

VisiTime Guide

Borland VisiBroker[®] 7.0

Borland[®]

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

Refer to the file [deploy.html](#) for a complete list of files that you can distribute in accordance with the License Statement and Limited Warranty.

Borland Software Corporation may have patents and/or pending patent applications covering subject matter in this document. Please refer to the product CD or the About dialog box for the list of applicable patents. The furnishing of this document does not give you any license to these patents.

Copyright 1992–2006 Borland Software Corporation. All rights reserved. All Borland brand and product names are trademarks or registered trademarks of Borland Software Corporation in the United States and other countries. All other marks are the property of their respective owners.

Microsoft, the .NET logo, and Visual Studio are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

For third-party conditions and disclaimers, see the Release Notes on your product CD.

VB70VisiTime
March 2006
PDF

Contents

Chapter 1		
Introduction to Borland VisiBroker	1	
VisiBroker Overview	1	
VisiBroker features.	2	
VisiBroker Documentation	2	
Accessing VisiBroker online help topics in the standalone Help Viewer	3	
Accessing VisiBroker online help topics from within the VisiBroker Console.	3	
Documentation conventions	4	
Platform conventions	4	
Contacting Borland support.	4	
Online resources.	5	
World Wide Web.	5	
Borland newsgroups	5	
Chapter 2		
Using the VisiTime Service	7	
Time Service Overview	7	
How the Time Service Defines Time	7	
Time Service Components	8	
Universal Time Object	8	
Time Interval Object	8	
Time Service Services.	8	
Timer Event Service	8	
Secure Time Service	9	
		VisiTime Service. 9
		Starting the VisiTime Service. 9
		Starting Secure VisiTime Service 10
		Bootstrapping the VisiTime Service 10
		Bootstrapping Using ORBInitRef 11
		Bootstrapping Using ORBDefaultInitRef 11
		Bootstrapping Using the SmartAgent 12
		Running the Time Service In-process 12
		NTP Server Support for Time Source 12
		Specifying NTP Server Addresses and Failover 13
		Configuring the VisiTime Service. 13
		Creating Time Service Objects with the TimeService interface. 15
		Creating UTOs using the TimeService interface 16
		Creating TIOs using the TimeService interface 16
		Using the Timer Event Service 16
		Creating TimerEventHandlers. 17
		Setting Alarms for TimerEventHandlers. 18
		Cancelling a Timer and Unregistering a TimerEventHandler 19
		Friendly Time Object 19
		Index 21

Introduction to Borland VisiBroker

For the CORBA developer, Borland provides *VisiBroker for Java*, *VisiBroker for C++*, and *VisiBroker for .NET* to leverage the industry-leading VisiBroker Object Request Broker (ORB). These three facets of VisiBroker are implementations of the CORBA 2.6 specification.

VisiBroker Overview

VisiBroker is for distributed deployments that require CORBA to communicate between both Java and non-Java objects. It is available on a wide range of platforms (hardware, operating systems, compilers and JDKs). VisiBroker solves all the problems normally associated with distributed systems in a heterogeneous environment.

VisiBroker includes:

- VisiBroker for Java, VisiBroker for C++, and VisiBroker for .NET, three implementations of the industry-leading Object Request Broker.
- VisiNaming Service, a complete implementation of the Interoperable Naming Specification in version 1.3.
- GateKeeper, a proxy server for managing connections to CORBA Servers behind firewalls.
- VisiBroker Console, a GUI tool for easily managing a CORBA environment.
- Common Object Services such as VisiNotify (implementation of Notification Service Specification), VisiTransact (implementation of Transaction Service Specification), VisiTelcoLog (implementation of Telecom Logging Service Specification), VisiTime (implementation of Time Service Specification), and VisiSecure.

VisiBroker features

VisiBroker offers the following features:

- “Out-of-the-box” security and web connectivity.
- Seamless integration to the J2EE Platform, allowing CORBA clients direct access to EJBs.
- A robust Naming Service (VisiNaming), with caching, persistent storage, and replication for high availability.
- Automatic client failover to backup servers if primary server is unreachable.
- Load distribution across a cluster of CORBA servers.
- Full compliance with the OMG's CORBA 2.6 Specification.
- Integration with the Borland JBuilder integrated development environment.
- Enhanced integration with other Borland products including Borland AppServer.

VisiBroker Documentation

The VisiBroker documentation set includes the following:

