



DEVELOPING CUSTOM PICTOGRAMS FOR THE MOBILE WEB


Eduardo Casais
areppim AG
Bern, Switzerland

1. A MATTER OF TRADE-OFFS

Pictograms – miniature graphical representations of states, actions and objects – made their way into the mobile Web over 15 years ago. Several normalized (UNICODE, WAP) and proprietary (Japanese emojis, Openwave) mechanisms are in place to enrich Web applications with pre-defined images.

The various approaches, with correspondence tables between icon dictionaries, are reviewed in the article at <http://areppim.com/b2evolution/usrblogs/technotes/?p=37>. However, one cannot always rely upon these standards:

- ◆ Numerous devices do not support any standard pictographic facilities at all.
- ◆ Many others do not implement all symbols of the reference sets; fall-backs are therefore needed to cope with missing elements.
- ◆ A service may require images outside normalized dictionaries (see <http://areppim.com/b2evolution/usrblogs/technotes/?p=38>), like the sign for RSS feeds  or the indication of a hidden navigation menu .
- ◆ Finally, an organization may wish to enforce its own look-and-feel across platforms, since the typical appearance of default icons can be disparate:

Symbol	DoCoMo	iOS	Openwave	UNICODE
upper left				
password				
message				
pull face				

Programming techniques for the ad-hoc implementation of pictograms entail different compromises regarding the properties expected from the resulting icon resources – apart from their graphical quality:

- ◆ They can be inserted in HTML and CSS markup and manipulated like normal textual data (alignment, scale, colour, effects).
- ◆ Once installed, they are persistent: they survive the end of a browsing session and the attendant flushing of the client cache.

- They are accessible to all mobile Web applications running on the terminal: different Web pages need not load and store their own version of the icons – just as each municipality need not define its own variant of traffic signs.

In the following, we describe common methods and discuss their applicability. While the topic is treated in a resolutely mobile perspective, all techniques carry over directly to desktop Web development.

2. SYSTEM FONTS

The only approach that fulfils all the aforementioned requirements consists of installing an adequate typeface implementing UNICODE pictographs on the terminal, and then configuring the client software to use it as the default font. The requisite manipulations imply transferring a font package (the TrueType format is practically always supported) directly to the end-user device; what is usually straightforward with linux or Windows can become quite involved on a mobile operating system:

OS	Manual procedure summary
Android	Overwrite system fonts in folder <code>/system/fonts</code> . Requires a device management tool (ADB from the Android SDK). Requires (possibly transient) root privileges. http://yespiracy.com/forums/android/how-to-install-unicode-fonts-in-android-(using-adb)
Bada	Custom fonts can be added, but the system typeface itself appears to be non-replaceable. Requires a device management tool (Kies and sTune). http://badahub.com/2011/11/method-to-add-new-font-in-bada-2-0.html
Blackberry 7.x	Copy new fonts with a <code>.font</code> extension to folder <code>/Device Memory/appdata/rim/fonts</code> or <code>/store/appdata/rim/fonts</code> . Then change default font via the “Options – Screen and Keyboard” menu. http://www.berryreview.com/2012/01/30/how-to-add-fonts-to-your-blackberry-smartphone-without-an-app
iOS	Overwrite system fonts in folder <code>/System/Library/Fonts/Cache</code> . Application-specific fonts that do not replace default typefaces are made known to iOS by editing a configuration file. Requires root privileges (jailbreaking). http://blog.gauravgiri.com/2008/08/tutorial-adding-extra-fonts-to-iphone
Meego	Overwrite system fonts in folder <code>/usr/share/fonts/nokia/Nokia Pure/proportional</code> . Requires root privileges. http://nokiamobileblog.com/how-to-change-the-font-of-your-nokia-n9
Nokia S40	Reflash the device with a modified firmware substituting system fonts. http://www.youtube.com/watch?v=OGGeN86MzUw

