Developing a Backup Strategy White Paper

VERSION 2018

jade

Contents

Developing a Backup Strategy

The objective of this white paper is to provide guidance in developing a backup strategy for a JADE database. Hardware and operating system software selection and configuration are outside the scope of this article; however, we assume the requirements outlined in the *Environmental Considerations for Deploying JADE* white paper will be met.

Consider the information presented in this white paper when you construct your disaster recovery strategy. We assume that you are familiar with the operational characteristics of the database, and the available backup and recovery mechanisms as described in the *JADE Database Administration Guide*.

Finally, Appendix A of this document includes a checklist of *best practice* actions to use to qualify your backup strategy and Appendix B contains considerations for your database backup strategy.

For details about developing a backup strategy, see the following subsections.

What is a Backup?

A database backup is a representative copy of data. The copy includes important parts of your database such as the control file and data files, and your backed up transaction journals. When the original data is lost, use the backup to reinstate the physical files that constitute your JADE database.

In the event of a catastrophic failure, your database backup is the key to successfully recovering your data. Additionally, restoring and recovering a database from a backup can be operationally useful. By backing up a database from one computer and restoring and recovering the database from the backup to another computer, a copy of a database can be made quickly and easily.

By default, transaction journals are automatically removed. You must set the JADE initialization file **EnableArchivalRecovery** parameter to **true** so that the journals can be retained for recovery purposes.

Why Are Backups Important?

To establish that backups are important, consider the impact to revenue and customer satisfaction if your production database suddenly became unavailable, even for just five or ten minutes.

What if data files were lost due to media failure and you could not restore or recover them because you did not have a backup? From the perspective of your enterprise, the results could be grim.

To resume operations, you must be able to restore and recover your data quickly. A key factor to your success in this situation is a well-defined backup strategy.

Do I Need a Backup Strategy?

If you can confidently say to yourself "I don't need the data in this database", then no, you don't need a backup strategy.

If the data in the database is of any value to you or your business, it is imperative that a backup strategy be put in place.

You should assume that the database may at some time become damaged or unusable, whether from hardware failure, environmental instability, improper program operation, human error, or malicious intervention.

To get the backup in place, recovered, and online, you must have a valid database backup and sets of procedures to follow.

What is a Backup Strategy?

A backup strategy is more than simply taking a backup of a database and storing it somewhere so it can be fetched and loaded when trouble strikes.

Notes You must be able to get your database back, verified, and online within an expected timeframe. These are the important aspects of a backup strategy.

We cannot stress enough that the requirements outlined in the *Environmental Considerations for Deploying JADE* white paper must be met in order to have some confidence that your database can be restored, recovered, and brought on-line as expected.

It is most important that you verify all data that has been moved. Backed up files or restored files that were corrupted by hardware while they were being written are useless. A cable fault can corrupt data blocks being written to disk.

Knowing how long it takes to backup and verify the database, to restore and verify the database, to restore and verify transaction journals and to perform recovery is also important.

A backup strategy has four components:

- Backing up the database

  This involves making available verified copies of the database files and transaction journals. *Available* means accessible for use in getting your database online. Your requirements determine the forms and frequencies of the backups. They may be offline backups, online backups (or a mix of the two), or you may elect to implement a warm standby server or a Synchronized Database Environment (SDE), which would be immediately available. For details about SDE, see the *Synchronized Database Service (SDS)* white paper.

- Restoring the database

  This involves loading and verifying the database files and transaction journals so that a recovery can be performed. The number and size of your transaction journals is governed by your determination of how much database update activity you can afford to lose.

  In warm standby server and SDE implementations, the database is already resident and ready for recovery once any additional transaction journals are made available and verified. (Additional journal loading is unnecessary on SDS secondaries processing in journal block write mode.)

- Recovering the database

  This involves activating the database to perform roll-forward recovery through the transaction journals. This is a continuous state for warm standby server and SDS secondary implementations where takeover operations can be performed. The failure mode might be such that the transaction journal that was current is not retrievable. In this case, that journal contained the database update activity that you have lost.

