.obf and .obz Open Board File Formats

Introduction

AAC apps are relatively consistent in their interface and layout, so it seems reasonable to expect a way to transfer boards and board sets from one app to another. However, there isn't a standardized file format for doing so. The .obf (stands for Open Board Format) and .obz file extensions are meant to address this problem by providing a simple, flexible format for packaging potentially-self-contained board configurations.

Basics

.obf files are JSON files that contain the structure and layout of an individual board. There are a number of more advanced options, but at a basic level you define a list of buttons, a list of images, and a grid-based layout for the buttons (non-grid layouts are supported, I'll cover that below). The `format` attribute should be "open-board-0.1" to support versioning. All IDs should be strings (not numbers). Buttons, images and sounds must have a unique ID, but it doesn't need to be globally unique, just unique to the current .obf or .obz file. Note that image (and sound) files are referenced on button entries using `image_id` (or `sound_id`).

```
{
  "format": "open-board-0.1",
  "id": "1",
  "locale": "en",
  "url": "http://www.myaacapp.com/boards/123",
  "name": "Example Board",
  "description_html": "This is just a <b>simple</b> example board I put together.",
  "buttons": [
    {
      "id": "1",
      "image_id": "2",
      "label": "happy",
      "border_color": "rgb(0, 0, 55)",
      "background_color": "rgba(200, 255, 255, 0.2)"
    },
    {
      "id": "2",
      "label": "drinks",
      "image_id": "2"
    },
  ],
  "grid": {
    "rows": 1,
    "columns": 2,
    "order": [
      ["1","2"]
    ]
  },
  "images": [
    {
      "id": "1",
      "url": "http://cdn.myaacapp.com/happy.png",
      "width": 1024,
      "height": 768,
      "content_type": "image/png"
    }
  ]
}
```

.obz files at their most basic are one or more .obf files compressed as a zip file. If the .obz file contains more than one .obf file then a ./manifest.json file is required in the root of the zip file to define the contents. The manifest.json provides id-based mapping of all resources in the package, and a root attribute, which specifies the location within the zipped package of the primary board (note that id-to-path mappings are technically redundant but still required):

```
{
  "format": "open-board-0.1",
  "root": "boards/1.obf",
```

```
  "paths": {
    "boards": {
      "1": "boards/1.obf",
      "2": "boards/2.obf"
    },
    "images": { },
    "sounds": { }
  }
}
```

These examples reference external URLs for images. You can also include images and sounds inline using the `data` attribute with the value set as a data-URI:

```
  "images": [
    {
      "id": "1",
      "data": "...FTkSuQmCC",
      "width": 1024,
      "height": 768,
      "content_type": "image/png",
    }
  ]
```

Magic, right? If you're wondering how to handle proprietary symbol sets, there's more details farther down. When packaging as an .obz file instead of an .obf then you can also just include the files in the zipped package and reference them using the `path` attribute:

```
  "images": [
    {
      "id": "1",
      "path": "images/pic.png",
      "width": 1024,
      "height": 768,
      "content_type": "image/png",
    }
  ]
```

Path-based referencing is useful because then you can use the same file across multiple boards, potentially resulting in a smaller overall file size.

Linking to Other Boards

You can link to other boards within an .obz file or using URLs. To configure a button as a linked button, you define its `load_board` attribute:

```
  "buttons": [
```

```
    {
      "id": "2",
      "label": "drinks",
      "image_id": "2",
      "load_board": {
        "name": "Drinks",
        "data_url": "http://myboards.net/api/v1/boards/123",
        "url": "http://myboards.net/boards/123"
      }
    }
  ]
```

The `url` attribute allows you to redirect the user to a web site, in case the linked board is contained in another system. The `data_url` attribute is an endpoint that should return programmatic data about the board, but may require authentication. There's not really a clean way I can think of other than this to include settings that link to another board when importing a standalone .obf file, but at least within the same system this should prove helpful. The importing app may ignore links like this if they don't know how to handle them.