Symbian	Copy new fonts to folder <code>/Resource/Fonts</code> on the memory card, and rename them to match existing system fonts (which themselves should not be disturbed from the device built-in system memory). Old versions of S60 only accept fonts in the format GDR; translation from TTF is achieved through the tool KVT Symbian Font Converter. http://mynokiablog.com/2010/11/23/how-to-changing-fonts-in-symbian3-very-easy-done-in-a-minute-nokia-n8-c6-01-c7-e7 http://symbianworld.org/478-how-to-change-fonts-in-s60-3rd
WebOS	Add new fonts to folder <code>/usr/share/fonts</code> . Custom fonts overriding the default typeface must have a matching name. Requires a device management tool (WebOSQuickInstall). Requires root privileges (development mode). http://www.webosbuzz.com/hp-touchpad/1784-%5Btutorial%5D-unicode-fonts-touchpad-vietnamese-language.html
Windows	Application-specific fonts can be added to Windows Phone 7.x, but the system typeface itself is non-replaceable. On Windows Mobile 6.x, copy new fonts to folder <code>Windows/Fonts</code> before editing the registry. http://answers.microsoft.com/en-us/winphone/forum/wp7-wptips/add-fonts-to-windows-phone-7/b0e016fe-1f10-4be4-9e5e-fea8a12fbd80 http://forum.xda-developers.com/showthread.php?t=334781

Should the custom font be associated to a different writing logic (such as happens with many Asian languages), then keyboard layouts and rendering functions must be adjusted too, possibly affecting system libraries. A misconfiguration may well result in an unstable operating system, so all resources being altered should be backed up beforehand.

Because of its intricacies, specialized utilities have been published to facilitate font installation on popular devices (such as BytaFont for jailbroken iPhones). Nonetheless, the overall approach is only reasonably feasible in a controlled environment where handsets are configured centrally before being distributed to end-users – such as in a corporation or a public administration.

3. WEB FONTS

CSS includes a construct for downloading an external font to a user agent:

```
@font-face {
  font-family: 'IconicFill';
  src: url('http://site.ch/fonts/iconic_fill.eot');
  src: url('http://site.ch/fonts/iconic_fill.eot?#iefix')
      format('embedded-opentype'),
      url('http://site.ch/fonts/iconic_fill.woff')
      format('woff'),
      url('http://site.ch/fonts/iconic_fill.ttf')
      format('truetype'),
      url('http://site.ch/fonts/iconic_fill.svg#iconicfill')
      format('svg');
```

```
font-weight: normal;
font-style: normal
}
```

This clause instructs the browser to retrieve font *Iconic Fill* from the link matching a suitable format. Following the so-called “bulletproof syntax”, it includes a redundant declaration for files of type EOT and a URL pseudo-parameter to cater for limitations of Microsoft Internet Explorer; the identifier appended to the SVG declaration is indispensable. The font appears in a further CSS declaration:

```
.picto {
  font-family: 'IconicFill'
}
```

In the markup, one inserts the desired symbol with a numeric reference – or a letter or digit, if the font covers usual alphanumeric code points – and encloses it within an element of class `picto`:

```
<span class="picto">&#xE06E;</span><!-- phone call -->
```

Sometimes, HTML elements are systematically endowed with a pictographic marker, for instance when distinguishing different URL types. The following notation is then preferred – it eliminates clutter from the HTML markup, and enforces consistency by concentrating presentation characteristics in one CSS clause:

```
a[href^="tel:"]:before {
  /* Mobile phone symbol prepended to click-to-call links. */
  font-family: 'IconicFill';
  content: "\E06E "
}
input[type="file"]:after {
  /* Document symbol appended to file upload fields. */
  font-family: 'IconicFill';
  content: "\00E000"
}
<a href="tel:+41446313111">press centre</a>
Share file: <input type="file" name="upfile" />
```

HTML-5 capable devices may tailor individual elements through `data` attributes:

```
[data-picto]:before {
  font-family: 'IconicFill';
  content: attr(data-picto)
}
<h3 data-picto="&#xE074;">Detailed description</h3>
<!-- Magnifying glass symbol prepended to a section title. -->
```

Like other external resources, Web fonts may be evicted from the browser cache whenever the client software deems it appropriate, and utilization is restricted to the site that loaded them initially. Furthermore, mobile platforms impose restrictions on installable font packages (on Blackberry, for instance, they cannot exceed 90 kB). On the other hand, they exhibit two distinct advantages over custom system fonts:

- ◆ Since a Web font is always explicitly downloaded, all necessary symbols can be rendered even if the end-user re-configures the terminal with an unexpected default typeface that does not comprise the required glyphs.
- ◆ Since a Web font serves to style textual elements explicitly, it can be designed so that its symbols are judiciously placed in the UNICODE space – for instance within the *Basic Latin* block. This is valuable whenever the client does not recognize code points outside the *Basic Multilingual Plane*.

