Proper display of bidirectional structured text

Authors

- Aharon Lanin (Google)
- Gilead Almosnino (Microsoft)
- Lina Kemmel (IBM)
- Mati Allouche (former IBM)
- Mohamed Mohie (IBM)
- Tomer Mahlin (IBM)

Basic terminology

1. **Structured text** - text with inherent structure which must be preserved during display to assure its readability. This article focuses exclusively on the following types of structured text:
    a. URI (i.e. http://www.hello.world.com/folder)
    b. email address (i.e me@company.com)
    c. regular expression (i.e. [a-z]{1})
    In this article the term also appears abbreviated as STT (**st**ructured **t**ext).
2. **Bidi - Bidirectional**
    a. RTL character - character which belongs to a language with right-to-left script (e.g. Arabic, Hebrew, Farsi etc.)
    b. RTL language - language with RTL script (e.g. Arabic, Hebrew, Farsi etc.)
    c. Bidi language[1] - a synonym for RTL language
    d. RTL sentence / phrase / message - a sentence, message or phrase which belongs to a RTL language (e.g. Arabic, Hebrew, Farsi etc.)
    e. Bidi text / data / string - text which includes RTL characters but not necessarily belongs to a bidi language. It can be an English sentence that includes RTL, e.g a word in Hebrew or Arabic script.
3. **BTD** – **b**ase **t**ext **d**irection. The overall direction of a piece of text and a parameter to the UBA. LTR for English sentences. RTL for Arabic and Hebrew sentences. If the incorrect BTD is applied to a piece of text, it may

[1] Please note that neither characters nor languages themselves are bidirectional. Each separate character either has a well defined unique direction (e.g. LTR for a Latin-script character and RTL for a Hebrew character) or is neutral (e.g. punctuation and whitespace). A language by itself has no direction; it is only its written expression which has a direction through the script it uses. For convenience, the full and correct definition (e.g. language written with a bidirectional script, character which belongs to a bidirectional script) is usually replaced by a formally incorrect but significantly shorter description.

be displayed incorrectly.
    a. LTR – left-to-right (value of base text direction)
    b. RTL – right-to-left (value of base text direction)
    c. Auto (aka First Strong) – base text direction is derived from the text content - the first “strong” character wins.
4. **RTE** –Rich text editor (Dojo RTE, CKEditor)
5. **PII** – Program Integration Information – translatable text packaged with the product and used for product’s localization
6. **GUI** - Graphical User Interface
7. **UBA** - Unicode Bidirectional Algorithm governing proper reordering of text on logical platforms like HTML, Windows, Android, etc.
8. **ICU** - International Components for Unicode (http://site.icu-project.org/)

Conventions

The examples used in this article assume the following conventions:

- Upper case letters denote letters like Arabic and Hebrew which are to be read from right to left. Lower case letters denote letters like Latin or Greek which are to be read from left to right.
- If not specified otherwise, the examples are shown in **display order**. In other words, they appear in the same order as shown on the product GUI.
- Some examples are shown in **logical order** (the order in which characters are normally typed and spoken, also known as typing or chronological order; this is also the order in which characters are stored in memory, also known as storage order). Note that the content of storage is always shown with increasing addresses going from left to right.

Background

Display of bidirectional text on logical platforms (e.g., Android) is governed by the Unicode Bidirectional Algorithm (UBA) . The UBA is used by presentation systems on those platforms and provides satisfactory results for display of free form bidirectional text. However, not all bidirectional text is free form text. Structured text has a predefined, machine-readable syntax or internal structure, and sometimes structured text is also bidirectional. Unfortunately, the UBA was never designed for structured text, is unaware of its structure, and as a result can obfuscate its structure, making the text’s display misleading or even incomprehensible.

Preamble

Q: Why did the bidirectional structured text topic become important only now ?

A: Because RTL characters have been recently allowed in several important types of structured text and are starting to be used.

During the last decade, usage of of non-English data gradually became more and more widespread until one day it became ubiquitous. This advance started from content data (stored as part of file content on the file system or as part of database content in RDBMS). It spread to descriptive data or metadata, in the beginning as part of file or directory names on the file systems and entity names in RDBMS (such as database, table or column names), then in email addresses, URIs and even world wide web domain names (international domain names, international domain name part of e-mail address).

