Saving code on Espruino

Normally when you upload code to Espruino via the IDE, it is stored in Espruino's RAM. If you reset the board or power is lost, all your code will be lost.

However it's easy to save your code to flash memory and make it permanent.

Summary

On normal Espruino devices, just type `save()` on the left-hand side of the IDE and the current state of Espruino including all saved code will be written so that it is loaded at boot time.

`save()` doesn't work on Bangle.js watches, since you wouldn't want to save the current state of the device (including any Widgets/etc) into flash memory.

There are other methods of saving code that can be more efficient, or can allow you to do things like saving constants.

You can change methods using the down-arrow below the Upload icon in the IDE, or under **`Communications`** in the **`Settings`** window. See below for more information.

Boot Process

To understand how best to save data, it's best to know how Espruino loads saved code.

When Espruino starts up, it does a few things:

- If `BTN1` is pressed or if it reset because of a call to `reset()`, it sets `hasBeenReset` to **`true`**.
- If `hasBeenReset` wasn't set, it looks for a compressed image (`.varimg` in Storage) of the interpreter's state that was saved with `save()`. If it exists it unpacks it into RAM.
- (2v00 and later) Looks for files in Storage named `.boot0`, `.boot1`, `.boot2` and `.boot3` and executes them in sequence. (On Bangle.js these are *not* executed if `BTN1` is held down, but all other devices execute them each time)
- Looks in Storage for a file named `.bootrst` and executes it if it exists (see Save on Send below)
- If `hasBeenReset` **wasn't** set and `.bootrst` wasn't found in the last step, it looks in Storage for a file named `.bootcde` and executes it (see Save on Send below)
- Initialises any previously-initialised peripherals
- Runs any handlers registered with `E.on('init', function() { ... });`
- Runs a function called `onInit()` if it exists.

If Espruino is reset with `load()` it follows the same steps as above, with `hasBeenReset` to **`false`**. However in Espruino 2v05 and later, `load(filename)` will follow the same steps but will load the specified file instead of `.bootcde`/`.bootrst`.

There are two main ways to save code with Espruino:

`save()`

If you type `save()` on the left-hand side of the IDE after your code is uploaded, the contents of Espruino's RAM at that point will be compressed and written in to flash memory.

This includes:

- All variables and functions
- Any watches created with `setWatch`
- Timeouts and intervals created with `setTimeout` and `setInterval`
- All Pin states

When power is next applied, Espruino will load the information back out of flash and will resume where it left off. You can think of this a bit like 'hibernate' on a PC.

This is the standard way of saving code in normal Espruino devices (**it is not suitable for Bangle.js**), and it means that you can interact with your code on the left-hand side of the IDE, changing variables and functions, and can then save everything - including your changes.

For instance, if you upload the code `var t = E.getTemperature()` and type `save()`, `t` will contain the temperature of the device *at the time that you uploaded* - **not** at the time the device started, or even the time you typed `save()`.

However, this means that any code that was executed at upload time will not be re-executed. For instance you may have some external hardware like an LCD display connected to Espruino - after applying power, the LCD will need to be initialised (since it can not remember its state). In this case you can create a function called `onInit` (or add a `E.on('init', function() { ... })` listener) that is automatically called by the interpreter when it initialises.

Once code is saved, you can return the interpreter to a 'clean' state with `reset()`. This won't clear out any of the saved data in flash, so if you reboot the device (or call `load()`) it will re-load your previously saved state. To completely clear out saved code, run `reset()` and *then* run `save()` to save the clean state back into flash memory.

Pros

- Once you have your software working, you can just `save()` and it will keep working
- You can make changes to `save()`d code and can then type `save()` again to save your changes
- JS code isn't stored in flash as plain text, so is harder for a malicious user to extract
- `E.setFlags({pretokenise:1})` will allow JavaScript code in RAM to be heavily compacted, and to execute more quickly.

Cons

- Not as memory efficient since everything is stored in RAM
- Some formatting of the code inside functions may change, and comments outside functions won't be saved
- Any code that has to be run at startup needs to be placed in a function called `onInit`, or a `E.on('init', function() { ... })` event handler

Gotchas

- Unless you use `onInit` or `E.on('init', ...)`, code won't run at boot time.
- Since `setWatch` and `setInterval` are remembered, if you call them in `onInit` and then `save()` multiple times, you can end up with multiple copies. You can use `clearInterval()` and `clearWatch()` in `onInit` to avoid that.

