knipp

Universal Acceptance Training

Werkzeuge und Unterstützung in der Software-Entwicklung

Michael Bauland

2023-04-21

knipp

Agenda

- Auffrischung
- E-Mail-Server
- Grundlagen: Zeichensätze und Character Encoding
- UA in der Software-Entwicklung (Python/Java)
  - Ein- und Ausgabe
  - Dateien lesen und schreiben
  - Normalisierung
  - IDN
  - EAI
- UASG Document Hub

knipp

Disclaimer

- Dieser Vortrag basiert zum Teil auf einem von der UASG veröffentlichten Vortrag im Zusammenhang mit dem UA Day
  - Abrufbar unter: https://uasg.tech/wp-content/uploads/2023/03/UA-Programming-Training.pptx
- Beispiele und Texte teilweise angepasst

knipp

Auffrischung – Universal Acceptance

Ziel

- Alle Domainnamen und E-Mail-Adressen funktionieren in allen Software-Produkten.

Auswirkung

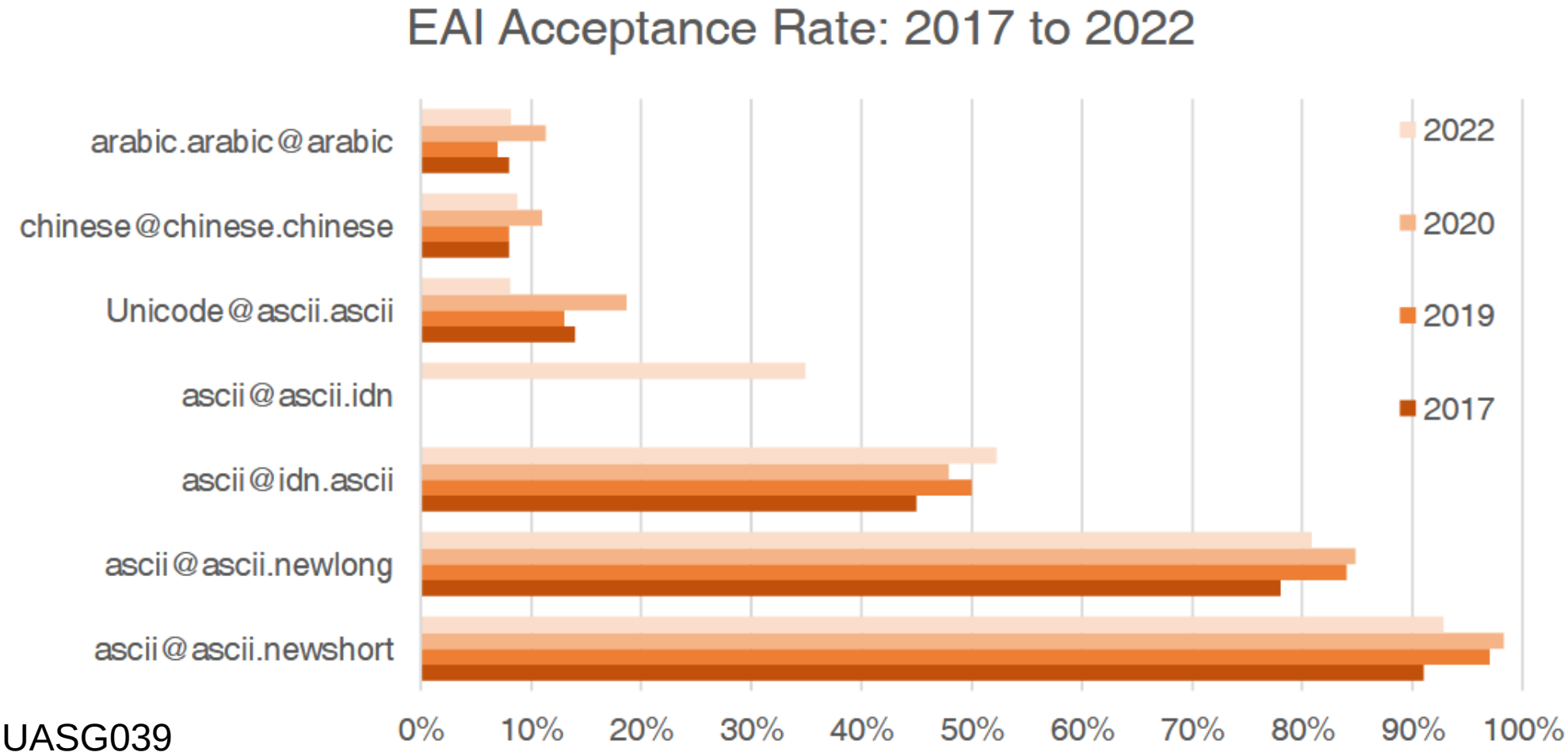
- Auswahl für Benutzer fördern
- Wettbewerb verbessern
- Breiteren Zugang für Endbenutzer anbieten

