I have read this document. I thank Dr Raymond Doctor for preparing it, and for the Devanagari team for providing it for review. It is a thoughtful and broad response to the initial questions, and I believe it is extremely useful for the purposes of this project.

Many thanks for your kind remarks as well as the detailed reading of the white paper, which must have taken quite some time. I have tried to answer to some of your comments not as a justification but more as a means of explaining the underlying motivation. My comments are provided in red for easy reading.

I have some detailed comments, which I lay out below. I have keyed my comments to the section numbers in the paper.

2.1, Abstract Character.

This section argues that the term Abstract Character can be understood as meaning the same as Glyph. But the Unicode Standard explicitly denies that these are the same:

- An abstract character has no concrete form and should not be confused with a glyph.

- An abstract character does not necessarily correspond to what a user thinks of as a “character” and should not be confused with a grapheme. (see Unicode Standard 6.0, 3.4 D 7, p 66)

Since one goal is to cleave to external sources for definitions, I think we must say here that Glyph and Abstract Character are not the same. Moreover, the example here is in terms of the bilabial unvoiced stop /p/. I think that it is important to recognize that there is a siginficant link between a language and its writing system(s), but I am under the impression that the Unicode Technical Committee (henceforth, UTC) tries very hard to stay out of making linguistic decisions and to stick on the side of encoding writing systems. I suspect, therefore, that an abstract character is more like "the thing that looks like A with a round thingy over the top", which happens to be encoded in Unicode as U+00C5 LATIN CAPITAL LETTER A WITH RING ABOVE, U+0041 LATIN CAPITAL LETTER A plus U+030A COMBINING RING ABOVE, and U+212B ANGSTROM SIGN.

Nicholas Ostler had raised the same issue. All I can state is what I wrote in my response to his comments: I quote:

“Both you and Andrew have pointed this anomaly out. I have based myself on draft-ietf-appsawg-rfc3536bis-02, which states as under

> *glyph*
> *A glyph is an abstract form that represents one or more glyph images. The term "glyph" is often a synonym for glyph image, which is the actual, concrete image of a glyph representation having been rasterized or otherwise imaged onto some display surface. In displaying character data, one or more glyphs may be*
> *selected to depict a particular character. These glyphs are selected by a rendering engine during composition and layout processing. <UNICODE>*

The way I read (past tense) this text has led me to make this statement. However if I have misunderstood the text, I stand corrected.”

2.1, Language Character Repertoire

I think the discussion here is entirely congruent with the reason the JET guidelines originally conceived of variants in terms of a language rather than a script. But there is a difficult problem raised as a result, which is that in the context of zone policy, one has to make choices that resolve registration request conflicts in favour of one language or another. In the root zone (as well as in gTLDs), such a policy could be problematic, which is why there has been considerable effort to reduce issues to a script rather than a language. I'm not sure what to do about this sort of problem: any inter-language problems are going to be extremely difficult to address in these kinds of zones.

The same issue was raised by Nicholas and while I agree in principle to what both you and he state, I still feel that giving priority to Script over language was one of the biggest fallacies of Unicode. The confusion in the Arabic code-page could have been avoided, had the code-pages been language driven rather than a single script-driven code page. The same is the case with Devaanagari and many other languages to follow which use one single page.

I guess this is the linguist in me rather than the technical person arguing, but for me language came first, script followed; although I see that IDN's are all script driven and not language driven.