- Proving the strategy

  This involves initial and periodic failure drills with critical post-mortem analysis. You must prove the integrity of the strategy by picking a failure mode, dropping the database, and announcing to your team that the database has become unusable. You must prove that every step of your strategy is appropriate, executable, and produces the desired outcome in the expected timeframe.

  These periodic catastrophes highlight any flaws in your processes and enable you to confirm your ability to meet the established critical timeframe for ‘getting back online’.

External Flat Files

You need to be aware of potential mismatches between data stored in the database and data stored in flat files following recovery using stored backups. These mismatches may require addressing with application-specific code. Generally, if data is essential to the integrity of the information contained in the database, then it should be stored in the database, not in external flat files.

As a general concept, where an application has dependencies on volatile external flat files, you may need to store some sort of history or header information in the database for such files, so that the database can be linked to the state of the flat file; for example:

- Has the file been processed yet?
- Is the file on disk the same version as the database thinks it is?
- Should the file actually be there?
- Does the file even exist?

You may need to provide application code that can handle any or all of the above conditions.

Notes If no backup and recovery strategy is in place for files related to this database instance (for example, JADE binaries, third-party binaries, and miscellaneous files), consider including the backup and restoration of those files appropriately within your chosen database backup and restore processes.

Making your database available again in the most up-to-date state as is possible in the timeframe your business can accept is the reason you implement a backup strategy.

The up-time requirements of your database, how long you are prepared to accept it being unavailable when disaster strikes, and how much updating can be lost are the major factors you must consider when developing your strategy.

Availability Requirements

If your database can regularly be taken offline without impacting your business during some quiet period, you can implement a strategy utilizing offline backups captured during this downtime.

If your database is required to be online and available at all times, you must either implement a strategy utilizing online backups captured regularly during some lower-volume transaction processing period, or a warm standby server or SDE-based strategy.

Where online or offline backups are used, the volume of updating transactions and your recovery requirements determine the frequency of your database and transaction journal backups.

Recovery Requirements

When formulating your strategy, try to answer the following questions.

- How much data can I lose? (committed transactions recovery point)

  You can minimize the chances of losing journal data (and where practicable, you should) using various techniques such as mirroring to a separate physical volume. You cannot, however, eliminate the possibility that an event will occur that makes immediate access to the journal data on your drives impossible.

Generally, the frequency with which you want the transaction journal to switch, be verified, and transferred to a backup medium and re-verified is governed by the number of transactions on the database and the acceptable amount of journal data that can be lost if the system has a critical failure. The upper limit of journal data that may be lost is proportional to the number of transactions that occur in the period between journal switches.

The JADE initialization file [PersistentDb] section **JournalMaxSize** parameter specifies the journal file size, which when reached, will cause a switch to a new journal. Once writing focus has been switched from a journal, the journal can be verified and backed up.

It is common to view the issue in terms of time. A database may have a specified **JournalMaxSize** value of 30M bytes and produce one transaction journal per hour for the 10 hours per day that the database is online. Naturally there are busy and less-busy periods, but it is the human way to relate this to one journal being roughly an hour worth of updates.

It isn't true that you always lose this data, but in the worst case, is it acceptable to require the re-entry of about an hour's worth of business operation? Halve the journal file size and it is half the re-entry work. The cost of establishing what exactly requires re-entry is the same, in any case.

If the maximum amount of data loss that can be tolerated is close to zero, you should give serious consideration to implementation of an SDS environment. (See "Synchronized Database Service", later in this document.)

How much data will you risk losing?

- How quickly must I complete the recovery? (recovery time)

  The frequency of database backups has some bearing on the length of time it takes to recover from a failure. If a database with a high rate of transactions is backed up offline only once a week, it can take a significant amount of time to restore and verify the database and to restore and verify the transaction journals in preparation for recovery from a crash occurring late in the week. The recovery processing itself is normally only a fraction of the total overall time it takes to get the database back online.

  Supplementing the weekly offline backups mentioned above with mid-week overnight online backups would approximately halve the number of transaction journals requiring reloading, verification, and reapplication.

