Critical Values for I18n Testing

Tex Texin
Chief Globalization Architect
XenCraft

Abstract

In this session, we recommend specific data values that are likely to identify internationalization problems in software intended for global markets.

Based on years of global software experience, these data values are useful for functional or linguistic QA tests of internationalized software. In the last session, data value recommendations included character encoding, postal address, locale and other data types typically used in software and will trigger common internationalization problems. This presentation will offer specific test suggestions.

 3

Assumptions: QA you already know

Knowledge and practice of traditional i18n functional QA techniques are assumed
- Functional vs. Linguistic QA
- Pseudolocalization (Psèüdõlõcâlîzâtîõn)
- Testing with different locale settings
- Testing with native software (Operating System, Browsers, 3rd party software, etc.) and devices

- Security and other aspects are not addressed

 5

Test Results Possibilities

Possibilities	True	False
Positive	True Positive	False Positive
Negative	True Negative	False Negative

Results	True	False
+ (Tests Fail)	Actual Bug(s)	Imagined Bug(s)
- (Tests Pass)	Quality Assured	Hidden Bug(s)

True +	True -	False +	False -
You earned your money	The team did well!	Investigation time down the drain	Customer dissatisfaction, corporate embarassment, heads may roll

 7

Test Results Possibilities

Possibilities	True	False
Positive	True Positive	False Positive
Negative	True Negative	False Negative

Results	True	False
+ (Tests Fail)	Actual Bug(s)	Imagined Bug(s)
- (Tests Pass)	Quality Assured	Hidden Bug(s)

Of course, assuring quality assumes adequate test coverage.

Often, bugs can remain hidden *despite voluminous testing*.

Voluminous testing gives false confidence that coverage is adequate and quality is assured.

 8

Volume testing

It is a mistaken statistical premise that testing many random values will uncover problems.

- Automated testing can exercise many values, but repeats them each test session.
- Problem values are not randomly distributed.
- Testing without knowledge of the underlying architecture often misses problem areas.
- Testing hits only a small % of all possible values

- Unicode
  - 150 Scripts
  - 87,887 Ideographs
  - 137,929 Characters
  - 19,024,409,041 Combinations of 2 characters
  - Bazillions of Character Combinations for a string of N characters.

 10

Large test data sets hide problems

Big (random) data sets miss key problems

- Volume testing creates the incorrect impression that the software is robust

Numerous examples of catastrophic errors

- Intel FPU arithmetic error
- DG systems won’t boot on a certain date
- Text corruption/Mojibake

False Negatives can be consequential

- To the business
- And often easily experienced by users

 11

How to adequately test large data sets?

E.g. Text, Date-time, Arithmetic, Identifiers

Solutions?

- Test every possible value and permutation
  - At least once, possibly over a long period
  - Don’t test same values repeatedly
- User, random and other testing
- Use architecture knowledge to test risky areas
  - Limits, Boundaries, Syntax, Error cases
- Test with values that have found problems before
- There isn’t one answer short of testing all cases, and repeating in different sequences
  - Testing with problem-finding values is a minimum

 12

Goal: Critical Values For I18n Testing

Goal: Test values that provoke problems or raise issues for global software

- Use real-world business values where possible
  - Domains: Text, Date-Time, Identifiers (email, URL)
- Avoid exotic or contrived examples

 13

Text and Unicode

- Unicode is a large character set
- 137,000+ characters & millions of variants
  - Including combining characters
  - Varying length characters and strings
- Many characters are similar visually or functionally, and distinct in unexpected ways
- Explicit and implicit rules
- Locale-based behaviors
- Continually evolving

 14

Wide Loads: Full Width Characters

ASCII (et al) characters are duplicated

Full Width: ！＂＃＄％＆＇（）＊＋，－．／０１２３４５６７８９：；＜＝＞？＠ＡＢＣＤＥＦＧＨＩＪＫＬＭＮＯＰＱＲＳＴＵＶＷＸＹＺ［＼］＾＿｀ａｂｃｄｅｆｇｈｉｊｋｌｍｎｏｐｑｒｓｔｕｖｗｘｙｚ｛｜｝～