Just for your reference, am reproducing what I wrote to Nicholas Ostler:

> *The white paper was never meant to be an "appeal" for Language Tags. However the issue needs to be discussed and sorted out once and for all. It is a social as well as cultural issue to degrade language and place it on a rank lower to script, with script gaining primacy. While Unicode's preoccupation is with Script (and rightly so), somewhere language has to be given priority or else languages sharing a script. There are today 71936894 speakers of Marathi and over 366 million first-language speakers; an additional 121 million second-language speakers for Hindi, not to mention the 7 other official languages sharing Devanagari[1]. While I agree that it could become a « political football » I also feel that mixing all languages sharing a common script into one melting pot is just as bad a policy.*

2.1, Summing-up [some discussion of normalization]

Many thanks for this analysis. I agree with you completely but I am trying to safeguard the common user since all browsers do not satisfy either Unicode nor ICANN norms(see your comment in 2.3 Table 4) , I will answer all the comments at the end, except some small comments in cases which need commenting.

The discussion here suggests that it is important to discuss normalization and to test browsers to check that they actually perform such normalization. This issue has also come up in the Arabic VIP. I think some important distinctions are needed.

First, we need to ascertain whether we are talking about Unicode normalization or something else. Unicode normalization has four forms: NFC, NFKC, NFD, and NFKD. I could explain all of that here, except that I would just repeat what is in http://www.unicode.org/reports/tr15/tr15-33.html. If you haven't read and understood that, however, none of what I am about to say will make any sense.

I have read the RFC in question. The main point I wanted to make was that although there exist normative structures defined both by Unicode as well as partly by ICANN, browser behavior is controlled by the browser developer and as our policy white paper which is under development shows, different browers behave quite insidiously. To rephrase the popular expression Unicode proposes, the browser disposes (unfortunately) This is why we take Normalisation rather seriously. Moreover I see that IDNA2008 (which I should have read, I confess) does handle this to a large extent, but…..
(Please see the end of this discussion for some browser/font behavior instances).

There remain, as I understrand it, some kinds of normalization that are not actually covered by Unicode normalization. This normalization is generally linguistically sensitive. So, for instance, in some Arabic-script using writing, there are modifier dots that have no effect on the meaning of a piece of text, and can be written or not as a writer likes. (But I am told that they are not handled by the relevant Unicode normalization in this case. More below.) I was under the impression, after the Pune meeting, that no Devanagari-using language has this issue, but as I speak none I'm hardly going to be useful in coming up with counterexamples.

Agreed, among Devanagri driven languages there are no counter-examples but I fall back on y argument given above. An example will illustrate the insidousness of browser behaviour:

[1] http://www.oclc.org/languagesets/educational/languages/india.htm

Now, as for the normalization we need, we have two kinds that affect us. IDNA2003 -- specifically, the Nameprep step -- uses NFKC. This is not optional, but the use of compatibility equivalents was one of the things that people didn't like in IDNA2003 (because it meant that you could not be sure that the transformation to Punycode and back again didn't lose any data: things like final form sigma would get mapped away).

IDNA2008 solves this in two ways. First, it does not itself do any mapping. However, it defines U-label such that only things in NFC form can be U-labels. How the string in question gets into NFC is not part of the protocol, and is something that is supposed to happen before IDNA2008 takes over. Second, IDNA2008 disallows any character that is not stable when performing NFKC and caseFold (the actual rule is B: toNFKC(toCaseFold(toNFKC(cp))) != cp in RFC 5892 section 2.2. See http://tools.ietf.org/html/rfc5892#section-2.2). The goal is, roughly, to make sure that only stable characters are candidates for being used in the protocol, but otherwise to stick to NFC.

It might seem tempting, then, to try to ensure that applications are using just the right normalization in all cases, and to try to add policy where it is clear that some clients are not following the protocol. There are, however, two problems with this approach.

The first problem is that, if a client isn't using NFC for its U-labels, then it just isn't implementing IDNA2008. Similarly, if a client isn't performing NFKC on its strings, then it just isn't implementing IDNA2003. It is probably wise to have registration policies with some sort of sunset clause that results in the most restrictive policy for the intersection of these two protocols (effectively, one wants a policy that recognizes that IDNA2003 is still more widely deployed); but that is not the same as saying that one is going to test for bugs and implement around them. Testing for nonconforming client software on the Internet is shooting fish in a barrel, and if you try to make policy around the bugs in Internet client software, you will quickly go mad. But more importantly, at least some clients can accept and use raw UTF-8 as part of their DNS lookups, and that is regarded as a feature rather than a bug by the users. Such clients aren't implementing IDNA at all, but are using UTF-8 in the DNS. This is perfectly legal: DNS labels are not LDH-labels, but bits. It is only by convention (really, really widely held and widely regarded as protocol, but still a convention) that DNS labels are restricted to letters, digits, and hyphen.