- Borland *VisiBroker Installation Guide*—describes how to install VisiBroker on your network. It is written for system administrators who are familiar with Windows or UNIX operating systems.
- Borland *Security Guide*—describes Borland's framework for securing VisiBroker, including VisiSecure for VisiBroker for Java and VisiBroker for C++.
- Borland *VisiBroker for Java Developer's Guide*—describes how to develop VisiBroker applications in Java. It familiarizes you with configuration and management of the Visibroker ORB and how to use the programming tools. Also described is the IDL compiler, the Smart Agent, the Location, Naming and Event Services, the Object Activation Daemon (OAD), the Quality of Service (QoS), the Interface Repository, and the Interface Repository, and Web Service Support.
- Borland *VisiBroker for C++ Developer's Guide*—describes how to develop VisiBroker applications in C++. It familiarizes you with configuration and management of the Visibroker ORB and how to use the programming tools. Also described is the IDL compiler, the Smart Agent, the Location, Naming and Event Services, the OAD, the QoS, Pluggable Transport Interface, RT CORBA Extensions, and Web Service Support.
- Borland *VisiBroker for .NET Developer's Guide*—describes how to develop VisiBroker applications in a .NET environment.
- Borland *VisiBroker for C++ API Reference*—provides a description of the classes and interfaces supplied with VisiBroker for C++.
- Borland *VisiBroker VisiTime Guide*—describes Borland's implementation of the OMG Time Service specification.
- Borland *VisiBroker VisiNotify Guide*—describes Borland's implementation of the OMG Notification Service specification and how to use the major features of the notification messaging framework, in particular, the Quality of Service (QoS) properties, Filtering, and Publish/Subscribe Adapter (PSA).
- Borland *VisiBroker VisiTransact Guide*—describes Borland's implementation of the OMG Object Transaction Service specification and the Borland Integrated Transaction Service components.

- Borland *VisiBroker VisiTelcoLog Guide*—describes Borland's implementation of the OMG Telecom Log Service specification.
- Borland *VisiBroker GateKeeper Guide*—describes how to use the VisiBroker GateKeeper to enable VisiBroker clients to communicate with servers across networks, while still conforming to the security restrictions imposed by web browsers and firewalls.

The documentation is typically accessed through the Help Viewer installed with VisiBroker. You can choose to view help from the standalone Help Viewer or from within a VisiBroker Console. Both methods launch the Help Viewer in a separate window and give you access to the main Help Viewer toolbar for navigation and printing, as well as access to a navigation pane. The Help Viewer navigation pane includes a table of contents for all VisiBroker books and reference documentation, a thorough index, and a comprehensive search page.

Important Updates to the product documentation, as well as PDF versions, are available on the web at <http://www.borland.com/techpubs>.

Accessing VisiBroker online help topics in the standalone Help Viewer

To access the online help through the standalone Help Viewer on a machine where the product is installed, use one of the following methods:

- Windows**
- Choose Start|Programs|Borland Deployment Platform|Help Topics
 - or, open the Command Prompt and go to the product installation `\bin` directory, then type the following command:
- ```
help
```
- UNIX**
- Open a command shell and go to the product installation `/bin` directory, then enter the command:
- ```
help
```
- Tip** During installation on UNIX systems, the default is to not include an entry for `bin` in your `PATH`. If you did not choose the custom install option and modify the default for `PATH` entry, and you do not have an entry for current directory in your `PATH`, use `./help` to start the help viewer.

Accessing VisiBroker online help topics from within the VisiBroker Console

To access the online help from within the VisiBroker Console, choose Help|Help Topics.

The Help menu also contains shortcuts to specific documents within the online help. When you select one of these shortcuts, the Help Topics viewer is launched and the item selected from the Help menu is displayed.

Documentation conventions

The documentation for VisiBroker uses the typefaces and symbols described below to indicate special text:

Table 1.1 Documentation conventions

Convention	Used for
<i>italics</i>	Used for new terms and book titles.
<code>computer</code>	Information that the user or application provides, sample command lines and code.
bold computer	In text, bold indicates information the user types in. In code samples, bold highlights important statements.
[]	Optional items.
...	Previous argument that can be repeated.
	Two mutually exclusive choices.

Platform conventions

The VisiBroker documentation uses the following symbols to indicate platform-specific information:

Table 1.2 Platform conventions

Symbol	Indicates
Windows	All supported Windows platforms.
Win2003	Windows 2003 only
WinXP	Windows XP only
Win2000	Windows 2000 only
UNIX	UNIX platforms
Solaris	Solaris only
Linux	Linux only

Contacting Borland support

Borland offers a variety of support options. These include free services on the Internet where you can search our extensive information base and connect with other users of Borland products. In addition, you can choose from several categories of telephone support, ranging from support on installation of Borland products to fee-based, consultant-level support and detailed assistance.

For more information about Borland's support services or contacting Borland Technical Support, please see our web site at: <http://support.borland.com> and select your geographic region.

When contacting Borland's support, be prepared to provide the following information:

- Name
- Company and site ID
- Telephone number
- Your Access ID number (U.S.A. only)
- Operating system and version
- Borland product name and version
- Any patches or service packs applied

- Client language and version (if applicable)
- Database and version (if applicable)
- Detailed description and history of the problem
- Any log files which indicate the problem
- Details of any error messages or exceptions raised

Online resources

You can get information from any of these online sources:

World Wide Web	http://www.borland.com
Online Support	http://support.borland.com (access ID required)
Listserv	To subscribe to electronic newsletters, use the online form at: http://www.borland.com/products/newsletters

World Wide Web

Check <http://www.borland.com/bes> regularly. The VisiBroker Product Team posts white papers, competitive analyses, answers to FAQs, sample applications, updated software, updated documentation, and information about new and existing products.

