

Tutorial n. 001
(versione 1.3 / luglio 2010)

Applescript Studio e Sparkle

**Come inserire un sistema di aggiornamento
nei progetti XCode**

Idea tratta dal sito
<http://foolsworkshop.com/>

InerziaSoft

<http://www.inerziasoft.eu>

Indice

Storico revisioni.....	3
Licenza e altro	3
Introduzione.....	3
Sparkle.....	4
Strumenti.....	4
Procedimento	5
<i>Inserimento del framework.....</i>	<i>5</i>
<i>Nuova fase di build.....</i>	<i>6</i>
<i>File Info.plist.....</i>	<i>8</i>
<i>Menu Check for updates.....</i>	<i>9</i>
<i>Preferenze.....</i>	<i>13</i>
<i>Firma degli update.....</i>	<i>15</i>
<i>File AppCast.....</i>	<i>18</i>
Come proseguire.....	19
Conclusione.....	19
Nota.....	19
Appendice A: uso del Portachiavi.....	20
Appendice B: automatizzare XCode per Sparkle.....	23
<i>Target "Distribution".....</i>	<i>23</i>

Storico revisioni

rev	data	sezioni
1.0	ottobre 2008	tutte
1.2	gennaio 2009	Firma degli update: precisazioni e screenshot Appendice A: salvare i certificati nel Portachiavi Appendice B: automatizzazione di XCode
1.3	luglio 2010	nota nel caso di progetti con Garbage Collection modifica nello script per Snow Leopard (Appendice B)

Licenza e altro

Questo tutorial deve essere considerato come freeware: potete cioè scaricarlo ed utilizzarlo senza che nulla mi sia dovuto. Siete incoraggiati a distribuire questo documento in formato pdf e potete inoltre inserirlo in altri download, sempre che non venga fatto pagare alcunché all'utente finale. In cambio, chiedo di lasciare inalterati i riferimenti al nostro sito (inerziasoft.eu) e di citare la fonte.

Il contenuto è stato rivisto e molti errori sono stati eliminati; non posso però affermare che non ce ne siano altri, per cui non posso essere ritenuto responsabile di eventuali perdite di dati conseguenti alle azioni indicate in questo tutorial. Posso solo dire che tutta la sequenza di azioni è stata da me effettuata ed è documentata dalle figure prese direttamente dal vivo.

Se ritenete che questo tutorial possa avere un certo valore e non sapete come sdebitarvi... la soluzione è semplice: fate una piccola donazione, usando il link che trovate sul sito www.inerziasoft.eu alla sezione download!

Se avete trovato un errore, o per altre segnalazioni, vi prego di contattarci all'indirizzo info@inerziasoft.eu

Introduzione

Situazione: avete terminato la vostra applicazione e, dopo un breve periodo di test (sempre *troppo* breve, in verità!) ne avete immediatamente fatto l'upload sui siti di distribuzione del software oppure direttamente sul vostro sito.

Orrore! Dopo qualche giorno vi accorgete (o peggio, se ne accorge un utilizzatore del vostro software) di un sottile baco che pregiudica il funzionamento! Vi mettete subito all'opera e in breve (siete infatti molto bravi!) avete ottenuto un buon fix.

Ora potete subito fare un nuovo upload ma... e tutti quegli utenti che hanno già scaricato il software? Qualcuno non avrà ancora incontrato il malfunzionamento, ma altri sì e... invece di lamentarsi non hanno fatto altro che fare un drag & drop verso il cestino e nei loro ricordi la vostra applicazione resterà per un bel po' come "quella che era bacata"! Che bel danno!

Come sarebbe bello se poteste conoscere almeno l'email di tutti e dir loro di scaricare la nuova versione! Ora, con tutto lo spam che impera, non potreste certo chiedere una mail prima del download, specie se state proponendo un freeware: comunque, spesso ve ne verrebbe fornita una tenuta apposta per questi usi! Quanto sarebbe meglio che fosse la stessa applicazione a *chiamare casa* per verificare se esiste una nuova versione!

Bene: da un po' di tempo questo è possibile, in modo anche facile, semplicemente sfruttando un framework fornito gratuitamente!

Sparkle



Si trova sul sito della Sparkle [<http://sparkle.andymatuschak.org/>].

È un framework fatto per Cocoa ma, grazie alla potenza di XCode, l'ambiente di sviluppo fornito gratuitamente da Apple, può essere integrato anche in un progetto Applescript Studio. Cosa interessante... è gratis! Si tratta infatti di un progetto open-source, portato avanti da uno studente; se trovate, come penso, che il progetto sia veramente utile, potete fare una donazione, cosa che consiglio, utilizzando il link nella home page.

Probabilmente lo avrete già incontrato, anche se non sapevate di cosa si trattava; infatti, l'integrazione con il software ospite è veramente perfetta: durante il lancio di un'applicazione, se siete collegati ad internet, si apre una finestra che vi avverte dell'esistenza di una nuova versione, chiedendo se volete scaricarla. Se rispondete affermativamente, non si apre Safari sul sito, bensì inizia automaticamente il download, al termine del quale il file viene scompattato, l'applicazione attuale viene chiusa e spostata nel cestino e viene rilanciata la nuova versione. Aggiungete il fatto che se non siete collegati alla rete, il software rimanda il controllo, senza dare alcun errore.

È anche inoltre possibile inserire nelle preferenze la possibilità di gestire il menu che abilita o meno l'aggiornamento automatico. Tutto questo meccanismo è messo a disposizione dal framework di Sparkle.

Nel Basic Setup della documentazione sul sito Sparkle ci sono le istruzioni passo passo per ottenere tutto ciò. Tuttavia, è specializzato per chi lavora in Cocoa, con pochi esempi visivi, mentre le istruzioni per l'uso con Applescript Studio non sono immediate da trovare. Ecco il perché di questo tutorial, per il quale ho preso spunto dal sito [The AppleScript Studio Workshop > Adding a Check Updates Feature](#), purtroppo in inglese.

Da sempre sostengo che un programmatore dovrebbe conoscere a sufficienza l'inglese, ma spesso ci si ferma perché qualche sfumatura ci sfugge. Lo scopo del tutorial è quindi di mettere a disposizione di tutti questo utile strumento.

Per maggiori informazioni, potete poi consultare direttamente la documentazione sul sito, a cui si può accedere velocemente usando il link contenuto nel dmg.

Strumenti

Useremo i seguenti strumenti:

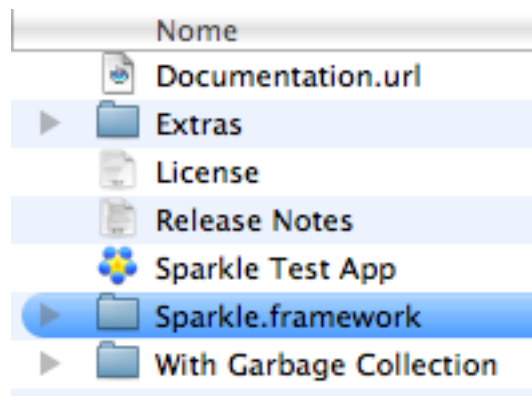
- XCode 3.1 su Leopard (Mac OS X.5.x) oppure XCode 3.2 su Snow Leopard;

- Interface Builder 3.1;
- una versione recente di Sparkle, che al momento significa 1.5b6;
- un sito web in cui posizionare un file di tipo RSS, che verrà usato per controllare l'esistenza di una versione più recente e per contenere quest'ultima.

Ovviamente, prima di partire scaricate Sparkle dal [sito](#)!

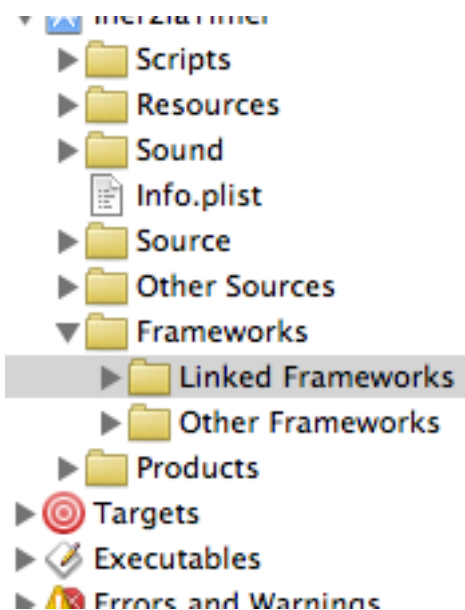
Procedimento

Apriamo il dmg appena scaricato, il cui contenuto è rappresentato in figura:



Il link alla documentazione rimanda al sito e vediamo anche un'applicazione di prova, ma quello a cui siamo interessati è la cartella *Sparkle.framework*.

Apriamo un progetto Applescript Studio dentro XCode (come esempio, useremo un progetto *InerziaSoft*, abbiamo scelto *InerziaTimer*) e cominciamo a dare un'occhiata alla parte laterale della finestra, dove si svolgeranno la maggior parte delle azioni (la promessa che abbiamo fatto è infatti di non scrivere righe di codice):

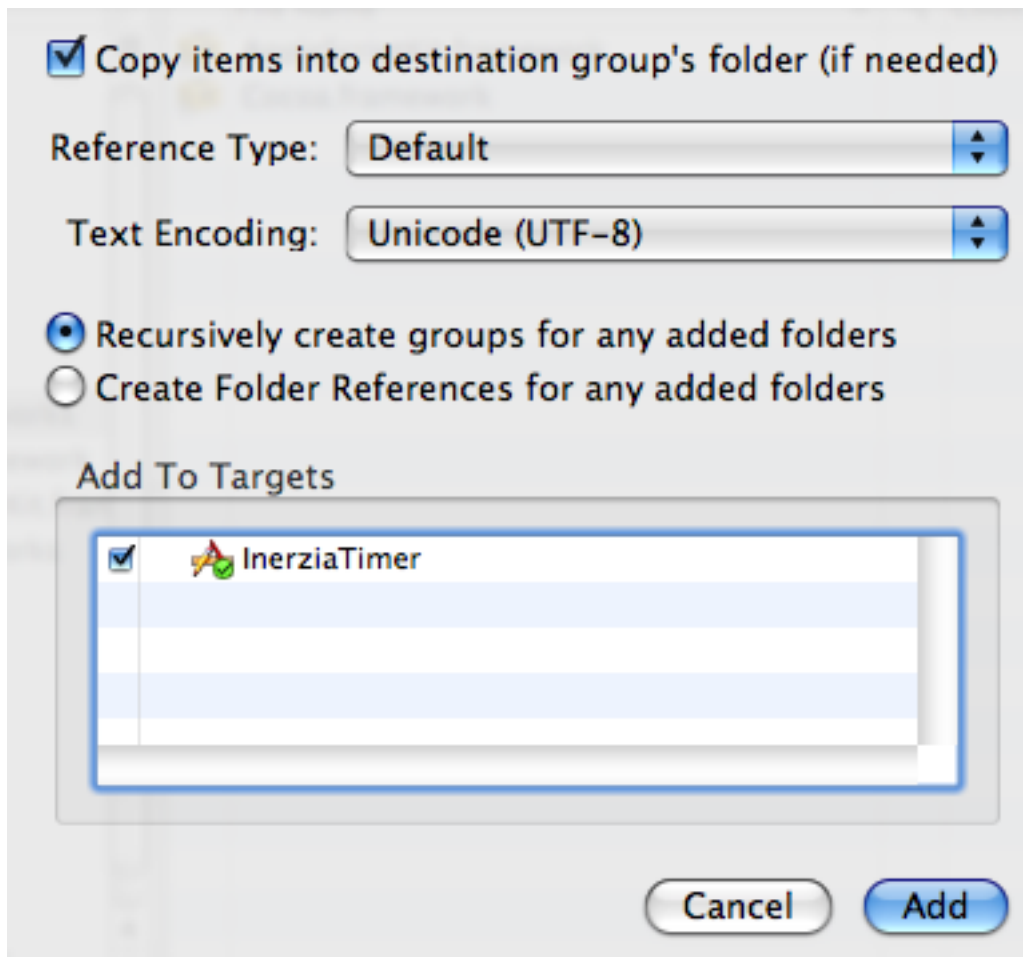


Inserimento del framework

Verso metà lista, troviamo la cartella *Frameworks* all'interno della quale se ne trova un'altra dal nome *Linked Frameworks*.

