Umlaut-Domänen unter Linux

Zeichenkünstler

Das globale Dorf ist seit dem 1. März ein Stück weltoffener geworden: In Domainnamen sind jetzt deutsche Umlaute sowie asiatische und arabische Schriftzeichen erlaubt. Damit eröffnen sich für Entwickler und Benutzer neue Möglichkeiten, es ergeben sich aber auch neue Probleme. Thomas Drilling, Barbara Grün, Ronald Woelfel

Hinter dem, was die Internetprovider hierzulande einfach Umlaut-Domain nennen, verbirgt sich ein komplexes Verfahren, das weit über die Kodierung der deutschen ä-, ö- und ü-Zeichen hinausgeht. Die internationalisierten Domainnamen (IDN, Internationalized Domain Name) reißen allerdings nicht nur alte Barrieren ein, sie schaffen leider auch neue. Für Linux-Anwender – und nicht nur für die – stellen sich damit wichtige Fragen: Welche Anwendungen sind bereits IDN-tauglich? Welche werden es in absehbarer Zeit sein? Wo hilft nur der Griff in die Trickkiste?

Bisher erlaubte der RFC 1034 für Domainnamen nur die Buchstaben A bis Z, die Ziffern und das Minuszeichen aus dem Ascii-Zeichensatz. Nun unterstützen die Registrare verschiedener Toplevel-Domains auch Namen, die Unicode enthalten. Dabei stehen theoretisch rund 95000 unterschiedliche Zeichen zur Auswahl.

Praktisch ist der freie Zeichenvorrat im deutschsprachigen Raum allerdings begrenzt. So hat das deutsche DENIC nur genau 92 zusätzliche Zeichen aus den Codesets Latin-1-Supplement und Latin-Extended-A zugelassen [9], die Schweizer sind noch viel vorsichtiger und begnügen sich vorerst mit 31 neuen Zeichen aus dem Latin-1-Supplement.

Diese Einschränkung verhindert eine Reihe potenzieller Fehler von vornherein. So mögen japanische Schriftzeichen als Blickfang inmitten lateinischer Lettern zwar reizvoll sein, doch wären für den Benutzer Probleme mit fehlenden Fonts programmiert. Außerdem vermindert sich die Gefahr, grafisch ähnliche Zeichen zu verwechseln. Nicht alles, was möglich wäre, hat wirklich Sinn.

Für Serverdienste bleibt alles beim Alten

Administratoren dürfen sich freuen: Sie brauchen weder in den sensiblen DNS-Dienst, noch in die Konfiguration des Webservers einzugreifen, denn diese Dienste operieren auch künftig nur mit ihrem beschränkten Ascii-Code. Umstellen müssen sich dagegen die Clients, in erster Linie die Browser. Von ihnen sind bisher beispielsweise Netscape (ab 7.1), Mozilla (ab 1.4), Konqueror (ab 3.2 mit der GNU IDN-Library) und Galeon (ab 1.3.14) fit für IDN.

Tippt der Besitzer eines derart befähigten Browsers eine Unicode-URL ein, wandelt das Programm sie in einem mehrstufigen Prozess in ein spezielles Ascii-Format und befragt damit den Nameserver nach der zugehörigen IP. So kommen Web- wie Nameserver nur mit Namen in Kontakt, die aus dem bisher gültigen Zeichenvorrat der 37 Ascii-Zeichen gebildet wurden.

Aus einem Domainnamen entstehen auf diese Weise zwei Versionen, die je nach Verwendungszweck eingesetzt werden: Benutzer und IDN-fähige Applikationen arbeiten mit der Unicode-Version – alle anderen Anwendungen werden mit der daraus berechneten Ascii-Form bedient. (siehe Abbildung 1).

Dieses Konzept mündete im RFC 3490: Internationalizing Domain Names in Applications (IDNA). Der im März 2003 verabschiedete Standard sieht vor, dass jener Teil eines Domainnamens, der Unicode-Zeichen enthält, mit dem Präfix »xn--« gekennzeichnet wird. Daher verweigert die zentralen Registrierungsstelle für DE-Domains DENIC bereits seit Anfang 2002 Domains mit jeweils einem Minuszeichen an dritter und vierter Stelle.

Punycode

Das Beispiel aus Abbildung 2 zeigt, wie die Kodierung der Sonderzeichen genau bewerkstelligt wird: Aus dem Domainnamen »grün-wölfel.de« wird dabei die Ascii-Variante »xn-grn-wlfel-47a6d.de«.

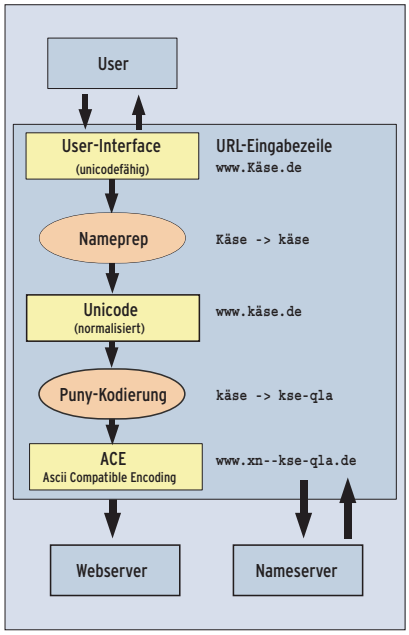

Abbildung 1: Der lange Marsch vom User-Interface bis zum Webserver: IDN-fähige Anwendungen müssen die gesamte Übersetzung in das Ascii-Format selbst übernehmen. Erst der fünfte Schritt liefert die Zeichenkette, die ein Nameserver in eine IP-Adresse umsetzen kann.