knipp

Kategorien von Domainnamen und E-Mail-Adressen

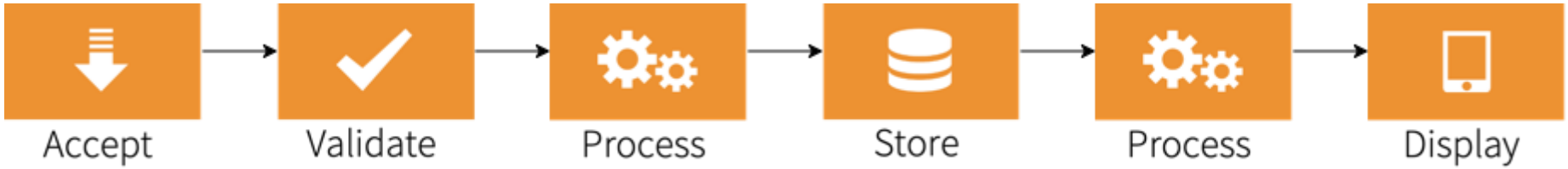
- Es ist inzwischen möglich Domainnamen und E-Mail-Adressen in verschiedenen Sprachen zu haben.
- Domainnamen
  - Neue Top-Level-Domainnamen — example.eus
  - Längere Top-Level-Domainnamen — example.bauhaus
  - Internationalisierte Domainnamen — أكان.بازار
- Internationalisierte E-Mail-Adressen (EAI=Email Address Internationalization)
  - ASCII@IDN (keine EAI) — michael@grün.de
  - UTF8@ASCII (EAI) — björn@example.com
  - UTF8@IDN (EAI) — björn@grün.de
  - UTF8@IDN; rechts-nach-links Skript (EAI) — بازار@أكان.بازار