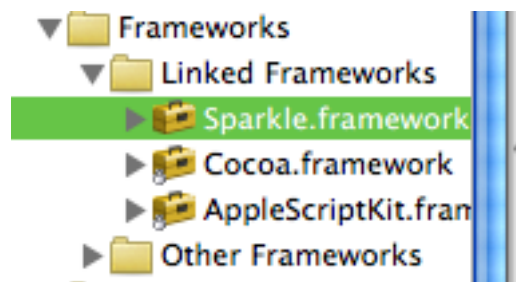
Mettete una finestra del Finder vicina a quella di XCode e fate un bel drag&drop dello *Sparkle.framework* (dal dmg) sulla cartella *Linked Framework* del progetto XCode. Se stiamo lavorando su un progetto XCode *Garbage Collected*, dovremo usare il framework che si trova dentro la cartella *With Garbage Collection*.

Non appena rilasciamo il mouse, compare un dialogo:



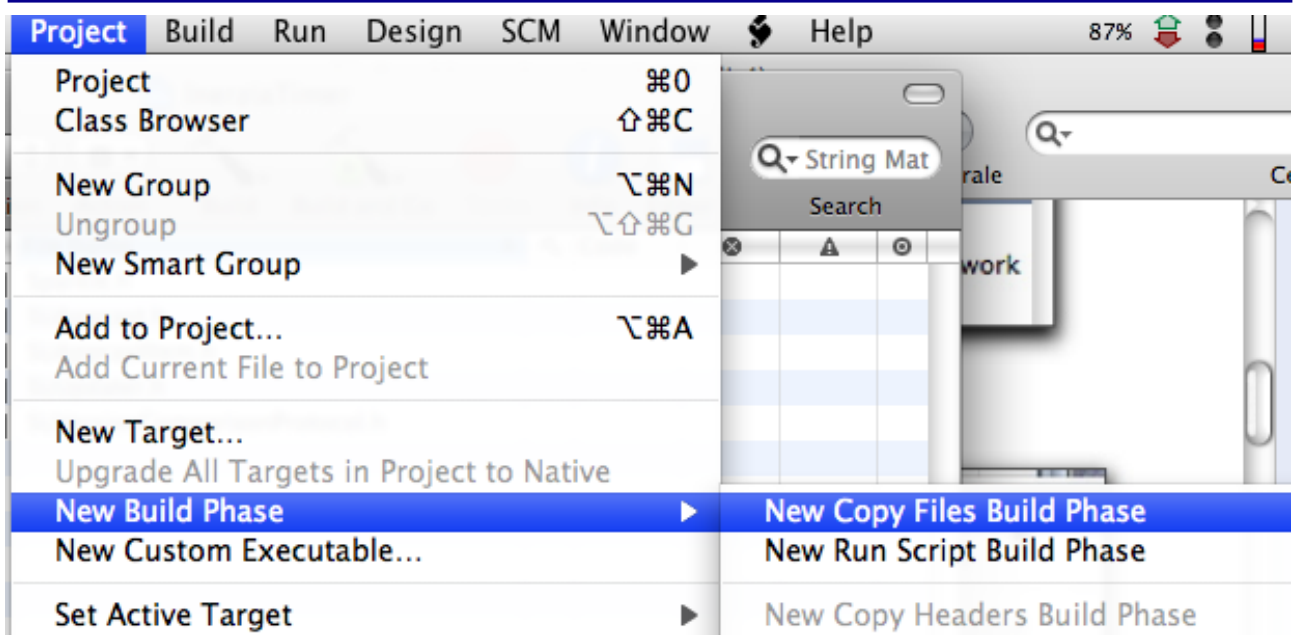
nel quale mettiamo un check su *Copy Items into...* e click su *Add*.

Come risultato ci troviamo ad avere il framework nel progetto:

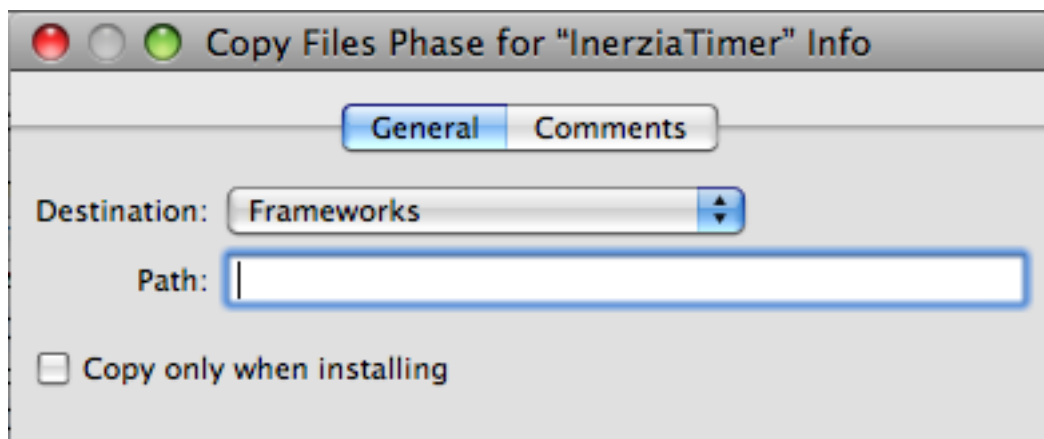


Nuova fase di build

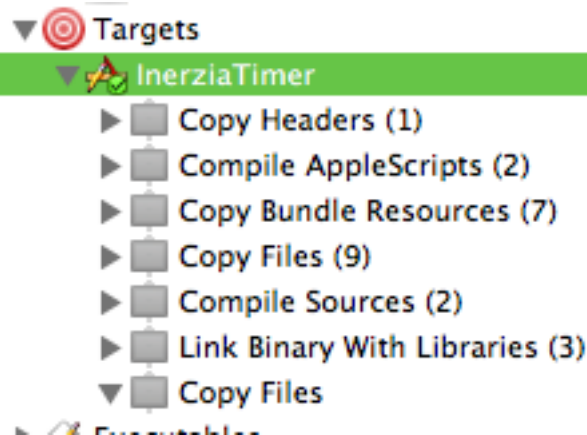
Ora dobbiamo fare in modo che il nuovo framework venga considerato nella compilazione, per cui dal menu Project andiamo su New Build Phase, da cui scegliamo il sotto-menu New Copy Files Build Phase:



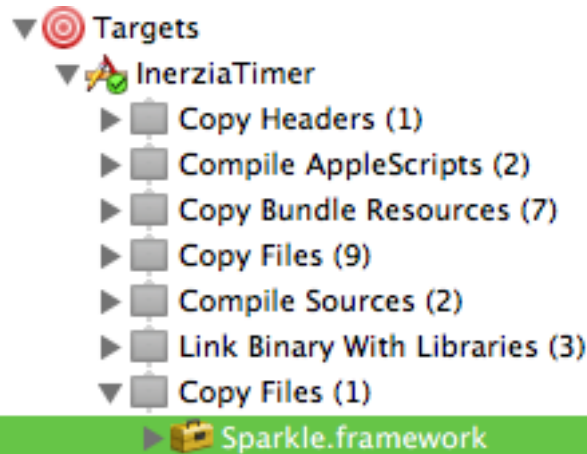
Una volta scelto il sotto-menu, compare una finestra:



in cui come Destination sceglieremo *Frameworks* e poi chiuderemo con un click sul pallino rosso di chiusura. Se ora andiamo ad espandere la parte Target, vedremo alla fine della lista una nuova voce "Copy Files", al momento vuota:



Allora prendiamo col mouse lo Sparkle.framework dentro la cartella *Linked Frameworks*, dove l'abbiamo messo prima e con un drag&drop portiamolo sul *Copy Files* appena creato ed il risultato comparirà come nella figura seguente:



Finora non abbiamo ancora scritto una riga di codice!

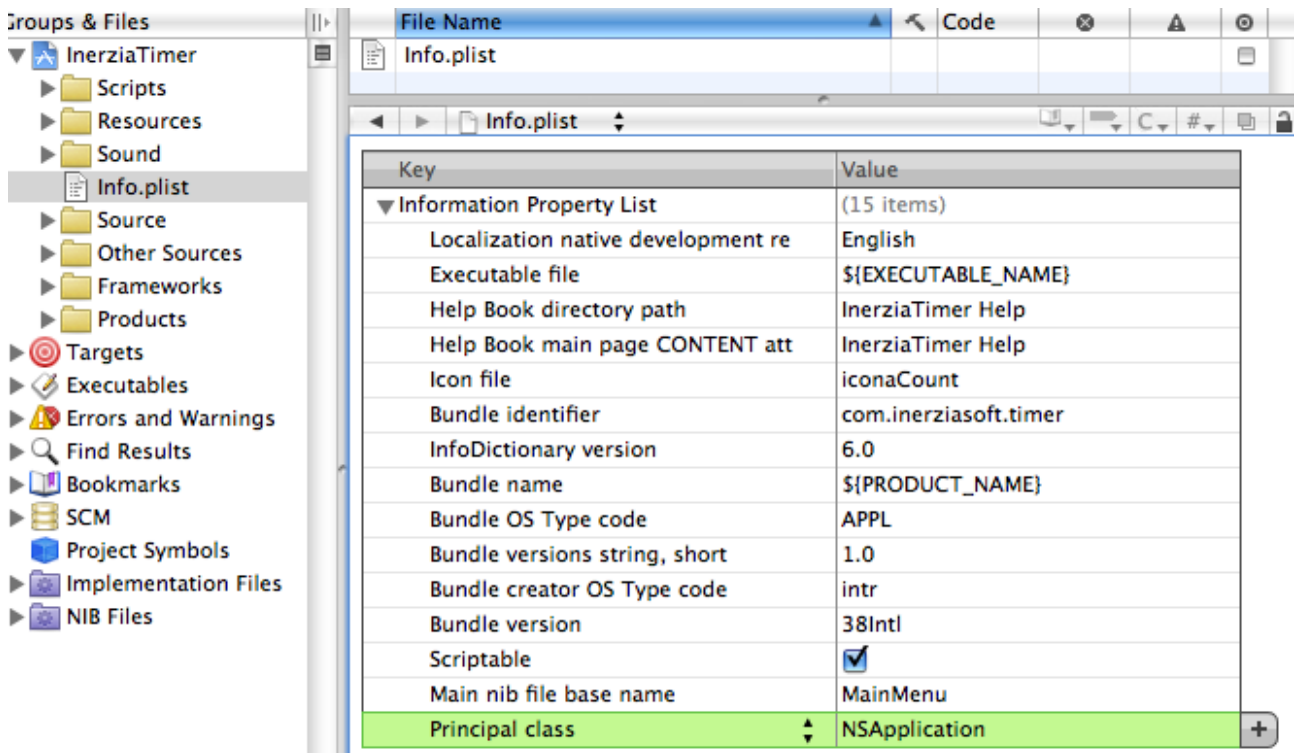
File Info.plist

Ora però dobbiamo creare un link al file XML che andremo a posizionare sul nostro sito web e che conterrà le informazioni sull'aggiornamento. Per fare ciò, dobbiamo modificare il file `Info.plist` contenuto nel progetto, aggiungendo una nuova chiave:

```
<key>SUFeedURL</key>
```

```
<string>http://miosito.com/nomeApplicazione.xml</string>
```

dove il valore (una stringa) della chiave è l'indirizzo di dove verrà salvato il file sul nostro sito. L'indirizzo deve essere assoluto; sarà il framework di Sparkle a farne buon uso. Per effettuare l'aggiunta, facciamo click su `Info.plist` e si aprirà l'editor corrispondente:



Se clicchiamo su un qualunque campo, comparirà un segno “+” sulla destra: se lo clicchiamo, verrà aggiunto un nuovo record al file plist, Notiamo che quando andremo a cliccare sul nome del campo, comparirà un menu a tendina; ma noi dobbiamo scrivere “SUFeedURL”, che nel menu non esiste, per cui scriviamo senza preoccuparci. Passiamo all’altro campo e inseriamo l’URL del file xml; alla fine, un click al di fuori termina l’ingresso.