  If the maximum down-time that can be tolerated is measured in minutes rather than hours, you should give serious consideration to implementation of an SDS environment. (See "Synchronized Database Service", later in this document.)

  How long can your system be down?

Transaction Journal Backups

Automatic restart recovery after a system failure utilizes the current (online) transaction journal set to recover the database to an operational state.

In a full recovery from a backup, all necessary transaction journal files are used. To effect a full recovery of the database to an operational state from the backup, all archived (offline) journal files created since the backup was performed are required, as well as the current (online) transaction journal set.

For protection against media failure in production systems, always locate the online and archived transaction journals on a different physical volume to the database files, and mirror the transaction journal volume.

You may not be able to fully recover your database if you do not backup consecutively numbered journal files.

Caution Ensure that archived transaction journals are backed up.

Use automated journal close actions, the JADE Database utility, or the JADE database administration framework to perform transaction journal backups. The transaction journal files can be optionally verified (which we recommend).

Specify automated close actions using the JADE initialization file [PersistentDb] section **JournalCloseAction** parameter.

The JADE database administration framework provides the **JournalTransferEvent** event, which is caused when a transaction journal is transferred. This event is signaled to notify a monitoring application that an active journal has become offline and can be backed up. You can use the **backupJournal** method, which supports verification and compression, for this purpose.

The resultant backup file can be verified using the **verifyJournal** method. For details, see the **JadeDatabaseAdmin** class in the *JADE Encyclopaedia of Classes*).

You can perform scripted or manual journal verification by using the JADE Database utility **verifyJournal** function.

Note We cannot stress enough the importance of validating the content of the files written.

If you have adequate disk space, don't remove the archived journals after they have been backed up. Keeping the archived journals resident facilitates faster recovery from a backup, should the need arise.

Offline Full Backups

An *offline* backup involves shutting down the database server before starting the backup and then restarting the database after the backup is complete.

As the database is shut down during the backup process, neither online users nor background processes can access the database for the duration of the backup. You must therefore schedule sufficient time to perform the backup and ensure that the periods when the database is unavailable are acceptable to users.

Use the JADE Database utility to perform offline backups. This ensures that the database is in the correct state and that all necessary database files are backed up. The database files can be optionally verified while being read (which we recommend) and the backup files compressed while being written.

When a file backup completes, the MD5 checksum is automatically recomputed and compared with the value calculated when writing the file.

If you move database backups across media (for example, from disk to disk or tape to disk), it is essential that you verify the resulting files. Use the JADE Database utility **verifyChecksums** function to recalculate and compare the file checksums with those that were originally written to the **backupinfo** file.

Note We cannot stress enough the importance of validating the content of the files written.

It is possible that the operational window of time you have during which you would like to be able to perform an offline backup is too small, due to the size of the database. If so, you will need to perform online backups or to implement a warm standby server or SDE-based strategy.

Online Updating Full Backups

A backup that is performed while the database is active for both read and write access is referred to as a *full online*, or *hot*, backup. As the database can be updated during the backup, the backed up data may be in an inconsistent state. Special begin and end backup records are written to the transaction journal, to bound the backup operation.

The transaction journal file or files must be retained so that backup recovery may occur; that is, the recovery process can recover the restored database files to the fully consistent state as at the time the end backup record was written.

Use the JADE database administration framework to integrate online backup services into your applications or to build standalone database administration applications. Alternatively, the **RootSchema** provides a database backup service that you can incorporate directly into any of your applications. For details, see Chapter 7 of the *JADE Developer's Reference*.

When a file backup completes, the MD5 checksum is automatically recomputed and compared with the value calculated when writing the file.

If you move database backups across media (for example, from disk to disk or tape to disk), it is essential that you verify the resulting files. Use the JADE Database utility **verifyChecksums** function to recalculate and compare the file checksums with those that were originally written to the **backupinfo** file.

Note We cannot stress enough the importance of validating the content of the files written.

Online Quiesced Full Backups

A backup that is performed while the database is in a read-only state is referred to as a *quiesced online*, or *warm*, backup.

