Smart.IO App Presentation

Richard Man, richard@imagecraft.com
September 12th, 2017

The Smart.IO app is rather different from the majority of apps in the iOS "ecosystem". Rather than being a standalone program, it's meant to act as the "UI interpreter" for ImageCraft's patent pending Smart.IO toolkit. Therefore, in order to understand what Smart.IO app does, we also need to briefly describe the Smart.IO toolkit. (https://imagecraft.com/smartio/)

What does Smart.IO do?

Smart.IO allows embedded designers to create a modern UI for any embedded design. The most important attribute of the Smart.IO toolkit is also one that, for a number of people, takes a few moments to sink in:

"Let's emphasize this most important point: an embedded designer does NOT need to write ANY BLE/wireless or app code. None, zero, zilch, nada. Smart.IO does that for them, and the UI runs on any iOS device." With Smart.IO, hardware prototyping and even production units are no longer limited by "hardware knobs, bits, and bobs".

Components of Smart.IO

The Smart.IO toolkit is made up of both a hardware module and software components. This table lists the major components and which documents are applicable to each of them.

Component	**Description**	**Relevant Document**
Smart.IO hardware module	Provides hardware for BLE communication and the firmware support to implement the Smart.IO API	Smart.IO Hardware Integration Guide
Host Interface Layer	Source code in Standard C that an embedded user compiles with their host MCU firmware. Converts the Smart.IO API calls into command stream for the Smart.IO firmware.	Host Interface Layer
Smart.IO API	The API that the embedded users call to build and interact with the UI	Smart.IO Software API Smart.IO User Interface Elements
iOS Smart.IO app	A remote display manager. Smartphone	

	app that interfaces with a Smart.IO-enabled device and provides the UI for the device	

Smart.IO App Architecture

One may look at the Smart.IO app as a remote display manager: for example, the embedded system uses a simple C API function call (e.g. `SmartIO_MakeSlider`), and the app displays an interactive slider. When the app user (e.g. the user of the embedded device) adjusts the slider, the updated value is sent back to the embedded firmware for processing by the embedded device.

Once paired with a Smart.IO-enabled device, the embedded device sends UI commands through the BLE interface and the app interprets the commands, usually resulting in some UI element(s) being displayed.

A key innovation with Smart.IO API is that we utilize the concept of GUI slices, so that the generated UI will look good on **any** iOS device regardless of screen resolution, from the iPhone 5 to the iPad Pro. The embedded firmware engineer does not need to worry about screen resolution, exact pixel placement, etc.

In addition to providing a remote display manager, the Smart.IO toolkit provides all the hardware and API necessary to access the app, and takes care of all the BLE communication details, so that the embedded firmware only needs to make high level UI-centric API calls. (Since these details are not relevant to the Smart.IO app itself, we will not describe them further in this document.)