You may want to check these URLs in particular:

- http://www.borland.com/products/downloads/download_visibroker.html (updated VisiBroker software and other files)
- <http://www.borland.com/techpubs> (documentation updates and PDFs)
- <http://info.borland.com/devsupport/bdp/faq/> (VisiBroker FAQs)
- <http://community.borland.com> (contains our web-based news magazine for developers)

Borland newsgroups

You can participate in many threaded discussion groups devoted to the Borland VisiBroker. Visit <http://www.borland.com/newsgroups> for information about joining user-supported newsgroups for VisiBroker and other Borland products.

Note These newsgroups are maintained by users and are not official Borland sites.

Using the VisiTime Service

This section describes the VisiTime Service, a complete implementation of the OMG Time Service Specification, Version 1.1. The OMG Time Service specification defines two types of services which are implemented in VisiBroker:

- **Basic Time Service:** provides an interface to create objects representing time (a time stamp, for example) and intervals of time.
- **Timer Event Service:** provides an interface to manage Timer Event Handler objects. These objects are used to generate time based events based on user defined time settings.

Time Service Overview

According to the OMG Time Service Specification, the OMG Time Service was created to allow a user to obtain the current time as well as an error estimate associated with it. Additionally, the Time Service was to provide a means of tracking events by ascertaining the order in which events occur, generate time-based event triggers or “alarms”, and compute the interval between two events.

How the Time Service Defines Time

The OMG Time Service Specification defines time using the Universal Time Coordinated (UTC) representation. The UTC representation uses hundreds of nanoseconds (10^{-7} seconds) as its basic unit of time, with its base time set at 15 October 1582 00:00:00 GMT. A range of approximately 30,000 years A.D. is supported by the UTC representation.

Similarly, the UTC representation defines a intervals of time or “relative time”. Like regular time, the basic unit of a relative time is 10^{-7} seconds. Ranges can span approximately plus-or-minus 30,000 years.

The Time Service relies on the presence of an underlying time source that provides the time and performs any necessary time synchronization. If the underlying time source meets the security criteria set out in Appendix A of the OMG Time Service Specification, then the Time Service is able to provide secure time as well.

Time Service Components

The Time Service defines two types of CORBA objects that can be used by applications. These objects are the Universal Time Object (UTO), and the Time Interval Object (TIO). Utilizing these two objects, a CORBA Time Service must provide for the ability to:

- Getting the current time with associated inaccuracy in a UTO object called `universal_time`.
- Getting the current time and associated inaccuracy in a UTO object if the criteria for secure time source can be met via the `secure_universal_time` object.
- Creating a UTO object to represent arbitrary time called a `new_universal_time` object.
- Creating a UTO object from UtcT structure, an object called `uto_from_utc`.
- Creating a TIO, known as a `new_interval`.

Universal Time Object

The UTO interface corresponds to an object that contains UTC time and provides means to manipulate time in that object. UTO is an immutable object; it does not allow modifying the value of time contained in it. A UTO also provides for operations to be performed on basic time, such as comparing UTOs, comparing a UTO to a TIO interval, and getting the constituent parts of the UTO object.

Time Interval Object

Like a UTO, a TIO is an immutable object that represents a time interval and provides operations on time intervals. Methods are provided to get the interval value stored in the TIO object, determine overlapping between a TIO and one or more UTOs, and convert a TIO into a UTO.

Time Service Services

In addition to providing time objects that can be manipulated and used by applications, the Time Service also specifies a Timer Event Service and a Secure Time Service. The Timer Event Service provides a means for timer alarms to trigger events, which can be responded to using callback objects. The Secure Time Service allows only specified users of the system to set the time and/or specify the source of time.

Timer Event Service

The Timer Event Service provides a mechanism by which you can receive notifications when an event gets triggered. In other words, Timer Event Service provides a kind of alarm service. Your programs can register a `CosEventComm::PushConsumer` callback object with the Timer Event Service and obtain a special event handler object that provides operations to set and cancel alarms. When an alarm goes off, the Timer Event Service sends a notification to the callback object.

A Timer Event Handler object holds information about an event that is to be triggered at a specific time and the action to be taken when the event is triggered. The action taken is basically a call on the `push` method on the `CosEventComm::PushConsumer` object registered as the event handler. This method takes a `CORBA::Any` which contains the data to be pushed (the data is also specified when the event handler is registered with the event service). The following operations are provided by the Timer Event Handler interface:

- Querying whether an event has been triggered with the `time_set` method.
- Querying the status of the Timer Event Handler with the `status` method.
- Setting the time for an event to trigger an alarm with the `set_timer` method.

- Cancelling a trigger that has yet to go off with the `cancel_timer` method.
- Setting the data to be pushed when the event is triggered with the `set_data` method.

Alarms can be set using absolute or relative time definitions. They can also be set to occur periodically. The Timer Event Service interface provides operations for the complete lifecycle for the Timer Event Handler. The following operations are provided by the Timer Event Service interface:

- Registering an event handler and specify the callback object and the event data with the `register` method.
- Un-registering a previously registered event handler with the `unregister` method.
- Getting the time at which an event was triggered with the `event_time` method.