The database is placed in a quiescent state by allowing current active transactions to complete and then flushing modified buffers from cache to the stable database. In this mode, the physical database files contain all committed updates to the database and as the database files are opened with shared read access, external backup processes can safely copy the database files as a consistent database image.

During a quiesced backup, updating transactions are not permitted and attempts to execute database transactions raise an exception. When a backup is performed in a quiescent state, the physical database files are guaranteed to contain all database updates. Unlike a full online backup, a quiesced online backup does not require backup recovery when the database files are restored.

Use the JADE database administration framework to integrate online backup services into your applications or to build standalone database administration applications. Alternatively, the **RootSchema** provides a database backup service that you can incorporate directly into any of your applications. For details, see Chapter 7 of the *JADE Developer's Reference*.

The database files are verified while being read and optionally compressed while being written.

When a file backup completes, the MD5 checksum is automatically recomputed and compared with the value calculated when writing the file.

If you move database backups across media (for example, from disk to disk or tape to disk), it is essential that you verify the resulting files. Use the JADE Database utility **verifyChecksums** function to recalculate and compare the file checksums with those that were originally written to the **backupinfo** file.

Non-JADE Backups

It is *never* valid to backup a JADE database that is not closed, quiesced, or conditioned for external third-party snapshot, by using non-JADE backup software.

When a third-party tool is used to backup a JADE database, the backup must be taken when the database is closed, quiesced, or conditioned for an external third-party snapshot.

The preferred option is that the backup should be taken when the database is closed, as this allows use of the batch JADE Database utility (**jdbutilb**) custom backup support **convertToBackup** feature.

This verifies the files and creates the **restoreinfo** file that contains MD5 file checksums, and an operational log in the database directory. If the **convertToBackup** operation is successful, you can then perform the required action with the database image; for example, copy the files to tape.

If ever the files are copied to a new disk location, they can be verified using the JADE Database utility **verifyChecksums** function, which will recalculate and compare the file checksums with those that were originally written to the **restoreinfo** file.

If the backup is to be taken while the database is quiesced, call the **JadeDatabaseAdmin** class **changeDbAccessMode** method, specifying **Mode_Archive** and **Usage_ReadOnly**. To return the database to operational state when the backup is complete, call the **changeDbAccessMode** method, specifying **Mode_Default** (the usage parameter value is not used when the mode is specified as **Mode_Default**). For details, see the **JadeDatabaseAdmin** class in the *JADE Encyclopaedia of Classes*.

You can also do this externally, by using the **jomChangeAccessMode** Application Programming Interface (API) call. For details, see "Changing the Database Access Mode", in Chapter 3 of the *JADE Object Manager Guide*. This process results in a consistent image of the database being backed up, but the database was not verified during the backup and the resultant image does not contain the checksum information necessary to check the integrity of the image when it is restored.

Note When a primary changes to archive mode, the SDS service, if active, is stopped. The service is restarted as necessary when exiting from archive mode.

If the backup is to be taken while the database is being updated, call the **JadeDatabaseAdmin** class **changeDbAccessMode** method, specifying **Mode_Snapshot** and **Usage_Update**. This conditions the database for snapshot recovery, as follows.

1. A system quietpoint is established and a checkpoint taken (**reason=start snapshot**).
2. A copy of the control file conditioned for snapshot recovery is made (**_control.snap**), and control state is marked as *snapshot active*.

   The **_control.snap** file must be copied in the backup.

To exit from snapshot mode when the backup is complete, call the **JadeDatabaseAdmin** class **changeDbAccessMode** method, specifying **Mode_Default**. This transitions the database out of snapshot mode, as follows.

1. A checkpoint is taken (**reason=end snapshot**).
2. An **END_SNAPSHOT** record is written to the journal (recovery from the start snapshot checkpoint record up to this record is necessary to restore file integrity).
3. Snapshot active control state is cleared.

