Versant Asynchronous Replication (VAR) Manual

Release 7.0.1.4

VERSANT

Versant History, Innovating for Excellence

In 1988, Versant's visionaries began building solutions based on a highly scalable and distributed object-oriented architecture and a patented caching algorithm that proved to be prescient.

Versant's initial flagship product, the Versant Object Database Management System (ODBMS), was viewed by the industry as the one truly enterprise-scalable object database.

Leading telecommunications, financial services, defense and transportation companies have all depended on Versant to solve some of the most complex data management applications in the world. Applications such as fraud detection, risk analysis, simulation, yield management and real-time data collection and analysis have benefited from Versant's unique object-oriented architecture.

For more Information please visit **www.versant.com**

Versant US

Versant Corporation

255 Shoreline Drive, Suite 450, Redwood City, CA 94065

Ph +1 650-232-2400, Fx +1 650-232-2401

Versant Europe

Versant GmbH

Wiesenkamp 22b, 22359 Hamburg, Germany

Ph +49.40.60990-0, Fx +49.40.60990-113

Table of Contents

CHAPTER 1

Versant Asynchronous Replication (VAR)

This Chapter gives detailed explanation on the Usage of Versant Asynchronous Replication (VAR).

The Chapter covers the following in detail:

- Introduction
- Features
- Concepts
- Actions
- Procedures
- Architecture
- Identifying Objects To be Replicated
- Replication Knowhows
- Dealing with Update Conflict
- VAR Administration
- Usage Notes
- License Details

INTRODUCTION TO VAR

In many applications, there is a need to replicate data, typically to improve availability, to improve performance by geographically co-locating databases with the applications that access the databases, to isolate decision support systems from on-line production systems, and to help in recovery from failures using warm-standby systems.

Replication can be achieved in two ways — Synchronous, where consistency of data between replicas is strictly maintained, and Asynchronous, where data consistency is loosely maintained.

Synchronous replication increases the duration of application transactions and cannot be completed until all the replicas have been brought to the same state.

Asynchronous replication does not prolong the transaction much, as it updates the replicas in a separate transaction from the application transaction. Therefore, asynchronous replication is most suitable where certain latency before achieving consistency among replicas is permissible.

Versant provides both synchronous as well as asynchronous replication. Versant provides synchronous database replication and roll forward archiving utilities and interface methods.

Alternately, you may want to create your own *scheme* for asynchronous database replication using event notification mechanisms.

This chapter lists the features offered by Versant Asynchronous Replication, and introduces concepts and usage of Versant Asynchronous Replication.

VAR Concepts

While synchronous database replication is useful for achieving fault tolerance and data safety, the, asynchronous replication is useful for achieving performance in a distributed environment.

For example, suppose that you have numerous regional offices accessing a common database. If you used asynchronous replication to maintain ten replicated local databases, users in each regional office would be making local database connections and the number of users per database would be reduced. In some situations, this approach would dramatically improve performance.

VAR FEATURES

Versant Asynchronous replication provides two modes of replication: Default and Advanced.

The Default mode uses Versant APIs for replication and supports only Master-Slave configuration. This mode is suitable for LAN based replication where network is fast and reliable.

The Advanced mode provides snapshot-based replication and supports peer-to-peer and Master-Slave configurations. This mode is suitable for WAN based replication where the network is slow and unreliable. The Advanced mode provides customizable messaging, transport mechanism and conflict detection and resolution.

Features provided by Default mode

Versant Asynchronous Replication offers the following core features that are provided by Default as well as Advanced mode.

Flexible configuration

The architecture allows for various Master-Slave configurations such as:

- Master-Slave configuration with one master primary site and multiple fragmented slave replica sites
- Master-Slave configuration with multiple primary sites and one consolidated replica site

Multiple language support

Objects created using any of the Versant language interfaces, that is, C++/Versant, Java Versant or C/Versant, can be replicated to and from Versant databases.

Differential Update

Objects that have been created, modified or deleted are replicated. Versant event notification is used to register the updates, deletions and create events on classes of objects specified by the user. An event daemon in the database at each site monitors changes and creates requests for replication.

Transactional or batch replication

Versant Asynchronous Replication 6.0 supports transactional and batch replication. In a transactional replication, a replication transaction contains objects that have changed in the same user transaction. In batch replication, objects changed in a batch of user transactions are replicated in a single replication transaction. Irrespective of whether a replication is transactional or batch based only the part of the transaction that is of interest to that site is applied at the receiving site.

Event or API Based replication

Two methods are provided for automatic or explicit replication. The first method uses Versant event notification to monitor updates to database objects, and automatically creates replication requests. The set of objects to be monitored can be specified in various ways. In the second method, applications use Java or C++ API to create replication requests explicitly. The second method, therefore, has more control over specifying the set of objects to be replicated and the timing of replication.

Recovery from failures

All the objects representing meta data used by Versant Asynchronous Replication are stored persistently to facilitate recovery and synchronization after network or machine failures.

Administrative tools

The product includes a set of command line utilities to help administer the replication sites and to check and fix corrupted meta data.

Logging

Logging facility is provided to report errors and status of replication.

Preserving application events

Events are raised irrespective of whether the updates occurred due to replication from other sites or due to an application. However, it is guaranteed that changes are replicated only once to all the target sites.

Features provided by Advanced mode

In addition to the features provided by the Default mode, the Advanced mode provides the following features:

Peer-to-peer configuration

Besides supporting different Master-Slave configurations, Advanced replication supports N-way peer-to-peer or symmetric replication with mechanisms for update conflict detection and resolution.

Conflict Detection and Resolution

You can avoid update conflicts by allowing changes to objects to occur at a single designated site. In applications where this restriction cannot be supported, Versant Asynchronous Replication allows update conflicts to be detected and resolved. In order to detect update conflicts, an administrator can assign ownership of database objects to primary sites. Updates made to objects at non-primary sites are propagated to the primary site for replication. The primary site is then a single point where update conflicts can be detected and resolved regardless of where the update actually occurred. The target sites receive their updates only from the owner site.

Versant Asynchronous Replication provides a default mechanism for detecting conflicts. You can optionally customize the detection and resolution of conflicts by implementing the interfaces specified for conflict detection and resolution.

Conflict prevention and avoidance

Mechanisms are provided for preventing conflicts by synchronously replicating to the owner sites, from where replication is carried out asynchronously to the other sites. Conflict avoidance is supported by enabling the administrator to assign ownership of database objects to their primary sites, and avoiding updates to those objects at the other sites.

Customizable transport

Interfaces are provided to customize the transport mechanism used. Default implementation uses Java sockets. Versant connections are not used to communicate from one site to another. The default supports replication over a LAN as well as a WAN.

Customizable message format

Message formats can be customized to support replication of parts of database objects instead of replicating objects (Java-serialized) in their entirety as default.

Explicit Replication

In Explicit mode, replication operates in two states, active and passive. In active state, replication requests are only consolidated without actually replicating the data. Whenever the state is changed from active to passive, objects specified by the consolidated requests are replicated first and then processing of replication requests is started.

VAR CONCEPTS

This section lists the terminology and introduces some of the concepts useful in understanding the usage of the product.

User Database: Database to or from which data needs to be replicated.

Source Database: Database where you create, modify and delete objects.

Destination Database: Database to which object creations, updates and deletions are replicated.

Replication Repository: The database in which all administrative information and the data required for recovery is stored. In this release, the user database is used as the replication repository.

Replica Site or Site: An addressable entity that consists of a user-database. Each site has a replication repository, which is typically the same as the user database. In Default replication each site address has an address that consists of fully qualified database name. For Advanced replication the address consists of an IP address of the host machine and the port number. You can have multiple sites on a single machine. A site can be a source site or a destination site depending on whether the user database at this site is a source database or a destination database.

Update Conflict: Creating updates or deletion of the same set of objects at more than one site in such a way that each site is unaware of the changes made to objects at the other sites. The replication receiver then identifies potential conflicts and uses the user provided or default conflict detection or conflict resolution code to detect and resolve the conflicts.

Replication Channel: An entity that

- specifies the destination replica sites or destination to which given sets of objects are to be replicated
- optionally specifies one of the replica sites, called the owner site for the channel, where any update conflicts between the set of objects are detected and resolved
- optionally allows for identification of the set of objects to be monitored for changes and replicated to the destinations

There are two types of replication channels— non-event based and event based. The primary difference between the two types of channels if the manner in which the set of objects to be replicated is identified. In a typical symmetric or peer-to-peer replication, a channel has one owner site and all other sites as the destination for the channel.

Non-event based Replication Channel: A channel that specifies the destination sites and optionally the owner site. The owner site determines the destination sites for the objects to be replicated, conflicts to be detected and resolved for objects replicated over that channel. Applications can specify the objects that are to be replicated over the channel using the replication API.

Event based Replication Channel: A channel that specifies a set of objects to be monitored for changes using Versant event notification. The set is identified by specifying the names of classes of objects or VQL queries, which the objects should qualify, or the LOIDs of the objects to be monitored. Whenever, an object in the set changes it is automatically marked for replication.

Replication Request: A replication request tracks all the objects that need to be replicated. A replication request is associated with one or more replication channels. Replication request can be created in two different ways — by the request engine running in an event daemon, or by applications using an API to explicitly specify the objects to be replicated.

Replication Request Engine: A software module that runs with a Versant event daemon at each source database. It registers and monitors events raised by the database due to changes that occur in the set of objects defined by replication channels. It then collects all the events that belong to the same transaction and creates persistent replication request objects for use by the replication receiver.

Default replication specific

Replicator: A server process at the Master replica site, which waits for replication requests. Replicates objects specified by the request to Slave replica sites.

Advanced Replication specific

Owner Site: The site that owns a replication channel. An object replicated over the replication channel is replicated from the owner site. Request to replicate objects over a channel originates as the result of updates to objects at any site and the owner site is responsible for replicating the changes to the other destination sites. The idea of ownership of channels is optional, and the administrator can enable or disable this for channels. Conflict detection occurs at the owner site. Therefore, an owner site should be assigned to a channel, if conflict detection and resolution has to be performed.

Replication Message: The actual message sent from a source site to a destination site. It is the transformation of a replication-request. It consists of all the information in the replication-request, as well as an image of the objects associated with the replication request.

Replication Sender: A server process at each replica site that waits for replication messages to be placed on the replication message queue by the replication receiver. If not the owner site, the replication sender forwards the message to the owners of the replication channels associated with the replication message. If it is one of the owner sites, the replication sender, sends the message to the other destinations specified by the replication channels that it owns.

Replication Receiver: A set of server processes, each having a different function, at each replica site. The replication receiver processes replication messages received from other sites, performs conflict detection and resolution (if conflict detection is enabled), updates local databases based upon messages received and prepares messages for other sites.

VAR ACTIONS

When you create an asynchronous database application, the following Versant mechanisms are useful.

Create replica utility

Create a replica database that is a copy of a source database using a command line utility. The replica database name and database identifier must not exist on the replicated system.

```
UNIX   createreplica
Win    createrep
```

Copy objects

Copy objects to a target database without changing their logical object identifiers.

```
c     o_moveobjs()
c++   migrateobjs()
```

Set database for vstr read operations

For a vstr of objects, specify the database from which they are to be fetched.

```
c        o_setobjsdb()
c++      setobjsdb()
```

VAR PROCEDURES

A typical approach to asynchronous database replication is to first create a replica database and then maintain it with a combination of event notification and object copying methods and procedures.

Typically, maintenance will be done by setting up a process for each database which uses event notification to watch for any changes. When a change is made, it propagates it to the other databases using routines such as `o_moveobjs()` or `copyAllFromDB:`.

Instead of using event notification to propagate changes as they happen, it is also possible to write a program which propagates a batch of changes all at once. For example, if you have, say, a thousand salesmen carrying around copies of a master database on notebook computers, at the end of the day they could return to their offices, plug into the network, and run a program which copies new objects back to the master database.

Disadvantages

One disadvantage of asynchronous replication is that there is a lag time between when a change is made in one database and when it becomes visible in the other databases. As a result, there is a window where the databases are inconsistent. This can lead to a situation where different users at different times make conflicting updates to an object without knowing that it has been changed by another user. If you are dealing with such a situation, you might want to refer to the chapter on "Optimistic Locking" which explains how you can create and use timestamps to coordinate updates to unlocked objects.

For some applications, the disadvantages of asynchronous replication will not matter, but for others it will. Because of the great flexibility available, it is often possible to come up with workarounds for dealing with the inherent problems.

For more information on optimistic locking, please refer to the Chapter “Locks and Optimistic Locking” in the *Versant Database Fundamentals Manual*.

Example C++/Versant

Following is one way to perform asynchronous database replication using event notification and object migration routines.

It is provided as an example, rather than as the only way asynchronous database replication can be done.

The code for the following example is for C++/Versant. Utility names are for UNIX.

The central element in the following approach is the event notification daemon process. It acts not only as an event dispatcher, but also as an object replication server.

