Integration with Time Machine

Jim Dovey
aka Alan Quatermain of AwkwardTV

http://alanquatermain.net/

What we'll cover

- Who will use this?
- Concepts and design
  - UI elements, events
- Public API (a whole *two* functions!)
- Private API
- An example Cocoa controller class

What we won’t cover

- Snapshots
  - Used by Xcode ?
  - Useful for managing non-bundled collections of discrete files
- Triggering backups programatically
  - BUBackUpNow( ) function

Who will use this?

- Applications managing collections of data
  - Address Book, Mail, iPhoto
  - iTunes, iCal
  - Library, Ledgers, CRM
- Apps with a desire to handle partial dataset restorations
  - CoreData

Concepts and Design

Time Machine User Interface

- One large fullscreen window
- A collection of images
  - Time Machine ‘windows’ aren’t (necessarily) actual windows
  - Each instance is an image, usually taken from a simple window via CGContextXXX() functions.
- Time Machine scrolls through these windows for you
- Your app is alerted when a real window is required, and your app handles display & input for that window.

Events and Callbacks

Time Machine handles the interface for you— you only have to provide some callback routines.

```
BURegisterStartTimeMachineFromDock(...);
BURegisterRequestSnapshotImage(...);
BURegisterTimeMachineDismissed(...);
BURegisterTimeMachineRestore(...);
```

The ‘events’ posted by Time Machine include the startup request, actions, dismissal (cancel), restore (one or all), activate/deactivate snapshot windows, and requests for snapshot or thumbnail images.

API

Public API

Apple has released two functions:

```
CSBackupIsItemExcluded(CFURLRef item, Boolean * byPath);
CSBackupSetItemExcluded(CFURLRef item, Boolean exclude,
Boolean byPath);
```

These routines allow you to inform the backup system of cache files or other oft-changed data which need not be backed up.

Anything further than this requires that we resort to accessing the private API…

Private API

- Request notification of Time Machine invocation
- Provide callbacks for the Time Machine engine, then start Time Machine itself
  - If in a non-applicable state, don’t start time machine
  - Modal loops, active document is untitled/unsaved
- Answer callbacks to provide snapshot window images corresponding to backup data
- Handle activation and deactivation of individual snapshots
- Restore if so requested, or else revert to prior state upon dismissal.

Startup

- When your app starts, call BURegisterStartTimeMachineFromDock();
    - Your callback returns nothing and takes no arguments.
- The callback will fire when the user clicks the Time Machine icon in the dock. It's still up to you to launch the Time Machine UI, however.

```
typedef void (*BUStartTimeMachineCallBack)(void);
void BURegisterStartTimeMachineFromDock(BUStartTimeMachineCallBack
         cb);
void BUStartTimeMachine(int windowNumber, CFURLRef urlForWindow,
         BUAction flags);
```

Data Callbacks

- Upon receiving the startup call, you register your other callbacks, to provide data and handle events
- Time Machine provides request callbacks for window snapshots and for thumbnail images, but we'll just use snapshots.
- To generate a snapshot image, create a window for the data at the given URL, and call BUUpdateSnapshotImage(), providing the CG window number (using -[NSWindow windowNumber]) and the provided URL as parameters.

```
typedef void (*BURequestSnapshotImageCallBack)(void * token,
        CFURLRef backupURL);
void BURegisterRequestSnapshotImage(void * token,
        BURequestSnapshotCallBack callback);
void BUUpdateSnapshotImage(int windowNumber, CFURLRef url);
```

Snapshot Events

- You must provide callbacks to be notified when snapshots are focussed or blurred.
- When these callbacks are called, the application must display or remove a window at the given coordinates.
- When done processing, call BUActivatedSnapshot() or BUDeactivatedSnapshot() as appropriate.

```
typedef void (*BUActivateSnapshotCallBack)(void * token, CFURLRef
         backupURL, CGRect workingBounds);
typedef void (*BUDeactivateSnapshotCallBack)(void * token, CFURLRef
         backupURL);
void BURegisterActivateSnapshot(void * token,
         BUActivateSnapshotCallBack callback);
void BURegisterDeactivateSnapshot(void * token,
         BUDeactivateSnapshotCallBack callback);
void BUActivatedSnapshot(int windowNumber, CFURLRef url);
void BUDeactivatedSnapshot(int windowNumber, CFURLRef url);
```

