Rocket UniData

U2 Data Replication User Guide

Version 8.2.1

July 2017
UDT-821-REP-1

Notices

Edition

Publication date: July 2017
Book number: UDT-821-REP-1
Product version: Version 8.2.1

Copyright

Trademarks

Examples

This information might contain examples of data and reports. The examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

License agreement

Note: This product may contain encryption technology. Many countries prohibit or restrict the use, import, or export of encryption technologies, and current use, import, and export regulations should be followed when exporting this product.

Corporate information

Rocket Software, Inc. develops enterprise infrastructure products in four key areas: storage, networks, and compliance; database servers and tools; business information and analytics; and application development, integration, and modernization.

Website: www.rocketsoftware.com

Rocket Global Headquarters
77 4th Avenue, Suite 100
Waltham, MA 02451-1468
USA

To contact Rocket Software by telephone for any reason, including obtaining pre-sales information and technical support, use one of the following telephone numbers.

Country	Toll-free telephone number
United States	1-855-577-4323
Australia	1-800-823-405
Belgium	0800-266-65
Canada	1-855-577-4323
China	400-120-9242
France	08-05-08-05-62
Germany	0800-180-0882
Italy	800-878-295
Japan	0800-170-5464
Netherlands	0-800-022-2961
New Zealand	0800-003210
South Africa	0-800-980-818
United Kingdom	0800-520-0439

Contacting Technical Support

The Rocket Community is the primary method of obtaining support. If you have current support and maintenance agreements with Rocket Software, you can access the Rocket Community and report a problem, download an update, or read answers to FAQs. To log in to the Rocket Community or to request a Rocket Community account, go to www.rocketsoftware.com/support.

In addition to using the Rocket Community to obtain support, you can use one of the telephone numbers that are listed above or send an email to support@rocketsoftware.com.

Contents

Chapter 1: Introduction and terminology

U2 Data Replication is used to automate the delivery of read-only copies of UniData files and transactions from the database server to other UniData systems. It can operate in Real-Time, Immediate, Deferred, or Delayed mode. Replication gives you the added ability to set up a standby / failover and failback relationship between servers.

Replication can be used for offloading an inquiry workload. For example, it gives you the ability to have a dedicated reporting system taking the potential resource-heavy user reports away from the production system, allowing more resources for time-critical response screens.

Replication can be used for distributing data to regional centers for local processing. For example, you could move data to regional centers for processing by that center, and you could also have the regional center publish its results back to the central server as well.

Replication can be used for standby systems (Disaster Recovery (DR) / High Availability). It gives you the ability to have a DR system up and running quickly.

The system where the source data is maintained and from which it is distributed is called the *publisher*. A system requesting copies of file updates from the publisher is called a *subscriber*.

A single publisher can be associated with multiple subscribers, and a single subscriber can be associated with multiple publishers.

Replication is publisher driven meaning the publisher pushes transactions to the subscriber.

To identify the files to replicate, you create replication groups that are based on accounts. U2 Data Replication supports both account-level and file-level replication. In account level replication, all data files in the VOC file of an account are replicated. You can configure one account-level replication group and one or more file-level groups for each account. A file-level group is configured in the `repconfig` file. To replicate all supported file level commands, you must have an account level group defined for the account.

In general, for each account that you want to replicate, configure one account level group and one or more file-level groups. This approach distributes the replication load across multiple sets of processes, buffers, and log files and reduces the time required to apply updates. If you have a low volume of updates to replicated files in an account, create one account-level group for the account.

At UniData 8.2, the following commands are replicated:

- DIR file operations:
  - ED or AE
  - COPY
  - CLEAR.FILE
- File-level commands:
  - CREATE.FILE
  - DELETE.FILE
  - CLEAR.FILE
  - CNAME
  - SETFILE
- Other commands:
  - BASIC
  - CATALOG
  - DELETE.CATALOG

UniData does not support the following commands for account-level replication at this release:

- CREATE.INDEX, DELETE.INDEX, BUILD.INDEX, DISABLE.INDEX, ENABLE.INDEX, UPDATE.INDEX
- RESIZE
- CLEAR.ACCOUNT
- SAVE.LIST, COPY.LIST, MERGE.LIST, DELETE.LIST
- CREATE.TRIGGER, DELETE.TRIGGER

Note: The commands that UniData does not support for replication can be executed on the subscribing system; they are not blocked.

At this time, U2 Data Replication does not support the following:

- External Database Access (EDA) files

  Note: The subscribed file can be an EDA file, but you are unable to publish an EDA file.

- Network File Access (NFA) files
- The ability of the replication writer processes to execute a LOGIN or LOGOUT paragraph on a subscribing system
- Account level operations on the following files: `_XML_`, `_EDAMAP_`, `_EDAXMAP_`, `_PH_`, `_KEYSTORE_`, `_ENCINFO_`, `_DEBUG_`, and `_MENU.FILE_`

Replication and the Recoverable File System

You can use U2 Data Replication with or without the Recoverable File System (RFS). However, when used with RFS, the replication failover process makes immediate recovery possible when the primary server fails. U2 Data Replication relies on the RFS crash and media recovery processes to ensure physical and logical consistency of the database. In addition, failover of immediate replication relies on the RFS crash recovery process to generate replication recovery logs to restore possible missing transactions during the failover process.

RFS protects the integrity of a file. Replication provides a copy of the file(s) on another system; it does not directly protect a local system. RFS and replication are not substitutes for one another – they are complementary technologies.

In UniData versions earlier than 7.1, RFS is available on UNIX platforms only. In UniData version 7.1 and later, RFS is available on both UNIX and Windows platforms.

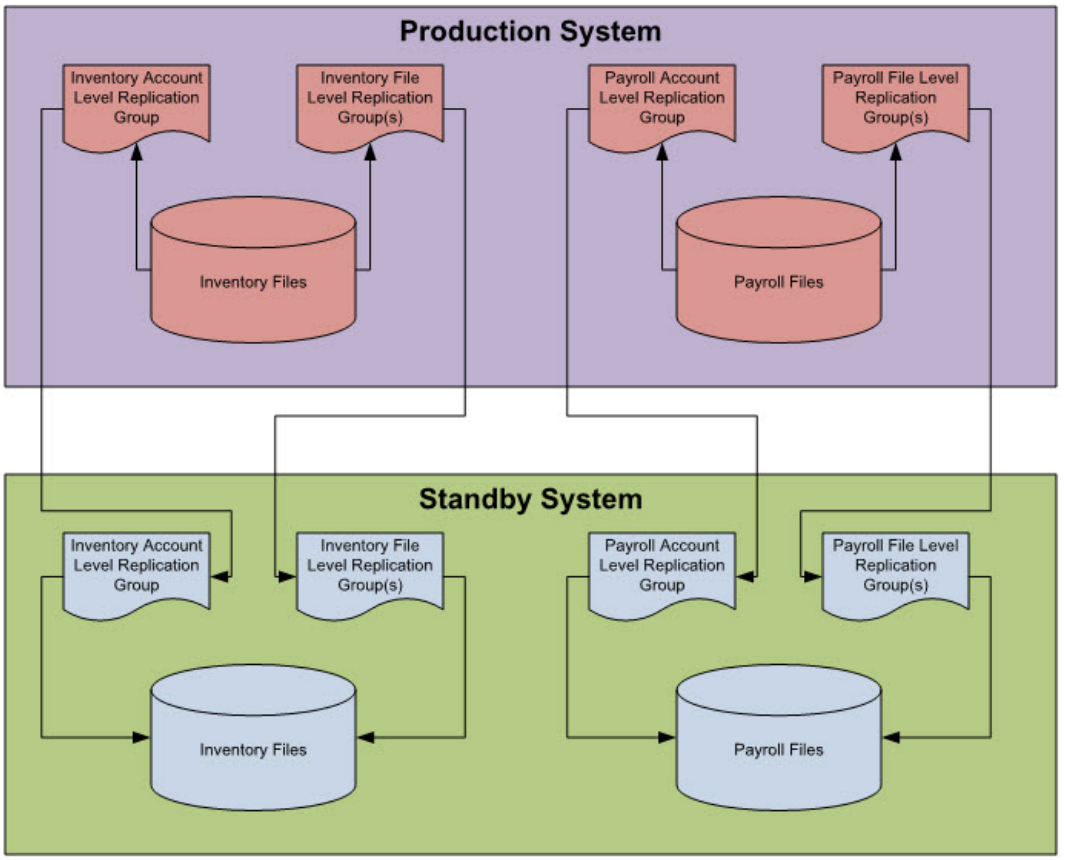
Replication topologies

One publisher and one subscriber

The most common replication topology has a production server that is the publisher and a standby server that is the subscriber. If the production server fails, the standby server can take over.

The following figure illustrates this topology. The production server publishes two accounts: Inventory and Payroll. Each of these accounts is being published by the use of two replication groups. Each replication group has one account-level group and one file-level group (more file-level groups can be

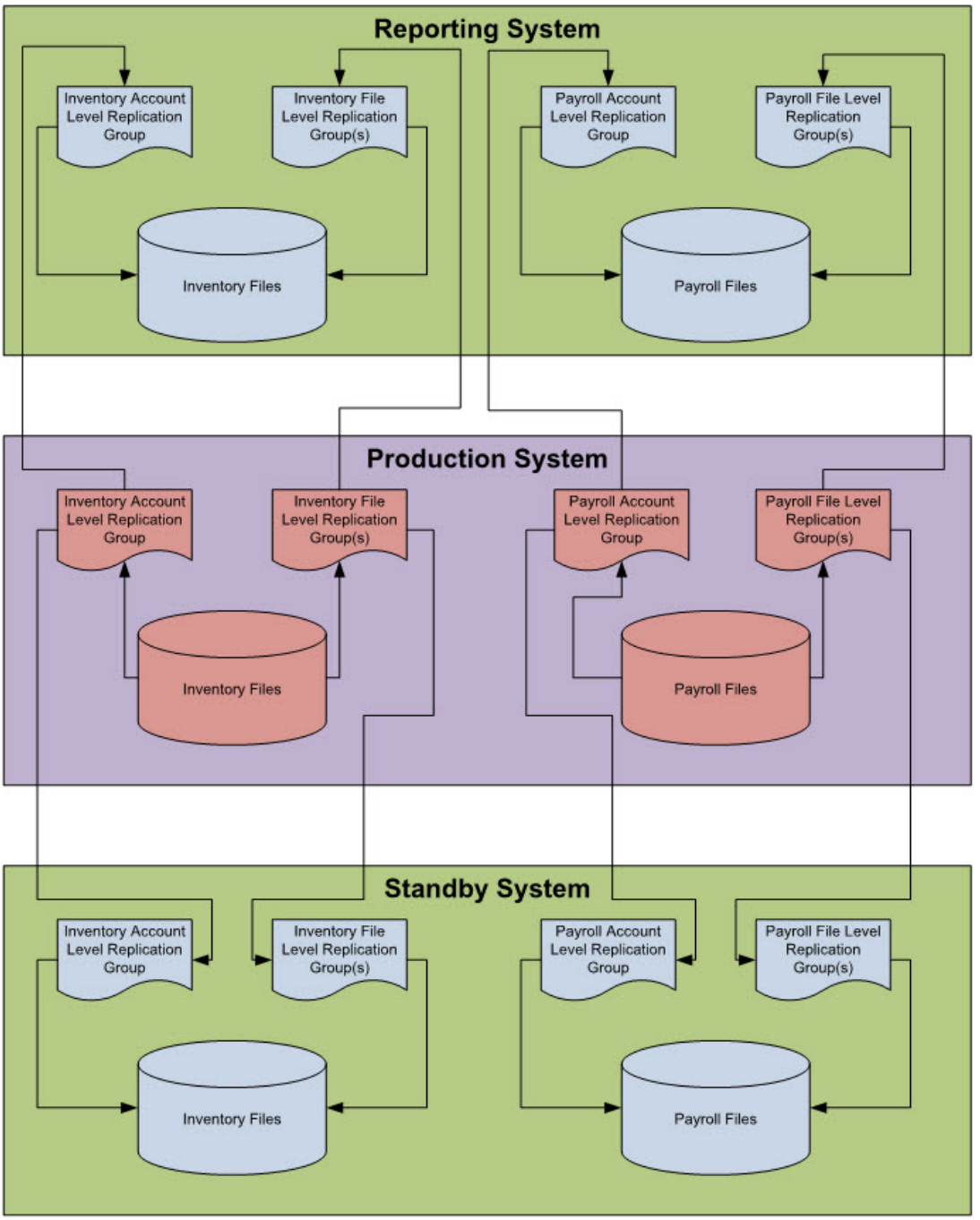
added for performance as needed). The standby server is the subscriber that receives the changes from the production server, which is the publisher.



One publisher and two subscribers

In a three-server topology, the publisher has two subscribers.

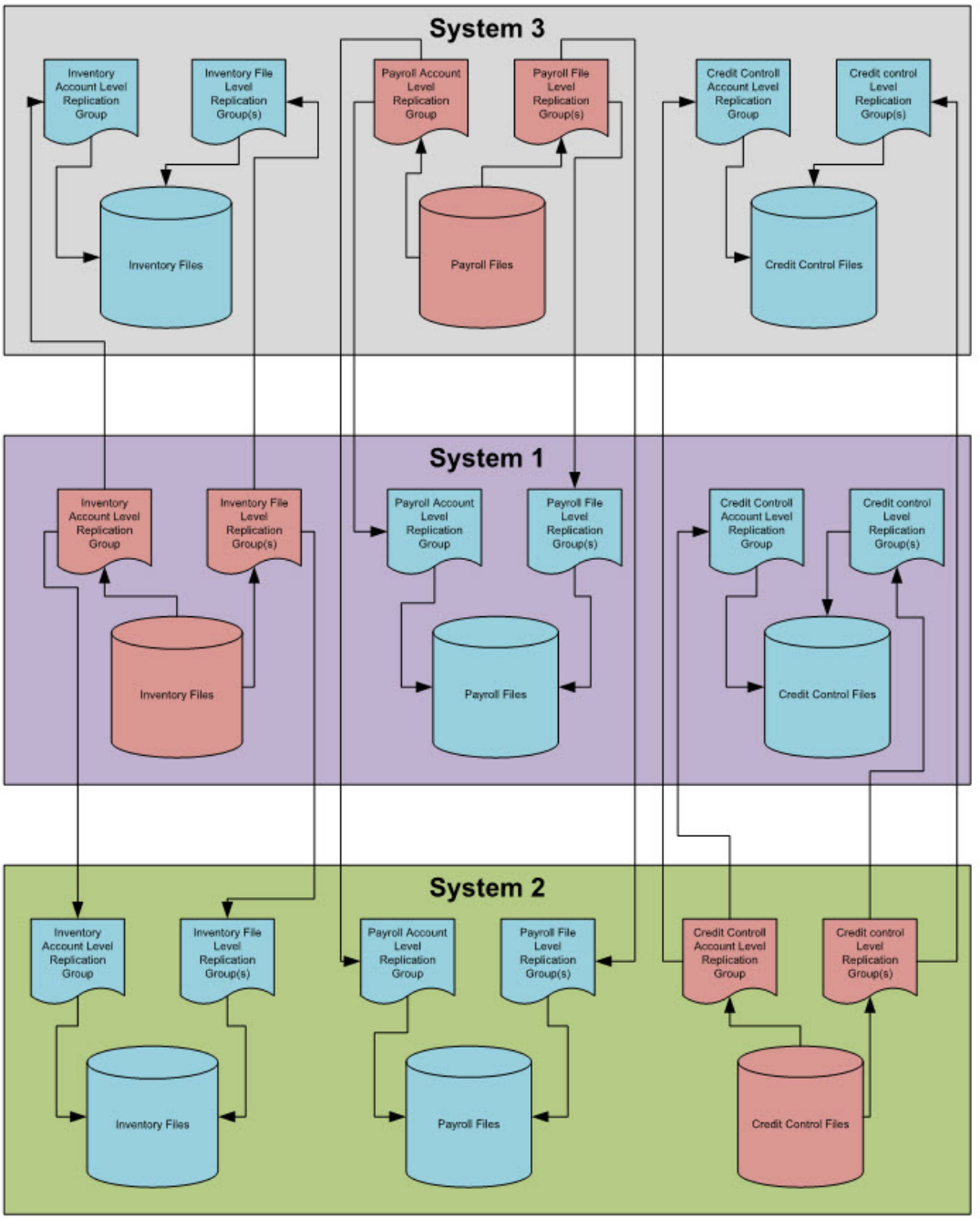
The following figure shows that the subscribers receive updates from the publisher. A typical configuration uses one subscriber as a standby, or failover, server. The second subscriber is a reporting server that is used to offload some of the reporting from the production server.



Multi-way replication

In multi-way replication, each server is a publisher and a subscriber.

The following figure shows three servers and three accounts: Credit Control, Inventory, and Payroll. Each server publishes one account and subscribes to the other two accounts. This type of topology illustrates how you can spread load and resilience across multiple servers.



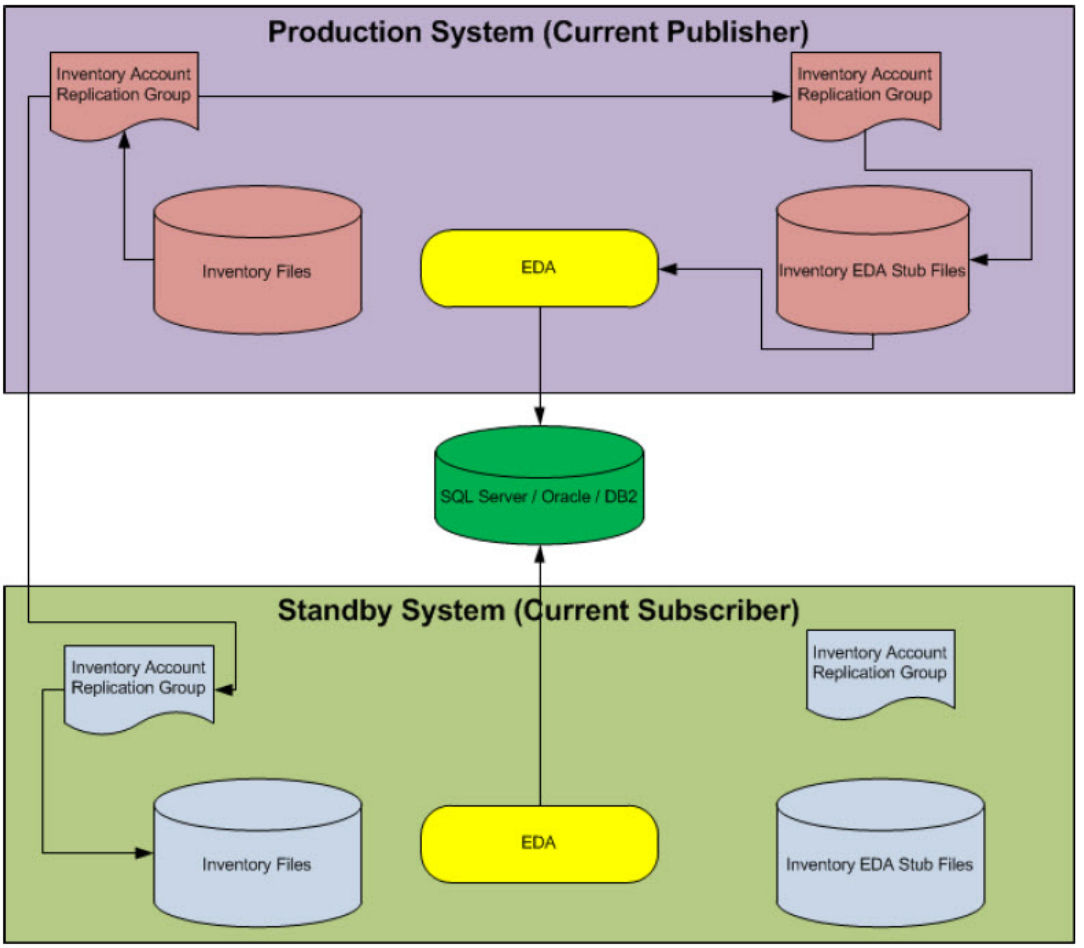
Note: A system can be both a publisher and subscriber, but an individual file can only be either published or subscribed.

External Database Access (EDA) replication to an account on the same server

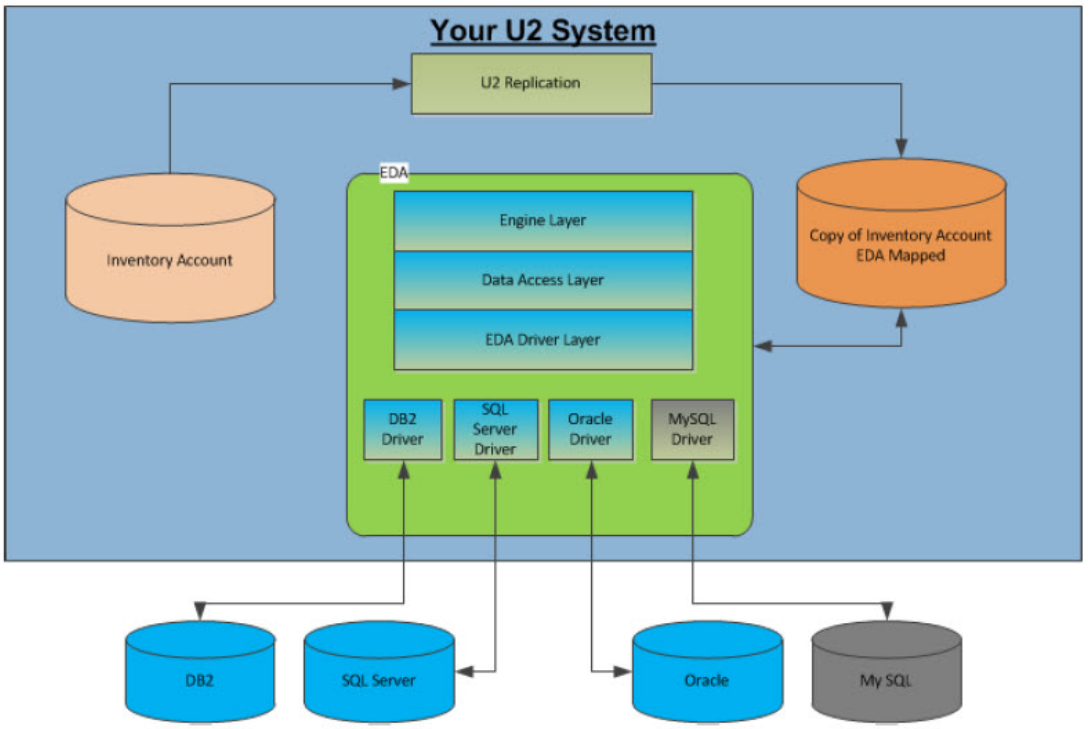
Starting with UniData version 7.3, replication to an External Database Access (EDA) account on the same server is supported. Prior to 7.3, you could replicate from a publishing system to an EDA account set up on a subscribing system, but this required two servers.

To support this new feature, an ACCTNAME phrase was added to `repsys` configuration file. Additionally the syntax for the ACCOUNT and DISTRIBUTION phrases in the `repconfig` file has changed. For more information, see Replication configuration file, on page 20.

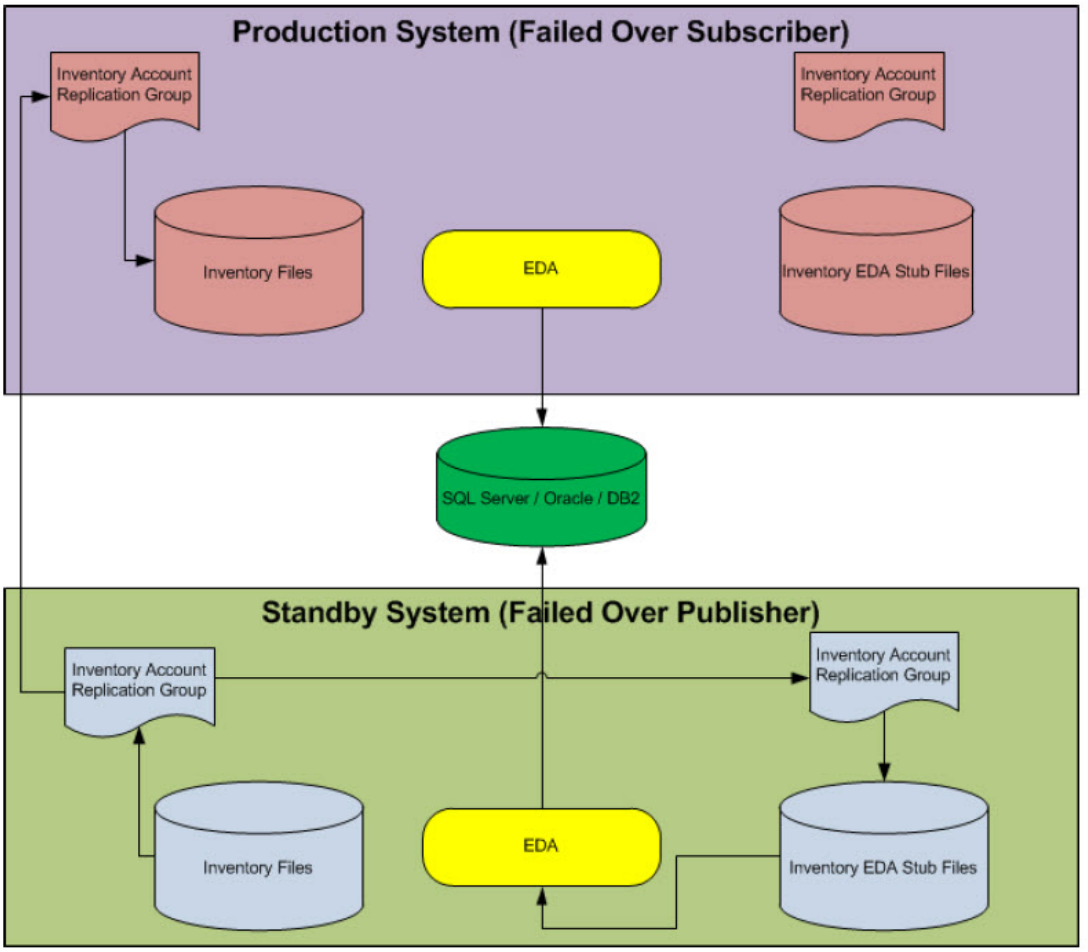
The following figure illustrates how you can also configure the EDA account access in a failover configuration.



Here is a more detailed view of the EDA box noted in the previous diagram. When a UniData account is mapped to EDA files, the UniData files become stub files.



The following figure shows that you are still able to access the EDA files in the external database after having failed over to the original subscribing server.



Note: For more information on how to configure and manage EDA Replication, please refer to the EDA Replication chapter in the *External Database Access (EDA)* guide.

Terminology

This section describes terminology used throughout this manual.

Replication objects

The following objects are used in the U2 Data Replication system:

- Replication object
- Publishing object
- Subscribing object

Replication object

A replication object can be any UniData database file, including static files, dynamic files, sequentially hashed files, directory files, and multilevel files. Each replication object has a unique object name and an account in which it is defined. The object name is the VOC name in the account.

Publishing object

A publishing object is the replication object being replicated. One publishing object may have multiple corresponding subscribing objects on multiple replication sites.

Subscribing object

A subscribing object is a replica of the publishing object. The subscribing object does not have to have the same physical structure as the publishing object. For example, a subscribing object could be a sequentially hashed file, while its publishing object is a static hashed file. A subscribing object must have the same object name as the publishing object.

Replication system

A replication system is a UniData system in a replication environment. The replication system replicates data from one system (publishing server), to other systems (subscribing servers), or both.

Replication group

In a replication environment, a replication group is a group of replication objects belonging to one UniData account. The replication group is the basic unit of replication. Organizing related database objects into a replication group makes it easier to administer many objects together.

You can create and use a replication group to organize the database objects necessary to support a particular database application. You can define multiple replication groups in the same account, but one replication object can belong to only one replication group.

You can replicate one replication group from one publishing account to multiple subscribing accounts.

On a publishing server, a publishing group is the replication group defined in a publishing account of a local system and replicated to remote replication systems.

On a subscribing server, a subscribing group is a replication group defined on a remote replication system and replicated to a local subscribing account on the remote system.

Group distribution

Group distribution defines a copy or replica of objects in a replication group. A replication group must have one publisher distribution and one or more subscriber distributions. A publisher distribution defines the replication system where the group is published. A subscriber distribution defines the replication system(s) that the group subscribes to and the replication type.

You can set up a replication environment that uses two subscribers. Each subscriber distribution is not dependent on other subscriber distributions. For example, the following configures a non-standby reporting subscriber named REPORTING and a delayed standby subscriber named STANDBY:

```
DISTRIBUTION=P:PRIMARY
```

```
DISTRIBUTION=I:REPORTING
DISTRIBUTION=IB480:STANDBY
```

The next example configures one standby subscriber and one delayed standby subscriber:

```
DISTRIBUTION=P:PRIMARY
DISTRIBUTION=IB:STANDBY1
DISTRIBUTION=IB480:STANDBY2
```

The next example configures two delayed standby subscribers:

```
DISTRIBUTION=P:PRIMARY
DISTRIBUTION=IB120:STANDBY1
DISTRIBUTION=IB480:STANDBY2
```

Replication

In a replication environment, a replication is the data replication from a publishing server to a subscribing server. You can have multiple replication groups in one replication.

On a specific system, you usually use a remote system name to represent the replication from a local system to the remote system, or from the remote system to the local system.

Each replication is defined as either one of the following:

- **Standby**: Used to support a standby failover system.
- **Non-standby**: Used to support report applications only.

Types of replication

When you define servers for replication, you specify the replication type, which controls when replication happens. Consider the replication type when you configure the DISTRIBUTION parameter.

Note: If you do not enable transaction processing, each updated record in a file is a single transaction.

The following are available types of replication:

- **Real-time**: In real-time replication, transactions on the publishing server are not committed until all transaction logs have been verified as stored on the subscriber. Updates to the database files on the subscriber may not have completed yet. If replication is suspended, the transaction is not committed until all of the logs for the transaction are written to the log reserve file on the publisher. If the publishing server fails over to the real-time subscribing server, all committed transactions on the publishing server are guaranteed to apply to the standby server. Real-time replication is best for resilience, but worst for performance.
- **Immediate**: In an immediate replication, the publisher's transactions are committed when all logs associated with the transaction have been written to the shared memory replication buffer on the publishing server. The publishing server sends a transaction log to the subscribing server immediately for processing or, if replication is suspended, properly stored in the publisher's log reserve file.

  This type of replication has better performance than real-time replication, but comes with a slight risk of the last-updated data being lost or temporarily unavailable if the publisher fails. Immediate replication is subject to Transaction Boundaries, if used. If the publishing server fails and cannot be restarted, data will be lost. If the publishing server can be restarted and recovered the last update

can also be recovered. though it will not be available on the failover system until recovery steps have been completed. You can use immediate replication for a standby failover system.

- **Deferred**: In deferred replication, the publisher stores transactions in log files and sends the log files to the subscriber at regular intervals. The subscribing server connects to the publishing server, retrieves all of the logs, and synchronizes its database with the publishing database. Deferred replication is used only in non-standby replication.

  Deferred replication has the best performance, but bears the greatest exposure for data loss. The subscriber data is not current with publisher data; however, this mode does allow a quarantine period. A quarantine period means that the subscriber is safe from unwanted events on the publisher for the deferred period.

  You cannot use deferred mode for a standby failover system.

- **Delayed**: Delayed replication delays updates to the subscribing server by a specific amount of time. The publishing server can failover to the subscriber system up to a specified time. This will allow you to cancel or void some unwanted updates during the failover to protect the database from accidental misuse or malicious damage.

  In delayed replication, you can define the delay time period for the subscriber and the publisher. The time period can be defined differently for each system. The time period defined on the subscriber system takes effect at runtime. On the publisher, it takes effect after failover.

Disabling replication

If necessary, U2 Data Replication can be disabled by the administrator without requiring any interruption to the users on the system.

U2 Data Replication can be disabled for any of the following reasons:

- The administrator determines that it is no longer necessary for the replication system to be running, for example, if problems with the subscribing server occur or changes are necessary for the requirements of replication.
- The replication system experiences an unrecoverable failure, for example, the communication link fails between the publisher and subscriber and the log area fills up before the communication link can be restored. A similar situation can occur if a power failure arises at the subscribing server.

Replication logs disk percentage

The replication logs disk percentage is a system-wide limit of disk usage for replication logs. The value is the maximum percentage the replication log files can consume of the total space available on the file system in which the logs are configured. The default configuration parameter is 95%. U2 Data Replication is immediately disabled if this limit is reached, and a full resynchronization of the subscriber is required after this event. If the replication log disk is shared with data files or other applications, you should properly define this percentage to prevent application or database failure due to a growing replication log file size.

The percentage can be configured by REP_DISABLE_DISK_PCT in the replication configuration file. For example,

```
REP_DISABLE_DISK_PCT=95
```

Note: If you turn off replication pacing, the defined replication logs disk percentage is still in effect. U2 Data Replication will be disabled if the replication log reach the maximum disk percentage.

For additional information on the REP_DISABLE_DISK_PCT parameter, see udtconfig parameters, on page 46.

Replication server definition

The replication server definition file (`repsys`), located in `/usr/ud82/include` on UniData for UNIX or `udthome\include` on UniData for Windows platforms, defines all replication systems in a replication environment. This file contains definitions for the publishing server and all subscribing servers that replicate data from or to the publishing server. Each server definition includes a user-defined server name, network address, database version, and other information related to the server and to replication from or to the server.

For more information, see Replication server definition, on page 51.

Replication configuration file

The replication configuration file (`repconfig`), located in `/usr/ud82/include` on UniData for UNIX or `udthome\include` on UniData for Windows platforms, defines all replication groups and usage on the system and their distributions. Each replication group has a unique global name, a list of files in the group, a local account location, a publishing server from which it replicates, and all subscribing servers to which it replicates.

For more information, see Replication configuration file, on page 55.

Data link compression

Data link compression can improve performance by reducing the amount of data that is transferred from the publisher to the subscriber. When data link compression is enabled, the publisher compresses data packets before sending them. The subscriber receives the data packets and decompresses them. This reduces the amount of data being transferred, mitigating the overhead associated with slow network connections.

Field-level replication

Field-level replication saves system space by generating smaller replication files and reducing the amount of data being replicated between the publisher and subscriber. With field-level replication, only updated fields are written to replication logs instead of entire records. Various parameters and configurables, described in the sections that follow, affect field-level replication.

Field-level replication, introduced at UniData v8.2, occurs under the following conditions:

- `FIELDWRITE` commands in your application generate field-level updates.

  The UniBasic `FIELDWRITE` command allows you to update the specified fields in an existing record. For those using replication, `FIELDWRITE` generates field-level replication logs. For more information, please see the `FIELDWRITE` documentation in the *UniBasic Commands Reference* manual.

- `WRITEV` commands generate field-level updates.

The UniBasic `WRITEV` command updates a specified attribute or attributes in a file regardless of lock status. The `WRITEV` command releases locks set by the same process. For more information, please see the `WRITEV` documentation in the *UniBasic Commands Reference* manual.

- The FIELD_UPDATE_THRESHOLD parameter is configured so that `WRITE` commands generate field-level updates.

  For more information, please see the FIELD_UPDATE_THRESHOLD documentation in the *Administering UniData on Windows and UNIX Platforms* manual.

- You define field-level replication objects using the FILE phrase. Fore more information, please see the FILE documentation in Replication configuration file.

Replication pacing

Replication pacing gracefully slows the pace of publisher database updates when replication becomes overflowed. This reduces the likelihood of the replication logs overflowing and ultimately disabling U2 Data Replication.

Pacing gracefully slows the database updates to prevent too many replication overflows to the log files. As the publisher process slowly paces the updates, the subscriber is able to catch up according to priorities set by the administrator. Administrators can define a session priority level, as described in Session priority level, on page 22, to guide the pacing mechanism in recognizing when to slow down and when to resume normal speeds. Instead of overflowing the log file, the number of data updates sent to the log file is reduced, minimizing the possibility of disablement.

Replication pacing criteria and weight

User applications are delayed when the pacing criteria are reached. The pacing criteria include:

- Replication overflows that cause the system to slow down. For example, a large batch job might generate too many updates in a short period of time. This could overwhelm the subscriber and cause replication logs to build up on the publisher.
- Events that might cause U2 Data Replication failure.

Each pacing criterion has a pacing weight. When a session meets all of the pacing criteria, the pacing weight is added to the session. After the update finishes or the transaction is committed, the session totals the pacing weights, and calculates the delay time, and forces the process to wait for that amount of time.

The following table shows the pacing criteria and their weights in milliseconds.

Criterion	Pacing weight
Log Info Buffer (LIB) is *varname* full. The REP_DISABLE_DISK_PCT parameter specifies the percentage, which is 95% by default.	10
Log is added to the log info extension (LIFE) file.	100
Log is added into a Log Extension File (LEF) other than the replication buffer.	4 + 1/gb
The Transaction Control Record (TCR) is created in the Transaction Control Area (TCA) overflow file.	100
Replication logs exceed 80% of maximum replication log size. The REP_DISABLE_DISK_PCT parameter specifies the percentage, which is 95% by default. Only for groups that have LIEF/LEF/LRF overflow.	150

Note: The values in the previous table cannot be changed. To modify the results, the group would need to be modified by setting the REP_PACING value, or the SET.PRIORITY.LEVEL value would need to be modified on a session-by-session basis. Any changes to these values will impact the number of milliseconds that it takes to receive results.

Session priority level

The session priority is the user-defined session execution priority that allows the U2 database scheduler to tune its degradation time if necessary. The values range from 0 to 9; 0 is the highest and 9 is the lowest. The default value is 3.

U2 Data Replication pacing uses the priority level of a session to adjust database-update delay. Sessions that have the highest priority (level 0) do not have any database-update delay. Phantom processes carry the priority level from their parent processes, unless explicitly set in the phantom execution.

SET.PRIORITY.LEVEL command

The ECL verb SET.PRIORITY.LEVEL is used to set the session priority in regards to replication pacing.

The value of SET.PRIORITY.LEVEL is 3 by default, and can range from 0-9, with 0 the highest priority. If you call the verb with no value, it displays the currently assigned value.

For more information, see the *UniData Commands Reference Guide*.

SYSTEM(517)

Used with replication pacing, SYSTEM(517) displays the total degradation time in milliseconds for a session.

SET.PRIORITY.LEVEL fits into the degradation calculation as follows:

<Total Degradation Weight> = Sum(<Degradation Weight>* REP_PACING for the group)

Degradation time for the session defined in milliseconds = <Total Degradation Weight> * SET.PRIORITY.LEVEL

For more information on SYSTEM(517), see the *UniBasic Commands Reference Guide*.

Replication pacing level calculation

The pacing level defines the overall delay level of the group. The higher the pacing parameter is, the longer a delay will occur if the other pacing criteria are in effect. The value is an integer from 0 to 255, where 0 turns off replication pacing for the group. The default value is 5.

To define the pacing level for each group, use the REP_PACING parameter in the `repconfig` file.

When the `udt` or `tmudt` or `tm` (UniData) or `uvsh` (UniVerse) process finishes a single update or transaction commit, all pacing weights are added together to calculate the delay.

The following formula shows the calculation:

Group pacing weight=*Sum*(*Pacing weights*)***Group pacing level*

Delay time=(*Group pacing weight*)***Session priority level**10 microseconds

If there are multiple groups, the *group pacing weight* is the total of all groups being updated.

The process sleeps for the delay time, and then continues with the next instruction.

Replication logs

A replication log represents a transaction or an update to a file. To replicate data, UniData transfers this log to a subscribing server.

- Replication logs are created by a `udt` process and initially written to the appropriate replication buffer for that object's replication group
- If the Recoverable File System (RFS) is active, the replication logs are created by a `tm` process instead of a `udt` process.
- In a replication group, each replication log has a unique replication log sequence number (RepLSN).

Data update log

A data update log is a log representing a data update and contains the following information:

- Log Header
  - RepLSN
  - Length of log
  - Transaction ID
  - Operation flag: Insert, Delete, CLEAR.FILE, or Transaction
  - Pub-Done flag
  - Sub-Done flag
- Log Body
  - File ID
  - Record ID
  - Record contents – including indexed virtual attribute values

Transaction log

A transaction log is a special log representing a transaction other than data updates. Each transaction generates a pair of transaction logs in the replication group:

- Transaction Begin Log
- Transaction End Log

Replication failover log

This is a replication log generated by the subscriber process when it is failing over to a standby system. UniData sends these logs to the original publishing server as part of the resynchronization process.

Replication recovery log

The replication recovery log is a replication log generated from the Recoverable File System (RFS) crash recovery process in order to recover missing transactions during the crash.

Replication log files

U2 Data Replication creates and manages log files on disks to support various aspects of replication. The location of these logs is determined by the REP_LOG_PATH parameter in the `udtconfig` file. The

default location for the replog directory is `$UDTHOME/log/replog` on a UNIX system, or `%UDTHOME%\log\replog` on a Windows system.

The logs created on publishing and subscribing servers have the same names and directory structure. Some of the logs are not used when replication runs normally on a subscribing server, but the logs are required if a publishing server fails over.

One additional log file exists on a subscribing system. This is the Sub Packet File (`.spf`) that is used by the dual-threaded subscriber process.

Replication buffer

When UniData starts, the replication system allocates replication buffers for all replication groups. A replication buffer is a buffer in shared memory that transfers replications logs between the “reader” and “writer” processes. For a publishing group, the writers are the udt and tm processes, and the reader is the publisher process. For a subscribing group, the writer is a subscriber process, and the readers are replication writer processes.

Replication buffer extension file

The replication buffer extension file is an extension of the replication buffer, used to store very large logs and other log files when the replication buffer is full.

The replication buffer extension file is a directory with a control file and part files.

Replication log reserve file

The replication log reserve file saves replication logs on the publishing server when replication is suspended because of a network or server failure, a subscribing server crash, or when an administrator issues the `suspend` command.

The reserved logs are later sent to the subscribing server to synchronize the data on the publishing and subscribing servers. For deferred replication, the replication log reserve file saves replication logs on the publishing server and sends the logs to the subscribing server during synchronization.

When replication is suspended on a subscribing server, the subscriber process saves the logs, which do not have pub-commit notifications from the publisher, and writes them to the replication log reserve file. When you failover the subscribing server, these logs are reloaded into the replication buffer.

The replication log reserve file is a directory with a control file and part files.

Replication subpacket store

The replication subpacket store stores data packets sent from the publisher before the udsub process loads and processes the data. The replication subpacket store consists of the subpacket file and the subpacket buffer.

Each time the udsub process starts a sync operation, the subpacket file is created or truncated. This file has the same format as the replication log reserve file, with a `.spf` file extension.

The subpacket buffer consists of two parts to store the log info and log body, respectively. The log info portion of the buffer has a fixed size of 2300 bytes, while the body buffer size is 128K.

While replication is running, the sublistener thread saves data packets to the subpacket buffer first. If this buffer is full, the sublistener thread saves the data packets in the subpacket file.

If you suspend replication, the udsub process dumps all unsaved replication logs from the subpacket buffer to the subpacket file in case of a failover. If you issue the `failover` command, the replication

logs are loaded into the replication buffer and processed as part of the failover process. If you do not issue a `failover` command, the unloaded data logs are discarded during the next sync operation.

System-level replication log files

The following are types of system-level replication files.

Replication log files	Description
`.udrepsyslog`	Saved replication configuration file (`.udrepsyslog`) is a binary version of the `repsys` and `repconfig` files combined. When the database starts, the repmanager compares the differences between this binary version on disk and the text versions of `repsys` and `repconfig` to determine any changes made to replication configuration after the database was stopped. Beginning at UniData 7.3.0, this file was renamed and moved from the replication log directory to the UniData include directory (where you can find other configuration files such as `udtconfig`, `repconfig`, or `repsys`). The include directory is `/usr/udversion/include` on a UNIX machine and `%UDTHOME%\INCLUDE` on a Windows machine. On UNIX, this is also a hidden file. The file name is `.udrepsyslog`. (Please note the leading '.' in the file name.) This change was part of the replication disablement feature. If you disable replication, you are likely to clear out the replication log directory, as the logs are no longer meaningful. The `.udrepsyslog` file includes status information about whether replication is disabled. As this file is preserved in the include directory, the UniData daemons and replication can be restarted in the correct disablement mode.
`reptemp.tca`	The TCA temp file (`reptemp.tca`) is a temporary file that extends the transaction control area in shared memory. This area coordinates cross-group transactions for applications that use transaction processing to combine multiple updates into a single transaction.

Group-level replication log files

The following are types of replication log files.

Note: The LARGE_RECSZ parameter applies to all group-level log file types. If a log record exceeds the size specified by the LARGE_RECSZ parameter in the `repconfig` file, a body*nn* file on disk records the log body information. An info*nn* file in shared memory stores the log header.

Replication log files	Description
`group_name.gstt`	A group status file (`group_name.gstt`), including log sequence numbers (LSNs) for the replication group, and the group failover status. It is updated when the replication status changes, or at an RFS checkpoint if the Recoverable File System is enabled (even if none of the files in the group are RFS files). An LSN, or log sequence number, is generated by the publisher as replication logs are created. The subscriber then uses this number to maintain the correct order of operations. Additionally, on a publishing system, if the updates to files in a replication group stop, the `udpub` process enters a sleep mode, waiting for additional updates to process. Starting at UniData 6.1.15 and 7.1.2, the group status file is updated before the udpub process starts sleeping.
`group_name.lef`	A log extension directory is an extension of the shared memory replication buffer. When the replication buffer is full, possibly because of a slow network connection to a subscribing server, files in this directory store the contents of replication log files until they can be processed. A `.lef` directory contains the following types of files: ▪ info*nn* files contain log header information. ▪ body*nn* files contain log body information. The MAX_LRF_FILESIZE parameter in the `udtconfig` file controls the maximum size of each info and body file. The files are sequentially numbered. **Note:** Prior to 7.1.8, UniData supported only two log extension files per replication group. The `group_name.lef` file contained log body information and the `group_name.leif` file contained log header information. Each of these files (not directories) was limited to a maximum size of 2GB. The directory architecture introduced at 7.1.8 for the Log Extension Files removes that restrictive size limit. On a publishing server, `udt` and `tm` (if RFS is active) processes write records to the replication buffer and extension files as needed. Records are read from the buffer by `udpub` processes and sent to the subscribing server. If replication is suspended, the `udpub` processes write to the log reserve file. On a subscribing server, subscriber (`udrw`) processes write to the replication buffer, and replication writer processes read from the replication buffer and update data files.

Replication log files	Description
`group_name.lrf`	Log reserve files are used when two situations occur. First, when replication on a publishing server is temporarily suspended, log reserve files preserve logs for later processing. When the replication group is synched and restarted, the publishing server sends the log reserve files to the subscribing server. Second, if a subscribing server is slow to process and commit logs to the database, the publisher might move some logs from shared memory to the log reserve file. An `.lrf` directory contains the following types of files: ▪ ldstat records the status of the log reserve file. ▪ info*nn* contains log header information. ▪ body*nn* contains log body information.
`group_name.recv`	The recovery log file contains replication logs generated by the recoverable file system during database crash recovery. It has same structure as the log reserve file. The `group_name.recv` directories contain three types of files: ▪ ldstat records the status of the recovery file. ▪ info*nn* contains log header information. ▪ body*nn* contains log body information.
`group_name.failover`	The failover log file is a work file that is used during the failover process. It has same structure as the log reserve file. The `group_name.failover` directories contain three types of files: ▪ ldstat records the status of the failover file. ▪ info*nn* contains log header information. ▪ body*nn* contains log body information.
`group_name.spf`	The subscriber packet file was added to support multi-threaded subscriber performance enhancement. A subscriber packet file stores packets from the publishing server before they are loaded into the replication buffer and made available to replication writer processes. It exists only on a subscribing system. You will also see it on a publishing system if that system was involved in a failover after the replication log directory was last cleared. It is used to store packets received from a publishing system before they are loaded into the replication buffer and made available to the replication writer processes. The subscriber packet file is written to if the subscriber process sub packet buffer overflows during packet processing. All unsaved packets in the sub packet buffer will also be flushed to the `.spf` file when replication suspends. These packets are used if a failover command is issued or truncated when a sync command is processed. This directory contains the following types of files: ▪ ldstat records the status of the subscriber packet file. ▪ info*nn* contains log header information. ▪ body*nn* contains log body information.

Replication log examples

The following figure shows the log files and directories for a publishing server that has four replication groups. This server has not failed-over or recovered since the logs were cleared and replication was restarted. This is the typical list of log files that you see on a normal system.

```
# ls -l replog

total 440
-rw-r-----   1 root     system       101480 Apr 10 14:48 inventory_acct_g0.gstt
drwxrwxrwx   2 root     system          256 Apr 10 13:10 inventory_acct_g0.lef
drwx------   2 root     system          256 Apr 10 13:10 inventory_acct_g0.lrf
-rw-r-----   1 root     system         6728 Apr 10 14:48 inventory_file_g1.gstt
drwxrwxrwx   2 root     system          256 Apr 10 13:10 inventory_file_g1.lef
drwx------   2 root     system          256 Apr 10 13:10 inventory_file_g1.lrf
-rw-r-----   1 root     system        97784 Apr 10 14:48 payroll_acct_g2.gstt
drwxrwxrwx   2 root     system          256 Apr 10 13:10 payroll_acct_g2.lef
drwx------   2 root     system          256 Apr 10 13:10 payroll_acct_g2.lrf
-rw-r-----   1 root     system         4376 Apr 10 14:48 payroll_file_g3.gstt
drwxrwxrwx   2 root     system          256 Apr 10 13:10 payroll_file_g3.lef
drwx------   2 root     system          256 Apr 10 13:10 payroll_file_g3.lrf
-rw-rw-rw-   1 root     system         1024 Apr 10 13:10 reptemp.tca
-rw-r-----   1 root     system         2000 Apr 10 13:10 udrepsys.log
#
```

The following figure shows a list of the logs that were produced after the publishing server crashed and replication failed over to the subscribing server, and then replication failed back to the original publishing server.

Because the publishing server ran as a subscribing server for a period of time, `group_name.spf` logs were also created. `group_name.failover` and `group_name.recy` are created too.

```
# ls -l replog
total 448
drwx------   2 root     system          256 Mar 18 08:01 inventory_acct_g0.failover
-rw-r-----   1 root     system       103328 Mar 18 08:02 inventory_acct_g0.gstt
drwxrwxrwx   2 root     system          256 Mar 18 07:58 inventory_acct_g0.lef
drwx------   2 root     system          256 Mar 18 08:02 inventory_acct_g0.lrf
drwx------   2 root     system          256 Mar 18 08:00 inventory_acct_g0.recv
drwx------   2 root     system          256 Mar 18 08:02 inventory_acct_g0.spf
drwx------   2 root     system          256 Mar 18 08:01 inventory_file_g1.failover
-rw-r-----   1 root     system         6896 Mar 18 08:02 inventory_file_g1.gstt
drwxrwxrwx   2 root     system          256 Mar 18 07:58 inventory_file_g1.lef
drwx------   2 root     system          256 Mar 18 08:02 inventory_file_g1.lrf
drwx------   2 root     system          256 Mar 18 08:00 inventory_file_g1.recv
drwx------   2 root     system          256 Mar 18 08:02 inventory_file_g1.spf
drwx------   2 root     system          256 Mar 18 08:01 payroll_acct_g2.failover
-rw-r-----   1 root     system        98120 Mar 18 08:02 payroll_acct_g2.gstt
drwxrwxrwx   2 root     system          256 Mar 18 07:58 payroll_acct_g2.lef
drwx------   2 root     system          256 Mar 18 08:02 payroll_acct_g2.lrf
drwx------   2 root     system          256 Mar 18 08:00 payroll_acct_g2.recv
drwx------   2 root     system          256 Mar 18 08:02 payroll_acct_g2.spf
drwx------   2 root     system          256 Mar 18 08:01 payroll_file_g3.failover
-rw-r-----   1 root     system         4376 Mar 18 08:02 payroll_file_g3.gstt
drwxrwxrwx   2 root     system          256 Mar 18 07:58 payroll_file_g3.lef
drwx------   2 root     system          256 Mar 18 08:02 payroll_file_g3.lrf
drwx------   2 root     system          256 Mar 18 08:00 payroll_file_g3.recv
drwx------   2 root     system          256 Mar 18 08:02 payroll_file_g3.spf
-rw-rw-rw-   1 root     system         1024 Mar 18 08:01 reptemp.tca
#
```

Each of the log file types that are directories at this level have the same file structure within their directory. The one exception is that the `.lef` directories do not have `ldstat` files. For example, here are the initial `inventory_acct_g0.lrf` and `inventory_acct_g0.lef` directory contents:

```
# ls -l replog/inventory_acct_g0.lrf
total 24
-rw------- 1 root system 4096 Jun 27 13:58 body1
-rw------- 1 root system 4096 Jun 27 13:58 info1
-rw------- 1 root system 4096 Jun 27 13:58 ldstat
#
#
# ls -l replog/inventory_acct_g0.lef
total 16
-rw-rw-rw- 1 root system 4096 Jun 27 13:58 body1
-rw-rw-rw- 1 root system 4096 Jun 27 13:58 info1
#
```

Here is a sample listing of the `replog` directory on a subscribing server. Note the additional Sub Packet File (`.spf`) log files used by the dual-theaded subscriber process.

```
# ls -l replog
total 448
-rw-r----- 1 root system 105368 Jun 27 13:58 inventory_acct_g0.gstt
drwxrwxrwx 2 root system 256 Jun 24 08:20 inventory_acct_g0.lef
drwx------ 2 root system 256 Jun 27 13:58 inventory_acct_g0.lrf
drwx------ 2 root system 256 Jun 27 13:58 inventory_acct_g0.spf
-rw-r----- 1 root system 5024 Jun 27 13:58 inventory_file_g1.gstt
drwxrwxrwx 2 root system 256 Jun 24 08:20 inventory_file_g1.lef
drwx------ 2 root system 256 Jun 27 13:58 inventory_file_g1.lrf
drwx------ 2 root system 256 Jun 27 13:58 inventory_file_g1.spf
-rw-r----- 1 root system 98856 Jun 27 13:58 payroll_acct_g2.gstt
drwxrwxrwx 2 root system 256 Jun 24 08:20 payroll_acct_g2.lef
drwx------ 2 root system 256 Jun 27 13:58 payroll_acct_g2.lrf
drwx------ 2 root system 256 Jun 27 13:58 payroll_acct_g2.spf
-rw-r----- 1 root system 3840 Jun 27 13:58 payroll_file_g3.gstt
drwxrwxrwx 2 root system 256 Jun 24 08:20 payroll_file_g3.lef
drwx------ 2 root system 256 Jun 27 13:58 payroll_file_g3.lrf
drwx------ 2 root system 256 Jun 27 13:58 payroll_file_g3.spf
-rw-rw-rw- 1 root system 1024 Jun 24 08:20 reptemp.tca
-rw-r----- 1 root system 2016 Jun 24 08:20 udrepsys.log
#
```

Manually deleting replication log files

In general, the cleanup of replication log files happens automatically. However, there are cases when you need to manually delete log files.

Some of the log files can grow quite large, however. The most common log growth observed is in the log reserve files on a publisher when replication is suspended for an extended period of time. A database administrator needs to consider what to do if the log files have grown, and decide when it is safe to manually remove them.

The only time that it is safe to remove log files manually is after a complete refresh of the replicated database on the subscribing server (from a clean copy of the files on the publishing server). Before you restart the database and replication on the publishing or subscribing server, remove all log files and directories from the replication log directory. Then when the database restarts on each server, the replication manager recreates all logs with appropriate sizes and status information. Again, this assumes that you created a copy of the publishing server's replication accounts on the subscribing server.

The log extension files (`.lef/body1` and `info1`) are truncated to the minimum size (4096) when the database and replication is restarted. For example, `$UDTBIN/stopud` followed by `$UDTBIN/startud`. Prior to UniData 7.1.10, the higher numbered body and info files were removed. At 7.1.10, they are truncated to 0 bytes, but remain in the directory.

The TCA temp file (`reptemp.tca`) is truncated to the minimum size (1024 KB) when the database and replication restarts. For example, `$UDTBIN/stopud` followed by `$UDTBIN/startud`.

The recovery log file (`group_name.recv`) is needed only during RFS crash recovery. It could be removed after a recovery has been completed. As it typically is not excessively large, it is left there to assist U2 Technical Support with problem diagnostics.

The failover log file (`group_name.failover`) is needed only during failover operations. It could be removed after a failover has completed. As it typically is not excessively large, it is left there to assist U2 Technical Support with problem diagnostics.

The subscriber packet file (`group_name.spf`) is truncated or created on a subscribing system by the sync operation (either via `$UDTBIN/ud_repadmin sync` command or via `$UDTBIN/started` with repsys AUTORESUME=1). It is updated as needed in a subscribing system during normal operations and is used during a failover operation. If replication has suspended and no failover is planned before the next sync operation, you can manually delete these files.

The log reserve files (`group_name.lrf`) on a publishing server generally are deleted or truncated after the logs are processed and sent to the subscribing server.

As an example, with replication suspended, updates resulted in the creation of lrf directory files for a group of body1, body2, body3 and info1, info2. After replication resumes (via "`ud_repadmin` sync") and all replication logs are received and committed on the subscriber (they reach the SubDone state as shown in the Replication Monitor utility), the body and info files revert to their original state with only body1 and info1 (plus ldstat) present, at a size of 4096 KB.

The `ud_repadmin reset` command triggers sequence numbers (LSNs) for the group. Consequently, pending updates are abandoned.

Keep in mind that using `$UDTBIN/startud -i` and `ud_repadmin reset` have the same effect: LSNs are reset and pending updates are abandoned.

For performance reasons, when replication is running normally, LEF `body1` and `info1` files never become smaller. If the files reach the value of the MAX_LRF_SIZE parameter, they remain at that size until the database is stopped and restarted. After the database is restarted, `body1` and `info1` return to 4096 KB.

If additional body and info files are created, such as `body2` or `info2` and `info3`, these files are never removed. When replication is running normally, if these files are no longer needed, they shrink to 0 KB. These 0 KB files also persist after stopping and starting the database.

Replication messages

The following types of messages are used with U2 Data Replication:

- **Pub-commit notification**: Sent from the publisher to the subscriber to notify the subscriber that a specific transaction has been committed on the publishing server.
- **Sub-got notification**: Sent from the subscriber to the publisher verifying that it received the replication logs of a specific transaction.
- **Sub-commit notification** Sent from the subscriber to the publisher verifying that a specific transaction was committed on the subscribing server.

Cross-group transaction

A cross-group transaction (CGT) is a transactional update where the files being updated in a transaction span multiple replication groups.

Prior to UniVerse 11.3.1 and UniData 8.2.1, U2 Data Replication only handled CGTs in a synchronous manner. The synchronized methodology involves updating the subscriber only when the logs from all replication groups involved in the transaction are on the subscriber and successfully loaded into the replication buffer. The subscriber processes do not send a sub-done acknowledgement back to the publisher until all logs associated with the transaction have been committed.

The synchronized methodology allows for the consistency of every transaction and its atomic properties to be kept during a hard failover. However, the replication performance of CGTs can be severely degraded using the synchronized methodology. This can happen when the load across different replication groups is not balanced and the site has heavy data volume. The impact might be so severe that the subscriber will continue to fall further behind and never catch up with the publisher.

At UniVerse 11.3.1 and UniData 8.2.1, asynchronous CGT processing has been added to improve replication performance. In an asynchronous methodology, each replication group handles CGTs separately as a single-group transaction. Writer processes (RWs – uvrw for UniVerse, and udrw for UniData) only wait on logs from their own group to arrive and load before they apply the logs to the subscriber database. The subscriber process for a group sends the sub-done acknowledgement when all of the logs for its own group are committed, without waiting for other involved replication groups. This methodology improves the overall throughput on the subscriber as it eliminates the potential for some replication groups to pause processing while waiting for the other groups in a CGT to complete.

Asynchronous CGT functionality is enabled using the REP_ASYNCHRONOUS_TP `udtconfig` file parameter. When using the asynchronous CGT functionality (REP_ASYNCHRONOUS_TP = 1), consideration should be given to increasing the default value of the TCA_SIZE `udtconfig` file parameter. The Transaction Control Area (TCA) is used more heavily when the asynchronous CGT functionality is enabled. With the default setting in TCA_SIZE, the buffer can overflow and significantly impact U2 Data Replication performance. While the performance impact will be seen primarily on the subscriber, increasing the default value in TCA_SIZE to 16000 or 32000 is recommended on both the publisher and subscriber. This will maintain a consistent configuration in the event of a failover.

Having information recorded on all transactional updates is an integral part of the recovery operation in the event of a hard failover when using asynchronous CGT functionality. The TRANS_COMMIT_LOG file in the $UDTHOME account records transactional updates. When logging is enabled, information on each transaction is recorded in the TRANS_COMMIT_LOG file when a transaction is committed. The transaction commit logging functionality is enabled using the TP_COMMIT_LOGGING `udtconfig` file parameter, described in udtconfig parameters, on page 46. The information recorded when each transaction is committed is displayed in the following example:

```
>LIST DICT TRANS_COMMIT_LOG WITH TYPE = "D"

DICT TRANS_COMMIT_LOG    05:27:53am  19 May 2016  Page    1

Field......... Type & Field........ Conversion.. Column......... Output
 Depth
Name.......... Field. Definition... Code........ Heading........ Format
 Assoc
               Number

@ID            D    0                           TP COMMIT LOG   10R
 S
USERID         D    1                           User ID         10L
 S
ACCT           D    2                           Account Name    10L
 S
```

```
FNAME           D    3                              File Name       10L
 M TP_UPD
ISDICT          D    4                              Is Dict?        2L
 M TP_UPD
OP              D    5                              Operation       2L
 M TP_UPD
RECID           D    6                              Record ID       10L
 M TP_UPD
```

By default, the TRANS_COMMIT_LOG file ships as a dynamic file, but the configuration can be changed as needed. For example, it could be set up as a distributed file that exists on the subscriber. Each user account has a Q pointer to the TRANS_COMMIT_LOG file in $UDTHOME. If desired, the physical location of TRANS_COMMIT_LOG can be moved from $UDTHOME by adjusting the Q pointer entry in each user account.

Note: The transaction commit logging functionality can still be used even without U2 Data Replication so that transactional updates can be logged independent of U2 Data Replication. Although as noted, it is a key part of the failover strategy when using U2 Data Replication and asynchronous CGTs.

The asynchronous methodology allows for the possibility that some updates in the transaction can be committed before updates in other replication groups have been completed. In a hard failover scenario, this could result in a partial transactional update on the subscriber. For example, with a CGT involving three replication groups, updates in two groups might have been processed while the updates in the third group were still pending at the time of the hard failover. In this scenario, the subscriber would have committed updates to some files in the transaction but not all.

To handle this situation, U2 Data Replication has been enhanced to search for any incomplete transactions on the subscriber during a failover operation. These incomplete transactions are recorded and written to the REP_FAILOVER_LOG file in $UDTHOME. The information recorded in the file is displayed in the following example:

```
>LIST DICT REP_FAILOVER_LOG WITH TYPE = "D"

DICT REP_FAILOVER_LOG    05:49:36am  19 May 2016  Page    1

               Type &
Field......... Field. Field........ Conversion.. Column......... Output
 Depth &
Name.......... Number Definition... Code........ Heading........ Format
 Assoc..

@ID            D    0                           Failover Log    6L
 S
TIMESTAMP      D    0                           Failover Timest 8L
 S
                                                 amp
STATUS         D    1                           Failover Status 6R
 S
PUBSYS         D    2                           Publisher System 10L
 M INCOMP_TP
TID            D    3                           Transaction Id  10L
 M INCOMP_TP
PUB_TSTAMP     D    4                           Pub Sys Stamp   8L
 M INCOMP_TP
```

A user-defined failover action script can be developed to review and correct these transactions. This script should be defined using the FAILOVER_ACTION phrase available in the `repsys` file. If the failover script is defined, it is used automatically by U2 Data Replication at the conclusion of the failover processes. The transaction ID and transaction timestamp recorded in the REP_FAILOVER_LOG file can be used to build the record ID to the TRANS_COMMIT_LOG file. And based on the application

logic, the user-defined script can use the information in the TRANS_COMMIT_LOG file to repair the incomplete transaction. A copy of the TRANS_COMMIT_LOG file must be on the subscriber in the event of a hard failover. One method is to use real-time replication to replicate the TRANS_COMMIT_LOG file in its own replication group.

