Support for intelligent handling of mobile and desktop versions of the websites.

Supporting desktop and mobile versions of web sites.

Many businesses have a requirement to support both *desktop* and *mobile* versions of their web site(s). In some cases, it done in a fashion where mobile traffic is directed to a *different* website, for example m.acme.com, while sending desktop traffic is sent to the usual www.acme.com.

Some sites would offers both mobile and desktop versions of their content *unified* under *same* domain – for example www.acme.com, tailoring output to match device capabilities.

Mobile site would normally render content in a simplified way that is more suitable for limited screen size and limited Javascript, HTML and CSS standards support in mobile browsers. Some sites distinguish between different mobile devices, offering more content rich versions of the content for some, more capable devices while offering the simple version to the less capable devices.

Likewise, due to limited screen sizes of mobile devices, sites cannot serve the same number of ads per page, for mobile users, as opposed to much larger number of ads per desktop versions of the site. This simple fact can point of contention with many a marketing department. Business people would rather serve a desktop page to an end user as opposed to delivering mobile version of same, due to higher ad revenues.

To support rendering of device-specific content, you need to have some kind of user-agent detection logic deployed at both web sites, analyzing user agents strings in incoming requests and redirecting user traffic as appropriate. For example, when a request coming for www.acme.com is detected to be coming from an Android mobile device, a response is sent to redirect the user browser to m.acme.com.

Likewise, similar logic can be deployed at m.acme.com and when request is detected as coming from a desktop browser, the response is sent back that redirects the browser to go desktop version of the site at www.acme.com.

There are a number of challenges related to supporting such dual-setups.

Reliable detection of mobile devices.

As simple as it sounds, this is less than trivial issue. Most detection methods rely on analysis of the request's User-Agent header. Two different techniques could be used: pattern-driven UA matching and DB-driven UA-matching. While pattern method is self-explanatory, the DB-driven one relies on some kind of database that contains, verbatim, letter-for-letter, all UA strings that you need to act on.

Once source of such information could the Internet – there're a number of commercial and open source projects that have such UA DB available.

No matter the method, as new devices seem to appear every week, you will need to update your matching logic to stay current.

aiCache offers industry's leading support for high-performance and seamless device detection using both of these methods, so that you don't have to do it at your application level.

Deploying device detection logic.

If you were to let the code on origin servers decide what version of content to serve, you'd need to pass such request straight to origin server, effectively negating all benefits of content caching. Clearly a non-starter !

aiCache offers industry's leading support for high-performance and seamless device detection so that you don't have to do it at your application level .

Having a strategy for handling of search bots/spiders.

Most sites don't want to allow spidering/indexing of mobile site's content and instead would rather prefer Googles of the world indexing their main desktop sites. Again, it boils down in most cases to revenue-per-page issue we described above. When someone runs a search and your site pops up in the results, they'd want the link to point to desktop version.

So you would probably want to prohibit spidering of your mobile site's content – for example by hosting appropriate robots.txt file on your mobile site.

When a request from a spider comes to your main site, you need to make sure to let it through and not redirect it to the mobile version of the site by mistake.

You can easily configure aiCache to handle the bots as special cases so you don't have to do it in your code.

Supporting different URL structure.

Some site have their desktop and mobile versions served from completely different environments – the code, the backend databases, the datacenters could all be different. But most importantly, the URL structure is likely to be different too.

For example, desktop version of acme.com could serve "US News" section as www.acme.com/US_NEWS . The same section is served by mobile site as m.cnbc.com/content/node_id=123 . Notice how very different the two URL are. Outside of the

home page, which is hopefully served as "/" in both cases, dozens of other URLs pointing to assorted section fronts, product listings, published stories etc , could be completely different.

Now imagine a mobile user sharing a link to m.cnbc.com/content/node_id=123 with a desktop user. When opened in a desktop browser this URL is likely to result in a PNF (404 Page Not Found) error, unless it is somehow intelligently rewritten to www.acme.com/US_NEWS

aiCache offers industry's leading support for seamless device-driven URL rewriting – so the proper version of an article or a section front is delivered to proper devices! As a matter of fact, it is so advanced, you can fold both mobile and desktop sites under one site, while making sure everything works 100% and proper content is served to proper devices !

Supporting unified mobile/desktop site.

Some sites have their desktop and mobile versions served from same content management system (CMS) – *unified* or *folded* under the same www site. Such setup has challenges of its own. To simplify, imagine an iPhone user requesting home page, for example www.acme.com/ Most CMSs would render a simplified version of the home page, taking into amount limited screen size and limited capabilities of the mobile device (iPhone in case, but it could an Android device or Windows mobile).

At about the same time, a different user requests the home page, but this time she is using a desktop browser. In response to the request, the CMS would render out completely different looking page from one that the same CMS delivered in response to the request made from an iPhone.

Likewise, there could be requests for the home page, coming from assorted search bots – in response to which you might want to deliver a different version of content.

Now think about thousands of different URLs that your site provide and needing to tailor output of most of them to the capabilities of the requesting device.

