PRIMEBASE
REPLICATION SERVER

July 2008
PrimeBase Systems GmbH
Max-Brauer-Alle 50 - D – 22765 Hamburg - Germany
www.primebase.com - e-mail: info@primebase.com

PRIMEBASE REPLICATION SERVER

***NOTE** for K4 Users: This documenation will describe the Replication with any database. Please read the pbrs_for_k4.pdf for a special K4 Replication Server instruction. We recommend to read this documentation, too to learn the basics.*

1. Replication Server Basics

The PrimeBase Replication Server (PBRS) is a platform for the replication and distribution of data stored by PrimeBase SQL Database Servers.

Using the replication server you can synchronize several copies of a database, and keep them up-to-date in real-time. Changes can be transmitted bi-directionally if required.

If the connection between databases is interrupted, then the replication server will buffer the changes and apply them as soon as the duplicate database is online again.

The replication server is fully programmable, allowing you to modify the standard replication procedures for your own purposes. For example, databases can be replicated in part, instead of completely as is the default.

The replication server can also be used for other purposes such as logging changes, or notification. For example it is possible to send an e-mail when certain data is modified in the database.

Using the PrimeBase Open Server in combination with the replication server it is possible to replicate data from a PrimeBase database to 3rd Party database systems such as Oracle or some other ODBC data source.

Important new features

The PBRS will store it's replication state not longer only in the Master database on the destination PBDS. Each database will store it's own state information itself to increase the amount of parallel running replications and to increase the speed of the replication progress. This feature is avaiable since PBRS version 4.2.70. See chapter 2 step 11 for details.

The PBDS has a new backup option for databases which contains much blob data. This will speed up the inital backup. This feature is avaiable since PBDS version 4.2.65. See chapter 2 step 9 for details.

Please read this documentation for further explanations of the features.

How Does Replication Work?

Each replication server is associated with a particular database server. The replication server is responsible for the outward transmission of data stored by the server. It does this by monitoring the transaction logs written by the server. As a result, the performance of the source database server is not affected at all by the replication activity.

The replication server reads the logs as they are written, and as soon as a transaction is committed it is scheduled for replication. Replication is done by one or more replication processes run by the replication server. Each replication process has its own configuration parameters and runs independently of the other replication activity. Configuration parameters include the replication destination and the replication process class, which determines the procedures run by the process.

The replication process receives the changes from the replication server in the form of 3 basic commands:

- beginTransaction
- performModification

• commitTransaction

Each modification concerns one row of a particular table and includes so-called “before” and “after images” of the data. This tells the replication process exactly what has changed.

What the replication process then does with this information depends on the replication procedures used (which is, as mentioned above, determined by the class of the replication process). The standard duplication class translates the commands into standard SQL statements BEGIN, COMMIT, INSERT, UPDATE and DELETE. These statements are then sent to the destination database server, using the standard SQL client/server interface.

By sub-classing the basic or any other standard replication process class you can fully control the behavior of your replication processes. To do this you will use the PrimeBase native language, PBT, a Java-like programming language which includes standard SQL statements.

The full-power of PBT is available to you when programming a replication process class. Among the features of PBT are TCP/IP Sockets, HTTP and FTP access, e-Mail, Unicode, XML, Blobs and SQL cursors.

Installing a Replication Server

You need three things for the installation of the PrimeBase Replication Server:

- A valid PrimeBase 4.2 or higher Database Server installation.
- The PBRS binary, a valid activation key (‘pbrs.key’ file) and the environment file (‘pbrs.env’).
- The source code of the standard replication process classes.

The PBRS binary must be started with the current working directory set to the directory containing the database server installation. This is the directory containing the ‘databases’ (db) and ‘setup’ folders of the server.

Alternatively the binary can be started with the –r option specifying the path to the server installation directory.

The source code of the standard replication process classes are in a folder called ‘replication’. This folder is placed in the database server’s ‘setup’ folder. Also place the replication server’s ‘pbrs.env’ file in the database server’s setup folder.

Once started, the replication server can be monitored using the PrimeBase console (PBCON). To shutdown the server enter ‘#halt’ on the console. Other console commands are displayed by entering ‘#help’.

To activate the replication server, place the ‘pbrs.key’ file in the installation directory and restart the replication server.

Setting System Parameters and Logging

The system parameters of the replication server are stored in the ‘pbrs.env’ file in the ‘setup’ folder. This file can be edited with the PrimeBase Environment Editor (PBEE).

To set a system parameter, start PBEE in the installation directory of the server. A list of environment files will be printed. You can open one of the files by entering the number of the file.

After this, enter ‘help’ for a description of the commands. A list of system parameters can be obtained using the ‘list’ command. The ‘set’ command is used to alter system parameters.

Most of the system parameters listed concern the PrimeBase Virtual Machine (PBVM) that is built into the replication server. The PBVM is used by the replication server to run the replication processes.

The only parameters directly used by the replication server are the registration information (Serial Number, Registration Name, Activation Key and Expiry Date), and the protocol parameters (Protocol Log File, Protocol Log Level and Protocol

Display Level).

The registration information is set when the replication server reads a ‘pbrs.key’ file (see “Installing a Replication Server” above).

