VisiTime ガイド

Borland VisiBroker® 7.0

Borland®
Excellence Endures™

Borland Software Corporation
20450 Stevens Creek Blvd., Suite 800
Cupertino, CA 95014 USA
www.borland.com

2006 年 5 月 11 日初版発行

著者：Borland Software Corporation

発行：ボーランド株式会社

PDF

目次

第 1 章

Borland VisiBroker の概要

Borland は，CORBA 開発者に向けて，業界最先端の VisiBroker オブジェクトリクエストブローカー（ORB）を活用するために *VisiBroker for Java*，*VisiBroker for C++*，および *VisiBroker for .NET* を提供しています。この 3 つの VisiBroker は CORBA 2.6 仕様の実装です。

VisiBroker の概要

VisiBroker は，CORBA が Java オブジェクトと Java 以外のオブジェクトの間でやり取りする必要がある分散配布で使用されます。幅広いプラットフォーム（ハードウェア，オペレーティングシステム，コンパイラ，および JDK）で使用できます。VisiBroker は，異種環境の分散システムに関連して一般に発生するすべての問題を解決します。

VisiBroker は次のコンポーネントからなります。

- VisiBroker for Java，VisiBroker for C++，および VisiBroker for .NET（業界最先端のオブジェクトリクエストブローカーの 3 つの実装）。
- VisiNaming Service - Interoperable Naming Specification バージョン 1.3 の完全な実装。
- GateKeeper - ファイアウォールの背後の CORBA サーバーとの接続を管理するプロキシサーバー。
- VisiBroker Console - CORBA 環境を簡単に管理できる GUI ツール。
- コモンオブジェクトサービス - VisiNotify（通知サービス仕様の実装），VisiTransact（トランザクションサービス仕様の実装），VisiTelcoLog（Telecom ログサービス仕様の実装），VisiTime（タイムサービス仕様の実装），VisiSecure など。

VisiBroker の機能

VisiBroker には次の機能があります。

- セキュリティと Web 接続性を容易に装備できます。
- J2EE プラットフォームにシームレスに統合できます（CORBA クライアントが EJB に直接アクセスできる）。
- 堅牢なネーミングサービス（VisiNaming）とキャッシュ，永続的ストレージ，および複製によって高可用性を実現します。
- プライマリサーバーにアクセスできない場合に，クライアントをバックアップサーバーに自動的にフェイルオーバーします。
- CORBA サーバークラスタ内で負荷分散を行います。
- OMG CORBA 2.6 仕様に完全に準拠します。
- Borland JBuilder 統合開発環境と統合されます。
- Borland AppServer などの他の Borland 製品と最適に統合されます。

VisiBroker のマニュアル

VisiBroker のマニュアルセットは次のマニュアルで構成されています。

- *Borland VisiBroker インストールガイド* — VisiBroker をネットワークにインストールする方法について説明します。このマニュアルは，Windows または UNIX オペレーティングシステムに精通しているシステム管理者を対象としています。
- *Borland VisiBroker セキュリティガイド* — VisiSecure for VisiBroker for Java および VisiBroker for C++ など，VisiBroker のセキュリティを確保するための Borland のフレームワークについて説明しています。
- *Borland VisiBroker for Java 開発者ガイド* — Java による VisiBroker アプリケーションの開発方法について記載されています。Visibroker ORB の設定と管理，およびプログラミングツールの使用方法について説明します。また, IDL コンパイラ, スマートエージェント，ロケーションサービス，ネーミングサービス，イベントサービス，オブジェクトアクティベーションデーモン (OAD), Quality of Service (QoS), インターフェースリポジトリ，および Web サービスサポートについても説明します。
- *Borland VisiBroker for C++ 開発者ガイド* — C++ による VisiBroker アプリケーションの開発方法について記載されています。Visibroker ORB の設定と管理，およびプログラミングツールの使用方法について説明します。また, IDL コンパイラ, スマートエージェント，ロケーションサービス，ネーミングサービス，イベントサービス，OAD, QoS，プラグイン可能トランスポートインターフェース，RT CORBA 拡張機能，Web サービスサポート，およびインターフェースリポジトリについても説明します。
- *Borland VisiBroker for .NET 開発者ガイド* — .NET 環境による VisiBroker アプリケーションの開発方法について記載されています。
- *Borland VisiBroker for C++ API リファレンス* — VisiBroker for C++ に付属するクラスとインターフェースについて説明します。
- *Borland VisiBroker VisiTime ガイド* — Borland による OMG Time Service 仕様の実装について説明します。
- *Borland VisiBroker VisiNotify ガイド* — Borland による OMG 通知サービス仕様の実装について説明します。通知メッセージフレームワークの主な機能として, 特に Quality of Service (QoS) のプロパティ, フィルタリング, および Publish/Subscribe Adapter（PSA）の使用方法が記載されています。

- *Borland VisiBroker VisiTransact ガイド* — Borland による OMG Object Transaction Service 仕様の実装および Borland Integrated Transaction Service コンポーネントについて説明します。
- *Borland VisiBroker VisiTelcoLog ガイド* — Borland による OMG Telecom Log Service 仕様の実装について説明します。
- *Borland VisiBroker GateKeeper ガイド* — Web ブラウザやファイアウォールによるセキュリティ制約の下で，VisiBroker GateKeeper を使用して，VisiBroker のクライアントがネットワークを介してサーバーとの通信を確立する方法について説明します。

通常，マニュアルにアクセスするには，VisiBroker とともにインストールされるヘルプビューアを使用します。ヘルプは，スタンドアロンのヘルプビューアからアクセスすることも，VisiBroker コンソールからアクセスすることもできます。どちらの場合も，ヘルプビューアを起動すると独立したウィンドウが表示されるため，このウィンドウからヘルプビューアのメインツールバーにアクセスしてナビゲーションや印刷を行ったり，ナビゲーションペインにアクセスすることができます。ヘルプビューアのナビゲーションペインには，すべての VisiBroker ブックとリファレンス文書の目次，完全なインデックス，および包括的な検索を実行できるページがあります。