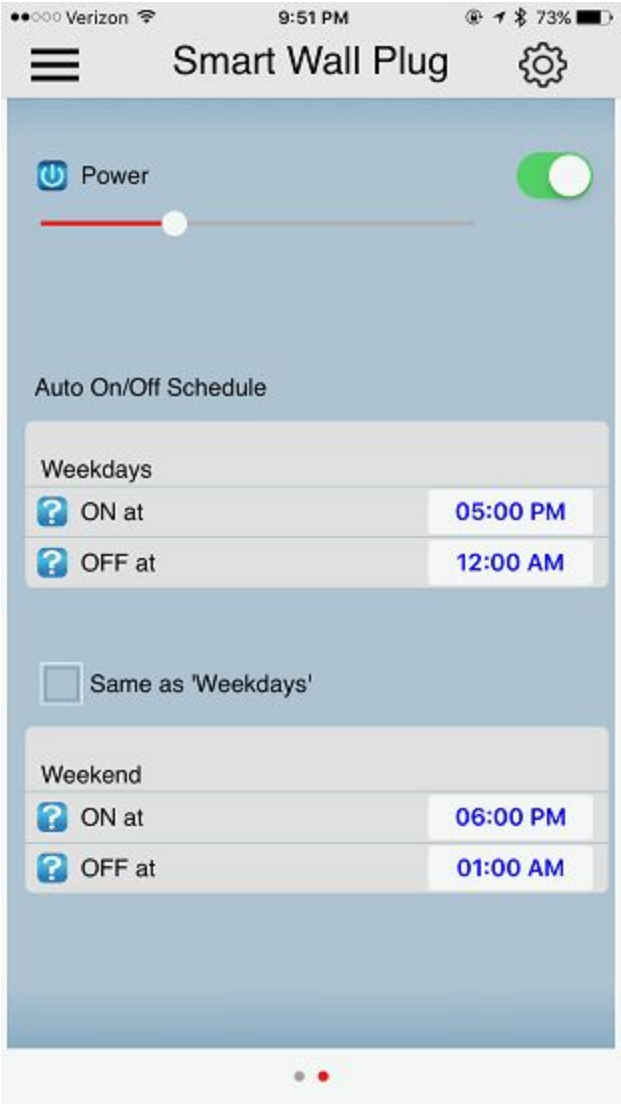
Smart.IO is designed such that customized app versions for a specific product can be generated for a given client so that their version of the control app, while built on top of the Smart.IO technology, can display the look and feel of any native apps.

Example of a Smart.IO-enabled Smart Wall Plug

Consider, for example, a common programmable wall plug. Typically it has 3-5 buttons, a small LCD, and some hard-to-read “quick setup” instructions crammed onto the inside panel in tiny print, or located on a small easy-to-misplace instruction pamphlet. Moreover, often the buttons are required to serve multiple purposes, and the only “user-interaction” is to push the buttons in various sequences to achieve this.

Obviously, a modern programmable wall plug can be more optimally controlled by an app. The user interface can be much more self-explanatory, using UI controls available on the iOS. For example, here’s a sample screen:

Sample UI for a smart wall plug

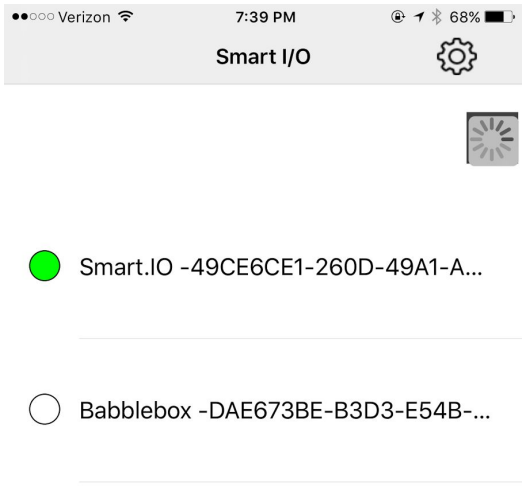


Everything is self-explanatory, with touch control. In fact, this is a sample screen from the Smart.IO app. The embedded firmware only needs to make about two dozen API calls to generate this screen, with no wireless or app coding necessary.

Some Sample Screens and Additional Features

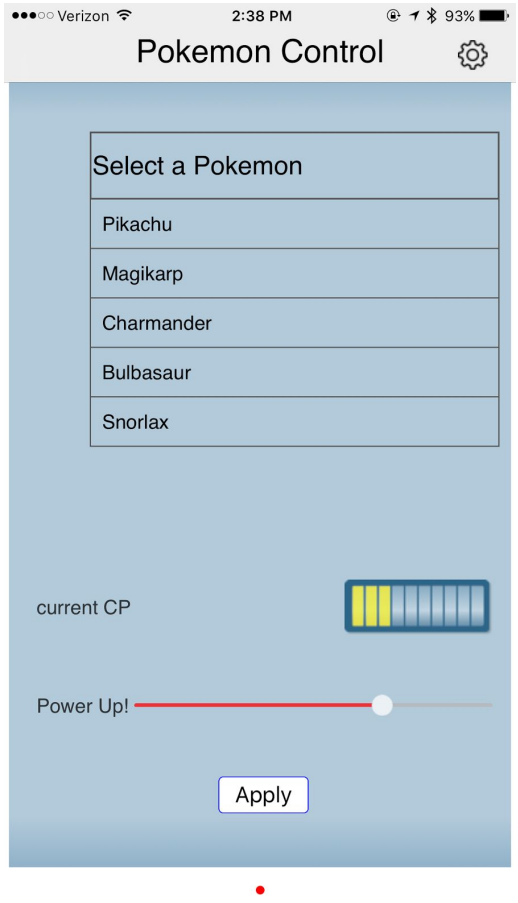
The Smart.IO app is designed to look for BLE UUIDs that conforms to a certain signature. The app and the device must perform handshaking protocols to ensure that they are compatible devices.

