ausregistry INTERNATIONAL

The TLD Specialists

Experiences Building an IDN Domain Name Registry

Chris Wright
CTO - AusRegistry International
ICANN no. 39, Cartagena, Colombia
9th December 2010

AusRegistry International

- Located in Melbourne, Australia
  - Involved in Domain Name Industry since 1999
  - ICANN Accredited Registrar since 2000
  - .au Registry Operator since 2002
- Domain Name Registry Services
  - Registry Systems and Software Provider
  - Consultancy Services
  - Our software and consultancy services have been used by several other TLDs including IDN enabled ccTLDs

Overview

- You may be considering applying for an IDN gTLD, or for an IDN under the ccTLD fast track program
- Alternatively you may want to allow IDNs under your ASCII TLD
- At some point, it is likely that you will need to begin supporting IDNs in your Registry Solution
  - Some notes from our experiences
  - Share what we have learnt
  - High-Level overview of part of our solution
  - Some interesting points to think about

So why did we implement IDNs?

- We supply Registry software & services to other TLDs
- We need to remain innovative and up-to-date
- We need to provide what our customers want
- We believe IDNs are integral to furthering the reach of the Internet

Our goals

- Implement IDNs in an RFC compliant way
- Do so generically and flexibly
- Ensure implementation is easily maintainable
- Ensure implementation may be customised if required by customers
- Configurable to suit various local policies without sacrificing performance, security or stability

As responsible TLD managers we must...

- Minimise public, Registrant and Registrar confusion
- Protect against phishing and other misdirection style attacks
- Maintain
  - high security standards
  - high performance standards
  - policy rich controls (where relevant)
- Protect the reputation of our namespace
- Manage our TLD the way an important asset should be managed

With that in mind...

- Some of the more important aspects to consider in a more responsible implementation of IDNs include:
  - Developing IDN specific policy
  - Fully Internationalising your Registry Platform
  - Blocking similar registrations
  - Bundles
  - Variants of your IDN zone
  - Effects on DNS
  - Security considerations
  - Performance impacts
  - Effects on Registrars, Registrants and end users
  - Implications for Registry Website & other interfaces

Registry Implementation

IDNA – Internationalised Domain Names in Applications

- Whilst it is a protocol in the dictionary definition of the term
- It is NOT a protocol in the sense that DNS, HTTP or EPP are protocols
- It's essentially three main things:
  - A way of converting a Unicode string into an ASCII string so that it can be used in the DNS protocol
  - A sequence of steps that a Registry must follow before accepting a name for registration
  - A sequence of steps that an Application must follow when looking up a name in the DNS

Why must we understand all of IDNA?

- IDNA assumes any required pre-processing has been performed by Registrars including:
  - ensuring the name is in Unicode NFC form
  - any other local processing that may be required (but is not defined in the IDNA specification) eg. case folding / lower casing

  However…

- To maintain the integrity of the Registry it is important to check that all rules have been followed.

Steps a Registry Must Follow

- Verify that the name is in NFC form, reject if not
- If domain provided in A-label format, generate U-label version using punycode
- If domain provided in U-label form it is strongly advised not to accept it to avoid any ambiguities
- Validate that both A-label form and U-label form are in fact related, reject if not
- Reject any name with leading combining marks
- Reject any name that contains consecutive hyphens in the 3rd and 4th positions (in the U-label)

Steps a Registry Must Follow (cont.)

- Verify that the domain contains only valid code points as defined by the IDNA standards, reject if it doesn't
- Apply the joiner rules (context j rules), reject if these rules fail
- Verify that for each context o code point, a rule exists in the standard and that when the rule is applied the domain name is still valid, reject if any of these rules fail
- If the domain contains any right-to-left characters apply the BIDI rules, reject if any fail

Basic Implementation Summary

- Implementing these steps is relatively simple as they are well defined in the protocol
- A simple implementation of these can be achieved very quickly
- However there are many methods that can be used to efficiently implement these steps in an elegant manner

Now that we have a valid IDN name

What else do we need to do?

Zone specific processing (policy)

- What needs to be done, how and why it should be done, is not documented anywhere
- However there are some VERY important steps that should be followed:
  - Checking for duplicate names (including complex equivalencies such as those created by the use of combining marks etc. – think variants or bundles)
  - Apply local policies
  - Validating against our language rules
  - Checking reserved lists

Checking for Duplicate Names

- Duplicate domains are domains that are considered ‘the same’ as one another
- For ASCII domains ‘the same’ is simply a case insensitive compare, e.g.
  - example.com
  - Example.com
  - EXAMPLE.com
  - ExAmPlE.com
- In this particular case this is enforced by the DNS protocol

However with IDNs...