Dieser String, der als ACE-Form (Ascii Compatible Encoding) bezeichnet wird, kommt bei der Namensauflösung zum Zug. Er beginnt mit dem erwähnten Präfix, gefolgt von den Zeichen, die keiner Umschreibung bedürfen.

Die Sonderzeichen und deren Position in der Zeichenkette kodiert eine besondere Ziffern-Buchstaben-Kombination (47a6d), die ein Minuszeichen abtrennt. Die speziell für IDN-Domains entwickelte Kodierungsvorschrift RFC 3492 **[6]** – der so genannte Punycode – gründet sich auf dem allgemeineren Bootstring-Algorithmus. Der Informatiker erwartet von diesem Code Eineindeutigkeit (Bijektivität), die gewährleistet, dass jedem Punycode nur genau ein Ascii-Pendant entspricht (und umgekehrt). Der Anwender möchte, dass die ACE-Form des Domainnamens nicht zu lang wird.

Punycode erfüllt beide Anforderungen. Lange Namen wären nicht nur dort unbequem, wo der Benutzer sie eintippen muss, sie liefen auch Gefahr, mit der Obergrenze von maximal 63 Zeichen Länge zu kollidieren. Ob dieses Limit eingehalten wird, erweist sich erst nach der Konvertierung, für die sich geeignete CGIs oder Applets auf vielen Webseiten finden (darunter auch der des DENIC). Das weiter unten vorgestellte Kommandozeilen-Tool IDN bietet diesen Service ebenfalls.

Vorwäsche

Noch vor der Kodierung mit dem Punycode-Algorithmus werden die Namen normalisiert. Die dafür geltenden Regeln definiert der RFC 3491 mit seinen Anhängen. Diese Namepreparation (Nameprep) wandelt alle Groß- in Kleinbuchstaben. Außerdem fasst sie aufeinander folgende Zeichen dort, wo das möglich ist, zu einem zusammen. Das betrifft beispielsweise die in der Schweiz gebräuchliche Umschreibung der deutschen Umlaute mit nachfolgenden Pünktchen. Damit wird aus »U + 0075 U + 0308« ein »U + 00FC«. Zusätzlich löst das so genannte Casemapping (oder Casefolding) Ligaturen wie das deutsche ß in einzelne Zeichen auf. Deshalb ist das ß in Domainnamen tabu. Wie der Algorithmus genau funktioniert, beschreibt der Kasten „Punycode im Detail“.

Wer eigene Applikationen auf IDN-Tauglichkeit prüfen möchte, muss eine Umlaut-Domain ordern. Zu ersten Tests genügt es aber auch schon, für eine vorhandene Ascii-Domain eine Subdomain mit dem »xn--«-Präfix und dem daran anschließenden Punycode-String anzulegen. Wie auch immer: Mit solchen Unicode-Domains kommen alle Anwendungen auch ohne Update klar – wenn sie die Punycode-Darstellung verwenden (im Beispiel: »http://www.xn-grn-wlfel-47a6d.de/«).

IDN unter Linux

Doch wie kommt man zu einem solchen ACE-String? Eine Alternative sind die schon erwähnten Webfrontends, eine zweite bietet das Kommandozeilen-Tool IDN. Es kann die erläuterten Konvertierungsschritte entweder einzeln ausführen oder in einem Zug URLs in beide Richtungen konvertieren: »idn --idna-to-ascii www.grün-wölfel.de« ergibt: »www.xn-grn-wlfel-47a6d.de«. Umgekehrt entstünde daraus mit »--idna-to-unicode« wieder das Unicode-Unikat »www.grün-wölfel.de«.

Sucht man die Ascii-Darstellung eines einzelnen Unicode-Zeichens oder die Unicode-Codepoints der Ausgabe eines mit Nameprep normalisierten Namensbestandteils, hilft der Debug-Modus von IDN weiter. »idn --debug« liefert ein Array, bei dem jedes Feld den Unicode-Codepoint der eingegebenen und der ausgegebenen Zeichen anzeigt:

```
input[0] = U+0078
input[1] = U+006e
input[2] = U+002d
input[3] = U+002d
```

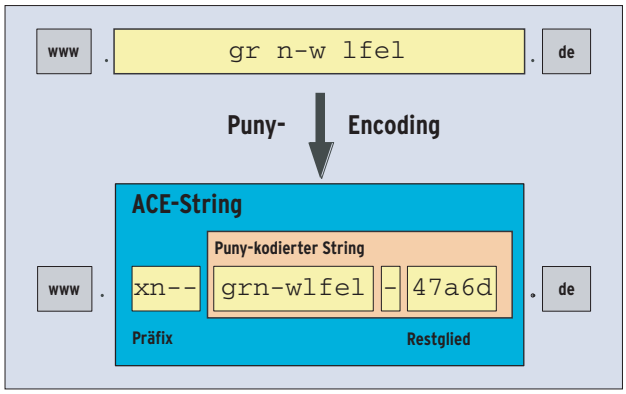

Abbildung 2: Die Konvertierung von Unicode nach ACE erzeugt Domainnamen in dem für Web- und Nameservern vertrauten Ascii.

