Handbuch diakritische Zeichen Grundlagen - Recht - Technik	Konvention diakrit 1.2.0 Ergebnis der AG

Kurzbeschreibung:	Dieses Dokument erklärt die Grundlagen von Unicode, die rechtliche Rahmenbedingungen zum Einsatz von diakritischen Zeichen und beschreibt die technische Handhabung in der Praxis.		
Autor(en):	Larissa Naber Peter Danner **Peter Reichstädter**	Projektteam / Arbeitsgruppe:	AG-II BKA/Bereich IKT-Strategie
Beiträge von:	Markus Triska Nicolas Knotzer		BMF

Stelle:	Vorgelegt am:	Angenommen am:	Abgelehnt am:

Änderungen von Version 1.1.0 auf 1.2.0

3.1.3 (Empfehlung: Unicode): Statt expliziter Auflistung der relevanten Unicode-Ranges wird auf die neue Konvention "Diakritische Zeichen" (DZ) verwiesen.

3.1.9 (Relevante Unicode Ranges): Spalte 4 (behördliche Verwendung) entfernt, da sich diese aus der Transkriptionstabelle in der Konvention DZ ergibt.

3.1.10 (Empfehlung: Zeichen): Transkriptionstabelle und Tabelle der Sprachcodes entfernt, die Regelung erfolgt in der Konvention DZ.

3.1.11 (Empfehlung: Zeichencodierung): Abschnitt entfernt, da in Konvention DZ geregelt.

Contents

Dieses Dokument verwendet die Schlüsselwörter MUSS, DARF NICHT, ERFORDERLICH, SOLLTE, SOLLTE NICHT, EMPFOHLEN, DARF, und OPTIONAL zur Kategorisierung der Anforderungen. Diese Schlüsselwörter sind analog zu ihren englischsprachigen Entsprechungen MUST, MUST NOT, REQUIRED, SHOULD, SHOULD NOT, RECOMMENDED, MAY, und OPTIONAL zu handhaben, deren Interpretation in RFC 2119 festgelegt ist.

Chapter 1

Sonderzeichen und diakritische Zeichen

Einleitung

Das vorliegende Dokument soll Behörden und ihre Dienstleister beim Umsetzen von Applikationen mit diakritischen Zeichen unterstützen.

1.1 Überblick

1.1.1 Über dieses Dokument

Erstellung

Dieses Dokument wurde in XML mit UTF-8 Codierung erstellt und anschließend mittels XSLT in die HTML und PDF Version konvertiert. Das Handbuch steht zusätzlich auch als Online-Version, mit ausführbaren Beispielen, zur Verfügung.

Wünsche, Anregungen und Beschwerden

Bitte im dafür vorgesehenen Bugzilla eintragen. Ein Account wird auf Nachfrage eingerichtet. Für Kommarch-Mitglieder ist der Zugang auch im Kommarch-Mailinglisten-Archiv zu finden.

Download

- HTML Version als ZIP File (siehe separates Paket Mustercode/Beilagen)
- PDF Version (siehe separates Paket Mustercode/Beilagen)
- Mustercode als ZIP File (siehe separates Paket Mustercode/Beilagen)
- Quellen als ZIP File (siehe separates Paket Mustercode/Beilagen)

1.1.2 Wie man dieses Dokument lesen soll

Dieses Dokument gliedert sich in drei Abschnitte:

- Einleitung (siehe Kapitel 1)
- Problemstellung (siehe Kapitel 2)
- Handhabung in der Praxis (siehe Kapitel 3)

Die Einleitung (siehe Kapitel 1) behandelt Unicode ganz allgemein, während das Kapitel Problemstellung (siehe Kapitel 2) auf die rechtlichen und linguistischen Grundlagen eingeht. Die Kapitel Einleitung (siehe Kapitel 1) und Problemstellung (siehe Kapitel 2) haben informativen Charakter, während das Kapitel Handhabung in der Praxis (siehe Kapitel 3) eher normativen Charakter hat.

Die ersten beiden Kapitel des Handbuchs sollten sequentiell gelesen werden. Bei ausreichenden Vorkenntnissen können diese Kapitel aber übersprungen werden. Das dritte Kapitel ist in Abschnitte gegliedert, die jeweils einen Themenkomplex näher betrachten, z. B.: Eingabe, Ausgabe, Datenbanken, Register, ...

1.1.3 Empfehlung

Empfehlungen werden durch die rote Überschrift Empfehlung und in der HTML-Version dieses Handbuchs zusätzlich durch den roten Rahmen um die eigentliche Empfehlung gekennzeichnet. Empfehlungen sollten nur in begründeten Ausnahmefällen ignoriert werden.

1.2 Unicode

Einleitung

Um sich den Sonderzeichen und diakritischen Zeichen nähern zu können, sind einige Grundlagen zu Unicode notwendig, die in diesem Kapitel vermittelt werden.

Ressourcen

- W3C Internationalization
- On the Goodness of Unicode Einstieg in Unicode
- The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets (No Excuses!) Einstieg in Unicode

1.2.1 Unicode

Unicode

Unicode ist ein internationaler, standardisierter Zeichensatz (character set) der derzeit (Q4/2005) in Version 4.1 vorliegt. Unicode ist die Kurzform für *„Universal Character Encoding“* und ist als Superset aller bisherigen diesbezüglichen internationalen, nationalen und Industriestandards zu sehen. Mit diesem Zeichensatz können fast alle lebenden und sehr viele tote Sprachen geschrieben werden. Im Gegensatz zu früheren Ansätzen, welche die gleichzeitige Darstellung von nur wenigen Sprachen im selben Text erlauben (Beispiele dafür sind Latin-1, Latin-2, ...), können mit Unicode alle unterstützten Sprachen **gleichzeitig** geschrieben werden.

Der Unicode Standard encodiert dabei eigentlich keine Sprachen, sondern Skripts – gemeint ist damit, dass wenn mehrere Sprachen sich ein gemeinsames Set von Symbolen teilen (zum Beispiel ist dies bei westeuropäischen Sprachen der Fall), so wird die Vereinigung all dieser Symbole in eine gemeinsame Sammlung von Zeichen – eben ein Skript zusammengefasst.

Beispiele von Skripts in Unicode sind:

- Latein
- Griechisch
- Kyrillisch
- Armenisch

- Hebräisch
- Arabisch
- Devanagari
- Bengali
- . . .
- Tibetisch
- Japanisch Kana
- Koreanisch Hangul
- . . .

Der Unicode Standard ordnet jedem Zeichen eine Laufnummer (Codepoint) zu. Unicode Zeichen werden üblicherweise in der Form U+(nn)nnnn beschrieben, wobei (nn)nnnn der hexadezimale Code des Zeichens ist, zum Beispiel: U+0123 = ġ. Der Unicode Zeichenraum erstreckt sich über den Adressbereich von 0 bis 0xFFFF (bzw. 0x10FFFF) - somit fasst er voll belegt knapp über 1,1 Millionen Zeichen. Des Weiteren wird der Unicode Zeichenraum in 16 Ebenen (planes) unterteilt, die jeweils 64K (65536) Zeichen aufnehmen können. Zusätzlich verfügt jedes Zeichen über einen Namen (z. B. U+0061 LATIN SMALL LETTER A = a), und es werden auch weitere Informationen definiert (zugehöriger Groß-/Kleinbuchstabe, Richtung, alphabetische und semantische Eigenschaften, udglm.).

Hinweis: Aufgrund der eindeutigen Zuweisung eines Codepoints (beispielsweise für das lateinische „A“ – U+0041; für das tibetanische Om U+0F00, usw.), kommt es oft zu Fehleinschätzungen, Unicode sei einfach ein neuer 16-Bit Code (jedes Zeichen hat 16 Bits und daher gäbe es maximal 65535 Zeichen). Diese Interpretation ist nicht korrekt! Die Schreibweise U+xxxx ist historisch bedingt.

Hinweis: In Unicode werden nur „abstrakte“ Zeichen kodiert. Alle anderen Aktionen, die mit einem eingegebenen Unicode Zeichen im Rahmen der Verarbeitung geschehen (also z. B. das Rendering am Schirm: Größe, Form, Orientierung, etc.), werden nicht im Unicode Standard definiert. Dies wird oft falsch verstanden: die visuelle Repräsentation eines Zeichens - „Glyph“ (siehe Kapitel 1.2.1) - ist nicht Teil des Unicode Standards.

Die Industrie nimmt den Unicode Standard hinsichtlich einer Internationalisierung und Standardisierung ihrer Software bzw. Produkte rasch an und inkorporiert diesen als Basis in ihre Produkte (Betriebssysteme, Datenbanken, ...).

Organisation

Die derzeit definierten knapp 100.000 Unicode Zeichen (dies sind ungefähr 9% der maximal möglichen Anzahl) befinden sich in 4 von 16 definierten Unicode Ebenen:

- **Ebene 0 (00000000 – 0000FFFF) – BMP – Basic Multilingual Plane:** enthält den überwiegenden Teil der Zeichen, die hauptsächlich genutzt werden.
- **Ebene 1 (00010000 – 0001FFFF) – SMP – Supplemental Multilingual Plane:** enthält weitere Zeichen und Symbole (überwiegend ausgestorbene Sprachen, musikalische Symbole, künstliche Schriftsysteme, diverse Alphabete, Zeichen zur Verwendung als mathematische Symbole, u. a.). Beispiel: 10330-1034F – gotisch, 103A0-103DF – persische Keilschrift, usw.
- **Ebene 2 (00020000 – 0002FFFF) – SIP – Supplemental Ideographic Plane:** enthält Ideographische Zeichen für CJK (= Chinesisch, Japanisch, Koreanisch; überwiegend Zeichen, die aktuell nicht mehr in Gebrauch sind).
- **Ebene 14 (000E0000 – 000EFFFF) – SSP – Supplementary Special Purpose Plane:** enthält Kontrollzeichen, die in Ebene 0 keinen Platz gefunden haben.
- **Ebene 15 (000F0000 – 000FFFFF):** für den privaten Gebrauch reserviert.
- **Ebene 16 (00100000 – 0010FFFF):** für den privaten Gebrauch reserviert.

Innerhalb der Ebenen existieren weitere Unterteilungen in *Allocation Areas* und *Character Blocks*. Die *Allocation Areas* unterscheiden grob: Alphabete, Symbole, Ideographische Zeichen, usw.

Außerdem sind die Zeichen innerhalb einer *Allocation Area* in Böcken (*Character Blocks*) gruppiert. Es gibt Blöcke für diverse Schriftsysteme, wie lateinische Schrift (Basis, erweitert A, erweitert B, ...), griechische Schrift, kyrillische Schrift, chinesisch-japanisch-koreanisch vereinheitlichte Zeichen, ... und Blöcke für diverse mathematische und andere Sonderzeichen (z. B. Währungssymbole, Pfeile, ...).

ISO/IEC 10646

Die Entwicklung eines universalen Zeichensatzes wurde ursprünglich von zwei Standardisierungsgremien Unicode und die ISO begonnen. Der initiale ISO Entwurf wurde jedoch nicht angenommen, da er sehr stark von den existierenden Systemen abwich.

Unicode hingegen berücksichtigte von vornherein, dass bereits eine Vielzahl unterschiedlicher Kodierungen millionenfach verwendet wurde. Unicode-basierte Systeme sollten daher herkömmlich kodierte Daten mit geringem Aufwand handhaben können. Hierzu wurde für die unteren 256 Zeichen die weit verbreitete ISO 8859-1-Kodierung (Latin1) beibehalten ebenso wie die Kodierungsarten verschiedener nationaler Normen, z. B. ISCII für indische Schriften, die in der ursprünglichen Reihenfolge lediglich in höhere Codebereiche verschoben wurden.

Aus diesem Grunde setzte sich Unicode durch und seit 1992 kooperieren nun beide Standardisierungsgremien. Der Unicode Standard V3.0 entspricht zu 100% dem ISO Standard ISO/IEC 10646-1:2000 (kurz: UCS – Universal Multiple-Octet Character Set):

- Unicode 1.1 ISO/IEC 10646-1:1993
- Unicode 3.0 ISO/IEC 10646-1:2000
- Unicode 3.2 ISO/IEC 10646-2:2001
- Unicode 4.0 ISO/IEC 10646-3:2003

ISO/IEC 10646 wurde insbesondere auch durch die Einführung von zusätzlichen Codierungen (Encodings) (siehe Kapitel 1.2.2), wie UTF-8 und UTF-16, zu einem begleitenden Standard.

Ressourcen

- unicode.org
- Was ist Unicode? (unicode.org)
- ISO/IEC 10646
- Allen Woods Unicode Page
- IBM Unicode Browser

1.2.2 Glyphen

Glyphen sind grafische Repräsentationen von Zeichen. Zur einer Zeichendefinition gehört neben der Codierung (Zeichenzuordnung zur Zeichenposition) auch die grafische Darstellung (Glyphe). Ein und dieselbe Glyphe kann dabei durchaus für mehrere Zeichen verwendet werden.

Veranschaulicht werden kann das anhand eines Beispieles. Obwohl ein a immer ein a ist, schaut es doch in jeder Schriftart (Font) anders aus: a, a, a; auch die Verwendung unterschiedlicher Stile (a, *a*, **a**, ^{a}) ändert nichts daran, dass der Buchstabe ein a bleibt.

Der Unicode Standard behandelt nur Zeichen, keine Glyphen. Auch bei bestehender Unicodeunterstützung ist das Endergebnis immer von den vorhandenen Glyphen abhängig. Wenn der gewählte Font einen bestimmten Glyphen nicht unterstützt, erscheint das Zeichen nicht (es erscheint stattdessen ein leeres Rechteck oder der Hex-Code des Zeichens, etc.) Vielen Unicode-Zeichen ist gar keine Glyphe zugeordnet. Auch sie gelten als „characters“. Neben den Steuerzeichen wie Zeilenvorschub (U+000A), Tabulator (U+0009) usw. sind allein 19 Zeichen explizit als Leerzeichen definiert, sogar solche ohne Breite, die u. a. als Worttrenner gebraucht werden, für Sprachen wie Thai oder Tibetisch, die ohne Wortzwischenraum geschrieben werden.

Digraphen (und Trigraphen, Tetragraphen, ...): Digraphen sind Buchstabenpaare (bzw. drei, vier, ... Buchstaben), die in einer Sprache als so eng zusammengehörig betrachtet werden, dass sie zum Beispiel auch bei der alphabetischen Sortierung wie ein Buchstabe behandelt werden (zwei Grapheme, die sich oft einem Phonem zuordnen lassen). Beispiele: ll im Spanischen, dž, lj, nj im Kroatischen, usw.

Trennen von Diakritika: Trägt ein Buchstabe mehrere Diakritika darüber oder darunter, werden diese normalerweise vertikal gestapelt. Für Ausnahmefälle, in denen zwei Diakritika nebeneinander stehen müssen, sieht Unicode vor, dass ein *combining graphem joiner (CGJ)* (U+034F) dazwischengestellt wird.

Mehr Informationen finden sich im praktischen Teil unter Fonts (siehe Kapitel 3.6).

1.2.3 Codierungen (Encodings)

Wozu Codierungen?

Der Unicode Zeichenvorrat kann in verschiedenen Encoding-Schematas/Formaten gespeichert werden. Unicode ordnet jedem Zeichen eine Laufnummer zu. Computer können aber nicht direkt mit diesen Nummern umgehen, sondern sie müssen erst mittels Codierung in eine Form gebracht werden, mit der ein Computer umgehen kann.