Practical Web services for converting a font definition into several formats, with the associated CSS font-face declaration, can be found at <http://www.font2web.com>, <http://fontface.codeandmore.com/index.php> and <http://www.fontsquirrel.com>. Another free online font converter is <http://www.freefontconverter.com>.

4. IMAGES STORED LOCALLY

The Web Local Storage mechanism of HTML-5 serves to manage key-value pairs kept persistently on the terminal. This makes it possible to avoid downloading graphical resources anew at every browsing session, as happens with Web fonts. The approach can be applied effectively to resources other than pictograms – such as static pictures that appear repeatedly in a Web site. It consists of storing images as BASE64-encoded strings locally, and, subsequently, retrieving them to fill in placeholders for pictograms in the HTML pages before these are rendered.

The landing pages of the mobile Web site include a Javascript function, activated by the `window.onload` event, that initializes the persistent store with all pictograms. We assume the existence of an ancillary routine `hasLocalStorage` that determines whether the browser supports the required capability.

```
function initPictoStorage () {
  if (hasLocalStorage ()) {
    // Check if the current version of the pictogram set is
    // already present to avoid useless re-installations.
    var curVersion = '...';
    if (curVersion != window.localStorage.getItem
        ('pictocurversion')) {
      // Retrieve pictogram set via AJAX/JSON. One could
      // instead declare it statically as an array here.
      var pictoHTTP = new XMLHttpRequest ();
      pictoHTTP.onreadystatechange = function () {
        if (this.readyState == 4) {
          if (this.status == 200) {
            // Store each pictogram locally with
            // a key prefixed by "picto". Add or
            // update the version as a last step.
            // Give up immediately upon a problem.
            // Instead of JSON.parse, a Javascript
            // eval() call can be used.
            var pictograms = JSON.parse
              (this.responseText);
```

```
        pictograms.push ({"pid" : "curversion",
                          "pval" : curVersion});
    for (var i = 0;
        i < pictograms.length; i++) {
        try {
            window.localStorage.setItem
                ('picto' + pictograms[i].pid,
                pictograms[i].pval);
        } catch (e) { break; }
    }
    // Whatever happens, initialize images.
    SetAllPicto ();
}
// Select image package to download depending
// on the pixel density of the display
var pictoFile;
if (window.devicePixelRatio !== undefined) {
    pictoFile = (window.devicePixelRatio >= 1.5
                ? 'hdpictograms.json'
                : 'pictograms.json');
} else {
    pictoFile = 'pictograms.json';
}
pictoHTTP.open ('GET',
                'http://site.ch/' + pictoFile,
                true);
pictoHTTP.send ();
} else {
    // No local storage, so initialize to fall-back images.
    setAllPicto ();
}
}
```

Images are declared in a JSON file with a property list binding pictogram names (“pid”) to their BASE64-coded bitmap (“pval”):

```
[ { "pid" : "phone",    "pval" : "iVBORw0K ... RK5CYII=" },
  { "pid" : "photo",   "pval" : "iVBORw0K ... TkSuQmCC" }, ... ]
```

Each pictogram is defined in the (X)HTML markup as an image, styled with a specific class *picto* and explicitly typed via a `data-picto` attribute, but with `src` left empty. For instance, one may attach a telephone icon to a click-to-call link as follows:

```
<a href="tel:+41446313111"><img class="picto" alt="call"
src="" data-picto="phone" />press centre</a>
```