Akzeptanz von E-Mail-Adressen weltweit



Quelle: UASG039

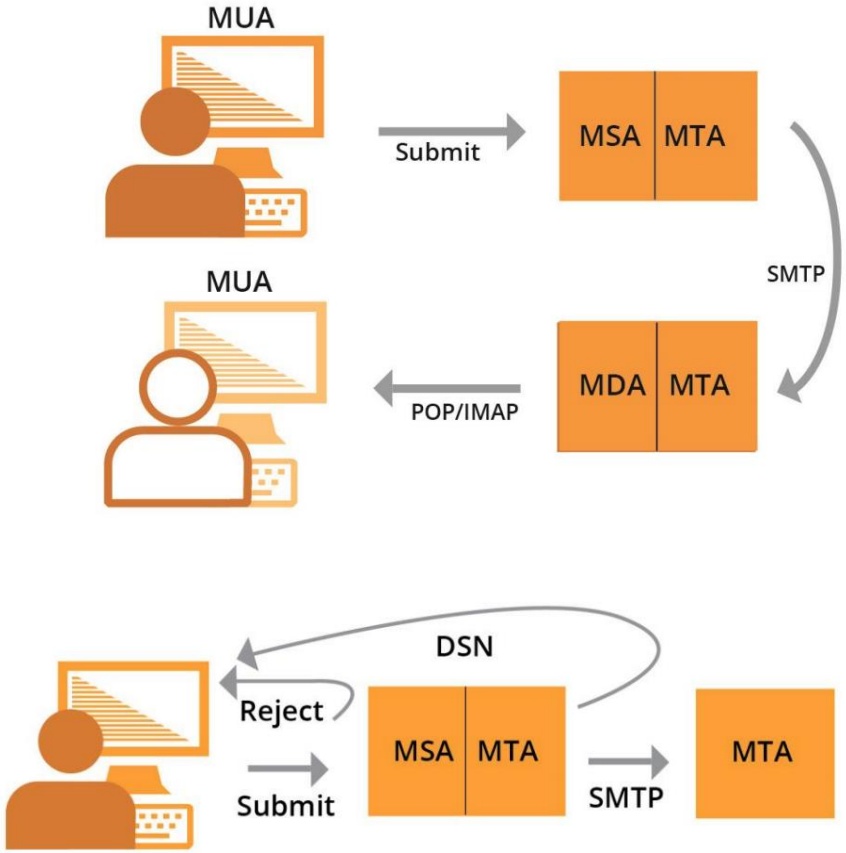
Universal Acceptance Readiness Framework



- **Accept**: Benutzer können Zeichen ihres lokalen Skripts eingeben.
- **Validate**: Die Software akzeptiert die Zeichen und erkennt sie als gültig.
- **Process** (1): Das System führt die Operation mit den Zeichen aus.
- **Store**: Das Speichersystem speichert die Zeichen korrekt.
- **Process** (2): Das System lädt die Daten und führt weitere Operationen aus.
- **Display**: Die Information wird anschließend korrekt angezeigt.

E-Mail-Systeme und EAI-Unterstützung

- Alle E-Mail-Agents müssen so konfiguriert sein, dass sie EAI schicken und empfangen.
  - MUA – Mail User Agent: Ein Client, der benutzt wird, um E-Mails zu schicken, zu empfangen und zu verwalten.
  - MSA – Mails Submission Agent: Ein Server, der E-Mails von MUAs erhält und weiter verarbeitet.
  - MTA – Mail Transfer Agent: Ein Server, der E-Mails von und an Internet-Hosts schickt. Ein MTA kann E-Mails von MSAs erhalten und/oder an MDAs ausliefern.
  - MDA – Mail Delivery Agent: Ein Server, der eingehende E-Mails verarbeitet und üblicherweise in Mailboxes speichert.



knipp

E-Mail-Systeme und EAI-Unterstützung

- Technische Konfiguration und Set-Up von Client und Server wird hier nicht weiter Thema sein.
- UASG bietet einen umfangreichen technischen EAI-Überblick in UASG012.
- Gabriella Schittek vermittelt weiterführende Schulungen speziell zum diesem Thema an. Bei Interesse
  - E-Mail schicken oder
  - Direkt im Anschluss ansprechen

knipp

Zeichen

Ein String besteht aus einzelnen Zeichen (Characters).

```
System.out.println ('h' + 'e' + 'l' + 'l' + 'o');
```
532

```
System.out.println ("h" + "e" + "l" + "l" + "o");
```
hello

```
char[] chArray = {'h', 'e', 'l', 'l', 'o'};
System.out.println ("hello".equals (String.copyValueOf
(chArray)));
```
true

knipp

Code Point

- Ein Code Point ist ein Wert oder eine Position eines Zeichens in einem gegeben Zeichensatz.
- Beispiele von Zeichensätzen:
  - US-ASCII
  - Windows-1252
  - ISO-8859-1
  - Unicode UTF-8