Abbildung 3: Die 92 Neuen – diese Zeichen unterstützen deutsche Registrare seit März 2004 in Domainnamen.

Punycode im Detail

Jede erdenkliche Zeichenkombination in einen möglichst kurzen ACE-String verwandeln – das ist das Ziel des Punycode-Algorithmus. Er wandelt dafür aber nicht nur unerlaubte Zeichen in Zahlencodes, sondern errechnet eine Darstellungsform, die gleichzeitig Positionsangaben enthält, sodass es möglich ist, die einzelnen Codes direkt hintereinander zu notieren.

Vom Ursprung der Deltas

Delta heißt der Zahlenwert, der Art und Position eines Sonderzeichens repräsentiert. Um den Punycode-Anhang kurz zu halten, bezieht sich die in den Werten gespeicherte Positionsangabe stets auf den jeweiligen Vorgänger. Auch für die Unicode-Codepoints ist das Identifikationsmerkmal der Abstand zur Codenummer des Vorgängers. Am Beispiel des Labels »böse-patäntbrüder« soll die Bestimmung der Deltas demonstriert werden. Die Variablennamen »m«,»n« und »h« werden dabei wie im Pseudocode des RFC verwendet. Für die Berechnung des ersten Deltawerts gilt:

delta = (m-n)*(h+1)+Versatz

Dabei ist »m« die Codenummer des zu kodierenden Zeichens. Die Reihenfolge bestimmt die Unicode-Tabelle, begonnen wird deshalb mit dem ä (siehe Abbildung 5). Da die Unicode-Codepoints erst jenseits der ersten 128 Zeichen vom Ascii-Code abweichen, wird für den ersten Deltawert der Initialwert »n« auf 128 gesetzt. Für alle weiteren Durchläufe nimmt »n« jeweils die Codenummer des zuletzt bearbeiteten Zeichens an.
»h« ist die Länge des Strings ohne Sonderzeichen (hier »bse-patntbrder«, also 14 Zeichen). In der Folge wird dieser Wert für jedes neu eingefügte Sonderzeichen um eins erhöht. Der Versatz gibt die Position des Sonderzeichens innerhalb des Strings an, immer gezählt vom linken Rand. Der verbleibende Rest fließt in die Berechnung des nächsten Werts mit ein. Für das erste einzufügende Unicode-Zeichen, das »ä«, wird von dem String »bse-patäntbrder« ausgegangen. Damit ergibt sich für den Versatz der Wert 7:

delta = (228-128)*15+7 = 1507

Für alle weiteren Deltas erhöht sich »h« entsprechend. Zu ihrer Berechnung dient:

delta = Versatz+
(m-n-1)*(h+1)+
Rest+1

Stellt man sich nun eine Tabelle vor, die jeweils so breit wie der String inklusive des aktuell einzufügenden Zeichens ist, dann repräsentiert die vertikale Achse den Abstand der Codepoints untereinander. Ein Überschreiten der Zeilenbreite entspricht dem Wechsel zum nächsten Codepoint. Die Summe aller zurückgelegten Felder ist der Wert Delta. Mit dem Faktor (m-n-1) wird dadurch, ausgehend von der Unicode-Nummer des vorherigen Sonderzeichens, ein Sprung nach unten vollzogen (Abbildung 4). Der gedachte Cursor befindet sich damit genau eine Zeile (m-n-1) über der des zu kodierenden Unicode-Zeichens.

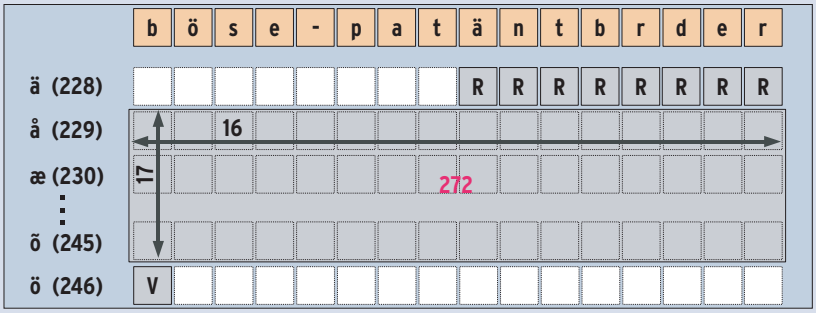

Abbildung 4: Vom ä zum ö: Die Entfernung setzt sich zusammen aus dem vertikalen Abstand (17 ganze Zeilen à 16 Felder = 272), dem (Zeilen-)Rest R und dem horizontalen Versatz V bis zur Einfügeposition. Als Delta ergibt sich 16*17+8+1=281.

Deltas fließen zusammen

Diese Dezimalzahlen könnten nun jede einzeln für sich zur Basis 36 konvertiert werden, wodurch drei kurze alphanumerische Strings entstünden.
Fügte man diese direkt aneinander, wäre jedoch die ursprüngliche Information aufgrund des fehlenden Trenners unwiederbringlich verloren.
Um dieses Problem zu lösen, wird eine Zahlendarstellung mit gemischten Basen verwendet. Im Gegensatz zu herkömmlichen Stellenwertsystemen, bei denen sich die Wertigkeit jeder Stelle aus ihrer Position ergibt (Einer, Zehner, Hunderter ...), wird die Wertigkeit hier über einen Schwellwert bestimmt.
Das Prinzip der gemischten Basen lässt sich am besten am Dekodierungsvorgang veranschaulichen, der Einfachheit halber mit gegebener Schwellwertfolge 235555 und der Ziffernfolge 734251. Beides sind oktale Angaben, die bei der (echten) Punycode-Dekodierung durch alphanumerische Strings zu ersetzen sind, indem die Ziffern zur Basis 36 interpretiert werden.

