:: How To ::

Thunderbird Extension for Beginners

Terminologia

In questa sezione vediamo alcuni termini che serviranno nelle successive fasi e che potrebbero risultarvi piuttosto oscuri!!

XUL

È la tecnologia basata su XML il cui compito è quello di permettere di scrivere di interfacce grafiche portabili. Usando JavaScript è possibile associarvi del codice che reagisce agli eventi della finestra e dei suoi componenti (click, inizializzazione, focus, ...). Un buon sito dove trovare informazioni riguardo questa tecnologia è: http://www.xulplanet.com/ .

XPCOM

Essendo Mozilla un ambiente portabile sono stati creati dei componenti COM simili a quelli presenti in Windows. Fortunatamente sono abbastanza facili da utilizzare all'interno di script JavaScript come se fossero semplici oggetti di JavaScript stesso. La difficoltà sta nel riuscire a determinare il componente XPCom che è di interesse. Un'ulteriore difficoltà di questi componenti sta nel fatto che non tutti sono "freeze", cioè bloccati per il resto della loro vita, e quindi è possibile che da una versione all'altra di Mozilla alcuni funzionino parzialmente o non funzionino del tutto. Sempre sul sito http://www.xulplanet.com/ è possibile trovare una buona raccolta di tutti i componenti XPCom (e le relative interfacce) che sono disponibili.

RDF

Questa è una delle tecnologie più oscure e "meno utili" della piattaforma ma senza di questa… non è possibile fare funzionare nulla in quanto i file di installazione e di configurazione sono scritti in questo linguaggio, molto simile all'XML. In realtà non vi sarà necessario conoscere RDF, i file necessari potete semplicemente copiarli da questo documento oppure cercare su internet altre estensioni e andare a dare una sbirciatina ai file RDF di queste.

Creare un'estensione Passo-Passo

In questa sezione vediamo come costruire un estensione per Thunderbird partendo dalla struttura delle cartelle, passando per l'aggiunta dei file necessari, fino alla scrittura effettiva del codice.

PASSO 0 :: PREREQUISITI

Per poter creare un'estensione per Thunderbird, o in generale per la suite Mozilla, è necessario avere alcune minime conoscenze di base; prima di tutto è necessaria una minima conoscenza di XML per poter creare le interfacce grafiche utilizzando la tecnologia XUL. Inoltre, è necessario conoscere JavaScript: per creare script complessi è necessario utilizzare in modo molto pesante questo linguaggio. Una minima conoscenza di CSS può infine essere d'aiuto. Ovviamente è anche necessario sapere come funziona Thunderbird, o meglio come installare e disinstallare i pacchetti.
È possibile trovare approfondimenti per queste tecnologie nella sezione "Bibliografia".

PASSO 1 :: SCARICARE IL MATERIALE NECESSARIO

Ovviamente per prima cosa è necessario avere una copia funzionante di Thunderbird (e, per esperienza personale, è meglio che questa non sia la copia che viene utilizzata per lavoro!!), se possibile l'ultima versione.

!! Nota Pratica !!

Consiglio di utilizzare una "versione vergine" e inutilizzata per il semplice motivo che è molto semplice fare danni nella creazione delle estensioni, niente che non possa essere riparato ma che richiede una certa "esperienza": ad esempio potreste trovarvi tutto ad un tratto senza cartelle, e questo non è il massimo se sono cartelle di lavoro!!

Se si prevede di creare un sistema internazionalizzato per più lingue è altamente consigliato scaricare tutte le versioni di thunderbird che si intende usare.
Per poter creare i file XPI (i file che contengono le estensioni e non sono altro che semplici file ZIP) consiglio di utilizzare il comando JAR presente nell'installazione di Java (http://java.sun.com/) .

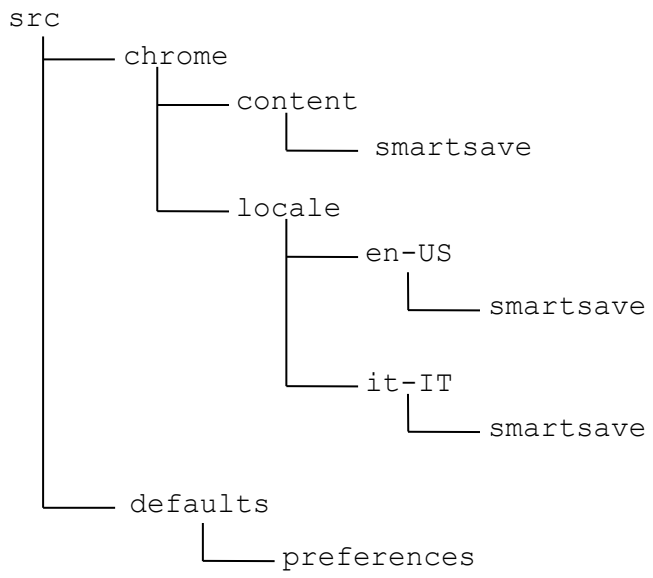
!! **Nota Pratica !!**

Dovremo usare in modo abbastanza pensante il comando jar quindi si consiglia, se già non lo si è fatto, di mettere nella vostra variabile d'ambiente PATH (o simili) il percorso del comando jar.

Un'ultima cosa da scaricare è un semplice editor di testo che evidenzia la sintassi. Uno abbastanza comodo è EditPlus 2 (http://www.editplus.com/) perché esistono delle estensioni che evidenziano la sintassi per file XUL e JavaScript, qualsiasi editor va comunque bene.
Non è necessario nient'altro: né compilatori, né linker, né interpreti di vario genere.

PASSO 2 :: COSTRUIRE LA STRUTTURA DELLE CARTELLE

Per prima cosa è necessario costruire la struttura delle cartelle. Supponendo che vogliate creare un'estensione chiamata *SmartSave* dovete creare una struttura delle cartelle come da figura.



La directory `src` è la cartella che contiene i sorgenti e sostanzialmente può avere un nome qualsiasi. È però comodo che si chiami in questo modo perché negli esempi e negli script successivi verrà utilizzata.

La directory `deafults/preferences` contiene delle impostazioni di preferenze del sistema. Non è necessariamente indispensabile ma è consigliata.

La directory `chrome/content/smartsave` contiene tutto il codice che effettivamente farà

funzionare l'estensione: all'interno di questa andranno posizionati i file XUL, JavaScript ed eventuali altri file ausiliari (immagini, suoni, ecc…).

La directory `chrome/locale/en-US/smartsave` contiene tutte i file che serviranno per la localizzazione delle stringhe in lingua inglese (in particolare relativamente agli stati uniti). Per contro i file nella directory `chrome/locale/it-IT/smartsave` sono realtivi a file scritti per la localizzazione italiana. Mozilla si preoccuperà in modo automatico di determinare quale sia la directory più opportuna, a noi basterà utilizzare il percorso relativo `chrome://smartsave/locale` !

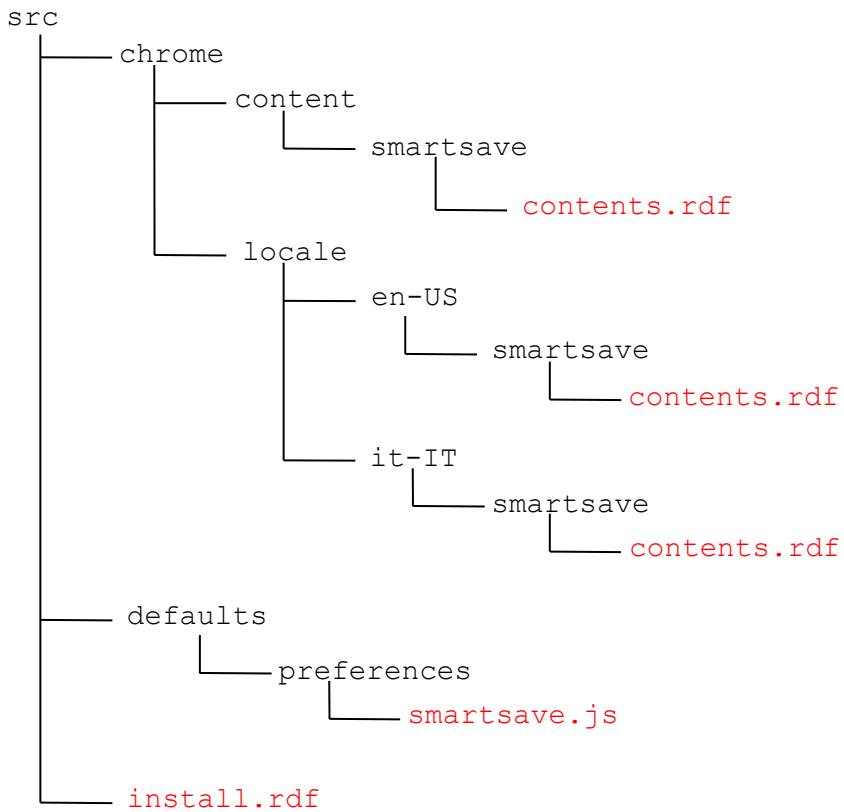
PASSO 3 :: INSERIRE I FILE INDISPENSABILI

A questo punto la successiva cosa da fare è quella di inserire tutti i minimi file necessari al funzionamento dell'estensione in particolare sono necessari un file che sovrintenderà all'installazione chiamato `install.rdf` da posizionare nella cartella `src`, un file `smartsave.js` da posizionare in `src\default\preferences\`, un file `contents.rdf` da posizionare in `src\chrome\content\smartsave`, in `src\chrome\locale\en-US\smartsave` e in `src\chrome\locale\it-IT\smartsave`.