Action Callbacks

- Two main actions: restore and dismiss
- Restore provides a flag to indicate whether to restore all items or just a selection.
- Dismissal only triggers *after* the Time Machine UI has gone away.
- To programatically dismiss, call `BUTimeMachineAction(1);`

```
typedef void (*BUTimeMachineDismissedCallBack)(void * token);
typedef void (*BUTimeMachineRestoreCallBack)(void * token, CFURLRef
        backupURL, CFURLRef liveURL, Boolean restoreAll,
        CFDictionaryRef userInfo);
void BURegisterTimeMachineDismissed(void * token,
        BUTimeMachineDismissedCallBack callback);
void BURegisterTimeMachineRestore(void * token,
        BUTimeMachineRestoreCallBack calback);
void BUTimeMachineAction(BUAction action);
```

Cocoa Controller

AQTimeMachineController

- Implemented in Objective-C 2.0
- Singleton class
- Designed to handle most of the work for you
  - You shouldn’t need to call BUxxxx() methods yourself
- You implement a delegate to provide application-specific data
- Ideally this delegate should be concerned only with Time Machine, and should be your *only* Time Machine-handling class

Properties

- @property(assign) id<AQTimeMachineDelegate> __weak delegate;
    - Synchronized access, non-retaining
- @property NSRect workingBounds;
    - The current snapshot bounds set by Time Machine
- @property BOOL changedItemsOnly;
    - YES if the UI should only show changed items
- @property BOOL inTimeMachine;
    - Check to see if Time Machine actions should be performed

General Functions

- + (AQTimeMachineController *) timeMachineController;
  - Fetch the singleton instance
- - (BOOL) canEnterTimeMachine;
  - A simple check, will call the delegate
- - (IBAction) browseBackups: (id) sender;
  - When you want your own Time Machine button
- - (void) dismissTimeMachine;
  - Close down the Time Machine UI
- - (void) invalidateSnapshotImages;
  - When your UI has changed, updates snapshots

Controller Tasks

- Handles Time Machine startup notifications
  - *Requires a delegate to be set prior to this*
- Stores the window state of the initial window, and restores this state when Time Machine is dismissed
  - Miniaturized, visible
- Maintains a list of window controller to URL mappings, one for each snapshot window
- Handles updates to snapshot images
- Activates and deactivates snapshots, notifying delegate
- Calls delegate when a restore action is requested

AQTimeMachineController Code

Delegate Tasks

- Determines whether the app can enter Time Machine
- Creates and returns controllers and data paths for the live window and any snapshot windows requested
- Implements data restoration
- Optionally:
  - Performs setup before & after entering Time Machine
  - Performs actions before & after snapshot activation/ deactivation
  - Makes any changes required for ‘show changed items only’
  - Any app-specific cleanup when Time Machine is dismissed

An NSDocument-based Delegate

Useful Data

- Keep a record of all snapshot NSDocuments, indexed by path or URL
- Keep track of the current document
- Store any document user-interface state which is likely to change while in Time Machine
    - Search box contents, list selections
- Ensure that no documents are editable while in Time Machine

-canEnterTimeMachine

- Check for modal panels:
  - [[[NSRunLoop mainRunLoop] currentMode] isEqualToString: NSModalPanelRunLoopMode]
- Check for an open & stored current document:
  - [[NSDocumentController sharedDocumentController] currentDocument]
- Document must have window controllers
- No sheet should be attached:
  - [[ctrl window] attachedSheet]

Snapshot window controllers

- You can create NSDocuments for backup snapshots, but it’s a good idea to limit them a little
  - Create using -[NSDocumentController makeDocumentWithContentsOfURL:ofType:error:]
  - Use -makeWindowControllers to setup the controllers, rather than letting NSDocument put itself onscreen

Updating snapshots

- Implement the optional notification handlers to store and set data at appropriate times
- Store UI state:
  - Before Time Machine activates
  - When deactivating snapshots
- Set UI state:
  - When activating snapshots
  - When restoring or dismissing Time Machine
- Also install your own handlers to invalidate & update snapshots in response to user activity
  - Notifications, delegates, KVO

Example Delegate Code

…now, only the future awaits

For more information and updates to this material, visit my website:
http://alanquatermain.net/