Bis zum Unicode Standard 3.0 umfasste der gesamte Unicoderaum nur den Bereich 0x0000-0xFFFF, also 65536 Zeichen. Dieser Bereich entspricht im Uni-

code 4 Standard der Basic Multilingual Plane (BMP) (siehe Kapitel 1.2). Daraus ergab sich automatisch ein 2-Byte Encoding (UCS-2, UTF-16), bei dem jedes Zeichen durch 2 Bytes repräsentiert wird. Es gibt **3 wichtige Encodierungsformate: UTF-8, UTF-16 und UTF-32**. Neben diesen Hauptformen gibt es noch eine Reihe anderer (UTF-7, UTF-7,5, UTF-EBCDIC, ...). Die Abkürzung UTF steht für die Kurzform von *UCS Transformation Format*.

UTF-16

UTF-16 ist die natürliche Wahl zum Codieren von Unicode-Texten, da ein starres Encodingsystem fixer Länge benutzt wird. Jedes Zeichen wird durch 2 Bytes (bzw. 4 Bytes) repräsentiert. Eine Codierung von Zeichen innerhalb der BMP benötigt 2 Bytes. Zeichen, die außerhalb der BMP liegen, werden mit Hilfe sogenannter *Surrogate Pairs* zusammengesetzt und belegen dadurch 4 Bytes. Die Surrogate Zeichen befinden sich zwischen 0xD800 und 0xDBFF (High Surrogates) sowie 0xDC00 und 0xDFFF (Low Surrogates)

Mit UTF-16 kann man 1.114.112 (2^{16} basis + 2^{20} Surrogate) Zeichen darstellen, was den 16 Unicode Planes entspricht und genug Platz für alle lebenden und toten Sprachen bietet. UTF-16 findet vor allem im asiatischen Raum Verwendung.

Die UTF-16 Codierung hat leider einige gravierende Nachteile:

- Sie ist nicht ASCII kompatibel
- Texte, die überwiegend aus alphabetischen Zeichen bestehen, sind doppelt so groß wie in anderen gängigen Codierungen (beispielsweise wie in ISO-8859-x).

In UTF-16 können UCS-4 Zeichen als Paare von UCS-2 gespeichert werden. Dies entspricht dem UCS-2 Standard der ISO (hier werden nur bis zu 65.535 verschiedene Zeichen unterstützt).

UTF-8

UTF-8 ist ein variabel langer Multi-Byte Zeichensatz. Da nicht alle Systeme mit 16-Bit umgehen können und die UTF-16 Codierung darüberhinaus gravierende Nachteile hat, wurde die UTF-8 Codierung entwickelt. UTF-8 benutzt eine *variable length* Codierung, das heißt jedes Zeichen wird als Folge von 1 bis 4 Bytes dargestellt. UTF-8 ist außerdem ASCII kompatibel, was bedeutet, dass ein Text der nur ASCII Zeichen enthält, sich in UTF-8 Codierung nicht von einem echten ASCII Text unterscheidet. Texte mit ideographischen Zeichen sind zwar wesentlich länger, Texte, die überwiegend aus lateinischen Zeichen bestehen, sind aber nur unwesentlich länger als in herkömmlichen Codierungen. Aus diesem Grund ist UTF-8 die dominierende Codierung für lateinische Schrift.

UTF-8 unterstützt den selben Zeichraum wie UTF-16 (alle 1.114.112 möglichen Unicode Zeichen) und könnte theoretisch den gesamten Umfang von UCS-4 abdecken.

Das UTF-8 Encoding ist definiert in ISO 10646-1:2000 Annex D, ebenso in RFC 3629, und unter Abschnitt 3.9 des Unicode Standards (4.0).

UTF-7

UTF-7 ist eine Adaptierung des Unicode Standards an die 7-Bit Welt (E-Mail Systeme!). Außer in zwingenden Gründen sollte UTF-7 nicht zum Einsatz kommen, weil es nicht nur ein *variable length* encoding darstellt, sondern auch die Codierung ein und des selben Zeichens von seiner Umgebung abhängig macht, was den Grundprinzipien von Unicode widerspricht.

UCS4 / UCS 2

UCS-4 ist eine 4 Byte (32-Bit) Codierung, die es ermöglicht, ausnahmslos alle (zukünftigen) Unicode Zeichen sämtlicher Ebenen ohne *Tricks* (Surrogate Pairs) darzustellen. Natürlich sind in UCS-4 codierte Texte sehr *sperrig* und UCS-4 ist von geringer praktischer Bedeutung.

UCS-2 ist die 2 Byte/16-Bit Variante von UCS-4. Da UCS-2 ausschließlich die BMP (Ebene 0) codieren kann, hat diese Codierung mit der Einführung von Unicode 4 stark an Bedeutung verloren und wurde komplett von UTF-16 verdrängt.

Anmerkung: Endianess UTF-16, UTF-32, UCS-2 und UCS-4 codierte Zeichen gibt es in *Big Endian (BE)* und *Little Endian (LE)* Variante, in Abhängigkeit der Plattform. Um erkennen zu können welche *Endianess* vorliegt, beginnen die codierten Texte mit einem speziellen Zeichen U+FEFF, dem BOM (Byte Order Mark), mit dem Namen *ZERO WIDTH NO-BREAK SPACE*, dessen Bytefolge (FFFE oder FEFF) mit der *Endianess* variiert. (siehe Kapitel 3.5.10)

UTF-32

UTF-32 ist eine, auf den Unicoderaum eingeschränkte, Variante des UCS-4 der ISO und unterstützt ebendso wie UTF-16 oder UTF-8 den gesamten Unicode Zeichenraum von 1.114.112 Zeichen. Diese Codierungsform wird praktisch noch nicht verwendet. Die obersten 2 Bytes sind Null Bytes; Die BOM für Big/Little Endian Darstellung sieht hier wie folgt aus: 0000FEFF für BE und FFFE0000 für LE.

UTF-EBCDIC

UTF-EBCDIC ist das Großrechner Pendant zur UTF-8 und stellt Kompatibilität zum EBCDIC Standard her. Weil für den EBCDIC Standard mehr als 128 Zeichen notwendig sind, ist der Algorithmus anders als bei UTF-8. Es kommt ein Zwischenformat namens UTF-8-Mod oder I8 zum Einsatz. Außerhalb von Großrechnerumgebungen hat UTF-EBCDIC keine Bedeutung und es eignet sich auch nicht zum Datenaustausch in heterogenen Umgebungen.

Ressourcen

- IANA character sets
- Coverage of European languages by ISO Latin alphabets

1.2.4 Dynamisch erzeugte Zeichen und Normierung

Dynamisch erzeugte Zeichen (Dynamic Composition)

Unicode bietet die Möglichkeit neue Zeichen dynamisch zu erzeugen, indem man ein Basiszeichen mit einem Kombinationszeichen (combining mark) combiniert. Jedes Kombinationszeichen kann dabei mit jedem Basiszeichen verbunden werden. Es können zum Beispiel Umlaute wie Ö als Folge von 2 Zeichen, nämlich O und ¨ (U+0308), dem Umlaut oder Tremma/Diarese Zeichen geschrieben werden. Die Combining Marks befinden sich in einem eigenen Block (Combining diakritic marks) von U+0300 bis U+036F.

Die geläufigsten diakritischen Zeichen liegen als separate Zeichen vor (static precomposed forms), schon alleine um kompatibel zu bestehenden anderen Standards zu sein.

Außer den Combining Marks, die hauptsächlich für diakritische Zeichen genutzt werden, existieren noch sogenannte *Presentation Forms*, die Ligaturen oder ähnliches beinhalten. Die Presentation Forms für lateinische Zeichen sind im Block von U+FB00 bis U+FB4F zu finden.

Begriffe

Äquivalenz Viele Texte lassen sich mit *static precomposed forms* und durch *dynamic composition* erzeugen. Der Standard bietet für alle static precomposed forms ein Äquivalenzmapping auf dynamic composed sequences. Beide Varianten gelten in Unicode als äquivalent und vollkommen gleichwertig. Lassen sich mehrere unterschiedliche Sequenzen erzeugen, so gelten auch alle diese als gleichwertig. Es wird empfohlen in den ersten Umsetzungsphasen die verwendeten Zeichen auf precomoposed forms einzuschränken (siehe Kapitel 3.1.6). In weiteren Ausbaustufen sollte dann zur Vermeidung von Kompatibilitätsproblemen und um

allgemein gültige String-Vergleichsoperationen durchführen zu können, eine eindeutige Darstellung erreicht werden. Dafür kommen die Normalisierungsformen (*Unicode Normalization Forms*) zum Einsatz

Combining Sequence (Kombinationsfolge)

Base character Ein Zeichen das sich nicht graphisch mit seinem Vorgängerzeichen verbindet und weiters kein Control- oder Format- Zeichen ist.

Combining character Ein Zeichen das sich in mit einem vohergehendem *base character* verbindet (Akzente, Diakrite, Hebräische Punkte, ...). Ein combining character hat alleine keine Bedeutung.

Nonspacing mark Ein combining character, der keinen Platz auf der Grundlinie beansprucht. Es kann sein, dass ein *nonspacing* mark die Plazierung seines *base characters* beeinflusst (zum Beispiel ist î breiter als ein i).

Spacing mark Ein *combining character* der kein *nonspacing mark* ist. Für die lateinische Schrift ohne Bedeutung.

Combining character sequence Eine Serie aus einem *base character* und ein oder mehr *combining characters*.

Static Precomposed Forms Ein einzelnes Zeichen, das einer bestimmten *combining sequence* entspricht, zum Beispiel ä, ö, g̊

Decomposition (Dekomposition)

Decomposable character Ein Zeichen, das äquivalent zu einer Folge anderer Zeichen ist, zum Beispiel: é = e +´. Es gibt zwei unterschiedliche Formen der Dekomposition: Canonical und Compatibility Decomposition.

Canonical decomposable character Das Zeichen Å (U+00C5 LATIN CAPITAL LETTER A WITH RING ABOVE) kann als Folge von A (U+0041 LATIN CAPITAL LETTER A) und ˚ (U+030A, COMBINING RING ABOVE) dargestellt werden. Zeichen können auch in Folgen von mehr als 2 Zeichen dekomponiert werden. Nach erfolgter Dekomposition wird ein *canonical ordering* durchgeführt.

Canonical ordering Wird ein *canonical decomposable character* mit 2 oder mehr combining marks in eine combining sequence umgewandelt, gibt es mehrere Möglichkeiten diese Sequenz zu bilden. Da die Reihenfolge der *combining marks* Einfluß auf das Ausehen des resultierenden Zeichens hat, kann die Reihenfolge nicht beliebig sein. Zur Anordnung wird die sog. *combining class* Eigenschaft (=Kombinationsklasse) herangezogen. Diese wird

durch eine Zahl von 0 bis 255 repräsentiert, die bei den *combining characters* bestimmt, in welcher Reihenfolge diese typografisch interagieren und somit bei der Sortierreihenfolge mitbestimmend sind. Die *combining characters* werden nach aufsteigender Kombinationsklasse sortiert, wobei sich die Reihenfolge von *combining characters* der selben *combining class* nicht ändert.

Compatibility decomposable characters *Compatibility characters* sind Zeichen die nur deshalb im Unicodestandard vorkommen, weil sie in bestehenden Zeichensätzen (wie ISO-8859-x) bereits vorgekommen sind. Sie dienen in erster Linie dazu, die *round-trip* Umcodierungen zu ermöglichen. Das große A mit einem Ring (Å) kommt im Unicode Standard an zwei Stellen vor: als U+00C5 (LATIN CAPITAL LETTER A WITH RING ABOVE) und als U+212B (ANGSTROM SIGN). Das Angstrom sign ist nur aus Rückwärtskompatibilitätsgründen enthalten. Es kann im Allgemeinen durch U+00C5 ersetzt werden. Diese Dekomposition kann aber verlustbehaftet sein. Im lateinischen Schriftsystem spielen *compatibility decomposable characters* nur eine untergeordnete Rolle. Außer dem Angstrom Zeichen gibt es keine *compatiblity characters* die bei der Eingabe Probleme bereiten könnten.

Unicode Normalization Forms (Normierungsformen)

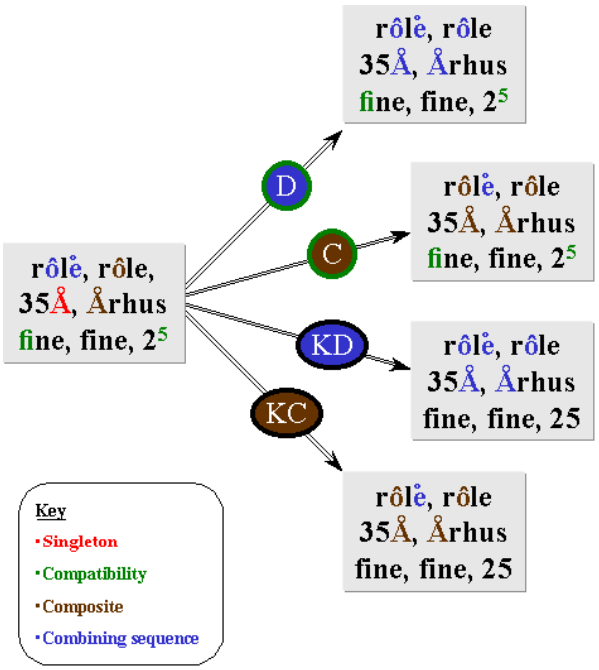
Da Zeichen auf mehrere Arten ausgedrückt werden können, müssen die verschiedenen Arten klassifiziert werden. Man unterscheidet:

- D - Canonical Decomposition
- C - Canonical Decomposition + Canonical Composition
- KC - Compatibility Decomposition
- KD - Compatibility Decomposition + Canonical Composition

An einem Beispiel illustriert bedeutet das

Form	Normalisation
D	a + ff + a + i + r + e + ´
C	a + ff + a + i + r + é
KC	a + f + f + a + i + r + e + ´
KD	a + f + f + a + i + r + é

Details entnehmen Sie bitte der folgenden Grafik (© Unicode Consortium).



Aus Gründen visueller Informationsübermittlung sollte diese Abbildung in farbiger Darstellung betrachtet werden.

Bei allen Normalisierungsformen werden *singleton characters* ersetzt. Bei den Formen C und D werden Zeichen mit *compatibility composition* beibehalten - bei den Formen KC und KD werden auch diese ersetzt. Letzteres kann zu Informationsverlust führen, falls die Information nicht zusätzlich durch Markup/Styling vermittelt werden kann. Der Einsatz von KC oder KD muss wohlüberlegt sein.

Die Formen D und KD ersetzen *precomposed characters* durch ihre kanonischen Dekompositionen, während C und KC kombinierte Zeichenfolgen (*combining character sequences*) auf ihre Komposite mappen.

Die Normalisierungsform C benutzt *canonical composite characters* wo immer möglich und behält den Unterschied zwischen Zeichen die *compatibility equivalents* sind. Form C ist die vom W3C bevorzugte Form für XML und verwandte Standards. Die Form C mappt Zeichenfolgen NICHT zu *compatibility composites* - aus f f wird also nicht ff (U+FB00).

ASCII Texte sind allen Normalisierungsformen gegenüber invariant.

Unicode Implementierungen die *combining marks* nicht unterstützen (Level 1 in ISO/IEC 10646-1) nutzen Normalisierung C.

Zusammenfassung Behandlung von *compatibility composites* im Quelltext

- Formen C und D behalten *compatibility composites* bei
- KC und KD ersetzen *compatibility composites*

- Keine der Formen erzeuget *compatibility composites* die nicht bereits vorhanden waren

In der Praxis kommt meist die Normalisierungsform C (NFC, Normalization Form C) zum Einsatz. Dies ist auch die vom W3C bevorzugte Form. Es wird empfohlen, wenn Normalisierungen durchgeführt werden, diese vom W3C bevorzugte Form C zu verwenden (siehe Kapitel **??**). Die vom Unicode Consortium bevorzugte Normalisierung ist, aufgrund der noch weitergehenden Zusammenführungsmöglichkeiten und damit erweiterten Normaliserung, die Form K (NFK, Normalization Form K).