Secure Time Service

Only administrators authorized by the system security policy may set the time and specify the source of time. Once this is guaranteed the administrator can configure the Time Service to return secured time. With this in place it can be safely assumed that the underlying time source is secured and calling a `secure_universal_time` operation on the Time Service interface will return a secured time. If the underlying time source is not secured, a `CosTime::TimeUnavailable` exception will be raised upon invocation of the `secure_universal_time` operation on the Time Service interface.

VisiTime Service

The VisiTime Service is a factory for creating Universal Time Objects and Time Interval Objects.

Starting the VisiTime Service

The VisiTime Service can be started by using the `timeserv` launcher located in the `bin` directory of your VisiBroker installation. Running this command starts both the VisiTime Service and Timer Event Service. The command syntax is:

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

You can also start the Time Service using the VBJ launcher:

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

The following driver options are available:

Option	Description
<code>-install <service-name></code>	(Windows only) Install as an NT service using the name provided. This option cannot be used when starting the Time Service using <code>vbj</code> .
<code>-remove <service-name></code>	(Windows only) Uninstalls this NT service. This option cannot be used when starting the VisiTime Service using <code>vbj</code> .

The general driver options are also available. See *VisiBroker for Java Developer's Guide* or *VisiBroker for C++ Developer's Guide* for more information.

The following VisiTime Service options are available:

Option	Description
-?, -h, -help, -usage	Print usage information.
-props <properties-file>	Use the supplied properties file as the configuration file when starting up the VisiTime Service. Note that a property defined in this file will get overridden if the same property is also passed on the command line.

Starting Secure VisiTime Service

When the underlying time source is secure and follows the guidelines given in Appendix A of the OMG Time Service specification, then the VisiTime Service can be started as a secure Time Service. Calls to `TimeService::secure_universal_time` would succeed in this case. Note that here security only refers to the security of the underlying time source. To start a secured VisiTime Service:

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

Bootstrapping the VisiTime Service

There are three ways to start a client application to get the initial reference to the VisiTime Service. These are:

- Using the `ORBInitRef` command-line option.
- Using the `ORBDefaultInitRef` command-line option.
- Using the Smart Agent.

When using either of the command-line options, client applications can make use of the ORB's `resolve_initial_references` method to obtain the Time Service or the Timer Event Service. For example:

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

// Get reference to Time Service
CORBA::Object_var obj_t = orb->resolve_initial_references("CosTimeService");
CosTime::TimeService_var time_svc = CosTime::TimeService::_narrow (obj_t.in());

// Get reference to Timer Event Service
CORBA::Object_var obj_te = orb-
>resolve_initial_references("CosTimerEventService");
CosTimerEvent::TimerEventService_var timer_svc =
    CosTimerEvent::TimerEventService::_narrow
(obj_te.in());
...
```

```

Java // Get reference to Time Service
      org.omg.CosTime.TimeService timeSvc = org.omg.CosTime.TimeServiceHelper.narrow(

      orb.resolve_initial_references("CosTimeService"));

      // Get reference to Timer Event Service
      org.omg.CosTimerEvent.TimerEventService timerSvc =

      org.omg.CosTimerEvent.TimerEventServiceHelper.narrow(

      orb.resolve_initial_references("CosTimerEventService"));
      ...

```

Bootstrapping Using ORBInitRef

The most common usage scenario for `ORBInitRef` is to use a `corbaloc` URL to specify the initial reference. Other URL schemes are also possible. For example, using the IOR string or the file URL (Java only) to specify the name of the file containing Time Service IOR. For example, the following commands bootstrap the Time Service and Timer Event Service running on port 5566 to the client application:

```

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

```

Bootstrapping Using ORBDefaultInitRef

Like `ORBInitRef`, `ORBDefaultInitRef` commonly uses `corbaloc` URLs to specify initial references. Other URL schemes are valid as well, depending on your implementation. The following command bootstraps both the Time Service and the Timer Event Service to the client application, using `ORBDefaultInitRef`:

```

C++ <client_application> -ORBDefaultInitRef corbaloc::<host>:5566
Java vbj <client_application> -ORBDefaultInitRef corbaloc::<host>:5566

```

You can also specify the `ORBDefaultInitRef` as a property with the `vbj` command starting the client application. The following command also bootstraps the Time Service, but specifies `ORBDefaultInitRef` as a property:

```

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

```

Bootstrapping Using the SmartAgent

Client applications can also make use of the VisiBroker `bind` method to get the initial reference to the Time Service and the Timer Event Service from the SmartAgent. In Java the `TimeServiceHelper` and `TimerEventServiceHelper` classes are used to perform the bind. When executing the method, you specify the name of the Time Service and Timer Event Service to which you're connecting (and in Java, the ORB hosting them). For example:

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

        // Get reference to Time Service
        CosTime::TimeService_var time_svc =
        CosTime::TimeService::_bind("VBTimeService");

        // Get reference to Timer Event Service
        CosTimerEvent::TimerEventService_var timer_svc =

        CosTimerEvent::TimerEventService::_bind("VBTimerEventService");
Java    org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,null);

        // Get reference to Time Service
        org.omg.CosTime.TimeService timeSvc =
        org.omg.CosTime.TimeServiceHelper.bind(orb,

        "VBTimeService");

        // Get reference to Timer Event Service
        org.omg.CosTimerEvent.TimerEventService timerSvc =

        org.omg.CosTimerEvent.TimerEventServiceHelper.bind(orb,

        "VBTimerEventService");
```

Running the Time Service In-process

The VisiTime service has the ability to run in-process or co-located with Java applications. You need not make any application code changes when switching from out-of-process to in-process mode. Enabling the in-process Time Service is controlled through the VisiBroker property `vbroker.time.enableInProc`.

