

CS TimeClock P2P Replication Specification

Document Date: November 2008

Document Status: Version 2.01

Program Status: Implemented in CS TimeClocks version 1.01 and later.

© 2009 by CapeSoft Software (pty) Ltd

Contents

1. Introduction	3
2. Purpose.....	3
3. Method.....	3
4. Notes	4
5. Prerequisites.....	4
6. p2p Replication.....	4
7. TCP/IP Connections	5
8. Incoming Messages.....	6
9. Routing	6
10. Circular Routes	7
11. Time Synchronization	7
12. Notes for Programmers.....	7

1. Introduction

When multiple parent clocks are used on the same site, then it is advantageous for the clocks to communicate with each other. By doing this they can share information, like the current employee payroll and area status and ensure that all the clocks are synchronized to the same time.

Parent clocks can communicate with each other over a Local Area Network (LAN) or over a Wide Area Network (WAN).

This functionality is Peer-To-Peer (p2p) meaning that there is no central clock, or controller, which is in charge. All the clocks communicate with each other. In situations of high-clock density, or where minimal bandwidth use is required, then creating Routes between groups of peers is possible.

This program is almost completely hidden from the user of the clock. Apart from a small number of settings required at installation time there is no interaction between this functionality and the user.

The name of the program that performs this function is ***Replicate***.

This document contains information useful to the person configuring the clocks where there are multiple parent clocks. It also contains information useful to programmers who are integrating PC Software with the CS TimeClocks.

2. Purpose

1. Synchronizes the employee TNA direction and Area between multiple clocks.
2. Ensures that all clocks have the “same time”.

3. Method

Replicate employs a peer-to-peer (p2p) system for passing the updated status around. Thus there should be no single point of failure. If a clock is “offline” for a period of time then it is automatically resynchronized with the others when it reconnects.

In addition to the p2p approach, Replicate also supports the use of Routing Tables to forward messages received from one clock to another clock. Using Routing tables it is possible to greatly reduce bandwidth consumed between one group of clocks and another group of clocks. However routing adds a single point of failure to the system. If the router is offline, then all the clocks dependant on the router are also not updated.

p2p, and Routing are specifically configured by the user, or software managing the system. Replicate makes no attempt to auto-configure.

4. Notes

The field used to uniquely identify each clock in the system is the Serial Number of the parent clock. Thus unique serial numbers are very important to the well-being of the system. If a serial number should change (for example, if a parent clock in the system is replaced) then the **OtherDev** and **Route** tables for all the other parent clocks in the network should be updated accordingly. In addition, if new parent clocks are added to the system, then additional records need to be added to the otherdev table of all the other parent clocks.

5. Prerequisites

1. The clock must have a valid serial number
2. Init field in Thisdev must be set to 1
3. One or more entries must exist in the Otherdev table

The items above are required in order for replication on the clock to be activated. If any of these states change then a reboot of the clock is recommended.

Specifically if the otherdev table is blank then replication on the device is disabled. If you add one or more records to a blank otherdev table, then a reboot of the clock is recommended.

Programmers Note: To avoid a reboot of the clock, the replicate process can be manually started, from a terminal window using;
`/usr/local/bin/replicate --quiet &`

6. p2p Replication

The OtherDev table lists the peers for this clock. Once you have a working clock, adding entries to this table is the only additional thing you need to do in order to make replication work.

For each change (caused by a swipe) in the employee Payroll status, or employee Area status, a message is sent to each peer. The message contains the current status of the employee, as well as the time the status last changed.

The fields in the otherdev table should be set as follows;

Field	Description
Description	The description of the remote clock
Serial	The serial number of the remote clock. If the serial number entered here matches the serial number of this clock, then the entry is invalid and it is simply ignored.
IP	The cip address of the remote clock. This should match the cip field in the thisdev table on the remote clock.
Port	The Port number, as set in the cport field, of the thisdev table, on the remote clock. Note that the replication process uses this port number+1 as the port number for replication.
Resyncrqd	Used Internally. Set to 0 for new records.
Lastcatime	Used Internally. Set to 0 for new records.
Lastcstime	Used Internally. Set to 0 for new records.
PC	Set to 0 if this line should not be altered by PC software. Set to 1 if the record was created, and is maintained by remote PC software. If this is set to 2, then the PC Software should import the settings as set here, and reset the PC field to 1.

Note that the otherdev table of all the clocks should be “complete”. That is to say, if clock 1234 and 5678 exist, and you add 1234 as a peer to 5678, then you should also add 5678 as a peer to 1234.

If a message cannot be delivered then the resyncrqd field is set. If the message is delivered okay, then the lastcatime and lastcstime are updated. Note that these fields are cached in the replicate program, and may not be visible using the TCP/IP interface to the database. To force the cache to be flushed send a .r; command to the clock’s TCP/IP interface.

Tip: Since invalid peers (entries in the otherdev table with the same serial number as this device) are simply ignored, it is possible to load all devices with exactly the same list of otherdev entries.

7. TCP/IP Connections

Connections to peers are made on an on-demand basis. Either clock may make the initial connection, after which status updates can flow in either direction on that connection. Connections are automatically closed after a time-out period has elapsed.

The OtherDev table contains the connection details for all the peer clocks, including the IP address and Port number. This IP and Port number should match the values in the other clock’s ThisDev table, specifically it should match the cip and cport fields on the other clock.

Connections are made on a port number equal to the port number in the OtherDevtable + REPLICATE_PORT_OFFSET. At the time of writing the REPLICATE_PORT_OFFSET equate was set to 1.

