと、ノーティフィケーション・サービスではサブスクライバにイベントを配信できません。このインターフェイスには、インプリメントしなければならない 3 つのメソッドがあります。

このクラスの OMG IDL は次のとおりです。

```
Module CosNotifyComm
{
 interface StructuredPushConsumer : NotifyPublish {

    void push_structured_event(
          in CosNotification::StructuredEvent event)
      raises(CosEventComm::Disconnected);
    void disconnect_structured_push_consumer:
    // 次のオペレーションが継承される
    void offer_change(
          in CosNotification::EventTypeSeq added,
          in CosNotification::EventTypeSeq removed )
      raises ( InvalidEventType );
  };
}; //CosNotifyComm
```

CosNotifyComm::StructuredPushConsumer::push_structured_event

概要 構造化イベントを配信します。

OMG IDL
```
void push_structured_event(
        in CosNotification::StructuredEvent event)
    raises(CosEventComm::Disconnected);
```

例外 `CosEventComm::Disconnected`
サブスクライバではこの例外を発生させてはなりません。

説明 サブスクライブするときに使用します。このオペレーションは、サブスクライバのコールバック・オブジェクトによってインプリメントされ、構造化イベントが配信されるたびにノーティフィケーション・サービスによって呼び出されます。このオペレーションには、1 つの入力パラメータ (構造化イベント) があります。

注記 このオペレーションは、トランザクションでは呼び出されません。また、このオペレーションは呼び出されてもすぐに復帰しなければなりません。その理由は、このオペレーションが復帰するまで、ノーティフィケーション・サービスではほかのサブスクライバへのイベント配信が開始されないからです。

例 **注記** ここで紹介するコード例は全体の一部分です。完全なコード例については、第 4 章の 8 ページ「CosNotifyComm::StructuredPushConsumer インターフェイスのインプリメント」を参照してください。

C++ コード例

```
virtual void push_structured_event(
      const CosNotification::StructuredEvent& notification );
       {
       // イベントを処理
       }
```

Java コード例

CosNotifyComm::StructuredPushConsumer:: disconnect_structured_push_consumer

概要 呼び出されることはありません。

OMG IDL
```
void disconnect_structured_push_consumer;
```

説明 このオペレーションが呼び出されることはありません。サブスクライバ・アプリケーションでは、このオペレーションのスタブ・アウト・バージョンを用意する必要があります。

例 **C++ コード例**

```
virtual void push_structured_event(
      const CosNotification::StructuredEvent& notification );
  {
    throw new CORBA::NO_IMPLEMENT();
  }
```

Java コード例

```
public void disconnect_structured_push_consumer()
  {
    throw new CORBA::NO_IMPLEMENT();
  }
```

CosNotifyComm::StructuredPushConsumer::Offer_change

概要 呼び出されることはありません。

OMG IDL

```
void offer_change(
          in CosNotification::EventTypeSeq added,
          in CosNotification::EventTypeSeq removed )
      raises ( InvalidEventType );
```

例外 CosNotifyComm::InvalidEventType
サブスクライバではこの例外を発生させてはなりません。

説明 このオペレーションが呼び出されることはありません。サブスクライバ・アプリケーションでは、このオペレーションのスタブ・アウト・バージョンを用意する必要があります。

例 **C++ コード例**

```
virtual void offer_change(
      const CosNotification::EventTypeSeq& added,
      const CosNotification::EventTypeSeq& removed )
    {
      throw CORBA::NO_IMPLEMENT();
    }
```

Java コード例

```
public void offer_change(EventType[] added, EventType[] removed)
{
  throw new NO_IMPLEMENT();
}
```

例外のマイナー・コード

この節では、ノーティフィケーション・サービスの例外シンボルとマイナー・コードについて説明します。マイナー・コードは、Tobj_Events.idl ファイルと Tobj_Notification.idl ファイルにあります。それらのファイルは、tuxdir\include ディレクトリ (Microsoft Windows の場合) および tuxdir/include ディレクトリ (UNIX の場合) に配置されています。

表 2-4 と 表 2-5 は、それぞれ Tobj_Events 例外と Tobj_Notification 例外の例外シンボルと対応するマイナー・コードを示しています。CORBA システム・イベントにはマイナー・コード・フィールドがあり、それらのマイナー・コードも以下の表で定義されています。

注記 表の中の例外シンボルは、より高いレベルの例外 (CORBA::IMP_LIMIT、CORBA::CORBA::BAD_PARAM、CORBA::BAD_INV_ORDER、CORBA::INV_OBHJREF、および CORBA::OBJECT_NOT_EXIST) に基づいて構成されており、アルファベット順にリストされています。

表 2-4 Tobj_Events 例外のマイナー・コード

例外シンボル	定義	マイナー・コード (16 進数)
CORBA::IMP_LIMIT 例外		
Tobj_Events:: POST_DOMAIN_AND_TYPE_TOO_LONG この例外の発生元 ■ Tobj_SimpleEvents::Channel:: push_structured_event ■ CosNotifyChannelAdmin:: StructuredProxyPushConsumer:: push_structured_event	イベントをポストするときに、組み合わせた長さが 31 文字を超えるドメイン名と型名が指定されました。	5455580D
Tobj_Events:: POST_DOMAIN_CONTAINS_SEPARATOR この例外の発生元 ■ Tobj_SimpleEvents::Channel:: push_structured_event ■ CosNotifyChannelAdmin:: StructuredProxyPushConsumer:: push_structured_event	イベントをポストするときに、「.」文字の含まれているドメイン名が指定されました。	54555802
Tobj_Events::POST_EMPTY_DOMAIN この例外の発生元 ■ Tobj_SimpleEvents::Channel:: push_structured_event ■ CosNotifyChannelAdmin:: StructuredProxyPushConsumer:: push_structured_event	イベントをポストするときに、空のドメイン名が指定されました。	5455580B

表 2-4 Tobj_Events 例外のマイナー・コード (続き)

例外シンボル	定義	マイナー・コード (16 進数)
Tobj_Events::POST_EMPTY_TYPE この例外の発生元 ■ Tobj_SimpleEvents::Channel::push_structured_event ■ CosNotifyChannelAdmin::StructuredProxyPushConsumer::push_structured_event	イベントをポストするときに、空の型名が指定されました。	5455580C
Tobj_Events::POST_SYSTEM_EVENTS_UNSUPPORTED この例外の発生元 ■ Tobj_SimpleEvents::Channel::push_structured_event ■ CosNotifyChannelAdmin::StructuredProxyPushConsumer::push_structured_event	イベントをポストするときに、BEA Tuxedo システム・イベント (つまり、ドメイン名が「TMEVT」で、型名が「.」文字で始まるイベント) のポストが試行されました。	54555804
Tobj_Events::POST_TYPE_CONTAINS_SEPARATOR この例外の発生元 ■ Tobj_SimpleEvents::Channel::push_structured_event ■ CosNotifyChannelAdmin::StructuredProxyPushConsumer::push_structured_event	イベントをポストするときに、「.」文字の含まれている型名が指定されました。	54555803
Tobj_Events::POST_UNSUPPORTED_PRIORITY_VALUE この例外の発生元 ■ Tobj_SimpleEvents::Channel::push_structured_event ■ CosNotifyChannelAdmin::StructuredProxyPushConsumer::push_structured_event	イベントをポストするときに、Variable ヘッダで「Priority」フィールドが追加されましたが、そのフィールドの値が 1 ～ 100 の範囲の「short」に設定されませんでした。	54555801

表 2-4 Tobj_Events 例外のマイナー・コード (続き)

例外シンボル	定義	マイナー・コード (16 進数)
Tobj_Events:: POST_UNSUPPORTED_VALUE_IN_ANY この例外の発生元 ■ Tobj_SimpleEvents:: Channel::push_structured_event ■ CosNotifyChannelAdmin:: StructuredProxyPushConsumer:: push_structured_event	イベントをポストするときに、構造化イベント・フィールドの「Any」の 1 つにサポートされていない型 (構造体、共用体、シーケンスなど) が挿入されました。そのサポートされていない型は、Variable ヘッダの値フィールド、フィルタ処理可能データの値フィールド、または remainder_of_body フィールドに挿入されています。	54555800
Tobj_Events:: SUB_DOMAIN_AND_TYPE_TOO_LONG この例外の発生元 ■ Tobj_SimpleEvents::Channel:: subscribe ■ CosNotifyChannelAdmin:: StructuredProxyPushSupplier:: connect_structured_push_consumer	サブスクライブするときに、組み合わせた長さが 255 文字を超えるドメイン名と型名が指定されました。	54555809
Tobj_Events:: SUB_DOMAIN_BEGINS_WITH_SYSEV この例外の発生元 ■ Tobj_SimpleEvents::Channel:: subscribe ■ CosNotifyFilter::Filter:: add_constraints	サブスクライブするときに、「.」文字で始まるドメイン名が指定されました。	54555805
Tobj_Events::SUB_EMPTY_DOMAIN この例外の発生元 ■ Tobj_SimpleEvents::Channel:: subscribe ■ CosNotifyFilter::Filter:: add_constraints	サブスクライブするときに、空のドメイン名が指定されました。	54555807

表 2-4 Tobj_Events 例外のマイナー・コード (続き)

例外シンボル	定義	マイナー・コード (16 進数)
Tobj_Events::SUB_EMPTY_TYPE この例外の発生元 ■ Tobj_SimpleEvents::Channel::subscribe ■ CosNotifyFilter::Filter::add_constraints	サブスクライブするときに、空の型名が指定されました。	54555808
Tobj_Events::SUB_FILTER_TOO_LONG この例外の発生元 ■ Tobj_SimpleEvents::Channel::subscribe ■ CosNotifyFilter::Filter::add_constraints	255 文字を超えるデータ・フィルタ式が指定されました。	5455580A
Tobj_Events::SUB_NAME_TO_LONG この例外の発生元 ■ Tobj_SimpleEvents::Channel::push_structured_event ■ CosNotifyChannelAdmin::StructuredProxyPushConsumer::push_structured_event	サブスクライブするときに、127 文字を超えるサブスクリプション名が指定されました。	5455580E
Tobj_Events::TRANSIENT_ONLY_CONFIGURATION この例外の発生元 ■ Tobj_SimpleEvents::Channel::subscribe ■ CosNotifyChannelAdmin::StructuredProxyPushSupplier::connect_structured_push_consumer	永続的なサブスクリプションの作成が試行されましたが、システムが一時的なサブスクリプションのみをサポートするようにコンフィギュレーションされていました。	54555806

表 2-4 Tobj_Events 例外のマイナー・コード (続き)

例外シンボル	定義	マイナー・コード (16 進数)
CORBA::BAD_PARAM 例外		
Tobj_Events::INVALID_CHANNEL_ID この例外の発生元 ■ Tobj_SimpleEvents::ChannelFactory::find_channel	シンプル・イベント API を使用してチャネルを検索するときに、無効なチャネル ID (つまり、Tobj_SimpleEvents::DEFAULT_CHANNEL ではないチャネル ID) が指定されました。	54555813
Tobj_Events::INVALID_SUBSCRIPTION_ID この例外の発生元 ■ Tobj_SimpleEvents::Channel::unsubscribe ■ CosNotifyChannelAdmin::ConsumerAdmin::get_proxy_supplier ■ Tobj_SimpleEvents::Channel::exists	シンプル・イベント API を使用してアンサブスクライブするときに、無効なサブスクリプション ID (存在しないサブスクリプション ID または CosNotification サブスクリプション ID) が指定されました。 CosNotification サービス API を使用してサブスクリプションを検索するときに、無効なサブスクリプション ID (存在しないサブスクリプション ID またはシンプル・イベント API サブスクリプション ID) が指定されました。 BEA シンプル・イベント API を使用して exists オペレーションを呼び出すときに、CosNotification subscription_id が渡されました。	54555812

表 2-4 Tobj_Events 例外のマイナー・コード (続き)

例外シンボル	定義	マイナー・コード (16 進数)
Tobj_Events:: SUB_INVALID_FILTER_EXPRESSION この例外の発生元 ■ Tobj_SimpleEvents::Channel:: subscribe ■ CosNotifyFilter::Filter:: add_constraints	サブスクライブするときに、無効なデータ・フィルタ式が指定されました。フィルタ式が無効になるのは、式に構文エラーがあるか、式のフィールド名の 1 つが FML フィールドとして定義されていない場合です。 データのフィルタ処理が行われるすべてのフィールドが格納される FML フィールド・テーブルが正しく作成されていることを確認し、フィールド・テーブル・ファイルが検索できるように UBBCONFIG ファイルが正しくコンフィギュレーションされていることを確認してください。	54555810

表 2-4 Tobj_Events 例外のマイナー・コード (続き)

例外シンボル	定義	マイナー・コード (16 進数)
Tobj_Events:: SUB_UNSUPPORTED_QOS_VALUE この例外の発生元 ■ Tobj_SimpleEvents::Channel:: subscribe ■ CosNotifyChannelAdmin:: StructuredProxyPushSupplier:: set_qos	サブスクライブするときに、サブスクリプションの無効なサービス品質が指定されました。 シンプル・イベント API の場合では、このことは指定されたサービス品質が次の要件のいずれかを満たしていないことを意味します。 ■ シーケンスの長さは 1 でなければなりません。 ■ 名前は Tobj_SimpleEvents::SUBSCRIPTION_TYPE でなければなりません。 ■ 値は Tobj_SimpleEvents::TRANSIENT_SUBSCRIPTION または Tobj_SimpleEvents::PERSISTENT_SUBSCRIPTION のいずれかでなければなりません。 CosNotification サービス API の場合では、このことは指定されたサービス品質が次の要件のいずれかを満たしていないことを意味します。 ■ サービスの品質には、名前が Tobj_Notification::SUBSCRIPTION_TYPE で、値が Tobj_Notification::TRANSIENT_SUBSCRIPTION または Tobj_Notification::PERSISTENT_SUBSCRIPTION の名前 / 値ペアが含まれていなければなりません。 ■ サービス品質の内容は、名前が Tobj_Notification::SUBSCRIPTION_NAME で、値がサブスクリプションの管理名の文字列である名前 / 値ペアでもかまいません。	54555811

表 2-4 Tobj_Events 例外のマイナー・コード (続き)

例外シンボル	定義	マイナー・コード (16 進数)
CORBA::INV_OBHJREF		
Tobj_Events:: SUB_NIL_CALLBACK_REF この例外の発生元 ■ Tobj_SimpleEvents::Channel:: subscribe ■ CosNotifyChannelAdmin:: StructuredProxyPushSupplier:: connect_structured_push_consumer	サブスクライブするときに、イベントを受信するコールバック・オブジェクトの NIL オブジェクト・リファレンスが指定されました。	54555830

表 2-5 Tobj_Notification 例外のマイナー・コード

例外シンボル	定義	マイナー・コード (16 進数)
CORBA::IMP_LIMIT 例外		
Tobj_Notification:: SUB_ADD_CONS_ON_TIMED_OUT_FILTER この例外の発生元 ■ CosNotifyFilter::Filter:: add_constraints	フィルタの作成後、CosNotification サブスクライバは 5 分以上経ってからそのフィルタの add_constraints を呼び出しました。そのフィルタは破棄されており (時間切れ)、サブスクライバでは新しいフィルタを作成しなければなりません。	54555858
Tobj_Notification:: SUB_ADD_CONS_TO_ADDED_FILTER この例外の発生元 ■ CosNotifyFilter::Filter:: add_constraints	プロキシに既に追加されているフィルタの add_constraints が CosNotification サブスクライバによって呼び出されました。	5455585E

表 2-5 Tobj_Notification 例外のマイナー・コード (続き)

例外シンボル	定義	マイナー・コード (16 進数)
Tobj_Notification:: SUB_ADDED_TIMED_OUT_FILTER この例外の発生元 ■ CosNotifyChannelAdmin:: StructuredProxyPushSupplier:: add_filter	フィルタを作成してその「add_constraints」を呼び出した後、CosNotification サブスクライバは 5 分以上経ってからフィルタをプロキシに追加する add_filter を呼び出しました。そのフィルタは破棄されており (時間切れ)、サブスクライバでは新しいフィルタを作成しなければなりません。	5455585D
Tobj_Notification:: SUB_ADD_FILTER_AFTER_CONNECT この例外の発生元 ■ CosNotifyChannelAdmin:: StructuredProxyPushSupplier:: add_filter	プロキシへの接続後に、CosNotification サブスクライバが add_filter を呼び出しました。	54555852
Tobj_Notification:: SUB_CANT_SET_QOS_AFTER_CONNECT この例外の発生元 ■ CosNotifyChannelAdmin::Structured ProxyPushSupplier::set_qos	プロキシへの接続後に、CosNotification サブスクライバが set_qos を呼び出しました。	54555856
Tobj_Notification:: SUB_MULTIPLE_CALLS_TO_ADD_CONS この例外の発生元 ■ CosNotifyFilter::Filter:: add_constraints	CosNotification サブスクライバがフィルタで add_constraints を複数回呼び出しました。	54555859
Tobj_Notification:: SUB_MULTIPLE_CALLS_TO_SET_FILTER この例外の発生元 ■ CosNotifyChannelAdmin:: StructuredProxyPushSupplier:: add_filter	CosNotification サブスクライバがプロキシで add_filter を複数回呼び出しました。	54555851

表 2-5 Tobj_Notification 例外のマイナー・コード (続き)

例外シンボル	定義	マイナー・コード (16 進数)
Tobj_Notification:: SUB_MULTIPLE_CALLS_TO_SET_QOS この例外の発生元 ■ CosNotifyChannelAdmin:: StructuredProxyPushSupplier:: set_qos	CosNotification サブスクライバがプロキシで set_qos を複数回呼び出しました。	54555855
Tobj_Notification:: SUB_MULTIPLE_CONSTRAINTS_IN_LIST この例外の発生元 ■ CosNotifyFilter::Filter:: add_constraints	CosNotification サブスクライバがフィルタで add_constraints を呼び出したときに、複数の項目から成る制約のリストが渡されました。つまり、サブスクライバは 1 つのデータ・フィルタではなくデータ・フィルタのリストを送信しようとしました。	5455585A
Tobj_Notification:: SUB_MULTIPLE_TYPES_IN_CONSTRAINT この例外の発生元 ■ CosNotifyFilter::Filter:: add_constraints	CosNotification サブスクライバがフィルタで add_constraints を呼び出したときに、複数のドメイン / 型セットから成る制約が渡されました。つまり、サブスクライバは 1 つのイベント型ではなくイベント型のリストを送信しようとしました。	5455585B
Tobj_Notification:: SUB_NIL_FILTER_REF この例外の発生元 ■ CosNotifyChannelAdmin:: StructuredProxyPushSupplier:: add_filter	CosNotification サブスクライバが add_filter に NIL フィルタのオブジェクト・リファレンスを渡しました。	54555853

表 2-5 Tobj_Notification 例外のマイナー・コード (続き)

例外シンボル	定義	マイナー・コード (16 進数)
Tobj_Notification:: SUB_NO_CUSTOM_FILTERS この例外の発生元 ■ CosNotifyChannelAdmin:: StructuredProxyPushSupplier:: add_filter	CosNotification サブスクライバが、デフォルトのフィルタ・ファクトリで作成されたものではないフィルタ・オブジェクトを add_filter に渡しました。たとえば、CosNotification サブスクライバが「カスタム」フィルタ処理を実行するために CosNotifyFilter::Filter インターフェイスをインプリメントし、それらのフィルタ・オブジェクトの 1 つを add_filter に渡しました。	54555854
Tobj_Notification:: SUB_SET_FILTER_NOT_CALLED この例外の発生元 ■ CosNotifyChannelAdmin:: StructuredProxyPushSupplier:: connect_structured_push_ consumer	CosNotification サブスクライバが、プロキシに接続する前にプロキシに対する add_filter を呼び出しませんでした。	54555850
Tobj_Notification:: SUB_SET_QOS_NOT_CALLED この例外の発生元 ■ CosNotifyChannelAdmin:: StructuredProxyPushSupplier:: connect_structured_push_ consumer	CosNotification サブスクライバが、プロキシに接続する前にプロキシに対する add_filter を呼び出しませんでした。	54555857

表 2-5 Tobj_Notification 例外のマイナー・コード (続き)

例外シンボル	定義	マイナー・コード (16 進数)
Tobj_Notification:: SUB_SYSTEM_EVENTS_UNSUPPORTED この例外の発生元 ■ CosNotifyChannelAdmin:: StructuredProxyPushSupplier:: set_qos	CosNotification サブスクライバが、ドメイン名「TMEVT」と「.」で始まる型名を渡しました。つまり、CosNotification サブスクライバは Tuxedo システム・イベントをサブスクライブしようとしました。システム・イベントのサブスクライブはサポートされていません。それはシンプル・イベント API でのみサポートされています。	5455585C
Tobj_Notification:: SUB_UNSUPPORTED_CLIENT_TYPE この例外の発生元 ■ ConsumerAdmin:: obtain_notification_push_ supplier ■ SupplierAdmin:: obtain_notification_push_ consumer	プロキシを作成するときに、CosNotification のサブスクライバまたはポスト元が CosNotifyChannelAdmin::STRUCTURED_EVENT 以外のクライアント型を渡しました。	5455585F

表 2-5 Tobj_Notification 例外のマイナー・コード (続き)

例外シンボル	定義	マイナー・コード (16 進数)
CORBA::OBJECT_NOT_EXIST 例外		
Tobj_Notification::SUBSCRIPTION_DOESNT_EXIST この例外の発生元 ■ StructuredProxyPushSupplier::add_filter ■ StructuredProxyPushSupplier::set_qos ■ StructuredProxyPushSupplier::connect_structured_push_consumer ■ StructuredProxyPushSupplier::disconnect_structured_push_supplier **注記** connect_structured_push_consumer はこの例外を発生される可能性があります。その理由は、ユーザがプロキシを作成し、ntsadmin ユーティリティを使用してサブスクリプションを削除してから、プロキシで connect_structured_push_consumer を呼び出すことができるからです。	CosNotification サブスクライバが既に破棄されているプロキシのメソッドを呼び出しました。そのプロキシは、次のいずれかのアクションによって破棄されています。 ■ CosNotification サブスクライバがプロキシとの接続を解除した。 ■ CosNotification サブスクライバがプロキシ作成から 5 分以上経ってからプロキシに接続した。つまり、サブスクリプションの完了に 5 分以上を要した。 ■ 管理者が ntsadmin ユーティリティを使用してサブスクリプションを破棄した。	54555880

3 BEA シンプル・イベント API の使い方

この章では、BEA シンプル・イベント API と C++ および Java プログラミング言語を使用してノーティフィケーション・サービス・アプリケーションを作成するための開発手順を説明します。

ここでは、次の内容について説明します。

- 開発プロセス
- ステップ 1: イベントをポストするアプリケーションの記述
- ステップ 2: イベントをサブスクライブするアプリケーションの記述
- ステップ 3: ノーティフィケーション・サービス・アプリケーションのコンパイルと実行

開発プロセス

表 3-1 は、ノーティフィケーション・サービス・アプリケーションの開発プロセスの概略です。

表 3-1 開発プロセス

手順	説明
1	イベントの設計
2	イベントをポストするアプリケーションの記述
3	イベントをサブスクライブするアプリケーションの記述

表 3-1 開発プロセス (続き)

手順	説明
4	ノーティフィケーション・サービス・アプリケーションのコンパイル

これらのステップは、以降の節で詳しく説明されています。

イベントの設計

イベントの設計は、あらゆるノーティフィケーション・サービスの基本です。イベントの設計は、一致するサブスクリプションに配信される情報の量だけでなく、ノーティフィケーション・サービスの効率と性能にも影響します。したがって、計画を慎重に行って、ノーティフィケーション・サービスが現在のニーズだけでなく将来の規模拡大にも対応できるようにする必要があります。イベント設計の説明については、第 2 章の 7 ページ「イベントの設計」を参照してください。

ステップ 1: イベントをポストするアプリケーションの記述

次のタイプの CORBA アプリケーションは、イベントをポストできます。

注記 BEA Tuxedo 8.0 では、Java のクライアントと共同クライアント / サーバがサポートされていますが、Java サーバはサポートされていません。Java サーバは BEA WebLogic Enterprise 製品のバージョン 5.0 と 5.1 ではサポートされていましたが、BEA WebLogic Enterprise がリリース 8.0 で BEA Tuxedo と結合された時点でそのサポートは除外されました。

- C++ のクライアント、共同クライアント / サーバ、およびサーバ
- Java のクライアントと共同クライアント / サーバ

- 外部 ORB クライアント

イベントをポストするために、アプリケーションでは最低でも次の機能をインプリメントする必要があります。

- イベント・チャネル・ファクトリのオブジェクト・リファレンスを取得し、それを利用してイベント・チャネルを取得します。
- イベントを作成してポストします。

以降の節では、これらの各機能について説明します。

イベント・チャネルの取得

最初にイベント・チャネルを取得しないと、クライアント・アプリケーションではイベントをポストできません。

この開発ステップは、リスト 3-1 で示されています。リスト 3-1 は、BEA シンプル・イベント API を使用する Notification Service サンプル・アプリケーションに基づいています。

イベント・チャネル・ファクトリのオブジェクト・リファレンスを取得するために、"Tobj_SimpleEventsService" 環境オブジェクトを使用して Bootstrap オブジェクトの resolve_initial_references メソッドが呼び出されます。取得したオブジェクト・リファレンスは、チャネル・ファクトリを取得するために使用します。チャネル・ファクトリは、イベント・チャネルを取得するために使用します。リスト 3-1 と リスト 3-2 は、C++ および Java のコード例を示しています。