Code (m)	②	①	③				
	böse-patäntbrüder						
ä 228	V	VVVVVV	RRRR	RRR			
ö 246	V	RRRRRRRRRR		RRR			
ü 252	VVVVVVVVVVVV			RRR			

Abbildung 5: Die Konvertierungsschritte in einer tabellarischen Darstellung.

Zurück zu den Deltas

Das in der Tabelle gezeigte Schema veranschaulicht die Schritte zur Ermittlung der Deltas. Die ersten drei Zeilen lassen sich bereits ausfüllen. Zuerst werden die Grenzen gezogen: Das Ende einer Zahl ist dann erreicht, wenn eine Oktalziffer kleiner ist als der zugehörige Schwellwert: digit(j) < t(j). Die Schwellwertfolge beginnt dann an dieser Stelle neu. Tatsächlich handelt es sich also um die Oktalziffernfolgen (mit gemischter Basis) 734 und 251. Wie für gewöhnliche Stellenwertsysteme ist auch hier definiert, dass die Gewichtung der am wenigsten signifikanten Stelle 1 beträgt: w(0) := 1
Dieses Least Significant Digit (LSD) steht ganz links. Mit w(j) = w(j-1) * (Basis-t(j-1)) erfolgt die Berechnung der fehlenden Werte. Es gilt:

für w(1): 1* (8-2)
für w(2): 6*(8-3)

Da auch die Folge der Gewichtungen wie die der Schwellwerte mit jeder neuen Zahl neu beginnt, sind die weiteren Gewichtungen in diesem Beispiel unwichtig. Die Dezimal-Umwandlung ist nun vergleichbar mit dem Dekodieren einer Zahl eines beliebigen Zahlensystems:

Delta1: 1*7+6*3+30*4 = 145
Delta2: 1*2+6*5+1*30 = 62

Die Kodierung erfolgt umgekehrt. Der Schwellwert ist so zu bestimmen, dass er die jeweils noch unbekannte Gewichtung übersteigt und damit das Ende signalisiert. Diese Gleichung mit zwei Unbekannten löst der Punycode-Algorithmus, indem er von der letzten Stelle der Kodierung ausgeht und dafür das maximale »t« von 26 (dezimal) vorgibt. Zusätzlich vermindert er t(j) von Ziffer zu Ziffer um einen Bias. Das bewirkt im Durchschnitt kürzere Ziffernfolgen je kodiertem Delta. Für eigene Experimente eignet sich die Beispiel-Implementierung [10].

Schließlich eignet sich IDN auch dazu, Unicode-Strings in Argumente für Kommandozeilen-Utilities zu konvertieren, etwa so:

```
ping `$(CHARSET=ISO-8859-1); echo ➥
"www.käse.de"` | idn --quiet -a
```

Jenseits von Latin-1

Für jene unter den 92 neuen Zeichen, die nicht im Zeichensatz ISO 8859-1 enthalten sind, ergibt sich ein weiteres Problem: Wie kann man sie eingeben und wie werden sie auf dem Bildschirm dargestellt? Eine Lösung bietet ein Unicodefähiges Terminal unter X-Window (etwa Uxterm) zusammen mit einem Unicode-Zeichensatz. Noch besser geeignet ist das Webfrontend des Libidn-Autors Simon Josefsson unter **[2]**. Es stellt alle Optionen der jeweils aktuellen IDN-Utilities zur Verfügung. Zudem bietet das Programm die Möglichkeit, die HTML-Seitenkodierung der Ausgabe frei zu bestimmen. Bei 180 verschiedenen Kodierungen dürften nur sehr ausgefallene Wünsche offen bleiben.

Bei den meisten der neuen Zeichen handelt es sich um Zusammensetzungen aus lateinischen Buchstaben und diakritischen Zeichen, also den Pünktchen, Häkchen und Kringeln, die für die Betonung oder Aussprache des Basiszeichens stehen. Wer Deadkeys auf seiner Tastatur duldet, kann diese Kombinationen leicht eingeben. So zaubert etwa die Taste für den Accent aigu, gefolgt von einem [E] ein é auf den Bildschirm. Vielfältigere Möglichkeiten bietet die Compose-Taste, unter X-Window auch Multikey genannt. Häufig entspricht sie der Tastenkombination [Shift] + [AltGr]. Die Tastenfolge [N],[~] legt dann die Tilde über das n, statt es als eigenständiges Zeichen daneben abzubilden.

Ein Blick in »/usr/X11R6/lib/X11/locale/iso8859-1/Compose« zeigt, was auf diese Weise – im Rahmen von ISO-8859-1 – bereits möglich ist. Mit aktivierter UTF-Unterstützung und einer anderen Compose-Tabelle lässt sich die Beschränkung auf 255 Zeichen überwinden. Der Eingabe von Ogoneks (Nasalhaken) oder quer gestrichenen Buchstaben und anderen Exoten steht dann nichts mehr im Wege.