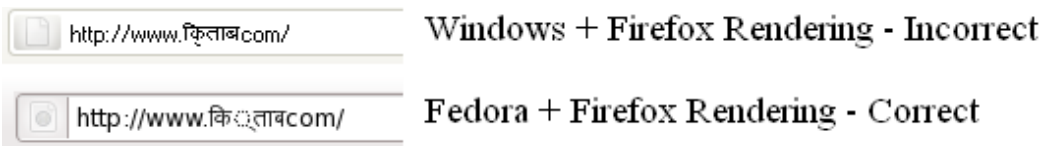
MY comment to the above. Once again while I agree with you completely, I return to my main fear: the browser. Hence the need to involve actively browser developers (which we tried at the Pune meet) in the process.

The issues related to these characteristics of the browsers belong to two broad categories:

1. Rendering Engine related issues
2. Font related issues

Rendering Engine related issues :

Whenever some text is submitted to a Unicode Enabled application, the rendering engine breaks this text in the form of syllables. These syllable formation rules have not been standardized, nor has Unicode given any specific rules pertaining to the same. Thus the behavior of different rendering engines is different and depends on the understanding of the language/script of the implementing body which seldom is perfect. This is exemplified in the cases given below:



Font related issues :

In case of rendering of Domain Names in browsers, font that gets applied on the domain name in address bar of the browser plays major role. Each operating system has a specific font which act as a default font for every script/language the OS supports. The browser uses default font provided by the OS for displaying the domain name in the address bar.

Similar to the rendering engine, the font implementation also varies from vendor to vendor. And thus the same Domain Name can be seen differently depending on the font properties, orthography adopted by the font, hinting, weight, kerning etc. There is a strong need for a central authority which will bring consensus in these implementations. (optimistic remark!!!!)

I trust these examples will justify my fear of “malfaisance” (which I hope is unfounded). The Devanagari policy white paper under development, provides as an annexure six PDF files of analysis of Browser behavior (different browsers x different OS: Linux, Mac, Windows [Chrome yet to be tested]) and compliance to Unicode as well as ICANN IDNA2008

Your comments in 2.3 Table 4 below only increase my worry since we have no control over the Browser behavior nor the rendering Engine for Indic.

2.3.2.1 Orthographic alternanats

While I like the term "alternants" as a term we might want to adopt, I am having a hard time understanding why the examples we see in this section do not lead us directly to say, "Color and colour are alternants, and by alternant policy the registrant of .color must also received .colour too." If there _is_ no principled reason why not, then why isn't the converse true. That is, if the case in English works out the same, what is so special about the non-English cases that the answer should not be, "If you want two labels, register (and, I presume, pay for) two labels"?

Agreed, point well taken. This is what was discussed at the Pune meet (the paper was written in June and submitted on the very day of the Pune meet) and the point you have raised was acceptable to all members present at the Pune meet..

2.3, Table 4

First, I should note that I think this table is very helpful. I appreciate its development.
In the first row of this table, item 2, there is the observation that new versions of Unicode will cause problems. This is true, but it is true _forever_ under IDNA2008. That was part of the point. Registries will always need to have sunrise provisions to cope with the fact that new characters will show up in later Unicode versions. A policy, of course, could be, "We won't register them," but then you still have the problem of code points that are PVALID under an earlier version of Unicode, but become DISALLOWED under later versions.