リスト 3-1 イベント・チャネルの取得 (C++)

```
// シンプル・イベント・チャネル・ファクトリのオブジェクト・リファレンスを取得
CORBA::Object_var channel_factory_oref =
  bootstrap.resolve_initial_references(
                    "Tobj_SimpleEventsService");

Tobj_SimpleEvents::ChannelFactory_var channel_factory =
  Tobj_SimpleEvents::ChannelFactory::_narrow(
                     channel_factory_oref.in());

// チャネル・ファクトリを使用してデフォルト・チャネルを取得
Tobj_SimpleEvents::Channel_var channel =
  channel_factory->find_channel(
                     Tobj_SimpleEvents::DEFAULT_CHANNEL);
```

リスト 3-2 イベント・チャネルの取得 (Java)

```
// シンプル・イベント・チャネル・ファクトリのオブジェクト・リファレンスを取得
org.omg.CORBA.Object channel_factory_oref =
      bootstrap.resolve_initial_references(
                "Tobj_SimpleEventsService");

// チャネル・ファクトリを使用してデフォルト・チャネルを取得
ChannelFactory channel_factory =
    ChannelFactoryHelper.narrow(channel_factory_oref);

Channel channel =
    channel_factory.find_channel(DEFAULT_CHANNEL.value);
```

イベントの作成とポスト

イベントをポストするには、まずイベントを作成しなければなりません。以下のリストは、Notification Service サンプル・アプリケーションに基づいています。

リスト 3-3 と リスト 3-4 は、それぞれ C++ と Java でのインプリメンテーションを示しています。イベント・チャネルにニュースを報告するために、このアプリケーションでは次の手順が実行されます。

1. イベントを作成し、ドメイン名と型名を設定します。コード・サンプルでは、ドメイン名は「News」、イベント型は「Sports」に設定されます。
2. ニュース記事を格納するフィールドをイベントのフィルタ処理可能データに追加して、その追加フィールドの名前を「Story」に設定し、値を記事の文字列に設定します。
3. push_structured_event オペレーションを使用して、イベントをノーティフィケーション・サービスにポストします。

リスト 3-3 イベントの作成とポスト (C++)

```
// イベントを作成
CosNotification::StructuredEvent notification;

// ドメインを「News」に設定
notification.header.fixed_header.event_type.domain_name =
    CORBA::string_dup("News");
```

```
// 型をニュース・カテゴリに設定
notification.header.fixed_header.event_type.type_name =
    CORBA::string_dup("Sports");

// 記事を格納するフィールドを 1 つイベントのフィルタ
// 処理可能データに追加。フィールドの名前は「Story」、
// 値は記事の文字列に設定
notification.filterable_data.length(1);
notification.filterable_data[0].name =
     CORBA::string_dup("Story");
notification.filterable_data[0].value <<= "John Smith wins again";

// イベントをポスト

// ドメインが「News」で、型がニュース・カテゴリと一致する
// イベントをサブスクライブしたサブスクライバがこのイベント
// を受信する

channel->push_structured_event(notification);
```

リスト 3-4 イベントの作成とポスト (Java)

```
// イベントを作成
StructuredEvent notification = new StructuredEvent();
// ヘッダの下位構造を作成
notification.header = new EventHeader();
notification.header.fixed_header = new FixedEventHeader();
    notification.header.fixed_header.event_type = new EventType();

// ドメインを「News」に設定
notification.header.fixed_header.event_type.domain_name = "News";

// 型をニュース・カテゴリに設定
notification.header.fixed_header.event_type.type_name = "Sports";

// このサンプルでは使用しないので、イベント名は空の
// 文字列に設定する
    notification.header.fixed_header.event_name = "";
// このサンプルでは使用しないので、Variable ヘッダは空にする
    notification.header.variable_header = new Property[0];

// 記事を格納するフィールドを 1 つイベントのフィルタ
// 処理可能データに追加。フィールドの名前は「Story」、
// 値は記事の文字列に設定
notification.filterable_data = new Property[1];
notification.filterable_data[0] = new Property();
notification.filterable_data[0].name = "Story";
notification.filterable_data[0].value = orb.create_any();
notification.filterable_data[0].value.insert_string(John Smith
wins again");
```

```
// このサンプルでは使用しないので、残りの本文は
// 新しい（空の） Any に設定する
notification.remainder_of_body = orb.create_any();

// イベントをポスト
channel.push_structured_event(notification);
```

ステップ 2: イベントをサブスクライブするアプリケーションの記述

次のタイプの CORBA アプリケーションは、イベントをサブスクライブできます。

注記 BEA Tuxedo 8.0 では、Java のクライアントと共同クライアント / サーバがサポートされていますが、Java サーバはサポートされていません。Java サーバは BEA WebLogic Enterprise 製品のバージョン 5.0 と 5.1 ではサポートされていましたが、BEA WebLogic Enterprise がリリース 8.0 で BEA Tuxedo と結合された時点でそのサポートは除外されました。

- C++ の共同クライアント / サーバとサーバ
- Java 共同クライアント / サーバ
- 外部 ORB クライアント

イベントをサブスクライブするために、アプリケーションでは最低でも次の機能をインプリメントする必要があります。

- `push_structured_event` オペレーションをサポートする CosNotifyComm OMG IDL インターフェイスをインプリメントします。
- イベント・チャネル・ファクトリのオブジェクト・リファレンスを取得し、それを利用してイベント・チャネルを取得します。
- コールバック・オブジェクトのオブジェクト・リファレンスが含まれるサブスクリプションを定義および作成します。
- CosNotifyComm::StructuredPushConsumer インターフェイスをインプリメントするコールバック・オブジェクトを作成します。

CosNotifyComm::StructuredPushConsumer インターフェイスのインプリメント

コールバック・オブジェクトでイベントを受信するためには、push_structured_event オペレーションをサポートする CosNotifyComm::StructuredPushConsumer インターフェイスをインプリメントする必要があります。一致するサブスクリプションがあるイベントが発生すると、ノーティフィケーション・サービスではコールバック・オブジェクトのこのオペレーションを呼び出してイベントをサブスクライバ・アプリケーションにプッシュします。

CosNotifyComm::StructuredPushConsumer インターフェイスでは、offer_change および disconnect_structured_push_consumer というオペレーションも定義されます。これらのオペレーションはノーティフィケーション・サービスで呼び出されないので、CORBA::NO_IMPLEMENT をスローするスタブ・アウト・バージョンをインプリメントする必要があります。

リスト 3-5 と リスト 3-6 は、このインターフェイスが C++ でどのようにインプリメントされるのかを示しています。

リスト 3-5 CosNotifyComm::StructuredPushConsumer インターフェイスのサンプル・インプリメンテーション (NewsConsumer_i.h)

```
#ifndef _news_consumer_i_h
#define _news_consumer_i_h

#include "CosNotifyComm_s.h"

// サーバント・クラスでニュース・イベントを受信するためには、
// CosNotifyComm::StructuredPushConsumer idl インターフェイス
// をインプリメントする必要がある

class NewsConsumer_i : public
POA_CosNotifyComm::StructuredPushConsumer

{
  public:

    // このメソッドはニュース・イベントの発生時に呼び出される

    virtual void push_structured_event(
      const CosNotification::StructuredEvent& notification
    );
```

```
    // OMG の CosNotifyComm::StructuredPushConsumer idl インターフェイス
    // では、offer_change メソッドと disconnect_structured_push_consumer
    // メソッドが定義される。ノーティフィケーション・サービスからそれらの
    // メソッドが呼び出されることはないので、ただ単に CORBA::NO_IMPLEMENT
    // 例外をスローさせる

    virtual void offer_change(
      const CosNotification::EventTypeSeq& added,
      const CosNotification::EventTypeSeq& removed )
    {
      throw CORBA::NO_IMPLEMENT();
    }

    virtual void disconnect_structured_push_consumer()
    {
      throw CORBA::NO_IMPLEMENT();
    }
};
#endif
```

リスト 3-6 CosNotifyComm::StructuredPushConsumer インターフェイスのサンプル・インプリメンテーション (NewsConsumer_i.cpp)

```
#include "NewsConsumer_i.h"
#include <iostream.h>

//-------------------------------------------------------------
// Subscriber.cpp は「News」イベントのシンプル・イベント・
// サブスクリプションを作成し、イベントが NewsConsumer_i オブジェクト
// に配信されるようにする。ニュース・イベントが発生すると（ユーザが
// Reporter アプリケーションを実行してニュース記事を報告すると発生する）、
// 次のメソッドが呼び出される

void NewsConsumer_i::push_structured_event(
  const CosNotification::StructuredEvent& notification )
  {
// イベントのフィルタ処理可能データの最初のフィールドから
// 記事を抽出する
char* story;
notification.filterable_data[0].value >>= story;

// コードを単純化するために、「story」は「null」ではないと想定

// イベントを出力
cout
    << "-----------------------------------------------------"
    << endl
    << "Category : "
    << notification.header.fixed_header.
                          event_type.type_name.in()
    << endl
```

```
    << "Story    : "
    << story
    << endl;
...
}
```

イベント・チャネルの取得

このステップは、イベント・ポスト元とイベント・サブスクライバの両方で同じです。このステップの説明については、第 3 章の 7 ページ「CosNotifyComm::StructuredPushConsumer インターフェイスのインプリメント」を参照してください。

コールバック・オブジェクトの作成

イベントを受信するためには、アプリケーションはサーバであることも必要です。つまり、アプリケーションでは、サブスクライバのサブスクリプションと一致するイベントが発生したときに呼び出すことができるコールバック・オブジェクトをインプリメントしなければなりません。

コールバック・オブジェクトを作成する手順は次のとおりです。

注記 この手順は、BEA Tuxedo CORBA 共同クライアント / サーバでの手順です。BEA Tuxedo CORBA サーバでもイベントをサブスクライブできます。

1. コールバック・オブジェクトを作成します。コールバック・オブジェクトは、BEAWrapper Callback API または CORBA ポータブル・オブジェクト・アダプタ (POA) を使用してインプリメントできます。
2. サーバントを作成します。
3. コールバック・サーバントのオブジェクト・リファレンスを作成します。

BEAWrapper コールバック・オブジェクトとそのメソッドの詳細については、『BEA Tuxedo CORBA プログラミング・リファレンス』の「共同クライアント / サーバ」を参照してください。

注記　BEAWrapper コールバック・オブジェクトを使用してコールバック・オブジェクトを作成する方法は、以下に説明します。POA を使用してコールバック・オブジェクトをインプリメントする方法については、『BEA Tuxedo CORBA サーバ間通信』を参照してください。

リスト 3-7 と リスト 3-9 は、それぞれ C++ および Java で、BEAWrapper コールバック・オブジェクトを使用してコールバック・オブジェクトを作成する方法を示しています。コード例では、`NewsConsumber_i servant` が作成され、`start_transient` メソッドを使用して一時的なオブジェクト・リファレンスが作成されます。

リスト 3-7　一時的なオブジェクト・リファレンスでコールバック・オブジェクトを作成するサンプル・コード (Introductory サンプル・アプリケーションの Subscriber.cpp)

```
// このクライアントではコールバックをサポートする必要があるので
// コールバック・ラッパー・オブジェクトを作成する

BEAWrapper::Callbacks wrapper(orb.in());

NewsConsumer_i* news_consumer_impl = new NewsConsumer_i;

CORBA::Object_var news_consumer_oref =
      wrapper.start_transient(
        news_consumer_impl,
        CosNotifyComm::_tc_StructuredPushConsumer->id()
      );

CosNotifyComm::StructuredPushConsumer_var
    news_consumer =
      CosNotifyComm::StructuredPushConsumer::_narrow(
        news_consumer_oref.in()
       );
```

サブスクリプションの作成

サブスクライバでイベントを受信するためには、ノーティフィケーション・サービスをサブスクライブする必要があります。一時的なサブスクリプションまたは永続的なサブスクリプションのいずれかを作成できます。

Introductory サンプル・アプリケーションの抜粋である リスト 3-8 と リスト 3-11 は、それぞれ C++ および Java で一時的なサブスクリプションを作成する方法を示しています。

次の手順を実行する必要があります。

1. サブスクリプションのサービス品質 (QoS) を一時的または永続的のいずれかに設定します。
2. subscription_name (オプション)、domain_name、type_name、および data_filter (オプション) を指定します。
3. サブスクリプションを作成します。サブスクリプションは、domain_name、type_name、data_filter (オプション)、およびサービスの品質 (QoS) を設定し、サブスクライバのコールバック・オブジェクトのオブジェクト・リファレンスをノーティフィケーション・サービスに提供します。

リスト 3-8 一時的なサブスクリプションの作成 (C++)

```
// サービスの品質を TRANSIENT に設定
CosNotification::QoSProperties qos;
qos.length(1);
qos[0].name =
   CORBA::string_dup(Tobj_SimpleEvents::SUBSCRIPTION_TYPE);
qos[0].value <<=
   Tobj_SimpleEvents::TRANSIENT_SUBSCRIPTION;

// 型をニュース・カテゴリに設定
const char* type = "Sports";
// サブスクリプションを作成する。ドメインを「News」に設定し、
// データ・フィルタを 30 を超える年齢に設定する
Tobj_SimpleEvents::SubscriptionID subscription_id =
  channel->subscribe(
    subscription_name,
          "News", // ドメイン
          "Sports",    // 型
    "Age > 30",  // データ・フィルタ
    qos,
    news_consumer.in()
  );
```

注記 データのフィルタ処理を利用する場合は、コンフィギュレーション・タスクも行う必要があります。データ・フィルタ処理のコンフィギュレーション要件については、第 7 章の 2 ページ「データ・フィルタのコンフィギュレーション」を参照してください。

C++ および Java の Advanced サンプル・アプリケーションのコードである リスト 3-9 と リスト 3-13 は、ノーティフィケーション・サービスの永続的なサブスクリプションを作成するコード記述手順を示しています。永続的なサブスクリプションを作成する手順は、前述の一時的なサブスクリプションを作成する手順と同じです。

注記 コード例では news_consumer コールバック・オブジェクトのオブジェクト・リファレンスが永続的であると想定されていますが、一時的なコールバック・オブジェクト・リファレンスで永続的なサブスクリプションを作成することもできます。一時的と永続的を対比させたコールバック・オブジェクト・リファレンスの説明については、表 2-3 を参照してください。

リスト 3-9 永続的なサブスクリプションの作成 (Advanced の Subscriber.cpp)

```
CosNotification::QoSProperties qos;
qos.length(1);
qos[0].name =
  CORBA::string_dup(Tobj_SimpleEvents::SUBSCRIPTION_TYPE);
qos[0].value <<= Tobj_SimpleEvents::PERSISTENT_SUBSCRIPTION;

CosNotifyComm::StructuredPushConsumer_var
  news_consumer =
    CosNotifyComm::StructuredPushConsumer::_narrow(
        news_consumer_oref.in()
      );

Tobj_SimpleEvents::SubscriptionID sub_id =
 channel->subscribe(
    subscription_info.subscription_name(),
          "News",      // ドメイン
          "Sports",    // 型
          "",          // データ・フィルタなし
    qos,
    news_consumer.in()
  )
);
```

C++ 共同クライアント / サーバ・アプリケーションのスレッドに関する考慮事項

共同クライアント / サーバ・アプリケーションは、まずクライアント・アプリケーションとして機能し、その後にサーバ・アプリケーションにスイッチすることができます。そのために、共同クライアント / サーバ・アプリケーションでは次の呼び出しを行って、スレッドの制御を完全にオブジェクト・リクエスト・ブローカ (ORB) に移します。

```
orb -> run();
```

共同クライアント / サーバ・アプリケーションのサーバ部分のメソッドで `ORB::shutdown()` が呼び出されると、共同クライアント / サーバ・アプリケーションのサーバ部分で `ORB::run()` が呼び出された後にすべてのサーバ・アクティビティが停止し、制御が文に戻ります。この状況でのみ、制御は共同クライアント / サーバ・アプリケーションのクライアント部分に戻ります。

クライアント・アプリケーションのスレッドは 1 つだけなので、共同クライアント / サーバ・アプリケーションのクライアント部分では共同クライアント / サーバ・アプリケーションのサーバ部分と CPU を共有する必要があります。この共有は、共同クライアント / サーバ・アプリケーションに実行すべきサーバ・アプリケーション作業があるかどうかを ORB でときどき確認することで実現します。その確認を実行するには、次のコードを使用します。

```
if ( orb->work_pending() ) orb->perform_work();
```

サーバ・アプリケーションの作業が完了すると、ORB は共同クライアント / サーバ・アプリケーションに戻り、クライアント・アプリケーションの機能が実行されます。共同クライアント / サーバ・アプリケーションでは、ORB での確認をときどき必ず行わなければなりません。確認を行わないと、共同クライアント / サーバ・アプリケーションでは呼び出しが処理されません。

共同クライアント / サーバ・アプリケーションが要求でブロックされている間は、ORB でコールバックを提供できないことに注意してください。共同クライアント / サーバ・アプリケーションが別の BEA Tuxedo CORBA サーバ・アプリケーションのオブジェクトを呼び出した場合、その応答を待つ間 ORB はブロックされます。ブロックされている間は ORB ではコールバックを提供できないので、コールバックは要求が完了するまでキューに入れられます。

ステップ 3: ノーティフィケーション・サービス・アプリケーションのコンパイルと実行

ノーティフィケーション・サービス・アプリケーション開発の最後のステップでは、アプリケーションをコンパイル、ビルド、および実行します。そのためには、次の手順を実行する必要があります。

1. ノーティフィケーション・サービスとイベント・ポスト元アプリケーションおよびイベント・サブスクライバ・アプリケーションの間のインターフェイスを定義するために必要なクライアント・スタブ・ファイルとスケルトン・ファイルを生成します。イベント・ポスト元アプリケーションとしては、クライアント、共同クライアント / サーバ、またはサーバが考えられます。イベント・サブスクライバ・アプリケーションとしては、共同クライアント / サーバまたはサーバが考えられます。
2. アプリケーション・コードをコンパイルし、スケルトン・ファイルおよびクライアント・スタブ・ファイルに対してリンクします。
3. アプリケーションをビルドします。
4. アプリケーションを実行します。

クライアント・スタブ・ファイルとスケルトン・ファイルの生成

クライアント・スタブ・ファイルとスケルトン・ファイルを生成するには、アプリケーションで使用されるノーティフィケーション IDL ファイルごとに idl コマンドを実行する必要があります。表 3-2 は、各タイプのサブスクライバで使用する idl コマンドを示しています。

表 3-2 idl コマンドの要件

言語	BEA Tuxedo CORBA 共同クライアント / サーバ	BEA Tuxedo CORBA サーバ
C++	idl -P	idl
Java	idltojava	BEA Tuxedo 8.0 以降ではサポートされていない

次に、idl コマンドの例を示します。

```
>idl -IC:\tuxdir\include C:\tuxdir\include\CosEventComm.idl
```

表 3-3 は、BEA シンプル・イベント・インターフェイスを使用する各タイプのノーティフィケーション・サービス・アプリケーションで必要な IDL ファイルを示しています。

表 3-3 ノーティフィケーション・サービス・アプリケーションで必要な IDL ファイル

アプリケーション・タイプ	必要な OMG IDL ファイル
イベント・ポスト元 (クライアント、共同クライアント / サーバ、またはサーバ)。スタブはすべてのファイルで必要	CosEventComm.idl CosNotification.idl CosNotifyComm.idl Tobj_Events.idl Tobj_SimpleEvents.idl

表 3-3 ノーティフィケーション・サービス・アプリケーションで必要な IDL ファイル

アプリケーション・タイプ	必要な OMG IDL ファイル
サブスクライバ (サーバまたは共同クライアント / サーバ)。スタブはすべてのファイルで必要。スケルトンは CosNotifyComm.idl ファイルで必要	CosEventComm.idl CosNotification.idl CosNotifyComm.idl Tobj_Events.idl Tobj_SimpleEvents.idl

アプリケーションのビルドと実行

ビルドの手続きは、ノーティフィケーション・サービス・アプリケーションのタイプによって異なります。表 3-4 は、各タイプのノーティフィケーション・サービス・アプリケーションをビルドするために使用するファイルのコマンドと種類を示しています。

表 3-4 アプリケーションのビルド要件

アプリケーション・タイプ	クライアント	共同クライアント / サーバ	サーバ
C++ のイベント・ポスト元	buildobjclient コマンドを使用して、アプリケーション・ファイルと IDL スタブをコンパイルします。	-P オプションを設定した buildobjclient コマンドを使用して、アプリケーション・ファイルと IDL スタブをコンパイルします。	buildobjserver コマンドを使用して、アプリケーション・ファイルと IDL クライアント・スタブをコンパイルします。
C++ のイベント・サブスクライバ	適用されません。	-P オプションを設定した buildobjclient コマンドを使用して、アプリケーション・ファイル、IDL スタブ、IDL スケルトンをコンパイルし、BEAWrapper ライブラリとリンクします。	buildobjserver コマンドを使用して、アプリケーション・ファイル、IDL スタブ、および IDL スケルトンをコンパイルします。

表 3-4 アプリケーションのビルド要件 (続き)

アプリケーション・タイプ	クライアント	共同クライアント / サーバ	サーバ
Java のイベント・ポスト元	javac コマンドを使用して、アプリケーション・ファイルと IDL スタブをコンパイルします。	javac コマンドを使用して、アプリケーション・ファイルと IDL ファイルをコンパイルします。	BEA Tuxedo 8.0 以降ではサポートされていません。
Java のイベント・サブスクライバ	適用されません。	javac コマンドを使用して、アプリケーション・ファイル、IDL ファイル、および IDL スケルトンをコンパイルします。	BEA Tuxedo 8.0 以降ではサポートされていません。

リスト 3-10 は、Microsoft Windows システム上の C++ ポスト元アプリケーション (Reporter.cpp) で使用するコマンドを示しています。C++ の実行可能ファイルを作成するために、idl コマンドが必要な IDL ファイルで実行され、buildobjclient コマンドによって C++ クライアント・アプリケーション・ファイルと IDL スタブがコンパイルされます。

リスト 3-10 C++ Reporter アプリケーションのビルドと実行のコマンド (Microsoft Windows)

```
# idl コマンドを実行
idl -IC:\tuxdir\include C:\tuxdir\include\CosEventComm.idl \
C:\tuxdir\include\CosNotification.idl \
C:\tuxdir\include\CosNotifyComm.idl \
C:\tuxdir\include\Tobj_Events.idl \
C:\tuxdir\include\Tobj_SimpleEvents.idl

# buildobjclient コマンドを実行
buildobjclient -v -o subscriber.exe -f " \
    -DWIN32                             \
    Reporter.cpp                        \
    CosEventComm_c.cpp                  \
    CosNotification_c.cpp               \
    CosNotifyComm_c.cpp                 \
    Tobj_Events_c.cpp                   \
    Tobj_SimpleEvents_c.cpp             \
    "
```

```
# アプリケーションを実行
is_reporter
```

リスト 3-11 と リスト 3-12 は、それぞれ Microsoft Windows および UNIX 上の C++ サブスクライバ・アプリケーション (Subscriber.cpp) で使用するコマンドを示しています。C++ の実行可能ファイルを作成するために、-P オプションを設定した buildobjclient コマンドによって、共同クライアント / サーバ・アプリケーション・ファイル (Subscriber.cpp と NewsConsumer_i.cpp)、IDL スタブ、および IDL スケルトン (CosNotifyComm_s.cpp) がコンパイルされます。

リスト 3-11 C++ Subscriber アプリケーションのビルドと実行のコマンド (Microsoft Windows)

```
# idl コマンドを実行
idl -P -IC:\tuxdir\include C:\tuxdir\include\CosEventComm.idl \
C:\tuxdir\include\CosNotification.idl \
C:\tuxdir\include\CosNotifyComm.idl \
C:\tuxdir\include\TobjvEvents.idl \
C:\tuxdir\include\Tobj_SimpleEvents.idl

# buildobjclient コマンドを実行
buildobjclient -v -P -o subscriber.exe -f "  \
  -DWIN32                                   \
  Subscriber.cpp                            \
  NewsConsumer_i.cpp                        \
  CosEventComm_c.cpp                        \
  CosNotification_c.cpp                     \
  CosNotifyComm_c.cpp                       \
  CosNotifyComm_s.cpp                       \
  Tobj_Events_c.cpp                         \
  Tobj_SimpleEvents_c.cpp                   \
c:\tuxdir\lib\libbeawrapper.lib             \
"

# アプリケーションを実行
is_subscriber
```

リスト 3-12 C++ Subscriber アプリケーションのビルドと実行のコマンド (UNIX)

```
# idl コマンドを実行
idl -P -I/usr/local/tuxdir/include
/usr/local/tuxdir/include/CosEventComm.idl \
```

```
/usr/local/tuxdir/include/CosNotification.idl \
/usr/local/tuxdir/include/CosNotifyComm.idl \
/usr/local/tuxdir/include/Tobj_Events.idl \
/usr/local/tuxdir/include/Tobj_SimpleEvents.idl

# buildobjclient コマンドを実行
buildobjclient -v -P -o subscriber -f "     \
  Subscriber.cpp                            \
  NewsConsumer_i.cpp                        \
  CosEventComm_c.cpp                        \
  CosNotification_c.cpp                     \
  CosNotifyComm_c.cpp                       \
  CosNotifyComm_s.cpp                       \
  Tobj_Events_c.cpp                         \
  Tobj_SimpleEvents_c.cpp                   \
"

# アプリケーションを実行
is_subscriber
```