The ‘Protocol Log File’ specifies the name of the protocol log file. By default the file is ‘pbrs.log’, and is created in the ‘setup’ folder.

The ‘Protocol Log Level’ parameter determines what level of protocol statements are written to the log, and the ‘Protocol Display Level’ parameter determines which statements are printed to the console. Levels that can be set range from 0 to 3 where: 0=Off, 1=Errors, 2=Warnings, 3=Trace.

If you set ‘Protocol Log Level’ to 3 (trace) then the current date is added to the log file name to prevent the file from becoming too large.

Replication Process Configuration

The replication processes to be run by the server are described in the XML configuration file `'processes.xml'` which is located in the ’setup/replication’ folder. The format of the file is given by the following example:

```
<processlist>
<process>
      <enabled options="yes/no">yes</enabled>
      <name>Replicate Database</name>
      <database>reptest</database>
      <class>DuplicateProcess</class>
      <connection>reptest_copy</connection>
      <frequency unit="s">10</frequency>
      ...
</process>.
...
<processlist>
```

Any number of processes may be specified with the process list. Each process has the following attributes:

Enabled

Use this switch to enable or disable a single replication process. If this paramter is missing, the replication server will set the process to enabled. **Attention**: Set „enabled“ to „no“ only before you start the initial backup. Doing this while replication is running all transactions for this process will be ignored and you have to re-setup the replication.

Name

The name of the replication process. This name is used for logging and maintaining the process state and **must be unique**. Maximum length for a name is 32 characters.

Database

This is the name of the source database that will be replicated by the process.

***Note**: If you use the same database name on the source and on the destination machine, it will be easier for you to handle a failover scenario. We will use a scenario like this in this documentation, so pay attention on which server (source or destination PBDS) we are, when we use the database.*

Class

The class of the replication process, which determines the procedures to be used

during replication.

Connection

This is the alias of the connection definition which determines the destiniation of replication. Connection definitions are stored in the ‘connect.def’ file, and contain all the information required to connect to the destination server. Also included is **login details** and the **name of the destination database**.

Connection definitions can be specified using the Automation Client PBAC. Note that a connection definition for a replication process must include the **user name and password of the system administrator**.

Frequency

The frequency determines the pause before the process is restarted, if the process aborts due to an error. For example, if the process cannot connect to the destination server. ‘unit’ is either ‘s’ for seconds, or ‘m’ for minutes.

Duplicatekey

Values are allow or disallow. If set to allow, the PBRS will do an update instead of an insert. In case of disallow the PBRS will throw an error and you have to solve the problem manuel before the replication could go on.

Duplicatekeytablelist

If you only want to allow the Duplicatekey setting on certain tables, you could define a list of the tables here.

Initialphase

This determines how the replication process is initiated. The default is copy-backup. Rollforward will start with the current log, without copying a backup first. You have to make sure, that source and destination database are identical. All active transactions have to be written to the transaction log file. We dont recommend to use the rollforward option.

Keynotfound

Allow "key not found" error: When this error occurs, an insert is done instead of an update. Values are: allow / disallow

Missingblobs

This option tells the replication server to ignore missing blob data or corrupt blob data in the source database. The replicataion process will continue, but you will find a worning in the log file.

Values are: allow / disallow. The default is disallow.

Trace

Turn debug trace on or off with this option. Values are: on / off

Update

This option tells the replication process to only update those columns that have been changed. This options is usefull to avoid conflicts during bi-directional replication. Values are: all / modified

Maintaining the Replication State

The replication state is a set of parameters issued by the replication server but stored by the replication process. The replication state contains all the information the replication server needs to determine which modification should be performed by the replication process next. Each replication process has its own replication state.

When the beginTransaction command is sent to the replication process it includes

the replication state. The state must be stored and committed along with the changes that are preformed by the transaction.

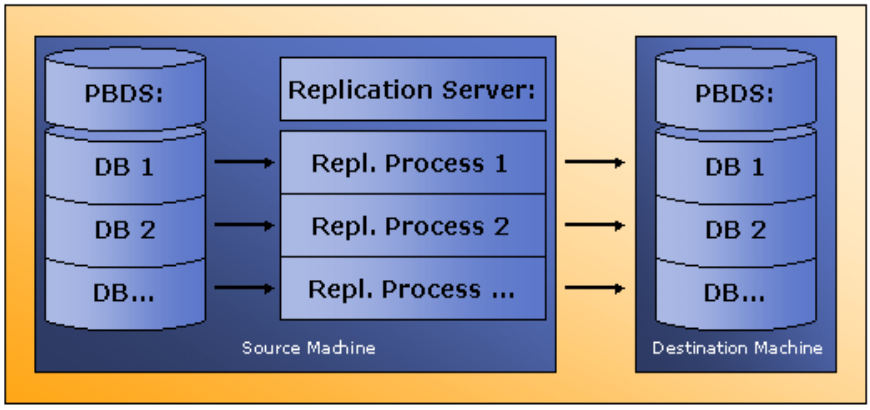
When a replication process starts it sends its current replication state to the replication server. The replication server then knows what modification to send to the process next. After this point a modified replication state is sent with each beginTransaction command, which is, in turn, stored by the replication process.