Ressourcen

- Unicode Normalization Forms Version Unicode Standard Annex #15
- Beispiele aus dem Annex
- Normalization Charts

Chapter 2

Problemstellung

Einleitung

Dieses Kapitel behandelt die sprach- und schriftbezogenen Grundlagen zum Thema Sonderzeichen und diakritische Zeichen

2.1 Sonderzeichen und diakritische Zeichen

Dieser Abschnitt befasst sich mit den für die österreichische Verwaltung bedeutsamen Sonderzeichen und diakritischen Zeichen.

2.1.1 Das lateinische Alphabet

Einleitung

Zum lateinischen Alphabet zählen streng genommen nur die Buchstaben a-z und A-Z. Damit lassen sich mehrere Sprachen wie zum Beispiel Latein, Italienisch, Englisch oder Niederländisch schreiben. Streng genommen lassen sich weder Italienisch noch Niederländisch mit ASCII schreiben, sogar Englisch benützt diakritische Zeichen in französischen Lehnwörtern. Das lateinische Alphabet reicht aber bei weitem nicht für alle Sprachen aus, die man grundsätzlich mit lateinischen Buchstaben schreibt.

Darüberhinaus werden auch viele Sprachen, die gewöhnlich mit anderen Alphabeten, Abjaden, Silbenschriften oder Ideogrammen geschrieben werden, mit lateinischen Zeichen umschrieben. Diese Umschriften erfordern meist diakritische Zeichen oder Erweiterungen am Alphabet.

Unicode

Der Unicode Standard kennt derzeit über 700 Zeichen, die dem lateinischen Schriftsystem zugerechnet werden. Darunter befindet sich natürlich eine große Zahl an diakritischen Kombinationszeichen.

Die folgenden Unicode Blöcke enthalten lateinische Zeichen

- Latin 1
- Latin-1 Supplement
- Latin Extended A
- Latin Extended B
- Latin Ext. Additional
- Combining diacritic marks
- Alphabetic Presentation Forms

Streng genommen zählen auch die Zeichen der IPA Extensions zu den lateinischen Zeichen. Da die Zeichen dieses Blocks aber ausschließlich für die phonetische Schrift gebraucht werden, sind sie hier nicht von Belang.

Details unter Zeichen für europäische Sprachen (siehe Kapitel 3.1)

Ressourcen

- Everson Typography - Alphabeths of Europe
- Omniglot.com Schriftsysteme im Vergleich
- Wikipedia: Latin Alphabet
- Wikipedia: Alphabets derived from the Latin

2.1.2 Diakritische Zeichen

Einleitung

Diakritsche Zeichen (diacritic marks) sind Zeichen, die an anderen Zeichen (im Allgemeinen Buchstaben) angebracht werden und deren Bedeutung bzw. Betonung oder Aussprache verändern. Diakritische Zeichen sind in vielen Sprachen gebräuchlich, das gleiche Zeichen kann jedoch je nach Sprache eine unterschiedliche Bedeutung haben. Dies kann sogar soweit gehen, dass eine Kombination von Buchstabe und diakritischem Zeichen in einer Sprache ein eigenständiges Zeichen mit einem eigenen Laut ist, während diese in anderen Sprachen nur die Betonung angibt. Die deutschen Umlaute gelten z. B. als eigene Zeichen, ein ä im Französischen hingegen bezeichnet eine Diarese, verlangt also, dass der Vokal getrennt von seinem Vorgänger ausgesprochen wird.

Diakritische Zeichen in Unicode

Grundsätzlich gibt es zwei Möglichkeiten diakritische Zeichen zu erzeugen: Durch dynamische Komposition (siehe Kapitel 1.2.3) und indem man ein passendes vorgefertigtes Zeichen (precomposed character) auswählt. Der Unicode Standard fordert, dass beide Methoden zu unterstützen sind.

Ressourcen

- Wikipedia: Diakritische Zeichen
- Diakritsche Zeichen

2.1.3 Sonderbuchstaben und diakritische Zeichen

Einleitung

Außer den lateinischen Buchstaben a-z, A-Z benötigen einige Sprachen noch weitere Buchstaben.

Echte Sonderbuchstaben

Zeichen	Code	Name
Ð	U+00D0	Eth (Isländisch)
ð	U+00F0	eth (Isländisch)
Æ	U+00C6	AE (Isländisch, Dänisch, Norwegisch)
æ	U+00E6	ae (Isländisch, Dänisch, Norwegisch)
Þ	U+00DE	Thorn (Isländisch)
þ	U+00FE	thorn (Isländisch)
ß	U+00DF	scharfes s (Deutsch)

Uneigentliche Sonderbuchstaben

Diese Zeichen sind zwar Ableitungen von lateinischen Buchstaben durch diakritische Zeichen, gelten aber in den zugehörigen Sprachen als eigene Zeichen. Das macht sich vor allem in der Transkription bemerkbar.

Zeichen	Beschreibung
Ä, Ö, Ü, ä, ö, ü	Deutsche Umlaute
Å, å	Dänisch, Norwegisch, Schwedisch
Ø, ø	Dänisch, Norwegisch

Ressourcen

- Alphabets derived from the Latin Alphabet

2.1.4 Ligaturen und Präsentationsformen

Einleitung

Außer den Sonderbuchstaben und diakritischen Zeichen gibt es noch zwei weitere Arten von *Sonderzeichen*: Ligaturen (Präsentationsformen) und Digraphen.

Ligaturen

Ligaturen (z. B. U+FB03 ffi) dienen rein der Verbesserung des Schriftbildes. Sie sollten in Eigennamen keine Verwendung finden. Eine Ausnahme bilden die Ligaturen mit Zeichencharakter (Æ, Œ, ß, ...), die aber zu den *gewöhnlichen* lateinischen Zeichen zählen.

Digraphen

Insbesondere Ostsprachen, aber auch andere Sprachen setzen Digraphen oder Trigraphen ein, um eine Umschrift für Zeichen des Kyrillischen zu haben. Im

Deutschen zum Beispiel ch und sch. Diese, im Deutschen üblichen, Digraphen bzw. Trigraphen werden allerdings nie als eigene Zeichen geschrieben, sondern immer aus den einzelnen Bestandteilen zusammengesetzt. Anders ist der Fall für die in den Ostsprachen geläufigen Digraphen, denen der Unicode Standard eigene Zeichen zuordnet. Aber auch in den osteuropäischen Ländern ist es eher üblich, diese Zeichen auf dem Computer durch zwei separate Zeichen zu repräsentieren, auch wenn sie beim Abzählen der Zeichen als nur ein Zeichen gezählt werden.

Zeichen	Code	Beschreibung
	U+01C4	DŽ (Serbokroatisch, Slovakisch)
	U+01C5	Dž (Serbokroatisch, Slovakisch)
	U+01D6	dž (Serbokroatisch, Slovakisch)
	U+01C7	LJ (Serbokroatisch)
	U+01C8	Lj (Serbokroatisch)
	U+01C9	lj (Serbokroatisch)
	U+01CA	NJ
	U+01CB	Nj
	U+01CC	nj
	U+01F1	DZ (Slovakisch, Ungarisch)
	U+01F2	Dz (Slovakisch, Ungarisch)
	U+01F3	dz (Slovakisch, Ungarisch)

Ressourcen

- Unicode.org: Alphabetic Presentation Forms (Ligaturen)

2.2 Rechtlicher Rahmen

Maßgebliche Rechtsquelle für die Schreibweise von Personennamen im Personenstandswesen ist das *Personenstandsgesetz (PStG)*. § 11 Abs. 1 PStG trägt der Verpflichtung Rechnung, die Österreich durch die Ratifizierung des Übereinkommens über die Angabe von Familiennamen und Vornamen in den Personenstandsbüchern, BGBl. Nr. 308/1980, übernommen hat. Auf die sich aus diesem Übereinkommen ergebende Pflicht wird auch in den § 5 Abs. 3 bis 5 der *Personenstandsverordnung (PStV)* Bezug genommen.

Auszug aus dem Personenstandsgesetz

Personennamen *§ 11. (1) Personennamen sind aus der für die Eintragung herangezogenen Urkunde buchstaben- und zeichengetreu zu übernehmen. Sind in der Urkunde andere als lateinische Schriftzeichen verwendet worden, müssen die Regeln für die Transliteration beachtet werden.*
(2) Zur Ermittlung des durch Abstammung erworbenen Familiennamens sind, soweit die Person, auf die sich die Eintragung bezieht, nicht anderes beantragt, nur die Urkunden der Person(en) heranzuziehen, von der (denen) der Familienname unmittelbar abgeleitet wird.
(3) Ist für den Familiennamen einer im § 2 Abs. 2 angeführten Person oder der Person(en), von der (denen) der Familienname abgeleitet wird, oder für den Vornamen einer im § 2 Abs. 2 angeführten Person eine vom rechtmäßigen Familiennamen (Vornamen) abweichende Schreibweise gebräuchlich geworden, ist auf ihren Antrag der Familienname (Vorname) in der gebräuchlich gewordenen Schreibweise einzutragen. Der Antrag bedarf der Zustimmung des Ehegatten, wenn dieser den gleichen Familiennamen führt und dem Personenkreis des § 2 Abs. 2 angehört.
(4) Auf Antrag einer im § 2 Abs. 2 angeführten Person ist in alle sie betreffende Eintragungen in den Personenstandsbüchern ein Vermerk (§ 13 Abs. 2) in sinngemäßer Anwendung des Abs. 3 einzutragen.
(5) Die Eintragung des Personennamens nach Abs. 3 und 4 ist für alle weiteren dieselbe Person betreffenden Eintragungen maßgebend; die nunmehrige Schreibweise des Familiennamens oder Vornamens ist auch in den früheren dieselbe Person betreffenden Eintragungen zu vermerken (§ 13 Abs. 2). Das gleiche gilt für die Schreibweise des Familiennamens des Ehegatten, der dem Antrag nach Abs. 3 und 4 zugestimmt hat, und des zur Zeit der Eintragung minderjährigen Kindes, das dem Personenkreis des § 2 Abs. 2 angehört, wenn es seinen Familiennamen vom Antragsteller ableitet.

Auszug aus der Personenstandsverordnung

Verordnung des Bundesministers für Inneres vom 14. November 1983, BGBl. Nr. 629/1983, zur Durchführung des Personenstandsgesetzes

§ 5. (1) Ist Aufgrund einer fremdsprachigen Urkunde einzutragen, so hat die Partei (§ 15 Abs. 7 Z 1 und 2 des Gesetzes) eine von einem allgemein beeideten gerichtlichen Dolmetscher oder Übersetzer angefertigte Übersetzung vorzulegen. Trifft die Vorlagepflicht nicht eine Partei, so hat die Personenstandsbehörde die Übersetzung selbst anfertigen zu lassen. Die im Volksgruppengesetz, BGBl. Nr. 396/1976, und den dazu ergangenen Durchführungsbestimmungen enthaltenen besonderen Regelungen für die Übersetzung von in der Sprache der Volksgruppe abgefassten Urkunden bleiben unberührt.
(2) Ist die fremdsprachige Urkunde in lateinischer oder in der früher gebräuchlichen deutschen Schrift abgefasst, so kann auf eine Übersetzung verzichtet werden, wenn die für die Eintragung maßgebenden Daten, auch ohne Übersetzung, verständlich sind, oder wenn der Standesbeamte die fremde Sprache hinreichend beherrscht.
(3) Wird auf Grund einer Urkunde eingetragen, die Personen- oder Ortsnamen in lateinischer oder in der früher gebräuchlichen deutschen Schrift enthält, so müssen die Namen buchstaben- und zeichengetreu wiedergegeben werden. Entspricht einem früheren deutschen Schriftzeichen kein lateinisches, so ist eine Transliteration vorzunehmen.
(4) Wird auf Grund einer Urkunde eingetragen, die Personen- oder Ortsnamen in fremder Schrift enthält, so müssen die Namen so weit wie möglich durch Transliteration wiedergegeben werden.
(5) Bei der Transliteration sind folgende von der Internationalen Normenorganisation (ISO) empfohlene Normen einzuhalten:

1. *ISO/R 9 (zyrillisch - lateinisch)*
2. *ISO/R 233 (arabisch - lateinisch)*
3. *ISO/R 259 (hebräisch - lateinisch)*
4. *ISO/R 843 (griechisch - lateinisch)*

Auszug aus dem Übereinkommen über die Angabe von Familiennamen und Vornamen in den Personenstandsbüchern (StF: BGBl. Nr. 308/1980)

Artikel 2 *Soll von einer Behörde eines Vertragsstaates eine Eintragung in ein Personenstandsbuch vorgenommen werden und wird zu diesem Zweck eine Abschrift eines Personenstandseintrags oder ein Auszug aus diesem oder eine andere Urkunde vorgelegt, die die Familiennamen und Vornamen in den gleichen Schriftzeichen wiedergibt wie in denjenigen der Sprache, in der die Eintragung*

vorgenommen werden soll, so sind diese Familiennamen und Vornamen buchstabengetreu ohne Änderung oder Übersetzung wiederzugeben. Die in diesen Familiennamen und Vornamen enthaltenen diakritischen Zeichen sind ebenfalls wiederzugeben, selbst wenn die Sprache, in der die Eintragung vorgenommen werden soll, solche Zeichen nicht kennt.

Die Pflicht zur zeichengetreuen Wiedergabe nach § 5 Abs. 3 PStV bezieht sich auf die Wiedergabe diakritischer Zeichen, die den üblichen lateinischen Schriftzeichen beigefügt werden. Fehlt für ein fremdes Schriftzeichen eine Entsprechung in der lateinischen Schrift, so ist eine Transliteration vorzunehmen, dies ist die möglichst genaue Wiedergabe des Lautwertes eines fremden Schriftzeichens, § 5 Abs. 4 PStV.

Dem Personenstandsgesetz folgend, sind grundsätzlich alle lateinischen Schriftzeichen zu unterstützen.

2.3 Behördenpraxis

Anwendungsbereich

Durch die Verwendung diakritischer Zeichen im Zentralen Melderegister, im Stammzahlenregister, im Standarddokumentenregister und auf der Bürgerkarte ist deren Verwendung auch dort empfehlenswert, wo Anwendungen mit diesen Registern elektronisch verknüpft sind, bzw. wo die Bürgerkarte verwendet wird.

Behördenpraxis

In der Behördenpraxis reduziert sich die Zahl der Sonderzeichen mitunter aufgrund der Tatsache, dass beglaubigte Übersetzungen oftmals keine oder nur eine geringe Zahl von Sonderzeichen aufweisen.

Folgende Sprachen werden in der Praxis besonders häufig benötigt:

- Sprachen der in Österreich anerkannten Minderheiten
- Sprachen der EU-Mitgliedsstaaten
- Sprachen der EU-Kandidatenländer bzw. Beitrittswerber

Europäische Sprachen mit lateinischer Schrift: Albanisch, Asturisch, Baskisch, Bosnisch, Bretonisch, Cornisch, Dänisch, Deutsch, Englisch, Estnisch, Färöresisch, Finnisch, Französisch, Friaulisch, Gälisch, Galizisch, Grönländisch, Holländisch, Isländisch, Irisch, Italienisch, Jèrriais, Katalanisch, Korsisch, Kroatisch, Lettisch, Lithauisch, Lombardisch, Lützleburgisch, Luxemburgisch, Maltesisch, Niederländisch, Norwegisch, Okzitanisch, Piemontesisch, Polnisch, Portugiesisch, Rätoromanisch, Rumänisch, Sami, Schottisch, Schwedisch, Slowakisch, Slowenisch, Sorbisch, Spanisch, Tschechisch, Türkisch, Tzimbrisch, Ungarisch, Walisisch.