リスト 3-13 は、リモートの Java ポスト元アプリケーションをリンク、ビルド、および実行するために使用するコマンドの例を示しています。

リスト 3-13 Java Reporter アプリケーションのリンク、ビルド、および実行のコマンド

```
# idltojava コマンドを実行
idltojava -IC:\tuxdir\include C:\tuxdir\include\CosEventComm.idl \
  C:\tuxdir\include\CosNotification.idl C:\tuxdir\include\CosNotifyComm.idl \
  C:\tuxdir\include\Tobj_Events.idl C:\tuxdir\include\Tobj_SimpleEvents.idl

# Java ファイルをコンパイル
javac -classpath C:\tuxdir\udataobj\java\jdk\m3envobj.jar Reporter.java

# Java .class ファイルを Java アーカイブ (JAR) ファイルに結合
jar cf reporter.jar Reporter.class org\omg\CosEventComm \
  org\omg\CosNotification org\omg\CosNotifyComm \
  com\beasys\Tobj_Events com\beasys\Tobj_SimpleEvents

# Reporter アプリケーションを実行
java -DTOBJADDR=//BEANIE:2359 -classpath \
  reporter.jar;C:\tuxdir\udataobj\java\jdk\m3envobj.jar Reporter
```

4 CosNotification サービス API の使い方

この章では、CosNotification サービス API と C++ および Java プログラミング言語を使用してノーティフィケーション・サービス・アプリケーションを作成するための開発手順を説明します。

ここでは、次の内容について説明します。

- 開発プロセス
- ステップ 1: イベントをポストするアプリケーションの記述
- ステップ 2: イベントをサブスクライブするアプリケーションの記述
- ステップ 3: ノーティフィケーション・サービス・アプリケーションのコンパイルと実行

開発プロセス

表 4-1 は、ノーティフィケーション・サービス・アプリケーションの開発プロセスの概略です。

表 4-1 開発プロセス

手順	説明
1	イベントの設計
2	イベントをポストするアプリケーションの記述
3	イベントをサブスクライブするアプリケーションの記述
4	ノーティフィケーション・サービス・アプリケーションのコンパイル

これらのステップは、以降の節で詳しく説明されています。

イベントの設計

イベントの設計は、あらゆるノーティフィケーション・サービスの基本です。イベントの設計は、一致するサブスクリプションに配信される情報の量だけでなく、ノーティフィケーション・サービスの効率と性能にも影響します。したがって、計画を慎重に行って、ノーティフィケーション・サービスが現在のニーズだけでなく将来の規模拡大にも対応できるようにする必要があります。イベント設計の説明については、第 2 章の 7 ページ「イベントの設計」を参照してください。

ステップ 1: イベントをポストするアプリケーションの記述

次のタイプの CORBA アプリケーションは、イベントをポストできます。

注記 BEA Tuxedo 8.0 では、Java のクライアントと共同クライアント / サーバがサポートされていますが、Java サーバはサポートされていません。Java サーバは BEA WebLogic Enterprise 製品のバージョン 5.0 と 5.1 ではサポートされていましたが、BEA WebLogic Enterprise がリリース 8.0 で BEA Tuxedo と結合された時点でそのサポートは除外されました。

- C++ のクライアント、共同クライアント / サーバ、およびサーバ
- Java のクライアントと共同クライアント / サーバ
- 外部 ORB クライアント

イベントをポストするために、アプリケーションでは最低でも次の機能をインプリメントする必要があります。

- イベント・チャネル・ファクトリのオブジェクト・リファレンスを取得し、それを利用してイベント・チャネルを取得する。

- イベントを作成してポストする。

以降の節では、これらの各機能について説明します。

イベント・チャネルの取得

最初にイベント・チャネルを取得しないと、クライアント・アプリケーションではイベントをポストできません。

この開発ステップは、リスト 4-1 で示されています。リスト 4-1 は、CosNotification サービス API を使用する Introductory サンプル・アプリケーションの Reporter.cpp ファイルのコードです。

イベント・チャネル・ファクトリのオブジェクト・リファレンスを取得するために、"NotificationService" 環境オブジェクトを使用して Bootstrap オブジェクトの resolve_initial_references メソッドが呼び出されます。取得したオブジェクト・リファレンスは、チャネル・ファクトリを取得するために使用します。チャネル・ファクトリは、イベント・チャネルを取得するために使用します。リスト 4-1 と リスト 4-2 は、C++ および Java のコード例を示しています。

リスト 4-1 イベント・チャネルの取得 (Reporter.cpp)

```
// CosNotification チャネル・ファクトリのオブジェクト・リファレンスを取得
CORBA::Object_var channel_factory_oref =
      bootstrap.resolve_initial_references(
        "NotificationService" );

CosNotifyChannelAdmin::EventChannelFactory_var
   channel_factory =
       CosNotifyChannelAdmin::EventChannelFactory::_narrow(
          channel_factory_oref.in() );

// チャネル・ファクトリを使用してデフォルト・チャネルを取得
CosNotifyChannelAdmin::EventChannel_var channel =
   channel_factory->get_event_channel(
       Tobj_Notification::DEFAULT_CHANNEL );
```

リスト 4-2 イベント・チャネルの取得 (Reporter.java)

```
import org.omg.CosNotification.*;//CosNotification API の一部
import org.omg.CosNotifyChannelAdmin.*;import // CosNotification API
```

```
                                                    // の残り
com.beasys.Tobj_Notification.*; // CosNotification API を使用する
                                // ときに必要な固有の定数
import com.beasys.Tobj.*;
import com.beasys.*;
import org.omg.CORBA.*;

import java.io.*;
// CosNotification チャネル・ファクトリのオブジェクト・リファレンスを取得
org.omg.CORBA.Object channel_factory_oref =
      bootstrap.resolve_initial_references("NotificationService");

EventChannelFactory channel_factory =
       EventChannelFactoryHelper.narrow(channel_factory_oref);

// チャネル・ファクトリを使用してデフォルト・チャネルを取得
EventChannel channel =
      channel_factory.get_event_channel(DEFAULT_CHANNEL.value);
```

イベントの作成とポスト

イベントをポストするには、SupplierAdmin オブジェクトを取得し、そのオブジェクトを使用してプロキシを作成し、イベントを作成して、そのイベントをプロキシにポストする必要があります。

リスト 4-3 と リスト 4-4 は、それぞれ C++ と Java でのインプリメンテーションを示しています。

リスト 4-3 イベントの作成とポスト (Reporter.cpp)

```
// イベントをポストする側なので、
// SupplierAdmin オブジェクトを取得する
CosNotifyChannelAdmin::SupplierAdmin_var supplier_admin =
      channel->default_supplier_admin();

// サプライヤ管理を使用してプロキシを作成する。イベントはそのプロキシ
// にポストされる （シンプル・イベント・インターフェイスの場合、イベント
// はチャネルにポストされる）
CosNotifyChannelAdmin::ProxyID proxy_id;
CosNotifyChannelAdmin::ProxyConsumer_var generic_proxy_consumer =
     supplier_admin->obtain_notification_push_consumer(
        CosNotifyChannelAdmin::STRUCTURED_EVENT, proxy_id );

CosNotifyChannelAdmin::StructuredProxyPushConsumer_var
   proxy_push_consumer =
```

```
      CosNotifyChannelAdmin::StructuredProxyPushConsumer::_narrow(
         generic_proxy_consumer );

// イベントをポストできるようにプロキシに接続
proxy_push_consumer->connect_structured_push_supplier(
       CosNotifyComm::StructuredPushSupplier::_nil() );

...
// イベントを作成
  CosNotification::StructuredEvent notification;

// ドメインを「News」に設定
notification.header.fixed_header.event_type.domain_name =
    CORBA::string_dup("News");

// 型をニュース・カテゴリに設定
notification.header.fixed_header.event_type.type_name =
    CORBA::string_dup("Sports");

// 記事を格納するフィールドを 1 つイベントのフィルタ
// 処理可能データに追加。フィールドの名前は「Story」、
// 値は記事の文字列に設定
notification.filterable_data.length(1);
notification.filterable_data[0].name =
           CORBA::string_dup("Story");
notification.filterable_data[0].value <<= "John Smith wins again";

// イベントをポスト
// ドメインが「News」で、型がニュース・カテゴリと一致する
// イベントをサブスクライブしたサブスクライバがこのイベント
// を受信する

proxy_push_consumer->push_structured_event(notification);

...
// 接続を解除
proxy_push_consumer->disconnect_structured_push_consumer();
```

リスト 4-4 イベントの作成とポスト (Reporter.java)

```
// イベントをポストする側なので、
// サプライヤ管理オブジェクトを取得する
SupplierAdmin supplier_admin =
    channel.default_supplier_admin();

// サプライヤ管理を使用してプロキシを作成する。イベントはそのプロキシ
// にポストされる （シンプル・イベント・インターフェイスの場合、イベント
// はチャネルにポストされる）
IntHolder proxy_id = new IntHolder();
   ProxyConsumer generic_proxy_consumer =
      supplier_admin.obtain_notification_push_consumer(
          ClientType.STRUCTURED_EVENT, proxy_id );
```

```
m_proxy_push_consumer =
     StructuredProxyPushConsumerHelper.narrow(
          generic_proxy_consumer);

// イベントをポストできるようにプロキシに接続
m_proxy_push_consumer.connect_structured_push_supplier(null);

...
// イベントを作成
StructuredEvent notification = new StructuredEvent();
notification.header = new EventHeader();

// ヘッダの下位構造を作成
notification.header.fixed_header = new FixedEventHeader();
notification.header.fixed_header.event_type = new EventType();

// ドメインを「News」に設定
notification.header.fixed_header.event_type.domain_name = "News";

// 型をニュース・カテゴリに設定
notification.header.fixed_header.event_type.type_name = "Sports";

// このサンプルでは使用しないので、イベント名は空の
// 文字列に設定する
notification.header.fixed_header.event_name = "";

// このサンプルでは使用しないので、Variable ヘッダは空にする
notification.header.variable_header = new Property[0];

// 記事を格納するフィールドを 1 つイベントのフィルタ
// 処理可能データに追加。フィールドの名前は「Story」、
// 値は記事の文字列に設定
notification.filterable_data = new Property[1];
notification.filterable_data[0] = new Property();
notification.filterable_data[0].name = "Story";
notification.filterable_data[0].value = orb.create_any();
notification.filterable_data[0].value.insert_string("John Smith
wins again");

// このサンプルでは使用しないので、残りの本文は
// 新しい（空の） Any に設定する
notification.remainder_of_body = orb.create_any();

m_proxy_push_consumer.push_structured_event(notification);

...
// 接続を解除
proxy_push_consumer.disconnect_structured_push_consumer();
```

ステップ 2: イベントをサブスクライブするアプリケーションの記述

次のタイプの CORBA アプリケーションは、イベントをサブスクライブできます。

注記 BEA Tuxedo 8.0 では、Java のクライアントと共同クライアント / サーバがサポートされていますが、Java サーバはサポートされていません。Java サーバは BEA WebLogic Enterprise 製品のバージョン 5.0 と 5.1 ではサポートされていましたが、BEA WebLogic Enterprise がリリース 8.0 で BEA Tuxedo と結合された時点でそのサポートは除外されました。

- C++ の共同クライアント / サーバとサーバ
- Java 共同クライアント / サーバ
- コールバックをサポートする外部 ORB クライアント

イベントをサブスクライブするために、アプリケーションでは最低でも次の機能をサポートする必要があります。

- `push_structured_event` オペレーションをサポートする CosNotifyComm OMG IDL インターフェイスをインプリメントします。
- イベント・チャネル・ファクトリのオブジェクト・リファレンスを取得し、それを利用してイベント・チャネルを取得します。
- コールバック・オブジェクトのオブジェクト・リファレンスが含まれるサブスクリプションを定義および作成します。
- `CosNotifyComm::StructuredPushConsumer` インターフェイスをインプリメントするコールバック・オブジェクトを作成します。

CosNotifyComm::StructuredPushConsumer インターフェイスのインプリメント

コールバック・サーバント・オブジェクトでイベントを受信するためには、push_structured_event オペレーションをサポートする CosNotifyComm::StructuredPushConsumer インターフェイスをインプリメントする必要があります。一致するサブスクリプションがあるイベントが発生すると、ノーティフィケーション・サービスではサブスクライバ・アプリケーションのサーバント・コールバック・オブジェクトのこのオペレーションを呼び出してイベントをサブスクライバ・アプリケーションに配信します。

CosNotifyComm::StructuredPushConsumer インターフェイスでは、offer_change および disconnect_structured_push_consumer というオペレーションも定義されます。これらのオペレーションはノーティフィケーション・サービスで呼び出されないので、CORBA::NO_IMPLEMENT をスローするスタブ・アウト・バージョンをインプリメントする必要があります。

リスト 4-5 と リスト 4-6 は、このインターフェイスが C++ でどのようにインプリメントされるのかを示しています。

リスト 4-5 CosNotifyComm::StructuredPushConsumer インターフェイスのサンプル・インプリメンテーション (NewsConsumer_i.h)

```
#ifndef _news_consumer_i_h
#define _news_consumer_i_h

#include "CosNotifyComm_s.h"

// サーバント・クラスでニュース・イベントを受信するためには、
// CosNotifyComm::StructuredPushConsumer idl インターフェイス
// をインプリメントする必要がある

class NewsConsumer_i : public
POA_CosNotifyComm::StructuredPushConsumer

{
  public:

    // このメソッドはニュース・イベントの発生時に呼び出される

    virtual void push_structured_event(
      const CosNotification::StructuredEvent& notification
    );
```

```
    // OMG の CosNotifyComm::StructuredPushConsumer idl インターフェイス
    // では、offer_change メソッドと disconnect_structured_push_consumer
    // メソッドが定義される。ノーティフィケーション・サービスからそれらの
    // メソッドが呼び出されることはないので、ただ単に CORBA::NO_IMPLEMENT
    // 例外をスローさせる

    virtual void offer_change(
      const CosNotification::EventTypeSeq& added,
      const CosNotification::EventTypeSeq& removed )
    {
      throw CORBA::NO_IMPLEMENT();
    }

    virtual void disconnect_structured_push_consumer()
    {
      throw CORBA::NO_IMPLEMENT();
    }
};
#endif
```

リスト 4-6 CosNotifyComm::StructuredPushConsumer インターフェイスのサンプル・インプリメンテーション (NewsConsumer_i.cpp)

```
#include "NewsConsumer_i.h"
#include <iostream.h>

//-------------------------------------------------------------
// Subscriber.cpp は「News」イベントのシンプル・イベント・
// サブスクリプションを作成し、イベントが NewsConsumer_i オブジェクト
// に配信されるようにする。ニュース・イベントが発生すると（ユーザが
// Reporter アプリケーションを実行してニュース記事を報告すると発生する）、
// 次のメソッドが呼び出される

void NewsConsumer_i::push_structured_event(
  const CosNotification::StructuredEvent& notification )
  {
// イベントのフィルタ処理可能データの最初のフィールドから
// 記事を抽出する
char* story;
notification.filterable_data[0].value >>= story;

// コードを単純化するために、「story」は「null」ではないと想定

// イベントを出力
cout
    << "-----------------------------------------------------"
    << endl
    << "Category : "
    << notification.header.fixed_header.
                          event_type.type_name.in()
    << endl
```

```
    << "Story    : "
    << story
    << endl;
...
}
```

イベント・チャネル、ConsumerAdmin オブジェクト、およびフィルタ・ファクトリ・オブジェクトの取得

最初にイベント・チャネル、ConsumerAdmin オブジェクト、およびフィルタ・ファクトリ・オブジェクトを取得しないと、アプリケーションではサブスクリプションを作成できません。リスト 4-7 は、それぞれ C++ と Java でのインプリメンテーションを示しています。

イベント・チャネル・ファクトリのオブジェクト・リファレンスを取得するために、"NotificationService" 環境オブジェクトを使用して Bootstrap オブジェクトの resolve_initial_references メソッドが呼び出されます。取得したオブジェクト・リファレンスは、チャネル・ファクトリを取得するために使用します。チャネル・ファクトリは、イベント・チャネルを取得するために使用します。最後に、イベント・チャネルは ConsumerAdmin オブジェクトと FilterFactory オブジェクトを取得するために使用します。

リスト 4-7 イベント・チャネル、ConsumerAdmin オブジェクト、およびフィルタ・ファクトリ・オブジェクトの取得 (Subscriber.cpp)

```
// CosNotification チャネル・ファクトリのオブジェクト・リファレンスを取得
CORBA::Object_var
  channel_factory_oref =
     bootstrap.resolve_initial_references(
          "NotificationService" );

CosNotifyChannelAdmin::EventChannelFactory_var
   channel_factory =
      CosNotifyChannelAdmin::EventChannelFactory::_narrow(
          channel_factory_oref.in() );

// チャネル・ファクトリを使用してデフォルト・チャネルを取得
CosNotifyChannelAdmin::EventChannel_var channel =
```

```
   channel_factory->get_event_channel(
      Tobj_Notification::DEFAULT_CHANNEL );

// チャネルを使用してコンシューマ管理とフィルタ・ファクトリを取得
CosNotifyChannelAdmin::ConsumerAdmin_var consumer_admin =
    channel->default_consumer_admin();

CosNotifyFilter::FilterFactory_var filter_factory =
    channel->default_filter_factory();
```

コールバック・オブジェクトの作成

イベントを受信するためには、アプリケーションはサーバであることも必要です。つまり、アプリケーションでは、サブスクライバのサブスクリプションと一致するイベントが発生したときに呼び出すことができるコールバック・オブジェクトをインプリメントしなければなりません。

コールバック・オブジェクトを作成する手順は次のとおりです。

注記 この手順は、共同クライアント / サーバでの手順です。BEA Tuxedo CORBA サーバでもイベントをサブスクライブできます。

1. コールバック・ラッパー・オブジェクトを作成します。コールバック・ラッパー・オブジェクトは、BEAWrapper コールバック・オブジェクトまたは CORBA ポータブル・オブジェクト・アダプタ (POA) を使用してインプリメントできます。
2. サーバントを作成します。
3. コールバック・サーバントのオブジェクト・リファレンスを作成します。

BEAWrapper コールバック・オブジェクトとそのメソッドの詳細については、『BEA Tuxedo CORBA プログラミング・リファレンス』の「共同クライアント / サーバ」を参照してください。

注記 BEAWrapper コールバック・オブジェクトを使用してコールバック・オブジェクトを作成する方法は、以下に説明します。POA を使用してコールバック・オブジェクトをインプリメントする方法については、『BEA Tuxedo CORBA サーバ間通信』を参照してください。

リスト 4-8 は、それぞれ C++ および Java で、BEAWrapper コールバック・オブジェクトを使用してコールバック・オブジェクトを作成する方法を示しています。コード例では、NewsConsumber_i servant が作成され、start_transient メソッドを使用して一時的なオブジェクト・リファレンスが作成されます。

リスト 4-8 一時的なオブジェクト・リファレンスでコールバック・オブジェクトを作成するサンプル・コード (Introductory サンプル・アプリケーションの Subscriber.cpp)

```
// このクライアントではコールバックをサポートする必要があるので
// コールバック・ラッパー・オブジェクトを作成する
BEAWrapper::Callbacks wrapper(orb.in());

NewsConsumer_i* news_consumer_impl = new NewsConsumer_i;

// このサーバントの一時的なオブジェクト・リファレンスを作成
CORBA::Object_var news_consumer_oref =
       wrapper.start_transient(
         news_consumer_impl,
         CosNotifyComm::_tc_StructuredPushConsumer->id()
          );

CosNotifyComm::StructuredPushConsumer_var
    news_consumer =
       CosNotifyComm::StructuredPushConsumer::_narrow(
           news_consumer_oref.in() );
```

サブスクリプションの作成

サブスクライバでイベントを受信するためには、ノーティフィケーション・サービスをサブスクライブする必要があります。一時的なサブスクリプションまたは永続的なサブスクリプションのいずれかを作成できます。

サブスクリプションを作成するには、次の手順を実行する必要があります。

1. ノーティフィケーション・プロキシ・プッシュ・サプライヤを作成し、それを利用して StructuredProxySupplier オブジェクトを作成します。
2. サブスクリプションのサービスの品質 (QoS) を設定します。QoS は、一時的または永続的に設定できます。
3. フィルタ・オブジェクトを作成し、domain_name、type_name、および data_filter (オプション) を割り当てます。

4. フィルタをプロキシに追加します。
5. サブスクリプションのコールバック・オブジェクト・リファレンスを渡してプロキシに接続します。

Introductory サンプル・アプリケーションのコードである リスト 4-9 は、それぞれ C++ および Java で一時的なサブスクリプションを作成する方法を示しています。

リスト 4-9 一時的なサブスクリプションの作成

```
// 新しいサブスクリプションを作成 ( この時点ではまだ完全ではない )
CosNotifyChannelAdmin::ProxyID subscription_id;
CosNotifyChannelAdmin::ProxySupplier_var generic_subscription =
      consumer_admin->obtain_notification_push_supplier(
          CosNotifyChannelAdmin::STRUCTURED_EVENT,
          subscription_id );

CosNotifyChannelAdmin::StructuredProxyPushSupplier_var
     subscription =
      CosNotifyChannelAdmin::StructuredProxyPushSupplier::_narrow(
            generic_subscription );
      s_subscription = subscription.in();

  // サービスの品質を設定する。これでサブスクリプション名
  // とサブスクリプションのタイプ (TRANSIENT) が設定される
  CosNotification::QoSProperties qos;
  qos.length(2);
  qos[0].name =
      CORBA::string_dup(Tobj_Notification::SUBSCRIPTION_NAME);
  qos[0].value <<= subscription_name;
  qos[1].name =
     CORBA::string_dup(Tobj_Notification::SUBSCRIPTION_TYPE);
  qos[1].value <<=
     Tobj_Notification::TRANSIENT_SUBSCRIPTION;

  subscription->set_qos(qos);

  // フィルタを作成 ( ドメイン、型、およびデータ・フィルタの指定に使用 )
  CosNotifyFilter::Filter_var filter =
          filter_factory->create_filter(
            Tobj_Notification::CONSTRAINT_GRAMMAR );
  s_filter = filter.in();

    // フィルタ処理パラメータを設定
    // ( ドメイン = "News"、型 = "Sports"、およびデータ・フィルタなし )
    CosNotifyFilter::ConstraintExpSeq constraints;
    constraints.length(1);
    constraints[0].event_types.length(1);
    constraints[0].event_types[0].domain_name =
          CORBA::string_dup("News");
```

```
    constraints[0].event_types[0].type_name =
          CORBA::string_dup("Sports");
    constraints[0].constraint_expr =
          CORBA::string_dup(""); // データ・フィルタなし

    CosNotifyFilter::ConstraintInfoSeq_var
add_constraints_results = // この戻り値は無視
            filter->add_constraints(constraints);

    // フィルタをサブスクリプションに追加
    CosNotifyFilter::FilterID filter_id =
       subscription->add_filter(filter.in());

    // サブスクリプションの名前、タイプ、およびフィルタ処理の各パラメータ
    // が設定されたので、イベントを配信するコールバック・オブジェクトの
    // オブジェクト・リファレンスを渡してサブスクリプションを完成する
    subscription->connect_structured_push_consumer(
         news_consumer.in() );
```

ステップ 3: ノーティフィケーション・サービス・アプリケーションのコンパイルと実行

ノーティフィケーション・サービス・アプリケーション開発の最後のステップでは、アプリケーションをコンパイル、ビルド、および実行します。そのためには、次の手順を実行する必要があります。

1. ノーティフィケーション・サービスとイベント・ポスト元アプリケーションおよびイベント・サブスクライバ・アプリケーションの間のインターフェイスを定義するために必要なクライアント・スタブ・ファイルとスケルトン・ファイルを生成します。イベント・ポスト元アプリケーションとしては、クライアント、共同クライアント / サーバ、またはサーバが考えられます。イベント・サブスクライバ・アプリケーションとしては、共同クライアント / サーバまたはサーバが考えられます。
2. アプリケーション・コードをコンパイルし、スケルトン・ファイルおよびクライアント・スタブ・ファイルに対してリンクします。
3. アプリケーションをビルドします。
4. アプリケーションを実行します。

クライアント・スタブ・ファイルとスケルトン・ファイルの生成

クライアント・スタブ・ファイルとスケルトン・ファイルを生成するには、アプリケーションで使用されるノーティフィケーション IDL ファイルごとに idl コマンドを実行する必要があります。表 4-2 は、各タイプのサブスクライバで使用する idl コマンドを示しています。

表 4-2 idl コマンドの要件

言語	BEA Tuxedo CORBA 共同クライアント / サーバ	BEA Tuxedo CORBA サーバ
C++	idl -P	idl
Java	idltojava	BEA Tuxedo 8.0 以降ではサポートされていません。

次に、idl コマンドの例を示します。

```
>idl -IC:\tuxdir\include C:\tuxdir\include\CosEventComm.idl
```

