

PC Connectivity: How To Write Backup Aware Software for Symbian OS v9

Paul Newby, PC Connectivity

Revision 1.4, June 2006

Contents

1	INTRODUCTION	3
1.1	PURPOSE AND SCOPE	3
1.2	WHAT YOU SHOULD DO AS A DEVELOPER	3
2	OVERVIEW AND BACKGROUND.....	4
2.1	FILE LOCKS AND PUBLIC FILES	4
2.2	PRIVATE DATA AND DATA CAGING REQUIREMENTS.....	5
2.3	ACTIVE AND PASSIVE BACKUP.....	5
2.4	BASE AND INCREMENTAL BACKUP AND RESTORE.....	6
2.5	FULL AND PARTIAL BACKUP AND RESTORE OPERATIONS	6
3	BACKUP AWARE BEHAVIOUR	7
3.1	BACKUP AND RESTORE SIGNALLING	7
3.2	DEFAULT BEHAVIOUR FOR GUI APPLICATIONS.....	7
3.3	SERVERS THAT HOLD LOCKS FOR CLIENTS	8
3.4	SERVERS AND APPLICATIONS THAT HOLD OTHER LOCKS	9
3.5	REBOOT AFTER RESTORE.....	9
4	EXPECTED BEHAVIOUR FROM DATA OWNERS	10
4.1	CRITERIA FOR BACKING UP PRIVATE DATA.....	10
4.2	CRITERIA FOR CHOOSING ACTIVE OR PASSIVE BACKUP	11
4.3	OWNERS OF PRIVATE DATA TO BE ACTIVELY BACKED UP	12
4.4	OWNERS OF PRIVATE DATA TO BE PASSIVELY BACKED UP	13
4.5	OWNERS THAT DO NOT WANT PRIVATE DATA BACKED UP	14
4.6	OWNERS WITH DATA THAT RESIDES WITH A PROXY (E.G. THE CENTRAL REPOSITORY)	14
4.7	OWNERS OF DBMS FILES TO BE BACKED UP.....	15
4.8	OWNERS OF SYSTEM DATA (INSTALLED APPLICATIONS) TO BE BACKED UP.....	15
5	BACKUP REGISTRATION FILE FORMAT	16
5.1	CONTENT OF THE BACKUP REGISTRATION FILE	16
5.2	EXAMPLES OF BACKUP REGISTRATION FILES	17
5.3	ADDITIONAL BACKUP REGISTRATION FILES	18
6	TESTING BACKUP AND RESTORE.....	19
APPENDIX A	BACKUP REGISTRATION FILE DTD.....	20
APPENDIX B	ACTIVE BACKUP CLIENT API.....	21

1 Introduction

1.1 Purpose and scope

Symbian OS includes software to carry out backup and restore operations from a connected PC. In order for this software to function correctly, it requires some co-operation from other software (servers and applications) running on the phone. This document describes the required co-operation, how an application or server needs to behave in order to participate in backup and restore operations.

This document is intended for any developer creating software to run on a Symbian OS phone. This document covers behaviour of Symbian OS v9.1 and later versions.

1.2 What you should do as a developer

If you are responsible for an application that owns data then read through the following steps:

1. Familiarize yourself with the backup and restore concepts so that you can then make the right decisions regarding the backup and restore of your application and its data. Read through Section 2.
2. Decide what mechanism will be required in order for your application to become backup aware, refer to Section 3 for details.
3. Decide which data needs to be backed up and restored and how it should be dealt with. This will require the following decisions to be made, more details can be found in Section 4:
 - a. Decide whether or not you have public or private data that needs to be backed up.
 - b. Decide how you will react to backup and restore operations (i.e. do you need active or passive backup and restore).
 - c. If you need to implement active backup then do so.
 - d. Decide whether or not you wish to have your system files backed up.
4. Produce a backup registration file that covers the decisions made above. Refer to Section 5 for details.

2 Overview and Background

2.1 File Locks and Public Files

Backup works by copying files and data from the phone to the PC. Restore works by copying the files and data in the other direction. The backup and restore operations rely on being able to copy files when they need to. If a file is locked then it cannot be copied. In normal phone use, a range of servers and applications will have files open for reading or writing and may have updates pending on some files.

Files in the public area of the filing system can be backed up and restored directly by the PC Connectivity subsystem but this requires all other processes to free up file locks when required. Platform security (introduced in Symbian OS v9), and specifically data caging, provides for ownership of private data files (see Section 2.2). Ownership of public data files is more complex.

- A backup operation needs to read files so processes must relinquish exclusive-locks on files but can retain read-locks (although in practice many processes just relinquish all locks for the sake of safety and simplicity). In order for a backup operation to take place, applications and servers must flush any pending updates to files and allow all files to be read (but cached data can be retained as backup will not alter data files). When the backup has taken place, servers and applications can re-take file locks and carry on.
- A restore operation requires exclusive access to files so processes must relinquish all locks on files. In order for a restore operation to take place, applications and servers must discard all cached data and allow files to be written or overwritten. When the restore has taken place, servers and applications must reload their data from files that can be expected to have changed.

