



Shepherd Replication Services Administration Guide

J&J Computer Consulting Shepherd Server Publications

Overview

Shepherd Replication Services (SRS) provide a method by which directory data can cascade down to other Shepherd servers that run off of a single directory. Using SRS, an administrator can make changes at a single Shepherd system, and those changes can quickly replicate to all of the other Shepherd servers in the network.

Currently, SRS provides a top-down replication service where changes propagate down to other systems, but changes cannot merge back up the chain. Eventually, SRS will adapt to a bi-directional replication scheme.

The purpose of this document is to identify the steps necessary to configure Shepherd Replication Services and the options available for configuring SRS to fit your needs.

SRS Concepts

SRS uses the Hook and Agent features of the Shepherd Engine. With SRS, a single Shepherd server gets designated as the "Master" server. All additions, updates, and deletions are performed on the Master server. Using Hooks, the Master server logs all changes to the directory for replication to "Slaves". An Agent running on the Master wakes up at a specified interval and processes all of the changes logged on the Master server by performing the same operations on the Slaves using the Shepherd.Admin Service on the Slaves.

The Slave systems reject all add, modify, and delete operations on their directories except for those performed by the Master during replication. The Slave systems use Hooks to catch and reject all change operations on the directory.

Slaves also have the option of serving as Master to other Slave systems if necessary. In this situation, a special Slave Hook logs data replicated by the Master and continues to reject all other add, delete, or update requests. The same Agent used on the Master also runs on the slave to replicate data to other Slaves.

All of the files needed to set up replication can be found in the "replicate" subdirectory of the installation directory. A sample LDIF file is provided for a simple configuration in setup\replicate.ldif.

Setting up the Master

To setup the Master server, complete the following steps:

1. Add the attribute **replicationInterval** to the ShepherdServer object that represents the Master server. For the value, enter a number of minutes for the replication Agent to sleep between replications.
2. Configure the SRS Agent. Create entries similar to the following for the Agent:

```
dn: cn=replication, c=US
cn: replication
objectclass: ShepherdAgent
agentModule: replicate\sdsrepl.agt
hostServer: cn=Master, c=US
log: cn=replicationErrorLog, c=US
log: cn=replicationAuditLog, c=US
```

```
dn: cn=replicationErrorLog, c=US
cn: replicationErrorLog
objectclass: Log
logType: error
filename: log\replerror.log
```

```
dn: cn=replicationAuditLog, c=US
cn: replicationAuditLog
objectclass: Log
logType: audit
filename: log\replaudit.log
```

3. Create the Master SRS Hooks as follows:

```
dn: cn=sdsrMasterModifyHook, c=US
cn: sdsrMasterModifyHook
objectclass: ShepherdHook
hostServer: cn=Master, c=US
hookMethod: modifyObject
api: replicate\sdsrepl.dll!sdsMasterReplicationHook
```

```
dn: cn=sdsrMasterDeleteHook, c=US
cn: sdsrMasterDeleteHook
objectclass: ShepherdHook
hostServer: cn=Master, c=US
hookMethod: deleteObject
api: replicate\sdsrepl.dll!sdsMasterReplicationHook
```

```
dn: cn=sdsrMasterAddHook, c=US
cn: sdsrMasterAddHook
objectclass: ShepherdHook
hostServer: cn=Master, c=US
hookMethod: addObject
api: replicate\sdsrepl.dll!sdsMasterReplicationHook
```

These Hooks initiate logging of add, modify, and delete operations.

The Master should now be ready to support replication. However, keep in mind that no replication will take place until Slaves are configured. The Master will log and process replication data without sending it anywhere until Slaves are configured.