表 4-3 は、各タイプのノーティフィケーション・サービス・アプリケーションで必要な IDL ファイルを示しています。

表 4-3 ノーティフィケーション・サービス・アプリケーションで必要な IDL ファイル

アプリケーション・タイプ	必要な OMG IDL ファイル
イベント・ポスト元 (クライアント、共同クライアント / サーバ、またはサーバ)	CosEventChannelAdmin.idl CosEventComm.idl CosNotification.idl CosNotifyChannelAdmin CosNotifyComm.idl CosNotifyFilter Tobj_Events.idl Tobj_Notification.idl

表 4-3 ノーティフィケーション・サービス・アプリケーションで必要な IDL ファイル

アプリケーション・タイプ	必要な OMG IDL ファイル
サブスクライバ (共同クライアント / サーバまたはサーバ)	CosEventChannelAdmin.idl CosEventComm.idl CosNotification.idl CosNotifyChannelAdmin CosNotifyComm.idl CosNotifyFilter Tobj_Events.idl Tobj_Notification.idl

アプリケーション・コードのコンパイルとリンク

コンパイルとリンクの手続きは、ノーティフィケーション・サービス・アプリケーションのタイプによって異なります。表 4-4 は、各タイプのアプリケーションをコンパイルするために使用するコマンドとファイルの概要を示しています。

表 4-4 アプリケーションのビルド要件

アプリケーション・タイプ	クライアント	共同クライアント / サーバ	サーバ
C++ のイベント・ポスト元	buildobjclient コマンドを使用して、アプリケーション・ファイルと IDL スタブをコンパイルします。	-P オプションを設定した buildobjclient コマンドを使用して、アプリケーション・ファイルと IDL スタブをコンパイルします。	buildobjserver コマンドを使用して、アプリケーション・ファイルと IDL クライアント・スタブをコンパイルします。
C++ のイベント・サブスクライバ	適用されません。	-P オプションを設定した buildobjclient コマンドを使用して、アプリケーション・ファイル、IDL スタブ、および IDL スケルトンをコンパイルします。	buildobjserver コマンドを使用して、アプリケーション・ファイル、IDL スタブ、および IDL スケルトンをコンパイルします。

表 4-4 アプリケーションのビルド要件 (続き)

アプリケーション・タイプ	クライアント	共同クライアント / サーバ	サーバ
Java のイベント・ポスト元	javac コマンドを使用して、アプリケーション・ファイルと IDL スタブをコンパイルします。	javac コマンドを使用して、アプリケーション・ファイルと IDL ファイルをコンパイルします。	BEA Tuxedo 8.0 以降ではサポートされていません。
Java のイベント・サブスクライバ	適用されません。	javac コマンドを使用して、アプリケーション・ファイル、IDL ファイル、および IDL スケルトンをコンパイルします。	BEA Tuxedo 8.0 以降ではサポートされていません。

リスト 4-10 は、Microsoft Windows システム上の C++ Reporter アプリケーション (Reporter.cpp) で使用するコマンドを示しています。C++ の実行可能ファイルを作成するために、idl コマンドが必要な IDL ファイルで実行され、buildobjclient コマンドによって C++ クライアント・アプリケーション・ファイルと IDL スタブがコンパイルされます。

リスト 4-10 C++ Reporter アプリケーションのビルドと実行のコマンド

```
# idl コマンドを実行
idl -IC:\tuxdir\include C:\tuxdir\include\CosEventComm.idl \
C:\tuxdir\include\CosEventChannelAdmin \
C:\tuxdir\include\CosNotification.idl \
C:\tuxdir\include\CosNotifyComm.idl \
C:\tuxdir\include\CosNotifyFilter.idl \
C:\tuxdir\include\Tobj_Notification.idl

# buildobjclient コマンドを実行
buildobjclient -v -o is_reporter.exe -f "\
    -DWIN32                                \
    Reporter.cpp                           \
    CosEventComm_c.cpp                     \
    CosEventChannelAdmin_c.cpp             \
    CosNotification_c.cpp                  \
    CosNotifyComm_c.cpp                    \
    CosNotifyFilter_c.cpp                  \
    CosNotifyChannelAdmin_c.cpp            \
    Tobj_Events_c.cpp                      \
    Tobj_Notification_c.cpp "
```

```
# アプリケーションを実行
is_reporter
```

リスト 4-11 と リスト 4-12 は、それぞれ Microsoft Windows および UNIX 上の C++ Subscriber アプリケーション (Subscriber.cpp) で使用するコマンドを示しています。C++ の実行可能ファイルを作成するために、-P オプションを設定した buildobjclient コマンドによって、共同クライアント / サーバ・アプリケーション・ファイル (Subscriber.cpp と NewsConsumer_i.cpp)、IDL スタブ、および IDL スケルトン (CosNotifyComm_s.cpp 用) がコンパイルされます。

リスト 4-11 C++ Subscriber アプリケーションのビルドと実行のコマンド (Microsoft Windows)

```
# idl コマンドを実行
idl -P -IC:\tuxdir\include C:\tuxdir\include\CosEventComm.idl \
C:\tuxdir\include\CosEventChannelAdmin \
C:\tuxdir\include\CosNotification.idl \
C:\tuxdir\include\CosNotifyComm.idl \
C:\tuxdir\include\CosNotifyFilter.idl \
C:\tuxdir\include\CosNotifyChannelAdmin \
\C:\tuxdir\include\Tobj_Events.idl \
\C:\tuxdir\include\Tobj_Notification

# buildobjclient コマンドを実行
buildobjclient -v -P -o is_subscriber.exe  -f " \
   -DWIN32                                     \
    Subscriber.cpp                             \
    NewsConsumer_i.cpp                         \
    CosEventComm_c.cpp                         \
    CosEventChannelAdmin_c.cpp                 \
    CosNotification_c.cpp                      \
    CosNotifyComm_c.cpp                        \
    CosNotifyComm_s.cpp                        \
    CosNotifyFilter_c.cpp                      \
    CosNotifyChannelAdmin_c.cpp                \
    Tobj_Events_c.cpp                          \
    Tobj_Notification_c.cpp                    \
    C:\tuxdir\lib\libbeawrapper.lib            \
    "

# アプリケーションを実行
is_subscriber
```

リスト 4-12 C++ Subscriber アプリケーションのビルドと実行のコマンド (UNIX)

```
# idl コマンドを実行
idl -P -I/usr/local/tuxdir/include
/usr/local/tuxdir/include/CosEventChannelAdmin \
/usr/local/tuxdir/include/CosEventComm.idl \
/usr/local/tuxdir/include/CosNotification.idl \
/usr/local/tuxdir/include/CosNotifyComm.idl \
/usr/local/tuxdir/include/CosNotifyFilter.idl \
/usr/local/tuxdir/include/CosNotifyChannelAdmin \
/usr/local/tuxdir/include/Tobj_Events.idl \
/usr/local/tuxdir/include/Tobj_SimpleEvents.idl

# buildobjclient コマンドを実行
buildobjclient -v -P -o subscriber -f "     \
  Subscriber.cpp                            \
  NewsConsumer_i.cpp                        \
  CosEventComm_c.cpp                        \
  CosEventChannelAdmin_c.cpp                \
  CosNotification_c.cpp                     \
  CosNotifyComm_c.cpp                       \
  CosNotifyComm_s.cpp                       \
  CosNotifyFilter_c.cpp                     \
  CosNotifyChannelAdmin_c.cpp               \
  Tobj_Events_c.cpp                         \
  Tobj_SimpleEvents_c.cpp                   \
  -lbeawrapper                              \
"
# アプリケーションを実行
is_subscriber
```

リスト 4-13 は、リモートの Java Reporter アプリケーションおよび Java Subscriber アプリケーションをリンク、ビルド、および実行するために使用するコマンドの例を示しています。

リスト 4-13 Java Reporter アプリケーションのリンク、ビルド、および実行のコマンド

```
# idltojava コマンドを実行
idltojava -IC:\tuxdir\include C:\tuxdir\include\CosEventComm.idl \
  C:\tuxdir\include\CosEventChannelAdmin.idl \
  C:\tuxdir\include\CosNotification.idl C:\tuxdir\include\CosNotifyComm.idl \
  C:\tuxdir\include\CosNotifyFilter.idl \
  C:\tuxdir\include\CosNotifyChannelAdmin.idl \
```

```
  C:\tuxdir\include\Tobj_Events.idl \
  C:\tuxdir\include\Tobj_Notification.idl

# Java ファイルをコンパイル
javac -classpath C:\tuxdir\udataobj\java\jdk\m3envobj.jar Reporter.java

# Java .class ファイルを Java アーカイブ (JAR) ファイルに結合
jar cf reporter.jar Reporter.class org\omg\CosEventComm \
  org\omg\CosEventChannelAdmin org\omg\CosNotification org\omg\CosNotifyComm \
  org\omg\CosNotifyFilter org\omg\CosNotifyChannelAdmin com\beasys\Tobj_Events \
  com\beasys\Tobj_Notification

# Reporter アプリケーションを実行
java -DTOBJADDR=//BEANIE:2359 -classpath \
  reporter.jar;C:\tuxdir\udataobj\java\jdk\m3envobj.jar Reporter
```

5 Introductory サンプル・アプリケーションのビルド

ここでは、次の内容について説明します。

- 概要
- Introductory サンプル・アプリケーションのビルドと実行

概要

Introductory サンプル・アプリケーションでは、ニュースのレポータが記事をポストし、ニュースのサブスクライバがその記事を消費するニュース閲覧環境をシミュレートします。

Introductory サンプル・アプリケーションは、2 つのインプリメンテーションが用意されています。1 つは BEA シンプル・イベント・アプリケーション・プログラミング・インターフェイス (API) が使用される C++ プログラミング言語のインプリメンテーション、もう 1 つは CosNotification サービス API が使用される Java のインプリメンテーションです。

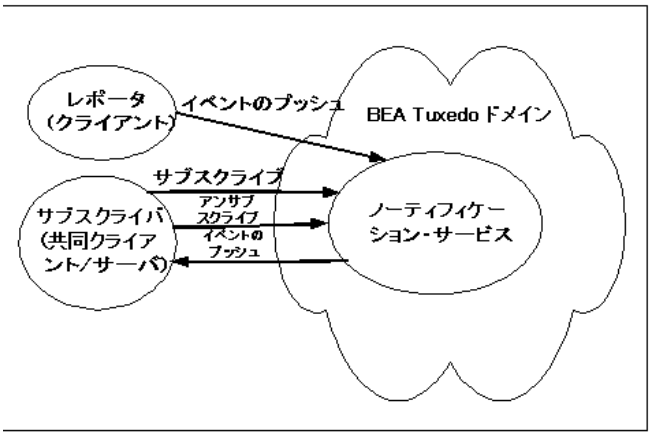
Introductory サンプル・アプリケーションは、Reporter と Subscriber の 2 つのアプリケーションとノーティフィケーション・サービスで構成されます。Reporter アプリケーションは、ユーザにニュース記事の入力を要求し、そのニュース記事を BEA Tuxedo CORBA ノーティフィケーション・サービスにイベントとしてポストするクライアント・アプリケーションをインプリメントします。Subscriber アプリケーションは、イベントをサブスクライブおよびアンサブスクライブするときにはクライアントとして機能し、イベントを受信するときにはサーバとして機能する共同クライアント / サーバ・アプリケーションをインプリメントします。イベントを受信するために、サブスクライバではノーティフィケーション・サービスによってイベント配信が必要なときに呼び出されるコールバック・オブジェクトをインプリメントします。

Introductory サンプル・アプリケーションは、ノーティフィケーション・サービスの最もシンプルな使い方を示します。このサンプル・アプリケーションでは、BEA シンプル・イベント API、CosNotification API、一時的なサブスクリプション、および一時的なオブジェクト・リファレンスの使い方が例示されます。永続的なサブスクリプションまたはデータのフィルタ処理の使い方は示されません。永続的なサブスクリプションおよびデータのフィルタ処理が使用されるサンプル・アプリケーションについては、「第 6 章 Advanced サンプル・アプリケーションのビルド」を参照してください。

この Introductory サンプル・アプリケーションでは、次の 2 つの実行可能ファイルが提供されます (図 5-1 を参照)。

- ノーティフィケーション・サービスにイベントをポストする Reporter アプリケーション。コールバック機能を持たないクライアントです。
- ノーティフィケーション・サービスをサブスクライブし、イベントを受信する Subscriber アプリケーション。サブスクライバは、イベントをサブスクライブするときにはクライアントとして機能し、イベントを受信するときにはサーバとして機能する共同クライアント / サーバです。

図 5-1 Introductory サンプル・アプリケーションの構成要素



イベント・ポスト元 (Reporter アプリケーション) では、構造化イベントの `domain_name`、`type_name`、および `filterable_data` フィールドを使用してイベントを作成します。ドメイン名は業界を定義します。このアプリケーションでは、domain_name は「News」に設定します。`type_name` はその業界でのイベントの種類を定義し、ニュース記事のカテゴリに設定されます (「Sports」など)。この値はアプリケーションのユーザが指定します。`filterable_data` フィール

ドでは、「Story」というフィールドを追加します。このフィールドは、ポストされるニュース記事のテキストを格納します。このテキストもアプリケーションのユーザが指定します。

Subscriber アプリケーションでは、構造化イベントの domain_name フィールドと type_name フィールドを使用してノーティフィケーション・サービスのサブスクリプションを作成します。サブスクリプションでは、domain_name が「News」の内容の固定文字列として定義されます。実行時に、Subscriber アプリケーションではユーザに「ニュース・カテゴリ」を問い合わせ、その入力を使用してサブスクリプションの type_name フィールドを定義します。当然、両アプリケーション (レポータとサブスクライバ) のユーザが協力し、サブスクリプションの「ニュース・カテゴリ」文字列がイベントと一致するようにしなければなりません。そうしないと、サブスクライバにイベントは配信されません。サブスクリプションでは、記事の本文が文字列であり、その記事が構造化イベントの filterable_data フィールドの最初の名前 / 値ペアにあるものと想定され、filterable_data フィールドのチェックが行われることはありません。

イベントをポストするために、Reporter アプリケーションでは push_structured_event メソッドを使用してニュース・イベントをノーティフィケーション・サービスにプッシュします。各イベントについて、Reporter アプリケーションでは「ニュース・カテゴリ」(「Sports」など) と記事 (複数行のテキスト文字列) をユーザに問い合わせます。

Subscriber アプリケーションでは、ノーティフィケーション・サービスを呼び出してニュース・イベントをサブスクライブします。各サブスクリプションについて、Subscriber アプリケーションでは「ニュース・カテゴリ」(「Sports」など) をユーザに問い合わせます。Subscriber アプリケーションでは、ニュース・イベントを受信して処理するためのコールバック・オブジェクトも (NewsConsumer_i サーバント・クラスを通じて) インプリメントします。Subscriber がサブスクライブするときには、このコールバック・オブジェクトのオブジェクト・リファレンスがノーティフィケーション・サービスに渡されます。一致するイベントが発生すると、つまりサブスクリプションとニュース・カテゴリが一致するイベントが Reporter によってポストされると、ノーティフィケーション・サービスではコールバック・オブジェクトの push_structured_event メソッドを呼び出してサブスクライバのコールバック・オブジェクトにイベントを配信します。このメソッドでは、イベントが出力され、ノーティフィケーション・サービスの unsubscribe メソッドを呼び出してサブスクリプションが取り消され、Subscriber がシャットダウンされます。簡

略化のため、push_structured_event メソッドでは domain_name、type_name、length、および name フィールドが一致し、記事が value フィールドにあるものと想定されます。

注記 「ニュース・カテゴリ」は、Reporter のユーザと Subscriber のユーザが合意した文字列です。このサンプルには固定のカテゴリはありません。したがって、Reporter のユーザと Subscriber のユーザが両者とも、カテゴリが要求されたときに (大文字と小文字の区別およびスペースの有無を含めて) 同じ文字列を入力しなければなりません。

このサンプルを実行するには、Reporter アプリケーションと Subscriber アプリケーションを少なくとも 1 つずつ起動する必要があります (それぞれ複数を実行することも可能)。Reporter によってポストされたイベントは、(「ニュース・カテゴリ」に基づいて) 一致するすべてのサブスクライバに配信されます。

また、サブスクライバは必ずイベントをポストする前に起動してください。そうしないと、イベントが失われます。

Introductory サンプル・アプリケーションのビルドと実行

Introductory サンプル・アプリケーションをビルドして実行するには、次の手順を行う必要があります。

1. "TUXDIR" 環境変数と "JAVA_HOME" 環境変数が適切なディレクトリ・パスに設定されていることを確認します。

   注記 "JAVA_HOME" 環境変数は、Java アプリケーションの場合のみ必要となります。

2. Introductory サンプル・アプリケーションのファイルを作業ディレクトリにコピーします。
3. ファイルの保護属性を変更して、書き込みと実行を可能にします。
4. UNIX の場合は、make ファイルがパスに含まれているようにします。Microsoft Windows の場合は、nmake ファイルがパスに含まれているようにします。

5. アプリケーションの環境変数を設定します。
6. サンプルをビルドします。
7. システムを起動します。
8. Subscriber アプリケーションと Reporter アプリケーションを実行します。
9. システムをシャットダウンします。
10. ディレクトリを元の状態に戻します。

これらの手順は、次の節でさらに詳しく説明されます。

環境変数の設定を確認する

Introductory サンプル・アプリケーションをビルドして実行する前に、システムで TUXDIR 環境変数が設定されていることを確認する必要があります。ほとんどの場合、この環境変数はインストールの過程で設定されます。しかし、正確な情報が反映されているかどうかを確認することが必要です。

表 5-1 は、Introductory サンプル・アプリケーションを実行するために必要な環境変数のリストです。

表 5-1 Introductory サンプル・アプリケーションで必須の環境変数

環境変数	説明
TUXDIR	BEA Tuxedo ソフトウェアをインストールしたディレクトリのパスです。次に例を示します。 **Windows** TUXDIR=c:\tuxdir **UNIX** TUXDIR=/usr/local/tuxdir
JAVA_HOME (Java アプリケーションのみ)	JDK ソフトウェアをインストールしたディレクトリのパスです。次に例を示します。 **Windows** JAVA_HOME=c:\JDK1.2.2 **UNIX** JAVA_HOME=/usr/local/JDK1.2.1

インストール時に定義された環境変数の情報が正しいかどうかを確認するには、次の手順を行います。

Windows

1. [スタート] メニューの [設定] をクリックします。
2. [設定] メニューの [コントロール パネル] をクリックします。

   コントロール・パネルが表示されます。
3. [システム] アイコンをクリックします。

   [システムのプロパティ] ウィンドウが表示されます。
4. [詳細] タブをクリックします。

   [詳細] ページが表示されます。
5. TUXDIR および JAVA_HOME の設定を確認します。

UNIX

```
ksh prompt>printenv TUXDIR
ksh prompt>printenv JAVA_HOME
```

設定を変更するには、次の手順を行います。

Windows

1. [システムのプロパティ] ウィンドウの [詳細] ページで、設定を変更する環境変数をクリックします。
2. [値] フィールドに環境変数の正しい情報を入力します。
3. [OK] をクリックして変更内容を保存します。

UNIX

```
ksh prompt>export TUXDIR=directorypath
ksh prompt>export JAVA_HOME=directorypath
```

または

```
csh> setenv TUXDIR=directorypath
csh> setenv JAVA_HOME=directorypath
```

Introductory サンプル・アプリケーションのファイルを作業ディレクトリにコピーする

Introductory サンプル・アプリケーションのファイルと common ディレクトリのファイルをローカル・マシンの作業ディレクトリにコピーする必要があります。

注記 アプリケーション・ディレクトリと common ディレクトリは同じ親ディレクトリにコピーしなければなりません。

ファイルは、次のディレクトリに配置されています。

Windows

C++ の Introductory サンプル
*drive:**tuxdir*\samples\corba\notification\introductory_simple_cxx
*drive:**tuxdir*\samples\corba\notification\common

Java の Introductory サンプル
*drive:**tuxdir*\samples\corba\notification\introductory_cos_java
*drive:**tuxdir*\samples\corba\notification\common

UNIX

C++ の Introductory サンプル
/usr/local/tuxdir/samples/corba/notification/
introductory_simple_cxx
/usr/local/tuxdir/samples/corba/notification/common

Java の Introductory サンプル
/usr/local/tuxdir/samples/corba/notification/
introductory_simple_cxx
/usr/local/tuxdir/samples/corba/notification/common

表 5-2 と 表 5-4 のファイルは、BEA シンプル・イベント API を使用してインプリメントされる C++ Introductory サンプル・アプリケーションをビルドして実行するために使用します。表 5-3 と 表 5-4 のファイルは、CosNotification API を使用してインプリメントされる Java Introductory サンプル・アプリケーションをビルドして実行するために使用します。

表 5-2 introductory_sample_c++ ディレクトリのファイル

ファイル	説明
Readme.txt	Introductory サンプル・アプリケーションの説明と、環境の設定およびアプリケーションのビルドと実行の手順が記述されています。
setenv.cmd	Microsoft Windows システムの環境を設定します。
setenv.ksh	UNIX システムの環境を設定します。
makefile.nt	Microsoft Windows システムの makefile。
makefile.mk	UNIX システムの makefile。
makefile.inc	makefile.nt ファイルと makefile.mk ファイルで使用される共通の makefile。
Reporter.cpp	レポータのコード。
Subscriber.cpp	サブスクライバのコード。
NewsConsumer_i.h および NewsConsumer.cpp	ニュース・イベントを受信するためにサブスクライバで使用されるコールバック・サーバント・クラス (Subscriber アプリケーション用)。

表 5-3 introductory_cos_java ディレクトリのファイル

ファイル	説明
Readme.txt	Introductory サンプル・アプリケーションの説明と、環境の設定およびアプリケーションのビルドと実行の手順が記述されています。
setenv.cmd	Microsoft Windows システムの環境を設定します。
setenv.ksh	UNIX システムの環境を設定します。
makefile.nt	Microsoft Windows システムの makefile。
makefile.mk	UNIX システムの makefile。

表 5-3 introductory_cos_java ディレクトリのファイル

ファイル	説明
makefile.inc	makefile.nt ファイルと makefile.mk ファイルで使用される共通の makefile。
Reporter.java	レポータのコード。
Suscriber.java	サブスクライバのコード。
NewsConsumer_i.java	ニュース・イベントを受信するためにサブスクライバで使用されるコールバック・サーバント・クラス (Subscriber アプリケーション用)。

表 5-4 は、Introductory サンプル・アプリケーションで使用されるほかのファイルのリストです。

表 5-4 Introductory サンプル・アプリケーションで使用されるほかのファイル

ファイル	説明
次のファイルは、common ディレクトリに配置されています。	
common.nt	Microsoft Windows システムの makefile シンボル。
common.mk	UNIX システムの makefile シンボル。
introductory.inc	管理対象の makefile。
ex.h	例外を出力するユーティリティ (C++ のみ)。
client_ex.h	例外を処理するクライアント・ユーティリティ (C++ のみ)。
ShutdownManager.java	ノーティフィケーション・サービス Java サンプルのメインとサーバントがサーバのシャットダウンを調整するのを支援するクラス。 **注記** このファイルは Java アプリケーションでのみ必要です。

表 5-4 Introductory サンプル・アプリケーションで使用されるほかのファイル (

ファイル	説明
次のファイルは、\tuxdir\include ディレクトリに配置されています。	
CosEventComm.idl	CosEventComm モジュールを宣言する OMG IDL コード。
CosNotification.idl	CosNotification モジュールを宣言する OMG IDL コード。
CosNotifyComm.idl	CosNotifyComm モジュールを宣言する OMG IDL コード。
Tobj_Events.idl	Tobj_Events モジュールを宣言する OMG IDL コード。
Tobj_SimpleEvents.idl	Tobj_SimpleEvents モジュールを宣言する OMG IDL コード。 **注記** このファイルは、BEA シンプル・イベント API を使用して開発されたアプリケーションでのみ必要です。
次のファイルは、CosNotification サービス API を使用して開発されたアプリケーションでのみ必要です。	
CosEventChannelAdmin.idl	CosEventChannelAdmin モジュールを宣言する OMG IDL コード。
CosNotifyFilter.idl	CosNotifyFilter モジュールを宣言する OMG IDL コード。
CosNotifyChannelAdmin.idl	CosNotifyChannelAdmin モジュールを宣言する OMG IDL コード。
Tobj_Notification.idl	Tobj_Notification モジュールを宣言する OMG IDL コード。

Introductory サンプル・アプリケーションのファイルの保護属性を変更する

BEA Tuxedo CORBA ソフトウェアのインストール時に、サンプル・アプリケーションのファイルは読み取り専用に設定されます。Introductory サンプル・アプリケーションのファイルを編集またはビルドするには、作業ディレクトリにコピーしたファイルの保護属性を次のように変更する必要があります。

Windows

1. DOS ウィンドウで、ディレクトリを作業ディレクトリに変更します (cd)。
2. `prompt>attrib -r drive:\workdirectory\*.*`

UNIX

1. ディレクトリを作業ディレクトリに変更します (cd)。
2. `prompt>/bin/ksh`
3. `ksh prompt>chmod u+w /workdirectory/*.*`

UNIX システムでは、次のように `setenv.ksh` のパーミッションを変更して実行可能にすることも必要です。

```
ksh prompt>chmod +x setenv.ksh
```