重要 Web サイト http://www.borland.com/techpubs には，PDF 版のマニュアルと最新の製品マニュアルがあります。

スタンドアロンヘルプビューアからの VisiBroker オンラインヘルプトピックへのアクセス

製品がインストールされているコンピュータでスタンドアロンのヘルプビューアからオンラインヘルプにアクセスするには，次のいずれかの手順を実行します。

Windows

- ［スタート｜プログラム｜ Borland VisiBroker ｜ Help Topics］の順に選択します。
- または，コマンドプロンプトを開き，製品のインストールディレクトリの ¥bin ディレクトリに移動し，次のコマンドを入力します。

```
help
```

UNIX コマンドシェルを開き，製品のインストールディレクトリの /bin ディレクトリに移動し，次のコマンドを入力します。

```
help
```

ヒント UNIX システムにインストールするときの指定で，PATH エントリのデフォルトに bin を含まないようにします。カスタムインストールオプションを選択して PATH エントリのデフォルトを変更せず，PATH に現在のディレクトリのエントリがない場合は，./help を使用してヘルプビューアを起動できます。

VisiBroker コンソールからの VisiBroker オンラインヘルプトピックへのアクセス

VisiBroker コンソールから VisiBroker オンラインヘルプトピックにアクセスするには，［Help ｜ Help Topics］を選択します。

［Help］メニューには，オンラインヘルプ内のいくつかの文書へのショートカットもあります。ショートカットの 1 つを選択すると，ヘルプトピックビューアが起動し，［Help］メニューで選択した項目が表示されます。

マニュアルの表記規則

VisiBroker のマニュアルでは，文中の特定の部分を表すために，次の表に示す書体と記号を使用します。

表 1.1　マニュアルの表記規則

表記規則	用途
italic	新規の用語およびマニュアル名に使用されます。
`computer`	ユーザーやアプリケーションが提供する情報，サンプルコマンドライン，およびコードです。
bold computer	本文では，ユーザーが入力する情報を示します。サンプルコードでは，重要なステートメントを強調表示します。
[ ]	省略可能な項目。
...	繰り返しが可能な直前の引数。
|	二者択一の選択。

プラットフォームの表記

VisiBroker マニュアルでは，次の記号を使用してプラットフォーム固有の情報を示します。

表 1.2　プラットフォームの表記

記号	意味
Windows	サポートされているすべての Windows プラットフォーム
Win2003	Windows 2003 のみ
WinXP	Windows XP のみ
Win2000	Windows 2000 のみ
UNIX	すべての UNIX プラットフォーム
Solaris	Solaris のみ
Linux	Linux のみ

Borland サポートへの連絡

ボーランド社は各種のサポートオプションを用意しています。それらにはインターネット上の無償サービスが含まれており，大規模な情報ベースを検索したり，他の Borland 製品ユーザーからの情報を得ることができます。さらに Borland 製品のインストールに関するサポートから有償のコンサルタントレベルのサポートおよび高レベルなアシスタンスに至るまでの複数のカテゴリから，電話サポートの種類を選択できます。

Borland のサポートサービスの詳細や Borland テクニカルサポートへの問い合わせについては，Web サイト `http://support.borland.com` で地域を選択してください。

ボーランド社のサポートへの連絡にあたっては，次の情報を用意してください。

- 名前
- 会社名およびサイト ID
- 電話番号
- ユーザー ID 番号（米国のみ）
- オペレーティングシステムおよびバージョン
- Borland 製品名およびバージョン
- 適用済みのパッチまたはサービスパック
- クライアントの言語とそのバージョン（使用している場合）
- データベースとそのバージョン（使用している場合）

- 発生した問題の詳細な内容と経緯
- 問題を示すログファイル
- 発生したエラーメッセージまたは例外の詳細な内容

オンラインリソース

ネットワーク上の次のサイトから情報を得ることができます。

Web サイト http://www.borland.com/jp/

オンラインサポート http://support.borland.com（ユーザー ID が必要）

リストサーバー 電子ニュースレター（英文）を購読する場合は，次のサイトに用意されているオンライン書式を使用してください。

http://www.borland.com/products/newsletters

Web サイト

定期的に http://www.borland.com/jp/products/visibroker/index.html をチェックしてください。VisiBroker 製品チームによるホワイトペーパー，競合製品の分析，FAQ の回答，サンプルアプリケーション，最新ソフトウェア，最新のマニュアル，および新旧製品に関する情報が掲載されます。

特に，次の URL をチェックすることをお勧めします。

- http://www.borland.com/products/downloads/download_visibroker.html（最新の VisiBroker ソフトウェアおよび他のファイル）
- http://www.borland.com/techpubs（マニュアルの更新および PDF）
- http://info.borland.com/devsupport/bdp/faq/（VisiBroker の FAQ）
- http://community.borland.com（英語，開発者向けの弊社 Web ベースニュースマガジン）

Borland ニュースグループ

Borland VisiBroker を対象とした数多くのニュースグループに参加できます。VisiBroker などの Borland 製品のユーザーによるニュースグループへの参加については，http://www.borland.com/newsgroups を参照してください。

メモ これらのニュースグループはユーザーによって管理されているものであり，ボーランド社の公式サイトではありません。

第 2 章

VisiTime サービスの使い方

ここでは，OMG Time Service 仕様バージョン 1.1 を完全に実装する VisiTime サービスについて説明します。OMG Time Service 仕様は，VisiBroker に実装される 2 種類のサービスを定義します。

- **基本のタイムサービス：**時間（タイムスタンプなど）と時間間隔を表すオブジェクトを作成するインターフェースを提供します。
- **タイマーイベントサービス：**タイマーイベントハンドラのオブジェクトを管理するインターフェースを提供します。これらのオブジェクトは，ユーザー定義の時間設定に基づく時間基準のイベントを生成します。