Regardless of whether the Time Service is using in-process or out-of-process execution mode, user applications will use `orb.resolve_initial_references("CosTimeService")` and `orb.resolve_initial_references("CosTimerEventService")` to obtain initial references to the Time Service and Timer Event Service respectively. There would be a difference in the bootstrapping mechanism for in-process and remote Time Service respectively. User applications should not specify the `ORBInitRef` property with in-process Time Service. Instead, they must enable the VisiBroker property `vbroker.time.enableInProc=true`. If `ORBInitRef` is used together with `vbroker.time.enableInProc=true`, only `ORBInitRef` will take effect.

NTP Server Support for Time Source

By default, the VisiTime Service implementation uses the System Time as the Time Source. Alternatively it can be configured to use a NTP Server as a Time Source. This is controlled through the VisiBroker property `vbroker.time.ntp.addr`.

Specifying NTP Server Addresses and Failover

The value for the `vbroker.time.ntp.addr` can be one or a sequence of comma-separated strings representing the NTP Server addresses. Both IPv4 and IPv6 format addresses can be specified as well. For example, consider three NTP server addresses given here:

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

The first address, `foo.com`, relies on the internal DNS lookup. Since no port is specified, the default NTP port `123` is used. The second entry, `[fe220::103:baaa:fbbb:fedf]:123`, is an IPv6 format address enclosed in square brackets. Here, the port is defined specifically as `123`. The final entry, `101.121.145.100:124` is the familiar IPv4 format, with the port number `124` specified as well.

The VisiTime Service will first try to contact the first NTP Server in the sequence. If the address is valid and the server is available, the time of the NTP Server will be returned to the caller. Assuming that the first server in the list was not available, the implementation will transparently fail over to the second in the list and so on until it retrieves the required time value from one of the Server in the list. If all of the Servers are unreachable, VisiTime Service will throw an exception to the caller. Depending on the method called, the exception can be either `CosTime::TimeUnavailable` or a CORBA system exception such as `COMM_FAILURE`.

Configuring the VisiTime Service

The VisiTime Service can be configured using the VisiBroker Console, using properties specified on the command line, or using properties specified in a properties file. The following properties are provided for the VisiTime Service.

Table 2.1 General Properties

Property	Default	Description
<code>vbroker.time.name</code>	(none)	Specifies a name for this Time Service. This name is used to identify a particular Time Service in the Console or through Server Manager.
<code>vbroker.time.listener.port</code>	0	The listener port for the Time Service. The default value of 0 means any random port will be picked. This property does not take effect if the listener port is set through the Server Manager's <code>vbroker.se.iiop_tp.scm.iiop_tp.listener.port</code> property.
<code>vbroker.time.timeRefFile</code>	(none)	Specifies the name of file where Time Service IOR is written. Not effective when the Time Service is run in in-process execution mode.
<code>vbroker.time.timerEventRefFile</code>	(none)	Specifies the name of file where the Timer Event Service IOR is written. This property is not effective for in-process Time Service.
<code>vbroker.time.enableInProc</code>	false	Java only. Run the Time Service as in-process. It should be specified on the Time Service client and not the Time Service itself.
<code>vbroker.time.leapSeconds</code>	0	Adds leap seconds to the time returned by the Time Source. A leap second is a second added to Coordinated Universal Time (UTC) to make it agree with astronomical time to within 0.9 second. The current value is 23 seconds (since June 30, 1972). Use this property in cases when the time source attached to Time Service is not corrected for leap seconds.

Table 2.2 Secure Time Service Properties

Property	Default	Description
vbroker.time.source.secured	false	Tells the Time Service that the Time Source is a secured one. When this property is <code>true</code> , a call to <code>secure_universal_time</code> will always succeed. Otherwise, it throws the <code>TimeUnavailable</code> exception.

Table 2.3 Timer Event Service Properties

Property	Default	Description
vbroker.time.threadMax	0	Sets the maximum number of threads in the Timer Event Service thread pool.
vbroker.time.threadMin	5	Sets the minimum number of threads in the Timer Event Service thread pool.
vbroker.time.threadMaxIdle	100	Sets the time in seconds after which an idle thread will be removed from the pool. However, the number of threads in the pool will be kept to the value of <code>threadMin</code> .