Scope of discussion

Q: What is the nature of the problem ? Is it display specific or does it pertain to data processing as well ?
A: The problem is specific to display only. Text with internal structure or syntax that includes RTL characters may not be properly displayed to the end user. However, improper display does not negatively affect the way this text is processed.

Q: If the problem is specific to display only, how severe it is for the end user ? Is there a chance the user can get used to the improper display of bidi data ?
A: Depending on the syntax and the actual data used, the display may vary from normal to unusual to unexpected to misleading to completely incomprehensible. Consequently, if the problem of **st**ructured **t**ext (STT) display is not handled appropriately, the program graphical front end may appear to the bidi end user not simply unfriendly (in the best case) but as broken (in the worst case).

Q: What is the scope of the problem ? In which contexts structured text can appear ?
A: It is important to point out that in general case, applications not only need to properly display structures text, they also need to allow the end user to enter structured text. A structured text value needs to be displayed the same during entry and subsequent display.

As a result of increased usage of non-English data, applications working with data sources supporting usage of such data (i.e. applications exchanging data with file systems, RDBMS, email servers, web servers, or applications used for data/solution/process modeling and editing of source code etc.) are forced to face working with this type of data in sometimes very unexpected contexts.

Since bidirectional structured text can and does pop up in just about any application, and most application developers don't have sufficient background about bidi much less about bidirectional structured text, the means of handling it properly must be **extremely easy to use** for the average application developer.

It must be noted that in order to support proper handling and correct display of non-English data, a wide variety of aspects must be addressed. In this article, we focus on specifics of

bidirectional data display in a very unique context: display of data having complex internal structure or syntax (referred as “structured text”).

Basics of bidirectional text display

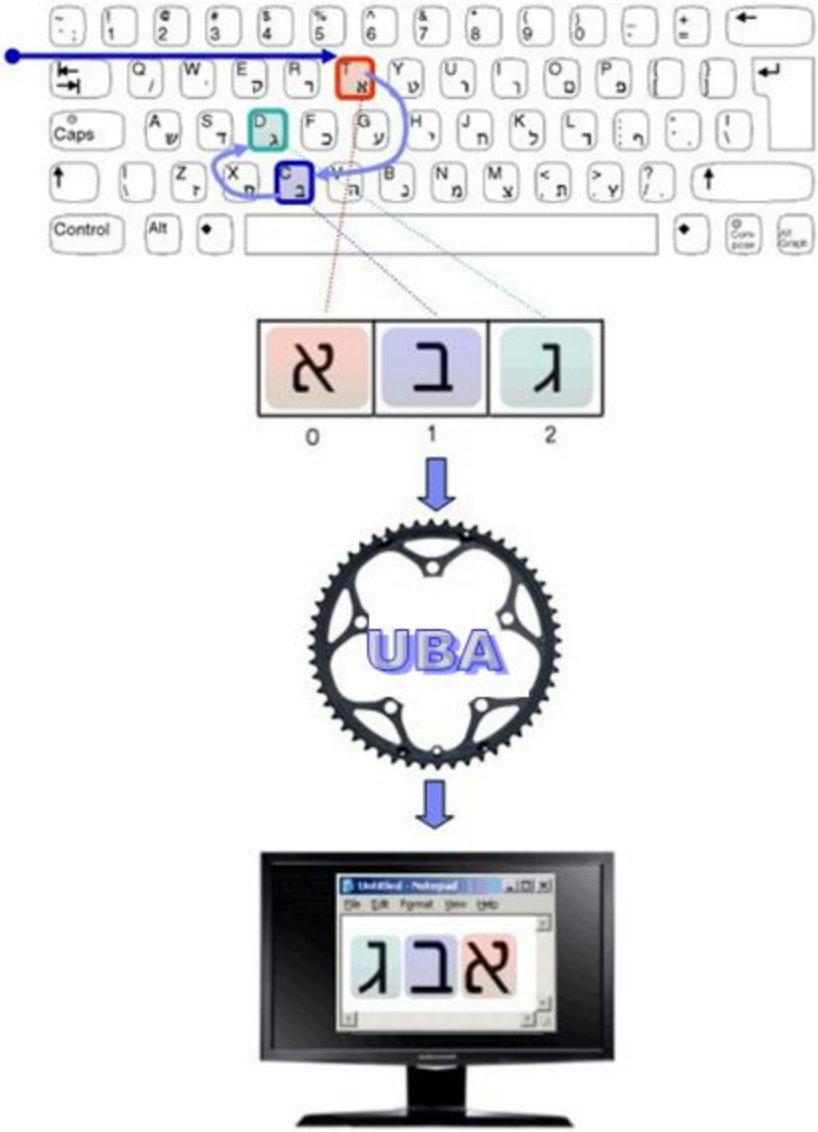
What is so special about display of bidi text as part of data having complex internal structure or syntax? To answer this question the reader should understand the mechanism used on many presentation systems for bidi text rendering. In most general terms, this mechanism is based on the Unicode Bidirectional Algorithm (UBA). Let us take the following simplified example for the sake of illustration. Assume a text buffer including the following bidi text:

גבא

On presentation systems designed for logical data (e.g., Windows), the end user will see the following text on the screen:

אבג

To achieve this result, the OS uses an implementation of the UBA and applies it on the text buffer before rendering it on the screen. For illustration, please see the figure below:



In addition, let us clarify the following concepts: bidirectional character type, strong directional character, character direction, directional run and base text direction.

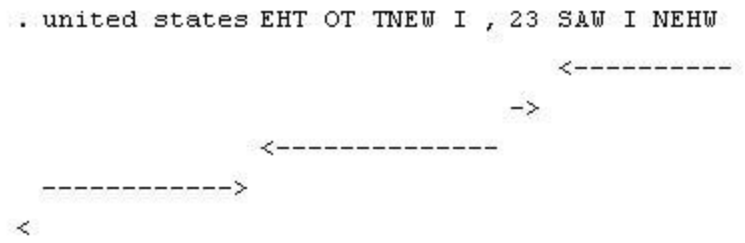
Bidirectional character type: the Unicode standard assigns to each character its own invariant "bidirectional character type" (see http://www.unicode.org/reports/tr9/#Bidirectional_Character_Types).

Strong directional character: Characters with the Left-to-Right, Right-to-Left and Right-to-Left Arabic types are called "strong directional" (or in short "strong") characters since they have their own predefined reading direction and may affect the reading direction of neighboring characters. The reading direction of characters which are not "strong" depends on their context.

Numbers are a special case: on one hand, their reading direction is always LTR, but they don't affect the reading direction of neighboring characters. Note that even numbers displayed with Arabic-Indic digits have a LTR character direction.

Character direction refers to the reading direction of a given occurrence of a character. For strong characters and for digits, the character direction is fixed. For other types of characters it depends on the context.

Directional run(s): Bidirectional text, as its name implies, may contain segments which are read from right to left and segments which are read from left to right. Consider the following example which shows a basically right-to-left sentence with a mixture of RTL words (in Arabic, Hebrew, Urdu, etc.) and LTR words (for example, in English). Upper case letters represent letters in the Arabic or Hebrew scripts. The example sentence contains 5 directional runs:



In this sentence,

- the segment " SAW I NEHW" is a right to left directional run.
- "23" is a number which is read from left to right, i.e. twenty three, it is a left to right directional run.
- " EHT OT TNEW I ," is a right to left run.
- "united states" is a left to right run.
- the final full stop by itself is a right to left run.

This why it is called "bidirectional text".

A directional run is a maximal sequence of adjacent characters which are read in the same direction, thus it is a more formal term for what was called "segment" above.