In this way, replication can continue after any kind of interruption. For example, replication can continue even if the destination server is not available for a while, once the server comes back on line. Stopping and starting the replication server also has no affect on the replication process.

When a replication process starts for the first time, it has no state. In this case, the replication process sends a “NULL” state to the replication server. This is a state in which all the parameters are set to zero. The replication server then knows that replication process has started for the first time and initiates replication.

2. Starting the Replication System

In this section we describe how to initiate the standard, single-directional, duplication of a database.[1]



single-directional replication

We assume the source and destination databases are on different hosts, and will use the following setup in our examples:

Source Machine:	
Host Name:	source.eg.com
Server Installation Location:	/Applications/PrimeBase/Server1
Server Name:	PrimeServer
Database Name:	reptest
System Administrator Name:	Administrator
Password:	sa
Destination Machine:	
Host Name:	dest.eg.com
Server Installation Location:	/Applications/PrimeBase/Server1
Server Name:	PrimeServer
Database Name:	reptest
System Administrator Name:	Administrator
Password:	sa

1. The replication server is also capable of bi-directonal replication; in this case each database server acts both as source and destination for replication.

1. Upgrade/Install the source and destination Database Servers

To use the standard replication class, both source and destination database servers must be at least version 4.2.

Upgrading/Installing PrimeBase under UNIX or Mac OS X

Download and install the appropriate PrimeBase Database Server package.

Run the installation script. To upgrade, select and existing installation directory. To install a new server, enter a new installation directory or select the default.

Upgrading/Installing PrimeBase under Windows

Download and unpack the appropriate PrimeBase Database Server archive.

If you wish to upgrade an existing installation, copy the executable files (.EXE and .DLL) and replace the executables in the current server installation directory.

2. Install the Replication Server

Install the replication server in the installation location of the source server as described in the section “Installing a Replication Server” above.

3. Create a Connection Definition for the Destination Database

Create a text file called ‘connect.def’ in the ‘setup’ directory of the source server’s installation directory.

Start the automation client (PBAC) in the source server’s installation directory. Enter ‘a’ to add a new connection, and then enter the following information:

Connection alias:	reptest_dest
Runtime server connection:	n
Server brand (Database Server):	1
Server name:	PrimeServer
Protocol (TCP/IP):	1
Server address:	dest.eg.com
Users name:	Administrator
Password:	sa
Default database:	reptest
User name for the database:	(hit Enter)

NOTE: *It’s ok that you get an error, if you try to use this connection until you created the Default database („reptest“ in the example) on the target machine.*

Note that the connection definition is created in the ‘connect.def’ file in the location indicated by the automation client. Since you created a local ‘connect.def’ file before starting the automation client this file will be used.

If there is no ‘connect.def’ file in the local ‘setup’ folder, then the global ‘connect.def’ file will be used. However, the global ‘connect.def’ should not be used by the replication server, since an accidental change in the destination connection definition will cause the replication process to fail.

4. Define and Configure the Replication Process

Define the replication process by editing the text file ‘processes.xml’ which is located in the ‘setup/replication’ directory of the source server.

After editing ‘processes.xml’ it should look like this (besides comments):

```
<processlist>
<process>
      <enabled options="yes/no">no</enabled><!-- set it to no at
this step -->
      <name>Replicate Test Database</name>
      <database>reptest</database> <!-- its the source database -->
      <class>DuplicateProcess</class>
      <connection>reptest_dest</connection>
      <frequency unit="s">30</frequency>
      ...
</process>
<processlist>
```

Comments may be placed in an XML file using the delimiters ‘<!--‘ and ‘-->’. For example:

```
<!-- The process list contains several replication processes -->
```

For more information on the configuration parameters please refer to the section “Replication Process Configuration” above.

5. Configure the Source Server for Replication

Two settings need to be made to the source database server in order to permit replication of a database belonging to the server.

To set a system parameter you need to connect to the server and open the master database. This can be done as follows:

Start the PrimeBase Console (PBCON) on the source machine and login as system administrator (User: ‘Administrator’, password: ‘sa’ in our example).

Now enter the following:

```
open database master;
#go
```

System parameters are altered using the SET VARIABLE statement (see below for details). You can list the current settings of the system parameters by entering:

```
#status
```

After setting all parameters, enter the following to logout:

```
#cls
```

Enter CTRL-R to exit the console, leaving the server running in the background.

Enable Replication on the Database Server

Set the system parameter EnableReplication to TRUE **on the source and on the destination database**, see step 11 for more information. Do this as follows:

```
set variable EnableReplication = $true;
#go
```

This will set the system parameter as required.

This setting tells the database server that a replication server is active and causes the server to write additional information to the transaction log files that is required by the replication server.

When replication is enabled, you will also be prevented from using the OPEN TABLE FOR IMPORT statement. The reason is that this statement does not write the inserted data to the transaction log.