You can also do this externally, by using the:

- **jomChangeAccessMode** Application Programming Interface (API) call. For details, see "Changing the Database Access Mode", in Chapter 3 of the *JADE Object Manager Guide*.
- **jdbadmin** program **StartSnapshot** and **EndSnapshot** database actions, to condition the database for recovery from a non-JADE backup and to take the database out of snapshot mode, respectively. (For details, see "StartSnapshot" and "EndSnapshot", respectively, in Chapter 2 of the *JADE Database Administration Guide*.)

Caution It is essential that the backup processing occurs entirely while the database is in snapshot mode, and that as a minimum, journals from the start snapshot checkpoint to the **END_SNAPSHOT** record will be available for snapshot recovery.

If the database is accidentally closed while it is in snapshot mode, do *not* restart the database until the backup has completed. When restarted, the database will perform a snapshot recovery.

To recover from a third-party backup taken while in snapshot mode, perform the following actions.

1. Restore the system using the backup software tools.
2. Copy any necessary journals into place.
3. Initiate recovery with the JADE Database utility or by starting the database.

Snapshot recovery is performed when database recovery (from database open, or JADE Database utility-initiated recovery) detects the database is in snapshot mode. The **_control.dat** file is replaced with the **_control.snap** file and a roll-forward recovery is performed. If the recovery was JADE Database utility-initiated, the roll-forward termination condition is that specified by you; otherwise the roll-forward recovery is to end-of-last-journal.

Roll-forward recovery logic looks for the **END_SNAPSHOT** audit record. This indicates the point at which file integrity is known to be restored, assuming that the snapshot completed and the recovery is not due to restart.

Recovery due to restart will encounter the **LOG_TRAILER** or a header discontinuity, set snapshot state to incomplete, and then continue recovery as normal.

Note If a roll-forward termination condition was specified and that condition is met before the database can establish an end-snapshot condition, the recovery will fail with a 3192 error (*Snapshot recovery terminated before end-snapshot condition established*).

The end-snapshot condition is satisfied when an **END_SNAPSHOT** record is encountered, or in its absence, a **DATABASE_CLOSE** or **DATABASE_OPEN** record is encountered. Snapshot state is set to recovered, and recovery then continues as normal.

If the third-party backup tool is file-based rather than block-based, it will not be able to copy **_control.dat**, which is opened in exclusive mode by the database. This file can be excluded from the backup and when the database is restored, the **_control.dat** file can be recreated by copying **_control.snap** as **_control.dat**.

It is usual for third-party block-based backup tools to copy blocks changed since the last backup, and blocks that change during the backup. Therefore at the end of the backup, the image will contain all updates to journals in the **journals\current** directory.

Restoring such a backup and initiating the system without otherwise restoring journals will behave as a snapshot recovery due to restart.

To achieve the same behavior when the third-party backup tool is file-based, however, requires that the backup tool copies the journals *after* the data files have been copied.

Warm Standby Server

It may be desirable to maintain a copy of your database on another machine in standby mode. You can quickly and easily make a standby copy of a database by restoring and recovering a backup on another machine.

As transaction journals become archived (or offline) on your primary system, you can then copy them to the standby system, verify them, and apply them to the database.

When a transaction journal switch occurs, a journal switch control record is written to the end of the journal. When this record is processed by roll-forward recovery and it is the final record in the transaction journal, the recovery process knows that later transaction journals can exist so the database state is maintained such that a further roll-forward recovery from that point is possible.

When the batch JADE Database utility (**jdbutilb**) is run to recover the database, it processes the next journal and whatever in-order later journals exist in the directory.

The interactive JADE Database utility (**jdbutil**) processes journals in the same manner except that if a journal switch control record is found at the end of the last journal in the directory, it waits for the next journal file to be provided or for the user to finish or abort the recovery processing.

Both versions of the JADE Database utility support continuing roll-forward recovery over multiple process executions. Any other open of the database cancels the preserved state that enables continuous roll-forward recovery.

While warm standby servers can provide adequate functionality, a fully automated and capability-rich standby server implementation is available with the Synchronized Database Service, described in the next section.