Principal class	NSApplication
SUFeedURL	http://miosito.com/nomeApplicazione.xml

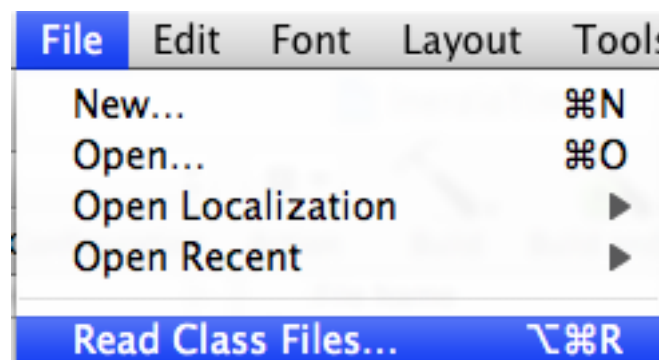
Nel nostro Info.plist potrebbero essere aggiunte altre chiavi relative a Sparkle; vediamo solo le più importanti:

- `SUEnableAutomaticChecks`: se messo a *true* abilita il meccanismo dei check automatici (può essere cambiato solo se inseriamo una preferenza utente, come vedremo in seguito).
- `SUShowReleaseNotes`: se *false* non mostra le note di rilascio (se la chiave non c’è, le note sono sempre mostrate... sempre che ci siano!). Da notare che dalla versione 1.5b6, le note non vengono mostrate se l’indirizzo è del tipo file://, per evitare possibili letture indesiderate sul proprio hard disk.
- `SUPublicDSAKeyFile`: il nome della chiave pubblica DSA (vedremo in seguito il significato).

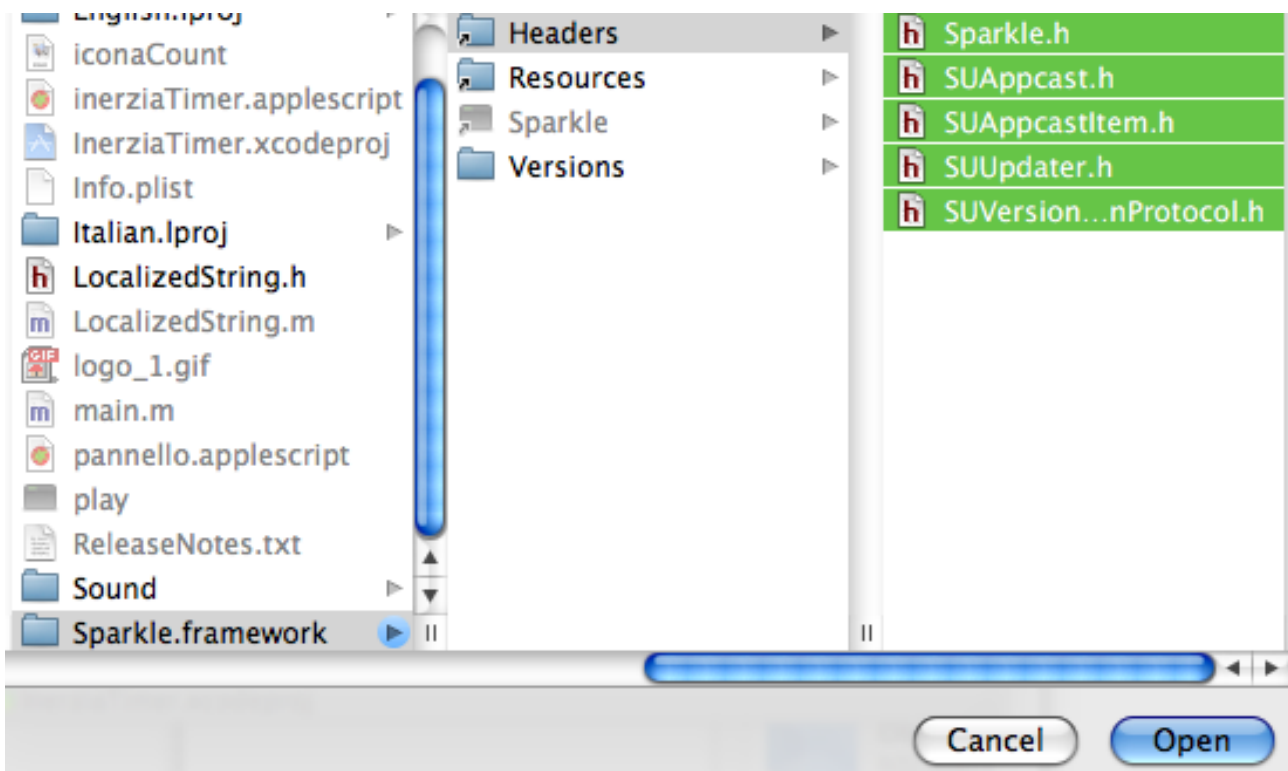
Menu Check for updates...

Abbiamo ancora da fare qualche operazione sul progetto XCode. Per prima cosa dobbiamo creare un legame con il framework, per gestire un menu che lancerà il controllo manuale dell’esistenza di una nuova versione.

Nel progetto, un click sulla cartella *Resources*, localizziamo il file dell’interfaccia *MainMenu.nib* e apriamolo con un doppio click (si avvierà Interface Builder). Dal menu *File* apriamo la voce *Read Class File...*:

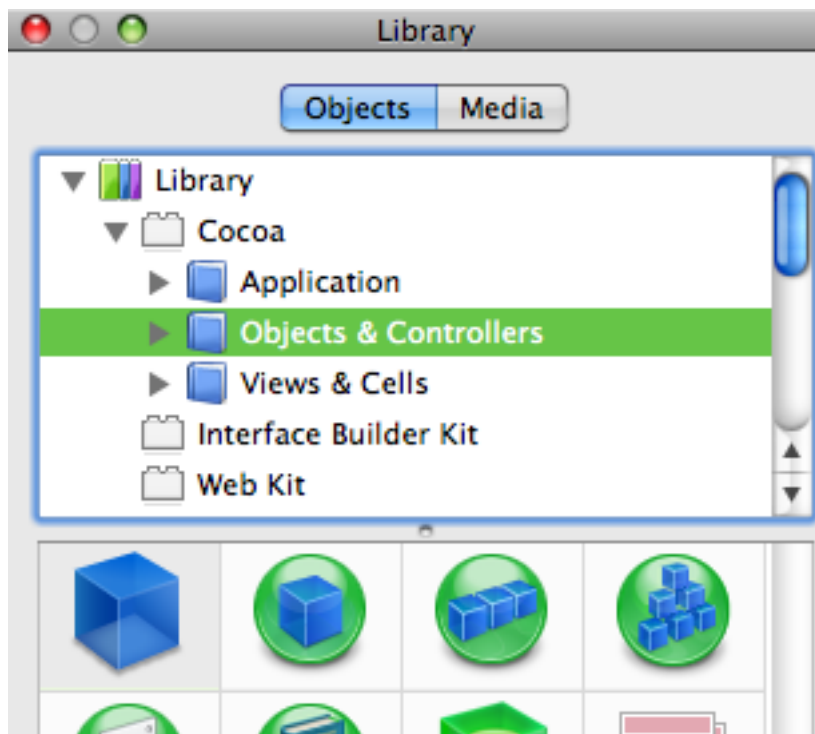


Si apre un normale dialogo di apertura; navighiamo fino alla cartella del nostro progetto, localizziamo al suo interno la cartella *Sparkle.framework*; all’interno di questa troviamo degli alias (si riferiscono sempre all’interno del framework), click sulla cartella *Headers* e selezioniamo tutti i file lì contenuti:

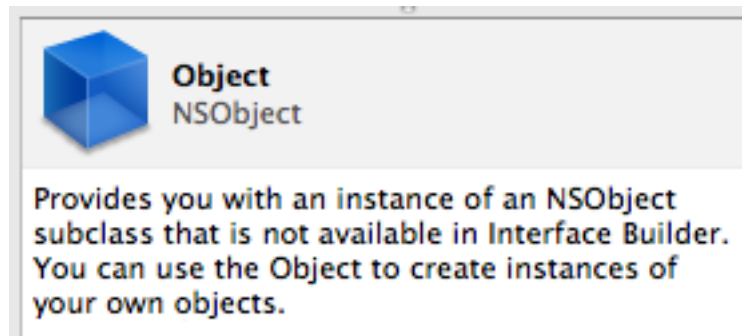


che avranno tutti l'estensione .h (si tratta appunto di header); clicchiamo su Open e il dialogo si chiuderà. Abbiamo aggiunto gli header dal framework.

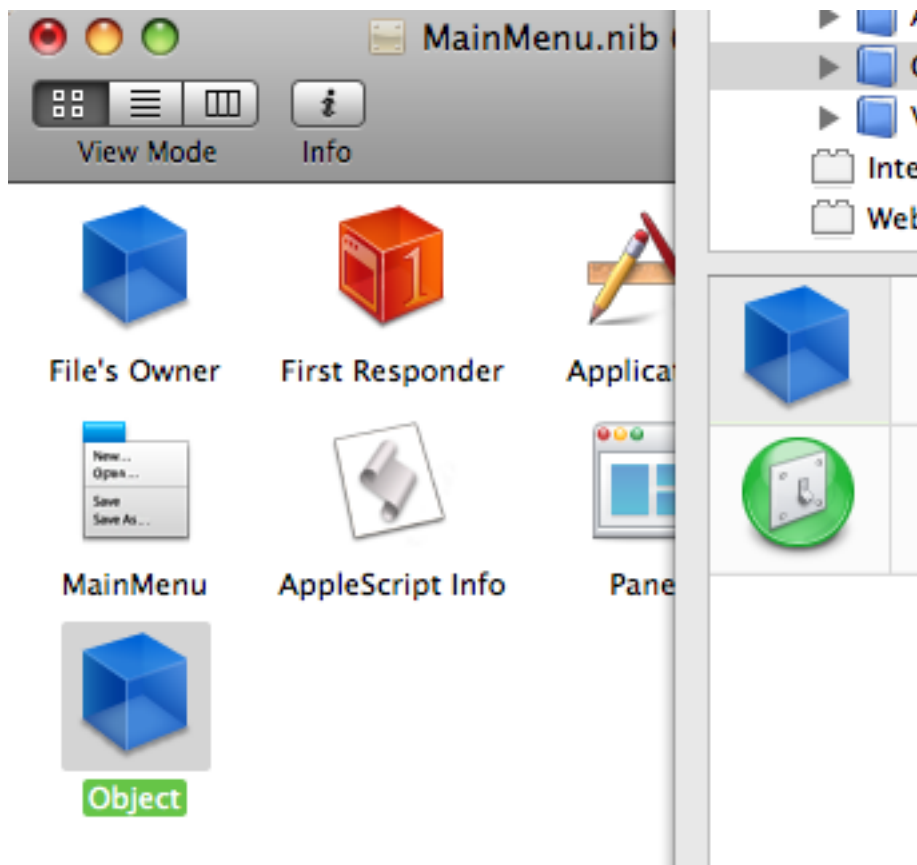
Sempre nell'Interface Builder, nella finestra Library, click su Objects and Controllers:




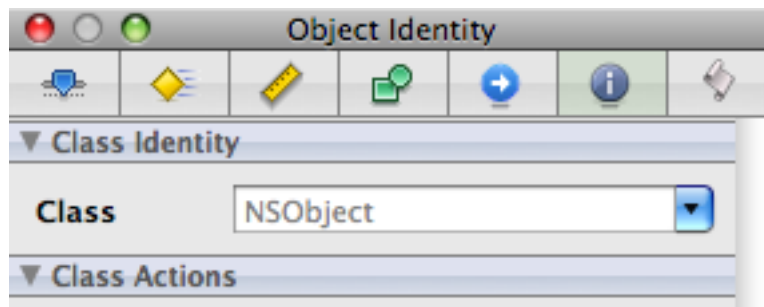
Se selezioniamo la prima icona (il cubo blu), nella parte inferiore della finestra compare il nome dell'oggetto (NSObject), che in questo caso ci dice che si tratta di un oggetto generico:



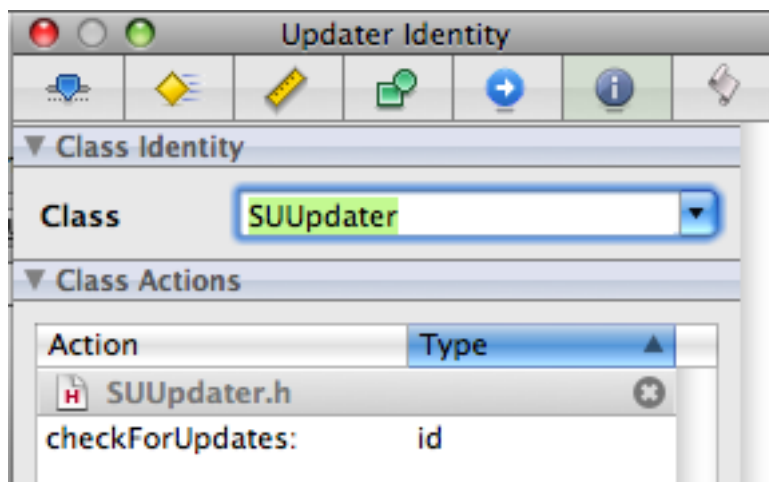
La scritta ci avverte che è proprio quello che cerchiamo: un oggetto di solito non disponibile nell'Interface Builder. Prendiamo quindi l'icona del cubo blu e facciamo un drag&drop sulla lista della finestra MainMenu.nib: otterremo una copia chiamata Object, con la stessa icona:



Selezioniamo il nuovo oggetto, apriamo l'inspector (menu Tools - Inspector oppure Command-Shift-I) e clicchiamo su  :

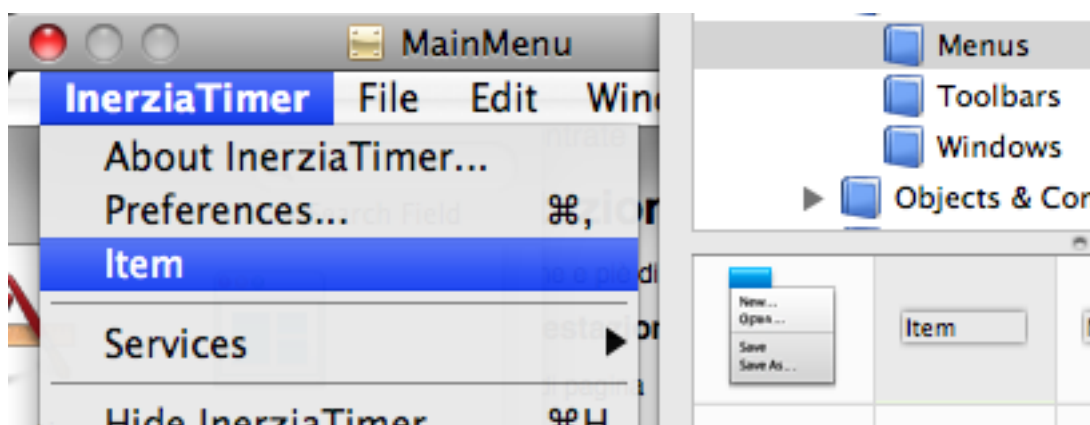


e nel campo Class scriviamo “SUUpdater”; se abbiamo fatto tutto per bene (cioé se avremo fatto leggere gli header) la parola si autocompleta man mano che la scriviamo. Al termine premiamo Return ed otterremo:



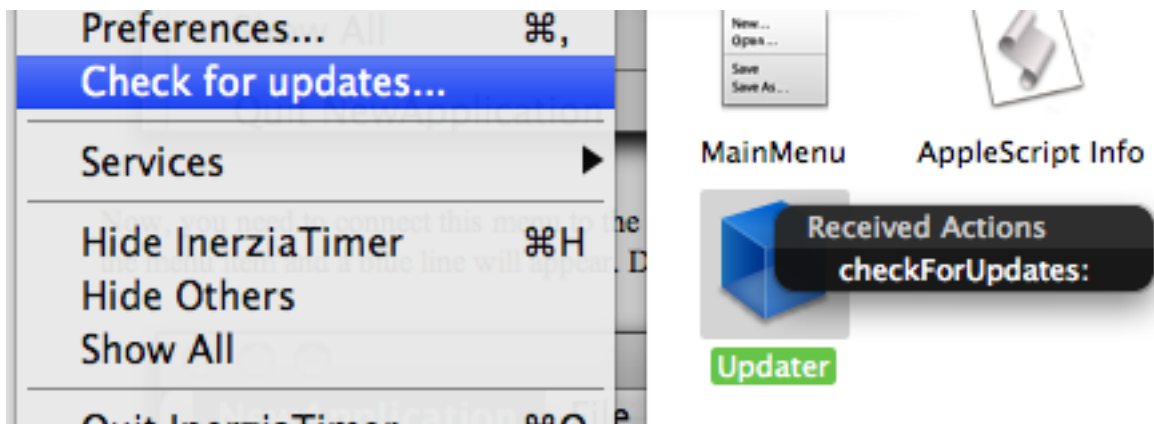
dove è stata aggiunta la Class Action cercata. Notiamo anche che il cubo in blu ha preso il nome di “Updater”.

Ora dobbiamo aggiungere il nuovo menu: di solito questo si trova nel menu con il nome dell'applicazione; per aggiungerlo, agiamo come al solito, trascinando un nuovo menu item dalla libreria nel menu voluto (lo apriremo dall'elenco dei file):



e gli cambieremo nome: dato che stiamo lavorando su un MainMenu.nib in inglese, lo chiameremo “Check for updates...”; le regole dell'interfaccia richiedono l'aggiunta dei tre puntini di sospensione, per indicare che quel comando aprirà un dialogo. Ora non ci resta che collegare il nuovo menu con il cubo “Updater”: tenendo premuto il tasto Control,

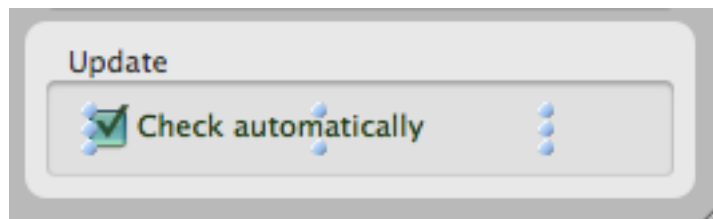
trasciniamo dal nuovo menu al cubo. Se tutto è a posto, un messaggio ci avvisa dell'azione:



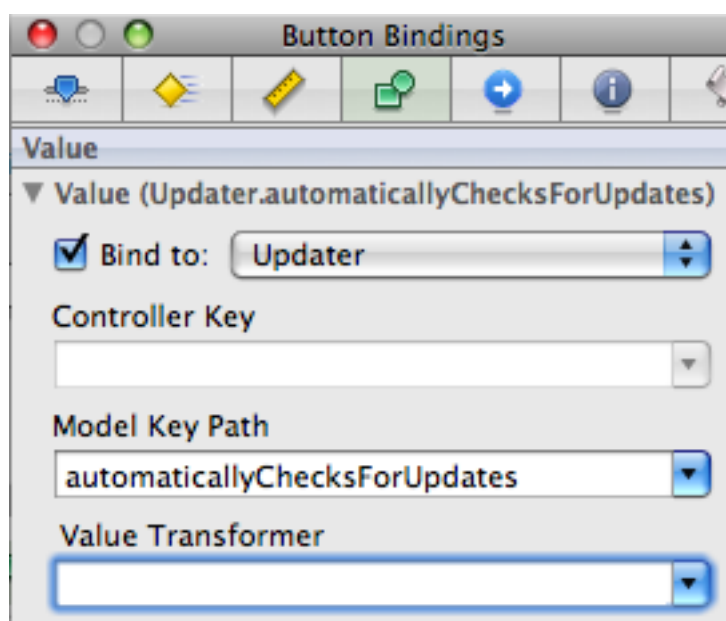
Preferenze

Visto che siamo stati così bravi, facciamo un'ultimo passo: inseriamo una preferenza con la quale l'utente può scegliere se il check della versione viene fatto automaticamente o meno.

Apriamo da Interface Builder la finestra che avremo creato per gestire le preferenze e aggiungiamo un check box (se non avevamo una finestra delle preferenze, possiamo chiederci se è il caso di farne una apposta solo per il controllo automatico...) e clicchiamo sul check box appena creato:

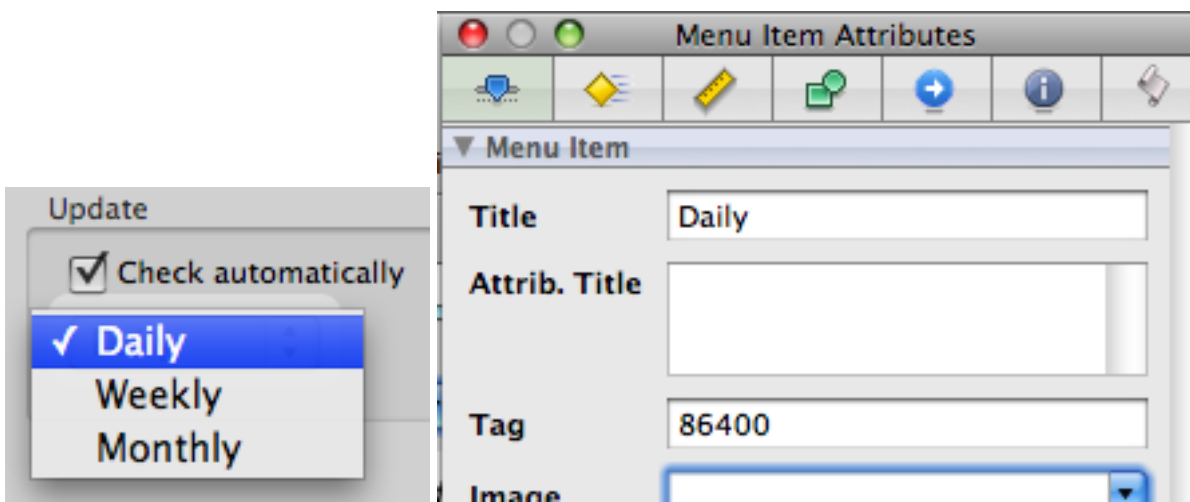


Poi andiamo nella finestra dell'Inspector e clicchiamo sull'icona 

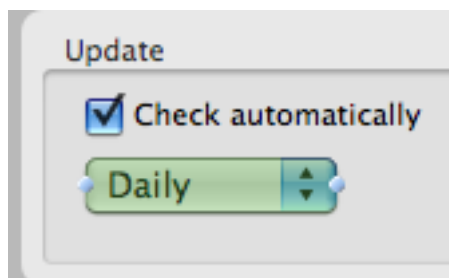



Click sul check box “Bind to:” e dal menu a discesa scegliamo *Updater*; nel campo Model Key Path scriviamo esattamente “`automaticallyChecksForUpdates`” e lasciamo le altre scelte come le abbiamo trovate. In questo modo abbiamo creato un legame (*binding*) tra il valore del check box ed il parametro nel framework che controlla il check automatico; tale valore verrà automaticamente salvato nelle preferenze e quindi la scelta si manterrà finché l’utente non la cambierà. Queste impostazioni valgono dalla versione 1.5b5 di Sparkle: con le precedenti avremmo dovuto usare *Shared Preferences* nel menu a discesa e scrivere `SUEnableAutomaticChecks` nel Model Key Path.

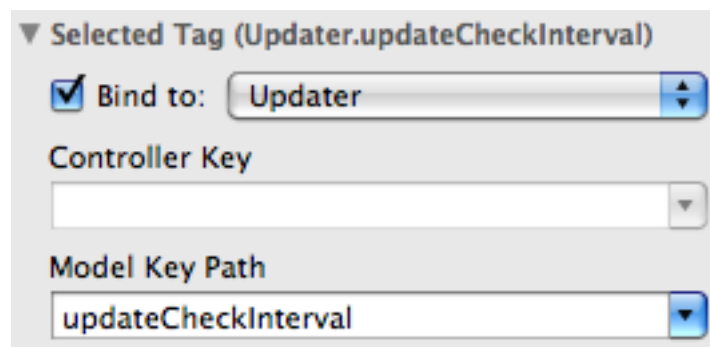
È anche possibile impostare la frequenza dei controlli automatici, che di solito avvengono al lancio dell’applicazione. Basta impostare un menu popup con le frequenze volute e impostare il tag al valore del ritardo in secondi (cioè 86400 per una volta al giorno, 604800 per una volta alla settimana, 2629800 per una volta al mese).