Wer das alles zu aufwändig findet, der bedient sich einfach mit Cut&Paste aus der Gnome Unicode Charmap **[7]**. Aber auch das hat einen Haken: Zu ihrer Darstellung muss ein entsprechender Font installiert sein, was in einem Terminal oder erst recht auf der Konsole nicht immer gewährleistet ist.

Schriftenvielfalt

Die Welt der Schriftzeichen ist vielfältig und die Kommunikationsprobleme mit anderen Sprach- und Schriftkulturen bestehen nicht erst seit Internet-Zeiten. Schon lange schreibt man Nicht-Ascii-Schriften in lateinische Buchstaben um – entweder ihrer Aussprache nach (Transliteration) oder ihrer Zeichen-Entsprechung nach (Transskription).

Da für Domainnamen nur ein begrenzter Zeichenvorrat zur Verfügung steht, ist diese Notlösung etwa für einen Chinesen, dessen Sprache immerhin 20000 Ideogramme kennt, kaum verwendbar. Hinzu kommt, dass Transliterationen von der Rechtschreibung der Zielsprache abhängen: Der russische Präsident erscheint in deutschen Tageszeitungen als Wladimir Putin, in der französischen Presse hingegen als Vladimir Poutine.

Zudem gibt es für die Transskription keine 1:1-Umsetzung fremder Zeichen in die des Ascii-Alphabets – weil es sich nicht um Buchstaben, sondern um Silben oder gar um komplex zusammengesetzte (logographische) Zeichen handelt, die völlig unabhängig von der Aussprache und einem Alphabet als Bedeutungsträger fungieren.

Es gibt durchaus Regeln und Konventionen, um die Umschreibung fremder Zeichen zu vereinheitlichen und die Willkür durch die jeweiligen Sprachgepflogenheiten einzugrenzen, aber sie vermögen die grundsätzliche Schwäche dieses Systems nicht zu beseitigen.

Whois und IDN

Was nützen internationale Domainnamen ohne einfache Prüfung, ob sie auch registriert werden können? Die neueste

RIPE-Version von Whois [8] erledigt dies (und vieles mehr):

```
whois -h whois.denic.de -C ISO-8859-1 ➥
-T dn bowlingbahnöl.de
```

Bei allen Ausgaben werden die Unicode- und die ACE-Form zur Kontrolle ausgegeben:

```
domain:      bowlingbahnöl.de
domain-ace:  xn--bowlingbahnl-fjb.de
```

Wer die Konsole oder das Terminal in den UTF-8-Modus setzt, kann »-C UTF-8« verwenden, was das Zeichensatz-, nicht immer aber das Fontproblem löst.

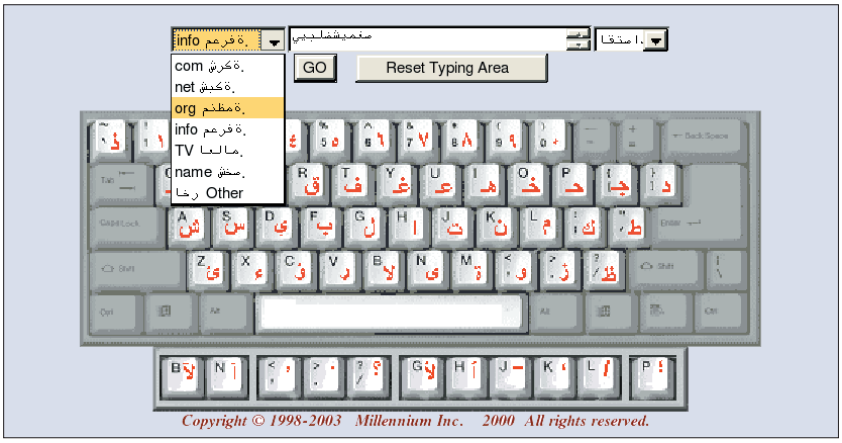

Abbildung 6: Nicht alle Schriften lassen sich so einfach eintippen. Für das Eingeben der arabischen Zeichen beispielsweise hat sich ein Registrar etwas Besonderes einfallen lassen: Ein Java-Applet, bei dem der Bediener die einzelnen Zeichen per Maus auswählt.

Libidn – eine für alle

Grundlage des IDN-Tools und sehr vieler IDN-fähiger Applikationen ist die hervorragend dokumentierte GNU-Bibliothek Libidn von Simon Josefsson [1]. Die Funktionen dieser Bibliothek entsprechen dabei weitestgehend den Optionen des IDN-Tools.

Die aktuelle Version 0.42 vom 20. März 2004 berücksichtigt die Beschränkungen, die der jeweiligen Toplevel-Domain auferlegt sind (92 Sonderzeichen für DE-Domains), lediglich für Norwegen und Frankreich. Das IDN-Tool antwortet auf einen Test mit einem für NO unzulässigen Zeichen tatsächlich mit »string rejected by TLD test«. Ein Omega in einem Domainnamen für die Toplevel-Domain DE beanstandet IDN nicht. Um dieses Manko abzustellen, hat Thomas Jacob das Projekt „TLD checking patch“ auf Berlios ins Leben gerufen. Das aktuelle Patch [3] ändert die Libidn-Sourcen so ab, dass immerhin die Regelungen für AT-, CH-, DE-, DK-, INFO-, LI-, PL- und SE-Domains berücksichtigt werden.