Steps to perform a asynchronous database replication.

1. Create a replica database with the `createdb` utility.

   The database name and the database identifier must be unique to the network system, because the same `osc-dbid` database system identifier file should be used for all databases involved.

For more information on creating a database, please refer to the Chapter "Database Creation", in the *Versant Database Administration Manual*.

See also the "setdbid" and "createdb" utilities in Chapter "Database Utilities", in the *Versant Database Administration Manual.*

2. Follow normal event notification procedures to initialize the environment, register events, and provide for cases when the replica database is down.
3. Fetch an event with `readevent()`.
4. Retrieve the logical object identifier for the changed object from the `o_event_msg` struct of the received event.
5. Migrate the changed object to the replication database.

For example:

```
o_event_msg event;
::dom->readevent(...); // fetch an event
loid objId = event->obj_event_raised; // get loid
LinkAny     object;
if ((object = ::dom->getcod(objId)) != NULL) {
    // Remember, the system will generate 'begin-event' and
    // 'end-event' events whose obj_event_raised fields
    // will be filled with all 0's. So the check is
    // recommended.
LinkVstrAny objvstr, dummy;
```

```
objvstr.add(object);
try {
   ::dom->migrateobjs(
         objvstr,
         DBNAME1, // source db
         DBNAME2, // replication db
         &dummy,
         O_MV_SYNC);
   ::dom->commit();
}
catch (PError &err) {
      // migration failed
}
end_try;
       // migration succeeded
```

Depending upon your needs, either the event notification daemon or any other application can copy objects to the replication database. In a real application, you might also want to copy objects in groups, rather than one at a time, or copy objects only if they met some condition. You can use these routines to implement periodic synchronization using either a push or a pull model.

Following is code that uses a loop to read events:

```
while(TRUE)
{
   try {
      dom->readevent(..., O_EV_IPC_NOWAIT);
   }
   catch (PError &err) {
      // Ping server here.
      ping_server();
      continue;
   }
   end_try;
// Process/dispatch event here.
...}
```

You might want the daemon to utilize a busy-waiting event handling loop since the option `O_EV_IPC_NOWAIT` is specified. It is possible to put the daemon to sleep for a while before pinging the server. However, if there is heavy event traffic, the event queue may overflow while the daemon is sleeping.

Following is code that pings the database server:

```
#define PING_EVENT 9999
#define DB  "mydb"
// Ping if the database server still alive.
// Return 1 if server is alive.
int ping_server()
{
   try {
      dom-> sendeventtodaemon(DB, PING_EVENT,
            0, NULL, 0, NULL);
   }
   catch (PError &err) {
      return 0;
   }
   end_try;
   return 1;
}
```

The daemon should ignore these ping events in its event loop.

If the server goes down, you might also want to notify client applications. When the server comes back, the daemon will need to re-initialize the event notification environment and possibly notify client applications.

NOTE:-Versant provides a Event Toolkit - a mechanism for delivery and processing of all user specified events to user-specified engines for processing.

Customization of the transport protocol and event registration is also possible. Please contact support for more information.

IMPORTANT:- Versant Event Toolkit will not be a part of the VOD installation and will be available under consultancy package.

ARCHITECTURE

Default Replication

Databases are monitored by an event daemon for registered events. When you register an event, you specify the classes or objects you want to monitor and the events you want to monitor. For example, you may want to monitor a specified set of classes and replicate any new, updated or deleted objects.

You can also use an interface method to specifically request that a specified change be replicated in other databases.

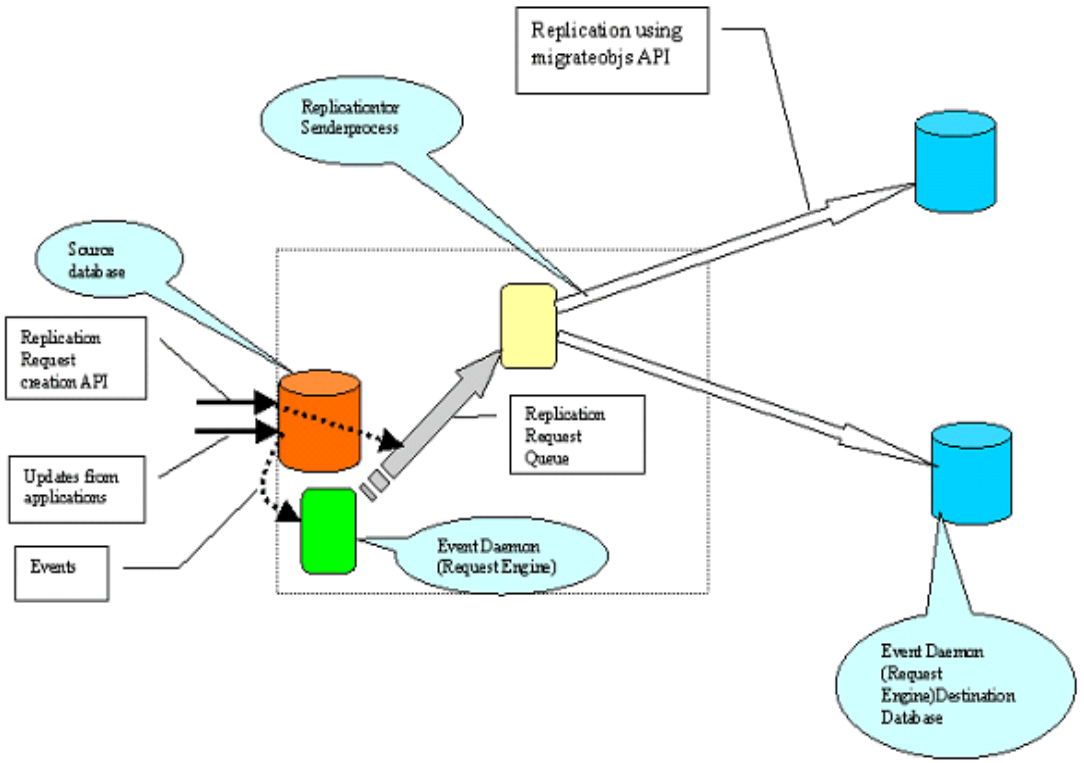
All replication registrations and replication requests are stored in a user database as persistent objects.

When an event of interest occurs or when a specific replication request is made, the replication engine creates a replication message containing the object changes to be replicated.

Each replication request is associated with a single transaction. All of the changes in a transaction are grouped by replication channel. A replication channel defines the set of objects that could be replicated from the source database. Out of this set, a replication request identifies a subset of objects for replication, that is, those objects that were created, modified or deleted in the same transaction.

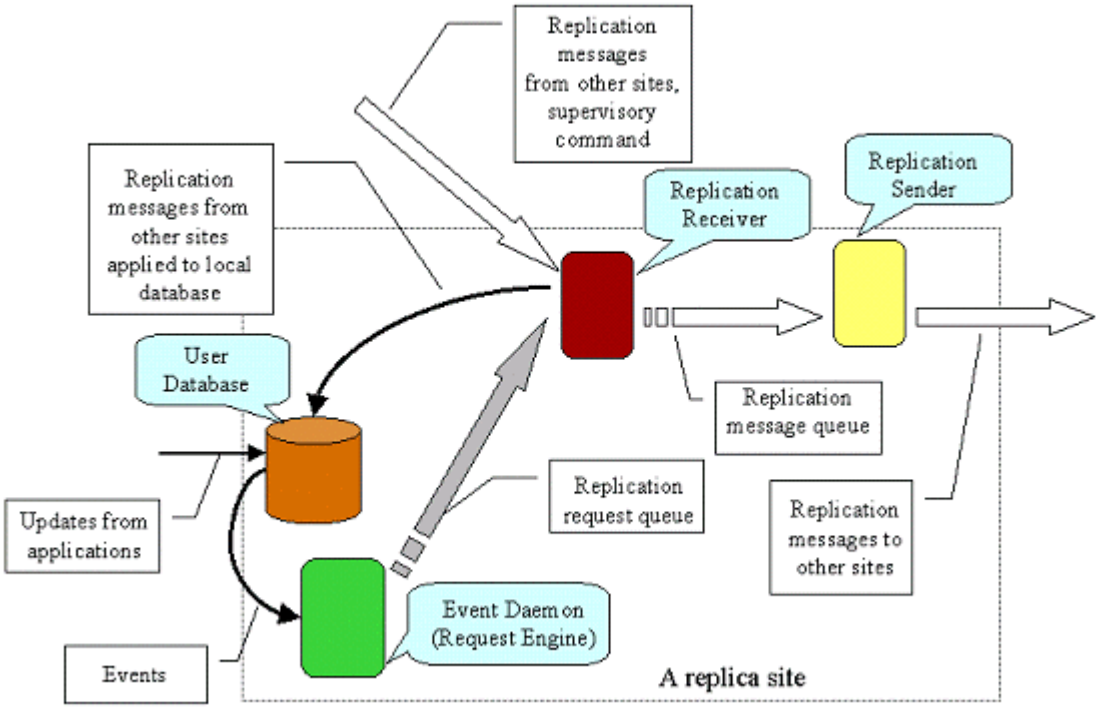
Replicator processes replication requests, associated in the source database and replicates objects specified by the requests to destination databases. The process uses Versant API (`migrateobjs`) to replicate objects.

The following is a visual representation of a Default replication.



Advanced Replication

Advanced replication uses the same mechanism as that of Default replication for identifying objects to be replicated. This mode uses customizable messaging and transport mechanisms for replicating objects. The following is a detailed visual representation of a replica site.



The replication receiver does the following:

- processes replication messages sent from other sites, filtering the messages to find those objects that belong to a channel on the local site.
- processes replication requests queued in the local database or for a batch of replication requests (if batching is enabled).
- performs conflict detection and resolution for changes to objects on the owner site (if conflict detection is enabled).
- updates local databases as appropriate.
- creates replication messages for other sites based upon requests made by the local request engine and then places the messages on a queue for use by the replication sender.
- processes administration and supervision messages made by an administration utility.

The replication sender takes the replication messages from the replication message queue and sends them to their destinations.

Replication messages are persistent objects, so the replication sender can be recovered and the messages queued for replication restored.

In explicit mode, if active state is enabled, the requests are consolidated in consolidated requests. Whenever the state is changed from active to passive, the replication messages are created for the objects specified by the consolidated requests. The replication -receiver starts processing the replication requests, once all the consolidated requests are processed.

IDENTIFYING OBJECTS TO BE REPLICATED

You can identify the set of objects to be replicated using either of the two different procedures:

Identifying Objects Using Methods

You can use replication interface methods to specify objects that need to be replicated. This approach is suitable where the number of applications used to update the databases is few and can be customized to use the methods to identify the objects for replication.

This approach is also suitable in cases where applications make large number of updates, replicate changes periodically, or replicate a closure or other related objects. The objects to be replicated should be determined during application development.

When you identify objects using methods, you will use non-event-based channels to identify the objects to be replicated.

If you use this approach, updates performed using database tools and other utilities will not be replicated.

Identifying Objects Using Event Notification

You can use event-based channels to identify the set of objects to be replicated. This allows you to use the asynchronous replication facility without modifying existing applications.

When an event-based channel is created one or more event registrations are created in the source database. Updates to the source database trigger events which are monitored by event daemon and replication requests are created automatically.

The following are the subtypes of event-based channels:

Class-based Replication Channel: Set of classes being replicated.

VQL-based Replication Channel: The channel specifies a VQL query for channel membership.

Object-based Replication Channel: Set of objects specified by their object id's.

During application development, you need not determine the set of objects to be replicated.

In both API based replication and event-based replication, a replication request specifies the logical object identifiers of the objects to be replicated. By the time the objects are replicated, the state of the objects may have changed.

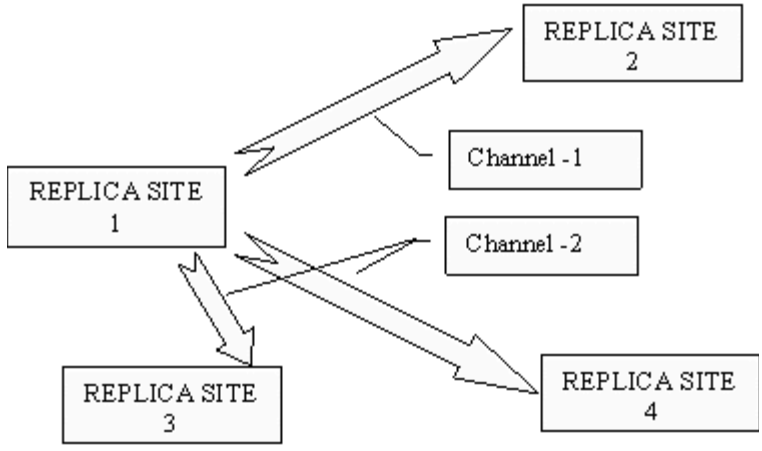
In a Master-Slave configuration, updates occur only at the master site. In Advanced mode channels defined at the slave sites need not have any event-registration information, although the channels at the master site could be defined as event-based. This implies that the same channel can be event-based at the master site and non-event based at the slave sites.