Transaction control record

A transaction control record is a record in a shared memory buffer used to control a cross-group transaction.

For each cross-group transaction that occurs on the publishing server, a transaction control record is created on the subscribing server. The transaction control record contains the following information, which is used to synchronize the processes that deal with the transaction:

- Transaction ID
- Total number of replication groups in the transaction
- Number of groups that received all log files
- Number of groups that committed the transaction
- Number of groups that have aborted the transaction
- A list of replication writers waiting for the transaction

Transaction control area

The transaction control area is a shared memory buffer used for transaction control records. Each subscribing server has one transaction control area.

The transaction control area is configured using the TCA_SIZE parameter.

Processes

There are six different UniData processes (executables) that are spawned on publishing or subscribing systems to perform the tasks involved in replication.

Replication manager daemon (repmanager)

The replication manager, which runs on each publishing server and subscribing server, performs the following tasks:

- Reads and loads replication configuration information when the database starts
- Creates replication processes for each replication group
- Monitors and controls replication processes
- Responds to commands from ud_repadmin and XAdmin
- Reconfigures the replication environment
- Manages failover replication, recovery, and data synchronization of replication data

Note: The repmanager process is often referred to as RM in messages recorded in the `rm.log` and `rm.errrlog` files.

Publisher process

The publisher process (`udpub`) runs on the publishing server, reads the logs of a replication group from the replication buffer, and sends the logs to the subscribing server.

Publisher listener process

The publisher listner process (`publistener`) creates a process that receives information about the status of replication logs from a remote subscriber process and updates the information in the replication buffer.

Publisher-syncing process

To synchronize the publishing and subscribing servers, the publisher-syncing process (`pubsyncer`) runs on the publishing server and sends replication logs from the replication log reserve file to the subscribing server to synchronize subscriptions to their publishers. The `pubsyncer` exists only while the systems are syncing - after replication has been suspended for a period of time.

Subscriber process

A subscriber (`udsub`) runs on the subscribing server, receives logs from the publisher server, and writes the logs to the replication buffer.

The subscriber process consists of two major threads. The sublistener thread receives TCP/IP data packets from the publishing server through a socket connection, then stores the data packets in the sub packet store and sends an acknowledgment back to the publishing server. The sub packet store includes a sub packet buffer, a subscriber process local memory buffer that is non-configurable, and a sub packet file, a disk file that temporarily stores data packets when the sub packet buffer is full and when replication is suspended. The sublistner thread passes the publisher's request to the sub main thread via local memory inter-thread buffer. The sub main thread creates the sublistner thread when the database starts and stops the thread when replication is suspended. The sub main thread loads the replication logs from the sub packet store to the replication buffer and coordinates with replication writer processes to apply the logs on the subscribing database. It also manages cross-group transactions updates and monitors the replication buffer and transaction control area for overflow. This thread also runs commands from the replication manager.

If the subscriber has a delayed time period defined as part of a delayed replication, then the subscriber and replication writer processes are changed. The replication writer process gets the current time from the system clock and compares it to the log's timestamp. If the time difference is less than the subscriber delay time period, the replication writer process places itself into the reader waiting list instead of applying the log update. The subscriber process continues to load logs into the replication buffer and compares the time difference of the last log in the replication buffer and the current time. If the difference is bigger than the subscriber delay time period, the subscriber wakes up the replication writer processes from the reader waiting list.

Replication writer process

The replication writer process (`udrw`) runs on the subscribing server, reads logs from the replication buffer, and applies the updates to the files. A replication group can have one or more replication writer processes.

Additionally, while replication is starting up, one replication writer process is launched for each UniData replication account. The replication writer process reads the `repconfig` file and loads replication object information into the replication buffer. Replication object information is the definition of files being replicated into groups that they are being replicated to. The `udrw` process is spawned on all replication systems, whether publisher or subscriber. These processes exit after loading the replication objects. On a subscribing server, new `udrw` processes are then launched to perform the subscribing object (file) updates.

Here is the first part of the `$UDTBIN/rm.log` file on a publishing system that shows the replication writer processes being launched to load the replication objects into the replication buffer:

```
RW(741404) is started to load rep objects' inum/dnum for account /disk1/udt72/inventory
RW(225450) is started to load rep objects' inum/dnum for account /disk1/udt72/payroll
```

Note: The following should be acknowledged for replication object loading:

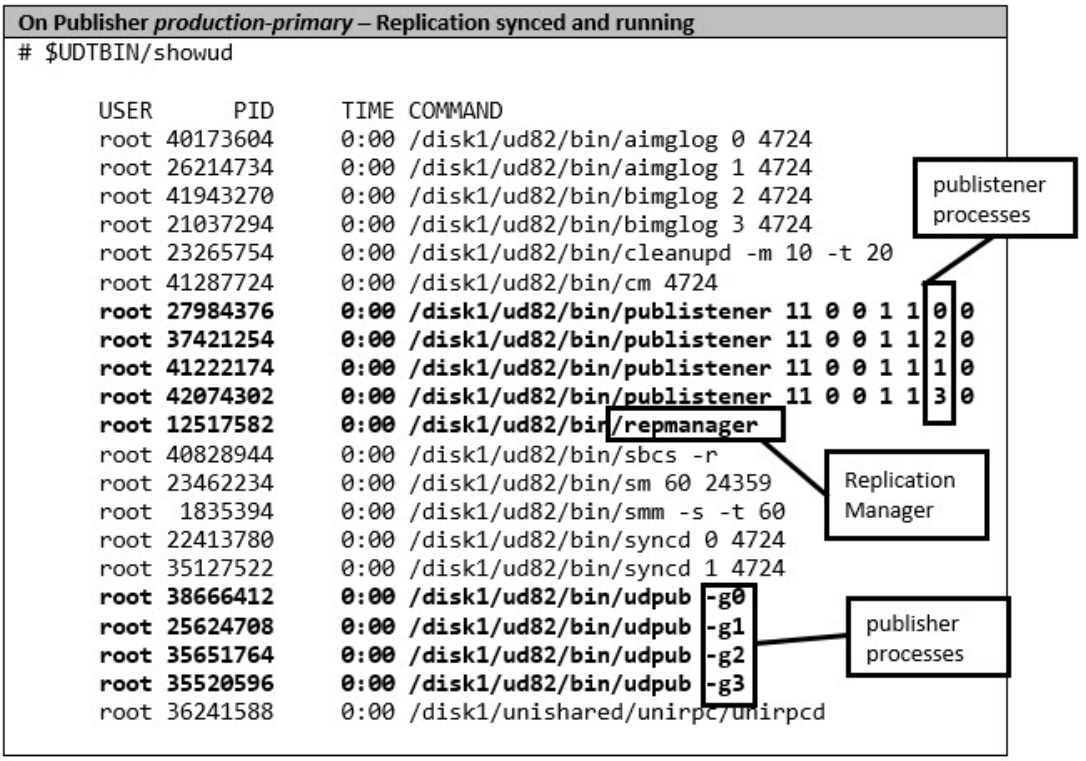
- Prior to UniData version 7.2.0, one `udrw` process was launched for each replication group.
- As of UniData version 7.2.0:
  - One `udrw` process is launched for loading each replicated UniData account.
  - No messages are recorded in `rm.log` for this activity or in `rm.errlog` for problems encountered during this process.
- As of UniData version 7.2.3, the status message shown in the above example was added. In addition, if there are any errors during the process, messages are now recorded in `rm.log` and `rm.errlog`.

Replication processes on a publishing server

The following `showud` output is from a publishing server that is synced and running normally.

- The server has four replication groups, 0-3.
- The server has one `repmanager` process.
- Each replication group has one `udpub` and one `publistener`.

- The groups are numbered from 0 to 3 (in the order of their appearance in the `repconfig` file).
  - For `udpub` processes, this is noted in the –gn argument.
  - For the `publistener` processes, the group number is the next-to-last argument.



Note:

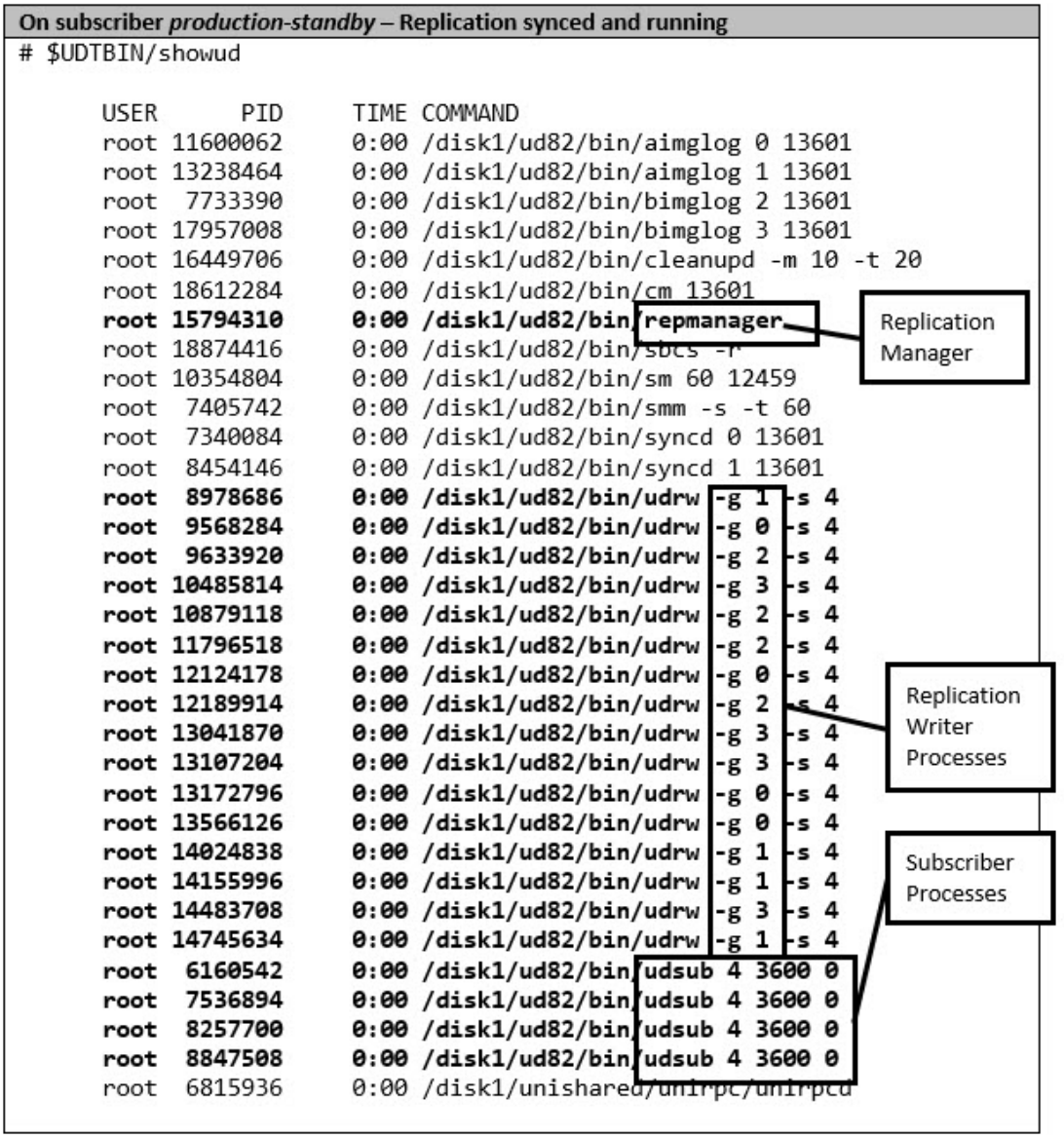
- When replication suspends, the `publistener` processes exit.
- Replication writer (`udrw`) processes are present when replication starts, but exit after loading the replication objects into the replication buffer.
- When synchronization occurs, one `pubsyncer` process is launched for each replication group.
- If a replication group has multiple subscribing server DISTRIBUTION parameters defined, one `publistener` process is launched for each subscribing server.
  - In such a configuration, there is still only one `udpub` process for each group.

Replication processes on a subscribing server

The following `showud -a` output is from a subscribing server that is synced and running normally.

- The server is configured with four replication groups.
- The server has one repmanager process.
- Each group has one subscriber (`udsub` process.

- Each group has a set of replication writer (`udrw`) processes.
  - There are four `udrw` processes for each group.
  - The replication group number is shown in the "-g n" option on each `udrw` process.
  - The number of `udrw` processes per group is set by `repconfig` N_REPWRITER.



Note:

- When replication suspends, all subscriber and replication writer processes exit.
- When replication starts, one replication writer process is launched for each group. After the logs are loaded into the replication buffer, the process exits.

IPC structures

UniData uses UNIX Inter Process Communication (IPC) structures extensively. They include message queues, shared memory, and semaphores. Where these facilities do not exist in Windows, UniData has implemented the same or similar functionality. For example, Windows memory mapped files are used instead of shared memory.

The standard UNIX command for listing IPC structures is ipcs. UniData provides the shell-level `ipcstat` command, which lists IPC structures in the same format as the ipcs command and includes the process that is associated with the structure.

Note: The `ipcstat` listings included in this section were produced on a UNIX system with RFS and replication enabled. They have been edited to remove all non-UniData structures (which were annotated as "unknown" in the original output).

Note: Prior to UniData 7.1.8 on Windows, the output of the `ipcstat` command did not display all RFS and replication structures.

Message queues

When you enable replication on a publisher or subscriber, two message queues are created.

The replication manager uses these queues to perform the following tasks:

- Communicate with the `ud_repadmin` process
- Receive responses from the publisher process (`udpub`) on a publishing server
- Receive responses from the subscriber process (`udsub`) on a subscribing server
- Communicate with the replication manager on a remote system via the `repconn` process. For a publisher, the remote system is the subscriber. For the subscriber, the remote system is the publisher.

If you issue a `ipcstat` command, the output lists the following queues:

- rm R8.2 (request)
- rm R8.2 (reply)

If kernel settings are insufficient (typically `msgmni`) to create the necessary message queues, you will see a message similar to the following in `$UDTBIN/rm.log` when the database is trying to start:

```
# cat $UDTBIN/rm.log
Starting: Wed May  2 12:57:57 2007
Starting: Wed May  2 12:57:57 2007
Wed May 2 12:57:57 Exit: RM cannot create reply msg Q, errno=28.
```

Semaphores

When replication is enabled on a server, semaphores are created to control access to resources, such as replication buffers.

The NUSERS and NSEM_PSET parameters in the `udtconfig` file control the number of semaphores that are created. The number of semaphores created also depends on whether RFS is active on your system. The following formula is used to determine the number of semaphores to create. In the formula, *groups* is the total number of replication groups defined in the replication configuration file.

NUSERS + *group* + 1 / NSEM_PSET

The following formula determines the number of semaphores created if RFS is enabled:

((NUSERS * 2) + *group* + 1 + N_AIMG + N_BIMG + N_ARCH + N_SYNC) / NSEM_PSET

To display a list of semaphores, issue the `ipcstat -s` command. Each semaphore is displayed as rm R8.2 (waiting).

If kernel settings are insufficient to create the necessary semaphores for replication, the `$UDTBIN/rm.log` will display the following message when the database tries to start:

```
Exit: smm: cannot allocate semaphore for replication waiting node 864
errno 28
```

Shared memory

When the database starts on a server that has replication enabled, one or more shared memory segments are created to store information that is associated with replication. The first segment that is created is annotated in an `ipcstat` listing as rm *version* (ctl)

The first part of this shared memory segment is the control section, which is relatively small compared to the replication buffer sections. The following section contains the following items:

- Semaphore information
- Waiting lists
- A transaction control area, which is used to manage cross-group transactions
- A replication table that lists the files that are in the `repconfig` file

The replication table, which is the largest part of the control section, is populated with inode and dnode information for each replication object when replication starts.

When considering the size of shared memory used by replication, the most significant part is the replication buffer that is created for each replication group and is used to store and manage replication logs. The size of each replication buffer is:

- You define the N_LOGINFO, REP_BUFSZ, and N_REPWRITER parameters in the `repconfig` file for each replication group.
- NbrOfDistributions is the number of DISTRIBUTION parameters that you define in the `repconfig` file for the group.

If the sum of the replication buffers for all of the groups plus the size of the control section exceeds the value of the MAX_REP_SHMSZ parameter in the `udtconfig` file, additional shared memory segments are created. These segments are annotated as rm *version* (shmbuf).

Additional considerations:

- The size of a replication buffer cannot exceed the value of the MAX_REP_SHMSZ parameter in the `udtconfig` file.
  - Each replication buffer must fit into one shared memory segment.
- The value of the MAX_REP_SHMSZ parameter cannot exceed the kernel setting for the maximum size of a shared memory segment (typically shmmax).

Predicting the number and size of the shared memory segments that will be created is a bit tricky. At startup, the size of the control section and each replication buffer is calculated. Replication buffers are placed in the ctl segment based on size, not the order in which the group is listed in the `repconfig` file. If some replication buffers do not fit within the control segment, a buffer segment is created to contain as many buffers as will fit. The size of the buffer segment is only large enough to contain the buffers. There is no fixed size for the buffer segments in shared memory.

```
# $UDTBIN/ipcstat -m
IPC status from /dev/mem as of Tue Jun 13 07:19:39 MST 2017
```

```
T         ID     KEY        MODE       OWNER    GROUP
Shared Memory:
m 491782147 0x45041076 --rw-rw-rw-     root   system  -> smm R8.2 (ctl)
m  91226120 0xffffffff --rw-rw-rw-     root   system  -> sm R8.2 (sysbuf)
m 525336588 0x65041076 --rw-rw-rw-     root   system  -> sm R8.2 (ctl)
m 373293078 0xffffffff --rw-rw-rw-     root   system  -> rm R8.2 (shmbuf)
m 521142304 0xffffffff --rw-rw-rw-     root   system  -> rm R8.2 (shmbuf)
m 339738662 0xffffffff --rw-rw-rw-     root   system  -> rm R8.2 (ctl)
m 999292968 0xffffffff --rw-rw-rw-     root   system  -> sm R8.2 (sysbuf)
m 550502442 0xffffffff --rw-rw-rw-     root   system  -> rm R8.2 (shmbuf)
m 652214316 0xffffffff --rw-r--r--     root   system  -> sbcs R8.2
m 504365104 0xffffffff --rw-rw-rw-     root   system  -> rm R8.2 (shmbuf)
m 941621297 0xffffffff --rw-rw-rw-     root   system  -> smm R8.2 (glm)
m 183500850 0xffffffff --rw-rw-rw-     root   system  -> smm R8.2 (shmbuf)
m 502267955 0xffffffff --rw-rw-rw-     root   system  -> sm R8.2 (sysbuf)
```

If kernel settings are insufficient (typically shmmni) to create the necessary shared memory segments, a message similar to the following appears in `$UDTBIN/rm.log` when the database is trying to start:

```
# cat $UDTBIN/rm.log
Starting: Wed May  2 13:37:41 2007
Starting: Wed May  2 13:37:41 2007
Wed May 2 13:37:41 RM: Create shm fails
Wed May 2 13:37:41 RM exit: Setup shm config failed.
Wed May 2 13:37:41 Exit: RM Exit(1)!
```

If you configure UniData so that it tries to create a replication shared memory segment that is larger than the kernel allows, the very same error is generated in `$UDTBIN/rm.log`.

If you get the error displayed above, you need to:

1. Determine if there are enough available shared memory identifiers on the system to allow for the creation of additional segments.
2. Determine if your `udtconfig` and `repconfig` settings would attempt to create a segment larger than allowed by the kernel shmmax setting (using the formula for calculating the segment size that is provided above).

Network

A publishing server uses TCP/IP to interact with a subscribing server over a network. Replication logs and messages are sent to remote systems in TCP/IP packets.

Two unirpcservices components are created for replication when you install UniData.

- unirep82
- rmconn82

These services can be found in the appropriate unishared directory for your UniData installation. Here is a series of commands demonstrating how to find this on a UNIX system:

```
unirep82 /disk1/ud82/bin/udsub * TCP/IP 0 3600
rmconn82 /disk1/ud82/bin/repconn * TCP/IP 0 3600
unirep /disk1/ud82/bin/udsub * TCP/IP 0 3600
rmconn /disk1/ud82/bin/repconn * TCP/IP 0 3600
```

On Windows, the default location for the unirpcservices file is: `C:\U2\unishared\unirpc\unirpcservices`

```
unirep82 C:\u2\ud82\bin\udsub.exe * TCP/IP 0 3600
rmconn82 C:\u2\ud82\bin\repconn.exe * TCP/IP 0 3600
unirep C:\u2\ud82\bin\udsub.exe * TCP/IP 0 3600
rmconn C:\u2\ud82\bin\repconn.exe * TCP/IP 0 3600
```

If the publishing and subscribing servers are behind different firewalls, you must configure the firewalls to allow access for TCP/IP port 31438, which is the default port number for UniRPC.

UDTBIN logs and errlogs

Replication processes log messages and errors to the following files in the `UDTBIN` directory:

- `rm.log`
- `rm.errlog`

The replication manager log file, `rm.log`, contains status messages, records of events, and errors encountered. Reviewing this log is often key to understanding how well replication is functioning on your system. Errors that are logged in the replication manager error log file, `rm.errlog`, are also recorded in the `rm.log`.

rw.errlog

This error log is updated by the `repwriter` processes for problems accessing files defined in `repconfig`. Replication writer processes are spawned on both the publisher and the subscriber when replication is starting up. They read the `repconfig` file and load all of the inodes and dnodes for the configured files into shared memory buffers. If a file specified in `repconfig` does not exist, failure to open it is noted in these `errlog` files and the file is not included in replication.

On a subscribing server, after the initial set of replication writer processes completes loading information into shared memory, the processes exit. A new set of replication writer processes is then launched. These processes read replication logs from the replication buffer and apply updates to the subscriber files. If there are any errors in applying updates, the messages are recorded in the replication writer error log file (`udrw.errlog`).

Note: Prior to 7.2, each replication writer process created a unique error log file. The name of each file was in the format: `rw`*`repwriter PID`*`.errlog`. Starting at 7.2, all replication writer errors are written to the single `rw.errlog` file.

---

udpubn.errlog

n is a group number that is automatically assigned, starting with 0.

To ensure that you can determine the replication groups to which the errors pertain, include a number, starting with 0, in the name of each replication group that you create.

These files contain details of errors that individual publisher processes encounter. Typical errors are often associated with network and TCP/IP.

udsubn.errlog

n denotes the group number starting from 0 as compared to the number of groups in the `repconfig` file.

These files contain details of errors that the individual subscriber processes encounter. Typical errors are often associated with network and TCP/IP. The messages are written to these error logs and to the `rm.log`.

Note: At UniData 7.2, if a file is invalid on the subscribing server, such as unsubscribed, or does not exist, replication will report the cancellation of only one record level update; however, it will report all file level operations. If replication is suspended and synced again, it will reset and report the first cancellation of record operations.

Chapter 2: Installing and configuring U2 Data Replication for UniData

This chapter describes how to install and configure U2 Data Replication for UniData.

Setting up replication

Choose a server on which to create and test your replication configurations. The only requirement for the server is that you must be able to stop and start the database, as necessary, during the testing phase. If you are adding a new server to the environment, for example, to implement fail over, the new server is the ideal candidate.

When you know that replication is working correctly, copy the `repsys`, `repconfig`, and `repacct.def` files to the other server. Then, if the file system on the other server is different, modify the account paths in the `repconfig` file to match the server they were copied from.

Compatibility with previous UniData releases

Many customers implement U2 Data Replication on servers running the same operating system and the same UniData release. However, replication does not require this. You can define a publishing server on an AIX server and subscribe to updates on a Windows or Linux system - or any combination of platforms supported for UniData.

When running replication, you may have different releases of UniData on your publishing and subscribing servers. Sometimes, however, enhancements or fixes to replication result in a change to the communication protocol used. The protocol version is determined by the version of UniData that is being used. All systems in a replication configuration must be running the same protocol. Additionally, replication log structure changes can require the installation of compatible releases on all servers, even if the protocol has not changed.

The following table summarizes compatible ranges of UniData releases for replication systems:

Earliest release	Latest release	Notes
6.0.6	7.1.6	Protocol 10002
7.1.7	7.1.7	Protocol 10003
7.1.8	7.1.16	LEF log structure change
7.1.17	7.1.current	System byte/order fix
7.2.0	7.2.current	Protocol 10004 - Account-level replication
7.3.0	7.3.7	Protocol 10006 - EDA single server and wildcard patterns in EXCLUDED_FILE phrase
8.1.0	8.1.0	Protocol 10007 – Data link compression and performance monitoring
8.1.1	8.1.current	Protocol 10008 – Replication pacing and disablement Replication sub-protocol: 1 Replication log file version: 4

Earliest release	Latest release	Notes
8.2.0	8.2.current	Protocol 10009 - Field level replication, delayed standby replication, and asynchronous cross group transactions Replication sub-protocol:1 Replication log file version: 4

If you do not follow these rules about the UniData release level, replication between two systems will not work.

If you are upgrading from one version of UniData to another version of UniData where the replication protocol level, sub-protocol level, or replication log file version has changed, we recommend completing the following steps:

1. Ensure the subscribing server has received and applied all the updates from the publisher server before commencing the upgrade.
2. Perform a controlled shut down of UniData on both the publisher and subscriber (i.e. do not use `stopud -f`)
3. Upgrade the publishing server first and choose not to start UniData after the upgrade.
4. Clear the contents of the directory as specified by the configurable REP_LOG_PATH in `udtconfig`.
   a. Note if you are using UniData 7.3 and above, please also delete the `.udrepsyslog` file from the `/usr/udnn/include` directory on UNIX and `$UDTHOME\INCLUDE` on Windows
5. You can now start UniData on the publishing server.
6. Repeat Steps 3 to 5 for the subscribing server.

Note: Rocket U2 recommends the minimum version that should be used for replication is UniData 8.2.1.

Upgrading to UniData 8.2.1

After upgrading to UniData 8.2.1, you must clear the replication log directory.

You must upgrade both the publishing server and the subscribing server for replication to run correctly.

These changes are required due to the new Protocol=10009.

For more information on upgrading, please refer to Step 3 and Step 4 of the previous section.

Installing U2 Data Replication for UniData

The UniData installation process prompts whether you want to install U2 Data Replication as part of the standard udtinstall script. The following example illustrates this installation prompt:

```
Turn on the Replication SubSystem? [No]:
```

If you want to install U2 Data Replication for UniData at this time, enter `Yes`.

If you answer yes to this prompt, the installation script then prompts for the name of the replication system ID, as shown in the following example:

```
Replication System ID? [production-standby]
```

The default response to this prompt is the hostname, followed by the version of UniData you are installing.

Note: For detailed information about the UniData installation process, see *Installing and Licensing UniData 8.1 Products*.

You can install replication at a later time by creating the `repsys` file, `repconfig` file, adding the replication rpc services to the rpcservices table, and setting the REP_FLAG parameter in the `udtconfig` file to 1.

Files created by the installation process

UniData performs the following tasks if you choose to install the U2 Data Replication system during the installation process:

- Creates the replication server definition (`repsys`) file, located in `/usr/ud82/include` on UniData for UNIX or `udthome\include` on UniData for Windows platforms.
- Creates the replication configuration file (`repconfig`), located in `/usr/ud82/include` on UniData for UNIX or `udthome\include` on UniData for Windows platforms.
- Creates the `repacct.def` file, located in `/usr/ud82/include` on UniData for UNIX or `udthome\include` on UniData for Windows platforms.
- Inserts the local server definition in the replication server definition file
- Adds the replication rpc services to the rpcservices table.
- Sets the REP_FLAG parameter in `udtconfig` to 1.

Preparing to configure replication

Complete the following steps to prepare replication for configuration.

1. Confirm that the publishing server and the subscribing servers use the same version of the database, or at a minimum, use versions that have compatible replication protocols.
2. Choose a meaningful logical name for each server. Replication logs, error logs, and the replication monitor display these names.
3. Make a list of the accounts that you want to replicate. Include the full file system path to each account.

   You can only use either the account name or the file path but not a combination of both.
4. Create “work” copies of all accounts you want to replicate on the subscribing servers. The contents of the data files do not need to match the publishing server at this point, but all of the files and appropriate VOC entries for those files must be present.

Setting up replication configuration files for the first time

Three files need to be set up on every machine where replication is enabled: `repsys`, `repconfig`, and `repacct.def`. Each time the database starts, it uses and checks these files.

When replication starts, the configuration files on all of the servers are checked to ensure that the parameters match or are appropriate.

A system administrator can edit the replication server definition file (`repsys`) and the replication configuration file (`repconfig`), to define and configure the replication groups. UniData loads the new configuration parameters when you execute the `ud_repadmin reconfig` command.

udtconfig file

udtconfig contains parameters that define the UniData kernel, including those that directly affect replication. The values can be changed to get the best performance or to accommodate more resources for replication.

The udtconfig file is major-release specific. It is located in the usr/udversion/include directory on a UNIX machine, or in the %UDTHOME%\INCLUDE directory on a Windows machine.

To modify the udtconfig file, use a text editor or the Configuration Editor in XAdmin.

The following is a sample replication section from the udtconfig file

```
#
# Section 7 Replication parameters
#
REP_FLAG=1
TCA_SIZE=2048000
MAX_LRF_FILESIZE=1073741824
N_REP_OPEN_FILE=8
MAX_REP_DISTRIB=1
REP_CP_TIMEOUT=300
MAX_REP_SHMSZ=0
REP_LOG_PATH=/disk2/logs/udreplication/ud82
UDR_CONVERT_CHAR=0
REP_DISABLE_DISK_PCT=95
TP_COMMIT_LOGGING=0
REP_ASYNCHRONOUS_TP=0
FIELD_UPDATE_THRESHOLD=0
MAX_REP_GROUPS=512
#
```

udtconfig parameters

The following table lists the udtconfig file parameters that pertain to U2 Data Replication:

Parameter	Description
MAX_LRF_FILESIZE	The maximum log reserve file size, in bytes. The default value is 134,217,728 (128 MB) This file is used primarily when replication is suspended. This value applies to the size of each body<nn> or info<nn> file in each `<group_name>.lrf` directory. The `.lrf` directories reside in the `udtconfig` REP_LOG_PATH directory. The valid range for this setting is: ▪ Low: 4096 ▪ High: 2,147,483,136 (2GB – 512 bytes) Recommended high setting: 2GB -64MB = 2,080,374,784 **Note:** We recommend a setting of no more than 2GB minus a value that is greater than the length of the largest record your application replicates. For example: 2GB - 64MB = 2,080,374,784 As long as your operating system settings for maximum file size are not exceeded, you may set the value higher than the default setting. Setting a higher value can result in fewer files for Replication to open and manage, boosting the performance of Replication. You should also consider disk requirements for your replication log directories. If your LRF or LEF files have grown to multiple files (such as body2, body3), when the storage is no longer needed, these files are truncated back to 4KB. The primary LEF files (body1 or info1) are not truncated during normal running. They remain at MAX_LRF_FILESIZE until the database is restarted. **Note:** This same parameter controls the maximum size of the following subfiles: ▪ Log Extension File (LEF) subfiles ▪ Sub Packet File (SPF) subfiles on a subscribing system
MAX_REP_DISTRIB	Reserved for internal use.
MAX_REP_GROUPS	The maximum number of semaphores needed to run replication with RFS enabled and meet the required number of replication groups. The replication manager also uses MAX_REP_GROUPS during replication startup and reconfiguration to see if the user has allocated a valid number of replication groups. The default value is 512. If MAX_REP_GROUPS is set to negative, SMM will not start and an error message will be reported. When the number of replication groups defined in the `repconfig` file is more than MAX_REP_GROUPS, an error message similar to the following will be reported to the `rm.log` and `rm.errlog` files: `MAX_REP_GROUPS is too small in udtconfig to start UniData, please increase it to at least a value of #` # represents the actual number of replication groups defined in the `repconfig` file.

Parameter	Description
MAX_REP_SHMSZ	The maximum shared memory buffer segment size for a replication group. This value includes consideration for REP_BUFSZ, N_LOGINFO, distributions, and group control information. The default value is 1,073,741,824 (1 GB) and it must be less than 2 GB. If MAX_REP_SHMSZ_GB is configured, then the maximum shared memory buffer size for a replication group is the sum of MAX_REP_SHMSZ_GB * 1GB + MAX_REP_SHMSZ.
MAX_REP_SHMSZ_GB	The maximum shared memory segment size, in GB, for a replication group. If this value is configured, then the maximum shared memory buffer size for a replication group is the sum of MAX_REP_SHMSZ_GB * 1GB + MAX_REP_SHMSZ.
N_REP_OPEN_FILE	The maximum number of open replication log files for a udt or tm process. The default value is 8.
REP_ASYNCHRONOUS_TP	Specifies how to handle cross-group transactions. When set to 1, cross-group transactions are handled in asynchronous mode. When set to the default value of 0, transactional updates are handled synchronously, and the transaction is committed on the subscriber after all replication groups have been updated.
REP_CP_TIMEOUT	Specifies the checkpoint manger (`cm`) daemon timeout interval for replication at checkpoint. The default value is 200 seconds. If this value is set to 0, the `cm` daemon will not time out. When running Replication on a database with the Recoverable File System (RFS) enabled, replication is suspended if an RFS checkpoint does not complete within the number of seconds specified by this `udtconfig` parameter. The default setting is 300 seconds. This parameter typically comes into play on a subscribing server with cross-group Transaction Processing (TP) semantics active. If TCA_SIZE is too small, the process that flags replication logs as committed during a checkpoint may be very slow. Setting this parameter higher allows the checkpoint to complete without triggering a suspension of replication. If this parameter is set to 0, replication will never time out during a checkpoint – though this setting is not recommended. If there is truly a checkpoint problem, it would be appropriate for replication to suspend. If you need to increase the timeout parameter, we recommend settings in the range of 1200 to 2400. This can occur on a publishing system as well – generally if the RFS logs are under-configured and have overflowed extensively to disk. Here is an example of the type of message that is logged in the `$UDTBIN/rm.log`: Fri Apr 6 14:14:27 2007 Group inventory_acct_g0 Replication to standby is suspended(RFS CM Requested).

Parameter	Description
REP_DISABLE_DISK_PCT	This parameter defines the system-wide limit of disk usage for replication logs. The value is the maximum percentage the replication log files can consume of the total space available on the file system in which the logs are configured. The default configuration parameter is 95%. U2 Data Replication is immediately disabled if this limit is reached, and a full resynchronization of the subscriber is required after this event. If the replication log disk is shared with data files or other applications, you should properly define this percentage to prevent application or database failure due to a growing replication log file size. For example, `REP_DISABLE_DISK_PCT=95`. In the event that you start to approach the percentage defined to disable U2 Data Replication, a warning message is sent to the `rm.log` file. The warning is sent when the total replication log file sizes reaches 80% of the maximum replication log size defined by REP_DISABLE_DISK_PCT. Part of the replication disablement process is to suspend UniData I/O and suspend replication. The replication suspension process will copy any existing log files in the `LEF` directories to the `LRF` directories. If the `LEF` directories contain a large amount of data, the copy process can result in the disk becoming full and ending the disablement process. When this happens, the file I/O suspension flag can be left on. To check if the I/O suspension flag is on, use the `dbpause_status` command at the shell prompt. If set, the `dbresume` command will release the suspension flag. This parameter can be set to a lower value to avoid any failure that may occur during disablement or cause the system to hang.
REP_FLAG	The REP_FLAG parameter enables or disables U2 Data Replication. The following options are available: ▪ 0 - U2 Data Replication is off. ▪ 1 - U2 Data Replication is on. The default value is 0.
REP_LOG_PATH	The full path to the replication log files. The default value is `$UDTHOME/log/replog`. If replication is suspended for an extended period of time, the size of the log reserve files on the publishing server can become very large. Conversely, when a sync command is subsequently issued, subscriber packet files on the subscribing server can temporarily become large. Put the replication log directory on a file system that has adequate available disk space. If the log directory becomes full, replication is automatically suspended. For performance reasons, put the replication logs on a disk that does not also contain data files and transaction logs.

Parameter	Description
TCA_SIZE	The maximum number of entries in the transaction control area (TCA). The TCA is used when more than one replication group is configured, and there are cross-group transactions. ▪ On UNIX: The default value is (NUSERS * 64), with a minimum setting of 2048. ▪ On Windows: Apply this same formula and adjust TCA_SIZE accordingly. If you are using transaction processing, set the value to at least the number of users on the system. If you are not using transaction processing, this parameter is irrelevant.
TP_COMMIT_LOGGING	Specifies whether to use transaction commit logging. The default value is 0. Valid values are: 0, 1, 2, or 3. ▪ 0 - Default; Transactions are not logged. ▪ 1 - The transaction attempts to write a log record during commit. The transaction itself will continue to commit in spite of any errors encountered in writing to the TRANS_COMMIT_LOG file. ▪ 2 - The transaction attempts to write a log record during commit. The entire transaction is aborted if the attempt to write to the TRANS_COMMIT_LOG file fails. ▪ 3 - The function behaves like option 1, except that if the TRANS_COMMIT_LOG file is a UVNet file, the updates to it are in no-wait mode.
UDR_CONVERT_CHAR	When this value is set to 1, if the publishing server and the subscribing server have a different I18N group, UniData converts marks and SQLNULL marks to those on the local machine on the data passed between the two systems. The default value is 0. **Note:** For performance reasons, be sure to set UDR_CONVERT_CHAR to 0 unless it is required by the language group settings on your servers.
MAX_REP_DISTRIB	Reserved for internal use.
MAX_REP_SHMSZ	The maximum shared memory buffer segment size. The default value is 67,108,864 (64 MB). **Note:** The maximum recommended value for MAX_REP_SHMSZ is 2,080,374,784 (2GB minus 64MB). With a certain combination of configuration parameters, it is possible that replication will not start when both SHM_MAX_SIZE and MAX_REP_SHMSZ are set to 2GB.

Two `udtconfig` RFS parameters need to be considered when configuring RFS and replication on a system. They are sometimes overlooked when configuring a subscribing system. In the case of N_AFT, one customer restored a second copy of their accounts on their active subscribing system from a backup tape. When they started UniData processes that opened these additional files, they filled the RFS Active File Table. When the replication writer processes then tried to open additional files to apply replication logs, they failed. This triggered a suspension of replication.

Parameter	Description
BPF_NFILES	The per-process logical limit for the total number of recoverable files that can be opened with UniBasic OPEN statements at one time. If more recoverable files are opened, UniData closes files and then reopens them, causing heavy overhead. This parameter cannot exceed N_AFT. For optimal performance on a subscribing system, this parameter should large enough to accommodate the total number of recoverable files in a single replication group.
N_AFT	The system-wide limit on the number of recoverable files and indexes that can be open at one time. This is the number of slots in the system buffer Active File Table (AFT). Parameter is like MAX_OPEN_FILE but pertains only to recoverable files. A dynamic file counts as one file. Even if more than one user opens the same file, it is counted only once. **Note:** On a subscribing system, it must be greater than the total of all RFS files defined for all the replication groups.

The following is partial sample of an RFS section from the `udtconfig` file:

```
# 3.1 RFS flag
SB_FLAG=1

# 3.2 File related parameters
BPF_NFILES=80
N_PARTFILE=500

# 3.3 AFT related parameters
N_AFT=200
N_AFT_SECTION=1
N_AFT_BUCKET=101
N_AFT_MLF_BUCKET=23
N_TMAFT_BUCKET=19
```

Replication server definition

The replication server definition defines all servers in the replication environment. This file, called `repsys`, resides in `/usr/ud82/include` on UniData for UNIX, or `udthome\include` on UniData for Windows platforms.

The `repsys` file is text-based, so it can be modified with your own preferred editor, or it can be modified using the **Replication** → **System Definition** in XAdmin.

A system section definition starts with a SYSTEM phrase. One section ends when another SYSTEM phrase is defined, or at the end of the `repsys` file.

The following table describes the types of phrases that U2 Data Replication can include.

Phrase	Description
ACCTNAME=*account_name*:*account_path*	Use the ACCTNAME phrase to define an account name in the system. U2 Data Replication uses this account name, together with the system name, to identify a secondary replication. U2 Data Replication also uses the account name, together with the group name, to identify a secondary group instance at runtime. The account name is unique in a replication system. *account_name* can be a string combination of characters, numbers, and special characters, and has a maximum length of 127 bytes. If the ACCTNAME defines an account name, you can use that name in `repconfig` to replace the account path. If `repconfig` uses an account path and the account is not defined by the ACCTNAME phrase in the `repsys` file, U2 Data Replication generates a default account name when you start UniData using the following rules: ▪ When UniData starts, U2 Data Replication searches the system account table for local accounts. The system account table is located in `UDTHOME/sys/UD.ACCOUNT`. ▪ If the last directory name in the account path is unique on the system, U2 Data Replication uses that name. If it is not unique, U2 Data Replication precedes the account name with the parent directory name and an underscore (“_”) until it finds a unique name. For example, if the account path is `/disk1/ud82/demo`, U2 Data Replication tries to use “demo” as the account name. If “demo” already exists, it tries “ud82_demo.” If that already exists, it tries “disk1_ud82_demo.” If U2 Data Replication cannot form a unique name when it reaches the maximum number of characters, it tries to replace the last character with a digit from 0 through 9. If the name is still not unique, it tries to replace the second to last character with a digit, and so forth. U2 Data Replication sends the account name of the publishing account in the handshake of the SYNC operation. For secondary replication, U2 Data Replication also sends the secondary subscribing account name. The account names must match the account names defined on the subscribing server.

Phrase	Description
AUTHORIZATION=1	For remote systems with a static IP address, the publishing server can always trust the subscribing server because the IP address is defined in the `repsys` file. However, if the remote system is a DHCP system, the publishing server cannot verify the IP address. In order to verify the subscribing server, set the AUTHORIZATION phrase value to 1 on the subscribing server location. U2 Data Replication performs an authorization check when a SYNC request is received from that subscribing server. If you define a subscribing server location with an AUTHORIZATION phrase, you must define a connection user name and password on the remote subscribing server in order for the SYNC request to succeed. You can define or change the connection user and password through XAdmin or the repadmin tool with the following command: **ud_repadmin setconnect** *replication* where *replication* is the remote system location defined in the `repsys` file. The repadmin tool prompts for the connection user name and password.
AUTORESUME=[1|0]	Specifies whether replication will be synchronized and resume automatically when the database starts and after a reconfiguration. Specify one of the following values: ▪ 1: Synchronize and start replication ▪ 0: Do not synchronize and start replication **Note:** We recommend setting this value to 1; however, when performing various maintenance tasks, it may be useful to set this value to 0 so UniData will not synchronize automatically with Replication after each step.
DHCP=1	Specifies whether the server uses a dynamic IP address. Specify one of the following values: ▪ 1: The server uses DHCP. ▪ 0: The server does not use DHCP. If you define a remote system as a DHCP server, U2 Data Replication automatically updates the IP address of the remote system when it receives a SYNC request from that IP address. The request will fail if the IP address has not been set or is out of date. If you define a local system as a DHCP system, U2 Data Replication automatically sends the current IP address in the SYNC request to the remote server. If the remote system is a DHCP system, you must set the AUTORESUME parameter (0) to false to avoid a SYNC request failure.
EXCEPTION_ACTION=*path*	Specify the full path to the script that runs if replication fails. **Note:** If you include a sync command in your script, the script should be included on both the publishing and subscribing server; however, if you issue a sync command in your script to automatically re-sync replication, the script should only be included on the subscribing server.

Phrase	Description
HOSTNAME=*host_name or IP_address*	The host name or IP address of the server. It is a required entry in the `repsys` file to define the host name of the replication system location. A system can have only one host name. **Note:** We recommend make sure the choice of using either a hostname or an IP address is consistent across all the machines involved in replication. If you use a hostname on some machines and an IP address on others, replication may report a mismatch in the `repsys` file when you attempt to start UniData.
SYNCINTVAL=*minutes*	Specifies the number of minutes to wait between automatic synchronization. To configure manual synchronization, specify 0 for the *sync_interval*. The SYNCINTVAL parameter applies only to subscribing groups that have deferred replication on the remote system. The parameter does not apply to publishing groups.

SYSTEM=*system_name*	Specifies a user-defined name for a replication system in the replication environment.
	
	You must specify a user-defined name for the local system in the `repsys` file. Normally, UniData adds the local definition to the `repsys` file if you install U2 Data Replication during the UniData installation, as shown in the following example:
	
	`# pg repsys`
	`SYSTEM=production-standby`
	`HOSTNAME=dentas`
	`VERSION=82`
	
	You must include one SYSTEM parameter for each system in the replication environment. When defining a remote system location, you also define the replication from the local system.
	
	system_name is a unique name that you give to the server. *system_name* can contain a combination of alphabetic characters, number, and any of the following characters: ~ ! @ $ % ^ & * - + . / \. The default is hostname_udtversion.

TIMEOUT=*seconds*	Defines the number of seconds to wait for packets from the remote system before suspending replication. The publishing server sends a packet to the subscribing server approximately every 4 seconds when replication is idle. The subscribing server then sends a packet back to the publishing server. If the subscribing server location has the TIMEOUT phrase defined, the `publistener` process counts the time that has elapsed between packets being received. If the amount exceeds the value of the TIMEOUT phrase, replication is suspended. In the value of TIMEOUT is 0, no timeout occurs. It is recommended that you do not set the value of the TIMEOUT phrase to less than 300 seconds, which is equal to 5 minutes.
Version=*version_number*	Specifies the version number of UniData. The version number must be 60 or higher.

Examples

The following examples show a replication system that has three servers, primary-production, production-standby, and production-reporting. Each server runs one UniData system.

Note, the replication server definition file is identical for all three servers.

The following is an example of the primary-production server:

```
#Definition of system primary-production
SYSTEM=primary-production
HOSTNAME=dentap
VERSION=82
AUTORESUME=1
DHCP=0
AUTHORIZATION=0
SYNCINTVAL=0
TIMEOUT=300
EXCEPTION_ACTION=/disk1/ud82/bin/excep_script
FAILOVER_ACTION=/disk1/ud82/bin/failover_script
```

The following is an example of the production-standby server:

```
#Definition of system production-standby
SYSTEM=production-standby
HOSTNAME=dentas
VERSION=82
AUTORESUME=1
DHCP=0
AUTHORIZATION=0
SYNCINTVAL=0
TIMEOUT=300
EXCEPTION_ACTION=/disk1/ud82/bin/excep_script
FAILOVER_ACTION=/disk1/ud82/bin/failover_script
```

The following is an example of the production-reporting server:

```
#Definition of system production-reporting
SYSTEM=production-reporting
HOSTNAME=dentar
VERSION=82
AUTORESUME=1
DHCP=0
AUTHORIZATION=0
SYNCINTVAL=0
TIMEOUT=300
EXCEPTION_ACTION=/disk1/ud82/bin/excep_script
FAILOVER_ACTION=/disk1/ud82/bin/failover_script
```

Replication configuration file

The replication configuration file (`repconfig`), located in the `/usr/ud82/include` on UniData for UNIX and `%UDTHOME%\INCLUDE` on UniData for Windows platforms, defines all replication groups, usage, and distribution within replication systems.

The `repconfig` file is text-based, so it can be modified with your own preferred choice of editor, or it can be modified using the **Replication** → **Publishing / Subscribing groups** in XAdmin.

A group phrase starts a group definition section. The group definition section defines contents, distributions, and configurable parameters of a replication group. One group section ends when another group phrase is defined, or at the end of the `repconfig` file. A new group would be started after all of the configurable parameters for the group were defined.

The syntax for the ACCOUNT and DISTRIBUTION phrases in `repconfig` accommodates the optional `repsys` ACCTNAME definition. There is also a new *Replication Type* of 'E' that is associated with EDA Replication.

At 8.2, the FAILOVER_ACTION phrase was added. Additionally, a new delayed time value that is associated with delayed replication has been added.

Parameter	Description
GROUP=*group_name*	Specifies a group definition. *group_name* is a user-defined name for the group. **Note:** You cannot have two groups defined as the same group name. The replication manager daemon checks the system's `repconfig` file to verify there is only one group name defined. If two or more groups are defined with the same name during startup, the replication manager and UniData fail to start. If attempted during a reconfiguration, `uv_repadmin reconfig -verbose` displays an error and U2 Data Replication uses the previous configuration. In both cases, an error is logged to the `rm.errlog` file. We suggest including "acct" in group name to differentiate it from file-level replication groups. For instance, for the INVENTORY account example we use in this guide, we chose **inventory_acct_g0** As in the example noted, we suggest including a group number in the group name. Numbering should start with 0 and reflect the order in which the replication group exists in the `repconfig` file. This can help you identify the replication daemons that are associated with each replication group when you review active processes on your system.

Parameter	Description
LEVEL=[ACCOUNT|FILE]	Specifies the replication level of the replication group. By default, a replication group is a file-level group. The LEVEL phrase defines the replication level of a replication group. By default, a replication group is a file-level group. While this phrase is optional for a file-level group, we recommend defining it for every group for the purpose of clarity. **Note:** UniData will attempt to verify all file references in the VOC of the specified ACCOUNT. If a file reference cannot be verified, a warning message will be written to the `rm.log` and `rm.errlog` files. In a FILE level group, only files defined as a member of that group will generate a warning message if they cannot be verified. Account-level replication replicates updates to all files in an account, except those defined in the Excluded File phrase. You can define multiple file-level groups in an account, and one account-level group. If a file is defined in a file-level group, UniData replicates updates through the replication group where the file is defined. If you do not explicitly define the file in a group, UniData replicates the updates through the account-level group. **Note:** If you want to replicate all of the file-level commands supported at version 7.2 – you must have an account-level group defined for the account.

Parameter	Description

ACCOUNT=*account_name* | *account_path*	Specifies the local account where the replication group resides.
	
	account_name is a valid account name defined in a repsys ACCTNAME phrase or a valid local account defined in the `UDTHOME/sys/UD.ACCOUNT` file.
	
	account_path is the full path to the directory where the account resides. The actual subscribing system *account_path* may be different than the publishing system, due to different disk and file system layouts on the subscribing system.
	
	This path is used to specify the VOC location that will be opened to find all of the files for an account-level replication group and all files explicitly defined in any FILE phrase. The VOC pointer for each file is opened when Replication starts up and is used to load each file's inode/dnode into the replication buffer.
	
	Combinations of account paths and account names cannot be used. If the account name is defined first in the `repconfig` file, an error similar to the following displays:
	
	`Duplicate account names/paths are defined in repconfig line 4 and 52. Use either account names or account paths exclusively.`
	
	If the account path is defined first, then an error similar to the following displays:
	
	`Account is not accessible(2). RM: Load Replication object fails!`
	
	When using XAdmin to configure replication, the account configurable in the `repconfig` file needs to be an account name from the `UD.ACCOUNT`. If the account name specified is an absolute path that does not point to a correct entry in `UD.ACCOUNT` file, it will not work.

Parameter	Description
FILE= [SUB_WRITEABLE] [*Field Level Keyword*] [DICT | DATA] *filename* [,*subfile*]	Each GROUP section can contain multiple file phrases. A file section defines a replication object in a replication group, and a FILE phrase defines the file name. The *filename* is the logical file name defined in the VOC file. If you specify SUB_WRITEABLE, you can write to the file you specify on the subscribing server. On the publishing server, this does not take effect until system failover. You can use the *Field Level Keyword* to define the replication object to be a field-level replication file, and you can optionally add a *Field Update Threshold* to define the threshold at which automatic field-level update and replication occurs. The syntax for *Field Level Keyword*, which must be all one word with no extra characters, is as follows: *Field Level Keyword*:=FIELD_LEVEL[*Field Update Threshold*] The *Field Update Threshold* is an integer in kilobytes and applies to the specified file only. If a record is updated to a file with length exceeding the threshold, the database generates a field-level replication log; otherwise, the entire record is replicated. **Note:** Automatic field-level update does not occur if a WRITE operation writes an identical record value to an existing file. If the record value is unchanged by the WRITE, no replication log is generated, and the database is not updated. While a value specified for *Field Update Threshold* here applies only to the specified file, you also can set *Field Update Threshold* in the udtconfig file to apply to all files. The value specified in the FILE phrase overrides any value specified in the udtconfig file. **Examples:** The following FILE phrase declares that the replication object DATA STUDENT is sub-writeable, and field-level updates and replication are triggered if the record length exceeds 80 kilobytes: FILE= SUB_WRITEABLE FIELD_LEVEL:80 DATA STUDENT Continued on next page...

Parameter	Description
...Continued: **FILE**= [SUB_WRITEABLE] [*Field Level Keyword*] [DICT | DATA] *filename* [,*subfile*]	...Continued The following FILE phrase declares that the replication object EMPLOYEE has field-level updates and replication on all WRITE operations regardless of record length: `FILE= FIELD_LEVEL EMPLOYEE` **Note:** By default, subscribed files are not writeable on a subscribing system. This is by design. This prevents changes to data on a subscribing system so an administrator can rely on the data matching the publishing system in case of a failover to the subscriber. This should be considered carefully when designating any file as SUB_WRITEABLE. Customers requested the ability to allow a small number of files to be writeable on a subscriber. Typically, these files would be control files that are updated when a user launches the customer's application. Without this capability, they could not start their application to run reports. If you specify DICT, only the dictionary portion of the file is replicated. If you specify DATA, only the DATA portion of the file is replicated. Otherwise, UniData replicates both the dictionary and data portion of the file. **Note:** `repconfig` files from releases prior to 7.2 are fully compatible with 7.2. They do not need to be changed when upgrading a replication system. You can still specify both DATA and DICT in a FILE phrase. You will automatically have both DICT and DATA replicated when you specify a filename in a FILE phrase. If the *filename* you specify is a multilevel file and you do not specify a *subfile* name, UniData replicates the multilevel file and all its part files. It does not include newly created part files after UniData starts. An account-level group can also have file sections. You might want to define a file explicitly in an account-level group if the file has a symbolic link or VOC pointer and you want to clearly define the primary replication file, or you want to define the file with special options. By default, UniData automatically adds the VOC file to the account-level group. **Warning:** If you define a file in more than one replication group, UniData will use one and only one file definition. The file UniData chooses is arbitrary.

Parameter	Description
EXCLUDED_FILE=[DICT | DATA] *filename*[,*subfile*]	Specifies the name of a file or a pattern for the names of files to exclude from replication. In the `repconfig` or `repacct.def` files, define the pattern for the files you want to exclude using the EXCLUDED_FILE phrase. This is typically used in an account-level group, but may be used in a file-level group as well. In an account-level group, you can exclude files that have VOC pointers, but do not need to be replicated. Candidates might include: ENGLISH.MSG, HELP.FILE, AE_COMS, AE_DOC, _MAP_, and ERRMSG. If neither DICT nor DATA is specified, both the data portion and the dictionary are excluded. If no subfile is specified for a multilevel file, all parts of that file are excluded. **Note:** If the same file is named in a FILE phrase and an EXCLUDED_FILE phrase, the file is may or may not be excluded If a file is explicitly named in an EXCLUDED_FILE phrase (without using 7.3.1 wildcard syntax), the file is excluded – even if it is also named in a FILE phrase. If the file is included in an EXCLUDED_FILE phrase due to wildcard syntax evaluation, it will still be included if it is explicitly named in a FILE phrase. Use a wildcard patterns when your application creates temporary files and you do not know the names of the files in advance. **EXCLUDED_FILE**={DICT | DATA} [...]*filename*[,*subfile*][...] where the ellipses (…) are wildcards. If a wildcard exists in the EXCLUDED_FILE phrase, it describes an excluded pattern rather than an excluded file. For example, the phrase `EXCLUDED_FILE=...WORK.FILE...` excludes any file that contains `WORK.FILE`. `EXCLUDED_FILE=WORK...` excludes any file that has `WORK*` at the file directory level, data files, and dictionary files. `EXCLUDED_FILE=WORK.FILE,...` excludes all subfiles of the level file `WORK.FILE` and all level files that begin with `WORK.FILE`. The level file is the file defined at the top level in the VOC and is not a subfile. `EXCLUDED_FILE=...WORK.FILE,...` excludes all files that end with `WORK.FILE` and all subfiles of all level files that end with `WORK.FILE`.

Parameter	Description
… Continued: **EXCLUDED_FILE**=[DICT | DATA] *filename*[,*subfile*]	Continued …. In account-level replication, the RESERVED_FILE_SPACE parameter defines the number of dynamic objects for both included and excluded objects. Each dynamic excluded object created for a file uses one slot in the dynamic object table. The default `repacct.def` file contains the following excluded file phrase: `EXCLUDED_FILE=VI.TMP...`
DISTRIBUTION= *replication_type*: *system_name* [*account_identifier*]	Specifies the publishing server from which the group replicates or a subscribing server to which the group replicates. Each GROUP section must have one DISTRIBUTION parameter that defines the publishing server and a minimum of one DISTRIBUTION parameter that defines the subscribing server. *replication_type* defines the distribution and replication type. See the following tables for a description of these subscribing types. *system_name* is one of the systems previously defined in the `repsys` file. One replication group must have one, and only one, local system distribution. Optionally, you can use the LOCAL keyword to represent the local system name. *account_identifier* defines a secondary distribution. *account_identifier* must satisfy the following rules: ▪ *replication_type* can only be a nonstandby replication system ▪ *system_name* can be either the publishing server name or the subscribing server name ▪ *account_identifer* must be a different account than the primary account of the group defined by the ACCOUNT phrase **Note:** Each group section must contain one DISTRIBUTION parameter that defines the publishing server and one or more DISTRIBUTION parameters that define subscribing servers.
REP_PACING=*value*	Defines the replication pacing level in a group. *value* is the level from 0 to 255. 0 turns off pacing functionality. The default value is 5.

Parameter	Description
RFSFAILOVER=*system_name*	Defines the behavior of U2 Data Replication when a publishing group starts. If you are running RFS, when UniData starts, it checks to see if the system was properly shut down when UniData stopped. If it was not, UniData runs crash recovery to recover the database. When a publishing group starts after crash recovery finishes, it must determine if the group failed over to a standby server. If not found, it checks and uses the RFSFAILOVER setting. This setting is used by the DBA to define the intention to failover and which system to failover to. If it is not defined, this indicates that the DBA's preference is to not failover. U2 Data Replication first tries to locate the failover system by connecting to its standby subscribing servers. If one standby server failed over to become the new publishing server for the group, the system that was recovered becomes a subscriber of the group. If U2 Data Replication does not locate a standby server that failed over, it relies on the failover phrase to determine if the group needs to failover, and to which failover system. If you do not define the failover phrase, U2 Data Replication restarts the replication group as a publishing group. **Note:** If you define a failover phrase, you can still failover to a different standby server than the one defined by the failover phrase using the `FAILOVER` command to change the publishing server.

As described in the previous table, DISTRIBUTION should contain a *replication_type*, which defines the distribution and replication type. The following replication types are available:

Replication type	Description
R	Real-time subscribing server
R[B | E]	Real-time standby subscribing server, used for failover
I	Immediate subscribing server
I[B | E]	Immediate standby subscribing server, used for failover
D	Deferred subscribing server
P	Publishing server

Note: If 'E' mode is specified on a remote Secondary Distribution, the publisher does not sync it at run time. It does not have any effect on other type of distributions. This is designed for a configuration that includes a local EDA replicated account as well as a hot-standby replication server. While there is a comparable EDA configured account on the subscribing system, updates are only written to the local replicated EDA account when that system is the active publishing system.

Hot standby servers	Description
B*delayed_time_period*	Used to denote that the subscribing server is a hot standby server that can be failed over to if required. The optional *delayed_time_period* defines the subscriber delay time period. This option is not allowed with deferred replication (D).

Optional switches	Description
E	Turns on EDA mode, which allows for replication objects to be EDA mapped on multiple systems, but ensures that only one system will be applying the updates to the EDA source at any one time. The system that receives and applies the updates is determined by the status of the publisher or subscriber and any failed over status.
C	Turns on Data link compression, on page 20. When C is specified in the subscribing type, the data transfer from the publisher to the subscriber is compressed.

Configuration phrases

Configuration phrases define configurable parameters for the replication group and are also used in the `repconfig` file. The following table describes the configurable parameters.

Parameter	Description
RFS Failover System	Defines the behavior of U2 Data Replication when a publishing group starts. If you are running RFS, when UniData starts, it checks to see if the system was properly shut down when UniData stopped. If it was not, UniData runs crash recovery to recover the database.
N_LOGINFO	Specifies the maximum number of replication logs, which are created for each file update, to load in the replication buffer. If the number of logs exceeds this value, the excess replication logs are stored in the log extension file (LEF) on disk. The default value is 4096 replication logs. For best performance, avoid writing replication logs to the LEF. For an active group, it would be reasonable to increase N_LOGINFO to 8192. If you change this setting, you should adjust the REP_BUFSZ so it is large enough to hold all of the logs.
REP_BUFSZ	Specifies the size of the replication buffer that stores the log body for the replication group. The default value is 1048576 bytes. **Note:** Increasing the size of this buffer can aid the performance of both the publishing and subscribing systems by allowing logs to be processed in shared memory. Consider adjusting N_LOGINFO as well. Use the following formula: REP_BUFSZ = N_LOGINFO * Average Record Size When calculating the average record size for the files in a replication group, you need to include key length, data length, and the length of all indexed virtual fields.
LARGE_RECSZ	This parameter defines the largest record size that is managed in the shared memory replication buffer. When a record is larger than the value of LARGE_RECSZ, the database stores it in the LEF instead of the replication buffer. The default value is 64 KB. **Note:** The log size calculation includes the length of the file name, record key, data record, and indexed virtual field values. For performance reasons, most records should be handled in shared memory.

Parameter	Description

RESERVED_FILE_SPACE	Defines the number of file slots to reserve for an account-level group to allocate for files that will be created in an account after replication starts. File slots correspond to the available rows in the replication object table. The default value is 500 file slots.
	`RESERVED_FILE_SPACE=<number of reserved files>`
	If the RESERVED_FILE_SPACE table is full, a `CREATE.FILE` command fails with an error similar to the following example:
	`Replication object table is full: Enable Replication object TEST3 failed.`
	`Creating file TEST3 failed.`
	Note: If your reserved space becomes full, you can provide fresh reserve slots without having to stop and restart the database. Use the "$UDTBIN/ud_repadmin reconfig" command to reset your available slots to the number you specified in the RESERVED_FILE_SPACE phrase.
	To avoid stopping and restarting the database, perform the following steps:
	1. Increase the value of the RESERVED_FILE_SPACE parameter for the replication group.
	2. Run the `reconfig` command on each subscribing server for the replication group.
	3. Run the `reconfig` command on the publishing server.
	You will need to run `reconfig` first on all subscribing servers defined in the distributions for that replication group, and then on the publishing system.

Pacing level	The pacing level defines the overall delay level of the group. The higher the pacing parameter is, the longer a delay will occur if the other pacing criteria are in effect. The value is an integer from 0 to 255, where 0 turns off replication pacing for the group. The default value is 5.

When the RESERVED_FILE_SPACE is full, the `create.file` command fails and a message similar to the following example is displayed:

```
Replication object table is full: Enable Replication object TEST3
 failed.

Creating file TEST3 failed.
============>>>> NOTE: CREATE.FILE failed! <<<<<<==============

Note - file created but cannot be replicated. Table full = fatal error.
```

To reload the replication table and specify additional slots without stopping the database, use the `ud_repadmin reconfig` command.

Replication writer configuration phrases

Replication writer configuration phrases define configurable parameters for replication writer processes. These parameters are used only by a subscribing group. We recommend that you define this for the publishing group if you want to failover the publishing group to a standby subscribing server. The group is treated as a subscribing group after the failover.

The following table describes the replication writer configurable parameters.