Every page featuring pictograms defines a function that, when `window.onload` fires, walks through every `img` element, checks whether it belongs to class *picto*, and

if so, fetches the appropriate BASE64 string and assigns it to the `src` attribute. If data cannot be found in the local store, a URL to an external bitmap is used instead.

```
function setAllPicto () {
    // Retrieve all images in the HTML page and set the src of
    // those identified as pictograms with a proper value.
    var localStorageOn = hasLocalStorage ();
    var allImages = document.getElementsByTagName ("img");
    for (var i = 0; i < allImages.length; i++) {
        var isPicto = allImages[i].className.match (/picto/);
        if (isPicto != null) {
            // Retrieve the right BASE64 string stored locally.
            var localPicto = null;
            var pictoName = allImages[i].getAttribute
                ("data-picto");

            if (localStorageOn) {
                localPicto = window.localStorage.getItem
                    ('picto' + pictoName);
            }
            if (localPicto != null) {
                // Initialize img element with BASE64 string.
                allImages[i].setAttribute ("src",
                    'data:image/png;base64,' +
                    localPicto);
            } else {
                // The pictogram is not found locally, so fall
                // back to a normal URL to an external bitmap.
                allImages[i].setAttribute ("src",
                    'http://site.ch/pictograms/' +
                    pictoName + '.png');
            }
        }
    }
}
```

Finally, a CSS declaration for class *picto* ensures that pictograms are scaled to match the dimension of the surrounding text, with possible spacing adjustments:

```
img.picto {
    margin-left: 2px;
    margin-right: 2px;
    border-style: none;
    outline-style: none;
    /* Size and alignment chosen to resemble characters. */
    height: 0.75em;
    width: 0.75em;
    vertical-align: baseline
}
```

Images cannot be embedded as seamlessly as text, but they entail one major benefit over font-based pictograms: the flexibility to design rich polychrome symbols. Sharing of pictograms put in local storage is of course limited to pages from the

same host and domain, but a bigger drawback is that the technique is rendered inoperative whenever Javascript is switched off in the browser.

5. BACKGROUND CSS IMAGES

Absent the local Web storage facility, one can still rely upon BASE64 encoded images to deliver pictograms to terminals. Placing them inside a style sheet is preferable, since the CSS file can be cached and shared among several pages, whereas the direct inclusion of static strings in HTML results in code duplication and ultimately larger payloads to be transmitted over the air.

One CSS class is reserved for each pictogram, and the corresponding bitmap drawn as background of the empty textual space allocated for this purpose:

```
[class]="picto":before {
  /* Reserve exactly one em white space for each pictogram.
     Set position and scale in individual rules. */
  content: "\002003"
}
.picto-phone:before {
  /* Prepend the symbol of a telephone. */
  background: url('data:image/png;base64,iVBORw0K ... K5CYII=')
    no-repeat center center;
  background-size: 0.75em /* Fits nicely as a character */
}
.picto-photo:before {
  /* Prepend the symbol of a photographic camera. */
  background: url('data:image/png;base64,iVBORw0K ... kSuQmCC')
    no-repeat 50% 50%;
  background-size: 0.75em
}
```

High-density displays are taken into account by setting up separate versions of the style sheet for each pixel ratio of interest, and then instructing client software to download the relevant one through appropriate CSS media queries – possibly complemented by conditional statements to cater for Internet Explorer. In (X)HTML:

```
<!-- Default CSS rules for devices with normal pixel ratio. -->
<link rel="stylesheet" type="text/css" href="sitestyles.css" />
<!-- CSS rules for high-density screens, full media query. -->
<link rel="stylesheet" type="text/css" href="hdsitestyles.css"
media="only screen and (-webkit-min-device-pixel-ratio: 1.5),
  only screen and (min--moz-device-pixel-ratio: 1.5),
  only screen and (-o-min-device-pixel-ratio: 3/2),
  only screen and (min-device-pixel-ratio: 1.5),
  only screen and (min-resolution: 144dpi),
  only screen and (min-resolution: 1.5dppx)" />
```

One inserts pictograms in the markup by stating the relevant CSS class name:


```
<span class="picto-photo">Photography</span> is allowed.  
<a href="tel:+41446313111" alt="call" class="picto-phone">  
press centre</a>.
```

Developers will have no difficulty finding an abundance of on-line services converting data files to BASE64 strings – like <http://yellowgreen.de/image-to-base64-string-encoder> or <http://www.motobit.com/util/base64-decoder-encoder.asp> – as well as a few standalone programs (notably datauri, <https://github.com/nzakas/datauri>). More advanced utilities parse entire CSS style sheets, extract URL to external media, and replace them with embedded BASE64 data; a couple of examples are spritebaker (on-line Internet service, <http://www.spritebaker.com>) and cssembled (standalone command line tool, <https://github.com/nzakas/cssembled>).