ASCII: !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~

- Test Case: Use full width (Ideographic) "＊" and other characters in keywords and operators
  - Full Width characters may be syntax equivalents.
    - Inconsistent across products, even where standards exist

SELECT ＊ FROM Employees WHERE LastName='龍' AND (FirstName='陳元' ＯＲ FirstName='Jackie')

 15

The Invisible Man and Invisible Girl

White space (space, tab, return, et al) often needs trimming or other special handling

1. No-Break Space (U+00A0,) often missed
   - Is Group separator in some locales (e.g. **1 234 567,89**)
2. Full width (ideographic) space (U+3000)
3. Other spaces

En Space U+2002
Em Space U+2003
Three-Per-Em Space U+2004
Four-Per-Em Space U+2005
Six-Per-Em Space U+2006
Figure Space U+2007
Punctuation Space U+2008
Thin Space U+2009
Hair Space U+200A
Zero Width Space U+200B

 16

The Invisible Man and Invisible Girl

- Test Data NBSP, Full Width Space

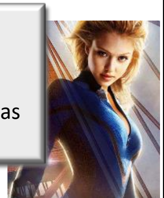
Although No-Break Space is not usually typed in by users, it is often copy/paste into fields.

If NBSP or Full Width Space are not trimmed, searches can fail, or duplicate entries can occur

Duplicate Entries	String	String+NBSP	String+FWSP	NBSP+String

Search for "String", won't match "String+NBSP".

Create User "String+NBSP" won't fail (as expected) if "String" already exists.



The usual suspects

Unicode Storage Test Values

Background
- UTF-8 characters can be 1-4 bytes (1-4 octets)
- UTF-16 characters can be 1-2 words (1-2 16-bit units)
- U+10000-U+1FFFFF are 4 bytes or 2 words

Typical software text storage errors
- Assume 1 character =1 byte or 1 word
- Assume 1 UTF-8 character <= 3 bytes
- Assume maximum length string will not occur and less is adequate
- Assume text operations do not change length (e.g. case)
- Memory allocation or string indexing “off by one” errors
- Inconsistent definitions of maximum length (e.g. UI <> DB)
- Performance degrades or severe memory leaks with long strings

Unicode Text Examples

- One byte “A”
- Two Bytes “É” or Three bytes “E” + “´”
- Three Bytes “€”
- Four Bytes “𠜎”

Test various length characters and strings to find mistakes in string indexing, line wrapping, etc.
- E.g. wrapping a line in the middle of a character.
- E.g. Family Emoji: 15 Characters for a single emoji

Man, Dark Skin Tone, Joiner, Bald, Joiner,
Woman, Med. Skin Tone, Joiner, Red Hair, Joiner,
Girl, Light Skin Tone, Joiner,
Boy, Medium Light Skin Tone

Unicode Storage Test Values

Maximum length strings using maximum length characters (Supplementary characters > U+FFFF)

Frequently used 4-byte Supplementary characters
- U+2070E, U+20731, U+20779, U+20C53, U+20C78, U+20C96, U+20CCF, U+20CD5, U+20D15 and more
- See i18nguy.com/unicode/supplementary-test.html

- Provokes memory overruns, off by one errors, errors where the validation rules are incorrect

Test1: create maximum length strings with them

Unicode Storage Test Values

Test character substitution near boundaries

4 Characters 4 Bytes	A	A	A	A		
4 Characters 5 Bytes	A	A	A	É		

4 Characters 4 Bytes	A	A	A	A	
4 Characters 5 Bytes	A	É		A	A

Test Case	Rationale
1. Fill field to maximum length with ASCII characters 2. Replace one character with a Supplementary character	Test that field supports full count of characters not bytes OR If counting bytes, that validation prevents overruns
1. Fill field to maximum length-1 (or -2, -3) with ASCII characters 2. Replace one character with a Supplementary character	Verify that validation prevents overruns

 21

Case rules

This building is for ~~eels~~ congress. (congrès vs congres)

Palais des congrès

Background

- Case character maps can be more complex than English

Typical software errors

- Assuming that string length doesn't change
- Assuming that case rules are reversible