Perl, Python, PHP und Pike

Ebenfalls von Thomas Jacob stammt das Perl-Modul Net::LibIDN (Paketname »libnet-libidn-perl«) [5], das auch Perl-Programmierern den Genuss der verschiedenen Libidn-Funktionen gönnt.

Unicode – alles gut?

Schon der Name „American Standard Code for Information Interchange“ deutet an, dass Ascii als Partner für den Informationsaustausch keinesfalls die gesamte Welt in Betracht zieht. Die Erfolgsgeschichte des Internets hat diesem Standard allerdings zu einer Verbreitung auch in solche Regionen verholfen, in denen er seinem Zweck nur sehr bedingt gerecht werden kann. Daher entwickelten sich schon bald neben dem für die westliche Welt gängigen Standard ISO 8859-1, der auf 8-Bit-Ascii beruht, andere Zeichensätze und Kodierungen.

Hieroglyphen inklusive

Mit Unicode wurde Ende der 80er Jahre ein erweiterbarer, lebendiger Standard geschaffen, der tatsächlich alle bekannten Zeichen und Symbole aller Sprachen dieser Erde repräsentiert. Jedes einzelne ist in einer Tabelle festgehalten, die ihm eine eindeutige Positionsnummer (Codepoint) zuordnet.

Zeichen steht hierbei nicht nur für Buchstaben, auch Ziffern, Silben-, Satz- und sogar Steuerzeichen, die etwa die Änderung der Schreibrichtung (von links-rechts nach rechts-links oder umgekehrt) mitten im Text markieren, haben in dieser Tabelle ihren Platz gefunden. Und sie ist offen für weitere, bisher noch unbekannte Zeichen.

Die ersten 256 Zeichen des Unicodes stimmen mit dem bei uns verbreiteten Zeichensatz ISO Latin-1 (ISO 8859-1) überein. Bis zur Unicode-Version 3.2 wurde zunächst nur ein weiteres Byte zur Kodierung hinzugezogen. Damit war Platz geschaffen für etwa 65 000 zusätzliche Zeichen. Mit diesem Zwei-Byte-Wert ist auch heute noch die Kodierung aller wichtigen Zeichen – auch der 20 000 chinesischen – möglich (Basic Multilingual Plane, BMP).

Die Aufnahme altägyptischer Hieroglyphen und weiterer ausgefallener Zeichen machte weitere Bits notwendig. Mit Version 4.0 entschied sich das Unicode-Konsortium für Vier-Byte-Werte, von denen derzeit nur die ersten 21 Bit für effektiv rund 95 000 real existierende Zeichen verwendet werden.

Die Speicherung eines Unicode-Codepoint benötigt mindestens 2 Byte pro Zeichen, wodurch sich die Dateigröße im Vergleich zu der Ascii-Version verdoppelt. Das Format UTF-8 behebt dieses Manko, indem es die 128 Ascii-Codes mit nur einem Byte dargestellt. Je nach Position des Zeichens in der Unicode-Tabelle benötigen manche mit UTF-8 kodierten Zeichen aber auch 2 bis 3 Bytes. Das bedeutet, dass UTF-8 innerhalb der westlichen Unicode-Zeichenfamilie durchaus effizienter ist – bei der Verarbeitung von Zeichen, die in der Unicode-Tabelle weit von den Ascii-Zeichen entfernt platziert sind, können dagegen einzelne Zeichen mehr Speicher beanspruchen als mit konsequenter 16-Bit BMP-Kodierung.

Die Darstellung von Unicode-Zeichen in Ascii-Texten wie zum Beispiel den RFCs variiert, das Zeichen Pi mit der hexadezimalen Positionsnummer »03c0« kann als »U+03c0« oder »<03c0>« geschrieben werden. Um die zirka 95 000 Zeichen auch sehen zu können, muss der jeweils passende Font installiert sein.

Grenzenlose Kommunikation

Webseiten, die den entprechenden Tag im Header vorweisen, teilen dem Browser gleich mit, welcher Zeichensatz und welche Schriftart zur Anzeige erforderlich sind. Ist der geforderte Font auf dem Server hinterlegt, stellt die grenzenlose Kommunikation zumindest kein technisches Problem mehr dar. Die Verwendung von Unicode-Zeichen in HTML und XML ist problemlos mit der Notation »&#xUNICODE;« (zum Beispiel »π«) möglich.

Die Fehler, die »make test« meldet, kann man getrost ignorieren – verhindern lassen sie sich nur mit dem erwähnten Libidn-Patch. Eine Beispiel-Anwendung des Moduls könnte so aussehen:

```
use Net::LibIDN ':all';
print idn_to_ascii("www.grün-wölfel.de");
```

Python erfreut seine Anhänger mit dem wohl umfassendsten Support für IDN. Mit Version 2.3 erlaubt der IDNA-Codec Konvertierungen zwischen ACE- und Ascii-Strings. Das Besondere an Python ist jedoch, dass viele der bestehenden Python-Skripte keine Anpassung benötigen und damit automatisch IDN-tauglich sind. Dafür sorgt die transparente Wandlung von URLs und Hostnamen in Funktionsparametern.