Replicate Transaction Logs

By default the PrimeBase Database Server deletes transaction logs when they are moved offline, since an offline log is no longer needed for system recovery. However, it may be that logs are required for restoring a database in the future, and for this reason it is possible to tell the server to archive the offline logs. This is done by setting the system parameter OfflineFunction to “Archive”.

When a replication server is active, it takes over the task of deleting the transaction logs that are no longer online. This means that if the OfflineFunction is currently set to “Delete” on the source server, then your need to change this to “Replicate”.

Enter the following on the server console:

```
set variable OfflineFunction = "Replicate";
#go
```

This setting tells the server to pass responsibility of deleting the offline log to the replication server. The replication server will now delete the offline log when all replication processes have finished reading the log.

Note that if the OfflineFunction is currently set to “Archive” on the source server, then it need not be set to “Replicate”. The replication server will read the archive logs if necessary. Of course, archived logs are never deleted by the replication server.

***NOTE**: Change EnableReplication and OfflineFunction back to the original values, if you dont longer want to use Replication. Otherwise no blob and no transaction log will be automatically deleted by the PBDS.*

6. Add Primary Keys to Source Database

In order to replicate a database, all tables must have a primary key. An index must also be created on the primary key columns. A primary key is a unique row identifier for the table. It may be a single column or a combination of columns.

The replication process will generate an error if it encounters a table without a primary key, so use the browser-based Admin Server to create primary keys for all your tables before continuing.

For our example we will create a new database on the source server and insert

some data. Connect to the source server using PBAC and run the following script[1]

```
CREATE DATABASE reptest;
CREATE TABLE Common.reptab

(
      id          INTEGER          NOT NULL,
      string      VARCHAR(20)      NULL,
      number      DECIMAL(10,2)    NULL,
      when        TIMESTAMP        NULL
);

CREATE PRIMARY KEY pk_reptab ON reptab.id;
CREATE INDEX i_pk_reptab ON reptab(id);
CREATE COUNTER integer cnt_reptab;
CREATE DEFAULT def_pk_reptab ON reptab.id AS SERIAL cnt_reptab;
CREATE DEFAULT def_reptab_when ON reptab.when AS NOW;

INSERT reptab(string, number) VALUES ("PNEF xys", 123.45);
INSERT reptab(string, number) VALUES ("ARC22", 123456.45);
INSERT reptab(string, number) VALUES (NULL, 86745);
INSERT reptab(string, number) VALUES (NULL, NULL);
INSERT reptab(string, number) VALUES ("ABCD 1234 nx", 9.2);
INSERT reptab(string, number) VALUES ("AB", 11.11);
INSERT reptab(string, number) VALUES ("ABC ABC", 123456.12);
INSERT reptab(string, number) VALUES ("zzz TIME 78", 0);
INSERT reptab(string, number) VALUES ("A ABC DEF", NULL);

#go
```

After this, you can enter the following to see what data has been inserted:

```
SELECT * FROM reptab; printall;
#go
```

Note that ‘reptab’ table in our example uses an automatically generated primary key called a counter. The ‘when’ column is also automatically set to the current time by default.

7. Create the Destination Database

You can create a database with an identical schema on the destination server using the browser-based Admin Server or using the „schema.dal“ script in the „setup/scripts“ folder. We recommend the browser interface only for this tutorial. If you want to replicate a larger database you have to use the „schema.dal“ script, which will be discribed further on. This script will ensure that all database definitions are identical on source and destination.

Using the browser interface

The destination database must have the identical schema to the source database. In other words, the structure of the source and destination databases must be identical.

Use a browser to connect to the Admin Server on the source database server. Once you have logged in click on the “Database Schema” tab and then the help button in the bottom left-hand corner. Follow the instructions **”To Duplicate a Database”**. Set the destination DBMS using the “reptest_dest” connection alias.

NOTE: *Clear the source DBMS before you run the script (do this by clicking on the name of the source DBMS and then click on the button "Clear Source..." in the dialog at the bottom of your browser window). This will ensure that the data already in*

1. You can run all scripts in this documentation by copying the code from this PDF-file and pasting it into the console

the source database is not transferred to the destination database by the Admin Server. This is important because a pre-requisite to replication is ***that the destination database must be empty****.*

For our test example, we already have the SQL statements that create the database, so you can simply connect to the destination server and run the script above without the INSERT statements, instead of using the Admin Server to duplicate the database schema.

If you insert data into the destination by accident, then you must delete the data before starting replication. In our example this can be done as follows:

```
DELETE reptab;
#go
```

Note that it is possible to change a database schema once replication has been started. However the replication process will stop until schema changes have been applied to both copies of the database.

Click on „run“ to start the transfer.

Using the „schema.dal“ script

Start „pbcon“, connect to the source PBDS on which your database is running and login as administrator. Type in:

```
EXECUTE FILE "schema.dal";
OPEN DATABASE reptest;
makescript("reptest.sql");
#go
```

You will see the output of the table definitions in the console. The same output is written to the file „reptest.sql“ in the directory where the source PBDS is running. You could define your own file name instead of „reptest.sql“.

Copy this file to the same location on the destination PBDS, start „pbcon“ and login as administrator. Type in:

```
DROP DATABASE reptest; // This step is only necessary, if the data-
base alread exist, because of an former replication
CREATE DATABASE reptest;
OPEN DATABASE reptest;
EXECUTE FILE "reptest.sql";
#go
```

The schema of the database is no identical on source and destination.