Setting Up a Slave

To set up a Slave server, complete the following steps:

1. Add a **slave** attribute to the Slave's Master ShepherdServer object that contains the distinguished name of this slave. Note that the Master could also be a Slave itself under some circumstances.
2. Add the **ipHostNumber** attribute to the Slave's ShepherdServer object. The ipHostNumber should contain the IP address on which the Shepherd.Admin Service is running for the Slave.
3. Add the **ipServicePort** attribute to the Slave's ShepherdServer object. The ipHostPort should contain the TCP/IP port number on which the Shepherd.Admin Service is running for the Slave.
4. Add the **slaveUser** attribute to the Slave's ShepherdServer object. The slaveUser should point to the distinguished name of the user in the directory under which the Master should login for replication services. All other users' modification requests will be rejected by the Slave. If this Slave will replicate data to other Slaves, this user must also be the user set in the shepherd.ini file of the Slave.
5. If this Slave will replicate to other Slaves, go to step 6. If this Slave will not replicate to other Slaves, enter Hook entries similar to the following:

```
dn: cn=sdsrSlaveModifyHook, c=US
cn: sdsrSlaveModifyHook
objectclass: ShepherdHook
hostServer: cn=Slave1, c=US
hookMethod: modifyObject
api: replicate\sdsrepl.dll!sdsSlaveReplicationHook
```

```
dn: cn=sdsrSlaveDeleteHook, c=US
cn: sdsrSlaveDeleteHook
objectclass: ShepherdHook
hostServer: cn=Slave1, c=US
hookMethod: deleteObject
api: replicate\sdsrepl.dll!sdsSlaveReplicationHook
```

```
dn: cn=sdsrSlaveAddHook, c=US
cn: sdsrSlaveAddHook
objectclass: ShepherdHook
hostServer: cn=Slave1, c=US
hookMethod: addObject
api: replicate\sdsrepl.dll!sdsSlaveReplicationHook
```

These entries cause the Slave to reject all modification requests except those sent by the user specified in the slaveUser attribute of the Slave's ShepherdServer object. Proceed to step 7.

6. Complete the following entries for a Slave that will replicate to other Slaves:

```
dn: cn=sdsrSlaveModifyHook, c=US
cn: sdsrSlaveModifyHook
objectclass: ShepherdHook
hostServer: cn=Slave1, c=US
hookMethod: modifyObject
api: replicate\sdsrepl.dll!sdsSlaveReReplicationHook
```

```
dn: cn=sdsrSlaveDeleteHook, c=US
cn: sdsrSlaveDeleteHook
objectclass: ShepherdHook
hostServer: cn=Slave1, c=US
hookMethod: deleteObject
api: replicate\sdsrepl.dll!sdsSlaveReReplicationHook
```

```
dn: cn=sdsrSlaveAddHook, c=US
cn: sdsrSlaveAddHook
objectclass: ShepherdHook
hostServer: cn=Slave1, c=US
hookMethod: addObject
api: replicate\sdsrepl.dll!sdsSlaveReReplicationHook
```

These entries cause the Slave to reject all modification requests by users other than the slaveUser specified in the ShepherdServer. They also log all changes completed for the slaveUser for replication to other Slaves.

To complete setting up chained replication service for this slave, an Agent entry must be made for the SRS Agent and a replicationInterval must be added to the Slave's ShepherdServer object. Refer to Steps 1 and 2 of the previous section, Setting Up the Master, for instructions.

7. Shut down the Master (IMPORTANT!).
8. Copy the file pointed to by the file field in the dsp section of the shepherd.ini file to the Slave and points its shepherd.ini entries to the new file.
9. Start the Master.
10. Start the Slave.

At this point, all modifications made to the Master directory should appear on the Slave after the

replicationInterval has passed. Consult the SRS Agent's log files if there are problems with replication.

Fault Tolerance

The design of SRS helps it resist potential problems due to software or hardware failures. Using a status file, the SRS Agent keeps close track of which operations have completed successfully and which have not completed successfully. In some cases, a failure will result in commands being replayed on a Slave. However, this should not cause the Master and Slave directories to fall out of sync.

One area that could result in different data appearing on the Slave and Master relates to the SRS Hooks' logging of data. Since the Hooks execute prior to SDS completing the operation, it is possible that a failure after the Hook but before the SDS operation could cause changes to replicate to Slaves that do not appear on the Master.

In real world scenarios, however, the administrator should recognize that the previous attempt at changing data on the Master failed. The administrator can then run the change again. As a result, the change will replay on the Slaves but cause no damage and bring the Masters and Slaves back in sync.

If at any point Masters and Slaves become badly out of sync, order can be restored by shutting down all Masters and Slaves, deleting all .log and .status files from the replicate subdirectory of all Master and Slave systems, copying the Master's directory database to each of the Slaves, and starting back up all Masters and Slaves.