Parameter	Description
N_REPWRITER	The number of replication writer processes for the replication group. Specify more than one replication writer per group, but do not specify a value that is significantly higher than the number of CPUs on the subscribing server. On a subscribing server, NUSERS parameter in the `udtconfig` must be large enough to accommodate the total number of replication writer processes configured for all groups.
RW_UID= *user_ID*	**UNIX only.** The user ID of the replication writer process. By default, each replication writer starts as root. If you define RW_UID, the replication writer process changes the user ID to the one you specify. For more information on the RW_UID parameter, see Miscellaneous, on page 165.
RW_GID= *group_ID*	**UNIX only.** The group ID of the replication writer process. If you define RW_GID, the replication writer process changes the group ID to the one you specify.
RW_REEVALUATE _VF	UniData evaluates indexed virtual fields on the publishing server before writing the results to the database, then transfers these values to the subscribers along with the record. You can choose to use these values directly from the publisher, or have the replication writer processes reevaluate these values. The following are valid values: ▪ 0 ▪ 1 By default, the value is 1. The replication writer process reevaluates the virtual field on the subscribing server. If you want to use the values without reevaluating them, set this parameter to 0. **Note:** Most administrators set this parameter to 0 so they are certain that the index values stored on the subscriber match those on the publisher. Additionally, the performance of replication is better, as the replication writer processes don't have to perform the work required to calculate the values of the virtual field formulas.
RW_SKIP _TRIGGER	If this value is set to 1, the default, the replication writer process ignores a trigger, even if a trigger exists on the subscribing file. If you do not want to ignore the trigger, set this value to 0. **Note:** Setting this parameter to 0 might result in unwanted updates on the subscribing server, such as attempting to update a read-only file.
RW_IGNORE _ERROR	By default, when a replication writer process encounters inconsistency in a replication log, it writes an error message to the `rm.errlog` and `rm.log` file and continues processing. The following are valid values: ▪ 0 ▪ 1 By default, the value is 0. At the default setting, UniData will suspend replication if it finds inconsistency in a replication log.

Parameter	Description
RW_QUEUESZ	Defines the size of the intelligent write queue. The intelligent write queue is used to optimize update locks and record updates on the subscriber when repeated updates of the same record are detected.

Replication configuration (repconfig) file example

The following example illustrates a `repconfig` file for an account level replication of the INVENTORY account. The location of the INVENTORY account is defined in the `UD.ACCOUNT` file or in the `repsys` file. The system primary-production is the publishing server that is replicating the account to production-standby as an immediate standby subscribing server and to production-reporting as an immediate subscribing server.

In most installations, the `repconfig` file will be identical on all three systems although this is not a requirement of replication and the configuration file can be tailored on each system if required.

```
# Definition of group inventory_acct_g0
GROUP=inventory_acct_g0
LEVEL=ACCOUNT
ACCOUNT=INVENTORY
# Definition of replication objects
EXCLUDED_FILE=PORTWORK...

# Definition of replication distributions
DISTRIBUTION=P:primary-production
DISTRIBUTION=IB:production-standby
DISTRIBUTION=I:production-reporting

#Define failover distribution
RFSFAILOVER=production-standby

# Definition of tuneable parameters
N_LOGINFO=15360
REP_BUFSZ=15360000
N_REPWRITER=4
RW_UID=
RW_GID=
RW_QUEUESZ=0
RW_IGNORE_ERROR=1
RW_REEVALUATE_VF=0
RW_SKIP_TRIGGER=1
LARGE_RECSZ=64
REP_PACING=5
RESERVED_FILE_SPACE=500
# End of definition of group udreptest_acct_g0
```

Example default account level replication file

The `repacct.def` file, located in the `/usr/ud82/include` directory on UNIX or the `UDTHOME\include` directory, includes a default list of files to include in or exclude from each account-level replication group. The `repacct.def` file can also be modified to include information defined by the user.

The following is an example of the `repacct.def` file:

```
# Definition of default excluded objects for account replication
EXCLUDED_FILE=&report&
EXCLUDED_FILE=_REPORT_
EXCLUDED_FILE=_SCREEN_
EXCLUDED_FILE=_ENCINFO_
```

```
EXCLUDED_FILE=_DEBUG_
EXCLUDED_FILE=_PH_

EXCLUDED_FILE=ENGLISH.MSG
EXCLUDED_FILE=ERRMSG
EXCLUDED_FILE=HELP.FILE
EXCLUDED_FILE=CTLGTB
EXCLUDED_FILE=UD.ACCOUNT
EXCLUDED_FILE=UD_SQLTABLES
EXCLUDED_FILE=...AUDLOG...

EXCLUDED_FILE=AE_DOC
EXCLUDED_FILE=AE_XCOMS
EXCLUDED_FILE=AE_COMS
EXCLUDED_FILE=AE_SCRATCH

EXCLUDED_FILE=EDA_DRIVER
EXCLUDED_FILE=EDA_DATASOURCE
EXCLUDED_FILE=EDA_EXCEPTION
EXCLUDED_FILE=_EDAMAP_
EXCLUDED_FILE=_EDAXMAP_

FILE=SUB_WRITEABLE DATA VOC
FILE=SUB_WRITEABLE &MAP&
FILE=SUB_WRITEABLE _MAP_
FILE=SUB_WRITEABLE _XML_
FILE=SUB_WRITEABLE _KEYSTORE_
FILE=SUB_WRITEABLE SAVEDLISTS
FILE=SUB_WRITEABLE MENUFILE
FILE=SUB_WRITEABLE __V__VIEW
FILE=SUB_WRITEABLE privilege
```

Server account file

The server account (`UD.ACCOUNT`) file is a hashed file that is in the `$UDTHOME/sys` directory and is created by default during the initial install. It contains records that map account names to file system paths. It is maintained by the XAdmin client tool in the **Accounts** menu. Starting with UniData version 7.1, it is also updated by the `newacct` command.

The `UD.ACCOUNT` file is required by XAdmin if you use server account table to set up or administer Replication.

The server account table contains records that map account names to file system paths. The XAdmin client maintains this table, which is displayed in the **Accounts** menu and stored in the `UD.ACCOUNT` file in the `$UDTHOME/sys` directory.

Configuring login credentials

DHCP and authorization

If a remote system in a Replication environment is assigned a dynamic IP address, the DHCP should be set to 1 under that system definition in repsys. This allows the remote system to update the current IP address when a sync request is sent.

A typical usage would be when the publishing system is a statically IP-addressed UNIX machine and the subscribing system is a dynamically IP-addressed Windows machine. The DHCP needs to be set in the publisher's repsys, under the definition of the subscribing system. The initial sync request must be issued by the subscriber to update the current IP address in the publisher's shared memory.

Obviously, this raises a security issue, since any system can claim it is the subscriber and can replicate data from the publisher. To prevent an unauthorized system attempting to make a connection to the publisher, a DBA can set AUTHORIZATION in repsys, under the subscribing server definition, to ask the subscriber to provide login name and password for the initial sync request. On the subscribing server, a DBA can set up the login name and password of the publisher by using either XAdmin or the `ud_repadmin setconnect` command. The login name and password are then encrypted and saved in a replication system file. Any sync request issued by the subscriber is sent to the publishing system with the saved login name and password pair.

Even though AUTHORIZATION was implemented for DHCP, it can be used to define a static IP addressed subscribing system as well, if a DBA wants to ensure login checking for the request sent from the subscriber.

Configuring replication

Complete the following steps to complete configuring U2 Data Replication.

1. On the publishing server, open the `repsys` file, and define the publishing and subscribing servers using XAdmin or a text editor.
2. On the publishing server, start the database and replication.
   a. Set REP_FLAG=1 in the file `udtconfig`.
   b. Run `$UDTBIN/startud`
3. Confirm that replication processes started.

   Note: Since replication setup has not been completed on the other sever, the sync command will fail.

4. Review the `UDTBIN rw.errlog` file for replication object-loading errors.
   a. In the VOC files of the replicated accounts on both the subscribing and publishing server, correct errors that involve invalid file pointers.
5. Copy the `repsys`, `repconfig`, and `repacct.def` files to the directory where UniData was installed on the subscribing server. If the subscribing server uses different paths to the accounts, edit the ACCOUNT parameters in the `repconfig` file on the subscribing server.

   Note: Create a master version of the `repsys` and `repconfig` files on one server. Then copy the files to the other servers and modify the file paths and file lists, as necessary. This approach minimizes possible mismatches between the files when you first start to use U2 Data Replication.

6. Confirm that the subscribing server has the same set of accounts that the publishing server has.
7. On the subscribing server, start the database and replication.
   a. Set REP_FLAG=1 in the file `udtconfig`.
   b. Run `$UDTBIN/startud`
8. If the AUTORESUME parameter in the `repsys` file is set to 0, you must use XAdmin or following command to establish the replication connection between the publishing and subscribing servers:

   ```
   $UDTBIN/ud_repadmin sync
   ```

9. Confirm that all required processes started on the publishing and subscribing servers.
10. Update several files in each replicated account on the publishing server. Then confirm that the subscribing server received and replicated the updates.

Essential tuning

Before going live with replication, you should review your configuration files and implement some changes that will allow replication to perform well. Some of these changes are based on guesses and your knowledge of how heavily certain files are updated by your application. After you have gone live, you will be able to monitor replication and perform additional tuning.

The primary way to tune replication for performance is to split up the load between different replication groups.

However, here are some initial configuration changes that you can make:

1. Determine which files in each account don't need to be replicated.
    a. There is no need to do incur the overhead of processing unneeded replication logs. In addition to your application work files, some standard UniData files you may want to consider include: ENGLISH.MSG, HELP.FILE, AE_COMS, AE_DOC, _MAP_, ERRMSG
    b. Add EXCLUDED_FILE phrases to each account-level group definition in the `repconfig` file. If the files occur in every account consider adding the EXCLUDED_FILE phrases to the `repacct.def` file as it will then be implemented for all account level groups.
2. Determine if your application requires any files to be writeable on the subscriber.
    a. By default, replicated UniData files on a subscribing server cannot be written to. Even if you just want to use the subscribing server as a report server, many applications update files when launching user menus. If you want to replicate such files (instead of exclude them), you can define them as writeable on the subscribing server. While this is not a performance issue, it may be required to allow access to replicated files on a subscriber from within your application.
    b. Add FILE phrases for those files to each account-level group definition in the `repconfig` file. Include the SUB_WRITEABLE keyword in the FILE phrase.
        a. If you have a common file name in multiple account level groups you should consider adding the phrase to the `repacct.def` file. This will avoid you having to define the phrase multiple times for each account group that requires it.
3. Review the total number of replication objects loaded for each replication group.
    a. If the number of objects in a single group exceeds the NFILES setting in the `udtconfig` file, you should consider creating one or more file-level replication groups for that account. The goal is that no groups will have more than NFILES objects. The purpose of this configuration change is to avoid the overhead of file closing and reopening by the replication writer processes on the subscribing sever.
    b. With the database started and replication active, review the number of objects in each replication group using either XAdmin or the shell-level `uvreptool` utility. For more information, see Reviewing the number of objects in replication groups using XAdmin, on page 71 and Reviewing the number of objects in a replication group using the reptool utility, on page 72.
    c. Create one or more file-level replication groups to divide up the replication objects that were loaded by default in the account-level group.
        a. Use XAdmin or a text editor. The `repconfig` file is in the directory where UniData was installed.
        b. Choose replication group names that are meaningful.
        c. We suggest including "file" in the group name to differentiate it from account-level replication groups. For instance, in the INVENTORY file-level group example we use in this guide, we chose inventory_file_g1
        d. As in the example noted, we suggest including a group number in the group name. Numbering should start with 0 and reflect the order in which the replication group exists in the `repconfig` file. This can help you identify the replication daemons that

are associated with each replication group when you review active processes on your server.

e. In the `repconfig` file, insert a FILE parameter for each file to include in the new replication group.

4. For each account, distribute the frequently updated files among multiple file-level replication groups.
   a. As a database or system administrator, you probably have a good idea which files on your system are the most heavily updated – either with a high volume of updates or with very large records. Distribute some of these hot files into separate replication groups. When your system is live, you will be able to monitor actual update activity to get a better understanding of which files are the really hot files.
   b. If you already created new file-level groups based on the NFILES review above, create FILE phrases in different groups for the hot files.
5. Restart UniData and replication to implement the configuration tuning changes.
   a. Stop UniData on the publishing and subscribing server.
      - Ask users to exit, stop phantom processes, etc.
      - Run `$UDTBIN/stopud`

      Note: If you run `stopud -f` and have RFS enabled, next, run `startud` to let the database recover from the forced shutdown. Then run `stopud`.

   b. Clear the replication log directories on both the publishing and subscribing servers.
      a. As we are still just setting things up, the contents of the data files on the subscriber do not need to match the publisher. Clearing the replication logs could potentially discard pending updates.
      b. Remove all files and directories from the directory defined in `udtconfig` REP_LOG_PATH on both servers.
   c. Start the UniData daemons on both servers.
      a. Run `$UDTBIN/startud`.
   d. If necessary, issue a sync command (on either server) to establish the replication connection between your publishing and subscribing servers. This is necessary only if the `repsys` parameter AUTORESUME is 0.
      - Use XAdmin or issue the shell-level command: `$UDTBIN/ud_repadmin sync`
   e. Confirm all appropriate daemons have launched on the publishing and subscribing servers.
   f. Update records in replicated files on the publishing server and confirm that the updates arrived at the subscribing server and were applied to the data files.
   g. If replication is not running as expected, review this procedure again.

Reviewing the number of objects in replication groups using XAdmin

Use XAdmin to review the number of objects in each replication group.

1. In XAdmin, from the U2 Resource pane, select the subscriber server. From the Admin Tasks view, expand **Replication** then double-click **Configuration**.
2. In the Replication pane, click the **Diagnosis Utility** tab.
3. Next, select the **Groups** tab and select a replication group.
4. Click **Objects**.

   The number of object for the replication groups selected displays.

Reviewing the number of objects in a replication group using the reptool utility

You can also review the number of objects in each replication group using the shell-level `reptool` utility.

1. Select **2: Replication Groups** and select a replication group to review.
2. Select **2: Objects in Group**.
3. From the end of the display, find and sum:

   Total Static Objects in Group = 102

   Total Dynamic Objects in Group = 96
4. Enter `0` to return to the Replication Groups prompt and check another group, or enter `0` again to exit the utility.

Recommended configuration and tuning

The following are some recommended tuning and configuration suggestions.

- Consider setting up floating IP addresses for your replication servers.

  If you are using replication to provide a failover server in the event of problems with the production server, you should set up machine names with a floating IP address. This allows client connections to easily switch from the publishing server to the subscribing server without having to change IP connection information on the client PC.
- Create, test, and implement an EXCEPTION_ACTION script.

  We recommend that you create and implement an EXCEPTION_ACTION script. This is enabled by defining the full path to the script in this optional parameter in the `repsys` file. If replication is suspended unexpectedly, the script will run automatically. Customers typically create a script that notifies the system administrator in some manner, such as email. Optionally, after a period of time, the script could attempt to run a `sync` command to reconnect the publishing server to the subscribing server.

  For sample scripts, see Sample EXCEPTION_ACTION script for UNIX, on page 108 and Sample EXCEPTION_ACTION script for Windows, on page 115.

  Place the script on the subscribing server only. Update the `repsys` file on that server to include the EXCEPTION_ACTION parameter for the publishing server definition.
- Minimize cross-group transactions if your application uses transaction processing semantics.

  If your application uses transaction processing, minimize the number of cross-group transactions that are generated. A cross-group transaction occurs when updates within one transaction are applied to files in two or more replication groups. To improve performance, group files that are involved in typical transactions in a single replication group, or reduce the number of groups involved in a transaction.

Going live

You just need to schedule time to refresh the data files on the subscribing system and re-initialize the replication logs. The downtime on the publishing system needs to be as long as it takes to create a clean backup of your files. If you have your disks mirrored and are able to create a backup from a split-off mirror, your downtime will be minimal. You need only interrupt the users on your publishing system long enough to stop UniData, clear the replication logs, and split off a mirror.

For more information, see Refreshing the subscriber with a backup of the publisher (publisher suspended), on page 192.

After you enable replication in your production environment, review the database update activity that each replication group handles. Then, if necessary, move files to balance the load across multiple replication groups.

Chapter 3: ud_repadmin tool

The `ud_repadmin` tool is an interface between the database administrator and the replication manager daemon. This tool sends commands to the replication manager and receives feedback. Issue the `ud_repadmin` command from the operating system-level.

Syntax

`ud_repadmin` *`command [options] [targets]`*

The following table describes each parameter of the syntax.

Parameter	Description
command	The command to send to the replication manager. Each command is documented later in this chapter.
options	The options you want to send with the command. One command can have multiple options, separated by a space. Details of each option are documented with the command later in this chapter.
targets	See ud_repadmin targets, on page 74.

ud_repadmin targets

A *target* of the `ud_repadmin` tool is a replication, a replication group, or a distribution of a replication group. A replication is all data replicated from a remote system to the local system, or from a local system to a remote system. A target definition of ALL represents all replications on the system. One `ud_repadmin` command can have multiple targets.

Syntax

Targets are separated by commas, and have the following syntax:

`ALL` | *`{replication | group} [,replication | group...]`*

The following table describes each parameter of the *targets* syntax.

Parameter	Value		
replication	The replication to or from the system you specify.		
	PUBLISHING *system_name*	The replication from the system you specify to the local system.	
	SUBSCRIBING *system_name*	The replication from the local system to the system you specify.	
	system_name	Replication to and from the system you specify.	
group	The group that is the target of the command.		
	GROUP *group_name*	The name of the replication group.	
		DISTRIB *system_name*	The distribution of the group you specify.
		none	All distributions of the group.

If the command you issue through the `ud_repadmin` tool is successful, `ud_repadmin` returns 0. If the command is not successful, `uv_repadmin` returns an error code.

You can specify the -verbose option with any `ud_repadmin` command you issue to provide more detailed information about the results or errors associated with the command.

suspend command

Use the `suspend` command before shutting down the publishing or subscribing server.

In suspend mode, communication between the publisher and the subscriber is interrupted. The publishing server saves the replication log files to the replication logs rather than transferring them to subscribing servers. The subscribing server and all replication writer processes finish updating existing logs in the replication buffer and then stop.

Syntax

suspend *targets*

If you specify *targets* as ALL, UniData suspends all replications on the system. If you specify a replication, UniData suspends the replication from or to the system you specify. If you specify target as a replication group, UniData suspends all distributions of the group you specify.

Note: After the database restarts, replication remains suspended. To automatically resume live replication after a suspension, set the AUTORESUME parameter in the server definition file.

Suspending replication does not affect deferred replication.

When to issue the suspend command

Suspend replication before you perform the following task:

- Stopping the publishing or subscribing server
- Clearing the replication logs
- Performing database maintenance tasks, such as creating a backup, on a subscribing server

You should not issue a `suspend` command before attempting a failover or reconfig (these are described later).

Example

The following is an example of successfully suspending replication.

```
# $UDTBIN/ud_reapadmin suspend -verbose ALL
Command succeeded!
Publishing Group inventory_acct_g0 is REP_SUSPENDED:DBA Ordered.
Publishing Group inventory_file_g1 is REP_SUSPENDED:DBA Ordered.
Publishing Group payroll_acct_g2 is REP_SUSPENDED:DBA Ordered.
Publishing Group payroll_file_g3 is REP_SUSPENDED:DBA Ordered.
#
```

To make sure that the command was processed successfully, you should examine the `rm.log` file on both systems.

Note: The `tail` command is used in the next two examples to show only the results in the `rm.log` file that are applicable to the command that has just been entered. Blank lines have also been inserted into the output to visually group the messages. `tail` is a UNIX shell command.

Publishing server

```
# tail -9 $UDTBIN/rm.log
Thu Jun 10 13:08:21 2010: Request DBA Suspend to ALL received.

DBA Ordered Suspended GROUP inventory_acct_g0, SYSTEM standby successfully
DBA Ordered Suspended GROUP inventory_file_g1, SYSTEM standby successfully
DBA Ordered Suspended GROUP payroll_acct_g2, SYSTEM standby successfully
DBA Ordered Suspended GROUP payroll_file_g3, SYSTEM standby successfully

DBA Ordered Suspended Publishing Group inventory_acct_g0 successfully
DBA Ordered Suspended Publishing Group inventory_file_g1 successfully
DBA Ordered Suspended Publishing Group payroll_acct_g2 successfully
DBA Ordered Suspended Publishing Group payroll_file_g3 successfully
#
```

Subscribing server

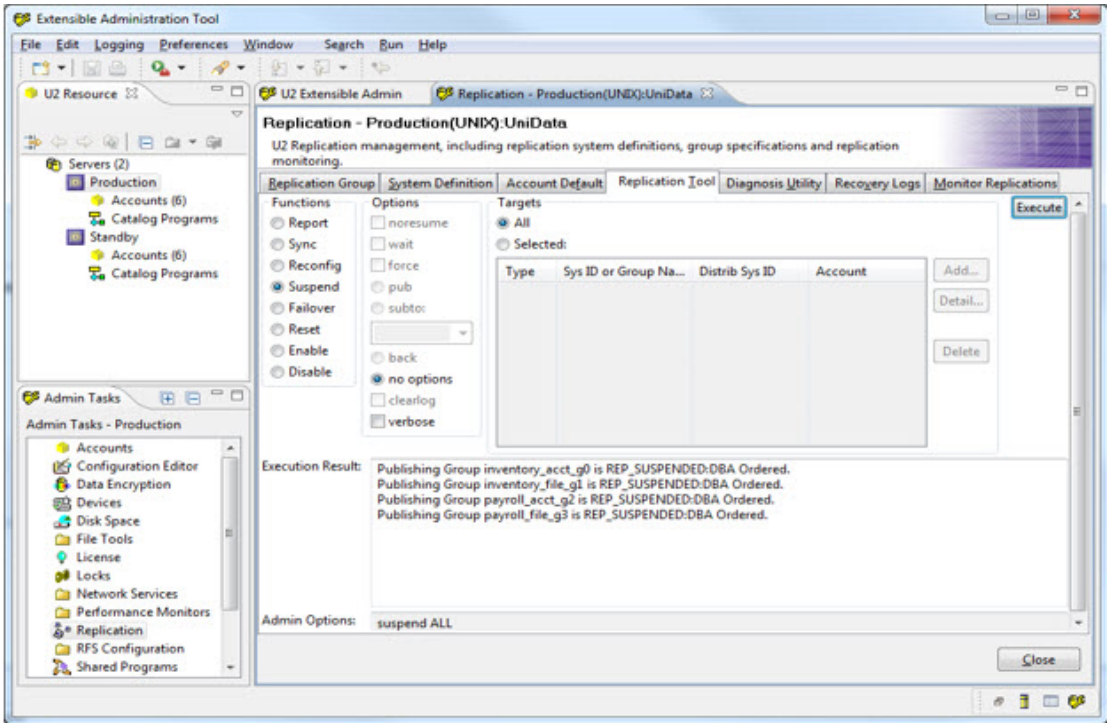
```
# tail -8 $UDHOME/rm.log
Tue Jul 19 03:19:18 2011 Subscribing Group inventory_acct_g0 started
suspending(Publisher Requested).
Tue Jul 19 03:19:18 2011 Subscribing Group payroll_acct_g2 started
suspending(Publisher Requested).
Tue Jul 19 03:19:18 2011 Subscribing Group inventory_file_g1 started
suspending(Publisher Requested).
Tue Jul 19 03:19:18 2011 Subscribing Group payroll_file_g3 started
suspending(Publisher Requested).
Tue Jul 19 03:19:18 2011 Subscribing group inventory_acct_g0 stopped
successfully.
Tue Jul 19 03:19:18 2011 Subscribing group payroll_acct_g2 stopped
successfully.
Tue Jul 19 03:19:18 2011 Subscribing group inventory_file_g1 stopped
successfully.
Tue Jul 19 03:19:18 2011 Subscribing group payroll_file_g3 stopped
successfully.
#
```

Look closely at the contents of the `rm.log` file. On the publisher, you see the messages that the suspend command was successfully issued and processed. You can also see that the contents of `rm.log` file on the subscriber show that the suspend command was successfully received and processed there as well.

Also note that the contents of `rm.log` on the subscriber show that the suspend command was “Publisher Requested”. If the suspend command had been issued from the subscriber, the contents of the `rm.log` file on the publisher would state that the suspend command was requested from the subscriber.

Note: When examining the contents of the `rm.log` file on your system, if the sequence or subsequent result of the events seems not to reflect those shown in the example, this may be an indication that you need to take steps to correct the error or to let the commands complete before continuing.

The following figure, from UniData, displays the results of the `suspend` command, executed from the **XAdmin** → **U2 Replication** → **Configuration** → **Replication Tool**.



Also note that the subscriber `udrw` processes exit when replication is suspended, as shown in the `showud` output below.

```
# $UDTBIN/showud
      USER     PID      TIME COMMAND
      root  536696      0:00 /disk1/ud72/bin/aimglog 0 25883
      root  598052      0:00 /disk1/ud72/bin/aimglog 1 25883
      root  249878      0:00 /disk1/ud72/bin/bimglog 2 25883
      root  671848      0:00 /disk1/ud72/bin/bimglog 3 25883
      root  663718      0:00 /disk1/ud72/bin/cleanupd -m 10 -t 20
      root  368694      0:00 /disk1/ud72/bin/cm 25883
      root  606392      0:00 /disk1/ud72/bin/repmanager
      root  319682      0:00 /disk1/ud72/bin/sbcs -r
      root  774274      0:00 /disk1/ud72/bin/sm 60 2617
      root  675934      0:00 /disk1/ud72/bin/smm -t 60
      root  377046      0:00 /disk1/ud72/unishared/unirpc/unirpcd
```

sync command

The `sync` command synchronizes subscribing servers to their publishing servers. The publishing server establishes a connection to the subscribing server and invokes the subscribing process. UniData reads and transfers replication logs from the publishing server to the subscribing server. The subscribing server then applies the updates to the database.

Syntax

sync [-noresume | -wait | -force] *target*

If you specify *targets* as ALL, UniData synchronizes replications on all systems. If you specify *targets* as a replication system, UniData synchronizes replications from and/or to the replication system you specify. If you specify *target* as a group, UniData synchronizes all replications of the group you specify.

The speed of the synchronization process depends on how many pending logs UniData must synchronize, and the speed of the network connection. The `sync` command normally returns when all targets are in the synchronization mode.

By default, the `sync` command resumes a live replication. The following table describes the options available with the `sync` command.

Option	Description
-noresume	UniData suspends a live replication when it finishes synchronizing the replication systems. When the first synchronization is complete, U2 Data Replication does not look for more updates.
-wait	If you specify this option, the `SYNC` command does not return until all synchronizations you specify succeed or fail.
-force	The replication writer processes ignore the errors encountered during the synchronization process, if any. If you do not specify this option, the synchronization process fails if an error is encountered.

Note: After UniData starts, the replications to all systems are suspended. You can choose to set the AUTORESUME phrase in the `repsys` file to automatically resume live replication.

The `sync` command may fail if the network connection fails, or if the distribution of a replication conflicts between the publishing server and the subscribing server. UniData returns an error code to indicate the failure. You can use the `report` command to check the details of the error.

Note: If you issue the `sync` command when a live replication is running, the command has no effect.

When to issue the sync command

You should issue the `sync` command in the following circumstances:

- After UniData starts on the subscribing server if you do not set the AUTORESUME phrase in the `repsys` file.
- After you suspend live replication from or to a system and you want to resume replication.
- After failing over or reconfiguring a replication if you do not set the AUTORESUME phrase in the `repsys` file.
- After UniData starts on the publishing server while some of the subscribing servers are running and you do not set the AUTORESUME phrase in the `repsys` file.

Example

The following is an example of how to use the sync argument. In this example, we use the `sync` command to resume replication after it was suspended.

First, check to two servers to see if replication is suspended.

Publishing server

```
# $UDTBIN/ud_repadmin report -detail -verbose
reporting ALL...
Replication publishing to standby is REP_SUSPENDED:DBA Ordered.
GROUP inventory_acct_g0, SYSTEM standby is REP_SUSPENDED:DBA Ordered.
GROUP inventory_file_g1, SYSTEM standby is REP_SUSPENDED:DBA Ordered.
GROUP payroll_acct_g2, SYSTEM standby is REP_SUSPENDED:DBA Ordered.
GROUP payroll_file_g3, SYSTEM standby is REP_SUSPENDED:DBA Ordered.
```

```
#
```

Subscribing server

```
# $UDTBIN/ud_repadmin report -detail -verbose
reporting ALL...
Replication subscribing from primary is SUB_STOP:Publisher Requested.
GROUP inventory_acct_g0, SYSTEM primary is SUB_STOP:Publisher
Requested.
GROUP inventory_file_g1, SYSTEM primary is SUB_STOP:Publisher
Requested.
GROUP payroll_acct_g2, SYSTEM primary is SUB_STOP:Publisher
Requested.
GROUP payroll_file_g3, SYSTEM primary is SUB_STOP:Publisher
Requested.
#
```

Once the status of replication has been confirmed on both the publisher and subscriber, use the sync argument to resume the replication. In the following example, the `ud_repadmin sync` command from the subscribing server.

```
# $UDTBIN/ud_repadmin sync -verbose ALL
Command succeeded!
Replication subscribing from primary is REP_SYNCING:Sync Succeeded.
```

The output from the `ud_repadmin` command indicates that the `sync` command has worked. Next, examine the contents of the `rm.log` file to confirm the results.

```
# tail -6 $UDTBIN/rm.log
Thu Jun 10 14:23:15 2010: Request Sync to ALL received.

DBA Ordered Synced Replication subscribing from primary successfully
Thu Jun 10 14:23:16 2010 Subscribing Group inventory_file_g1 synchronized
successfully to publisher and resumed running.
Thu Jun 10 14:23:16 2010 Subscribing Group inventory_acct_g0 synchronized
successfully to publisher and resumed running.
Thu Jun 10 14:23:16 2010 Subscribing Group payroll_acct_g2 synchronized
successfully to publisher and resumed running.
Thu Jun 10 14:23:16 2010 Subscribing Group payroll_file_g3 synchronized
successfully to publisher and resumed running.
#
```

The contents of the `rm.log` file confirms that the subscriber has received and successfully processed the command. Next, examine the contents of the `rm.log` file on the publisher.

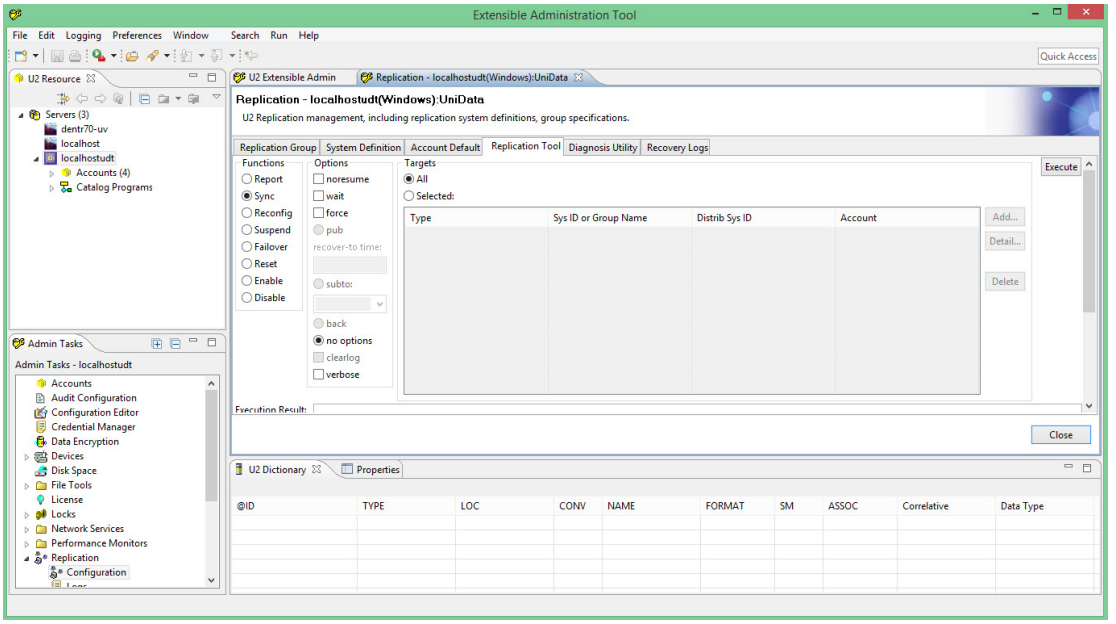
```
# tail -10 $UDTBIN/rm.log
Thu Jun 10 14:23:15 2010: Remote request Sync to Replication publishing to
standby received.

Thu Jun 10 14:23:15 2010 Group inventory_acct_g0 Replication to standby
synchronized successfully and resumed running.
Remote System Requested Synced GROUP inventory_acct_g0, SYSTEM standby
successfully
Thu Jun 10 14:23:15 2010 Group payroll_acct_g2 Replication to standby
synchronized successfully and resumed running.
Remote System Requested Synced GROUP payroll_acct_g2, SYSTEM standby
successfully
Thu Jun 10 14:23:15 2010 Group inventory_file_g1 Replication to standby
synchronized successfully and resumed running.
Remote System Requested Synced GROUP inventory_file_g1, SYSTEM standby successfully
Thu Jun 10 14:23:15 2010 Group payroll_file_g3 Replication to standby
```

```
synchronized successfully and resumed running.
Remote System Requested Synced GROUP payroll_file_g3, SYSTEM standby
successfully

Remote System Requested Synced Replication publishing to standby successfully
#
```

The contents of the publisher's `rm.log` file confirm the receipt of the sync command from the subscribing server (noted as a "Remote System Requested" and that the four groups were successfully synchronized and replication resumed running.

> **Note:** When examining the contents of the `rm.log` file on your system, if the sequence or subsequent result of the events seems not to reflect those shown in the example, this may be an indication that you need to take steps to correct the error or to let the commands complete before continuing.

Here are the results of the sync command, using UniData, executed from the **XAdmin** → **U2 Replication** → **Configuration** → **Replication Tool**:



failover command

The `failover` command changes the replication direction on a local system, either from the local system to the publishing server or subscribing server, or changes the subscribing source distribution.

Syntax

`failover` `[-verbose] [-pub {`*`failover recover-to time`*`} | -subto`
`system_name` `| -back] {`*`target ...`*`}`

The following table describes the `failover` options.

Option	Description
-verbose	Displays messages to the screen.

Option	Description
-pub	The `failover` command changes the replication to a publishing server on the local system. The local distribution of the replication must be a standby backup system.
failover recover-to time	Specified timestamp to which the failover process will recover. Used with delayed standby replication, failover will apply its logs with the replication log generated before this timestamp, but abandon the logs generated after it. The format of this timestamp is: *mm:dd:HH:MM{:SS{:yyyy}}{GMT}* *mm*: Month *dd*: Day *HH*: Hour, using a 24-hour clock *MM*: Minute number *SS*: Second number. Default value is 00. *yyy4*: 4-digit year. Default is current year. *GMT*: GMT time zone if specified; otherwise, local time
-subto	If you specify the -subto option, UniData either changes a replication from a publishing server to a subscribing server, or changes the subscribing source to the system you specify with *system_name*.
-back	UniData changes the replication back to the original configuration setting.
target	A replication, a replication group, or a distribution of a replication group.

Example

Suppose that ServerA crashes, and you want to failover to ServerB. On Server B, execute the following command:

```
ud_repadmin failover -pub
```

On ServerC, change the subscribing source from ServerA to ServerB with the following command:

```
ud_repadmin failover -subto ServerB
```

On ServerA, you should change the replication from a publishing server to a subscribing server on ServerB using the following command:

```
ud_repadmin failover -subto ServerB
```

If you want to revert to the original replication directions, execute the following command on ServerA and on ServerC:

```
ud_repadmin failover -back ServerB
```

In this case, ServerB represents all replication groups replicating data from ServerB to a local system (ServerA or ServerC).

On ServerB, you should execute the following command to change all groups currently publishing to ServerA to their original distribution setting:

```
ud_repadmin failover -back ServerA
```

Note: The `failover` command does not change the distribution definition in the `repconfig` file. Instead, it records the failover status in an internal group status file for all replication groups involved. Even after you stop and restart UniData, U2 Data Replication maintains this failover status. Issue another `failover` command, or issue the `reconfig` command to change the failover status.

After UniData starts, the replications to all systems are suspended. You can choose to set the AUTORESUME phrase in the `repsys` file to automatically resume live replication.

reconfig command

Use the `reconfig` command to reconfigure replication configuration while the database is running.

Reconfigure replication when the following situations occur:

- The `repsys` file, `repconfig` file, or both files changed, and you want to update the configuration and restart replication while the database is running.
- The inode of a file in a replication group changed, and you do not want to restart the database.

Note: The UniData `RESIZE` or `memresize` command changes the inode for a file. However, because the database automatically adjusts the inodes that are stored in memory for the file, you do not need to run the `reconfig` command.

Syntax

`reconfig`

After you issue the `reconfig` command, the replication manager completes the following tasks:

1. Issues the `dbpause` command to block any writes to the database
2. Stops all replications
3. Reloads the `repconfig` and `repsys` files
4. Reallocates system resources
5. Restarts each replication group on the server
6. Issues the `dbresume` command to continue processing on the database

Note: After UniData resumes processing, the replications to all systems are suspended. You can choose to set the AUTORESUME phrase in the `repsys` file to automatically resume live replication.

Example

The following illustrates how to issue the `repconfig` command on the publishing server.

```
# $UDBIN/ud_repadmin reconfig -verbose
reconfiging ALL...
Command succeeded!
Publishing Group inventory_acct_g0 is PUB_RUNNING:.
Publishing Group inventory_file_g1 is PUB_RUNNING:.
Publishing Group payroll_acct_g2 is PUB_RUNNING:.
Publishing Group payroll_file_g3 is PUB_RUNNING:.
#
```

Next, examine the contents of the `rm.log` to see what happened with the `repconfig`.

```
#tail -44 $UDTBIN/rm.log
Thu Jun 10 14:29:01 2010: Request DBA Reconfigure to ALL received.

RM: Force RFS group commit.
RM: Pause database activity.
CheckPoint time before ForceCP: Thu Jun 10 14:26:18 2010
.CheckPoint time after ForceCP:  Thu Jun 10 14:29:03 2010
.CP has been forced successfully.
DBpause successful.
RM: Force RFS group commit.

Thu Jun 10 14:29:04 2010 Group inventory_file_g1 Replication to standby is
suspended (DBA Ordered).
Stoped Publishing Group inventory_file_g1 successfully.
Thu Jun 10 14:29:05 2010 Group payroll_file_g3 Replication to standby is
suspended (DBA Ordered).
Thu Jun 10 14:29:05 2010 Group inventory_acct_g0 Replication to standby is
suspended (DBA Ordered).
Thu Jun 10 14:29:05 2010 Group payroll_acct_g2 Replication to standby is
suspended (DBA Ordered).

Thu Jun 10 14:29:05 2010 Publishing group inventory_file_g1 stopped
successfully.
Thu Jun 10 14:29:05 2010 Publishing group inventory_acct_g0 stopped
successfully.
Thu Jun 10 14:29:05 2010 Publishing group payroll_file_g3 stopped successfully.
Thu Jun 10 14:29:05 2010 Publishing group payroll_acct_g2 stopped successfully.
Stoped Publishing Group payroll_file_g3 successfully.
Stoped Publishing Group inventory_acct_g0 successfully.
Stoped Publishing Group payroll_acct_g2 successfully.

RW(696476) is started to load rep objects' inum/dnum for account
/disk1/udt72/inventory
RW(471174) is started to load rep objects' inum/dnum for account
/disk1/udt72/payroll
RM: Reload Configuration successfully!

udpub for group inventory_acct_g0(0) is started, pid = 594040.
udpub for group inventory_file_g1(1) is started, pid = 618612.
udpub for group payroll_acct_g2(2) is started, pid = 798726.
udpub for group payroll_file_g3(3) is started, pid = 774338.
Started Publishing Group inventory_acct_g0 successfully!
Started Publishing Group inventory_file_g1 successfully!
Started Publishing Group payroll_acct_g2 successfully!
Started Publishing Group payroll_file_g3 successfully!

RM: Resume database activity.
DBresume successful.

Reconfigured Replications successfully


Thu Jun 10 14:29:05 2010 Group inventory_file_g1 Replication to standby
synchronized successfully and resumed running.
Auto Resuming Synced GROUP inventory_file_g1, SYSTEM standby successfully
Thu Jun 10 14:29:05 2010 Group inventory_acct_g0 Replication to standby
synchronized successfully and resumed running.
Auto Resuming Synced GROUP inventory_acct_g0, SYSTEM standby successfully
Thu Jun 10 14:29:06 2010 Group payroll_acct_g2 Replication to standby
synchronized successfully and resumed running.
Auto Resuming Synced GROUP payroll_acct_g2, SYSTEM standby successfully
Thu Jun 10 14:29:06 2010 Group payroll_file_g3 Replication to standby
synchronized successfully and resumed running.
Auto Resuming Synced GROUP payroll_file_g3, SYSTEM standby successfully
Auto Resuming Synced Replication publishing to standby successfully
```

Pausing UniData - dbpause

Suspend replication

Reload buffer from repconfig

Restart udpub processes

Resume UniData - dbresume

Sync Replication

You can see that the contents of the `rm.log` file detail the steps that were described when a `reconfig` command is issued. The contents of the `rm.log` file also show that the command was completed successfully.

You should also examine the contents of the `rm.log` on the subscribing server to determine if the commands issued as a result of the `reconfig` were successfully received and processed.

```
# tail -12 $UDTBIN/rm.log

Thu Jun 10 14:29:04 2010 Subscribing Group inventory_file_g1 started suspending
(Publisher Requested).
Thu Jun 10 14:29:04 2010 Subscribing Group inventory_acct_g0 started suspending
(Publisher Requested).
Thu Jun 10 14:29:04 2010 Subscribing Group payroll_acct_g2 started suspending
(Publisher Requested).
Thu Jun 10 14:29:04 2010 Subscribing Group payroll_file_g3 started suspending
(Publisher Requested).
Thu Jun 10 14:29:05 2010 Subscribing group inventory_acct_g0 stopped
successfully.
Thu Jun 10 14:29:05 2010 Subscribing group inventory_file_g1 stopped
successfully.
Thu Jun 10 14:29:05 2010 Subscribing group payroll_acct_g2 stopped
successfully.
Thu Jun 10 14:29:05 2010 Subscribing group payroll_file_g3 stopped
successfully.
```

Messages resulting from the suspend issued by reconfig.

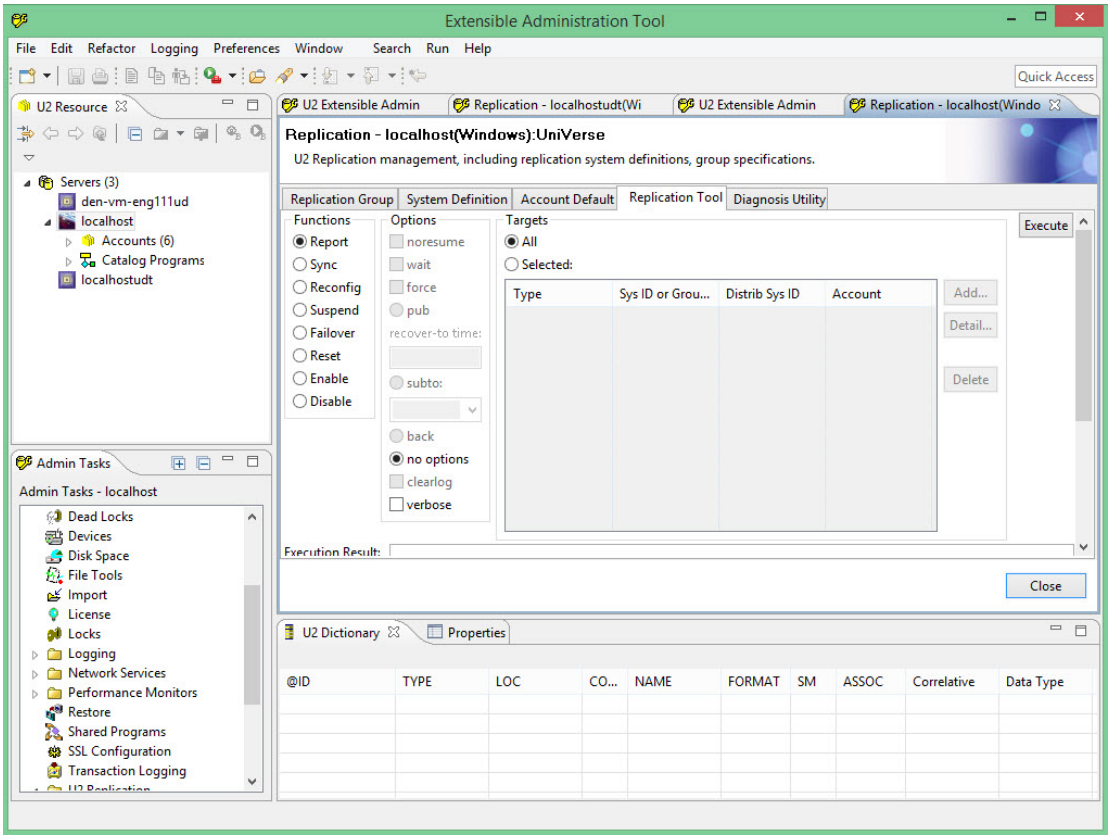
```
Thu Jun 10 14:29:06 2010 Subscribing Group inventory_file_g1 synchronized
successfully to publisher and resumed running.
Thu Jun 10 14:29:06 2010 Subscribing Group inventory_acct_g0 synchronized
successfully to publisher and resumed running.
Thu Jun 10 14:29:07 2010 Subscribing Group payroll_acct_g2 synchronized
successfully to publisher and resumed running.
Thu Jun 10 14:29:07 2010 Subscribing Group payroll_file_g3 synchronized
successfully to publisher and resumed running.
#
```

Messages resulting from the sync issued at the end of the reconfig process.

You can see the successful suspension of replication, which is noted as “Publisher Requested.” You can then see the successful synchronization and restart on the subscriber after the `reconfig` completed on the publisher. In this example, AUTORESUME was set to 1.

Note: When examining the contents of the `rm.log` file on your system, if the sequence or subsequent result of the events seems not to reflect those shown in the example, this may be an indication that you need to take steps to correct the error or to let the commands complete before continuing.

Here are the results of the `reconfig` command, using UniData, executed from the **XAdmin** → **U2 Replication** → **Configuration** → **Replication Tool**:



reset command

Use the `reset` command to clear the replication logs in the replication log reserve file.

Syntax

`reset` `[-noresume]` *`target`*

Use the `reset` command after you copy or store database files, since the remaining replication logs are no longer useful.

Use the `reset` command with a deferred replication, or a live replication in suspension mode. If the *target* you specify is running, the `reset` command has no effect.

Unless you specify the `-noresume` option, a live replication returns to running mode from suspension mode when the `reset` command finishes processing.

Example

The following are examples of the `reset` command on the publishing server.

```
# $UDBIN/ud_repadmin reset -verbose ALL
Command succeeded!
Replication publishing to standby is REP_RUNNING:.
#
```

Examine the contents of the `rm.log` file to make sure the command was successfully received and processed.

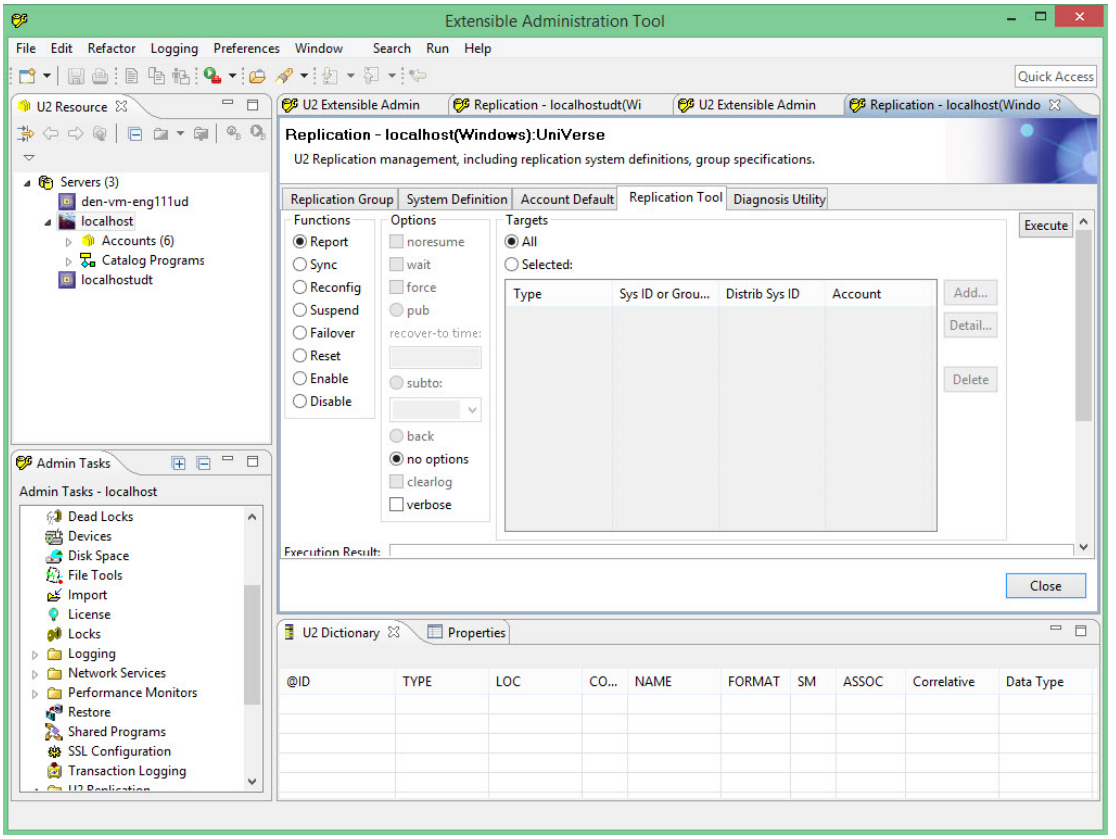
```
# tail -6 $UDHOME/rm.log
```

```
Tue Jul 19 05:15:30 2011: Request DBA Reset to ALL received.
Reset GROUP inventory_acct_g0, SYSTEM standby successfully
Reset GROUP inventory_file_g1, SYSTEM standby successfully
Reset GROUP payroll_acct_g2, SYSTEM standby successfully
Reset GROUP payroll_file_g3, SYSTEM standby successfully
Reset Replication publishing to standby successfully
#
```

The contents of the `rm.log` file confirm that the command was successful. In the `rm.log` file on the subscriber, you will not see anything relating to this command, as no action needs to be taken on the subscribing server.

Note: When examining the contents of the `rm.log` file on your system, if the sequence or subsequent result of the events seems not to reflect those shown in the example, this may be an indication that you need to take steps to correct the error or to let the commands complete before continuing.

Here are the results of the `reset` command, using UniData, executed from the **XAdmin → U2 Replication → Configuration → Replication Tool**:



report command

Use the `report` command to display the current replication status.

Syntax

report [-detail | -xml] *target*

Each line of the report represents the status of a replication. Each line has the following format:

```
[PUBLISHING | SUBSCRIBING] server_name=Status:Reason
```

The *server_name* is the name you specify in the `repsys` file.

Status is the current status of the replication. The following table lists valid statuses.

Status	Description
RUNNING	The replication is running.
SUSPEND	The replication is in suspension mode.
SUSPENDING	The replication is in the process of suspending processing.
SYNCING	The replication is synchronizing data.
RECONFIGURING	The replication is in the process of reconfiguring.
SHUTDOWN	The replication is shutting down.
STOPPED	The replication stopped normally.
EXIT	The replication exited abnormally.

If the replication status is SUSPEND, the reason for the suspension is also provided. The following table lists the reasons:

Reason	Description
STARTUP	The local server is starting or restarting.
DBA_ORDERED	The administrator suspended replication.
REMOTE_REQUEST	A remote server requested that replication be suspended.
SYS_RECONFIG	Replication is in suspension mode after a reconfiguration.
REMOTE_SYS_DOWN	The remote server shut down.
RPC_FAILURE	The connection failed and cannot be reestablished.
RPC_TIMEOUT	A sending or receiving process timed out.
SUBSCRIBER_FAILURE	The subscribing server failed to write a replication log. The detailed error message is in the `rm.errlog`.
TRANSACTION_ABORT	A transaction on the subscribing server failed. The detailed error message is in the `rm.errlog`.
FATAL_ERROR	A internal or fatal error occurred, and the server cannot recover.

Note: Local servers are the machines where the commands are running. Remote servers are all other servers involved with replication.

If the status is EXIT, the reason for the exit is also provided. The following table lists the reasons:

Reason	Description
HAS_FAILOVER	The subscribing server failed over to the publishing server.
DISTRIBUTION_MISMATCH	A mismatch exists between a distribution on the publishing server and a distribution on the subscribing server.
FATAL_ERROR	An internal or fatal error occurred, and the server cannot recover.

If you specify the -detail option, the status of each replication group is reported after the replication status. The syntax of the replication group status is:

```
group_name=status:reason
```

where *group_name* is the replication group name defined in the `repconfig` file. The *status* and *reason* are the same as those for the replication status.

The -xml option reports detailed replication status information in XML format, regardless of whether you specify -detail.

The following example illustrates a replication status report in XML format:

```
<? Xml version="1.0"?>
<REP_REPORT LOCAL_SYSTEM="Trigger_71" DATE="Thu Oct 5 16:02:46 2006">
    <REPLICATION REMOTE_SYSTEM = "Juneau_71"REPTYPE = "STANDBY
 IMMEDIATE
PUBLISHING" STATUS = "REP_RUNNING" REASON = "Sync Succeeded" >
<GROUP NAME = "reptest1" STATUS = "REP_RUNNING" REASON = "Sync
 Succeeded" />
<GROUP NAME = "reptest2" STATUS = "REP_RUNNING" REASON = "Sync
 Succeeded" />
<GROUP NAME = "reptest3" STATUS = "REP_RUNNING" REASON = "Sync
 Succeeded" />
    </REPLICATION>
    <REPLICATION REMOTE_SYSTEM = "Pinto_71" REPTYPE = "FAILOVER
 IMMEDIATE
PUBLISHING" STATUS = "REP_RUNNING" REASON = "Sync Succeeded" >
<GROUP NAME = "reptest1" STATUS = "REP_RUNNING" REASON = "Sync
 Succeeded" />
<GROUP NAME = "reptest2" STATUS = "REP_RUNNING" REASON = "Sync
 Succeeded" />
    </REPLICATION>
</REP_REPORT>
```

The following example shows the DTD for the XML report shown above:

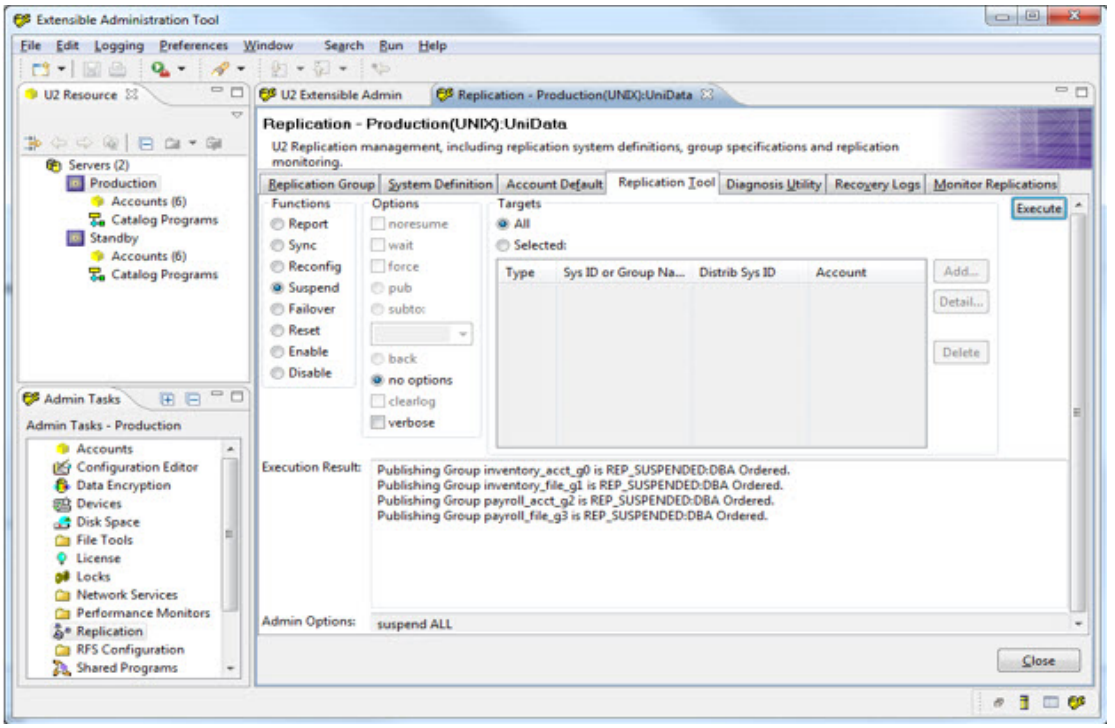
```
<!-REP_REPORT.DTD -->
<! ELEMENT REP_REPORT (REPLICATION*)>
<! ATTLIST REP_REPORT
  LOCAL_SYSTEM CDATE #REQUIRED
  DATE CDATE #REQUIRED>
<! ELEMENT REPLICATION (GROUP*)>
<! ATTLIST REPLICATION
  REMOTE_SYSTEM CDATE #REQUIRED
  REPTYPE CDATA #REQUIRED
  STATUS CDATA #IMPLIED
  REASON CDATA # IMPLIED >
<! ELEMENT GROUP EMPTY>

<! ATTLIST GROUP
  NAME CDATA #REQUIRED
  STATUS CDATA #IMPLIED
  REASON CDATA #IMPLIED>
```

Any report request sent via the `ud_repadmin` command is also logged in the `rm.log` file.

```
# $UDTBIN/ud_repadmin report -detail -verbose
reporting ALL...
Replication from/to standby is REP_RUNNING:Sync Succeeded.
GROUP inventory_acct_g0, SYSTEM standby is REP_RUNNING:Sync
Succeeded.
GROUP inventory_file_g1, SYSTEM standby is REP_RUNNING:Sync
Succeeded.
GROUP payroll_acct_g2, SYSTEM standby is REP_RUNNING:Sync Succeeded.
GROUP payroll_file_g3, SYSTEM standby is REP_RUNNING:Sync Succeeded.
#


# tail -2 $UDTBIN/rm.log
```

```
Tue Jul 19 05:28:38 2011: Request DBA Report to ALL received.
#
```

The following figure, from UniData, displays the results of the `suspend` command, executed from the **XAdmin** → **U2 Replication** → **Configuration** → **Replication Tool**.



The `report` command also sets shell-level return codes. This facilitates scripting of the command that you can use in scripts.

If you issue the `report` command on a publishing server and specify a single replication group, the status for the group is returned. The following table lists the group status codes.

```
$UDTBIN/ud_repadmin report group <group name>
```

For example:

$UDTBIN/ud_repadmin report group inventory_acct_g0

Replication status return codes

The following table describes the replication status return codes that are returned by `ud_repadmin report` command.

Return code	Code name	Description
1	PUB_STOP	The publishing group stopped normally
2	PUB_EXIT	The publishing group quit abnormally
3	PUB_SHUTDOWN	The publishing group is shutting down
4	PUB_RUNNING	The publishing group is running
5	PUB_DO_RECONFIG	The publishing group is performing a reconfiguration

Return code	Code name	Description
7	PUB_DO_FAILOVER	The publishing group is performing a failover
11	REP_RUNNING	Publishing replication is running normally
12	REP_SUSPENDED	Publishing replication is suspended
13	REP_SYNCING	Publishing replication is synchronizing
14	REP_DO_SUSPEND	Publishing replication is in the process of suspending
16	REP_EXIT	Publishing replication is exited
21	SUB_STOP	The subscribing replication group stopped normally
22	SUB_EXIT	Subscribing replication has quit abnormally
23	SUB_SHUTDOWN	Subscribing replication is shutting down
24	SUB_RUNNING	Subscribing replication is running
25	SUB_DO_RECONFIG	Subscribing replication is performing a reconfiguration
27	SUB_DO_SUSPEND	Subscribing replication is suspending
28	SUB_SYNCING	Subscribing replication is syncing
29	SUB_DO_FAILOVER	Subscribing replication is performing a failover
30	SUB_RESYNCING	Subscribing replication is resynchronizing

setconnect command

The `setconnect` command is used on the subscriber to provide login credentials to a publishing server so the publisher can confirm that the subscribing server has permissions to subscribe from it. Although the `setconnect` command was designed to be used with DHCP, it can also be used in a static IP environment to ensure that a subscribing server is allowed to subscribe.

Syntax

```
uv_repadmin setconnect replication
```

After entering this command, you are prompted for the user name and password that you want the *replication* publisher to use in validation. These credentials are used only if the AUTHORIZATION value is set to 1.

disable command

The `disable` command is part of the replication disablement feature that was added at UniData 7.3.0. For more information on this feature, please see the section in this document named: Disabling U2 Data Replication.

For more information, see Disabling U2 Data Replication, on page 106.

Syntax

```
ud_repadmin disable [-clearlog] [-verbose]
```

The –clearlog option clears the replication log files from the replication log directory.

You must be logged in as root or administrator to disable U2 Data Replication.

Example

The following example illustrates how to issue the disable command and what you should expect to see:

```
# $UDTBIN/ud_repadmin disable -clearlog -verbose
Performing disable for ALL...
Command succeeded!

Publishing Group inventory_acct_g0 is PUB_SHUTDOWN:DBA Ordered.

Publishing Group inventory_file_g1 is PUB_SHUTDOWN:DBA Ordered.

Publishing Group payroll_acct_g2 is PUB_SHUTDOWN:DBA Ordered.

Publishing Group payroll_file_g3 is PUB_SHUTDOWN:DBA Ordered.
#
```

Note: The group status files and the `reptemp.tca` files are not removed from the `replog` directory when the -clearlog option is used. Here is the content of the `replog` directory after issuing the disable command with the -clearlog option:

```
# ls -l
total 472
-rw-r-----    1 root     system       112964 Jan  8 15:39 inventory_acct_g0.gstt
-rw-r-----    1 root     system         7496 Jan  8 15:39 inventory_file_g1.gstt
-rw-r-----    1 root     system       103388 Jan  8 15:39 payroll_acct_g2.gstt
-rw-r-----    1 root     system         5648 Jan  8 15:39 payroll_file_g3.gstt
-rw-rw-rw-    1 root     system         1024 Dec 21 15:21 reptemp.tca
```

enable command

The `enable` command is part of the replication disablement feature that was added at UniData 7.3.0.

For more information on this feature, see Disabling U2 Data Replication, on page 106.

```
ud_repadmin enable [-verbose]
```

You must be logged in as root or administrator to enable U2 Data Replication.