6. SPRITES

The possibly oldest materialization of Web sprites may not be as well-known as CSS background image sprites, but it optimizes graphical resources to a comparable degree. It is especially well-suited for elements that are all identical in size – a constraint unlikely to prove troublesome in the case of pictograms.

Assuming the sprite is composed of eleven images N pixels wide by N pixels tall, arranged consecutively without separation in one column, one sets up a kind of viewport matching the dimensions of a single character through which the relevant portion of the sprite is rendered:

```
.pictowrap {  
    position: relative;      /* To shift the underlying image. */  
    width: 1em;             /* Set container dimensions the */  
    height: 1em;            /* ... same as surrounding text. */  
    overflow: hidden;       /* Only one sub-image is shown. */  
    display: inline-block   /* Image is embedded within text. */  
}  
[class|=“picto”] {  
    position: absolute;     /* To be able to shift the image */  
    left: 0;                /* ... flush to left of container. */  
    width: 1em;             /* Scale image to fill container */  
    height: 11em;           /* ... and keep height proportional.*/  
    outline-style: none;  
    border-style: none  
}  
.picto-doc {  
    top: -2em                /* third image inside sprite */  
}  
.picto-phone {  
    top: -8em                /* ninth image inside sprite */  
}
```

The second rule relying upon an attribute selector can of course be folded into the directives for individual pictograms. Although in principle equivalent, a horizontal sprite is not recommended, as many a mobile browser chokes on images that do not

fit naturally within the screen width. The final (X)HTML code includes a semantically meaningful `alt` attribute attached to the pictogram and looks like this:

```
<h3><span class="pictowrap"></span>Appendix</h3>
```

If one can afford the software brittleness induced by pixel-oriented declarations, then CSS background sprites may play a role – particularly when optimizing purely decorative assets such as graphical separators, repeating background textures, or when tailoring bullets in unordered lists with icons. Interestingly, these customization capabilities are also supported by old-fashioned built-in pictograms of the Openwave browser.

There are numerous on-line sprite generators, configurable interactively (alignment and padding of icons, colour table, image output format, etc). The site <http://www.designdim.com/2011/02/8-best-css-sprites-generator-resources> lists a selection of such utilities. SmartSprites (<http://csssprites.org>) is a more elaborate tool to produce CSS sprites and manage the accompanying style sheets. A tutorial on sprites can be found at <http://coding.smashingmagazine.com/2009/04/27/the-mystery-of-css-sprites-techniques-tools-and-tutorials>.

7. MINIMAL FALL-BACK

If the methods proposed so far turn out to be inapplicable – as it frequently occurs with low-end phones, older WAP handsets and PDAs running legacy software – the only recourse left is the default already presented in section 5: pictograms implemented as external images. Assuming style rules for class *picto* identical to those listed in the aforementioned section, one obtains a simple markup:

```
<a href="tel:+41446313111">press centre</a>
```

Unsurprisingly, this technique entails practically none of the performance benefits associated with built-in pictograms.

The matrix below assesses the compatibility of major mobile browsers against every solution described in the present article. The software versions under consideration encompass a broad range of device classes and generations. The table reveals that, while the SVG format is recommended for icons and pictograms because of the well-behaved properties of vector images under transformations, developers must be ready for a (still) relatively inconsistent support for this standard across browsers, and be wary of using full-fledged SVG instead of its “tiny” profile.