If you need to open a firewall between two clocks then, in order for replicate to work, the port number (cip+REPLICATE_PORT_OFFSET) must be opened. The default for this is port 5124. Ideally connections should be possible in either direction as the clocks are peers. If a connection is only possible in 1 direction, then the system will still work, however some employee updates may be delayed.

8. Incoming Messages

Incoming messages are checked against the existing emp table, and if necessary updates to the emp table are made.

By default incoming messages are applied, but not forwarded to any other clock, unless one or more Routing entries exists.

9. Routing

Routing is less ideal than p2p because it creates one or more single-points-of-failure. However to reduce bandwidth over slow, or expensive connections, or to reduce the load on specific clocks, routing can be employed. Care should be taken when setting up routing paths not to create circular routes.

Routing is configured via the Route table. This table is set uniquely for each clock. The table contains 3 fields;

ID
FromSerial
ToSerial

The ID field is a unique numeric identifier in the table. Routes are processed in ascending ID order.

All messages received from the FromSerial clock are forwarded to the ToSerial clock.

A clock can appear in multiple routes, either in the FromSerial or ToSerial field. A clock may appear in the routing table as a FromSerial in some records, and as the ToSerial in other records.

Both the FromSerial clock, and the ToSerial Clock should be peers of this clock. If either the FromSerial, or ToSerial clocks are not peers of this clock, then the route is invalid. If either the FromSerial or ToSerial fields contain the serial number of this clock, then the route is invalid. Invalid routes in the route table do no harm, they are ignored.

10. Circular Routes

A circular route is created when Clock A routes messages to B, B routes messages to C and C routes messages to A. In this scenario a message introduced into the system will continually flow from one clock to the next.

Circular routes are bad and can cause a major disruption to the system. At the time of writing no attempt is made to detect, or suppress, circular routes. A future version of the program may improve circular route handling.

11. Time Synchronization

All clocks need to have the “same time” within reasonable boundaries. Specifically the time difference between any 2 clocks must be less than the time it takes to clock on one clock, and then clock on the other. A recommended time difference of less than 3 seconds is recommended, and this is the degree of accuracy maintained by the replicate program.

Whenever a connection is made between two clocks, the current time is sent from the primary clock to the secondary clock. The secondary clock adjusts its own time if it is necessary to do so.

The method for determining which clock is the Primary, and which is the Secondary is automatic, and dependant on the value in the TimePriority field in the ThisDev table. If two clocks have the same Time Priority then the clock which has the lower serial number of the pair is automatically the Primary clock.

By default all clocks have an equal TimePriority (0) and hence the Serial Number rule applies.

Note: If you wish to change the time of any clock in the system, then it is necessary to change the time on all the clocks. In order to do this you need to change the time on the clock which has the highest TimePriority, or, if multiple clocks have the same TimePriority then the clock with the lowest serial number (of that group of clocks.) Failure to set the time on the right clock will result in the time being overwritten the following time the clock communicates with the Primary clock.

Internally all times are GMT, not Local. Therefore it is possible to use Replicate to maintain employee area and direction, even if the clocks involved are in different time zones. When you set the time of the clock using PcServer though you set it to Local Time not GMT time. The conversion from GMT to Local is done internally.

12. Notes for Programmers

12.1 Incoming commands on the TCP Port

Command	Name	Description
.w;	Who Am I	The remote clock wants to know the serial number of this clock. A .a; command is sent as a reply.
.aX1;	I Am	The remote clock on this connection has serial number X1.
.sX2;	Set Time	Set the time of this device to X2.
.r;	Refresh Cache	The local cache (of the otherdev and route tables) needs to be refreshed.
.uX3;	Update Employee	Update the employee status as per X3.

Packet Name	Description
X1	A numeric number, from 1 to 2 billion. Eg .A1234;
X2	A numeric number, containing the Linux standard time, relative to GMT to set this clock to. The time is converted to local time before being applied to the clock.
X3	A colon separated string containing the following items; Area – A numeric code for the area number the employee is now in. T&A -- The current T&A status of the employee. 0=Out, 1=In. AreaTime – The time (GMT) when the employee entered that area. T&ATime – The time (GMT) when the employee clocked in, or out. Employee – A string containing the employee number. For example; 12:1:8765432:8765432:001

12.2 Internal caching

The OtherDev and Route tables are cached internally by the Replicate program. If either table is changed, then the Replicate program needs to be refreshed. This is done by sending it a `MYSIG_REFRESH_DATABASE` signal. This is done in one of the following ways;

If the PcServer program is being used to update the database then sending a command
`.r;`
 to the PcServer will ensure that the Replicate program is refreshed.

If you are connected to the Replicate TCP port directly then a command of
`.r;`
 will trigger a refresh of these fields.

If none of these approaches are suitable then a clock reboot is advised.

Note that refreshing the cache will first flush the existing cached data. Specifically the fields `resyncrqd`, `lastcatime` and `lastcstime` may be overwritten. If you wish to specifically overwrite these fields then the Replicate program should be stopped before editing these fields.

12.3 Terminating the Replicate Process

The Replicate process is under the control of the Watchdog program. By default, if it terminates in an unexpected way then the Watchdog program will restart it.

To terminate the process without it being restarted by the Watchdog identify the processid (using ps) and then do a simple kill. For example;
kill 894

You may then restart the Replicate process by typing
replicate &

If replicate is started in this way it will re-attach itself to the Watchdog process.

To terminate the Replicate process so that it will be automatically restarted by the Watchdog (within approximately 30 seconds) use the kill -9 command. For example;
kill -9 894

12.4 Command Line Parameters

The standard logging command line parameters apply.

- quiet
- noisy
- verbose
- debug (default)
- errors (default)
- noverbose
- nodebug
- noerrors

There are no command line parameters specific to Replicate.