!! **Nota Pratica !!**

I file contents.rdf pur avendo lo stesso nome hanno contenuto diverso!!! Quindi non fate l'errore di scriverne uno e copiarlo in tutti gli altri: la vostra estensione potrebbe venire installata ma non funzionerebbe come dovrebbe.

Praticamente la nuova strutture delle cartelle deve essere quella definita nel disegno seguente, dove in rosso sono evidenziati i file che devono essere aggiunti.



Vediamo adesso qual è il significato dei vari file e quale deve essere il loro contenuto.

FILE SRC\INSTALL.RDF

Questo file RDF permette a Mozilla di installare la vostra estensione. Esso contiene tutte le informazioni necessarie per individuare i file XUL che vi serviranno successivamente, la localizzazione, quali versioni di Thunderbird sono supportate ecc…
Il contenuto del file deve essere sostanzialmente il seguente:

```
<?xml version="1.0"?>
<RDF xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:em="http://www.mozilla.org/2004/em-rdf#">

    <Description about="urn:mozilla:install-manifest">

        <em:id>{031984cf-187c-40a2-bc8c-3ca45ccdb3e8}</em:id> 1
        <em:name>SmartSave Thunderbird Extension</em:name> 2
        <em:version>0.1.0</em:version> 3
        <em:description>My first Thunserbird Extension !! </em:description> 4
        <em:creator>Stefano Anelli</em:creator> 5
        <em:iconURL>chrome://smartsave/content/smartsave.gif</em:iconURL> 6


        <em:file>
            <Description about="urn:mozilla:extension:file:smartsave.jar">
                <em:package>content/smartsave/</em:package> 1
                <em:locale>locale/en-US/smartsave/</em:locale> 2
                <em:locale>locale/it-IT/smartsave/</em:locale>
            </Description>
        </em:file>

        <em:optionsURL>chrome://smartsave/content/settings.xul</em:optionsURL> 3

        <em:targetApplication>
            <Description>
                <em:id>{3550f703-e582-4d05-9a08-453d09bdfdc6}</em:id> 4
                <em:minVersion>0.8</em:minVersion> 7
                <em:maxVersion>1.5.0.*</em:maxVersion> 8
            </Description>
        </em:targetApplication>

    </Description>

</RDF>
```

Al momento alcuni particolari potrebbero sembrarvi oscuri, ma fidatevi, la cosa funziona e tutto vi sarà un po' più chiaro proseguendo con la lettura.

Il file ha sempre sostanzialmente la stessa struttura; in grassetto sono evidenziate le cose più importanti da modificare e in rosso i numeri delle linee. In particolare queste hanno il seguente significato:

1. In questo tag `<em:id>` è *necessario* inserire un ID valido per la propria estensione, l'ID deve avere la stessa struttura di quello mostrato (deve in particolar avere tutte le lineette (-) nel medesimo posto. È abbastanza probabile che Mozilla rifiuti il vostro ID le prime volte senza un motivo preciso, il mio consiglio è di partire da quello fornito come esempio e di modificarne alcuni valori poco per volta;

2. In questo tag è necessario inserire un nome per la vostra applicazione, non esiste nessuna particolare limitazione a questo campo ovviamente;

3. In questo campo dovete inserire la versione del vostro progetto; questo valore verrà mostrato dall'extension manager.

4. In questo campo inserite una breve descrizione di cosa è in grado di fare la vostra estensione; questo valore verrà mostrato dall'extension manager.

5. In questo campo dovete inserire il vostro nome, ovvero il nome dell'autore dell'estensione.

6. Questo tag, *opzionale*, permette di aggiungere un'icona alla vostra estensione che verrà visualizzata dall'extension manager e servirà a dare un tocco di stile alla vostra applicazione;

7. In questo campo dovete inserire qual è la versione minima di Thunderbid su cui la vostra estensione è in grado di girare; se siete incerti su quale valore dare lasciate il valore qui impostato;

8. A differenza del campo precedente in questo caso viene definita la versione massima di Thunderbird su cui può girare la vostra estensione. Il campo può contenere anche il carattere speciale `*` per indicare qualsiasi versione o sottoversione. Non è consigliato mettere un numero di versione troppo alto: potrebbero intervenire dei cambianti nel cuore di Mozilla e la vostra applicazione smettere di funzionare o funzionare in modo inaspettato e non corretto.

!! **Nota Pratica !!**

State attenti a modificare tutte le occorrenze di smartsave col nome del vostro progetto, ad esempio nel contenuto del tag `<em:optionsURL>` e nell'attributo `about` del tag `Description` all'interno del tag `<em:file>`.

Adesso analizziamo brevemente le altre linee di codice che sono un po' meno importanti ma comunque interessanti (in questo caso sono le linee con numerazione verde) :

1. In questa linea è necessario inserire il percorso relativo del contenuto della vostra applicazione. In linea di principio ha sempre la stessa struttura, ovviamente al posto di smartsave dovete mettere il nome del vostro progetto;

2. Questa riga, e in modo similare quella sottostante, definisce che esiste una localizzazione (in questo caso in la lingua inglese per gli stati uniti);

3. Questa riga definisce quale deve essere l'interfaccia grafica da lanciare quando, all'interno dell'estension manager, l'utente clicca sul bottone Opzioni.

4. Questa linea contiene l'ID dell'applicazione Thunderbird. Se volete creare un'estensione per Firefox o per la suite Mozilla dovete necessariamente modificare questi valori con i rispettivi ID.

FILE *SRC\CHROME\CONTETENT\SMARTSAVE\CONTENTS.RDF*

Questo file RDF ha lo scopo di definire in modo generale il contenuto dell'applicazione, e sostanzialmente è molto simile al file `install.rfd`, differisce sostanzialmente solo nella sintassi; il contenuto del file è il seguente:

```
<?xml version="1.0"?>
<RDF:RDF xmlns:RDF="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:chrome="http://www.mozilla.org/rdf/chrome#">

  <!-- list all the packages being supplied by this jar -->
  <RDF:Seq about="urn:mozilla:package:root">
    <RDF:li resource="urn:mozilla:package:smartsave"/>
  </RDF:Seq>

  <!-- package information -->
  <RDF:Description about="urn:mozilla:package:smartsave" (1)
        chrome:displayName="SmartSave Thunderbird Extension v0.1"
        chrome:author="Stefano Anelli"
        chrome:authorURL="http://www.mysite/mozilla"
        chrome:name="smartsave"
        chrome:description="Now you can save you messages in a smarter way!"
        chrome:extension="true"
       chrome:settingsURL="chrome://smartsave/content/settings.xul">
  </RDF:Description>

 <!-- overlay information -->
 <RDF:Seq about="urn:mozilla:overlays">
   <RDF:li resource="chrome://messenger/content/messenger.xul"/> (2)
 </RDF:Seq>

 <RDF:Seq about="chrome://messenger/content/messenger.xul">
  <RDF:li>chrome://smartsave/content/smartsave.xul</RDF:li> (3)
 </RDF:Seq>
</RDF:RDF>
```

Di questo file deve essere modificata solo la sezione marcata col numero (1). Questa sezione è molto simile a quella presente nel file precedente e gli attributi di `<RDF:Description>` sono abbastanza auto esplicativi e non hanno particolare impatto sul contenuto dell'estensione.

La linea marcata invece con (2) ha lo scopo di definire a quale applicazione fra Thunderbird, Firefox o Mozilla Suite noi vogliamo andare a fare delle aggiunte. In questo caso dichiariamo che vogliamo andare a fare delle aggiungere al messenger, e quindi a Thunderbird. L'indicazione effettiva di quale file verrà utilizzato per aggiungere componenti grafici a Thunderbird è definita invece nella sezione (3).

FILE *SRC\CHROME\LOCALE\EN-US\SMARTSAVE\CONTENTS.RDF*

Questo file RDF contiene informazioni circa la localizzazione dei file, in questo caso in lingua inglese. Il contenuto del file è il seguente:

```
<?xml version="1.0"?>
<RDF:RDF xmlns:RDF="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:chrome="http://www.mozilla.org/rdf/chrome#">

  <!-- list all the skins being supplied by this package -->
  <RDF:Seq about="urn:mozilla:locale:root">
    <RDF:li resource="urn:mozilla:locale:en-US"/>
  </RDF:Seq>
```

```
  <!-- locale information -->
  <RDF:Description about="urn:mozilla:locale:en-US">
    <chrome:packages>
      <RDF:Seq about="urn:mozilla:locale:en-US:packages">
        <RDF:li resource="urn:mozilla:locale:en-US:smartsave"/>
      </RDF:Seq>
    </chrome:packages>
  </RDF:Description>

  <!-- Version Information.  State that we work only with major version of
this
       package. -->
  <RDF:Description about="urn:mozilla:locale:en-US:smartsave"
   chrome:localeVersion="1.5a"/>
</RDF:RDF>
```

Il file non ha bisogno di particolari modifiche; è necessario solo sostituire SmartSave col nome del vostro progetto ed eventualmente la stringa en-US con la versione della localizzazione che state creando.