- There are many more cases where duplicate registrations may exist e.g.

Convention, visually confusing or historic	**Non-visual reasons**	**Technical reasons**
café.com cafe.com	١١١١١.com 11111.com	أ.com (U+0627,U+0654) أ.com (U+0623)

- No single, simple rule can be applied, i.e. just lower casing does not help

Duplicate Example

- ASCII John’s Cafe (because of convention)
  - johnscafe.com ← Sacrificing the é
- IDN John’s Café (because now we can)
  - johnscafé.com
- Shouldn’t the two be considered the same name? i.e. Duplicates?

Implementing duplicates – The variant generation method

- The idea that one character is a variant of another character e.g.
  - ‘e’ and ‘é’
- When a domain is created using one representation the other representation is also considered registered or ‘blocked’
  - cafe.com
  - café.com
- This is done by ‘calculating’ all of the variants

Implementing duplicates – The variant generation method (cont.)

- This can happen at time of registration in which case all the variants are then stored for later comparisons

  or

- This can happen on input to all commands (obviously very inefficient)

Implementing duplicates – The variant generation method

- Calculating and storing duplicates introduces overhead
- Consider a name where there is only one variation of several of the characters in the name e.g.

e → é

cafeeeeeeeeeeeeeeee.com
cafeeeeeeeeeeeeeeeé.com
cafeeeeeeeeeeeeeeée.com
cafeeeeeeeeeeeeeeéé.com
.
.
caféééééééééééééééé.com

In this fictitious case there is 2 ^ 16 combinations i.e. 65,536 variations

Implementing duplicates – The variant generation method

- If we have a domain name with just 32 characters in it, each with one variant we would have over 4 billion variants
- There has to be a better way!
- And there is...

Implementing duplicates – The canonical method

- Canonical representation of domain names isn’t new
- ASCII domain names use the concept, its built into the protocol - lowercase
- The overall premise is that we assign each character a canonical form

What do we mean by character?

- A character, for the sake of this discussion, is a sequence of one or more code points that represents one particular component of a word.

a

is a character

أ

(single code point) is a character

أ

(multiple code points) is a character

Assigning canonical form

- Each character is assigned a canonical form
- You can think of it as the base form of the character
- In most cases it just be the character itself
- Sometimes another code point entirely
- Sometimes nothing at all
- The actual character chosen doesn't really matter – its just a concept

Using the canonical form

- Define all canonical mappings for your zone
- Perform a simple substitution of each character for its canonical equivalent
  - This generates the canonical form of the label being registered
- Use this canonical form of the label as the unique key for the domain registration representing ALL forms of the domain name (without each of those forms having to be generated and/or stored)

Using the canonical form

- In our zone we allow the following characters with the canonical mappings listed:

a	→	a
c	→	c
e	→	e
é	→	e
f	→	f

Using the canonical form – An example

- We register the name cafe’.com and compute the canonical form

  café.com → cafe.com

- The domain is café.com but the unique label is cafe. So when someone tries to register cafe.com we compute the canonical form

  cafe.com -> cafe.com

- But this will NOT be allowed as a domain with that canonical label is already registered

Using the canonical form – Another example

- The name cafeeeeeeeeeeeeeeee.com maps to
  cafeeeeeeeeeeeeeeee.com → cafeeeeeeeeeeeeeeee.com

  as does
  caféeéeéeéeéeéeéeée.com → cafeeeeeeeeeeeeeeee.com

  as does
  caféééééééééééééééé.com → cafeeeeeeeeeeeeeeee.com

- So by storing the canonical form and checking all new registration attempts against it we have blocked all other registrations without actually having to calculate them all!

More on canonical...

- Mapping names to a canonical form is nothing new
  - Exactly what happens in existing domain name registries when we lower case names
  - Implied canonical mapping between upper case and lower case (implemented by a function)
  - Just also happens to be enforced by the DNS protocol itself

Making canonical work for us

- Just as we lower case the domain name provided to Registry functions such as:
  - Search
  - Domain Check / Update
  - Reserved List Matching
  - WHOIS
  - Etc.
- If we apply the canonical mapping to IDN names passed to registry functions everything just works

Benefits of using canonical

- It just works
- Its linear time regardless of the size of the domain names and desired variant configuration
- It provides speed and efficiency benefits, especially when compared to variant generation methods
- It saves space and memory
- Its a simple algorithm that is easy to implement, less error prone and easier to optimise

Bundles

Why Bundles?