Failure cases

- Upper(ß) => "SS" Lower(SS) => "ss"
- Length(eßen) = 4 Length(Upper(eßen)) = 5

 22

Case rules

Example real world failure

- Case-insensitive compare used to determine if the record changed and should be saved
  - If (**Upper(CurrentVal) <> Upper(InputVal)**) save-changes;
  - Fails if the User edits the field and the change is not saved

Description	CurrentVal	InputVal	Result
Locale = fr_CA	"résumé"	"resume"	SUCCESS! "RÉSUMÉ"<>"RESUME"
Locale = fr_FR	"résumé"	"resume"	FAIL! "RESUME"="RESUME"
Different lowercase characters with the same uppercase	"essen"	"eßen"	FAIL! "ESSEN"="ESSEN"

 23

Case rules

"ß" U+00DF is a useful character for i18n testing

Tests for poor case mappings assumptions

Test Data	Assumption	Reality	Failure Case
ß (U+00DF)	case rules are reversible	Upper(ß) => "SS" Lower(SS) => "ss"	X=Upper(Lower(X))
ß (U+00DF)	string length doesn't change with case	Length(eßen) = 4 Length(Upper(eßen)) = 5	X = mem(4) X = Upper("eßen")
ß (U+00DF)	Unique strings remain unique with case changes	Upper(eßen)=ESSEN Upper(essen)=ESSEN	X="eßen" Y="essen" X<>Y Upper(X)=Upper(Y)

 24

Example Tests For Case Rules

Test for uppercasing overrunning memory

Data Case 1: Use maximum length run of "ß"

- "ßßßßß" 5 chars become 10: "SSSSSSSSSS"

Data Case 2: Edit a maximum length string, replace one character with "ß"

Enter up to 8 characters:	Enter up to 8 characters:	Enter up to 8 characters:
testdata	testdatß	testdatß
Storage	Storage	Storage
t e s t d a t a	t e s t d a t ß	T E S T D A T S S

Locale-dependent case rules

Case mappings are locale-dependent

- Turkish is special as it changes case of ASCII characters

Language	Case Rules	Notes
English	Upper(i) = I	Dotted i uppercases to Dotless I
Turkish	Upper(i) = İ Upper(ı) = I	Dotted i uppercases to Dotted İ Dotless ı uppercases to Dotless I

Recommended: Test with Turkish locale

- Even if you don't localize for or plan to support Turkey

Test Data	Assumption	Failure Case
i (U+0069) I (U+0049) İ (U+0130) ı (U+0131)	Case rules for ASCII characters do not change	Upper(exit)<>EXIT

Text me!

Character Tests

Syntax Character Tests

Software uses many programming languages and protocols. Each has its own syntax with "special" characters and escapes.

Note: A syntax can rely on a sequence of characters.
Note: A syntax can rely on pairings of characters.
e.g. " " or /* */

Language / Protocol	Example Syntax Characters	escape	Examples
HTML, XML	& < > ' "	&	& €
URL	? & # + / . : = %	%	tex.com?a=san+jos%C3%A8+%26+CA
Javascript	' " \ // /* */ \u	\	\n \\ \u20AC

Syntax Character Tests

Potential Failure cases

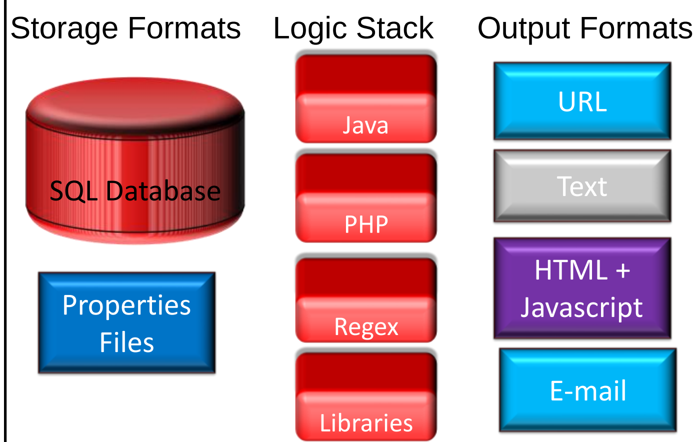
- Syntax characters are not escaped
- Syntax characters are escaped twice
- Syntax characters used in 2 or more languages, are escaped and unescaped in different orders

Problems can occur because data comes from different sources, goes thru different code paths, and is intended for different uses.