Table 2.4 Time Service Logger Properties

Property	Default	Description
vbroker.time.logLevel	C++: 0 Java: emerg	Specifies the logging level of message that will be logged. When set to the default value the system logs messages when the system is unusable, or in a panic condition. Acceptable values are: <ul style="list-style-type: none"> ■ emerg (0): indicates some panic condition. ■ alert (1): a condition that requires user attention—for example, if security has been disabled. ■ crit (2): critical conditions, such as a device error. ■ err (3): error conditions. ■ warning (4): warning conditions—these may accompany some troubleshooting advice. ■ notice (5): conditions that are not errors but may require some attention, such as upon the opening of a connection. ■ info (6): informational, such as binding in progress. ■ debug (7): debug conditions understood by developers.
vbroker.time.logger.output	stdout	The name of the file where the logger output is written. Default is to print to screen.
vbroker.time.logger.appName	TimeService	The name of the application to appear in the log output.
vbroker.log.enable	false	To see the debug log statements from this service, set this property to <code>true</code> . For the various source names options for debug log filtering, see the Debug Logging properties section of the <i>VisiBroker for C++ Developer's Guide</i> .

Table 2.5 NTP Server Configuration Properties

Property	Default	Description
vbroker.time.ntp.addr	(none)	Specifies the NTP server's address and port. The value for this property is specified as follows: <pre>addr<:port>[, addr<:port>]</pre> Where <code>addr</code> is the host name such as <code>myhost.com</code> or an IP address. Both IPv4 and IPv6 addresses are supported. IPv6 addresses must be enclosed in square brackets. The port is optional. If not specified, the default Time Service port 123 is used. When multiple addresses are specified, then NTP server failover happens if communication with one of the servers fails. The Time Service will try all the servers before throwing a <code>TimeUnavailable</code> exception.
vbroker.time.ntp.timeout	5000	The time in milliseconds to wait for a reply from the NTP server. If multiple NTP servers are specified then failover to next server happens after the timeout expires.

Creating Time Service Objects with the `TimeService` interface

The VisiTime Service interface `TimeService` provides methods for creating UTOs and TIOs, but doesn't provide any methods to deactivate/destroy these Objects. VisiBroker's `TimeService` implementation uses the default servant-based dispatch mechanism limiting the number of these objects, meaning that for any number of these references the real servant processing the request is only one. You will not, therefore, need to be concerned with a large number of Time Service objects—UTOs and TIOs—being created. You use the `TimeService` interface to create UTOs and TIOs. Before creating these objects, you must resolve to the Time Service and narrow it (using the `TimeServiceHelper` in Java). The following code samples explain how to do this:

```
C++
    //Initialize the ORB
    CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);

    //Resolve the TimeService interface
    CORBA::Object_var obj_t = orb->resolve_initial_references("CosTimeService");

    //Narrow the TimeService interface
    CosTime::TimeService_var time_svc = CosTime::TimeService::_narrow (obj_t.in());

Java
import org.omg.CORBA.ORB;
import org.omg.CosTime.*;
...
//Initialize the ORB
ORB orb = ORB.init(args, null);

//Resolve the TimeService interface
org.omg.CORBA.Object obj = orb.resolve_initial_references("CosTimeService");

//Narrow it properly using the Helper
TimeService timeService = TimeServiceHelper.narrow(obj);
```

Once you have resolved to and narrowed the `TimeService` interface, you can use it to create UTOs and TIOs.

Creating UTOs using the TimeService interface

Use the `TimeService` method `universal_time()` to create a Universal Time Object containing the current time. For example,

```
C++    CosTime::UTO_var uto = time_svc-> universal_time();
Java   UTO uto = timeService.universal_time();
```

creates a Universal Time Object `uto` whose time value is the current time at the execution of the method.

You can also create a UTO containing a relative time of your choosing (not obtained using a Time Source) using the `new_universal_time` method. You provide three arguments to this method:

- the 64-bit time value. This is the number of hundreds of nanoseconds that have elapsed since base time and is a C++ `CORBA::ULongLong` or Java `long` data type.
- the time inaccuracy value.
- the `Tdft` value, a C++ `CORBA::Short` or Java `short` data type.

For example:

```
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);
```

Creating TIOs using the TimeService interface

You can create TIOs using the `TimeService` interface. The `new_interval` method takes two arguments of type `CORBA::ULongLong` (C++) or `long` (Java), which are the bounds of the time interval expressed as hundreds of nanoseconds since base time. For example:

```
C++    //Create a TIO that represents a specific interval
        CosTime::TIO_var tio = time_svc->new_interval((CORBA::ULongLong)10000000,
                                                    (CORBA::ULongLong)20000000);
Java   //Create a TIO that represents a interval
        TIO tio = _timeService.new_interval(10000000L, 20000000L);
```

Using the Timer Event Service

This section explains how to resolve to a Timer Event Service, obtain `TimerEventHandlers`, set alarms using the `TimerEventHandlers`, cancel an alarm that was previously set, and unregister a `TimerEventHandler`.

Before creating and utilizing `TimerEventHandlers`, you must resolve to the Timer Event Service itself, as well as the ORB's standard Event Service providing the `PushConsumer` object. For example:

```
C++    //Initialize the ORB
        CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);

        //Resolve the TimerEventService
        CORBA::Object_var obj_t = orb-
        >resolve_initial_references("CosTimerEventService");
        CosTime::TimerEventService_var time_evsvc =
            CosTime:: TimerEventService::_narrow (obj_t.in());
```

```

//Resolve to the EventService
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.*;
...

//Initialize the ORB
ORB orb = ORB.init(args, null);

//Resolve the TimerEventService
TimerEventService timerEventService=TimerEventServiceHelper.narrow(

_orb.resolve_initial_references("CosTimerEventService"));

//Resolve to the EventService
EventChannel channel =

EventChannelHelper.narrow(_orb.resolve_initial_references("EventService"));

```

Creating TimerEventHandlers

the Timer Event Service provides an operation to register a `CosEventComm::PushConsumer` together with a `CORBA::Any` that provides event data. Internally, the `TimerEventHandler` is created and the event data and `PushConsumer` are associated with the it. You can at any point change the event data, but the `PushConsumer` is immutably associated with the `TimerEventHandler` and cannot be changed.