- Sometimes blocking is just not enough
- In some scenarios it make sense that a Registrant can make use of multiple versions of a name e.g.
  - cafe.com
  - café.com

In simple terms...

- Its the same as the generating variant model, so it has the same issues
  - If in our zone configuration we said that we wanted the following character variant ‘provisioned’ or used to create ‘bundles’

    ١ → 1

  - And then we registered the name

    ١١١١١١١١.com

  - We still end up with...

Example

- The following variants to be provisioned

  ١١١١١١١١.com
  ١١١١١١١1.com
  ١١١١١١1١.com
  ١١١١١١11.com
  .
  .
  .
  11111111.com

- Which in this case would be 256 variants to be calculated, stored and provisioned in the zone file
- Canonical mappings can’t help us here

Bundles (cont.)

- Character variants for blocking of registrations make all combinations important
- ... But when considering bundling.. If we look at the reason people desire variants, another option is presented

Continuing our example...

- In this case it makes sense that someone may enter either of the following domains:

  ١١١١١١١١.com
  11111111.com

- But does it really make sense that someone would type the following domains names:

  ١1١1١1١1.com
  ١١١١1111.com

- All combinations need to be blocked (which canonical mappings will do) , yet only two out of the 256 variants provisioned in the DNS are required.

Introducing Mutual Exclusion

Mutual Exclusion

- Mutual exclusion is not a new concept, it is used everywhere in modern-day life
- If we apply it to domain name variants we can achieve the desired behaviour e.g.

Primary Grouping	**Sub-Grouping**
Numerals	English Numerals e.g. 1,2,3,4,5...
	Arabic Numerals e.g. ١٢٣٤٥...

So the rule is…

- If a domain name contains any characters that are in one sub-group, it is not allowed to contain any characters from other sub-groups of the same primary group to be provisioned in the DNS
- i.e. The characters in one sub-group are mutually exclusive to the characters in another subgroup

Returning to our example...

Primary Grouping	Sub-Grouping
Numerals	English Numerals e.g. 1,2,3,4,5...
	Arabic Numerals e.g. ١٢٣٤٥...

- These are allowed:

  ١١١١١١١١.com

  11111111.com

- But these are not:

  ١1١1١1١1.com

  ١١١١1111.com

Other bundling considerations

- Allowing Registrants to turn parts of a bundle off or on
  - How?
- Impacts on other services offered
  - e.g. DNSSEC
- Charging model
  - Should there be one?
- Flow on effects to accounting and reporting
  - Is a bundle of three domains one registration or three?

Validating Local Language Rules

What are local language rules?

