Top-Level functions in CCL cocoa apps

Background
A top-level function is one that will be invoked whenever the application throws to the toplevel. It can be set by calling (%set-toplevel <function>). This does not immediately invoke it. Rather, this just makes it the one that will be invoked if a throw to the top-level subsequently occurs. I had to work through how this all works for my own benefit, so hopefully this explanation will make it easier for others.

save-application
When save-application is called, a toplevel-function arg can be provided (see below for what happens if this argument is not provided)

save-application passes this argument directly to %save-application-internal

%save-application-internal
%save-application-internal first looks at the application-class argument also passed through save-application. If provided and different from gui::*application* (which contains the current application instance) it creates one. In the cocoa IDE the value of gui::*application* will be an instance of GUI::COCOA-APPLICATION. Note that this is not an Objective-C application class, this is a lisp application class.

If the toplevel-function arg was provided it becomes the one used, otherwise it creates a toplevel function that simply applies the function #'toplevel-function to the application-class instance just discussed. This method should be suitable for use when an app is started. See the toplevel-function method defined for instances of the class 'cocoa-application (defined in start.lisp).

%save-application-internal calls save-image to save the image to a file passing in the toplevel-function we just identified.

save-image
save-image is a fairly convoluted function that creates toplevel functions within other toplevel functions in order to make everything work out ok. It is called with two arguments: a save-function to be invoked to do the actual image save to a file and a top-level function that is to be made part of the image so that it will be invoked when the image is executed.

First it creates a toplevel function that will set up the image to be saved and then actually save it. It calls %set-toplevel with that newly created toplevel function and then invokes it by calling (toplevel) which does a throw to the top level. Let's call this new toplevel function TL1.

TL1 first calls %set-toplevel with an exit function so that if any error occurs during subsequent steps the Lisp application will just exit. The next thing it does is funcall all gui::*save-exit-functions*. Then it calls (%set-toplevel ...)with yet another constructed toplevel function that we will call TL2. TL2 is the toplevel function that will be included in

the saved application image and called when that image is executed. Finally TL1 funcalls the save-function argument that was passed into save-image.

TL2 is the toplevel function that will be called when the image is subsequently executed. It first calls %set-toplevel with the toplevel function argument provided as input to save-image. This is the same toplevel function that was determined in %save-application-internal. If a toplevel-function argument is provided to save-image, then this is that same function. Then TL2 calls (restore-lisp-pointers) to set up all internal pointers correctly in the memory image. So the effect of using TL2 as the toplevel function when the image is executed is that it will set up the user-specified toplevel function as the one to be called the next time there is a throw to the toplevel, restore memory to the way it was before the dump, and then (effectively) throw to the top-level to invoke the user-specified toplevel function.

The default toplevel function works as follows:

- Checks AppKit version number to make sure it is new enough and exits otherwise
- Sets *standalone-cocoa-ide* to t
- Starts swank listener if that is configured
- Checks to see whether FD 0 is /dev/null and if so sets the have-interactive-terminal-io slot of the current thread to nil and tries to start up the altConsole application
- Calls start-cocoa-application

start-cocoa-application

- Locally defines yet another toplevel function called cocoa-startup
- Interrupts the *cocoa-event-process* thread (the initial thread) and causes it to call %set-toplevel using the new cocoa-startup function as its argument and then (effectively) throw to the top-level to invoke cocoa-startup.

cocoa-startup

- Starts up a housekeeping thread
- If *NSApp* is null, runs init-cocoa-application (below) to set it
- Sets the application image (using image from the main bundle called NSApplicationIcon
- Sets the ccl::application-ui-object slot in the lisp application instance to the value of *NSApp*
- If the delegate for the *NSApp* object still isn't set (see init-cocoa application below for how this might have already been done), then set the delegate to the value of the application-proxy-class-name key that was passed to start-cocoa-application. If no value was passed for this key it defaults to *default-ns-application-proxy-class-name*, which is set to the class LispApplicationDelegate (as of October 2010).
- Calls run-event-loop (below)

run-event-loop

- %set-toplevel to nil
- change-class of *cocoa-event-process* to 'appkit-process
- call (event-loop) which sets up an appropriate error-handling mechanism and calls #/run on *NSApp* just like Objective-C programs do

init-cocoa-application

- Creates the shared defaults object by calling #/standardUserDefaults on ns:ns-user-defaults
- Opens the main bundle for the app and retrieves its info.plist
- Sets the process name to the value of the CFBundleName key in the info.plist
- Creates an instance of the class specified by the value of the NSPrincipalClass key in the info.plist; thereby making it the value of #&NSApp
- If the app instance didn't already set its own delegate, and if the CCLDelegateClass key in the info.plist specifies a class, then create an instance of that class and make it the application's delegate
- Load the nib file specified by the value of the NSMainNibFile key from the info.plist