Once you have resolved the Timer Event Service and Event Service, and you have obtained a channel from the latter, you can create the Event Handler implementation. To do so, you must follow these six steps:

- 1 Create a `ProxyPushSupplier` object to push the event data to the consumer.
- 2 Create a `PushConsumer` object to receive the event data.
- 3 Associate the `ProxyPushSupplier` with its `PushConsumer`.
- 4 Obtain a `ProxyPushConsumer` object from the event channel. This is the object that will be registered with the Timer Event Service.
- 5 Create the event data with a new `CORBA::Any`.
- 6 Create the event handler by executing the Timer Event Service's `register` method, using the `ProxyPushConsumer` and the `CORBA::Any` objects as arguments.

The following tables show source code used to execute each of the steps above:

Table 2.6 C++ code

Step	Code
1	<pre> //Create a ProxyPushSupplier CosEventChannelAdmin::ConsumerAdmin_var cns_adm = channel->for_consumers(); CosEventChannelAdmin::ProxyPushSupplier_var pushSupplier = cns_adm->obtain_push_supplier(); </pre>
2	<pre> //Create the PushConsumer, PushView here is the implementation of PushConsumer PushView* view = new PushView(); </pre>
3	<pre> //Connect the PushConsumer pushSupplier->connect_push_consumer(view->_this()); </pre>

Table 2.6 C++ code (continued)

Step	Code
4	<pre>//Get a ProxyPushConsumer from the Event Channel CosEventChannelAdmin::SupplierAdmin_var sup_adm = channel->for_suppliers(); CosEventChannelAdmin::ProxyPushConsumer_var proxy = sup_adm->obtain_push_consumer();</pre>
5	<pre>//Create the data that we want to receive when the event is triggered CORBA::Any any ; any <<="my data";</pre>
6	<pre>//Register the PushConsumer and the event data to obtain a TimerEventHandler CosTimerEvent::TimerEventHandler_var eventHandler = time_evsvc->register(proxy,any);</pre>

Table 2.7 Java code

Step	Code
1	<pre>//Create a ProxyPushSupplier ProxyPushSupplier pushSupplier = channel.for_consumers().obtain_push_supplier();</pre>
2	<pre>//Create the PushConsumer, PushView here is the implementation of PushConsumer PushView view = new PushView();</pre>
3	<pre>//Connect the PushConsumer pushSupplier.connect_push_consumer(view._this(orb));</pre>
4	<pre>//Get a ProxyPushConsumer from the Event Channel ProxyPushConsumer proxy = channel.for_suppliers().obtain_push_consumer();</pre>
5	<pre>//Create the data that we want to receive when the event is triggered Any any = orb.create_any(); Any.insert_string("my data");</pre>
6	<pre>//Register the PushConsumer and the event data to obtain a TimerEventHandler TimerEventHandler eventHandler = timerEventService.register(proxy,any);</pre>

Setting Alarms for TimerEventHandlers

In order to use your newly-created `TimerEventHandler`, you set alarms using the `EventTimer` interface. The `set_timer` method is used to set an alarm. It takes two arguments: the type of alarm and a UTO object. Three types of alarms are available:

- `TTAbsolute`: the alarm is triggered at an absolute time specified by the UTO.
- `TTRelative`: the alarm is triggered at the UTO relative to the current time (the UTO represents time from the current absolute time, not the time base).
- `TTPeriodic`: the alarm occurs periodically, repeating at each relative time specified by the UTO.

To set an alarm, you must:

- 1 Create a Timer Event Service object.
- 2 Create a new UTO that will be used to trigger the alarm.
- 3 Use the Event Handler's `set_timer` method to set the alarm.

For example, the following code sets an alarm for a `TimerEventHandler` object called `eventHandler`:

```
C++    //Create an UTO that represents relative time
        CosTime::UTO_var uto =
            time_svc-
            >new_universal_time((CORBA::ULongLong)10000000,0,(CORBA::Short)0);

        //set a periodic timer on the TimerEventHandler, this alarm would trigger after
        //every 1
        //second (10000000/10000) second has elapsed and the event data will be pushed
        //to the
        //PushConsumer that was previously registered
        eventHandler->set_timer(CosTimerEvent::TTPeriodic,uto);
```

```

Java //Create an UTO that represents a relative time
      UTO uto = timeService.new_universal_time(10000000L,0,(short)0);

      //set a periodic timer on the TimerEventHandler, this alarm would trigger after
      //every 1
      //second (10000000 /10000)second has elapsed and the event data will be pushed
      //to the
      //PushConsumer that was previously registered
      eventHandler.set_timer(TimeType.TTPeriodic,uto);

```

Note The Timer Event Service minimum relative interval for which an alarm can be set is 1 millisecond. Any value less than 1 millisecond will be transparently converted to 1 ms.

