# When to Use Transactional NTFS - Win32 apps

By jwmsft

Archived: 2026-04-06 02:05:48 UTC

An application can use Transactional NTFS (TxF) to preserve the integrity of data on disk during unexpected error conditions. In general, an application should consider using TxF if the application is flushing files and using other techniques to maintain data integrity. TxF can perform better and simplify the application's error handling code while improving error recovery and reliability. The following sections in this topic provide examples of scenarios for you to use TxF.

The updating of a file is a common and typically simple operation. However, if the system or application fails while an application is updating information on a disk, the result can be catastrophic, because the user data can be corrupted by a file update operation that is partially completed. Robust applications often perform complex sequences of file copies and file renames to ensure that data is not corrupted if a system fails.

TxF makes it simple for an application to protect file update operations from system or application failure. To update a file safely, the application opens the file in transacted mode, makes the necessary updates, and then commits the transaction. If the system or application fails during the file update, then TxF automatically restores the file to the state that it had before the file update began, which avoids file corruption.

TxF is even more important when a single logical operation affects multiple files. For example, if you want to use a tool to rename one of the HTML or ASP pages on a website, a well-designed tool would also fix all links to use the new file name. However, a failure during this operation leaves the website in an inconsistent state, with some of the links still referring to the old file name. By making the file rename operation and the link fixing operation a single transaction, TxF ensures that the file rename and link fix succeed or fail as a single operation.

TxF isolates concurrent transactions. If an application opens a file for a transactional read while another application has the same file open for a transactional update, TxF isolates the effects of the two transactions from one another. In other words, the transactional reader always views a single, consistent version of the file, even while that file is in the process of being updated by another transaction.

An application can use this functionality to allow customers to view files while other customers make updates. For example, a transactional web server can provide a single, consistent view of files while another tool concurrently updates those files.

Note

TxF does not support concurrent updates by multiple writers in different transactions. TxF supports only a single writer with multiple concurrent and consistent readers.

Transactions used with transacted file systems may also be used with the transacted registry. Updates to the file and the registry are coordinated with a single transaction.

By using Distributed Transaction Coordinator (DTC) transactions or System.Transactions, updates made to SQL, MSMQ, and other transactional resources can be coordinated with transacted file updates. For more information, see DTC's IKernelTransaction.

TxF does not support the following transaction scenarios:

- Transactions on network volumes, for example on file shares. TxF is not supported by the CIFS/SMB protocols.
- Transactions on any file system other than NTFS.
- Transacted operations against files cached by client-side caching.
- File access using object IDs.
- Any shared writer scenario.
- Any situation where a file is opened for an extended period of time (days or weeks).

---