Base text direction refers to the order of directional runs in a sentence.
For LTR sentences (e.g., in English) the proper base text direction is LTR even if they include Arabic (or Hebrew, Urdu, Farsi) words, while for RTL sentences (e.g., in Arabic or Hebrew) the proper base text direction is RTL even if they include English words or numbers.
Determining if a sentence is English or Arabic/Hebrew is not always easy.
Let us review the sample sentence used above. Since this sentence is an Arabic/Hebrew sentence, it has a right to left base text direction, the directional runs must be laid out on the line one after the other from right to left, giving (with each directional run bracketed with square brackets for clarity):

[.][united states][EHT OT TNEW I ,][23][SAW I NEHW]

One should not confuse **character direction** with **base text direction**.
Character direction refers to the effective direction (which may be affected by the context) of specific occurrences of characters. It corresponds to the reading order of letters in word(s). The character direction is uniform within a directional run.
Base text direction refers to the ordering of directional runs in a sentence.
Consecutive English words form an LTR directional run even when embedded in an Arabic, Hebrew or Urdu sentence. Similarly, consecutive Arabic/Hebrew words form an RTL directional run even when embedded in an English sentence. However, the ordering of successive directional runs depends on the value of the base text direction. Examples below illustrate some typical cases.

The natural base text direction for Arabic, Hebrew or Urdu text is Right-To-Left, while for English text it is Left-To-Right. This natural text direction is inherent to the script used for the text and independent from the component direction (GUI mirroring mode) and from the user locale. Namely, even if the GUI of a product is not mirrored, bidirectional text originated by the end user (bidirectional text which the user directly or indirectly introduces and sees via the product's GUI) should be displayed according to its natural base text direction.

The base text direction of Bidi text affects its display. Let us consider how the following string (that we list in logical order, i.e. the order in which the text is pronounced) is going to be displayed when we change its base text direction:

hello world !

With Left-To-Right base text direction the string will be displayed as follows:

hello world !

With Right-To-Left base text direction the same string will be displayed as follows:

! hello world

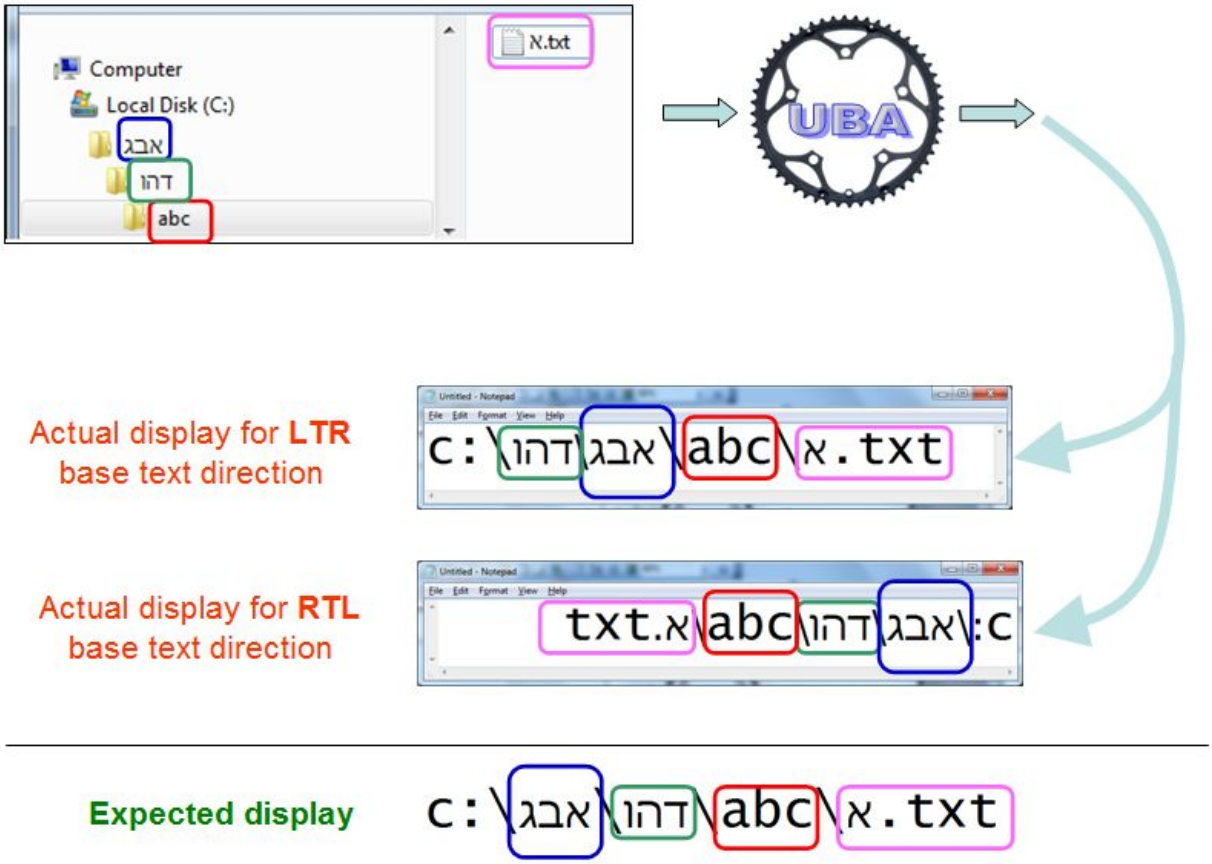
Note that in the second case the exclamation mark appears on the left side of the string.
Note also that the string will **not** be displayed as