Alle weiteren (nicht-europäische) Sprachen mit lateinischer Schrift werden in der gängigen Praxis zumeist nicht unterstützt. Die dazu notwendigen Sonderzeichen (ca. 700) unterscheiden sich nur marginal von einander und deren korrekte Eingabe in maschinenbasierte Systeme setzt ein hohes Maß an Verständnis für die unterschiedlichen Sprachen voraus. Dazu zählen beispielsweise:

Afaan Oromo, Afrikaans, Aymara, Azeri, Cheyenne, Comanche, Filipino, Hausa, Hawaiian, Kiribati, Kurdish, Ido, Indonesian, Lojban, Malay, Maori, Nahuatl, Navajo, Naxi, Quechua, Samoan, Swahili, Tagalog, Turkmen, Vietnamese, Volapük, Yoruba, Zulu.

Chapter 3

Diakritische Zeichen in der Praxis

Dieses Kapitel bietet einen Überblick über den Umgang mit diakritischen Zeichen. Es werden die Bereiche Zeichenauswahl, Datenein- und Ausgabe, Datenspeicherung und -verarbeitung sowie Datenaustausch behandelt. Empfehlungen sind als solche hervorgehoben und sollten nur in begründeten Ausnahmefällen missachtet werden.

3.1 Empfehlungen hinsichtlich Sprachen, Zeichen und Komposition

Einführung

Dieses Kapitel beinhaltet die von behördlichen Anwendungen zu unterstützenden Zeichen/Sprachen und die zu verwendenden Codierungen und Normalisierungen.

3.1.1 Zeichen für europäische Sprachen

Die Testseite zu folgendem Verweis beinhaltet zur Veranschaulichung die Sonderzeichen der in Europa gebräuchlichen Sprachen mit lateinischer Schrift.

- Europäische Sprachen und ihre Sonderzeichen (siehe separates Paket Mustercode/Beilagen)

Aus dieser Testseite geht hervor, dass sich fast alle europäischen Sprachen mit lateinischer Schrift in Basic Latin, Latin-1 Supplement und Latin Extended A schreiben lassen. Einzig Bosnisch-Kroatisch-Serbisch und Rumänisch erfordern theoretisch mehr Zeichen, doch die Praxis zeigt, dass diese Zeichen auch in den Ursprungsländern nicht benutzt werden, weil sie vom dort überwiegend gebräuchlichen ISO-8859-2 latin-2 Zeichensatz nicht unterstützt wurden.

3.1.2 Empfehlung: Zu unterstützende Sprachen

Die gewählten Sprachen sind europäische Sprachen (inkl. Türkisch) mit lateinischer Schrift.

- Albanisch
- Asturisch
- Baskisch
- Bosnisch
- Bretonisch
- Dänisch
- Deutsch
- Englisch
- Estnisch
- Färöresisch

- Finnisch
- Französisch
- Friaulisch
- Gälisch
- Galizisch
- Grönländisch
- Holländisch
- Isländisch
- Italienisch
- Katalanisch
- Kroatisch
- Lettisch
- Litauisch
- Luxemburgisch
- Maltesisch
- Norwegisch
- Okzitanisch
- Polnisch
- Portugiesisch
- Rätoromanisch
- Rumänisch
- Sami (Nördl.)
- Schwedisch
- Slowakisch
- Slowenisch
- Sorbisch

- Spanisch
- Tschechisch
- Türkisch
- Ungarisch
- Walisisch

3.1.3 Empfehlung: Unicode

Zu unterstützen sind die Unicode-Codepoints aus der separaten Konvention "Diakritische Zeichen" (DZ) in der letztgültigen Version in den in der Konvention genannten Kodierungen.

3.1.4 Empfehlung: Hoheitliche und privatwirtschaftliche Verwaltung

Applikationen, die sich aus dem Personenstandswesen ableiten, müssen ehestmöglich auf Unicode umgestellt werden (z. B.: Staatsbürgerschaftswesen (Einbürgerungen), ...). Andere Applikationen sollten so rasch wie möglich (zum Beispiel im Zuge des nächsten größeren Releases) umgestellt werden, um Inkonsistenzen zu vermeiden. Applikationen der privatwirtschaftlichen Verwaltung können dabei nachrangig behandelt werden. Mehr dazu unter Rechtlicher Rahmen (siehe Kapitel 2.2).

3.1.5 Empfehlung: Sprachen der anerkannten Minderheiten

Ist eine Unicode Unterstützung technisch nicht umsetzbar, können die Sprachen der anerkannten Minderheiten (siehe Kapitel 2.2) durch den Einsatz der ISO-8859-2 (Latin-2) Codierung unterstützt werden.

3.1.6 Empfehlung: Verzicht auf dynamische Komposition

Diakritische Zeichen sollten durch ihre vorgefertigte Zeichen (precomposed characters) dargestellt werden, nicht durch Zeichen+diakritischer Marker (dynamische Komposition).

3.1.7 Empfehlung: Sonderzeichen der Ostsprachen

Die bosnisch/kroatisch/serbischen Digraphen sollten durch zwei Zeichen geschrieben werden. Die im Rumänischen gebräuchlichen S, s, T, t mit Komma darunter: Ș, ș, Ț, ț (U+0218 bis U+012B) können in erster Näherung durch S und T mit Cedille: Ş, ş, Ţ, ţ (U+015E, U+015F, U+0162, U+0163) ersetzt werden.

3.1.8 Empfehlung: Latin Extended B

Neue Desktop Entwicklungen für den behördlichen Einsatz sollten nach Maßgabe der Möglichkeiten bereits auch die relevanten Zeichen aus Latin Extended B (Bereich U+0180 bis U+024F) unterstützen.

3.1.9 Relevante Unicode Ranges

Die folgende Tabelle listet alle Unicode Blöcke, die lateinische Zeichen beinhalten, mit ihren Unterteilungen laut den Codecharts des Unicode Consortiums auf.

Von	Bis	Bezeichnung
		Latin 1
0000	001F	C0 Controls
0020	002F	ASCII punctuation and symbols
0030	0031	ASCII digits
003A	0040	ASCII punctuation and symbols
0041	005A	Upercase Latin
005B	0060	ASCII punctuation and symbols
0061	007A	Lowercase Latin
007B	007e	ASCII punctuation and control
		Latin-1 Supplement
0080	009F	C1 Controls
00A0	00BF	Latin punctuation and symbols
00C0	00D6	Letters
00D7		Math
00D9	00F6	Letters
00F7		Math
00F8	00FF	Letters
		Latin Extended A
0100	007F	European Latin
		Latin Extended B
0180	01BF	African, Asian, archaic European and phonetic
01C0	01C3	African clicks
01C4	01CC	Croatian digraphs (2)
01CD	01DC	Pinyin
01DD	01FF	Additions
0200	0217	Slovenia, Croatian
0218	021B	Romanian
021C	0229	Misc. additions
022A	0233	Livonian
0234	0236	Sineinlogy
		Latin Ext. Additional
1E00	1E9B	Latin general Extension
1EA0	1EF9	Vietnamese
		Combining diacritic marks
0300	0333	Ordinary diacritics
0334	0338	Overstruck diacritics
0339	033F	Additions
0340	0341	Vietnamese tone marks (deprec.)
0342	0345	Greek
0346	034E	IPA
034F		Grapheme joiner
0350	0357	Uralic
035D	0362	Double Diacritics
0363	036F	Medivial superscript letters
		Alphabetic Presentation Forms
FB00	FB06	Latin ligatures

3.1.10 Empfehlung: Zeichen

Relevante Zeichen

Die zu unterstützenden Zeichen und Kodierungen sind in der Konvention “Diakritische Zeichen” in der jeweils letztgültigen Version aufgelistet bzw. beschrieben.

3.2 Schnittstellen

3.2.1 Formate für den Datenaustausch

SOAP Webservices

SOAP Webservices setzen XML Nachrichten (siehe Kapitel 3.2.1) zum Informationsaustausch ein. Diese sind laut W3C in UTF-8 mit Normaliserungsform C zu halten. Auf diese Weise können alle für den allgemeinen Gebrauch empfohlenen Sonderzeichen und diakritischen Zeichen übermittelt werden.

Services mit Sonderzeichenpflicht Services, die aus gesetzlicher Sicht zur Führung von Sonder- und diakritischen Zeichern verpflichtet sind, müssen darauf achten, dass zur Eintragung angelieferte Daten auch selbige Sonderzeichen beinhalten.

Beim Lesezugriff auf Daten sind die Applikationen angehalten, die Suche auch über vereinfachte Daten (siehe Kapitel 3.4.3) zu ermöglichen. Antworten sind generell in der korrekten diakritischen Form zu übermitteln.

Es steht der Applikation frei, ein Interface anzubieten, dass Sonderzeichen außerhalb des ISO-8859-1 Raumes vereinfacht (siehe Kapitel 3.4.3) oder als Escapesequenz (&#xuuuu) liefert. Eine solche Vereinfachung kann auch über einen Proxy-Server erfolgen, oder clientseitig über eine Library gelöst werden.

Services ohne Sonderzeichenpflicht Liegt keine gesetzliche Verpflichtung vor, sollten neugeschaffene, oder redesignte Dienste trotzdem diakritische Zeichen unterstützen, um möglichst rasch eine Homogenisierung zu erreichen.

REST Webservices

Die Übergabe von Daten als GET oder POST Parameter sollte ebenfalls in UTF-8 Form erfolgen. Details unter URL Encoding (siehe Kapitel 3.2.6)

3.2.2 XML

Informationen des W3C

XML ist per Definition Unicode-tauglich. Die Defaultcodierung ist UTF-8. Jeder konforme XML-Parser muss diese Codierung unterstützen (alle anderen sind optional). Das W3C verlangt außerdem *early normalization* (Normalisierung vor weiterer Verarbeitung) in der Normalisierungsform C (NFC).

XML bietet überdies die Möglichkeit, den kompletten Unicode-Zeichensatz auch in NICHT-XML Codierungen einzubringen. Die gebräuchlichen Escape-Darstellungen wie zum Beispiel ģ erzeugen auch in ISO-8859-1 codierten Dateien das Unicodezeichen U+0123.

3.2.3 Empfehlung: Codierung

Die vom W3C empfohlene Defaultcodierung UTF-8 und die Normalisierungsform C sind beizubehalten.

3.2.4 Empfehlung: Element- und Attributnamen

Für Element- oder Attributnamen sollen ausschließlich A-Z, a-z, 0-9 und _ verwendet werden. Weiters sollten sich unterschiedliche Element- oder Attributnamen nicht nur durch die Groß- und Kleinschreibung unterscheiden, um Komplikationen in der programmatischen Verarbeitung vermeiden.

3.2.5 Webinterfaces

Problemstellung und Vorgangsweise

Beim Einsatz von diakritischen Zeichen in Webformularen muss darauf geachtet werden, dass die Zeichen auch in UTF-8 Codierung übergeben werden. Dazu sind mehrere Schritte erforderlich:

- Die Seite, die das Formular enthält, muss bereits in UTF-8 codiert sein (notfalls ist ein Sonderzeichen in einem Kommentar einzubauen, um die ASCII Kompatibilität zu überlisten).

  ```
  <meta http-equiv="content-type" content="text/html; charset=UTF-8">
  ```

- XHTML Dokumente sollten zusätzlich einen XML Prolog mit Encoding Angabe besitzen

  ```
  <?xml version="1.0" encoding="UTF-8" ?>
  ```

- Das Formular muss als einzige erlaubte Codierungsvariante UTF-8 enthalten

  ```
  <form action="out.php" method="GET" accept-charset="UTF-8">
  ```

- Es kann nötig sein, im <form> Element zusätzlich noch

  ```
  enctype="multipart/form-data"
  ```

  anzugeben, um ältere Browser zur Kooperation zu überreden.

- Der Webserver darf nicht angewiesen sein, das Encoding von Webseiten zu überschreiben (z. B: bei Apache die Direktive AddDefaultCharset), oder er muss angewiesen sein, das Encoding auf UTF-8 zu setzen.

Beispiel

```
<html>
<head>
<title>html utf test: input</title>
<meta http-equiv="content-type" content="text/html; charset=UTF-8">
</head>
<body>
  <!-- ä ß ǵ  UTF-8 enforcer comment -->
  <form action="out.php"
        method="GET"
        accept-charset="UTF-8"
        enctype="multipart/form-data">

     <input type="text" name="in" value="ä ǵ ḧ" />
     <input type="submit" />
  </form>

</body>
</html>
```

3.2.6 Empfehlung: Durchgängiger Einsatz

Insbesondere bei Webapplikationen sollte UTF-8 **durchgängig** eingesetzt werden, um implizierte Umcodierung zwischen Seiten mit unterschiedlicher Codierung zu unterdrücken.

Ressourcen

- W3C Internationalisation Information on default charsets on webservers
- W3C Tutotrial zu (X)HTML und Encodings
- W3C - i18n: The HTTP charset parameter
- Beispiele zur Verarbeitung in diversen Programmiersprachen finden sich im Kapitel Programmatische Behandlung (siehe Kapitel 3.4.5)

3.2.7 URL-Encoded Information

Datenübermittlung per URL Encoding

URL-Codierung macht aus beliebigen Zeichenketten reine ASCII-Strings, deren mögliche Zeichen weiters auf die in URLs zulässigen Zeichen eingeschränkt sind. Die zu codierenden Strings können im Prinzip in beliebiger Codierung vorliegen. Eine Decodierung und weitere Verarbeitung ist allerdings nur dann sicher möglich, wenn das ursprüngliche Encoding bekannt ist (mehr Informationen im Abschnitt UTF-8 Codierung erkennen. (siehe Kapitel 3.5.10)).

URL Encoding wird in RFC 1738 und Folgenden definiert. Zu den verbotenen Zeichen zählen zum Beispiel: Leerzeichen, CR, LF, TAB, +, ?, &, ...

Die Defaultcodierung für HTTP ist ISO-8859-1. Dieses Encoding kann aber durch die HTTP charset Direktive überschrieben werden. Dies passiert zum Beispiel wenn diakritische Zeichen aus einem UTF-8 codierten Formular heraus übergeben wird.

Beispiel

```
aus
abc def+=&%äöü
wird
abc+def%2B%3D%26%25%E4%F6%FC%22
```

3.2.8 Empfehlung: URL Codierung in UTF-8

URL-codierte Strings zur Informationsübermittlung sollen als Ausgangsformat UTF-8 besitzen, auch wenn die vorkommenden Zeichen in ISO-8859-1 codiert werden könnten.

3.3 Dateneingabe

Dieses Kapitel beschäftigt sich mit den Möglichkeiten der Dateneingabe durch Bürger und Sachbearbeiter und der Datenübernahme von der Bürgerkarte.

3.3.1 Dateneingabe in Webinterfaces

Einleitung

Dieser Abschnitt beschäftigt sich mit der Eingabe von Sonderzeichen in Webinterfaces. Das Problem der korrekten Übergabe von Zeichen via Webinterfaces wird in Abschnitt Schnittstellen/Webinterfaces (siehe Kapitel 3.2.4) behandelt.

Problemstellung

Um Sonderzeichen einzugeben, die sich nicht über die Tastatur eingeben lassen, müssen externe Hilfsmittel zum Einsatz kommen. Es stehen folgende Möglichkeiten zur Auswahl:

- Boardmittel (Charactermaps) werden in Kapitel Eingabe/Desktopapplikationen (siehe Kapitel 3.3.2) behandelt
- Webbasierte Eingabehilfen in JavaScript

Anforderungen

- Nur Eingabe der zulässigen Sonderzeichen
- Einfache Unterscheidung ähnlicher Zeichen
- Schneller Zugriff für Vielschreiber

Möglichkeiten

Die im Folgenden aufgelisteten Möglichkeiten setzen, abgesehen von *dead keys*, JavaScript voraus. Dead keys sind Tasten auf einer Tastatur, die kein unmittelbares Ergebnis erzeugen, wenn sie gedrückt werden, aber das Ergebnis des darauf folgenden Zeichens ändern.Ohne Einsatz von JavaScript muss man sich auf die von der jeweiligen Plattform angebotenen Methoden verlassen.