タイムサービスの概要

OMG Time Service 仕様によれば，OMG Time Service は，現在の時刻およびそれに関連付けられているエラー予測を取得するために作成されています。さらに，タイムサービスは，イベントが発生する順番を確認してイベントを追跡する手段を提供し，時間基準のイベントトリガーまたは「アラーム」を生成し，2 つのイベントの間隔を計算します。

タイムサービスが時間を定義する方法

OMG Time Service 仕様は，協定世界時（UTC）を使って時間を定義します。UTC は，基本の時間単位に 100 ナノ秒（10^{-7} 秒）を使用し，基本時間は 1582 年 10 月 15 日 00:00:00 グリニッジ標準時です。UTC 表現では，西暦 30,000 年前後までの範囲がサポートされます。

同様に，UTC 表現は時間間隔または「相対時間」も定義します。相対時間の基本単位は，通常の時間と同様に 10^{-7} 秒です。範囲は，± 30,000 年程度です。

タイムサービスは時間を提供する基底の時間ソースを使用し，必要なすべての時間の同期を実行します。基底の時間ソースが OMG Time Service 仕様の付録 A に設定されているセキュリティ条件を満たす場合，そのタイムサービスは安全時間も提供できます。

タイムサービスのコンポーネント

タイムサービスは，UTO（Universal Time Object）と TIO（Time Interval Object）というアプリケーションで使用できる 2 種類の CORBA オブジェクトを定義します。この 2 つのオブジェクトを使用して，CORBA Time Service では次の機能を提供する必要があります。

- universal_time という UTO オブジェクトで現在の時刻と関連付けられた誤差を取得する。
- secure_universal_time　　オブジェクトで安全時間ソースの条件が満たされる場合は，UTO オブジェクトで現在と時間と関連付けられた誤差を取得する。
- new_universal_time オブジェクトという任意の時間を表す UTO オブジェクトを作成する。
- UtcT 構造体から uto_from_utc という UTO オブジェクトを作成する。
- new_interval という TIO を作成する。

Universal Time Object

UTO インターフェースは UTC 時間を格納するオブジェクトに対応し，オブジェクトの時間を操作する手段を提供します。UTO は変換に自由の利かないオブジェクトなので，格納されている時間の値は変更できません。UTO では，UTO の比較，UTO と TIO 間隔の比較，UTO オブジェクトの構成要素の取得など，基本時間で実行される操作ができます。

Time Interval Object

UTO と同様に，TIO は変換に自由の利かないオブジェクトで，時間間隔を表し，時間間隔の操作を提供します。TIO オブジェクトに保存されている間隔値を取得し，TIO と 1 つ以上の UTO のオーバーラップを特定し，TIO を UTO に変換するメソッドがあります。

タイムサービスのサービス

タイムサービスは，アプリケーションで使って操作できる時間オブジェクトを提供するだけでなく，Timer Event Service と Secure Time Service を指定します。Timer Event Service は, コールバックオブジェクトを使って応答できるイベントをタイマーアラームがトリガーする手段を提供します。Secure Time Service では，システムの指定されたユーザーだけが時間の設定や時間ソースの指定ができます。

Timer Event Service

Timer Event Service は，イベントがトリガーされたときに通知を受信するメカニズムを提供します。つまり，Timer Event Service はある種のアラームサービスを提供します。プログラムで Timer Event Service を使って CosEventComm::PushConsumer コールバックオブジェクトを登録し，アラームを設定およびキャンセルする操作を提供する特別なイベントハンドラを取得できます。アラームがオフになると，Timer Event Service はコールバックオブジェクトに通知を送信します。

Timer Event Handler オブジェクトは, 特定の時間にトリガーされるイベントに関する情報，およびイベントがトリガーされる際に実行されるアクションに関する情報を格納します。実行されるアクションは，基本的にイベントハンドラとして登録される CosEventComm::PushConsumer オブジェクトの push メソッドの呼び出しです。このメソッドは，プッシュするデータを含む CORBA::Any オブジェクトを受け取ります。データはイベントハンドラをイベントサービスに登録する際も指定します。Timer Event Handler インターフェースでは，次の操作を実行できます。

- time_set メソッドを使用して，イベントがトリガーされているかどうかを照会する。
- status メソッドを使用して，Timer Event Handler の状態を照会する。
- set_timer メソッドを使用して，イベントがアラームをトリガーする時間を設定する。

- cancel_timer メソッドを使用して，作動していないトリガーをキャンセルする。
- set_data メソッドを使用して，イベントがトリガーされたときにプッシュするデータを設定する。

アラームは，相対的または絶対的な時間定義を使って設定できます。定期的に発生するように設定することもできます。Timer Event Service インターフェースでは，Timer Event Handler の存続期間全体を操作できます。Timer Event Service インターフェースでは，次の操作を実行できます。

- register メソッドを使用して，イベントハンドラを登録し，コールバックオブジェクトとイベントデータを指定する。
- unregister メソッドを使用して，以前に登録されたイベントハンドラの登録を解除する。
- event_time メソッドを使用して，イベントがトリガーされた時間を取得する。

Secure Time Service

システムセキュリティポリシーで承認されている管理者だけが時間および時間ソースを指定できます。承認されている管理者は，安全時間を返すようにタイムサービスを設定できます。これにより，基底の時間ソースの安全を確保でき，タイムサービスインターフェースの secure_universal_time オペレーションの呼び出しは確実に安全時間を返します。基底の時間ソースが安全ではない場合，タイムサービスインターフェースで secure_universal_time オペレーションを呼び出すと CosTime::TimeUnavailable 例外が発生します。

VisiTime サービス

VisiTime サービスは，UTO（Universal Time Object）と TIO（Time Interval Object）を作成するファクトリです。