Example

The following example illustrates how to issue the enable command and what you should expect to see:

```
# $UDTBIN/ud_repadmin enable -verbose
Performing enable for ALL...
Command succeeded!

Publishing Group inventory_acct_g0 is PUB_RUNNING:.

Publishing Group inventory_file_g1 is PUB_RUNNING:.

Publishing Group payroll_acct_g2 is PUB_RUNNING:.

Publishing Group payroll_file_g3 is PUB_RUNNING:.
```

Chapter 4: Managing U2 Data Replication through the command line

You can define publishing and subscribing groups and change their definitions directly from the command line

To administer U2 Data Replication from the command line, perform the following tasks:

- Starting replication
  The value of the REP_FLAG parameter in the `udtconfig` file controls whether replication starts. For replication to start, this parameter must be set to 1. Then replication starts and stops as part of the normal startud and stopud scripts.
- Stopping the database and replication
- Defining subscribing groups from the command line
  Define a subscribing replication group from the command line.
- Defining publishing groups from the command line
  Define a publishing replication group from the command line.
- Changing an existing replication group definition from the command line
  Reconfigure U2 Data Replication from the command line.
- Enabling and disabling U2 Data Replication through the command line
  You can enable or disable U2 Data Replication using the command line.
- Replication exception action programs
  A replication exception action program defines the action to take if replication abnormally ends. The `repmanager` process initializes the action defined in the trigger whenever it receives a replication exception notice.
- Reconfiguring live systems
- Resizing a replicated file on a subscribing server
  UniData provides two commands for an administrator to use for resizing files:
- Listing replication files
  Use the `LIST.REPLICATION.FILE` command to list replication files for a specified account or file.
- Replication files with multiple VOC pointers
  If a file has multiple definitions in the VOC file, in the same or different account, UniData loads each VOC item name in different replication file entries. When a udt or tm process updates a record in the file, the process searches for the primary replication file entry in the replication file table by the i-no/d-no, beginning from the static replication file entries. If there is no match, UniData searches the dynamic replication file entries. UniData uses the first match for replication, generating a replication log in the associated replication group, and sending the VOC file name to the subscriber for publication. This VOC name is referred to as the primary replication file.
- Error messaging
  When you stop replication, detailed error messages are written to the replication log file. The following example lists messages that were written when a publisher process terminated:
- Refreshing the subscriber database
  To refresh the subscribing server, copy the database files from the publishing server to the subscribing server.
- Error handling
  With replication disablement, the runtime error handling of U2 Data Replication changes. If U2 Data Replication incurs a fatal error, U2 Data Replication is disabled and continues running UniData in `DISABLE` mode rather than crashing the publishing server.

- Recovering the system
  At UniData 7.3, the -D option has been added to the `startud` command. This starts UniData and U2 Data Replication in DISABLE mode.
- Monitoring U2 Data Replication through the command line
  Use the REPLOGGER to monitor U2 Data Replication at the command line.

Starting replication

The value of the REP_FLAG parameter in the `udtconfig` file controls whether replication starts. For replication to start, this parameter must be set to 1. Then replication starts and stops as part of the normal startud and stopud scripts.

The following example, on UNIX, shows UniData being started with replication. The component that relates to replication is the message `RM is Started` where RM means replication manager.

```
# $UDTBIN/startud

Using UDTBIN=/disk1/ud72/bin

All output and error logs have been saved
to /disk1/ud72/bin/saved_logs directory.

SMM is started.
Unirpcd is started
SBCS is started.
SM is started.
RM is started.
CLEANUPD is started.

UniData R7.2 has been started.

#
```

If the replication manager fails to start, the output of the start command indicates the component that failed and the log file to find the source of the problem.

Replication messages and errors are logged in the `rm.log` and `rm.errlog` files.

After replication successfully starts and is running normally, the following error files are automatically created. Regularly view these files.

- `rw.errlog`- This file contains all errors encounters by replication writer processes. Before UniData 7.2, replication created individual `rw`*`pid`*`.errlog` files for each replication writer process, where *pid* was the process ID of a replication writer process.
- `udpub`*`n`*`.errlog` (where n is the group number, starting with 0; the sequence of groups is defined in the `repconfig` file) – These files contain details of errors that individual publisher processes encounter.
- `udsub`*`n`*`.errlog` (where *n* is the group number starting from 0; the sequence of groups is defined in the `repconfig` file) – These files contain details of errors encountered by individual replication subscriber processes (`udsub` on UNIX, `udsub.exe` on Windows).

The following is the output from `showud` after a successful start on a publishing server with RFS enabled. (You can tell it's a publishing server because of the `udpub` and `publistener` processes.) The server has four `udpub` processes, numbered 0 to 3, that correspond each group in the `repconfig` file.

On publisher *production-primary* – Replication synced and running

```
# $UDTBIN/showud

    USER     PID     TIME COMMAND
    root  446544     0:00 /disk1/ud72/bin/aimglog 0 20561
    root  585750     0:00 /disk1/ud72/bin/aimglog 1 20561
    root  524526     0:00 /disk1/ud72/bin/bimglog 2 20561
    root  344122     0:00 /disk1/ud72/bin/bimglog 3 20561
    root  872460     0:00 /disk1/ud72/bin/cleanupd -m 10 -t 20
    root  618532     0:00 /disk1/ud72/bin/cm 20561
    root  634916     0:00 /disk1/ud72/bin/publistener 11 0 0 1 1 0 0
    root  696454     0:00 /disk1/ud72/bin/publistener 11 0 0 1 1 3 0
    root  729326     0:00 /disk1/ud72/bin/publistener 11 0 0 1 1 1 0
    root  786508     0:00 /disk1/ud72/bin/publistener 11 0 0 1 1 2 0
    root  774282     0:00 /disk1/ud72/bin/repmanager
    root  897172     0:00 /disk1/ud72/bin/sbcs -r
    root  766184     0:00 /disk1/ud72/bin/sm 60 24974
    root  819396     0:00 /disk1/ud72/bin/smm -t 60
    root  553130     0:00 /disk1/ud72/bin/syncd 0 20561
    root  798908     0:00 /disk1/ud72/bin/syncd 1 20561
    root  573662     0:00 /disk1/ud72/bin/udpub -g0
    root  631010     0:00 /disk1/ud72/bin/udpub -g1
    root  176276     0:00 /disk1/ud72/bin/udpub -g2
    root  504062     0:00 /disk1/ud72/bin/udpub -g3
    root  905450     0:00 /disk1/unishared/unirpc/unirpcd
```

publistener processes

Replication Manager

Publisher processes

The following is the output from `showud` on the matching subscribing server. In the sample `repconfig` file used in this guide, each account-level replication group has the parameter N_REPWRITER=4, and each file-level replication group has the parameter N_REPWRITER=3. Therefore, a total of 13 replication writer processes (`udrw`) are launched for the four replication groups. There is one subscriber process (`udsub`) active for each group.

On subscriber *production-standby*

```
# $UDTBIN/showud
     UID     PID     TIME COMMAND
      USER     PID     TIME COMMAND
      root  671822     0:00 /disk1/ud72/bin/aimglog 0 27180
      root  319546     0:00 /disk1/ud72/bin/aimglog 1 27180
      root  647264     0:00 /disk1/ud72/bin/bimglog 2 27180
      root  352366     0:00 /disk1/ud72/bin/bimglog 3 27180
      root  618730     0:00 /disk1/ud72/bin/cleanupd -m 10 -t 20
      root  663648     0:00 /disk1/ud72/bin/cm 27180
      root  368878     0:00 /disk1/ud72/bin/repmanager
      root  934022     0:00 /disk1/ud72/bin/sbcs -r
      root  725158     0:00 /disk1/ud72/bin/sm 60 3523
      root  422032     0:00 /disk1/ud72/bin/smm -t 60
      root  405742     0:00 /disk1/ud72/bin/udrw -g 2 -s 4
      root  491726     0:00 /disk1/ud72/bin/udrw -g 0 -s 4
      root  540812     0:00 /disk1/ud72/bin/udrw -g 0 -s 4
      root  581744     0:00 /disk1/ud72/bin/udrw -g 1 -s 4
      root  589904     0:00 /disk1/ud72/bin/udrw -g 3 -s 4
      root  635036     0:00 /disk1/ud72/bin/udrw -g 0 -s 4
      root  651288     0:00 /disk1/ud72/bin/udrw -g 1 -s 4
      root  700552     0:00 /disk1/ud72/bin/udrw -g 0 -s 4
      root  708662     0:00 /disk1/ud72/bin/udrw -g 1 -s 4
      root  712780     0:00 /disk1/ud72/bin/udrw -g 3 -s 4
      root  749584     0:00 /disk1/ud72/bin/udrw -g 1 -s 4
      root  766004     0:00 /disk1/ud72/bin/udrw -g 3 -s 4
      root  815302     0:00 /disk1/ud72/bin/udrw -g 2 -s 4
      root  844020     0:00 /disk1/ud72/bin/udrw -g 2 -s 4
      root  905222     0:00 /disk1/ud72/bin/udrw -g 3 -s 4
      root  962584     0:00 /disk1/ud72/bin/udrw -g 2 -s 4
      root  250032     0:00 /disk1/ud72/bin/udsub 4 3600 0
      root  569446     0:00 /disk1/ud72/bin/udsub 4 3600 0
      root  614414     0:00 /disk1/ud72/bin/udsub 4 3600 0
      root  987316     0:00 /disk1/ud72/bin/udsub 4 3600 0
      root  377046     0:00 /disk1/unishared/unirpc/unirpcd
```

Replication writer processes

Subscriber processes

Note: Before UniData 7.2, the subscriber processes on UNIX systems were named `unirepnn` (where *nn* represented the UniData major release – for example unirep71). They were not displayed by `showud` unless you included the -a option. At 7.2, the subscriber processes have been renamed `udsub`. This requires that the 7.2 version of the unirpcd daemon is active. If you upgrade to 7.2 from a prior release and fail to install the new unirpcd daemon (perhaps because it was shared by other versions of UniData or UniVerse on the server), the UNIX subscriber processes are launched as unirepnn processes, but still run properly.

Parent topic: Managing U2 Data Replication through the command line

Example log file for starting replication and synchronizing

The following examples are copies of the `rm.log` file showing a successful start and synchronization of two replication systems.

Note the messages indicating that the subscriber sever is not running and that auto sync failed in the publisher's `rm.log` file. In the example below, this is because the publisher sever was started when the subscriber sever was not running. This results in the `Subscriber system is not running` message. The `repconfig` files have the AUTORESUME parameter set to 1, which results in an attempt to synchronize the systems on start-up (which fails at this point because only one system is operational). This is the expected behavior. The messages should be considered transitional.

Also note the messages in the subscriber's `rm.log` file. When the subscribing sever was started, the subscribing groups were synchronized successfully and resumed running automatically (again because the AUTORESUME parameter is set to 1).

If the AUTORESUME parameter was set to 0 instead of 1, to complete the start-up of replication, the `ud_repadmin` tool would need to be used to enter a synchronization command. This would bring the servers back into synchronization so they would continue working correctly. As this step is easy to forget, we recommend that you set AUTORESUME to 1.

A final note before examining the logs: Any interaction with the system from an administrator using the `ud_repadmin` tool is also logged to the files. You can see the requests detailed in the logs.

On publisher *production-primary*

```
# cat $UDTBIN/rm.log
Starting: Tue May 18 15:46:42 2010

Starting: Tue May 18 15:46:42 2010

RW(471160) is started to load rep objects' inum/dnum for account
/disk1/udt72/inventory
RW(503814) is started to load rep objects' inum/dnum for account
/disk1/udt72/payroll

UniData System Version: 7.2

Replication Protocol: 10004

IPC facilities:

  q - 28311572 (request queue
  q - 28311571 (reply queue)
  m - 24121331 (ctl)
  s - 28311573 (waiting)
  s - 28311572 (waiting)
  s - 28311571 (waiting)
  s - 28311570 (waiting)
  s - 28311569 (waiting)
  s - 28311568 (waiting)
  s - 35651586 (waiting)
  s - 35651587 (waiting)
  s - 35651588 (waiting)
  s - 35651589 (waiting)
  s - 36700166 (waiting)
  s - 34603015 (waiting)

Replication start normal running, pid=176348

udpub for group inventory_acct_g0(0) is started, pid = 471162.
udpub for group inventory_file_g1(1) is started, pid = 503816.
udpub for group payroll_acct_g2(2) is started, pid = 594026.
udpub for group payroll_file_g3(3) is started, pid = 696466.

Started Publishing Group payroll_file_g3 successfully!
Started Publishing Group inventory_file_g1 successfully!
Started Publishing Group payroll_acct_g2 successfully!
Started Publishing Group inventory_acct_g0 successfully!

Successfully Started.
Tue May 18 15:46:43 Subscriber system is not running(2210)!
Auto Resuming Synced GROUP inventory_file_g1, SYSTEM standby failed
Tue May 18 15:46:43 Subscriber system is not running(2210)!
Auto Resuming Synced GROUP inventory_acct_g0, SYSTEM standby failed
Tue May 18 15:46:43 Subscriber system is not running(2210)!
Auto Resuming Synced GROUP payroll_acct_g2, SYSTEM standby failed.
Tue May 18 15:46:43 Subscriber system is not running(2210)!
Auto Resuming Synced GROUP payroll_file_g3, SYSTEM standby failed.
```

Loading repconfig replication object information into replication buffers

Record of specific IPC structures created by Replication daemons

Repmanager PID

Starting udpub processes

"auto" sync failed as subscribing system is not started yet -- **as expected**

```
Tue May 18 15:46:43 Subscriber system is not running(2210)!
Auto Resuming Synced Replication publishing to standby failed.

Tue May 18 15:47:25 2010: Remote request Sync to Replication publishing to
standby received.
```

Subscribing system started and made sync request

```
Tue May 18 15:47:25 2010 Group inventory_acct_g0 Replication to standby
synchronized successfully and resumed running.
Remote System Requested Synced GROUP inventory_acct_g0, SYSTEM standby
successfully
Tue May 18 15:47:25 2010 Group inventory_file_g1 Replication to standby
synchronized successfully and resumed running.
Remote System Requested Synced GROUP inventory_file_g1, SYSTEM standby
successfully
Tue May 18 15:47:25 2010 Group payroll_acct_g2 Replication to standby
synchronized successfully and resumed running.
Remote System Requested Synced GROUP payroll_acct_g2, SYSTEM standby
successfully
Tue May 18 15:47:25 2010 Group payroll_file_g3 Replication to standby
synchronized successfully and resumed running.
Remote System Requested Synced GROUP payroll_file_g3, SYSTEM standby
successfully
Remote System Requested Synced Replication publishing to standby successfully
```

Sync completed successfully – all groups reached "resumed running" status

```
Tue May 18 15:50:32 2010: Request DBA Report to ALL received.
#
```

Log of "ud_repadmin report –detail –verbose" command being run by administrator

Stopping the database and replication

Note: The following figures assume that all UniData processes have already exited and you are not using `stopud -f` to force the database down. As in all examples, RFS is enabled.

Parent topic: Managing U2 Data Replication through the command line

Stopping the database and replication on the publishing server

After all UniData processes have exited, when you login to UniData as `root` and run `stopud` on the publishing server (followed by `showud`), you should see the following output:

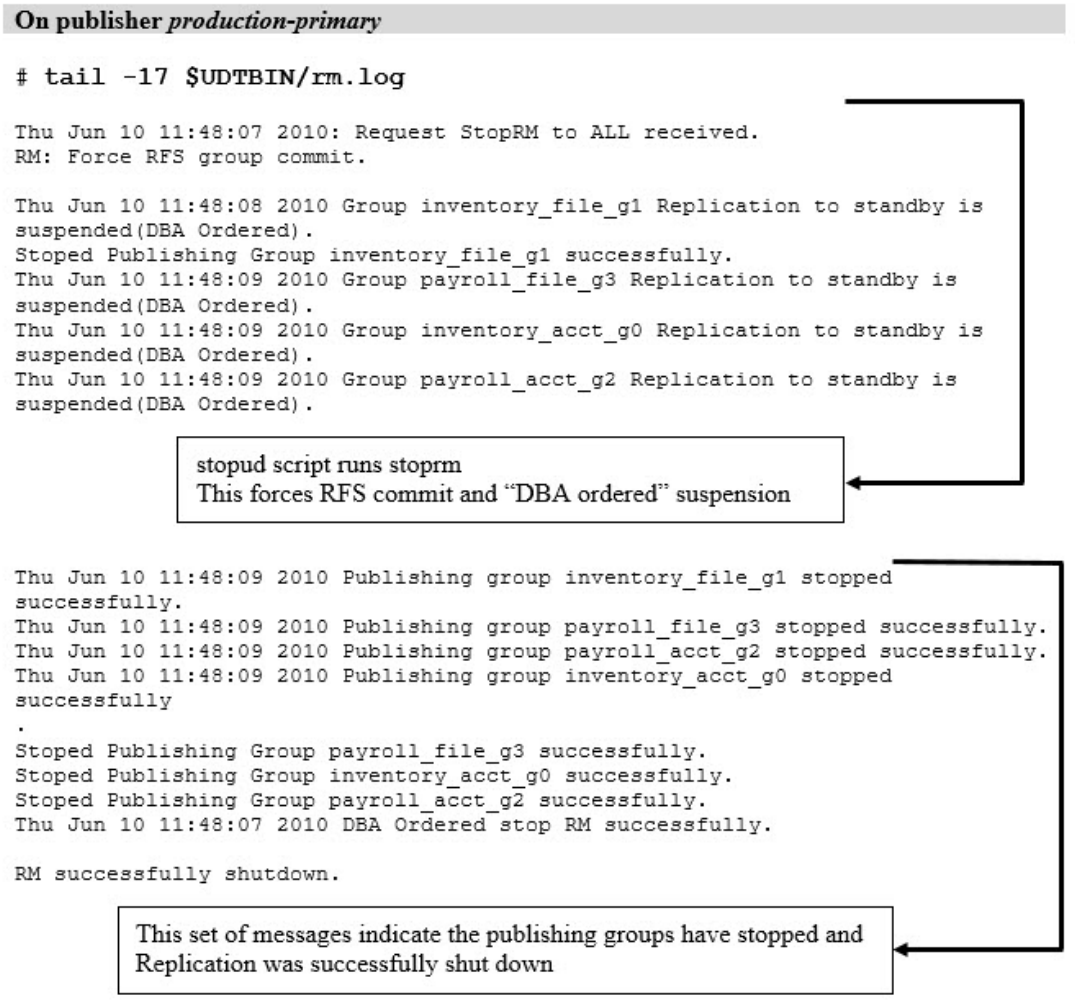
On publisher *production-primary*

```
# $UDTBIN/stopud

Using UDTBIN=/disk2/ud72/bin

RM stopped successfully.
SM stopped successfully.
CLEANUPD stopped successfully.
SBCS stopped successfully.
SMM stopped successfully.
Unirpcd stopped successfully

Unidata R7.2 has been shut down.
#
#
# $UDTBIN/showud
      USER     PID     TIME COMMAND
#
```

Replication Manager stopped

No output from showud indicates all UniData daemons have exited – including **repmanager**, **publistener** processes and **udpub** processes

As replication shuts down during the `stopud` processing, entries are made in the `$UDTBIN/rm.log`.



Replication is suspended when UniData is shut down on the publishing system. On the subscribing server, you see the same results as if an administrator had run a `$UDTBIN/ud_repadmin suspend` command on the publishing server:

- The replication writer (`udrw`) processes exit.
- The subscriber processes exit.
- The suspension request and status is noted in the `$UDTBIN/rm.log`.

On subscriber *production-standby*

```
# $UDTBIN/showud
      USER      PID      TIME COMMAND
      root   405532      0:00 /disk1/ud72/bin/aimglog 0 12483
      root   635134      0:00 /disk1/ud72/bin/aimglog 1 12483
      root   663702      0:00 /disk1/ud72/bin/bimglog 2 12483
      root   536666      0:00 /disk1/ud72/bin/bimglog 3 12483
      root   422062      0:58 /disk1/ud72/bin/cleanupd -m 10 -t 20
      root   696514      0:29 /disk1/ud72/bin/cm 12483
      root   250088      0:00 /disk1/ud72/bin/repmanager
      root   319632      0:00 /disk1/ud72/bin/sbcs -r
      root   614482      0:01 /disk1/ud72/bin/sm 60 5367
      root   676070      0:06 /disk1/ud72/bin/smm -t 60
      root   377046      0:00 /disk1/ud72/unishared/unirpc/unirpcd
#
```

showud output indicates all udrw and udsub processes have exited (they are not listed)

On subscriber *production-standby*

```
# tail -8 $UDTBIN/rm.log
Thu Jun 10 11:48:11 2010 Subscribing Group inventory_file_g1 started suspending
(Publisher Requested).
Thu Jun 10 11:48:11 2010 Subscribing Group inventory_acct_g0 started suspending
(Publisher Requested).
Thu Jun 10 11:48:11 2010 Subscribing Group payroll_file_g3 started suspending (Publisher
Requested).
Thu Jun 10 11:48:11 2010 Subscribing Group payroll_acct_g2 started suspending (Publisher
Requested).
```

rm.log messages indicate that the "Publisher Requested" Replication suspension

```
Thu Jun 10 11:48:12 2010 Subscribing group inventory_file_g1 stopped
successfully.
Thu Jun 10 11:48:12 2010 Subscribing group inventory_acct_g0 stopped
successfully.
Thu Jun 10 11:48:12 2010 Subscribing group payroll_file_g3 stopped
successfully.
Thu Jun 10 11:48:12 2010 Subscribing group payroll_acct_g2 stopped
successfully.

#
```

rm.log messages indicating each group has successfully suspended Replication

Stopping the database and replication on the subscribing server

After all UniData processes have exited, when you log in to UniData as `root` and run `stopud` on the subscribing server (followed by `showud`), you should see the following output:

On subscriber *production-standby*

```
# $UDTBIN/stopud

Using UDTBIN=/disk1/ud72/bin

RM stopped successfully.
SM stopped successfully.
CLEANUPD stopped successfully.
SBCS stopped successfully.
SMM stopped successfully.
Unirpcd stopped successfully

Unidata R7.2 has been shut down.

# $UDTBIN/showud
      UID      PID      TIME COMMAND
#
#
```

Replication Manager stopped

No output from showud indicates all UniData daemons have exited – including **repmanager**, **subscriber** processes and **udrw** processes

As Replication shuts down during the stopud processing, entries are made in the rm.log.

On subscriber *production-standby*

```
# tail -13 $UDTBIN/rm.log

Thu Jun 10 12:43:01 2010: Request StopRM to ALL received.
RM: Force RFS group commit.

Thu Jun 10 12:43:07 2010 Subscribing group inventory_acct_g0 stopped successfully.
Stoped Subscribing Group inventory_acct_g0 successfully.
Thu Jun 10 12:43:07 2010 Subscribing group payroll_file_g3 stopped successfully.
Stoped Subscribing Group payroll_file_g3 successfully.
Thu Jun 10 12:43:07 2010 Subscribing group inventory_file_g1 stopped successfully.
Stoped Subscribing Group inventory_file_g1 successfully.
Thu Jun 10 12:43:07 2010 Subscribing group payroll_acct_g2 stopped successfully.
Stoped Subscribing Group payroll_acct_g2 successfully.
Thu Jun 10 12:43:01 2010 DBA Ordered stop RM successfully.

RM successfully shutdown.

#
```

stopud script runs stoprm. This forces RFS commit and suspends subscriber processes

These messages indicate that the subscribing groups were stopped and Replication was successfully shut down

Defining subscribing groups from the command line

Define a subscribing replication group from the command line.

About this task

For more information, see Configuring replication, on page 69.

Procedure

1. Log on to the subscribing server.
2. Edit the `repsys` file, located in `/usr/ud82/include` on UniData for UNIX or `udthome\include` on UniData for Windows platforms. Define all replication systems in the replication environment to which U2 Data Replication may connect.
3. Edit the `repconfig` file located in the `/usr/ud82/include` on UniData for UNIX or `udthome\include` on UniData for Windows platforms.

Define all the replication groups you want to replicate from the publisher. Make sure you use the same group name used on the publisher.

- Copy the file list of the group from the publisher.
- Define the publishing server according to the definition on the publisher.
- Define the subscribing server, and define the replication type.
- If you are going to use this system as a standby subscriber, define all other distributions of the group and their replication types, according to the definition on the publisher.
- Set the replication group configurable parameters.
- Define the subscribing account path in replication.

4. Create the subscribing account, then copy all files from the publishing server to the subscribing server.
5. Restart UniData, or execute the `ud_repadmin reconfig` command to load the replication group definition and allocate system resources.

Parent topic: Managing U2 Data Replication through the command line

Defining publishing groups from the command line

Define a publishing replication group from the command line.

About this task

For more information, see Configuring replication, on page 69.

1. Log on to the publishing server.
2. Edit the `repsys` file, located in `/usr/ud82/include` on UniData for UNIX or `udthome\include` on UniData for Windows platforms. Define all replication systems in the replication environment to which U2 Data Replication may connect.
3. Edit the `repconfig` file, located in `/usr/ud82/include` on UniData for UNIX or `udthome\include` on UniData for Windows platforms.

   For each replication group, define the following information:

   - Assign a unique replication group name.
   - Define the replication level.
   - Define the location of the publishing account.
   - Define excluded files for account-level replication.
   - Define all files belonging to the file-level replication group.
   - Define the distribution for the group, including the publishing server and all subscribing servers to which it replicates.
   - Define the replication type for each of the distribution systems.
   - Set the replication group configurable parameters.
4. Restart UniData, or execute the `ud_repadmin reconfig` command.

Parent topic: Managing U2 Data Replication through the command line

Changing an existing replication group definition from the command line

Reconfigure U2 Data Replication from the command line.

1. Log on to the replication system you want to reconfigure.
2. Make your changes in the `repsys` and `repconfig` files.
3. Synchronize the reconfigured group from its publisher or to its subscriber.
4. Execute the `RECONFIG` command to request that the replication manager reconfigure the replication group.

The replication manager executes the `dbpause` command to block writes to the database. If you are running RFS, the `dbpause` command also forces a checkpoint.

The replication manager requests that all replication groups suspend replication.

The replication manager reads the new `repconfig` and `repsys` files, and compares them to the original file.

The replication manager checks the consistency of the definition, and checks each of the groups that are changing to see if there are any replication logs not synchronized to its subscribers.

The replication manager reallocates shared memory segments, reloads the replication configurations, and the Inode and device number of each replication object.

The replication manager executes the `dbresume` command to resume processing.

If you selected the AUTORESUME option, the replication manager synchronizes and resumes all real-time and immediate replication groups to its publishers and subscribers. If you did not select the AUTORESUME option, you should synchronize manually.

Parent topic: Managing U2 Data Replication through the command line

Enabling and disabling U2 Data Replication through the command line

You can enable or disable U2 Data Replication using the command line.

Parent topic: Managing U2 Data Replication through the command line

Enabling and disabling U2 Data Replication using ud_repadmin

Use the `ud_repadmin` command to enable or disable U2 Data Replication from the operating system level.

The replication `ud_repadmin enable` mode is the normal running mode for U2 Data Replication. If U2 Data Replication is disabled, it is enabled if the system administrator issues the `ENABLE` command, or UniData is started with the -i option.

When the replication manager receives an `ud_repadmin enable` request, it performs the following tasks:

- Issues the `dbpause` command.
- Clears the replication disable flag and saves it in the `udrepsys.log` file.

- Resets the log sequence number for all groups.
- Purges the replication log files.
- Starts the publishing processes for all publishing groups.
- issues the `dbresume` command.
- If the AUTORESUME parameter is enabled, issues the `SYNC` command to all replications.

To enable U2 Data Replication, use the enable option:

```
ud_repadmin enable [-verbose]
```

The -verbose option displays messages to the screen.

In replication `ud_repadmin disable` mode, all replication functions stop, but the replication manager continues to run. The following actions occur when U2 Data Replication is disabled:

- All replications are suspended.
- All publishing processes stop.
- The udt and tm process stops generating replication logs.
- The replication manager accepts only the `ENABLE`, `RECONFIG`, and `STOP` commands.
- The `SYNC`, `SUSPEND`, and `RESET` commands do not proceed until replication is enabled.
- The `ud_repadmin disable` mode persists after stopping, reconfiguring, and starting the database.

If one of the following events occurs, U2 Data Replication automatically switches to `ud_repadmin disable` mode:

- You issue the `ud_repadmin disable` command.
- A fatal replication error occurs. The error handling procedure sends a request to disable replication.
- You use the `startud -D` command to start the database.

When the replication manager receives a `ud_repadmin disable` request, it performs the following tasks:

- Suspends all active replications.
- Sets the replication disable flag and saves it in the `udrepsys.log`.
- Issues the `dbpause` command.
- Exits the publishing processes.
- If the ClearLog option was specified, purges the replication log files for all replication groups.
- Issues the `dbresume` command.
- If the `DISABLE` request was automatically issued in response to an error, the exception action script runs the program if one is defined on the local server.

When the replication manager starts with the disable option, it loads the replication configuration, sets the replication disable flag and saves it in the `udrepsys.log`, and continues running in `ud_repadmin disable` mode.

To disable U2 Data Replication, use the disable option:

```
ud_repadmin disable [-clearlog] [-verbose]
```

The -clearlog option clears the replication log files. The -verbose option displays messages to the screen.

You must be logged in as root to enable or disable U2 Data Replication.

Enabling or disabling U2 Data Replication using startud and stopud

You can use the `startud` command to enable U2 Data Replication. Use `stopud` to disable it.

Enable: `startud -D`

The -D option starts UniData with U2 Data Replication disabled. You cannot use this option with the "-i" or "-m" options. If you start UniData after a crash with RFS enabled, the "-D" option disables U2 Data Replication and restarts UniData with full RFS crash recovery.

If you use the -i option with the `startud` command, U2 Data Replication will be enabled when the database starts. This option should be used with caution, as it triggers broader actions than merely enabling U2 Data Replication. If you use the -i option on an RFS system after a crash, RFS recovery is disabled. This could result in file corruption to recoverable files. With U2 Data Replication, the LSN numbers are also reset and the LRF logs abandoned.

Disable: `stopud -f`

The -f option forces UniData daemons to stop unconditionally, which kills all active udt processes, including U2 Data Replication.

Disabling U2 Data Replication

When replication is suspended, the publisher continues generating replication logs and reserves them for the next synchronization. If replication is suspended for too long, the publishing server might accumulate enough replication logs to fail the replication log file system. In an extreme case, a full file system can crash the entire publisher system. For this reason, disablement of replication prior to filling the disk entirely may provide for a more graceful recovery.

If an RFS-enabled publishing server crashes, the current recovery has two steps. First, the RFS recovery process applies before image logs to the database files and restores the physical consistency of the files. Then the replication recovery process handles the after image logs. If there is no intentional failover, the replication recovery process applies the logs to the database. If there is a failover or intentional failover, the RFS after image logs are used to resynchronize to the failed-over system. This mechanism ensures data consistency and quick synchronization should failover occur after the primary server crash. However, if replication recovery fails or some other serious failures occur and the replication system cannot recover, the DBA may want to disable replication and have full RFS recovery to quickly overcome the failure and recover the system. The new U2 Data Replication disablement mechanism is addressing this function.

Replication exception action programs

A replication exception action program defines the action to take if replication abnormally ends. The `repmanager` process initializes the action defined in the trigger whenever it receives a replication exception notice.

The exception action program is a shell script on UNIX, and a batch program on Windows. Define the full path to the trigger using the EXCEPTION_ACTION clause in the `repsys` file. For example, if you define a replication action trigger as `UDRepExceptionAction.sh` in the `/usr/u2` directory, set the EXCEPTION_ACTION parameter as:

```
EXCEPTION_ACTION=/usr/u2/UDRepExceptionAction.sh
```

U2 Data Replication passes parameters to the replication action trigger through environment variables. You can use the following environment variables in the replication action trigger program:

- UNIREP_REMOTESYS – the remote system name of the replication.
- UNIREP_REPTYPE – the type of replication.
- UNIREP_GRPNAME – the replication group name where the exception occurred.
- UNIREP_ERRCODE – the exception error code.
- UNIREP_ERRSTRING – the exception error string.

If a replication has multiple replication groups, the interruption of each group is a separate event that executes the trigger for that group. If a single group belongs to multiple replications, each interruption of the replication triggers the replication action trigger to run. UNIREP_REMOTESYS and UNIREP_REPTYPE define the replication where the exception occurred.

Use the exception action program to automatically notify the system administrator that an exception occurred. Do not use the exception action program to automatically failover to a standby server.

A replication exception action program runs only when an unexpected event suspends replication. If a system administrator issues a command that suspends replication, the exception action program is not invoked.

When the exception action program runs, the replication manager process writes a message similar to the following example in the replication log:

`Execute Replication Exception Action program` *`program_path`*

You can write and deploy an exception action program that runs when replication is suspended or disabled unexpectedly. Register the script using the `EXCEPTION_ACTION` parameter in the local system `repsys` file. If U2 Data Replication is suspended or disabled unexpectedly, the program runs and returns environment variables that describe the error.

The following table lists the environment variables.

Environment variable	Description
UNIREP_GRPNAME	The replication group that triggered replication disablement or suspension.
UNIREP_REMOTESYS	The local server name defined in the `repsys` file.
UNIREP_REPTYPE	Empty.
UNIREP_ERRCODE	The replication error code that triggered replication disablement or suspension.
UNIREP_ERRSTRING	The error message pertaining to the error code.
UNIREP_REPSTATUS	The replication status. Valid values are `REP_SUSPENDED` and `REP_DISABLED`.
UNIREP_PID	The process ID that triggered replication disablement or suspension. This value is empty if the status is `REP_SUSPENDED`.
UNIREP_OSERRNO	The operating system error code that triggered replication disablement or suspension. This value is empty if the status is `REP_SUSPENDED`.

Parent topic: Managing U2 Data Replication through the command line

Sample EXCEPTION_ACTION script for UNIX

Here is a sample UNIX file that can give you some ideas on how to construct an EXCEPTION_ACTION script that is appropriate for your system.

```
#!/bin/sh
#
# Replication Status Code that returned by ud_repadmin report command
#
# (c) Rocket Software 2014 - 2017 All Rights Reserved
# Disclaimer of Warranties. Rocket Software disclaims to the fullest
extent authorized by law any and all other warranties, whether express
or implied, including, without limitation, any implied warranties
# of  merchantability or fitness for a particular purpose. Without
limitation of the foregoing, Rocket Software expressly does not warrant that:
# (a) the software will meet your requirements [or expectations];
# (b) the software or the software content] will be free of bugs,
errors, viruses or other defects;
# (c) any results, output, or data provided through or generated by the
software will be accurate, up-to-date, complete or reliable;
# (d) the software will be compatible with third party software;
# (e) any errors in the software will be corrected.
#
#   1.3.1 JDS 25/02/14 First Proposed Customer Release
#   1.3.5 JDS 03/06/15 Change to all_synced check for disablement
#   1.4.7 JDS 22/01/16 Small Spelling Correction
#   1.4.8 JDS 27/01/16 Change to ignore publishing group errors on Recovery
#                      This is for sites with mixed publishing and subscribing groups
#                      On the same machine
#   1.5.0 JDS 16/11/16 Add configurable for CGT (Cross Group Transactions) and
also try to detect CGT
#                      If CGT is detected or set Script will sync all groups at once to
#                      avoid Replication Stall of one by one sync and CGT
#   1.5.1 JDS 20/12/16 Modify CGT sync call to remove DISTRIB phrase
#   1.5.2 JDS 22/12/16 Correct version number and correct $filenane typo to $filename
#                      and deal with a blank file for last run time
#   1.5.3 JDS 27/03/17 Email notification if it is found that script is already running
#
# Publishing group Status Codes
export PUB_STOP=1 #/* group is normally stopped */
export PUB_EXIT=2 #/* group has quit abnormally.*/
export PUB_SHUTDOWN=3 #/* group is doing shutdown */
export PUB_RUNNING=4 #/* group is in running mode */
export PUB_DO_RECONFIG=5 #/* group is doing reconfigure */
export PUB_DO_FAILOVER=7 #/* group is doing failover */
# Publishing Replication Status Codes
export REP_RUNNING=11 #/* replication is in running mode */
export REP_SUSPENDED=12 #/* replication is in suspension mode */
export REP_SYNCING=13 #/* replication is in syncing mode */
export REP_DO_SUSPEND=14 #/* replication is doing suspend */
export REP_DO_SYNC=15 #/* replication is doing sync */
export REP_EXIT=16 #/* replication exit */
export REP_DISABLED=17 #/* replication disabled */
# Subscribing Replication/group status codes.
export SUB_STOP=21 #/* group is normally stopped */
export SUB_EXIT=22 #/* group has quit abnormally.*/
export SUB_SHUTDOWN=23 #/* group is doing shutdown */
export SUB_RUNNING=24 #/* replication is in running mode */
export SUB_DO_RECONFIG=25 #/* group is doing reconfigure */
export SUB_DO_SUSPEND=27 #/* replication is suspending */
export SUB_SYNCING=28 #/* replication is doing sync */
```

```
export SUB_DO_FAILOVER=29 #/* group is doing failover */
export SUB_RESYNCING=30 #/* group is doing re-syncing */
# Script Version
export REP_EXCEP_SCRIPT_VERSION=1.5.3
export CROSS_GROUP_TRANSACTION=0
export debugrequest=0
export versionrequest=0
while getopts ":xv" opt; do
case $opt in
x ) debugrequest=1 ;;
v ) versionrequest=1 ;;
esac
done
if [ $debugrequest = 1 ]
then
  set -x
fi
if [ $versionrequest = 1 ]
then
 echo "Script Version is :"$REP_EXCEP_SCRIPT_VERSION
 exit 0
fi
# Check UDTHOME
udthome="$UDTHOME"
if [ "$udthome" = "" ]
then
  export UDTHOME=/apps/ud
fi
udtbin="$UDTBIN"
if [ "$udtbin" = "" ]
then
  export UDTBIN=$UDTHOME/bin
fi
# Number of minutes to check before running script again
export PUBCHECKMIN=2
export SUBCHECKMIN=2
export DISCHECKMIN=2
# Sleep is in seconds not minutes
export RESYNCSLEEP=120
#
export MAILRECS="someone@you.com,someoneelse@you.com"
# Show exception time
export dstring=`date`
export reportfilename=$UDTHOME'/RepExcept.errlog'
export scriptrunningname=$UDTHOME'/RepExcept.running'
if [ -f $scriptrunningname ]
then
  echo $scriptrunningname" found script already running - exiting"
  echo "Subject: Exception Script is Already Running - New Request Rejected"
  | cat - $reportfilename | /usr/lib/sendmail -F $machinename -t $MAILRECS
  exit 5
fi
echo "Run Started at :"$dstring > $scriptrunningname
echo "UniData Replication Exception called at:"$dstring > $reportfilename
echo "Script Version is:"$REP_EXCEP_SCRIPT_VERSION >> $reportfilename
export machinename=`uname -a | awk '{print $2}'`
echo "Run on machine "$machinename >> $reportfilename
echo "Is this the publisher or subscriber" >> $reportfilename
export cmd="$UDTBIN/ud_repadmin report $UNIREP_REMOTESYS"
$cmd
returncode=$?
echo "returncode=$returncode" >> $reportfilename
```

```
export publisher=0
export subscriber=0
export disabled=0
if [ $returncode -lt 17 ]
then
  publisher=1
fi
if [ $returncode -gt 17 ]
then
  subscriber=1
fi
if [ $returncode = "17" ]
then
  disabled=1
fi
if [ $publisher = 1 ]
then
  echo "System is a publisher" >> $reportfilename
fi
if [ $subscriber = 1 ]
then
  echo "System is a subscriber" >> $reportfilename
fi
if [ $disabled = 1 ]
then
  echo "Replication is Disabled !" >> $reportfilename
fi

# Check Last Run Time and avoid running multiple times
export checkfilename=$UDTHOME'/RepExcept.runtime'
export ddatetime=`date +"%Y %j %H %M"`
ddatetime=`echo $ddatetime | sed 's/ //g'`
export run_check=0
if [ ! -f $checkfilename ]
then
  # Put the current date time into the log file
  echo $ddatetime > $checkfilename
  run_check=1
else
  ldatetime=`cat $checkfilename`
  if [ "$ldatetime" = "" ]
  then
    echo "Empty Last Run File Found" >> $reportfilename
    echo $ddatetime > $checkfilename
    run_check=1
  else
    if [ $publisher = 1 ]
    then
      tdatetime=`expr $ldatetime + $PUBCHECKMIN`
    fi
    if [ $subscriber = 1 ]
    then
      tdatetime=`expr $ldatetime + $SUBCHECKMIN`
    fi
    if [ $disabled = 1 ]
    then
      tdatetime=`expr $ldatetime + $DISCHECKMIN`
    fi
    echo "Last run at "$ldatetime >> $reportfilename
    echo "Current "$ddatetime >> $reportfilename
    echo "Next "$tdatetime >> $reportfilename
    if [ $ddatetime -gt $tdatetime ]
```

```
    then
      run_check=1
      echo $ddatetime > $checkfilename
    fi
  fi
fi
echo "Run Check "$run_check >> $reportfilename
if [ $run_check = 0 ]
then
  rm $scriptrunningname
  exit
fi
# Show environment variables passed from replication system
echo "UNIREP_REMOTESYS="$UNIREP_REMOTESYS >> $reportfilename
echo "UNIREP_REPTYPE="$UNIREP_REPTYPE >> $reportfilename
echo "UNIREP_GRPNAME="$UNIREP_GRPNAME >> $reportfilename
echo "UNIREP_ERRCODE="$UNIREP_ERRCODE >> $reportfilename
echo "UNIREP_ERRSTRING="$UNIREP_ERRSTRING >> $reportfilename
echo "UDTBIN=$UDTBIN" >> $reportfilename
echo "Return Error Codes" >> $reportfilename
echo "NO ERROR=0" >> $reportfilename
echo "PUB_STOP="$PUB_STOP >> $reportfilename
echo "PUB_EXIT="$PUB_EXIT >> $reportfilename
echo "PUB_SHUTDOWN="$PUB_SHUTDOWN >> $reportfilename
echo "PUB_RUNNING="$PUB_RUNNING >> $reportfilename
echo "PUB_DO_RECONFIG="$PUB_DO_RECONFIG >> $reportfilename
echo "PUB_DO_FAILOVER="$PUB_DO_FAILOVER >> $reportfilename
echo "REP_RUNNING="$REP_RUNNING >> $reportfilename
echo "REP_SUSPENDED="$REP_SUSPENDED >> $reportfilename
echo "REP_SYNCING="$REP_SYNCING >> $reportfilename
echo "REP_DO_SUSPEND="$REP_DO_SUSPEND >> $reportfilename
echo "REP_DO_SYNC="$REP_DO_SYNC >> $reportfilename
echo "REP_EXIT="$REP_EXIT >> $reportfilename
echo "REP_DISABLED="$REP_DISABLED >> $reportfilename
echo "SUB_STOP="$SUB_STOP >> $reportfilename
echo "SUB_EXIT="$SUB_EXIT >> $reportfilename
echo "SUB_SHUTDOWN="$SUB_SHUTDOWN >> $reportfilename
echo "SUB_RUNNING="$SUB_RUNNING >> $reportfilename
echo "SUB_DO_RECONFIG="$SUB_DO_RECONFIG >> $reportfilename
echo "SUB_DO_SUSPEND="$SUB_DO_SUSPEND >> $reportfilename
echo "SUB_SYNCING="$SUB_SYNCING >> $reportfilename
echo "SUB_DO_FAILOVER="$SUB_DO_FAILOVER >> $reportfilename
echo "SUB_RESYNCING="$SUB_RESYNCING >> $reportfilename
# Run ud_repadmin report command
export cmd="$UDTBIN/ud_repadmin report -detail $UNIREP_REMOTESYS"
echo "Now running:"$cmd >> $reportfilename
$cmd >> $reportfilename
echo "showud" >> $reportfilename
$UDTBIN/showud >> $reportfilename
echo "Please See "$reportfilename" for more information" >> $reportfilename
# SendMail of Failure
echo "Attempting First SendMail" >> $reportfilename
if [ $disabled = 1 ]
then
   echo "Subject: REPLICATION DISABLED" | cat - $reportfilename |
   /usr/lib/sendmail -F $machinename -t $MAILRECS
   echo "Replication Disabled Email Sent" >> $reportfilename
else
   echo "Subject: REPLICATION SUSPENDED" | cat - $reportfilename |
   /usr/lib/sendmail -F $machinename -t $MAILRECS
   echo "Replication Suspended Email Sent" >> $reportfilename
fi
```

```
if [ $publisher = 1 ]
then
  echo "System Reported it was a publisher no need to try sync" >> $reportfilename
  rm $scriptrunningname
  exit 0
fi
if [ $disabled = 1 ]
then
  echo "System Reported Replication was disabled no need to try sync" >>
  $reportfilename rm $scriptrunningname
  exit 0
fi
# Check all groups are suspended before proceeding
# Not Needed for Publisher or if Replication is disabled
export all_suspended=0
export try=0
while [ $all_suspended = 0 -a $try -lt 10 ]
do
  sleep 5
  all_suspended=1
  menu_str="2\n0\n0\n0\n\n\n\n"
  for grpname in `printf $menu_str | $UDTBIN/reptool | grep "^Group"
  | awk '{print $3}'`; do
     if [ $all_suspended ]
     then
       # Run ud_repadmin report command
     export cmd="$UDTBIN/ud_repadmin REPORT GROUP $grpname DISTRIB $UNIREP_REMOTESYS"
     echo "Try Counter is "$try >> $reportfilename
     echo "Now running:"$cmd >> $reportfilename
     $cmd
     returncode=$?
     echo "returncode=$returncode" >> $reportfilename
     if [ $returncode -gt 30 ]
     then
 # Report command failed
        echo "Report command failed, exit." >> $reportfilename
        exit 2
        rm $scriptrunningname
     elif [ $returncode = $REP_SUSPENDED -o $returncode = $SUB_STOP
     -o $returncode = $SUB_EXIT ]
     then
        echo "Group $grpname is suspended." >> $ filename
     else
        all_suspended=0
     fi
  fi
  done
  try=`expr $try + 1`
done

if [ $all_suspended = 0 ]
then
   echo "Number of Try's Exceeded to Confirm All Groups Suspended"
   >> $reportfilename
   echo "Subject: UNABLE TO CONFIRM SUSPENSION OF ALL GROUPS
   (NO SYNC WILL BE ATTEMPTED)" | cat - $reportfilename |
   /usr/lib/sendmail -F $machinename -t $MAILRECS
   echo "Unable to confirm suspension email sent" >> $reportfilename
   rm $scriptrunningname
   exit 3
fi
echo "Subject: CONFIRMED SUSPENSION OF ALL GROUPS - Resync Attempt in 2 Minutes" |
```

```
cat - $reportfilename | /usr/lib/sendmail -F $machinename -t $MAILRECS
echo "Confirmed suspension email sent" >> $reportfilename
# Try automatic resync and resume - only run on subscriber
echo "Sleeping for $RESYNCSLEEP" >> $reportfilename
sleep $RESYNCSLEEP
echo "Woke up from Sleep" >> $reportfilename

echo "Cross Group Transaction Currently set to $CROSS_GROUP_TRANSACTION"
>> $reportfilename
if [ $CROSS_GROUP_TRANSACTION = 0 ]
then
  echo "Testing for Transactions" >> $reportfilename
  menu_str="6\n1\n0\n0\n\n\n"
  trans_cnt=`printf $menu_str | $UDTBIN/reptool | grep "^TCR Created:"
  | awk '{print $3}'`
  if [ $trans_cnt = 0 ]
  then
    echo "No Transactions Seen Leaving CGT Unset" >> $reportfilename
  else
    echo "Transactions Found so setting CGT" >> $reportfilename
    CROSS_GROUP_TRANSACTION=1
  fi
fi

export all_synced=1
export group_syncing=1
if [ $CROSS_GROUP_TRANSACTION = 0 ]
then
menu_str="2\n0\n0\n0\n\n\n"
for grp_name in `printf $menu_str | $UDTBIN/reptool | grep "^Group"
  | awk '{print $3}'`; do
   cmd="$UDTBIN/ud_repadmin sync -wait -verbose GROUP $grp_name
   DISTRIB $UNIREP_REMOTESYS"
   echo "Now running:"$cmd >> $reportfilename
   $cmd >> $reportfilename
   returncode=$?
   echo "Return code=$returncode" >> $reportfilename
   if [ $returncode = 0 ]
   then
      echo "Group Sync Worked" >> $reportfilename
   else
      export cmd="$UDTBIN/ud_repadmin report GROUP $grp_name"
      echo "Now running:"$cmd >> $reportfilename
      $cmd
      returncode=$?
      echo "returncode=$returncode" >> $reportfilename
      if [ $returncode = $REP_SYNCING -o $returncode = $REP_DO_SYNC
      -o $returncode = $REP_RUNNING  -o $returncode = $SUB_RUNNING
      -o $returncode = $SUB_SYNCING -o $returncode = $SUB_RESYNCING ]
      then
         group_syncing=1
      else
         group_syncing=0
      fi
      if [ $returncode -lt 17 ]
      then
         group_syncing=1
         echo "group appears to be a publishing group so and sync
               failures are ignored"
      fi
      echo "group_syncing=$group_syncing" >> $reportfilename
      if [ $group_syncing = 0 ]
```

```
      then
         echo "Group Sync Failed" >> $reportfilename
         all_synced=0
      fi
   fi
   sleep 1
done
else
   cmd="$UDTBIN/ud_repadmin sync -wait -verbose $UNIREP_REMOTESYS"
   echo "Now running:"$cmd >> $reportfilename
   $cmd >> $reportfilename
   returncode=$?
   echo "Return code=$returncode" >> $reportfilename
   if [ $returncode = 0 ]
   then
      echo "Whole Sync Worked" >> $reportfilename
   else
      export cmd="$UDTBIN/ud_repadmin report"
      echo "Now running:"$cmd >> $reportfilename
      $cmd
      returncode=$?
      echo "returncode=$returncode" >> $reportfilename
      if [ $returncode = $REP_SYNCING -o $returncode = $REP_DO_SYNC
      -o $returncode = $REP_RUNNING  -o $returncode = $SUB_RUNNING
      -o $returncode = $SUB_SYNCING -o $returncode = $SUB_RESYNCING ]
      then
         group_syncing=1
      else
         group_syncing=0
      fi
      if [ $returncode -lt 17 ]
      then
         group_syncing=1
         echo "group appears to be a publishing group so and sync failures are ignored"
      fi
      echo "group_syncing=$group_syncing" >> $reportfilename
      if [ $group_syncing = 0 ]
      then
         echo "Whole Sync Failed" >> $reportfilename
         all_synced=0
      fi
   fi
   sleep 1
fi

echo "all_synced flag = "$all_synced >> $reportfilename
if [ $all_synced = 1 ]
then
 echo "Subject: Sync Commmands Worked - Please Check Systems" |
 cat - $reportfilename | /usr/lib/sendmail -F $machinename -t $MAILRECS
else
 echo "Subject: SYNC COMMANDS FAILED - PLEASE CHECK SYSTEMS" |
 cat - $reportfilename | /usr/lib/sendmail -F $machinename -t $MAILRECS
 rm $scriptrunningname
 exit 4
fi
rm $scriptrunningname
exit 0
```

Sample EXCEPTION_ACTION script for Windows

Here is a sample Windows `.bat` file that can give you some ideas on how to construct an EXCEPTION_ACTION script that is appropriate for your system.

Note: If you include a `sync` command in your script to automatically resync replication after an unexpected suspension, be sure to include this script on either the publishing server or the subscribing server, but not both. As a best practice, the subscribing server is recommended. If scripts on both servers are firing off repeated sync commands, replication can become confused.

```
: Simple UniData Data Replication Exception Action script example

@echo off

: Define Replication Groups
set REP_GROUPS=inventory_acct_g0 inventory_file_g1 payroll_acct_g2 payroll_file_g3

: Show exception time
echo @prompt set date=$d$_set time=$t$h$h$h > {a}.bat
%comspec% /e:2048 /c {a}.bat > {b}.bat
for %%v in ({b}.bat del) do call %%v {?}.bat
echo UniData Replication Exception called at: %date% %time%
echo UDTBIN=%UDTBIN%

: Show environment variables passed from replication system
echo UNIREP_REMOTESYS=%UNIREP_REMOTESYS%
echo UNIREP_REPTYPE=%UNIREP_REPTYPE%
echo UNIREP_GRPNAME=%UNIREP_GRPNAME%
echo UNIREP_ERRCODE=%UNIREP_ERRCODE%
echo UNIREP_ERRSTRING=%UNIREP_ERRSTRING%

: Now Paging DBA
: [Insert your DBA alert code here - e.g. sendmail]

: Sleep a bit to allow cross-group suspension to complete
: Next line  emulates SLEEP 60
ping 1.0.0.0 -n 1 -w 60000 > %TMP%\NUL

: Display status of each replication group
: Using ud_repadmin REPORT command on each of the groups

for %%g in (%REP_GROUPS%) do ud_repadmin.exe REPORT -DETAIL GROUP
%%g DISTRIB %UNIREP_REMOTESYS%

: Assuming suspension was temporary network interrupt, try to sync and resume

: Next line  emulates SLEEP 60
ping 1.0.0.0 -n 1 -w 60000 > %TMP%\NUL

echo Now running: ud_repadmin.exe sync -wait -verbose %UNIREP_REMOTESYS%
ud_repadmin.exe sync -wait -verbose %UNIREP_REMOTESYS%
:end
```

Reconfiguring live systems

Parent topic: Managing U2 Data Replication through the command line

Moving files to other replication groups

To improve load balancing, you can move a file from one replication group to another group in the same account. It does require making the database on the publishing sever unavailable for the time it takes to pause UniData and back up the account. You will not need to stop UniData.

Note: `ud_repadmin reconfig` can be used for different types of `repconfig` changes. In this particular case, we are considering the activity of moving a file from one replication group to another (typically for load balancing). A simple `ud_repadmin reconfig` may not suffice on an active system as `reconfig` does not update all pending logs (to change the replication group for all files impacted by the `repconfig` changes).

1. Update the `repconfig` file with your changes on both PUB and SUB servers.
2. On SUB: Stop UniData.
   a. `$UDTBIN/stopud`
3. On PUB: Activate the new `repconfig` file.
   a. `$UDTBIN/ud_repadmin reconfig`
   b. **Optional:** Use the XAdmin `reconfig` command while connected to PUB.
4. On PUB: Reset Replication.
   a. `$UDTBIN/ud_repadmin reset`
      This discards any preserved logs as we will not want to apply them on SUB.
   b. Use the XAdmin `reset` command while connected to PUB.
5. On PUB: Pause the UniData database.
   a. `$UDTBIN/dbpause`
6. On PUB: Confirm `dbpause` has successfully completed.
   a. `$UDTBIN/dbpause_status`
7. On PUB: Make a backup of all replicated files in the accounts affected by the `repconfig` changes
   If available, minimize pause time by splitting disk mirrors and making the data backup from the mirror.
8. On PUB: Allow the database to resume.
   a. `$UDTBIN/dbresume`
      All updates and processing on PUB can continue at this point.
9. On SUB: Restore the account backup made on the PUB during the `dbpause`.
10. On SUB: Reconfirm the `repconfig` file changes made on the PUB have been made to the `repconfig` file on the SUB.
11. On SUB: Remove all files/directories in REP_LOG_PATH.
    This avoids any replication log “out of sequence” errors after replication is synced.
12. On SUB: Start UniData.
    a. `$UDTBIN/startud`
13. On PUB or SUB: Issue a `sync` command if `repsys` AUTORESUME=0.
    a. `$UDTBIN/ud_repadmin sync`
    b. **Optional:** Use the XAdmin `sync` command.
14. On PUB or SUB: Confirm replication has synced and “resumed running” for all groups.
    a. Review `$UDTBIN/rm.log` or use replication monitor

Changing the numbers of replication writers

For performance reasons, you might want to increase or decrease the number of replication writer processes on a subscribing server. You can use the `ud_repadmin reconfig` command on the subscribing server and avoid interrupting application processing on the publishing server.

Note: All of the commands listed in this procedure are performed on the subscribing server.

1. Suspend Replication.

   While this step is not required, it allows the subsequent `reconfig` command to complete more quickly.

   a. `$UDTBIN/ud_repadmin suspend`
2. Confirm that replication has fully suspended.
   a. Run `$UDTBIN/showud` and confirm all `udrw` processes have exited, or examine the `$UDTBIN/rm.log` to confirm that there are “stopped successfully” messages for all replication groups.
3. Update the N_REPWRITER parameter in the UniData `include\repconfig` file to reflect the new number of replication writers you want for each replication group.

   Note: If the subscribing server is configured as a failover server, make the same changes to the `repconfig` file on the publishing server. Then if a failover occurs and the publishing server becomes a subscribing server, the failed-over subscribing server will launch the same number of replication writer processes.
4. Reconfigure replication on the subscribing server.
   a. `$UDTBIN/ud_repadmin reconfig`
5. If needed, sync replication.
   a. If `repsys` AUTORESUME=0, run `$UDTBIN/rep_admin sync`
6. Confirm the replication has synced and “resumed running” for all groups.
   a. Review `$UDTBIN/rm.log` or use replication monitor.
7. Confirm that the expected number of replication writer processes are now running.
   a. Run `$UDTBIN/showud` and review the active `udrw` processes for each replication group.

Resizing a replicated file on a subscribing server

UniData provides two commands for an administrator to use for resizing files:

- `RESIZE` - ECL command
  - For static files only at UniData 7.1 and earlier
  - For dynamic or static files at UniData 7.2
- `$UDTBIN/memresize` – shell-level command
  - For static or dynamic files at all UniData versions

At UniData 7.1 and earlier, if the VOC file of the account is being replicated, any resize command fails with errors similar to the following:

```
# $UDTBIN/memresize BANKS
Resize BANKS mod(,sep) = 0(,-1) type = -1 memory = 8000 (k)
RESIZE file BANKS to191.
```

```
errno=9: Bad file number
Tue Jul 6 11:47:05 cwd=/disk1/replication/udt/payroll 31744:write error in
U_blkwrite for file 'VOC', key '', number=30
Tue Jul 6 11:47:05 cwd=/disk1/replication/udt/payroll 31744:write error in
U_blkwrite for file 'VOC', key '', number=30
31744:write error in U_blkwrite for file 'VOC', key '', number=30
Tue Jul 6 11:47:05 cwd=/disk1/replication/udt/payroll 1:write error in
add_to_group for file 'VOC', key 'rsztempJmugUa', number=30
Tue Jul 6 11:47:05 cwd=/disk1/replication/udt/payroll 1:write error in
add_to_group for file 'VOC', key 'rsztempJmugUa', number=30
1:write error in add_to_group for file 'VOC', key 'rsztempJmugUa', number=30
Tue Jul 6 11:47:05 cwd=/disk1/replication/udt/payroll 1:write error in
U_append_strtuple for file 'VOC', key 'rsztempJmugUa', number=30
Tue Jul 6 11:47:05 cwd=/disk1/replication/udt/payroll 1:write error in
U_append_strtuple for file 'VOC', key 'rsztempJmugUa', number=30
1:write error in U_append_strtuple for file 'VOC', key 'rsztempJmugUa', number=30
'rsztempJmugUa' does not exist in VOC file.
'rsztempJmugUa' does not exist in VOC file.
'rsztempJmugUa' does not exist in VOC file.
rsztempJmugUa is not a file name in this directory
Cannot open temp file, RESIZE failed.
memresize failed.
Tue Jul 6 11:47:05 Detected calling of U_exit_proc() by udt with
its tm is still alive(udtno: 15, tm pid:409786).
Tue Jul 6 11:47:05 Detected calling of U_exit_proc() by udt with
its tm is still alive(udtno: 15, tm pid:409786).
```

Unless you are using the CONCURRENT keyword with `RESIZE` (available at UniData 7.2 or later), no other processes should be accessing the file you are resizing. On a subscribing server, you can stop all updates to a replicated file from a publishing server by suspending replication. If the file is configured as SUB_WRITEABLE (7.2 and later), be aware that a local process could still update the file.

Additionally, if the file is configured as a recoverable file (and RFS is active), the resize commands will fail if any udt (or udrw replication writer) process has the file OPEN. An error similar to the following is displayed:

```
# $UDTBIN/memresize BANKS
Resize BANKS mod(,sep) = 0(,-1) type = -1 memory = 8000 (k)
BANKS is opened by other users, try later again.
memresize failed.
```

Parent topic: Managing U2 Data Replication through the command line

Listing replication files

Use the `LIST.REPLICATION.FILE` command to list replication files for a specified account or file.

`LIST.REPLICATION.FILE` `[ALL |DICT |DATA]` *`filename`*`[,`*`subfile`*`]` `NO.PAGE`

If you specify ALL, UniData displays all active replication files in the current account. If the local account is an account-level replication account, this command displays the account as well.

If you specify a file name, UniData displays all replication files contained in the replication file table related to that file. If you do not specify ALL or *filename*, UniData obtains the file names from an active select list, or prompts for input.

The `LIST.REPLICATION.FILE` command displays the following information about each file:

- FTYPE – The file type:
  - LF – LF directory
  - DICT – Dictionary file
  - DATA – Hashed file
  - QDICT – Q-pointer dictionary file
  - QDATA – Q-pointer data file
  - QLF – Q-pointer LF directory
  - ACCT – The account-level replication account
- FILE NAME – The VOC file name in the replication file table. If the file is not in the local account, the path is also listed.
- REPLICATION – The replication definition for the file:
  - PUBLISHED – The file is published
  - SUBSCRIBED – The file is subscribed
  - EXCLUDED – The file is excluded from replication
  - INACTIVE – The file is included in replication, but is currently disabled
- PRIMARY – Whether the file is the primary replication file.
- GROUP – The name of the replication group

Parent topic: Managing U2 Data Replication through the command line

Replication files with multiple VOC pointers

If a file has multiple definitions in the VOC file, in the same or different account, UniData loads each VOC item name in different replication file entries. When a udt or tm process updates a record in the file, the process searches for the primary replication file entry in the replication file table by the i-no/d-no, beginning from the static replication file entries. If there is no match, UniData searches the dynamic replication file entries. UniData uses the first match for replication, generating a replication log in the associated replication group, and sending the VOC file name to the subscriber for publication. This VOC name is referred to as the primary replication file.

When you issue a `DELETE.FILE` command to physically delete a file, all entries that have the same i-no/d-no are disabled. If you delete a VOC pointer, symbolic link, or physical link that points to the file, only the corresponding entry in the replication table is disabled. If you disable the primary file, the next matched file is used as the primary file.

When you enable a replication file, UniData only enables the corresponding VOC file name and account ID, it does not automatically search and enable other VOC file names that point to the same file.

When you replicate a file-level command, UniData uses the corresponding replication file entry for the target file regardless if it is the primary replication file or a secondary replication file. UniData always replicates record-level commands through the primary replication file.

If a file has multiple definitions in the VOC file, define only one VOC name in the included file list so UniData knows which file name and which replication group to use with replication.

If the primary file reference VOC entry is missing on the subscriber, the updates will fail and errors will be reported to the `rm.log` and `rw.errlog` error logs.

The following is an example of the error message you could see:

```
Wed Jun 14 05:16:30 2017, RW report:
```

```
RW(0, 29523):RW File is not subscribed(0, 0); (LSN=10)Insert on
SALES.FILE ('REC6') is canceled..
Wed Jun 14 05:08:15 2017
RW(0,28959) report:
Unable to locate file SALES.FILE in VOC
```

Note: Defining a FILE=filename phrase in the `repconfig` file will force the primary file reference to be the defined file name. Otherwise, the assignment of the primary file reference is based on which VOC pointer is encountered first when loading the replication object table.

Parent topic: Managing U2 Data Replication through the command line

Error messaging

When you stop replication, detailed error messages are written to the replication log file. The following example lists messages that were written when a publisher process terminated:

```
Wed Oct 5 10:53:17 RM: sigchld received: pid=393236, dsig=9
Replication daemon on group 0(393236) is killed(9)!
Wed Oct 5 10:54:06 RM: Force stop group acct_grp(0)
Wed Oct 5 10:54:06 RM: kill publistener 454762
RM trying to kill 454762 by sig 9
RM: kill 454762
Wed Oct 5 10:54:06 RM: kill Publisher 393236
RM trying to kill 393236 by sig 9
RM: kill 393236
Wed Oct 5 10:52:48 2011: Fatal error from process 393236: Group=0,
Rep Error=2560:Publisher killed; OS Error=0:Error 0
Replication start disabling replication!
RM: Force RFS group commit.
Replication is Disabled!
RM: Pause database activity.
CheckPoint time before ForceCP: Wed Oct 5 10:55:53 2011
.CheckPoint time after ForceCP: Wed Oct 5 10:58:55 2011
.CP has been forced successfully.
DBpause successful.
RM: Force RFS group commit.
Stopped Publishing Group acct_grp successfully.
RM: Disable Replication.
RM: Resume database activity.
DBresume successful.

Execute Replication Exception Action program
/disk1/repacct/repeventtrig.sh
Publisher killed, RM disabled replication successfully.
```

Parent topic: Managing U2 Data Replication through the command line

Refreshing the subscriber database

To refresh the subscribing server, copy the database files from the publishing server to the subscribing server.