Linking within an .obz file is a lot more straightforward. The `load_board` attribute should have a `path` attribute defined that points to the linked .obf file:

```
  "buttons": [
    {
      "id": "2",
      "label": "drinks",
      "image_id": "2",
      "load_board": {
        "id": "1",
        "name": "Drinks",
        "data_url": "http://myboards.net/api/v1/boards/123",
        "url": "http://myboards.net/boards/123",
        "path": "boards/3.obf"
      }
    }
  ]
```

Which can then be used to parse the next .obf file. Note that it is recommended that an importing app import all .obf files in the .obz package, regardless of whether they are eventually referenced through the root board, but this behavior is not required for compliance.

Additional Parameters

In addition to linking and referencing resources, there are a bunch of other things you can configure. The idea is to be flexible without going crazy, but if there are things you think should be here that aren't, please make a comment.

Button Ordering

In a grid-based layout (absolute-positioned is defined below) you define your button order as a list of ids. Each id should match the unique id provided for the corresponding button in the `buttons` list. The grid is defined using the board's `grid` attribute. Any buttons that are empty or unused should have a `null` placeholder in the grid's `order` attribute, which is an array of arrays.

```
{
  "grid": {
    "rows": 3,
    "columns": 6,
    "order": [
       ["1","2","3","4","5","6"],
       ["1",null,null,null,null,"6"],
       ["9",null,"23",null,null,"4"]
    ]
  }
}
```

Proprietary Symbol Sets

If you're using a proprietary or licensed symbol set then it's likely a violation of terms of use to have a public image URL or embed the images in shareable data files like an .obf. In that case you can define the `symbol` attribute on the image, which includes a few sub-attributes. Standardization of symbol sets across apps is desired (i.e. if two apps use SymbolStix, it should be possible to pass .obf files back and forth and the boards should remain intact). *NOTE: I haven't used any proprietary symbol sets, so I'm definitely open to suggestions on the best way to organize this attribute.* If an app does not have access to the specified symbol set, it may fail or notify the user of alternative images.

```
  "images": [
    {
      "id": "1",
      "symbol": {
        "set": "symbolstix",
        "filename": "happy.png"
      }
      "width": 1024,
      "height": 768,
      "content_type": "image/png",
    }
  ]
```

Different Sound than Labelled

Sometimes you want to make a different sound than is provided by the label (i.e. the button says "happy" but you want it to say "I am happy"). In these cases you can define the `vocalization` attribute to whatever you would like. If it is not defined, then the `label` attribute will be used to generate the sound.

```
"buttons": [
  {
    "id": "1",
    "image_id": "2",
    "label": "happy",
    "vocalization": "Are you as happy as me?"
  }
]
```

Spelling and Specialty Actions

Some buttons will be used to perform actions other than vocalizing a word or phrase. For example, a keyboard-style board would have individual letters which should be combined to make single words, not pronounced independently, and would also have a “spacebar” button for ending words. Other common actions include returning to the home screen, clearing the current sentence, etc. as well as more advanced functions like copy and paste. All of these options are defined using the `action` attribute on a button.

To specify buttons as spelling-style buttons, you define `action` as “+something”, i.e. “+a” to add an “a” to a word, “+oo” to add “oo” to a word, etc.

```
"buttons": [
  {
    "id": "1",
    "image_id": "2",
    "label": "F",
    "action": "+f"
  }
]
```

To specify other actions, you define action as “:something”, i.e. “:space” to finish the previous word and start a new word, “:clear” to clear the current sentence, etc.

```
"buttons": [
  {
    "id": "1",
    "image_id": "2",
    "label": "clear",
```

```
    "action": ":clear"
  }
]
```

Not all communication apps support these types of specialty actions, and if your app doesn't support them but you find them in an imported file, you should notify the user. The list of suggested supported actions are:

- `+something` - append letter(s) to a currently-spelled word
- `:space` - finish spelling and start a new word
- `:home` - return to the user's home board
- `:speak` - speak the current sentence box contents
- `:clear` - clear the sentence box
- `:backspace` - remove the last entry in the sentence box

This list will probably grow over time, and will be split into suggested and optional actions. In the mean time, if you have actions that are custom to your app, you can use the `ext_` notation covered in more detail below (in this case still prefaced by a colon).

```
"buttons": [
  {
    "id": "1",
    "image_id": "2",
    "label": "clear",
    "action": ":ext_speakify_flip_buttons"
  }
]
```

If a button needs to support multiple actions, you can define it in the optional `actions` attribute. Some AAC systems will only support a single action per button, so you should include a single `action` attribute as a fallback in such a case.

```
"buttons": [
  {
    "id": "1",
    "image_id": "2",
    "label": "clear and home",
    "action": ":clear",
    "actions": [":clear", ":home"]
  }
]
```

Color Definitions

Buttons can define both background and border colors. Text color is currently non-configurable and should be handled by the communication app based on its implementation. Color attributes are string values and can be represented as rgb or rgba values. If alpha is not supported (i.e. rgba) then it is suggested that the communication app convert the color into a similar non-alpha color based on its implementation (note this probably should be something more intelligent than “just strip off the alpha” or the color won’t really match as expected).

```
"buttons": [
  {
    "id": "1",
    "image_id": "2",
    "label": "happy",
    "background_color": "rgb(255, 255, 255)",
    "border_color": "rgba(255, 0, 0, 0.5)"
  }
]
```

Absolute Positioning

The default layout is defined using the `grid` attribute, and `grid` should always be defined in case an importing app doesn’t support absolute positioning of buttons. However, if an app supports absolute positioning, you can define `top`, `left`, `width`, and `height` attributes for all buttons, in which case they will be rendered using absolute positioning. All four attributes are required for all buttons, and have values defined as a percentage (from 0.0 and 1.0).

```
"buttons": [
  {
    "id": "1",
    "image_id": "2",
    "label": "happy",
    "left": 0.102,
    "top": 0.2345,
    "width": 0.08,
    "height": 0.11
  }
]
```

The above example would position the button (screen_width * .102) from the left, (screen_height * .2345) from the top, with a width of (screen_width * .08) and a height of (screen_height * 0.11). With absolute positioning, buttons are rendered in the order they are specified so later buttons could potentially be drawn on top of earlier buttons.

String Lists

As an optional enhancement to support boards that can be rendered in multiple languages, obf allows for the inclusion of multiple string lists. String lists can be defined for multiple locales, and are essentially a dictionary where the keys match values set for `label` and `vocalization` attributes on buttons. String lists are defined via the `strings` attribute on the root level and are mapped by locale. If no matches are found in the strings dictionary for a given button, the parser can use the attribute's value as-is. It's up to the app to decide whether to fall back in cases with a more general locale (i.e. using "`en`" if nothing is defined for "`en-us`")

```
{
  "locale": "en",
  "strings": {
    "en": {
      "happy": "happy",
      ":time": "time"
    },
    "es": {
      "happy": "contento",
      ":time": "hora"
    },
    "fr": {
      ":time": "temps"
    }
  }
  "buttons": [
    {
      "id": "1",
      "image_id": "2",
      "label": "happy",
      "border_color": "rgb(0, 0, 55)",
      "background_color": "rgba(200, 255, 255, 0.2)"
    },
    {
      "id": "2",
      "label": ":time",
      "image_id": "2"
    },
    {
      "id": "3",
      "label": "Brian",
      "image_id": "2"
    },
  ]
}
```

In the above example the English `strings` matches some of the attributes defined on the buttons. The `label` "Brian" is not translated, so the app will fall back to using the attribute as-is. The Spanish strings are also, but the French strings are missing a translation for "Brian" and

also “happy”, so the app can fall back to the strings as-is for both of these labels, or search alternate string lists for a different fallback.