Browser	Web fonts	Web local storage	CSS data URI	Image sprites	SVG element
Android	✓ ≥ 2.2	✓ ≥ 2.1	✓ ≥ 1.5	✓ ≥ 1.5	✓ ≥ 3.0
Blackberry	✓ ≥ 6.0	✓ ≥ 6.0	✓ ≥ 6.0	✓ ≥ 4.6	✓ ≥ 6.0
Chrome	✓ ≥ 18.0	✓ ≥ 18.0	✓ ≥ 18.0	✓ ≥ 18.0	✓ ≥ 18.0

Browser	Web fonts	Web local storage	CSS data URI	Image sprites	SVG element
IE Mobile	✓ ≥ 10.0	✓ ≥ 9.0	✓ ≥ 9.0	✓ ≥ 9.0	✓ ≥ 9.0
Meego	✓ ≥ 8.5	✓ ≥ 8.5	✓ ≥ 8.5	✓ ≥ 8.5	✓ ≥ 8.5
MIB	✗ ≤ 2.2.2	✗ ≤ 2.2.2	✗ ≤ 2.2.2	✗ ≤ 2.2.2	✗ ≤ 2.2.2
NetFront	✗ ≤ 4.2	✗ ≤ 4.2	✗ ≤ 4.2	✗ ≤ 4.2	⚠ ≥ 3.3 ³
Nokia S40	✗ ≤ dp 2	✓ ≥ dp 1.1	✗ ≤ dp 2	✓ = S40 6 th	⚠ = S40 6 th 2
Obigo	✗ ≤ Q7.1	✗ ≤ Q7.1	✗ ≤ Q7.1	✓ ≥ Q7.1	⚠ ≥ Q7.1 ³
Opera Mini	✗ ≤ 7.1	✗ ≤ 7.1	✓ ≥ 4.0	✓ ≥ 4.0	✓ ≥ 4.0
Opera Mobile	✓ ≥ 10.0	✓ ≥ 11.0	✓ ≥ 10.0	✓ ≥ 10.0	✓ ≥ 10.0
Openwave	✗ ≤ 7.2	✗ ≤ 7.2	✗ ≤ 7.2	✗ ≤ 7.2	✓ ≥ 7.0
Safari	✓ ≥ 5.0 ⁵	✓ ≥ 5.0 ⁵	✓ ≥ 5.0 ⁵	✓ ≥ 5.0 ⁵	✓ ≥ 5.0 ⁵
SEMC	✗ ≤ 4.0	✗ ≤ 4.0	✗ ≤ 4.0	✗ ≤ 4.0	⚠ ≥ 4.0 ³
Symbian	✗ ≤ 8.3	✓ ≥ 8.3	⚠ ≥ 7.3 ¹	✓ ≥ OSS3.0	✗ ≤ 8.3
TSS	✗ ≤ 2.5	✗ ≤ 2.5	✗ ≤ 2.5	✗ ≤ 2.5	✗ ≤ 2.5
UC Browser	✗ ≤ 8.9	✗ ≤ 8.9	✗ ≤ 8.9	⚠ ≥ 7.9 ⁴	✗ ≤ 8.9



















1. Works directly on block-level elements, not on pseudo-elements.
2. Only available on devices running Opera Mini.
3. SVG availability depends on model, implementation often severely limited.
4. Works in paragraphs, not within headings.
5. Older versions not tested.

8. RESOURCES ON THE INTERNET

Instead of firing up Gimp or FontForge to design custom pictograms, developers can simply look to the wealth of (typo)graphical resources available on the Internet.

The sites <http://www.alanwood.net/unicode/fontsbyrange.html> and <http://www.babelstone.co.uk/unicode/fontlist.html> are the first stops for determining which typefaces implement a specific UNICODE block. Symbola (<http://users.teilar.gr/~g1951d>) is essentially restricted to ASCII, Greek and Cyrillic,

but encompasses many more pictograms – including non-standard ones. Quivira (<http://www.quivira-font.com/index.php>) offers better support for European and Asian scripts (e.g. Armenian, Georgian, Hebrew, Thai, Vietnamese), with a partial, but growing, set of pictograms. Both free typefaces implement mathematical signs as well as glyphs for musical notation and several ancient languages.

Relevant UNICODE block	Quivira 3.8	Symbola 7.07	Remarks
Enclosed alphanumerics			Circled digits
Geometric shapes			Triangles for “up”, “next”...
Miscellaneous symbols			Weather, games, zodiac...
Dingbats			Assorted pictograms
Misc. symbols and arrows			Orientated arrows
Enclosed alphanum. supplement			Squared “free”, “new”...
Misc. symbols and pictographs			Assorted pictograms
Emoticons			Smileys
Transport and map			Vehicles, tourist info...