- E.g. Text in properties files destined for HTML vs e-mail vs javascript alert vs URL, et al
  - Each has different escape requirements

 29

Syntax Character Tests



 30

Syntax Character Tests

Recommendation: Test with syntax-significant characters everywhere

& + / \ " ' ( ) ? . # @ _ - ~

Escape/Unescape example Ampersand "&"

Character	HTML Escape	URI Escape
&	&	%26

Example: M&M	Step 1	Step 2
Encode M&M HTML followed by URI	M&M	M%26amp;M
Decode M%26amp;M URI first	M&M	M&M
Decode M%26amp;M HTML first	M%26amp;M	M&M

 31

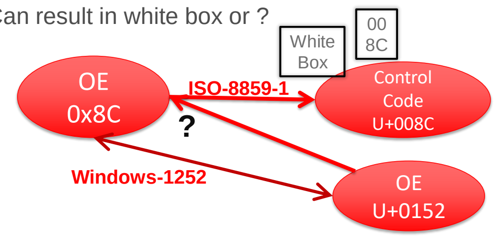
ISO-8859-1 vs Windows-1252

Very common error: incorrect charset name

- For CSV, HTML, XML, e-mail, and other file formats, DB, HTTP and other protocols

Background

- Convert "ISO-8859-1" from/to Unicode is incorrect
  - Can result in white box or ?



 33

ISO-8859-1

Every code point is used. C1 control codes are legacy of mainframe and minicomputers

0	0 - 1F Control Codes																															
20		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
80	80-9F C1 Control Codes																															
A0		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
C0	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E0	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

 34

Windows 1252

C1 control codes replaced by characters. Some code points are not (yet) assigned (81, 8D, 8F, 90, 9D)

0	0 - 1F Control Codes																															
20		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
80	€		‚	ƒ	„	…	†	‡	ˆ	‰	Š	‹	Œ		Ž			‘	’	“	”	•	–	—	˜	™	š	›	œ		ž	Ÿ
A0		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
C0	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E0	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

 35

ISO-8859-1 vs Windows-1252

Test Recommendations

- Use characters in 0x80-0x9F (128-159)
  - Dashes: – —
  - Punctuation: Smart quotes ‘’“”
  - Ellipses …
  - Left/Right Single Pointing Quotation Marks: ‹ ›
  - Characters: OE Ligature "Œ œ", Trademark "™"

1. Œ easy to use in text data & pseudolocalization
2. Use Ÿ and ÿ for case tests
   - ÿ U+00FF is in both encodings
   - Ÿ U+0178 0x9F (159) is only in Windows-1252

 36

Double Decode Detector

Common error is double conversion to UTF-8

- Often data goes in and out without error being noticed

	Start	Convert Windows-1252 to UTF-8		Convert Back to Windows-1252	
		1st time	2nd time	1st time	2nd time
Most characters succeed	Â 0xC3	Âƒ 0xC3 0x83	Âƒ‚ƒ 0xC3 0x83 0xC2 0x83	Âƒ 0xC3 0x83	Â 0xC3
Test Data showing error	Á 0xC1	Â? 0xC3 0x81	**Error 0x81 doesn't exist**		

- Recommend Test Character Á (U+00C1) A-acute
  - Or others with byte value of 81, 8D, 8F, 90, 9D

 37

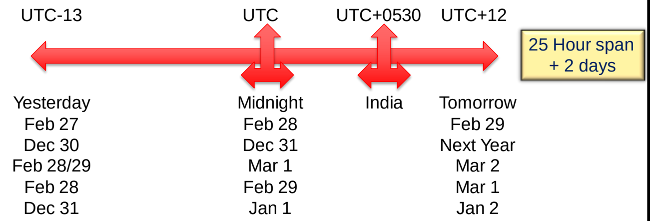
From time out of mind

Date-time Tests

The Date-Time Continuum

Time zone affects date as well as time

- Often overlooked, but important to test
- Use data that crosses datelines, dates, months
- Use time zones with offsets other than 00