8. Optional: Set the Destination Database to READONLY

If replication is not bi-directional (as in this example) then it is recommended that you set destination database to READONLY. This is just a precautionary measure to ensure that an update is not made to the destination database by mistake.

To set the database to READONLY, connect to the destination server, or login at the console and enter the following command:

```
alter database ("reptest", readonly=$true);
#go
```

The database is now set to READONLY. An SQL INSERT, UPDATE or DELETE statement will generate an error. The replication process, however, is exempt from this restriction.

Note that before you can make any modifications to the destination database

directly you will first have to enter the command:

```
alter database ("reptest", readonly=$false);
#go
```

9. Backup the Source Database to Start Replication

Start both the source and the destination database servers.

Start the PrimeBase Console (PBCON) on the source machine and select the replication server. Check that the server has started correctly and that no errors have appeared on the console. For example, that no error occurred when connecting to the destination server. Allowed is an error which tells you, that no backup is avaiable.

To initiate replication perform a backup of the database to be replicated. You can do this as follows:

Start the PrimeBase Console (PBCON) on the source machine, connect to the PBDS and login as system administrator (User: 'Administrator', password: 'sa' in our example).

Now enter the following:

```
BACKUP DATABASE reptest;
#go
```

Change the „enabled" option in the processes.xml to „yes" (see step 4 for processes.xml description) and then start the replication server in the source server's installation directory. Replication will begin as soon as the backup is completed. The replication process begins by copying the source database backup image to the destination database. After transfer of the backup is complete, the replication process starts duplicating any changes you have made to the database after the backup.

There is a new backup feature in the PBDS since version 4.2.65 for databases which contains very much blobs:

```
BACKUP DATABASE <dbname> WITHOUT BLOBS;
#go
```

A „backup without blobs" will only copy the tables which is much faster, than copy every blob, too. The PBRS knows which backup belongs to which original database and will merge the table data from the backup with the blobs from the original database. Because of the handling of the blobs in PrimeBase and the settings EnableReplication and OfflineFunction described in step 5 no blob will be deleted until the PBRS finished transfering the backup and all corresponding blobs.

Set the EnableReplication flag on the destination database back to $false (see step 5 and 11 for details).

10. Observing Replication in Progress

If trace is on, then the SQL statements performed during replication can be observed on the replication server's console.

Connect to the source server and enter the following statements:

```
INSERT reptab(string, number) VALUES ("aaabbbccc", 121212.12);
#go
```

```
UPDATE reptab SET string = "AB AB AB AB" WHERE id = 3;
#go
```

```
UPDATE reptab SET string = "x x x x" WHERE string = "AB AB AB AB";
#go
```

```
DELETE reptab WHERE id = 5;
#go
```

As each statement is executed you will see how the change is propagated to the destination database.

Note that it is recommended to turn trace off during production as this can affect performance. By default trace is on for the standard duplication process. Trace is disabled by setting the instance variable 'traceOn' to $false, in the file 'duplicateprocess.pbt' in the 'setup/replication' directory. After modifying this file you need to restart the replication server for the change to take affect.

11. The Replication.Processes Table

The Replication.Processes table is created by the standard duplicate replication process. The table is created on the destination server in the Master database and in the database which you want to replicate (also on the destination server). The Replication.Processes table in the Master database is used to have an overview over all running replications and the Replication.Processes table in the database which you want to replicate is used the by the replication process to store its current replication state. A Replication.Process table per database is new a feature in PBRS 4.2.70, to increment the number of databases, which can be replicated in parallel. The PBRS will detect older versions which are only using the Replication.Processes table in the Master database and will automatically create the table in the database which will be replicated. The PBRS is copying the state information from the Master database to the new table.

You can view the Replication.Processes table created by our example replication by connecting to the destination server in the example, and entering the following:

```
open database Master;
SELECT * from replication.processes; printall;
#go
```

The table contains one entry for each replication process that is sending modifications to the server. The actual state is not longer stored in the Master database. Execute the same SELECT statement on the database which you are replicating.

IMPORTANT: EnableReplication (see step 5 for details) must be set on the destination database server to allow the creation of the table „Processes". After you started the replication process you have to set EnableReplication back to $false on the destination database.