File src\default\preferences\smartsave.js

L'utilità di questo file è molto limitata, e molto probabilmente può anche non essere incluso nella vostra estensione. Comunque, metterlo non vi costa niente.
Il suo scopo è presumibilmente quello di settare una stringa nelle preferenze, una stringa che non ha nessun utilità pratica ai fini del progetto. Il contenuto del file è comunque questa unica linea di codice:

```
pref("extensions.{031984cf-187c-40a2-bc8c-3ca45ccdb3e}.description",
"chrome://smartsave/locale/smartsave.properties");
```

L'unica cosa da fare è modificare l'ID presente nella stringa con quello del vostro progetto, e ovviamente cambiare il nome smartsave con quello del vostro progetto.

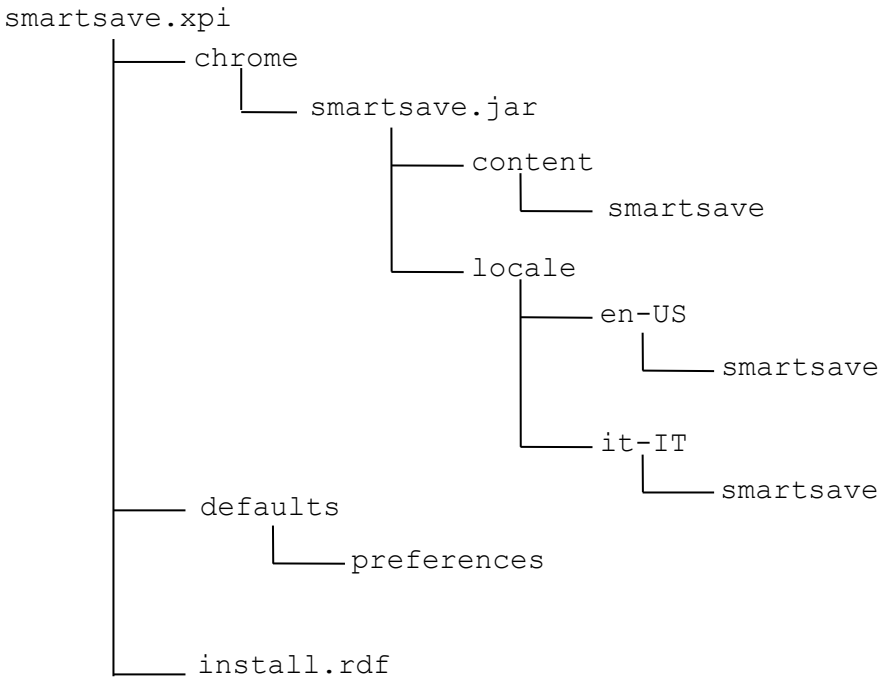
Passo 4 :: Costruire "il compilatore"

A questo punto la prossima cosa da fare è di costruire il compilatore. Spiegamoci meglio. Mozilla non ha bisogno che nessun codice venga compilato, ha bisogno però che il codice venga impacchettato all'interno di file ZIP con nomi particolari.

!! **Nota Pratica !!**

Questa operazione, per quanto possa sembrare semplice potrebbe rubarvi parecchio tempo per affinare la creazione dello script di compilazione; in particolare, state attenti al fatto che i file da creare sono file di tipo ZIP e quindi assolutamente **NON** RAR, pena… la vostra estensione non viene nemmeno installata perché ritenuta non valida !!

La struttura che deve essere ottenuta all'interno dei file ZIP è la seguente (sono visualizzate solamente le cartelle (ad eccezione del file `install.rdf`); il loro contenuto è quello specificato precedentemente e quello che creeremo nelle prossime sezioni):



Lo script di compilazione per Windows è il seguente (utenti di sistemi Unix-like o MacOS non dovrebbero avere difficoltà a tradurlo per la propria piattaforma) :

```
@echo off

echo Cancello i file che non mi servono più
del src\chrome\smartsave.jar (1)
del SmartSave.xpi (2)
del /S *.bak (3)
del /S /Q "C:\Documents and Settings\Stefano\Dati
applicazioni\Thunderbird\Profiles\8mz1qqvk.default\extensions\{031984cf-187c-
40a2-bc8c-3ca45ccdb3e8}" (4)
rmdir /S /Q "C:\Documents and Settings\Stefano\Dati
applicazioni\Thunderbird\Profiles\8mz1qqvk.default\extensions\{031984cf-187c-
40a2-bc8c-3ca45ccdb3e8}" (5)

echo Creo il file chrome\smartsave.jar
cd src\chrome
jar cvfM smartsave.jar -C ./fuffa -C ./content
jar uvfM smartsave.jar -C ./fuffa -C ./locale

cd..

echo Creo il file SmartSave.xpi

jar cvfM ..\SmartSave.xpi install.rdf -C ./fuffa -C ./chrome
jar uvfM ..\SmartSave.xpi -C ./fuffa -C ./defaults
```

Vediamo cosa fanno brevemente le linee e come possono o devono venire modificate.

1. Questa linea ha il compito di eliminare il file `src\chrome\smartsave.jar`; questo file è un prodotto delle vostre future "compilazioni" ed è importante che venga eliminato.

2. Questa linea ha il compito di eliminare l'archivio generale SmartSave.xpi, che, come per il precedente punto è il prodotto (finale) del vostro progetto.

3. Se usate editor come EditPlus vi verranno creati tanti file di backup .bak; non è strettamente necessario eliminarli, ma se non lo fate questi verranno inclusi nella vostra estensione occupando spazio inutilmente.

4. Questa linea elimina tutti i file del vostro progetto che sono rimasti dalla vostra installazione, dovete però necessariamente sostituire il percorso fornito al comando col vostro percorso. Come determinate il percorso? Facendo una semplice ricerca all'interno dell'area dove il sistema mette i vostri dati personali usando come parametro di ricerca l'ID della vostra applicazione.

5. La linea è molto simile alla precedente e anche qui dovete modificare il percorso. Il suo scopo è quello di eliminare la cartella che conteneva il vostro progetto sul vostro profilo.

Il resto del codice **NON** deve essere modificato in quanto crea in modo corretto la struttura del file `smartsave.xpi`. Le cartelle finte `./fuffa` servono giusto per fare in modo che il "compilatore funzioni".

In realtà il compilatore ha un baco; nel file `smartsave.xpi` vengono inserire anche le cartelle `chrome\content` e le cartelle `chrome\locale` anche se non sono necessarie. Una volta che la vostra estensione è pronta potete cancellare; non cancellatele tutte le volte: lasciarle li comporta solo un file `.xpi` più grande ma comunque funzionante.

!! **Nota Pratica !!**

L'operazione di compilazione può essere particolarmente pericolosa per la vostra installazione di Thunderbird; in particolare **NON** dovete mai eseguire lo script di compilazione prima di avere disinstallato l'estensione e riavviato nuovamente Thunderbird: se lo fate la vostra installazione andrà in crisi e potrebbe mostrarvi una interfaccia grafica incoerente. La soluzione è però semplice: reinstallate l'estensione !!

Passo 5 :: Scrivere il codice XUL per l'overlay

Se volete scrivere delle estensioni per Thunderbird è molto probabile che vogliate modificare l'interfaccia grafica dell'applicazione stessa, ad esempio volete aggiungere altri voci ai menu o ai menu contestuali che appaiono facendo un right-click su una mail, oppure aggiungere nuovi bottoni alle toolbar, …
Sostanzialmente quello che avete intenzione di fare è chiamato nel gergo di XUL un overlay. Esistono vari modi di fare un overlay, noi vediamo quello più semplice. Per prima cosa dovete creare il file `src\chrome\content\smartsave.xul`.

!! **Nota Pratica !!**

Il file che qui descriviamo può avere in realtà un nome qualsiasi, se lo chiamate in modo diverso dovete cambiare la linea segnata col numero tre in verde nel file `install.rdf` col nuovo nome che gli avete assegnato.

Il contenuto *base* del file deve essere il seguente:

```
<?xml version="1.0"?>
```

```
<!-- Aggiunta di eventuali DTD -->

<!-- DEFINIZIONE DELL'OVERLAY -->
<overlay id="smartsaveOverlay"
      xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">

      <!-- Aggiunta di eventuali script -->


     <!-- Contenuto dell'overlay -->

</overlay>
```

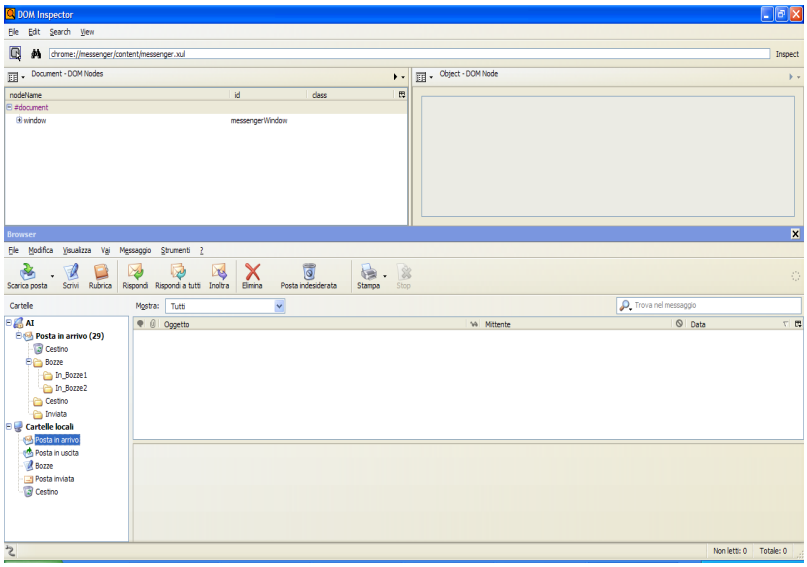
Nel file l'unica cosa da modificare è l'attributo ID dell'overlay, modificandolo con un altro "nome di fantasia".