1. Run the `ud_repadmin suspend` command on either the publishing server or the subscribing server.
2. On the publishing server, run the following command to purge any pending logs on the publishing server:

```
ud_repadmin reset -noresume
```

3. On the publishing server, run the following command to pause the database:

   ```
   dbpause
   ```

   If you are sure that no applications are running on the publishing server, you can skip this step. For information about the `dbpause` command, see the *UniData Commands Reference*.
4. Copy the files that you are replicating from the publishing server to the subscribing server.
5. If you paused the database, run the `dbresume` command to resume it. For information about this command, see the *UniData Commands Reference*.
6. Complete one of the following steps:
   - If the database on the subscribing server is stopped, run the following command to start it:

     ```
     startud -i
     ```
   - If the database is already running, run the following command to reconfigure the replication configuration:

     ```
     ud_repadmin reconfig
     ```
7. If the AUTORESUME parameter is not enabled on the subscribing server, run the following command on the publishing server or on the subscribing server:

   ```
   ud_repadmin sync
   ```

   If AUTORESUME is enabled, synchronization starts automatically when the database starts.

Related topics

- reconfig command, on page 82
- suspend command, on page 75
- sync command, on page 77

Parent topic: Managing U2 Data Replication through the command line

Error handling

With replication disablement, the runtime error handling of U2 Data Replication changes. If U2 Data Replication incurs a fatal error, U2 Data Replication is disabled and continues running UniData in `DISABLE` mode rather than crashing the publishing server.

The following system errors will cause the publishing server to be disabled:

- A udt or tm process encounters a log extension file (LEF) error when handling replication logs.
- The publishing server cannot access the log reserve file (LRF).
- The publishing process is stopped or abruptly ends.

Applications continue to run in DISABLE mode.

We recommend that you write and deploy a U2 Data Replication exception action script. If U2 Data Replication is disabled due to an unexpected event, the replication manager executes the script.

On a subscribing server, when a subscribing process or replication writer process encounters an LEF or LRF failure, U2 Data Replication is suspended.

Parent topic: Managing U2 Data Replication through the command line

Recovering the system

At UniData 7.3, the -D option has been added to the `startud` command. This starts UniData and U2 Data Replication in DISABLE mode.

If the Recoverable File System (RFS) is enabled and you are starting UniData after a system crash, the crash recovery behaves differently between the DISABLE mode and the ENABLE mode. Regardless if the system is in DISABLE mode because of starting UniData with the -D option or disabled because of a system crash, full RFS recovery is performed, and before image and after image log files are applied.

In ENABLE mode, the restart process applies only the before image log files and generates replication recovery logs from the after image logs. If a failover occurred, U2 Data Replication uses the replication recovery logs in the synchronization process. Otherwise the Replication Writer processes apply the replication recovery logs to the database.

The DISABLE mode supports recovering from a failed recovery process. If you try to recover the system when U2 Data Replication is in ENABLE mode and the recovery fails, you can still try to recover the system in DISABLE mode.

The following steps describe the steps for the second recovery if the first recovery fails:

1. The init_sm process creates the Replication Manager process. If you specify the -D option when starting UniData, the Replication Manager disables U2 Data Replication.
2. The init_sm process starts the Restart process for recovery.
3. The Restart process applies the before image logs to the database.
4. If U2 Data Replication is disabled, the Restart process applies the after image log files to the database. Otherwise, the Restart process generates replication recovery logs from the after image logs.
5. The Restart process resets the before image and after image logs.
6. The init_sm process stops the Replication Manager process and starts the sm daemon.
7. A new Replication Manager is started. If U2 Data Replication is disabled, the Replication Manager writes the replication recovery logs to the database. If U2 Data Replication is enabled, the Replication Manager starts a replication procedure that either generates a failover log from the replication recovery logs if the replication has failed over, or reapplies the replication recovery logs to the database if the replication has not failed over.

Parent topic: Managing U2 Data Replication through the command line

Monitoring U2 Data Replication through the command line

Use the REPLOGGER to monitor U2 Data Replication at the command line.

Parent topic: Managing U2 Data Replication through the command line

Monitoring replication with REPLOGGER

REPLOGGER is a BASIC program that allows you to monitor replication. REPLOGGER collects information from the replication system using the replication administration reptool and writes the information to a sequential log file. The information logged by REPLOGGER describes how the replication system performs over a period of time. Using REPLOGGER helps you understand and analyze how your replication system responds to changing workloads. REPLOGGER can produce

a large amount of data, so to aid in analyzing and visualizing the data, a Java tool called the U2 Replication Analyzer is available on github to graph the result sets.

Note: You must be a super user to run REPLOGGER. REPLOGGER should be run on the subscriber.

1. At the ECL or TCL prompt, enter `REPLOGGER`.

   The REPLOGGER BASIC program starts and prompts you to enter the following information:

   - the number of replication groups; if this is a single machine replication, this number should be doubled because it counts both publisher and subscriber
   - whether to clear previous logs
   - the capturing interval time, in seconds
   - the maximum log file size, in megabytes

   You can alternatively enter the answers to the prompts directly in the command line when starting REPLOGGER, as shown in the following example:

   ```
   REPLOGGER 1 y 2 100
   ```

2. After you enter the prompts, press Enter.

   The REPLOGGER BASIC program starts. The results are written to the `rep_monitor.log` file in the current account.

   The following is an example of REPLOGGER executed in a Linux environment.

```
Current working directory is /disk1/u2test/test81_97837/testRFS.
:REPLOGGER
Replication Logging Tool. Ver 2.1

Program Starts: 12:02:03 04 Nov 2015

Please enter the number of replication groups:
?1
Clear the previous logs:(y/n)
?Y
Old logs have been cleared.
Enter the capturing interval in seconds:
?5
Enter the maximum log file size in Mega Bytes(MBs):
?100
Replication Logging Tool. Ver 2.1

Program Starts: 12:02:03 04 Nov 2015

Waiting for data...

Press Ctrl+C to exit.

[root@den-ci125 testRFS]# cat rep_monitor.log
GROUP.NO DATE.TIME PUB.DONE SUB.GOT SUB.AVAIL SUB.DONE LIEFSTART
LEFNO.START LEFNO.LAST LEFPOS.START LEFPOS.END USER.TIME(%)
 SYS.TIME(%) TCA.MODULO
TOTAL.PG TCA.FILE.ACCESS TCR.CREATE TCR.DROP
 0 10:36:22 14 SEP 2015 44935 44935 44936 44523 0 1 1 0 0 2 21 2457
 4096 0 0 0
 0 10:36:25 14 SEP 2015 49306 49306 49307 46947 0 1 1 0 0 1 25 2457
 4096 0 0 0
 0 10:36:28 14 SEP 2015 52541 52541 52542 48404 0 1 1 0 0 1 24 2457
 4096 0 0 0
 0 10:36:32 14 SEP 2015 54991 54991 54992 49544 0 1 1 0 0 1 28 2457
 4096 0 0 0
```

```
 0 10:36:35 14 SEP 2015 57275 57275 57276 50600 0 1 1 0 0 2 29 2457
 4096 0 0 0
 0 10:36:38 14 SEP 2015 59157 59157 59158 51464 0 1 1 0 0 2 31 2457
 4096 0 0 0
 0 10:36:41 14 SEP 2015 60001 60001 60002 52303 0 1 1 0 0 1 33 2457
 4096 0 0 0
 0 10:36:44 14 SEP 2015 60001 60001 60002 53059 0 1 1 0 0 1 36 2457
 4096 0 0 0
 0 10:36:47 14 SEP 2015 60001 60001 60002 53854 0 1 1 0 0 1 46 2457
 4096 0 0 0
 0 10:36:50 14 SEP 2015 60001 60001 60002 54737 0 1 1 0 0 1 46 2457
 4096 0 0 0
 0 10:36:54 14 SEP 2015 60001 60001 60002 55531 0 1 1 0 0 2 46 2457
 4096 0 0 0
 0 10:36:57 14 SEP 2015 60001 60001 60002 56312 0 1 1 0 0 2 46 2457
 4096 0 0 0
 0 10:37:00 14 SEP 2015 60001 60001 60002 57029 0 1 1 0 0 1 46 2457
 4096 0 0 0
 0 10:37:03 14 SEP 2015 60001 60001 60002 57748 0 1 1 0 0 1 47 2457
 4096 0 0 0
 0 10:37:06 14 SEP 2015 60001 60001 60002 58414 0 1 1 0 0 1 46 2457
 4096 0 0 0
 0 10:37:09 14 SEP 2015 60001 60001 60002 59091 0 1 1 0 0 1 48 2457
 4096 0 0 0
 0 10:37:12 14 SEP 2015 60001 60001 60002 59705 0 1 1 0 0 1 18 2457
 4096 0 0 0
 0 10:37:15 14 SEP 2015 60001 60001 60002 60001 0 1 1 0 0 0 0 2457
 4096 0 0 0
 0 10:37:19 14 SEP 2015 60001 60001 60002 60001 0 1 1 0 0 0 0 2457
 4096 0 0 0
 0 10:37:22 14 SEP 2015 60001 60001 60002 60001 0 1 1 0 0 0 0 2457
 4096 0 0 0
 0 10:37:25 14 SEP 2015 60001 60001 60002 60001 0 1 1 0 0 0 0 2457
 4096 0 0 0
 0 10:37:28 14 SEP 2015 60001 60001 60002 60001 0 1 1 0 0 0 0 2457
 4096 0 0 0
 0 10:37:31 14 SEP 2015 60001 60001 60002 60001 0 1 1 0 0 0 0 2457
 4096 0 0 0
 0 10:37:34 14 SEP 2015 60001 60001 60002 60001 0 1 1 0 0 0 0 2457
 4096 0 0 0
 0 10:37:37 14 SEP 2015 60001 60001 60002 60001 0 1 1 0 0 0 1 2457
 4096 0 0 0
 0 10:37:40 14 SEP 2015 60001 60001 60002 60001 0 1 1 0 0 0 0 2457
 4096 0 0 0
 0 10:37:43 14 SEP 2015 60001 60001 60002 60001 0 1 1 0 0 0 0 2457
 4096 0 0 0
[root@den-ci125 testRFS]#
```

Checking replication status with $UDTBIN/ud$UVBIN/uv_repadmin

In addition to using the replication status monitor, you can review the replication status on your systems with the `$UDTBIN/ud_repadmin` command.

If replication is running properly on the publishing server, you should see a REP_RUNNING status for each replication group you have configured.

If replication is running properly on the subscribing server, you should see a status of SUB_RUNNING for each replication group.

Note: You can produce reporting details for each replication group through the XAdmin report interface, but it is a bit cumbersome. By default, only the overall replication system status is displayed.

Reviewing the replication log

You'll want to review the `rm.log` file in the `UDTBIN` directory as your primary source to check for errors or to view the administration activity that has occurred for replication. In addition, a `rm.log` is used to record problems. To simplify problem solving, all `rm.log` messages are also recorded in `rm.log` – giving you a single log to review on each of your systems.

Displaying active processes

If a replication daemon process exits abnormally, there is usually an entry in the `$UDTBIN/rm.log`, but not always (for example, if a process core dumps, it has no ability to write an error to the `rm.log`). Run `$UDTBIN/showud` on all systems and review the output to confirm that the daemons required for your configuration are present.

On a publishing server there should be:

- One `repmanger` process.
- One `udpub` process for each replication group.

If replication is successfully synced to a subscribing server, there should be:

- One `publistener` process for each replication group for each synced subscribing server. For instance, if there are two subscribing server DISTRIBUTIONs defined for a replication group, there will be two `publistener` processes spawned for that group when both subscribers are connected and active.

Continuing with the systems and `repconfig` used throughout this guide:

```
# $UDTBIN/showud
USER PID TIME COMMAND root 463040 0:00 /disk1/ud72/bin/aimglog 0 18945
root 729118 0:00 /disk1/ud72/bin/aimglog 1 18945
root 524378 0:00 /disk1/ud72/bin/bimglog 2 18945
root 553068 0:00 /disk1/ud72/bin/bimglog 3 18945
root 827556 0:00 /disk1/ud72/bin/cleanupd -m 10 -t 20
root 819258 0:01 /disk1/ud72/bin/cm 18945
root 356410 0:00 /disk1/ud72/bin/publistener 11 0 0 1 1 0 0
root 360564 0:00 /disk1/ud72/bin/publistener 11 0 0 1 1 1 0
root 430142 0:00 /disk1/ud72/bin/publistener 11 0 0 1 1 3 0
root 782520 0:00 /disk1/ud72/bin/publistener 11 0 0 1 1 2 0
root 737298 0:00 /disk1/ud72/bin/repmanager
root 332006 0:00 /disk1/ud72/bin/sbcs -r
root 725212 0:00 /disk1/ud72/bin/sm 60 124
root 811076 0:00 /disk1/ud72/bin/smm -t 60
root 856198 0:00 /disk1/ud72/bin/udpub -g0
root 467072 0:00 /disk1/ud72/bin/udpub -g1
root 684264 0:00 /disk1/ud72/bin/udpub -g2
root 770158 0:00 /disk1/ud72/bin/udpub -g3 root
753810 0:00 /disk1/unishared/unirpc/unirpcd
```

On Windows systems, you can run the task manager, sort the process listing by image name, and find the database process names to review. Sets of processes (like `udpub` or `publistener`) are listed together and are fairly easy to review. On a subscribing server, there should be:

- One `repmanger` process.

If replication is synced to a publishing system and running, there should be:

- N_REPWRITER `udrw` processes for each replication group.
- One subscriber process for each replication group: `udsub`.

The `showud` output below is from a subscribing system that is synced and running normally.

- The server is configured with four replication groups.
- The `repmanger`, replication writer (`udrw`), and subscriber processes are highlighted.
- There is one `repmanger` process for the server.
- There is one subscriber process for each of the four groups configured.
- There is one set of replication writer processes for each of the four groups
  - There are four `udrw` processes for each group.
  - The replication group number is shown in the "-g n" option on each `udrw` process.
  - The number of `udrw` processes per group is configured by setting the N_REPWRITER parameter in the `repconfig` file.

```
# $UDTBIN/showud
USER PID TIME COMMAND root 729114 0:00 /disk1/ud72/bin/aimglog 0 10055
root 970880 0:00 /disk1/ud72/bin/aimglog 1 10055
root 696528 0:00 /disk1/ud72/bin/bimglog 2 10055
root 712768 0:00 /disk1/ud72/bin/bimglog 3 10055
root 827640 0:06 /disk1/ud72/bin/cleanupd -m 10 -t 20
root 770204 0:04 /disk1/ud72/bin/cm 10055
root 753800 0:00 /disk1/ud72/bin/repmanager
root 737482 0:00 /disk1/ud72/bin/sbcs -r
root 262350 0:00 /disk1/ud72/bin/sm 60 25984
root 786660 0:00 /disk1/ud72/bin/smm -t 60
root 245882 0:00 /disk1/ud72/bin/udrw -g 1 -s 4
root 327856 0:00 /disk1/ud72/bin/udrw -g 0 -s 4
root 331828 0:00 /disk1/ud72/bin/udrw -g 2 -s 4
root 356412 0:00 /disk1/ud72/bin/udrw -g 3 -s 4
root 389312 0:00 /disk1/ud72/bin/udrw -g 2 -s 4
root 462986 0:00 /disk1/ud72/bin/udrw -g 2 -s 4
root 475366 0:00 /disk1/ud72/bin/udrw -g 0 -s 4
root 577574 0:00 /disk1/ud72/bin/udrw -g 2 -s 4
root 655582 0:00 /disk1/ud72/bin/udrw -g 3 -s 4
root 839782 0:00 /disk1/ud72/bin/udrw -g 3 -s 4
root 872540 0:00 /disk1/ud72/bin/udrw -g 0 -s 4
root 897178 0:00 /disk1/ud72/bin/udrw -g 0 -s 4
root 901168 0:00 /disk1/ud72/bin/udrw -g 1 -s 4
root 1003708 0:00 /disk1/ud72/bin/udrw -g 1 -s 4
root 1024146 0:00 /disk1/ud72/bin/udrw -g 3 -s 4
root 1081500 0:00 /disk1/ud72/bin/udrw -g 1 -s 4
root 520308 0:07 /disk1/ud72/bin/udsub 4 3600 0
root 802968 0:07 /disk1/ud72/bin/udsub 4 3600 0
root 815332 0:07 /disk1/ud72/bin/udsub 4 3600 0
root 999492 0:07 /disk1/ud72/bin/udsub 4 3600 0
root 716826 0:00 /disk1/unishared/unirpc/unirpcd
```

Note: These processes are created via UniRPC when a connection to a subscribing server is established during the sync process. They receive replication logs from the publishing server for a replication group and write the logs to the local replication buffer. The `udrw` processes read the logs from the buffer and update the subscriber database files. If errors occur, the processes may also create files named `udsubn.errlog` in the `$UDTBIN` directory.

On Windows systems, you can run the task manager, sort the process listing by image name, and find the database process names to review. Sets of processes (like `udrw` processes) are listed together and are fairly easy to review.

Viewing replication activity

An undocumented diagnostic tool called uvreptool is designed for use by UniData software engineers to peer into replication shared memory structures. It was never intended to be used by database administrators. However, it is reasonably easy to use to look at some log counters maintained in memory to see if replication is perking along. You may find the XAdmin interface to `reptool` easier to use.

Note: One of the benefits of `reptool` is its ability to create a memory dump that can be replayed by the tool. This `reptool` dump is included in all of the `udtdiag` script dumps. It gives U2 Support and Development an opportunity to look at a snapshot of replication internal structures from your system at a later time (typically after you have already taken remedial steps on your system). The syntax to produce a `reptool` dump on disk is: `$UDTBIN/reptool -s some.dump.name`

To look at the log counters:

- Launch the tool from the UNIX or Windows shell.
- Select **2: Replication Groups**.
- Choose a group to display (0 in the example below).
- Select **3: Replication Buffer Pointers**.

```
# $UDTBIN/reptool
Replication System Information
==============================
1: Replication Configuration
2: Replication Groups
3: Replication Semaphores
4: Replication Waiting Mechanism
5: Replication IPCs
6: Replication TCRs
7: Replication LSNs
0: Quit (or back to previous menu)
Selection = 2
Group 0: inventory_acct_g0
Group 1: inventory_file_g1
Group 2: payroll_acct_g2
Group 3: payroll_file_g3
Which group to display (0 -3): 0
Information about Group inventory_acct_g0(0)
============================================
1: General Group Info
2: Objects in Group
3: Replication Buffer Pointers
```

```
4: Distributions
5: Waiting Lists
6: Logs In Replication Buffer
7: Last RM Command
8: RW process flags
9: Object Updates Report
0: Quit (or back to previous menu)
Selection = 3
RB pointers of Group inventory_acct_g0(0):
infoBufPosition: (start=826, end=826)
logBufPosition: (start=379185, end=379185)
LIEFStart: 0
LEFno: (first=1,last=1)
LEFPosition: (start=0, end=0)
LRFInfoFno: (first=1,last=1)
LRFBodyFno: (first=1, last=1)
LRFBodyPosition: (Start=0, End=0)
LRFLSNs: (first=0,next=0)
infoBufStartLSN: 9019
currentReadLSN: 9019
localDoneLSN: 9018
nextAvailLSN: 9019
localGotLSN: 0
cpLSN(valid): 9018(9018)
TP_resolved: 0
RFS_cp_no: 3
```

The two most interesting counters in this display are:

- nextAvailLSN
- localDoneLSN

LSN is an abbreviation meaning Log Sequential Number.

The nextAvailLSN is simply a “next ID counter”. As each replication log is created on the publisher, it is assigned a sequential numeric ID or key.

If updates are being made to files in the group you are examining, you should expect to see that number increase each time you redisplay the replication buffer pointers. After this display, the submenu is redisplayed. You can enter “3” repeatedly to view activity in this group. If you want to view a different group, enter “0” to drop back one menu level and choose another group.

On a publisher, localDoneLSN is incremented when an update is committed to the local data files. If this is not increasing, there may be a problem with transaction logging or the replication subsystem.

If you want to confirm that logs are being passed to the subscriber and processed, watch the remoteDoneLSN counter that is displayed in the menu selection “4:Distributions” (as shown below). If all processing is caught up and complete, this should be one less than nextAvailLSN.

```
Information about Group inventory_acct_g0(0)
============================================
1: General Group Info
2: Objects in Group
3: Replication Buffer Pointers
4: Distributions
5: Waiting Lists
6: Logs In Replication Buffer
7: Last RM Command
8: RW process flags
9: Object Updates Report
0: Quit (or back to previous menu)
Selection = 4
Total 1 distributions in group inventory_acct_g0(0)
```

```
Group 0, Name 'inventory_acct_g0', Distribution 0:
replication name = primary->standby
reptype = STANDBY IMMEDIATE SUBSCRIBING
distrib_status = REP_RUNNING
pre_status = REP_SYNCING
next_status = NULL
Sync flags = 0x0()
returnCmd = 0
reason(19): Sync Succeeded
returnReason(0):
conn_id = 0
syncer_pid = -1
listener_pid = 14614652
lastSyncLSN = 0
remoteGotLSN = 9018
returnGotLSN = 9018
remoteAvailLSN = 9018
remoteDoneLSN = 9018
packet_sent = 395
packet_rcvd = 393
total_data = 379185
last_sent = 395
```

The remoteDoneLSN should be climbing along with the localDoneLSN (previous screen). It may trail local- DoneLSN a bit, but it should not be very far behind on a reasonably tuned system. The remoteDoneLSN line in the distribution display on a publishing system represents replication logs that have been committed to the database on the subscribing system.

Here is the corresponding XAdmin view of `reptool`:

1. Define your servers and connect to the publisher.
2. Choose the **Replication** option from the left menu.
3. Select **Diagnosis Utility**.
4. Select the **Groups** tab.
5. Select **Buffer Pointers**.

To watch activity, click the **Refresh** button.

It is easy to change which group you are viewing by using the up and down errors in the **Groups** box.

As you can see, for the purposes of demonstrating the utilities consistently, there is no activity occurring on the demonstration publishing system. The nextAvailLSN is not changing in subsequent screen displays.

In order to complete the picture (and answer the question, “Is replication happening?”), you need to examine the buffer pointers on the subscribing system as well. The same logic applies. nextAvailLSN should be climbing and localDoneLSN should not be far behind.

Note: If the replication writer (`udrw`) processes are overly busy, you may see a lag in localDoneLSN on a subscribing server.

You may find the XAdmin interface easier to use than the shell-level `$UDTBIN/reptool` command.

Chapter 5: Managing U2 Data Replication through XAdmin

To access the replication tools, in XAdmin, select the server you want to use. From the **Admin Tasks** pane, expand **Replication** (**U2 Replication** in UniVerse), then double-click **Configuration**.

The Replication pane opens and displays the database type that you are using.

There are four replication tool options available for use in XAdmin:

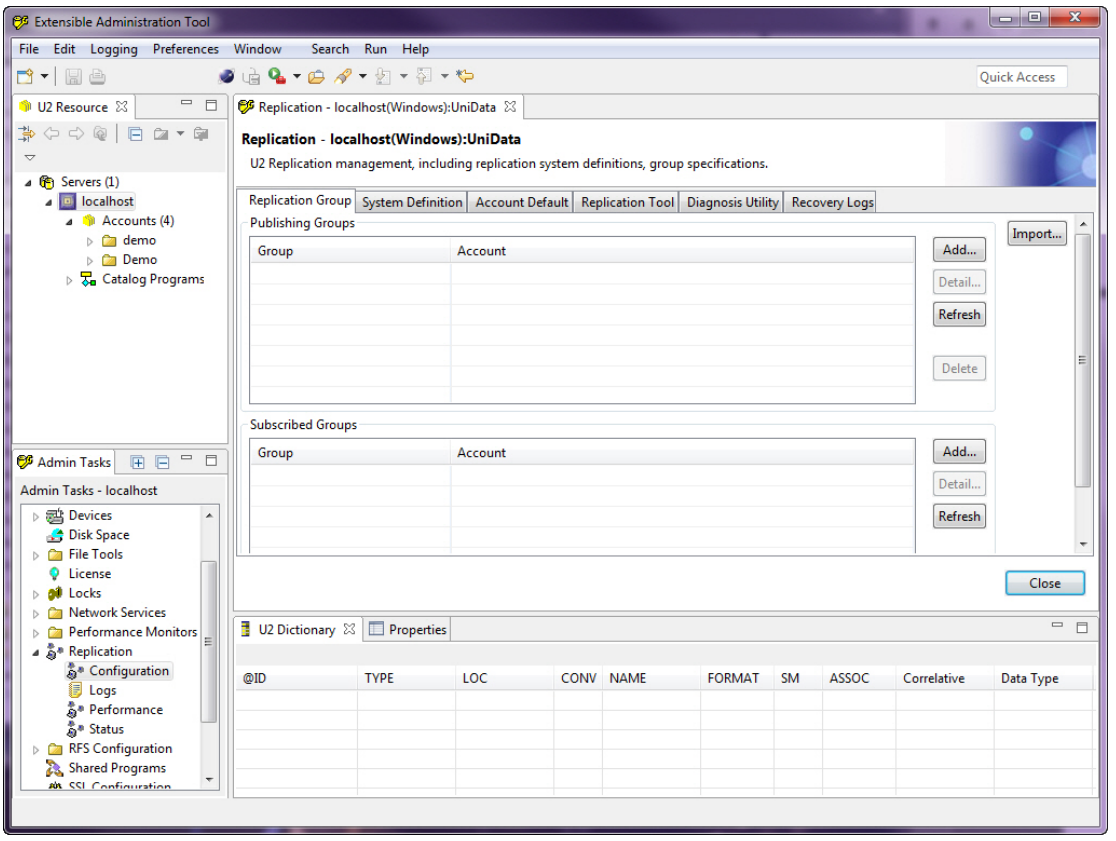
- **Configuration**

  Note: The **Recovery Logs** tab in this option is only available with UniData.

- **Logs**
- **Performance**
- **Status**

Configuration

To configure U2 Data Replication, select **Configuration** from the Admin Tasks list. In the following example, UniData is the displayed database.



The following tabs are available when the **Configuration** option is selected:

- Replication group
  Use XAdmin to add publishing and subscribing groups.
- System definition
  Use the **System Definition** tab to define replication servers in XAdmin.
- Account default
  Use the **Account default** tab to view files that are included and excluded.
- Replication tool
  Use the **Replication Tool** tab to start replication from XAdmin.
- Diagnosis utility
  The diagnosis utility is used by U2 support for diagnosing problems encountered with U2 Data Replication. With the utility, you can view information such as the current configuration, groups, semaphores, and other general information that can help you troubleshoot any issues you have with replication.
- Recovery logs
  The replication recovery log is a UniData file located in `$UDTHOME/sys`. This file records replication recovery. The resynchronization of a publishing group on a system running RFS generates a record in the replication recovery log that records the publishing group recovery. The database also writes the logs representing missing transactions during recovery when resynchronizing after failing over to a system running immediate replication to this file.

Replication group

Use XAdmin to add publishing and subscribing groups.

- Adding a publishing group
  After you have defined the replication systems, add a replication group to the publisher.
- Adding a subscribing group
  After you have added a publishing group, add a subscribing group.
- Editing replication group definitions
  You can edit a replication group definition without stopping and restarting the database.

Parent topic: Configuration

Adding a publishing group

After you have defined the replication systems, add a replication group to the publisher.

1. In XAdmin, from the U2 Resource pane, select the publisher server. From the Admin Tasks view, expand **Replication**, and then double-click **Configuration**.
   The Replication pane displays.
2. Click the **Replication Group** tab.
3. To add a publishing group, from the **Publishing Groups** area, click **Add**.
4. In the dialog box that displays, enter the ID of the publishing group in the **Group ID** field.
5. Specify the path of the publishing account by selecting the account where the files you want to publish reside from in the **Account** list.
6. To define a replication level, select the level of replication you want from the **Level** list. You can select either **ACCOUNT** or **FILE** replication.
   If you select **ACCOUNT**, replication will occur at the account level for the account you selected. This means every file in that account will be replicated.

If you select **FILE**, only the files selected in the Files area will be replicated for the account you selected.

7. To add files to the publishing group, in the Files area, click **Add**.

   A dialog box displays with all the files for your account listed. If you selected **ACCOUNT** in step 5, you can still select specific files for reasons such as making them sub-writable. If you selected **FILE** in step 5, select the specific files that you want to publish.

   When you are done adding files, click **Finish**.
8. By default, both the Data and the Dict cells of the file are selected. If you do not want to publish the data portion of the file, clear the **Data** check box (cell). If you do not want to publish the dictionary portion of the file, clear the **Dict** check box (cell).
9. **UniData only.** If you want to update the threshold level of any files you selected, select the cell for that file in the **Field Update Threshold** column, and then enter `Y` or `N`.
10. **UniData only.** If you are using account-level replication, select files to exclude. To specify any files that are excluded, in the **Excluded Files** area, click **Add**.

    Select the specific files that you want to exclude, and click **Finish**.
11. Set any of the configuration parameters necessary for your environment in the **Configuration** area.

    For detailed information about these configuration parameters, see Configuration phrases, on page 64.
12. Set any of the configuration parameters necessary for your environment in the **RW Configuration** area. The following parameters are available:

Parameter	Description
RW_REEVALUATE_VF	The database evaluates indexed virtual fields on the publishing server before writing the results to the database, then transfers these values to the subscribers along with the record. You can choose to use these values directly from the publisher, or have the replication writer processes reevaluate these values. The following are valid values: ▪ 0 ▪ 1 By default, the value is 1. The replication writer process reevaluates the virtual field on the subscribing server. If you want to use the values without reevaluating them, set this parameter to 0. **Note:** Most administrators set this parameter to 0 so they are certain that the index values stored on the subscriber match those on the publisher. Additionally, the performance of replication is better, as the replication writer processes don't have to perform the work required to calculate the values of the virtual field formulas.
RW_SKIP_TRIGGER	If this value is set to 1, the default, the replication writer process ignores a trigger, even if a trigger exists on the subscribing file. If you do not want to ignore the trigger, set this value to 0. **Note:** Setting this parameter to 0 might result in unwanted updates on the subscribing server, such as attempting to update a read-only file.

Parameter	Description
RW_IGNORE_ERROR	By default, when a replication writer process encounters inconsistency in a replication log, it writes an error message to the `rm.errlog` and `rm.log` file and continues processing. The following are valid values: ▪ 0 ▪ 1 By default, the value is 0. At the default setting, the database will suspend replication if it finds inconsistency in a replication log.
RW_VERIFY_PARTITION	If this value is set to 1, the replication writer processes verify the partition algorithm of distributed files part files. If the publishing server and the subscribing server have exactly the same definition for distributed files, you can set this value to 0 to skip partition verification. The default value is 1.
RW_UID	**UNIX only.** The user ID of the replication writer process. By default, each replication writer starts as root. If you define RW_UID, the replication writer process changes the user ID to the one you specify.
RW_GID	**UNIX only.** The group ID of the replication writer process. If you define RW_GID, the replication writer process changes the group ID to the one you specify.

13. Click **Finish**.

Parent topic: Replication group

Adding a subscribing group

After you have added a publishing group, add a subscribing group.

1. In XAdmin, from the U2 Resource pane, select the subscriber server. From the Admin Tasks view, expand **Replication**, and then double-click **Configuration**.

   The Replication pane displays.
2. Click the **Replication Group** tab.
3. Decide whether to manually add a subscribing group or import the group configuration from the publishing server. To add one manually, continue to the next step. To import the configuration, click **Import**.
   a. In the dialog box that displays, select the server where you set up the publishing group.
   b. Select the version number.
   c. Select whether you want to merge or replace the subscribing configuration file with the remote one.
   d. Click **Finish**.

      The configuration is imported, and the values are automatically populated in the dialog box that displays.
   e. Verify that your configuration is correct, and click **Finish**.

      You have successfully added a subscribing group, and you can move on to Excluding files from replication.
4. To add a subscribing group, from the **Subscribing Groups** area, click **Add**.

5. In the dialog box that displays, select the publishing server from which this subscribing server is receiving data from the **From System** list.
6. Select the group ID from the publishing server in the **Group ID** list.

   When you select the group ID, XAdmin populates the file list and configuration parameters from the same group ID on the publishing server.
7. Select the path of the account to which you want to replicate data in the **Into Account** field. XAdmin automatically populates the **Account Path** field.
8. Click the file you want to receive. You can select multiple files by clicking the files while holding down the CTRL key.
9. By default, both the Data and the Dict cells of the file are selected. If you do not want to receive the data portion of the file, clear the **Data** check box (cell). If you do not want to receive the dictionary portion of the file, clear the **Dict** check box (cell).
10. If you want to be able to update the file on the subscribing server, select the SUB_WRITEABLE check box (cell).
11. **UniData only.** If you want to update the threshold level of any files you selected, select the cell for that file in the **Field Update Threshold** column.
12. **UniData only.** If you are using account-level replication, select files to exclude. To specify any files that are excluded, in the **Excluded Files** area, click **Add**.

    Select the specific files that you want to exclude, and click **Finish**.
13. In the **Distributions** area, click the value in the **Type** field, then select the ellipsis button (...) that displays.
14. In the dialog box that displays, select the type of replication you want and the account for the subscriber.

    When choosing Realtime or Immediate subscribing, the subscribing server can optionally be defined as a standby server. This can be done by selecting the **Hot Standby** check box. A standby server can also be optionally configured for delayed standby replication.

    When selecting the **Hot Standby** check box, a **Delayed Time Period** field displays. Enter the amount of time you want to delay the updates on the subscriber in minutes. When using delayed standby replication, the updates are immediately sent to the subscriber but are not applied until the delayed time period has elapsed.

    Click **Finish**.
15. Set any of the configuration parameters necessary for your environment in the **Configuration** area.

    For detailed information about these configuration parameters, see Configuration phrases, on page 64.
16. Set any of the configuration parameters necessary for your environment in the **RW Configuration** area. The following parameters are available:

Parameter	Description
N_REPWRITER	The number of replication writer processes for the replication group. Specify more than one replication writer per group, but do not specify a value that is significantly higher than the number of CPUs on the subscribing server. On a subscribing server, NUSERS parameter in the `udtconfig` must be large enough to accommodate the total number of replication writer processes configured for all groups.

Parameter	Description
RW_REEVALUATE_VF	The database evaluates indexed virtual fields on the publishing server before writing the results to the database, then transfers these values to the subscribers along with the record. You can choose to use these values directly from the publisher, or have the replication writer processes reevaluate these values. The following are valid values: ▪ 0 ▪ 1 By default, the value is 1. The replication writer process reevaluates the virtual field on the subscribing server. If you want to use the values without reevaluating them, set this parameter to 0. **Note:** Most administrators set this parameter to 0 so they are certain that the index values stored on the subscriber match those on the publisher. Additionally, the performance of replication is better, as the replication writer processes don't have to perform the work required to calculate the values of the virtual field formulas.
RW_SKIP_TRIGGER	If this value is set to 1, the default, the replication writer process ignores a trigger, even if a trigger exists on the subscribing file. If you do not want to ignore the trigger, set this value to 0. **Note:** Setting this parameter to 0 might result in unwanted updates on the subscribing server, such as attempting to update a read-only file.
RW_IGNORE_ERROR	By default, when a replication writer process encounters inconsistency in a replication log, it writes an error message to the `rm.errlog` and `rm.log` file and continues processing. The following are valid values: ▪ 0 ▪ 1 By default, the value is 0. At the default setting, the database will suspend replication if it finds inconsistency in a replication log.
RW_VERIFY_PARTITION	If this value is set to 1, the replication writer processes verify the partition algorithm of distributed files part files. If the publishing server and the subscribing server have exactly the same definition for distributed files, you can set this value to 0 to skip partition verification. The default value is 1.

Parameter	Description
RW_UID	**UNIX only.** The user ID of the replication writer process. By default, each replication writer starts as root. If you define RW_UID, the replication writer process changes the user ID to the one you specify.
RW_GID	**UNIX only.** The group ID of the replication writer process. If you define RW_GID, the replication writer process changes the group ID to the one you specify.
RW_QUEUESZ	Defines the size of the intelligent write queue. The intelligent write queue is used to optimize update locks and record updates on the subscriber when repeated updates of the same record are detected.

17. Click **Finish**.

Parent topic: Replication group

Editing replication group definitions

You can edit a replication group definition without stopping and restarting the database.

1. In XAdmin, from the U2 Resource pane, select the subscriber server. From the Admin Tasks view, expand **Replication**, and then double-click **Configuration**.
2. From the **Replication Group** tab, select the group to update, and then click **Detail**.
3. Edit the group definition, and then click **Finish**.

   You must synchronize the reconfigured group from its publisher or to its subscriber after making your changes.
4. To activate the change, enter the `ud_repadmin reconfig` command.

Parent topic: Replication group

System definition

Use the **System Definition** tab to define replication servers in XAdmin.

- Defining replication servers

  Once you have opened the Replication pane, define the replication servers. This task needs to be performed on the publisher and subscriber servers.

Parent topic: Configuration

Defining replication servers

Once you have opened the Replication pane, define the replication servers. This task needs to be performed on the publisher and subscriber servers.

1. In XAdmin, from the U2 Resource pane, select the subscriber server. From the Admin Tasks view, expand **Replication**, and then double-click **Configuration**.
2. To define replication systems, click the **System Definition** tab.
3. To add a new system, click **Add**.
4. In the Replication Server Definition dialog box, complete the following fields:

Field	Description
System ID	Enter a unique ID for the system.
Host Name	The name or IP address of the host. Select the **DHCP** check box to the **Host Name** field to check if the system is enabled for DHCP.
Version	Version of the U2 server
Auto Resume	Determines whether U2 Data Replication will automatically sync and resume running when the database starts or after completing a repconfig. The default setting is **Yes** and is typically the desired behavior for most users. Set to **No** only if a specific action must be completed on the server that needs to be done prior to starting replication. Once the action has been completed, manually issue a sync to start replication.
Sync Interval	Interval of synchronization in minutes.
Connect Authorization	Select the **Connect Authorization** check box to check if the remote system should be authorized.
Timeout	The number of seconds the system waits before suspending replication if no packets are received from the remote system.
Exception Action	Enter the name of the exception action file, or click **Browse** to locate the file.
Failover Action	Enter the name of the failover action file, or click **Browse** to locate the file.
Account Definitions	Define an account for the replication system. To define an account for replication, click **Add**. **Note:** You should only complete this field if the account definition has not already been defined in the `UD.ACCOUNT` file.

5. If you clicked add, in the dialog box that displays, complete the following fields:

Option	Description
Account Name	Name of the account.
Account Path	Full path to the account. The account information is written to the `repsys` file. You can refer to the account name in other areas of replication without having to specify the full path.

6. Click **Finish**.
7. If you selected **Connection Authorization** previously, select the system and click **Set Connection** to define the connection user name and password to the publishing server.
8. In the dialog box that displays, enter the login name and the corresponding password. Re-enter the password. Click **Finish**.

Parent topic: System definition

Account default

Use the **Account default** tab to view files that are included and excluded.

- Excluding files from replication
  You can view the account defaults to see which files are automatically included (replicated) and excluded.

Parent topic: Configuration

Excluding files from replication

You can view the account defaults to see which files are automatically included (replicated) and excluded.

1. In XAdmin, from the U2 Resource pane, select a server. From the Admin Tasks view, expand **Replication**, and then double-click **Configuration**.
2. Click the **Account Default** tab on the Replication pane.
   The Files area lists the files that are included in replication. The Excluded Files area lists the files that are automatically excluded.
3. **Optional:** If you want to include or exclude a file, click **Add** next to Files or Excluded Files.
   In the dialog box that displays, enter the file name, and then select any applicable specifications
   - **Data**
   - **Dict**
   - **SUB_WRITEABLE**
   - **Field Update Threshold** (UniData only)
4. When you are done, click **Finish**.

Parent topic: Account default

Replication tool

Use the **Replication Tool** tab to start replication from XAdmin.

- Starting replication
  After you have defined the replication systems and added both a publishing and subscribing group, you can start replication.
- Enabling or disabling U2 Data Replication through XAdmin
  Enable U2 Data Replication through XAdmin using the Replication pane.

Parent topic: Configuration

Starting replication

After you have defined the replication systems and added both a publishing and subscribing group, you can start replication.

Procedure

1. In XAdmin, from the U2 Resource pane, select the publishing or subscribing server. From the Admin Tasks view, expand **Replication**, and then double-click **Configuration**.
   The Replication pane displays.

2. Click the **Replication Tool** tab.
3. From the **Functions** section, select the type of administrator option that you want to execute.
   - **Report**: Reports the current status of a replication. This command is useful after a failure or failover occurs. For detailed information about this command, see report command, on page 86.
   - **Sync**: Synchronizes subscribing servers to their publishing servers. The publishing server establishes a connection to the subscribing server and invokes the subscribing process. UniVerse or UniData reads and transfers replication logs from the publishing server to the subscribing server. The subscribing server then applies the updates to the database. For detailed information about this commands, see sync command, on page 77.
   - **Reconfig**: Reconfigures the replication configuration while UniData or UniVerse is running. For detailed information about this command, see reconfig command, on page 82.
   - **Suspend**: Suspends a live replication. In a suspended mode, UniVerse or UniData interrupts the connection between the publisher and the subscriber. The publishing server saves the replication log files to the replication logs reserve file rather than transferring them to the subscribing servers. The subscribing server and all replication writer processes stop after they finish updating existing logs in the replication buffer. For detailed information about this command, see suspend command, on page 75.
   - **Failover**: Changes the replication direction on a local system, either from the local system to the publishing server or subscribing server, or changes the subscribing source distribution. For detailed information about this command, see failover command, on page 80.
   - **Reset**: The reset command clears saved replication logs in the replication log reserve file. Use the reset command after you copy or store database files, since the remaining replication logs are no longer useful. For detailed information about this command, see reset command, on page 85.
   - **Enable**: Enables replication.
   - **Disable**: Disables replication. For more information, see Disabling replication, on page 19.
4. The choices in the **Options** section become available based on which function you select. Select the appropriate options for this replication.
5. Select whether you want a target definition of **All** or **Selected**.

   A target is a replication, a replication group, or a distribution of a replication group. A replication is all data replicated from one server to another server. A target definition of **All** represents all replications on the server. One command can have multiple targets.

   If you want to execute the command against all targets, leave **All** selected in the **Targets** area.

   If you want to select the replication to execute the command against, select the **Selected** option in the **Targets** area of the replication tool dialog box, then click **Add**. In the dialog box that displays, enter the target information, then click **Finish**.
6. Click **Execute** to start the replication process.

   Any output from the command displays in the **Status** box.

Parent topic: Replication tool

Enabling or disabling U2 Data Replication through XAdmin

Enable U2 Data Replication through XAdmin using the Replication pane.

For more detailed steps about enabling or disabling U2 Data Replication through XAdmin, see Managing U2 Data Replication through XAdmin, on page 130.

1. To enable or disable U2 Data Replication through XAdmin, from the **Admin Tasks** view, click **Replication**, then click the **Replication Tool** tab.

2. To enable U2 Data Replication, click the target for which you want to enable replication, then click **Enable**.
3. To disable U2 Data Replication, click the target for which you want to disable replication, then click **Disable**.

   You must be logged in as root to enable or disable U2 Data Replication.

Parent topic: Replication tool

Diagnosis utility

The diagnosis utility is used by U2 support for diagnosing problems encountered with U2 Data Replication. With the utility, you can view information such as the current configuration, groups, semaphores, and other general information that can help you troubleshoot any issues you have with replication.

To access the diagnosis utility, in XAdmin, select the server you want to diagnose. From the **Admin Tasks** view, expand **Replication**, and then double-click **Configuration**. Click the **Diagnosis Utility** tab.

You can use the command line to display the same information as the diagnosis utility. For more information, see Listing replication files, on page 118.

Parent topic: Configuration

Recovery logs

The replication recovery log is a UniData file located in `$UDTHOME/sys`. This file records replication recovery. The resynchronization of a publishing group on a system running RFS generates a record in the replication recovery log that records the publishing group recovery. The database also writes the logs representing missing transactions during recovery when resynchronizing after failing over to a system running immediate replication to this file.

To view the logs from XAdmin, from the Admin Tasks view, expand **Replication**, and then double-click **Configuration**. Click the **Recovery Logs** tab.

Select the time stamp for which you want to view the logs from the **Time Stamp** drop-down list, then select the replication group in the **Group ID** list. XAdmin displays the recovery status.

The replication recovery log has two associated files:

- `REP_RECV_LOG`: Records the recovery of publishing groups and the keys of missing transactions.
- `REP_RECV_REC`: Records the records and virtual attribute values of missing transactions.

- REP_RECV_LOG

  The `REP_RECV_LOG` file contains the following information:

Parent topic: Configuration

REP_RECV_LOG

The `REP_RECV_LOG` file contains the following information:

- Replication group name that was recovered
- Timestamp of the recovery

- Missing logs:
  - Insert/Delete/Update
  - Replication account
  - Replication object name
  - Key of the record
  - @ID of the record in REP_RECV_REC representing the old record of the missing log
  - @ID of the record in REP_RECV_REC representing the new record of the missing log
  - @ID of the record in REP_RECV_REC representing virtual attribute values of the old record
  - @ID of the record in REP_RECV_REC representing virtual attribute values of the new record

You can execute any UniData commands against these files to query the information they contain.

Example

The following examples are taken from a `REP_RECV_LOG` file on the publishing system after crash, failover, and recovery. The first example shows what a `REP_RECV_LOG` looks like for a group that did not have any updates that needed to be recovered.

```
:LIST REP_RECV_LOG 'inventory_acct_g01358959856' ALL
LIST REP_RECV_LOG 'inventory_acct_g01358959856' ALL 06:47:12 Jan 24 2013 1
REP_RECV_LOG inventory_acct_g01358959856
Group Name inventory_acct_g0
Timestamp 1422179456
Operation
Status
Rep Object Name
Record Key
LSN Number
New Tuple Atid
Old Virtual Atid
New Virtual Atid
Is Dict?
Rep Account /disk1/udt/inventory
Recovery Status finished
Failover system standby
1 record listed
```

This example shows a group that had updates recovered by UniData when the daemons were restarted on the production-primary machine. The failover command was run on the production-standby system before UniData was restarted on the production-primary system.

```
:LIST REP_RECV_LOG 'payroll_file_g31358959856' ALL NO.PAGE
LIST REP_RECV_LOG 'payroll_file_g31358959856' ALL NO.PAGE 06:44:22 Jan 24 2013 1
REP_RECV_LOG payroll_file_g31358959856
Group Name payroll_file_g3
Timestamp 1422179456
Operation Insert
Status
Rep Object Name EXEMPTIONS
Record Key 7175
LSN Number 1
New Tuple Atid 1
Old Virtual Atid
New Virtual Atid
Is Dict?
Operation Insert
Status
```

```
Rep Object Name STAFF
Record Key NEXT
LSN Number 2
New Tuple Atid 2
Old Virtual Atid
New Virtual Atid
Is Dict? DICT
Operation Insert
Status
Rep Object Name STAFF
Record Key 40192
LSN Number 3
New Tuple Atid 3
Old Virtual Atid
New Virtual Atid
Is Dict?
Rep Account /disk1/udt/payroll
Recovery Status finished
Failover system standby
1 record listed
```

Viewing recovery logs in XAdmin

To view recovery logs using XAdmin, select the **Replication** option, and then select **Recovery Logs**. In the window, choose a **Time Stamp** and **Group ID** from the drop-down lists.

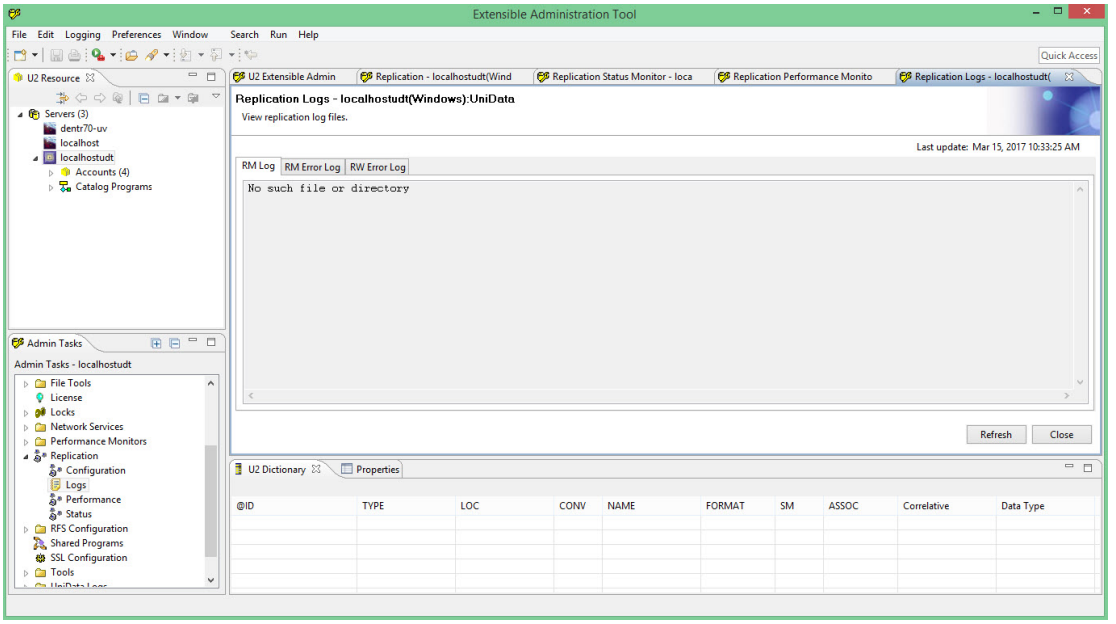
The file `REP_RECV_REC` contains the actual records from the files.

At present, there is no tool to allow a database administrator to push / pull these records onto the subscriber / publisher. Without the assistance of a tool, the DBA must have a very good understanding of the underlying application and associated data to resolve these data issues.

Parent topic: Recovery logs

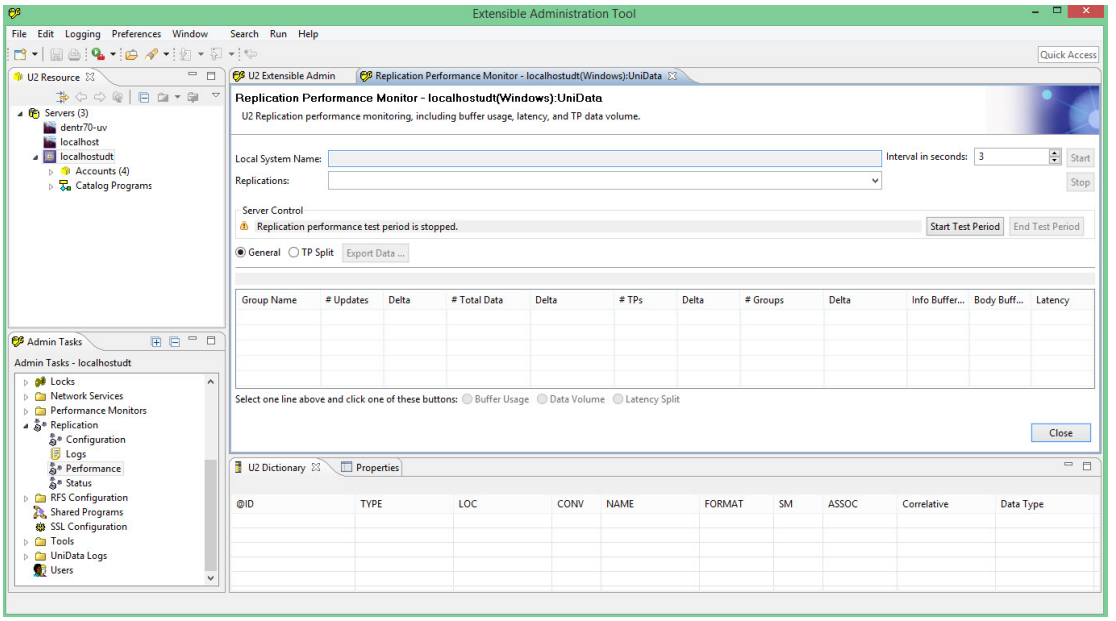
Replication logs

In XAdmin, from the Admin Tasks list, click **Logs**. The **Logs** option allows you to view replication log files. In the following example, UniData is the displayed database.



Performance

To monitor U2 Data Replication, click **Performance** from the Admin Tasks list. In the following example, UniData is the displayed database.



Monitoring replication through XAdmin

The replication monitoring tool monitors connection status, data transferring, and whether the publisher and subscriber systems are synchronized.

1. To access the replication monitor, in XAdmin, select the publishing server. From the Admin Tasks view, expand **Replication**, then double-click **Performance**, and then double-click **Status**.

The following two tabs displays: the **Replication Performance Monitor** tab and the **Replication Status Monitor** tab. These tabs work together to monitor replication.

You can monitor two types of replication:

- Publishing replication: The data replication from the local system to a remote subscribing server, including all replication groups involved. A publishing server can have more than one publishing replication defined.
- Subscribing replication: The data replication from a remote publishing server to the local system, including all replication groups involved. A subscribing server can have more than one subscribing replication defined.

2. **Optional:** You can perform the previous step again but for the subscribing server. This step is not required, but it can be useful because you can see how the publisher and the subscriber communicate with each other.
3. Click the **Replication Performance Monitor** tab.
   a. Select the name of the replication you want to monitor from the **Replications** list.

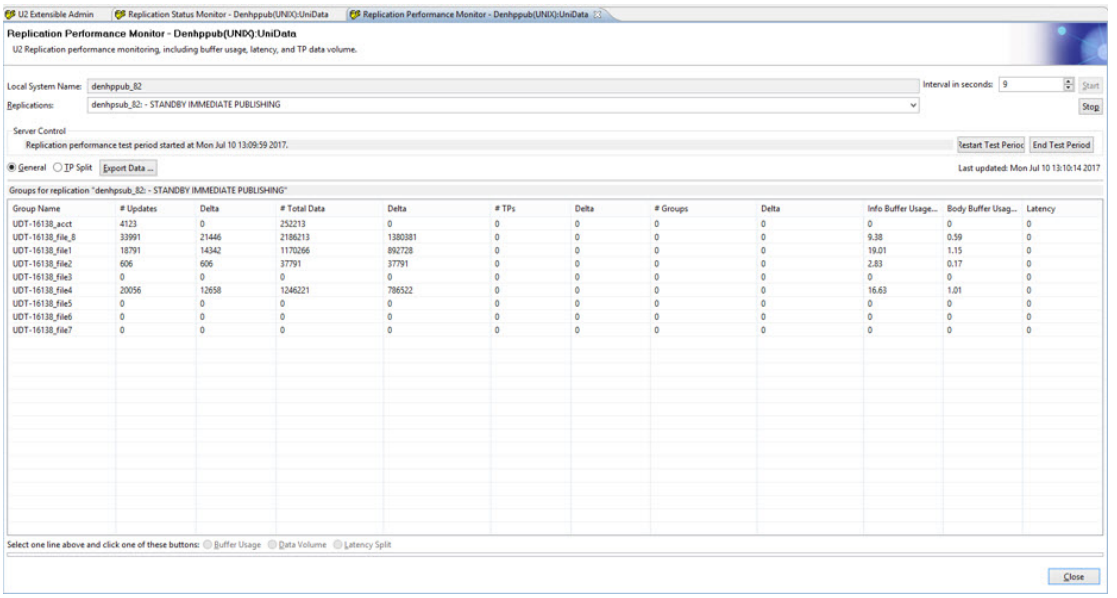
      Each replication is assigned a unique name on one system, consisting of the replication type and the remote system name. A replication type can be one of the following:

      - Immediate
      - Standby immediate
      - Realtime
      - Standby realtime
      - Deferred
      - Delayed

      Note: If a replication has failed over, it still belongs to the same replication as it did before the failover, but the name changes.

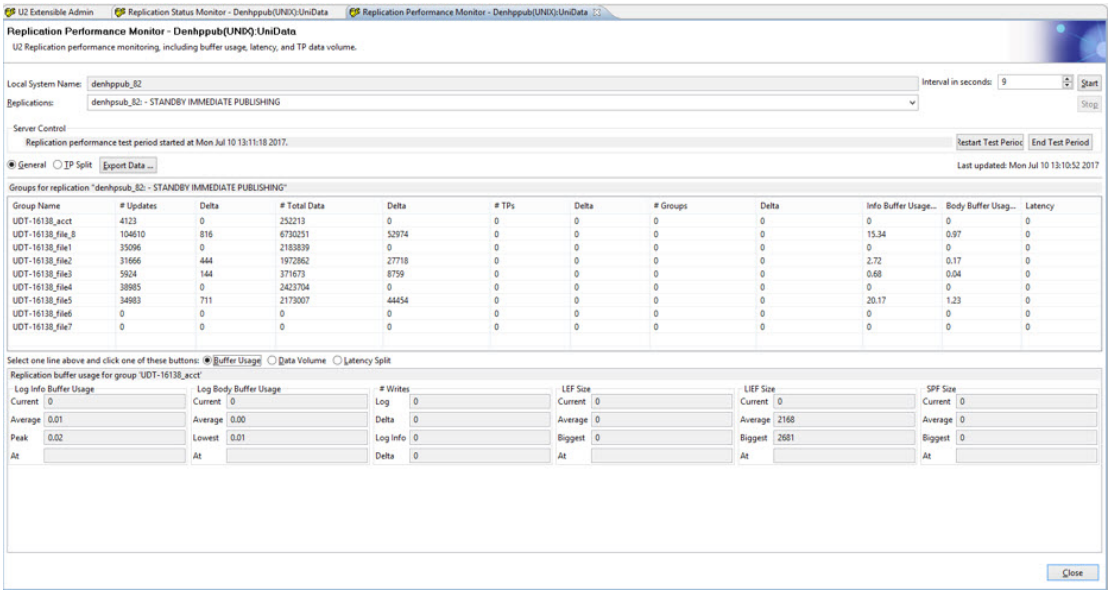
   b. Select the number of seconds to refresh the monitor in the **Interval in seconds** field. The default interval is 3 seconds.
   c. Next to the **Server Control** field, click **Start Test Period**.
   d. Next to the **Interval in seconds** field, click **Start**.

      The replication group status table displays the current status of each replication group belonging to the replication you specify. The following example shows the replication group status table:

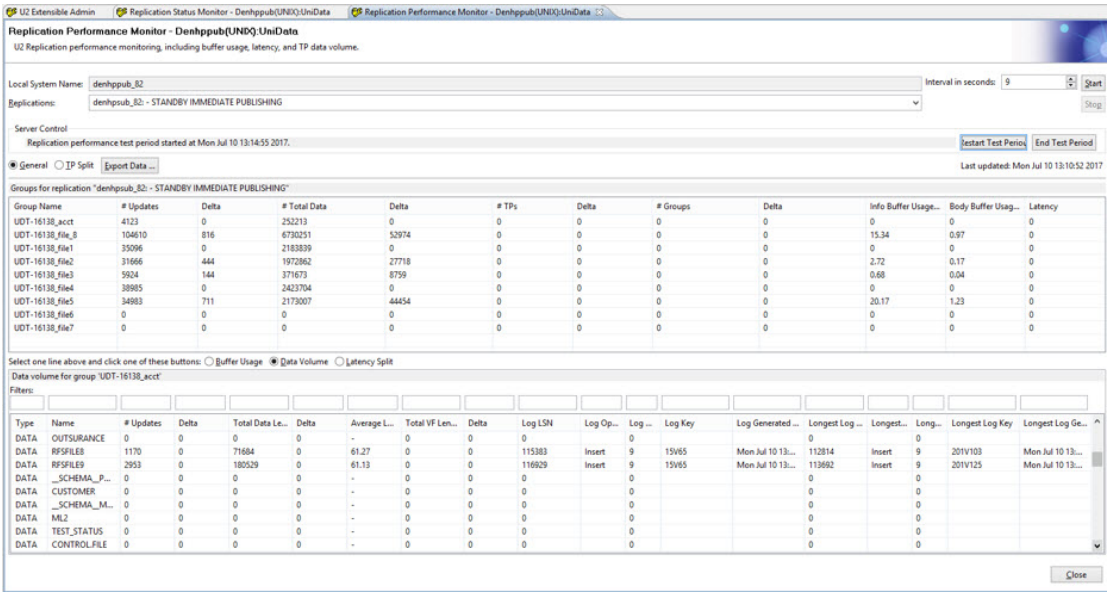


e. Select a row, then below the table, select either **Buffer Usage**, **Data Volume**, or **Latency Split** to view the extended results displayed below the table.

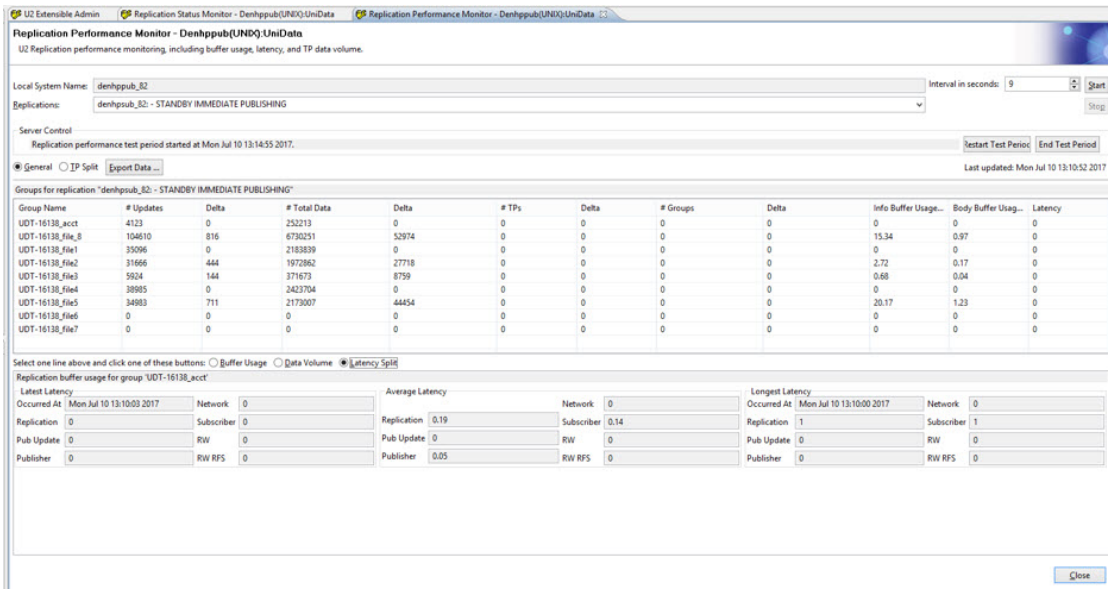
The following example shows the replication group status table with **Buffer Usage** selected:



The following example shows the replication group status table with **Data Volume** selected:



The following example shows the replication group status table with **Latency Split** selected:



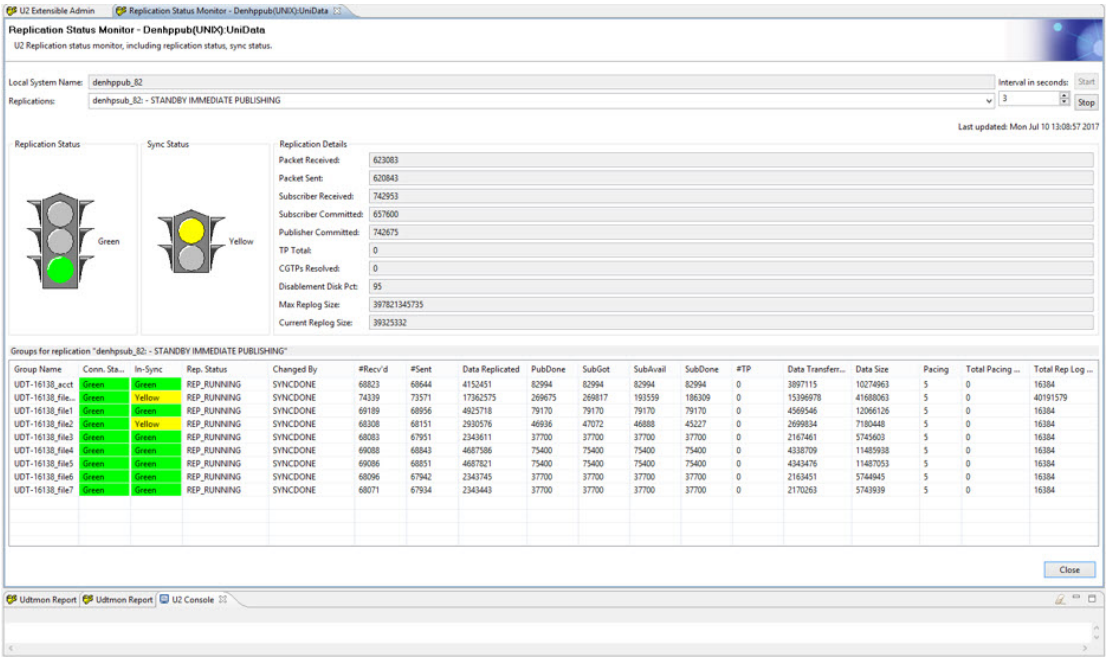
f. By default, the table displays with **General** selected. You can also view the table by test period by selecting **TP Split** from above the table.