This is the structure of the Replication.Processes table:

```
CREATE DOMAIN Replication.ProcessID INTEGER;
CREATE TABLE Replication.Processes

(
ID                Replication.ProcessID   NOT NULL,
Name              SysName                 NOT NULL,
ServerID          SysName                 NOT NULL,
DatabaseName      SysName                 NOT NULL,
CreationTime      TIMESTAMP               NOT NULL,
ModifyTime        TIMESTAMP               NOT NULL,
DatabaseID        INTEGER                 NOT NULL,
BackupNo          INTEGER                 NOT NULL,
Phase             INTEGER                 NOT NULL,
TableID           INTEGER                 NOT NULL,
TableOffset       INTEGER                 NOT NULL,
LogRestartID1     INTEGER                 NOT NULL,
LogNumber1        INTEGER                 NOT NULL,
LogOffset1        INTEGER                 NOT NULL,
LogRestartID2     INTEGER                 NOT NULL,
LogNumber2        INTEGER                 NOT NULL,
LogOffset2        INTEGER                 NOT NULL,
TransactionNo     INTEGER                 NOT NULL,
Key               (Name, ServerID, DatabaseName)
);

CREATE PRIMARY KEY Replication.ProcessesPk
ON Replication.Processes.ID;
CREATE INDEX Replication.ProcessesInd1
ON Replication.Processes(ID);
CREATE CANDIDATE KEY Replication.ProcessesCk
ON Replication.Processes.Key;
CREATE INDEX Replication.ProcessesInd2
ON Replication.Processes(Key);
CREATE COUNTER INTEGER Replication.ProcessCnt = 0;
CREATE DEFAULT Replication.ProcessesPkDef
ON Replication.Processes.ID
AS SERIAL Replication.ProcessCnt;
CREATE DEFAULT Replication.ProcessesCrDef
ON Replication.Processes.CreationTime
AS NOW;
CREATE DEFAULT Replication.ProcessesMoDef
ON Replication.Processes.ModifyTime
AS NOW;
```

The columns `Name`, `ServerID` and `DatabaseName` are used to identify the replication process.

`Name` is the name of the replication process as specified in the ‘processes.xml’ file. `ServerID` is a unique identifier of the server, and contains the serial number of the source server. `DatabaseName` is the name of the source database.

The columns `CreationTime` and `ModifyTime` indicate when the replication server started, and when the last modification was performed by the process.

`Phase` will show you the actual state information of the replication process:

0: Initial Phase
1: Copy Backup Phase
2: Roll Forward Phase
3: Initial Roll Forward Phase

***Note**: Make sure not to start a new backup to the same location while the replication process is in the Copy Backup Phase.*

The remaining columns store the various parameters of the replication state.

3. Restarting a Replication Process

***NOTE**: Do a new backup BEFORE deleting the replication process entry in the Master!Replication.Processes table (on the destination server).*

Restarting a replication process means that the process is reset to its initial “NULL” state. When this is done to the standard duplication processes, it begins by first copying the latest backup of the source database to the destination database. Following this, replication continues with the modification that occurred after the backup.

Normally, after a replication process has been started there is no need to restart it. But there are a number of cases in which you will have to restart replication:

- The source database was unmounted and then mounted again or a backup was mounted instead.
- Data was imported into the source database using OPEN TABLE FOR IMPORT.
- A log file required by the replication server has gone missing.
- A RESTORE DATABASE command has been performed on the source database.

In any of these cases you can perform the following steps to restart replication.

***NOTE:** The only thing that will cause replication to restart is if the replication state is deleted, or set to NULL. Any other error or problem will only cause the replication process to pause, wait for the period of time specified by the frequency parameter and the retry.*

1. Shut down the Replication Server

Start the PrimeBase console on the source server, and select the replication server. Now enter the following in the console:

```
#halt
```

The replication server will shut down.

2. Backup the Source Database

Make a backup of the source database. When replication begins again, it will start from this backup. How to make a backup is described in the section “Backup the Source Database to Start Replication” above.

***NOTE:** that a backup may be performed as usual at any time during replication. It does not affect the behavior of replication, and will not cause replication to be restarted.*

3. Delete all the Data in the Destination Database

All the data in the destination database must be deleted before replication is restarted. You can do this by dropping, and then re-creating the database. Alternatively you can delete the contents of all the tables, but before this can be done you need remove the READONLY lock on the database:

```
alter database ("reptest", readonly=$false);
#go
```

Now execute the following statements:

```
OPEN DATABASE reptest;
DESCRIBE TABLES INTO c;
FOR EACH c {
      DELETE c->owner.c->name;
}
#go
```

After this you can set the destination database to READONLY again:

```
alter database ("reptest", readonly=$true);
#go
```

4. Delete the Replication State

To restart the replication process you need to delete the entry for this process. As mentioned above, the replication information is stored in the table Replication.Processes, in the Master database and in the database which you want to replicate.

This can be done as follows:

```
OPEN DATABASE Master;
DELETE Replication.Processes WHERE name = "Replicate Test Database"
and DatabaseName = "reptest";
CLOSE DATABASE Master;
OPEN DATABASE <YourDatabaseName>;
DELETE Replication.Processes WHERE name = "Replicate Test Database"
and DatabaseName = "reptest";
#go
```

Note that some care must be taken when deleting a replication entry when there is more than one replication process that have the same database server as destination. The Replication.Processes table in the Master database contains one row for each replication process (the Replication.Processes table in the database which you want to replicate contains only one entry). It is best to first select the data from the table, and then select the row to be deleted based on the columns, `Name`, `ServerID` and `DatabaseName`, which uniquely identify a replication process.

Restart the Replication Server

You can now restart the replication server running in the source database server's installation directory. The replication process will immediately begin by transferring the data in the most recent backup to the destination database.

4. Programming the Replication Server

As mentioned before, the PrimeBase Replication Server is fully programmable. A standard duplication implementation is provided, but this implementation can be extended or replaced completely.