! world hello

when the base text direction is RTL because even though the base text direction is RTL, consecutive English words always constitute a LTR directional run, so that each word is displayed LTR and the progression of words in the run is also LTR.

Plain text vs. structured text

For plain text, the mechanism described in the previous section generally provides satisfactory results. However for more complex cases the results provided by the UBA are suboptimal. The figure below illustrates default and expected display of file path including bidi characters as part of folder names.



Please observe that neither base text direction (LTR or RTL) results in a sensical display of file path in this example.

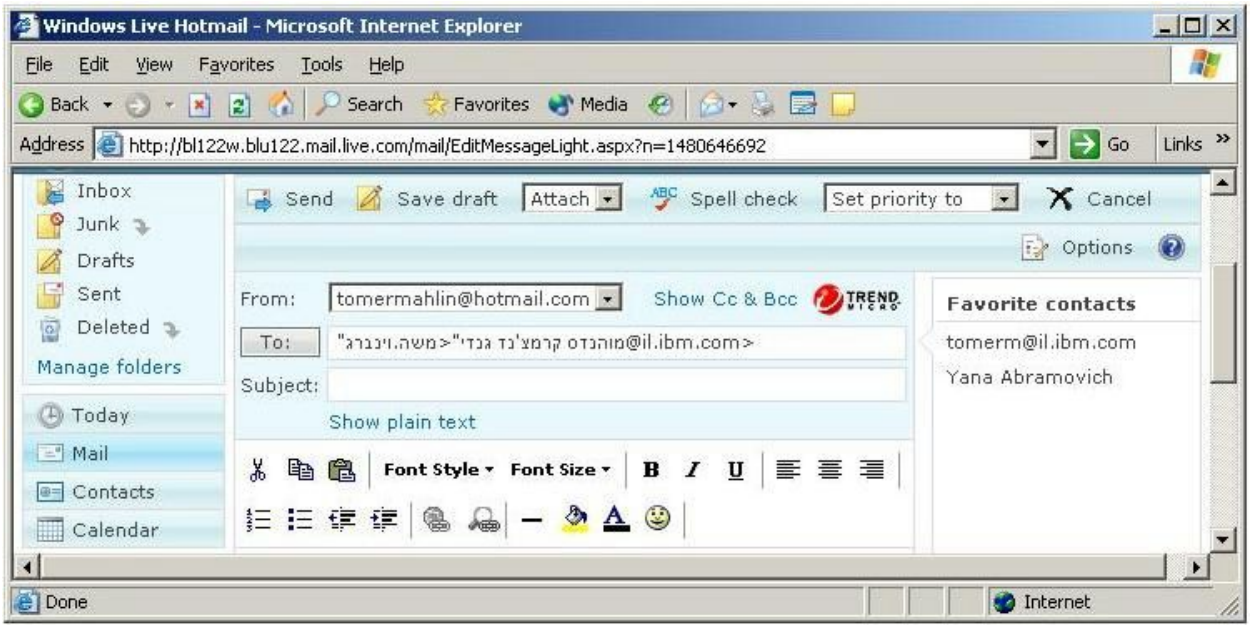
Examples of incorrect display of structured text

URL or File Path

Please see the example at the end of previous section.

Email address

Let us consider an email address typed into the To field of the Hotmail application shown on the figures below.