This is a feature, and not a bug, of IDNA2008. IDNA2003 was pegged to a particular version of Unicode, but that didn't work.
When you upgrade your computer and get a new version of Unicode, you can't tell which version of Unicode you're using. So many of the putative IDNA2003-implementing clients out there aren't actually implementing any standardized protocol at all, because they have a later version of Unicode and can't tell. The only way to solve this issue is to forbid UTC from changing Unicode, and we can't do that.

I entirely agree with your observation. In fact this has led to creation of legacy methods of inputting and constitutes at least for Indic scripts once of the main set of variants.

However we have to ensure that the policy for a given language is Latest Unicode compliant. The only bright point is that as far as I know all pertinent characters (for Indian scripts) with the exception of a couple are present in Unicode and hence the policy will not undergo much mutation.

A later row of the table discusses case marking. I think this is not a problem for anyone, even though some think it is. IDNA2008 is designed specifically so that upper-case letters aren't allowed in U-labels. It's true that this is frustrating when compared with the special processing of A-Z. There's nothing that can be done about that special processing, however, because it's a built in part of the DNS protocol.
This point was raised to show that there are things which need to be accepted. The French (to the highest degree), English and Germans(to a lesser degree) have accepted the absence of Apostrophe (Consider that in a paragraph of a French newspaper of around 300 words, at least 80 have apostrophes.). Case is an issue that German and other case-sensitive languages can do without.

The last line of this table is potentially useful input to a specific policy, but I do not know how any general policy can be made on the basis of browser issues. This is for two reasons.
First, there is absolutely nothing that we can do about different browsers on the Internet. We cannot detect what browsers people are using when they do DNS lookups, so we cannot react usefully to such differences. Second, it is not only browsers that are relevant. DNS names are used all over the place: in mail agents, in SIP VOIP clients, in system configuration files that only system administrators ever see, in system logs, and in machine-to-machine transactions in which no human is directly involved most of the time.
This touches 2.1 (unless I am off tangent) Please see the comment there. Within a holistic picture of IDN's the weakest link is the browser and we can really not do much about it.
2.3.3, Problem of the Preferred Variant

In this section we have a proposal for a way of identifying a number of alternants for one another, and then picking which of those is to be allocated and delegated. Suppose we have a set of alternants, {a1..an}. These are each expressions of the Archivariant. The central proposal of this section is that, on registration request for any one of {a1..an}, the registry is checked for conflicting members of any other set. If there is no conflict, the entire set {a1..an} is allocated to the registrant. This allocation does not, however, automatically result in delegation. At this point, it is a matter of registry policy how many of the individual alternants, if any, are to be registered; and what that will cost.

Agreed. This could be a possible solution and in my opinion the only viable one to the issue

I would like, however, to distinguish a couple of terms here that I think are used in more than one way. I want to say that "bundling" is not what has happened in the case of www.color.com and www.colour.com: those are actually just separate registrations that happen to resolve to the same IP address (note that they don't go to "the same page". One of them is a redirect to the other, and if you visit http://www.colour.com you will be redirected to http://www.color.com. This is an http-level redirect and not something done in the DNS). I think it would be better to use the term "bundling" for the case where the registry automatically performs the linking of the different registrations together somehow, as happens here when the allocation of all members of {a1..an} go to the same registrant. Point well taken and accepted. Many thanks for the useful suggestion

By the same token, I think it would be good to distinguish reserved and blocked names. We might want to say that a reserved name is not allocated at all. All two-character domains in the top level that are not actually registered are in fact reserved: nobody is allowed to register them until ISO allocates a country code. But a name that is allocated but not delegated is blocked: nobody else can register the name because, in effect, it is already registered.

Point well taken and accepted. Many thanks for the useful suggestion

I hope these comments are useful.

Extremely useful and I hope to all others who will read this thread. Many thanks to you and Nicholas Ostler for taking time off to read through the doc and give such useful inputs.