環境を設定する

環境を設定するには、次のコマンドを入力します。

Windows

```
prompt>.\setenv.com
```

UNIX

```
ksh prompt>. ./setenv.ksh
```

Introductory サンプル・アプリケーションをビルドする

makefile を実行するには、make コマンドを使用します。makefile は、Microsoft Windows および UNIX の両方でサンプル・アプリケーションをビルドするために使用します。UNIX では、make を使用します。Microsoft Windows では、nmake を使用します。

makefile の概要

makefile では、次の手順が自動的に行われます。

1. 環境設定コマンド (setenv.cmd) が実行済みであることを確認します。環境変数が設定されていない場合は、エラー・メッセージを画面に出力して終了します。
2. common.nt コマンド・ファイル (Microsoft Windows) または common.mk コマンド・ファイル (UNIX) をインクルードします。このファイルでは、サンプルで使用される makefile シンボルが定義されます。それらのシンボルにより、UNIX および Microsoft Windows の makefile でビルド規則をプラットフォームに依存しない makefile に委譲することができます。
3. makefile.inc コマンド・ファイルをインクルードします。このファイルでは、is_reporter および is_subscriber の実行可能ファイルがビルドされ、不要なファイルとディレクトリがクリーン・アップされます。
4. introductory.inc コマンド・ファイルをインクルードします。このファイルでは、UBBCONFIG ファイルが作成され、tmloadcf -y ubb コマンドを実行して TUXCONFIG ファイルが作成されます。これはプラットフォームに依存しない makefile の一部であり、Introductory サンプル・アプリケーションで共通の管理上のビルド規則を定義します。

makefile を実行する

makefile を実行する前に、次の確認作業を行う必要があります。

- アプリケーションをビルドおよび実行するための適切な管理者権限があることを確認します。

- Microsoft Windows では、nmake がマシンのパスに含まれていることを確認します。
- UNIX では、make がマシンのパスに含まれていることを確認します。

Introductory サンプル・アプリケーションをビルドするには、make コマンドを次のように入力します。

Windows

```
nmake -f makefile.nt
```

UNIX

```
make -f makefile.mk
```

Introductory サンプル・アプリケーションを起動する

Introductory サンプル・アプリケーションを起動するには、次のコマンドを入力します。

1. BEA Tuxedo システムを起動するには、次のように入力します。

   ```
   prompt>tmboot -y
   ```

   このコマンドでは、次のサーバ・プロセスが開始されます。

   - TMSUSREVT

     ノーティフィケーション・サービスで使用される、BEA Tuxedo システムに付属の EventBroker サーバ

   - TMNTS

     サブスクリプションおよびイベントのポストの要求を処理する BEA Tuxedo ノーティフィケーション・サービス・サーバ

   - TMNTSFWD_T

     一時的なサブスクリプションのあるサブスクライバにイベントを転送する BEA Tuxedo ノーティフィケーション・サービス・サーバ

   - ISL

     IIOP リスナ / ハンドラ・プロセス

2. Subscriber アプリケーションを起動するには、次のように入力します。

   C++ の場合 : `prompt>is_subscriber`
   Microsoft Windows 上の Java の場合 : `prompt>java %IC_SUBSCRIBER%`
   UNIX 上の Java の場合 : `prompt>java $IC_SUBSCRIBER`

   Subscriber をもう 1 つ起動するには、別のウィンドウを開き、ディレクトリを作業ディレクトリに変更して (`cd`)、環境変数を設定し (`setenv.cmd` または `setenv.ksh` を実行)、プラットフォームに適した起動コマンドを入力します。

3. Reporter アプリケーションを起動するには、新たにウィンドウを開いて次のように入力します。

   C++ の場合 : `prompt>is_reporter`
   Microsoft Windows 上の Java の場合 : `prompt>java %IC_REPORTER%`
   UNIX 上の Java の場合 : `prompt>java $IC_REPORTER`

   Reporter をもう 1 つ起動するには、別のウィンドウを開き、ディレクトリを作業ディレクトリに変更して (`cd`)、環境変数を設定し (`setenv.cmd` または `setenv.ksh` を実行)、プラットフォームに適した起動コマンドを入力します。

Introductory サンプル・アプリケーションの使い方

Introductory サンプル・アプリケーションを使用するには、Subscriber アプリケーションを使用してイベントをサブスクライブし、Reporter アプリケーションを使用してイベントをポストする必要があります。サブスクライブは、各イベントをポストする前に行う必要があります。そうしないと、イベントは失われます。

注記 Subscriber アプリケーションは、イベントを 1 つ受信した後にシャットダウンされます。

Subscriber アプリケーションを使用したイベントのサブスクライブ

次の手順を行います。

1. Subscriber アプリケーションを起動すると (`prompt>is_subscriber`)、次のプロンプトが表示されます。

   ```
   Name?                  ( スペースなしで名前を入力する )
   Category (or all)?  ( ニュース・カテゴリまたは「all」を入力する )
   ```

   ニュースのカテゴリとしては任意の文字列を入力できます。つまり、ニュース・カテゴリの決まったリストは存在しません。ただし、Reporter アプリケーションを使用してイベントをポストするときには、必ず、ニュース・カテゴリに同じ文字列を使用してください。

2. Subscriber アプリケーションは、サブスクリプションを作成し、イベント受信の準備ができた時点で「Ready」を出力します。イベントを 1 つ受信すると、Subscriber はシャットダウンされます。

   注記 Reporter アプリケーションを使用してイベントをポストする前に、必ず Subscriber アプリケーションを使用してイベントをサブスクライブする必要があります。そうしないと、イベントは失われます。

Reporter アプリケーションを使用したイベントのポスト

次の手順を行います。

1. Reporter アプリケーションを起動すると (`prompt> is_reporter`)、次のプロンプトが表示されます。

   ```
   (r) Report news
   (e) Exit

   Option?
   ```

2. ニュースを報告するには `r` を入力します。次のプロンプトが表示されます。

   ```
   Category?
   ```

3. ニュース・カテゴリを入力します。ニュース・カテゴリは、Subscriber アプリケーションで入力したカテゴリと (スペースの有無や大文字と小文字の区別を含めて) まったく同じに入力する必要があります。

ニュース・カテゴリを入力すると、次のプロンプトが表示されます。

```
Enter story (terminate with '.')
```

4. 記事を入力します。複数行にまたがってもかまいません。記事を終わらせるには、1 行にピリオド (「.」) のみを入力して改行します。

   この記事とカテゴリが一致するサブスクライバが記事を受信して出力します。記事を受信すると、サブスクライバは自動的にシャットダウンされます。

5. 続けてニュース記事を送信および受信するには、新たにサブスクライバを起動して、また別の記事を報告します。ニュースの報告が終了したら、終了 (e) オプションを選択します。

   注記 Subscriber アプリケーションは、イベントを 1 つ受信した後にシャットダウンされます。したがって、Reporter アプリケーションを使用してイベントをポストする前に、必ず Subscriber アプリケーションを使用してイベントをサブスクライブする必要があります。そうしないと、イベントは失われます。

システムのシャットダウンとディレクトリのクリーン・アップ

次の手順を行います。

注記 Reporter プロセスと Subscriber プロセスが停止していることを確認してください。

1. システムをシャットダウンするには、任意のウィンドウで次のように入力します。

   ```
   prompt>tmshutdown -y
   ```

2. ディレクトリを元の状態に復元するには、任意のウィンドウで次のように入力します。

   Windows

   ```
   prompt>nmake -f makefile.nt clean
   ```

   UNIX

   ```
   prompt>make -f makefile.mk clean
   ```

6 Advanced サンプル・アプリケーションのビルド

ここでは、次の内容について説明します。

- 概要
- Advanced サンプル・アプリケーションのビルドと実行

概要

Advanced サンプル・アプリケーションでは、ニュース・レポータが記事をポストし、ワイヤ・サービスがその記事をイベントとしてノーティフィケーション・サービスにポストし、ニュース・サブスクライバが記事を消費するニュース閲覧環境をシミュレートします。

Advanced サンプル・アプリケーションは、2 つのインプリメンテーションが用意されています。1 つは BEA シンプル・イベント・アプリケーション・プログラミング・インターフェイス (API) が使用される Java プログラミング言語のインプリメンテーション、もう 1 つは CosNotification サービス API が使用される C++ のインプリメンテーションです。

Advanced サンプル・アプリケーションは、BEA Tuxedo CORBA ノーティフィケーション・サービスを使用するレポータ・アプリケーション、サブスクライバ・アプリケーション、およびワイヤ・サービス・アプリケーションで構成されます。レポータ・アプリケーションは、クライアント・アプリケーションをインプリメントします。このアプリケーションは、ニュース記事の入力をユーザに促し、アプリケーション固有の IDL を使用して WireService サーバを呼び出します。WireService サーバは、イベントをポストします。サブスクライバは、共同クライアント / サーバ・アプリケーションをインプリメントします。このアプリケーションは、イベントをサブスクライブおよびアンサブスクライブするときにはクライアントとして機能し、イベントを受信するときにはサーバとして機能し

ます。イベントを受信するために、サブスクライバはノーティフィケーション・サービスによってイベント配信が必要なときに呼び出されるコールバック・オブジェクトをインプリメントします。

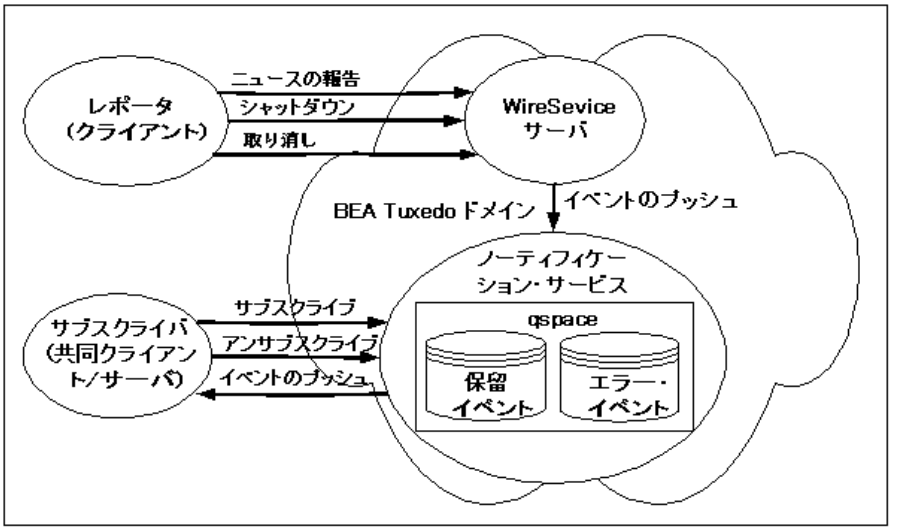
注記 UNIX システムでは、ポートが再び利用可能になるまでに時間がかかるので (実際の時間はプラットフォームによって異なる)、サブスクライバを即座に再起動することはできません。あまりにも早く再起動すると、`CORBA::OBJ_ADAPTER` 例外が発生します。この例外が発生した場合は、少し待ってから再試行してください。Solaris システムでは、ポートが利用可能になるまで最高で 10 分かかります。ポートがまだ使用中かどうかを確認するには、"`Restart -a | grep <the port number>`" というコマンドを使用します。

この Advanced サンプル・アプリケーションでは、BEA シンプル・イベント API、CosNotification サービス API、一時的および永続的なサブスクリプション、およびデータのフィルタ処理の使い方を例示します。

この Advanced サンプル・アプリケーションでは、次の 3 つの実行可能ファイルを使用します (図 6-1 を参照)。

- イベントをポストする WireService アプリケーション。WireService アプリケーションは、ノーティフィケーション・サービスのクライアントであり、BEA Tuxedo CORBA サーバでもあります。このアプリケーションは、Reporter アプリケーションで使用される OMG IDL インターフェイスをインプリメントします。
- WireService のメソッドを呼び出してニュース記事を報告する Reporter アプリケーション。WireService ではその記事をイベントに変換し、ノーティフィケーション・サービスを使用してイベントをポストします。レポータは純粋なクライアントです。
- ノーティフィケーション・サービスをサブスクライブし、イベントを受信する Subscriber アプリケーション。サブスクライバは、イベントをサブスクライブするときにはクライアントとして機能し、イベントを受信するときにはサーバとして機能する共同クライアント / サーバです。

図 6-1 Advanced サンプル・アプリケーションの構成要素



イベント・ポスト元 (WireService アプリケーション) では、構造化イベントの domain_name、type_name、および filterable_data フィールドを使用して 3 つのイベント (ニュース・イベント、サブスクライバ・シャットダウン・イベント、およびサブスクライバ取り消しイベント) を作成します。

- ニュース・イベント

  このイベントでは、ドメイン名は文字列であり、アプリケーションによって事前に「News」として設定されます。型名は文字列であり、Reporter アプリケーションのユーザによって実行時に定義されます。型名は、ニュースのカテゴリ (「Sports」など) に設定されます。フィルタ処理可能データには、名前 / 値ペアが格納されます。その名前は「Story」で、値はポストされるニュース記事の本文である文字列です。

- サブスクライバ・シャットダウン・イベント

  このイベントでは、ドメイン名は文字列であり、アプリケーションによって事前に「NewsAdmin」として設定されます。型名は文字列であり、アプリケーションによって事前に「Shutdown」として設定されます。フィルタ処理可能データは使用しません。

- サブスクライバ取り消しイベント

  このイベントでは、ドメイン名は文字列であり、アプリケーションによって事前に「NewsAdmin」として設定されます。型名は文字列であり、

アプリケーションによって事前に「Cancel」として設定されます。フィルタ処理可能データは使用しません。

Subscriber アプリケーションでは、構造化イベントの domain_name、type_name、および filterable_data フィールドを使用して 2 つのサブスクリプションを作成します。1 つはニュース記事を処理するニュース・サブスクリプション、もう 1 つは Cancel イベントと Shutdown イベントを処理するシャットダウン・サブスクリプションです。実行時に、Subscriber アプリケーションではノーティフィケーション・サービスに対するそれら 2 つのサブスクリプションを確立します。

- ニュース・サブスクリプション

  Subscriber アプリケーションでは、構造化イベントの domain_name、type_name、および filterable_data フィールドを使用してノーティフィケーション・サービスのサブスクリプションを作成します。サブスクリプションでは、ドメイン名が「News」の内容の固定文字列として定義されます。実行時に、Subscriber アプリケーションではユーザに「ニュース・カテゴリ」と「キーワード」を問い合わせ、その入力を使用してサブスクリプションの type_name フィールドと data_filter フィールドを定義します。当然、両アプリケーション (リポーターとサブスクライバ) のユーザが協力し、サブスクリプションの「ニュース・カテゴリ」文字列と「キーワード」文字列がイベントと一致するようにしなければなりません。そうしないと、サブスクライバにニュース・イベントは配信されません。サブスクリプションでは、記事の本文が文字列であり、その記事が構造化イベントの filterable_data フィールドの最初の名前 / 値ペアにあるものと想定され、filterable_data フィールドのチェックが行われることはありません。

- シャットダウン・サブスクリプション

  Subscriber アプリケーションでは、構造化イベントの domain_name フィールドと type_name フィールドを使用してノーティフィケーション・サービスのサブスクリプションを作成します。サブスクリプションでは domain_name が「NewsAdmin」の内容の固定文字列として定義され、type_name が「Shutdown」または「Cancel」のいずれかの文字列として定義されます。filterable_data フィールドは空の文字列です。

Reporter アプリケーションは、Shutdown イベントと Cancel イベントを生成するだけでなく、ニュースを報告するためのユーザ・インターフェイスをインプリメントします。ノーティフィケーション・サービスを直に使用してイベントをポストするのではなく、WireService サーバのメソッドを呼び出します。

WireService サーバでは、ノーティフィケーション・サービスを使用して次の 3 種類のイベントをポストします。

- 「News」イベント (ニュースをサブスクライバに配信するために使用)
- 「Shutdown」イベント (サブスクライバを一時的にシャットダウンするために使用)
- 「Cancel」イベント (サブスクライバを永続的にシャットダウンするために使用)

ノーティフィケーション・サービスでは、これらのイベントをサブスクライバに配信します。

サブスクライバでは、ノーティフィケーション・サービスを使用してニュース・イベントの永続的なサブスクリプションを作成します。サブスクライバは、ニュース・イベントを受信して処理するための永続的なコールバック・オブジェクトを (`NewsConsumer_i` サーバント・クラスを通じて) インプリメントします。サブスクライバがサブスクライブするときには、このコールバック・オブジェクトのオブジェクト・リファレンスがノーティフィケーション・サービスに渡されます。一致するイベントが発生すると、ノーティフィケーション・サービスではこのコールバック・オブジェクトの `push_structured_event` メソッドを呼び出してイベントをサブスクライバにプッシュします。このメソッドはイベントを出力します。

サブスクライバでは、ノーティフィケーション・サービスを使用して Shutdown イベントと Cancel イベントの一時的なサブスクリプションも作成します。サブスクライバは、それらのイベントを受信して処理するためのコールバック・オブジェクトも (`ShutdownConsumer_i` サーバント・クラスを通じて) インプリメントします。

サブスクライバが実行されると、そのサブスクライバはユーザに名前の入力を要求します。このユーザが初めてサブスクライバ・プログラムを実行すると、サブスクライバでは News イベントの永続的なサブスクリプションが作成されます。そのために、サブスクライバはサブスクライブするニュース記事の種類とサブスクライバが動作するポート番号の入力をユーザに要求します。サブスクライバはこのポートで動作し、サブスクライブして、サブスクリプション ID、フィルタ ID (CosNotification API 使用の場合)、およびポート番号をファイルに書き込みます (ファイルの名前は *<user_name>*`.pstore`)。サブスクライバは、次回の実行時には、ユーザに名前の入力を要求し、ファイル *<user_name>*`.pstore` を開いて、このユーザのサブスクリプション ID、フィルタ ID (CosNotification API 使用の場合)、およびポート番号をファイルから読み取ります。これで、ニュー

ス・コールバック・オブジェクトのオブジェクト・リファレンスが永続的であるため、実行のたびに同じポート番号を使用しなければならない、という要件が満たされます。

Subscriber では一時的なサブスクリプションを作成して Shutdown イベントと Cancel イベントを受信するので、一時的なサブスクリプションはサブスクライバが実行およびシャットダウンされるたびに作成および破棄されます。このサブスクリプション ID は、ファイル *<user_name>*.pstore に書き出されません。

サブスクライバが Shutdown イベントを受信すると、シャットダウン / コールバック・サブスクリプションが破棄されますが、News サブスクリプションはそのまま維持されます。サブスクライバがシャットダウンされた後で、それが再起動される前に News イベントがポストされた場合、そのイベントはサブスクライバが再起動された時点で配信されるか、またはエラー・キューに入れられます。ntsadmin ユーティリティを使用すると、エラー・キューからイベントを削除するか、配信を再試行できます。

イベントが再配信されるか、エラー・キューに入れられるかは、サブスクライバがすぐに再起動されるかどうかによって決まります。すぐに再起動されるかどうかは、キューの再試行パラメータによって決まります。キューの再試行パラメータの値については、advanced.inc (ノーティフィケーション・サンプルの common ディレクトリ) を参照してください。

News イベントには、カテゴリ (見出しなど) と記事 (複数行のテキスト文字列) という 2 つのパートがあります。Subscriber アプリケーションは、ニュース・カテゴリの入力をユーザに求めます。サブスクライバでは、カテゴリがその文字列と一致するニュース・イベントをサブスクライブします。Reporter アプリケーションは、ニュース・カテゴリと記事の入力をユーザに要求します。レポータでは、ワイヤ・サービスのメソッドを呼び出して対応するニュース・イベントをポストします。イベントは、そのカテゴリのニュースをサブスクライブしたサブスクライバだけに配信されます。

注記 カテゴリは文字列です。Reporter のユーザと Subscriber のユーザは同じ文字列を使用しなければなりません。このサンプルには固定のカテゴリはありません。したがって、Reporter のユーザと Subscriber のユーザは両者とも、カテゴリが要求されたときに (大文字と小文字の区別およびスペースの有無を含めて) 同じ文字列を入力しなければなりません。

このサンプルではデータのフィルタ処理も利用します。ユーザが初めて Subscriber を実行すると、そのユーザは「キーワード」の入力を要求されます。カテゴリが一致し、記事にキーワードが含まれているイベントがサブスクライバに配信されます。たとえば、ユーザが「none」というキーワードを入力した場

合、データのフィルタ処理は行われません。その場合、ユーザは選択したニュース・カテゴリのすべてのイベントを受信することになります。「smith」というキーワードを入力した場合、そのキーワードは "Story %% '.*smith.*'" に変換されます。この場合、文字列の含まれる「Story」というフィールドが存在し、その文字列が任意数の文字で始まり、リテラル文字列「smith」が含まれ、任意数の文字で終わっているイベントのみがサブスクリプションで受け入れられます。

このサンプルを実行するには、Reporter と Subscriber を少なくとも 1 つずつ実行する必要があります (それぞれ複数を実行することも可能)。Reporter によってポストされたイベントは、(カテゴリに基づいて) 一致するすべてのサブスクライバに配信されます。

また、サブスクライバは必ずイベントをポストする前に起動してください。サブスクライバが起動する前にポストされたイベントは配信されません。

Advanced サンプル・アプリケーションのビルドと実行

Introductory サンプル・アプリケーションをビルドして実行するには、次の手順を行う必要があります。

1. "TUXDIR" 環境変数と "JAVA_HOME" 環境変数が適切なディレクトリ・パスに設定されていることを確認します。

   注記 "JAVA_HOME" 環境変数は、Java アプリケーションの場合のみ必要となります。

2. Introductory サンプル・アプリケーションのファイルを作業ディレクトリにコピーします。
3. ファイルの保護属性を変更して、書き込みと実行を可能にします。
4. UNIX の場合は、make ファイルがパスに含まれているようにします。Microsoft Windows の場合は、nmake ファイルがパスに含まれているようにします。
5. アプリケーションの環境変数を設定します。
6. サンプルをビルドします。

7. システムを起動します。
8. Subscriber アプリケーションと Reporter アプリケーションを実行します。
9. システムをシャットダウンします。
10. ディレクトリを元の状態に戻します。

これらの手順は、次の節でさらに詳しく説明されます。

環境変数の設定を確認する

Advanced サンプル・アプリケーションをビルドして実行する前に、システムで TUXDIR 環境変数が設定されていることを確認する必要があります。ほとんどの場合、この環境変数はインストールの過程で設定されます。しかし、正確な情報が反映されているかどうかを確認することが必要です。

表 6-1 は、Callback サンプル・アプリケーションを実行するために必要な環境変数のリストです。

表 6-1 Callback サンプル・アプリケーションで必須の環境変数

環境変数	説明
TUXDIR	BEA Tuxedo ソフトウェアをインストールしたディレクトリのパスです。次に例を示します。 **Windows** TUXDIR=c:\tuxdir **UNIX** TUXDIR=/usr/local/tuxdir
JAVA_HOME (Java アプリケーションのみ)	JDK ソフトウェアをインストールしたディレクトリのパスです。次に例を示します。 **Windows** JAVA_HOME=c:\JDK1.2 **UNIX** JAVA_HOME=/usr/local/JDK1.2

インストール時に定義された環境変数の情報が正しいかどうかを確認するには、次の手順を行います。

Windows

1. [スタート] メニューの [設定] をクリックします。
2. [設定] メニューの [コントロール パネル] をクリックします。

   コントロール・パネルが表示されます。
3. [システム] アイコンをクリックします。

   [システムのプロパティ] ウィンドウが表示されます。
4. [詳細] タブをクリックします。

   [詳細] ページが表示されます。
5. TUXDIR and JAVA_HOME の設定を確認します。

UNIX

```
ksh prompt>printenv TUXDIR
ksh prompt>printenv JAVA_HOME
```

設定を変更するには、次の手順を行います。

Windows

1. [システムのプロパティ] ウィンドウの [詳細] ページで、設定を変更する環境変数をクリックします。
2. [値] フィールドに環境変数の正しい情報を入力します。
3. [OK] をクリックして変更内容を保存します。

UNIX

```
ksh prompt>export TUXDIR=directorypath
ksh prompt>export JAVA_HOME=directorypath
```

Advanced サンプル・アプリケーションのファイルを作業ディレクトリにコピーする

Advanced サンプル・アプリケーションのファイルをローカル・マシンの作業ディレクトリにコピーする必要があります。

注記 アプリケーション・ディレクトリと common ディレクトリは同じ親ディレクトリにコピーしなければなりません。

Advanced サンプル・アプリケーションのファイルは、次のディレクトリに配置されています。

Windows

C++ の Advanced サンプル
*drive:**tuxdir*\samples\corba\notification\advanced_cos_cxx
*drive:**tuxdir*\samples\corba\notification\common

Java の Advanced サンプル
*drive:**tuxdir*\samples\corba\notification\advanced_simple_java
*drive:**tuxdir*\samples\corba\notification\common

UNIX

C++ の Advanced サンプル
/usr/local/tuxdir/samples/corba/notification/advanced_cos_cxx
/usr/local/tuxdir/samples/corba/notification/common

Java の Advanced サンプル
/usr/local/tuxdir/samples/corba/notification/advanced_simple_java
/usr/local/tuxdir/samples/corba/notification/common

表 6-2 と 表 6-4 のファイルは、BEA シンプル・イベント API を使用してインプリメントされる Java Advanced サンプル・アプリケーションをビルドして実行するために使用します。表 6-3 と 表 6-4 のファイルは、CosNotification API を使用してインプリメントされる C++ Advanced サンプル・アプリケーションをビルドして実行するために使用します。