Character picker - Auflistung aller Sonderzeichen in Unicode Reihenfolge

+ Deckt sich mit Desktopapplikationen
− Unübersichtlich

Dynamische Komposition - Zeichen und diakritische Marker kombinieren

+ platzsparend
+ flexibel
– unerlaubte und unmögliche Zeichen können erzeugt werden

Auflistung nach diakritischem Marker

+ platzsparend
+ Buchstaben ohne diakritische Marker in Dropdown Feldern
+ es gibt weniger relevante diakritische Marker als Buchstaben mit denen sie kombinieren
– Echte Sonderbuchstaben müssen separat abgehandelt werden

Auflistung nach Basisbuchstaben

+ platzsparend
– Diakritische Marker im Auswahlfeld - eher unleserlich
– echte Sonderbuchstaben müssen separat abgehandelt werden

Buchstaben - diakritische Marker Matrix

+ übersichtlich
– sperrig
– echte Sonderbuchstaben müssen separat abgehandelt werden

Auswahl nach Sprache

+ sehr zuverlässig
– setzt Kenntnis der Sprache voraus

Reihung nach Häufigkeit

+ schneller Zugriff
– Lernphase

Eingabe mit "dead keys"

+ intuitive Nutzungf
– Es können nicht alle Zeichen erzeugt werden

Zulässige Kombinationen

Diakrit. Marker	Kombinierbare Buchstaben
ˇ Caron/Hacek	AaCcDdEeIiLlNnOoRrSsTtUuZz
´ Acute	AaCcEeIiLlNnOoRrSsUuYyZ
` Grave	AaEeIiOoUu
ˆ Circumflex	AaCcEeGgHhIiJjOoSsUuWwYy
˜ Tilde	AaIiNnOoUu
Macron	AaEeIiOoUu
˘ Breve	AaGgIiOoUu
˙ Punkt	CcEeGgIiZz
¨ Diaresis	AaEeIiOoUuYy
Slash	LlOo
° Ring	AaUu
Cedilla	CcGgKkLlNnRrSsTt
Querstrich	DdHhTt
˝ Doppelacut	OoUu
Ogonek/Haken	AaEeIiUu
¸ Komma	SsTt

Fallstricke

Nicht alle diakritischen Marker lassen sich von Laien eindeutig erkennen: lokale Gewohnheiten und typographische Konventionen erschweren die Zuordnung oft beträchtlich. Hier einige Beispiele:

Groß	Klein	Kommentar
Ģ (U+0122)	ģ (U+0123)	Lettisch, Cedille unter G wird zu Akut auf g
Ť (U+0164)	ť (U+0165)	Tschechisch, Kroatisch, Hacek auf T wird zu Apostroph hinter t
Ď (U+010E)	ď (U+010F)	Tschechisch, Kroatisch, Hacek auf D wird zu Apostroph hinter d
Đ (U+00D0)	ð (U+00F0)	Isländisch, aus D mit Querstrich wird Sonderbuchstabe, Verwechslung mit Đ (U+0110)
Đ (U+0110)	đ (U+0111)	Rumänisch, Verwechslung mit Đ (U+00D0)
Ų (U+0172)	ų (U+0173)	Combining ogonek, zentriert unter dem großen U, rechts unter kleinem u

3.3.2 Empfehlung: Eingaben immer normalisieren

Alle Benutzereingaben sollten sofort nach Form C normalisiert werden, da die Quelle und Normalisierung der eingegebenen Daten immer unbekannt ist.

Ressourcen

- UniView, JavaScript-basierte Unicodetabelle
- Beispiel für eine JavaScript-basierte Zeichentabelle (siehe Kapitel 3.5.3)
- Echte Sonderzeichen/Sonderbuchstaben (siehe Kapitel 2.1.2)

3.3.3 Dateneingabe auf Personal Computern

Microsoft Windows

Mit dem Betriebssystem Windows (alle Versionen) laufende PCs, bieten eine Applikation (Zeichentabelle) zum Auswählen von Sonder- und diakritischen Zeichen. Sie ist, wenn installiert, im Menu Zubehör enthalten. Die Bildschirmtastatur (schnell erreichbar durch Drücken der Tasten Windows+U) ermöglicht die komfortable Eingabe von Zeichen, vor allem, wenn verschiedene Eingabegebietsschemata gleichzeitig benutzt werden sollen, da dann die Tastenbeschriftungen immer aktuell zum jeweils aktiven Tastaturlayout angezeigt werden. Weiters bietet die Microsoft Office Suite (alle Versionen) eine zusätzliche Applikation zur Zeichenauswahl.

Ressourcen

- Verwenden der Bildschirmtastatur unter Windows (siehe separates Paket Mustercode/Beilagen)
- Microsoft Windows XP - Die technische Referenz

Unix/Linux mit Gnome oder KDE und OpenOffice

Sowohl die gebräuchlichen Desktopumgebungen für Unix/Linux als auch das weit verbreitetete OpenOffice Paket bieten Tools zur Zeichenauswahl an. Das OpenOffice Tool ist nach den Unicodeblöcken gegliedert. Es ist fraglich, ob dies für Laien hilfreich ist.

Apple/Macintosh

Apple bietet eine sehr komfortable Zeichentabelle namens Zeichenpalette.

Ressourcen

- Siehe auch: Darstellung auf PCs (siehe Kapitel 3.6.3)
- Apple Unicode Keyboard Layouts

3.3.4 Datenübernahme von Bürgerkarten

Grundlagen

Daten werden in Bürgerkarten-Infoboxen, im UTF-8 Encoding, gespeichert. Die Bürgerkarte unterstützt grundsätzlich den gesamten Unicode Zeichenumfang (Details zu Formaten für die Anzeige der Bürgerkarten-Umgebung können der Spezifikation entnommen werden). In der Praxis sind dem Zeichenumfang allerdings Grenzen gesetzt. Die Personenbindung wird vom BMI aufgebracht und beschränkt sich gemäß den Empfehlungen in diesem Dokument, auf lateinische Zeichen, genauer gesagt, Buchstaben aus den Unicodeblöcken Basic Latin, Latin-1 Supplement, Latin Extended A, Latin Extended B, Latin Extended Additional. Zeichen liegen in Normalisierungsform C vor.

A-Trust Bürgerkarten

A-Trust schränkt die Gruppe der erlaubten Zeichen auf folgende ein:

Kleinbuchstaben: àáâãäåæą ćçčċĉ ďđð èéêëęě ğġĝ ħĥ ìıíîï ĵ ĺľ ńňñ òóôõőöø ŕř śšşŝß ţť þ ùúûüŭ ýÿ źżż

Grossbuchstaben: ÀÁÂÃÄÅÆĄ ĆÇČĊĈ ĎĐÐ ÈÉÊËĚĘ ĞĠĜ ĦĤ İIÌÍÎÏ Ĵ ŁĽ ŃŇÑ ÒÓÔÕŐÖØ ŔŘ ŚŠŞŜß ŢŤ Þ ÙÚÛÜŬÝ Ÿ ŹŽŻ

Es werden also keine Zeichen aus Latin Extended-B unterstützt. Trotz der Zusicherung, dass die oben erwähnte Einschränkung auf die technischen Randbedingungen des Kartenaufdrucks zurück geht, werden keine Sonderzeichen auf die Karte aufgebracht (außer Umlauten und ß).

A1 Signatur

Die A1 Signatur unterstützt den benötigten Zeichumfang. Derzeit liegen keine genaueren Informationen vor.

e-Card

Die e-Card unterstützt alle vom Hauptverband der Sozialversicherungsträger unterstützten Zeichen des WGL/4 Sets (siehe Kapitel 3.6) sowohl im Zertifikat als auch im Aufdruck

- Unicode range U+0020 through U+00BB
- Unicode range U+00BC through U+017E
- Unicode range U+017F through U+1EF3
- Unicode range U+2013 through U+FB02

Ressourcen

- Formate für die Anzeige der Bürgerkarten-Umgebung

3.4 Persistierung

Einleitung

Dieses Kapitel behandelt die Persistierung von diakritischen Zeichen in Datenbanken und LDAP Verzeichnissen. Weiters werden Algorithmen zur Vereinfachung von diakritischen Zeichenketten vorgestellt und *Best Practises* für die Umsetzung von sonderzeichentoleranten Suchen erläutert.

Im Prinzip kann davon ausgegangen werden, dass jede aktuelle Version der verbreiteten Datenbanken Unicode zumindest in dem für Österreich relevanten Ausmaß unterstützt. Ältere Systeme (Großrechner) lassen sich im Allgemeinen auch auf Unicode umrüsten, jedoch erfordert diese Art der Umstellung Know-how, das den Rahmen dieses Handbuchs sprengen dürfte.

Informationen über XML finden sich im Abschnitt Schnittstellen (siehe Kapitel 3.2.1)

3.4.1 Unicode Unterstützung gängiger Datenbanken

Großsysteme

DB2 Mit DB2 UDB for OS/390 und z/OS Version 7 können Daten in Unicode Tabellen gespeichert und Hostvariablen in Unicode verarbeitet werden. In DB2 UDB for z/OS ab Version 8 gibt es durchgängigen Unicode Support, d. h. der gesamte SQL-Statement String, einschließlich Objektnamen etc., kann in Unicode verarbeitet werden. JOINs zwischen EBCDIC, ASCII und Unicode Tabellen sind möglich. Über eine Erweiterung der CASE Funktion mit CCSID Angabe kann die SQL Verarbeitung beeinflusst werden (z. B. Sortierreihenfolgen ...). Der DB2 z/OS Unicode Support unterstützt hierbei UTF-8 und UTF-16. Die Zuordnung, ob UTF-8 oder UTF-16 verwendet wird, erfolgt über die DB2 Datentypen (CHAR, VARCHAR. CLOB –> UTF-8; GRAPHIC, VARGRAPHIC oder DBCLOB –> UTF-16).

DB2 CLI Unicode Funktionen akzeptieren neben ASCII Strings auch Unicode Strings. Unicode Strings müssen dazu in UCS-2 Codierung in plattformspezifischer Endianess vorliegen. Die ODBC API Funktionen haben Suffixe die die String Codierung wiedergeben: die unicodetauglichen enden auf *W*, die ANSI Varianten haben kein Suffix.

- DB2 UDB for z/OS Internationalization Guide
- Introduction to Unicode (part I)
- Introduction to Unicode (part II)
- DB2 Database Design: Using Unicode
- DB2 Database Development: Unicode CLI aplications - und folgende Seiten

- DB2 Database Development: JDBC and Unicode Streams

Oracle Oracle bietet eine umfassende Unicodeunterstützung, inklusive *supplementary characters*. Es stehen 4 Zeichensätze zur Verfügung, die zum UTF-8 korrespondieren:

- AL32UTF8: 1-3 Bytes, Surrogate Paare brauchen 4 Bytes. In Oracle Version 9i und höher. Wird auch Unicode 4.0 unterstützen.
- UTF8: 1-3 Bytes. In Oracle Version 8i und höher. UTF8 bleibt auf Unicode 3.0 stehen.
- UTFE: auf EBCDIC basierten Plattformen (UTF-EBCDIC äquivalent).
- AL24UTFFSS: obsolet ab Version 9i. Nur für Oracle Version 7.3 bis 8i. Unicode 1.1 Implementierung.

Oracle hat einen Zeichensatz, der zum UTF-16 korrespondiert:

- AL16UTF16: kann nicht als Datenbank-Zeichensatz, sondern nur als National Character Set verwendet werden. In Oracle Version 9i und höher. Wird auch Unicode 4.0 unterstützen. AL16UTF16 ist der Default National Datenbank Zeichensatz ab Version 9i und höher.

Die Wahl des Datenbank-Zeichensatzes ist vom Zeichensatz der Betriebsumgebung (Betriebssystem) völlig unabhängig. Für die Verarbeitung ist entscheidend, in welchem Zeichensatz der Client (PC oder Applikationsserver) arbeitet. Die Festlegung des Zeichensatzes am Client geschieht durch die Variable NLS_LANG. Verwendet der Client einen nicht Unicode fähigen Zeichensatz, muss man Konversionsproblemen mit Übersetzungstabellen beikommen. Daher ist dringend zu empfehlen (siehe Kapitel 3.4.2), Clients mit kompatiblen Zeichensätzen zu verwenden, um diese Schwierigkeiten zu vermeiden.

Oracle Datentypen und Zeichensätze:

- CHAR, VARCHAR2, BLOB, CLOB, LONG sind Single-Byte und Multi-Byte (Unicode und Nicht-Unicode Zeichensätze) fähig
- NCHAR, NVARCHAR2 und NCLOB verwenden immer den National Database Character Set und sind nur Multi-Byte Unicode fähig

Bei Unicode basierten Datenbanken können auch alle Bezeichner Unicode Zeichen enthalten. Oracle bietet grundsätzlich die Möglichkeit, die gesamte Datenbank oder nur einzelne Spalten mit Unicode Support anzulegen.

Beispiel

```
CREATE DATABASE sample
...
CHARACTER SET AL32UTF8
NATIONAL CHARACTER SET AL16UTF16
  ...
```

Des weiteren steht ein Toolkit für den *Globalization Support* (ehem. *National Language Support (NLS)*) und eine sehr ausführliche Dokumentation (siehe Resourcen) zur Verfügung.

- Globalization Support Guide
- Unicode Support diverser Oracle Versionen
- Unicode Datenbanken oder Unicode Felder?

Mittelgroße Datenbank Systeme

Firebird/Informix Firebird unterstützt Unicode zumindest seit Version 1.5.

Informix Informix bietet umfassende Unicode Unterstützung über das Unicode Blade Module.

- Unicode DataBlade Module User's Guide, Version 1.0

Ingres Ingres unterstützt Unicode seit Version 2.6. Aktuelle Versionen bieten erweiterte Unterstützung für Unicode. Ingres benutzt UTF-16 intern und im Frontend und setzt die Normalisierungsform D ein. Darüber hinaus kommt dieses System mit weitreichenden Möglichkeiten zur Suche und einer (automatischen) Zeichensatzkonvertierung.

- Unicode in Ingres

MaxDB (früher SAPDB) MaxDB unterstützt Unicode und speichert Daten nach eigener Ausage intern in UTF-16/UCS-2, was darauf schließen lässt, dass derzeit nur Unicode 3 unterstützt wird.

- MaxDB und Unicode
- Unicode und ODBC

Microsoft SQL Server Der Microsoft SQL Server unterstützt Unicode auf Feldbasis durch die Datentypen NCHAR, NVARCHAR sowie NTEXT und verwendet UCS-2 als Encoding Schema. String Funktionen sind Unicode sicher. Weiters bietet der Microsoft SQL Server mehrere Möglichkeiten für Unicode Collations (Sortierfolgen). Es ist zu beachten, dass Microsoft generell UTF-16 als Encoding Schema auf Windows 2000 und höher gewählt hat.

```
CREATE TABLE [dbo].[unitest] (
[ID] [int] IDENTITY (1, 1) NOT NULL ,
[firstname] [nvarchar] (100) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
[lastname] [nvarchar] (100) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
) ON [PRIMARY]
GO
```

- Using Unicode Data
- Using Collations

MySQL Mysql unterstützt Unicode (3) ab Version 4.1 durch UCS-2 und UTF-8. Darüber hinaus unterstützt MYSQL für UCS und UTF-8 mehrere Collation Algorithmen, unter anderem auch den Unicode Collation algorithmen utf8_unicode_ci, der sich besonders für Deutsch eignet, weil er auch die Erweiterung ß -> ss beherrscht. Umlaute werden wie das Basiszeichen sortiert.