Licensing

You can define licenses for boards, images and sounds, including proper attribution. If no license is defined for a resource then the importing app should assume it is private, all rights reserved.

```
{
  "format": "open-board-0.1",
  "id": "1",
  "url": "http://coughdropaac.com/boards/123",
  "name": "Example Board",
  "license": {
    "type": "CC-By-SA",
    "Copyright_notice_url": "https://creativecommons.org/licenses/by-sa/4.0",
    "source_url": "http://coughdropaac.com/boards/123?optional",
    "author_name": "Bob Jones",
    "author_url": "http://about.me/bobjones",
    "author_email": "email_or_link@dontneedboth.com"
  }
  "images": [
    {
      "id": "1",
      "url": "http://cdn.coughdropaac.com/happy.png",
      "width": 1024,
      "height": 768,
      "content_type": "image/png",
      "license": {
        "type": "CC-By",
        "copyright_notice_url": "https://creativecommons.org/licenses/by/4.0",
        "source_url": "http://coughdropaac.com/optional/url",
        "author_name": "Sarah Miller",
        "author_url": "http://facebook.com/bobjones",
        "author_email": "email_or_link@dontneedboth.com"
      }
    }
  ]
}
```

Extensions

If there are additional parameters you need that aren’t available, you are free to add them to the file, just make sure to prefix them with `ext_` to prevent any collisions on the future. It is

recommended that you include the name of your app in the attribute name (i.e. for an app called “speakify” you could define `ext_speakify_name`, `ext_speakify_animation`, etc.)

```
"buttons": [
  {
    "id": "1",
    "image_id": "2",
    "label": "happy",
    "ext_speakify_app_store_id": "com.facebook.Facebook"
  }
]
```

url vs. data_url

In addition to defining `url` attributes on images and sounds, you can also define the `data_url` attribute, which is the URL to a programmatic endpoint (API call) for retrieving information about the resource. These endpoints may or may not require authentication, depending on the app.

Parsing Guidelines and Gotchas

There are cases where an .obf file may include some ambiguity. The goal of this section is to help address that.

Numerical IDs

It has been observed that some implementers have been using numerical IDs instead of string IDs. This is contrary to the spec, but you may see it in the real world. The easiest fix is to cast all IDs as strings (button IDs, image IDs, grid order IDs, etc.) before trying to use them.

ID Uniqueness

It’s worth repeating that board, image and sound ids do not need to be globally unique, but should be unique to the current .obf or .obz file. When multiple .obf files are zipped up together in a single .obz file, the image and sound ids should be unique to that .obz file, which is a stronger constraint than is required for the .obf files when bundled independently (i.e. if two .obf files both have images with id of “2”, when they are independent that is fine, but when they are combined into a single .obz file the package is invalid). As such, it may actually be in your best interest to use globally-unique or system-wide-unique ids to make your life easier.

Images and Sounds with Multiple References

There is nothing preventing an image or sound record in an .obf file from including multiple file references (i.e. a data-URI as well as a remove URL or zipped file attribute). If multiple references are included, the app should use the first reference found in the following order:

- `data`
- `path`
- `url`
- `symbol`

Future Work

Currently I'm working on a ruby gem (library) that will provide better support for importing and exporting from other systems. I also plan to host a website that will provide public documentation as well as a validator and .obf-to-.pdf and .obz-to-.pdf converters.

Some board systems have a hard-coded header that manages common actions like "speak" and "back", and others don't. What to do, what to do...

Define spelling buttons, so you can create a board that is a keyboard.

Option to "say it now without adding to the sentence" for quick fires. Also additional button-specific settings and overrides.

Consider adding optional word-type attribute (noun, verb, etc.).

Changes

3/4/2019 - Added Changes section, clarified location of manifest.json for .obz files

Additional References

- http://www.oatsoft.org/Software/concept-coding-framework-ccf/
- http://sensorysoftware.com/answers/xml-schema-for-the-grid-2/
- https://github.com/willwade/AAC-Tools/tree/master/Extractors/SymbolDB