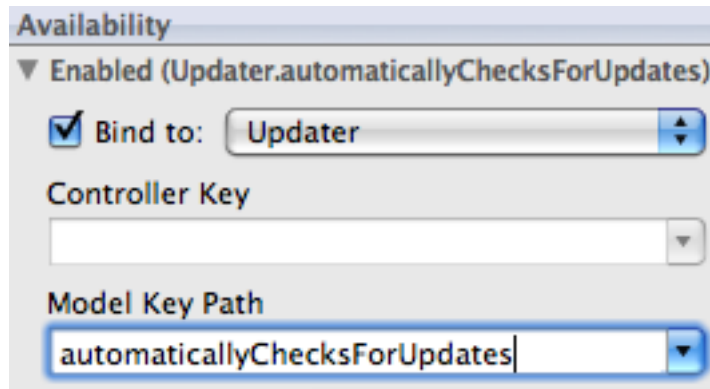
Ora selezioniamo il popup:



andiamo nell’Inspector, sezione Bindings , clicchiamo su Selected Tag e riempiamo come in figura:



e poi facciamo in modo che il popup sia abilitato se il check box inserito prima (check automatically) è selezionato, per cui, sempre con il popup selezionato, click su *Enabled* e riempiamo di conseguenza:



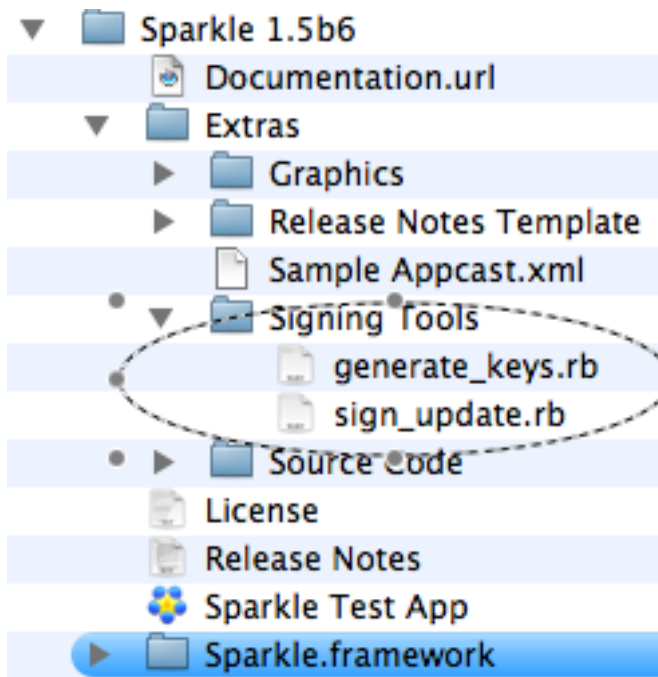
In questo modo, se non vogliamo i controlli automatici, anche il popup non sarà attivo, come ci si deve aspettare in una corretta interfaccia grafica.

Con questa versione di Sparkle, si potrebbe anche chiedere all'utente se è d'accordo sullo spedire dati anonimi sulla configurazione del proprio Mac, oppure se vuole fare il download automatico della nuova versione, senza attendere conferma. La procedura è identica alle precedenti, con la differenza che le chiavi di Bindings sono rispettivamente *sendsSystemProfile* e *automaticallyDownloadsUpdates*. Teniamo conto che di solito gli utilizzatori di un software odiano queste due possibilità, per cui lasciamo loro la scelta!

Firma degli update

Ancora un argomento che di solito viene sottovalutato: che faremmo se ci venisse all'improvviso chiesto di fare un download di qualcosa di imprecisato da un sito web sconosciuto? Molto probabilmente diremmo di no! D'accordo che Mac OS X è difficilmente attaccabile da un virus o da un qualunque malware, ma non esageriamo. Se poi durante il download ci venisse richiesta la password di amministratore... drizzeremmo le orecchie! Se invece il sito si qualificasse con un bel certificato, allora saremmo più tranquilli. Allora perché non offrire ai nostri utenti qualcosa di cui possano fidarsi? In altre parole, a meno che non usiamo una connessione sicura (SSL), è bene fare in modo che ci sia un controllo di quello che l'utente sta per scaricare: prevenire è sempre meglio che curare!

Possiamo facilmente immaginare qualche malintenzionato che si mette in ascolto sulla rete e quando cattura un messaggio indirizzato al nostro server, si sostituisce a questo e l'utente si trova sul proprio hard disk un qualcosa pronto a fare danni. Considerando questo rischio, Andy Matuschak, l'autore di Sparkle, a partire dalla versione 1.5b6, ha disposto che l'utilizzo del certificato sia obbligatorio, a meno che non ci troviamo su una connessione sicura. Di solito, l'utilizzo delle chiavi di sicurezza è un argomento un po'



scargarle dal repository dove si trovano assieme al resto del codice: attualmente l'indirizzo è <http://bazaar.launchpad.net/~andymatuschak/sparkle/main/files>. I file da scaricare sono `generate_keys.rb` e `sign_update.rb`.

oscuro, ma Sparkle arriva con tutto quanto serve per costruirsi una coppia di chiavi asimmetriche che assicureranno l'utente che l'update è proprio generato da noi. Come?

Per prima cosa si genera la coppia di chiavi, con un piccolo script in linguaggio Ruby; tale linguaggio arriva assieme al Mac OS X, per cui lo abbiamo tutti. Da notare che tale script utilizza OpenSSL, un sistema di crittatura *open source*. All'interno della cartella Extras sul dmg di Sparkle si trova lo script, chiamato `generate_keys.rb`; per ottenere le chiavi, useremo il terminale.

Attenzione: nelle versioni beta, spesso le due funzioni `.rb` per la firma dell'update non sono inserite nel dmg. Bisogna andare a

Supponiamo di aver copiato il contenuto del dmg di Sparkle nella cartella Downloads come appare in figura. Apriamo il terminale e scriviamo il comando:

```
ruby "Downloads/Sparkle 1.5b6/Extras/Signing Tools/generate_keys.rb"
```

Naturalmente se invece avevamo copiato la cartella di Sparkle da un'altra parte, metteremo il percorso corretto; l'importante è il comando ruby e che tutto il percorso sia compreso dentro i doppi apici. Per questo motivo, se decidete di far scrivere l'indirizzo al Finder trascinando sul Terminale il comando ruby, dovrete poi togliere i caratteri \ che il Terminale aggiunge per trattare gli spazi: essendo tutto racchiuso tra virgolette, i \ verrebbero letti come caratteri normali.

Se tutto va bene, la risposta sarà prima

```
Generating DSA parameters, 2048 bit long prime  
This could take some time  
..+..+++++
```

che avverte che dovremo attendere un po' e nel frattempo vedremo comparire dei punti intervallati da segni +. Alla fine avremo il messaggio

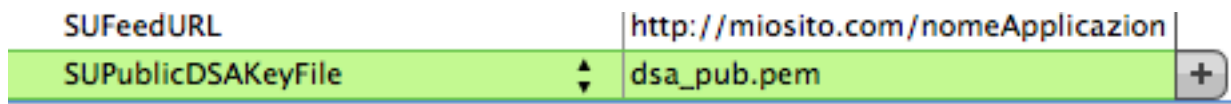
```
Generating DSA key, 2048 bits  
read DSA key  
writing DSA key  
Generated private and public keys: dsa_priv.pem and dsa_pub.pem.  
BACK UP YOUR PRIVATE KEY AND KEEP IT SAFE!  
If you lose it, your users will be unable to upgrade!
```

Il programmino ha generato le due chiavi, chiamandole `dsa_priv.pem` (privata) e `dsa_pub.pem` (pubblica).

Troveremo le due chiavi nella cartella a cui puntava il Terminale al momento dell'esecuzione del comando `ruby`; dato che all'apertura il Terminale punta sempre alla nostra cartella `Home`, se non abbiamo cambiato nulla, li troveremo le due chiavi.

Come dice l'avviso sul Terminale, copiamo la nostra chiave privata in un posto sicuro: se qualcuno ne venisse in possesso, potrebbe presentarsi al nostro posto e i nostri utenti non potrebbero distinguerlo da noi! Allo stesso modo, se la perdessimo, non avremmo alcun modo di fare un update! Quindi, facciamone una o due copie, per sicurezza. Per chi condivide un Mac assieme ad altri utilizzatori, può essere interessante conservare queste chiavi dentro... al portachiavi del Mac! Nell'Appendice A viene riportato come fare.

Copiamo invece la chiave pubblica nella cartella `Resources` del nostro progetto XCode; anche questa non dovrà essere persa (la coppia così costruita è unica), ma questa almeno la potremmo trovare in ogni copia del nostro software! Dovremo modificare il file `Info.plist` del progetto, aggiungendo la key `SUPublicDSAKeyFile`, con valore uguale al nome della chiave pubblica (`dsa_pub.pem` se non gli abbiamo cambiato nome).



Quando avremo un update da mettere a disposizione, dovremo *firmarlo*; in altre parole useremo la nostra chiave privata per ottenere l'impronta dell'update. Utilizzeremo a questo scopo un'altra routine in ruby, anche questa presente nel dmg di Sparkle, con l'ovvio nome di `sign_update.rb`. Utilizzando lo stesso percorso di prima, il comando sarà del tipo

```
ruby "Downloads/Sparkle 1.5b6/Extras/Signing Tools/sign_update.rb"
percorsoZip percorsoChiave
```

(il comando va scritto tutto di seguito, senza dare nessun return durante la scrittura - il Terminale andrà a capo senza spezzare il comando) dove `percorsoZip` punta al file dell'update nel formato scelto (zip, dmg,...), mentre `percorsoChiave` porta alla chiave privata. Il risultato del comando (che non è immediato, dovremo attendere un po' di tempo dipendente dalle dimensioni del file zip) è una stringa del tipo

```
MC0CFCL3xNRIyN40y1C93oU2eFzJJ5f3AhUAKHkQg5Tse0uD0rJlNaxuxpZ2aQ=
```

che useremo nel prossimo passo, inserendola nell'AppCast.

Riassumendo, i passaggi per un update sicuro sono:

1. ricavare una coppia di chiavi pubblica-privata con lo script `generate_keys`;
2. inserire la chiave pubblica nella cartella `Resources` del progetto XCode;
3. aggiungere la key `SUPublicDSAKeyFile` nel file `info.plist`;
4. ottenere la firma del nostro update con lo script `sign_update`;
5. inserire la firma nell'AppCast (vedremo tra un attimo).

Nel momento del download sulla macchina dell'utente, Sparkle controllerà che la firma sia corretta, usando la chiave pubblica inserita nell'applicazione: se lo è, l'update viene permesso, altrimenti no e l'utente è salvo.

Nota: la generazione delle chiavi può essere fatta una volta sola per tutte le nostre applicazione e per tutti gli update. Infatti, la chiave privata è a tutti gli effetti simile alla nostra carta di identità: è sempre la stessa, qualunque uso ne facciamo e ci identifica. Quello che cambia da una volta all'altra è soltanto la firma, che dipende sia dalla chiave privata che dal file che metteremo a

disposizione. La chiave pubblica viene utilizzata per controllare che il file sia proprio quello previsto, ma non può essere usata per firmare qualcosa (proprio per questo è pubblica!).

File AppCast

Finalmente abbiamo terminato tutto quello che andava fatto sul progetto XCode: notiamo ancora una volta che non è stato necessario scrivere una riga di codice, ma abbiamo soltanto sfruttato le caratteristiche di XCode, oltre naturalmente al framework di Sparkle!

Però... non vorrete mica ottenere tutto senza fatica, no? Ci manca ancora un tassello! Infatti, abbiamo fornito all'applicazione l'URL a cui cercare il file che descrive l'aggiornamento: almeno quello dovremo crearlo!

Si tratta, come già detto, di un file tipo RSS (cioè, un xml); nella documentazione inserita nel dmg di Sparkle c'è un esempio completo: nella cartella `Extras` possiamo trovare `Sample AppCast.xml`. La cosa importante è ricordarsi di inserire il nome del pacchetto della nuova versione dell'applicazione, che deve essere nominato come "Nome Applicazione_2.1.zip", cioè deve essere zippata ed il nome, compresi gli eventuali spazi, deve essere seguito dal carattere di sottolineatura, a cui segue il numero di versione (separato da punti). Possono anche essere usati i formati `dmg`, `tar.gz` e `tar.bz2`.