Replication does not, for example, have to have a different database as destination. A replication process could observe changes and log the modifications in the same database. Also mentioned before are such functions as e-mail notification or an XML export. Any process you write need not apply to the entire database, but only to tables and columns of your choice.

Programming the replication server is simple. The code of the standard duplication process provides a good example. In this section we will describe the standard class provided with the replication server and how to extend these classes to perform your own replication tasks.

Standard Replication Server Classes

ReplicationProcess in 'replicationprocess.pbt'

This is the basic abstract replication process class. A custom replication process must extend this class. The methods to be implemented are:

- ReplicationState getState(ReplicationID id);
- void beginState(ReplicationID id, ReplicationState state);
- void commitState(ReplicationID id, ReplicationState state);
- void modifyImage(ReplicationID id, ReplicationImage image);
- void handleError(ReplicationID id, ReplicationError err);

Note that when you implement a replication process you do not deal with the commands beginTransaction, commitTransaction and performModification which are sent by the replication server directly. These commands are received by the ReplicationProcess class which then calls the methods beginState, commitState and modifyImage listed above.

The replication ID, represented by the ReplicationID is the unique identifier of your replication process. It contains the fields: `serverID`, `processName` and `databaseName`.

ReplicationState getState(ReplicationID id);

The method getState() must return the current replication state of the process. This method will be called when the replication process is booted.

void beginState(ReplicationID id, ReplicationState state);

The method beginState() indicates that the modifications that follow all belong to one transaction.

void modifyImage(ReplicationID id, ReplicationImage image);

The modifyImage() method performs the actual replication task. To do this it receives the replication "image" which contains all the information about a change to a particular row in the database. The ReplicationImage class is described below.

void commitState(ReplicationID id, ReplicationState state);

The methods commitState() indicates the end of the modification transaction. At this point the replication process should save the replication state and commit the transaction. The state saved here must be returned by next call to getState().

void handleError(ReplicationID id, ReplicationError err);

The handleError() method is called when the replication server receives an exception thrown by the replication process. Extending or overwriting the implementation of this method is optional. By default the method just logs the error and the stack trace at the time of the error. After calling handleError() the replication process is terminated by the replication server.

ReplicationImage in 'replicationimage.pbt'

As mentioned above, the replication image contains all the information about the modification of one particular row in the database. The class includes the following fields:

```
public varchar    table;
public varchar    key;
```

The table field contains the fully qualified name of the table modified.

The key field contains a comma delimited list of the names of the primary key columns of the table. This primary key is used in order to uniquely identify the data-

base row during replication.

In addition to these fields the replication also contains a “before” and “after component”, which are stored as database cursors. The before and after components of the image each contain, at most one row. However, if the row was just inserted, then before component will be empty, and if the row was deleted, then after will be emtpy.

ReplicationImage provides a number of methods that help you to manipulate and examine the image data. Methods are also provided which generate SQL statements from the image (insertSQL, updateSQL, deleteSQL and selectSQL), and execute these statements (performInsert, performUpdate and performDelete).

boolean hasBefore()

This methods returns $true if the before component of the image is not empty.

boolean hasAfter()

This methods returns $true if the after component is not empty.

generic getBeforeValue(varchar column)

This methods returns the value of a particular column in the before component of the image.

generic getAfterValue(varchar column)

This methods returns the value of a particular column in the after component.

boolean modified(varchar column)

This methods returns $true if the column has been modified. To do this, the method compares the values of the before and after components. Note that if a component is empty, then the value of all columns in the component is considered to be NULL.

DuplicateProcess in ‘duplicateprocess.pbt’

This class is the standard duplicate process implementation. The class extends ReplicationProcess. The following sections describe the implementation of the abstract methods provided by the ReplicationProcess class.

As mentioned before, the DuplicateProcess class has a field called traceOn. When set to $true, the process logs the SQL statements executed by the destination server.

By default the statements are only logged to the replication server console. If you turn on trace logging on the replication server the statements will be written to the ‘pbrs.log’ file (see “Setting System Parameters and Logging” above).

ReplicationState getState(ReplicationID id);

The implementation of this method selects and returns the current state of the replication process from the Replication.Processes table in the Master database.

If the table does not exist it is created. The replication ID is used to locate the replication processes state in the table. If the state is not in the table, a NULL state is inserted.

void beginState(ReplicationID id, ReplicationState state);

The method performs a BEGIN TRANSACTION statement, and then updates the process replication state in the Replication.Processes table. The update also sets the column ModifyTime to the current time.

void modifyImage(ReplicationID id, ReplicationImage image);

The duplicate process implementation of this method updates the destination data-

base to reflect the change made to the source database.

Duplicate key errors are ignored by default during the phase in which the backup of the source database is copied to the destination database.

void commitState(ReplicationID id, ReplicationState state);

The method executes a COMMIT TRANSACTION statement on the destination database. This commits the changes made during the beginState() and modifyImage() calls.

