UniData dynamic files or Recoverable Files (RFS) can become corrupt on some 7.1 releases on Windows® platforms

Duplicate drive level file identifiers on UniData Windows® systems can result in file corruption and unexpected file or record locks. Your system is at risk if you have more than 65,536 files of any type on a Windows® drive that UniData files reside on and you are running one of the following releases:

UniData Windows® releases affected: 7.1.5, 7.1.6, 7.1.9, 7.1.12

Background:

When UniData opens a file, it relies on the operating system to provide a unique file identifier for that file on a device. On a Windows® drive this is referred to as the 'FileID'. This FileID is used in a number of areas within UniData, including:

Dynamic file current modulo table
Record lock table
Group lock table, including file-level locks
Recoverable file system (RFS) active file table
Data Replication

The Windows® FileID that UniData calculated and used was not always unique. Two or more distinct UniData files on a drive could have been represented by the same FileID if the Windows® generated INBR exceeded 65,536.

Note: On Windows® the FileID is a combination of INBR and INBRH (high) – as displayed in LIST.READU DETAIL.

How can I tell if my system is at risk for any problems?

1. You must be running UniData on a Windows® server.
2. You must be running UniData release 7.1.5, 7.1.6, 7.1.9 or 7.1.12
3. You must have more than 65,536 files of any type on a lettered drive where UniData files reside.

Note: You can determine if any UniData files on a drive resolve to duplicate FileIDs by running a utility program that is available from the IBM U2 Client Support team. Please contact your U2 support provider to initiate a request for this utility.

Resolution:

If you determine that your system is at risk, you should promptly order and install UniData release 7.1.20 for Windows®. Upgrading to UniData version 7.2.0 when it is released will also correct this problem. If you do not need fixes or enhancements contained in 7.1 point releases, you can also downgrade to release 7.1.0 to avoid this problem.

What kinds of problems can occur?

Dynamic file issues:

UniData keeps track of the current modulo of an open dynamic file in shared memory using the device ID and FileID as a key. Due to the apparent duplicate FileID, a UniData process may have accessed or updated a file using an incorrect modulo. This could result in issues with reads/deletes, writes or opens. The write errors could result in data corruption by placing a record in the wrong group or attempting to write a record to a group that was larger than the biggest modulo of the file on disk.

How can you determine if you have experienced any of these problems with dynamic files on your Windows® system?

1. Recognizing if a READ or DELETE failed in your application would depend on how your application handles unexpected results.

2. If a WRITE placed a record in the wrong group, you can see this by running %UDTBIN%\guide on your files. You would see an error similar to the following in the GUIDE_ERRORS.LIS output file:

Group 0, block 1, record number 7 = "FC0088BBL" is in wrong group, key length 9, hash value 48

3. If a WRITE failed because the actual modulo of the file was too small, you would see an error similar to the following recorded in the %UDTBIN%\udt.errlog:

Fri May 23 11:43:01 cwd=D:\ud\DEMO 1:U_pread return error. fd=540, blkbuf addr=1042D518, blksize=1024, blkoff=3072. ret=0, errno=0. Read error in U_blkread for file 'PRODUCT.CATEGORY', key '', number=2

4. UniData also stores the number of primary and overflow part files associated with a dynamic file in shared memory. If a UniData process tried to OPEN a dynamic file and found fewer part files than expected from the shared memory entry, the OPEN would have failed and a message similar to the following would have been written to the udt.errlog:

Tue Jun 17 22:58:33 cwd=d:\ud\demo 1:U_open_large_file error in U_openfile for file 'TRANS', key '', number=0

Note: The problems listed above were experienced by a customer and reported to the U2 support group. The issues listed below could potentially happen (and may have been experienced by customers) but have not been reported or recognized as associated with the root cause issue. The problem descriptions are based on our understanding of how UniData uses file identifiers.

Recoverable File (RFS) Corruption – either static or dynamic files:

In addition to file access issues (noted above) and locking issues (noted below), static or dynamic RFS files could become corrupt.

With RFS enabled, UniData maintains its own cache in a shared memory segment known as the system buffer. The blocks in the system buffer are associated with FileIDs of RFS files recorded in the Active File Table. If there were duplicate FileIDs encountered, data associated with one file may have been flushed to disk and updated to the wrong physical file, causing physical and logical corruption. This would have occurred silently. The corruption will be reported if you run guide on your files.

Once this corruption occurs, it is more likely that UniData will shut down unexpectedly when a UniData process tries to access the corrupted file.

Unexpected failure to acquire a record lock:

UniData record locks (READU, etc) are stored based on the FileID of the UniData file. If two different UniData files share a duplicate FileID, a process requesting a lock for a record key in one file will have to wait if that same key is already locked in the other file.
There would not be any data integrity issues in this case. The impact would be minimal but depends on how your application is coded to handle a request for a lock that is not available.

In addition to a possible delay, waiting for a bogus lock could also result in an unexpected 'deadly embrace' situation. If this had occurred, the final lock request would have been denied and the process requesting that lock would have exited. This event would have been recorded in the udt.errlog as well.

UniData Group and File-level Lock issues:

UniData group locks are maintained by the UniData engine to prevent concurrent updates to a group in a hashed file, dynamic file or node on an index file. UniData may also block updates to an entire file by setting a file-level lock. These locks are managed by the UniData engine and cannot be controlled by an application running on UniData. These locks are also stored on the basis of the FileID.

1. If two different UniData files share a duplicate FileID, a process requesting a lock for a group in one file will have to wait if that same group is already locked in the other file. The only impact is that processes requesting group locks may have incurred more re-tries due to this problem.

2. Under the very rare circumstance where a data file and its own alternate index file happen to share the same FileID, a group lock on the primary file could have been released prematurely during an update to a record. This premature release would have exposed the group to corruption if another process had been waiting to update that group.

3. Additionally, a group could have been exposed during updates to a file that included a virtual field index. If the virtual field index called a subroutine that updated other UniData files, there was some risk of a premature release of a group lock on the primary file (again, if the files in question shared a FileID and the same group numbers were being updated). This rare combination of events could have lead to corruption of the group in the primary file.

4. Regarding file-level locks, your application may have been unexpectedly blocked from running operations that require obtaining a file-level lock. Examples of such commands include:
BUILD.INDEX without use of the CONCURRENT keyword
RESIZE and memresize
CLEAR.FILE

UniData Data Replication issues:

When UniData Replication launches, a process runs that builds a table in shared memory that describes all of the files configured as replication objects. The FileID is part of that table.

1. On a Publishing system, if two files designated to be replicated shared the same FileID, updates to one of those files may have been written to the other file when replicated to the subscriber. This would have resulted in data on the Subscribing system being inconsistent with the Publishing system. There

would have been no problems with the data integrity in files on the Publishing system (beyond those already noted above) as a result of implementing Replication.

2. On a Subscribing system, there would have been no additional integrity or corruption problems even if any UniData files designated as Replication objects shared the same FileID. There may have been delays in acquiring locks to apply updates to the files, but the updates would have been applied.