You might have a sinking feeling in your stomach just about now, thinking to yourself – there's no way this could be all handled and cached by aiCache. All of these requests will now have to go straight to my origin servers and there's no way my infrastructure can possibly handle this much traffic !

aiCache offers full support for device-intelligent caching in this scenario, with no custom programming required on your part !

Letting users have a choice.

So it might be clear to most that a lesser mobile device should be served a simplified mobile version of the requested page. What about 7" Android tablet – what version should this one receive – same simplistic page, a full blown desktop version or something in the middle ?

How about 10" Android Tab or an iPad ? What about Windows 8 tablet – these will likely support all of the latest HTML/CSS/JS and Flash standards just as good as their desktop counterparts. In addition to defaulting users to presentation (mobile vs desktop) of your choosing, you can also let users decide what site they want to see.

And you guessed it, aiCache supports that too!

aiCache's method and apparatus of supporting device-specific seamless and transparent content selection, caching and filling .

Here's the logic that aiCache uses to deliver on the Holy Grail of desktop/mobile-and-everything-in-between intelligent content serving.

Before we can conquer, we must divide. There are about 16000 different user agent strings in existence today and the number is growing every day. aiCache can use pattern-driven and/or exact-UA-match-driven to compress this madness down to more manageable number of User-Agent-derived ***tags.***

To do so, you can define, at website level, a series UA *patterns* along with their tags. Likewise, a setting could be set pointing to a file that matches complete UA strings to tags.

For example, you can decide to render your content in 3 different stylings: **mobile_simple**, **tablet** and **default** desktop style. As desktop styling is most common, aiCache doesn't require a custom tag for it, but let's use **mobile_simple** and **tablet** as two other *tags*.

Now, we tell aiCache how to match User-Agent string to each of these tags: 2 explicit tags and one default.

To configure UA-to-tag patterns, simply define them, at website level, via **ua_tag_pattern** For example (incomplete list)

```
ua_tag_pattern     .*BlackBerry8.*    mobile_simple
ua_tag_pattern     .*iPhone.*    mobile_simple
ua_tag_pattern     .*iPad.*    tablet
```

To configure UA-to-tag exact-matching, simply define them in a separate file, where each line contain tag and complete UA, in that order, separated by space. Line starting with # are ignored. For example (incomplete list) place the following into a file called *useragentfile:*

```
# Match firefox to "default" tag
default Mozilla/5.0 (Windows NT 6.1; WOW64; rv:13.0) Gecko/20100101
Firefox/13.0.1
# match Ipad to tablet
tablet Mozilla/5.0 (iPad; U; CPU OS 3_2 like Mac OS X; en-us)
AppleWebKit/531.21.10 (KHTML, like Gecko) version/4.0.4 Mobile/7B367
Safari/531.21.10
```

You can create such file based, for example, on a WURFL database. It is up to you to secure proper right and access to the UA database – some are free, some require a payment etc.

Next, you can process the file using a custom script and output the file format that aiCache expects. For example, using WURFL, you can look at screen size attribute and assign tags based on that value. Possibilities are endless, but again it is up to you to produce the output in the format that aiCache expects and as you can see, the format is very simple

Next we point to that file via **ua_tag_file** setting, at website level. For example (incomplete list)

```
ua_tag_file     useragentfile
```

Now, you need to tell aiCache "run the UA matching logic for these requests" by setting **ua_tag_process** flag at pattern level. This way aiCache runs the matching logic only when told so, instead doing for each and every request. While you might want to run this logic for home page URL, you don't want to run it for thousands of URLs that request auxiliary content – JS, CSS and image files.

The rest of configuration depends on your setup:

Different sites (www.acme.com and m.acme.com), different URL structure.

We'd want to catch what we think are mobile requests, on the "main" site and rewrite-redirect them to the mobile site. Likewise, we want to catch what we think are desktop requests at mobile site and rewrite redirect them to the desktop site.

Let's concentrate on desktop site for now. First let's match, to the best of our abilities, all of the known mobile devices to **mobile** tag. You can use both pattern and exact matching methods as described above. Internet has many mobile-matching patterns, ready for use – but do be warned that they need constant upkeep, due to new devices, new ROMs, new versions of browsers appearing daily.

As mentioned earlier, it is often time *more important* to err on the site of keeping the requests on main site, as opposed to redirecting a request to mobile site.

Next, in the matching pattern section, you can specify an exact match string, that is to be matched against the UA tag (in our case, these will be **mobile**), along with URL rewrite string. For example, we catch requests from mobile devices – by matching to the tag of **mobile**, directed at /news.html and /technews.html on the main site and rewrite/redirect them to point to different urls on the mobile site instead. The setting is called **ua_url_rewrite** and you'd use at pattern level:

```
pattern /news.html simple 30
ua_tag_process
....
ua_url_rewrite mobile .+ http://m.acme.com/render?id=22

pattern /technews.html simple 30
ua_tag_process

....
ua_url_rewrite mobile .+ http://m.acme.com/render?id=31
```

Note that you have full power of regular expressions in the rewrite pattern, so you capture part of the orginal URL and use that in the rewritten URL.