Because of byte ordering or operating system differences, the format of physical database files differs between hardware architectures and/or operating systems. You therefore cannot interchange database files directly by backing up a database on one machine and then restoring the database on a machine with a different operating system or hardware byte order.

Synchronized Database Service

The Synchronized Database Service (SDS) is the name given to the optional software service in a JADE system that keeps one or more secondary databases automatically synchronized with a primary database.

SDS automates the otherwise-manual processes of maintaining a standby database server, which can be used if the primary database is taken offline for routine maintenance, becomes damaged, or is lost entirely.

SDS provides several management interfaces, including an API, JADE initialization file parameters, and the JADE SDS Administration utility that presents a graphical user interface for monitoring and controlling an SDS environment.

You can synchronize secondary databases to a journal file boundary or to a journal block boundary within the current transaction journal, as specified in the JADE initialization file [SyncDbService] section **SyncMode** parameter, which you can set to **JournalSwitch** or to **JournalBlockWrite**.

A JADE Synchronized Database Environment (SDE) combines loosely connected and geographically dispersed primary and secondary databases into a robust, easily managed disaster recovery solution. It is a superior data recovery solution compared to backup and recovery from tape, which is generally incapable of meeting high-availability recovery time objectives. JADE's SDS facility provides hot standby secondary databases, which you can use for both disaster recovery and for offloading query workloads.

For further details, see the *Synchronized Database Service (SDS)* white paper and the *JADE Synchronized Database Service (SDS) Administration Guide*.

Verify, Verify, Verify…

If you do not incorporate verification into your strategy, you are courting disaster. For its intended purpose, a corrupt database backup is worse than useless.

Whenever data is moved, verify the database before opening it (using the JADE Database utility **verifyChecksums** function) and verify journals (using the JADE Database utility **verifyJournal** function).

Don’t Be Afraid to Ask For Advice

The administration of databases was once the domain of specialists. It is a different world today, and JADE is more than just a Database Management System (DBMS). *You* can be its administrator.

The decisions you are required to make when formulating, implementing, and proving your strategy might seem daunting at first.

As you become more familiar with the concepts and the mechanics of the processes involved, you will find that while you have to be particular and correctness is important, database administration isn’t *too* difficult.

The strategy decisions you are making are data-based in that it is the nature of the data in the database and your attachment to that data that drives your decisions.

It is for this reason that you might implement different strategies for different databases in your organization.

Then again, for the sake of simplicity, you may decide to treat your smaller and less business-critical databases in the same manner as your primary business database because it makes operational sense, piggy-backing on the primary backup strategy.

Every set of circumstances is different.

If in doubt, ask the people who make JADE, by contacting your local JADE support center or JADE Support if your JADE licenses include support. Alternatively, see the JADE Forums (https://forums.jadeworld.com).

Appendix A Database Backup '*Best Practice*' Checklist

The following is a checklist of database backup best practices.

- Schedule and take backups on a regular basis
- When performing backups, never overwrite the last good backup until the current backup is completed and verified
- Consider and implement the use of RAIDed disk volumes for storage of all volumes associated with your system, including database files, journals, other flat files etc.
- After journals have been released, verify them by using the JADE Database utility **verifyJournal** feature, and back them up.
- Place archived journals on a different physical disk and/or machine from the current journals
- Store backups and associated journals securely, both off-site and on-site
- Regularly perform a physical certification of the database, by using the JADE Database utility **certify** feature, and investigate and address any errors or warnings
- Regularly test the restore and recovery process, using recent backups
- After you move database files from one location to another (for example, from disk-to-disk or tape-to-disk), perform the JADE Database utility **verifyChecksums** feature *before* opening the database for recovery or for other use
- After you move journals from disk to disk or from tape to disk, verify them again by using the JADE Database utility **verifyJournal** feature

Appendix B Backup Strategy Considerations

This appendix elaborates further on considerations for your backup strategy.

Data Loss Strategy

Although there are many technical and procedural safeguards in place to protect against the loss of data, there remains the remote possibility of a situation developing that results in the loss of previously captured or committed data.