Notice how the email address is displayed in the To field:

Hebrew: "מוהנדס קרמצ'נד גנדי"<משה.וינברג@il.ibm.com>

Arabic: "كلام عربي" <رامي.سعيد@eg.ibm.com>

The general syntax of email address according to RFC2822 is as follows:

"[display name]"<[dot-atom]@[Internet domain]>

Note that in the **display** the different parts do not appear in the proper order. In addition the order of the atoms in the [dot-atom] portion is inverted as well. The proper display of the email address above is:

Hebrew: "מוהנדס קרמצ'נד גנדי"<וינברג.משה@il.ibm.com>

Arabic: "كلام عربي" <سعيد.رامي@eg.ibm.com>

Regular expression

Different engines support different syntax of regular expressions. In this article we use examples which are supported by Java regular expression (java.util.regex - http://java.sun.com/j2se/1.4.2/docs/api/java/util/regex/Pattern.html). Let us consider the following sample of regular expression:

([aj-ko-txyz]{1,23})|(helloworld)

This expression can be "translated" as follows: we are looking for one of two types of strings. It can be either a constant (i.e. helloworld) or a string which appears at least 1 time (but no more than 23 times) and which includes characters exclusively from the following range: a, any character from j to k, any character from o to t, x, y, or z.

When bidi characters are used the display (without STT handling) is as follows:

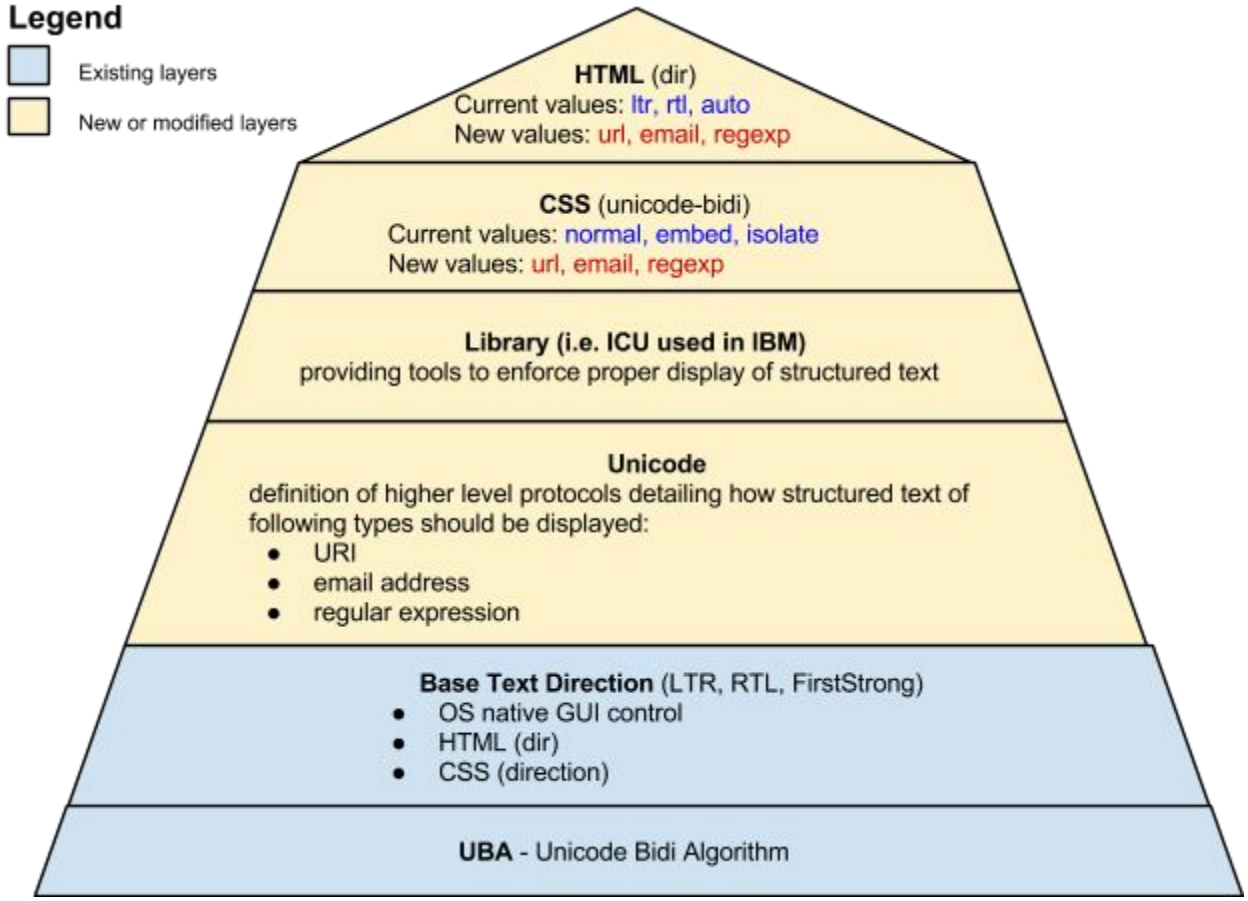
Hebrew: **([שלום)|({1,23}[פקרת-טנ-אגworld)**