Elf Funktionen für PHP

Für PHP existiert ebenfalls ein Libidn-API [4], mit dem die IDN-Funktionalität – in Form elf einzelner Funktionen – zur Verfügung steht. Voraussetzung ist natürlich, dass die IDN-Unterstützung in PHP bei Configure auch einkompiliert wurde (»--with-idn[= *Verzeichnis*]«).
Auch die noch nicht sehr verbreitete, aber durchaus leistungsfähige objektorientierte Skriptsprache Pike verfügt

Gastkommentar: Herausforderung IDN

Der Einsatz von Umlauten in einem URI hat zweifellos etwas Abenteuerliches an sich. Schon bei Elementen wie »Übersicht.html« warten etliche Fallen. Jeder Browser interpretiert sie anders. Je nach eingestelltem Zeichensatz (etwa ISO 8859-15 oder UTF-8) sendet mein Browser einen anderen Namensstring an den Server. So gelingt es dem Webmaster nicht, sicherzustellen, dass seine Seiten immer funktionieren. Bei den Domänen setzt die IETF zwar recht konsequent auf Unicode, sie reduziert das Problem damit aber nur zum Teil.
Der Ansatz, alle Zeichensatzprobleme in der Applikation lösen zu wollen, wird uns Anwender, Programmierer und Admins mit zahlreichen neuen Problemen konfrontieren. Jede Applikation muss die DNS-Namen an verschiedenen Stellen verarbeitet, zum Beispiel beim Darstellen einer Fehlerseite. Wer eine Umlaut-Domain eintippt, in einer Rückmeldung aber den Punycode-String sieht, hat es schwer bei der Fehlersuche.

Tippfehler sind nicht mehr zu erkennen

Bisher konnte ich als Anwender noch halbwegs erkennen, ob das Problem auf meiner Seite liegt oder ob der Server nicht korrekt arbeitet. In Zukunft wird die Mischung aus Umlautdarstellung und Punycode-String zusätzlich Verwirrung stiften. Tippfehler sind kaum noch zu erkennen. Ist die Domäne »xn--lcke--Ora.*Toplevel*« falsch geschrieben oder kann ich den Server nur nicht finden, weil eine Applikation noch nicht mitspielt?
Dazu kommen neue Probleme beim Vergleich der Servernamen in SSL-Zertifikaten. Durch bloßes Hinschauen kann ich nicht mehr sicher erkennen, ob die Umlaut-Domänen im Zertifikat wirklich der Website gehören, auf der ich etwas bestellen will, oder ob ich auf einer falschen Seite gelandet bin. Auch die Empfangsbestätigung per Mail ist ein Negativ-Beispiel, hier verlieren wir ein gutes Stück Zuverlässigkeit. Bis alle diese Wege fehlerarm funktionieren, wird noch einige Zeit vergehen. Ich denke sogar, dass wir noch lange darauf warten müssen, bis alle wichtigen Applikationen IDN brauchbar unterstützen.
Bis dahin werden sich die Anwender mit dem Mut zur »xn--lcke-Ora« wappnen müssen oder weiterhin reine Ascii-Namen verwenden. Fast jede Internet-fähige Anwendung benutzt heute die Namen und muss gegebenenfalls auf die recht komplizierte Kodierung erweitert werden. Hier sind noch Berge an Quellcode zu versetzen, dabei wird auch die Sicherheit unserer Anwendungen leiden. Als ein – möglichst abschreckendes – Vorbild mögen die vielen Exploits dienen, die Zugriffsfilter dadurch umgehen, dass sie Namen mit einer der vielen passenden Unicode-Kodierungen angeben.
Das eigentliche Problem ist trotz des beträchtlichen Aufwands aber noch immer nicht gelöst: Auf eine E-Mail-Adresse wie »Jürgen@Müller.de« werden wir noch einige Jahre warten müssen. *(Dirk Meyer/fjl)*

Dirk Meyer ist Diplom-Informatiker und Experte für Netzwerktechnik, Sicherheit, Linux und FreeBSD sowie Internet Service Provider seit 1992.

über guten Unicode- und IDNA-Support. Pike ab Version 7.6 unterstützt Methoden wie zum Beispiel »to_unicode()« und »nameprep()«, wodurch auch eine schrittweise Konvertierung möglich ist. C-Programmierer können sich der Libidn direkt bedienen, ausführliche Dokumentationen mit Beispielprogrammen finden sie auf den Libidn-Seiten. Dort sind auch Alternativen zur GNU-Bibliothek zu finden. Erwähnenswert ist Verisigns Software-Development-Kit für C und Java auf Linux- und Win32-Plattformen.

Ein Schritt vor und zwei zurück?

Die Aussicht auf mehr Freizügigkeit in der Gestaltung der Domainnamen hat jedoch nicht nur Zustimmung, sondern auch Kopfschütteln und laute Bedenken ausgelöst. So wird etwa die Auffassung vertreten, dass es besser wäre, sich der historisch entstandenen Beschränkung auf den Ascii-Code anzupassen. Doch das Web sollte auch für jene unkompliziert nutzbar sein, die sonst zu einem ständigen Wechsel von Zeichensatz und vielleicht sogar Schreibrichtung gezwungen wären. Hier sorgen die IDNs dafür, dass die Domain von Herrn Li für den gesamten asiatischen Raum, in dem chinesische Ideogramme weit verbreitet sind, mühelos zu erreichen ist. Aber letztlich auch wieder nur dort.

Alle Domainnamen mit Zeichen, die nicht auf jeder Tastatur leicht zu finden sind, darunter die Umlaute, werden zumindest für die Direkteingabe von einem anderen Sprachraum aus schwieriger zu erreichen sein. Ein einfacher Zugang kann nur über einen Link realisiert werden. Die noch fehlende Unterstützung der IDN durch die Suchmaschinen schränkt die Erreichbarkeit von solchen Domains weiter ein.