Da notare che potete inventarvi anche un altro schema (p.es. `applicazione_data.zip`); basta che poi usiate lo stesso schema nel file XML e sul vostro sito. Il differenziare nomi e versioni vi permetterà di mantenere sul sito tutte le versioni di ogni software, senza problema di ambiguità. Questo vi aiuterà in chiarezza.

L'esempio fornito nel dmg di Sparkle è esplicativo. Possiamo anche lasciare all'interno tutti gli aggiornamenti: l'aggiunta di uno di essi si riflette soltanto nell'aggiunta di un elemento `<item>`. Diamo quindi solo un'occhiata alla struttura di questi elementi; i nodi importanti sono nel link delle note di rilascio (`sparkle:releaseNotesLink`) e nel tag `enclosure`, dove vengono definiti il link all'update, la versione dell'update, la firma e la versione minima del sistema operativo:

```
<item>
  <title>Version 3.0 (5 bugs fixed; 2 new features)</title>
  <sparkle:releaseNotesLink>
    http://sito.com/releaseNote/versione30.html
  </sparkle:releaseNotesLink>
  <pubDate>Sat, 13 Aug 2008 11:14:01 +0000</pubDate>
  <enclosure url="http://www.sito.com/download/miaApplicazione_3.0.zip"
    sparkle:version="3.0" length="465792"
    sparkle:dsaSignature="MC0CFCL3xNRIyN4...Qg5Tse0uD0rJlNaxuxxpZ2aQ="
    sparkle:minimumSystemVersion="10.3.9"
    type="application/octet-stream" />
</item>
```

1. La `minimumSystemVersion` è necessaria solo se la nuova applicazione ha cambiato i requisiti: l'annuncio verrà mostrato solo se il sistema operativo dell'utente potrà far girare anche la nuova. Consiglio comunque di metterla sempre, così non la dimenticheremo quando ci dovesse servire.

2. La `dsaSignature` è costituita dalla stringa di firma ottenuta nella sezione precedente (qui abbiamo messo dei puntini solo per non confondere troppo: nella realtà ci sarà sempre tutta e chiusa tra virgolette, come per tutti gli altri valori).
3. Se usiamo numeri di build interni per `CFBundleVersion` ed una versione commerciale per `CFBundleShortVersionString`, si può fare in modo che Sparkle non mostri i numeri interni; basta inserire, nel tag `enclosure`, la versione interna per `sparkle:version` (p.es. "b123") e il numero commerciale per `sparkle:shortVersionString` (p.es. "3.0").
4. Il file dell'applicazione aggiornata deve avere lo stesso nome di quella che andrà a sostituire, a parte la versione, come spiegato sopra.
5. Le eventuali *release note* sono raccolte in un normale file html: se tale file esiste, verrà mostrato nel dialogo di richiesta di update.

Ancora due informazioni:

- Sparkle non effettuerà il controllo al primo lancio dell'applicazione, ma dal secondo in avanti;
- se esiste una nuova versione, Sparkle chiede se si vuole aggiornare subito; le altre possibilità sono di saltare l'aggiornamento o ricordarlo più tardi.

Come proseguire

Ora avete tutte le armi per poter utilizzare questo utile framework. Potete prendere una qualunque applicazione in Applescript e aggiungervi il meccanismo di aggiornamento. Quando vi sentirete a vostro agio, potete scorrere nel dettaglio le istruzioni riportate sul sito di Sparkle, per capire il perché dei vari passaggi. Quando vi sentite sicuri, date una scorsa alle Appendici, non ve ne pentirete!

Conclusione

Sparkle presenta ulteriori possibilità se usato con Cocoa, come per esempio, l'aggiornamento di un Preference Pane; d'altra parte, è impensabile che un pannello di preferenze venga fatto in Applescript Studio! E comunque lo stesso Cocoa permette cose non previste da un progetto AS: se vogliamo dare libero sfogo al nostro estro creativo informatico, il consiglio è, prima o poi, di passare al Cocoa e al suo Objective-C!

Nota

Questo tutorial dovrebbe essere sufficientemente esplicativo; tuttavia, potrebbe contenere degli errori; inoltre, potrei averne fatta una nuova versione per seguire i cambiamenti nelle versioni di Sparkle. Per cui consiglio di tornare periodicamente sul [nostro sito](#), per verificare se per caso fosse pronta una nuova versione; questa si differenzierà dall'attuale per il numero di versione che segue il nome del file: la stesa versione è riportata nell'intestazione di ogni pagina (in questo caso il numero è 1.2). Se fossero diverse... siete davanti ad una nuova versione di questa guida!

Appendice A: uso del Portachiavi



Dove conservare delle chiavi? Ovvio: nel portachiavi!

Questa filosofia spicciola ha senso se stiamo usando un Mac, dove tutte le password, certificati e documenti da proteggere da occhi indiscreti possono essere immagazzinati in modo sicuro utilizzando Accesso Portachiavi, nella cartella Utility, che si presenta con l'icona a fianco.

La chiave privata e quella pubblica non sono altro che file di testo; di questo ci possiamo facilmente accertare facendone aprira una da TextEdit:

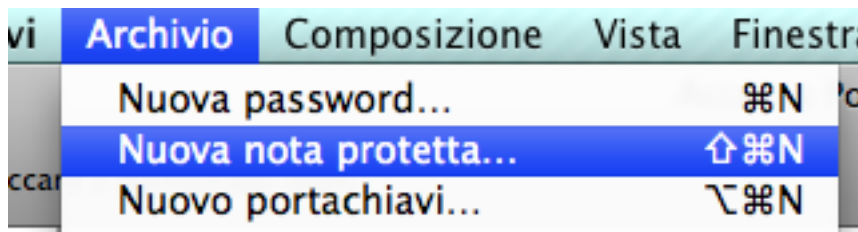
```

TextEdit  Archivio  Composizione
dsa_pub.p
---BEGIN PUBLIC KEY-----
[DOjCCAi0GBYqGSM44BAEwggIgAoIBAQDb/8td7DYN
]CqF4QYh1EKokm7k7Dk54wL+PesbWoHvgso0SDdAWt
jWlhz0A4ydPn/8mRLJRr6jaszHEdSNFLNM3BUlgygp
noiwvsWdN0oFrcKHZ0jhrvr0s904BTsLCq0asSS3yJ
+pcfUPHvQBqw6o29jGU8hf7BTCNxTwsJZhtY368Sr3

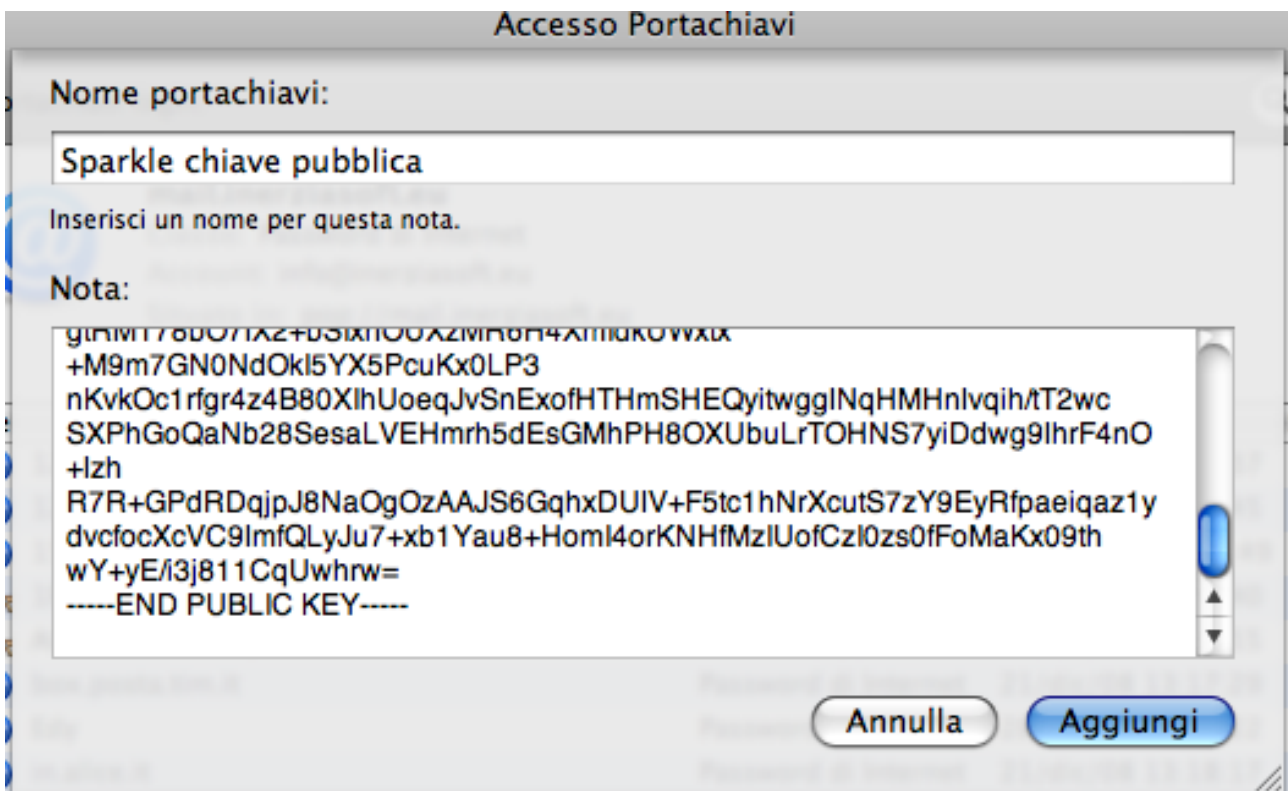
```

Allora possiamo salvarle entrambe come Note Protette nel nostro portachiavi.

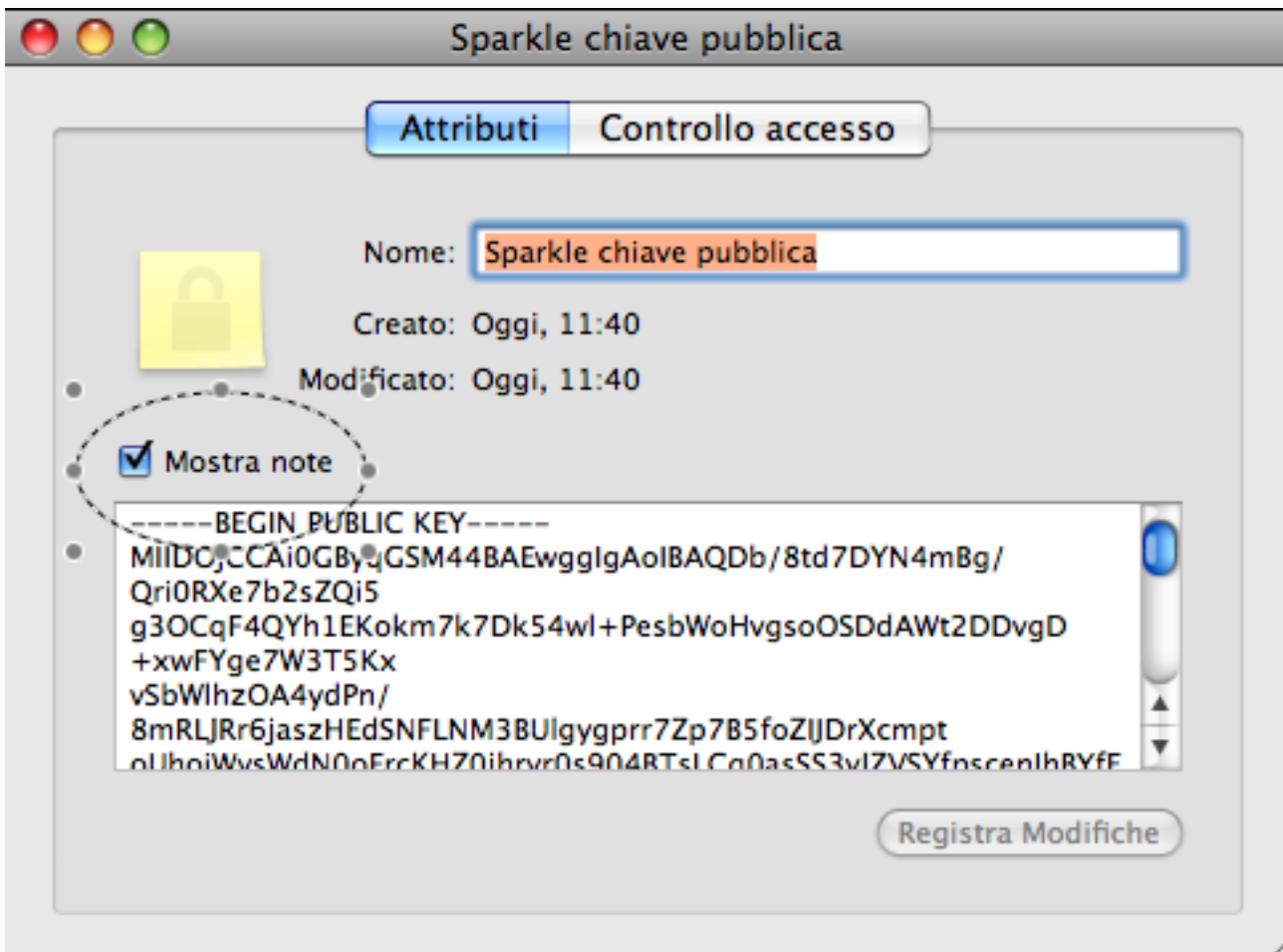
Apriamo Accesso Portachiavi e ci troviamo davanti ad una finestra che lista tutte le nostre password, relativa al portachiavi (in inglese KeyChain) che viene aperto al login. Andiamo nel menu Archivio e scegliamo Nuova Nota Protetta:



Si aprirà un dialogo allegato alla finestra, in cui andremo a inserire il nome della nota con cui vogliamo salvarla e tutto il contenuto del file della chiave, comprese le scritte iniziali e finali che delimitano la PUBLIC KEY (ottenuto con un copia e incolla dalla finestra di Text Edit: notiamo che non si può fare un drag & drop, poiché ci troveremmo solo con il percorso del documento contenente la chiave!):

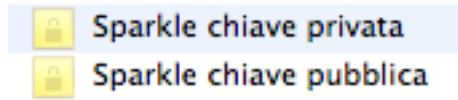


Dopo un click su Aggiungi troveremo la nostra chiave nell'elenco degli oggetti del portachiavi e se facciamo un doppio click potremo vederla:



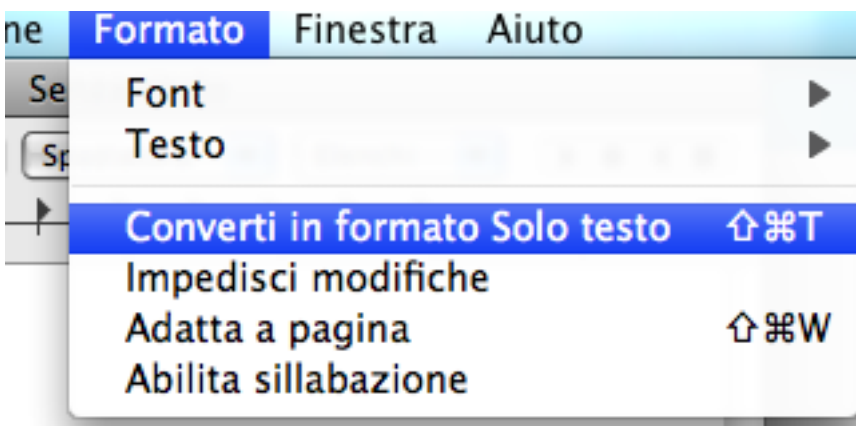
Il contenuto della nota diventa visibile solo se mettiamo il check su *Mostra note* e forniamo la password di amministratore e clicchiamo su **Consenti**; questa è una buona forma di protezione: non è consigliabile fare click su **Consenti sempre**. altrimenti chiunque abbia accesso al computer potrebbe visualizzare la chiave.

Ripetiamo la procedura con la chiave privata, ovviamente chiamandola “Sparkle chiave privata”. Avremo quindi nell’elenco entrambe le chiavi:



Dopo queste operazioni, possiamo cancellare entrambe le chiavi dal finder. Quando ci serviranno, potremo facilmente ricuperarle: basterà riaprire Accesso Portachiavi, fare doppio click su una di esse e fornire la password per visualizzare il contenuto della nota protetta. Potremo quindi copiarle in file di testo per utilizzarle: quella privata per firmare un update e quella pubblica per inserirla nella cartella Resources di un progetto XCode.

Facciamo solo attenzione al fatto che il file da utilizzare deve essere un file di testo (estensione `txt`), mentre di default Text Edit utilizza il formato `rtf`. Per evitare problemi, prima di incollare il contenuto della chiave nella finestra di Text Edit, scegliamo **Converti in formato Solo testo** dal menu **Formato** di Text Edit:



e dopo aver incollato possiamo salvare senza problemi. Naturalmente potremo usare il nome originale (`dsa_pub.pem`, `dsa_priv.pem`) o un qualunque altro; basta che li usiamo anche nei comandi ruby di firma dell’update e nel file `Info.plist` per la chiave pubblica.

È utile ricordare che, se dovessimo cambiare computer o per qualche motivo riformattare l’hard disk, ricordiamo di copiare le nostre chiavi prima di cancellare tutto! Notiamo che il comando Esporta del portachiavi è sempre disabilitato, per cui dovremo copiare a mano le informazioni. È anche possibile una procedura che evita di fare tutto a mano, per la quale però rimando al sito Apple.

Nell’Appendice B, è riportato un metodo per automatizzare le operazioni di firma direttamente in XCode, che fa uso delle Note Protette del Portachiavi.

Appendice B: automatizzare XCode per Sparkle

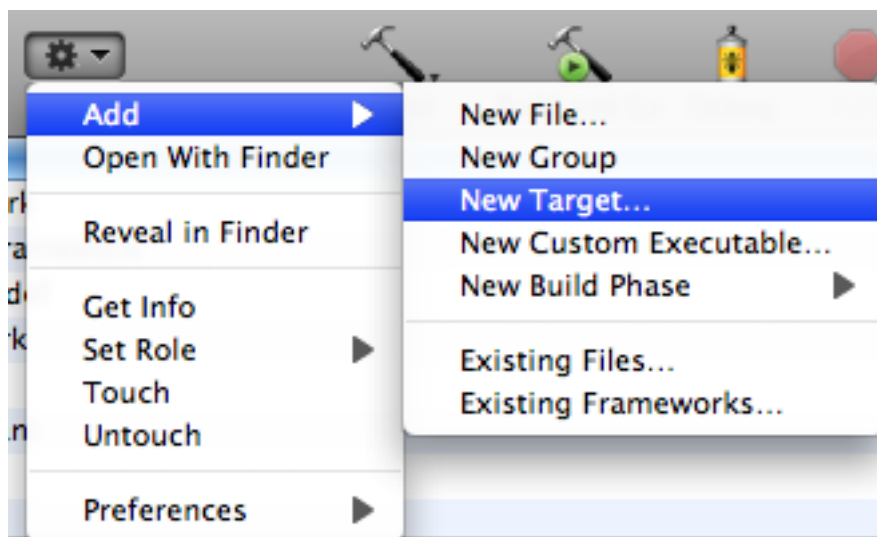
Ogni volta che finiamo il lavoro su una nuova versione di un nostro software, se abbiamo inserito il meccanismo di aggiornamento di Sparkle, dobbiamo anche costruire il necessario per farlo funzionare. Come abbiamo visto, si tratta di compilare come Release la nuova versione, zipparla e aggiungere le nuove informazioni nel file xml presente sul nostro sito; nulla di trascendentale, ma può diventare noioso. Ecco un procedimento che ben si presta all'automatizzazione e che fa uso delle chiavi private registrate nel Portachiavi.

Se abbiamo conoscenze approfondite di Unix possiamo costruirci un file che faccia tutto per noi; se invece siamo pigri... useremo un bel procedimento, mostrato da Marc Liyanage nel suo blog all'indirizzo <http://www.entropy.ch/>, che si basa appunto sulle chiavi registrate nel Portachiavi del Mac e che fa uso delle possibilità di XCode.

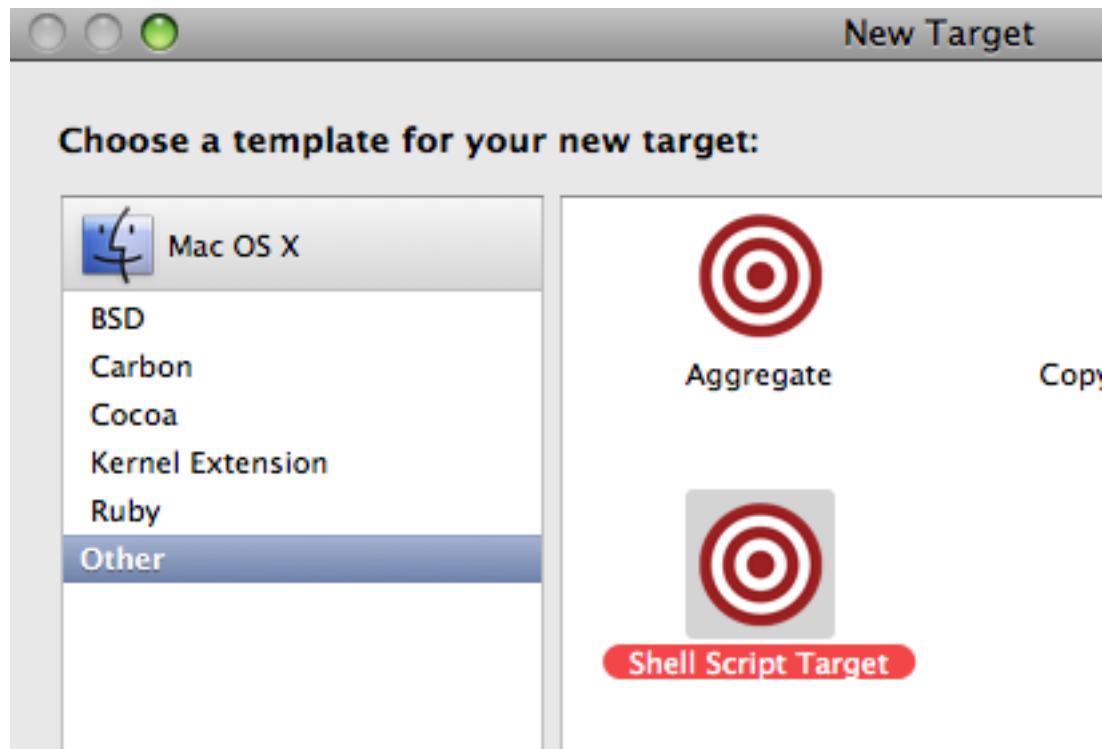
Per una caratteristica del Portachiavi, anche se possiamo cambiare il nome visualizzato ad una Nota Protetta, per qualche ragione il nome interno resta invariato. Per cui, per essere sicuri che tutto funzioni, non cambiamo il nome delle Note che contengono le chiavi. Se per caso le avessimo cambiate, è meglio rifare la procedura dall'inizio, creando note con un nome che non cambieremo più.

Target "Distribution"

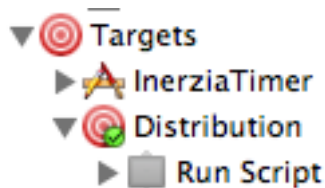
Per prima cosa, aggiungiamo al nostro progetto XCode un nuovo target, che chiameremo *Distribution*: dal pulsante Action nella Toolbar, scegliamo Add->New target...



Viene esposta una finestra dalla quale dovremo scegliere il tipo di target; nel nostro caso, non sarà un'applicazione da costruire ma un semplice script della shell, per cui andremo a selezionare sulla sinistra la categoria **Other** e, al suo interno, il tipo **Shell Script Target**, come indicato nella figura seguente:

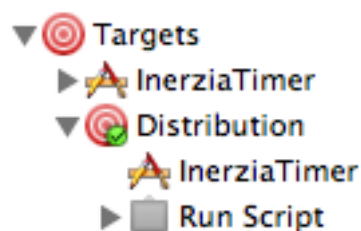


Un click su **Next** e compare un dialogo dove andremo a mettere il nome (*Distribution*, ma può essere qualsiasi - l'ho mantenuto così per coerenza di XCode, che è in inglese). Click su **Finish** e usciamo anche dal dialogo seguente.