REPLICATION KNOWHOWS

Where Replication Occurs

Each replication request includes the details of the replication channels involved. The replication channels have a list of replica sites. So, while processing a replication request, the destinations of the replicated objects are known.

The following illustration shows how a database can be logically partitioned by defining multiple replication-channels so that only the specific data will be replicated to a destination.



Single replication sender, multiple channel replication

How Replication Occurs

Default Replication

Default replication uses Versant's `o_migrateobjs()` API to replicate objects to destination sites. The replicator establishes Versant connection to destination sites. After replicating objects, it commits the replication transaction.

Advanced Replication

Interfaces are provided for customizable message creation and transport mechanisms. Default messaging uses Java serialization coupled with GZIP compression. The default transport of replication messages uses Java TCP/IP sockets.

The replication request specifies the objects that need to be replicated. These objects are serialized using Java serialization, compressed to form replication messages. The replication-messages are streamed across to destination sites using Java sockets. In each of the destination sites, the replication receiver receives the stream, converts the objects to a canonical format and updates the destination databases transactionally.

When Replication Occurs

Transactional replication

When a new replication-request is created, it starts off the replication as a replication-request, which represents all the objects that were changed in a transaction.

Batch based replication

Frequency of replication depends on following configurable parameters:

- `MAXBATCHSIZE` — maximum number of user transactions per batch
- `MAXBATCHPERIOD`— *maximum* duration between two consecutive replication.

Replication starts off, whenever the number of replication-requests are at least equal to `MAX-BATCHSIZE,` or when the time elapsed after creation of the first replication-request in a batch is equal to `MaxBatchPeriod.`

Explicit replication

In Explicit mode, replication operates in two states, active and passive. In passive state, replication requests are only consolidated without actually replicating the data. Whenever the state is changed from active to passive, objects specified by the consolidated requests are replicated first and then processing of replication-requests is started.

DEALING WITH UPDATE CONFLICT

If the same set of objects can be updated from more than one replica site, that is, peer-to-peer configuration is needed then Advanced mode should be used as it provides support for conflict detection and resolution. This section describes in detail how to use Advanced mode in peer-to-peer configuration.

When you create a channel, you must assign one of the replica sites to be the owner of the channel. The owner site then detects and resolves conflicts for all of the channels it owns.

Though the updates may occur at any site, objects are replicated only from the owner site. If required the owner site can be moved to a different site.

Avoiding conflicts

You can avoid update conflicts if:

- no object belongs to more than one channel
- all updates to objects that belong to a replication channel occur at one site, though it may not be the owner site

Objects updated at one site are routed for replication to a single owner site to avoid update conflict. If a set of objects is to be updated at different sites, you can still avoid conflicts by ensuring that the updates occur at different times. For example, if the sites are separated by time zone and the updates occur during business hours at those sites.

Conflict detection and resolution

In cases where conflicts are unavoidable, conflicts can be detected and resolved.

Conflict detection and resolution is user configurable. In some applications, multiple sites may not modify the same set of objects at the same time. In such cases, you can turn off the conflict detection and resolution. In other applications, it may be important to detect a conflict and resolve it.

Conflict detection

By default, conflict detection is disabled and all requests for replication are queued at the source site and processed on a first-in-first-out basis. This mode is suitable for Master-Slave configuration. In peer-to-peer configurations, we recommend enabling conflict detection and resolution (by specifying it in the configuration file.)

The database administrator can enable conflict detection. If conflict detection is enabled, replication messages from different source sites are queued up at the owner site of the channel. You can specify a configurable period to retain the replication message in the queue after it is processed. This enables the detection of update conflicts, if the same objects are updated at different sources within the configurable time. At each owner site, a potential conflict is detected by comparing two replication messages `m1` and `m2`, where:

- `m2` is the current replication message being processed and `m1` is to be sent or processed by conflict detection
- `m1` and `m2` are from different sources
- `m1` and `m2` consists of one channel owned by the local site.
- `m1` is older than `m2` as per local timestamp

The replication messages are from different sources, which implies that there could be a potential conflict. In case of conflict, the two replication messages are handed over to a conflict detection module. You can over ride the default implementation provided. The default implementation flags a conflict when the replication receiver receives the new replication message by looking for a non-null intersection of the objects, specified in the two replication messages. The following table lists the outcome of the default conflict detection algorithm based on the intersection of various sets of objects in the two replication messages.

`C1` = Set of newly created objects in m1.

`M1` = Set of modified objects in m1.

`D1` = Set of deleted objects in m1.

`C2` = Set of newly created objects in m2.

`M2` = Set of modified objects in m2.

`D2` = Set of deleted objects in m2.

`C2 intersection C1`	Must be null. Non-null set is an error since 2 sites should not create objects with the same logical object identification.
`C2 intersection M1`	Must be null. Cannot modify non-existent objects.
`C2 intersection D1`	Must be null. Cannot delete non-existent objects.
`M2 intersection C1`	Conflict if not null.
`M2 intersection M1`	Conflict if not null.
`M2 intersection D1`	Conflict if not null.

`D2 intersection C1`	Conflict if not null.
`D2 intersection M1`	Conflict if not null.
`D2 intersection D1`	No Conflict even if not null.

Conflict resolution

If a conflict is detected, the conflict resolution module is invoked. The current message being processed and all replication messages that it conflicts with are passed to the conflict resolver. The default resolution is to discard the current replication message and log the details of the conflict. You can override the default behavior. For example, you could create a conflict resolver that attempts to resolve the conflict by modifying the current replication message. You can do so by using information from the older replication messages, and re-routing it back to the source of the replication message to correct the update.

For example, conflict handling from regional offices would be possible if an update from the head office is not already queued. In such conditions, the primary site will optionally discard the request from the regional office. The transaction at the regional office is committed. Therefore to undo the transaction you have to modify the replication message at the primary site, that will replicate the current information at the primary site.

Besides default conflict resolver, Compensating Transaction resolver is provided. The Compensating Transaction resolver creates compensating message for the conflicting message. The compensating message is sent back to the source site. The compensating message will undo the changes represented by the conflicting message when it is applied at the source site.

For more information refer to “VAR Configuration” on page 56, in "Chapter 2 - VAR Reference".

Guidelines for defining replication channels

To ensure replication consistency at all levels, certain guidelines must be followed while defining a channel and assigning an ownership to these channels.

- An object must not belong to more than one channel

  This ensures that if an object is updated at two sites, the conflict is detected and resolved at a single owner site.

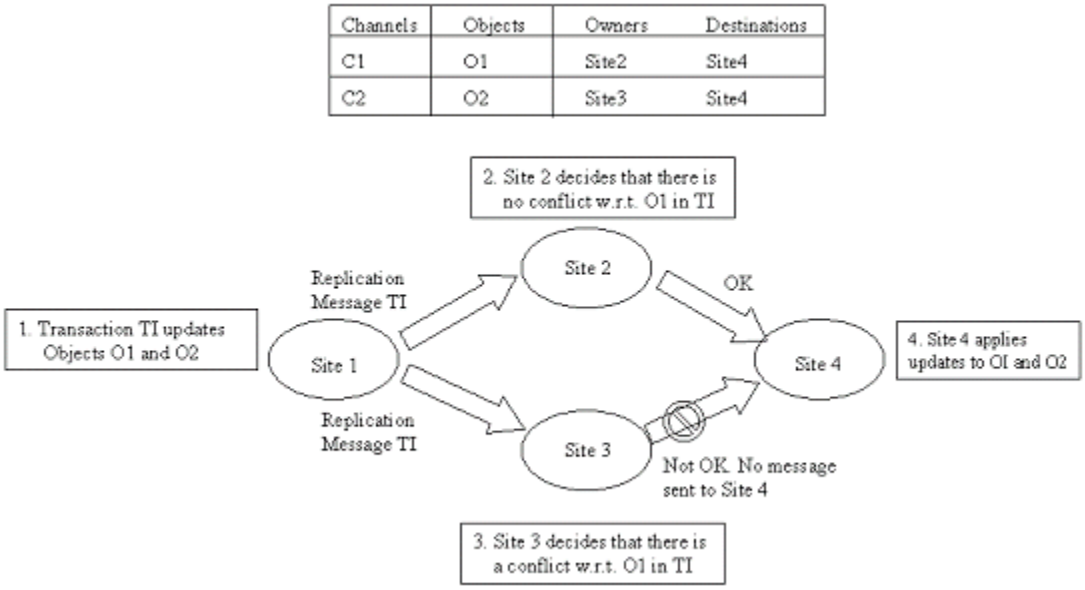
To follow this guideline, you can use VQL queries:

- A channel should include all the related objects that can be updated in a single transaction. Conversely, related objects should not be placed in more than one channel.

This restriction ensures that all the objects involved in a transaction are queued up at one site for conflict handling and replication. Else few objects may be handled for conflicts at one site and few others at another site. This behavior may result in a different conflict with different outcomes of conflict handling for multiple owners.

You must ensure that a single site owns all the related objects belonging to one or more channels. With this, you can be sure that the replication message representing the transaction will not be fragmented for conflict handling at multiple sites.

However this release does not enforce the guidelines, but if the rules are not followed the system attempts to ensure a deterministic outcome. For example, an owner site owns some of the objects updated in a transaction and a different owner owns the remaining objects updated in the same transaction. If one of the owners conclude that there has been no conflict, the entire replication message and the transaction is considered conflict free. The following illustration shows replication through multiple owners.



Replication through multiple owners

VAR ADMINISTRATION

There are two types of users of Versant Asynchronous Replication - the Administrator and the Developer.

The Developer is involved only if there is a need to customize the defaults supported by the product, or when the objects to be replicated are to be identified using the replication methods. For example, to satisfy a requirement to customize the conflict detection and resolution for an application, the Developer would implement the necessary methods. Or, in a C++ application, the Developer could use the C++ API to specify the set of objects that need to be replicated in a transaction.

The API section of the VAR Reference Chapter in this manual, describes the modules and APIs that the Developer would use.

The Administrator is responsible for the installation and configuration of the product. The Administrator is also responsible for initializing the replication repository, starting and stopping the replication processes. Configuring the system, monitoring the replication and handling of system errors that may be encountered during replication is also handled by the Administrator.

General administration and typical usage of the utilities is described in what follows.

For more information, on the utilities, software components and configuration file entries, refer to Chapter 2 “VAR Reference” on page 55.

The two key entities in the system that the Administrator will have to deal with are replica sites and the replication channels.

Configuring database

Before using Versant Asynchronous Replication:

- Check if the databases used for replication have different DBID’s.
  If the DBIDs are same you are likely to get an exception `EVJ_NO_SUCH_ATTR`. If you get this error, recreate and reinitialize one of the replicating databases.

Before using Versant Asynchronous replication, configure database:

- Edit `profile.be` file of the database.
  Set event registration mode to transient and event message mode to persistent.

```
event_registration_mode    transient
event_msg_mode             persistent
```

- Load replication schemas and initialize the database

```
varinitdb                  dbname
```

Administering replica sites

Replica site can be administered using the Java utility `varsiteadmin`.

- You must specify the replication-repository (database name) when using this utility
- The utility connects to the replication-repository database using a Versant connection (in this release, the repository database is the same as the user database).

In the Default Replication Mode, the Master site needs to be administered as replication processes run only at the Master site. However, in Advanced Replication Mode, every site must be administered separately.

Replica site administration involves the following tasks:

Creating a new site: In Default mode, it consists of adding information about the Slave site at the Master site.

In Advanced mode, it consists of creating site information at the new site, and adding the new site information to all the other sites. The site information includes the name of the new site, the address of the new site, that is, the hostname and port number.

Information related to the name and address of a particular replica site must be the same at all the replica sites. This applies to peer-to-peer configuration where each site needs to know about all the other sites. However, in a Master-Slave configuration, the slave site need not know about the other sites.

The utility `varsiteadmin` is used for creating site information at the new site and at the other sites.

Creating a new site also involves updating replication channel information to add the replica site to the list of destinations of relevant replication channels. The utility `varchadmin` described in the following section should be used for managing replication channels.

Setting up or updating the configuration file: The configuration file specifies the following parameters:

- log level and log directory location
- whether to use Default or Advanced replication
- whether to use transactional or batch replication
- location of the Java VM to be used by the replication processes, and the Java VM options to be used.
- conflict detection related parameters
- parameters to optionally turn on or off replication components
- customization of transport, messaging and conflict detection and resolution

Starting replication processes: The utility `varsiteadmin` is used to start the replication processes. The event daemon and the database should be started simultaneously before the applications start updating the database, so that events are not lost.