表 6-2 advanced_simple_java ノーティフィケーション・ディレクトリに配置されているファイル

ファイル	説明
Readme.txt	Advanced サンプル・アプリケーションの説明と、環境の設定およびアプリケーションのビルドと実行の手順が記述されています。
setenv.cmd	Microsoft Windows システムの環境を設定します。
setenv.ksh	UNIX システムの環境を設定します。
makefile.nt	Microsoft Windows システムの makefile。
makefile.mk	UNIX システムの makefile。
makefile.inc	makefile.nt ファイルと makefile.mk ファイルで使用される共通の makefile。
Reporter.java	レポータのコード。
Subscriber.java	サブスクライバのコード。
NewsConsumer_i.java	ニュース・イベントを受信するためにサブスクライバで使用されるコールバック・サーバント・クラス (Subscriber アプリケーション用)。
ShutdownConsumer_i.java	Shutdown イベントと Cancel イベントを受信するためにサブスクライバで使用されるコールバック・サーバント・クラス (Subscriber アプリケーション用)。
WireService.xml	WireService サーバのサーバ記述ファイル。
WireService_i.java	WireService インターフェイスをインプリメントします。
WireServiceFactory_i.java	WireService ファクトリ・インターフェイスをインプリメントします。
WireServiceServer.java	WireService サーバのコード。

表 6-3 と 表 6-4 のファイルは、Advanced サンプル・アプリケーションをビルドして実行するために使用します。

表 6-3 advanced_cos_c++ ノーティフィケーション・ディレクトリに配置されているファイル

ファイル	説明
Readme.txt	Advanced サンプル・アプリケーションの説明と、環境の設定およびアプリケーションのビルドと実行の手順が記述されています。
setenv.cmd	Microsoft Windows システムの環境を設定します。
setenv.ksh	UNIX システムの環境を設定します。
makefile.nt	Microsoft Windows システムの makefile。
makefile.mk	UNIX システムの makefile。
makefile.inc	makefile.nt ファイルと makefile.mk ファイルで使用される共通の makefile。
Reporter.cpp	レポータのコード。
Subscriber.cpp	サブスクライバのコード。
NewsConsumer_i.h および NewsConsumer.cpp	ニュース・イベントを受信するためにサブスクライバで使用されるコールバック・サーバント・クラス (Subscriber アプリケーション用)。
ShutdownConsumer_i.h および ShutdownConsumer.cpp	Shutdown イベントと Cancel イベントを受信するためにサブスクライバで使用されるコールバック・サーバント・クラス (Subscriber アプリケーション用)。
WireServiceServer.cpp	WireService サーバのコード。
News.icf	WireService インターフェイスの ICF ファイル。
WireService_i.h および WireService.cpp	WireService インターフェイスをインプリメントします。

表 6-4 は、Advanced サンプル・アプリケーションで使用されるほかのファイルのリストです。IDL ファイル以外のファイルは、ノーティフィケーション common ディレクトリに配置されています。

表 6-4 Advanced サンプルで使用されるほかのファイル

ファイル	説明
次のファイルは、common ディレクトリに配置されています。	
News.idl	WireService サーバの IDL 定義
news_flds	データのフィルタ処理を実行するために使用する FML フィールド定義とニュース・イベント
common.nt	Microsoft Windows システムの makefile シンボル
common.mk	UNIX システムの makefile シンボル
advanced.inc	管理対象の makefile
ex.h	例外を出力するユーティリティ (C++ のみ)
client_ex.h	例外を処理するクライアント・ユーティリティ (C++ のみ)
server_ex.h	例外を処理するサーバ・ユーティリティ
次のファイルは、\tuxdir\include ディレクトリに配置されています。	
CosEventComm.idl	CosEventComm モジュールを宣言する OMG IDL コード
CosNotification.idl	CosNotification モジュールを宣言する OMG IDL コード
CosNotifyComm.idl	CosNotifyComm モジュールを宣言する OMG IDL コード
Tobj_Events.idl	Tobj_Events モジュールを宣言する OMG IDL コード

表 6-4 Advanced サンプルで使用されるほかのファイル (続き)

ファイル	説明
Tobj_SimpleEvents.idl	Tobj_SimpleEvents モジュールを宣言する OMG IDL コード **注記** このファイルは、BEA シンプル・イベント API を使用して開発されたアプリケーションでのみ必要です。
次のファイルは、CosNotification サービス API を使用して開発されたアプリケーションでのみ必要です。	
CosEventChannelAdmin.idl	CosEventChannelAdmin モジュールを宣言する OMG IDL コード
CosNotifyFilter.idl	CosNotifyFilter モジュールを宣言する OMG IDL コード
CosNotifyChannelAdmin.idl	CosNotifyChannelAdmin モジュールを宣言する OMG IDL コード
Tobj_Notification.idl	Tobj_Notification モジュールを宣言する OMG IDL コード

Advanced サンプル・アプリケーションのファイルの保護属性を変更する

BEA Tuxedo ソフトウェアのインストール時に、Advanced サンプル・アプリケーションのファイルは読み取り専用に設定されます。Advanced サンプル・アプリケーションのファイルを編集またはビルドするには、作業ディレクトリにコピーしたファイルの保護属性を次のように変更する必要があります。

Windows

1. ディレクトリを作業ディレクトリに変更します (cd)。
2. `prompt>attrib -r drive:\workdirectory\*.*`

UNIX

1. ディレクトリを作業ディレクトリに変更します (cd)。

```
2. prompt>/bin/ksh
3. ksh prompt>chmod u+w /workdirectory/*.*
```

UNIX オペレーティング・システム・プラットフォームでは、次のように setenv.ksh のパーミッションを変更して実行可能にすることも必要です。

```
ksh prompt>chmod +x setenv.ksh
```

環境を設定する

環境を設定するには、次のコマンドを入力します。

Windows

```
prompt>.\setenv.com
```

UNIX

```
prompt>. ./setenv.ksh
```

Advanced サンプル・アプリケーションをビルドする

makefile を実行するには、make コマンドを使用します。makefile は、Microsoft Windows および UNIX の両方でサンプル・アプリケーションをビルドするために使用します。Microsoft Windows では、nmake を使用します。UNIX では、make を使用します。

makefile の概要

makefile では、次の手順が自動的に行われます。

1. 環境設定コマンド (setenv.cmd) が実行済みであることを確認します。環境変数が設定されていない場合は、エラー・メッセージを画面に出力して終了します。

2. common.nt コマンド・ファイル (Microsoft Windows) または common.mk コマンド・ファイル (UNIX) をインクルードします。このファイルでは、サンプルで使用される makefile シンボルが定義されます。それらのシンボルにより、UNIX および Microsoft Windows の makefile でビルド規則をプラットフォームに依存しない makefile に委譲することができます。

3. makefile.inc コマンド・ファイルをインクルードします。このファイルでは、is_reporter、is_subscriber、および AS_WIRESERVICE の実行可能ファイルがビルドされ、不要なファイルとディレクトリがクリーン・アップされます。

4. advanced.inc コマンド・ファイルをインクルードします。このファイルでは、tmadmin コマンドと qadmin コマンドを実行してトランザクション・ログおよび永続的なサブスクリプションで必要なキューが作成されます。さらに、UBBCONFIG ファイルが作成され、tmloadcf -y ubb コマンドを実行して TUXCONFIG ファイルが作成されます。

makefile を実行する

makefile を実行する前に、次の確認作業を行う必要があります。

- アプリケーションをビルドおよび実行するための適切な管理者権限があることを確認します。
- Microsoft Windows では、nmake がマシンのパスに含まれていることを確認します。
- UNIX では、make がマシンのパスに含まれていることを確認します。

Advanced サンプル・アプリケーションをビルドするには、make コマンドを次のように入力します。

Windows

```
nmake -f makefile.nt
```

UNIX

```
make -f makefile.mk
```

Advanced サンプル・アプリケーションを起動する

Advanced サンプル・アプリケーションを起動するには、次のコマンドを入力します。

1. BEA Tuxedo システムを起動するには、次のように入力します。

   ```
   prompt>tmboot -y
   ```

   このコマンドでは、次のサーバ・プロセスが開始されます。

   - TMSUSREVT

     ノーティフィケーション・サービスで使用される、BEA Tuxedo システムに付属の EventBroker サーバ

   - TMNTS

     サブスクリプションおよびイベントのポストの要求を処理する BEA Tuxedo CORBA ノーティフィケーション・サービス・サーバ

   - TMNTSFWD_T

     一時的なサブスクリプションのあるサブスクライバにイベントを転送する BEA Tuxedo CORBA ノーティフィケーション・サービス・サーバ。このサーバは、一時的なサブスクリプションで必須です。

   - TMNTSFWD_P

     永続的なサブスクリプションのあるサブスクライバに永続的なイベントを転送する BEA Tuxedo CORBA ノーティフィケーション・サービス・サーバ。このサーバは、永続的なサブスクリプションで必須です。

   - TMQUEUE

     メッセージ・キュー・マネージャは、tpenqueue(3) および tpdequeue(3) を呼び出してプログラムの代わりにキューのメッセージの登録と取り出しを行う BEA Tuxedo システム付属のサーバです。このサーバは、永続的なサブスクリプションで必須です。

   - TMQFORWARD

     メッセージ転送サーバは、後で処理するように tpenqueue(3c) を使用して格納されたメッセージを転送する BEA Tuxedo システム付属のサーバです。このサーバは、永続的なサブスクリプションで必須です。

- WIRE_SERVICE_SERVER

  Reporter アプリケーションからイベントを受信し、そのイベントをノーティフィケーション・サービスにポストする、Advanced サンプル・アプリケーション用に特別にビルドされたサーバ。News、Shutdown、および Cancel という 3 タイプのイベントが受信およびポストされます。

- ISL

  IIOP リスナ / ハンドラ・プロセス

2. Subscriber アプリケーションを起動するには、次のように入力します。

   C++ の場合 : prompt>is_subscriber
   Microsoft Windows 上の Java の場合 : prompt>java %IC_SUBSCRIBER%
   UNIX 上の Java の場合 : prompt>java $IC_SUBSCRIBER

   Subscriber をもう 1 つ起動するには、別のウィンドウを開き、ディレクトリを作業ディレクトリに変更して (cd)、環境変数を設定し (setenv.cmd または setenv.ksh を実行)、プラットフォームに適した起動コマンドを入力します。

3. Reporter アプリケーションを起動するには、新たにウィンドウを開いて次のように入力します。

   C++ の場合 : prompt>is_reporter
   Microsoft Windows 上の Java の場合 : prompt>java %IC_REPORTER%
   UNIX 上の Java の場合 : prompt>java $IC_REPORTER

   Reporter をもう 1 つ起動するには、別のウィンドウを開き、ディレクトリを作業ディレクトリに変更して (cd)、環境変数を設定し (setenv.cmd または setenv.ksh を実行)、プラットフォームに適した起動コマンドを入力します。

Advanced サンプル・アプリケーションの使い方

Advanced サンプル・アプリケーションを使用するには、Subscriber アプリケーションを使用してイベントをサブスクライブし、Reporter アプリケーションを使用してイベントをポストする必要があります。サブスクライブは、各イベントをポストする前に行う必要があります。そうしないと、イベントは失われます。

Subscriber アプリケーションを使用したイベントのサブスクライブ

次の手順を行います。

1. Subscriber アプリケーションを初めて起動すると (prompt>is_subscriber)、次のプロンプトが表示されます。

   ```
   Name?                ( スペースなしで名前を入力する )
   Port (e.g. 2463)     (このサブスクライバが動作するポートの番号を入力する )
   Category (or all)    ( ニュース・カテゴリまたは「all」を入力する )
   Keyword (or none)    (配信するすべての記事に含まれている必要のあるキーワードを入力する )
   ```

   注記 Subscriber アプリケーションが Reporter アプリケーションからの Shutdown イベントによってシャットダウンされた場合 (Shutdown イベントでは永続的なサブスクリプションは取り消されない)、Subscriber アプリケーションの以降の起動では、名前以外は入力を要求されません。Subscriber アプリケーションは、名前以外の情報を <user_name>.pstore ファイルから取り出します。このようにすることで、同じポート番号が必ず使用されるようになります (永続的なサブスクリプションでは不可欠)。

   Subscriber アプリケーションが Reporter アプリケーションからの Cancel イベントによってシャットダウンされた場合 (Cancel イベントでは永続的なサブスクリプションを含むすべてのサブスクリプションが取り消される)、Subscriber アプリケーションの以降の起動では、名前、ポート番号、カテゴリ、およびキーワードの入力が要求されます。

2. ニュースのカテゴリとしては任意の文字列を入力できます。つまり、ニュース・カテゴリの決まったリストは存在しません。ただし、Reporter アプリケーションを使用してイベントをポストするときには、必ず、ニュース・カテゴリに同じ文字列を使用してください。

   同様に、キーワードでも文字列を入力できます。キーワードも決まったリストは存在しません。したがって、レポータを実行して記事を入力するときには、その記事に同じ文字列が含まれているようにしてください。そうしないと、記事はサブスクリプションに配信されません。

ユーザ名、カテゴリ (または「all」)、およびキーワード (オプション) の入力に基づいて初めて Subscriber アプリケーションが実行されたときには、ニュースのサブスクリプションが作成されます。次以降の実行時には、サブスクライバではこのサブスクリプションが再利用されます。Subscriber アプリケーションは常に、イベントを受信する準備ができた時点で「Ready」を出力します。

Subscriber アプリケーションは、サブスクリプションを作成し、イベント受信の準備ができた時点で「Ready」を出力します。

注記 Reporter アプリケーションを使用してイベントをポストする前に、必ず Subscriber アプリケーションを使用してイベントをサブスクライブする必要があります。そうしないと、イベントは失われます。Subscriber アプリケーションで News イベントの永続的なサブスクリプションが作成される場合でも、そのサブスクリプションは Subscriber アプリケーションが起動するまで作成されません。

注記 別のウィンドウを開いてこの手順を繰り返せば、複数のサブスクライバを起動できます。

Reporter アプリケーションを使用したイベントのポスト

次の手順を行います。

1. Reporter アプリケーションを起動すると (`prompt> is_reporter`)、次のプロンプトが表示されます。

```
(r) Report news
(s) Shutdown subscribers
(c) Cancel Subscribers
(e) Exit

Option?
```

2. ニュースを報告するには `r` を入力します。次のプロンプトが表示されます。

```
Category?
```

3. ニュース・カテゴリを入力します。ニュース・カテゴリは、Subscriber アプリケーションで入力したカテゴリと (スペースの有無や大文字と小文字の区別を含めて) まったく同じに入力する必要があります。

   ニュース・カテゴリを入力すると、次のプロンプトが表示されます。

```
Enter story (terminate with '.')
```

4. 記事を入力します。複数行にまたがってもかまいません。記事を終わらせるには、1 行にピリオド (「.」) のみを入力して改行します。サブスクライブするときにキーワードを入力した場合は、(スペースの有無や大文字と小文字の区別を含めて) まったく同じ文字列が記事に含まれているようにしてください。

   カテゴリとキーワード (指定された場合) がこの記事と一致するサブスクライバが記事を受信して出力します。

5. 「s」オプションを選択すると、Shutdown イベントがポストされ、そのイベントがすべてのサブスクライバによって受信されて、それらのサブスクライバがシャットダウンされます。サブスクライバがシャットダウンされていても、再び「r」オプションを使用すれば別のニュース記事をポストできます。ノーティフィケーション・サービスはそのニュース記事を保留キューに入れますが、News イベントのサブスクリプションは永続的であるため、依然として有効です。サブスクライバを再起動すると、サブスクライバはこの 2 番目のニュース記事を受信します (再起動の遅延によってイベントがエラー・キューに移動された場合を除く)。なぜなら、ニュース記事の永続的なサブスクリプションがサブスクライバによって作成されているからです。

   注記 `ntsadmin retryerrevents` コマンドを使用すると、エラー・キューのイベントを保留キューに移動することができます。

6. 「c」オプションを選択すると、Cancel イベントがポストされ、そのイベントがすべてのサブスクライバで受信されます。サブスクライバは、ニュースのサブスクリプションを取り消し、シャットダウンされます。サブスクライバを再起動するときには、新しいサブスクリプションを作成することになるので、ポート、カテゴリ、およびキーワードを再び入力する必要があります。

7. ニュースの報告が終了したら、終了 (e) オプションを選択します。

   注記 別のウィンドウを開いてこの手順を繰り返せば、複数のレポータを起動できます。レポータによって報告されたニュース記事はすべて、一致するすべてのサブスクライバに配信されます。システムをシャットダウンするときには、その前に必ずすべてのレポータを終了してください。

システムのシャットダウンとディレクトリのクリーン・アップ

Reporter と Subscriber のプロセスが停止しているのを確認してから、次の手順を行ってください。

1. システムをシャットダウンするには、任意のウィンドウで次のように入力します。

   ```
   prompt>tmshutdown -y
   ```

2. ディレクトリを元の状態に復元するには、任意のウィンドウで次のように入力します。

   Windows

   ```
   prompt>nmake -f makefile.nt clean
   ```

   UNIX

   ```
   prompt>make -f makefile.mk clean
   ```

7　CORBA ノーティフィケーション・サービスの管理

ここでは、次の内容について説明します。

- はじめに
- ノーティフィケーション・サービスのコンフィギュレーション . この節では、以下の内容について説明します。
  - データ・フィルタのコンフィギュレーション
  - ホストとポートの設定
  - トランザクション・ログの作成
  - イベント・キューの作成
  - UBBCONFIG ファイルと TUXCONFIG ファイルの作成
- ノーティフィケーション・サービスの管理
- ノーティフィケーション・サービスの管理ユーティリティとコマンド
- ノーティフィケーション・サーバ

はじめに

BEA Tuxedo CORBA ノーティフィケーション・サービスは、BEA Tuxedo の EventBroker システムおよびキューイング・システムに重ね合わせて使用します。つまり、CORBA ノーティフィケーション・サービスを管理する場合は、それらのほかの BEA Tuxedo システムも管理する必要があります。ノーティフィケーション・サービスの管理には、BEA Tuxedo ユーティリティの `tmadmin`、`qmadmin`、および `ntsadmin` を使用します。

ノーティフィケーション・サービスの管理は、コンフィギュレーションと管理という 2 つの関連するタスクで成り立ちます。それらの領域は別々に説明しますが、実際には相互に関連しています。したがって、コンフィギュレーションを完全に理解するためには、管理についても理解する必要があります (その逆も同じ)。

ノーティフィケーション・サービスのコンフィギュレーション

イベント・ノーティフィケーション・サービス・アプリケーションを実行するには、まず次のコンフィギュレーション要件を満たさなければなりません。

- データのフィルタ処理または BEA Tuxedo ATMI 相互運用性を利用する場合は、フィルタ処理または相互運用を行うフィールドを記述する BEA Tuxedo ATMI FML フィールド定義ファイルを作成します。
- 永続的なサブスクリプションを使用する場合は、次のタスクを行います。
  - 共同クライアント / サーバを使用する場合は、コールバック・オブジェクトのオブジェクト・リファレンスのホストとポート番号を設定します。
  - トランザクション・ログを作成します。
  - イベントを保持するキューを作成します。
- システム・コンフィギュレーション・ファイル (`UBBCONFIG`) および `TUXCONFIG` ファイルを作成します。

データ・フィルタのコンフィギュレーション

データのフィルタ処理または BEA Tuxedo ATMI 相互運用性をサブスクライバ・アプリケーションで利用する場合は、次の手順を行ってサブスクリプションでデータのフィルタ処理を利用する必要があります。

1. フィルタ処理を行うフィールドを記述する BEA Tuxedo ATMI FML フィールド・テーブル定義ファイルを作成します (リスト 7-2 を参照)。
2. アプリケーションの起動時にフィールド定義ファイルの位置をノーティフィケーション・サービス・サーバに渡せるように、UBBCONFIG ファイルで FML フィールド・テーブル定義ファイルの配置場所を指定します (リスト 7-3 を参照)。

リスト 7-1 の中で、**太字**のコードはイベント・ポスト元アプリケーションでデータのフィルタ処理がどのようにインプリメントされるのかを示します。名前 / 値ペア billing および patient_account の含まれるサブスクリプションだけがイベントを受信します。

リスト 7-1 BEA シンプル・イベント API を使用したデータ・フィルタ処理のサンプル (C++)

```
CosNotification::StructuredEvent notif;

notif.header.fixed_header.event_type.domain_name =
       CORBA::string_dup("HEALTHCARE");

notif.header.fixed_header.event_type.type_name =
       CORBA::string_dup("HMO");

// このイベントの名前と値に基づいて追加フィルタを
// 指定する

notif.filterable_data.length(2);
notif.filterable_data[0].name = CORBA::string_dup("billing");
notif.filterable_data[0].value <<= CORBA::Long(1999);
notif.filterable_data[1].name =
          CORBA::string_dup("patient_account");
notif.filterable_data[1].value <<= CORBA::Long(2345);

// 構造化イベントをチャネルにプッシュ
testChannel->push_structured_event(notif);
```

リスト 7-2 は、データのフィルタ処理を利用するために必要な FML フィールド・テーブル定義ファイルを示しています。

リスト 7-2 データ・フィルタ処理の FML フィールド・テーブル・ファイル

```
*base 2000
```

```
#Field Name      Field #  Field Type    Flags     Comments
#-----------     -------  ----------    ------    --------
billing          1        long          -         -
patient_account  2        long          -         -
```

リスト 7-3 は、環境変数ファイル (envfile) の内容を示しています .envfile には、FML フィールド定義ファイルの位置を格納します。

注記 環境変数ファイルにはどのような名前でも使用できますが、その名前は ENVFILE コンフィギュレーション・オプション n (UBBCONFIG ファイルの SERVERS セクション) で指定された名前と同じでなければなりません。

リスト 7-3 データ・フィルタ処理のための envfile の指定 (Microsoft Windows)

```
FLDTBLDIR32=D:\tuxdir\EVENTS_Samples\ADVANCED_Simple_cxx\common
FIELDTBLS32=news_flds
```

リスト 7-4 は、Advanced サンプルの UBBCONFIG ファイルで FML フィールド・テーブル・ファイルの位置がどのように指定されるのかを示しています (**太字**部分)。

リスト 7-4 UBBCONFIG ファイルでの FML フィールド定義ファイルの指定

```
*SERVERS
TMSYSEVT
  SRVGRP  = NTS_GRP
  SRVID   = 1
TMUSREVT
  SRVGRP  = NTS_GRP>>$@
  SRVID   = 2
  ENVFILE = "D:\tuxdir\EVENTS_Samples\ADVANCED_Simple_CXX\envfile"
TMNTS
  SRVGRP = NTS_GRP
  SRVID = 3
  ENVFILE = "D:\tuxdir\EVENTS_Samples\ADVANCED_Simple_CXX\envfile"
  CLOPT = "-A -- -s TMNTSQS"
TMNTSFWD_T
  SRVGRP = NTS_GRP
  SRVID = 4
  ENVFILE = "D:\tuxdir\EVENTS_Samples\ADVANCED_Simple_CXX\envfile"
```

```
TMNTSFWD_P
  SRVGRP = NTS_GRP
  SRVID = 5
  ENVFILE = "D:\tuxdir\EVENTS_Samples\ADVANCED_Simple_CXX\envfile"
```

ホストとポートの設定

コールバック・オブジェクトに関するオブジェクト・リファレンスのホストとポート番号の要件は次のとおりです。

- 一時的なコールバック・オブジェクトの場合は、どのポートでも問題はなく、ORB で動的に取得できます。
- 永続的なコールバック・オブジェクトの場合は、コールバック・オブジェクトのオブジェクト・リファレンスが作成された同じポートでコールバック・オブジェクトの要求を受け付けるように ORB をコンフィギュレーションする必要があります。

ポート番号は、動的な範囲ではなくユーザの範囲で指定します。ユーザの範囲でポート番号を割り当てれば、共同クライアント / サーバ・アプリケーションが使用するポートの衝突を防ぐことができます。

ホストとポートを設定する方法は、プログラミング言語によって異なります。

- C++ サブスクライバ・アプリケーションでのホストとポートの設定

  C++ サブスクライバ・アプリケーションの場合、共同クライアント / サーバ・アプリケーションで使用する特定のポートを指定するには、共同クライアント / サーバ・アプリケーションのプロセスを開始するコマンド行で次のように指定します。

  ```
  -ORBport nnnn  -IRBid BEA_IIOP
  ```

  nnnn は、共同クライアント / サーバ・アプリケーションのコールバック・オブジェクトの呼び出しを作成するとき、および呼び出しをリッスンするときに ORB で使用されるポートの番号です。

  このコマンドは、共同クライアント / サーバ・アプリケーションのコールバック・オブジェクトのオブジェクト・リファレンスが永続的である必要があるとき、および共同クライアント / サーバ・アプリケーションを停止して再起動する必要があるときに使用します。このコマンドを使用しない場合、

ORB ではランダムなポートが使用されます。共同クライアント / サーバ・アプリケーションが停止して再起動されるときにランダムなポートが使用される場合、共同クライアント / サーバ・アプリケーションの永続的なコールバック・オブジェクトの呼び出しは失敗します。

ポート番号は、CORBA::orb_init メンバ関数の argv 引数に対する入力の一部です。argv 引数が渡されると、ORB はその情報を読み取って、そのプロセスで作成されたオブジェクト・リファレンスのポートを確立します。