Arabic: **([مرحبا)|({1,23}[كهوي-زغ-جworld)**

Obviously (based on coloring) the integrity of the regular expression is not preserved.

General approach for resolution of the problem

General approach for resolution of the problem is depicted on the following diagram.



- At the moment display of bidirectional text is guided by UBA and higher level protocols such as HTML, CSS controlling direction of text.
- The suggestion is to define a specific set of higher level protocols recommended for use on certain types of structured text (i.e. postulated in Unicode) specifying expected display of structured text including bidirectional characters. Let us call those protocols *“STT protocols”*.
- At next step, ICU (or any proprietary library provided by Google, Microsoft etc.) will provide necessary tools to assure text is displayed in accordance with STT protocol(s). For example it can provide a function(s) which:
  receives
  (a) an STT (i.e. “c:\file.txt”) and
  (b) type of STT protocol (i.e. FILEPATH)

returns

(a) text which will be properly displayed by any UBA compliant rendering engine

- At the next step CSS will provide new unicode -bidi property values associated with specific STT types. For example: unicode-bidi will support following new values: uri, email, regexp

- At the next step HTML will provide a markup associating specific text with specific STT protocol. For example: dir will support following new values: uri, email, regexp

- Finally, browsers supporting those CSS / HTML will enforce display of text using appropriate tools mentioned above (such as ICU).