Monitoring replication processes: The utility `varsiteadmin` is used to find the status and statistics of the replication processes. Additionally log files can be used to get more information about the process. A log file is a plain text file containing the progress details of the replication, warnings and errors during replication. Separate files are used to log messages from different processes.

Stopping the replication processes: The utility `varsiteadmin` is used to stop replication processes, except the event daemon at a particular site. Event daemon should be stopped after all the user applications are stopped so that events are not lost. The utility `vedstop-driver` can be used to stop the event daemon.

Updating a replica site: In the Default mode, you need not update the site information, as address of a site is the database name itself.

In the Advanced mode, you may need to change the address of a site. Updating a replica site involves changing the address of the replica site at all the replica sites. It is important that the address details of a particular replica site be the same at all the replica sites.

This applies to peer-to-peer configuration where each site needs to know about every other site. However, in a Master-Slave configuration, the slave sites need not know about the other sites.

The utility `varsiteadmin` is used for updating site information.

Removing a replica site: In the Default mode, you must remove the information of a Slave site from the Master site using the utility `varsiteadmin`. In the Advanced mode you can remove the information of the site from all the replica sites using the utility `varsiteadmin`. In both the modes removal also involves updating replication channel information to remove the replica site from the list of destinations of relevant replication channels.

Listing site information: The utility `varsiteadmin` can be used to list information about a specific site or all the sites.

Enabling/Disabling: The utility `varsiteadmin` can be used to set the explicit replication state.

Administering channels

The utility `varchadmin` is used for administering replication channels:

- The utility operates on a specific replication repository (user-database) at a site
- The utility uses Versant connections as well as socket connections to Versant event daemon for the given database

In the Default Replication Mode, a channel should be created at the Master site, as replication processes runs only at the Master site. However, in Advanced Replication Mode, a channel should be created at all the sites that subscribe to the channel.

You must ensure that the channel definitions are the same at all the involved replica sites. You can use the utility to ensure channel consistency across multiple sites.

The utility `varchadmin` can be used to:

- create replication channels
- update replication channels
- change ownership of channels
- add or remove channel destinations
- enable or disable replication channels, this is applicable to event based channels only
- remove channels
- list all channels or a specific channel at a site or database
- compare channel definition across multiple sites or databases

On-line or off-line?

In this release, certain restrictions on the usage of the administration utilities exist with respect to the site being administered or all the sites being on-line or off-line. The restrictions also depend on the configuration peer-to-peer or Master-Slave of the replica sites.

The following site administration tasks typically require the entire system, that is, all the sites to be off-line

- creating a new replica site
- updating a replica site
- removing a replica site

The following channel administration tasks typically require the replica sites involved with the channel to be off-line

- creating a new replication channel
- updating a replication channel
- removing a replication channel

Tasks that can be done on-line are:

- listing information of the replica sites
- listing status or statistics information at a site
- listing information of the replication channels
- comparing replication channel definitions at multiple sites
- enabling or disabling replication channels
- in Advanced mode, enabling or disabling replication to other sites.

Customization support

Customization support is applicable only to the Advanced mode of replication. Implementing one or more of the Java interfaces provided as part of the Versant Asynchronous Replication APIs can customize several components. Default implementations are provided to address most of the basic needs. The following lists the components that can be customized.

Object reader and writer: A few examples of customization includes:

- Replicate a subset of the attributes of objects
- Transform the attributes of an object before replication

Messaging and transport: The default implementation using Java serialization and sockets can be customized to suit application requirements.

Conflict handling: Conflict detection and resolution can be customized. Besides default conflict resolver, Compensating Transaction resolver is provided, which creates compensating messages for the conflicting messages.

Handling schema evolution

This release does not support schema evolution. An error is reported if the objects received at a site have different schema than that of the local schemas.

NOTE:- You can define a channel on class `class` and class `attribute` to propagate changes to schema objects as well. But this needs testing for various language bindings and class definitions.

Comparing replicas

The utility `varcompare` can be used by the administrator to compare the two sites and list the differences. The inputs are the two database names and the class names of the objects to be compared at the two sites. If class names are not specified then the objects of all the user classes are compared. The utility uses a normal Versant connection to access the two databases and lists the Logical Object ID's of objects found in

- the database of one site but not at the other site
- the database of both the sites but differ in value

How to recover from failures?

Default Replication

If a network failure occurs while replicating objects, then the current replication transaction is rolled back and attempts are made periodically to connect to all the receiving sites. For all other failures, the replicator process will shut down and will record the fatal failures in the log file. Whenever replication is restarted, replicator starts processing replication requests from where it had left off.

Advanced replication

Network failures

If a network failure occurs while sending a replication message, attempts are made periodically, till a connection is established with the receiving site. Meanwhile, messages destined to that site are queued up. After establishing the connection the messages are sent in the original order. The receiver, on the other hand, detects and discards any duplicate replication messages. This assures a single message delivery even on encountering network failures.

Process failure while sending a replication-message to a site

If a replication process fails, a driver process attempts to restart the process for a configurable number of times. If unsuccessful, all the processes are shut down and the fatal failures recorded in the log file.

Each replication process reads replication requests or messages from persistent queues. A persistent counter is maintained to track the last request processed and the last message sent. Thus, when the process restarts, it can start processing replication requests and the messages from where it had been discontinued.

Using utility `varcheckmeta`

Utility `varcheckmeta` checks consistency of meta data. It prompts to run with additional options if it detects discrepancies in the meta data. Therefore, if the replication does not start or hangs then run `varcheckmeta` to check whether meta data is consistent.

How to synchronize a new replica site?

The following procedure describes how to synchronize a new replica site without stopping databases at other replica sites.

- no need to stop user applications
- stop replication at Master site if it is Master-Slave configuration and at all the sites if it is peer-to-peer configuration. Do not stop event daemon if event based channels are used.
- take back-up of database at Master site if it is Mater-Slave configuration or one of the peer database if it is peer-to-peer configuration using `vbackup` utility.
- use backup to create new site database and remove meta data using `varrmmeta` utility.
- initialize the database using `varinitdb` utility and create new site as explained in Administering replica sites section.
- start replication at the new site and at all other sites where replication was stopped.

For more information on vbackup utility, please refer to Chapter “Database Utilities” in the *Versant Database Administration Manual.*

How to synchronize replica sites when there is long term failure?

The following procedure describes how to synchronize new replica site without stopping databases when the replica sites are out of sync for a long time, for example: due to network failure.

- no need to stop user application but if API based channels are used then user applications should stop creating request objects.
- stop replication at all the sites. Also stop event daemon if event based channels are used.

If Master-Slave configuration is used

- remove meta data from Master site using `varrmmeta` utility and remove databases at the Slave sites.
- reinitialize master site using `varinitdb` utility and recreate channels and sites.
- restart event daemon if event based channel is used, or notify applications to create requests
- take backup of Master site database using `vbackup` utility

- use backup to create slave site databases and remove meta data using `varrmmeta` utility. Create sites and channels at slave sites
- start replication at all the sites.

If peer-to-peer configuration is used

- remove meta data at all the sites using `varrmmeta` utility
- reinitialize all the sites using `varinitdb` utility and recreate channels and sites
- restart event daemon if event based channel is used or notify applications to create requests
- synchronize all the sites using `vstream` utility or you will have to develop a custom utility which synchronize sites using some policy
- start replication at all the sites

USAGE NOTES

This section describes the methods involved in setting up new sites and creating channels.

Default Replication

This is used if you need to set-up a replica database for a master database where all updates to master will be replicated to replica database. This configuration is useful for improving performance of the master database, as all read-only operations will be performed on replica database. Steps required to set-up the configuration:

Site	Database name	Host name
`Master`	`masterdb (existing)`	`masterserver`
`Replica`	`replicadb (new)`	`replicaserver`

Channels:

Channel name	Destinations	Criteria
`AllObjects`	`Replica`	Instances of all the classes

Set up Replica site

- Create and set-up the replica database using Versant ODBMS utilities
  ```
  makedb -g replicadb
  createdb replicadb
  ```
- Copy information from the masterdb to replicadb using Versant ODBMS utility `vcopydb`
  ```
  vcopydb -nocreate masterdb@masterserver replicadb@replicaserver
  ```

NOTE:- Please refer to the section on synchronizing new site if the database `masterdb` cannot be stopped.

Set up Master site

Edit the Versant database profile.be file of `masterdb@masterserver`

- Change the event registration mode parameter to transient and event message mode to persistent.

```
event_registration_mode    transient
 event_msg_mode            persistent
```

- Load the replication schema and initialize replication repository at `masterdb@masterserver`

  `varinitdb masterdb@masterserver`
- Create site information in the database `masterdb`

  `varsiteadmin create -database masterdb -name Master -address masterdb@masterserver`
- Add site information about the replica database

  `varsiteadmin add -database masterdb -name Replica -address replicadb@replicaserver`

Start event daemon of the database masterdb

```
start veddriver masterdb master_ved.cfg
```

where `master_ved.cfg` is the configuration file name for the event daemon.

Create replication channel to monitor instances of all the classes at the Master site

```
varchadmin create -name AllObjects -database masterdb -site Replica
-class all
```

Start replication processes at the Master site

```
varsiteadmin start -database masterdb -config master.cfg
```

The file `master.cfg` is the configuration file for the replication processes at Master site. At this point any application creating, updating or deleting any objects at the Master site will trigger asynchronous replication of those objects to the Replica site. You need not start replication processes at the replica site.

Stop the replication server processes

```
varsiteadmin stop -database masterdb -config master.cfg
```

Advanced Replication

The assumption is that there exists an existing database in New York `nydb@nyserver` that needs to be replicated to a new database `parisdb@parisserver` located in Paris. In this example, it is assumed that all changes to the `nydb@nyserver` need to be replicated. This is achieved by using a special channel named all to include every class in the channel. It is also assumed that the requirement is to allow peer-to-peer replication (that is, updates can occur at both sites).

For example, we need to set up two sites - one at New York, and another at Paris with the following configuration:

Site	Database name	Host name	Port number
New_York	nydb (existing)	nyserver	6000
Paris	parisdb (new)	parisserver	7000

Channels:

Channel name	Owner	Destinations	Criteria
AllObjects	New York	Paris	Instances of all the classes

Set up site at Paris

- Create and set-up the database at Paris using Versant utilities.

  ```
  makedb -g parisdb
  createdb parisdb
  ```

- Copy information from the database at New York to the one at Paris using Versant utility `vcopydb`:

  ```
  vcopydb -nocreate parisdb@parisserver nydb@nyserver
  ```

NOTE:- Please refer the section on "Synchronizing new site" if the database at New York cannot be stopped.

- Edit the Versant database `profile.be` file for both `nydb@nyserver` and `parisdb@parisserver`

- Change the `event_registration_mode` parameter to transient and `event_msg_mode` to persistent

```
event_ registration_ mode     transient
event_msg_mode                persistent
```

- Load the replication schema and initialize the replication-repository in both the databases

```
varinitdb nydb@nyserver
varinitdb parisdb@parisserver
```

- Create site information in the database `parisdb` at Paris.

```
varsiteadmin create -database parisdb -name Paris -address paris-
server:7000
```

- Add site information of the new Paris site.

```
varsiteadmin add -database parisdb -name NewYork -address nyser-
ver:6000
```

Start the event daemon of the database at Paris.

- Start `veddriver paris.cfg`

  where `paris.cfg` is the name of the configuration file for the event daemon.

Create replication channel to monitor instances of all the classes at the Paris site.

```
varchadmin create -name AllObjects -database parisdb -site Paris
-owner NewYork -class all
```

Set up site at New York.

- Create site information in the database `nydb` at New York.

```
varsiteadmin create -database nydb -name NewYork -address nyser-
ver:6000
```

- Add site information of the new Paris site.

```
varsiteadmin add -database nydb -name Paris -address parisser-
ver:7000
```

Start event daemon of the database `nydb` at New York.

- Start veddriver nydb ny.cfg

  where `ny.cfg` is the name of the configuration file for the event daemon.

Create replication channel to monitor instances of all the classes at the New York site.

```
varchadmin create -name AllObjects -database nydb -site Paris -owner
 NewYork -class all
```

Start replication processes at the New York site.

```
varsiteadmin start -database nydb -name NewYork -config nysite.cfg
```

The file `nysite.cfg` is the configuration file for the replication processes at New York.

Start replication processes at the Paris site.

```
varsiteadmin start -database parisdb -name Paris -config paris
site.cfg
```

The file `parissite.cfg` is the configuration file for the replication processes at Paris.

At this point, creating, updating or deleting any objects at the New York site will trigger asynchronous replication of those objects to the Paris site.

Optionally check the status of the replication processes at the Paris or New York site.

```
varsiteadmin status -database parisdb -config parissite.cfg
```

Stop replication.

To stop the replication server processes, for example, at the Paris site:

```
varsiteadmin stop -database parisdb -name Paris -config parissite.cfg
```

Explicit Replication in Advanced mode

Suppose, we need to set up two sites - one at New York and another at Paris in master slave configuration.