2.2 Private data and data caging requirements

With the introduction of platform security, directories are divided into public and private areas. Public files can be viewed and altered by any application (although file locks may still get in the way). However, processes can define private data areas where they store files that only they can operate on. The system also defines some areas as private to store executables and read-only data. The choice of whether and what files should be stored in public and private areas is devolved to the servers and applications (with some audits). All of the restrictions can be overridden by executables that have the requisite privileges (the use of these is minimized on principle).

As a principle, the backup process attempts to protect private data in the following ways:

- Application executables will not be backed up without the consent of the developer. This means that if you develop an application and forget about backup, you should not find your application backed up. It is normally meaningless to restore an application without at least some of its private data.
- Private data will not be backed up without the consent of the data owner. This means that if you don't explicitly enable the private data to be backed up, you should not discover it has been exposed without your knowledge.
- Private data may be encrypted before delivery to the backup client (normally a PC). The backup process supports encryption and provides the hooks for key access but does not guarantee that a useful key will be used. (This relies on the phone manufacturer to include a specific mechanism to provide the encryption key). It should be assumed that the user will have access to the encryption key and so application developers should not assume that data will be protected from the user. For this reason, encryption of backed up data may be omitted.

2.3 Active and passive backup

There are two ways of backing up private data:

1. *Active backup of private data.* In this model, the process which owns the data, registers with the secure backup engine using a registration file. The secure backup engine will start any process registered for active backup if not already started. The data owning process then responds to a central signal when a backup or restore operation takes place and actively provides its private data to or receives it from the secure backup engine. This requires that the data-owning process include specific code to take part in backup and restore operations, and that it must be running when a backup or restore takes place. In this model the data-owning process registers with the secure backup engine but exercises complete control of which data is backed up and restored.
2. *Passive backup of private data.* In this model, the process which owns the data, registers with the secure backup engine using a registration file and records whether executables, private files or directories of private files should be backed up. The data-owning process then releases file locks for private files in the same way as for public files (described above) and the files are backed up by the secure backup engine (which has the required capability to access private data files belonging to other processes).

It can be seen that passive backup is very simple for developers to implement – all that is required is a registration file that defines which private files should be backed up or restored and the same type of lock-releasing behaviour already required for public files. Active backup requires more effort to implement but provides the data-owning process with more control over the data backed up and restored.

It should be noted that data-owning processes can also list public files or directories that should be backed up or restored along with their private data in cases of a partial backup or restore.

2.4 Base and incremental backup and restore

Backup is intended to be a routine operation for the user. Therefore, it should be as quick and painless as possible. For this reason, Symbian OS implements base and incremental backups. A base backup (sometimes described as a full backup) is a backup of all or part of a drive that includes all the selected files. In contrast, an incremental backup is a backup of all or part of a drive that only includes files that are new or changed since a previous backup. An incremental backup is thus smaller than a base backup and will take less time.

It should be noted that there are (at least) two possible patterns of use for incremental backups.

1. The first pattern involves taking one base backup and then a series of incremental backups with each increment including only the files that are new or changed since the last increment. This pattern minimises the time taken for each increment but has some drawbacks for restoring private files (because all the increments are required for restore operations).
2. The second pattern involves taking one base backup and then a series of incremental backups with each increment including only the files that are new or changed since the base backup. In this pattern, the time taken for an incremental backup is not minimised and may approach the time taken for a base backup but a restore operation only requires the base and a single increment.

The choice of which pattern to use will be made by the PC software - not by the phone software.

It should be noted that an incremental backup cannot be created purely based on a date and time stamp. If a new file is added with an old time-stamp then a time-based increment would omit it. Therefore, any incremental backup requires a list of the files included in the preceding backup.

2.5 Full and partial backup and restore operations

By default, a backup (whether base or incremental) aims to include all files on a specified drive. In practice, the user may only care about data belonging to one or more specific applications. By omitting other data files the time taken for the backup (or restore) is reduced. It can be seen that the choice between full and partial backups is orthogonal to the choice between base and incremental backups.

Versions of Symbian OS prior to v9 do not support partial backup or restore well. This is because all files are public and Symbian OS provides no means of associating specific files with specific applications. Some phone manufacturers have provided (at least partial) solutions to this problem but the extension of these categories to third-party software is problematical. Similarly, some PC suites have provided partial restore but the association of files with applications is problematical.

Because platform security requires a process (application or server - basically a private data owner) to specify the private files to be backed up, there is a close association of private files with specific applications. In addition, it is possible to associate public files with specific applications. However, there is no guarantee that more than one data owner will not lay claim to any public file and there is no guarantee that all public files will be associated with a data owner.

An additional possible benefit of partial backup and restore is that a data owner may be able to participate in backup or restore without requiring all file locks to be freed. For example, if a data owner only deals with its own private files then it may be possible to back up its data in isolation. This would allow backup operations to have less impact on the user.