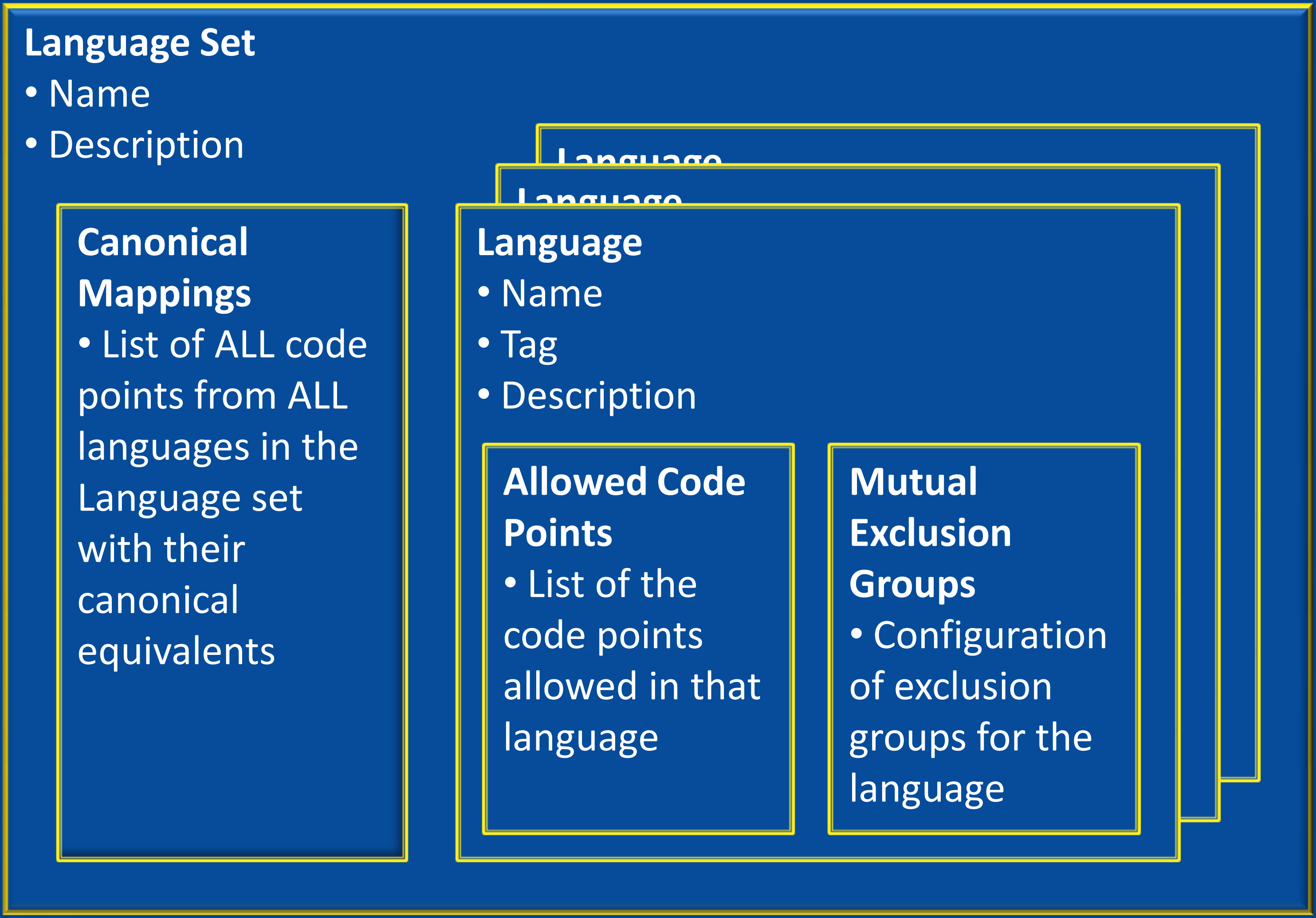
- In short, they are and can be anything
  - Which unicode code points make up the language
  - Handling of edge cases
    - ae → æ
    - ss → ß
    - Final form sigma
  - and so on
- Important that the business rule engine is flexible and customisable enough to handle these requirements

Putting it all together

How can we represent IDN configuration?

- In a generic way
- That reduces the management and configuration overhead
- That is easily understood by non-technical people

Our Solution



Language Set
- Name
- Description

Canonical Mappings
- List of ALL code points from ALL languages in the Language set with their canonical equivalents

Language
- Name
- Tag
- Description

Allowed Code Points
- List of the code points allowed in that language

Mutual Exclusion Groups
- Configuration of exclusion groups for the language

Just the tip of the iceberg!

Performance Implications

- TLD Registries include performance and SLTs
- Validation rules and cross checking that now needs to be performed has to be implemented as streamlined as possible, especially when performing domain availability checks
- A lot of ASCII 'tricks' or optimisations are invalidated e.g.
  - Byte size != string length
  - Byte equivalency is not the only case of equality any more
  - Lower casing is not the only pre-processing required for uniqueness checks
- Multi-zone registries with mixed IDN and non-IDN zones will even incur a performance hit on the non-IDN enabled zones as certain checks still need to be performed

Effects on Registrars, Registrants and End Users

- It is different to ASCII domains
- Registrars have a harder job to do now
  - Interpret what the Registrant wants
  - Turn it into something remotely protocol valid (to map or not to map?)
  - Explain all of this to the Registrant
- Provide tools to Registrars
- Ensure consistent message to Registrants and end users

Many other areas to consider

- IDNA – Internationalised Domain Names in Applications
  - Registry Systems are also Applications in fact they are a collection of many different applications
  - We have to implement the application and the Registry portion of IDNA
- Changing Language rules in an already established zone!
- Effects on EPP
  - Command Extensions
  - Protocol Extensions
  - Returning Variants
- Security Considerations
  - Puny Code Overload
  - Puny Code Reverse Engineering
  - Handling of supplementary characters

Many more areas to consider

- Internationalising your Registry
- Unicode versions understood by software in use
- Registrars, Registrants
- Effects on DNS
  - Increase in zonefile size
  - DNAME vs NS records
  - Increase in complexities
- Infrastructure Requirements
- IDNs, variants & DNSSEC

IDNs are hard (to do right)...

- However…
- There are many creative and innovative solutions to all of the issues I have mentioned
- Start experimenting & share knowledge
- Help is out there – come and see us

ausregistry INTERNATIONAL

The TLD Specialists