VisiTime サービスの起動

VisiTime サービスは，インストールしている VisiBroker の bin ディレクトリにある timeserv 起動プログラムで起動できます。このコマンドを実行すると，VisiTime サービスと 8 ページの「Timer Event Service」が起動します。コマンド構文は次のとおりです。

UNIX
```
timeserv [driver_options] [timeserv_options] &
```

Windows
```
timeserv [driver_options] [timeserv_options]
```

タイムサービスは，VBJ 起動プログラムを使って起動することもできます。

```
vbj [driver_options] com.borland.vbroker.CosTime.TimeServer
```

次のドライバのオプションがあります。

オプション	説明
-install <service-name>	（Windows のみ）提供される名前を使って NT サービスとしてインストールします。このオプションは，vbj を使ってタイムサービスを起動する場合は使用できません。
-remove <service-name>	（Windows のみ）この NT サービスをアンインストールします。このオプションは，vbj を使って VisiTime サービスを起動する場合は使用できません。

一般的なドライバのオプションも使用できます。詳細については，『VisiBroker® for C++ 開発者ガイド』か『VisiBroker® for Java 開発者ガイド』の共通オプションを参照してください。

次の VisiTime Service オプションがあります。

オプション	説明
-?, -h, -help, -usage	コマンドの使用方法を表示します。
-props <properties-file>	VisiTime サービスの起動では，設定ファイルとして提供されているプロパティファイルを使用します。このファイルに定義されているプロパティは，コマンドラインで同じプロパティが渡されると上書きされます。

安全な VisiTime サービスの起動

基底の時間ソースが安全で OMG Time Service 仕様の付録 A のガイドラインにしたがっている場合，VisiTime サービスは安全なタイムサービスとして起動できます。この場合，TimeService::secure_universal_time への呼び出しは成功します。ここでのセキュリティは，基底の時間ソースのセキュリティだけを指します。安全な VisiTime サービスを起動するには，次のようにします。

UNIX
```
timeserv -J-Dvbroker.time.source.secured=true &
```

Windows
```
start timeserv -J-Dvbroker.time.source.secured=true
```

VisiTime サービスのブートストラップ

クライアントアプリケーションを起動して VisiTime サービスの初期リファレンスを取得するには，次の 3 つの方法があります。

- ORBInitRef コマンドを使用する。
- ORBDefaultInitRef コマンドを使用する。
- スマートエージェントを使用する。

どちらのコマンドラインを使用する場合も，クライアントアプリケーションは ORB の resolve_initial_references メソッドを使ってタイムサービスまたは Timer Event Service を取得できます。次に例を示します。

C++
```
...
CORBA::ORB_var orb = CORBA::ORB_init (argc, argv);

// タイムサービスへのリファレンスを取得します。
CORBA::Object_var obj_t = orb->resolve_initial_references("CosTimeService");
CosTime::TimeService_var time_svc = CosTime::TimeService::_narrow (obj_t.in());

// Timer Event Service へのリファレンスを取得します。
CORBA::Object_var obj_te = orb->resolve_initial_references("CosTimerEventService");
CosTimerEvent::TimerEventService_var timer_svc =
                            CosTimerEvent::TimerEventService::_narrow (obj_te.in());
...
```

Java
```
// タイムサービスへのリファレンスを取得します。
org.omg.CosTime.TimeService timeSvc = org.omg.CosTime.TimeServiceHelper.narrow(
                                        orb.resolve_initial_references("CosTimeService"));

// Timer Event Service へのリファレンスを取得します。
org.omg.CosTimerEvent.TimerEventService timerSvc =
                           org.omg.CosTimerEvent.TimerEventServiceHelper.narrow(
                           orb.resolve_initial_references("CosTimerEventService"));
...
```

ORBInitRef を使用するブートストラップ

最も一般的な ORBInitRef の使い方は，corbaloc URL を使って初期リファレンスを指定することです。その他の URL スキーマも使用できます。たとえば，IOR 文字列やファイル URL（Java のみ）を使ってタイムサービス IOR を含むファイル名を指定します。たとえば，次のコマンドは，ポート 5566 で実行されているタイムサービスと Timer Event Service をクライアントアプリケーションにブートストラップします。

C++

```
<client_application> -ORBInitRef CosTimeService=corbaloc::<host>:5566/CosTimeService

<client_application> -ORBInitRef CosTimerEventService=corbaloc::<host>:5566/
CosTimerEventService
```

Java

```
vbj <client_application> -ORBInitRef CosTimeService=corbaloc::<host>:5566/CosTimeService

vbj <client_application> -ORBInitRef CosTimerEventService=corbaloc::<host>:5566/
CosTimerEventService
```

ORBDefaultInitRef を使用するブートストラップ

ORBInitRef と同様に，ORBDefaultInitRef も corbaloc URL を一般に使って初期リファレンスを指定します。インプリメンテーションによっては，ほかの URL スキーマも使用できます。次のコマンドは，ORBDefaultInitRef を使ってタイムサービスと Timer Event Service をアプリケーションにブートストラップします。

C++

```
<client_application> -ORBDefaultInitRef corbaloc::<host>:5566
```

Java

```
vbj <client_application> -ORBDefaultInitRef corbaloc::<host>:5566
```

クライアントアプリケーションを起動する vbj コマンドで ORBDefaultInitRef をプロパティとして指定することもできます。次のコマンドもタイムサービスをブートストラップしますが，プロパティとして ORBDefaultInitRef を指定します。

```
vbj -DORBDefaultInitRef=corbaloc::<host>:5566 <client_application>
```

スマートエージェントを使用するブートストラップ

クライアントアプリケーションでは, VisiBroker bind メソッドを使ってスマートエージェントからタイムサービスまたは Timer Event Service への初期リファレンスを取得できます。Java では，TimeServiceHelper クラスと TimerEventServiceHelper クラスを使ってバインドを実行します。メソッドを実行する場合は，接続するタイムサービスと Timer Event Service の名前を指定します(Java では, サービスをホストする ORB)。次に例を示します。