On platform security-enabled Symbian OS phones, installed applications are no longer backed up by default as was the case on earlier releases of Symbian OS. The backup and restore of installed applications is now requested via a tag in the backup registration file – more details can be found in Section 4.8.

3 Backup aware behaviour

3.1 Backup and restore signalling

The signalling method provided to inform processes that a backup or restore is taking place is via the publish-and-subscribe API. The publish-and-subscribe server maintains a list of subscribers expressing an interest in the backup and restore flag. Any published changes to this flag by the backup and restore engine results in all subscribers being notified. The flag provides information on whether a backup or restore operation is in progress, whether a backup is base or incremental and whether the operation is full or partial.

Subscribing to the flag is done via RProperty, which must be used in conjunction with an active object to be notified when the value changes. The following key and category values should be used (these are defined in `epoc32\include\connect\sbdefs.h`):

Category : KUidSystemCategoryValue

Key : KUidBackupRestoreKey

The following code fragment demonstrates this:

```
#include <e32property.h>

RProperty iProperty;

iProperty.Attach(KUidSystemCategory, KUidBackupRestoreKey);
CActiveScheduler::Add(this);
iStatus = KRequestPending;
iProperty.Subscribe(iStatus);
SetActive();

// In RunL, to get the state:
TInt backupStateValue = 0;
iProperty.Get(backupStateValue);
```

See the technical paper [Publish and Subscribe](#) for more information about the publish-and-subscribe API.

Applications being made backup aware should use publish-and-subscribe (P&S), however there's another mechanism still present but should not be used in favour of P&S which will be deprecated in a future OS release: the Base backup server. It is now only used to manage GUI applications during a backup or restore operation. When a backup or restore takes place, a signal is passed from a backup client on the PC or phone and a number of special servers take action. The Base backup server maintains a list of servers that have registered an interest in backup and restore and invokes observers to take appropriate action. When a backup or restore is complete, the Base backup server informs its observers that normal service has been resumed.

3.2 Default behaviour for GUI applications

The behaviour for GUI applications is deliberately straightforward: by default, all GUI applications are politely terminated when a backup or restore operation takes place. As a result of most applications exiting, many system servers release file locks. This technique avoids GUI application developers having to implement backup- and restore- aware code in most cases; the relevant server and exit code is implemented within the Uikon subsystem.

At its simplest, a GUI application can take no specific backup or restore action. If a GUI application does not have the 'system' status it will be terminated when file locks are required and will be re-started after the backup or restore. The terminate-restart behaviour will ensure that direct and

(most) indirect file locks will be freed. A potential optimization for a GUI application is to store the current view and state when terminated and to adopt the same view when it restarts (this choice may be required or deprecated as part of UI design guidelines outside the scope of this document). This will make for a seamless restart after a backup, i.e. the application will appear the same to the user, as if it had not terminated and restarted.

The UIKON backup server terminates most running GUI applications (all GUI applications that are not registered as system applications). When a backup or restore is complete, the UIKON backup server restarts the GUI applications that were terminated. A GUI application that must not be terminated during a backup or restore, a 'system' GUI application, must behave in the same way as a server (see below) and manage its own file locks directly. Only phone manufacturers should develop such applications because of the possible side effects, the default behaviour should be to accept termination.

3.3 Servers that hold locks for clients

Some servers access files only as a result of calls from clients. As long as these servers only have well-behaved clients they need take no special action. Their clients will either terminate (if they are GUI applications) or will drop connections (if they are well-behaved servers or system applications) so the server will then drop locks on files.

However, the developer should be aware of the sequence of actions during a backup or restore as some behaviours, which might appear to be performance optimizations, could prevent a successful backup:

- If a server was to keep a file open after all attached clients had closed that file (for example, to improve performance in case a client was likely require it again), then the backup and restore operations would be blocked.
- If a server was to keep cached data in case the same file was re-opened then a restore operation could be corrupted.

If such optimizations are required then the server must become backup-aware and drop file locks promptly and flush data caches on backup and restore operations.

3.4 Servers and applications that hold other locks

Servers (and other applications) that do not terminate during a backup or restore operation and do not drop file locks because of their clients' behaviour need to become backup aware in their own right.

If a process is not liable to external events during backup and restore operations then it simply needs to react to backup and restore events and release file locks. However, if a process can receive external events during a backup or restore (such as a telephone call, an incoming message or an external request for some action) then the process needs to ensure that the event does not interfere with any files during the backup or restore. Examples of possible interference include:

- Writing to call log files when a telephone call is received;
- Storing an incoming message in the messaging store;
- Applying an incoming configuration message.
- How the process handles the event is up to the process developers but some possibilities include:
 - Ignoring incoming events (only acceptable if they will be automatically re-sent later);
 - Caching incoming events in memory or in a safe private file and then applying them after the backup or restore operation.

3.5 Reboot after restore

The reboot after restore flag is currently ignored. This is due to Symbian OS, currently, providing no standard way to ensure a reboot of a phone. Applications should be written in a manner which will not require a reboot as a reboot can not be guaranteed. (When Symbian OS support a standard mechanism for rebooting a phone then the functionality will be enabled.)