Sulla destra ci troviamo quindi con un nuovo target, con il nome che abbiamo impostato e al suo interno un blocco Run Script, che ci dice che abbiamo impostato le cose giuste.

Inoltre, per automatizzare al meglio le operazioni di distribuzione del nostro prodotto, vorremmo poter fare in un colpo solo la compilazione della nostra applicazione in modo *Release* e la costruzione dell'elemento `<item>` del file appcast; vediamo come ciò possibile.



Apriamo gli elementi target nella parte sinistra della finestra di XCode, come appare nello screenshot qui sopra. Con il mouse, prendiamo il target principale (quello con il nome della nostra applicazione) e ne facciamo un drag & drop sul target *Distribution* appena creato: possiamo rilasciare quando appare una riga blu subito sotto il target: in questo modo abbiamo creato un alias del nostro software nel nuovo target (infatti non compare il triangolino grigio che indica la presenza di operazioni al suo interno). Se abbiamo fatto tutto correttamente, la situazione è quella nella figura a fianco, in cui l'alias dell'applicazione compare prima del *Run Script*: quando faremo la compilazione del target *Distribution*, verrà prima compilata la nostra applicazione e poi verrà lanciato lo script.

Ora dobbiamo inserire lo script che verrà lanciato quando sceglieremo questo target. Per far ciò, facciamo doppio click sul blocco grigio *Run Script*: si aprirà una finestra con il tab *General* selezionato, in cui andremo a scrivere `/bin/bash` (non aggiungete un carattere / alla fine!) nel campo *Shell* (lo script ha bisogno di caratteristiche disponibili solo in questa shell), mentre nel campo *Script* inseriremo lo script di Marc, che per comodità riporto per intero qui sotto, avendo soltanto tradotto gli avvisi.

Seguite il codice riportato qui, poiché indico dove deve essere personalizzato per il proprio progetto ed ho corretto qualche piccola imprecisione. Ogni riga non deve essere interrotta ed il return deve essere dato solo alla fine della riga, anche se il codice in questa pagina è andato a capo. Per facilitare la copiatura, ho aggiunto i numeri di riga: non dovete assolutamente copiarli! Sono solo un aiuto per capire dove comincia la riga e dove andare a capo con un Return. Non copiate nemmeno quanto scritto su fondo bianco (sono miei commenti al codice);

```
01- set -o errexit
02-
03- [ $BUILD_STYLE = Release ] || { echo Il progetto viene distribuito con la build
  "Release"; false; }
```

Legge la versione da Info.plist (CFBundleVersion); se invece siete abituati a inserire la versione nella chiave CFBundleShortVersionString, inserite questa al posto dell'altra! Notate che c'è uno spazio prima della chiave:

```
04- VERSION=$(defaults read "$BUILT_PRODUCTS_DIR/$PROJECT_NAME.app/Contents/Info"
  CFBundleVersion)
```

Il seguente è l'URL per il download: sostituite con il vostro

```
05-
06- DOWNLOAD_BASE_URL="http://www.mioSito.com/download"
```

Il seguente invece è l'URL alla release note: anche qui mettete quello giusto; il file è costruito come nomeApplicazione_versione.html; se ne volete uno fisso, sostituite l'ultima parte:

```
07- RELEASNOTES_URL="http://www.mioSito.com/Notes/${PROJECT_NAME}_${VERSION}.html"
```

Qui costruisce automaticamente il nome del file da scaricare; però deve essere previsto uno spazio. Se non è così, potete scrivere tra virgolette direttamente il nome file; non c'è bisogno di costruire lo zip, farà tutto questo script!

```
08- ARCHIVE_FILENAME="${PROJECT_NAME}_${VERSION}.zip"
09- DOWNLOAD_URL="$DOWNLOAD_BASE_URL/$ARCHIVE_FILENAME"
```

Qui sotto mettete tra virgolette il nome usato nel Portachiavi per la chiave privata:

```
10- KEYCHAIN_PRIVKEY_NAME="Sparkle chiave privata"
11-
12- WD=$PWD
13- cd "$BUILT_PRODUCTS_DIR"
14- rm -f "$PROJECT_NAME".zip
15- zip -qr "$ARCHIVE_FILENAME" "$PROJECT_NAME.app"
16-
17- SIZE=$(stat -f %z "$ARCHIVE_FILENAME")
18- PUBDATE=$(date +"%a, %d %b %G %T %z")
```

Ora ricava la firma dell'update caricando la chiave privata.

```
19- SIGNATURE=$(
20-   openssl dgst -sha1 -binary < "$ARCHIVE_FILENAME" \
21-   | openssl dgst -dss1 -sign <(security find-generic-password -g -s
  "$KEYCHAIN_PRIVKEY_NAME" 2>&1 1>/dev/null | perl -pe '($_) = /"(.)"/; s/\\012/\\n/g') \
22-   | openssl enc -base64
23- )
```

Attenzione: Snow Leopard ha cambiato la risposta del comando security, che ora è un XML, per cui se stiamo usando MacOS X.6, le righe da 19 a 23 devono essere sostituite dalle seguenti:

```
19- SIGNATURE=$(
20-   openssl dgst -sha1 -binary < "$ARCHIVE_FILENAME" \
```

```

21-     | openssl dgst -dss1 -sign <(security find-generic-password -g -s
"$KEYCHAIN_PRIVKEY_NAME" 2>&1 1>/dev/null | perl -pe '($_) = /"(.+)"/; s/\\012/\\n/g' |
perl -MXML::LibXML -e 'print XML::LibXML->new()->parse_file("-")->findvalue(q{//string
[preceding-sibling::key[1] = "NOTE"]}))' \
22-     | openssl enc -base64
23- )

```

```

24- [ $SIGNATURE ] || { echo Non posso caricare la chiave privata
"$KEYCHAIN_PRIVKEY_NAME" dal Portachiavi; false; }
25-
26- cat <<EOF

```

ora stampa sullo schermo tutto l'elemento <item> da aggiungere al file XML

```

27-     <item>
28-         <title>Version $VERSION</title>
29-         <sparkle:releaseNotesLink>$RELEASENOTES_URL</
sparkle:releaseNotesLink>
30-         <pubDate>$PUBDATE</pubDate>
31-         <enclosure
32-             url="$DOWNLOAD_URL"
33-             sparkle:version="$VERSION"
34-             type="application/octet-stream"
35-             length="$SIZE"
36-             sparkle:dsaSignature="$SIGNATURE"
37-         />
38-     </item>
39- EOF

```

Se qualcosa non dovesse funzionare, controllate bene ogni carattere, in particolare le virgolette, gli apici e naturalmente gli 'a capo';

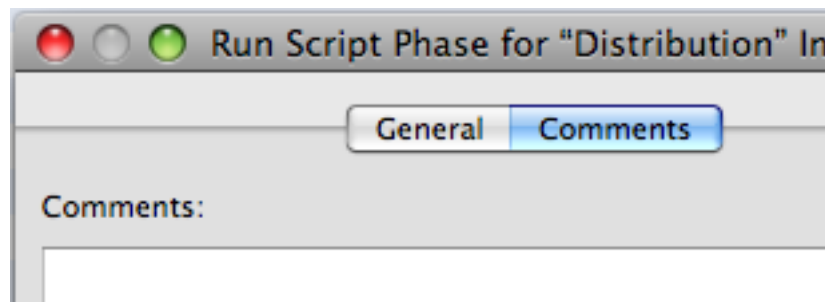
Riassumendo, questo script effettuerà le seguenti operazioni:

1. fa creare ad XCode una build del vostro software, di tipo *Release*;
2. crea uno zip dell'applicazione così costruita;
3. legge la chiave privata dal Portachiavi: per farlo potrebbe chiedervi la password di amministratore, che permette di sbloccare il portachiavi: se vedete un messaggio come il seguente

fornite pure la vostra password (ve la chiederebbe anche se seguiste la procedura manuale, al momento di aprire il Portachiavi); per sicurezza, scegliete **Consenti una volta**;

4. elabora l'impronta del file e la firma con la chiave privata, convertendola poi a Base64;
5. costruisce il blocco xml <item> da aggiungere al file appcast: possiamo usare il copia-incolla

Togliamo il check su *Show environment variables in build log*: ci semplificherà la vita nel leggere il risultato. Abbiamo inserito lo script e abbiamo fatto le opportune modifiche agli URL per la nostra situazione: ora possiamo scrivere qualche commento cliccando sull'apposito tab:



Infatti, tutti sappiamo quanto sia importante commentare il codice (vero, eh?) e questo non fa eccezione. Chiudiamo la finestra.

Supponiamo ora di aver terminato il lavoro sul nostro progetto in fase di debug: siamo pronti per il rilascio. Effettuare i seguenti passaggi:

1. dal menu **Build** scegliere Build Results (viene mostrata la finestra del build);
2. dal menu pop-up di questa finestra, scegliere *Distribution* come *Active Target*;
3. dallo stesso menu pop-up, scegliere *Release* come *Active Configuration*.

Le stesse scelte possono essere fatte anche dal menu **Project**; nel pop-up della finestra sono riassunte in un menu unico.

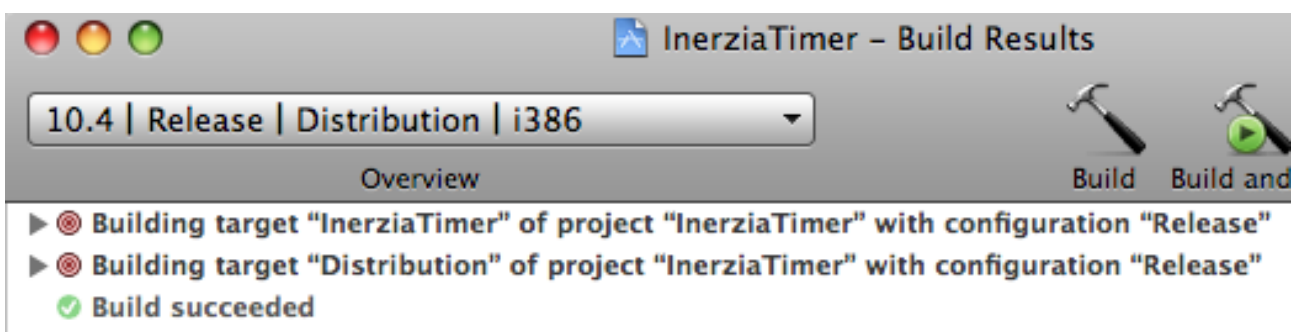
4. Nella Toolbar della stessa finestra, cliccare su Build, oppure premere Cmd-B, oppure dal menu Build-->Build.

Parte la build ed il primo prodotto trovato è la nostra applicazione, che viene compilata come *Release*; terminata la build dell'applicazione, parte lo script. Se ci siamo dimenticati di impostare il modo Release, lo script si ferma con il messaggio

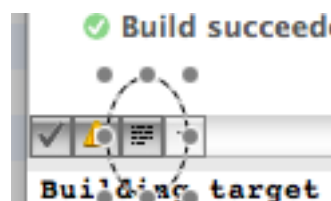
Il progetto viene distribuito con la Build 'Release'

Infatti è inutile proseguire: il software non funzionerà su un Mac senza XCode.

Se invece la build è corretta, lo script ci chiede il permesso per accedere alla chiave privata nel nostro Portachiavi (potrebbe anche chiedere la password amministratore) e viene eseguito fino in fondo. Nella parte superiore della finestra Build Results compaiono i vari step effettuati:



Per vedere i risultati, in particolare quelli dello script, se non è già visibile, dobbiamo mostrare la parte inferiore della finestra, cliccando sul terzo tastino:



Questa parte dovrebbe essere sempre visibile, ma spesso XCode la nasconde; dopo aver cliccato sul tastino indicato, potrebbe essere necessario spostare la divisione della finestra con il mouse.

In questa parte inferiore della finestra ci sono tutti i messaggi e risultati generati durante la build. Dato che lo script è stato eseguito per ultimo, andiamo fino in fondo, dove troviamo il blocco <item> da copiare e incollare nel file xml! Naturalmente, prima di farlo, meglio controllare che sia tutto a posto: i banchi si nascondono da ogni parte!