Updates at New York needs to be sent to Paris periodically, for e.g. At 8.00 PM everyday, all the updates which happened at New York during a day should be sent to Paris.

Sites:

Site	Database name	Host name	Port number
New York	nydb (existing)	nyserver	6000
Paris	parisdb (new)	parisserver	7000

Channels:

Channel name	Owner	Destinations	Criteria
AllObjects	New York	Paris	Instances of all the classes

1. Set up site at Paris
   - Create and set-up the database at Paris using Versant ODBMS utilities.

     ```
     makedb -g parisdb
     createdb parisdb
     ```
   - Copy information from the database at New York to the one at Paris using Versant ODBMS utility `vcopydb`

     ```
     vcopydb -nocreate nydb@nyserver parisdb@parisserver
     ```

NOTE:- The Versant ODBMS utility `vstream` can also be used. However, it requires a transfer of files between sites, using ftp for example. The advantages are that it does not need Versant connection across sites.

For more information on `vstream` utility, please refer to Chapter "Database Utilities" in the *Versant Database Administration Manual.*

- Edit the Versant database profile.be file for both `nydb@nyserver` and `parisdb@parisserver`.
- Change the `event_registration_mode` parameter to transient and `event_msg_mode` to `persistent`.

  ```
  event_ registration_ mode     transient
  event_msg_mode                persistent
  ```
- Load the replication schema and initialize the replication-repository in both the databases

```
varinitdb nydb@nyserver
varinitdb parisdb@parisserver
```

- Create site information in the database parisdb at Paris.

```
java com.versant.VARSiteAdmin create -database parisdb -name Paris -
address parisserver:7000
```

- Add site information about the new Paris site.

```
varsiteadmin add -database parisdb -name NewYork -address nyser-
ver:6000
```

2. Create replication channel to receive changes from site at NewYork.

```
varchadmin create -name AllObjects -database parisdb -site Paris
```

3. Set up site at New York.
   - Create site information in the database `nydb` at New York.

```
varsiteadmin create -database nydb -name NewYork -address nyser-
ver:6000
```

   - Add site information about the new Paris site.

```
varsiteadmin add -database nydb -name Paris -address parisser-
ver:7000
```

4. Start event daemon of the database `nydb` at New York.
   - Start `veddriver ny.cfg`

     where `ny.cfg` is the name of the configuration file for the event daemon.

5. Create replication channel to monitor instances of all the classes at the New York site.

```
varchadmin create -name AllObjects -database nydb -site Paris -owner
NewYork -class all
```

6. Start replication processes at the New York site.

```
varsiteadmin start -database nydb -config nysite.cfg
```

   The file nysite.cfg is the configuration file for the replication processes at New York.

7. Start replication processes at the Paris site.

```
varsiteadmin start -database parisdb -name Paris -config parissite.cfg
```

   The file `parissite.cfg` is the configuration file for the replication processes at Paris.

   At this point, any application creating, updating or deleting any objects at the New York site will trigger asynchronous replication of those objects to the Paris site.

8. Set passive state of the explicit replication at New York site.

```
varsiteadmin explicit -database nydb -config nysite.cfg -state passive
```

   The file `nysite.cfg` is the configuration file for the replication processes at New York.

This will accumulate changes being done at New York Site.

9. Change state of explicit replication to active at New York site when changes at New York site needs to be replicated to Paris site.

   ```
   varsiteadmin explicit -database parisdb -config nysite.cfg -state active
   ```

   The file `nysite.cfg` is the configuration file for the replication processes at New York.

   At this point, a group of messages will be created and sent to Paris site. Each message will contain snapshot of 1000 object by default, which were modified when explicit replication was in passive state.

Conflict detection and resolution in Advanced Replication

Suppose, we need to set up two sites, one at New York, and another at Paris with changes happening at both the sites. New York site is assigned as the owner where conflicts are detected and resolved. To keep the sites in sync after a conflict is detected, Compensating Transaction resolver is used.

Sites:

Site	Database name	Host name	Port number
New York	nydb (existing)	nyserver	6000
Paris	parisdb (new)	parisserver	7000

Channels:

Channel name	Owner	Destinations	Criteria
AllObjects	New York	Paris	Instances of all the classes

1. Set up site at Paris
   - Create and set-up the database at Paris using Versant ODBMS utilities.

     ```
     makedb -g parisdb
     createdb parisdb
     ```

- Copy information from the database at New York to the one at Paris using Versant ODBMS utility vcopydb

  ```
  vcopydb -nocreate nydb@nyserver parisdb@parisserver
  ```

NOTE:- The Versant ODBMS utility vstream can also be used. However, it requires a transfer of files between sites, using ftp for example. The advantages are that it does not need Versant connection across sites.

For more information on vstream utility, please refer to Chapter "Database Utilities" in the *Versant Database Administration Manual.*

- Create site information in the database `parisdb` at Paris.

  ```
  java com.versant.VARSiteAdmin create -database parisdb -name Paris -address parisserver:7000
  ```

- Add site information about the new Paris site.

  ```
  varsiteadmin add -database parisdb -name NewYork -address nyserver:6000
  ```

2. Start the event daemon of the database at Paris.
   - Start `veddriver paris.cfg`

     where `paris.cfg` is the name of the configuration file for the event daemon.
3. Create replication channel to monitor instances of all the classes at the Paris site.

   ```
   varchadmin create -name AllObjects -database parisdb -site Paris -owner NewYork -class all
   ```

4. Set up site at New York.
   - Create site information in the database nydb at New York.

     ```
     varsiteadmin create -database nydb -name NewYork -address nyserver:6000
     ```

   - Add site information about the new Paris site.

     ```
     varsiteadmin add -database nydb -name Paris -address parisserver:7000
     ```

5. Start event daemon of the database `nydb` at New York.
   - Start `veddriver ny.cfg`

     where `ny.cfg` is the name of the configuration file for the event daemon.
6. Create replication channel to monitor instances of all the classes at the New York site.

   ```
   varchadmin create -name AllObjects -database nydb -site Paris -owner NewYork -class all
   ```

7. Start replication processes at the New York site.

   ```
   varsiteadmin start -database nydb -config nysite.cfg
   ```

   The file `nysite.cfg` is the configuration file for the replication processes at New York.

   The `nysite.cfg` must start conflict resolver by specifying the following in the configuration file.

   ```
   ConflictDetection=true
   ConflictCheckPeriod=60
   ConflictResolverClass=com.versant.var.conflict.VARCompensat-
   ingTxResolver
   ```

8. Start replication processes at the Paris site.

   ```
   varsiteadmin start -database parisdb -name Paris -config paris-
   site.cfg
   ```

   The file `parissite.cfg` is the configuration file for the replication processes at Paris.

   At this point, any application creating, updating or deleting any objects at the New York or Paris site will trigger asynchronous replication of those objects to the other site.

9. Create conflicts by updating same set of objects simultaneously at both the sites.

   The conflict handler running at New York site will detect the conflict and will create compensating messages for the conflicting messages. Use statistics command of the `varsiteadmin` tool to find out how many conflicts were detected.

   ```
   varsiteadmin statistics -database parisdb -name Paris -config
   parissite.cfg
   ```

10. Use varcompare utility to check whether both the sites are in sync.

    ```
    varcompare nydb@nyserver   parsdb@parisserver
    ```

LICENSE DETAILS

Release 6.0.5 onwards, Versant Asynchronous Replication is licensed as a separate component.

To use Versant Asynchronous Replication on Versant Databases the user will have to acquire the license for the Versant databases between which the replication is going to occur.

Hence the replicating database and the replica database installations will need the Versant Asynchronous replication license.

CHAPTER 2

VAR Reference

This Chapter gives detailed explanation about the Versant Asynchronous Replication configuration, components and utilities.

The Chapter covers the following in detail:

- Configuration
- Utilities Reference
- Components
- Interface Reference
- Customization
- Logging

VAR CONFIGURATION

The following is a reference to the possible configuration file entries organized by logical function and a sample config file is also presented.

Logging specific parameters:

`LogLevel`

Determines the verbosity of logging. Errors and warnings are always logged.

The log level can have the value 2, 3 or 4. The default value is 3.

```
LogLevel=2 (only Errors and Warnings)
LogLevel=3 (Terse debug),
LogLevel=4 (Verbose debug)
```

`LogDirectory`

Specifies the location of the log files. This can be specified as either an absolute or a relative path.

The default is the current working directory. Note that the backslash character (\) should be escaped with another backslash character (\).

For example,

```
LogDirectory=D:\\VAR\\Logs
```

Replication specific parameters:

`AdvancedReplication`

Enables advanced replication.

The default is `false`.

For example,

```
AdvancedReplication=true
```

`MaxBatchSize`

Specifies the maximum number of requests that can be made in a batch during batch replication.

The default value is `1`.

For example,

```
MaxBatchSize=10
```

MaxBatchPeriod

Specifies the maximum duration in seconds between two consecutive replication requests.

The default value is `0`.

For example,

```
MaxBatchPeriod=60
```

RequestRetryInterval

Determines how frequently the request processor polls the database to get the next transaction replication request. Specifies the maximum interval between request retries in milliseconds.

This parameter is also used to wait once all requests have been processed. The wait time is a random number of milliseconds inside this interval.

The default value is `10000` milliseconds.

For example,

```
RequestRetryInterval=60000
```

RMIRegPort

Specifies a particular port for RMI registry.

This port should be passed as a parameter while running the `varstatrmi` utility.

The default value is `1099`.

For example,

```
RMIRegPort=5065
```

UseGroupRead

Enables a group read operation for faster replication. In case of lock conflicts, objects will be read one by one.

`UseGroupRead` will improve performance when there are no lock conflicts and none of the objects have been deleted. However, in case of heavy lock conflicts, the performance will remain the same as earlier. So you should choose to set `UseGroupRead` to true only when you know that there will be no or few conflicts.

The default setting is false.

For example,

```
UseGroupRead = true
```

Java specific parameters:

`JavaVM`

Specifies the Java VM to be used for starting up replication processes.

The default is '`java`'.

Note that the backslash character (\) should be escaped with another backslash character(\).

For example,

```
JavaVM=D:\\jdk\\bin\\java
```

Options:

`JavaVMOptions`

Options to be used when invoking the Java VM.

The default value is `-Xmx128m.`

For example,

```
JavaVMOptions=-Xmx32m
```

Process management parameters:

`MaxRestarts`

Specifies the maximum number of process restarts. Determines the number of times the replication server driver attempts to restart a failed process.

Determines the number of times the `VARVDBReplicator` attempts to reconnect to the slave databases for basic replication. After each failing reconnect attempt the `VARVDBReplicator` sleeps `MaxReconnectInterval` seconds.

The default value is `3`.

For example,

```
MaxRestarts=5
```

`MaxLockRetries`

Specifies the maximum number of retries made by replication server driver before giving up on getting a lock on a `VARTxRequest`.

The default value is `3`.

For example,

```
MaxLockRetries=5
```

`MaxLockRetryInterval`

Specifies the maximum time interval in seconds before retrying to acquire the lock after the lock wait timeout occurs on an object.

The wait time is a random number of seconds inside this interval.

The default value is `5`.

For example,

```
MaxLockRetryInterval=10
```

`StartTxRequestProcessor`

Parameter to enable/disable request processor (Boolean).

The default is `true`.

For example,

```
StartTxRequestProcessor=false
```

StartTxMessageSender

Parameter to enable/disable message sender (Boolean).

The default is `true`.

For example,

```
StartTxMessageSender=false
```

StartTxMessageApplier

Parameter to enable/disable message applier (Boolean).

The default is `true`.

For example,

```
StartTxMessageApplier=false
```

RetryAllErrors

Retries if any error occurs on the slave site. Once set to true, this will force replication server driver to attempt to recover from any kind of exception. If set to false the replication will retry only for the handled network errors and lock errors.

This parameter is used in combination with `MaxRestarts` and `MaxLockRetries`.

The default is `false`.

For example,

```
RetryAllErrors=true
```

Sender specific parameters:

MaxReconnectInterval

Specifies the maximum interval in number of seconds between attempts to reconnect to a site.

The default is `30` seconds.

For example,

```
MaxReconnectInterval=60
```

SenderMaxCommitSize

Specifies the maximum messages sent during a commit-batch.

The default value is `1`.

For example,

```
SenderMaxCommitSize=50
```

SenderMaxCommitPeriod

Specifies the maximum duration of a commit-batch in seconds.

The default is `0` seconds.

For example,

```
SenderMaxCommitPeriod=60
```

Receiver specific parameters:

ReceiverMaxCommitPeriod

Specifies the maximum duration of a commit-batch in seconds.

The default is `0` seconds.

For example,

```
ReceiverMaxCommitPeriod=60
```

ReceiverMaxCommitSize

Specifies the maximum messages that can be received during a commit-batch.

The default is `1`.