We're not tagging the requests from any other browsers, so they will be processed (cached etc) as usual.

Clearly, you'd want to catch and properly rewrite-redirect all URLs of significance – but not more than that. For example, you would't want to apply this processing to the static content etc, as it is not likely to be ever requested from your main site by any of the mobile browsers.

Different sites (www.acme.com and m.acme.com), same URL structure.

This scenario is rather unlikely, but configuring it is very easy. We still match mobile requests and tag them with **mobile .** The only difference from above is that we don't rewrite the URLs and instead, forward users to the same URLs on different site.

```
pattern /USnews.html simple 30
ua_tag_process

....
ua_url_rewrite mobile .+ http://m.acme.com/USnews.html


pattern /technews.html simple 30
ua_tag_process

....
ua_url_rewrite mobile .+ http://m.acme.com/technews.html
```

Same site (www.acme.com), same URL structure.

Again, configuring it is very easy. We still match mobile requests and tag them with **mobile .** And that is it! aiCache will still cache content as per your configuration. To make sure content

is cached so that differently-tagged versions of it don't collide, the UA **tag** is added to the cached request signature.

To make it easier for the origin servers to decide what version of content to render, aiCache sends the matched tag to the origin servers via X-UA-Rewrite header. Complete User-Agent string is also forwarded to the origin server. It is up you what you want to base your decision making on: the **tag** that aiCache conveniently forward for you, or the entire User-Agent string.

Same site (www.acme.com), different URL structure, different origins.

This setup allows you to fold what presently are two different websites – the main and the mobile, under the same site, even if they are running different CMS and are hosted out of different datacenters

As usual, do your best to match known mobile UA to **mobile** tag. Again, you can have a number of tags and very evolved matching, we're simplifying here.

Next we rewrite the URLs that are different between desktop and mobile URL structure. You can rewrite "behind the scenes" to where visitors are not aware of the rewrites or issue redirect to the proper URLs instead, it is up you. Possibilities are endless.

We also specify a different OS Tags in the rewrites so that mobile content is filled from different origin servers

```
pattern /USnews.html simple 30
ua_tag_process

....
ua_url_rewrite mobile .+ /render?id=123 2
# Notice how OS tag of 2 is specified as last param above


pattern /technews.html simple 30
ua_tag_process

....
ua_url_rewrite mobile .+ /render?id=223 2
# Notice how OS tag of 2 is specified as last param above


# Main site's origin, default tag of 0 implied
origin 1.1.1.1
origin 1.1.1.2

# Mobile site's origin, notice different network and we specify os tag of 2
origin 2.2.2.1 80 2
origin 2.2.2.2 80 2
```

Same site (www.acme.com), same URL structure, different origins.

Very similar to the setups above. Keep the URLs intact, while specifying different os tags.

Overriding TTL based on UA tag.

In the 3 scenarios above, you might want to override the TTL for certain UA tags, possibly in addition to selecting different origin servers. For example, when rendering pages for mobile devices, you might want to render the ads into the page server-side and disable caching for such pages, while still caching the same URLs for desktop browsers, as these render ads "client-side", using Javascript.

The TTL is optional 5th parameter in the **ua_url_rewrite** directive. It is to follow os tag and you must provide os tag, even if it is just the default os tag of 0, if you want to specify the TTL.

For example:

```
# Default caching of 30 secs
pattern /USnews.html simple 30
ua_tag_process

....
ua_url_rewrite mobile .+ /render?id=123 0 10
# But modify to 10 seconds when serving mobile users, use default os tag of
0
```

Dropping requests based on UA tag.

To drop requests, match UA to special tag of **drop.** Enable processing for the URLs where you want to apply this drop logic, via **ua_tag_process** at pattern level.

Simplify UA tagging with default tag.

You can tag requests with special tag of **default**. When such tag is assigned, aiCache treats it as if no tag was assigned. This might simplify you matching logic. For example, you might want to match known desktop user agents to **default** before continuing to match the remaining patterns.

Note that aiCache will first try to use UA pattern, before going to UA tag file.

Letting users have a choice.

You can force-assign a tag to a request by using a cookie, this way aiCache won't attempt to obtain a tag based on the value of the User-Agent string and will use the tag value from the cookie instead.

Use **ua_tag_cookie** setting to specify the cookie name. It is up to you when and how to set this cookie, if ever. When setting the cookie, we recommend using the session cookie that expires when user closes their browser. Otherwise the user will be pegged to a desktop or a mobile site for an extended period of time.

You can provide a link saying "Choose main site" and "Choose mobile site" in the page headers. When clicked, custom JavaScript logic could be then downloaded and executed – that sets appropriate cookie value.

While not obvious, you can delegate setting of the UA tags entirely to your server-side or client-side code, if you so desire. Simply set the **ua_tag_cookie** to the value of your choosing and provide matching tag-driven rewriting rules via **ua_url_rewrite** setting .