Since their corresponding TrueType files weigh from 1.3 to 2.2 MB, these typefaces are suitable solely as system fonts. One must recur to specialized font packages, amounting to just a few KB to a few tens of KB, for binding downloadable resources to mobile Web pages. Fortunately, there is an expanding supply of such fonts, both free and commercial ones, professionally designed and covering numerous symbols. Overlapping collections are presented at <http://css-tricks.com/flat-icons-icon-fonts>, <http://owltastic.com/2011/08/simple-interface-design-icons>, <http://www.delicious.com/simurai/Icon-Fonts> and <http://www.iconsguide.com>.









































A plethora of icon sets, in bitmap or vector format, can be procured on the Internet. The choice is facilitated to some extent by repositories offering a query mechanism (by keyword, size, license, etc) such as <http://www.icojoy.com> and <http://findicons.com>. The frequent round-ups at <http://www.smashingmagazine.com> constitute a further source of information. We emphasize a couple of extreme cases:

- ◆ The set published by FatCow (<http://www.fatcow.com/free-icons>) comprises some 3000 colourful icons originally intended for desktop applications. Several of them are used in the compatibility tables of the present article.
- ◆ Pixelated (<http://wplifeguard.com/pixelated-icon-set>) and BacktoPixel (<http://www.icojoy.com/articles/28>) feature icons with a minimalist design and particularly small dimensions (10x10 and 9x9 pixels).

The sheer number of pictographic resources, the diversity of application domains, and the variety of requirements make any attempt at a comparative evaluation of available font and icon sets a daunting endeavour. We present here a

small sample illustrating the stylistic range exhibited by current packages, based on a handful of pictograms relevant for modern mobile Web applications:

- ◆ One pictogram present in every standard: the symbol for making a phone call, generally used for click-to-call links.
- ◆ Two symbols absent from pictographic standards, indicating a hidden menu (the “navicon” or approximations thereof), and an RSS feed.
- ◆ Two symbols available in some optional dictionaries: the cog traditionally designating a “settings” menu, and a sign denoting photographic functions.

Package	type	phone	menu	feed	config	photo	Attribution
Farm Fresh Web	icon						FatCow Web Hosting http://www.fatcow.com/free-icons
Web0.2ama	icon						Christian Burprich http://chrfb.deviantart.com
Frankfurt	icon						Patricia Clausnitzer http://pc.de/icons
105 loops	icon						Pranav Pramod http://dribbble.com/pranav
Iconic	font, icon						P. J. Onori http://somerandomdude.com/work/iconic
Typicons	font						Stephen Hutchings http://typicons.com
Entypo	font						Daniel Bruce http://www.entypo.com
Font Awesome	font						Dave Gandy http://fontawesome.github.com/Font-Awesome

There is a greater design and colour variability with icons, whereas fonts provide flat, black and white glyphs, to be livened up with CSS.

The Web service at <http://icomoon.io/app>, running on HTML-5 capable browsers, is invaluable to fine-tuning pictographic resources:

- ◆ Developers can mix and match symbols from various packages – including one’s own pictograms uploaded from SVG images or SVG font definitions – and then generate both a set of image files and a Web font out of them.
- ◆ The Web font is produced in WOFF, TTF, SVG and EOT formats, with a CSS boilerplate for `@font-face` declarations and related rules. It is very compact, as it excludes superfluous glyphs. Characters are assigned to the *Basic Latin Block*, the *Private Use Area*, or individual codes in UNICODE.







- Image files are produced in SVG and PNG; corresponding sprites are also generated, with a CSS file containing the necessary rules to use them.

One should of course study the relevant license for the permission to create and use derivative works from a font or icon package processed through IcoMoon.







9. CONCLUDING REMARKS

Beyond issues of portability, each mode of implementing custom pictograms produces subtly different representations.







Web Fonts

In normal  text. In very small  text. Inside a click-to-call  [press centre](#) link.
~~In a stricken  text.~~ In violet  text. In shadowed  text.







Local Web Storage

In normal  text. In very small  text. Inside a click-to-call  [press centre](#) link.
~~In a stricken  text.~~ In violet  text. In shadowed  text.







Data URI

In normal  text. In very small  text. Inside a click-to-call  [press centre](#) link.
~~In a stricken  text.~~ In violet  text. In shadowed  text.