Die Funktionen REGEXP / RLIKE sind nicht Multi-Byte sicher.

```
create table unitest (
   ID int primary key not null auto_increment,
   firstname varchar(100),
   lastname varchar(100)
)
character set utf8;
```

- MySQL Unicode support
- Unicode charactersets and Collation

PostgreSQL Postgres unterstützt Unicode seit Version 7.1.0, wobei Unicode-Support in diesem Fall für UTF-8 Unterstützung steht. Multi-Byte-Support kann bei der Konfiguration vorgenommen (./configure –enable-multibyte[=UNICODE]), oder per Datenbank festgelegt werden (createdb -E UNICODE test). Multi-Byte-Support muss beim Build aktiviert sein, um Unicode nutzen zu können. Das Anlegen einer Datenbank kann auch über ein SQL Kommando erfolgen:

```
CREATE DATABASE test WITH
  ENCODING = 'UNICODE';
```

Postgres bietet außerdem automatische Zeichenkonvertierung zwischen Client und Server. Die Pattern-matching Funktionen sind Multi-Byte sicher.

- Character Sets
- String Funktionen
- Pattern matching Functions

Sybase Sybase unterstützt Unicode seit System 11 SQL Server durch UTF-8 Codierung. Konsequenterweise wird nun seit der Version 12.5 des Adaptive Server Enterprise UTF-16 Encoding unterstützt.

Minimal Systeme/Desktop Datenbanken

MS Access MS Access unterstützt Unicode in UTF-16 Codierung seit der Version 2000.

SQLite Ab Version 3 wird Text in SQLite im UTF-8, UTF-16BE oder UTF-16LE Format gepeichert. Das Format wird bei der Anlage der der Datenbank ausgewählt.

3.4.2 Empfehlung: Datenbankclients mit kompatiblen Zeichensätzen verwenden

Bei Anwendungen mit Anbindung zu Datenbanken wird empfohlen, diese Clients mit zur Datenbank kompatiblen Zeichensätzen zu verwenden, um Schwierigkeiten bei auf Unicode befähigten Datenbanken zu vermeiden (Konversionsprobleme), bzw. die Notwendigkeit der Verwendung von Übersetzungstabellenzu zu verhindern.

3.4.3 Unicode Unterstützung in LDAP

Status: Unterstützt

LDAP v3 unterstützt Unicode in der UTF-8 Codierung. Beim Einsatz von LDIF sollten Zeichen außerhalb des ASCII Bereiches in UTF-8 Base64 codiert vorliegen.

Ressourcen

- RFC 2253 - Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names
- Beispiele für die LDIF Codierung

3.4.4 Best Practices

Duale Datenfelder

Zur Vereinfachung der Suche sollten indizierte Felder mit Sonderzeichen doppelt geführt werden: einmal mit Sonderzeichen und einmal ohne, sofern eine solche Funktionalität nicht von der Datenbank selbst zur Verfügung gestellt wird.

Regeln für die Transkription Die Vereinfachungen können auch der Tabelle der zu unterstützenden Zeichen (siehe Kapitel 3.1) entnommen werden.

Zeichen	Transkription
À, Á, Â, Ã, Å, Ò, Ó, ...	A, O, ...
à, á, â, ã, ò, ó, ...	a, o, ...
Ä	AE
ä	ae
Ö	OE
ö	oe
Ü	UE
ü	ue
ß	ss
Æ	AE
æ	ae
Œ	OE
œ	oe
Þ	TH
þ	th
Ŋ	N
ŋ	n
Đ	D
ð	d

3.4.5 Fallback für ältere Umgebungen

Entwicklungsumgebungen

Während die generelle Unicode Unterstützung namhafter Datenbanken als sehr gut zu bezeichnen ist, hinken die zugehörigen Entwicklungsumgebungen stellen-

weise hinterher. Insbesondere erkennt man einen Paradigmenwechsel weg vom Desktop-Client hin zum Webinterface. In den meisten Fällen lassen sich Probleme durch das Verwenden von Webinterfaces beheben. Ist diese Möglichkeit nicht gegeben, kann man auf Escapesequenzen (&#xuuu; – oder benutzerfreundlicher – HTML Entities) zurückgreifen, falls selbige Applikationen überwiegend von geschultem Personal genutzt werden. Ein Mapping für die Escapesequenzen kann durch stored procedures erfolgen.

Ressourcen

- HTML Entities

3.5 Programmatische Behandlung

Einleitung

Mittlerweile sind fast alle gängigen Programmiersprachen unicodefähig. Insbesondere Sprachen, die vorrangig zur Webprogrammierung oder XML Programmierung herangezogen werden, bieten eine gute Unicodeunterstützung.

Allgemeine Ressourcen

- Make your Programs Unicode aware - Information mainly for Linux/Unix systems

Sprachspezifische Ressourcen finden sich bei der jeweiligen Sprache.

3.5.1 JAVA

Status: Unterstützt

Allgemeine Informationen JAVA unterstützt Unicode und speichert Strings intern in UTF-16 bzw. UTF-8 Codierung. Zeichenbehandlung in J2SE 5 basiert auf Version 4.0 des Unicode Standards, J2SE 1.4 benutzt Version 3.0 und J2SE 1.3 verwendet Version 2.1 des Unicode Standards. Zum Einlesen und Ausgeben von Unicode Text in/aus Dateien, empfiehlt es sich, die -Reader und -Writer Methoden anstatt der InputStream und OutputStream Methoden zu verwenden (siehe Kapitel 3.5.2). Allerdings geht JAVA davon aus, dass das plattformspezifische Default-Codierungsformat vorliegt. Da dieses Format häufig ISO-8859-1 ist, wird beispielsweise UTF-8 Input beim Einlesen ruiniert. Aus diesem Grunde sollte immer das Quell- oder Zielencoding aufgeführt werden. Die XML Reader wählen das richtige Encoding automatisch.

```
BufferedReader rdr =
    new BufferedReader(
       new InputStreamReader(new FileInputStream("infile.txt"),
                             "UTF-8"));
String line = rdr.readLine();
```

JAVA unterstützt Unicode auch im Sourcecode. Alle Unicodezeichen können als Escape-Sequenzen angegeben werden (in der Form \unnnn).

Mit Hilfe eines Unicode fähigen Editors können diakritische Zeichen aber auch direkt eingegeben werden. Dann muss beim Compiler-Aufruf allerdings das Encoding angegeben werden, da JAVA immer die plattformspezifische Defaultcodierug (meist ISO-8859-1) erwartet. Seit Version 1.4 benutzt JAVA die ISO Namen für die diversen Encodings. Bei älteren Versionen weichen die Namen teilweise stark ab.

```
javac -encoding UTF-8 HelloWorld.java
```

Bei der Ausgabe ist darauf zu achten, dass das Ausgabemedium Unicode-tauglich ist (Kommandozeilen sind das häufig nicht). Swing GUIs sind Unicode-fähig, bei JSP/Webappliaktionen hängt die korrekte Ausgabe von der richtigen HTTP-Header Information und den Fähigkeiten des Browsers ab.

JAVA Servlets Bevor ein Objekt geholt wird (Stream oder Writer), sollte der Content Type entsprechend gesetzt werden:

```
resource.setContentType ("text/html;charset=utf-8");
```

Beim Einsatz eines Writers, wird die Umwandlung des Encodings von JAVA automatisch erledigt.

JSP Für JSP sollte die passende Codierung im page Aufruf gewählt werden:

```
<%@ page contentType="text/html; charset=UTF-8" %>
```

Jeglicher Output wird danach automatisch in das gewählte Encoding konvertiert.

Ressourcen

- sun: JAVA Tutorial: Internationalization
- sun: JAVA Tutorial: Working with Text
- sun: Internationalization FAQ
- Unicode in JAVA
- IBMs extensive Unicode library für JAVA

3.5.2 Empfehlung: JAVA StreamReader und -Writer Methoden verwenden

Zum Einlesen und bei der Ausgabe von Unicode Text in/aus Dateien bzw. allgemeinen Datenströmen, wird empfohlen, die zugehörigen -Reader und -Writer Methoden anstatt der InputStream und OutputStream Methoden zu verwenden.

3.5.3 .NET: VB, C#

Status: unterstützt

ASP und ASP.NET setzen die Codierung unabhängig vom Dokumenttyp über die charset-Methode des response-Objekts. Diese Methode muss gleich zu Beginn aufgerufen werden, weil die HTTP-Header Information nach dem Senden der ersten Daten nicht mehr manipuliert werden kann. Um die Codierung zu ändern, benutzt man

```
<%response.Charset="utf-8"%>
```

In ASP.Net können Content-Type und Encoding auch gemeinsam über Response.ContentEncoding gesetzt werden. Darüber hinaus gibt es die Möglichkeit, die Defaultcodierung über die Dateien Web.config oder Machine.config zu ändern (defaultmäßig ist bereits UTF-8 eingestellt).

Beispiel

Das folgende Beispiel (siehe separates Paket Mustercode/Beilagen) illustriert den Zugriff auf eine MSSQL Datenbank mittels ASP.Net.

```
<%@ Language=VBScript %>
<html>
  <head>
   <meta name="GENERATOR" Content="Microsoft Visual Studio .NET 7.1">
  </head>
  <body>
    <h2>Unicode Mini Test Page</h2>
    <form action="in.asp" method="get">
      <table>
<tr>
  <td>Vorname</td>
  <td><input type="text" size="30" name="fname" /></td>
</tr>
<tr>
      <td>Nachname</td>
  <td><input type="text" size="30" name="lname" /></td>
</tr>
<tr>
```

```
  <td colspan="2" align="center">
            <input type="submit" value="Hinzuf&uuml;gen" />
          </td>
</tr>
      </table>
    </form>
    <h2>Inhalt der Datenbank</h2>
    <table border="1">
      <tr>
     <th>Vorname</th><th>Nachname</th>
      </tr>

      <%
provStr = "Provider=sqloledb;Server=XPEN;Database=UnicodeTest; & _
                 User ID=sa;Password=''"

if Request.item("fname") <> ""  and Request.item("lname") <> "" then
Set conn = CreateObject("ADODB.Connection")
conn.Open provStr
strSQL = "INSERT INTO unitest (firstName, lastName) VALUES" & _
               "('" & Request.item("fname") & "', '" & _
               Request.item("lname") & "')"
conn.Execute strSQL
conn.close
Set conn = nothing
end if

Set rs = CreateObject("ADODB.Recordset")
strSQL = "SELECT * FROM unitest ORDER BY lastname"
rs.open strSQL, provStr
while not rs.EOF
Response.Write("<tr><td>"  & rs("firstName"))
Response.Write("</td><td>"  & rs("lastName") & "</td></tr>")
rs.MoveNext
wend
rs.close
Set rs = Nothing

%>
    </table>

    <h2>Information</h2>
    <p>Dieses Programm setzt eine MS SQLServer DB voraus.
```

```
       Die Datenbank kann mit folgendem Script
       erzeugt werden :</p>

    <pre>
CREATE TABLE [dbo].[unitest] (
[ID] [int] IDENTITY (1, 1) NOT NULL ,
[firstname] [nvarchar] (100) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
[lastname] [nvarchar] (100) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
) ON [PRIMARY]
GO
    </pre>
   <p>Der SQLServer in unserem Beispiel heißt XPEN. Auf ihn wird mit dem
    User <i>sa</i> ohne Passwort zugegriffen.</p>
  </body>
</html>
```

Ressourcen

- ASP.Net Unicode

3.5.4 JavaScript

Status: Unterstützt

Seit Version 1.3 sind JavaScript-Strings immer in Unicode. Es gibt keinen Character-Typ, aber einzelne Unicode-Zeichen können als Escape Sequenzen eingeschleust werden (der Form \unnnn).

JavaScript normalisiert intern NICHT - es erwartet gemäß W3C Normalisierungsvorgaben die Normalisierungsvariante C (siehe Kapitel 1.2.3) (Canonical Decomposition + Canonical Composition).

Beispiel

Ein einfacher Sonderzeichen Browser (Komplett sichtbar nur in der Online-Version dieses Handbuchs)

```
function cappend(c){
```

```
 document.forms[0].unitest.value = document.forms[0].unitest.value + c;
}
```

```
Klicken Sie auf ein Sonderzeichen um es einzufügen:
Zeichenauswahl:
  ģ
  ā
  ą
  đ

   Geben Sie hier Sonderzeichen ein:
```

Ressourcen

- ECMAscript Specification

3.5.5 PHP

Status: Unterstützt

PHP unterstützt UTF-8 seit Version 3.06. Es können UTF-8 Zeichen bis zu 4 Bit Breite verarbeitet werden. Durch die Multibyte String Extension unterstützt PHP auch UTF-16 und UTF_32 in LE und BE Ausprägung mit gängigen String-operationen wie mb_ereg().

3.5.6 Empfehlung: PHP Multi-Byte String Operationen benutzen

Beim Einsatz von UTF-8 oder anderen Unicode encodings müssen die Multibyte Varianten der String Funktionen zum Einsatz kommen.

3.5.7 Empfehlung: PHP: HTTP-Header für content-type benutzen

Bevor man Teile der HTML Seite schickt, müssen die korrekten HTTP-Header gesendet werden. PHP bietet dafür die Funktion header(). Wird das Encoding

nicht mit der Funktion header() auf UTF-8 gesetzt, stellt der Webserver defaultmäßig ISO_8859-1 ein.

```
header("Content-type: text/html;
       charset=utf-8");
```

Dieser Header muss an oberster Stelle in der PHP Datei stehen.

```
<?
header("Content-type: text/html; charset=utf-8");
?>
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=UTF-8">
<title>PHP unicode database test</title>
</head>
<body>
....
```

Beispiel

Download unitest.php (siehe separates Paket Mustercode/Beilagen). Weitere Information: Unicode und MYSQL (siehe Kapitel 3.4), HTML Formulare und Unicode (siehe Kapitel 3.2.4)

```
<?
// Sample program to show UTF-8 information handling using a mysql database
// see end of programm for information on database structure

// important notice: this header MUST go first!
// Otherwise the encoding will be changed to ISO_8859-1
header("Content-Type: text/html; charset=UTF-8");
?>
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=UTF-8">
<title>PHP unicode database test</title>
</head>
<body>

<h2>Information</h2>
```

```
<p>Dieses Programm setzt folgende mysql Datenbank voraus:</p>

<pre>
create database unitest;
use database unitest;
create table unitest (
  ID int primary key not null auto_increment,
  firstname varchar(100),
  lastname varchar(100)
)
type=InnoDB character set utf8;
</pre>

<h2>Personen Datenbank</h2>
<form action="unitest.php" method="get" accept-charset="UTF-8">
   Vorname: <input type="text" name="firstname" size="40" /><br>
   Nachname: <input type="text" name="lastname" size="40" /><br>
   <input type="submit" name="submit" value="Hinzufuegen" /><br>
</form>

<?
// adapt these for your setup
$host ="127.0.0.1";
$user = "root";
$pw = "";
$db = "unitest";
$table = "unitest";

// connect to db server and select database
$server = mysql_pconnect($host, $user, $pw);
mysql_select_db($db);


// get variables - replace register_globals=on
foreach ($_GET as $k=>$v){
  $tmp=$k;
  $$tmp = $v;
}

// if new data was submitted add it to database
if ($firstname && $lastname) {
  $query = "insert into $table (firstname, lastname)";
  $query.= "values ('$firstname','$lastname')";
```

```
  @mysql_query($query) or die("failed: ".mysql_error());
 }

// show all entries
echo "<h2>Personen</h2>";
$query = "select * from $table";
$persons = mysql_query($query);
while ($person = mysql_fetch_row($persons)){
    foreach ($person as $p) echo "$p ";
echo "<br>";
}

// close server connection
mysql_close($server);
?>

</body>
</html>
```

Ressourcen

- Multibyte String Extension

3.5.8 Perl

Status: Unterstützt

Ab Version 5.6 speichert Perl strings intern im UTF-8 Format. Andere Codierungen können über die iconv library angebunden werden.