C++

```
CORBA::ORB_var orb = CORBA::ORB_init (argc, argv);

// タイムサービスへのリファレンスを取得します。
CosTime::TimeService_var time_svc = CosTime::TimeService::_bind("VBTimeService");

// Timer Event Service へのリファレンスを取得します。
CosTimerEvent::TimerEventService_var timer_svc =
                    CosTimerEvent::TimerEventService::_bind("VBTimerEventService");
```

Java

```
org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,null);

// タイムサービスへのリファレンスを取得します。
org.omg.CosTime.TimeService timeSvc = org.omg.CosTime.TimeServiceHelper.bind(orb,
                                                                "VBTimeService");

// Timer Event Service へのリファレンスを取得します。
org.omg.CosTimerEvent.TimerEventService timerSvc =
                            org.omg.CosTimerEvent.TimerEventServiceHelper.bind(orb,
                                                    "VBTimerEventService");
```

インプロセスでのタイムサービスの実行

VisiTime サービスは，インプロセスで実行するか，Java アプリケーションと同じ場所に置くことができます。アウトプロセスモードからインプロセスモードに切り替えるときにアプリケーションコードを変更する必要はありません。インプロセスのタイムサービスの有効化は，VisiBroker プロパティ vbroker.time.enableInProc で制御します。

タイムサービスがインプロセスまたはアウトプロセスのどちらの実行モードを使用している場合も，ユーザーアプリケーションは orb.resolve_initial_references("CosTimeService") と orb.resolve_initial_references("CosTimerEventService") を使ってタイムサービスと Timer Event Service への初期リファレンスを取得できます。インプロセスとリモートのタイムサービスでは，ブートストラッピングのメカニズムが異なります。インプロセスのタイムサービスで, ユーザーアプリケーションは ORBInitRef プロパティを指定することはできません。かわりに，VisiBroker プロパティ vbroker.time.enableInProc=true を有効にする必要があります。ORBInitRef を vbroker.time.enableInProc=true とともに使用すると，ORBInitRef だけが有効になります。

時間ソースの NTP サーバーサポート

デフォルトでは，VisiTime サービスのインプリメンテーションは，システム時刻を時間ソースとして使用します。または，NTP サーバーを時間ソースとして使用することもできます。これは，VisiBroker プロパティ vbroker.time.ntp.addr で制御します。

NTP サーバーアドレスとフェイルオーバーの指定

vbroker.time.ntp.addr の値は，NTP サーバーアドレスを表すカンマで区切られた 1 つ以上の文字列です。IPv4 形式と IPv6 形式のアドレスも指定できます。たとえば，次のようにして 3 つの NTP サーバーアドレスを指定します。

```
vbroker.time.ntp.addr=foo.com,[fe220::103:baaa:fbbb:fedf]:123,101.121.145.100:124
```

最初のアドレス，foo.com は，内部 DNS の検索に依存します。ポートが指定されていないので，デフォルトの NTP ポート 123 が使用されます。2 つめのアドレス，[fe220::103:baaa:fbbb:fedf]:123 は IPv6 形式のアドレスをブラケットで囲んでいます。ここで，ポートは 123 に定義されています。最後のアドレス，101.121.145.100:124 は一般的な IPv4 形式で，ポート番号 124 も指定されています。

VisiTime サービスは，最初に，一連のアドレスの 1 つめの NTP サーバーにコンタクトを試みます。アドレスが有効でサーバーが使用できる場合，NTP サーバーの時間が呼び出し元に返されます。リストの最初のサーバーを使用できない場合，インプリメンテーションは透過的に 2 番めにフェイルオーバーし，リストのサーバーから必要な時間値を取得できるまでこの処理を続けます。すべてのサーバーを使用できない場合，VisiTime サービスは呼び出し元に例外を生成します。呼び出されるメソッドにより，例外は CosTime::TimeUnavailable，または COMM_FAILURE などの CORBA システム例外になります。

VisiTime サービスの設定

VisiTime サービスは VisiBroker コンソール，コマンドラインで指定されるプロパティ，またはプロパティファイルで指定されるプロパティを使って設定できます。VisiTime サービスのために，次のプロパティが提供されています。

表 2.1　一般プロパティ

プロパティ	デフォルト値	説明
vbroker.time.name	（なし）	このタイムサービスの名前を指定します。この名前は，管理コンソールまたは Server Manager を使って特定のタイムサービスを指定するために使用します。
vbroker.time.listener.port	0	タイムサービスのリスナーポート。デフォルト値は，ランダムポートが選択される 0 です。このプロパティは，リスナーポートが Server Manager の vbroker.se.iiop_tp.scm.iiop_tp.listener.port プロパティで設定されている場合は無効です。
vbroker.time.timeRefFile	（なし）	タイムサービスの IOR が書き込まれている場所を指定します。タイムサービスがインプロセス実行モードで実行されている場合は無効です。
vbroker.time.timerEventRefFile	（なし）	タイマーイベントサービスの IOR が書き込まれている場所を指定します。このプロパティは，インプロセスタイムサービスでは無効です。
vbroker.time.enableInProc	false	Java のみ。タイムサービスをインプロセスで実行します。タイムサービス自体ではなく，タイムサービスクライアントで指定する必要があります。
vbroker.time.leapSeconds	0	時間ソースが返す時間にうるう秒を追加します。うるう秒は，協定世界時（UTC）に追加して天文時との誤差が 0.9 秒以内になるようにします。現在の値は 23 秒です（1972 年 6 月 30 日以降）。タイムサービスの時間ソースでうるう秒の修正がされていない場合はこのプロパティを使用します。

表 2.2　安全タイムサービスのプロパティ

プロパティ	デフォルト値	説明
vbroker.time.source.secured	false	時間ソースが安全であることをタイムサービスに通知します。プロパティが true の場合は，secure_universal_time の呼び出しは常に成功します。そうでない場合は，TimeUnavailable 例外が生成されます。