The reboot after restore flag will be made available to clients using the backup architecture so phone manufacturer-specific reboot mechanisms can be used.

4 Expected behaviour from data owners

If you are the developer responsible for a process (application, server or some other component) which accesses public files or which owns private data files then you need to go through a number of steps to make your process participate in the backup and restore operations:

1. Decide whether any of your private data should be backed up, and if so, how much of it.
2. If you have private data to back up then decide whether you will use the active or passive backup approach.
3. Based on the answers to the previous stages and on whether your process is a GUI application or not, decide how your process needs to respond to backup and restore events.

Note that for Java MIDlets, the registration file is created by the SystemAMS component because of Java's method for data caging MIDlets.

4.1 Criteria for backing up private data

If you are the developer responsible for creating or maintaining a process which owns private data then you will have to decide if your processes should take part in the backup of private data or not. In general, as much private data as possible should be backed up (to help the user) but there are some criteria, which indicate data which should not be backed up:

- If the private data must not be exposed to the user then it should not be backed up. Assume that all data backed up can be read and potentially altered by the user. This might change if a phone manufacturer implements encryption which does not rely on keys provided by the user but that is a phone manufacturer-specific decision and cannot be guaranteed.
- If the process cannot keep the private data stable during a backup or restore then they cannot safely back it up. This is a poor reason for not backing up data but may be a practical one. It depends on the sources of external events and whether they can be delayed or placed in a temporary location.
- If the private data can be easily recreated by other means then it may be excluded. This may not be a good criteria, but if a process would find it difficult to back up private data (perhaps for the reasons above) and the data could easily be created (for example by receiving configuration messages) then choosing not to back it up may be more acceptable.

In addition, consider the implications on other processes of backing up the data or not. If your data must remain consistent with that in public files then the private data must be backed up. If your data must remain consistent with that owned by another process then they should both back up data or neither.

4.2 Criteria for choosing active or passive backup

As described above, passive backup and restore of private data is simpler than active backup. It also takes less system resources because processes taking part in active backup must be running during a backup or restore. However, active backup allows the data-owning process more control over the data and this can have both security and performance gains.

The choice between passive and active backup may not be a black-and-white one - it may involve weighing up a number of factors. The following list includes a range of advantages and disadvantages and you need to consider all of these and make a decision.

1. The registration for passive data backup only specifies whole files or directories. It is not possible to backup only part of a file using passive backup. If you need to backup only part of a file you need to either split the file into two or use active backup.
2. The registration file that defines the files and directories to be subject to passive backup is created at system build time (for a system process) or at installation time (for an installed application). The registration files cannot be modified or created at runtime. If your process will create new directories after installation then the registration file cannot be extended for these. If you want these backed up then you need to either put them under a common directory which all gets backed up or use active backup and make a run-time decision on what to backup. This aspect may be partially or wholly addressed by installing additional backup registration files.
3. Backed up data is compressed en-route to speed up data transfer (this can be disabled). However, if your private data has a specific format and you can manage to backup only a fraction of it you may be able to make the backup of your data more efficient by implementing active backup.
4. Backups can be base backups or incremental backups. With passive backup (as with the backing up of public files) incremental backups include all relevant files which have changed or are new since the last backup took place. If a file contains record-based data (such as a database) then the addition or changing of one record will cause the whole file to require backup. If you implement active backup then you could only backup the changed or new records (plus recording deleted records in some way). This could increase the performance of backup and restore for your data.
5. Active backup requires your process to be started during a backup or restore operation. Your process will be started automatically as required by the secure backup engine if not already running. If your process normally runs at all times then this imposes no additional burden on the phone. However, if your process is not normally running then it will require additional resources during a backup or restore and you should consider if the benefits of active backup justify the additional burden.

One approach which has been considered by some developers and which may be useful is that of a hybrid approach. The secure backup engine allows a data owner to register as both an active backup client and a passive backup client. This was originally intended to allow a data owner to back up some data in each way but an alternative use is for the active backup client to prepare data for backup and then let the secure backup engine handle the actual transfer as a passive backup. The active backup client does not need to actually provide any data.

In order to support this, the secure backup engine should not retrieve data for passive backup until an active backup client has signalled that the data is ready (this set-up is configurable in the backup registration file). That is, if a data owner is registered to take part in both passive and active backup then the PC host or other client is expected to request active backup data before passive backup data.

When a restore operation is carried out, the data will be restored as passive data and the data owner is responsible for dealing with it when the restore operation is complete. In order to support

this, the PC host or other client is expected to restore passively backed up data before actively backed up data if a data owner has registered for both active and passive backup and restore.

4.3 Owners of private data to be actively backed up

Processes, which own private data that needs to be actively backed up, will have to register with the backup engine as an owner of private data for active backup.

If an application is designed for separate installation (as opposed to upgrade) then it will need to copy a registration file into the private directory during installation. (If an application is part of the platform then this involves creating a registration file as part of the build and ensuring that it is located in the relevant private directory.) The format of the backup registration files can be found in Section 5.