- Java サブスクライバ・アプリケーションでのホストとポートの設定

  Java サブスクライバ・アプリケーションの場合は、ホストとポートを設定するプロパティを渡すことができます。リスト 7-5 は、その方法を示しています。

リスト 7-5 Java サブスクライバ・アプリケーションでのホストとポートの設定

```
Properties prop = new Properties();
prop.put( "org.omg.CORBA.ORBClass", "com.beasys.CORBA.iiop.ORB" );
prop.put("org.omg.CORBA.ORBSingletonClass",
                  "Com.beasys.CORBA.idl.ORBSingleton");
prop.put( "org.omg.CORBA.ORBPort",  nnnn );
ORB orb = ORB.init(args, prop);
```

注記 java コマンド行でポートを設定することもできます。ポート番号を設定する java コマンド行の例を次に示します。

```
java -DTOBJADDR=//BEANIE:2359 \
-Dorg.omg.corba.ORBPort=portnumber -classpath...
```

トランザクション・ログの作成

永続的なサブスクリプションを使用する場合は、BEA Tuxedo キューイング・システムをコンフィギュレーションおよび起動する必要があります。キューイング・システムでは、トランザクション・ログが必要です。リスト 7-6 は、tmadmin ユーティリティを使用してトランザクション・ログを作成する方法を示しています。

リスト 7-6　トランザクション・ログの作成 (createtlog) (Microsoft Windows)

```
>tmadmin

>crdl -b 100 -z D:\tuxdir\EVENTS_Samples\ADVANCED_Simple_CXX\TLOG
>crlog -m SITE1
>quit
>
```

イベント・キューの作成

永続的なイベントを使用する場合は、BEA Tuxedo キューイング・システムをコンフィギュレーションおよび起動する必要があります。次の 2 つのイベント・キューを作成しなければなりません。

- TMNTSFWD_P

  これは、永続的なサブスクリプション用のイベント転送キューです。イベントはまずこのキューに入り、その後に一致する永続的なサブスクリプションに転送されます。イベントを最初の試行で配信できない場合、そのイベントはこのキューに保持されて、配信が繰り返し試行されます。イベントを正常に配信できないうちに設定可能な再試行回数の上限に達した場合、そのイベントはエラー・キューに移動されます。

  このキューでは、次のコンフィギュレーション・パラメータを設定する必要があります。

  - キューイングの順序 (先入れ先出し、など)
  - 順序を無視したキュー登録の処理方法
  - 再試行回数の上限 (イベントがエラー・キューに移動されるまでに再試行が行われる回数)
  - 再試行の時間間隔
  - キューがどのくらい一杯になったら管理上の介入が必要になるのか
  - 一杯になった後にキューのイベント数がどのくらい少なくなったら管理上の介入が必要になるのか
  - 管理上の介入のコマンドの定義

- TMNTSFWD_E

  これはエラー・キューです。このキューは、サブスクリプションに配信できないイベントを TMNTSFWD_P キューから受信します。このキューでは、TMNTSFWD_P 転送キューと同じコンフィギュレーション・パラメータを設定する必要があります。ただし、このキューはエラー・キューであり、エラーは管理上の介入によってのみ削除されるので再試行回数の上限と再試行の時間間隔のパラメータは意味を成しません。

これらのキューをコンフィギュレーションするには、次の手順を行います。

1. キュースペースのデバイスをディスク上に作成します。
2. キュースペースをコンフィギュレーションします。
3. キューを作成します。

この手順については、以降の節で説明します。

一時的および永続的なサブスクリプションのスペース・パラメータの値の割り出し

システムを調整して最大限の性能を引き出すためには、次のパラメータの最適な値を割り出す必要があります。

- 一時的な転送サーバ (TMNTSFWD_T) と永続的な転送サーバ (TMNTSFWD_P) の数
- IPC キュースペース (一時的なサブスクリプションで使用)
- /Q キューのサイズ (永続的なサブスクリプションで使用)

一時的なサブスクリプションの IPC キュースペース

一時的なサブスクリプションのスペース・パラメータの値は次のように割り出します。

1. 一時的なサブスクリプションのパイプラインを通る可能性のあるイベントの数、つまり特定の時点で配信されている可能性のあるイベントの数を割り出します。この数は、イベントの数とそれらのイベントを受信するサブスクライバの数を乗算すると算出できます。

2. イベントのサイズを割り出します。説明の便宜上、イベントは比較的小さく約 300 バイト以下と想定します。
3. 起動する必要のある一時的な転送サーバの数を割り出します (たいていは 1 つか 2 つ)。マシン上のプロセッサごとに 1 つから始めるのが理想的です。
4. 一時的なイベントを保持するのに必要な IPC キュースペースの量を割り出します。必要なスペースの量は、1000 バイトをパイプラインで許容するイベント数で乗算した値です。この値は、一時的な転送サーバの IPC キューの数で除算します。MSSQ セットを使用する場合は、1 つの IPC キューが一時的な転送サーバで共有されます。使用しない場合は、各転送サーバで独自の IPC キューが使用されます。

   たとえば、10 のイベントが 50 のサブスクライバにパイプラインで配信されると仮定し、2 つの一時的な転送サーバを起動して、それらのサーバで IPC キューが共有されないとすると (つまり MSSQ は不使用)、IPC キュースペースの必要量は次のようになります。

   10 イベント * 50 サブスクライバ * 1000 バイト / 2 転送サーバ = 250,000 バイト
5. システム・レジストリのエントリを変更して、IPC キューのサイズをこの値にコンフィギュレーションします。その方法はプラットフォームによって異なります。
   - Microsoft Windows システムの場合は、第 7 章の 14 ページ「Microsoft Windows での IPC パラメータの設定」を参照してください。
   - UNIX システムの場合は、システムに付属のリファレンス・マニュアルを参照してください。

永続的なサブスクリプションの /Q キュー・サイズ・パラメータ

永続的なサブスクリプションのスペース・パラメータの値は次のように割り出します。

1. 永続的なサブスクリプションのパイプラインを通る可能性のあるイベントの数、つまり特定の時点で配信されている可能性のあるイベントの数を割り出します。この数は、イベントの数とそれらのイベントを受信するサブスクライバの数を乗算すると算出できます。

2. イベントのサイズを割り出します。説明の便宜上、イベントは比較的小さく約 300 バイト以下と想定します。
3. 永続的なイベントを保持するのに必要な /Q キューのサイズを割り出します (保留キューとエラー・キューの両方)。次のように割り出します。
   a. ディスク・ページのサイズを割り出します。それはプラットフォームによって異なります。たとえば Microsoft Windows の場合、ディスク・ページは 500 バイトです。UNIX マシンの場合、ディスク・ページのサイズは 500 ～ 4000 バイトです。
   b. 1 つのイベントを格納するのに必要なディスク・ページの数を割り出します。たとえば、イベントごとに 1000 バイトが必要で、ディスク・ページのサイズが 500 バイトである場合は、イベントごとに 2 つのディスク・ページが必要になります。
   c. イベント全体で必要なディスク・ページの数を割り出します。たとえば、500 の保留イベントと 200 のエラー・イベントを許容し、1 つのイベントが 2 つのディスク・ページを占める場合は、1400 のディスク・ページが必要になります。
   d. qspace で必要なディスク・ページの数を割り出します。これは、イベントに必要なディスク・ページと qspace のオーバーヘッドのためのページの合計数です。たとえば、イベントで 1400 のディスク・ページが必要な場合、qspace では約 1450 のディスク・ページが必要です (50 ページはオーバーヘッド用)。
   e. qspace デバイスで必要なページ数を割り出します。これは、qspace に必要なディスク・ページとデバイスのオーバーヘッドのためのページの合計数です。たとえば、qspace で 1450 のディスク・ページが必要な場合、デバイスでは約 1500 のディスク・ページが必要です (50 ページはデバイスのオーバーヘッド用)。
4. `qmadmin` を使用して永続的なイベントの qspace を作成する場合は、まず最初にデバイスを作成します。サイズは、ステップ 3e で算出された値を使用します (約 1500 ページ)。次に、qspace のサイズを指定します。サイズは、ステップ 3d で算出された値を使用します (約 1450 ページ)。次に、保留キューおよびエラー・キューに存在できるイベントの数を指定します。以降の節では、qspace を作成およびコンフィギュレーションする方法を説明します。

ディスク上のキュースペースのデバイスの作成

qmadmin コマンド・ユーティリティを使用すると、ディスク上にキュースペースのデバイスを作成できます。

キュースペースを作成するには、まず汎用デバイス・リスト (UDL) でそのエントリを作成する必要があります。リスト 7-7 は、コマンドの例を示しています。

リスト 7-7　ディスク上のキュースペースのデバイスの作成 (UNIX)

```
prompt>qmadmin d:\smith\reg\QUE
qmadmin - Copyright (c) 1996-1999 BEA Systems, Inc.
Portions * Copyright 1986-1997 RSA Data Security, Inc.
All Rights Reserved.
Distributed under license by BEA Systems, Inc.
BEA Tuxedo is a registered trademark.
QMCONFIG=d:\smith\reg\QUE

> crdl d:\smith\reg\QUE 0 1100
Created device d:\smith\reg\QUE, offset 0, size 1100
 on d:\smith\reg\QUE
```

ディスク上でデバイスを作成する方法の詳細については、『BEA Tuxedo/Q コンポーネント』を参照してください。

キュースペースのコンフィギュレーション

キュースペースをコンフィギュレーションするには、qmdamin qspacecreate コマンドを使用します。キュースペースでは、IPC 資源が使用されます。そのため、キュースペースを定義する場合は、共用メモリ・セグメントとセマフォを割り当てることになります。qspacecreate コマンドを使用する最も簡単な方法は、プロンプトを表示することです。リスト 7-8 は、Advanced サンプル・アプリケーション用にコンフィギュレーションされたキュースペースの例を示しています。

リスト 7-8　キュースペースの作成

```
> qspacecreate
Queue space name: TMNTSQS
IPC Key for queue space: 52359
Size of queue space in disk pages: 1050
Number of queues in queue space: 2
Number of concurrent transactions in queue space: 10
Number of concurrent processes in queue space: 10
Number of messages in queue space: 500
Error queue name: TMNTSFWD_E
Initialize extents (y, n [default=n]): y
Blocking factor [default=16]:
```

リスト 7-8 で作成されるキュースペースでは、次のサイズ設定に注意してください。

Number of messages in queue space:500
このパラメータを 500 に設定すると、保留キューとエラー・キューで合計 500 のイベントを格納するスペースが確保されます。

Size of queue space in disk pages:1050
Microsoft Windows では、各ディスク・スペースのサイズが 500 バイトで、各イベントは 1000 バイトが必要です。その上、イベントごとに 2 つのディスク・ページを供給する必要があります。保留キューとエラー・キューに 500 のイベントが格納されると見積もっているので、1000 のディスク・ページを供給しなければなりません。また、qspace のオーバーヘッドのために 50 のディスク・ページが必要なので、qspace のサイズは 1050 ディスク・ページに設定します。最後に、デバイスのオーバーヘッドでも 50 のディスク・ページが必要なので、デバイスのサイズは 1100 ディスク・ページになります。この値は、`crdl` コマンドを使用して設定します (リスト 7-7 を参照)。

キュースペース作成の詳細については、『BEA Tuxedo/Q コンポーネント』を参照してください。

キューの作成

使用する各キューを作成するには、qmadmin qcreate コマンドを使用します。キューを作成するには、まず qmadmin qopen コマンドでキュースペースを開く必要があります。キュースペース名が指定されていない場合は、qopen で名前を入力するように求められます。

リスト 7-9 は、Advanced サンプル・アプリケーション用に作成される TMNTSFWD_P キューと TMNTSFWD_E キューの作成例を示しています。

リスト 7-9 キューの作成

```
> qopen
Queue space name: TMNTSQS

> qcreate
Queue name: TMNTSFWD_P
Queue order (priority, time, fifo, lifo): fifo
Out-of-ordering enqueuing (top, msgid, [default=none]): none
Retries [default=0]: 5
Retry delay in seconds [default=0]: 3
High limit for queue capacity warning (b for bytes used, B for
 blocks used, % for percent used, m for messages [default=100%]):
 80%
Reset (low) limit for queue capacity warning [default=0%]: 0%
Queue capacity command:
No default queue capacity command
Queue 'TMNTSFWD_P' created

> qcreate
Queue name: TMNTSFWD_E
Queue order (priority, time, fifo, lifo): fifo
Out-of-ordering enqueuing (top, msgid, [default=none]): none
Retries [default=0]: 2
Retry delay in seconds [default=0]: 30
High limit for queue capacity warning (b for bytes used, B for
 blocks used, % for percent used, m for messages [default=100%]):
 80%
Reset (low) limit for queue capacity warning [default=0%]: 0%
Queue capacity command:
No default queue capacity command
Q_CAT:1438: INFO: Create queue - error queue TMNTSFWD_E created
Queue 'TMNTSFWD_E' created

> q
```

キュー作成の詳細については、『BEA Tuxedo/Q コンポーネント』を参照してください。

Microsoft Windows での IPC パラメータの設定

Microsoft Windows システム向けの BEA Tuxedo ソフトウェア製品には、プロセス間通信のサブシステムである BEA Tuxedo IPC Helper (TUXIPC) が同梱されています。IPC Helper はほとんどのマシンではインストール時の設定で実行されますが、コントロール・パネル・アプレットにある [IPC Resources] ページを使用して、TUXIPC サブシステムを調整し、性能を最大化することもできます。

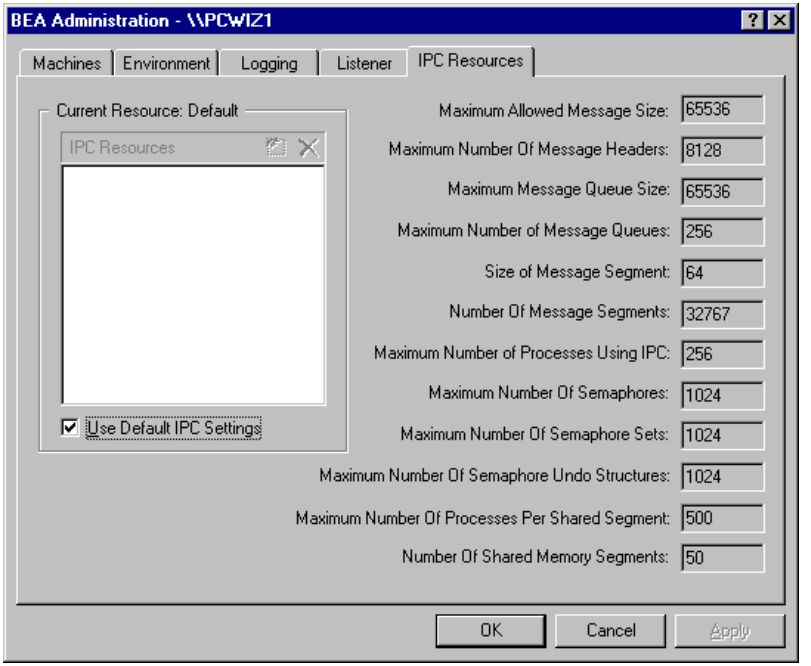
[IPC] コントロール・パネルの [IPC Resources] ページを表示するには、次の手順を行います。

1. [スタート] メニューの [設定] をポイントし、[コントロール パネル] を選択します。Microsoft Windows のコントロール・パネルが表示されます (図 7-1)。

図 7-1　Microsoft Windows のコントロール・パネル

2. [BEA Administration] アイコンをクリックします。[BEA Administration] コントロール・パネルが表示されます (図 7-2)。
3. [IPC Resources] タブをクリックします。[BEA Administration] コントロール・パネルの [IPC Resources] コントロール・パネル部分が表示されます (図 7-2)。

図 7-2 Microsoft Windows 向け BEA Tuxedo ソフトウェアの [IPC Resources] コントロール・パネル



BEA Tuxedo マシンの IPC 設定を定義するには、次の手順を行います。

1. [Current Resource: Default] ボックスで、[Use Default IPC Settings] チェック・ボックスをクリックしてクリアします。
2. 挿入ボックスをクリックします。
3. マシンの名前を入力して Enter キーを押します。

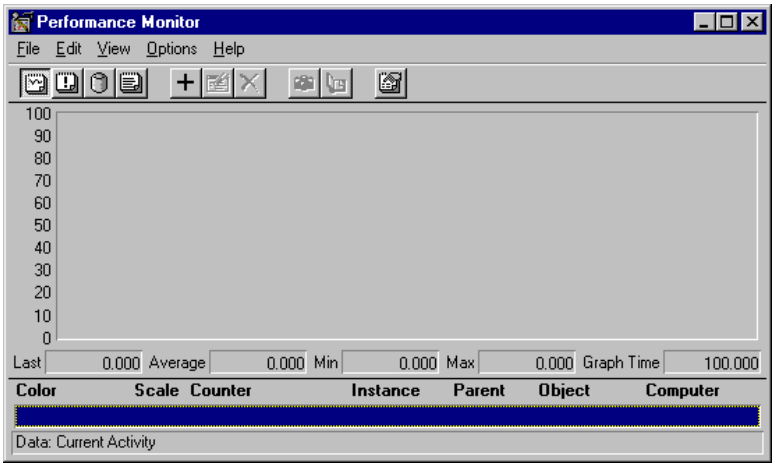
4. 設定する IPC 資源の横にあるフィールドをクリックし、必要な値を入力して、[Apply] をクリックします。[Apply] をクリックすると、変更が登録テーブルに保存されます。変更内容を有効にするには、`tuxipc.exe` サービスを一度終了してから再起動します。

5. [OK] をクリックしてコントロール・パネルを閉じます。

実行中の BEA Tuxedo サーバ・アプリケーションのパフォーマンスは、パフォーマンス・モニタで確認できます。

パフォーマンス・モニタを起動するには、[スタート] メニューの [プログラム] をポイントし、[管理ツール] の [パフォーマンス モニタ] を選択します。パフォーマンス・モニタの画面が表示されます (図 7-3)。

図 7-3　Microsoft Windows 向け BEA Tuxedo ソフトウェアのパフォーマンス・モニタ



UBBCONFIG ファイルと TUXCONFIG ファイルの作成

イベントのポスト元アプリケーションとサブスクライバ・アプリケーションが BEA Tuxedo ドメインの CORBA オブジェクト (この場合はノーティフィケーション・サービス) と通信するためには、ノーティフィケーション・サービスで UBBCONFIG ファイルが必要です。UBBCONFIG ファイルは、ノーティフィケーション・サービス・アプリケーションの開発の過程で記述する必要があります。そうしないと、アプリケーションをビルドして実行することができません。

UBBCONFIG ファイルを記述した後は、tmloadcf コマンドを使用して TUXCONFIG ファイル (実行時に使用) を生成します。したがって、TUXCONFIG ファイルはノーティフィケーション・サービス・アプリケーションの起動前に存在していなければなりません。TUXCONFIG ファイルは、バイナリ形式の UBBCONFIG ファイルです。tmloadcf コマンドの使用例を次に示します。

```
tmloadcf -y ubb
```

UBBCONFIG ファイルを記述する前に、ノーティフィケーション・サービス・アプリケーションのコンフィギュレーション要件をリストしておく必要があります。要件をリストするためには、サブスクリプションをサポートするために必要なサーバとプロセスを特定します。表 7-1 は、異なるタイプのサブスクリプションのコンフィギュレーション要件を示しています。

表 7-1 一時的および永続的なサブスクリプションのコンフィギュレーション要件

サポート対象のサブスクリプション	UBBCONFIG ファイルに含まれなければならないサーバとプロセス
一時的なサブスクリプション	TMUSREVT、TMNTS、および TMNTSFWD_T
永続的なサブスクリプション	TMUSREVT、TMNTS、TMNTSFWD_P、TMQUEUE, TMQFORWARD

IIOP が使用されるイベント・サブスクライバ・アプリケーションを使用する場合は、IIOP ハンドラ (ISH) に接続されていないコールバック・オブジェクトを呼び出せるようにアウトバウンド IIOP を有効にするパラメータで、UBBCONFIG ファイルの IIOP リスナ (ISL) コマンドをコンフィギュレーションする必要があ

ります。ISL コマンドの -O オプション (大文字の O) はアウトバウンド IIOP を有効にします。追加パラメータを使用すると、システム管理者はノーティフィケーション・サービス・アプリケーションの最適なコンフィギュレーションを取得できます。ISL コマンドの詳細については、『BEA Tuxedo アプリケーションの設定』を参照してください。

ノーティフィケーション・サービス・アプリケーションを開発するときには、UBBCONFIG ファイルの SERVERS セクションで次のタイプのサーバを設定できます。

- TMUSREVT

  tppost(3) からのイベント・レポート用メッセージ・バッファを処理し、EventBroker としてそれらのバッファをフィルタ処理して配信する BEA Tuxedo システム付属のサーバ (必須)。

- TMNTS

  サブスクリプションおよびイベントのポストの要求を処理する BEA Tuxedo ノーティフィケーション・サービス・サーバ (必須)。

- TMNTSFWD_T

  一時的なサブスクリプションのサブスクライバに一時的なイベントを転送する BEA Tuxedo ノーティフィケーション・サービス・サーバ (一時的なサブスクリプションで必須)。

- TMNTSFWD_P

  永続的なサブスクリプションのあるサブスクライバに永続的なイベントを転送する BEA Tuxedo ノーティフィケーション・サービス・サーバ。サブスクライバに配信できないイベントはエラー・キューに送信されます (永続的なサブスクリプションで必須)。

- TMQUEUE

  イベント・キューを管理する BEA Tuxedo サーバ (永続的なサブスクリプションで必須)。

- TMQFORWARD

  永続的なサブスクライバに転送できるように、イベントをノーティフィケーション・サービス TMNTSFWD_P サーバに転送する BEA Tuxedo サーバ (永続的なサブスクリプションで必須)。

- ISL

  BEA Tuxedo IIOP サーバ・リスナ / ハンドラ・プロセス (イベントのポスト元またはサブスクライバがローカル・ドメインの外側にある場合に必須)。

リスト 7-10 は、ノーティフィケーション・サービスの Introductory サンプル・アプリケーションの UBBCONFIG ファイルです。Introductory サンプル・アプリケーションでは一時的なサブスクリプションのみがサポートされており、永続的なサブスクリプションまたはデータのフィルタ処理はサポートされていません。

リスト 7-10 Introductory サンプルの UBBCONFIG ファイル

```
# この UBBCONFIG ファイルは一時的なサブスクリプションのみをサポートし、
# 永続的なサブスクリプションまたはデータのフィルタ処理はサポートしない
*RESOURCES
    IPCKEY 52359
    DOMAINID events_intro_simple_cxx
    MASTER SITE1
    MODEL SHM
#------------------------------------------------------------------
*MACHINES
    "BEANIE"
        LMID = SITE1
    APPDIR = "D:\tuxdir\EVENTS~1\INTROD~2"
    TUXCONFIG = "D:\tuxdir\EVENTS~1\INTROD~2\tuxconfig"
    TUXDIR = "d:\tuxdir"
    MAXWSCLIENTS = 10
    ULOGPFX = "D:\tuxdir\EVENTS~1\INTROD~2\ULOG"
#------------------------------------------------------------------
# 一時的なイベントを使用するので、グループがトランザクション
# に関与する必要はない
*GROUPS
    SYS_GRP
    LMID = SITE1
    GRPNO = 1
#------------------------------------------------------------------
*SERVERS
    DEFAULT:
    CLOPT = "-A"
    TMSYSEVT
    SRVGRP = SYS_GRP
    SRVID = 1
TMUSREVT
    SRVGRP = SYS_GRP
    SRVID = 2
TMFFNAME
    SRVGRP = SYS_GRP
    SRVID = 3
    CLOPT = "-A -- -N -M"
TMFFNAME
    SRVGRP = SYS_GRP
```

```
    SRVID = 4
    CLOPT = "-A -- -N"
TMFFNAME
    SRVGRP = SYS_GRP
    SRVID = 5
    CLOPT = "-A -- -F"
# ノーティフィケーション・サービス・サーバを起動
#
TMNTS
    SRVGRP = SYS_GRP
    SRVID = 6
# ノーティフィケーション・サービスの一時的なイベント転送サーバを起動
#
TMNTSFWD_T
    SRVGRP = SYS_GRP
    SRVID = 7
# クライアントのコールバックを使用するので -O を使用して ISL を起動
ISL
    SRVGRP = SYS_GRP
    SRVID = 8
    CLOPT = "-A -- -O -n //BEANIE:2359"
#--------------------------------------------------------------------
*SERVICES
```

リスト 7-11 のコード例は、ノーティフィケーション・サービスの Advanced サンプル・アプリケーションからの引用です。Advanced サンプル・アプリケーションでは、一時的なサブスクリプションと永続的なサブスクリプションおよびデータのフィルタ処理がサポートされています。