Complete data loss is unlikely, as long as regular copies are made and stored safely away from the production (or main) system. In addition, a stand-by system might be employed using SDS, which enables a much faster recovery in the event of a disaster. However, a hardware, low-level software, or firmware fault, or a combination of these faults might provide a circumstance in which a system needs to be recovered from a previous backup, and for some reason cannot be rolled forward to the last point of user data input. This would result in a period of recently applied transactions being lost.

For any IT system, you need to consider the consequences of data loss *prior* to the event occurring, in order to decide whether proactive strategies to mitigate the impact are warranted. Valid strategies might involve both technical and procedural or business-related actions.

Importantly, you need to consider the impact of data loss not just in terms of the specific application in which the data loss has occurred but also the effect on any other applications that interface with it. This can be a complicated situation that developers need to consider at the design stage of any systems that receive or send updating transactions with other systems.

Note There is no single solution available, but it must be given careful thought in the design of each interface from each system's perspective.

A basic design consideration in cases where separate systems update each other is to ensure that the interface has a form of sequence checking that can detect whether one of the systems has lost data; that is, has gone backwards in time. For example, if application **A** sends transactions to update application **B**, the interface at both ends might keep an incrementing sequence reference of the last processed transaction, which is compared before processing any new transaction. In the event that the sequence records are not synchronized, processing could be halted and action taken to address the problem.

You should consider design functionality to assist in the recovery in such a situation. For example, if application **B** has lost data, application **A** might be designed to retain the source of the transactions previously sent to application **B** in such a way that it has the ability to safely re-send previously sent transactions, if required.

Your system must have the ability to deal with a situation in which application **A** has lost data and therefore application **B** contains transactions as a result of data that longer resides in application **A**. The solution will be unique for each situation, depending on the type and inter-relationships of the data concerned.

Storing Data External to the Database

All persistent data should be stored in the JADE database. You should consider at risk of full or partial loss any data necessary for the operation of the solution that is represented in the form of non-database files (flat files), and you must be able to be recreate those flat files somehow, whether that be from the JADE system itself or from an external system or user.

If there are compelling reasons for the application to maintain data in the form of flat files external to the database, you must be mindful of maintaining both integrity and performance.

Referential Integrity

Data retained outside the database is not protected to the same level as persistent database data. This is because the database journaling process does not apply to flat files and also because the files could potentially be manipulated outside the application. For example, in the event of a disk subsystem failure, it is highly possible that the database will be restored to a timestamp that differs from that of the flat files. This will corrupt any references contained in the database to the flat file data.

Another significant consequence of maintaining data external to the database is that JADE's SDS functionality that can be used to create warm standby systems for data recovery will not maintain the non-database files; that is, the data recovery system will not have these files available.

You must consider the following suggestions to allow for unusual situations.

- If the database is to contain references to essential non-database file data, the application must provide check, resynchronization, and repair facilities that can be run on a regular basis or in the event that the integrity is suspect. Essentially, a facility must exist in the application to detect references (within the database) to missing files or orphan files, as well as a means of repairing them.

  Repair might be manual by the user once the corruptions have been identified. In order to achieve this functionality, it is likely that the database will need to be able to authenticate each file. This necessitates the use of algorithms such as MD5 file hashes and storing the result with the database reference to detect the authenticity of each file.

- Write access to entire directory tree down to file level is to remain exclusively the domain of the application. Read-only access can be permitted to other applications or users, if required.

Performance

In addition to the integrity issue discussed earlier in this appendix, there can also be performance issues if large quantities of flat files are stored in a Windows NTFS file system. A high volume of small flat files has an adverse impact on functions such as backup, defragmentation, virus checking, and general directory searching.

To avoid these issues, take the following precautions.

- Use a structured tree of data-dependent branches, to ensure that the quantity of files contained in any directory or folder is kept to a nominal level of files for file system performance.

- Maintain references to the entire tree, including the files in the database.

- You should consider the maximum number of files that can be practically stored on an NTFS volume to be approximately 250,000.

  Path references to a reasonably low level should therefore be soft-coded in the application, to allow file system separation, if required.