knipp

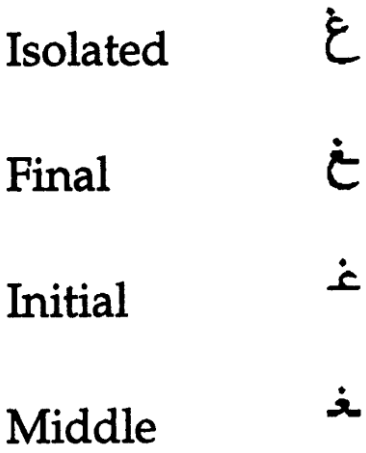
US-ASCII

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

knipp

Glyph

- Eine typographische Repräsentation eines Zeichens heißt Glyph.
  - Beispiel: a, *a*
  - Arabische Zeichen haben oft vier Glyphen, je nach Position im String, zum Beispiel das arabische Zeichen GHAIN:
  - Sprachen können von links nach rechts oder von rechts nach links geschrieben/angezeigt werden, aber die Daten werden immer in der Reihenfolge der Zeichen in einer Datei gelesen. Dies ist unabhängig von der Schreibrichtung.



knipp

Character Encoding

- Ein Character Encoding ist ein Mapping einer Zeichensatzdefinition zu den tatsächlichen Code Points, die benutzt werden, um die Daten zu repräsentieren.
- Ein Encoding beschreibt, wie Code Points zu Bytes kodiert werden und wie Bytes zu Code Points dekodiert werden.

knipp

Kurze Geschichte

- Basic ASCII: 7-bit Character: beschränkt auf max. 128 Zeichen
- Extended ASCII: 8-bit Character: beschränkt auf max. 256 Zeichen
- ASCII Encoding kann offensichtlich nicht genug Zeichen beinhalten, um alle Sprachen zu unterstützen
- Verschiedene Encoding-Systeme wurden entwickelt, die je nach Sprache und Skript Nummern ganz unterschiedlich Zeichen zuweisen. Führte (und führt auch immer noch) zu vielen Interoperabilitätsproblemen.

knipp

Unicode

- Ein Code für alle.
- Standard für digitale Repräsentation von Zeichen
- Unicode bietet eine einheitliche Möglichkeit, Text in jeder Sprache zu speichern, zu suchen und auszutauschen.
- Er wird von allen modernen Computern verwendet und ist die Grundlage für Textverarbeitung im Internet.
- Die Slots, um die Weltsprachen zu repräsentieren liegen im Bereich 0000 – 10FFFF. Auf https://unicode.org/charts/ sieht man alle Bereiche.

knipp

Unicode Encoding

- Unterschiedliche Character Encodings implementieren Unicode
  - UTF-8
  - UTF-16
  - UFT-32
- UTF-8 wird üblicherweise im Domain-Bereich verwendet.
- UTF-8 hat eine variable Länge für die Zeichenkodierung (1-4 bytes)
  - ASCII-Zeichen: 1 byte
  - Arabische Zeichen: 2 bytes
  - Devanagari Zeichen: 3 bytes
  - Chinesische Zeichen: 4 bytes

knipp

Beispiele in Python und Java

- Beispiele in diesem Vortrag stammen zum größten Teil von UASG
- Mitschreiben nicht nötig
- https://uasg.tech
  - Document Hub
  - Tags „Coding“
  - UASG 043 UA-Ready Code Samples in Java, Python, and JavaScript EN

knipp

Ein- und Ausgabe in Python

```
print ("Enter your input: ")

inputstr = input () #default character encoding is UTF-8

print ("Input data is: ")

print (inputstr)
```

knipp

Ein- und Ausgabe in Java

```
import java.util.Scanner;

class ReadWriteUnicode
{
  public static void main (String[] args)
  {
    Scanner scr = new Scanner (System.in);
    System.out.println ("Enter your input");
    // default character encoding is UTF-8
    String Input = scr.nextLine ();
    System.out.println ("Received input is: " + Input);
  }
}
```

knipp

Python – Datei lesen und schreiben

- UTF-8-Datei einlesen

```
file = open ("test1.iml", 'r', encoding='UTF-8')
for line in file:
    print (line)
file.close ()
```