Device scanning page



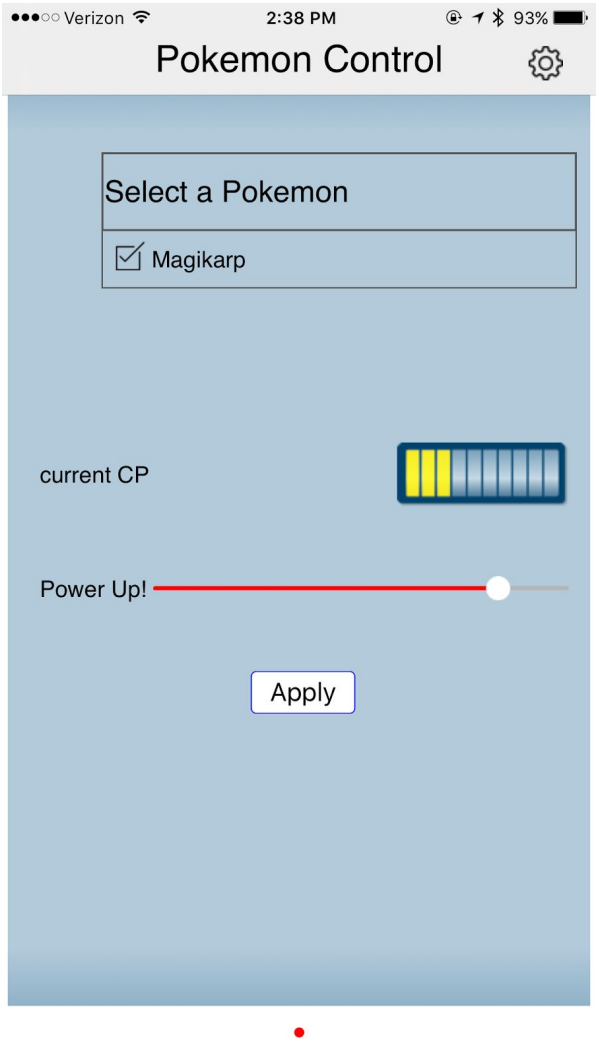
The Smart.IO API is designed to minimize BLE communication whenever possible. For example:

A simulated sample Pokémon game control app:



In this screencap, while it is hard to see, the slider control is slightly dimmed and cannot be changed. This is because the embedded (UI) designer is using a Smart.IO API call to make the enabling of the slider depend on the state of the expandable list: when no simple Pokémon selection is currently being made, the slider is disabled. This dependency check and slider enabling is handled entirely by the app, and does not involve communication to the embedded firmware (which is expensive in terms of BLE communication overhead).

Here's the screencap of the state when one Pokémon has been selected:



Again, while it may be hard to see from a screencap, the slider control is now at full brightness, and the slider can be adjusted by the app user.

Caching - an In-App Purchase Option

To encourage widespread adoption of the Smart.IO technology, the basic app is offered free to users. However, one potential problem with having the UI generation in the embedded firmware, is that the UI code effectively resides on the embedded system, and must be sent over to the app. This can take a bit of due to BLE overhead.

Therefore: a handy solution is caching of the UI generation instructions. A complex UI page that may take 5-6 seconds to generate could be reduced to about a couple of seconds once caching has been enabled. NOTE: caching is only enabled in the paid-for app (an in-app purchase option is available.)