A questo punto viene la parte più difficile: inserire qualcosa all'interno dell'overlay. Questa guida non copre XUL, ma spiega brevemente come inserire alcuni contenuti comuni a molte applicazioni.

Indicazioni Generali

La prima cosa da fare per scrivere codice XUL è… conoscere qualche nozione di XUL in generale, almeno sapere come funziona e quali sono le nozioni fondamentali. Nel libro "Rapid Application Developmen for Mozilla" (vedi la bibliografia per sapere dove poterlo scaricare gratuitamente) si trova un'estesa descrizione di tutti i componenti (finestre, menu, popuop, bottoni, label, …) che può essere (ed è) molto utile.

Dopo aver capito il funzionamento di XUL avrete capito che ogni componente può avere associato un ID univoco; questo vale anche per l'interfaccia grafica di Thunderbird che è stata scritta anch'essa in XUL. A questo punto sorge il problema di come determinare questi ID che ci serviranno come ancore per costruire la nostra interfaccia di overlay. Per farlo basta usare il DOM inspector per Thunderbird (attualmente scaricabile da https://addons.mozilla.org/firefox/1806/); questa estensione per thunderbird è abbastanza semplice da utilizzare e permette di visualizzare la struttura DOM del documento. Il documento che a noi interessa è: `chrome://messenger/content/messenger.xul` . Qui, con un po' di pazienza possiamo trovare tutti gli ID di tutti i componenti di Thunderbird.



AGGIUNGERE UNA VOCE DI MENU AL MENU FILE E TOOLS

In questa sezione vediamo mediante un esempio come aggiungere una voce di menu al menu File. Il codice qui riportato deve essere inserito all'interno del tag overlay:

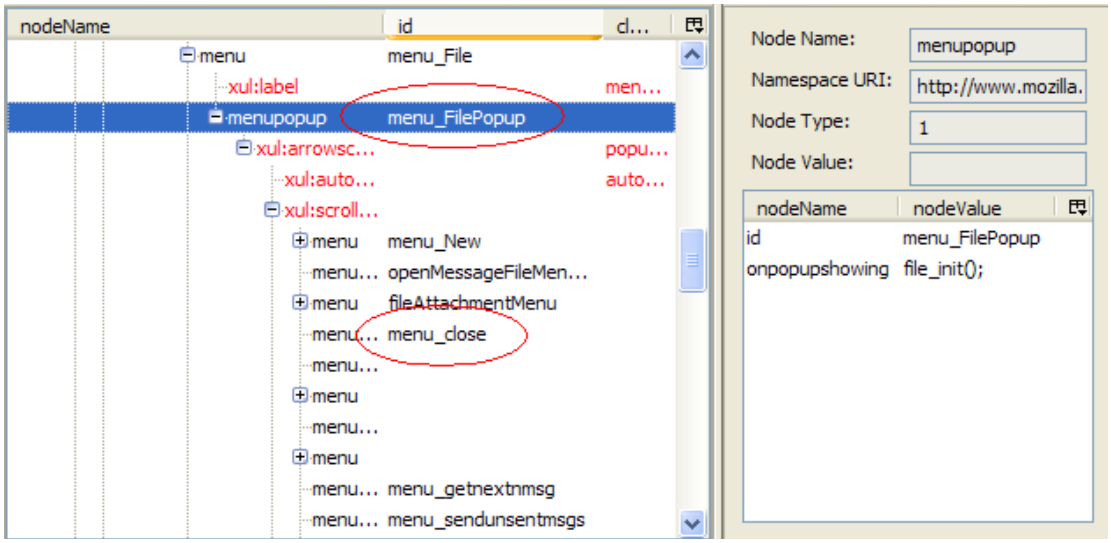
```
<menupopup id="menu_FilePopup">

     <menuitem id="menu_MyOwnMenuOnFile"
           label="Open my own window!"
           insertafter="menu_close"
           oncommand="DoSomething();">

      <menuseparator insertafter="menu_close" />

</menupopup>
```

Analizziamo adesso brevemente il codice. La prima riga ci permette di agganciarci al popup menu del menu file; in questo modo tutto quello che inseriremo all'interno verrà agganciato dove abbiamo deciso. A questo punto infatti inseriamo un primo tag `menuitem` che ci permette di aggiungere una nuova voce di menu con label `Do Something` e id `menu_MyOwnMenuOnFile`. L'ID può ovviamente essere una stringa qualsiasi ma consiglio di inserire un nome sensato per poter usare semplicemente il menu che abbiamo creato. La cosa più interessante da notare è l'attributo `insertafter` del tag `menuitem`. Questo attributo indica dove inserire il nuovo elemento: in questo esempio diciamo di inserirlo dopo il menu close, e glielo diciamo fornendo come valore di `insertafter` proprio l'ID del menu close (id che può essere ricavato sempre mediante l'utilizzo del DOM inspector che abbiamo scaricato ed installato precedentemente).



L'attributo `oncommand` (sempre del tag menuitem) indica quale azione deve essere eseguita dal codice JavaScript (nell'esempio viene richiamata la funzione `DoSomething()`)che andremo a scrivere in seguito. Come noterete infatti cliccando sulla voce di menu non succede nulla, o meglio non succede nulla di visibile, in realtà, se andata al menu "JavaScript Console..." nel menu "Tools" vedrete c'è stato generato un errore perché non viene trovata la funzione `doSomething()`. Quando aggiungeremo il codice questo errore ovviamente non ci sarà più.

Dopo aver aggiunto il `menuitem` il codice aggiunge un `menuseparator`, sempre dopo il menu close.

A questo punto se provate a compilare il codice, ad installarlo e a lanciare Thunderbird troverete una nuova voce di menu proprio come descritto precedentemente (sempre che non ci siano errori all'interno del vostro codice).

Adesso facciamo un altro esempio di aggiunta al menu file; in questo caso aggiungeremo non una singola voce di menu ma un intero sotto-menu. Il codice da scrivere è il seguente e inserito all'interno del tag `<menupopup id="menu_FilePopup">`:

```
<menu id="menu_MySubMenu"
      label="This is my submenu"
      insertafter="menu_close">

      <menupopup id="menu_MuSubMEnuPopup">

            <menuitem id="menu_FirstSubmenu"
                  label="Submenu One"
                  oncommand="runSubMenu('one');" />


            <menuitem id="menu_SecondSubmenu"
                  label="Submenu Two"
                  oncommand="runSubMenu('two');"/>


            <menuitem id="menu_ThirdSubmenu"
                  label="Submenu Three"
                  oncommand="anotherAction();"/>

      </menupopup>

</menu>
```

Non sono necessarie particolari spiegazioni riguardo il codice in quanto è molto simile al precedente con pochissime aggiunte, consiglio comunque in caso di dubbi di leggere prima di tutto un buon manuale XUL.

AGGIUNGERE UNA VOCE DI MENU CONTESTUALE A MESSAGGI E FOLDER

Per aggiungere un menu al menu contestuale dei messaggi è sufficiente aggiungere la seguente parte di codice alla sezione overlay:

```
<menupopup id="threadPaneContext">

      <menuitem id="context_MyContextInMails"
            label="Il mio menu contesuale sulle e-mail !!"
            oncommand="doSomethingWithThisMail();"/>

      <menuseparator />

</menupopup>
```

Come sempre l'unica difficoltà è conoscere l'ID del menu contestuale che in questo caso è `threadPaneContext`. L'unica differenza col resto del codice precedente è che non viene inserito l'attributo `insertafter`: in questo caso sia il `menuitem` sia il `menuseparator` vengono automaticamente aggiunti all'inizio del menu di popup.

Per aggiungere un menuitem al menu contestuale che appare facendo click destro sul pannello che contiene le cartelle potete utilizzare lo stesso codice precedente, con l'unica differenza è che dovete modificare l'ID precedente con `folderPaneContext`.

AGGIUNGERE UN BOTTONE ALLA TOOLBAR

Per aggiungere un bottone alla toolbar è sufficiente aggiungere il seguente codice:

```
     <toolbar id="mail-bar">

           <toolbarseparator/>

           <toolbarbutton id="toolbar_MyToolbarButton"
                 class="toolbarbutton-1"
                 tooltiptext="Questo è il tooltip del bottone !!"
                 image="chrome://smartsave/content/smartsave.gif"
                 label="Cliccami!!"
                 orient="vertical"
                 oncommand="MyButtonClick();" />

     </toolbar>
```

Il codice non ha bisogno di particolari spiegazioni a parte l'attributo `tooltiptext` che non fa altro che mostrare il tooltip del bottone, `image` che carica l'immagine tramite l'url.