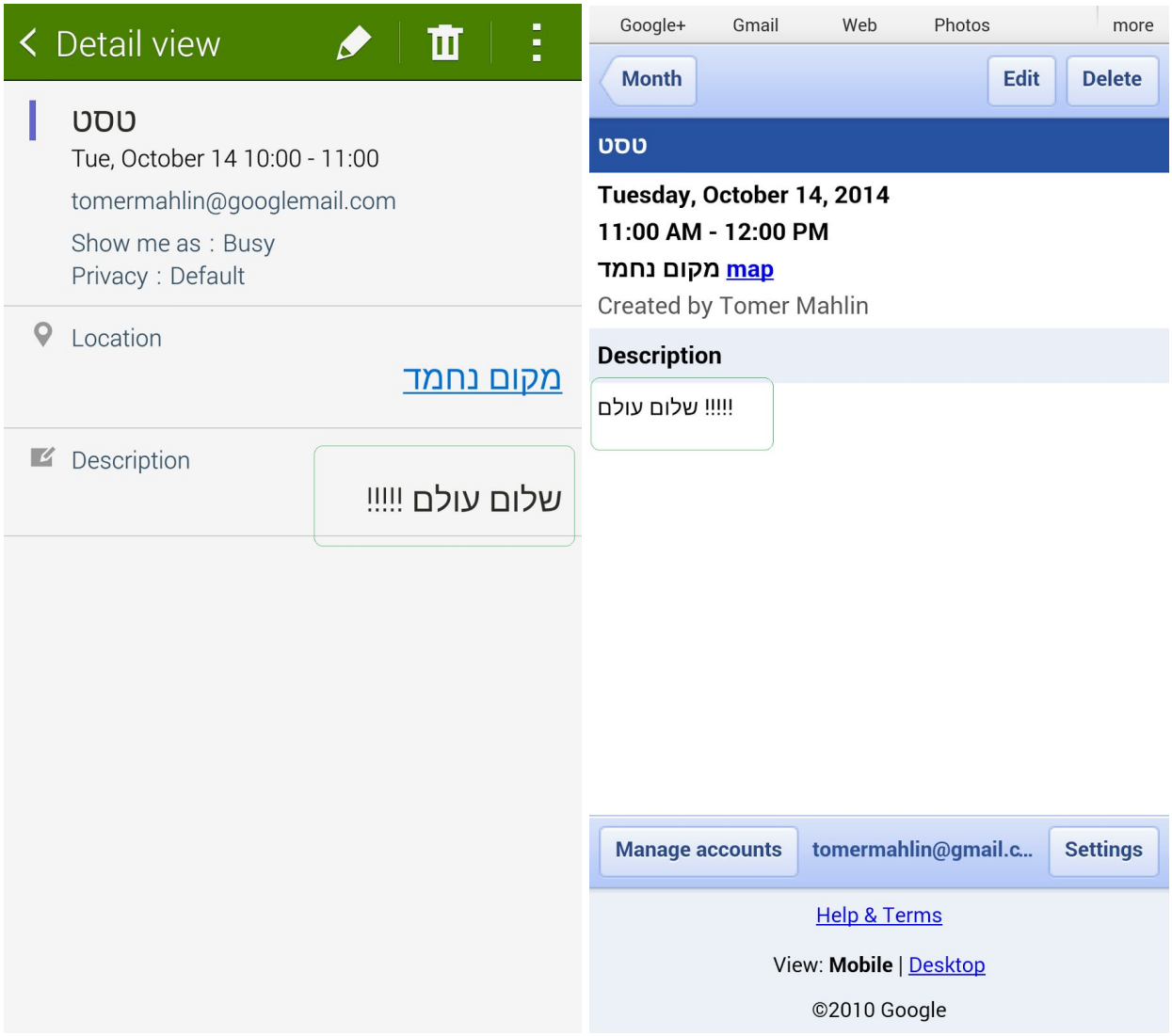
Observations and possible concerns

If problem is resolved via UCC how do we keep text safe ? Who and when remove them ?

Some of sample solutions already available in different technologies leverage UCC (Unicode Control Characters) injected into text (email, URL, regexp) to achieve expected display. If UCC are indeed used for resolution of the problem they must be used “on the glass”. In other words, they must be used for display purposes only and should not affect the content of the text. Removal of UCC is in the responsibility of the same technology responsible for rendering of text (i.e. browser, platform such as Eclipse etc.).
UCC are removed when text is retrieved from the platform for example via API or placed into the clipboard.

What is the migration plan for times at which some software will present the text properly while others won't ?

There is no migration plan. Proper display of specially formatted text is in responsibility of software implementing “STT protocols”. Some software will implement it, others will never implement it. Please observe that currently there are no migration plans even for display of old plain text. Different rendering platforms provide different out of the box rendering rules. As a result the same plain text is rendered differently on those platforms. For example images below show how description of scheduled event is shown in a native application (providing First-Strong [aka Contextual or Auto] out of the box display) and a web application (providing LTR out of the box display).



Thus there is no reason to expect a different approach for structured text.

What is the solution for plain text protocols ? For example what to do with plain text file including email address to assure it is properly displayed (i.e. in a Notepad, web browser etc.) ?

Proper display of structured text is expected only from those software in which “STT protocols” are implemented (i.e. HTML content shown in a web browser). Plain text fails to properly display mathematical expressions (even in English only). It also fails to properly display plain bidirectional text with natural text direction. This is simply because information about direction of each paragraph is not preserved by plain text protocols (as opposed to HTML protocol or any meta data based model used by rich text editors). Thus it is not expected from plain text protocols to properly display email addresses, URLs and regular expressions.

We (in Microsoft, Apple, Google, IBM etc.) are still not sure how exactly URL should be displayed.

This article is not about setting standard for STT display rules. It does not provide **any** details on STT protocols. How URL (or any other type of STT) should be displayed is subject to discussion as part of STT protocols standard review.
Instead, this article provides a significantly higher level overview of approach for problem resolution. This view does not depend on the details of STT protocols.