For example,

```
ReceiverMaxCommitSize=10
```

Default Reader specific parameters:

ReaderMaxLockRetries

Specifies the maximum number of retries for locking objects with RLOCK.

The default value is -1 i.e., infinite.

For example,

```
ReaderMaxLockRetries=3
```

ReaderUseNoLock

Determines whether to use RLOCK or NOLOCK to lock the objects to be replicated.

Using NOLOCK i.e., ReaderUseNoLock = true allows the objects from the source database with NOLOCK to be read. This avoids heavy lock conflicts and wait times for the replication on a heavy updating or long running transaction.

The default is false which implies that RLOCK will be applied on all created and modified objects.

For example,

```
ReaderUseNoLock=true
```

Default Writer specific parameters:

WriterUseStrictMode

Specifies whether to use strict mode or not.

The default is false.

For example,

```
WriterUseStrictMode=true
```

Default Transport specific parameters:

TransportReadTimeoutPeriod

Specifies period in seconds after which read will timeout.

The default value is `600`.

For example,

```
TransportReadTimeoutPeriod=300
```

Conflict detection specific parameters:

`ConflictDetection`

Specifies whether to use conflict detection mode or not.

The default is `false`.

For example,

```
ConflictDetection=true
```

`ConflictCheckPeriod`

Specifies the period in seconds for which replication-messages are used for conflict detection.

The default value is `30`.

For example,

```
ConflictCheckPeriod=60
```

Customization specific parameters:

`ConflictDetectorClass`

Name of conflict detection class.

The default is `com.versant.var.conflict.VARDefaultConflictDetector`.

For example,

```
ConflictDetectorClass=com.versant.var.conflict.VARDefaultConflictDe
tector
```

`ConflictResolverClass`

Name of conflict resolution class.

The default is `com.versant.var.conflict.VARDefaultConflictResolver`.

For example,

```
ConflictResolverClass=com.versant.var.conflict.VARDefaultConflictRe
solver
```

`ConflictResolverClass`

A compensating transaction can be generated and sent to slave site by using a class that generates compensating transaction.

For example,

```
ConflictResolverClass=com.versant.var.conflict.VARCompensatingTxRe
solver
```

`MessageFactoryClass`

Name of Message factory class - responsible for creating an instance of `txMessage`.

The default is `com.versant.var.message.VARDefaultMessageFactory`.

For example,

```
MessageFactoryClass=com.versant.var.message.VARDefaultMessageFactory
```

`ObjectReaderClass`

Name of Object reader class - responsible for reading in objects from the user database.

The default is `com.versant.var.message.VARDefaultObjectReader`.

For example,

```
ObjectReaderClass=com.versant.var.message.VARDefaultObjectReader
```

`ObjectWriterClass`

Name of Object writer class - responsible for writing objects to the user database.

The default is `com.versant.var.message.VARDefaultObjectWriter`.

For example,

```
ObjectWriterClass=com.versant.var.message.VARDefaultObjectWriter
```

`TransportClass`

Name of class implementing transport interface - responsible for providing transport mechanism.

The default is `com.versant.var.transport.VARDefaultTransport`.

For example,

```
TransportClass=com.versant.var.transport.VARDefaultTransport
```

Configuration File

The configuration file is a Java property file that specifies the parameters used by the replication process and utilities. The following is a sample configuration file:

```
# Log level
# Determines the verbosity of logging. Errors and warnings are always
# logged.(2 (only Errors and Warnings)Default: 3 (Terse debug),
# Max: 4 (Verbose debug)
LogLevel=4


# Log directory
# Specifies the location of the log files (Default: Current working
# directory. Can be specified as either absolute or relative path)
# Note that backslash (\) should be escaped with another backslash
# (\).
LogDirectory=D:\\VAR\\Logs


# Java VM to be used for starting up replication processes
# (Default: 'java')
# Note that backslash (\) should be escaped with another backslash
# (\).
JavaVM=D:\\jdk\\bin\\java
```

```
# Java VM Options
# Options to be used when invoking a Java VM
# (Default: -Xmx128m)
JavaVMOptions=-Xmx32m
# Select a particular port for RMI registry
# (Default: 1099)
# varstatrmi utility should be run passing this port as a parameter.
RMIRegPort=5065

# Batch replication
# max number of requests in a batch
# (Default: 1)
# MaxBatchSize=10
# maximum duration in seconds between two consecutive replication
# (Default: 0)
# MaxBatchPeriod=60

# Maximum time in milliseconds to retry a request
# Interval between request retries. This parameter is also used to
# wait once all requests have been processed.
# The wait time is a random number of milliseconds inside this inter
# val.
# (Default: 10000)
RequestRetryInterval=60000

# enable Advanced replication
# (Default: false)
# AdvancedReplication=true

# Following are the process management parameters

# Parameter to enable/disable request processor (Boolean)
# (Default: true)
StartTxRequestProcessor=false
# Parameter to enable/disable message sender (Boolean)
# (Default: true)
StartTxMessageSender=false

# Parameter to enable/disable message applier (Boolean)
# (Default: true)
```

```
StartTxMessageApplier=false

# Maximum number of process restarts
# Determines the number of times the replication server driver
# attempts to restart a failed process.
# For basic replication determines the number of times the VARVDBRep
# licator attempts to reconnect to the slave databases.
# After each failing reconnect attempt the VARVDBReplicator sleeps
# MaxReconnectInterval second.
#(Default: 3)
MaxRestarts=5

# Retries if any error occured on the slave site.
# Once set to true, this will force replication server driver to
# attempt to recover from any kind of exception.
# If set to false the replication will retry only for the handled net
# work errors and lock errors.
# This parameter is used in combination with MaxRestarts and
# MaxLockRetries.
# (Default: false)
RetryAllErrors=true

# Maximum number of retries made by replication server driver to
# before giving up on getting a lock on a VARTxRequest
# (Default: 3)
MaxLockRetries=5

# Maximum time interval in seconds before retrying to
# acquire the lock after the Lock wait time out occurs on an object.
# (Default : 5)
MaxLockRetryInterval=10

# Sender specific parameters

# Max reconnect interval
# Specifies the maximum interval in number of seconds between attempts
# to reconnect to a site. (Default:30 sec)
MaxReconnectInterval=60

# maximum messages sent during a commit-batch
# (Default : 1)
```

```
SenderMaxCommitSize=50

# maximum duration of a commit-batch in seconds
# (Default : 0 sec)
# SenderMaxCommitPeriod=60

# Receiver specific parameters

# maximum messages received during a commit-batch
# (Default : 1)
# ReceiverMaxCommitSize=10

# maximum duration of a commit-batch in seconds
# (Default : 0 sec)
# ReceiverMaxCommitPeriod=60

# Default Reader specific parameters
# maximum number of retries for locking objects with RLOCK
# (Default : -1 i.e infinite)
# ReaderMaxLockRetries=3
# (Default : false)
# ReaderUseNoLock=true

# Default Writer specific parameters

# Whether to use strict mode
# (Default: false)
# WriterUseStrictMode=true

# Default Transport specific parameters

# Specifies period in seconds after which read will timeout
# (Default: 600)
# TransportReadTimeoutPeriod=300
# Conflict detection specific parameters

# Conflict detection mode
# (Default: false)
ConflictDetection=true
```

```
# Period in seconds for which replication-messages are used
# for conflict detection
# (Default: 30)
ConflictCheckPeriod=60

# Customization specific parameters

# Name of Message factory class - responsible for creating an
# instance of txMessage.
# (Default: com.versant.var.message.VARDefaultMessageFactory)
MessageFactoryClass=com.versant.var.message.VARDefaultMessageFactory

# Name of class implementing transport interface - responsible for
# providing transport mechanism
# (Default: com.versant.var.transport.VARDefaultTransport)
TransportClass=com.versant.var.transport.VARDefaultTransport

# Name of Object reader class - responsible for reading in objects
# from the user database.
# (Default: com.versant.var.message.VARDefaultObjectReader)
ObjectReaderClass=com.versant.var.message.VARDefaultObjectReader

# Name of Object writer class - responsible for writing objects to the
# user database.
# (Default: com.versant.var.message.VARDefaultObjectWriter)
ObjectWriterClass=com.versant.var.message.VARDefaultObjectWriter

# Name of conflict detection class
# (Default: com.versant.var.conflict.VARDefaultConflictDetector)
ConflictDetectorClass=com.versant.var.conflict.VARDefaultConflict
Detector

# Name of conflict resolution class
# (Default: com.versant.var.conflict.VARDefaultConflictResolver)
ConflictResolverClass=com.versant.var.conflict.VARDefaultConflict
Resolver
# A compensating transaction can be generated and sent to
# slave site by using a class that generates compensating
# transaction.

# ConflictResolverClass=com.versant.var.conflict.VARCompensatingT
```

```
# xResolver
```

VAR UTILITY REFERENCE

Most of the utility commands can be invoked only when replication is off. Commands, such as `list`, that can be invoked either offline or online are specifically noted.

Replica site initialization utility

varinitdb

`varinitdb dbname`

This utility loads the Versant Asynchronous Replication schema into the replication repository (the user database that is specified as the parameter) and initializes the replication queues. You cannot use this product to replicate between sites without first running this utility for each of the databases involved in the replication.

For example:

```
varinitdb mydb
```

Channel administration utility

varchadmin

`varchadmin command parameters`

Replication can be configured and administered using the channel administration utility. This utility creates updates, destroys, disables, enables or lists channels. It is invoked at the command line.

The following are `commands` and their `parameters`:

compare

`varchadmin compare {required parameters}`

Compares channels.

This command compares the channel definitions in databases at multiple sites and reports the differences.

This command can be invoked either online or offline.

Required parameters are:

`-name channelname`

Specifies the channel name.

`-database "db1 db2..."`

Specifies the database names.

create

`varchadmin create {required parameters} [optional parameters]`

Creates a channel.

Required `parameters` are:

`-name channelname`

Specifies the channel name.

`-database dbname`

Specifies the database name.

`-site "site1 site2..."`

Specifies a list of sites.

An optional `parameter` to assign ownersite to a channel is:

`-owner ownername`

Specifies the name of the owner site.

Optional parameters to create an event-based channel (you can use only one of the following):

`-class "class1 class2..."`

Specifies a list of classes.

OR

-class all

Specifies all user classes including VAssociate and VHashtable.

OR

-query "vql query string"

Specifies a query predicate for selecting specific objects.

OR

-object "loid1 loid2..."

A set of objects specified by their logical object identifiers.

If none of the optional parameters for creating an event-based channel are specified, then a non-event-based channel is created.

An event-based channel is created to either monitor instances of one or more classes or, objects qualifying one or more VQL queries, or for a given set of objects specified using their LOIDs. In order to monitor instances of all the classes, you should specify `all` instead of a list of class names.

For example (entered on one line) to create a class based channel consisting of all user classes:

```
varchadmin create -name mychannel -database -mydb -site server_site
-class all
```

disable

```
varchadmin disable {required parameters}
```

Disables a channel.

This command can be invoked either online or offline.

This command can be used only for an event-based channel.

Required `parameters` are:

-name channelname

Specifies the channel name.

-database dbname

Specifies the database name.

enable

`varchadmin enable {required parameters}`

Enables a channel.

This command can be used for only an event based channel.

This command can be invoked either online or offline.

Required `parameters` are:

-name channelname

Specifies the channel name.

-database dbname

Specifies the database name.

help

`varchadmin help`

Provides help.

`varchadmin command -help`

Provides additional help for a specified command.

list

`varchadmin list {required parameters } [ optional parameters]`

Lists channels.

The `list` command can be invoked either online or offline.

Required `parameters` are:

-database dbname

Specifies the name of the user database.

Optional parameters are:

-name channelname

Lists only the specified channel.

For example:

```
varchadmin list -database mydb
varchadmin list –database mydb -name mychannel
```

remove

```
varchadmin remove {required parameters}
```

Removes a channel.

Required parameters are:

-name channelname

Specifies the channel name.

-database dbname

Specifies the database name.

update

```
varchadmin update {required parameters} [optional parameters]
```

Updates a channel.

Required `parameters` are:

-name channelname

Specifies the channel name.

-database dbname

Specifies the database name.

Optional `parameters` are:

For a class based channel:

`-addclass "class1 class2..."`

Specifies a list of classes to be added.

`-removeclass "class1 class2..."`

Specifies a list of classes to be removed.

For a query based channel:

`-query "vql query string"`

Specifies a query predicate for selecting specific objects.

For an object based channel:

`-addobject "loid1 loid2..."`

Specifies a set of objects to be added.

You cannot use this option with the `-query` option.

`-removeobject "loid1 loid2..."`

Specifies a set of objects to be removed.

You cannot use this option with the `-query` option.

To update a destination site:

`-addsite "site1 site2..."`

Specifies a list of sites to be added.

`-removesite "site1 site2..."`

Specifies a list of sites to be removed.

To assign or change the owner site of a channel:

`-owner ownername`

Specifies the owner site name.