> **Note:** To use the **TP Split** option, you must run the `uv_repadmin reconfig -tpstat` (UniVerse) or `ud_repadmin reconfig -tpstat` (UniData) command on the publisher or subscriber. A method to enable this in XAdmin will be added at a future release.

4. To export data in an XML format, click the **Export Data** button.
5. To stop the performance monitor, select **End Test Period** in the **Server Control** area or select the **Stop** button on the top right. If you select the **Stop** button, it will keep the test period enabled. To enable the view again , click **Start**.

6. Click the **Replication Status Monitor** tab.
   a. Select the name of the replication you want to monitor from the **Replications** list.
   b. Select the number of seconds to refresh the monitor in the **Interval in seconds** field. The default interval is 3 seconds.
   c. Next to the **Interval in seconds** field, click **Start**.

      Two traffic lights displays along with details about the replication, as shown in the following example:

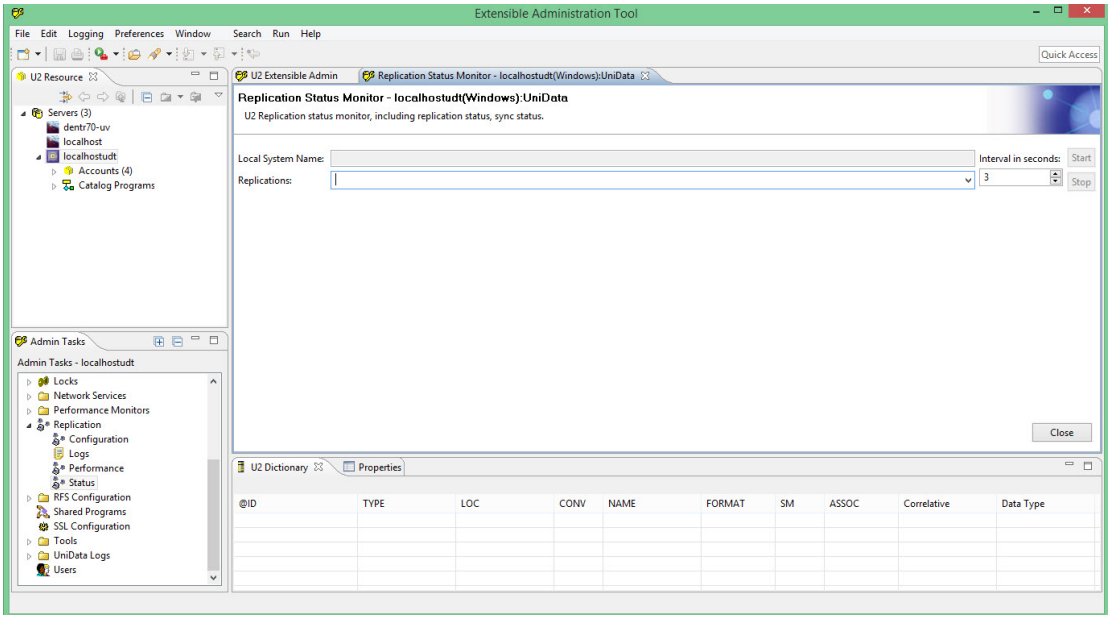


   For more information about the contents of this table, see Status, on page 147.

7. **Optional:** If you opened the **Replication Performance Monitor** tab and the **Replication Status Monitor** tab for both the publisher and the subscriber, repeat the previous two steps to start both tests.

Status

The following figure shows the **Replication Status Monitor** tab connected with UniData.



The following table describes the functionality of the **Replication Status Monitor** tab.

Field/Area	Description
Replication Status	The replication status indicates whether the publisher and subscriber are connected. The status can be one of the following: ▪ Green – The publisher and subscriber are connected for all groups involved in the replication. ▪ Yellow – At least one of the replication groups has been suspended by an administrator. ▪ Red – At least one of the replication groups has been terminated abnormally.
Sync Status	The sync status indicates whether the subscribing database is synchronized with the publishing database. The status can be one of the following: ▪ Green – The publishing and subscribing databases are synchronized. ▪ Yellow – There are pending updates that have not been applied to the subscribing database.
Replication Details	Provides specific details about replication as described below.

Field/Area		Description
	Packet Received	The number of packets received from the other party of the replication. Types of packets include data packets, confirmation packets, heartbeat packets, and other control packets. Monitoring this information indicates whether the physical connection between the publishing database and the subscribing database is satisfactory. This number is cumulative from the last time the database was started or U2 Data Replication was reconfigured.
	Packet Sent	The number of packets that have been sent to the other party of the replication. Types of packets include data packets, confirmation packets, heartbeat packets and other control packets. Monitoring this information indicates whether the physical connection between the publishing database and the subscribing database is satisfactory. This number is cumulative from the last time the database was started or U2 Data Replication was reconfigured.
	Subscriber Received	The number of data records that have been received by the subscriber. This number is cumulative from the last time the database was started or U2 Data Replication was reconfigured. **Note:** When monitoring a publishing replication, this number may be out of date if the replication status is not green.
	Subscriber Committed	The number of data records that have been committed on the subscribing database. This number is cumulative from the last time the database was started or U2 Data Replication was reconfigured. **Note:** When monitoring a publishing replication, this number may be out of date if the replication status is not green.
	Publisher Committed	The number of data records that have been committed on the publishing database. This number is cumulative from the last time the database was started or U2 Data Replication was reconfigured.
	TP Total	The sum of all transactions committed in the replication groups on the local system. This number is cumulative from the last time the database was started or U2 Data Replication was reconfigured.
	CGTPs Resolved	The sum of all transactions committed across more than one replication group on the local system. This number is cumulative from the last time the database was started or U2 Data Replication was reconfigured.
	Disablement Disk Pct	The replication log disk percentage defined by the `udtconfig` file parameter REP_DISABLE_DISK_PCT.
	Max Replog Size	The limit the replication log file size can reach before U2 Data Replication is disabled.
	Current Replog Size	The total replication log file size for all replication groups.
Replication group status table		The replication group status table displays the current status of each replication group belonging to the replication you specify. See the following table for a description of each column of this table.

The following table describes the columns in the replication group status table.

Column	Description
Group name	The name of the replication group.

Column	Description
Connection Status	The connection status between the publishing server and the subscribing server in the group. The status can be one of the following: ▪ Green – The publisher and subscriber are connected in this group. ▪ Yellow – The replication group has been suspended by an administrator. ▪ Red – The replication group has terminated abnormally.
In-Sync	Indicates whether the subscribing files are synchronized with the publishing files in the group. The in-sync status can be one of the following: ▪ Green – The publishing and subscribing databases are synchronized. ▪ Yellow – There are pending updates in this replication group that have not been applied to the subscribing database.
Rep. Status	The current replication group status. ▪ REP_RUNNING – The replication is running. ▪ REP_SYNCING – The replication is performing synchronization. ▪ REP_SUSPENDED – The replication is suspended. ▪ REP_DO_SUSPEND – The replication is in the process of suspending. ▪ REP_EXIT – The replication is suspended due to an abnormal termination. The status can be one of the following on the subscribing server: ▪ SUB_STOP – The replication is stopped. ▪ SUB_EXIT – The subscribing server has exited abnormally. ▪ SUB_SHUTDOWN – The replication on the subscribing server has been shut down. ▪ SUB_RUNNING – The replication on the subscribing server is running. ▪ SUB_DO_RECONFIG – A reconfiguration process is occurring on the subscribing server. ▪ SUB_DO_SUSPEND – The replication is suspended on the subscribing server. ▪ SUB_SYNCING – The replication is performing synchronization on the subscribing server. ▪ SUB_RESYNCING – The replication is performing resynchronization on the subscribing server. ▪ SUB_DO_FAILOVER – The subscribing server is performing a failover.
Changed By	The event or reason that caused the replication status to change. If the status changed due to an exception, this column displays the error category and error code. A detailed error string is available in the tool tips. See the following table for a description of each of the events.
#Recv'd	The number of packets received in the group. Monitoring this number indicates if the physical connection between the publishing server and subscribing server is satisfactory.
#Sent	The number of packets sent in the group. Monitoring this number indicates if the physical connection between the publishing server and subscribing server is satisfactory. This number is cumulative from the last time UniData was started or U2 Data Replication was reconfigured.

Column	Description
Data Replicated	The total amount of data, in bytes, replicated in the group. On a publishing server this field represents the total amount of data sent out. On a subscribing server, this field represents the total amount of data received from the publishing server. This number is cumulative from the last time UniData was started or U2 Data Replication was reconfigured.
SubGot	The log sequential number of the latest replication log received by the subscribing server. This number is cumulative from the last time UniData was started or U2 Data Replication was reconfigured.
SubAvail	The log has been loaded into the replication buffer and is available for the replication writer processes.
SubDone	The log sequential number of the latest replication log committed to the subscribing database. This number is cumulative from the last time UniData was started or U2 Data Replication was reconfigured.
PubDone	The log sequential number of the latest replication log committed to the publishing database. This number is cumulative from the last time UniData was started or U2 Data Replication was reconfigured.
#TP	The total number of transactions resolved in this group, including cross-group transactions. This number is cumulative from the last time UniData was started or U2 Data Replication was reconfigured.
Data Transfer	The total amount of transferred data that has been replicated.
Data Size	The total size of data replicated.
Pacing Level	The replication pacing level in a group, as defined in the REP_PACING parameter in the `repconfig` file.
Total Degradation	The total group pacing weight since the UniData system has started or U2 Data Replication was reconfigured.
Total Rep Log Size	The current replication log file size, including LEF and LRF.

The following table describes the valid events of reasons that appear in the Changed By column that can cause a change in the replication status.

Event/Reason	Description
AUTO_SYNC	Replication is automatically resuming.
CGTP_SUSPEND	Cross-group transaction processing is suspended.
CM_REQUEST	Checkpoint manager is requested (RFS only).
DBA_ORDER	The system administrator issued a request resulting in a change of status.
ENABLE	Replication is enabled.
FAILOVER	A failover has occurred.
PUB_REQUEST	A request was sent from the publishing server, or an event occurred on the publishing server resulting in a change of status.
PUB_STARTUP	The publishing server started.
RECONFIG	Replication is reconfiguring.
REMOTE_REQ	A subscribing server can request to execute a `SYNC` command on the publishing server.
REP_DISABLED	Replication is disabled.
SCHEDULED	The replication manager process schedules a `SYNC` command only for deferred replication.

Event/Reason	Description
SUB_REQUEST	A request was sent from the subscribing server, or an event occurred on the subscribing server resulting in a change of status.
SYNCDONE	A synchronization process succeeded.
SYS_STARTUP	The UniData system started.

Chapter 6: How U2 Data Replication works

The following are available types of replication:

- **Real-time**: In real-time replication, transactions on the publishing server are not committed until all transaction logs have been verified as stored on the subscriber. Updates to the database files on the subscriber may not have completed yet. If replication is suspended, the transaction is not committed until all of the logs for the transaction are written to the log reserve file on the publisher. If the publishing server fails over to the real-time subscribing server, all committed transactions on the publishing server are guaranteed to apply to the standby server. Real-time replication is best for resilience, but worst for performance.
- **Immediate**: In an immediate replication, the publisher's transactions are committed when all logs associated with the transaction have been written to the shared memory replication buffer on the publishing server. The publishing server sends a transaction log to the subscribing server immediately for processing or, if replication is suspended, properly stored in the publisher's log reserve file.

  This type of replication has better performance than real-time replication, but comes with a slight risk of the last-updated data being lost or temporarily unavailable if the publisher fails. Immediate replication is subject to Transaction Boundaries, if used. If the publishing server fails and cannot be restarted, data will be lost. If the publishing server can be restarted and recovered the last update can also be recovered. though it will not be available on the failover system until recovery steps have been completed. You can use immediate replication for a standby failover system.
- **Deferred**: In deferred replication, the publisher stores transactions in log files and sends the log files to the subscriber at regular intervals. The subscribing server connects to the publishing server, retrieves all of the logs, and synchronizes its database with the publishing database. Deferred replication is used only in non-standby replication.

  Deferred replication has the best performance, but bears the greatest exposure for data loss. The subscriber data is not current with publisher data; however, this mode does allow a quarantine period. A quarantine period means that the subscriber is safe from unwanted events on the publisher for the deferred period.

  You cannot use deferred mode for a standby failover system.
- **Delayed**: Delayed replication delays updates to the subscribing server by a specific amount of time. The publishing server can failover to the subscriber system up to a specified time. This will allow you to cancel or void some unwanted updates during the failover to protect the database from accidental misuse or malicious damage.

  In delayed replication, you can define the delay time period for the subscriber and the publisher. The time period can be defined differently for each system. The time period defined on the subscriber system takes effect at runtime. On the publisher, it takes effect after failover.

For all types of replication, the publishing server maintains network connections to its subscribers while it is running. UniData sends data updates on the publishing object to its subscribers through this connection immediately.

Single replication group

The following steps describe running replication through a single replication group.

1. On the publishing server, the database processes (`udt` or `tm` for RFS files) update records that are in the files of an account and creates replication logs in the replication buffer that is in shared memory.
2. If the replication buffer is full or the logs are large, the logs are temporarily stored in the log extension file for that group.

3. The publisher process (`udpub`) reads the logs from the replication buffer and sends them across the network to the subscribing server.
4. The sublistener thread of the subscriber process (`udsub`) receives the logs into the subpacket store.
5. The submain thread of the subscriber process (`udsub`) retrieves the logs from the subpacket store and loads them into the group's replication buffer in shared memory.
6. If the replication buffer is full or the logs are large, the logs are temporarily stored in the group's log extension file.
7. The replication writer processes (`udrw`) read the logs from the buffer and make the appropriate update to the files in the account specified for that group.
8. The subscriber process sends a log status to the publishing server. The following are the log statuses:
   - SubGot- The log was received.
   - SubAvail- The log is in the replication buffer.
   - SubDone- The update was committed to disk.
9. The publisher listener process (`publistener`) on the publishing server receives the log status and updates the logs in the replication buffer.
10. When the publishing server receives the sub-done log status from the subscriber, the publishing server deletes the replication log from the buffer.

Two replication groups

The following describes running replication with two replication groups.

- Adding the second replication group (annotated as group 1) results in the creation of additional components that are highlighted by the dotted line in the previous figure:
  - On the publishing system:
    - Replication buffer for group 1
    - Log extension file for group 1
    - Publisher process (`udub -g1`)
    - Publisher listener process (`publistener`)
  - On the subscribing system:
    - Subscriber process for group 1 (`udsub`)
    - Sub packet file for group 1
    - Replication buffer for group 1
    - Log extension file for group 1
    - Replication writer processes for group 1
  - TCP/IP connections for the new publisher, publisher listener, and subscriber processes

Data replication process

The following steps describe the process of moving replicated data form the publisher to the subscriber.

1. Before UniData commits an update, the `udt` or `tm` process generates a replication log for the update and writes it to the replication buffer.
2. In real-time replication, the `udt` or `tm` process waits for the response from the publisher process.

3. The publisher process reads the logs from the replication buffer and sends a package, through the network, to all subscribers.
4. When a subscriber receives a package, the subscriber process converts the data record, if necessary, copies the logs to the replication buffer, and sends a sub-got message to the publisher.
5. In real-time replication, the publisher process receives the sub-got notification and wakes up the waiting `udt` or `tm` process.
6. The `udt` or `tm` process commits the update to the database.
7. If pacing is enabled, the publisher calculates the pacing weights and, if necessary, sleeps to replication from being disabled.
8. If RFS is not running, the udt process writes a pub-commit notification to the replication buffer. If RFS is running, the aimglog process writes the pub-commit notification to the replication buffer when UniData writes the redo log to disk.
9. The publisher process sends the pub-commit notification to the subscribers in the next data package.
10. When the subscriber receives a pub-commit notification, it marks the pub-commit flag in the log in the replication buffer and wakes up the replication writer processes waiting for the log.
11. A replication writer process reads the log from the replication buffers once, and applies a logical lock for the record it is about to update. It then checks the pub-commit flag on the log and waits for the flag to be set by the subscriber. For the log of an update involved in a transaction, the replication writer process also has to wait for the arrival of the transaction-end log.
12. Next, the replication writer process updates the subscribing objects according to the contents of the replication log.
13. The replication writer process then updates the sub-commit flag in the log to indicate that the log has been committed on the subscribing server, and releases the logical lock for the record.
14. The subscriber sends back a sub-commit notification to the publishing server confirming that the update has been committed on the subscribing server.
15. When the publishing process receives the sub-commit notification message, it removes the logs from the replication buffer.

Suspending replication

System administrators can suspend replication when subscribing servers shut down, crash, network failures occur, or for their own reason.

Note: Pacing weights are impacted by buffer overflows to the disk. When replication is suspended, updates will overflow to the replication disk log files since no updates are being processed by the subscriber. Therefore, if you suspend replication with pacing enabled, your system will be impacted.

You can suspend replication by issuing the `suspend` command. For more information, see the suspend command, on page 75.

Example

To suspend a replication, the publishing server saves all replication logs without sub-commit notification to the replication log reserve file. If the subscriber did not crash, it saves all logs without pub-commit notification to the replication log reserve file. UniData then closes the connection between the publisher and the subscriber. On the subscriber, the `udpub` and `udrw` processes close and exit.

When you suspend a replication, the publishing server runs in suspension mode. The `udt` or `tm` process can read from or write to the database as usual. Instead of sending replication logs to the subscriber, the publisher saves them in the replication log reserve file.

The following section describes how U2 Data Replication behaves in suspension mode:

1. Before committing an update, the `udt` or `tm` process generates replication logs for the update, and writes them to the replication buffer.
2. For real-time replication, the `udt` or `tm` process waits for the response from the publisher process.
3. The publisher process reads the logs from the replication buffer and saves them to the replication log reserve file.
4. For real-time replication, the publisher process wakes up the waiting `udt` or `tm` process.
5. The `udt` or `tm` process commits the update to the database.
6. If you are not using RFS, the `udt` process writes the pub-commit notification to the replication buffer. If you are using RFS, the aimglog process writes the pub-commit notification to the replication buffer when it writes the RFS redo log to disk.
7. The publisher process removes the logs from the replication buffer.

Synchronizing replication

You must synchronize suspended replication to resume replication to the running mode.

To synchronize the replication, use the `sync` command on the publishing or subscribing server. For more information, see the sync command, on page 77.

How synchronization works

UniData completes the following steps when synchronizing suspended replication:

1. The publishing server establishes a connection to the subscribing server and invokes the subscriber process.
2. The subscriber process creates the replication writer processes to read the logs from the replication buffer. The subscriber then begins listening for a data package.
3. The publishing server creates the publisher listener process to listen to the return package from the subscriber process.
4. The publishing server creates the publisher-syncing tool process.
5. The publisher-syncing tool process reads the logs from the replication log reserve file and sends them to the subscriber.
6. When a package of replication log files arrive at the subscriber, the subscriber process converts the data record, if necessary, and copies the logs to the replication buffer. It then sends a sub-got notification message to the publisher.
7. A replication writer process reads the log from the replication buffers once, and applies a logical lock for the record it is about to update. For the log of an update involved in a transaction, the replication writer process also has to wait for the arrival of the transaction-end log.
8. The replication writer process updates the database according to the contents of the replication log, and marks the sub-commit flag of the log to indicate the log has been committed on the subscribing server. It then releases the logical lock for the record.
9. The subscriber process sends back the sub-commit notification message to the publishing server.
10. For every notification package received, the publisher listener process marks the sub-got and sub-commit flags of the logs.
11. When the publisher-syncing tool finishes sending all of the logs in the replication log reserve file, it terminates.
12. After the publisher-syncing tool terminates, the publisher process checks to see if any new replication logs were generated during the synchronization process. If new replication logs exist, it begins at step 4 and repeats the synchronization process.

13. The publishing server now resumes replication in running mode.

Deferred replication

In deferred replication, the publishing server does not keep connections to the subscribers active. UniData saves the data in the replication log reserve file instead of sending the data to the subscribers. Data synchronization is the only means of data propagation.

Deferred replication has two types:

- Periodic replication has a predefined period of time when the replication system will automatically synchronize its data. The predefined time period is specified by the SYNCINTVAL phrase in the `repsys` file.
- Manual-sync replication does not have a predefined synchronization period. Data propagation is achieved by the `sync` command.

Deferred replication is similar to the suspension mode of immediate replication. Following are the steps for data synchronization with deferred replication:

1. The publishing server establishes a connection to the subscribing server and invokes the subscriber process.
2. The subscriber process creates the replication writer processes to read the logs from the replication buffer. The subscriber then begins listening for a data package.
3. The publishing server creates the publisher listener process to listen to the return package from the subscriber process.
4. The publishing server creates the publisher-syncing tool process.
5. The publisher-syncing tool process reads the logs from the replication log reserve file and sends them to the subscriber.
6. When a package of replication log files arrive at the subscriber, the subscriber process converts the data record, if necessary, and copies the logs to the replication buffer. It then sends a sub-got notification message to the publisher.
7. A replication writer process reads the log from the replication buffers once, and applies a logical lock for the record it is about to update. For the log of an update involved in a transaction, the replication writer process also has to wait for the arrival of the transaction-end log.
8. The replication writer process updates the database according to the contents of the replication log, and marks the sub-commit flag of the log to indicate the log has been committed on the subscribing server. It then releases the logical lock for the record.
9. The subscriber process sends back the sub-commit notification message to the publishing server.
10. For every notification package received, the publisher listener process marks the sub-got and sub-commit flags of the logs.
11. When the publisher-syncing tool finishes sending all of the logs in the replication log reserve file, it terminates. Any replication logs created after the sync process was initiated will be deferred until the next scheduled sync operation.
12. The publishing server disconnects from the subscriber and returns replication to suspension mode.

Transaction handling

The following section describes how U2 Data Replication handles transaction processing:

1. At the beginning of the commit phase of a transaction, the tm process generates all replication logs for the updates of a replication object in the transaction. This process also generates a transaction-begin and transaction-end log for each of the transactions involved in the replication

group. A transaction-begin log contains the transaction ID, the number of replication groups involved in the transaction, and the number of updates within the replication group.

2. For each of the replication groups, the tm process writes the transaction-begin log, the replication logs of all updates in the replication group, and the transaction-end log to the replication buffer of the replication group. If you are using real-time replication, the tm process waits for the transaction-end log to ensure all logs of the transaction arrive at the subscribing server.
3. The publishing process sends the logs to the subscriber. In the case of deferred replication, UniData writes the logs to the replication log reserve file. When the publisher receives the sub-got notification message from the subscriber, it wakes up the tm process that is waiting.
4. The tm process writes the RFS redo logs, and commits the transaction to the database.
5. After the aimglog process flushes the redo logs to disk, it marks the pub-commit flag of the transaction-begin log.
6. The publishing process sends the pub-commit notification to the subscriber process.
7. When the subscriber process receives a transaction-begin log, it copies the log to the replication buffer. If the transaction is part of a cross-group transaction, it checks the transaction control area for the transaction control record for this transaction. If the transaction control record does not exist, it creates a new transaction control record and adds it to the transaction control area.
8. When the subscriber process receives the data update logs of a transaction, it copies the logs to the replication buffer.
9. When the subscriber process receives a transaction-end log, it sets the sub-got flag in the transaction-begin log. If the transaction is part of a multi-group transaction, it increases the number of sub-got groups in the transaction control record. If the number of sub-got groups is equal to the number of groups involved in the transaction, it then sets the sub-got flag in the transaction control record and wakes all replication writer processes waiting for the transaction control record, then sends a sub-got notification message to the publishing server.
10. The replication writer process reads one data update log from the replication buffer, applies a logical lock on the record it is about to update, then waits for the arrival of all of the replication logs involved in the transaction. It also waits for the pub-commit notification message from the publishing server.
11. The replication writer process updates the database according to the contents of the log files.
12. The replication writer process sets the sub-commit flag in the log, and increases the number of sub-commit logs in the transaction-begin log. If the number of sub-commit logs equals the number of logs in the group for the transaction, it increases the number of sub-commit groups involved in the transaction. If the number of sub-commit groups equals the number of groups in the transaction, it sets the sub-commit flag in the transaction control record to indicate the transaction has been committed on the subscribing server.
13. The subscriber process sends back a sub-commit notification message to the publishing server, and removes the transaction control record from the transaction control area.
14. The publishing server removes all logs related to the transaction from the replication buffer.

RFS checkpoint handling

When running RFS, the Checkpoint Manager (CM) process coordinates with the publisher processes to ensure that all replication logs of the current redo log set are either transferred to the immediate replication subscriber, or saved to the replication log reserve file before the CM switches the redo log sets. If the system crashes after this checkpoint, the restart process generates replication logs from the current redo log set.

Indexed virtual attributes

U2 Data Replication propagates the indexed virtual attribute values, as well as the record. The following two methods are available to propagate virtual attribute values:

The RW_REEVALUATE_VF phrase, in the `repconfig` file of each replication group, controls this behavior.

- When set to 0, the values evaluated on the publishing server are used. If the index is enabled, the udt or tm process evaluates all indexed virtual attribute values before updating the record in the database. These values are also passed as a record to the subscribing servers. The replication writer process then writes the values directly to the index file. When updating the subscriber using the values evaluated on the publishing server, the following rules apply:
  - The dictionary and index definition must be the same on the publisher and subscriber.
  - The virtual field indexes must have been created in the same order on both the publisher and subscriber.
  - The same number of virtual field indexes must exist on the publisher and subscriber.
  - The state (enabled/disabled) of the index must be the same on the publisher and subscriber.

  If these rules are not followed, virtual field index data may be written to the incorrect index on the subscriber.
- When set to 1, indexed virtual attributes on the subscribing server are reevaluated. This is the default setting. Before the replication writer process applies the replication log to the database, it evaluates all indexed virtual attribute values and saves them in the index file, if the index is enabled. If the index is disabled, it saves the record to the index log file, then the `UPDATE.INDEX` command evaluates the values.

File triggers

If you set the value of RW_SKIP_TRIGGER to 1, if a subscribing object has triggers defined, the replication writer process will not execute the trigger when applying the replication log. Your database administrator can configure the replication group to execute the trigger by setting the value of RW_SKIP_TRIGGER to 0 in the `repconfig` file. The default value is 0.

Note: If the trigger program updates replicated files, you will need to mark them as SUBWRITEABLE in order to update them on the subscribing server.

Supported file-level commands

UniData supports the following file-level commands:

- DIR file operations:
  - `ED`
  - `COPY`
  - `CLEAR.FILE`

- File-level commands
  - CREATE.FILE
  - DELETE.FILE
  - CLEAR.FILE
  - CNAME
  - SETFILE
- Other commands:
  - BASIC
  - CATALOG
  - DELETE.CATALOG
  - DELETE.LIST
  - COPY.LIST
  - SAVE.STACK
  - CREATE.ENCRYPTION.KEY
  - CREATE.ENCRYPTION.WALLET
  - GRANT.ENCRYPTION.KEY
  - REVOKE.ENCRYPTION.KEY
  - DELETE.ENCRYPTION.KEY
  - DELETE.ENCRYPTION.WALLET
  - WALLET.ADD.KEY
  - WALLET.REMOVE.KEY
  - SAVE.EDMAP

UniData does not support the following commands for replication:

- BUILD.INDEX
- CLEAR.ACCOUNT
- CREATE.INDEX
- CREATE.TRIGGER
- DECRYPT.FILE
- DECRYPT.INDEX
- DELETE.INDEX
- DELETE.LIST
- DELETE.TRIGGER
- DISABLE.INDEX
- EDA.CONVERT
- ENABLE.INDEX
- ENCRYPT.FILE
- ENCRYPT.INDEX
- MERGE.LIST
- RESIZE

- SAVE.LIST
- UPDATE.INDEX

Note: If you run the CREATE.INDEX, BUILD.INDEX, SET.INDEX, or DELETE.INDEX commands against a published file, UniData writes a message to the rm.log file indicating the command that was run, the account where it was run, and the file it was run against.

Note: The commands UniData does not support for replication can be executed on the subscribing server; they are not blocked.

File-level commands

To replicate file-level commands, you must publish the target file. If the excluded file list contains that file, or if the file is disabled, UniData does not replicate the file-level command.

If a file-level command creates a new file, and UniData successfully enables the file for publication, UniData replicates the command to the subscriber.

If a file-level command automatically updates a VOC or dictionary file. UniData replicates these updates, whether or not the VOC or dictionary file is published. These updates are also performed on the subscribing server when re-executing the command.

CREATE.FILE

When you execute the CREATE.FILE command, UniData enables the file for replication. If UniData successfully enables the file as a publishing file, it sends the command itself to the subscribing server for replication. When the subscribing server receives the CREATE.FILE log, the replication writer process applies an exclusive lock to the VOC entry for the file and re-executes the command.

After the replication writer process creates the file on the subscribing server, it enables the file for subscription. Updates on the file will also be replicated.

If you specify the PARTTBL option with the CREATE.FILE command, UniData does not replicate this option to the subscriber, but uses the default PARTTBL on the subscribing server.

DELETE.FILE

When you execute the DELETE.FILE command, UniData checks if the file is published. If so, UniData sends the command to the subscribing server for replication.

The DELETE.FILE command automatically disables the publication and subscription of the file.

Note: If UDT.OPTIONS 87 is ON on the publishing server, this also will be ON on the subscribing server with the DELETE.FILE log. The replication writer process will delete the remote file.

1. If you issue a DELETE.FILE command on a publishing server against a replicated file, the command may not be successfully applied to the subscribing server. This can occur if a UniData process on the subscribing server has the file open. It could be opened by a process running your application software on the subscribing server – perhaps for reporting.
   a. On UNIX systems, this situation is extremely unlikely and is only really applicable to Windows systems due to how windows behaves regarding open files. The command is successful on

the publishing system, but fails completely on the subscribing system – leaving all parts of the file in place. Messages similar to the following are recorded in the `$UVHOME/uvrm.log` and `uvrm.errlog` files:

```
uvrm.errlog
Mon May 14 13:28:01 2012, RW report:
RW(0, 832):RW failed to replicate(0, 0); (LSN=3)DELETE.FILE on DICT
WWWTEST ('WWWTEST') is abort.
Mon May 14 13:28:01 2012, RW report:
RW(0, 832):RW failed to replicate(0, 0); (LSN=3)DELETE.FILE on
WWWTEST ('WWWTEST') is abort.
```

b. On Windows systems, this situation applies to both recoverable and nonrecoverable files. This is due to the way the Windows operating system behaves regarding open files. With a nonrecoverable file on Windows, you may end up with the dictionary and VOC pointer for the file deleted, but the data portion remaining.

2. `DELETE.FILE` will leave the dictionary file and VOC entry on a subscribing server if the data portion of the file is replicated, but the dictionary file is not. This is an unlikely situation. You would need to have specifically excluded the dictionary file from replication with an EXCLUDED_FILE phrase in `repconfig` or `repacct.def`.

CNAME

The `CNAME` command changes both the physical name and VOC entry name for both the data file and its dictionary. The old file name must be an enabled publishing file or the `CNAME` command will not replicate on the subscribing server.

If you enable the new file as a replication file in a different replication group than the old file, UniData replicates the `CNAME` command in the group of the original file and does not synchronize it with other operations on the new file.

1. `CNAME` changes the name of a file. If you use this ECL command on a replicated file, it is possible that some updates to the file could be lost after the name is changed. This is a problem only if the new name for the file is in a different replication group than the original file name. For this situation to occur, the new file name would have had to been predefined in `repconfig` with a FILE phrase. If you have an account-level group created for the account where the `CNAME` command is run and neither of the filenames is explicitly defined, you will not have any problems with lost updates after `CNAME`.

   How can updates be lost? The `CNAME` command itself is replicated in the group where updates to the original file name are being processed. All replication logs within a group are applied sequentially on the subscribing server. Once the `CNAME` command on the publishing server is committed, the replication object used for the file is changed to the new name. New updates to the file are sent to the new replication group associated with the new file name. If there is a large backlog of replication logs pending for the original group (but the new group isn't busy), it is possible for the subscribing server to try to apply updates to the new name in the new group before the actual `CNAME` command is received and processed in the original group. As the new filename does not yet exist on the subscribing system (as a replication object in the new group), those logs will be canceled and the updates lost. When the `CNAME` command is committed on the subscribing system, subsequent updates to the new file name are properly applied to the data file.
2. `CNAME` could also change a file from being replicated to not being replicated (or vice-versa). Here are two possible scenarios where replication of updates to the file would stop:

- The new file name is already specified in an EXCLUDED_FILE phrase.
- The new file name is not defined in `repconfig` and there is no account-level group created for the account.

SETFILE

The `SETFILE` command creates a new VOC pointer, and also loads that pointer into the replication file table. If you delete the old VOC pointer using the `DELETE.FILE` command, the file may still replicate with the new VOC pointer. UniData replicates `SETFILE` as a file-level operation, so the new pointer on the subscribing server is also loaded into the replication file table.

There is a potential problem with data consistency on a subscribing system when the `SETFILE` command is used. This is similar to the problem (described above) with `CNAME`, but not quite the same. Updates to a record in a file could be applied out of sequence, if the `SETFILE` command changes the primary replication object for the file to an object in a different replication group. An out-of-sequence update could result in stale data being written to a record in a file.

Within a replication group, UniData guarantees that replication logs are applied to the subscribing server in the proper sequence. For a problem to occur, multiple updates would need to be made to a record in a file that span a `SETFILE` command that changes the primary replication object to one in another group. The original group would also need to be busier than the new group, so that older updates processed in the original group are handled later than subsequent updates in the new group on the subscribing server.

Under what circumstances could the primary replication object for a file change as the result of a `SETFILE` command?

1. The file is being replicated as a dynamic object (not explicitly named) in an account-level replication group. The new file reference created by the `SETFILE` command already exists as a `repconfig` FILE phrase in a different replication group. Since an explicit file reference takes precedence over an implicit reference, the new file reference becomes the primary replication object.
2. Both the original file reference and the new file reference (created by `SETFILE`) are defined in FILE phrases in different replication groups. Since FILE phrases are loaded with the same relative weight, the primary file reference might not change. It depends entirely on how the replication objects hashed in a table in shared memory.

Consider the following example for SETFILE DELIVERY DELIVERY_NEW with scenaio 1:

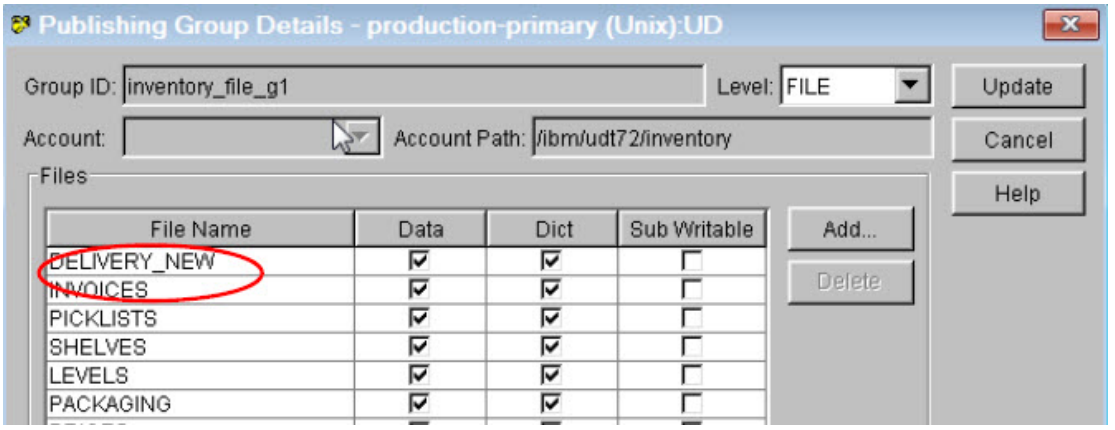
The DELIVERY file is being replicated in account-level group inventory_acct_g0.

```
:LIST.REPLICATION.FILE DELIVERY
FTYPE FILENAME REPLICATION PRIMARY GROUP
DICT DELIVERY PUBLISHED YES inventory_acct_g0
DATA DELIVERY PUBLISHED YES inventory_acct_g0
```

The replication object DELIVERY_NEW is not being replicated. There is no current VOC pointer by that name.

```
:LIST.REPLICATION.FILE DELIVERY_NEW
'DELIVERY_NEW' does not exist in VOC file.
DELIVERY_NEW does not exist in this directory
```

DELIVERY_NEW is defined in a FILE phrase in replication group inventory_file_g1.



During replication start-up, the DELIVERY_NEW object could not be loaded as there was no VOC entry for it. This is the expected behavior.

```
:!grep DELIVERY_NEW $UDTBIN/rm.errlog
Mon Aug 9 16:16:37 Unable to load object DELIVERY_NEW in the group inventory_file_g1
Mon Aug 9 16:16:37 Unable to load object DICT DELIVERY_NEW in the group inventory_file_g1
```

After running the ECL command “SETFILE DELIVERY DELIVERY_NEW”, the primary replication object for the DELIVERY file is now DELIVERY_NEW.

```
:LIST.REPLICATION.FILE DELIVERY
FTYPE FILENAME REPLICATION PRIMARY GROUP
DICT DELIVERY_NEW PUBLISHED YES inventory_file_g1
DICT DELIVERY PUBLISHED NO inventory_acct_g0
DATA DELIVERY_NEW PUBLISHED YES inventory_file_g1
DATA DELIVERY PUBLISHED NO inventoryacctg0 `
```

Adding single file to replication

As a database administrator, you may occasionally want to enable replication for a file that is not already replicated. This would be in an account for which there is no account-level replication group defined. If there is a file-level group defined, you can add the file to that group by creating a new FILE phrase in the `repconfig` file. To enable it, you would then either run `ud_repadmin reconfig` or restart the database.

Alternatively, assuming you have at least one account-level replication group defined on your system, you could use `SETFILE` in that account to create a VOC pointer to the non-replicated remote file. When you use `SETFILE` to create a pointer to a file, a new replication object is immediately added to the replication object table in shared memory. Any subsequent updates to the file via the new file reference are replicated.

Note: You must use `SETFILE` for this to occur. If you use an editor to manually create a new VOC pointer to the non-replicated file, subsequent updates are not replicated. Of course, when you restart UniData this new file reference in the VOC of the account-level enabled account will be loaded as a replication object and available for updates to the file or via a `ud_repadmin reconfig` command.

Miscellaneous

If replication is between a UNIX platform and a Windows platform, UniData does not convert slashes contained in the path.

Replicating file-level commands might result in the creation of a new operating system level file on the subscriber. The following rules determine the owner of the file:

- If the replication configuration file contains the RW_UID parameter, the owner of the new file is RW_UID.
- If the replication configuration file does not contains a RW_UID parameter and the user login name from the publisher exists on the subscriber, the user login name from the publisher is used.
- If the replication configuration file does not contains a RW_UID parameter or the user login name does not exist on the subscriber, root is for the owner of the file.

Account-level replication restrictions

Account-level replication does not replicate the following operations:

- BUILD.INDEX
- CREATE.INDEX
- CONFIGURE.FILE
- REBUILD.FILE
- shfbuild
- memresize
- convhash - Not available beginning at UniData 8.1.0.
- convmark
- convcode
- convtape
- convdata
- convidx
- convsecu
- udfile
- SETLANG

Chapter 7: Failover and recovery

Failover converts a standby subscribing server into a publishing server when the publishing server becomes unavailable.

You can configure a subscribing server of a live replication as a standby server. Then if the publishing server fails, your database administrator can choose to failover immediately to one of the standby servers. This server then becomes a publishing server after the failover. Other subscribing servers must then subscribe from this server.

When the primary server restarts, the server should become a subscriber from the new publishing server.

Failover of real-time replication guarantees that all committed updates on the primary server commit on the standby server as well. For immediate replication, there is a chance that some updates were committed on the primary server, but their logs might not have arrived on the standby server. These updates will be missing on the standby server after failover, but might be recovered after recovery and resynchronization of the primary server.

The following table describes the two types of failover classifications:

Type	Description
Soft	This is the type of failover in which the administrator has decided to make one of the subscribing servers the publishing server. The administrator normally initiates a soft failover for the purpose of performing scheduled maintenance or housekeeping tasks.
Hard	This is the type of failover in which the publishing server has suffered some type of catastrophic, unplanned failure, and one of the subscribing servers needs to become the publishing server until such time that the publishing system is able to take control again.

Note: After a failover, the original publishing server is referred to as the " failed over subscriber" and the original subscribing server is referred to as the "failed over publisher."

Soft failover

The first course of action is to make sure that the publishing and subscribing servers and associated replication groups are all working as expected. You can achieve this by using the `ud_repadmin` command.

Examples

The following example is from the publishing server:

```
# $UDTBIN/ud_repadmin report -detail
reporting ALL...
Replication from/to standby is REP_RUNNING:Sync Succeeded.
GROUP inventory_acct_g0, SYSTEM standby is REP_RUNNING:Sync
Succeeded.
GROUP inventory_file_g1, SYSTEM standby is REP_RUNNING:Sync
Succeeded.
GROUP payroll_acct_g2, SYSTEM standby is REP_RUNNING:Sync Succeeded.
GROUP payroll_file_g3, SYSTEM standby is REP_RUNNING:Sync Succeeded.
#
```

The next example is from the subscribing server:

```
# $UDTBIN/ud_repadmin report -detail
reporting ALL...

Replication from/to primary is SUB_RUNNING:Sync Succeeded.

GROUP inventory_acct_g0, SYSTEM primary is SUB_RUNNING:Sync Succeeded.

GROUP inventory_file_g1, SYSTEM primary is SUB_RUNNING:Sync Succeeded.

GROUP payroll_acct_g2, SYSTEM primary is SUB_RUNNING:Sync Succeeded.

GROUP payroll_file_g3, SYSTEM primary is SUB_RUNNING:Sync Succeeded.
#
```

The output of the `ud_repadmin` commands shows that both machines and all four replication groups are in a working state and the procedure for a failover can continue. If the output from the `ud_repadmin` commands differs from those shown in the example, it is an indication that you need to take steps to rectify any replication system problems before attempting a failover.

The `ud_repadmin` command is used to control which groups are to be failed over to which system. The `ud_repadmin` command can, if required, fail over a single group to another system. This functionality proves useful if a system is set up in such a way that users can be automatically directed to the machines and accounts they use. However, it is far more likely that all the groups on a system will be failed over, so the following example is based on this premise.

Before continuing with the failover procedure, all users should log off in a controlled manner to make sure that work on the replicated files has been completed. This is not completely necessary because after the failover, the files become read-only. However, logging off ensures that potential problems and extra decision-making are avoided later in the procedure. For example, when the files become read-only, any programs attempting to write to them fail with a write error, which might cause concern for users, which in turn would cause concern for the system administrator.

In the following example, two machines are used and the replication environment has four groups: inventory_acct_g0, inventory_file_g1, payroll_acct_g2, and payroll_file_3. The initial publishing server is SYSTEM primary running on HOSTNAME production-primary, and the initial subscribing server is SYSTEM standby running on HOSTNAME production-standby.

Step 1 is to issue the failover command on production-primary (the publisher), instructing the publishing server to subscribe to production-standby and become the failed over subscriber:

```
# $UDTBIN/ud_repadmin failover -verbose -subto standby
failovering ALL...
Command succeeded!
Publishing Group inventory_acct_g0 is PUB_STOP:Failed Over.
Publishing Group inventory_file_g1 is PUB_STOP:Failed Over.
Publishing Group payroll_acct_g2 is PUB_STOP:Failed Over.
Publishing Group payroll_file_g3 is PUB_STOP:Failed Over.
#
```

Note: After the failover command has been issued, the production-primary system is referred to as the “failed over subscriber”.

As previously mentioned, our examples run with AUTORESUME set to 1. At present, this results in extra messages in the `rm.log` file that appear to be errors; however, in the context of this process, they are not errors and should be considered transitional.

To verify the `failover` command, examine the contents of the `rm.log` file on production-primary.

```
# tail -n16 $UDTBIN/rm.log

Fri Jun 11 06:35:08 2010: Request DBA Report to ALL received.

Fri Jun 11 06:43:11 2010: Request DBA Failover to ALL received.

Fri Jun 11 06:43:11 2010 Group inventory_file_g1 Replication to standby is
suspended (DBA Ordered).
Fri Jun 11 06:43:11 2010 Group payroll_acct_g2 Replication to standby is
suspended (DBA Ordered).
Fri Jun 11 06:43:11 2010 Group inventory_acct_g0 Replication to standby is
suspended (DBA Ordered).
Fri Jun 11 06:43:11 2010 Group payroll_file_g3 Replication to standby is
suspended (DBA Ordered).
Fri Jun 11 06:43:11 2010 Publishing group inventory_file_g1 stopped
successfully.
Fri Jun 11 06:43:11 2010 Publishing group payroll_acct_g2 stopped successfully.
Fri Jun 11 06:43:11 2010 Publishing group inventory_acct_g0 stopped
successfully.
Fri Jun 11 06:43:11 2010 Publishing group payroll_file_g3 stopped successfully.
```

Messages resulting from the suspend command

```
RM: Failover group inventory_file_g1 to subscriber, subscribing to system
standby
RM: Failover group payroll_acct_g2 to subscriber, subscribing to system standby
RM: Failover group inventory_acct_g0 to subscriber, subscribing to system
standby
RM: Failover group payroll_file_g3 to subscriber, subscribing to system standby

Successfully failover ALL to standby
```

Messages indicating failover success

```
Tue May 15 11:28:42 No publishing group defined for the Replication(2060)!
Auto Resuming Synced Replication subscribing from standby failed.

#
```

Messages indicating expected sync command failure

This example shows the sequence of steps performed by the replication subsystem after the `failover` command was issued. The DBA `Failover` command was received, replication was suspended on the four groups, and the groups were successfully failed over to SYSTEM standby. The messages after the “Successfully failover ALL to standby” message are to be expected at this stage. The `failover` command has not yet been issued to SYSTEM standby. As AUTORESUME is set to 1, the “Auto Resuming” sync command also failed.

Before continuing the failover procedure, you need to confirm the success of the `ud_repadmin` command on the other system. This is accomplished by reviewing the `rm.log` file on production-standby to make sure the command was received and processed successfully.

```
# tail -11 $UDTBIN/rm.log

Fri Jun 11 06:43:11 2010: Remote request Sync to Replication publishing to
primary received.
Fri Jun 11 06:43:11 No publishing group defined for the Replication(2060)!
Remote System Requested Synced Replication publishing to primary failed.
```

Messages indicating expected sync command failure

```
Fri Jun 11 06:43:12 2010 Subscribing Group payroll_acct_g2 started suspending
(Publisher Requested).
Fri Jun 11 06:43:13 2010 Subscribing Group inventory_acct_g0 started suspending
(Publisher Requested).
Fri Jun 11 06:43:13 2010 Subscribing Group payroll_file_g3 started suspending
(Publisher Requested).
Fri Jun 11 06:43:13 2010 Subscribing Group inventory_file_g1 started suspending
(Publisher Requested).
Fri Jun 11 06:43:14 2010 Subscribing group payroll_acct_g2 stopped
successfully.
Fri Jun 11 06:43:14 2010 Subscribing group inventory_acct_g0 stopped
successfully.
Fri Jun 11 06:43:14 2010 Subscribing group payroll_file_g3 stopped
successfully.
Fri Jun 11 06:43:14 2010 Subscribing group inventory_file_g1 stopped
successfully
#
```

This example shows the sequence of steps that have been performed by the replication subsystem after the `failover` command was issued. The four replication groups have been suspended after a request initiated from production-primary (publisher) and have all stopped successfully. The messages related to the “No publishing group defined” and “Remote System Requested Synced” errors are to be expected for the reasons explained previously.

The next step is to issue the failover command on production-standby, instructing the subscribing system to become the failed over publisher:

```
# $UDTBIN/ud_repadmin failover -verbose -pub
failovering ALL...
Command succeeded!
Subscribing Group inventory_acct_g0 is PUB_RUNNING:.
Subscribing Group inventory_file_g1 is PUB_RUNNING:.
Subscribing Group payroll_acct_g2 is PUB_RUNNING:.
Subscribing Group payroll_file_g3 is PUB_RUNNING:.
#
```

Note: After the `failover` command has been issued, production-standby server is referred to as the “failed over publisher.”

Before continuing the failover procedure, you need to confirm the success of the `ud_repadmin` command. This is accomplished by reviewing the `rm.log` file on production-standby to make sure the command was received and processed successfully.

```
# tail -28 $UDTBIN/rm.log

Fri Jun 11 07:45:03 2010: Request DBA Failover to ALL received.

RM: udsub 749656 is started failover group inventory_acct_g0.
RM: udsub 634898 is started failover group inventory_file_g1.
RM: udsub 352400 is started failover group payroll_acct_g2.
RM: udsub 933992 is started failover group payroll_file_g3.

Fri Jun 11 07:45:05 2010 Subscribing group payroll_file_g3 stopped
successfully.
RM: Failover group payroll_file_g3 to publisher
udpub for group payroll_file_g3(3) is started, pid = 987354.
Fri Jun 11 07:45:06 2010 Subscribing group inventory_acct_g0 stoppe
successfully.
RM: Failover group inventory_acct_g0 to publisher
udpub for group inventory_acct_g0(0) is started, pid = 393310.
Fri Jun 11 07:45:06 2010 Subscribing group inventory_file_g1 stoppe
successfully.
RM: Failover group inventory_file_g1 to publisher
udpub for group inventory_file_g1(1) is started, pid = 323804.
Fri Jun 11 07:45:06 2010 Subscribing group payroll_acct_g2 stopped
successfully.
RM: Failover group payroll_acct_g2 to publisher
udpub for group payroll_acct_g2(2) is started, pid = 499854.
```

Start subscriber processes in failover mode to load any LRF logs created during previous suspension

Stop failover mode subscriber processes and launch publisher processes

```
Successfully failover ALL to publisher

Auto Resuming Synced GROUP payroll_acct_g2, SYSTEM primary successfully
Auto Resuming Synced GROUP inventory_acct_g0, SYSTEM primary successfully
Auto Resuming Synced GROUP payroll_file_g3, SYSTEM primary successfully
Auto Resuming Synced GROUP inventory_file_g1, SYSTEM primary successfully

Auto Resuming Synced Replication publishing to primary successfully
```

Sync initiated as AUTORESUME=1

```
Fri Jun 11 07:45:06 2010 Group payroll_acct_g2 Replication to primary
synchronized successfully and resumed running.
Fri Jun 11 07:45:06 2010 Group inventory_acct_g0 Replication to primary
synchronized successfully and resumed running.
Fri Jun 11 07:45:06 2010 Group payroll_file_g3 Replication to primary
synchronized successfully and resumed running.
Fri Jun 11 07:45:06 2010 Group inventory_file_g1 Replication to primary
synchronized successfully and resumed running.
#
```

All LRF logs successfully transferred to failed over subscriber

This example shows the sequence of steps that the Replication subsystem has performed after the `ud_repadmin failover` command was issued to make production-standby the publisher. The four groups have failed over successfully. Because AUTORESUME is set to 1 and the replication modes now match on both systems, replication was successfully synchronized and continued running.

Note: If you have multiple subscribing servers associated with the publishing server, at this point you need to let the other subscribing servers know which server is now acting as the publisher. For instance, if you have a reporting server in addition to your failover server, you need to run a `failover` command on the reporting server:

```
$UDTBIN/ud_repadmin failover –verbose –subto standby
```

You can now perform a final confirmation. A `showud` command on production-standby shows that the server is no longer running any subscriber-based processes. It is running only publisher-based processes.

```
# $UDTBIN/showud
      USER     PID     TIME COMMAND
      root  536696     0:00 /disk1/ud72/bin/aimglog 0 25883
      root  598052     0:00 /disk1/ud72/bin/aimglog 1 25883
      root  249878     0:00 /disk1/ud72/bin/bimglog 2 25883
      root  671848     0:00 /disk1/ud72/bin/bimglog 3 25883
      root  663718     0:01 /disk1/ud72/bin/cleanupd -m 10 -t 20
      root  368694     0:00 /disk1/ud72/bin/cm 25883
      root  422086     0:00 /disk1/ud72/bin/publistener 11 0 0 1 1 0 0
      root  446580     0:00 /disk1/ud72/bin/publistener 11 0 0 1 1 2 0
      root  815332     0:00 /disk1/ud72/bin/publistener 11 0 0 1 1 1 0
      root  991362     0:00 /disk1/ud72/bin/publistener 11 0 0 1 1 3 0
      root  606392     0:00 /disk1/ud72/bin/repmanager
      root  319682     0:00 /disk1/ud72/bin/sbcs -r
      root  774274     0:00 /disk1/ud72/bin/sm 60 2617
      root  675934     0:00 /disk1/ud72/bin/smm -t 60
      root  393310     0:00 /disk1/ud72/bin/udpub -g0
      root  323804     0:00 /disk1/ud72/bin/udpub -g1
      root  499854     0:00 /disk1/ud72/bin/udpub -g2
      root  987354     0:00 /disk1/ud72/bin/udpub -g3
      root  377046     0:00 /disk1/ud72/unishared/unirpc/unirpcd
```

Before continuing the failover procedure, the success of the `ud_repadmin` command needs to be confirmed on the other system. This is accomplished by reviewing the `rm.log` file to make sure the command was received and processed successfully.

```
# tail -4 $UDTBIN/rm.log
Tue Jul 19 06:55:44 2011 Subscribing Group payroll_acct_g2 synchronized
successfully to publisher and resumed running.
Tue Jul 19 06:55:44 2011 Subscribing Group payroll_file_g3 synchronized
successfully to publisher and resumed running.
Tue Jul 19 06:55:44 2011 Subscribing Group inventory_acct_g0 synchronized
successfully to publisher and resumed running.
Tue Jul 19 06:55:44 2011 Subscribing Group inventory_file_g1 synchronized
successfully to publisher and resumed running.
#
```

This example shows the steps that the replication subsystem has performed after the `ud_repadmin failover` command was issued to make production-standby the failed over publisher. The replication modes now match, and because AUTORESUME is set to 1, production-primary is now correctly running as the failed over subscriber.

Confirmation of the process can be performed by issuing a `showud` command on the production-primary server. This shows that it is no longer running any publisher-based processes and is running only subscriber-based processes.

```
# $UDTBIN/showud
      USER     PID     TIME COMMAND
      root  765964     0:00 /disk1/ud72/bin/aimglog 0 27947
      root  458758     0:00 /disk1/ud72/bin/aimglog 1 27947
      root  848054     0:00 /disk1/ud72/bin/bimglog 2 27947
      root  503822     0:00 /disk1/ud72/bin/bimglog 3 27947
      root  729114     0:01 /disk1/ud72/bin/cleanupd -m 10 -t 20
      root  176174     0:00 /disk1/ud72/bin/cm 27947
      root  634960     0:00 /disk1/ud72/bin/repmanager
      root  516200     0:00 /disk1/ud72/bin/sbcs -r
      root  897256     0:00 /disk1/ud72/bin/sm 60 5113
```

```
      root  524382     0:00 /disk1/ud72/bin/smm -t 60
      root  786618     0:06 /disk1/ud72/bin/syncd 0 27947
      root  553174     0:07 /disk1/ud72/bin/syncd 1 27947
      root  356598     0:00 /disk1/ud72/bin/udrw -g 2 -s 4
      root  376972     0:00 /disk1/ud72/bin/udrw -g 1 -s 4
      root  413910     0:00 /disk1/ud72/bin/udrw -g 1 -s 4
      root  508148     0:00 /disk1/ud72/bin/udrw -g 3 -s 4
      root  549044     0:00 /disk1/ud72/bin/udrw -g 0 -s 4
      root  561160     0:00 /disk1/ud72/bin/udrw -g 3 -s 4
      root  577656     0:00 /disk1/ud72/bin/udrw -g 2 -s 4
      root  676062     0:00 /disk1/ud72/bin/udrw -g 0 -s 4
      root  704548     0:00 /disk1/ud72/bin/udrw -g 3 -s 4
      root  708666     0:00 /disk1/ud72/bin/udrw -g 3 -s 4
      root  774340     0:00 /disk1/ud72/bin/udrw -g 2 -s 4
      root  794878     0:00 /disk1/ud72/bin/udrw -g 1 -s 4
      root  811030     0:00 /disk1/ud72/bin/udrw -g 1 -s 4
      root  823476     0:00 /disk1/ud72/bin/udrw -g 2 -s 4
      root  880740     0:00 /disk1/ud72/bin/udrw -g 0 -s 4
      root  913412     0:00 /disk1/ud72/bin/udrw -g 0 -s 4
      root  364638     0:00 /disk1/ud72/bin/udsub 4 3600 0
      root  594042     0:00 /disk1/ud72/bin/udsub 4 3600 0
      root  622638     0:00 /disk1/ud72/bin/udsub 4 3600 0
      root  798728     0:00 /disk1/ud72/bin/udsub 4 3600 0
      root  905450     0:00 /disk1/ud72/unishared/unirpc/unirpcd
#
```

The final two steps to confirm that the failover process has completed successfully are to (1) run `ud_repadmin` on both machines to report their new states and (2) make sure that files on the failed over subscriber are read-only.

Failed over subscriber production-primary

```
# $UDTBIN/ud_repadmin report -detail -verbose
reporting ALL...
Replication subscribing from standby is SUB_RUNNING:Sync Succeeded.
GROUP inventory_acct_g0,SYSTEM standby is SUB_RUNNING:Sync Succeeded.
GROUP inventory_file_g1,SYSTEM standby is SUB_RUNNING:Sync Succeeded.
GROUP payroll_acct_g2, SYSTEM standby is SUB_RUNNING:Sync Succeeded.
GROUP payroll_file_g3, SYSTEM standby is SUB_RUNNING:Sync Succeeded.
#
```

Failed over publisher production-standby

```
# $UDTBIN/ud_repadmin report -detail -verbose
reporting ALL...
Replication publishing to primary is REP_RUNNING:Sync Succeeded.
GROUP inventory_acct_g0,SYSTEM primary is REP_RUNNING:Sync Succeeded.
GROUP inventory_file_g1,SYSTEM primary is REP_RUNNING:Sync Succeeded.
GROUP payroll_acct_g2,SYSTEM primary is REP_RUNNING:Sync Succeeded.
GROUP payroll_file_g3,SYSTEM primary is REP_RUNNING:Sync Succeeded.
#
```

Failed over subscriber production-primary

```
# $UDTBIN/udt

UniData Release 7.2  Build: (3786)
(c) Copyright Rocket Software, Inc. 1988-2009.
All rights reserved.

Current UniData home is /disk1/ud72/.
Current working directory is /disk1/replication/udt72/inventory.
```

```
:AE CLIENTS 1
Top of "1" in "CLIENTS", 3 lines, 26 characters.
*--: FI
In /disk1/ud72/sys/CTLG/a/AE_AE at line 2214 Adding/updating record is
not allowed on non-writeable subscribing file CLIENTS(19904,2).
[AE] UniBasic WRITE failed, STATUS=1, check triggers.
Quit "1" in file "CLIENTS" unchanged.
:QUIT
#
```

Hard failover

A hard failover occurs when the publishing server has a catastrophic unplanned failure, and one of the subscribing servers temporarily becomes the publishing server until the publishing server can resume control. If a hard failover and failback were to occur on a server that uses by UniData's Recoverable File System (RFS) along with replication, these features provide extra protection and significantly enhance recoverability.

If the publishing server fails, examine the `rm.log` on the subscribing server to determine the state of replication on the subscriber server.

The following example shows the results of publishing server failure. In this case, the failure occurred because the administrator did not stop the database before shutting down the publishing server.

If you have access to the publishing server, check the logs to see what happened. Then depending on the cause of failure, you might be able to restart the database without having to fail over. This needs to be determined on a case-by-case basis, and is an option only if both systems are protected by RFS.

For this example, we assume that we need to fail over in order to allow work to continue, while the original system is recovered.

The first step is to examine the `rm.log` on the subscriber.

```
# tail -10 $UDTBIN/rm.log

Fri Jun 11 10:02:42 2010 Subscribing Group payroll_acct_g2 started suspending.
Fri Jun 11 10:02:42 PubListener process is killed(3906)!

Execute Replication Exception Action program /disk1/udt72/resync_script

Cross-Group Suspension Suspended GROUP payroll_file_g3, SYSTEM primary successfully
Cross-Group Suspension Suspended GROUP inventory_file_g1, SYSTEM primary successfully
Cross-Group Suspension Suspended GROUP inventory_acct_g0, SYSTEM primary successfully

Fri Jun 11 10:02:43 2010 Subscribing group payroll_acct_g2 stopped successfully.
Fri Jun 11 10:02:43 2010 Subscribing group payroll_file_g3 stopped successfully.
Fri Jun 11 10:02:43 2010 Subscribing group inventory_acct_g0 stopped successfully.
Fri Jun 11 10:02:43 2010 Subscribing group inventory_file_g1 stopped successfully.
```

The contents of the `rm.log` indicate that replication group inventory_acct_g0 started suspending after the UNIX shutdown process killed the group's `publistener` process on the publishing server.

```
Fri Oct 14 03:27:11 2011 Subscribing Group inventory_acct_g0 started
suspending(Publisher Requested).
Fri Oct 14 03:27:11 PubListener process is killed(3906)!
```

Since this suspension was not the result of a replication command (such as `$UDTBIN/ud_repadmin suspend`), the exception action script defined in the `repsys` file ran. This is a

user-written script that can be used to notify the administrator when an unexpected suspension or resynchronization occurs. In this case, no `ud_repadmin` commands were run in the script.

When one replication group suspends, the other groups on the server are automatically suspended. Here is an example of the message for group payroll_file_g3:

```
Cross-Group Suspension Suspended GROUP payroll_file_g3, SYSTEM primary successfully
```

The `rm.log` file then reports the successful suspension of all the replication groups. From this type of failure, we can assume that the publishing server detected a problem and the database shut down; however, it is far more likely that in the event of a total system failure, all the RPC connections would be lost.

We can fail over the subscriber to become the failed over publisher, as shown in the next example.

```
# $UDTBIN/ud_repadmin failover -verbose -pub
failovering ALL...
Command succeeded!

Subscribing Group inventory_acct_g0 is PUB_RUNNING:.

Subscribing Group inventory_file_g1 is PUB_RUNNING:.

Subscribing Group payroll_acct_g2 is PUB_RUNNING:.

Subscribing Group payroll_file_g3 is PUB_RUNNING:.
#
```

The next step is to examine the contents of the `rm.log` file to make sure the command was received and processed successfully.

```
# tail -30 $UDTBIN/rm.log

Fri Jun 11 10:16:54 2010: Request DBA Failover to ALL received.

RM: udsub 356366 is started failover group inventory_acct_g0.
RM: udsub 606410 is started failover group inventory_file_g1.
RM: udsub 589946 is started failover group payroll_acct_g2.
RM: udsub 569498 is started failover group payroll_file_g3.

Fri Jun 11 10:16:57 2010 Subscribing group payroll_file_g3 stopped
successfully.
RM: Failover group payroll_file_g3 to publisher
udpub for group payroll_file_g3(3) is started, pid = 614502.
Fri Jun 11 10:16:57 2010 Subscribing group inventory_file_g1 stoppec
successfully.
RM: Failover group inventory_file_g1 to publisher
udpub for group inventory_file_g1(1) is started, pid = 905376.
Fri Jun 11 10:16:57 2010 Subscribing group payroll_acct_g2 stopped
successfully.
Fri Jun 11 10:16:57 2010 Subscribing group inventory_acct_g0 stoppec
successfully.
RM: Failover group payroll_acct_g2 to publisher
RM: Failover group inventory_acct_g0 to publisher
udpub for group payroll_acct_g2(2) is started, pid = 323832.
udpub for group inventory_acct_g0(0) is started, pid = 712834.

Successfully failover ALL to publisher

Fri Jun 11 10:16:57 Subscriber system is not running(2210)!
Auto Resuming Synced GROUP inventory_file_g1, SYSTEM primary failed.
Fri Jun 11 10:16:57 Subscriber system is not running(2210)!
```

Start subscriber processes in failover mode to load any LRF logs created during previous suspension.