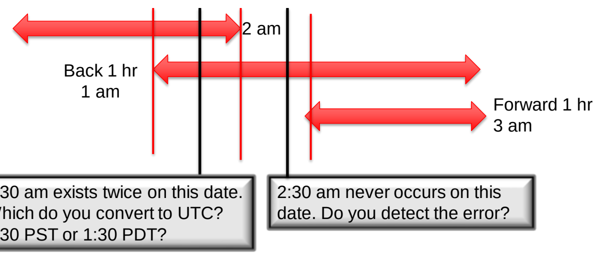


 39

The Date-Time Continuum

Daylight Savings

- Test gaps in single time zone
- Test gaps crossing multiple time zones
  - E.g. PST-BST, PST-BDT, PDT-BST, PDT-BDT



 40

The Date-Time Continuum

- Time zones change
- Test changing periods of Daylight Savings
  - How do you find dates that need updating?

 41

Knock Knock! Who's there?

Identity Tests

Identify yourself! Names, Postal Addresses

Vast differences in conventions

- Standards are not consistent
  - Due to differences in applications and use cases

Recommended Tests

- Country Code: UK
  - ISO 3166 code is GB   Postal code is UK
    Source of errors
    - ISO Code GB causes mail to be rejected.
    - Postal Code UK will fail validations using ISO Codes
  - DB & XML Identifiers should distinguish them
    - NOT <Country Code> but <PostalCountryCode> <ISOCountryCode>
- Countries with no post (zip) code
  - ~~Ireland~~, Somalia, Syria (Although these may be changing)

Identify yourself! Names, Postal Addresses

Field sizes are often inadequate

- Longest Surname MacGhilleseatheanaich
- Longest Street Name (Poland)
  Dwudziestego Pierwszego Praskiego Pulku Piechoty imienia Dzieci Warszawy
- Long Street Name with no spaces
  Carl-Philipp-Emanuel-Bach-Straße Frankfurt (Oder), Germany
- Longest City Name (Anglesey, Wales 58 characters)
  - Llanfairpwllgwyngyllgogerychwyrndrobwllllantysiliogogogoch

E-mail

E-mail tests

- Test syntax characters in email
  - "Téx" <Tex+1@xencraft.com>
- Test maximum size (254 bytes)
- Test International domain names in mail
  - TEX@globalização.biz (Need your own domain)
  - tex@xn--globalizao-n5a1c.biz
  - Verify equivalence for search

Web Address / International Domain Names

Test Cases for International Domain Names (IDN)

- Max length (255-2048)
  - Mobile and Personal device browsers have lower limits
    - http://thelongestlistofthelongeststuffatthelongestdomainnameatlonglast.com
    - http://www.thequickbrownfoxjumpsoveralazydog.com
- Syntax characters, escapes (&, /, + # ? . % :)
- International TLD, domain and subdomain names
  - http://www.xn--globalizao-n5a1c.biz
  - http://www.globalização.biz
  - http://globalização.globalização.biz
    - Testing subdomains are useful if you don't have an IDN
  - Test search equivalence of international and ASCII-fied names
  - Evaluate display
  - Test international domain names as parameters in URLs
    - google.com/search?q=http%3A%2F%2Fglobaliza%C3%A7%C3%A3o.biz

Web Addresses

Test Cases

- International path

  www.xencraft.com/globalização/globalização
  - Verify links in non-UTF-8 pages first convert to UTF-8

    Ç is %C3%A7 not %C7
- International query and fragment, including non-UTF-8

  www.xencraft.com/globalização/?a=çç#ã%C3%A7

Questions?

Tex Texin

TexTexin@Xencraft.com

Tex is an industry thought leader specializing in business and software globalization services. His expertise includes global product strategy, Unicode and internationalization architecture, and cost-effective implementation and testing. Over the past two decades, Tex has created numerous global products, led internationalization development teams, and guided companies in taking business to new regional markets.

Tex is a contributor to internationalization standards for software and on the Web.

Tex is a popular speaker at conferences around the world and provides on-site training on Unicode, internationalization, and globalization QA worldwide.

Tex is the author of the popular, instructional web site www.I18nGuy.com

Tex is founder and Chief Globalization Architect for XenCraft. XenCraft provides global business consulting and software design, implementation, test and training services on globalization product strategy and software internationalization architecture.