表 2.3　タイマーイベントサービスのプロパティ

プロパティ	デフォルト値	説明
vbroker.time.threadMax	0	Timer Event Service スレッドプールのスレッドの最大数を設定します。
vbroker.time.threadMin	5	Timer Event Service スレッドプールのスレッドの最小数を設定します。
vbroker.time.threadMaxIdle	100	アイドル状態のスレッドがプールから削除されるまでの時間を設定します。ただし，プール内のスレッド数は threadMin の値に維持されます。

表 2.4　タイムサービスロガーのプロパティ

プロパティ	デフォルト値	説明
vbroker.time.logLevel	C++: 0 Java : emerg	ログに記録されるメッセージのログレベルを指定します。デフォルト値に設定された場合は，システムを使用できないか，異常な状態の場合に，システムがメッセージを記録することを指定します。次の値を指定できます。 • emerg (0) : 何らかの異常な状態を示します。 • alert (1) : ユーザーが注意する必要がある状態です。たとえば，セキュリティが無効になっている場合です。 • crit (2) : 重大な状態です。たとえば，デバイスにエラーが発生した場合です。 • err (3) : エラー状態です。 • warning (4) : 警告状態です。トラブルシューティングの助言も表示される場合があります。 • notice (5) : 接続を開く場合など，エラーではないとしても注意する必要がある状態です。 • info (6) : 実行中のバインディングなどの情報を提供します。 • debug (7) : 開発者が理解できるデバッグ状態です。
vbroker.time.logger.output	stdout	ロガー出力を書き込むファイルの名前。デフォルトでは，画面に出力されます。
vbroker.time.logger.appName	TimeService	ログ出力に記載するアプリケーションの名前。
vbroker.log.enable	false	このサーバーのデバッグログステートメントを表示するには，このプロパティを true に設定します。デバッグログフィルタのさまざまなソース名オプションについては，『VisiBroker for C++ 開発者ガイド』の「デバッグログのプロパティ」を参照してください。

表 2.5　NTP サーバー設定のプロパティ

プロパティ	デフォルト値	説明
vbroker.time.ntp.addr	（なし）	NTP サーバーのアドレスとポートを指定します。このプロパティの名前は次のように指定します。 `addr<:port>[, addr<:port>]` addr は，myhost.com などのホスト名または IP アドレスです。IPv4 アドレスと IPv6 アドレスがサポートされます。IPv6 アドレスは，ブラケットで囲む必要があります。ポートは省略できます。指定しない場合は，デフォルトのタイムサービスポート 123 が使用されます。複数のアドレスが指定されている場合は，サーバーの 1 つとの通信が失敗すると NTP サーバーのフェイルオーバーが発生します。タイムサービスはすべてのサーバーを試してから TimeUnavailable 例外を生成します。
vbroker.time.ntp.timeout	5000	NTP サーバーからの応答を待つ時間（ミリ秒単位）。複数の NTP サーバーが指定されている場合は，タイムアウトが経過すると次のサーバーにフェイルオーバーします。

TimeService インターフェースでのタイムサービスオブジェクトの作成

VisiTime サービスインターフェース TimeService は, UTO と TIO を作成するメソッドを提供しますが，作成したオブジェクトを非アクティブ化または破棄するメソッドは提供しません。VisiBroker の TimeService インプリメンテーションは，オブジェクトの数を制限するデフォルトのサーバントベースのディスパッチメカニズムを使用します。つまりリファレンスの数がいくつであっても，要求を処理する実際のサーバントは 1 つだけです。したがって，大量のタイムサービスオブジェクト（UTO と TIO）が作成される心配はありません。UTO と TIO を作成するには，TimeService インターフェースを使用します。このオブジェクトを作成する前に, タイムサービスに解決して限定する必要があります (Java では TimeServiceHelper を使用)。次のコードに，この手順の例を示します。

C++
```
//ORB を初期化します。
CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);

//TimeService インターフェースを解決します。
CORBA::Object_var obj_t = orb->resolve_initial_references("CosTimeService");

//TimeService インターフェースを限定します。
CosTime::TimeService_var time_svc = CosTime::TimeService::_narrow (obj_t.in());
```

Java
```
import org.omg.CORBA.ORB;
import org.omg.CosTime.*;
...
//ORB を初期化します。
ORB orb = ORB.init(args, null);

//TimeService インターフェースを解決します。
org.omg.CORBA.Object obj = orb.resolve_initial_references("CosTimeService");

// ヘルパーを使って適切に限定します。
TimeService timeService = TimeServiceHelper.narrow(obj);
```

TimeService インターフェースを解決して限定すると，これを使って UTO と TIO を作成できます。

TimeService インターフェースを使用する UTO の作成

UTO オブジェクトを作成するには, TimeService の universal_time() メソッドを使用します。たとえば，次のようにします。

C++
```
CosTime::UTO_var uto = time_svc-> universal_time();
```

Java
```
UTO uto = timeService.universal_time();
```

メソッドを実行した時刻を値に持つ UTO uto オブジェクトを作成します。

new_universal_time メソッドを使用すれば，選択した相対時間（時間ソースを使って取得されていない）を含む UTO を作成することもできます。このメソッドに 3 つの引数を渡します。

- 64 ビットの時刻値。基本時間から経過した時間（100 ナノ秒単位）です。C++ では CORBA::ULongLong，Java では long データ型です。
- 時間の誤差値。
- C++ では CORBA::Short，Java では short の TdfT 値です。

次に例を示します。

C++
```
CosTime::UTO_var uto =
        time_svc-> new_universal_time((CORBA::ULongLong)10000000,0,(CORBA::Short)0);
```