- When uploading code with an `onInit()` or `E.on('init', ...)` function the function won't be called at upload time and to test you'll have to either `save()` or call `onInit()` manually.

Save on Send (to Flash)

`Save on Send` is an option in the Espruino IDE. Behind the scenes it uses the `E.setBootCode` command to save JS code directly into Espruino's flash memory. When Espruino boots up, it then executes the JavaScript code.

This is similar to the way you'd program a 'normal' microcontroller.

For instance, if you upload the code **`var t = E.getTemperature()`** with **`Save on Send`** enabled, `t` will be set to the temperature every time the device is powered on (in contrast to what happens when you use `save()`.

`Save on Send` (in the Communications section of the IDE) has three settings:

- **`No`** - code is uploaded to RAM, but can be saved with `save()` (as above). `RAM` is displayed below the upload icon.
- **`Yes`** - JavaScript code is saved to flash and loaded at boot time. **`Flash`** is displayed below the upload icon. If `reset()` is called, Espruino will remove all code from RAM and will not execute the saved JS code. This saves your JS code to a file in Storage called `.bootcde`.
- (deprecated) **`Yes,`** `execute even after reset()` - JavaScript code is saved to flash and loaded even after boot. ⚠**`Flash`** is displayed below the upload icon. If `reset()` is called, Espruino will remove all `save()`d code from RAM, but *will still execute the JS code that you saved*. See the 'Both Options' section. This saves your JS code to a file in Storage called `.bootrst`. **We've now removed this option from the Web IDE** as it is dangerous and is almost certainly not what is required in 99% of cases. If you still wish to use it, you can choose to save to Storage as `.bootrst`.

To remove any code saved with **`Save on Send`**, simply call `E.setBootCode()` with no arguments.

Pros

- Runs all code at boot-time, so there's no need for an `onInit()` function
- The code inside each function is kept in Flash memory, so doesn't use up as much RAM. This is also true for Modules if **`Modules`** `uploaded` **`as`** `functions` is enabled in the IDE.

Cons

- Your JavaScript code is stored in flash as plain text, so can easily be read out
- If you make changes using the left-hand side of the IDE, there is no way to save them
- `E.setFlags({pretokenise:1})` will have no effect, since a function's code will be kept in Flash (you can still add `"ram"` as the first item in a function to force it to be loaded into RAM and pretokenised)
- It isn't possible to run code at upload time - code only ever runs when the device powers on.

Gotchas

- If you turn on **`Save on Send`**, upload code, and then turn it off, you can be left with both bits of code in Espruino at the same time (see 'Both Options' below).
- If you call `save()` after having saved to flash using a method below, you may get **`Got`** `EOF expected ..., [ERASED]` errors. These happen because there were function definitions in RAM that referenced code in Flash that is no longer there.

To Storage

This is just like `Save on Send (to Flash)`, but on Espruino 2v05 and later you may call `load(filename)` to reset and load JavaScript code from a named file stored on Espruino.

Combining options

It is possible to combine `Save on Send` and `save()` - see Boot Process above for more information.

This allows you to write separate code with `Save on Send` that can ensure certain things are always done, regardless of the code saved with `save()`.

You can even add files called `.boot0`, `.boot1`, `.boot2` and `.boot3` to add extra code that doesn't interfere with code saved in other ways. You could for instance add the following code:

```
require("Storage").write(".boot0", `
WIFI_NAME = "MyWiFi";
WIFI_PASS = "HelloWorld123";
`);
```

To ensure that you always had the variables `WIFI_NAME` and `WIFI_PASS` defined regardless of what other code you uploaded.

For instance if you're making a device like the Espruino Home Computer then you might want to use `Save on send` or `.boot0` to save all the code that initialises the display and keyboard. The computer can then be programmed and its state saved with `save()`, but regardless of what is saved to the device you will always be able to rely on the display and keyboard being set up correctly.

Notes

You may be able to save code to Espruino that puts it into a state that stops you from reprogramming it. On most boards, holding down a button while applying power can be used to force the device to boot *without loading any of the saved code*. Take a look at the information page on your specific board for more information.