- UTF-8 Datei schreiben

```
file2 = open ("filepath", 'w', encoding='UTF-8')
data_to_write="اُن کے وکلا کی کوشش ہو"
file2.writelines (data_to_write)
file2.close ()
```

knipp

Java – Datei lesen

```
try
{
  FileInputStream fis = new FileInputStream ("filename");

  InputStreamReader isr =
    new InputStreamReader (fis, StandardCharsets.UTF_8);

  BufferedReader br = new BufferedReader (isr);
  String line;

  while ((line = br.readLine ()) != null)
  {
    System.out.println (line);
  }

  fis.close ();
}
catch (IOException ex)
{
  System.err.println (ex);
}
```

knipp

Java – Datei schreiben

```
try
{
   FileOutputStream fis =
     new FileOutputStream ("filename");

   OutputStreamWriter osw = new OutputStreamWriter (
     fis, StandardCharsets.UTF_8);

   BufferedWriter bw = new BufferedWriter (osw);
   bw.write (";("اُن کے وکلا کی کوشش ہو
   bw.flush ();
   fis.close ();
}
catch (IOException ex)
{
   System.err.println (ex);
}
```

knipp

Java – Datei schreiben

```
try
{
   FileOutputStream fis =
     new FileOutputStream ("filename");

   OutputStreamWriter osw = new OutputStreamWriter (
     fis, StandardCharsets.UTF_8);

   BufferedWriter bw = new BufferedWriter (osw);
   bw.write (";("اُن کے وکلا کی کوشش ہو
   bw.flush ();
   fis.close ();
}
catch (IOException ex)
{
   System.err.println (ex);
}
```

knipp

Java – Datei schreiben

```java
try
{
  FileOutputStream fis =
    new FileOutputStream ("filename");

  OutputStreamWriter osw =
    new OutputStreamWriter (fis, StandardCharsets.UTF_8);

  BufferedWriter bw = new BufferedWriter (osw);
  bw.write ("اُن کے وکلا کی کوشش ہو");
  bw.flush ();
  fis.close ();
}
catch (IOException ex)
{
  System.err.println (ex);
}
```

knipp

Normalisierung

- Für manche Glyphen gibt es in Unicode mehrere Möglichkeiten:
  - è = U+00E8
  - e und ` = è = U+0065 und U+0300
  - آ = U+0622
  - ◌ٓ und ا = آ = U+0627 und U+0653
- Dies führt dazu, dass Benutzer einen bestimmten String unterschiedlich kodiert eingeben können:
  - آدم (U+0622 U+062F U+0645)
  - آدم (U+0627 U+0653 U+062F U+0645)
- Zum Suchen, Sortieren, etc. braucht man normalisierte Strings.

knipp

Normalisierung

- In Unicode werden die folgenden vier verschiedene Normalisierungsformen aufgelistet:
  - Normalization Form D (NFD): Canonical Decomposition
  - Normalization Form C (NFC): Canonical Decomposition, gefolgt von Canonical Composition
  - Normalization Form KD (NFKD): Compatibility Decomposition
  - Normalization Form KC (NFKC): Compatibility Decomposition, gefolgt von Canonical Composition
- In Domainnamen wird NFC verwendet.

knipp

NFC Normalisierung – Python und Java

- Details muss man nicht wissen: Vorhandene Libraries nutzen!
- Python
  - `import unicodedata`
  - `unicodedata.normalize ('NFC', input)`
- Java
  - `import com.ibm.icu.text.Normalizer2;`
  - `Normalizer2 norm = Normalizer2.getNFCInstance ();`
  - `norm.normalize (input);`

knipp

Domainnamen

- Ein Domainname ist eine geordnete Menge von Labels/Strings, zum Beispiel www.example.co.uk
  - Die Top-Level-Domain (TLD) ist uk
  - Die Second-Level-Domain (SLD) ist co
  - Die Third-Level-Domain ist example
  - Die Fourth-Level-Domain ist www
- Anfänglich waren TLDs immer 2 oder 3 Zeichen lang.
- Jetzt gibt es auch längere TLDs (z.B. .vermögensberatung)
- Jedes Label maximal 63 Zeichen, gesamt maximal 255 Zeichen.