Java
```
UTO uto = timeService.new_universal_time(10000000L,0,(short)0);
```

TimeService インターフェースを使用する TIO の作成

TimeService インターフェースを使って TIO を作成できます。new_interval メソッドは，CORBA::ULongLong 型（C++）または long 型（Java）の 2 つの引数を受け取ります。これらは 100 ナノ秒で表される基本時間からの時間間隔の両端です。次に例を示します。

C++
```
// 特定の間隔を表す TIO を作成します。
CosTime::TIO_var tio = time_svc->new_interval((CORBA::ULongLong)10000000,
                                              (CORBA::ULongLong)20000000);
```

Java
```
// 間隔を表す TIO を作成します。
TIO tio = _timeService.new_interval(10000000L, 20000000L);
```

タイマーイベントサービスの使い方

ここでは，Timer Event Service の解決，TimerEventHandlers の取得，TimerEventHandlers を使用するアラームの設定，以前に設定されたアラームのキャンセル，TimerEventHandler の登録解除について説明します。

TimerEventHandlers を作成して利用する前に，PushConsumer オブジェクトを提供する ORB の標準イベントサービスだけでなく TimerEventService 自体にも解決する必要があります。次に例を示します。

C++
```
//ORB を初期化します。
CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);

//TimerEventService を解決します。
CORBA::Object_var obj_t = orb->resolve_initial_references("CosTimerEventService");
CosTime::TimerEventService_var time_evsvc =
                            CosTime:: TimerEventService::_narrow (obj_t.in());

//EventService に解決します。
CORBA::Object_var obj_ev = orb->resolve_initial_references("EventService");
CosEventChannelAdmin::EventChannel_var channel =
                      CosEventChannelAdmin::EventChannel::_narrow(obj_ev.in());
```

Java
```
import org.omg.CORBA.*;
import org.omg.CosEventComm.*;
import org.omg.CosEventChannelAdmin.*;
import org.omg.CosTime.*;
import org.omg.CosTimerEvent.*;
import org.omg.TimeBase.*;
...

//ORB を初期化します。
ORB orb = ORB.init(args, null);

//TimerEventService を解決します。
TimerEventService timerEventService=TimerEventServiceHelper.narrow(
                           _orb.resolve_initial_references("CosTimerEventService"));

//EventService に解決します。
EventChannel channel =
         EventChannelHelper.narrow(_orb.resolve_initial_references("EventService"));
```

TimerEventHandlers の作成

Timer Event Service では，イベントデータを提供する CORBA::Any とともに CosEventComm::PushConsumer を登録するオペレーションを提供できます。内部的には，TimerEventHandler が作成され，イベントデータと PushConsumer が関連付けられます。イベントデータはいつでも変更できますが，PushConsumer は TimerEventHandler に固定して関連付けられ，変更はできません。

Timer Event Service とイベントサービスを解決し，イベントサービスからチャネルを取得すると，イベントハンドラインプリメンテーションを作成できます。それには，次の 6 つの手順を実行する必要があります。

1 イベントデータをコンシューマにプッシュする ProxyPushSupplier オブジェクトを作成します。

2 イベントデータを受け取る PushConsumer オブジェクトを作成します。

3 ProxyPushSupplier をその PushConsumer と関連付けます。

4 イベントチャネルから ProxyPushConsumer オブジェクトを取得します。これは Timer Event Service に登録されるオブジェクトです。

5 新しい CORBA::Any でイベントデータを作成します。

6 引数として ProxyPushConsumer オブジェクトと CORBA::Any オブジェクトを使用し，Timer Event Service の register メソッドを実行してイベントハンドラを作成します。

次の表は，上の各ステップで実行されるソースコードです。

表 2.6 C++ コード

ステップ	コード

1	//ProxyPushSupplier を作成します。
	CosEventChannelAdmin::ConsumerAdmin_var cns_adm = channel->for_consumers();
	CosEventChannelAdmin::ProxyPushSupplier_var pushSupplier =
	cns_adm->obtain_push_supplier();

2	//PushConsumer を作成します。この PushView は PushConsumer のインプリメンテーションです PushView* view = new PushView();
3	//PushConsumer を接続します pushSupplier->connect_push_consumer(view->_this());
4	// イベントチャネルから ProxyPushConsumer を取得します。 CosEventChannelAdmin::SupplierAdmin_var sup_adm = channel->for_suppliers(); CosEventChannelAdmin::ProxyPushConsumer_var proxy = sup_adm- >obtain_push_consumer();
5	// イベントがトリガーされたときに受け取るデータを作成します。 CORBA::Any any ; any <<="my data";
6	//TimerEventHandler を取得するための PushConsumer とイベントデータを登録します。 CosTimerEvent:: TimerEventHandler_var eventHandler = time_evsvc- >register(proxy,any);

表 2.7 Java コード

ステップ	コード

1	//ProxyPushSupplier を作成します。
	ProxyPushSupplier pushSupplier =
	channel.for_consumers().obtain_push_supplier();

2	//PushConsumer を作成します。この PushView は PushConsumer のインプリメンテーションです PushView view = new PushView();

3	//PushConsumer を接続します
	pushSupplier.connect_push_consumer(view._this(orb));

4	// イベントチャネルから ProxyPushConsumer を取得します。 ProxyPushConsumer proxy = channel.for_suppliers().obtain_push_consumer();
5	// イベントがトリガーされたときに受け取るデータを作成します。 Any any = orb.create_any(); Any.insert_string("my data");
6	//TimerEventHandler を取得するための PushConsumer とイベントデータを登録します。 TimerEventHandler eventHandler = timerEventService.register(proxy,any);

TimerEventHandler のアラームの設定

新しく作成した TimerEventHandler を使用するには，EventTimer インターフェースを使ってアラームを設定します。アラームを設定するには set_timer メソッドを使用します。このメソッドは，アラームのタイプと UTO オブジェクトの 2 つの引数を受け取ります。使用できるアラームのタイプは 3 つあります。