Passo 6 :: Scrivere il codice XUL per l'interfaccia grafica

A questo punto è molto probabile che vogliate aggiungere alla vostra interfaccia grafica qualche finestra; adesso creeremo la finestra delle opzione della vostra estensione ma lo stesso principio può essere utilizzato per creare altre finestre. Per prima cosa dovete creare il file `src/chrome/content/smartsave/settings.xul`, e se notate è lo stesso nome inserito nel file `install.rdf`. Il codice da inserire per creare una finestra è il seguente:

```
<?xml version="1.0"?>

<!-- Carico gli stili da utilizzare-->
<?xml-stylesheet href="chrome://communicator/skin/" type="text/css"?>
<?xml-stylesheet href="chrome://messenger/skin/prefPanels.css"
                 type="text/css"?>


<dialog xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"
           buttons="accept, cancel"
           title="Titolo della finestra"
           id="smartsave-settings"
           orient="vertical" flex="0"
           persist="screenX screenY width height"
           onload="init();"
           ondialogaccept="onAcceptButton();">

     <label value="Ciao, questa è la mia finestra !!" />

</dialog>
```

La finestra (che in questo caso è in realtà una semplice finestra di dialogo) contiene in questo caso solo un semplice label, ma in realtà potete inserire qualsiasi cosa. Vediamo adesso brevemente il significato degli attributi più importanti:

- `title`: è il titolo della finestra;
- `buttons`: è una lista di tipi di bottoni che verranno aggiunti alla vostra finestra e che potete controllare, in questo caso noi controlleremo solamente il bottone di accept (OK);
- `persist`: una lista di attributi che devono rimanere validi tra una visualizzaione e l'altra della finestra stessa;
- `onload`: La funzione che deve essere esguita quando la finestra viene creata;

- `ondialogaccept`: La funzione da lanciare quando l'utente clicca sul bottone automatico "OK";

PASSO 7 :: SCRIVERE JAVASCRIPT PER L'OVERLAY

Per prima cosa dovere creare il file `src/chrome/content/smartsave/smartsave.js`, Inoltre dovete aggiungere la seguente linea di codice al file `smartsave.xul` all'interno del tag overlay:

```
<script type="application/x-javascript"
     src="chrome://smartsave/content/smartsave.js" />
```

Il codice da inserire nel file javascript non è particolarmente complicato dovete solo inserire le funzioni che avete dichiarato di usare nel file `smartsave.xul`:

```
function DoSomething(){
     alert("I've done something!!")
}

function runSubMenu(value){
     alert("You've chosen "+value);
}

function anotherAction(){
     alert("You've chosen third");
}

function doSomethingWithThisMail(){
     alert("You have selected some mails!!");
}

function MyButtonClick(){
     alert("You've clicked on the toolbar button!!");

     window.openDialog("chrome://smartsave/content/settings.xul");
}
```

Come già detto il codice che inserite nelle funzioni può fare qualcosa di più complesso, ad esempio la funzione `doSomethingWithThisMail` dovrebbe prelevare l'email selezionata e poi farne qualcosa. Per questi "trucchetti" e il funzionamento degli oggetti XPCom date un'occhiata alla sezione "Tips and Tricks per XPCom". L'unica cosa interessante è l'istruzione `window.openDialog()` presente nella funzione `MyButtonClick`: questo comando apre la finestra descritta dal file XUL passato come parametro sottoforma di URL.

PASSO 8 :: SCRIVERE JAVASCRIPT PER L'INTERFACCIA GRAFICA

Scrivere codice script per l'interfaccia è sostanzialmente la stessa cosa che farlo per l'overlay; in questo caso dovete creare il file `src/chrome/content/smartsave/settings.js` e aggiungere la seguente lina di codice al file `settings.js` all'interno del tag `dialog`:

```
<script type="application/x-javascript"
     src="chrome://smartsave/content/settings.js" />
```

A questo punto nel file potete inserire le seguenti linee di codice:

```
function oninit(){
     alert("Inizializzazione della finestra!!");
}

function ondialogaccept(){
     alert("You've clicked on the toolbar button!!");
```

```
}
```

Come per il passo pretendete le funzioni dovrebbero essere più intelligenti a seconda di quello che la vostra estensione deve fare.

PASSO 9 :: INTERNAZIONALIZZAZIONE

Molto probabilmente se volete distribuire la vostra estensione volete che il testo possa essere compatile con quello della versione di Thunderbird in uso dall'utente, ad esempio sa l'utente ha un versione inglese di Thunderbird vorrebbe leggere tutto il tetso in inglese, mentre se ha una versione italiana vorrebbe vedere le stringhe in italiano. In Mozilla questo è possibile in modo abbastanza semplice, anche se bisogna utilizzare due diverse tecniche per i file XUL e per i file JavaScript. Supponiamo in questo esempio che vogliate fare le versioni inglese e italiano.

INTERNAZIONALIZZAZIONE PER XUL

Per prima cosa dovete creare i seguenti due file: `src\chrome\locale\en-US\smartsave\smartsave.dtd` e `src\chrome\locale\it-IT\smartsave\samrtsave.dtd`. Nel primo file inseriremo le stringhe in inglese nel secondo quelle in italiano. A questo punto il contenuto dei file è il seguente:

`smartsave.dtd` (in en-US)

```
<!ENTITY smartsave.mystring1       "This is the localised string one!">
<!ENTITY smartsave.mystring2       "This is the localised string two!">
<!ENTITY smartsave.mystring3       "This is the localised string three!">
<!ENTITY smartsave.mystring4       "This is the localised string four!">
<!ENTITY smartsave.mystring5       "This is the localised string file!">
```

`smartsave.dtd` (in it-IT)

```
<!ENTITY smartsave.mystring1       "Stringa localizzata uno!">
<!ENTITY smartsave.mystring2       "Stringa localizzata due!">
<!ENTITY smartsave.mystring3       "Stringa localizzata tre!">
<!ENTITY smartsave.mystring4       "Stringa localizzata quattro!">
<!ENTITY smartsave.mystring5       "Stringa localizzata cinque!">
```

Adesso abbiamo due file localizzati. A questo non ci resta che inserire nei file XUL una delle seguenti due stringhe (prima del tag di root che nei nostri esempi era `dialog` oppure `overlay`) a seconda del tag di root che avete; per l'overlay:

```
<!DOCTYPE overlay SYSTEM "chrome://smartsave/locale/smartsave.dtd">
```

mentre per le finestre semplici o quelle di dialogo:

```
<!DOCTYPE window SYSTEM "chrome://smartsave/locale/smartsave.dtd">
```

A questo punto per inserire la stringa localizzata dovete semplicemente inserire (ad esempio al posto del valore del label il valore "`&smartsave.mystring2;`". Ad esempio il menu contestuale delle e-mail presente nel file `smartsave.xul` possiamo riscriverlo nel seguente modo (i cambiamenti sono evidenziati in grassetto):

```
<menupopup id="threadPaneContext">
     <menuitem id="context_MyContextInMails"
           label="&smartsave.mystring2;"
           oncommand="doSomethingWithThisMail();"/>
     <menuseparator />
</menupopup>
```

INTERNAZIONALIZZAZIONE PER JAVASCRIPT

Per i file JavaScript la cosa è un po' più complessa ed è necessario scrivere del codice in più ed utilizzare dei file di properties. Questi file hanno una struttura del tipo `chiave=valore`.
Dobbiamo creare prima di tutto i seguenti due file `src\chrome\locale\en-US\smartsave\smartsave.properties` (per l'inglese) e `src\chrome\locale\it-IT\smartsave\smartsave.properties` (per l'italiano). Il loro contenuto deve essere il seguente:

`smartsave.properties` (in en-US)

```
smartesave.keyone=This is the first key!
smartesave.keytwo=This is the second key!
smartesave.keyone=This is the third key!
```

`smartsave.properties` (in it-IT)

```
smartesave.keyone=Questa è la prima chiave!
smartesave.keytwo=Questa è la seconda chiave!
smartesave.keyone=Questa è la terza chiave!
```

A questo punto non ci resta che aggiungere la seguente funzione ai file JavaScript che necessitano della localizzazione:

```
function getString(ident){
     var strService =
Components.classes["@mozilla.org/intl/stringbundle;1"].getService(Components.i
nterfaces.nsIStringBundleService);
     var bundle =
strService.createBundle("chrome://smartsave/locale/smartsave.properties");
     return bundle.GetStringFromName(ident);
}
```

Questa funzione prende come parametro la chiave e ritorna il valore presente nel file di properties. Il file di properties è definito tramite l'URL in grassetto.

!! **Nota Pratica !!**

In questo esempio abbiamo chiamato il file `smartsave.properties` ma può aver un nome qualsiasi: l'importante è che modifichiate in modo corretto la funzione `getString` in modo che l'URL alla riga in grassetto punti al vostro file.