Auf Visitenkarten oder Anzeigen kann man die alternative Webadresse in Form des ACE-Strings drucken. Diese Lösung bietet einem chinesischen Anwender jedoch nicht mehr Komfort als die reine IP-Adresse. IDNs können also der entscheidende Schritt in Richtung auf mehr Informationsfreiheit sein, sie können allerdings auch zu einer neuen Regionalisierung und Auftrennung des Netzes in die unterschiedlichen Unicode-Untergruppen führen.

Fazit

Dank des IDNA-Standards berührt die Einführung der International Domain Names (IDN) weder das DNS noch HTTP und damit die Webserver. Die Umschreibung der nun erlaubten Unicode-Zeichen führt mittels Nameprep und anschließendem Punycode-Algorithmus zu einem ACE-String, der auf gewohnte Weise für die Namensauflösung herangezogen wird.

Die Unterstützung der Libidn in den wichtigsten Programmiersprachen ist zwar umfassend, doch noch sehr jung. So verwundert es nicht, dass derzeit nur sehr neue Browser IDNs unterstützen. Eine Änderung dieser Situation in den nächsten Monaten ist sehr wahrscheinlich. Solange die Anwendungen noch nicht von sich aus die Umwandlung leisten, steht mit dem Kommandozeilen-Tool IDN ein gutes Werkzeug zur Verfügung, das Konvertierungen in beide Richtungen durchführt. *(jcb)*

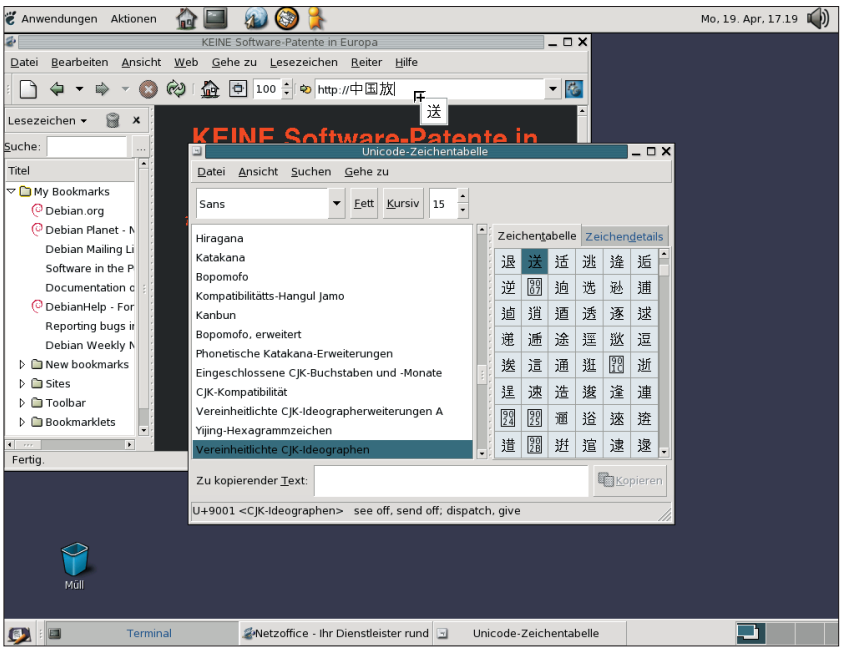

Abbildung 7: Mit der Gnome Unicode Charmap lassen sich beliebige Unicode-Zeichen per Drag & Drop oder Zwischenablage in andere Applikationen übertragen. Das ist immer noch bequemer, als den ACE-Code etwa einer japanischen Webseite per Hand einzugeben.

Infos

[1] Homepage der GNU IDN Library (Libidn): [http://www.gnu.org/software/libidn/]

[2] Webfrontend zum Testen des IDN-Kommandozeilen-Tools: [http://josefsson.org/idn.php]

[3] Libidn-Patch, damit Unicode-Beschränkungen der jeweiligen Toplevel-Domains berücksichtigt werden: [http://tldchk.berlios.de/]

[4] Dokumentation der PHP-IDN-Funktionen: [http://apache.bayour.com/phpdoc/html/ref.idn.html]

[5] Perl-Modul zur Nutzung der Libidn: [http://spaceship.berlios.de/index.php?page=idn.php]

[6] RFC 3492 über den Punycode: [http://www.faqs.org/rfcs/rfc3492.html]

[7] Unerlässlich bei Arbeiten mit Unicode-Zeichen ist die Gnome Unicode Charmap (Gucharmap): [http://gucharmap.sourceforge.net/]

[8] IDN-fähiger Whois-Client: [ftp://ftp.ripe.net/tools/ripe-whois-latest.tar.gz]

[9] Übersichtliche Darstellung der 92 neuen Sonderzeichen für DE-Domains mit der Möglichkeit, HTML-Entitäten zu kopieren: [http://www.grün-wölfel.de/idn_de_tabelle/]

[10] Punycode-Beispiel-Implementierung in PHP: [http://www.netzoffice.de/no/codeschnipsel.php]

[11] Afilias-FAQ zu IDN: [http://www.afilias.info/deutsch/idn_kunden_faq]

[12] Überblick über IDN-fähige Software unter Linux: [http://idn.isc.org/wiki/wiki.cgi?Other_IDN_Software]