Site Administration Utility

This utility creates, updates, removes, starts, stops or lists replication sites. This utility can also be accessed by JAVA API using class `com.versant.var.VARSiteAdmin`. Detailed description of how to use this API is available in javadoc.

varsiteadmin

```
varsiteadmin command parameters
```

This Java utility creates, updates, removes, starts, stops or lists replication sites.

The following are commands and their parameters.

add

```
varsiteadmin add {required parameters}
```

Adds a site.

This command adds information of a new site to another site. The name and address of a particular replica site should be the same for all the replica sites. This applies to peer-to-peer configuration and not to a master-slave configuration.

Required `parameters` are:

`-database dbname`

Specifies the database name.

`-name sitename`

Specifies the site name.

`-address address`

Specifies the site address.

create

```
varsiteadmin create {required parameters}
```

Creates a site.

This command initializes a new site by creating site information in the replication-repository at the new site.

Required `parameters` are:

`-database dbname`

Specifies the database name. The default is the user database.

`-name sitename`

Specifies the site name.

`-address address`

Specifies the site address.

help

`varsiteadmin help`

Provides help.

`varchadmin command -help`

Provides additional help for a specified command.

list

`varsiteadmin list {required parameters} [optional parameters]`

Lists sites.

This command lists information of all sites or the given site. It can be invoked either online or offline.

Required `parameters` are:

`-database dbname`

Either the name of the database to be kept as a replica of a source database or the name of the channel used to address the replica database.

Optional `parameters` are:

-name sitename

Specifies the name of the site.

For example, to list all sites:

```
varsiteadmin list -database mydb
```

To list a particular site:

```
varsiteadmin list –database mydb -name mychannel
```

statistics

```
varsiteadmin statistics {required parameters} [optional parameters]
```

Displays statistics.

This command displays the statistics for the replication queues at the local site. This command can be invoked either online or offline.

Required `parameters` are:

-database dbname

Either the name of the database to be kept as a replica of a source database or the name of the channel used to address the replica database.

Optional `parameters` are:

-config configuration_file

If the configuration file is not specified default configuration will be used. Therefore, config file must be specified if Advanced mode is used.

status

```
varsiteadmin status {required parameters} [optional parameters]
```

Displays status.

This command displays the status of the replication server processes and the modules at the local site. The command also displays the resource usage of the replication server processes running at the local site. This command can be invoked either online or offline.

Required `parameters` are:

-database dbname

Database name.

Optional `parameters` are:

-config configuration file

Specifies the configuration file.

If the configuration file is not specified default configuration will be used. Therefore, config file must be specified if Advanced mode is used.

-module module_name

Specifies the name of the module.

remove

```
varsiteadmin remove {required parameters}
```

Removes a site.

This command removes the details of a particular site from the replication repository on the specified site.

Required `parameters` are:

-database dbname

Specifies the database name.

-name sitename

Specifies the site name.

start

```
varsiteadmin start {required parameters} [optional parameters]
```

Starts replication at local site.

The command starts replication process at a site. Before the application starts updating the database, the event daemon and the database should be started simultaneously so that no events are lost.

Required `parameters` are:

`-database dbname`

Specifies the database name.

Optional `parameters` are:

`-config configuration_file`

Specifies the configuration file.

The configuration file contains the details of customization of transport, messaging and conflict and resolution. If not specified the default configuration will be used. Therefore `config. file` must be specified if Advanced mode is used.

stop

```
varsiteadmin stop {required parameters} [optional parameters]
```

Stops replication at local site.

This command stops replication processes at a site but does not stop the event daemon. The process can be invoked from a different site. Event daemon should be stopped using the Versant utility `stopdb`, so that no events are lost.

Required `parameters` are:

`-database dbname`

Specifies the database name.

Optional `parameters` are:

`-config configuration_file`

Specifies the configuration file.

If the configuration file is not specified default configuration will be used. Therefore config. file must be specified if Advanced mode is used.

update

```
varsiteadmin update {required parameters}
```

Updates a site.

This command updates information of the given site. Updating involves changing the address of all the replica sites.

Required `parameters` are:

`-database dbname`

Specifies the database name.

`-name sitename`

Specifies the site name.

`-address address`

Specifies the site address.

enable

`varsiteadmin enable (required parameters)`

Enables a site.

This command disables the given site and is valid only for Advanced mode. The replication sender process will start sending messages after the site is enabled.

Required `parameters` are:

`-database dbname`

Specifies the database name.

`-name sitename`

Specifies the site name.

disable

`varsiteadmin disable (required parameters)`

Disables a site.

This command disables the given site and is valid only for Advanced mode. The replication sender process will stop sending messages after the site is disabled.

Required parameters are:

-database dbname

Specifies the database name.

-name sitename

Specifies the site name.

explicit

```
varsiteadmin explicit (required parameters) [optional parameters]
```

Turns explicit replication `on` or `off`.

This command sets explicit replication on or off at the given site and is valid only for Advanced mode, in Master-Slave configuration.

Required `parameters` are:

-database dbname

Specifies the database name.

The optional `parameters` are:

-state {active/passive}

Specifies active/passive to set the explicit replication state to on/off respectively.

-config configuration_file

Specifies the name of the configuration file.

Specify absolute or relative path to the configuration file for advanced replication.

Utility to check and fix meta-data

varcheckmeta

```
VARCheckMeta -database <dbname> [{-repairMsgQ|-repairCircularQ|
```

```
                                                    -repairReqQ}]
```

This command scans the replication request queue and replication message queue for consistency and prompts to re-run with additional options if it detects any in-consistency with metadata. This utility should be run in passive mode. All other replication processes should not be running during this time.

Required `parameters` are:

`-database dbname`

Specifies the database name.

Optional `parameters` are:

`-repairCircularQ`
`-repairMsgQ`
`-repairReqQ`

These options should be specified only if prompted by the `varcheckmeta` utility.

Replica-site comparison utility

varcompare

```
varcompare {classes class1 class2..}[db1][db2]
```

This command compares the objects by identity and value of two databases at the replica sites. The command also lists the LOIDs and class names of objects:

- found only in single site of the database or
- found in both the sites but with a different value

The databases names are `db1,db2` and `class1,class2` are the class names of the objects to be compared. If the class objects are not specified, objects of all classes except those of the database system classes and the ones used by the asynchronous replication servers are compared. This command can be invoked either online or offline.

Meta information cleanup utility

varrmmeta

```
varrmmeta dbname
```

This command removes Asynchronous Replication related meta data from the database. The command is useful when user wants to reinitialize Asynchronous Replication.

Event Daemon Utility

veddriver

```
veddriver dbname cfgfile
```

Starts the event daemon. If the configuration file is not specified on command line then set environment variable `VED_CONFIGFILE` to specify name and location of the config. file.

For stopping the event daemon, use `vedstopdriver` utility.

The configuration file must include the location of `varrequesteng.dll` (request engine DLL) or `varrequesteng.so` (shared library).

The following example illustrates the format of the configuration file:

Entries	Explanation
`ChannelServicePort 6666`	The channel management service port number (Default:5030)
`LogFile ved.log`	The log file name. This file is a text file. There is no default: logging is turned off if not specified.

`LogLevel 2`	The details of replication progress, warnings and errors during the replication process. Log level: 0-Log errors and warnings 1-Terse debug 2-Verbose debug

`<EngineLibs>`
`varrequesteng.dll`
`</EngineLibs>`
	Tags to indicate the paths to event processing DLLs or shared libraries to be loaded. The replication-request creation engine must be specified.

Event Daemon Notification to Clients

The database clients might want to know if the event daemon is not running. Depending upon the application requirements the database can be configured to notify the clients/DBA about event daemon's existence.

For every commit, the database server checks for the existence of the event daemon. If the event daemon is down, it takes different actions based on the setting of the following database server process profile parameter;

```
event_daemon_notification off/on
```

`off` - Writes the error `6524`, `EV_DAEMON_NOT_RUN` to the database LOGFILE but doesn't abort the commit. This is the default behavior.

`on` - Aborts the commit and throws the error `6524`, `EV_DAEMON_NOT_RUN` to the application in addition to writing the error to the database LOGFILE.

Version Checking Utility

com.versant.var.Version

This utility displays the version information of VAR.

```
java com.versant.var.Version
```

Utility to repair sender queues

varsenderrepair

`VARSenderRepair dbname`

This utility is used to repair the sender queue i.e. the message queue at the sender site. This utility should be rarely used.

This utility should be run in passive mode. All other replication processes should not be running during this time. This utility could be run if it is found that the statistics for messages sent do not match with those of the number of requests processed at that site or in situations where the meta data could get corrupted for example frequent starts and stops of VARTxMessageSender.

Required `parameters` are:

`-database dbname`

Specifies the database name.

VAR COMPONENTS

Default Replication

Replication Event Daemon

- The Replication Event Daemon is the Replication Request Engine. It can be configured to run as a daemon or as a standalone process.

The Replication Event Daemon reads Versant events, creates replication requests, and puts them on the persistent request queue.

Replication Process

- Runs as a standalone process.
- Can be administered using the administration tool.
- Reads replication requests from the replication repository and replicates objects to destination sites.
- Logs errors and messages.

Advanced Replication

Replication Event Daemon

- The Replication Event Daemon is the Replication Request Engine. It can be configured to run as a daemon or as a standalone process.

The Replication Event Daemon reads Versant events, creates replication requests and puts them on the persistent request queue.

Replication Request Processor

- Runs as a standalone process. By default it is enabled, can be disabled if required.
- Reads replication requests from the replication repository, converts them to replication messages using the Object Reader interface and puts them on the persistent message queue.
- Logs errors and messages.

Message Receiver

- Runs as a thread within a driver process. The receiver cannot be disabled.
- Receives messages from replication servers, unpacks the messages into Versant objects and puts the replication messages on the persistent message queue.
- The Message Receiver also receives administrative messages, such as a STOP message from the Site administration utility.
- Logs errors and messages.

Replication Message Sender

- Runs as a standalone process (Java VM). The sender is enabled by default can be disabled if required.
- Reads the queued replication messages from the replication repository.
- Sends replication messages to the destinations specified for the channels.
- Logs errors and messages.

Replication Message Applier

- Runs as a standalone process. The Replication Message Applier is enabled by default, can be disabled if required
- Reads replication messages from the message queue in replication repository and applies changes to the local user database using the Object Writer Interface.
- Logs errors and messages.

Conflict Handler Process

- Runs as a standalone process. The conflict handler is disabled by default. It can be enabled by setting the configuration parameter `conflictDetection` to `true`.
- Reads replication messages from the message queue in replication repository and detects and resolves conflicts using conflict detection and resolution APIs.
- Logs errors and messages.

VAR INTERFACE REFERENCE

The Developer uses Versant Asynchronous Replication APIs to:

- identify, in an application, the objects to be replicated.
- customize the default implementation of conflict detection and conflict resolution.
- customize the default implementation of the object reader and object writer.

Identifying Objects

Use the replication interface to identify objects to be replicated.

A replication request represents a request to replicate a set of objects transactionally to the list of destinations for a channel.

The Replication APIs can be used to create a replication request. You can specify the set of objects to be created, modified or deleted in the replication request. Every replication request is associated with a specific channel. The Administrator should have specified the list of destinations for the channel. The Request Creation interface is available in C++ and Java.

In C++, you can use the class `VARTxData` to specify the objects that have been created, modified or deleted and need to be replicated over a channel. Please refer to the documentation for the C++ APIs for details on the usage and methods on this class.

In Java, you can use the class `com.versant.var.VARTxData` to specify the objects that have been created, modified or deleted and need to be replicated over a channel. Please refer to the documentation for the Java API (javadoc) for details on the usage and methods on this class.

In the demo directory, there are sample programs in C++ and Java that illustrates the use of the API to create replication requests.

VAR CUSTOMIZATION

The Versant Asynchronous Replication product can be customized in various ways. You can choose to customize how update conflicts are detected and resolved. You can also customize the object reader and object writer to replicate only the modified attributes of objects (instead of the default behavior, which is to send the entire object).

The customization is achieved by implementing the published APIs and specifying the name of the class that implements the interface in the configuration file.

NOTE:- The changes you make to the configuration file take affect only when the replication processes are restarted.

Customizing Update-Conflict Detection

You can customize conflict detection by implementing the interface `com.versant.var.conflict.VARIConflictDetector`. You can change the criteria, which constitute a conflict. One example is to use the source timestamp of transactions to determine the order of occurrence and use that information for conflict detection and resolving. Conflict resolution can be customized by implementing the interface `com.versant.var.conflict.VARIConflictResolver`.