Vediamo adesso come utilizzare la funzione in un file JavaScript, ad esempio potremmo modificare la funzione `ondialogaccept` presente nel file `settings.js` nel seguente modo:

```
function oninit(){
     alert(getString("smartsave.keytwo"));
}

function ondialogaccept(){
     alert(getString("smartsave.keyone"));
}

function getString(ident){
     var strService =
Components.classes["@mozilla.org/intl/stringbundle;1"].getService(Components.i
nterfaces.nsIStringBundleService);
     var bundle =
strService.createBundle("chrome://smartsave/locale/smartsave.properties");
```

```
      return bundle.GetStringFromName(ident);
}
```

A questo punto siete pronti per scrivere la vostra estensione: buona fortuna!!

Tips and Tricks per XPCom

Abbiamo già avuto modo di utilizzare XPCom per riuscire ad ottenere la localizzazione all'interno degli script senza però spiegare effettivamente in modo approfondito il come e il perché di questa tecnologia. In questa sezione analizzeremo in modo abbastanza sommario il funzionamento di XPCom e verranno forniti altri esempi pratici del suo utilizzo.

Le basi di XPCom

La prima cosa importante da tenere a mente è che gli oggetti XPCom sono effettivamente oggetti nel senso fornito dal paradigma Object-Oriented. Se vogliamo fare un paragone questi oggetti hanno molto in comune con gli oggetti scritti in Java: esattamente come in questo linguaggio esistono oggetti ed esistono interfacce. La differenza sta soprattutto nel modo in cui questi oggetti vengono costruiti e nel modo di funzionamento delle interfacce. Spieghiamoci meglio. Per costruire un oggetto dovete usare la seguente linea di codice:

```
var obj = Components.classes["<Stringa identificativa oggetto>"]
```

In realtà con questo oggetto potete farci ben poco; per poterlo utilizzare dovete per prima cosa costruine un'istanza oppure ottenerne un servizio a seconda del tipo di oggetto (come regola generale se la stringa identificativa dell'oggetto contiene la parola "Service" allora dovete ottenerne un servizio); per fare questo dovete invocare o il metodo `createInstance` oppure `getService` e ad entrambi dovete passare un riferimento ad un'interfaccia, questo perché nel vostro codice potete richiamare dei metodi su un oggetto se e solo se avete importato l'interfaccia che vi interessa. Questa interfaccia può essere "aggiunta" all'oggetto durante la creazione, mediante i `createInstance` o `getService` come detto in precedenza, oppure mediante il metodo `QueryInterface` a cui dovete passare il riferimento all'interfaccia. Vengono di seguito riportati alcuni esempi di codice che possono essere utili spunti per lo sviluppo tramite XPCom e JavaScript.

Dove trovare l'oggetto XPCom necessario

Spesso c'è la necessità di avere a disposizione alcune funzioni ma non sappiamo dove andare a prendere gli oggetti che le mettono a disposizione. Il mio consiglio è quello di andare a spulciare il codice di altre estensioni che fanno quello che interessa a voi, oppure di andare sul sito http://www.xulplanet.org: esiste una sezione completa di tutti gli oggetti XPCom presenti in Mozilla divisi per categoria.

Come determinare quali interfacce possiede un oggetto a Runtime

A volte potrebbe capitarvi di ritrovarvi nella situazione che abbiate un oggetto ma non riusciate a a capire quali sono le interfacce che questo possiede; questa situazione si presenta ad esempio quando ottenere un'interfaccia di tipo `nsIEnumerator`: questa fornisce un metodo `currentItem` che ritorna l'elemento corrente, il problema è che questo ritorna oggetti con interfaccia `nsISupport` che è l'interfaccia radice (è un po' come il tipo `Object` in Java).
Per riuscire a determinare le interfacce che questo supporta potete inserire nel vostro codice la seguente linea di codice (supponendo che `oggetto` sia l'oggetto XPCom di cui volete conoscere le interfacce implementate):

```
alert(oggetto);
```

In questo modo a video vi comparirà un messaggio il cui contenuto è una lista contente le interfacce possedute dall'oggetto.

MODIFICARE XUL DA JAVASCRIPT CONOSCENDONE L'ID

Questo "trucco" è inserito per completezza e perché spesso nel vostro codice vi capiterà di dover utilizzare oggetti scritti in XUL. Per ottenere l'oggetto XUL (che ha come attributi gli stessi attributi definiti nel codice XUL) conoscendone l'ID potete usare il seguente codice:

```
function getXULElement(idString){
     return document.getElementById(idString);
}
```

Riprendendo gli esempi precedenti potremmo scrivere questa riga di codice per modificare il testo del label che è presente nella voce di menu che abbiamo aggiunto al menu file:

```
getXULElement("menu_MyOwnMenuOnFile").label = "Il nuovo valore è questo!";
```

GESTIONE DELLE PREFERENZE

In questa sezione vedremo come poter settare delle impostazioni personali che Mozilla automaticamente ricorderà nel suo registro interno. Il meccanismo è abbastanza semplice. Per memorizzare una preferenza è sufficiente scrivere la seguenti linee di codice:

```
var pref = Components.classes["@mozilla.org/preferences-
service;1"].getService(Components.interfaces.nsIPrefBranch);

pref.setCharPref("smartsave.mycharpreference","MyString");
```

La prima riga richiama il componente XPCom opportuno, in particolare richiama l'oggetto `@mozilla.org/preferences-service;1` e poi come servizio viene richiamata l'interfaccia `nsIPrefBranch`.
La seconda linea invece permette di settare una preferenza di tipo stringa richiamando il metodo `setCharPref`. È possibile settare altri tipi di preferenze anche di tipo più complesso di stringhe come, ad esempio, oggetti. Per una lista completa rimando alla guida XPCom presente sul sito http://www.xulplanet.org/.

!! **Nota Pratica !!**

La chiave che qui abbiamo chiamato `smartsave.mycharpreference` può essere una stringa di testo qualsiasi, ma io consiglio si farla iniziare col nome del vostro progetto in modo che non vada ad interferire con chaivi di altri progetti.

Per ottenere invece il valore di una preferenza precedentemente settata, supponenedo che abbiate un oggetto chiamato pref definito come nel pezzo di codice precedente, potete utilizzare le seguenti linee di codice:

```
var myvalue = "";
try{
     myvalue = pref.getCharPref("smartsave.mycharpreference");
}catch(e){
     myvalue = "defaultvalue";
}
```

Il codice è abbastanza semplice: mediante `getCharPref` (il corrispettivo get del metodo presente nel listato precedente) è possibile ottenere la stringa che ha come chiave `smartsave.mycharpreferenze`; in questo caso `myvalue` avrebbe come valore il valore settato precedentemente (nell'esempio la stringa `"MyString"`). Il blocco `try/catch` è indispensabile

quando non si è certi che la preferenza che si vuole caricare sia già stata settata, ad esempio quando l'estensione viene eseguita per la prima volta e si vuole che parta con valori di default. Per questo motivo, proprio nella sezione catch viene inserita la stringa di default `"dafultvalue"`.

CARTELLE E MESSAGGI SELEZIONATI

In questa sezione vedremo come ottenere la lista dei messaggi selezionati o la cartella selezionata. Questa sezione potrebbe sembrare una banalità ma in realtà non è così semplice individuare questi valori, per questo è meglio introdurla. Per ottenere la lista dei messaggi selezionati potete utilizzare il seguente codice:

```
var dbv = GetDBView();
var messageArray = {}
var length = {}

var msgArray = dbv.getURIsForSelection(messageArray,length);

if(length.value == 0){
     return null;
}

msgArray = messageArray.value;
```

Mediante questo codice che potrebbe a prima vista sembrare un po' criptico, nella variabile msgArray otteniamo un array di URI che identificano univocamente ciascun messaggio selezionato. L'array è un semplice array di stringhe, per riuscire a leggere il contenuto della mail bisogna fare qualche passaggio in più che è spiegato più avanti nella sezione "Salvare una mail in locale".

Per ottenere la cartella selezionata, un oggetto con interfaccia `nsIMsgFolder` che rappresenta la cartella selezionata, potete utilizzare il seguente codice abbastanza simile al precedente:

```
var dbv = GetDBView();
folder = dbv.msgFolder;
```

L'oggetto `folder` è un oggetto di tipo `nsIMsgFolder` e come tale ha vari metodi ed attributi, ad esempio per sapere se il server è di tipo IMAP basta fare il seguente controllo

```
if(folder.server.type == 'imap'){ ... }
```

che ci servirà successivamente per scaricare i messaggi appunto da un server IMAP.

MESSAGGI CONTENUTI IN UNA CARTELLA

In questa sezione vedremo come, data una cartella locale o meglio un oggetto XPCom di tipo `nsIMsgFolder`, ottenere le cartelle e i messaggi contenuti.
Per ottenere la lista delle cartelle contenute potete utilizzare il seguente codice, supponendo che abbiate una variabile chiamato `folder` di tipo `nsIMsgFolder`:

```
var subs = folder.GetSubFolders();
subs.first();
try{
     do{
          var item = subs.currentItem();
          item.QueryInterface(Components.interfaces.nsIMsgFolder);

          DoSomethingWithGivenFolder();

          subs.next();

     }while(true);
}catch(e){}
```

Solo due note sul codice; il ciclo `while` è di tipo infinito in quanto quando viene richiamato il metodo `subs.currentItem()` questo lancerà un'eccezzione, ed ecco spigato anche la necessità di avere un `try/catch` esterno al ciclo. La funzione `DoSomethingWithGivenFolder()` ha invece la funzione di segnaposto, qui potente inserire quello deve fare il vostro codice con la sotto cartella selezionata presente nella variabile `item` di tipo `nsIMsgFolder`.