5. Errors and how to fix them

Each replication of a database is running in an own sessions in the PBRS. If an error in one session occurs, all other parallel running replication sessions of other databases are not concerned. If a session fails, the PBRS will close the session and will automatically create a new one after the defined delay time in the processes.xml file in the parameter „frequency“. Very much errors will be automatically fixed (like TCP-Connection failures) and the replication process in the session will then automatically restart. If errors occur on each restart again and again, you have to fix them manually.

A list of already kwown problems which might occur and a solution to fix them are described below:

ERROR -803 (0): Duplicate unique key on the „System.SysUsers“ table during „copy backup“ phase

This is only a warning and normal output. Users in the System.SysUsers table will be handled different, because 3 Default-Users always exist when you create a new database: System, Public and administrator (or the user who created the database). The PBRS will do an update for these default users keeping the Primary Key which might be different on the Master and Slave database. To avoid this error output, start the pbee (PrimeBase Environment Editor), open the pbrs.env and set ID 343 to 1 or 2.

ERROR -12667 (19): System level I/O on the virtual memory file (VM.SQL) failed (No such device).

Primary error.....: (-12667) System level I/O on the virtual memory file (VM.SQL) failed.
System error.....: (19) No such device.
Operation attempted.: Write
Seek position.....: 65536
Bytes transfered....: 65536
File/path name.....: databases\VM.PBRS

Start the pbee (PrimeBase Environment Editor), open the pbrs.env and set ID 310 to 100000000 (it’s the setting for the virtual memory cache).

ERROR -1: in "System Session" "replicationsystem.pbt"@client line 81: The backup used for replication, 1, is not the last backup

The replication process will start with the „copy backup“-phase on default. If you start a new backup during the actual backup will be copied to the destination database, you will get the above error. If this error accur, you have to restart the replication process again. Please make sure, that you have no automatic or manual backups running while the replication is in „copy backup“-phase. The state information is stored in the Replication.Process-Table on the destination database (see chapter 2 step 11 for details)

ERROR -12677 (0): Log file is currupted.
Primary error.....: (-12677) Log file is currupted.
Operation attempted.: Read
Seek position.....: 7865032
Bytes transfered....: 12
File/path name.....: databases/AA000829.LOG

Note: The values of seek position and „file/path name“ may be different in your case.

It might be possible to solve this error, if the log file isn't really currupted. The problem could be, that only the offset information in the Replication.Process table is wrong. If the same error occur after we finished the following steps, than it seems, that the LOG-File is damaged. In this case you have to restart the replication process. Here are the steps:

1. Stop the PBRS.
2. Connect to the destination PBDS and open the destination database in which the error occured. Type in:

```
SELECT ID, Name, DatabaseName, LogNumber1, LogOffset1, LogNumber2,
LogOffset FROM Replication.Processes; printall;
go
```

Name and DatabaseName will show you the selected replication. Look in LogNumber1 and LogNumber2 for the corresponding currupted LOG-File. Execute following Update-Statement (reset the readonly-setting, if necessary (chapter 2, step 8)). Replace <TheLogColumn> by LogOffset1 or LogOffset2 and <TheAboveID> by the value from the ID-Column from the result;

```
UPDATE Replication.Processes SET <TheLogColumn> = 0 WHERE ID = <The-
AboveID>;
go
```

3. Modify the processes.xml for the replication process which has the error and set the parameter „duplicatekey“ to allow. This is necessary, because we reset the Offset and the PBRS will execute already transferred transactions again. This will be now problem, when you set „duplicatekey“ to allow. You could switch this back to disallow, when the PBRS succesfully passed the position where the error occured.

ERROR -11705: in "User Session"
"sys.pbt"@client line 49: Invalid record header
1 SysError:throw sys.pbt 49
2 DuplicateProcess:throwError duplicateprocess.pbt 255
3 DuplicateProcess:performDelete duplicateprocess.pbt 309
4 DuplicateProcess:modifyImage duplicateprocess.pbt 380
5 ReplicationProcess:performModification replicationprocess.pbt
6 - Process:performModification(... 1
ERROR -11705 (0): "sys.pbt"@client line 49: Invalid record header.

Errors like this always belong to the destination database. The PBRS does not access the source database, he is only reading the RA*.LOG files on the source. Therefore he could only show you errors which belong to the destination database. You should find this error in the pbds.log on the destination PBDS, too. To fix this error, stop the PBRS and connect to the destination PBDS using PBCON. Login and type in (Replace <NameOfDB> with the name of the database and <NameOfTable> with the name of the table where the error occured)

OPEN DATABASE <NameOfDB>;
REORG TABLE <NameOfTable>;
go

After finishing the reorg, you could restart the PBRS again.

ERROR -12679 (2): System level I/O on a log file failed (No such file or direc-

tory).

Primary error.....: (-12679) System level I/O on a log file failed.
System error.....: (2) No such file or directory.
Operation attempted.: Open
File/path name.....: databases\AA000012.LOG

This error is normally a temporary problem. The PBDS will rename the AA*.LOG to RA*.log. The PBRS is reading the RA*.LOG and the AA*.LOG files. If the PBDS renames the file at the moment when the PBRS tries to open it, you will get the above error. The PBRS will close the session and when it will be reopened the error should not occur again.