You would also have to update the Versant Asynchronous Replication configuration file for each of the replica sites. In the configuration file, you would need to specify the entries for the `ConflictDetectorClass` and the `ConflictResolverClass`. To enable conflict detection the parameter ConflictDetection should be set to true. The following illustration shows a subset of the configuration file.

```
# Conflict detection mode
# (Default: false)
ConflictDetection=true

# Period in seconds for which replication-messages are retained
# for conflict detection after they are processed.
# (Default: 30)
ConflictCheckPeriod=30

# Name of conflict detection class
# (Default: com.versant.var.conflict.VARDefaultConflictDetector)
ConflictDetectorClass=com.versant.var.conflict.VARDefaultConflictDe
tector
```

```
# Name of conflict resolution class
# (Default: com.versant.var.conflict.VARDefaultConflictResolver)
ConflictResolverClass=com.versant.var.conflict.VARDefaultConflictRe
solver
```

Customizing the Object Reader and Object Writer

You can customize the default Object Reader and Object Writer.

You might want to do this for various reasons. For example, if you have implemented a scheme to determine which specific attribute is modified (say, using a bitmap), you may want to replicate only the modified attributes (instead of the default which is to replicate the entire object). Another use could be to encrypt or mangle some sensitive attributes before replicating the objects.

In order to customize the Object Reader and Object Writer, you would implement the interfaces `com.versant.var.message.VARIObjectReader` and `com.versant.var.message.VARIObject-Writer` respectively.

The default Object Reader and the Writer use Versant Data Service or Versant package. (`com.versant.common.dataservice`). This package allows canonical representation of persistent objects and classes that can be serialized using Java Object Serialization. The Versant package is bundled with Versant Asynchronous Replication release. You may use the Versant package to implement custom Object Reader and Writer.

You would also have to update the Versant Asynchronous Replication configuration file for each of the replica sites. In the configuration file, you would need to specify the entries for the `ObjectReaderClass` and the `ObjectWriterClass`.

As an illustration, a subset of the configuration file is shown below:

```
# Name of Object reader class - responsible for reading in objects
# from the user database.
# (Default: com.versant.var.message.VARDefaultObjectReader)
# ObjectReaderClass=CustomObjectReader

# Name of Object writer class - responsible for writing objects to the
# user database.
# (Default: com.versant.var.message.VARDefaultObjectWriter)
ObjectWriterClass=CustomObjectWriter
```

VAR LOGGING

Versant Asynchronous Replication logs messages to the log files in the directory specified in the site configuration file. There is no centralized logging and the Administrator needs to view the log files at every site independently.

The level of logging specified in the configuration file determines what is logged. The levels are:

1 Errors
2 Warnings
3 Terse Debug
4 Verbose Debug

Logging level 3 (Terse Debug) is the default.

Errors and Warnings are always logged, so you can only set the logging level to 2, 3 or 4.

If you change this setting, you need to restart the replication processes.

The Administrator is responsible for the purging of the log files. It is recommended that the log files be purged when the replication processes have been stopped. You may lose some log messages if the log files are being purged when the system is operational.

Every module logs messages to a different log file. The names of the log files are:

Default Replication

`VDBReplicator.log`

This file logs messages from the replicator process.

Advanced Replication

`Driver.log`

This file logs messages from the Driver process. The Driver starts the other processes. You should check this log to see if any of the process exited abnormally. It also gives information on attempts to restart failed process.

`MessageReceiver.log`

This file logs replication messages from other sites and supervision messages from the site administration utility `varsiteadmin`. Errors and warnings are also logged here. You should check this log to ensure that there are no failures while the message receiver is being started up. It would be useful to monitor this log when a replica site is a slave site in master-slave configuration or a receiving site in a peer-to-peer configuration.

`TxRequestProcessor.log`

This file logs the progress of creation of replication messages from queued replication requests. Errors and warnings are also logged here. This log is especially useful if a replica site is a master site in a master-slave configuration, or if, it is one of the source sites in a peer-to-peer configuration.

`TxMessageApplier.log`

This file logs the progress of the replication message applier and its errors and warnings. You should monitor this log when a replica site is a destination site in a peer-to-peer or master-slave configuration.

`TxMessageSender.log`

This file logs the progress of the replication message sender, it's errors and warnings. You should monitor this log when a replica site is a master site in a maser-slave configuration or source site in a peer-to-peer configuration.

`ConflictHandler.log`

This file logs potential conflicts detected by the conflict handler. You should monitor this log continuously when conflict detection is turned on. The default conflict resolver also logs detailed information about conflicts.

`TxMessageCleanup.log`

This file logs the progress of the replication message cleanup module and its warnings and errors.

Log Format

Message Identifier

Every time the replication processes start, the identifier is initialized to 0. It is sequentially incremented for every message in the log file.

TimeStamp

The date and time at which the log entry was created.

Log Level

Logging level used by the module.

Log Message

This is the actual message.

Typically, log messages have the sequence number and the transaction identification of the transaction being replicated.

For example: `[0, 9.0.32834]` in a log message represents the sequence number `0` and transaction identification number `'9.0.32834'`.

The sequence number indicates the position of the replication message in the replication message queue at each site.

Logging entry in configuration file

An example of the entry in the configuration file for logging is listed below. You should make sure the `LogDirectory` specified exists and that there is adequate disk space for logging. If the `LogDirectory` does not exist, the log files will be placed in the current directory.

```
# Log level
# Determines the verbosity of logging. Errors and warnings are always
# logged.( 2 only errors and warnings)
# Default:3 (Terse debug) , Max: 4 (Verbose debug)
LogLevel=4

# Log directory. Specifies the location of the log files (Default:
# Current working directory. Can be specified as either absolute or
# relative path)
# Note that backslash (\) should be escaped with another backslash
 (\).
LogDirectory=D:\\VAR\\Logs
```

Sample log files

A few sample log files are listed. The files illustrate the log messages at different levels of logging.

The following sample log shows the TERSE_DEBUG messages in the log file MessageApplier.log :

```
1|Tue Oct 27 15:01:36 PST 1998|TERSE_DEBUG|Session initialized||
2|Tue Oct 27 15:01:36 PST 1998|TERSE_DEBUG|Initialized persistent
message queue||
3|Tue Oct 27 15:01:36 PST 1998|TERSE_DEBUG|Starting to peek at the
 queue at counter 0||
4|Tue Oct 27 15:05:22 PST 1998|TERSE_DEBUG|Applying [0, 9.0.32834]||
5|Tue Oct 27 15:05:24 PST 1998|TERSE_DEBUG|Applied 10 new, 0
modified, 0 deleted||
6|Tue Oct 27 15:05:24 PST 1998|TERSE_DEBUG|Applied [0, 9.0.32834]||
```

The following sample log shows the TERSE_DEBUG and VERBOSE_DEBUG messages in the log file. This example shows the stack trace for the exception encountered:

```
46444|Wed Nov 11 08:42:33 PST 1998|VERBOSE_DEBUG|CommitHandler
scanning commit vector ...||
46445|Wed Nov 11 08:42:38 PST 1998|VERBOSE_DEBUG|CommitHandler
scanning commit vector ...||
46446|Wed Nov 11 08:42:39 PST 1998|ERROR|Handler
Socket[addr=abhimanyu.versant.com
/10.254.10.106,port=1333,localport=7000][Open]
Stack trace:
java.net.SocketException: Connection reset by peer
  at java.net.SocketInputStream.read(Compiled Code)
  at java.net.SocketInputStream.read(Compiled Code)
  at java.io.DataInputStream.readInt(Compiled Code)
  at com.versant.var.receiver.VARMessageReceiver$Handler.run(Compiled
Code) ||
46447|Wed Nov 11 08:42:39 PST 1998|TERSE_DEBUG|Handler
Socket[addr=abhimanyu.versant.com
/10.254.10.106,port=1315,localport=7000][Open]
started. ||
46448|Wed Nov 11 08:42:40 PST 1998|TERSE_DEBUG|Handler
Socket[addr=abhimanyu.versant.com
/10.254.10.106,port=1333,localport=7000][Open]
leaving the session||
```

CHAPTER 3

Trouble Shooting Guide

This Chapter gives detailed explanation about Trouble Shooting.

The Chapter explains the following in detail:

- Environment
- Event daemon - veddriver
- Replication
- Licensing Error Messages

ENVIRONMENT

When you get errors when you try to run the utilities `varinitdb`, `varsiteadmin` or `varchadmin`, or the demos make sure that the environment variables have been set properly.

Set PATH environment variable to locate Java VM and the VODBMS binaries

For UNIX platforms

```
setenv PATH
$VERSANT_ROOT/jre/bin:$VERSANT_ROOT/bin:$PATH
```

For Windows

```
set
PATH=%VERSANT_ROOT%\jre\bin;%VERSANT_ROOT%\bin;%PATH%
```

Set CLASSPATH environment variable to locate the required JVI and VAR jar files

For UNIX Platforms

```
setenv CLASSPATH .:$VERSANT_ROOT/jre/lib/rt.jar:$VERSANT_ROOT/lib/
jvi.jar:$VERSANT_ROOT/lib/var.jar:$CLASSPATH
```

For Windows

```
set
CLASSPATH=.;%VERSANT_ROOT%\jre\lib\rt.jar;%VERSANT_ROOT%\lib\jvi.jar;
%VERSANT_ROOT%\lib\var.jar;%CLASSPATH%
```

EVENT DAEMON - VEDDRIVER

veddriver is not starting

- veddriver uses port number 5030 for ChannelServicePort. Update veddriver configuration file if port is already used.
- If you are not providing configuration file as a command line parameter then check whether you have set `VED_CONFIGFILE` environment variable

veddriver is not creating request objects

- Check whether request engine (`varrequesteng.dll`) has been specified in the configuration file under `EngineLibs` subsection.
- Check whether following parameters are set in the database backend profile:

```
event_registration_mode     transient
event_msg-mode              persistent
```

REPLICATION

Check the logs to get more information about problems and to diagnose them. Note that each replication module logs into a separate log file. The log files will be under the directory specified in the configuration file using the parameter LogDirectory. If it has not been specified, the logs will be under the directory in which you started the replication processes using `varsiteadmin` utility.

Not able to start Replication

- If `varsiteadmin` start command is hanging then check whether `rmiregistry` process is running. If `rmiregistry` is not running, use `varstartrmi` to start the `rmiregistry` process.

  `rmiregistry` is the java remote object registry. A remote object registry is a bootstrap naming service that is used by RMI servers on the same host to bind remote objects to names. Clients on local and remote hosts can then look up remote objects and make remote method invocations. The `rmiregistry` process needs to be started in order to use RMI mechanism provided by JVM.

If replication is starting but it is stopping immediately:

- If a process is not able to get Versant session then check the transaction parameter in profile.be file and set the parameter to atleast 20.
- If you are using Advanced mode, check the `MessageReceiver.log` to make sure that it was able to bind to the port specified during site creation.
- Check whether you are specifying the configuration file while starting the replication if you want to use `Advanced` mode.
- Check addresses of the defined sites. In Default mode address of a site is the database name whereas in Advanced mode it is `hostname:portnumber` pair.

Replication is not making any progress

Use statistics and status commands of `varsiteadmin` utility to find out whether replication is making any progress. If replication is not making any progress:

- Check whether request objects are created in the database if event based channels are used.
- Check whether replicator process has logged any error in `VDBReplicator.log` file in Default replication mode.
- Check if the databases used for replication have the same DBID's. In this case you are likely to get an exception `EVJ_NO_SUCH_ATTR`. If you get this error, recreate and reinitialize one of the replicating databases.
- In Advanced mode

  At Sender site:

  - Check whether request processor is processing the requests. If not, check whether replicator process has logged any error in `TxRequestProcessor.log`.
  - If request processor is making progress, check whether sender has logged any error/warning messages in `TxMessageSender.log`.

  At Receiver site:

  - Check whether receiver is able to receive messages. If not, check whether replicator process has logged any error in `MessageReceiver.log`.
  - If the receiver is receiving messages, check whether applier has logged any error messages in `TxMessageApplier.log`.

Not able to stop the replication

- Use status command of `varsiteadmin` utility to find out which replication processes are running.
- Check whether you are specifying the configuration file while stopping the replication if you are using Advanced mode.
- Check if `rmiregistry` is running on desired port while stopping the replication. Otherwise replication will be thrown.

If replication is not stopping after many attempts then locate and kill all the jre processes using Windows Task Manager manually.

LICENSING ERROR MESSAGES

If you encounter any of the following error messages, rerun `vlicvrfy` utility to get further details about the cause of the error. If required contact Versant for new license key for VAR.

```
7180  LIC_ERR_ARGS                Error in arguments passed
7181  LIC_ERR_NODE                Component not licensed
7182  LIC_ERR_DATE                License expired
7183  LIC_ERR_ENVVARIABLE         Environment variable not set
7184  LIC_ERR_FILENOTFOUND        License file not found
7185  LIC_ERR_FILESYNTAX          Error in license file
7186  LIC_ERR_TEXTIGNORE          Ignoring text in license file
```

For more information on vlicvrfy utility, please refer to Chapter “Database Utilities” in the *Versant Database Administration Manual.*

Index