Supponendo che abbiate sempre una variabile `folder` di tipo `nsIMsgFolder` il codice per ottenere i messaggi presenti in una cartella è il seguente:

```
var msgs = fol.getMessages(msgWindow);
while(msgs.hasMoreElements()){
     var msg = msgs.getNext();
     msg.QueryInterface(Components.interfaces.nsIMsgDBHdr);

     var id = msg.threadId;
     var msgUri = uri+"/#"+id;

     DoSomethingWithTheMessage();
}
```

Il codice è abbastanza semplice, solo un paio di osservazioni. La variabile `msg` è di tipo `nsIMsgDBHdr` cioè descrive l'header di un messaggio e di per se non è utilissima ma la possiamo utilizzare per ottenere l'uri del messaggio, infatti supponendo che la variabile `uri` sia l'uri della cartella che contiene il messaggio in `msgUri` viene inserito proprio l'uri del messaggio. La funzione `DoSomethingWithTheMessage()` è anch'essa un segnaposto, al posto di questa potete inserire il vostro codice. La variabile `msgWindow` è una "variabile globale" di Mozilla di tipo `nsIMsgWindow` che a volte è richiesta come parametro di alcune funzioni, in questo caso è passata come parametro a `getMessages`.

SALVARE UNA MAIL IN LOCALE CONOSCENDONE L'URI

In questa sezione vedremo come data una mail, o meglio un oggetto nsIMsgHeader, questa possa essere salvata in locale. Il codice che ci interessa è la funzione descritta di seguito che prende tre parametri, il primo (`folder`) è una stringa che contiene la directory di destinazione del messaggio in locale, il secondo (`uri`) è ancora una stringa che contiene l'uri del messaggio che ci interessa, infine il terzo (`filename`) è ancora una stringa che contiene il nome del file che conterrà il messaggio. Il codice è il seguente:

```
function saveMsg(folder,uri, filename){

     var header =
messenger.messageServiceFromURI(uri).messageURIToMsgHdr(uri);

     filename =  folder+getFileSeparator()+filename;

     var file =
Components.classes["@mozilla.org/filespec;1"].createInstance(Components.interf
aces.nsIFileSpec);
     file.unicodePath = filename;

     if(header.folder.server.type == 'imap'){
           saveIMAPMessage(header, file, uri)
           return;
     }

     var inFile =
Components.classes["@mozilla.org/file/local;1"].createInstance(Components.inte
rfaces.nsILocalFile);
```

```
     inFile.initWithPath(header.folder.path.nativePath);

    var fileInputStream =  Components.classes["@mozilla.org/network/file-
input-stream;1"].createInstance(Components.interfaces.nsIFileInputStream);

     fileInputStream.init( inFile, 0x01, 0444, null );

    fileInputStream.QueryInterface(Components.interfaces.nsISeekableStream);

    fileInputStream.seek(0,header.messageOffset);

    var fileScriptableIO =
Components.classes["@mozilla.org/scriptableinputstream;1"].createInstance(Comp
onents.interfaces.nsIScriptableInputStream);

    fileScriptableIO.init(fileInputStream);

    var fileContent = fileScriptableIO.read( header.messageSize );

     fileScriptableIO.close();
     fileInputStream.close();


     var pref = Components.classes["@mozilla.org/preferences-
service;1"].getService(Components.interfaces.nsIPrefBranch);

     var outFile =
Components.classes["@mozilla.org/file/local;1"].createInstance(Components.inte
rfaces.nsILocalFile);

     outFile.QueryInterface(Components.interfaces.nsIFile);

     outFile.initWithPath(file.unicodePath);

     if(outFile.exists()) {
           try{
                 outFile.remove(false);
           }catch(e){}
     }


     var fileOutputStream = Components.classes['@mozilla.org/network/file-
output-stream;1'].createInstance(Components.interfaces.nsIFileOutputStream);
     try{
           outFile.create(outFile.NORMAL_FILE_TYPE, 0666);
           }catch(e){}

     fileOutputStream.init(outFile, 2, 0x200, false); /

           fileOutputStream.write(fileContent, fileContent.length);
           fileOutputStream.close();

     }

}
```

MESSAGGI SU IMAP

In questa sezione vedremo come riuscire a leggere quali messaggi contiene una cartella IMAP, in quanto questo procedimento è diverso per quello delle cartelle residenti in locale descritto nella sezione precedente. Supponendo che sappiate che una cartella risieda su IMAP per prima cosa

dovete fare un download dei messaggi per utilizzarli in modalità offline; senza questa operazione non riescireste nemmeno ad avere una lista dei messaggi presenti nella cartella. Per farlo potete utilizzare il seguente codice, supponendo che abbiate una variabile `folder` di tipo `nsIMsgFolder`:

```
folder.clearNewMessages()

listener = myUrlListener();
listener.fol = folder;
listener.exportfolder = exportfolder;
listener.uri = uri;

folder.downloadAllForOffline(listener, msgWindow);
```

La funzione `myUrlListener` è una funzione che mi ritorna un oggetto di tipo `nsIUrlListener`; il suo listato è il seguente:

```
function myUrlListener(){
     var listener = {
           fol : null,
           exportfolder : null,
           uri : null,

           QueryInterface : function(iid)  {
                 if (iid.equals(Components.interfaces.nsIUrlListener) ||
                       iid.equals(Components.interfaces.nsISupports))
                  return this;
                 throw Components.results.NS_NOINTERFACE;
                 return 0;
           },

           OnStartRunningUrl: function ( url )
           {
           },

           OnStopRunningUrl: function ( url, exitCode )
           {
                 try{

                       var msgs = this.fol.getMessages(msgWindow);
                       while(msgs.hasMoreElements()){

                             var msg = msgs.getNext();
                       msg.QueryInterface(Components.interfaces.nsIMsgDBHdr);
                             var id = msg.messageKey;
                             var msgUri =
this.uri.slice(0,this.uri.length)+"#"+id;
                             DoSomethingWithTheMessage();

                       }
                 }catch(e){
                       alert("Some error!!");
                 }
           }
     }
     return listener;
}
```

Il contenuto della funzione in questo caso è stato scritto in modo tale che quando tutti i messaggi sono stati completamente scaricati venga eseguito qualcosa (in particolare la funzione `DoSomethingWithTheMessage()`); il codice per ottenere i messaggi è molto simile al precedente solo che la struttura dell'url che creiamo è un po' diversa, in particolare viene agganciata la stringa

"#" e non "/#", anzi, viene addirittura tolto l'ultimo carattere / dell'url della cartella che contiene il messaggio.

Adesso vedremo come salvare effettivamente in locale un messaggio presente su un server IMAP, in quanto col procedimento precedente abbiamo letto i file presenti in una cartella, per poterli salvare dobbiamo usare il seguente codice:

```
     var mms =
messenger.messageServiceFromURI(uri).QueryInterface(Components.interfaces.nsIM
sgMessageService);
     var listener = myStreamListener();
     listener.file = file;
     listener.callback = realWriteIMAPMessage;
     var duri = new Object();
     mms.DisplayMessage(uri, listener, null, null, null, duri);
```

Nel codice la variabile `uri` è l'uri dell'email ottenuto col metodo precedente oppure come spiegato nella sezione "Cartelle e messaggi selezionati"; la variabile `file` è un oggetti con interfaccia di tipo `nsIFileSpec` che identifica il file su cui si vuole salvare il messaggio. La funzione `myStreamListener()` invece ritorna un ascoltatore da utilizzare nella funzione `DisplayMessage` che si metterà in ascolto per riuscire a leggere il contenuto della mail quando disponibile. Il suo contenuto è il seguente:

```
function myStreamListener(){
     var streamListener = {
             mStream : null,
            src: '',
            QueryInterface : function(iid)  {
                if (iid.equals(Components.interfaces.nsIStreamListener) ||
                    iid.equals(Components.interfaces.nsIMsgHeaderSink) ||
                    iid.equals(Components.interfaces.nsISupports))
                 return this;

                throw Components.results.NS_NOINTERFACE;

                return 0;
            },

            onStartRequest : function (aRequest, aContext) {
                this.mStream =
Components.classes['@mozilla.org/binaryinputstream;1'].createInstance(Componen
ts.interfaces.nsIBinaryInputStream);
                var channel =
aRequest.QueryInterface(Components.interfaces.nsIChannel);
                channel.URI.QueryInterface(Components.interfaces.nsIMsgMailNew
sUrl);
                channel.URI.msgHeaderSink = this
            },


            onStopRequest : function (aRequest, aContext, aStatusCode) {

                this.mStream = null;
                realWriteIMAPMessage(this.src, this.file);
            },

            onDataAvailable : function (aRequest, aContext, aInputStream,
aOffset, aCount) {
                this.mStream.setInputStream(aInputStream);
                var chunk = this.mStream.readBytes(aCount);
                this.src += chunk;
```

```
            },

            onStartHeaders: function() {
            },

            onEndHeaders: function() {
            },

            processHeaders: function(headerNameEnumerator,
headerValueEnumerator, dontCollectAddress) {
            },

            handleAttachment: function(contentType, url, displayName, uri,
isExternalAttachment) {
            },

            onEndAllAttachments: function() {
            },

            onEndMsgDownload: function(url) {
            },

            onEndMsgHeaders: function(url) {
            },

            onMsgHasRemoteContent: function(aMsgHdr) {
            },

            getSecurityInfo: function() {
            },

            setSecurityInfo: function(aSecurityInfo) {
            },

            getDummyMsgHeader: function() {
            }
     }
     return streamListener;
 }
```