- TTAbsolute：アラームは UTO に指定される絶対時間でトリガーされます。
- TTRelative：アラームは UTO で現在の時刻に対して相対的にトリガーされます（UTO は時間基準ではなく現在の絶対時間からの時間を表します）。
- TTPeriodic：アラームは UTO で指定される相対時間ごとに定期的に発生します。

アラームを設定するには，次のようにします。

1. 16 ページの「TimerEventHandlers の作成」オブジェクトを作成します。
2. アラームのトリガーに使用する新しい UTO を作成します。
3. イベントハンドラの set_timer メソッドを使ってアラームを設定します。

たとえば，次のコードは eventHandler という TimerEventHandler オブジェクトのアラームを設定します。

C++
```
// 相対時間を表す UTO を作成します。
CosTime::UTO_var uto =
        time_svc->new_universal_time((CORBA::ULongLong)10000000,0,(CORBA::Short)0);

// 定期的なタイマーを TimerEventHandler に設定します。このアラームは 1 秒
// (10000000/10000) が経過するたびにトリガーされ，イベントデータは事前に登録された
//PushConsumer にプッシュされます。
eventHandler->set_timer(CosTimerEvent::TTPeriodic,uto);
```

Java
```
// 相対時間を表す UTO を作成します。
UTO uto = timeService.new_universal_time(10000000L,0,(short)0);

// 定期的なタイマーを TimerEventHandler に設定します。このアラームは 1 秒
// (10000000/10000) が経過するたびにトリガーされ，イベントデータは事前に登録された
//PushConsumer にプッシュされます。
eventHandler.set_timer(TimeType.TTPeriodic,uto);
```

メモ Timer Event Service でアラームを設定できる相対間隔の最小値は 1 ミリ秒です。1 ミリ秒より小さい値は透過的に 1 ミリ秒に変換されます。

タイマーのキャンセルと TimerEventHandler の登録解除

イベントハンドラのタイマーをキャンセルするには，ハンドラの cancel_timer メソッドを実行するだけです。

C++
```
eventHandler->cancel_timer();
```

Java
```
eventHandler.cancel_timer();
```

イベントハンドラを完全に登録解除するには，イベントサービスの unregister メソッドを呼び出します。

C++
```
eventService->unregister(eventHandler);
```

Java
```
eventService.unregister(eventHandler);
```

フレンドリなタイムオブジェクト

64 ビットの時間表現を人が読み取ることができる年，月，日などの部分に変換する（その逆も可）フレンドリなインターフェースを持つ TimeI オブジェクトです。TimeI オブジェクトは，変換オブジェクトの表現として表示されます。一般的な使い方は，FriendlyTime::TimeService::time() オペレーションを使ってオブジェクトを作成します。これにより，時間が 0 に設定された TimeI オブジェクトが作成されます。この後，_set オペレーションを使用してさまざまな値の属性を設定できます。最後に属性時間を使って対応する TimeT 値を取得します。

逆に，時間属性で任意の TimeT 値を設定し，該当する属性から年，月などを取得できます。

フレンドリな時間オブジェクトの IDL は，次のとおりです。

```
module FriendlyTime {
 interface TimeI {
 attribute YearT year;
 attribute MonthT month;
 attribute DayT day;
 attribute HourT hour;
 attribute MinuteT minute;
 attribute SecondT second;
 attribute MicrosecondT microsecond;
 attribute TimeBase::TimeT time;
 void reset(); // すべての属性を 0 に設定します。
};
```

次のコードでは，フレンドリな時間オブジェクトを使用します。

C++

```
//ORB を初期化します。
CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);

//FriendlyTimeService を解決します。
CORBA::Object_var obj_t = orb->resolve_initial_references("CosTimeService");
FriendlyTime::TimeService_var time_svc = FriendlyTime::TimeService::_narrow
(obj_t.in());

//FriendlyTime から TimeI オブジェクトを取得します。
FriendlyTime::TimeI_var timeI = time_svc->time();

// 現在の時刻を UTO で取得します。
CosTime::UTO_var uto = time_svc-> universal_time();

//TimeI オブジェクトに現在の時刻を設定します。
timeI->time(uto->time());

//TimeI オブジェクトからさまざまな属性を人が読み取ることができる形式で取得し，標準出力
で出力します
//
cout << " Year is :" << timeI->year() << endl;
cout << " Month is :" << timeI->month() << endl;
cout << " Day is :" << timeI->day() << endl;
cout << " Hour is :" << timeI->hour() << endl;
cout << " Minute is :" << timeI->minute() << endl;
cout << " Second is :" << timeI->second() << endl;
cout << " MicroSecond is :" << timeI->microsecond() << endl;
```

Java

```
import org.omg.CORBA.ORB;
import org.omg.CosTime.*;
...
//ORB を初期化します。
ORB orb = ORB.init(args, null);

//FriendlyTimeService を解決します。
org.omg.FriendlyTime.TimeService friendlyTs =
      org.omg.FriendlyTime.TimeServiceHelper.narrow(
```

```
                        _orb.resolve_initial_references("CosTimeService"));

//FriendlyTime から TimeI オブジェクトを取得します。
org.omg.FriendlyTime.TimeI timeI = friendlyTs.time();

// 現在の時刻を UTO で取得します。
UTO uto = friendlyTs.universal_time();

//TimeI オブジェクトに現在の時刻を設定します。
timeI.time(uto.time());

//TimeI オブジェクトからさまざまな属性を人が読み取ることができる形式で取得し，標準出力
で出力します
System.out.println("Year is :"+ timeI.year());
System.out.println("Month is :"+ timeI.month());
System.out.println("Day is :"+ timeI.day());
System.out.println("Hour is :"+ timeI.hour());
System.out.println("Minute is :"+ timeI.minute());
System.out.println("Second is :"+ timeI.second());
System.out.println("MicroSecond is :"+ timeI.microsecond());
```

索引

ふ

へ

ま