Sprites

In normal  text. In very small  text. Inside a click-to-call  [press centre](#) link.
~~In a stricken  text.~~ In violet  text. In shadowed  text.

SVG image

In normal  text. In very small  text. Inside a click-to-call  [press centre](#) link.
~~In a stricken  text.~~ In violet  text. In shadowed  text.

The screen dump above, taken from an Opera Mobile 12.0 session with an HTML-5 page exercising all main techniques on the 105loops set, highlights their peculiarities regarding scale, alignment, and text formatting. In this test, Web fonts are black by default, whereas bitmap resources have been painted in various colours. Only those used in the “Web Local Storage” test are polychrome – but images in “Sprites”, “Data URI” and “SVG image” could have been too.

Mobile Web sites are geared towards delivering synthetic information on displays constrained in size, for immediate interactions in all situations of daily life. Pictograms stand out from the surrounding text, and thus quickly draw attention

towards important content; they identify data elements, making it easier to skim a page for an item of interest; they supplement styling attributes such as colour and font when these become inapplicable (monochrome display, disabled CSS).

However, pictograms may impair accessibility when the client software can neither handle them directly, nor map them to another appropriate representation (such as a vocal enunciation of the associated concept). Thus, screen readers are often confused by icon typefaces overriding the standard semantics of their underlying code points – such as when letter *O* serves to depict a clock. Developers can keep Web pages containing pictograms accessible in the following ways:

- ◆ Specify a meaningful `alt` attribute for pictograms implemented as external images with ``, as well as for Openwave icons and WAP fall-backs.
- ◆ Assign pictograms implemented as characters to adequate code blocks in UNICODE – such as “*Miscellaneous symbols and pictographs*” – or to the *Private Use Area*, whose members are generally passed over by screen readers but correctly displayed by normal browsers.
- ◆ When the pictogram is inserted in the HTML code, mark the enclosing element as an item to be ignored by screen readers:

```
<span class="picto" aria-hidden="true">&#xE06E;</span>
```

- ◆ Use an aural property to force screen readers to skip pictograms introduced in CSS markup via the `content` directive:

```
[data-picto]:before {  
    font-family: 'IconicFill';  
    content: attr(data-picto);  
    speak: none  
}
```

Support for ISO pictographs is becoming common in newer smartphones, but equivalent effects can be achieved on less well-endowed devices thanks to the methods explained in this article. Hence, nothing more prevents developers from taking advantage of pictograms to improve their mobile Web sites.

10. ACKNOWLEDGEMENTS

Many thanks to Sriram Sridharan from ScientiaMobile Inc. (www.scientiamobile.com) for his help in testing Internet Explorer Mobile.

REFERENCE

Eduardo Casais: *Developing custom pictograms for the mobile Web*, technical paper, areppim AG, Bern, Switzerland, 2013-02-11, 15 pages.

A version of this article has been published by mobiForge at

<http://mobiforge.com/designing/story/developing-custom-pictograms-mobile-web>

This paper can be downloaded in PDF format from areppim.com at

<http://areppim.com/b2evolution/usrblogs/technotes/?p=39>

© 2013 Eduardo Casais, areppim AG, Bern, Switzerland. All rights reserved.

ABOUT THE AUTHOR

Eduardo Casais has been working on mobile Web technologies since 1997. He led the development of content adaptation facilities in the Nokia WAP Gateway. He was also involved in projects dealing with transcoders for WWW on TV set-top-boxes. Eduardo Casais has been an invited expert to the Mobile Web Best Practices Working Group of the World-Wide-Web Consortium, where he participated in the elaboration of the Content Transformation Guidelines.

ABOUT AREPPIM AG

areppim AG develops Internet applications, with an emphasis on the display of quantitative information. The site <http://www.areppim.com> publishes data on a wide range of topics, presented as intuitive, content-rich charts and often accompanied by concise analyses. Naturally, data can also be accessed with mobile phones at <http://mobile.areppim.com> through a no-frills, mobile-optimized interface.

ADDRESS

areppim AG
Wankdorffeldstrasse 102
P.O. Box 261
CH-3000 Bern 22
Switzerland
e-mail: info@areppim.com