Bevor man Teile der HTML Seite schickt, müssen die korrekten HTTP-Header gesendet werden. Nach dem letzten Header einen doppelten Zeilenumbruch einfügen.

```
print "Content-Type: text/html; charset=utf-8\n\n";
```

Ressourcen

- Perl, Unicode and i18N FAQ
- iconv lib download from CPAN

3.5.9 Python

Status: Unterstützt

Python unterstützt Unicode ab Version 1.6. Es existieren zwei unterschiedliche String Typen: str (8-Bit Non-Unicode) und unicode (16-Bit Unicode String)

```
question = u'\u00bfHabla espa\u00f1ol?'  # ¿Habla español?
```

Python String Libraries sind Unicode sicher, das includiert auch die gettext Funktionen und Regular Expressions.

Bevor man Teile der HTML Seite schickt, müssen die korrekten HTTP-Header gesendet werden. Nach dem letzten Header einen doppelten Zeilenumbruch einfügen.

```
print "Content-Type: text/html; charset=utf-8\n\n"
```

Ressourcen

- Python und Unicode
- Dive into Python - Unicode

3.5.10 C/C++

Status: Unterstützt

C

Bedingt durch den Mangel an Strings in C, wird Unicode nur durch den wchar_t* Zeiger unterstützt. Dementsprechend gibt es Funktionen wprintf, wcslen, und wcslcat, die diese Unicode *Strings* bearbeiten. Diese wchar_t Zeichen sind allerdings 4 Bytes (32-Bit) lange Zeichen, was sich extrem negativ auf den Speicherbedarf von europäischen Zeichen auswirkt, dafür aber den Vorteil bietet, jedem Zeichen eine eindeutige Nummer zuordnen zu können (vermeidet Surrogate Paare in UTF-16).

In UTF-8 codierte Strings werden als *wide character strings* bezeichnet. Verwendet man wide character strings in Programmen auf Win32 Clients, muss bei der Kommunikation oft eine Konversion von UTF-8 in UCS-2 stattfinden, wofür das Flag CP_UTF8 für die WinAPI Konversionsfunktionen zu setzen ist:

- MultiByteToWideChar: Konversion von ANSI UTF-8 zu UCS-2
- WideCharToMultiByte: Konversion von UCS-2 zu ANSI UTF-8

Durch den Einsatz der Local-Library lässt sich auch der Output in UTF-8 oder UTF-16 erzwingen, die Unterstützung für UTF-32 ist allerdings besser.

Die libiconv bietet Unicode Support für Systeme, die von Grund auf keine Unicodeunterstützung bieten und ermöglicht das Konvertieren zwischen verschiedenen Codierungen.

C++

Für C++ gilt das für C Gesagte. Weiters bietet C++ noch ein bequemeres Unicode-String Interface über std::wstring

Ressourcen

- libiconv
- C and Unicode
- Microsoft: Unicode C/C++ Programming
- Unicode-enabling Microsoft C/C++ Source Code
- IBM ICU library for C

3.5.11 Unicode Codierung erkennen

BOM - Byte Order Mark

Da UTF-16 und UTF-32 Codierungen immer in einer Big Endian (BE) oder einer Little Endian (LE) Variante vorliegen, tragen sie im Allgemeinen ein sog. BOM (Byte Order Mark), um die Endianess auszudrücken.

Bytes	Codierung
00 00 FE FF	UTF-32, big-endian
FF FE 00 00	UTF-32, little-endian
FE FF	UTF-16, big-endian
FF FE	UTF-16, little-endian
EF BB BF	UTF-8
DD 73 66 73	UTF-EBCDIC
2B 2F 76 u. eine der folg. Sequ. [38 | 39 | 2B | 2F | 38 2D]	UTF-7

Anwendung von Byte Order Marks Byte Order Marks finden meist Verwendung am Beginn von Dateien oder von Datenstreams und agieren so als Signatur, zur Erkennung der jeweiligen Codierungsform. Treten sie inmitten eines Textes auf, so können diese Zeichen als ZERO WIDTH NON-BREAKING

SPACE (ZWNBSP), als Teil des Inhaltes eines Strings bzw. des Dateiinhaltes, behandelt werden. In diesen Fällen ist die Verwendung des Zeichens U+2060 (WORD JOINER) über ZWNBSP vorzusehen, damit nicht versehentlich eine BOM interpretiert wird. Byte Order Marks können aber auch als eigentliche BOM interpretiert werden und die Codierung umschalten, oder gänzlich ignoriert werden bzw. einen Fehler auslösen, wenn eine BOM bereits übergreifend vorgegeben wurde.

UTF-8

Der strukturierte Aufbau des *variable length encodings* UTF-8 macht es möglich, diese Codierung auch ohne Zuhilfenahme eines MIME-types oder eines encoding Attributs zu erkennen.

Methode Der der UTF-8 Codierung zu Grunde liegende Algorithmus erzeugt ein gleichmäßiges Bitmuster. Es wird immer die kürzestmögliche Multi-Byte Sequenz benutzt.

Aufgrund des charakteristischen Musters der High-Bits kann man sehr rasch auf die UTF-8 Codierung schließen und stellenweise sogar den Sprachraum eines Textes bestimmen.

Unicode Zeichen	Bit-Muster
0x00000000 - 0x0000007F:	0xxxxxxx
0x00000080 - 0x000007FF:	110xxxxx 10xxxxxx
0x00000800 - 0x0000FFFF:	1110xxxx 10xxxxxx 10xxxxxx
0x00010000 - 0x001FFFFF:	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
0x00200000 - 0x03FFFFFF:	111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
0x04000000 - 0x7FFFFFFF:	1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx

Beispiel 1: speichere A mit einem Circumflex in die Datenbank

```
  # A mit einem Dach Â = U+00C2 - liegt im 11 Bit Bereich, d.h.
  # Ergebnis muss sein: 110nnnnn 10mmmmmm
  # C2 = x00C2
0000 0000 1100 0010  # binärer Wert von C2 in 2 Bytes
```

```
  # Regel: gib die 6 rechten Bits nach mmmmmm und die nächsten 5 Bits
  #        nach nnnnn
  # UTF-8 Ergebnis:
1100 0011 1000 0010
  # dies entspricht: C3 82 (hex) = 50050 (dez); daher
  # Verwende CHR(50050) um Â in die Datenbank abzuspeichern!
```

Beispiel 2: speichere das Euro-Symbol in die Datenbank

```
  # € = U+20AC - liegt im 16 Bit Bereich, d.h.
  # Ergebnis muss ein 3 Byte-Wert sein 1110nnnn 10mmmmmm 10pppppp
  # 0x20AC:
0010 0000 1010 1100 # binärer Wert von 20AC in 2 Bytes
 # Regel: gib die 6 rechtesten Bits nach pppppp, die nächsten 6 rechten
  #        Bits nach mmmmmm und die nächsten 4 Bits nach nnnn
  # UTF-8 Ergebnis:
1110 0010 1000 0010 1010 1100
  # dies entspricht: E2 82 AC(hex) = 14 844 588(dez); daher
  # Verwende CHR(14844588) um € in die Datenbank abzuspeichern!
```

Speichert man somit von einem Windows PC aus, Daten in eine Datenbank, wird diese Konversion durchgeführt. Dies ergibt für die wichtigsten Sonderzeichen:

Unicode	Dezimalwert	Zeichen	HexWert in DB	DezWert in DB
U+00A7	167	§	C2 A7	49831
U+00C4	196	Ä	C3 84	50052
U+00D6	214	Ö	C3 96	50070
U+00DC	220	Ü	C3 9C	50076
U+00DF	223	ß	C3 9F	50079
U+00E4	228	Ä	C3 A4	50084
U+00F6	246	Ö	C3 B6	50102
U+00FC	252	Ü	C3 BC	50108
U+20AC	8364	€	E2 82 AC	14844588

UTF/UCS Converter Ein UTF/UCS Converter ist zu finden unter http://www.macchiato.com/unicode/convert.html - Verwendung:
· Input: UTF-16M auswählen (entspricht UCS-2); alternativ geht auch „Text“

· Output: UTF-8 auswählen
· Im Inputfenster den Unicode eingeben (zB 20AC für das €-Zeichen) oder (bei „Text“ direkt das € Zeichen)
· Als Ergebnis bekommt man im Outputfenster E2 82 AC
Dies bedeutet für das Einbringen von Sonderzeichen beispielsweise: statt des bisherigen CHR(220) für ein großes „Ü“ - CHR(50076) zu verwenden.

Ressourcen

- unicode.org: Byte Order Mark (BOM) FAQ
- UTF-8 Erkennung mit Regular Expressions
- Weitere Erkennungs- und Konvertierungsalgorithmen (in PHP, ist aber leicht auf andere Sprachen zu übertragen)

3.5.12 Signatur

Signatur und Kanonisierung

Die XMLDSig erfordert vor der eigentlichen Signatur die Kanonisierung des zu signierenden Dokuments. Dieser Kanonisierungsprozeß inkludiert auch eine Normalisierung auf die Form C, die die vom W3C und diesem Handbuch Normalisierungsform empfohlene (siehe Kapitel **??**) ist.

Ressourcen

- Canonical XML

3.6 Ausgabe

3.6.1 Unicode Fonts

Dezidierte Unicode Fonts

Echte Unicode Fonts gibt es mittlerweile für jede Plattform, obwohl keine dieser Fonts auch nur annähernd in der Lage ist, den Unicode Standard komplett mit Glyphen zu versehen. Alle im weiteren aufgelisteten Fonts verfügen über komplette Unterstützung der lateinischen Schrift.

Font	Glyphenzahl	Quelle
Code2000/Code2001	62,981 + BP1 Chars	Shareware/Freeware
MS Arial Unicode	51,180 (Unicode 2)	MS Office 2000+
Bitstream CyberBit	29,934	Freeware
Lucida Sans	2,929	JAVA/OpenOffice

Unicodeunterstützung bei Standard-Fonts + WGL4

Die mit Microsoft Systemen ausgelieferten Fonts richten sich nach dem WGL4 (Windows Glyph List 4) von Microsoft und Adobe. Der WGL4 Standard deckt die Windows Codepages 1250 (Eastern Europe), 1251 (Cyrillic), 1252 (US English = ANSI), 1253 (Greek) und 1254 (Turkish) ab und benutzt die Unicode Nummern für die Zeichen. Diese Codepages entsprechen den Unicode Blöcken Basic Latin, Latin-1, Latin Extended A und einem Bruchteil von Latin Extended B.

Die Zeichenzahl bei den mit Windows ausgelieferten System Fonts schwankt etwas, aber die mit Windows ausgelieferten Fonts Arial, Times New Roman, Tahoma, Verdana, ... unterstützen im Allgemeinen die folgenden Unicode Bereiche. Dadurch lassen sich alle europäischen Sprachen mit Ausnahme einiger Zeichen des Rumänischen (und der serbokroatischen Digraphen) schreiben.

- Basic Latin (alph. Zeichen komplett): U+0020 bis U+00BB
- Latin-1 Supplement (alph. Zeichen komplett) U+00BC bis U+00FF
- Latin Extended A (komplett): U+0100 bis U+017F
- Latin Extended B (spärlich) U+0192 und U+01FA bis U+01FF
- Combining Diacritic Marks (spärlich): U+02c7, U+02c9 und U+02d8 bis U+02dd

WGL4 Unterstützung für Latin Extended B (rudimentär)

Z	Nr.	Name, Bedeutung
f	U+0192	florin, latin small letter script f, florin sign
Ǻ	U+01FA	Aringacute, latin capital letter a with ring above and acute
ǻ	U+01FB	aringacute, latin small letter a with ring above and acute
Æ	U+01FC	AEacute, latin capital ligature ae with acute
æ	U+01FD	aeacute, latin small ligature ae with acute
Ǿ	U+01FE	Oslashacute, latin capital letter o with stroke and acute
ø	U+01FF	oslashacute, latin small letter o with stroke and acute
ˆ	U+02C6	circumflex, nonspacing circumflex accent

WGL4 Unterstützung für Combining Diacritic Marks (rudimentär)

Z	Nr.	Name, Bedeutung
ˇ	U+02C7	caron, modifier letter hacek
	U+02C9	macron, modifier letter macron
˘	U+02D8	breve
˙	U+02D9	dotaccent, dot above
° ;	U+02DA	ring, ring above
	U+02DB	ogonek
˜	U+02DC	tilde, nonspacing tilde
˝	U+02DD	hungarumlaut, modifier letter double prime

Fehlende Zeichen in WGL

Die folgenden Zeichen, die in europäischen Sprachen benutzt werden, sind in WGL 4 nicht enthalten. Dabei handelt es sich hauptsächlich um Ligaturen, die zwar in der jeweiligen Sprache als ein Buchstabe zählen, aber trotzdem als zwei Buchstaben geschrieben werden können. Ebenso wie die Rumänischen S und T mit Komma unter dem Zeichen (nicht zu verwechseln mit einer Cedille!), haben diese Buchstaben, mangels Umsetzung im ISO-8859-x Standard, keine Computer Tradition. Es ist aber zu erwarten, dass der Gebrauch mit dem wachsenden Einsatz von Unicode zunimmt.

Z	Nr.	Name
Ş	U+015E	Capital S with Comma beneath
ş	U+015F	Small s with Komma beneath
Ţ	U+021A	Capital T with Comma beneath
ţ	U+021B	Small t with Komma beneath
IJ	U+0132	Capital ligature IJ
	U+01C7	Small ligature ij
	U+01C4	Capital ligature DZ caron
	U+01C5	Ligature Capital Cz caron
	U+01C6	Small ligature dz caron
	U+01C7	Capital ligature LJ
	U+01C8	Ligature Lj
	U+01C9	Small ligature lj
	U+01CA	Capital ligature NJ
	U+01CB	Ligature Nj
	U+01CC	Small ligature nj
	U+01F1	Capital ligature DZ
	U+01F2	Ligature Dz
	U+01F3	Small ligature dz

Testseiten

- Lokale Testseite für Latin Characters (siehe separates Paket Mustercode/-Beilagen)
- Thesaurus Indogermanischer Text- und Sprachmaterialien Unicodetestseite
- W3C Teseite

Unicodeunterstützung je Unicode Block im Detail

Die folgende Übersicht stammt von Alan Wood.