In addition, the process must subscribe to the backup status flag described in Section 3.1 in order to detect when a backup or restore takes place. Processes which own private data that needs to be actively backed up will be started up if required by the secure backup engine in order to do the backup or restore. Therefore, they must check the backup flag on startup rather than making a normal startup (which may involve undesirable access to other servers and files).

Behaviour required of a process that takes part in active backup of private data:

1. When the backup status flag gets set to 'backup in progress' any data that has not yet been saved may need to be saved.
2. The process has to release locks on all public data files that require to be backed up (files that are excluded from backup can be kept locked but assume that all public files will be backed up) so they can be copied off and stop modify private data files. This may require that the process 'freezes' all sessions to external processes to avoid external events that would cause files to be written to (whether requests are blocked or errors returned depends on the context).
3. The process then tries to make a connection to the secure backup engine by name.
4. At the start of backing up private files, the process must ask the secure backup engine whether it is a base backup or an incremental backup using the APIs referenced in Appendix B and, if it is an incremental backup then it must wait until the file snapshot to use for the increment is provided.
5. The process then sends its private data as a stream using the APIs referenced in Appendix B. The owning process is responsible for taking its data files (all or whichever subset it wants to back up) and streaming them. Reference code will be provided to take all files in a private directory and stream them. Any other behaviour will have to be implemented by the data owners.

If the backup is a base backup then all the data in the relevant files should be provided.

If it is an incremental backup then it should only stream data corresponding to the changes since the snapshot. The data-owning process has a choice of how sophisticated its behaviour is in this regard. The simplest behaviour is to always do a base backup (this will be an option, even when an incremental backup has been requested). An alternative is to provide backups of files that have changed since the last backup. The final option is to provide only that data that has changed since the last backup. Which choice is made will depend on the nature of the data to be backed up.

It should be noted that an application may have been installed after the base backup so a process must handle this (no snapshot will be provided).

The data-owning process is responsible for providing a snapshot of files that can be used as a baseline for subsequent incremental backups.

6. The private data being sent may not be received all at once (multiple processes may be trying to supply data at the same time) and may not be received at all (if the backup is partial or has been broken off).

When the backup flag reverts to normal, files can be re-accessed and clients responded to. The process must handle the case where the backup data has only been partially received or has not been received at all.

Behaviour required of a process that responds to an active restore of private data:

1. When the backup status flag gets set to 'restore in progress' the process has to release locks on all public data files that require to be restored (files that are excluded from restore can be kept locked) so they can be overwritten. This may require that the process 'freezes' all sessions to external processes to avoid external events that would cause files to be read or written to (whether requests are blocked or errors returned depends on the context).
2. The process then tries to make a connection to the secure backup engine by name.
3. The process then receives its private data as a stream using the APIs defined in Appendix B. The owning process is responsible for receiving the data as previously provided and recreating the relevant files. Reference code will be provided to take a stream and create files in a private directory. Any other behaviour will have to be implemented by the data owners.

The first chunk of data received will be a base backup of (potentially) multiple files.

If incremental backups have been carried out the base backup data will be followed by a sequence of increments. They will be supplied in the correct (i.e. chronological) order. The receiving process is responsible for amending the data accordingly. If the process can only provide full backups then it will not receive increments.

4. The private data being restored may not be received immediately (multiple processes may be trying to receive data at the same time) and may not be received at all (if the restore is partial or has been broken off).
5. When the backup flag reverts to normal, files can be re-accessed and clients responded to. The process must handle the case where the restore data has only been partially received or has not been received at all. Any cached data must be flushed after a restore.

4.4 Owners of private data to be passively backed up

Processes, which own private data that needs to be passively backed up, will have to register with the backup engine as an owner of private data for passive backup.

If an application is designed for separate installation (as opposed to upgrade) then it will need to copy a registration file into the private directory during installation. (If an application is part of the platform then this involves creating a registration file as part of the build and ensuring that it is located in the relevant private directory.) The format of the backup registration files can be found in Section 5.

In addition, the process must subscribe to the backup status flag described in Section 3.1 in order to detect when a backup or restore takes place.

During the backup or restore, the process can exit or release locks on all public and private data files. If the process is a GUI app then the default behaviour will be for it to be terminated politely so no extra code is required.