Stop failover mode subscriber processes and launch publisher processes

```
Auto Resuming Synced GROUP inventory_acct_g0, SYSTEM primary failed.
Fri Jun 11 10:16:57 Subscriber system is not running(2210)!
Auto Resuming Synced GROUP payroll_acct_g2, SYSTEM primary failed.
Fri Jun 11 10:16:57 Subscriber system is not running(2210)!
Auto Resuming Synced GROUP payroll_file_g3, SYSTEM primary failed.
Fri Jun 11 10:16:57 Subscriber system is not running(2210)!
Auto Resuming Synced Replication publishing to primary failed.
#
```

Sync failed – as remote system is down – expected.

The contents of the `rm.log` show the receipt of the DBA-requested failover command, the successful failover of the replication groups, and the failure to synchronize because the remote system is not running. This is to be expected at this stage. This system is now the failed over publisher, so any updates to the replicated files are now logged to the replication log reserve files and are synchronized with the original publisher when the system is recovered.

Note: When examining the contents of the `rm.log` file on your system, if the sequence or subsequent result of the events seems not to reflect those shown in the example, this may be an indication that you need to take steps to correct the error or to let the commands complete before continuing.

Work can continue on the failed-over publisher, while you identify why the original publisher failed and what action to take to recover the system.

Failback after a soft failover

Before performing the failback procedure, all users should log off in a controlled manner to make sure that work on the replicated files has been completed. This is not completely necessary because after the failover, the files become read-only. However, logging off ensures that potential problems and extra decision-making are avoided later in the procedure.

The first step is that the failed over publisher server (the original subscriber), production-standby in our example, needs to be instructed to initiate the failback process. This is achieved by issuing the `ud_repadmin` command.

```
# $UDTBIN/ud_repadmin failover -verbose -back
failovering ALL...
Command succeeded!
Publishing Group inventory_acct_g0 is PUB_SHUTDOWN:DBA Ordered.
Publishing Group inventory_file_g1 is PUB_SHUTDOWN:DBA Ordered.
Publishing Group payroll_acct_g2 is PUB_SHUTDOWN:DBA Ordered.
Publishing Group payroll_file_g3 is PUB_SHUTDOWN:DBA Ordered.
#
```

Before continuing the failover procedure, examine the `rm.log` file to make sure the `ud_repadmin` command was received and processed successfully. After the `ud_repadmin failover -back` command, production-standby again becomes the subscriber.

```
# tail -17 $UDTBIN/rm.log

Fri Jun 11 08:35:32 2010: Request DBA Failover to ALL received.

Fri Jun 11 08:35:32 2010 Group inventory_file_g1 Replication to primary is
suspended(DBA Ordered).
Fri Jun 11 08:35:32 2010 Group inventory_acct_g0 Replication to primary is
suspended(DBA Ordered).
Fri Jun 11 08:35:32 2010 Group payroll_acct_g2 Replication to primary is
suspended(DBA Ordered).
Fri Jun 11 08:35:32 2010 Group payroll_file_g3 Replication to primary is
suspended(DBA Ordered).
Fri Jun 11 08:35:32 2010 Publishing group inventory_file_g1 stopped
successfully.
Fri Jun 11 08:35:32 2010 Publishing group inventory_acct_g0 stopped
successfully.
Fri Jun 11 08:35:32 2010 Publishing group payroll_acct_g2 stopped successfully.
Fri Jun 11 08:35:32 2010 Publishing group payroll_file_g3 stopped successfully.
```

All replication groups suspended and stopped

```
RM: Failover group inventory_file_g1 to original setting.
RM: Failover group inventory_acct_g0 to original setting.
RM: Failover group payroll_acct_g2 to original setting.
RM: Failover group payroll_file_g3 to original setting.
```

All replication groups reset to original settings

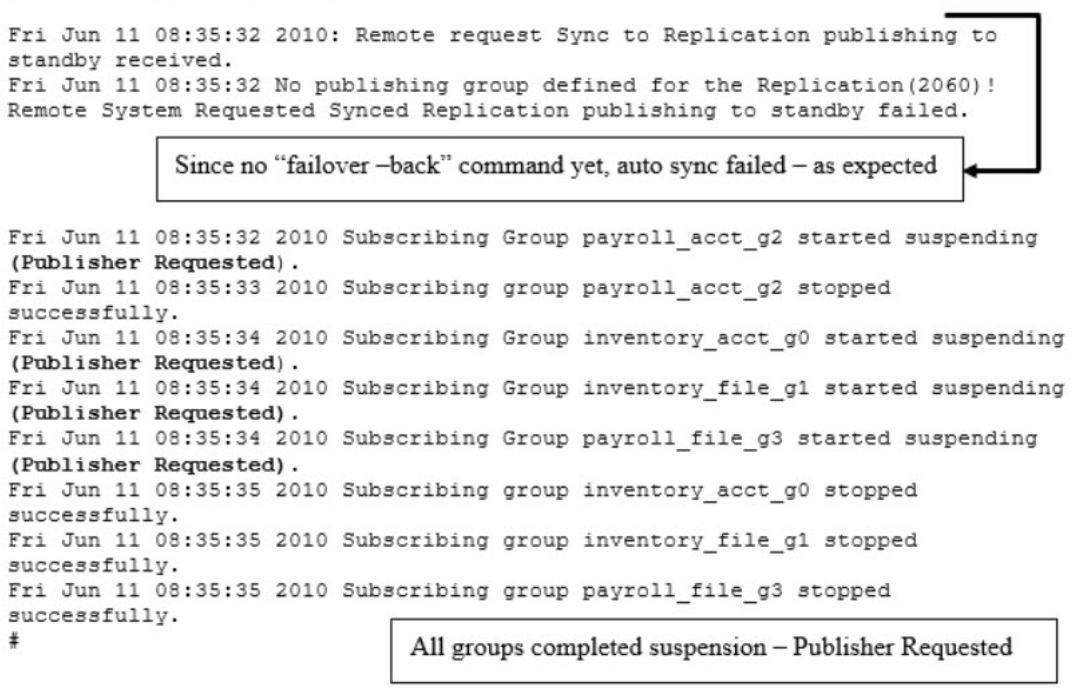
```
Successfully failover ALL to original configuration

Fri Jun 11 08:35:32 No publishing group defined for the Replication(2060)!
Auto Resuming Synced Replication subscribing from primary failed.
#
```

Since no "failover –back" command yet, auto sync failed – as expected

This example shows the steps that the Replication subsystem followed after the `ud_repadmin failover -back` command was issued. Publishing on the four groups stopped and the failover of the groups to their original settings was successful. The messages related to "No publishing group defined" and "Auto Resuming Synced" failures are to be expected at this stage, because we have yet to issue the `failback` command to the other machine and the auto sync was attempted because AUTORESUME is set to 1.

As discussed previously, setting AUTORESUME to 0 prevents these messages from appearing, but it requires the manual step of performing a sync call before continuing, and this can easily be missed. For this reason, we recommend setting AUTORESUME to 1.

Before continuing the failover procedure, examine the `rm.log` file on the other machine to make sure the `ud_repadmin` command was received and processed successfully.



This example shows the steps that the replication subsystem followed after the `ud_repadmin failover –back` command was issued from production-standby. The four replication groups were successfully suspended from a publisher request. The messages related to the "No publishing group defined" and "Remote System Requested Synced" failures are to be expected at this stage.

The procedure can continue by instructing the failed over subscriber (production-primary in our example) to become the publisher again through a failback. This can be done with the `ud_repadmin` command.

```
# $UDTBIN/ud_repadmin failover -verbose -back
failovering ALL...
Command succeeded!
Subscribing Group inventory_acct_g0 is PUB_RUNNING:.
Subscribing Group inventory_file_g1 is PUB_RUNNING:.
Subscribing Group payroll_acct_g2 is PUB_RUNNING:.
Subscribing Group payroll_file_g3 is PUB_RUNNING:.
```

Before continuing, examine the `rm.log` file to make sure the `ud_repadmin` command was received and processed successfully. After the failback has been issued, production-primary becomes the publisher again.

```
# tail -28 $UDTBIN/rm.log

Fri Jun 11 08:52:23 2010: Request DBA Failover to ALL received.

RM: udsub 798856 is started failover group inventory_acct_g0.
RM: udsub 594060 is started failover group inventory_file_g1.
RM: udsub 622640 is started failover group payroll_acct_g2.
RM: udsub 561164 is started failover group payroll_file_g3.

Fri Jun 11 08:52:25 2010 Subscribing group inventory_file_g1 stopped
successfully.
RM: Failover group inventory_file_g1 to original setting.
udpub for group inventory_file_g1(1) is started, pid = 508154.
Fri Jun 11 08:52:25 2010 Subscribing group payroll_file_g3 stopped
successfully.
RM: Failover group payroll_file_g3 to original setting.
udpub for group payroll_file_g3(3) is started, pid = 757956.
Fri Jun 11 08:52:25 2010 Subscribing group payroll_acct_g2 stopped
successfully.
Fri Jun 11 08:52:25 2010 Subscribing group inventory_acct_g0 stopped
successfully.
RM: Failover group payroll_acct_g2 to original setting.
RM: Failover group inventory_acct_g0 to original setting.
udpub for group payroll_acct_g2(2) is started, pid = 413916.
udpub for group inventory_acct_g0(0) is started, pid = 418000.

Successfully failover ALL to original configuration

Auto Resuming Synced GROUP inventory_file_g1, SYSTEM standby successfully
Auto Resuming Synced GROUP inventory_acct_g0, SYSTEM standby successfully
Auto Resuming Synced GROUP payroll_acct_g2, SYSTEM standby successfully
Auto Resuming Synced GROUP payroll_file_g3, SYSTEM standby successfully
```

Start subscriber processes in failover mode to load any LRF logs created during previous suspension

Stop failover mode subscriber processes and launch publisher processes with original configurations

```
Auto Resuming Synced Replication publishing to standby successfully
```

Sync initiated as AUTORESUME=1

```
Fri Jun 11 08:52:26 2010 Group inventory_file_g1 Replication to standby
synchronized successfully and resumed running.
Fri Jun 11 08:52:26 2010 Group inventory_acct_g0 Replication to standby
synchronized successfully and resumed running.
Fri Jun 11 08:52:26 2010 Group payroll_acct_g2 Replication to standby
synchronized successfully and resumed running.
Fri Jun 11 08:52:26 2010 Group payroll_file_g3 Replication to standby
synchronized successfully and resumed running.
#
```

All LRF logs successfully transferred to subscriber

This example shows the steps that the Replication subsystem followed after the `ud_repadmin failback` command was issued. The four replication groups were successfully failed back to their original state and restarted as publishing groups. As the replication modes between the two machines are now back in step and AUTORESUME is set to 1, the system has now completed its failback procedure.

The `showud` command provides a means for further verifying the status of the failback. This command shows that the subscriber-based processes are gone and the publishing-based processes are back.

```
# $UDTBIN/showud
USER PID TIME COMMAND
root 765964 0:00 /disk1/ud72/bin/aimglog 0 27947
root 458758 0:00 /disk1/ud72/bin/aimglog 1 27947
root 848054 0:00 /disk1/ud72/bin/bimglog 2 27947
root 503822 0:00 /disk1/ud72/bin/bimglog 3 27947
root 729114 0:01 /disk1/ud72/bin/cleanupd -m 10 -t20
root 176174 0:00 /disk1/ud72/bin/cm 27947
```

```
root 332030 0:00 /disk1/ud72/bin/publistener 11 0 0 1 1 1 0
root 602158 0:00 /disk1/ud72/bin/publistener 11 0 0 1 1 3 0
root 667900 0:00 /disk1/ud72/bin/publistener 11 0 0 1 1 0 0
root 917508 0:00 /disk1/ud72/bin/publistener 11 0 0 1 1 2 0
root 634960 0:00 /disk1/ud72/bin/repmanager
root 516200 0:00 /disk1/ud72/bin/sbcs -r
root 897256 0:00 /disk1/ud72/bin/sm 60 5113
root 524382 0:00 /disk1/ud72/bin/smm -t 60
root 786618 0:08 /disk1/ud72/bin/syncd 0 27947
root 553174 0:08 /disk1/ud72/bin/syncd 1 27947
root 418000 0:00 /disk1/ud72/bin/udpub -g0
root 508154 0:00 /disk1/ud72/bin/udpub -g1
root 413916 0:00 /disk1/ud72/bin/udpub -g2
root 757956 0:00 /disk1/ud72/bin/udpub -g3
root 905450 0:00 /disk1/ud72/unishared/unirpc/unirpcd
```

Before continuing, examine the `rm.log` file on production-standby to make sure the `ud_repadmin` command was received and processed successfully.

```
# tail -4 $UDTBIN/rm.log
Wed Nov 23 08:29:52 2011 Subscribing Group payroll_acct_g2
synchronized successfully to publisher and resumed running.
Wed Nov 23 08:29:53 2011 Subscribing Group inventory_acct_g0
synchronized successfully to publisher and resumed running.
Wed Nov 23 08:29:53 2011 Subscribing Group inventory_file_g1
synchronized successfully to publisher and resumed running.
Wed Nov 23 08:29:53 2011 Subscribing Group payroll_file_g3
synchronized successfully to publisher and resumed running.
```

This example shows the tasks that the Replication subsystem performed after the `ud_repadmin failback` command was issued. The four replication groups were successfully synchronized as the replication modes between the two machines are now back in step and AUTORESUME is set to 1.

Note: When examining the contents of the `rm.log` file on your system, if the sequence or subsequent result of the events seems not to reflect those shown in the example, this may be an indication that you need to take steps to correct the error or to let the commands complete before continuing.

Use the `showud` command on production-standby to further verify the status of the failback. This command shows the publisher-based processes are gone and the subscriber-based processes are back.

```
# $UDTBIN/showud
USER PID TIME COMMAND
root 536696 0:00 /disk1/ud72/bin/aimglog 0 25883
root 598052 0:00 /disk1/ud72/bin/aimglog 1 25883
root 249878 0:00 /disk1/ud72/bin/bimglog 2 25883
root 671848 0:00 /disk1/ud72/bin/bimglog 3 25883
root 663718 0:01 /disk1/ud72/bin/cleanupd -m 10 -t 20
root 368694 0:00 /disk1/ud72/bin/cm 25883
root 606392 0:00 /disk1/ud72/bin/repmanager
root 319682 0:00 /disk1/ud72/bin/sbcs -r
root 774274 0:00 /disk1/ud72/bin/sm 60 2617
root 675934 0:00 /disk1/ud72/bin/smm -t 60
root 352404 0:00 /disk1/ud72/bin/udrw -g 1 -s 4
root 393312 0:00 /disk1/ud72/bin/udrw -g 1 -s 4
root 422088 0:00 /disk1/ud72/bin/udrw -g 1 -s 4
root 516250 0:00 /disk1/ud72/bin/udrw -g 2 -s 4
root 540854 0:00 /disk1/ud72/bin/udrw -g 2 -s 4
root 569484 0:00 /disk1/ud72/bin/udrw -g 3 -s 4
```

```
root 589934 0:00 /disk1/ud72/bin/udrw -g 3 -s 4
root 594132 0:00 /disk1/ud72/bin/udrw -g 3 -s 4
root 618604 0:00 /disk1/ud72/bin/udrw -g 0 -s 4
root 643108 0:00 /disk1/ud72/bin/udrw -g 1 -s 4
root 659600 0:00 /disk1/ud72/bin/udrw -g 2 -s 4
root 688298 0:00 /disk1/ud72/bin/udrw -g 0 -s 4
root 712812 0:00 /disk1/ud72/bin/udrw -g 3 -s 4
root 749658 0:00 /disk1/ud72/bin/udrw -g 0 -s 4
root 933994 0:00 /disk1/ud72/bin/udrw -g 0 -s 4
root 987356 0:00 /disk1/ud72/bin/udrw -g 2 -s 4
root 446582 0:00 /disk1/ud72/bin/udsub 4 3600 0
root 815354 0:00 /disk1/ud72/bin/udsub 4 3600 0
root 905354 0:00 /disk1/ud72/bin/udsub 4 3600 0
root 991484 0:00 /disk1/ud72/bin/udsub 4 3600 0
root 377046 0:00 /disk1/ud72/unishared/unirpc/unirpcd
#
```

Failback after a hard failover

As with most systems running RFS, it is probably best to let UniData restart and follow its crash recovery procedure. In most cases, this ensures that UniData protects the physical integrity of the files, along with the transactional integrity of any updates. As covered earlier, when running RFS, UniData can make a decision on how to restart the replication system (as a publisher or as a failed over subscriber). Most of the time, it's best to restart the replication system as a failed over subscriber. Without RFS, it always restarts as a publisher. Also, with RFS, UniData is able to populate the `REP_RECV_LOG` files, which relate to possible lost transactions if you are using immediate mode.

In our example, both the publishing and subscribing servers are running with the added protection of RFS, as we strongly recommend.

```
# $UDTBIN/startud
Using UDTBIN=/disk1/ud72/bin
All output and error logs have been saved to ./saved_logs directory.
SMM is started.
Unirpcd has already been started
SBCS is started.
SM is started.
RM is started.
CLEANUPD is started.
UniData R7.2 has been started.
#
```

The contents of the `rm.log` and the RFS `sm.log` should be examined in order to see what happened during the UniData restart.

```
# cat $UDTBIN/rm.log
Starting: Fri Jun 11 12:30:51 2010
Starting: Fri Jun 11 12:30:51 2010

UniData System Version: 7.2

Replication Protocol: 10004

IPC facilities:

  q - 1048596 (request queue)
  q - 1048595 (reply queue)
  m - 1048586 (ctl)
  s - 1048597 (waiting)
  s - 1048596 (waiting)
  s - 1048595 (waiting)
  s - 1048594 (waiting)
  s - 1048593 (waiting)
  s - 1048592 (waiting)
  s - 1048591 (waiting)
  s - 1048590 (waiting)
  s - 1048589 (waiting)
  s - 1048588 (waiting)
  s - 1048587 (waiting)
  s - 1048586 (waiting)
```

Recovery process was automatically run, starting Replication as a failed over subscriber

```
Replication start replication recovery, pid=221340
RM: Start Replication Recovery for group inventory_acct_g0
RM: group inventory_acct_g0 has failed over to standby
RM: Failover group inventory_acct_g0 to subscriber, subscribing to system
standby
RM: RW 434420 is started recovering group inventory_acct_g0.
RM: Start Replication Recovery for group inventory_file_g1
RM: group inventory_file_g1 has failed over to standby
RM: Failover group inventory_file_g1 to subscriber, subscribing to system
standby
RM: RW 409828 is started recovering group inventory_file_g1.
RM: Start Replication Recovery for group payroll_acct_g2
RM: group payroll_acct_g2 has failed over to standby
RM: Failover group payroll_acct_g2 to subscriber, subscribing to system standby
RM: RW 405706 is started recovering group payroll_acct_g2.
RM: Start Replication Recovery for group payroll_file_g3
RM: group payroll_file_g3 has failed over to standby
RM: Failover group payroll_file_g3 to subscriber, subscribing to system standby
RM: RW 401620 is started recovering group payroll_file_g3.
Started Publishing Group inventory_file_g1 successfully!
Started Publishing Group payroll_file_g3 successfully!
Started Publishing Group payroll_acct_g2 successfully!
Started Publishing Group inventory_acct_g0 successfully!

Successfully Started.
Auto Resuming Synced Replication subscribing from standby successfully

Fri Jun 11 12:30:54 2010 Subscribing Group payroll_file_g3 synchronized
successfully to publisher and resumed running.
Fri Jun 11 12:30:54 2010 Subscribing Group inventory_file_g1 synchronized
successfully to publisher and resumed running.
Fri Jun 11 12:30:55 2010 Subscribing Group inventory_acct_g0 synchronized
successfully to publisher and resumed running.
Fri Jun 11 12:30:55 2010 Subscribing Group payroll_acct_g2 synchronized
successfully to publisher and resumed running.
#
```

Automatic sync as AUTORESUME=1

The `rm.log` file shows that during the restart of UniData, the replication recovery process was automatically run and UniData decided to start the replication system as a failed over subscriber. It

made this decision because the production-standby system had already been failed over and RFS was being used. This is noted in the following messages in the `rm.log`.

```
Replication start replication recovery, pid=221340

RM: Start Replication Recovery for group inventory_acct_g0
RM: group inventory_acct_g0 has failed over to standby
RM: Failover group inventory_acct_g0 to subscriber, subscribing to system standby
RM: RW 434420 is started recovering group inventory_acct_g0.

RM: Start Replication Recovery for group inventory_file_g1
RM: group inventory_file_g1 has failed over to standby
RM: Failover group inventory_file_g1 to subscriber, subscribing to system standby
RM: RW 409828 is started recovering group inventory_file_g1.

RM: Start Replication Recovery for group payroll_acct_g2
RM: group payroll_acct_g2 has failed over to standby
RM: Failover group payroll_acct_g2 to subscriber, subscribing to system standby
RM: RW 405706 is started recovering group payroll_acct_g2.

RM: Start Replication Recovery for group payroll_file_g3
RM: group payroll_file_g3 has failed over to standby
RM: Failover group payroll_file_g3 to subscriber, subscribing to system standby
RM: RW 401620 is started recovering group payroll_file_g3.

Started Publishing Group inventory_file_g1 successfully!
Started Publishing Group payroll_file_g3 successfully!
Started Publishing Group payroll_acct_g2 successfully!
Started Publishing Group inventory_acct_g0 successfully!
```

The `rm.log` also notes the automatic resynchronization of the two systems. At this stage, any updates that had been applied on the failed over publisher (production-standby) have been replayed onto the failed over subscriber (production-primary).

The contents of the RFS system manager log file (`sm.log`) can also be examined in order to see the combination of the RFS and Replication recovery.

```
# cat $UDTBIN/sm.log
----- SM (364674) is started at Jun 11 2010 12:30:21 -----

Unidata Environment : /usr/ud72/include

Fri Jun 11 12:30:21 SM: Restart_Flag = 1.
Starting: Fri Jun 11 12:30:31 2010

Starting: Fri Jun 11 12:30:31 2010

UniData System Version: 7.2

Replication Protocol: 10004

IPC facilities:

  q - 19 (request queue)
  q - 20 (reply queue)
  m - 10 (ctl)
  s - 10 (waiting)
  s - 11 (waiting)
  s - 12 (waiting)
  s - 13 (waiting)
  s - 14 (waiting)
  s - 16 (waiting)
  s - 17 (waiting)
  s - 18 (waiting)
  s - 19 (waiting)
  s - 20 (waiting)
  s - 21 (waiting)

Replication start in RFS recovery, pid=258200
Successfully Started.
recovery: RM started, pid:258200
Undo logfile[2] .....
The logfile[2] has been scanned.
Undo logfile[2] has been undone.
Undo logfile[3] .....
The logfile[3] has been scanned.
Undo logfile[3] has been undone.
Starting to restart ..........
restart: Report of Log-file Status.
   #     Type          Checkpoint-Number     Status
   0     after-image                   0          0
   1     after-image                   0          4
   2     before-image                  0          4
   3     before-image                  0          4
   4     after-image                   0          4
```

Previous database failure detected.
RFS recovery mode initiated

At UniData **7.1.10**, new status messages are recorded in the sm.log at appropriate points:
"RFS recovery has finished; now waiting Replication recovery…"
"Replication Recovery has finished, the system is normally running."

```
   5     after-image                  0         4
   6     before-image                 0         4
   7     before-image                 0         4
Link back all new blocks ....
Link new blocks successful !
Start the redo processing ....

Finish the redo processing.
All undo/redo recovery processing has been finished!!

Redo logfiles finished.

Step3: Check the file level commands...

File level commands checking finished.

Restart is successful!!!

*****!!! Restart Finished !!!*****

Fri Jun 11 12:30:51 2010: Request StopRM to ALL received.
Stoped Publishing Group inventory_acct_g0 successfully.
Stoped Publishing Group inventory_file_g1 successfully.
Stoped Publishing Group payroll_acct_g2 successfully.
Stoped Publishing Group payroll_file_g3 successfully.
Fri Jun 11 12:30:51 2010  stop RM successfully.

RM successfully shutdown.
RM stopped successfully.
recovery: RM stopped.
Checking log files .....
RFS recovery has finished; now waiting Replication recovery......
Fri Jun 11 12:30:52 Replication Recovery has finished, the system is normally
running.
```

This set of messages means that replication processes launched during RFS recovery have completed their work and exited. You need to review the rm.log to see the actual status of normal replication processes after this successful restart of the database.

The `sm.log` file indicates that UniData detected a system failure that occurred earlier:

```
SM: Restart_Flag = 1.
```

The other information in the `sm.log` shows the replay of the RFS logs and the subsequent recovery process of the replication system.

Note: When examining the contents of the `rm.log` and `sm.log` files on your system, if the sequence or subsequent result of the events seems not to reflect those shown in the example, this may be an indication that you need to take steps to correct the error or to let the commands complete before continuing.

A final check of the `rm.log` file on the failed over publisher also shows that the process was successful.

```
# tail -10 $UDTBIN/rm.log
Fri Jun 11 12:29:30 2010: Remote request Sync to Replication publishing to
primary received.

Remote System Requested Synced GROUP inventory_file_g1, SYSTEM primary
successfully
Remote System Requested Synced GROUP payroll_file_g3, SYSTEM primary
successfully
Remote System Requested Synced GROUP payroll_acct_g2, SYSTEM primary
successfully
Fri Jun 11 12:29:31 2010 Group inventory_file_g1 Replication to primary
synchronized successfully and resumed running.
Fri Jun 11 12:29:31 2010 Group payroll_file_g3 Replication to primary
synchronized successfully and resumed running.
Remote System Requested Synced GROUP inventory_acct_g0, SYSTEM primary
successfully
Fri Jun 11 12:29:32 2010 Group payroll_acct_g2 Replication to primary
synchronized successfully and resumed running.
```

```
Fri Jun 11 12:29:32 2010 Group inventory_acct_g0 Replication to primary
synchronized successfully and resumed running.

Remote System Requested Synced Replication publishing to primary successfully
#
```

The two systems are now failed over and running. The process for completing the failback is the same as described in the “Failback after a soft failover” section of this document.

However, before following those steps, please review the information in the replication recovery `REP_RECV_LOG`. These logs may contain information you will find useful in making sure that the logical integrity of your database is not compromised.

Data recovery

Data recovery relies on crash recovery and media recovery available with RFS. With RFS, UniData is able to recover both the physical structure and logical content of the database. Without RFS, your database administrator must manually recover the physical structure of the database by executing file diagnostic tools, such as guide, followed by file repair tools, if necessary.

Recovering the publishing server running RFS

If the primary publishing server crashes, you decide whether to recover the system as a primary publishing server or failover to a standby server. After successful failover, the original primary server should be recovered as a subscribing server of the new, failed over standby server.

When updating a publishing object, the RFS recovery process creates a replication recovery log and leaves it for the replication recovery process, rather than applying the update directly to the database. The replication recovery process determines each of the replication groups that must be failed over to a subscribing group, or starts the replication group as a publishing group.

If you start the replication group as a publishing group, the replication recovery process applies the replication recovery logs to the database. If a replication group is failed over as a subscribing group, UniData applies the missing updates that occurred during the period of the system crash in the data synchronization process.

The following steps describe the procedure for recovering a publishing server running RFS:

1. Recover the operating system, if necessary.
2. Restart UniData.
3. UniData determines if crash recovery is necessary.
4. If crash recovery is necessary, the restart process applies the bimg logs to make the database physically consistent.
5. The restart process applies the aimg logs. If a log for a publishing object exists, UniData generates a replication recovery log rather than applying the update to the database.
6. When the restart process finished, UniData starts the Replication Manager.
7. The Replication Manager determines whether the replication group recovers as a publishing group or a subscribing group, based on the following:
   - If the replication group does not configure any standby subscribing servers, it starts as a publishing group. Otherwise,
   - The Replication Manager tries to contact each of the remote Replication Managers of the standby subscribing servers to see if the replication group failed over to that system. If so,

the Replication Manager starts the group as a subscriber group subscribing to that system. Otherwise,

- The Replication Manager follows the failover definition in the replication group configuration to determine if it should start the group as a publishing group or a subscribing group to the specified system.

8. If UniData starts the group as a publishing group, the Replication Manager creates Replication Writer processes to apply replication recovery logs generated by the Restart process to the database, and saves the logs to the replication log reserve file.
9. For real-time replication, if UniData starts the group as a subscribing group, the Replication Manager removes the recovery replication logs. For immediate replication, the Replication Manager leaves the recovery replication logs for synchronization.

Recovering the publishing server not running RFS

If you are not running RFS, you should run UniData's file diagnostic and fixing tools to recover the physical structure of the database. Since there is no crash recovery process without RFS, there is no effective way to recover possibly lost transactions during the system failure. Failover to a real-time standby server is the only way to guarantee all committed transactions contain their full updates, and thus the consistency of the database.

When you restart UniData without RFS, a publishing group always restarts as a publishing group, even if a `failover` occurred on the standby server. In this case, the database administrator must manually failover the replication group to a subscribing group using the failover command after UniData restarts.

After a non-RFS crash of the publishing system, the best version of your data will likely be on the subscribing server you failed over to. After recovering the publishing server, you may want to consider refreshing the data on this system from the subscribing server (now the "failed over publisher") before failing back.

Recovery of a subscribing server

Recovery of a subscribing server is the same as normal recovery of a UniData system. The database administrator must manually start the system and run UniData's warmstart or file diagnostic and repair tools to recover the physical structure of the database. If you do not have RFS active on the subscribing server, you may want to consider refreshing the data from the publishing server to be sure the data file content is consistent with what exists on the publishing server.

Data resynchronization

Resynchronization is the first data synchronization after crash recovery. Synchronizing the data recovers possible missing updates that may have occurred during the crash, and makes the database consistent.

Resynchronization after restarting publishing server

When a publishing group restarts as a publishing group, the replication recovery process applies and saves replication recovery logs to the Replication Log Reserve File. The resynchronization process sends all replication recovery logs, as well as other logs in the Replication Log Reserve File, to synchronize the subscribing database to the publishing database.

Resynchronization after failover using real-time replication

Real-time replication guarantees that any committed update on the primary server delivered its logs to the standby server. After failover to a real-time standby subscribing server, UniData applies all committed updates and saves them as replication failover logs on the standby server. After recovery of the primary server is complete, the resynchronization process sends the replication failover logs to the primary server to synchronize it with the failed over standby server. This process also sends all follow-up replication logs to the primary server to ensure the system is up-to-date.

The resynchronization process is same described in Installing U2 Data Replication for UniData, on page 44.

Resynchronization after failover using immediate replication

After you fail over to an immediate standby subscribing server applications resume running on the standby system.

As described in Recovering the publishing server running RFS, on page 185, the RFS crash recovery process generates replication recovery logs for possible missing updates from the RFS crash recovery procedure. After you failover to an immediate standby subscribing server, applications resume running on the standby server. Resynchronization of the primary server to the failed over standby server not only synchronizes the database on the previous primary server to the failed over database, but also generates a crash log record, which contains the lost updates that occurred during the primary server crash.

The following steps describe the resynchronization process when using Immediate Replication.

1. The replication manager requests that the replication manager on the remote failed over system return the failover logs generated from the failover process.
2. The remote replication manager sends the request to the publisher processes.
3. The remote publishing server invokes the subscriber on the local system, and creates a connection between them.
4. The remote publishing server creates the publisher-syncing tool on the remote system. The publisher-syncing Tool reads the failover logs from the replication log reserve file and sends them to the subscriber.
5. The subscriber process invokes the replication writer process to apply the failover logs to the database.
6. The subscriber process compares the replication recovery logs generated by the RFS restart process to the failover logs and generates the failover log file
7. When the replication writer process reaches the failover end log, it generates a crash log record, including those replication recovery logs that do not exist in the failover log.
8. The replication manager synchronizes the primary server to the failed over system, making its database current.
9. If the REP_RECV_LOG contains recovery logs representing missing transactions during the crash, the database administrator must determine if the logs need to be applied to the failed over database. You can copy these logs using XAdmin, or write a program to accomplish this task.

Resynchronization after recovery from subscribing server or network failure

If a subscribing server crashes or a network failure occurs, the publishing server saves replication logs that do not contain sub-commit notification logs to the replication log reserve file. After recovery from

this type of failure, the resynchronization process applies the saved logs to the subscribing server so that it is consistent with the publishing server.

This process happens automatically if the AUTORESUME parameter is set to 1. Otherwise, you can use the `sync` command to force synchronization.

Chapter 8: Backing up and restoring

The purpose of taking any backup is to make sure that should it be needed, it can be used to recover the database after a catastrophic failure. The UniData Recoverable File System (RFS) can be used to protect UniData file structures from non-catastrophic hardware or machine failures, and can be used to aid recovery in the event of a catastrophic failure.

To lay the groundwork for a successful recovery, you must ensure that you do not inadvertently back up incomplete updates to UniData file structures, because you would then restore the database from corrupted UniData file structures. Although this may sound obvious, it is a fact that many administrators fail to secure the UniData database before taking a backup.

The following steps ensure that UniData, RFS, and replication are secured before you take the backup. There are many ways of taking backups (for example, with mirrored disks or fast tape drives), and some extra steps may be required depending on the hardware environment

The method we discuss here is to back up a UniData database that is using replication so the servers can be recovered in the event that a restore or refresh of the data is needed.

Each step must be completed successfully before proceeding to the next step. These steps allow you to secure the UniData environment without stopping the database. UniData continues running, but updates to the database remain disabled during the backup.

Note: We recommend backing up on the subscribing server, as this allows the publishing server to continue working while the subscriber is being backed up.

Complete the following tasks:

The `dbpause` command is a UniData system-level command that enables an administrator to temporarily pause the database and prevent updates to it. This feature was designed for use before performing a task that would otherwise require users to log off the system and the administrator to stop UniData, such as backing up data.

- Backing up the subscribing server
- Backing up the publishing server
- Pausing and resuming the database
  The `dbpause` command is a UniData system-level command that enables an administrator to temporarily pause the database and prevent updates to it. This feature was designed for use before performing a task that would otherwise require users to log off the system and the administrator to stop UniVerse, such as backing up data.
- Restoring the database
  Complete the following steps in the event that you need to restore both the subscriber and publisher back to a consistent point, possibly after a catastrophic failure. In general, we recommend taking the backup from the subscriber to bring both servers back to the same point in time. You could use the image from the publisher as well, but our recommendation is to use the back up from the subscriber.

Backing up the subscribing server

1. Confirm that replication is currently active by using the `ud_repadmin` tool.
2. Confirm that the subscribed files are “up to date”.

Note: This step is not necessary if you do not need to have a backup that exactly matches the publishing server data file contents.

   a. Confirm the LSN "done" numbers for each group on the subscriber match the publisher. In the replication monitor, review PubDone, and SubDone. In `reptool`, review localDoneLSN and remoteDoneLSN.
3. Suspend replication, using the `ud_repadmin` tool to issue the suspend command.
4. Confirm that replication has suspended properly.
5. Issue the `dbpause` command, if necessary.
   a. The suspend command stops updates to replicated files. You need to take the additional step of issuing the `dbpause` command if any files on the subscribing server are updated locally. This includes non-replicated files and any replicated files that are flagged as SUB_WRITEABLE.
   b. Issue the `dbpause_status` command to confirm that `dbpause` is active before the backup.
6. Perform the backup using a utility that allows you to back up open files.
   a. This step can be sped up if you have the ability to split your file system mirrors and perform the backup from the split-off mirror.
7. If you used `dbpause` before the backup, issue the `dbresume` command.
   a. Confirms that `dbpause` is not active by issuing the `dbpause_sync` command.
8. Resume replication, using the `ud_repadmin` tool to issue the `sync` command.
9. Confirm that all groups have reached the "resumed running" state.

Parent topic: Backing up and restoring

Backing up the publishing server

1. Issue the `dbpause` command.
2. Confirm that `dbpause` is active before the backup by issuing the `dbpause_status` command.
3. Perform the backup using a utility that allows you to back up open files.
4. Issue the `dbresume` command.
5. Confirm that `dbpause` is not active by issuing the `dbpause_status` command.

Parent topic: Backing up and restoring

Pausing and resuming the database

The `dbpause` command is a UniData system-level command that enables an administrator to temporarily pause the database and prevent updates to it. This feature was designed for use before performing a task that would otherwise require users to log off the system and the administrator to stop UniVerse, such as backing up data.

The `dbpause_status` command is a UniVerse system-level command that returns information regarding the status of `dbpause`. You can include this command in a start-up script to prevent users from starting a session if a `dbpause` is active. New udt sessions are automatically blocked if the database is paused.

The `dbresume` command is a UniVerse system-level command that enables an administrator to allow updates to the database after previously issuing a `dbpause` command.

Parent topic: Backing up and restoring

Restoring the database

Complete the following steps in the event that you need to restore both the subscriber and publisher back to a consistent point, possibly after a catastrophic failure. In general, we recommend taking the backup from the subscriber to bring both servers back to the same point in time. You could use the image from the publisher as well, but our recommendation is to use the back up from the subscriber.

1. On the publisher, issue the `stopud` command.
2. On the subscriber, issue the `stopud` command.
3. Restore the publisher with the backup of the subscriber.
4. Restore the subscriber with the backup of the subscriber.
5. Remove all the files and subdirectories in the replog directory on the publisher and the subscriber. (The REP_LOG_PATH parameter in the `udtconfig` file defines the location of this directory.)
6. Issue the `startud` command on the publisher.
7. Issue the `startud` command on the subscriber.

Parent topic: Backing up and restoring

Refreshing the subscriber with a copy of the publishing server

In case of an unforeseen event leading to an extended period of outage of the subscribing system, it may become necessary to refresh the subscriber with a copy of the publishing system.

In order to refresh the database files on a subscribing server, you must have a clean backup of the published data files from the publishing server. When creating a usable backup of UniData files on a server, UniData must either be stopped (`$UDTBIN/stopud`) or paused (`$UDTBIN/dbpause`). You can minimize the interruption to users by implementing mirrored disks that provide the capability of breaking the mirrors and taking a backup or snapshot from a split-off mirror set. With `dbpause`, writes to UniData files are blocked for the time it takes for the mirrors to be split. Users and web server applications generally do not need to disconnect from the system, as is the case if the database is stopped.

There are two different procedures for refreshing the subscribing server data files:

- Refreshing the subscriber with a backup of the publisher (publisher stopped)
  If you are able to make a backup of the publishing system data files while the database is stopped, use this procedure.
- Refreshing the subscriber with a backup of the publisher (publisher suspended)
  If you are able to make a backup of the publishing server data files while the database is paused, use this procedure.
- Refreshing the subscriber without a backup of the publisher
  This procedure can be considered an “online” refresh of the data files on a subscribing system. You would need to create programs that read and write all records in all published files on a publishing system to use this technique.
- Confirming that a data refresh is complete
  If you are refreshing a data account on a subscribing system, there is often a delay between restarting (or resuming) the database on the publishing system and issuing a sync command. During this delay, replication runs in suspended mode, and the publishing server accumulates log reserve files that need to be sent to the subscriber and applied to the database. As an administrator, it is important to know how to determine when the database files on the subscriber are up to date, after the database copy has been restored and the `sync` command issued.

Refreshing the subscriber with a backup of the publisher (publisher stopped)

If you are able to make a backup of the publishing system data files while the database is stopped, use this procedure.

1. Issue the database stop command on the subscribing server, if it hasn't been stopped already.
   a. `$UDTBIN/stopud`
2. Issue the database stop command on the publishing server.
   a. `$UDTBIN/stopud`
3. Remove all the files and subdirectories in the replog directory on the publisher and subscriber.
   The replog directory is defined in the `udtconfig` file parameter: REP_LOG_PATH
4. Back up the publishing server.
5. Issue the database start command on the publishing server.
   a. `$UDTBIN/startud`
   b. This allows the publishing server to continue working while the subscriber is refreshed.
6. Use the backup image from step 4 to refresh the subscribing server.
7. Issue the database start command on the subscribing server.
   a. `$UDTBIN/startud`

Parent topic: Refreshing the subscriber with a copy of the publishing server

Refreshing the subscriber with a backup of the publisher (publisher suspended)

If you are able to make a backup of the publishing server data files while the database is paused, use this procedure.

1. Issue the database `stopud` command on the subscribing server
   a. `$UDTBIN/stopud`
   b. This will suspend replication between the systems.
2. Issue the reset command on the publishing system with the –noresume option.
   a. `$UDTBIN/ud_repadmin reset –noresume`
3. Block UniData writes on the publishing server.
   a. `$UDTBIN/dbpause`
4. Back up all replicated accounts on the publishing server.
5. a. `$UDTBIN/dbresume`
   b. This allows the publishing server to continue working while the subscriber is refreshed.
6. Use the backup image from step 4 to refresh the subscribing server.
7. Issue the database start command on the subscribing server.
   a. `$UDTBIN/startud`
8. If AUTORESUME is not set to 1, issue a sync command to restart replication.
   a. `$UDTBIN/ud_repadmin sync`
   b. This command can be issued on the subscribing or publishing server.

Note: When a `stopud` command is issued, it should be allowed to finish gracefully. If the "force" option of `stopud` was used (`stopud -f`) and RFS is present, it is essential that:

- The `startud` command is issued and allowed to complete normally, so that UniData can complete its recovery.
- The `stopud` command (with no arguments) is then used to stop UniData and RFS gracefully.

Parent topic: Refreshing the subscriber with a copy of the publishing server

Refreshing the subscriber without a backup of the publisher

This procedure can be considered an "online" refresh of the data files on a subscribing system. You would need to create programs that read and write all records in all published files on a publishing system to use this technique.

Note: Although this can be achieved using the steps below, it requires careful consideration and review of the tuneables governing replication and the Recoverable File System if RFS is being used on the publishing or subscribing server. This is because it will greatly increase the throughput of the RFS components and log structures associated with it, and could result in serious performance degradation.

1. Issue the `stopud` command on the subscribing system.
2. Change the REP_FLAG parameter to 0 on the subscribing server.
3. Issue the `startud` command on the subscribing server.
4. Clear all of the subscribed files on the subscribing server via the use of the `CLEARFILE` command. (You might consider writing a script or program to accomplish this.)
5. Issue the `stopud` command on the subscribing server.
6. Change the REP_FLAG parameter to 1 on the subscribing server.
7. Issue the `startud` command with the "-i" option on the subscribing server.
8. If AUTORESUME is not set to 1, run `ud_repadmin sync` to restart replication. This command can be issued on the subscribing server or on the publishing server.
9. Read each record in each published file on the publishing server and write them back to file.

Note: When a `stopud` command is issued it should be allowed to finish gracefully. If the "force" option of `stopud` was used (`stopud -f`) and RFS is present, it is essential that:

- The `startud` command is issued and allowed to complete normally, so that UniData can complete its recovery.
- The `stopud` command (with no arguments) is then used to stop UniData and RFS gracefully.

Parent topic: Refreshing the subscriber with a copy of the publishing server

Confirming that a data refresh is complete

If you are refreshing a data account on a subscribing system, there is often a delay between restarting (or resuming) the database on the publishing system and issuing a sync command. During this delay, replication runs in suspended mode, and the publishing server accumulates log reserve files that need to be sent to the subscriber and applied to the database. As an administrator, it is important to know

how to determine when the database files on the subscriber are up to date, after the database copy has been restored and the `sync` command issued.

There are two phases to refreshing a data account on a subscribing server:

1. Receiving pending replication logs from the publishing server
2. Applying updates to the files on the subscribing server

On the subscribing server, replication writer processes are launched and poised to read the replication logs and apply them to the files. On the publishing server, one `pubsyncer` process is launched for each replication group. The `pubsyncer` process reads the log reserve files and sends the pending replication logs to the subscribing server. Subscriber processes receive the logs and load them into the replication buffers on the subscribing server. The logs might be stored in log extension files (LEF) on the subscribing server, depending on the size of the buffer and the number of logs being processed. The replication writer processes immediately start reading the logs from the buffers and apply the updates to the database files.

After the subscribing server receives all pending replication logs, messages similar to the following are written into the `rm.log` on the publishing and subscribing server:

```
Publishing system rm.log: "Mon Jun 21 12:30:00 2010 Group
inventory_acct_g0 Replication to standby synchronized successfully and
resumed running."
```

```
Subscribing system rm.log: "Mon Jun 21 12:28:33 2010 Subscribing Group
inventory_acct_g0 synchronized successfully to publisher and resumed
running."
```

When you see messages for all of the groups you have configured, you can be confident that the subscribing server has received all replication logs. Additionally, all `pubsyncer` processes on the publishing server will have exited.

In order to determine if all of the replication logs sent during the sync process have been applied to the database files, you need to look into the log sequence numbers (LSNs) for each replication group. You can use `reptool` from the shell or the XAdmin interface. On the publishing server (for instance), when remoteDoneLSN for a group equals or exceeds lastSyncLSN for that group, you can be assured that all of the updates that were queued up during the suspension period have been applied to the database files on the subscriber. You need to check each group to be sure that all logs have been applied. You can see this from either the publishing server or the subscribing server. The following screen capture shows a XAdmin session on a publishing server with logs from the sync operation in the process of being applied to the database files. For more information on configuring replication using XAdmin, see Managing U2 Data Replication through XAdmin, on page 130.

Parent topic: Refreshing the subscriber with a copy of the publishing server

Chapter 9: Restrictions

The chapter describes restrictions when using U2 Data Replication.

File types

A U2 Data Replication object can be any UniData database file, including a static hashed file, a dynamic hashed file, a sequentially hashed file, a multilevel file, a dir-type file, and a multi-dir type file. U2 Data Replication does not support a sequential file and other operating system-level files.

File names

U2 Data Replication supports file names that contain the standard ASCII displayable characters in their VOC entry. Use of control characters (for example, ASCII 10, 13 or 26) in file names is not supported usage and can cause unexpected results including the list of files in repconfig being misread or truncated. If it is necessary to replicate a file with control characters in its file name, then a synonym should be created in the VOC without control characters in the VOC file name and the synonym used for replication purposes.

Changing inode and device numbers

U2 Data Replication uses the inode and device number to identify a replication file. The inode and device numbers are loaded when the database starts and when you run the `RECONFIG` command. If you need to change the inode or device number for a file while the database is running, you must run the `RECONFIG` command to update the replication file. If you move a file to a new location without running the `RECONFIG` command, U2 Data Replication does not recognize the inode or device number of the file.

Note: The `RESIZE` ECL command might change the inode or device number, but when the command finishes running, it reloads the inode and device number. Therefore, you do not need to run the `RECONFIG` command after you resize a file.

A replication file can belong to only one replication group

Replication objects are sets of objects in replication groups. One replication object can belong to only one replication group. If there are multiple VOC pointers referencing a single UniData file, they should be defined only once in one replication group.

If you define one replication object more than once in different groups, only one of them will be used.

System files not supported as replication objects

U2 Data Replication does not support account-level operations on the following files:

- _HOLD_
- _PH_

- _DEBUG_
- _REPORT_
- _SCREEN_
- _XML_
- _EDAMAP_
- _EDAXMAP_
- MENUFILE

Certain ECL commands, such as CATALOG and SAVE.LIST, update system directory files. Replicating these updates may cause the subscribing server to be inconsistent.

No cascading replication

You cannot use a subscribing file as a publishing file.

Writeable subscriber files

You can update a file on the subscribing server if you explicitly define it as writeable in the `repconfig` file by using SUB_WRITEABLE in the FILE phrase. Updates made to a writeable file on the subscribing server are not replicated back to the publishing server.

You can update both record-level and file-level operations on a writeable subscriber file. If a file-level operation updates the VOC or dictionary file, you must specify these files as writable in the repconfig file.

Other command replication

UniData replicates the `BASIC` command, the `CATALOG` command, and the `DELETE.CATALOG` command through the account-level group. If you do not define an account-level group, the database does not replicate these commands, even if the target file is defined as a replicated file. If you define an account-level group. UniData replicates these commands, even if the target file is not defined as a replicated file.

Virtual field definitions

U2 Data Replication supports replication of indexed virtual field values. If you do not want to reevaluate virtual field values on the subscribing server, the subscribing object must have the same virtual field definition as on the publishing object. When updating the subscriber using the values evaluated on the publishing server, the following rules apply:

- The dictionary and index definition must be the same on the publisher and subscriber.
- The virtual field indexes must have been created in the same order on both the publisher and subscriber.
- The same number of virtual field indexes must exist on the publisher and subscriber.
- The state (enabled/disabled) of the index must be the same on the publisher and subscriber.

Sequence of updates

UniData ensures the sequence of updates for the same record on a publishing object as on the subscribing object. However, on a subscribing server, the sequence of updates for different records may not be the same as on the publishing server.

U2 Data Replication only keeps the operation sequence within the same replication group. If you replicate an account-level and a file-level operation that updates a file in different replication groups, we do not guarantee that updates will be processed in the same sequence. To keep subscribing and publishing files consistent, you should define all related files in the same replication group.

Transaction limitations

- One transaction can update data in files that belong to more than one replication group. However, those replication groups involved in one transaction should have the same subscribing distributions so the transaction is fully replicated to all of the distributions. The replication groups involved should also have the same replication type, either all real-time, all immediate, or all deferred.
- System administrators can define or reconfigure U2 Data Replication while UniData is running. The new configuration only applies to transactions after the `ud_repadmin reconfig` command is executed. UniData writes logs to the Replication Log Reserve File for transactions that were committed on the publisher, but not yet replicated on the subscriber. UniData generates this replication logs using the previous configuration. The synchronization process transfers these logs to the subscriber using the new configuration information. This may cause a different processing result.

  For example, if the new configuration definition removes a replication object from a replication group, the saved logs for the removed object, if any, are still sent to the subscriber as part of data synchronization. If you configure the subscriber to remove the replication object as well, the Replication Writer process will either ignore the log if you set the RW_IGNORE_ERROR, or suspend the replication.

  Make sure all data involved in a transaction is transferred before making changes to a replication group definition.
- Although transaction consistency will eventually occur on the subscribing server, an application running on the subscribing server may read partial changes of a transaction on the publisher.

System crashes during failover

If the subscribing server crashes during the failover process, you can recover the publishing server if you are running RFS with the crash recovery process. However, the data on the subscribing server may not be synchronized with the publishing server. You may have to recopy the data from the publishing objects to ensure synchronization.

Dictionary and data portions of a file

If you replicate both the data and dictionary portions of a file, they must belong to the same replication group.

Running the recoverable file system on the publishing server

If you are running the Recoverable File System on the publishing server, the `CNAME` command does not regenerate replication log files for record-level update that occurred prior to executing the `CNAME` command when restarting UniData after a system crash. The updates to a file before executing the `CNAME` command may be lost on the subscribing server.

Multiple locks

U2 Data Replication can apply multiple locks on the subscribing server. If your application is running on the subscribing server and you need to apply the same lock, you should follow the same locking sequence to avoid a deadlock.

Appendix A: Troubleshooting

The following sections describe various troubleshooting topics

UNIX kernel parameters

Because replication uses internal message queues and semaphore structures, you might need to increase some kernel parameters.

Parameter	Description
semmni	Maximum number of semaphore sets system-wide
semmns	Number of semaphores in the system
semmnu	Maximum number of semaphore undo structures
semmsl	Maximum number of semaphores per ID
maxnofiles	Maximum number of files that can be opened system-wide
maxfiles	Maximum number of files that can be opened by a single process

semmni and semmns are the most common parameters to change, but the others may need to be changed in step. The following example shows an error message indicating that the number of semaphores needs to be increased in the rm.log file.

```
"process:14295, _smm_semaget(),udt_semget() error, errno:28.
Exit: smm: cannot allocate semaphore for replication waiting node 8 errno 28
Mon Jun 20 11:24:46 RM: Load configuration to shm failed"
```

Determining which replication group is handling a specific file

In order for an update to a replicated file to be applied to the corresponding file on a subscribing system, that file must be handled in the same replication group on both the publishing system and the subscribing server.

- If you have identical accounts on both servers (including file definitions in all VOC files) and identical `repconfig` files, replication consistently loads files to the same groups.
- If you have different `repconfig` files or additional file pointers in VOC files on one side or the other, you might have missing updates on the subscribing server.

With account-level replication, any files in an account that are not explicitly included in replication (with a `repconfig` FILE phrase) are implicitly included by the presence of a file definition item in the VOC of the named account. During replication initialization, the VOC file is scanned for all file pointers. Each pointer is loaded into the replication buffer as a replication object.

- Entries for replication objects named in FILE phrases are referred to as static loaded.
- Entries for replication objects not named in FILE phrases are referred to as dynamic loaded.

Every file reference in the VOC for an account-level account is loaded into the replication buffer for that group. You may have multiple replication objects created for the same physical file. The file references loaded include:

- Main file references to files residing in the directory.
- Synonym file reference to files residing in the directory.

- File pointers to files residing in other accounts on the server.
  - These files may also be included in other replication group definitions because they are:
    - Specified with FILE phrases in other replication groups.
    - Included implicitly in another account-level group.
- On UNIX systems, file pointers might symbolic links to a physical file on another file system.

If you have multiple replication objects created for the same physical file (unique inode/dnode), only one object will be used to replicate updates from your publishing server to your subscribing server. This replication object is flagged as the PRIMARY object.

How does Replication decide which object to use as the primary replication object? The following precedence table is used when the objects are first loaded:

1. Excluded files
2. Included files
3. Account-level objects
   a. F Pointer

Here is the sequence of checking that is done:

- If the file is excluded in any group it is excluded
- If the file is named only once in a `repconfig` FILE phrase for a file-level group, this is primary.
- If the file is named only once in a `repconfig` FILE phrase for an account-level group, this is primary.
- If the file is not named via a `repconfig` FILE phrase whichever ‘F’ pointer entry is processed first during initialization of the account level groups becomes the primary.

Note: A file may be explicitly defined multiple times in one or more replication groups using different VOC file pointers A file may have no explicit `repconfig` entries, but has multiple VOC pointers – possibly in multiple account-level group accounts. If objects are referenced at the same level, then it’s just down to which reference is processed first during initial loading

---

Primary file/object and replication

- When an update is made to a file on a publishing system, the replication object table in shared memory is searched to find entries for that file's inode/dnode.
- Replication then uses the primary replication object found in the inode/dnode search to construct a replication log to send to the subscribing system. The significant attributes of that object used are:
  - The replication group name where the primary entry is found.
  - The file name reference stored for that replication object.
- When the log is received on the subscribing system, the replication objects table in shared memory for the group number specified is searched for the specific file name reference sent.
- If that file reference is found, the replication writer processing that log opens the file in the group's VOC using that file name.
- If the OPEN is successful, the update is applied and the log is flagged as “done”.
- On the subscribing server, there is no need to check whether the replication object used to apply updates to the file is the primary object or not. It is not used.

Note: This discussion about primary objects relates to record-level updates made to replicated files. When a file-level command is replicated (such as `DELETE.FILE`), the database uses the object in the replication group that is the same name as the file specified in the command – even if it is not the primary object.

To avoid confusion about which file reference is loaded as the primary object, create one – and only one - `repconfig` FILE phrase for the same physical file.

Now that we have a better understanding of what a PRIMARY replication object is and why understanding this could be important, we circle back to the initial question. In a complex application environment, how can an administrator determine what replication group is being used to replicate a specific file?

We provide a TCL command to list replication objects:

```
LIST.REPLICATION.FILE [ALL |DICT |DATA] filename[,subfile] NO.PAGE
```

Usage:

- If you specify ALL, the database displays all active replication files in the current account.
- If the local account is an account-level replication account, this command displays the account as well.
- If you specify a file name, the database displays all replication files contained in the replication file table related to that file.

If you do not specify ALL or filename, UniData obtains the file names from an active select list, or prompts for input.

Data displayed:

- FTYPE – The type of file.
- Valid file types are:
  - LF – LF directory
  - DICT – Dictionary file
  - DATA – Hashed file
  - ACCT – The account-level replication account
- FILE NAME – The VOC file name in the replication file table. If the replication file is not located in the local account, the `LIST.REPLICATION.FILE` command also lists the path to the file.
- REPLICATION – The replication definition for the file.
  - PUBLISHED – The file is published
  - SUBSCRIBED – The file is subscribed
  - EXCLUDED – The file is excluded from replication
  - INACTIVE – The file is included in replication, but is currently disabled
- PRIMARY – Whether the file is considered the primary replication file.
- GROUP – The name of the replication group

If you use the ALL argument, all files for the account you are logged on to are displayed. The files are listed in the sequence they are loaded into the shared memory replication buffer. You may spot problems by scanning this output, but the command is most useful for reviewing a specific file name that you have concerns about.

Here is an example of using the command to review a single file name. When you specify a file as the argument, `LIST.REPLICATION.FILE` opens that VOC pointer and determines the inode/dnode of

the physical file. It then lists all replication objects in shared memory that reference the physical file. In this example, we are looking at the LICENSES file in the inventory account.

```
>LIST.REPLICATION.FILE DATA LICENSES
FTYPE FILENAME REPLICATION PRIMARY GROUP
DATA LICN PUBLISHED YES inventory_acct_g0
DATA LICENSES PUBLISHED NO inventory_acct_g0
DATA LI_CENSES PUBLISHED NO inventory_acct_g0
```

Observations:

- Actual file name on disk: /disk1/udt72/inventory/LICN (though you cannot determine this from the command output).
- The file is not explicitly defined by any `repconfig` FILE phrase.
- Replication objects are loaded by the existence of VOC items in both inventory and payroll accounts (and that there are account-level groups setup for each account).
- The DATA keyword is used so dictionary objects are not displayed.
- Since there were no explicit references to this file, the replication object chosen by Replication happens to be for the VOC pointer LI_CENSES.
  - PRIMARY = YES for this FILENAME.
  - This is one of the synonym file pointers (observe the spelling difference).
  - Since this object was chosen as primary, the file name sent to the subscribing server for updates to the (operating system level) LICN file (via any of the file pointers) is LI_CENSES.
  - As long as the LI_CENSES VOC pointer exists in the inventory account on the subscribing system, updates will be applied to the file it references. If the VOC pointer does not exist on the subscribing server, updates will not be applied to the file on disk.

You may prefer to have the primary replication object for the /disk1/udt72/inventory/LICN file reference your main VOC file reference for this file: LICN. You can designate this VOC reference as the primary by adding one (and only one) FILE phrase to one of the groups you have configured for the inventory account.

Collecting diagnostic information

At the first hint of a problem with U2 Data Replication (on any of your publishing or subscribing servers), it is important to capture diagnostic information, ideally before you take any corrective actions. Replication is a large, complex product, particularly if RFS has been implemented on your systems. The easiest way to capture virtually all of the information you need to diagnose replication problems is to run `$UDTBIN/udtdiag` on all replication systems as close to the same time as possible. The only items missing from the `udtdiag` dump are the `replog` files and directories. As these can be quite large (and are rarely needed for problem analysis), we have chosen to skip dumping the `replog` files in a `udtdiag` dump.

Run the udtdiag script

On UNIX systems, `udtdiag` is a shell script. On Windows systems, it is a `.bat` file that invokes a few executables as well as a number of DOS commands. Whereas there are some optional arguments for `udtdiag` (take a look at the script/`.bat` file), the simplest way to run the script is as follows:

- Log on as a root or administrator user and access the operating system shell level.

- Change directories to a directory that has ample disk space available. Dumps can range from 5MB to 200 or 300MB, depending on what UniData software you are running and the size of your system. If you are running RFS with a large, active system buffer, your dump will be on the large end of the spectrum.
- Decide on a dump directory name. The directory should not already exist in the selected parent directory.
- Run `udtdiag <your.dump.dir.name>`. This assumes UDTBIN is in your path. Names such as “acme.pub.050109” (for a dump on the publishing system for Acme Products Company on May 1, 2009) can be useful to keep things organized.
- Repeat the same process for each system involved in a publishing/subscribing configuration.
- If you are providing the `udtdiag` dumps to U2 Technical Support for review, please tar and compress (UNIX) or zip (Windows) the dump directories (relative path only), and FTP them to our FTP server.
  - ftp ftp.rocketsoftware.com
  - Log on as user upload
  - Enter user password ToR0cket
  - bin
  - put <compressed tar or zipped dump directory>
  - quit
- Please advise U2 Technical Support of the names of the files you have posted and the support case with which they are associated.

Troubleshooting replication

The following sections are designed to help you recognize the kind of problem you are having and determine the steps to take to resolve it.

Problem recognition relies on understanding errors and messages created by the replication processes. Replication writes its messages and errors to the following files in the `$UDTBIN` directory:

- `rm.log`
- `rm.errlog`

When replication has started successfully and is running normally, it also uses the following error files:

- `rw.errlog` – These files contain errors encountered by the individual replication writer process.
- `udpubn.errlog` (where n is the group number starting from 0, as compared to the number of groups in the `repconfig` file) – These files contain details of errors encountered by the individual replication publisher processes on a publishing system.
- `udsubn.errlog` (where n is the group number, starting from 0 as compared to the number of groups in the `repconfig` file) – These files contain details of errors encountered by the individual replication subscriber processes on a subscribing system. The messages written into these error logs are also written into the `rm.log`.

Replication does not start

When the UniData daemons are started on a system with replication enabled, processes are launched to initialize replication buffers and prepare the system to sync to a remote server. This section describes how to observe and resolve problems with the replication daemons launching.

Note: If `repsys` AUTORESUME=1, during the `startud` process, replication will attempt to sync to another server defined in a replication distribution. For more information on syncing, see Replication does not sync, on page 210.

Recognizing the problem

- Starting the UniData daemons fails.
- Unable to launch a udt session in a UniData account.
- Error displayed to terminal where the failed `startud` command was run.
- Expected Replication daemons not started on the system.
- Error messages recorded in `$UDTBIN/ rm.log`, `rm.errlog`, or `rw.errlog`.

Resolving the problem

- Make some notes about recent activities on the system. What changed?
  - UniData or replication configuration changes?
    - Initial replication setup or changes to existing active configuration?
  - System kernel changes?
  - UniData data account layout or account location changes?
- Review `UDTBIN` replication logs and error logs on the problem system.
- If errors are found in the logs, review the specific errors listed in this section for a match.
- Before taking corrective action, generate a `udtdiag` dump that captures the current status.
  - If the error suggests a problem with an associated publishing or subscribing server, create a `udtdiag` dump on the other server as well.
- Make changes recommended in the Error Messages section to resolve the problem.
- Update your notes to include the time/date you made a change and the specific change made.
  - What was changed?
  - What commands did you run? When?
  - What did you observe after making a change?
  - Capture any messages displayed on your terminal.
- Need additional help to resolve your problem?
  - Create a second `udtdiag` dump.
  - If appropriate, create a second dump on an associated publishing or subscribing server.
  - Provide your detailed notes to your UniData support provider.
  - Provide all `udtdiag` dumps (before and after resolution attempts) to yourUniData support provider.

Error messages

Location	Error message
$UDTBIN/startud – text displayed	Couldn't start RM. Please check `/disk1/ud72/bin/rm.errlog` UniData not started.

`rm.log`, `rm.errlog`	`Wed Jun 10 10:53:00 repsys: Local replication system is not defined.`
	`Wed Jun 10 10:53:00 RM: Replication System definitions are inconsistent.`
	`Wed Jun 10 10:53:00 Exit: RM Exit(1)!`

This same set of errors can be returned by two different problems with the `repsys` file.

The most common problem is with how a HOSTNAME entry is specified in `repsys` as compared to the UNIX `/etc/hosts` entries. When setting up replication on a set of systems, you may just need to update `/etc/hosts` to match the specific system reference you use in `repsys`. Compare your `repsys` HOSTNAME entries to `/etc/hosts`.