リスト 7-11 Advanced サンプルの UBBCONFIG ファイル

```
# この UBBCONFIG ファイルは一時的なサブスクリプションと永続的な
# サブスクリプションおよびデータのフィルタ処理をサポートする
*RESOURCES
    IPCKEY 52363
    DOMAINID events_advanced_simple_cxx
    MASTER SITE1
    MODEL SHM
#--------------------------------------------------------------------
*MACHINES
    "BEANIE"
        LMID = SITE1
    APPDIR = "D:\tuxdir\EVENTS~1\ADVANC~1"
    TUXCONFIG = "D:\tuxdir\EVENTS~1\ADVANC~1\tuxconfig"
    TUXDIR = "d:\tuxdir"
    MAXWSCLIENTS = 10
    ULOGPFX = "D:\tuxdir\EVENTS~1\ADVANC~1\ULOG"
#
```

```
#   永続的なイベントを使用するので、トランザクション・ログが必要
#
     TLOGDEVICE = "D:\tuxdir\EVENTS~1\ADVANC~1\TLOG"
     TLOGSIZE = 10
#--------------------------------------------------------------------
*GROUPS
     SYS_GRP
     LMID = SITE1
     GRPNO = 1
#   ノーティフィケーション・サービス・サーバのヌル・トランザクション
#   ・グループを作成する
#
NTS_GRP
     LMID = SITE1
     GRPNO = 2
     TMSNAME = TMS
     TMSCOUNT = 2
#   永続的なイベントを使用するので、永続的なキューが必要
#   キュー・サーバのキュー・トランザクション・グループを作成
#
QUE_GRP
     LMID = SITE1
     GRPNO = 3
     TMSNAME = TMS_QM
     TMSCOUNT = 2
#
#   作成した QUE スペースをキュー・グループに管理させる
#   TMNTSQS として指定するキュースペースの名前は、作成したキュースペース
#   の名前と同じでなければならない
#
   OPENINFO = "TUXEDO/QM:D:\tuxdir\EVENTS~1\ADVANC~1\QUE;TMNTSQS"
#--------------------------------------------------------------------
*SERVERS
     DEFAULT:
     CLOPT = "-A"
#
#   キュー・サーバを起動
#   CLOPT の -s オプションで指定するキュースペースの名前は、
#   作成したキュースペースの名前と同じでなければならない
#
TMQUEUE
     SRVGRP = QUE_GRP
     SRVID = 1
     CLOPT = "-s TMNTSQS:TMQUEUE --  "
#
#   キュー転送サーバを起動し、イベントをノーティフィケーション・サービス


#   の永続的な転送サーバに転送させる
#
TMQFORWARD
     SRVGRP = QUE_GRP
     SRVID = 2
     CLOPT = "-- -i 2 -q TMNTSFWD_P"
     TMSYSEVT
```

```
    SRVGRP = NTS_GRP
    SRVID = 1
#
#  ユーザ EventBroker を起動する。ユーザ EventBroker が
#  「Story」fml フィールドの定義を見つけることができるよ
#  うに環境ファイルを渡す。これで、ユーザ EventBroker が
#  データのフィルタ処理を実行できるようになる
#
TMUSREVT
    SRVGRP = NTS_GRP
    SRVID = 2
    ENVFILE = "D:\tuxdir\EVENTS~1\ADVANC~1\envfile"
TMFFNAME
    SRVGRP = SYS_GRP
    SRVID = 1
    CLOPT = "-A -- -N -M"
TMFFNAME
    SRVGRP = SYS_GRP
    SRVID = 2
    CLOPT = "-A -- -N"
TMFFNAME
    SRVGRP = SYS_GRP
    SRVID = 3
    CLOPT = "-A -- -F"
#
#  ノーティフィケーション・サービス・サーバを起動する。
#  ノーティフィケーション・サーバがデータのフィルタ処理を実行
#  できるように環境ファイルを渡す。永続的なイベントを使用するので、
#  -s オプションを指定しなければならない。-s オプションは、TMNTSQS
#   としてキュースペースの名前を指定する。この名前は作成したキュー
#   スペースの名前と同じでなければならない
#
TMNTS
    SRVGRP = NTS_GRP
    SRVID = 3
    ENVFILE = "D:\tuxdir\EVENTS~1\ADVANC~1\envfile"
    CLOPT = "-A -- -s TMNTSQS"
#
#  ノーティフィケーション・サービスの一時的なイベント転送サーバを
#  起動する。サーバがデータのフィルタ処理を実行できるように環境
#  ファイルを渡す
#


TMNTSFWD_T
    SRVGRP = NTS_GRP
    SRVID = 4
    ENVFILE = "D:\tuxdir\EVENTS~1\ADVANC~1\envfile"
#
#  ノーティフィケーション・サービスの永続的なイベント転送サーバを
#  起動する。サーバがデータのフィルタ処理を実行できるように環境
#  ファイルを渡す
#
TMNTSFWD_P
```

```
    SRVGRP = NTS_GRP
    SRVID = 5
    ENVFILE = "D:\tuxdir\EVENTS~1\ADVANC~1\envfile"
#
#   クライアントのコールバックを使用するので -O を使用して ISL を起動
#
ISL
    SRVGRP = SYS_GRP
    SRVID = 4
    CLOPT = "-A -- -O -n //BEANIE:2363"
#-------------------------------------------------------------
*SERVICES
```

ノーティフィケーション・サービスの管理

ノーティフィケーション・サービス・アプリケーションをデプロイした後は、次の管理タスクを継続的に実行する必要があります。

1. データベースの同期
2. システムからのデッド・サブスクリプションのパージ
3. キュー使用率の監視
4. キューからの不要なイベントのパージ
5. エラー・キューのイベントの移動または削除

データベースの同期

複数の EventBroker をコンフィギュレーションする場合は、ノーティフィケーション・サービスのサブスクリプション・データベースで同期を取る必要があります。同期プロセスは時間を要し (イベントの配信に影響することがある)、ネットワーク・トラフィックを増大させるので、イベントのトラフィックが要求しない限りは複数の EventBroker をコンフィギュレーションしないでください。

複数の EventBroker をコンフィギュレーションするときには、TMUSREVT サーバの `-P` オプションを使用してデータベースの同期に要する時間をコンフィギュレーションできます。このオプションの設定方法については、『BEA Tuxedo のファイル形式とデータ記述方法』の `TMUSREVT(5)` を参照してください。

注記 データベースの同期に要する時間は、サブスクライバがサブスクライブしてからイベントを受信するまでの所要時間に影響します。また、サブスクライバがアンサブスクライブしてからイベントの受信が停止されるまでの所要時間にも影響します。

システムからのデッド・サブスクリプションのパージ

サブスクリプションは、サブスクライバが永続的なサブスクリプションを作成し、アンサブスクライブせずにシャットダウンしてから、再起動することなくノーティフィケーション・サービスに再接続しなかった場合、またはサブスクライバがどのイベントとも一致しないサブスクリプションを作成した場合にデッド状態になります。サブスクライバでは永続的なサブスクリプションを作成してから、アンサブスクライブせずにシャットダウンすることができますが、定期的に再接続して蓄積されたイベントを取得しないとエラーになります。ノーティフィケーション・サービスでは永続的なサブスクリプションと一致するイベントの配信が定期的に試行されるので、そのようなイベントはサブスクライバが接続されていない間蓄積され、キュースペースを消費し、システム・リソースを浪費します。

どのイベントとも一致することのないサブスクリプションは、意味を成さないので作成しないようにしてください。また、ポストされた各イベントは各サブスクリプションと照合しなければならないので、サブスクリプションはシステム・リソースを消費します。

表 7-2 の `ntsadmin` コマンドを使用すると、すべてのサブスクリプションを表示し、各サブスクリプションの保留キューとエラー・キューにどのくらいの数のイベントが入っているのかを確認できます。また、`ntsadmin` コマンドを使用してサブスクリプションを削除したり、エラー・キューのイベントを保留キューに移動したりすることもできます。`ntsadmin` ユーティリティの詳細については、第 7 章の 28 ページ「ntsadmin」を参照してください。

表 7-2 ntsadmin コマンドの概要

コマンド	説明
`subscriptions`	サブスクリプション・データベースにあるサブスクリプションをリストします。

表 7-2 ntsadmin コマンドの概要 (続き)

コマンド	説明
rmsubscriptions	サブスクリプション・データベースのサブスクリプションを削除します。
pendevents	保留イベント・キューにあるイベントの情報を表示します (永続的なサブスクリプションのみ)。
rmpendevents	保留イベント・キューのイベントを削除します (永続的なサブスクリプションのみ)。
errevents	イベント・エラー・キューのイベントをリストします (永続的なサブスクリプションのみ)。
rmerrevents	イベント・エラー・キューのイベントを削除します (永続的なサブスクリプションのみ)。

自動的にデッド・サブスクリプションを検出する手段はありませんが、ntsadmin ユーティリティを利用して、サブスクリプションがデッド状態かどうか、いつデッド状態になるのかを判断することはできます。

キュー使用率の監視

キューは、固定量のスペースを割り当てて作成します。このスペースは、イベントが蓄積されるにつれて消費されていきます。キューが一杯になると、それ以降はイベントをキューに入れることはできません。

キューの使用率を監視するには、qmadmin または ntsadmin を使用します (『BEA Tuxedo コマンド・リファレンス』の qmadmin(1) を参照)。

保留イベントを保持するためにキュースペースが作成されたときには、そのキュースペースで保持できるイベントの最大数が指定されました。たとえば、Advanced サンプル・アプリケーションでは、TMNTSQS キュースペースのイベントの最大数は 200 に設定されました (第 7 章の 7 ページ「イベント・キューの作成」を参照)。キュースペースの容量がわかっていれば、ntsadmin pendevents コマンドを使用してイベント・キューで保留されているイベントの数を確認できます。イベント・キューが一杯であるか、一杯に近い場合は、イベント最大数の設定を増やすか、イベント・キューの数を増やす必要があります。

注記 qmadmin qcreate コマンドでしきい値コマンド・オプション (cmd) を使用すると、キューがいっぱいに近くなったときに警告を生成することができます。このコマンドの詳細については、『BEA Tuxedo コマンド・リファレンス』の qmadmin(1) を参照してください。

キューからの不要なイベントのパージ

ntsadmin コマンドの rmerrevents および rmpendevents を使用すると、保留キューまたはエラー・キューからイベントをパージできます。

警告： キューからイベントを削除した後は、そのイベントを復元することはできません。イベントは完全に消滅し、サブスクライバ・アプリケーションがそのイベントを受信することはありません。

エラー・キューの管理

イベントの配信が事前に設定された回数試行された後、そのイベントはエラー・キューに移動します。イベントがエラー・キューに移動されたら、管理者はシステムからそのイベントをパージするか、そのイベントを保留キューに戻さなければなりません。イベントのパージについては、前節で説明しています。

エラー・キューから保留キューにイベントを移動するということは、イベント配信の再開を要求するということです。イベント配信の失敗はシステム・リソースを消費するので、配信の障害となっていた状況が修正されている場合以外は保留キューにイベントを戻さないでください。イベントを保留キューに戻すには、ntsadmin retryerrevents コマンドを使用します。

ノーティフィケーション・サービスの管理ユーティリティとコマンド

ここでは、次の内容について説明します。

- ntsadmin ユーティリティ
- ntsadmin コマンド
- ntsadmin ユーティリティの使い方

ntsadmin ユーティリティ

この節では、ntsadmin ユーティリティについて説明します。

ntsadmin

概要 BEA Tuxedo CORBA ノーティフィケーション・サービス管理コマンドのインタプリタです。

構文 ntsadmin

説明 ノーティフィケーション・サービスには、CORBA ノーティフィケーション・サービス・アプリケーションで次のタスクを実行するコマンドを備える管理コマンド・インタプリタ (ntsadmin) があります。

- サブスクリプションをリストする。
- サブスクリプションを削除する。
- 保留キューおよびエラー・キューにある構造化イベントの要約情報を表示する。
- 保留キューおよびエラー・キューにある構造化イベントを削除する。
- エラー・キューから保留キューに構造化イベントを移動する。

注記 アプリケーションに一時的なサブスクリプションしかない場合、ntsadmin を入力してプログラムを起動すると、永続的なサブスクリプション用のコマンドは無効になります。

注記 ntsadmin を使用するには、事前にノーティフィケーション・サービスが実行されていなければなりません。

コマンド・プロンプトで q を入力すると、ntsadmin プログラムを終了できます。Break キーを押すと、コマンドからの出力を終了できます。その後、プログラムは新しいコマンドの入力を要求します。

ntsadmin からの出力は、使用されているページ割りコマンドに従ってページ割りされます (paginate コマンドを参照)。

注記 subscription コマンドの出力は、verbose コマンドの設定によって異なります。

セキュリティ このユーティリティを使用できるのはシステム管理者のみです。

関連項目 TMNTS、TMNTSFWD_T、TMNTSFWD_P、qmadmin

ntsadmin コマンド

コマンドは、フル・ネームか略称で入力できます (略称がある場合は、フル・ネームの後にかっこで示されている)。コマンドの後には、適切な引数を入力します。角かっこ [] で示されている引数はオプションです。中かっこ {} で示されている引数は、相互に排他的なオプションの選択を示します。各コマンドには、次のオプションがあります。

オプション	定義
[-i identifier]	指定された場合は、identifier と一致するサブスクリプションを識別します。
[-n name]	指定された場合は、***name*** と一致するサブスクリプション名のサブスクリプションのみを識別します。空の文字列と一致する名前を指定するには (つまり名前のないサブスクリプション)、空の文字列を引用符 ("") で囲みます。 **注記** このオプションではワイルドカード文字 (*) がサポートされていないので、名前はサブスクリプション名と正確に一致しなければなりません。
[-t]	指定された場合は、QoS が一時的のサブスクリプションのみを示します。
[-p]	指定された場合は、QoS が永続的のサブスクリプションのみを示します。

ntsadmin コマンドを次に示します。

subscriptions (sub) [{-i *identifier* \|-n *name* \|-t \| -p}]
サブスクリプション・データベースにあるサブスクリプションをリストします。

注記 subscription コマンドの出力は、冗長モードが有効かどうかによって異なります (verbose コマンドについては後述)。リスト 7-12 は、冗長モードが有効な場合と無効な場合の subscription の出力例を示しています。

リスト 7-12　冗長モードが有効な場合と無効な場合の subscription コマンドの出力

```
> verbose on
Verbose mode is now on

> sub
          ID: 1000000006
        Name: marcello
         QoS: Transient
      Qspace: <N/A>
  Expression: stock trade\.quote
      Filter: stock_name %% 'BEAS' && price_per_share > 150

          ID: 1000000005
        Name: marcello
         QoS: Persistent
      Qspace: TMNTSQS
  Expression: stock trade\.sell
      Filter:

          ID: 1000000004
        Name: marcello
         QoS: Persistent
      Qspace: TMNTSQS
  Expression: stock trade\.buy
      Filter:

> verbose off
Verbose mode is now off

> sub
ID          Name                 Expression
--          ----                 ----------
1000000006  marcello         [T] stock trade\.quote
1000000005  marcello         [P] stock trade\.sell
1000000004  marcello         [P] stock trade\.buy
```

rmsubscriptions (rmsub) [{-i *identifier* |-n *name* |-t | -p]}[-y]
サブスクリプション・データベースからサブスクリプションを削除します。このコマンドは、**-y** が使用されている場合以外は確認を要求します。

このコマンドでは、削除されたサブスクリプションの数が表示されます。

pendevents (pevt) [{-i *identifier* |-n *name*}]
保留イベント・キューのイベントについての情報を表示します。

rmpendevents (rmpevt) [{-i *identifier* |-n *name* |-o}][-y]
保留イベント・キューのイベントを削除します。**-o** が指定された場合は、サブスクリプション・データベースに対応するサブスクリプションのないすべてのイベントが削除されます。

このコマンドは、**-y** が指定されている場合以外は確認を要求します。このコマンドでは、削除されたイベントの数が表示されます。

errevents (eevt) [{-i *identifier* |-n *name*}]
イベント・エラー・キューのイベントをリストします。

rmerrevents (rmeevt) [{-i *identifier* |-n *name* |-o}][-y]
イベント・エラー・キューのイベントを削除します。**-o** が指定された場合は、サブスクリプション・データベースに対応するサブスクリプションのないすべてのイベントが削除されます。

このコマンドは、**-y** が指定されている場合以外は確認を要求します。このコマンドでは、削除されたイベントの数が表示されます。

retryerrevents (reteevt) [{-i *identifier* |-n *name*}][-y]
イベント・エラー・キューのイベントの配信を再試行します。このコマンドでは、エラー・キューから保留キューにイベントが移動されます。

このコマンドは、**-y** が使用されている場合以外は確認を要求します。このコマンドでは、エラー・キューから保留キューに移動されたイベントの数が表示されます。

quit (q)
セッションを終了します。

echo (e) [{off |on}]
on に設定されている場合に入力コマンド行をエコーします。入力がない場合は、現在の設定が切り替えられ、新しい設定が出力されます。初期設定は **off** です。

help (h) [{*command* |all}]
ヘルプ・メッセージを出力します。***command*** が指定されている場合は、そのコマンドの略称、引数、および説明が出力されます。**all** が指定されている場合は、すべてのコマンドの説明が表示されます。引数を省略すると、すべてのコマンドの構文が表示されます。

paginate (page) [{off |on}]
出力をページ割りします。入力がない場合は、現在の設定が切り替えられ、新しい設定が出力されます。標準入力または標準出力のいずれかが端末デバイスではない場合を除いて初期設定は **on** です。ページ割りは、

標準入力と標準出力が両方とも端末デバイスである場合に限り有効にできます。シェル環境変数 PAGER を使用すると、出力のページングに使用されるデフォルトのコマンドを上書きできます。デフォルトのページング・コマンドは、ネイティブのオペレーション・システム環境に固有のページャです。たとえば、コマンド pg は UNIX オペレーティング・システムのデフォルトです。

verbose (v) [{on | off }]

冗長モードで出力を生成します。オプションが指定されない場合は、現在の設定が切り替えられ、新しい設定が出力されます。初期設定は **off** です。

! *shellcommand*

このコマンドを使用すると、シェルにエスケープして ***shellcommand*** を実行できます。

!!

このコマンドを使用すると、前回のシェル・コマンドを繰り返すことができます。

#[text]

このコマンドを使用すると、行をコメントとして指定できます。

<CR>

このコマンドを使用すると、前回のコマンドを繰り返すことができます。

ntsadmin ユーティリティの使い方

この節では、ntsadmin ユーティリティの使用例を紹介します。

リスト 7-13 は、ntsadmin を使用してエラー・キューから保留キューにイベントを移動する例を示しています。このタスクは次のように実行されます。

1. すべてのサブスクリプションで marcello を検索します。
2. 一意の subscription_id を使用して、エラー・キューにあるイベントの情報を表示します。
3. エラー・キューから保留キューにイベントを移動します。

リスト 7-13　エラー・キューから保留キューへのイベントの移動

```
D:\smith\reg>ntsadmin
ntsadmin - Copyright (c) 1996-1999 BEA Systems, Inc.
Portions * Copyright 1986-1997 RSA Data Security, Inc.
All Rights Reserved.
Distributed under license by BEA Systems, Inc.
BEA Tuxedo is a registered trademark.
INFO: /Q Qspace - TMNTSQS
INFO: /Q Device - D:\smith\reg\QUE (SITE1)

> subscriptions -n marcello
ID          Name                Expression
--          ----                ----------
1000000002  marcello         [T] stock trade\.quote
1000000001  marcello         [P] stock trade\.sell
1000000000  marcello         [P] stock trade\.buy

> verbose off
Verbose mode is now off

> eevt -i 1000000003
ID          Name             Count
--          ----             -----
1000000003  marcello         1
> reteevt -i 1000000003 -y
1 event(s) retried
```

リスト 7-14 は、ntsadmin を使用してサブスクリプションを削除し、イベントをパージする例を示しています。

リスト 7-14　サブスクリプションの削除

```
> rmsub -n BillJones -y
2 subscription(s) removed
> rmeevt -n marcello -y
1 event(s) removed
> rmpevt -n BillJones -y
No events removed
```

リスト 7-15 は、特定サブスクリプションの保留イベントを調べる方法を示しています。

リスト 7-15　保留イベントのチェック

```
> pevt -n marcello
ID          Name             Count
--          ----             -----
1000000003  marcello          1
```

ノーティフィケーション・サーバ

この節では、次のサーバについて説明します。

- TMTNS
- TMNTSFWD_T
- TMNTSFWD_P

ノーティフィケーション・サービスでは、次の BEA Tuxedo システム・サーバも使用されます。これらのサーバの説明については、『BEA Tuxedo のファイル形式とデータ記述方法』を参照してください。

- TMSYSEVT(5)
- TMUSREVT(5)
- TMQFORWARD(5)
- TMQUEUE(5)

TMNTS

概要 サブスクリプションおよびイベントのポストの要求を処理します。

構文

```
TMNTS SRVGRP="identifier" SRVID="number"
     [CLOPT="[-A] [servopts options]
     [--[-S queuespace]"]
```

説明 TMNTS は、サブスクリプションおよびイベントのポストのすべての要求を処理する BEA Tuxedo 提供のサーバです。

パラメータ -S *queuespace*

使用するキュースペースの名前です。このキュースペースには、TMNTSFWD_P および TMNTSFWD_E という 2 つのキューがなければなりません。このオプションは、永続的なサブスクリプションでのみ必須です。

注記 QoS が Persistent のサブスクリプションを使用する場合は、システムが作動する前にキュースペース、イベントを保持するキュー、およびエラー・キューを作成する必要があります。キュースペースの名前は、TMNTS サーバの CLOPT -S *queuespace* パラメータを使用して指定された *queuespace* 名と同じでなければなりません。イベント・キューは、TMNTSFWD_P という名前に設定する必要があります。エラー・キューは、TMNTSFWD_E という名前に設定する必要があります。

TMNTS サーバは、信頼性と可用性を高めるために複数を起動することができます。

TMNTS サーバは、イベントがトランザクションのコンテキストでポストされる場合はトランザクション・グループに属していなければなりません。

相互運用性 TMNTS は、バージョン 5.0 以上の BEA WebLogic Enterprise またはバージョン 8.0 以上の BEA Tuxedo 上で実行しなければなりません。

注記 TMNTS サーバは、TMUSREVT サーバと TMSYSEVT サーバが提供するサービスを利用します。したがって、システムが作動する前にそれらのサーバを起動する必要があります。一時的なサブスクリプションを使用する場合は、TMNTSFWD_T サーバもシステムの作動前に起動しなければなりません。永続的なサブスクリプションを使用する場合は、TMNTSFWD_P **サーバ、**TMQUEUE サーバ、および TMQFORWARD サーバもシステムの作動前に起動する必要があります。

例

```
*SERVERS
```

```
TMNTS SRVGRP = NTS_GRP  SRVID = 3
      CLOPT = "-A -- -s TMNTSQS"
```

関連項目　TMSYSEVT(5)、TMUSREVT(5)、TMQUEUE(5)、TMQFORWARD(5)、TMNTSFWD_P、TMNTSFWD_T(5)、UBBCONFIG(5)

TMNTSFWD_T

概要 イベントを一時的なサブスクライバに転送します。

構文
```
TMNTSFWD_T SRVGRP="identifier" SRVID="number"
       [CLOPT="[-A][--"]
```

説明 TMNTSFWD_T は、QoS が Transient のサブスクライバにイベントを転送する BEA Tuxedo 提供のサーバです。イベントの配信に関連付けられるトランザクション・コンテキストはありません。

注記 TMNTSFWD_T サーバは、信頼性と可用性を高めるために複数を起動することができます。

相互運用性 TMNTS は、バージョン 5.0 以上の BEA WebLogic Enterprise またはバージョン 8.0 以上の BEA Tuxedo 上で実行しなければなりません。

注記 TMNTSFWD_T サーバは、TMNTS サーバ、TMUSREVT サーバ、および TMSYSEVT サーバが提供するサービスを利用します。したがって、システムが作動する前にそれらのサーバを起動する必要があります。

例
```
*SERVERS

TMNTSFWD_T  SRVGRP = SYS_GRP  SRVID = 7
```

関連項目 TMSYSEVT(5)、TMUSREVT(5)、TMNTS(5)、TMNTSFWD_P、UBBCONFIG(5)。第 7 章の 8 ページ「一時的なサブスクリプションの IPC キュースペース」も参照してください。

TMNTSFWD_P

概要　イベントを永続的なサブスクライバに転送します。

構文

```
TMNTSFWD_P SRVGRP="identifier" SRVID="number"
       CLOPT="[-A] [--"]
```

説明　TMNTSFWD_P は、QoS が永続的であるサブスクライバにイベントを転送する BEA Tuxedo 提供のサーバです。イベントの配信に関連付けられるトランザクション・コンテキストはありません。

TMNTSFWD_P サーバは、信頼性と可用性を高めるために複数を起動することができます。

相互運用性　TMNTS は、バージョン 5.0 以上の BEA WebLogic Enterprise またはバージョン 8.0 以上の BEA Tuxedo 上で実行しなければなりません。

注記　TMNTSFWD_P サーバは、TMNTS サーバ、TMUSREVT サーバ、TMSYSEVT サーバ、TMQUEUE サーバ、および TMQFORWARD サーバが提供するサービスを利用します。したがって、システムが作動する前にそれらのサーバを起動する必要があります。

このサーバは、トランザクション・グループで起動する必要があります。

TMNTSFWD_P サーバは、TMQFORWARD サーバと同じ数だけ起動する必要があります。

例

```
*SERVERS

TMNTSFWD_P  SRVGRP = NTS_GRP  SRVID = 5
```

関連項目　TMSYSEVT(5)、TMUSREVT(5)、TMNTS、TMNTSFWD_T、servopts(5)、UBBCONFIG(5)

索引

は

ひ

ふ

ほ

ま

れ

ろ