Unterstützung für Latin Extended A

Windows/Unix Fonts Aboriginal Sans, Aboriginal Serif, AiPaiNutaaq (few), Akaash, Albany, Alkaios, ALPHABETUM Unicode, Andale Mono, Andale Sans, Andale Sans UI, Arabic Typesetting, Arial, Arial Black, Arial Narrow, Arial Unicode MS, Baekmuk Batang, Baekmuk Dotum, Baekmuk Gulim, Baekmuk Headline, Ballymun RO, Batang, BatangChe, Berling Antiqua, Bitstream Cyber-Base, Bitstream CyberBit, Bitstream CyberCJK, Book Antiqua, Bookman Old Style, Cardo, Caslon, Century, Century Gothic, Century Schoolbook, Chrysanthi Unicode, CN-Arial, CN-Times, Code2000, Comic Sans MS, Courier MonoThai,

Courier New, Cumberland, Doulos SIL, ESL Gothic Unicode, Everson Mono Unicode, Fixedsys Excelsior, Franklin Gothic, Franklin Gothic Cond, Free Monospaced, Free Sans, Free Serif, Frutiger Linotype, Gandhari Unicode, Garamond, Gentium, GentiumAlt, Georgia, Gulim Che, Gungsuh, GungsuhChe, Haettenschweiler, Hindsight Unicode, HY Shin Myeongjo Std Acro, Impact, jGaramond, Junicode, Khmer OS, Kidprint, Kozuka Mincho Pro Acro, Legendum, Linux Libertine, Lucida Bright, Lucida Console, Lucida Sans, Lucida Sans Typewriter, Lucida Sans Unicode, MD King KhammuRabi, MgOldTimes UC Pol, Microsoft Sans Serif, Ming(for ISO10646), Mistral, Monospace, Monotype Corsiva, MS Gothic, MS Mincho, MS PGothic, MS PMincho, MS Reference Sans Serif, MS Reference Serif, MS UI Gothic, MSung Std Acro, Naqsh, New Athena Unicode, New Gulim, NSimSun (few), NSimSun-18030 (few), Palatino Linotype, Sanskrit 2003, SImPL, SimSun (few), SimSun-18030 (few), Sylfaen, Tahoma, Thorndale, Thryomanes, Tibetan Machine Uni, Times New Roman, TITUS Cyberbit Basic, Trebuchet MS, TSC JSong S TT, Uqammaq (few), UVN Gia Dinh, UVN Giay Trang, Verdana, Vusillus Old Face

Mac Fonts AiPaiNutaaq (few), Apple Gothic (Central European), Arial, Century, Courier, Didot, Futura, Geneva, Gentium, GentiumAlt, #GothicMedium (Central European), #GungSeo (Central European), Hangang (Central European), Helvetica, Hiragino Kaku Gothic Pro, Hiragino Kaku Gothic Std (few), Hiragino Maru Gothic Pro, Hiragino Mincho Pro, Junicode, Lucida Grande, Monaco, MS Gothic, MS Mincho, MS PGothic, MS PMincho, #MyungjoNeue (Central European), #PCMyungjo (Central European), #PilGi (Central European), Seoul (Central European), SimSun (few), Skia, STFangsong, STHeiti, STKaiti, STSong, TektonPro, Times, Times New Roman, Trebuchet MS, Verdana, Zapfino

Unterstützung für Latin Extended B

Windows/Unix Fonts Windows: Aboriginal Sans, Aboriginal Serif, ALPHABETUM Unicode, Arial, Arial Unicode MS, Berling Antiqua, Bitstream CyberBase, Bitstream CyberBit, Bitstream CyberCJK, Cardo, Caslon, Chrysanthi Unicode, CN-Arial, CN-Times, Code2000, Courier New, Doulos SIL, ESL Gothic Unicode, Everson Mono Unicode, Fixedsys Excelsior, Free Monospaced, Free Serif, Frutiger Linotype, Gandhari Unicode, Gentium, GentiumAlt, Hindsight Unicode, jGaramond, Junicode, Kozuka Mincho Pro Acro, Legendum, Linux Libertine, Lucida Console, Lucida Sans (few), Lucida Sans Typewriter (few), Lucida Sans Unicode, MD King KhammuRabi, MgOldTimes UC Pol, Microsoft Sans Serif, Ming(for ISO10646), Monospace, MS Gothic, MS Mincho, MS PGothic, MS PMincho, MS Reference Sans Serif, MS Reference Serif, MS UI Gothic, MSung Std Acro, Naqsh, New Athena Unicode (few), NSimSun-18030 (few), SImPL,

SimSun-18030 (few), Tahoma (few), Thryomanes, Times New Roman, TITUS Cyberbit Basic, TSC JSong S TT, Verdana (few), Vusillus Old Face

Mac Fonts Arial, Gentium, GentiumAlt, Hiragino Kaku Gothic Pro, Hiragino Maru Gothic Pro, Hiragino Mincho Pro, Junicode, Lucida Grande, MS Gothic, MS Mincho, MS PGothic, MS PMincho, SimSun (few), STFangsong (few), STHeiti (few), STKaiti (few), STSong (few), Times New Roman, Verdana (few), Zapfino (few)

Ressourcen

- WGL/4 Zeichensatz
- Mozilla Browser font handling
- WGL4
- Alan Wood Unicode Ressources: Unicode Fonts
- Unicode Font Guide For Open Source Operating Systems
- Myfont: Languages and compatible Fonts
- Fieformat: Unicode Character Details, including font information
- Font ressource for Unicode scripts

3.6.2 Webinterfaces

Ausgabe auf dem Client

Die Ausgabe von Sonderzeichen auf Webseiten ist immer abhängig von den tatsächlich im System installierten Schriften (siehe Kapitel 3.6). Bei vorhandenen Schriften ist auf aktuellen Betriebssystemen kaum mit Beeinträchtigungen zu rechnen. Es gibt aber auch browser-spezifische Unterschiede.

Internet Explorer Die Unicode-Unterstützung des Microsoft Internet Explorers ist eher mäßig. Schriften können nur auf Basis der Schriftfamilie (*Latin-based*) gewählt werden, aber nicht auf Basis der Codierung (*Unicode*). Da der Bowser nicht in der Lage ist, in Abhängigkeit von vorhandener Codierungsinformation oder der Zeichenauswahl eine passende Schriftart zu wählen, bleibt das Ergebnis trotzt installierter Fonts oft hinter den Erwartungen zurück. Die voreingestellten Fonts liefern allerdings für österreichische Belange ausreichende Ergebnisse.

Bei installiertem Office Paket empfiehlt es sich, die im Paket enthaltenen Microsoft Arial Unicode Font als Default-Schriftart für Latin-based Scripts zu wählen (siehe Kapitel 3.6.3).

Mozilla/Firefox/Netscape Mozilla basierte Browser versuchen jedes Zeichen darzustellen, indem sie nach einer Font suchen, die eine Glyphe für das gewünschte Zeichen enthält. Abhängig von den installierten Fonts wird so eine sehr gute Abdeckung erzielt.

Serverbasierte Konvertierung

Um Probleme mit nicht unicodefähigen Browsern auszuschließen, ist es prinzipiell auch möglich Strings mit diakritischen Zeichen auf dem Server in Grafiken umzuwandeln, was natürlich die weitere Verwendung als Text ausschließt. Diese Methode sollte nur zusätzlich zur Übertragung als Text zum Einsatz kommen.

Die Technische Umsetzung kann durch kommerzielle Systeme (zum Beispiel Glyphgate), oder auch durch Methoden eines bereits eingesetzten Publishing Systems (z. B. Cocoon) erfolgen.

3.6.3 Empfehlung: MS Arial Unicode als Default-Schriftart einsetzen

Auf Windows basierten Systemen wird empfohlen, bei Verfügbarkeit der unter anderem in Microsoft Office enthaltenen Schriftart Microsoft Arial Unicode (24 Megabyte große Datei Arialuni.ttf), diese als Default-Schriftart für Latin-based Scripts einzusetzen.

3.6.4 Desktopsoftware

Microsoft Windows

Ab Windows NT unterstützt die Windows Plattform Unicode und speichert Text intern in UTF-16. Die Microsoft eigenen Entwicklungsumgebungen arbeiten i. A. mit Makros, sodass die entstehende Applikation per Präprozessoranweisung von ANSI auf Unciode Support umgeschaltet werden kann. Die Betriebssystemunterstützung wird über verschiedene Lokalisierungspackages bereitgestellt. Eine deutsche oder englische Betriebssystemversion unterstützt aber von Grund auf alle lateinischen Buchstaben. Die Unicode Unterstützung erstreckt sich auch auf Microsoft Office (siehe Kapitel 3.6.3).

Unix/Linux

Die meisten UNIX basierten Betriebssysteme bieten Unicode Unterstützung über das lokale Programm und die zugehörigen Umgebungsvariablen. Zusätzlich müssen noch die passenden Fonts (siehe Kapitel 3.6) installiert sein. Bei modernen Linux Distributionen ist das im Allgemeinen der Fall. Einige Kommandozeilen-Programme (zum Beispiel less) und ältere Standalone-Applikationen (zum Beispiel emacs mit mule) bieten eigene Unicode-Implementierungen.

OpenOffice

Open Office stellt intern alle Zeichen als Unicode dar. Die Zeichenauswahl erfolgt über die Tastatur (entsprechendes Keyboard-Layout vorausgesetzt) oder über den *Einfügen->Sonderzeichen* Befehl durch Auswahl aus einer Unicode Liste. Spellchecker und andere Tools sind ebenfalls in mehreren Sprachen erhältich (Lingucomponent Project).

Apple Macintosh

Apple Betriebssysteme (insbesondere Mac OS X) bieten eine sehr gute Unicode Unterstützung. Zum Arbeiten mit mehreren West- und Osteuropäischen Sprachen empfiehlt es sich die Tastaturbelegung *Extended Roman Unicode* zu benützen. Mac OS 9 verlangt zusätzlich noch das *Central European Language Kit*. Darüber hinaus bietet Apple noch eine sehr komfortable Zeichenpalette, die weit über den üblichen Funktionsumfang hinaus geht.

Empfehlung: Extended Roman Unicode als Tastaturbelegung benutzen

Auf Apple Macintosh basierten Systemen wird empfohlen, die Tastaturbelegung Extended Roman Unicode zu verwenden.

Weitere Informationen

Weitere Infomationen finden sich auch in Dateneingabe auf Personal Computern.

Ressourcen

- Alan Woods Unicode Ressources: Tools and Utilities
- A Quick Primer On Unicode and Software Internationalization Under Linux and UNIX
- Linux Unicode Programming
- Unicode HOWTO
- Gentoo UTf-8 System Guide,Gentoo UTF-8 Wiki, auch für andere Systemem brauchbar
- Unicode on Unix/Linux
- Mac: Anzeigen und Eingeben von Texten in verschiedenen Sprachen
- Apple: Getting Started With Internationalization
- sun: Globalisation Guide

- sun: JAVA Internationalization FAQ

3.6.5 Druckausgabe

Windows / Mac

Beide Systeme bieten WYSIWYG Printing. Sobald Text auf dem Bildschirm richtig angezeigt wird, wird er im Allgemeinen auch korrekt gedruckt. Ausnahmen entstehen nur durch schlechte Drucker(treiber)konfiguration (Verzicht auf Fontdownload).

Linux / Unix

Druck von Sonderzeichen auf Unix-basierten Systemen gestaltet sich etwas schwieriger. Während der Ausdruck von PDF Dateien normalerweise reibungslos verfläuft, sind Ausdrucke von Webseiten etwas komplizierter. Auf direktem Wege funktioniert das nur, wenn ein Postscript-fähiger Drucker mit entsprechendem Zeichensatz vorhanden ist, wovon man in der Regel aber nicht ausgehen kann. Als Alternativen bieten sich diverse externe Filterprogramme an, die in die Druckkette eingehängt werden. Details in den Ressourcen.

Ressourcen

- Linux Unicode Howto: Printing
- Mozilla: XPrint
- wprint
- Linuxprinting.org
- Yudit unicode editor printing tool

Chapter 4

Glossar

Ressourcen

Da die erklärungsbedürftigen Begriffe fast auschließlich aus dem Unicode-Umfeld stammen, wird an dieser Stelle nur auf das Glossar des Unicode-Gremiums verwiesen.

- unicode.org: Glossary

Chapter 5

Literatur

- The Unicode Consortium. The Unicode Standard, Version 4.0.0, defined by: The Unicode Standard, Version 4.0 (Boston, MA, Addison-Wesley, 2003. ISBN 0-321-18578-1)
- Unicode: A Primer. Tony Graham (Foster City, CA, M&T Books, 2000, ISBN 0-7645- 4625-2)
- Unicode Demystified. Richard Gillam (Boston, MA, Addison Wesley, ISBN 0-201-70052-2)

5.1 Externe Referenzen

- W3C Internationalization
- On the Goodness of Unicode
- The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets (No Excuses!)
- unicode.org
- Was ist Unicode? (unicode.org)
- ISO/IEC 10646
- Allen Woods Unicode Page
- IBM Unicode Browser
- RFC 3629
- IANA character sets

- Coverage of European languages by ISO Latin alphabets
- Combining diakritic marks
- Unicode Normalization Forms Version
- Beispiele aus dem Annex
- Normalization Charts
- Latin 1
- Latin-1 Supplement
- Latin Extended A
- Latin Extended B
- Latin Ext. Additional
- Combining diacritic marks
- Alphabetic Presentation Forms
- IPA Extensions
- Everson Typography - Alphabeths of Europe
- Omniglot.com
- Wikipedia: Latin Alphabet
- Wikipedia: Alphabets derived from the Latin
- Wikipedia: Diakritische Zeichen
- Diakritsche Zeichen
- Alphabets derived from the Latin Alphabet
- Unicode.org: Alphabetic Presentation Forms (Ligaturen)
- Latin 1
- Latin-1 Supplement
- Latin Extended A
- Latin Extended B
- Latin Ext. Additional

- Combining diacritic marks
- Alphabetic Presentation Forms
- 2-stellige Sprachcodes (ISO 639)
- W3C Internationalisation Information
- W3C Tutotrial zu (X)HTML und Encodings
- W3C - i18n: The HTTP charset parameter
- UniView, JavaScript-basierte Unicodetabelle
- Microsoft Windows XP - Die technische Referenz
- Apple Unicode Keyboard Layouts
- Details zu Formaten für die Anzeige der Bürgerkarten-Umgebung
- Formate für die Anzeige der Bürgerkarten-Umgebung
- DB2 UDB for z/OS Internationalization Guide
- Introduction to Unicode (part I)
- Introduction to Unicode (part II)
- DB2 Database Design: Using Unicode
- DB2 Database Development: Unicode CLI aplications
- DB2 Database Development: JDBC and Unicode Streams
- Globalization Support Guide
- Unicode Support diverser Oracle Versionen
- Unicode Datenbanken oder Unicode Felder?
- Unicode DataBlade Module User's Guide, Version 1.0
- Unicode in Ingres
- MaxDB und Unicode
- Unicode und ODBC
- Using Unicode Data
- Using Collations

- MySQL Unicode support
- Unicode charactersets and Collation
- Character Sets
- String Funktionen
- Pattern matching Functions
- RFC 2253 - Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names
- Beispiele für die LDIF Codierung
- HTML Entities
- Make your Programs Unicode aware
- sun: JAVA Tutorial: Internationalization
- sun: JAVA Tutorial: Working with Text
- sun: Internationalization FAQ
- Unicode in JAVA
- IBMs extensive Unicode library für JAVA
- ASP.Net Unicode
- ECMAscript Specification
- Multibyte String Extension
- Multibyte Varianten
- Multibyte String Extension
- Perl, Unicode and i18N FAQ
- iconv lib download from CPAN
- Python und Unicode
- Dive into Python - Unicode
- libiconv
- C and Unicode

- Microsoft: Unicode C/C++ Programming
- Unicode-enabling Microsoft C/C++ Source Code
- IBM ICU library for C
- http://www.macchiato.com/unicode/convert.html
- unicode.org: Byte Order Mark (BOM) FAQ
- UTF-8 Erkennung mit Regular Expressions
- Weitere Erkennungs- und Konvertierungsalgorithmen
- Canonical XML
- Code2000/Code2001
- Lucida Sans
- Thesaurus Indogermanischer Text- und Sprachmaterialien
- W3C Teseite
- Alan Wood
- WGL/4 Zeichensatz
- Mozilla Browser font handling
- WGL4
- Alan Wood Unicode Ressources: Unicode Fonts
- Unicode Font Guide For Open Source Operating Systems
- Myfont: Languages and compatible Fonts
- Fieformat: Unicode Character Details, including font information
- Font ressource for Unicode scripts
- Glyphgate
- Cocoon
- Lingucomponent Project
- Dateneingabe auf Personal Computern
- Alan Woods Unicode Ressources: Tools and Utilities

- A Quick Primer On Unicode and Software Internationalization Under Linux and UNIX
- Linux Unicode Programming
- Unicode HOWTO
- Gentoo UTf-8 System Guide
- Gentoo UTF-8 Wiki
- Unicode on Unix/Linux
- Mac: Anzeigen und Eingeben von Texten in verschiedenen Sprachen
- Apple: Getting Started With Internationalization
- sun: Globalisation Guide
- sun: JAVA Internationalization FAQ
- Linux Unicode Howto: Printing
- Mozilla: XPrint
- wprint
- Linuxprinting.org
- Yudit unicode editor printing tool
- unicode.org: Glossary