This one took me even longer then the part with the tabs, but I finally got keyboard shortcuts in my own zimlet working.

If you still don't know, what zimbra is, read my [blog post about tab navigation in zimlet](http://dennis.dieploegers.de/tab-navigation-in-zimbra-zimlets/) to get to know it.

Again following [[http://blog.zimbra.com/blog/archives/2006/09/look-ma-no-mouse-keyboard-navigation-and-shortcuts-in-the-zimbra-collaboration-suite-and-the-kabuki-ajax-toolkit.html|Ross Dargahi's blog post]] I finally understood how Zimbra deals with keypresses. And this time, it isn't even awkward or funny, it's just plain not really documented.

The basis of all Zimbra key-handling is the DwtKeyboardManager. That object's responsible for handling all keyboard events and pushing them over to the appropriate handler. Zimbra uses this to register a global keyboard handler for all its applications.

As a zimlet isn't a real application in the Zimbra sense, there's no integrated way of dealing with keyboard shortcuts. **But** the DwtKeyboardManager can handle multiple default handlers. So if the first one doesn't handle your shortcut, you can register another one, that probably does.

For this you first need to wrap up a class that inherits from DwtKeyMap. This is where your shortcuts are „registered" and assigned to a so-called „event code".

Let's build up an example. Like I said, we need to create a DwtKeyMap. I used my handler-JavaScript file to create one:

```
de_dieploegers_example_keyKeyMap = function () {
```

DwtKeyMap.call(this);
this._load(this._map, de_dieploegers_example_key,
de_dieploegers_example_keyKeyMap.MAP_NAME);

};

de_dieploegers_example_keyKeyMap.prototype = new DwtKeyMap(true);
de_dieploegers_example_keyKeyMap.prototype.constructor =
de_dieploegers_example_keyKeyMap;

That's the building block to generate your own key map. The _load-method generates a key map out of a properties file. This is quite cool, because it adds internationalization to short cuts. I will get back to this later.

Now we need a map name. That's an array of maps for key bindings. The idea behind this is, that you can have multiple handlers, each handling its own keymap. For our example we just have one handler that does the whole work. So we just add one name:

```
de_dieploegers_example_keyKeyMap.MAP_NAME = {};
de_dieploegers_example_keyKeyMap.MAP_NAME["de_dieploegers_example_key"] =
"DeDieploegersExampleKey";
```

```
// reverse map of above
de_dieploegers_example_keyKeyMap.MAP_NAME_R = {};
(function() {
for (var i in de_dieploegers_example_keyKeyMap.MAP_NAME) {
de_dieploegers_example_keyKeyMap.
MAP_NAME_R[de_dieploegers_example_keyKeyMap.MAP_NAME[i]] = i;
}
})();
```

The generation of the reverse map is something I simply copy/pasted from ZmKeyMap. I don't know exactly if and why this is needed, but I guess it's important.

Now comes the event code. We want to show a simple message:

```
de_dieploegers_example_keyKeyMap.SHOW_MESSAGE = „ShowMessage“;
```

You notice, that we didn't actually bind a key until now. As I mentioned, this is done by using the internationalization features of zimbra. Put these three lines in your .properties-file:

```
de_dieploegers_example_key.ShowMessage.description = Show a nice dialog
de_dieploegers_example_key.ShowMessage.display = Shift-D
de_dieploegers_example_key.ShowMessage.keycode = Shift+100
```

Let's have a look how the lines are built up:

..

The keymap-name is set in this line:

de_dieploegers_example_keyKeyMap.MAP_NAME[„de_dieploegers_example_key] = „DeDieploegersExampleKey“;

(the one in the brackets).

The event-code is set in this line:

de_dieploegers_example_keyKeyMap.SHOW_MESSAGE = „ShowMessage“;

(the one in the quotation marks)

The data is one of the following:

* description: Description of event
* display: Human readable Display of shortcut
* keycode: Machine readable keyboard shortcut

I believe, that the first two are only handled for Zimbra’s own applications and are displayed in the shortcut help (under „preferences“). I haven’t checked, though.

The last one is the most important: It’s the key scan code for the keyboard shortcut plus modifiers. Read Ross‘ blog post and Zimbra’s sources for more information. I think it’s basically modifier-names and ASCII-codes of keys.

That concludes the generation of the key map. Now we have to register our handler and the keymap. This is done in our zimlet’s init-method:

```
de_dieploegers_example_keyHandlerObject.prototype.init = function () {
```

var kbMgr;

kbMgr = appCtxt.getKeyboardMgr();

kbMgr.registerKeyMap(new de_dieploegers_example_keyKeyMap());
kbMgr.pushDefaultHandler(this);

};

Now, being a key handler we need to provide two other methods: getKeyMapName and (obviously) handleKeyAction:

```
de_dieploegers_example_keyHandlerObject.prototype.getKeyMapName = function ()
```

```
{
return "DeDieploegersExampleKey";
}
de_dieploegers_example_keyHandlerObject.prototype.handleKeyAction = function (
actionCode,
ev
) {
};
```

The key map tells DwtKeyboarManager what keymap we're referring to. It's defined in this line:

de_dieploegers_example_keyKeyMap.MAP_NAME[„de_dieploegers_example_key] = „DeDieploegersExampleKey“;

(the one in the quotation marks).

The handleKeyAction finally handles the keyboard shortcuts we registered. You can find the event-code „ShowMessage“ in the parameter actionCode and other event information in the parameter ev.

I'll leave the final creation of the dialog to you. Have fun with Zimbra!