Here is the logic used by the `repmanger` to check hostnames:

1. If the `repsys` HOSTNAME value is a string (as opposed to an IP address format *n.n.nn.nnn*), it compares the HOSTNAME string to the value returned by the UNIX function gethostname(). If it matches, the system is determined to be local.
2. If the first check fails:
   a. If HOSTNAME is a string, call UNIX gethostbyname().
   b. If HOSTNAME is an IP address, call UNIX gethostbyaddr().
3. If a match is found from checks in step 2, the system is determined to be local – otherwise it is not.

The second problem is that the `repsys` VERSION setting does not match the UniData version that the `uvrepmanger` process is running. This problem could occur after upgrading UniData from 7.1 to 7.2, for instance, and failing to modify VERSION (setting it to 72) in your `repsys` file.

The following table shows error messages displays when operating system semaphore settings are insufficient.

Location	Error messages
`$UDTBIN/startud – text` displayed	Couldn't start RM. Please check `/disk1/ud72/bin/rm.errlog` UniData not started.
`rm.log`	`process:29619, U_smm_semaget(), udt_semget() error, errno:28. Exit: smm: cannot allocate semaphore for replication waiting node 968 errno 28 Tue Jun 9 13:50:43 RM: Load configuration to shm failed Tue Jun 9 13:50:43 RM exit: Setup shm config failed. Tue Jun 9 13:50:43 Exit: RM Exit(1)!`
`rm.errlog`	`Tue Jun 9 13:50:43 RM: Load configuration to shm failed Tue Jun 9 13:50:43 RM exit: Setup shm config failed. Tue Jun 9 13:50:43 Exit: RM Exit(1)!`

For additional information, see UNIX kernel parameters, on page 199. A short-term solution may be to decrease `udtconfig` NUSERS setting, until you can get the kernel adjusted.

Note: In this particular case, the useful error information was contained in the `rm.log` and was missing from the `rm.errlog`.

Error message details:

- process: 29619 – The process ID of the `repmanger` daemon.
- U_smm_semaget(), udt_semget() – database functions for creating semaphores.
- errno 28 – UNIX error 28 ENOSPC from system call failure to create a semaphore.

Location	Error message
`$UDTBIN/startud – text` displayed	Couldn't start RM. Please check `/disk1/ud72/bin/rm.errlog` UniData not started.
`rm.log`, `rm.errlog`	`Tue Jun 9 13:38:33 Exit: RM cannot create request msg Q, errno=28.`

This is an indication that the operating system message queue configuration is insufficient to allow replication to start. To correct the problem, increase the kernel setting for the total number of message queues allowed system-wide (typically msgmni). Alternatively, in the short-term, you could try reducing the `udtconfig` settings on RFS-enabled servers for N_PGQ and N_TMQ.

Error message details:

- errno=28: UNIX error ENOSPC returned by system call to create a message queue.

Location	Error message
`$UDTBIN/startud – text` displayed	Couldn't start RM. Please check `/disk1/ud72/bin/rm.errlog` UniData not started.

`rm.log`, `rm.errlog`	`Starting: Tue Jun 9 14:20:27 2009`
	`Tue Jun 9 14:20:27 RM:`
	`Create shm fails(22)`
	`Tue Jun 9 14:20:27 RM exit:`
	`Setup shm config failed.`
	`Tue Jun 9 14:20:27 Exit: RM Exit(1)!`

This error occurs if the replication manager is unable to create the necessary shared memory segments for replication to start. For more information, see IPC structures, on page 38. This could be the result of the configuration settings requiring a shared memory segment that is larger than the kernel allows (shmmax).

This error can occur if the `udtconfig` parameters MAX_REP_SHMSZ and SHM_MAX_SIZE are set to the same value and the replication buffer configuration settings result in more than one shared memory segment needed to contain the buffers for all replication groups. To resolve this, reduce the value of MAX_REP_SHMSZ so that it is at least 64 MB smaller than SHM_MAX_SIZE.

Error message details:

- Create shm fails(22) – UNIX error 22 EINVAL is returned by shmget() system call.
  - If there are insufficient shared memory identifiers, shmget() returns UNIX error 28 ENOSPC instead.

Location	Error message
`$UDTBIN/startud – text` displayed	Couldn't start RM. Please check `/disk1/ud72/bin/rm.errlog` UniData not started.

`rm.log`, `rm.errlog`	`Tue May 19 12:13:06 Account /disk1/udt72/no_inventory is not accessable(2).`
	`Tue May 19 12:13:06 RM: Load Replication objects fails!`
	`Tue May 19 12:13:06 Exit: RM Exit(1)!`

This set of errors indicates that an ACCOUNT path specified in `repconfig` on a subscribing or publishing server does not exist. This fatal error occurs during the database start operation and as a result, UniData fails to start. This type of error often occurs when you copy a valid `repconfig` file from one system to another and the second server has a different file system layout. After you have corrected the specified ACCOUNT path in `repconfig`, all will be well.

Location	Error message
`$UDTBIN/startud – text` displayed	Couldn't start RM. Please check `/disk1/ud72/bin/rm.errlog` UniData not started.
`rm.log`, `rm.errlog`	`Tue Jun 9 15:23:52 RM: Save replication configuration failed` `Tue Jun 9 15:23:52 Exit: RM Exit(1)!`

This error is logged in `rm.errlog` and `rm.log` during the database start operation. The database fails to start. When replication starts, the replication configuration is saved in REP_LOG_PATH (as defined in `udtconfig`). If replication is unable to save its configuration in the path as defined, it records this `rm.errlog` message. In our example, the REP_LOG_PATH was invalid for the server.

Note: If the REP_LOG_PATH file does not exist, the database tries to create it for you. As long as the server can reach the specified path (and the directory can be created), this error does not occur.

Location	Error message
`$UDTBIN/startud -text` displayed	Couldn't start RM. Please check `/disk1/ud72/bin/rm.errlog` UniData not started.

`rm.log`, `rm.errlog`	`Sun Jun 7 20:48:38 Loading Account /disk1/udt72/payroll failed.`
	`Sun Jun 7 20:48:38 Reference rw.errlog for details.`
	`Sun Jun 7 20:48:38 Invalid account definition /disk1/udt72/payroll in replication group payroll_acct_g2`
	`Sun Jun 7 20:48:38 RM: Replication Configuration are inconsistent.`
	`Sun Jun 7 20:48:38 Exit: RM Exit(1)!`

`rw.log`	`RW Initial Loading(-1,852198) report: No more LCTs`
	`udt_exit(1) called in File shmbuf.c, Line 266`
	`Sun Jun 7 20:48:38 RW Initial Loading (-1,852198):Failed to initialize on Account /disk1/udt72/payroll(14022,0)`

If there are fewer LCTs available than replication accounts, `startud` fails during the phase in which `udrw` processes are launched to load the inodes and dnodes of all replication objects. One `udrw` process is launched for each replicated account. Each `udrw` occupies a slot in the local control table (LCT). If there are not slots available for each of these `udrw` processes, replication cannot be launched. This is a fatal error to starting the data server. To correct this problem, increase `udtconfig` NUSERS and run `startud`.

Note: This is fairly unlikely to occur at UniData 7.2, as only one `udrw` is launched to load replication objects for each account that has files replicated. Prior to version 7.2, one `udrw` process was launched for each replication group.

Location	Error message
`$UDTBIN/startud – text` displayed	Couldn't start RM. Please check `/disk1/ud72/bin/rm.errlog` UniData not started.

`rm.log`, `rm.errlog`	`Tue Jun 9 15:38:07 Loading Account /disk1/udt72/inventory failed.`
	`Tue Jun 9 15:38:07 Reference rw.errlog for details.`
	`Tue Jun 9 15:38:07 Invalid account definition /disk1/udt72/inventory in replication group inventory_acct_g0`
	`Tue Jun 9 15:38:07 RM: Replication Configuration are inconsistent.`
	`Tue Jun 9 15:38:07 Exit: RM Exit(1)!`

`rw.errlog`	`Tue Jun 9 15:38:07 RW Initial Loading(-1,3916) report: Locale en_US.UTF-8 does not belong to current language group 255/192/129 udt_exit(1) called in File shmbuf.c, Line 265`
	`Tue Jun 9 15:38:07 RW Initial Loading (-1,3916):Failed to initialize on Account /disk1/udt72/inventory(14022,0)`
	
	[additional set of errors for each account/ repwriter]

This set of errors occurs during the `startud` operation. UniData fails to start. A message similar to the following is displayed on the terminal:

`Couldn’t start RM. Please check /disk1/ud72/bin/rm.errlog`

This problem is associated with UniData language group settings and default locale/LANG settings for the operating system and root user. It occurs most often on Linux systems.

You need to add an additional line to the `$UDTHOME/sys/LANGGRP` file for the language group specified in the `rw.errlog` message– in this case en_US.UTF-8. For example:

```
255/192/129                 # Language Group 1
     ENGLISH C              # US English
     ENGLISH en_US.UTF-8    # Linux Default # Added line
```

Error message details:

- RW Initial Loading(-1,3916) – The repwriter PID encountering this error = 3916.
- Account /disk1/udt72/inventory(14022,0) – Database Err code (Intercall Developer's Guide appendix)
  - 14022 = IE_EINVAL – Invalid Argument

Note: At UniData 7.2, the default UniData `LANGGRP` file created during installation includes en_US.UTF-8 for Linux implementations. This installation change significantly reduces the occurrence of this replication startup failure.

Replication does not sync

The previous section describes problems associated with initializing core replication daemons and loading replication objects into shared memory buffers. After the `repmanger` and `udrw` processes have completed this process, the next step in bringing up replication is to connect a publishing server to a subscribing server, and then start the remaining replication daemons. This is accomplished by the administrative command `ud_repadmin sync`, or automatically during the database start process if the `repsys` parameter AUTORESUME=1. The `sync` command can also be issued in an EXCEPTION_ACTION script that launches as the result of an unexpected suspension. This section describes how to observe and resolve problems associated with the replication sync process.

Recognizing the problem

- Expected replication daemons not started on the server.
- Expected `rm.log` "resumed running" messages missing for one or more configured groups.
  - Here is an example from a publishing server `$UDTBIN/rm.log`:

    ```
    Sun Jun 7 21:18:51 2009 Group payroll_file_g3 Replication to
    standby synchronized successfully and resumed running.
    ```
- Error messages recorded in $UDTBIN `rm.log` and `rm.errlog`.

Resolving the problem

- Make some notes about recent activities on the system. What changed?
  - UniData or replication configuration changes?
    - Initial replication setup or changes to existing active configuration?
  - System kernel changes?
  - UniData data account layout or account location changes?
  - Sync failure after unexpected suspension?
- Review $UDTBIN replication logs and error logs on the problem system.
- If errors are found in the logs, review the specific errors listed in this section for a match.
- Before taking corrective action, generate a `udtdiag` dump that captures the current status.
  - If the error suggests a problem with an associated publishing or subscribing system, create a `udtdiag` dump on the other server as well.
- Make changes recommended in the following section to resolve the problem.
- Update your notes to include the time/date you made a change and the specific change made.
  - What was changed?
  - What commands did you run? When?
  - What did you observe after making a change?
  - Capture any messages displayed on your terminal.
- Need additional help to resolve your problem?
  - Create a second `udtdiag` dump.
    - If appropriate, create a second dump on an associated publishing or subscribing server.
  - Provide your detailed notes to your UniData support provider.
  - Provide all `udtdiag` dumps (before and after resolution attempts) to your UniData support provider.

Error messages

Location	Error message
`rm.log` on publishing server	Execute replication exception action program `/disk1/udt72/resync_script` [other messages about suspending / syncing]
`rm.log` on subscribing server	Execute replication exception action program `/disk1/udt72/resync_script` [other messages about suspending / syncing]

The `repsys` EXCEPTION_ACTION parameter lets you define a script to run if replication suspends unexpectedly. In your custom script, you can try to automatically resync replication – assuming that the cause for the suspension has been resolved. If you include the `ud_repadmin sync` command in your script, and the script is enabled on both the publishing server and the subscribing server, a `sync` command might be issued on one side while the other side is still suspending. The net result can be that the sync command issued in the scripts fails to allow replication to reestablish a connection and resume running.

As a best practice, if you include a `sync` command in your script, you should define an EXCEPTION_ACTION script on either the publishing or subscribing server, but not on both. We recommend locating the EXCEPTION_ACTION script on the subscribing server.

Location	Error message
`rm.log`, `rm.errlog`	`Tue Jun 9 13:05:39 2009, Group 0 report: udpub(inventory_acct_g0): Establish RPC connection to remote system standby error(81016) Tue Jun 9 13:05:39 Udpub Can not initiate Sync operation. Tue Jun 9 13:05:39 RPC Refused, Remote system may not start(81016)! DBA Ordered Synced GROUP inventory_acct_g0, SYSTEM standby failed.` [additional sets of message for each replication group]

This set of errors occurred because unirpcd was not running on the remote system. As unirpcd is required to initiate the TCP/IP connection between the servers, the `sync` command failed.

Error message details:

- error(81016) – Database error code (Intercall Developer's Guide, appendix): UVRPC_REFUSED.a
  - Connection refused, unirpcd not running.

The following error message is displayed in `rm.log` on subscribing server:

```
Mon Nov 28 07:28:04 2011: Request
Sync to ALL received.
Mon Nov 28 07:28:07 Invalid log
sequence received by subscriber(2254)!
Mon Nov 28 07:28:07 2011 Subscribing
group payroll_file_g3 exit:
Mon Nov 28 07:28:07 Invalid log
sequence received by subscriber(2254)!
Mon Nov 28 07:28:07 2011 Subscribing
```

```
group payroll_acct_g2 exit:
Cross-Group Suspension
Suspended GROUP inventory_file_g1,
SYSTEM primary successfully
Cross-Group Suspension Suspended
GROUP inventory_acct_g0,
SYSTEM primary successfully
Mon Nov 28 07:28:08 Invalid log
sequence received by subscriber(2254)!
DBA Ordered Synced Replication
subscribing from primary failed.
Mon Nov 28 07:28:08 2011 Subscribing
group inventory_acct_g0 stopped
successfully.
Mon Nov 28 07:28:08 2011 Subscribing
group inventory_file_g1 stopped
successfully.
```

The following error message is displayed in the `rm.errlog` on subscribing server:

```
Thu Jun 11 07:28:07 Invalid log
sequence received by subscriber(2254)!
Thu Jun 11 07:28:07 2009
Subscribing group payroll_file_g3
exit: Thu Jun 11 07:28:07
Invalid log sequence
received by subscriber(2254)!
Thu Jun 11 07:28:07 2009
Subscribing group payroll_acct_g2
exit: Thu Jun 11 07:28:08
Invalid log sequence
received by subscriber(2254)!
DBA Ordered Synced Replication
subscribing from primary failed.
```

The following error message is displayed in the `rm.errlog` on publishing server:

```
Thu Jun 11 07:28:07 Invalid log
sequence received by subscriber(2254)!
Remote System Requested Synced GROUP
payroll_file_g3, SYSTEM standby failed.
Thu Jun 11 07:28:07 Invalid log
sequence received by subscriber(2254)!
Remote System Requested Synced GROUP
payroll_acct_g2, SYSTEM standby
failed.
Thu Jun 11 07:28:07 Invalid log
sequence received by subscriber(2254)!
Remote System Requested Synced
Replication publishing to standby
failed.
RPCPID=716800(/disk1/ud72/bin/
publistener) - 07:28:09 -
read returns 0 errno=0
publistener(0): Connection broken by
subscriber(81002)!
Thu Jun 11 07:28:09 RPC
Connection Lost(81002)!
Thu Jun 11 07:28:09 2009
Group inventory_acct_g0 Replication
to standby is suspended.
Cross-Group Suspension Suspended GROUP
```

```
inventory_file_g1, SYSTEM standby
successfully
RPCPID=631026(/disk1/ud72/bin/
publistener) - 07:28:09
- read returns 0 errno=4
publistener(1): Connection broken
by subscriber(81002)!
```

The database keeps track of the current log sequence number (LSN) for each replication group in the replication buffers in shared memory. During a normal data server shutdown, the current LSNs are saved to disk and reloaded when replication restarts. In this case, the subscribing server crashed without allowing the database to preserve the LSNs to disk. The replication sync process checks the incoming logs to see if the LSN numbers are what are expected (based on reloading the status file from disk during replication startup). If there is a mismatch, the `Invalid log sequence` message is recorded in the logs and the sync fails.

To resolve this situation and restart replication, use the force option on the `sync` command:

```
$UDTBIN/ud_repadmin sync -force
```

This option tells the replication system on the subscribing server to reset the LSN numbers in memory to the LSN of the first log received from the publishing server for each group.

Note: For systems with the Recoverable File System enabled, the LSN numbers are automatically flagged to be reset during the RFS recovery process. If you have AUTORESUME=1 set in your `repsys` file, UniData will automatically sync without errors after recovery has completed. If AUTORESUME=0, you can also issue the `ud_repadmin sync` command immediately after recovery.

If AUTORESUME=0 and you gracefully stop and restart UniData after RFS crash recovery and before issuing a `sync` command, you may receive an “invalid log sequence” error. If so, use the -force option to allow the sync process to be successful.

Location	Error message

`rm.errlog` on subscribing server	`Thu Jun 11 08:19:21`
	`RW(1,524290) report:`
	`No more LCTs`
	`udt_exit(1) called in File shmbuf.c,`
	`Line 266`
	`Thu Jun 11 08:19:21 RW (1,524290):`
	`Failed to initialize on Account`
	`(14022,0)`

`rm.errlog` on subscribing server	`Thu Jun 11 08:26:12 2009 Subscribing`
	`Group inventory_acct_g0 started`
	`suspending.`
	`Thu Jun 11 08:26:12 Subscriber has`
	`no alive RWs(2358)!`

This error occurs during the sync operation, when the `repwriter` processes on a subscribing server are being spawned. This indicates that the value of `udtconfig` parameter NUSERS is too small to allow all of the configured `repwriter` processes to start. One error log is produced for each failure.

Note: If NUSERS is sufficient for at least some of the `repwriter` processes to start for all groups configured, there are no errors recorded in `rm.log` or `rm.errlog`. The `showud` command displays `udrw` processes, but not all `udrw` processes are running. This is not fatal to the sync process, so this problem may not be immediately apparent. Replication continues running on the subscribing server, but it may be under-performing because one or more groups have fewer than expected `repwriter` processes applying updates to the database files.

Location	Error message
`rm.log`, `rm.errlog` on publishing server	`Tue Jun 9 12:46:03 Publisher system is not defined or mis-matched on HOSTNAME/VERSION at subscriber(2204)! DBA Ordered Synced GROUP inventory_acct_g0, SYSTEM standby failed.` [additional messages for each replication group]

This set of errors occurred on a publishing server as the result of running a `ud_repadmin sync` command. The problem lies in the `repsys` files on the two servers involved in replication. In this case, the HOSTNAME defined for the publishing server in the `repsys` file on the subscribing server did not resolve to the same computer system defined in the `repsys` file on the publishing server. The mismatched definitions:

- Publishing server `repsys`: HOSTNAME=tshp11i
- Subscribing server `repsys`: HOSTNAME=tsaixpub

The simplest solution is to copy the correct `repsys` file to the system with the problematic `repsys` file. You may need to adjust ACCOUNT paths if the disk layout is not the same on each system.

This specific error message is just one of a series of messages that provide more guidance than the generic “configuration mismatch between publisher and subscriber” message. Here are the additional error messages you could see in a publishing system `rm.log` and `rm.errlog` after a `sync` command is issued:

Command	Description
SUB_INVALID_ACCT (2207)	Subscriber account is invalid.
SUB_NO_GROUP (2205)	No subscribing group defined for the replication. Group name is not defined at subscriber `repconfig`.
SUB_FAILEDOVER (2202)	The configured subscriber server has failed over to failed-over publisher. Publisher has not failed over, as the subscriber has.
SUB_NOT_FAILEDOVER (2201)	Replication has not failed over to failover subscriber at configured publisher. Publisher has failed over, but subscriber has not.
SUB_IS_PUB (2203)	Group is configured as a publishing group at subscriber.
SUB_REMOTE_SYS_NOT_FOUND (2204)	Publisher server is not defined or is mismatched on HOSTNAME/VERSION at subscriber.
INVALID_PUB (2208)	Publishing server is not defined as publisher at subscriber server.
SUBSYS_MISMATCH (2209)	Subscribing server definition mismatch on SYSNAME, HOSTNAME, or VERSION.
REPTYPE_MISMATCH (2206)	Replication type defined differently at subscriber.

Replication unexpectedly suspends

Replication can be suspended intentionally by an administrator using XAdmin or via a `ud_repadmin suspend` command at the shell. Suspension breaks the connection between the publishing and subscribing servers. The replication daemons launched by a `sync` command exit, as well. Replication logs generated on a publishing server are written to the log reserve file (LRF) instead of being sent to a subscribing server.

This section describes an unintended suspension that is triggered by a problem encountered in the replication environment – whether on the publishing server, the subscribing server, or the network connection between.

Recognizing the problem

- The `repsys` EXCEPTION_ACTION script has launched.
  - Depending on how this script is coded, you may have been emailed or paged.
- Expected replication daemons have exited.
- Error messages are recorded in $UDTBIN `rm.log`, `rm.errlog`, `udpubn.errlog`, and `udsubn.errlog`.

Resolving the problem

- Make some notes about recent activities on the system. What changed?
  - UniData or replication configuration changes?
    - Initial replication setup or changes to existing active configuration?
  - System kernel changes?
  - c data account layout or account location changes?
- Review operating system logs for indications of network problems.
- Review $UDTBIN replication logs and error logs on the problem system.
- If errors are found in the logs, review the specific errors listed in this section for a match.
- Before taking corrective action, generate a `udtdiag` dump that captures the current status.
  - If the error suggests a problem with an associated publishing or subscribing server, create a `udtdiag` dump on the other server as well.
- Make changes recommended in the “Error messages” section to resolve the problem.
- Update your notes to include the time/date you made a change and the specific change made.
  - What was changed?
  - What commands did you run? When?
  - What did you observe after making a change?
  - Capture any messages displayed on your terminal.
- Need additional help to resolve your problem?
  - Create a second `udtdiag` dump.
    - If appropriate, create a second dump on the publishing or subscribing server.
  - Provide your detailed notes to your UniData support provider.
  - Provide all `udtdiag` dumps (before and after resolution attempts) to your UniData support provider.

Error messages

The following error message displays in the `rm.log` on subscribing server:

```
Thu Jun 11 15:14:00 2009, Group 2
report:
U_Log_infobuf_append(): pwrite info
file fails. infooff=1020880, errno=0.

Thu Jun 11 15:14:00 2009 Subscribing
Group payroll_acct_g2 started suspending.
Thu Jun 11 15:14:00 Subscriber
failed on sub packet log file(2359)!
Cross-Group Suspension Suspended
GROUP payroll_file_g3, SYSTEM
primary successfully
Cross-Group Suspension Suspended
GROUP inventory_file_g1, SYSTEM
primary successfully
Cross-Group Suspension Suspended
GROUP inventory_acct_g0, SYSTEM
primary successfully

Thu Jun 11 15:14:01 2009, Group
2 report:
U_Log_flush_bodybuf(): pwrite log
body file fails. fno=1, bodyoff=0,
len=131072, errno=28.

Thu Jun 11 15:14:01 2009,
Group 2 report:
U_Log_flush_infobuf(): pwrite info
file fails. infooff=4096, errno=28.
```

The following error message displays in the `rm.errlog` on subscribing server:

```
Thu Jun 11 15:14:00 2009,
Group 2 report:
U_Log_infobuf_append(): pwrite info
file fails. infooff=1020880, errno=0.

Thu Jun 11 15:14:00 2009 Subscribing
Group payroll_acct_g2 started
suspending.
Thu Jun 11 15:14:00 Subscriber
failed on sub packet log file(2359)!

Thu Jun 11 15:14:01 2009,
Group 2 report:
U_Log_flush_bodybuf(): pwrite log
body file fails. fno=1, bodyoff=0,
len=131072,errno=28.

Thu Jun 11 15:14:01 2009,
Group 2 report:
U_Log_flush_infobuf(): pwrite info
file fails. infooff=4096, errno=28.
```

The following error message displays in the `udsub2.errlog` on subscribing server. This error is similar to the error displayed in the `udsub3.errlog`.

```
Mon Nov 28 15:14:00 2011
```

```
U_Log_infobuf_append(): pwrite info
file fails. infooff=1020880, errno=0.

Mon Nov 28 15:14:01 2011
U_Log_flush_bodybuf(): pwrite log
body file fails. fno=1, bodyoff=0,
len=131072, errno=28.

Mon Nov 28 15:14:01 2011
U_Log_flush_infobuf(): pwrite
info file fails. infooff=4096,
errno=28.
```

This error was encountered on a subscribing server when a subscriber process tried to update a replication log on disk and failed. In this case, the REP_LOG_PATH file system was full. Replication for the group that experienced this problem suspended and triggered suspension for all groups. This problem occurred during a sync operation after an extended suspension period. A large number of replication logs from the publishing server's log reserve files were being sent to the subscribing server by the `pubsyncer` processes.

Error message details:

- Subscriber failed on sub packet log file(2359) – LSN 2359 for group 2.
- pwrite – C runtime function to write to a file system file.
- info file – Log file containing log header information.
- body file – Log file containing log body information.
- infooff – Offset into info file for pwrite operation.
- bodyoff – Offset into body file for pwrite operation.
- len – Length of block to write at offset.
- errno=28 = UNIX error – ENOSPC – No space left on device

An error message similar to the following displays in the `rm.log` on publishing server:

```
RPCPID=893068(/disk1/ud72/bin/
publistener) - 07:42:57
- read returns 0 errno=0
publistener(2): Connection broken by
subscriber(81002)!
Fri Jun 12 07:42:58 RPC Connection
Lost(81002)!
Fri Jun 12 07:42:58 2009
Group payroll_acct_g2 Replication to
standby is suspended.
Cross-Group Suspension Suspended
GROUP inventory_acct_g0, SYSTEM
standby successfully
Cross-Group Suspension Suspended
GROUP inventory_file_g1, SYSTEM
standby successfully
Cross-Group Suspension Suspended
GROUP payroll_file_g3, SYSTEM
standby successfully
```

The following error message is displayed in the `rm.errlog` on publishing server:

```
Fri Jun 12 07:42:58 RPC Connection
Lost(81002)!
Fri Jun 12 07:42:58 2009
```

```
Group payroll_acct_g2 Replication to
standby is suspended.
```

The following error message display in the `rm.log` on subscribing server:

```
Fri Jun 12 07:42:58 2009 Subscribing
Group payroll_file_g3 started
suspending(Publisher Requested).
Fri Jun 12 07:42:59 2009 Subscribing
Group inventory_acct_g0 started
suspending(Publisher Requested).
Fri Jun 12 07:42:59 2009 Subscribing
Group inventory_file_g1 started
suspending(Publisher Requested).
Fri Jun 12 07:43:07 2009 Subscribing
group inventory_acct_g0 stopped
successfully.
Fri Jun 12 07:43:07 2009 Subscribing
group payroll_file_g3 stopped
successfully.
Fri Jun 12 07:43:07 2009 Subscribing
group inventory_file_g1 stopped
successfully.
```

This set of errors was triggered by losing the TCP/IP socket connection between a subscribing server `udsub` process and its publishing server `publistener` process. This is typically caused by a network problem between the two servers, but also occurs if one of the processes was killed. In this case, a `udsub` process was killed on the subscribing server. When one group fails, all groups are suspended. In order to restart replication, you need to initiate a `sync` command – either via XAdmin or with the shell-level `ud_repadmin sync` command. You should also review the stability and performance of the network connection between the servers.

Some messages to highlight:

- The publishing server reports “Connection broken by subscriber” as the problem originated by a `udsub` process terminating on the subscribing server.
- All groups are suspended with associated “Cross-Group Suspension” messages.
- If you are reviewing the `rm.log` on the subscribing server, the “Publisher Requested” message is an indication that you should look to the publishing server logs for more details.

Error message details:

- RPCID=893068 – Process ID of the `publistener` process on the publisher end of the broken connection.
- read returns 0 – Socket read returns 0 bytes.
- uvpublistener(2) – `publistener` process spawned for replication group 2.

- RPC Connection Lost(81002) – Database error code (Intercall Developer's Guide appendix)
  - 81002 – UVRPC_NO_CONNECTION. Connection is down.

Location	Error message
`rm.log` on publishing server	`RPCPID=26031(/disk1/ud72/bin/ publistener) - 03:18:32 - failed due to close or abortive close when SO_KEEPALIVE is set, errno=232 publistener(1): Connection broken by subscriber(81002)! RPCPID=26941() - 03:18:32 - low level write bombed errno=32`
`rmlog`, `rm.errlog` on publishing server	`Sat Sep 2 03:18:32 2006, Group 1 report: udpub(1): Send RPC SUSPEND packet error(81002) Sat Sep 2 03:18:32 RPC Connection Lost(81002)! ) Sat Sep 2 03:18:32 2006 Group repgroup1 Replication to standby is suspended. poll: Interrupted system call`
`udpub`*n*`.errlog` on publishing server	`Sat Sep 2 03:18:32 2006 udpub(1): Send RPC SUSPEND packet error(81002)`

This set of errors indicates that there are network and UniRPC connectivity issues. The “low level write bombed” error indicates that the socket connection established by UniRPC was closed on the subscribing server.

It is possible that the subscriber process is unable to keep up with the number of TCP/IP packets that have arrived, and the operating system has closed the socket connection. Generally, however, you

should review the stability and bandwidth of the network connection between the publishing and subscribing servers.

Location	Error message

`rm.log` on the subscribing server	`Fri Jun 12 15:46:24 2009 Subscribing Group repsupport_acc started suspending.`
	`Fri Jun 12 15:46:24 Subscriber has no alive RWs(2358)!`
	`Cross-Group Suspension Suspended GROUP payroll_acct_g2, SYSTEM primary successfully`
	`Cross-Group Suspension Suspended GROUP inventory_acct_g0, SYSTEM primary successfully`
	`Cross-Group Suspension Suspended GROUP inventory_file_g1, SYSTEM primary successfully`
	`Fri Jun 12 15:46:25 2009 Subscribing group repsupport_acc stopped successfully.`
	`Fri Jun 12 15:46:27 2009 Subscribing group inventory_file_g1 stopped successfully.`
	`Fri Jun 12 15:46:27 2009 Subscribing group inventory_acct_g0 stopped successfully.`
	`Fri Jun 12 15:46:27 2009 Subscribing group payroll_acct_g2 stopped successfully.`
	`Cross-Group Suspension Suspended GROUP payroll_file_g3, SYSTEM primary successfully`
	`Fri Jun 12 15:46:28 2009 Subscribing group payroll_file_g3 stopped successfully.`

`rm.errlog` on the subscribing server	`Fri Jun 12 15:46:24 2009 Subscribing Group repsupport_acc started suspending.` `Fri Jun 12 15:46:24 Subscriber has no alive Rws(2358)!`
`udsub`*`n`*`.errlog` on the subscribing server - in this case `udsub4.errlog`	`RPCPID=524428(/disk1/ud72/bin/udsub) - 15:46:24 - low level read select error errno=2`

This was a difficult problem to sort out as there isn't anything recorded in the logs that provides useful clues about the root cause. We eventually discovered the following:

- The customer had a virtual field index on a file that was a UniBasic subroutine.
- They had configured RW_REEVALUATE_VF=1 for the group in which the primary file resided.
- When a `repwriter` process tried to reevaluate the virtual field, a file open statement placed in the UniBasic program to evaluate the virtual field failed with a run-time error.

The easiest solution was to set RW_REEVALUATE_VF=0. The application actually had no need to reevaluate the virtual field index on the subscribing server. Alternatively, the UniBasic subroutine could have been recoded. As written, it relied on NAMED COMMON being initialized in the application

account LOGIN paragraph. As the `repwriter` processes do not run LOGIN paragraphs in an account, this initialization was not happening for the `repwriters`.

Note: The `Subscriber has no alive RWs` error means that all of the `repwriter` processes for a group have exited. In this case it was due to a fatal UniBasic run-time error. In addition to index virtual field evaluation, you may want to see if there are any problems with UniBasic code fired by a trigger.

Note: At UniData 7.2.3 this specific problem is handled differently:

- Error messages similar to the following are added to the `rw.errlog` for each attempted update - to assist with your problem diagnosis:
  - `Wed Jan 20 21:44:19 cwd=/disk1/replication/udt72/inventory/ repsupport 1:evaluate VF error in U_append_strtuple for file 'VF.TEST.FILE.MAIN', key '1', number=71`
  - `Wed Jan 20 21:44:19`
  - `RW(4,413748) report:`
  - `In BP/_VF.BASIC.TEST at line 3 Can not access unopened file. File variable not used in file operation`
  - `In BP/_VF.BASIC.TEST at line 3 Fatal error: READV error`
  - `1:evaluate VF error in U_append_strtuple for file 'VF.TEST.FILE.MAIN', key '1', number=71`
  - `Update VF.TEST.FILE.MAIN,'1' failed(30102,0)`
- Abort messages similar to the following are written to `rm.log` and `rm.errlog` for each attempted update:
  - `Wed Jan 20 14:44:19 2010, RW report:`
  - `RW(4, 413748):Succeeded(30102, 0); (LSN=31)Insert on VF.TEST.FILE.MAIN('1') is abort.`
- Replication does not suspend with this error.

Updates are not made on the subscribing server

When you set up replication and configure files to be replicated, you expect all updates to the file on the publishing server to be applied to the appropriate file on the subscribing server. This section describes the types of errors that can result in failure to apply updates to a subscribing server file.

Recognizing the problem

- Results from user-written application file auditing programs or manual checking of file contents.
- Error messages are recorded in $UDTBIN `rm.log` and `rm.errlog`.

Resolving the problem

- Make some notes about recent activities on the system. What changed?
  - UniData or replication configuration changes?
    - Initial replication setup or changes to existing active configuration?

    - System kernel changes?
    - UniData data account layout or account location changes?
- Review $UDTBIN replication logs and error logs on the problem system.
- If errors are found in the logs, review the specific errors listed in this section for a match.
- Before taking corrective action, generate a `udtdiag` dump that captures the current status.
    - If the error suggests a problem with an associated publishing or subscribing system, create a `udtdiag` dump on the other server as well.
- Make changes recommended in the “Error messages” section to resolve the problem.
- Update your notes to include the time/date you made a change and the specific change made.
    - What was changed?
    - What commands did you run? When?
    - What did you observe after making a change?
    - Capture any messages displayed on your terminal.
- Need additional help to resolve your problem?
    - Create a second `udtdiag` dump.
        - If appropriate, create a second dump on the publishing or subscribing server.
    - Provide your detailed notes to your UniData support provider.
    - Provide all `udtdiag` dumps (before and after resolution attempts) to your UniData support provider.

Error message

Location	Error message
`rm.log`, `rm.errlog`	`Mon Nov 28 14:19:56 Unable to load object NOT.IN.VOC in the group inventory_file_g1`

This indicates that the FILE phrase reference in the `repconfig` file for NOT.IN.VOC does not exist in the VOC file for the account path specified for that replication group.

This error occurs when replication is starting, during the phase in which inodes and dnodes for all replication files are loaded into a shared memory table. This file reference (replication object) is skipped. Any writes to this file fail at the UniData level because the file does not exist.

To avoid this error, you must either create the missing VOC entry for the file (if the file actually exists at the operating system level) or create the file in the appropriate UniData account. If this is not a valid file, remove the FILE=NOT.IN.VOC phrase from the `repconfig` file.

Location	Error message

`rw.errlog`	`Thu May 21 13:21:08`
	`RW Initial Loading(-1,647316) report:`
	`Failed on file /disk1/udt72/`
	`inventory:FILE.MISSING(2),`
	`realname=FILE.MISSING`
	`Failed on file /disk1/udt72/`
	`inventory:FILE.MISSING(2),`
	`realname=D_FILE.MISSING`

This set of errors occurs when a FILE reference in the `repconfig` file is found in the account VOC file, but the file definition item in the VOC itself is invalid. There is no physical file on disk at the location defined in the VOC pointer. This error occurs when replication is starting, during the phase in which inodes and dnodes for all replication files are loaded into a shared memory table. This file reference (replication object) is skipped, as it cannot be written to. You need to either remove this reference from the `repconfig` file or correct the file pointer in the VOC to resolve to a real file on disk.

This error occurs if the file is explicitly defined in `repconfig` with the FILE phrase:

```
FILE=FILE.MISSING
```

This error also occurs if the file is implicitly referenced by virtue of an ACCOUNT level group definition. The replication manager attempts to load all file references in the VOC for the ACCOUNT level group. When it discovers the invalid file pointer in the VOC of that account, it records the errors.

Error message details:

- RW Initial Loading(-1,647316) – The `repwriter` pid encountering this error = 647316.
- FILE.MISSING(2) – The UNIX error number encountered = 2 (ENOENT “No such file or directory”).

Location	Error message
`rm.log`, `rm.errlog` on the subscribing server	`Mon Nov 28 14:41:27 2011, RW report: RW(1, 340140):RW File is subscribed in wrong group(0, 0); (LSN=29624)Insert on SALVAGE ('TESTREC') is canceled.`

File SALVAGE is defined explicitly in a file-level group on the publisher, and defined implicitly in an account-level group on the subscriber. The accounts are the same, but the groups are different. This could be any combination.

In order for an update to be replicated, the replicated file must be defined in the same group number (either explicitly with the FILE phrase in `repconfig`, or implicitly as part of an account-level group). As a best practice, keep the `repconfig` file on your subscribing system identical to the `repconfig` file on your publishing server. If your file system layout is different from your publishing server, typically the only difference may be the ACCOUNT path definition.

Error message details:

- RW(1, 340140) – Repwriter (`udrw`) pid = 340140; replication group = 1.
- LSN=29624 – ID of the specific log sequence number that was canceled.
- SALVAGE('TESTREC')– Record id TESTREC in file SALVAGE.

Note: When an update to an object has been canceled, the log associated with this update is deleted. It is not preserved for later application to the subscribed file. The file on the subscriber will be missing updates until a data refresh can be scheduled.

There are two classes of errors that can trigger a canceled or abort message:

1. File open error
   a. In this case, only one cancelled message is recorded in the `rm.log` and `rm.errlog`.
   b. The file reference in the replication buffer is marked as unavailable.
   c. Any additional logs sent for this file are immediately discarded without any attempt to open the file to apply the update.
   d. You cannot tell how many updates have been lost by reviewing the `rm.log` and `rm.errlog` files.

e. Once you have corrected the underlying problem with the file, you need to suspend and sync replication in order to reload this file in the replication buffer.

2. Any other error
   a. If the error in applying the log is for some reason other than an inability to open the file (for instance insufficient write permissions), each log sent will generate a “canceled” or “abort” message.
   b. If you are able to correct the problem, subsequent logs can be processed and updates applied to the file without having to suspend/sync replication.
   c. The `rw.errlog` file will show each aborted update – including the file name, record key and type of operation (such as “insert” or “delete”).

Location	Error message
`rm.log`, `rm.errlog` on the subscribing server	`Mon Nov 28 07:40:35 2011, RW report: RW(0, 831508):RW file not found in VOC(14013, 13); (LSN=31152) Insert on STOREROOMS ('36229') is canceled.`

Even though the error message text says “file not found,” this error was generated when the `uvrw` process did not have permission to write to the STOREROOMS file. The numbers after “VOC” are the number of the error message documented in the *Intercall Developer's Guide* appendix and the UNIX error code, which together indicate “permission denied”. In this example, the RW_GID and RW_UID parameters that were defined for the replication group were more restrictive than the ownership and permissions of the STOREROOMS file at the operating system level.

Error message details:

- RW(0, 831508) – Repwriter (uvrw) pid = 831508; replication group = 0.
- VOC(14013, 13) – Database error code (*Intercall Developer's Guide* appendix); UNIX error code
  - 14013 – IE_EACCES. Permission denied.
  - 13 – EACCES. Permission denied.
- LSN=31152 – ID of the specific log sequence number that was canceled.
- STOREROOMS('36229') – Record ID 36229 in file STOREROOMS.

Note: When an update to an object has been canceled, the log associated with this update is deleted. It is not preserved for later application to the subscribed file. The file on the subscriber will be missing updates until a data refresh can be scheduled.

Location	Error message
`rm.log`, `rm.errlog` on the subscribing server	`Mon Nov 28 10:33:50 2011, RW report: RW(0, 917532):RW File is not subscribed(0, 0); (LSN=2002) Insert on CLIENTS('17534') is canceled.`

This error was triggered by running “DELETE.FILE CLIENTS” in a replicated account on the subscribing server. The `DELETE.FILE` command removed the file from group 0's replication object table in shared memory. When a `repwriter` process received a log for that file, it did not find it in the table. This resulted in the “not subscribed” message.

As there were multiple updates sent by the publishing server and four `repwriter` processes defined for the group, there were three additional messages in `rm.log`/`rm.errlog` that were very similar

to this. Each repwriter logged a single error and then stopped recording errors for additional logs received.

Error message details:

- RW(0, 917532) – Repwriter (udrw) pid = 917532; replication group = 0.
- LSN=2002 ID of the specific log sequence number that was canceled.
- CLIENTS('17534') – Record ID 17534 in file CLIENTS.

Note: When an update to an object has been canceled, the log associated with this update is deleted. It is not preserved for later application to the subscribed file. The file on the subscriber will be missing updates until a data refresh can be scheduled.

Location	Error message
rm.log, rm.errlog on the subscribing server	Mon Nov 28 11:11:12 2011, RW report: RW(0, 397478):RW file not found in VOC(0, 0); (LSN=3373) Insert on DELIVERY('36864') is canceled.

This error was generated by removing the VOC reference for the DELIVERY file (with “DELETE VOC DELIVERY”) in a replicated account on the subscribing system while replication was suspended. Replication was then restarted with a ud_repadmin sync command and updates sent from the publisher for this file.

The file was initially loaded as a valid replication object when replication started. Given that new repwriter processes are launched by the sync operation, they need to perform a standard file OPEN process when receiving a log for this file to process. Since the VOC pointer did not exist for the file, this error was generated. As multiple updates for this file were received, each repwriter for this group recorded a single error in the logs, resulting in a total of four errors of this nature.

Error message details:

- RW(0, 397378) – Repwriter (udrw) pid = 397378; replication group = 0.
- LSN=3373 – ID of the specific log sequence number that was canceled.
- DELIVERY('36864') – Record ID 36864 in file DELIVERY.

Note: When an update to an object has been canceled, the log associated with this update is deleted. It is not preserved for later application to the subscribed file. The file on the subscriber will be missing updates until a data refresh can be scheduled.

Indexes are corrupted

If your repconfig setting for RW_REEVALUATE_VF is “0”, updates to virtual field indexes for that group on a subscribing server are applied using the values sent from the publishing server. In this case, when replication applies the replication logs to a subscriber file, it expects any virtual field indexes on a file to be in the same slot or position as on the publishing server. It does not check to see that the virtual index expressions match on both servers. If the expressions do not match or the index fields are in a different order, the index updates are still applied and will result in logically corrupt indexes. No error message is produced. This is a silent failure.

Make sure that the indexes appear in the same order on the publishing and subscribing server. Review the output of the TCL LIST.INDEX filename command on each system to confirm that all index

names (both virtual and literal) appear in the same order. The alternate key length specified when the index is created should also be the same on both servers.

If the indexed fields are not listed in the same order, perform the following steps on the subscribing server to correct the problem:

1. Issue the `uv_repadmin suspend` command to suspend replication.
2. Issue the `DELETE.INDEX filename ALL` command to remove the indexes.
3. Issue the `CREATE.INDEX filename index_name` command(s) to create the indexes on the subscriber in the same order as they exist on the publisher and with the same key length.
4. Issue the `BUILD.INDEX filename ALL` command to build the index on the subscriber.
5. Issue the `uv_repadmin sync` command to synchronize replication.

Updates to newly created files are missing

This problem occurs only in versions 7.2.0 and 7.2.1. Here is the problem description for ecase 11157 from the 7.2.2 readme file:

When replicating a `CREATE.FILE` command, initial updates to that file may not have been made on the subscribing server. This problem occurred because the replication writer process first tried to open the file that was newly created on the publishing server. Since the file did not yet exist on the subscribing server, it generated a dummy structure that was not properly cleaned up, causing the record-level operation to fail. This problem has been fixed.

Miscellaneous problems

Error messages

Location	Error message
`rm.log`, `rm.errlog` on publishing or subscribing server	`Fri Aug 4 06:32:08 2006, File operation on replication object: Op=CREATEINDEX, Acct=/disk1/udt/ inventory, File=CLIENTS CREATE.INDEX CLIENTS Fri Aug 4 06:32:18 2006, File operation on replication object: Op=BUILDINDEX, Acct=/disk1/udt/ inventory, File=CLIENTS BUILD.INDEX CLIENTS Fri Aug 4 07:00:09 2006, File operation on replication object: Op=DELETEINDEX, Acct=/disk1/udt/ inventory, File=CLIENTS DELETE.INDEX CLIENTS`

Even with the account-level replication features added at UniData 7.2, not all file-level operations are replicated. UniData alternate index commands are not replicated. If you run commands such as `CREATE.INDEX`, `BUILD.INDEX`, or `DELETE.INDEX` on a replicated file, this activity is noted in the `rm.log` and `rm.errlog` files. These are informational messages, not errors.

Note: These commands are not blocked on an active subscribing system and can be run on the subscriber without interrupting replication.

Determining if a file is being published or subscribed

To determine if a file is currently being published or subscribed from within UniData, you can use FILEINFO from within Basic.

FILEINFO(filevariable, 24) returns the following values:

- 0 Indicates that the file is not included in any replication
- 1 Indicates that the file is a publication object
- 2 Indicates that the file is a subscription object
- 3 Indicates that the file is a writable subscription object

```
OPEN "CLIENTS" TO F.CLIENTS ELSE
CRT "Cannot Open CLIENTS file"
STOP
END
REPSTATUS = FILEINFO(F.CLIENTS, 24)
BEGIN CASE
CASE REPSTATUS = 0
CRT "File is not part of any replication"
CASE REPSTATUS = 1
CRT "File is a publication object"
CASE REPSTATUS = 2
CRT "File is a subscription object"
CASE REPSTATUS = 3
CRT "File is a writeable subscription object"
CASE 1
CRT "Unknown Status"
END CASE
```

Connecting client connections after failover

To make replication failover a seamless process for users, the administrator must consider the failover procedures that are deployed and the way in which the client connections are configured. This may be via standard client connections such as Dynamic Connect, wIntegrate, Telnet, Host Access, or other connections such as Web servers, UniObjects, ODBC, or, in fact, any TCP/IP-based connection.

The following example demonstrates a method that is as seamless as possible, and provides the administrators of the systems with minimal steps to accomplish the task.

- The primary machine has a fixed IP address of 192.1.1.1. We name this machine production-primary and map its IP address to 192.1.1.1 on the site's network infrastructure (such as DNS lookup servers).
- The standby machine has a fixed IP address of 192.1.1.2. We name it production-standby and map its IP address to 192.1.1.2 on the site's network infrastructure (such as DNS lookup servers).
- A machine name of "production" is created, which is assigned an IP address of 192.1.1.1 or 192.1.1.2 via the site's network infrastructure, depending upon the replication state. When working normally, it's assigned to 192.1.1.1 and when the systems are failed over, it's assigned to 192.1.1.2.
- All client-based TCP/IP connections should be set up to connect to the name of "production" and not an IP address or other machine name.

Having followed these steps, when a system is failed over or failed back, the administrators need to follow their process to ensure that correct mapping of the name "production" is applied to the

corporate network infrastructure (usually one or more DNS servers). Once that task is completed, the users of the system will be able to connect to the correct machine.

This also allows administrators, or other users if required, direct connection to either machine via the production-primary or production-standby names, providing complete control and access.

Tuning replication for performance on the publishing system

Balance processing load across multiple replication groups

The `udpub` process reads replication logs from the replication buffer and sends the logs over the network to the subscribing server. A single `udpub` process handles this work for each replication group.

1. Create multiple replication groups to distribute this load.
2. Monitor the number of updates processed for each group and rebalance groups to distribute the load evenly.
3. Use `reptool` or the replication diagnosis utility built into XAdmin to review the next AvailLSN number for each of your replication groups.
4. Move files to different replication groups or create new groups to balance the load.

Note: U2 Technical Support performed some testing to compare the performance gains from increasing the number of replication groups versus increasing the number of replication writer processes in each group. Increasing the number of groups significantly reduced the time taken for updates to be applied to the files on the subscribing server. This was much more effective than increasing the number of replication writer processes.

Monitoring data for updates made to each file in a replication group

As noted above, replication performance can be improved by rebalancing the transaction load processed by each of your replication groups. The XAdmin replication monitor provides excellent information about the number of logs, the number of TCP/IP packets, and the total bytes of data replicated through each replication group. Balancing the load is done by moving files to other replication groups. The question is, Which files are contributing the most load to the processing in each replication group?

Starting with UniData versions 7.1.22 and 7.2.2, the UniData `reptool` utility was enhanced to capture and report activity within a replication group at the individual file level. At a high level, the steps involved are:

1. Use the `reptool` utility to start file logging for one or more replication groups.
2. Run the application programs you want to monitor.
3. Use `reptool` to stop the logging for the group(s).
4. Use `reptool` to report the file activity for each group you had been logging.

The output report is sorted by the total log length captured for each file updated in the replication group during the reporting period. The files with the most data processed are shown first. The

details of the last log processed for each file are also shown. Here is an example of the type of output `reptool` produces:

```
Group inventory_acct_g0(0) update Report: start at Wed Jan 14 14:22:40 2009,
duration=0 hours,40 minutes,57 seconds:
Total 41 Objects updated in the counting period:
RFS Object[30]= SALVAGE (28776, 922337219576856)
Num of updates=700
Total log length=35290
Average log length=50
Total VF length=0
Last Update at Wed Jan 14 15:02:05 2009:
LSN=71581, op=Insert/Update, key=30153, udtno=10
Show next object in the report?
0: Back to previous menu
1: Show next Object
```

Steps for using `reptool` to gather this detailed information is fully described in a Rocket Knowledge Base Technote. To access this document:

1. Sign in to the Rocket Customer Portal:

   http://www.rocketsoftware.com/support
2. Select **Solutions** from the menu, then **Solution Search**.
3. Search for this title: *How to monitor updates to individual files within a U2 Data Replication group*.
4. Select **Exact Phrase** as the **Search Option**.

Minimize cross-group transactions if TP semantics used in your code

If you use transaction processing (TP) semantics in your application code, it is best to minimize the number of cross-group transactions that are generated. A cross-group transaction is where updates within a TP are applied to files in two or more replication groups. Try to group files contained in typical transactions in a single replication group. If you are unable to configure so that all files in a transaction are in a single group, reducing the number of groups involved in a transaction (for instance from three to two) will still improve your performance.

Tuning replication for performance on the subscribing server

Configure an adequate number of repwriter processes

The repwriter (`udrw`) processes read replication logs from the replication buffer on a subscribing server and apply those updates to the database files. To distribute this disk I/O activity, you should configure multiple repwriter processes per group. This is the repconfig parameter N_REPWRITER. It can be different for each replication group, if desired.

- The general recommendation is to have more than one repwriter, but not significantly more than the number of CPUs on the subscribing server.
- You can get a sense if the repwriters are lagging behind for a group by looking at the buffer pointers with `reptool` or the diagnosis utility in XAdmin. If localDoneLSN is consistently significantly behind nextAvailLSN, consider adding more repwriters for that group.

Configure TCA_SIZE for cross-group transaction processing

If you are using transaction processing (TP) semantics in your application code, you may want to increase `udtconfig` TCA_SIZE to handle cross-group transactions in shared memory, which is faster than using disk to manage the updates associated with a transaction. If the memory buffer is too small, the `reptemp.tca` file is created in the REP_LOG_PATH directory.

- Internal benchmarking suggests that TCA_SIZE=2048 or larger is appropriate. On UNIX, the default setting is (`udtconfig` NUSERS * 64), with a minimum setting of 2048. If you are using TP semantics on Windows, you still need to adjust this manually.
- If you have configured your subscribing server as a standby system for failover, you should make the same change on the publishing server (which will become a subscriber if you fail over to the original subscribing server).

Configure NUSERS to enable all repwriter processes

You may need to configure `udtconfig` NUSERS on your subscribing server. Each repwriter (`udrw`) process requires a slot in this table. These are the processes updating the subscribing server database files. If this value is slightly small, replication may start without the full complement of repwriter processes.

Do not under-configure MAX_LRF_FILESIZE

The `udtconfig` parameter MAX_LRF_FILESIZE is used to determine the maximum size of each body*nn* or info*nn* partfile in log reserve files (LRFs) and log extension files (LEFs). In terms of general tuning, having lots of small partfiles can slow down your replication processes.

The dual-threaded subscriber process makes use of an interim file when processing logs received from a publishing server. This sub packet file (SPF) is structured the same way as the LRF and LEF files, with multiple body and info files. MAX_LRF_FILESIZE is used to determine the maximum size of these partfiles as well. The default setting for MAX_LRF_FILESIZE (134,217,728) allows the subscriber process to perform reasonably well. If you manually change this parameter to a smaller value, you may encounter delays in receiving and processing logs on the subscribing server.

Note: The dual-threaded subscriber process was implemented to boost the performance of replication log processing. Do not under-configure MAX_LRF_FILESIZE and negate this benefit.

Restrict number of files per replication group to udtconfig NFILES

The number of files that a UniData process can keep open at the operating system level is determined by the `udtconfig` NFILES setting. If a process tries to open more files, UniData closes the least frequently used files to allow additional files to be opened. There is system overhead involved with

opening and closing files. The UniData repwriter (`udrw`) processes on a subscribing system obey the NFILES setting. A `udrw` process spawned for a replication group could potentially need to open all of the files in the group, depending on the logs it receives to process.

In order to avoid the unnecessary overhead of closing and reopening files, the total number of files in any single replication group should be fewer than the `udtconfig` NFILES setting on the subscribing server. This guidance can only be general, as you may have many files in a replication group that are never (or very rarely) updated (for example, the dictionary files). If you have dynamic files, the `udrw` process may need to open the `overnnn` files as well as the `datnnn` files.

How can I tell if temp close / reopen activities are occurring on my subscribing system?

UniData provides the ability to monitor system-wide activity associated with closing and re-opening UniData files when the NFILES threshold is crossed. You can use any one of the following three methods:

- Run `$UDTBIN/udtmon` from the shell on the subscribing system.
  - Monitor TempFile Close in the I/O subscreen.
- Monitor TempFile Close in the I/O subscreen.
  - Select **Performance Monitor**.
  - Add the TempFile Close token from the File I/O category.
  - Monitor TempFileClose activity.
- Connect to the subscribing server using the Extensible Administration Tool (XAdmin.)
  - Monitor TempFile Close activity in the UniData Monitor view.

How can I determine how many files are configured in a replication group?

Prior to this release, you had to name each file you wanted assigned to a replication group. Now, you can count the FILE entries in `repconfig` for a group. With account-level replication, UniData searches the VOC of the account you name in an ACCOUNT style group and loads the files (also known as replication objects) when replication starts. There are two ways to view the replication object table for a replication group. After starting UniData on the subscribing server, either:

- Run `$UDTBIN/reptool` from the shell.
  - From the menu, enter `2` (2: Replication Groups).
  - Enter the group number you want to check from the groups displayed.
  - Enter `2` (2: Objects in Group).
- Connect to the subscribing server with XAdmin.
  - Select **Replication**.
  - Click **Diagnosis Utility**.
  - Click **Groups** and select the group number to review.
  - Click **Objects**.

The output from both methods is the same. There are summary lines at the end of the detailed report. For instance:

Total Static Objects in Group = 20

Total Dynamic Objects in Group = 82

To determine the total number of files, add these two numbers together and subtract the number of replication objects that are marked as” Excluded”.

Do not reevaluate virtual field indexes

If you have created UniData alternate indexes with virtual fields on a replicated file, replication automatically sends the values of each virtual field index to the subscribing server in the replication logs. When processing these logs on the subscriber, you can choose to just apply the virtual field index values to the index or to reevaluate the virtual fields before applying the update to the index. This is controlled at the replication group level with a `repconfig` setting, RW_REEVALUATE_VF. If this parameter is set to 0, the logs are applied with the data sent from the publishing server. If you set the value to 1, the replication writer processes recalculate the values of the virtual field expressions before updating the index.

Generally, you would want the virtual field index values to be the same in the index on the subscribing server. An exception might be if you create an index that uses some unique information about the server UniData is running on, such as IP address or hostname. If you don't have this need, you can avoid the unnecessary processing on the subscriber server by setting RW_REEVALUATE_VF=0, thus boosting the performance of the replication writer processes.

Note: If you have virtual field indexes and set RW_REEVALUATE_VF=0, the indexes on the subscribing server must be configured identically to the publishing server.

- The dictionary items used to build the indexes must be identical.
- The order of the indexes in the file must be identical.
- The key length specified when the index is created must be identical.

If these settings are not correct, the updates are still applied to the virtual field indexes on the subscribing server – resulting in logically corrupt indexes. There is no error recorded in any replication error log to alert you to this situation.

You can check and confirm these settings by using the `LIST.INDEX` command on the publishing and subscribing servers for each account that contains replicated files with indexes that include virtual field expressions.

General tuning recommendations

Minimize log extension file (LEF) disk use

1. Configure N_LOGINFO for each group to allow for an adequate number of logs in shared memory.

   N_LOGINFO is the maximum number of replication logs that can be loaded into the shared memory buffer for a replication group. It is set in the `repconfig` file for each group. If the number of logs exceeds this value, the logs are stored in the log extension files (LEF) in the REP_LOG_PATH directory. Ideally, all logs should be handled in shared memory to maximize performance.

   Monitor LEF body and info file use for each group and increase N_LOGINFO to maximize shared memory use. The default setting is 512. Settings in the range of 8192 would not be excessive for a very active group.

   When you change N_LOGINFO, you should also recalculate and reset the replication buffer size for that group. REP_BUFSZ is the shared memory buffer size used to hold the log body information for a replication group. It should be set to N_LOGINFO * average record size. When calculating the average record size, include the key length, the data length, and the length of all indexed virtual fields.

Note: An easy way to see the average record size for all files/records being handled by a replication group is to use the XAdmin replication monitor. Watch an active system for a representative period of time and then divide data replicated by PubDone for each replication group.

2. Configure LARGE_RECSZ to allow logs for large records to be handled in shared memory.

   During the replication process, if a record log size (including the length of the file name, key, record, and indexed virtual field values) exceeds the LARGE_RECSZ value set in `repconfig` for that group, the log record is stored in the log extension file (LEF) on disk instead of in the replication buffer in shared memory. For the best performance, the log should be handled in shared memory.

   LARGE_RECSZ is expressed in kilobytes (KB). The valid range is from 1 to 65535. The default setting is 64. This `repconfig` parameter was added to UniData at version 7.1.12. Prior to that, this setting existed internally and was hard-coded to 16 KB.

   Note: If you increase LARGE_RECSZ to accommodate your data, be sure to review the `repconfig` REP_BUFSZ setting for the group as well. The overall buffer size needs to be bigger than LARGE_RECSZ to allow large records to be processed in shared memory.

How can I determine if I need to tune LARGE_RECSZ?

- Monitor the Log Extension File (LEF) use on disk in the REP_LOG_PATH. Do some groups have large, actively updated INFO or BODY files in their LEF directory? If you have already checked N_LOGINFO settings (as discussed above), you may want to review for large records in your files.
- Review FILE.STAT (or GUIDE_STATS.LIS) output for the files in the replication group. Do you have a large average record size for one or more of the files? Or a large standard deviation from average (suggesting some number of large records in the file)?
- Review the XAdmin replication monitor for the group while it is being actively updated. This number (Data Replicated / PubDone) is an average for all files in the group. It may be skewed by the bulk of the updates for files being small records, but may still be an indication of large record processing.
- You can monitor specific file updates in a group using `reptool`. Scan the report for large “Average log length” values.

Replication log disk location

Place your replication log directory (`udtconfig` REP_LOG_PATH) on different disks from your data files and from your transaction logs. The goal is to distribute disk I/O load across your disks and disk controllers to avoid any bottlenecks.

UniData upgrades and replication

It is possible to use the failover technology within replication to perform a phased upgrade of servers, such as upgrading one server while the other server is being used as the production server. However, the following steps assume both machines are upgraded at the same time.

If you want to perform a phased upgrade, please review the table below or check with U2 Technical Support to see if there are any prerequisites for the version or versions concerned. When running U2 Data Replication, you may have different versions of UniData on your publisher and subscriber

servers. From time to time, however, enhancements or fixes to replication result in a change to the communication protocol used. All systems in a replication configuration must be running the same protocol. Additionally, replication log structure changes can require the installation of compatible releases on all servers, even if the protocol has not changed.

The following table summarizes compatible ranges of UniData versions for replication systems:

Earliest release	Latest release	Notes
6.0.6	7.1.6	Protocol 10002
7.1.7	7.1.7	Protocol 10003
7.1.8	7.1.16	LEF log structure change
7.1.17	7.1.current	System byte/order fix
7.2.0	7.2.current	Protocol 10004 - Account-level replication
7.3.0	7.3.7	Protocol 10006 - EDA single server and wildcard patterns in EXCLUDED_FILE phrase
8.1.0	8.1.0	Protocol 10007 – Data link compression and performance monitoring
8.1.1	8.1.current	Protocol 10008 – Replication pacing and disablement Replication sub-protocol: 1 Replication log file version: 4
8.2.0	8.2.current	Protocol 10009 - Field level replication, delayed standby replication, and asynchronous cross group transactions Replication sub-protocol:1 Replication log file version: 4

If you do not follow these rules about the UniData release level, replication between two systems will not work.

Note: This UniData version compatibility information is also maintained in a U2 Knowledge Base item available on the web. To access this document:

1. Sign in with your U2 Techconnect ID at:

   https://u2tc.rocketsoftware.com/u2techconnect/main.asp?js=y
2. Under **Self help**, select the **U2 knowledge base** option.
3. Search for this title: *UniVerse Release compatibility for U2 Replication Publisher and Subscriber Systems*.

1. Sign in to the Rocket Customer Portal:

   http://www.rocketsoftware.com/support
2. Select **Solutions** from the menu, then **Solution Search**.
3. Search for this title: *UniData version compatibility for Replication*.
4. Select **Exact Phrase** as the **Search Option**.

First steps

Before performing a UniData upgrade, there are some simple steps you should follow to help ensure a successful upgrade and to get the systems working and configured as they were before the upgrade. These steps should also be used in conjunction with the *Installing and Licensing UniData Products* manual.

Make sure that all the replication logs from the publisher have been successfully delivered and successfully applied to the subscriber. (See the previous sections in this guide for information on how to determine this objective has been achieved.)

Stop UniData on the publisher and subscriber gracefully by issuing the `stopud` command without any force options.

It is prudent to perform a full backup of the machines at this point to provide a point of reference that can be restored to, should problems occur during the upgrade procedure.

Copying configuration files

Make a copy of the following files in the `usr/udnn/include` directory for UNIX or the `%UDTHOME%\INCLUDE` directory for Windows:

- `repconfig`
- `repsys`
- `udtconfig`
- `repacct.def`
- `logconfig`
- `arch_backup`
- `arch_restore`

The easiest way to do this is to copy each file so it can be referenced quickly after the upgrade. This could be done by copying each file to a file of the same name, but with a unique extension. For example, `repconfig` might be copied to `repconfig.pug`, and `repsys` to `repsys.pug`, and we might refer to the copied files as the “pug” files. The `.pug` extension is purely a naming convention for this manual and does not have any special meaning for the database.

Performing the upgrade

You should now be able to perform the upgrade in conjunction with the *Installing and Licensing UniData Products* manual.

After the upgrade has been successfully applied, the contents of each of the .pug files should be compared with the original version (for example, `repsys.pug` and `repsys`). This allows any values that have been changed, added, or overwritten through the upgrade process to be changed or corrected, ensuring that the systems are configured, where required, as before the upgrade.

After all the config files and values have been reviewed and adjusted, you are ready to restart the systems using the `startud` command.