For a fully customized app, further time reductions can be facilitated so that the UI becomes effectively is as fast as a native app.

List of UI Controls

It is beyond the scope of this document to describe all the UI controls. Full documentation is available on the webpage https://imagecraft.com/documentation/smart-io-documentation

The following input elements are provided:

UI Element	**Description**	**API Name**
On/off button	An on/off switch	`SmartIO_MakeOnOffButton`
3-pos button	A switch with 3 positions	`SmartIO_Make3PosButton`
Incrementer	Increment / decrement control	`SmartIO_MakeIncrementer`
Slider	Slider	`SmartIO_MakeSlider`
Expandable list	A collapsible list to select one item. No more than 6 to 8 items should be on the list.	`SmartIO_MakeExpandableList`
Picker	A scrollable list to select one item. For use when large number of items are needed.	`SmartIO_MakePicker`
Multi-selector	A single or double rows of typically 2 to 6 items.	`SmartIO_MakeMultiSelector`
Number selector	Select a number with a low and high range	`SmartIO_MakeNumberSelector`
Time selector	Select a time in hours and minutes	`SmartIO_MakeTimeSelector`

Calendar selector	Select a calendar date	`SmartIO_MakeCalendarSelector`
Weekday Selector	Select a weekday (MON-SUN)	`SmartIO_MakeWeekdaySelector`
OK button	A single button. The label can be modified.	`SmartIO_MakeOK`
Cancel/OK button	Two button choice. The labels can be modified.	`SmartIO_MakeCancelOK`
OK “Link” button	Same as an :OK button” except that the it is linked to another UI element. See text below this table.	`SmartIO_MakeOKLinkTo`
Checkboxes	A group of checkboxes where multiple items can be selected.	`SmartIO_MakeCheckboxes`
Radio buttons	A group of radio buttons where one item can be selected.	`SmartIO_MakeRadioButtons`
Text entry	A box where text can be entered.	`SmartIO_MakeTextEntry`
Password entry	Same as “text entry” except that each character is replaced by * in the display.	`SmartIO_MakePasswordEntry`
Number entry	Same as “text entry” except that only numbers are accepted.	`SmartIO_MakeNumberEntry`

Output Controls

UI Element	**Description**	**API Name**
Text Box	Display text in a box with specified width (in virtual pixels). Also allow slice icon, slice label, and box alignment.	`SmartIO_MakeTextBox`
Multiline Text	Display text in a box that takes the full width of the screen	`SmartIO_MakeMultilineBox`
Counter	Display numeric digits in a bound box	`SmartIO_MakeCounter`
Progress Bar	Display progress (percentage) in a bar	`SmartIO_MakeProgressBar`

Progress Circle	Display progress (percentage) in a circular “bar”	`SmartIO_MakeProgressCircle`
Horizontal Gauge	Display quantity (percentage) in a horizontal gauge	`SmartIO_MakeHGauge`
Vertical Gauge	Display quantity (percentage) in a vertical gauge	`SmartIO_MakeVGauge`
Semicircular Gauge	Display quantity (percentage) in a semicircular gauge	`SmartIO_MakeSemiCircularGauge`
Circular Gauge	Display quantity (percentage) in a circular gauge	`SmartIO_MakeCircularGuage`
Battery Level	Display a battery icon with the charge level (20% increment)	`SmartIO_MakeBatteryLevel`
RGB Led	Display a “led” with on/off state, and one of the RGB (Red Green Blue) colors.	`SmartIO_MakeRGBLed`
Custom Horizontal Gauge	Display quantity (percentage) in a horizontal gauge with custom colors	`SmartIO_MakeCustomHGauge`
Custom Vertical Gauge	Display quantity (percentage) in a vertical gauge with custom colors	`SmartIO_MakeCustomVGauge`

There are more features available in the Smart.IO toolkit, but this should suffice to give an overall idea of what the Smart.IO app is all about.