Da notare in questo codice è la funzione scritta in grassetto, questa funzione è quella che effettivamente salverà il messaggio una volta che lo stesso è stato completamente scaricato. Il codice di questa funzione è il seguente:

```
function realWriteIMAPMessage(fileContent, file){

     var sfile =
Components.classes["@mozilla.org/file/local;1"].createInstance(Components.inte
rfaces.nsILocalFile);

    sfile.initWithPath(file.unicodePath);

    sfile.QueryInterface(Components.interfaces.nsIFile);

    var stream = Components.classes['@mozilla.org/network/file-output-
stream;1'].createInstance(Components.interfaces.nsIFileOutputStream);

     var pref = Components.classes["@mozilla.org/preferences-
service;1"].getService(Components.interfaces.nsIPrefBranch);

     if(sfile.exists()) {
           try{
```

```
                sfile.remove(false);
            }catch(e){}
      }

      try{
            file.create(sfile.NORMAL_FILE_TYPE, 0666);
      }catch(e){}

      stream.init(sfile, 2, 0x200, false);
      stream.write(fileContent, fileContent.length);
      stream.close();

}
```

Il codice è abbastanza autoesplicativo e non necessita di ulteriori spiegazioni se non per il fatto che questa funzione se trova un file con lo stesso nome lo cancella per sostituirlo con quello nuovo. Per eliminare questo comportamento è possibile fare un controllo sull'esistenza del file e decidere se eliminarlo o meno.

CARTELLE SPECIALI (DRAFTS, INBOX, ...)

A volte potrebbe essere necessario per svariati motivi ottenere un oggetto `nsIMsgFolder` che punta alle cartelle speciali "Posta in Arrivo", "Posta in Uscita", "Bozze" o "Cartelle Locali". Ottenere il loro uri è abbastanza semplice; per farlo potete usare la seguente funzione:

```
function getSpecialFolder(folder){

      var url = "";
      var baseURL = "mailbox://nobody@Local%20Folders";

      if(folder == "INBOX"){
            url = baseURL + "/Inbox";
      } else if(folder == "SENT"){
            url = baseURL + "/Sent";
      } else if(folder == "DRAFTS"){
            url = baseURL + "/Drafts";
      } else if(folder == "ALL"){
            url = baseURL;
      }

      return url;
}
```

Il parametro `folder` è una stringa che deve assumere uno dei seguenti valori: INBOX, SENT, DRAFTS o ALL. In questo modo otterrete però solo l'URL del messaggio. Per ottenere l'oggetto `nsIMSgFolder` dovete utilizzare il seguente codice:

```
var mailbox = RDF.GetResource(url);
mailbox.QueryInterface(Components.interfaces.nsIMsgFolder);
```

Adesso potete utilizzare la variabile `mailbox` per ottenere ad esempio la lista dei messaggi contenuti in "Posta in arrivo" o le cartelle che contiene.

ALCUNE VARIABILI E FUNZIONI GLOBALI DI THUNDERBIRD

All'interno di Thunderbird esistono alcune variabili e funzioni globali che potete utilizzare per i vostri script, in particolare le più interessanti sono le seguenti: `document`, `navigator`, `messenger`, `msgWindow`, `GetDBView()`, `GenerateValidFilename()`. Vediamoli in dettaglio:

- `document`: è un oggetto fornito dall'interfaccia DOM. Esso permette di accedere alla struttura del codice XUL che contiene questo JavaScript. È utile soprattutto quando dovete far ottener/modificare valori XUL tramite JavaScript;
- `navigator`: è un oggetto fornito dall'interfaccia DOM. Esso permette di accedere alle caratteristiche dell'applicazione sovrastante, in particolare ad esempio al sistema operativo su cui gira l'applicazione.
- `messenger`: è un oggetto globale di tipo nsIMessenger fornito da XPCom, in alcune occasioni potrebbe risultarvi molto utile ad esempio per riuscire ad ottenere un messaggio a prtire da un URI;
- `msgWindow`: questa variabile di tipo `nsIMsgWindow` è utilizzata da alcune funzioni che vogliono una visione della finestra locale. Non è utile di per se, ma appunto deve essere utilizzata quando una funzione richiede come tipo di parametro appunto `nsIMsgWindow`.
- `GetDbView()`: questa funzione globale vi permette di ottenere una vista globale dell'interfaccia e quindi riuscire a determinare quali messaggi o quali cartelle sono stati selezionati rispettivamente tramite il metodo `getURIsForSelection` e l'attributo `msgFolder` (vedi gli esempi precedenti);
- `GenerateValidFilename()`: Questo metodo è necessario quando volete creare un file. Vuole due parametri: il primo è il nome del file mentre il secondo è l'estensione e ritorna il nome del file completo di estensione e ripulito di eventuali caratteri errati(ad esempio il carattere ":" nei sistemi Windows non può essere utilizzato nei nomi dei file o delle cartelle)

MODIFICARE IL TESTO DELLA STATUSBAR

Per modificare il testo della status bar potete utilizzare la seguente semplicissima funzione che non fa uso di oggetti XPCom:

```
function setStatusBar(text){
     document.getElementById("statusText").setAttribute("label", text);
}
```

Il parametro da passare `text` è una stringa che contiene il testo che deve essere visualizzato sulla barra di stato.

L'ID `statusText` è proprio quello dell'ID della status bar che contiene il testo e che può essere ricavato utilizzando il DOM Inspector di Thunderbird.

CREARE UNA DIRECTORY IN LOCALE

Per creare una directory sul vostro hard disk potete utilizzare la seguente funzione che vuole un percorso (assoluto o relativo) con il nome della cartella:

```
function createDir(name){
     var dir =
Components.classes["@mozilla.org/file/local;1"].createInstance(Components.inte
rfaces.nsILocalFile);
     dir.QueryInterface(Components.interfaces.nsIFile);
dir.initWithPath(name);
     if(!dir.exists()){      // Crearla se già esiste è un errore!!
           dir.create(Components.interfaces.nsIFile.DIRECTORY_TYPE, 0777);
     }
}
```

OTTENERE IL FILE SEPARATOR PER IL VOSTRO SISTEMA

Per ottenere il file separator (slash o back slash) del vostro sistema potete utilizzare la seguente funzione che non fa uso di XPCom:

```
function getFileSeparator(){
     var sep = "/";
```

```
      if (navigator.platform.toLowerCase().indexOf("win") != -1)
          sep = "\\";
    return sep;
}
```

GESTIONE DEI SUONI

Per eseguire un suono potete utilizzare la seguente semplice funzione:

```
function beep(){
     // Url del suono
     var soundURL = "chrome://smartsave/content/sound.wav";

     // Carico i componenti necessari ad eseguire il suono
     var   gSound =
Components.classes["@mozilla.org/sound;1"].createInstance(Components.interface
s.nsISound);
     var ioService = Components.classes["@mozilla.org/network/io-
service;1"].getService(Components.interfaces.nsIIOService);
     var url = ioService.newURI(soundURL, null, null);
     gSound.play(url);

}
```

Questa funzione utilizza il componente XPCom `@mozilla.org/sound;1` con l'interfaccia `nsISound` per poter eseguire il suono, mentre utilizza il componetene `@mozilla.org/network/io-service;1` con l'interfaccia `nsIIOService` viene utilizzato per caricare il file tramite l'URL. La funzione potrebbe essere migliorata facendo in modo che l'URL venga passato come parametro.

DOVE TROVARE ALTRI SUGGERIMENTI

Per poter trovare altre funzioni/script utili avete più opzioni. Per prima cosa vi consiglio si scaricare altre estensioni che fanno quello che vi serve o che magari lo fanno anche solo parzialmente e di studiarne il codice. Se il codice è scritto abbastanza bene e avete dimestichezza con JavaScript potete imparare molto. In secondo luogo potete andare a dare un'occhiata al sito www.mozdev.org, di solito si può trovare qualche bello script utile. Infine se proprio siete disperati potete fare un ultimo tentativo con Google, ma non sorprendetevi se troverete poca documentazione: non è facile trovarne di fatta bene per i componenti XPCOM e il più delle volte dovrete andare ad intuito.

Bibliografia

[1] http://www.mozilla.org

[2] Rapid Application Development with Mozilla
http://www.phptr.com/promotions/promotion.asp?promo=1484&redir=1&rl=1

[3] http://www.mozdev.org

[4] http://www.xulplanet.org

[5] http://www.mozilla.org/developer/

[6] http://www.ecma-international.org/publications/standards/Ecma-262.htm [JavaScript]

Indice