knipp

Internationalized Domain Names (IDNs)

- Jeder Domainname, der nicht ausschließlich aus LDH (Letter, Digit, Hyhpen; 0-9, a-z, -) besteht ist ein IDN
  - z.B. www.grün.de
- Es gibt zwei äquivalente Formen: U-Label und A-Label
  - Punycode-Algorithmus wandelt Formen um
  - grün ←→ xn--grn-ioa
- Wichtig für Konvertierung: Immer IDNA2008-Version nutzen (nicht IDNA2003)!
  - strasse.de ≠ straße.de = xn--strae-oqa.de

knipp

Punycode-Konvertierung – Python und Java

- Python
  - `import idna`
  - `idna.encode`
  - `idna.decode`
- Java
  - `import com.ibm.icu.text.IDNA;`
  - `import com.ibm.icu.impl.UTS46;`
  - `IDNA idnaInstance = new UTS46 (…);`
  - `idnaInstance.nameToASCII (Ulabel, output, info);`
  - `idnaInstance.nameToUnicode (Alabel, output, info);`

knipp

Validierung – Domainnamen

- Normalisieren
- Zu A-Label konvertieren
- Wenn Konvertierung keinen Fehler ergibt: gültig

knipp

E-Mail-Adressen

- Syntax: mailboxName@domainName
- domainName kann A-Label, U-Label oder gemischt sein
  - michael@xn--grn-ioa.de und michael@grün.de sind gleiche Adressen
    - Wichtig falls E-Mail-Adresse gleich Login-ID → Normalisieren!
- mailboxName kann ASCII oder Unicode (UTF-8) sein
  - Unicode → EAI
  - michael@أكان.بازار ist keine EAI
- Kein Punycode bei mailboxName
  - xn--grn-ioa@knipp.de und grün@knipp.de sind verschiedene Adressen

knipp

Validierung – E-Mail-Adressen

- Einfach: something@something
  - Regex: `^(.+)@(.+)$`
- Komplex (z.B. von owasp.org)
  - Regex: `^[a-zA-Z0-9_+&*-]+(?:\.[a-zA-Z0-9_+&*-]+)*@(?:[a-zA-Z0-9-]+\.)+[a-zA-Z]{2,}$`
  - Unterstützt keine EAI und keine Domain in U-Label-Format
- Regular Expressions nicht sinnvoll zur Validierung
  - Entweder müssten sie IDNA nachimplementieren oder haben eine statische Liste von „erlaubten“ TLDs.

knipp

Validierung – E-Mail-Adressen – Python

```
from email_validator import
validate_email,EmailNotValidError

try:
    a = "بازار@أكان.بازار"
    # As part of process it performs DNS resolution
    # Normalizes email addresses automatically
    # Supports internationalized domain names
    v = validate_email (a, check_deliverability=True)
    print (v)
    print (a + " is a valid email address")
except EmailNotValidError as e:
    print (a + " is not a valid email address: ")
    print (e)
```

knipp

Validierung – E-Mail-Adressen – Java

- Etwas aufwendiger
  - Details auf UASG-Seite

- Unsere Lösung im späteren Vortrag

knipp

UASG

- https://uasg.tech
- ICANN Community: https://community.icann.org/display/TUA

knipp

Fazit

- UA wird nicht immer vollständig von Software und Libraries unterstützt
  - Richtige Library / Framework wählen
  - Code anpassen, damit UA korrekt unterstützt wird