4.5 Owners that do not want private data backed up

A process which accesses public data but which does not wish to have private data backed up does not have to register for backup and restore but it does have to respond to the backup status flag:

1. If your process is a UI application which can exit during backup and restore then you need do nothing – you will be terminated politely as before. It should be noted that this means that the default behaviour is for private data of apps not to be backed up. The application needs to make a positive decision to get private data backed up. This ‘opt-in’ behaviour is a deliberate choice for security reasons – apps cannot find their private data backed up without their knowledge.
2. If your process is a non-UI application, or a UI application which will not terminate during backup and restore, which accesses public data then you will need to subscribe to the backup status flag and respond to it:
 - a. When the backup status flag goes to ‘backup in progress’ your process must release all locks or at least make all public data files readable. This may require that the process ‘freeze’ all sessions to external processes to avoid external events that would cause files to be written to (whether requests are blocked or errors returned depends on the context). When the backup flag reverts to normal, sessions can be re-activated and files re-accessed.
 - b. When the backup status flag goes to ‘restore in progress’ your process must change to a state where public files can be overwritten. This will involve releasing all public data files accessed and ensuring that external events cannot cause an attempt to re-access any files. When the restore is complete your process must re-load data files and refresh any data stored in cache.
 - c. Such a process needs to check the backup flag on start-up and, if it is set, should enter the correct state. This should not happen because such a process should not be started up during a backup or restore but the process should attempt to behave robustly just in case. If the process starts up with a backup or restore in progress then it should wait until the backup or restore is complete before attempting to access any public data files.

4.6 Owners with data that resides with a proxy (e.g. the central repository)

Data owners can have data stored within another data owner’s data cage, such as with the central repository. In this scenario, data from each data cage must be backed up. The data owner holding data on behalf of another data owner is known as a proxy. From the PC’s point of view, this data belongs to the data owner, but internally the data is being requested from the proxy via the active client interface. A data owner requiring data from one of these proxies must specify the proxy in the element `<proxy_data_manager />` using its secure ID. See example in Section 5.

Note that upon a backup or restore operation, the data owner will behave as an active data owner as far as the PC is concerned, although this requires no special action on behalf of the data owner itself.

Data stored in the central repository (CentRep) will be backed up if the proxy data manager tag is selected in the backup registration file specifying the central repository’s secure ID. Not all data may be backed up. Each value in CentRep has some metadata associated with it and one bit of the metadata is a backup flag. The value will be backed up only if the backup flag bit is set. It is the responsibility of the data owner to specify the correct settings within the repository initialization file.

CentRep backups are always base, as opposed to incremental, because the CentRep does not contain sufficient information to efficiently support incremental backups and because CentRep data is not expected to be sufficiently large to justify incremental backup.

4.7 Owners of DBMS files to be backed up

A data owner may be responsible for a number of databases that may need to be subject to back up and restore. Shared databases will be backed up if the policy number is specified in the data owners backup registration file using the `<dbms_backup>` tag. There is no need to specify database names as the DBMS will provide the list of the databases belonging to the data owner based on the policy identifier. See example in Section 5.

4.8 Owners of system data (installed applications) to be backed up

On platform security-enabled Symbian OS phones, installed applications are no longer backed up by default and now require the consent of the application designer. The backup and restore of an application's binaries and its data is also done separately and requested differently. For an applications data please refer to the sections above on active and passive backup, for binaries the `<system_backup/>` tag should be specified in the backup registration file. See Section 5 for more details.

The `<system_backup/>` tag triggers the secure backup engine to query which package the application belongs to and retrieves a list of all the files referenced by the package. Only the following read-only files will be backed up as system data (any file or directory not listed below will need to be backed up separately using one of the other mechanisms described previously):

- `\sys\bin*`
- `\resource*`
- `\private\123456\backup_registration*.xml`

Writeable files will not be backed up as system data as they will fail to be restored as system data.

Note: It is essential to add the backup registration file(s) to the package. If forgotten, nothing will be backed up or restored.

An additional mechanism exists to specify backup registration files for data owners that do not have private directories - this is to be used for DLLs which are installed as part of a package. To use this method, the backup registration file must be placed in a special location under the Secure Backup Engine's private directory:

```
\private\10202D56\import\packages\<package-id>\backup_registration.xml
```

(10202D56 is the SID of the Secure Backup Engine, and `<package-id>` should be replaced with the ID of the package containing the DLL)

Note: As the data owner does not have a private directory, the only methods that are valid in this backup registration file are system and public file backup.

5 Backup registration file format

5.1 Content of the backup registration file

A backup registration file includes the following information:

1. For passive data backup operations, a list of private directories and files that should be backed up. It is possible to list a directory for backup and then list sub-directories or files for exclusion. These directories and files are defined relative to the process private area so it is not possible to refer to private files owned by another process (a value of a single backslash denotes the whole of the private directory). If any files or directories do not exist then no error will be raised – i.e. it is acceptable to list directories which do not exist yet. If a data owner's data is such that incremental data will be inefficient then it is possible to specify that only base backups should be done (for example, if a data owner stores all of its data in one database file then any increment will always be the whole database file and it is more efficient to always do a base backup or a restore will involve transferring large amounts of redundant data).
2. A list of public files and directories to be backed up as part of a partial backup. The same syntax is used as for private files but with regard to the public part of the filing system. Any system files or private files will be ignored from this section (because they are not public files). As with files listed for passive backup, directories can be listed that do not yet exist.
3. For processes that require backup of system files (executables and resource files), a statement of the fact – it is not necessary to list the system files to be backed up as the list is available from package data. It should be noted that a package containing multiple data owners should have all its data owners backed up – if any of its data owners specify that system files are to be backed up then all the system files for the package will be backed up and restored as a unit.
4. For passive backup and partial backup of public files, should directories be cleared before a restore operation? I.e. should extra files be deleted?
5. Is active backup required? If so which process should be started or which server contacted. It is possible to invoke active backup and also request private and / or public files to be backed up. There is no guaranteed order of backup or restore (i.e. there is no guarantee of which data owner's data will be restored first) between SIDs although it is guaranteed that active data will be requested before passive on backup and active supplied before passive on restore.
6. Can this private data be backed up or restored selectively for a partial backup or restore? This is not necessarily only possible for a process that implements active backup (but a passive data owner will need to not keep files locked). This option is intended for the future where it may be possible to avoid terminating most processes during a backup or restore. At present it can be ignored as partial backup or restore will be achieved by using the normal backup or restore mode but only transferring data belonging to selected data owners.
7. Does this process require a reboot after a restore operation?
8. Is a shared database to be passively backed up? If so, then the security policy ID of the database to be backed up or restored must be specified.
9. Does the data owner have data stored in a proxy? If so, the secure ID of the proxy must be specified in the proxy data manager tag.
10. Does this process require a significant time to prepare for backup or restore?

The backup registration files will be stored in the root of the private directory of the corresponding process with a standard name (`backup_registration.xml`). By placing the backup registration files in the private data areas they are protected on the phone.

When a specific drive is backed up, the backup registration file is searched for on the drive being backed up. If no backup registration file is found then it is searched for in the corresponding private directory of the Z: drive. This is purely an administrative convenience to avoid processes having to copy backup registration files when it would not otherwise be necessary. If a backup registration file is found in the drive to be backed up then the Z: drive will not be searched – this allows the backup registration file on the Z: drive to be masked where necessary. One implication of this use of backup registration files from the Z: drive is that the backup registration file may refer to files (including private data files, public data files, databases, repositories and system files) that may not exist on a drive to be backed up. This is handled robustly by the backup process.

5.2 Examples of backup registration files

The following is an example of a backup registration file for a data owner that requires only passive backup of files in a sub-directory named 'preserve':

```
<?xml version="1.0" standalone="yes"?>
<backup_registration>
  <passive_backup>
    <include_directory name = "preserve"/>
  </passive_backup>
  <restore_requires_reboot = "no"/>
</backup_registration>
```

The following is an example of a backup registration file for a data owner that requires passive backup of all its files and also wants its system files to be backed up:

```
<?xml version="1.0" standalone="yes"?>
<backup_registration>
  <passive_backup>
    <include_directory name = "" />
  </passive_backup>
  <system_backup/>
  <restore_requires_reboot = "no"/>
</backup_registration>
```

The following is an example of a backup registration file for a data owner that has only central repository data:

```
<?xml version="1.0" standalone="yes"?>
<backup_registration>
  <proxy_data_manager SID="0x10202BE9" />
</backup_registration>
```

The following is an example of backup registration file for a data owner that requires an active backup only:

```
<?xml version="1.0" encoding="UTF-16" standalone="yes" ?>
<backup_registration version="1.0">
  <active_backup process_name="processname.exe"
    requires_delay_to_prepare_data="yes"
    active_type="activeonly" />
</backup_registration>
```

The following is an example of backup registration file for data owner which implements a proxy:

```
<?xml version="1.0" encoding="UTF-16" standalone="yes" ?>
<backup_registration version="1.0">
```

```

    <active_backup process_name="processname.exe"
                active_type="proxyonly" />
</backup_registration>

```

The following is an example of backup registration file for a data owner who wishes to back up its DBMS databases:

```

<?xml version="1.0" standalone="yes" ?>
<backup_registration>
  <dbms_backup_policy="AABBCCDD" />
</backup_registration>

```

5.3 Additional backup registration files

The backup registration file format allows the specification of directories for private data rather than having to specify all files individually. This means that some types of expansion (e.g., extra files or directories under a specified directory for private or public files) can be handled without changing backup registration files. However, the same technique may not handle additional system files (such as additional plug-in executables).

Therefore, additional backup registration files can be present and will be parsed. Additional backup registration files must have a name of the form `backup_registration*.xml` and be located in the root of the relevant private directory. Additional backup registration files can be distinguished from the principal backup registration file by the file names (the principal backup registration file has the name `backup_registration.xml`).

Additional backup registration files must only contain a subset of the normal backup registration file elements. They are intended to be purely additive of passive private, public and system files. They may not include an additional type of backup (i.e. they may not specify active backup or restore if it was not included in the principal backup registration file).

The effect of additional backup registration files is the same as if the passive backup, public and system file elements were included in the principal backup registration file.

All drives will be searched for additional backup registration files including the Z: drive. Additional backup registration files are always used in a backup and can not be masked.

Please note that additional registration files cannot be created or modified at runtime.

6 Testing backup and restore

The behaviour of any software should be tested and the behaviour of software with respect to backup and restore is no exception. Symbian tests the secure backup engine and the associated framework as part of testing Symbian OS and developers are responsible for testing their own components.