Canceling a Timer and Unregistering a TimerEventHandler

To cancel an event handler's timer, simply execute the handler's `cancel_timer` method:

```

C++ eventHandler->cancel_timer();
Java eventHandler.cancel_timer();

```

To unregister an event handler entirely, call the event service's `unregister` method:

```

C++ eventService->unregister(eventHandler);
Java eventService.unregister(eventHandler);

```

Friendly Time Object

This is an object with a friendly interface to convert the 64-bit time representation to human readable components like year, month, day etc and vice versa: the `TimeI` object. The `TimeI` object can be viewed as a representation conversion object. The general technique for using it is to create one using the operation `FriendlyTime::TimeService::time()`. This creates a `TimeI` object with time set to zero in it. Then the `_set` operations can be used to set the values of the various attributes. Finally, the attribute `time` can be used to get the corresponding `TimeT` value.

Conversely, one can set any `TimeT` value in the `time` attribute and then get the year, month, and so forth. from the appropriate attributes.

The IDL for the friendly time object is as follows:

```

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(); // set all attributes to zero
  };
}

```

The following code sample illustrates the usage of the friendly time object:

```

C++    //Initialize the ORB
CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);

//Resolve the FriendlyTimeService
CORBA::Object_var obj_t = orb->resolve_initial_references("CosTimeService");
FriendlyTime::TimeService_var time_svc = FriendlyTime::TimeService::_narrow(
obj_t.in());

//Get a TimeI object from the FriendlyTime
FriendlyTime::TimeI_var timeI = time_svc->time();

//Get the current time in a UTO
CosTime::UTO_var uto = time_svc-> universal_time();

//Set the current time in the TimeI object
timeI->time(uto->time());

//Get the various attributes from TimeI Object in a human readable format and
print to
//the standard output
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.*;
...
//Initialize the ORB
ORB orb = ORB.init(args, null);

//Resolve the FriendlyTimeService
org.omg.FriendlyTime.TimeService friendlyTs =
    org.omg.FriendlyTime.TimeServiceHelper.narrow(
        _orb.resolve_initial_references("CosTimeService"));

//Get a TimeI object from the FriendlyTime
org.omg.FriendlyTime.TimeI timeI = friendlyTs.time();

//Get the current time in a UTO
UTO uto = friendlyTs.universal_time();

//Set the current time in the TimeI Object
timeI.time(uto.time());

//Get the various attributes from TimeI Object in a human readable format and
print to //the standard output
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());

```


Index

Symbols

... ellipsis 4
[] brackets 4
| vertical bar 4

A

alarms, setting 18

B

bootstrapping using ORBDefaultInitRef 11
bootstrapping using ORBInitRef 11
bootstrapping using SmartAgent 12
bootstrapping VisiTime 10
Borland Developer Support, contacting 4
Borland Technical Support, contacting 4
Borland Web site 4, 5

C

cancelling timers 19
commands, conventions 4
components, time service 8

D

Developer Support, contacting 4
documentation 2
 .pdf format 3
 accessing Help Topics 3
 Borland Security Guide 2
 on the web 5
 platform conventions used in 4
 type conventions used in 4
 updates on the web 3
 VisiBroker for .NET Developer's Guide 2
 VisiBroker for C++ API Reference 2
 VisiBroker for C++ Developer's Guide 2
 VisiBroker for Java Developer's Guide 2
 VisiBroker GateKeeper Guide 3
 VisiBroker Installation Guide 2
 VisiBroker VisiNotify Guide 2
 VisiBroker VisiTelcoLog Guide 2
 VisiBroker VisiTime Guide 2
 VisiBroker VisiTransact Guide 2

F

failover, NTP 13
friendly time object 19

H

Help Topics, accessing 3

I

in-process time service 12
interface TimeService 15

N

Newsgroups 5
NTP failover 13
NTP server addresses 13
NTP support 12

O

online Help Topics, accessing 3
ORBDefaultInitRef, bootstrapping 11
ORBInitRef, bootstrapping 11
osagent, bootstrapping 12
overview 1, 7

P

PDF documentation 3
properties 13

R

running in-process 12

S

secure time services 9
server addresses, NTP 13
setting alarms 18
SmartAgent, bootstrapping 12
Software updates 5
starting secure service 10
starting service 9
Support, contacting 4
symbols
 brackets [] 4
 ellipsis ... 4
 vertical bar | 4

T

Technical Support, contacting 4
time definition 7
time interval object 8
time service configuration 13
time service services 8
time service, components 8
time source, NTP 12
timer event handler, unregistering 19
timer event handlers, creating 17
timer event service 8
timer event service, using 16
TimeService interface 15
TIO 8
TIO creation 16

U

universal time object 8
unregistering timer event handler 19
UTC 8
UTO 8
UTO creation 16

V

VisiBroker overview 1
VisiTime 7
VisiTime service 9

W

World Wide Web
Borland documentation on the 5
Borland newsgroups 5
Borland updated software 5