If a process is implementing active backup and restore then this should be tested. As a minimum, a simple sequence of base and incremental backups followed by a restore to a clean device, followed by smoke tests will demonstrate that the main functions of backup and restore work. Robustness tests should involve interrupting a backup or restore operation by disconnecting the phone from the PC and checking that the data owning process handles the interruption without corruption.

The developers of a component, subsystem or other set of software are responsible for testing to ensure that their private data is backed up and restored correctly. Here are some suggested tests for inclusion in a test plan:

- Run the software under test and create some data. Back it up to a PC, reformat the phone or obtain a new device and restore the data from the PC. Then run smoke tests to ensure that the expected data is present and that no expected data is missing
- Run the software under test and create some data. Back it up to a PC and then run the software again to edit the data or create more data. Restore the data from the PC (without reformatting the phone). Then run smoke tests to ensure that that data present is that which was backed up, not the later data and that the data is consistent.
- Run the software under test and create some data. Run a backup while the software is running to test that the software makes data available and does not interfere with the backup. If possible, run the software during the backup and attempt to create or edit data. This should not interfere with the backup. Run the same tests during a restore operation to test that the software does not interfere with the restore. Pay particular attention to what happens to data entered or edited during a backup or restore operation – is it lost or retained?

More thorough tests should include verifying the active restore client's handling of unexpected data versions, handling of out-of-sequence increments during restore and handling of incomplete or corrupt restore data but these will require specialized test software to be developed as the Symbian OS backup and restore framework tries to prevent these problems.

- After a restore, it is essential that the accuracy and completeness of the restored data is verified. This applies to passive backup data owners as much as active backup data owners. The Symbian OS backup and restore framework can only back up and restore the data that is specified or provided. If certain private files or directories are omitted then the Symbian OS backup framework will function correctly but the restored data will not be as expected.

Appendix A Backup registration File DTD

The DTD for the backup registration file is as follows:

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE backup_registration [
<!ELEMENT backup_registration
(passive_backup?, system_backup?, public_backup?, active_backup?,
cenrep_backup*, proxy_data_manager*, dbms_backup*, restore?) >
<!ATTLIST backup_registration
version CDATA #FIXED "1.0"
>
<!-- Include file or directory name should include path relative to private
directory for private data. To include all private data use "\".
Include file or directory name should include path relative to root of
drive for public data. This may or may not include the drive letter.
Inclusions with drive letter will be excluded from backups of other
drives. Inclusions without drive letter will be assumed to apply to all
drives.
-->
<!ELEMENT include_file EMPTY >
<!ATTLIST include_file
name CDATA #REQUIRED
>
<!-- Exclude file or directory name should be of the same form as the owning
<include_directory> and is not treated as relative to the
<include_directory>
-->
<!ELEMENT exclude EMPTY >
<!ATTLIST exclude
name CDATA #REQUIRED
>
<!ELEMENT include_directory (exclude*) >
<!ATTLIST include_directory
name CDATA #REQUIRED
>
<!ELEMENT passive_backup (include_directory|include_file)* >
<!ATTLIST passive_backup
supports_selective (yes|no) "no"
delete_before_restore (yes|no) "no"
base_backup_only (yes|no) "no"
>
<!ELEMENT public_backup (include_directory|include_file)* >
<!ATTLIST public_backup
delete_before_restore (yes|no) "no"
>
<!-- If the <system_backup> element is present then all executables
and resource files that were included in SIS files will be backed up.
The set of files does not need to be specified as it can be found from
package data.
They can only be restored after checking against signed hashes.
-->
<!ELEMENT system_backup >
<!-- <proxy_data_manager> indicates that the data owner represented by this
registration file has data that is accessed through a proxy which is
itself an active data owner.
-->
<!ELEMENT proxy_data_manager EMPTY >
<!ATTLIST proxy_data_manager
SID CDATA #REQUIRED
>
<!ELEMENT dbms_backup EMPTY >
<!ATTLIST dbms_backup
policy CDATA #REQUIRED
>
<!ELEMENT active_backup EMPTY >
```

```

<! ATTLIST active_backup
  process_name CDATA #REQUIRED
  requires_delay_to_prepare_data (yes|no) "no"
  supports_selective (yes|no) "no"
  supports_incremental (yes|no) "yes"
  active_type (activeonly|activeandproxy|proxyonly) "activeonly"
>
<! ELEMENT restore EMPTY >
  <! ATTLIST restore
    requires_reboot (yes|no) "no"
  >
]>

```

Appendix B Active Backup Client API

The active backup client class declarations can be found in the file `epoc32\include\connect\abclient.h` which requires `epoc32\include\connect\sbdefs.h`.

[Back to Developer Library](#)

Want to be kept informed of new articles being made available on the Symbian Developer Network?

[Subscribe to the Symbian Community Newsletter.](#)

The Symbian Community Newsletter brings you, every month, the latest news and resources for Symbian OS.