

## BIDI handling of Structured Text

### Authors:

Tomer Mahlin - IBM bidi Competency Center  
Matitiahu Allouche - IBM bidi architect

### 0. Terminology and Conventions

*Structured text* (a.k.a. complex expressions) - text having implied or inherent structure such as: URL, file path, email, Java code, XML, regular expression, date/time stamp etc. This term is abbreviated as STT.

The examples used in this proposal assume the following convention: 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.

### 1. The problem

Display of bidirectional text on logical platforms (e.g., Windows) 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 plain bidirectional text. However, not all bidirectional text is plain text. There are cases in which bidirectional text has a predefined syntax or an internal structure. Display of such text must follow this syntax. Unfortunately, the UBA is not aware of the internal structure of bidirectional text and does not produce satisfactory results in such cases. As a result the display of bidi text may become misleading or even incomprehensible

**Important: The problem is with display only and background related processing is not affected. However, the way the text is displayed on the screen makes it very hard to work with it (in the best case).**

### 2. What are the most common types of structured text a user can encounter?

The types of STT are broadly divided into two categories according to the level of the users.

**Regular end user** (types of STT encountered by almost every single end user)

- File path
- Email
- Regular expressions
- Date / time stamps
- Breadcrumb
- Messages with placeholders
- Concatenated text

**Developer or advanced end user** (types of STT encountered by advanced user e.g. programmer, system administrator, business analyst etc.)

- Java code
- Markup languages (e.g. XML)

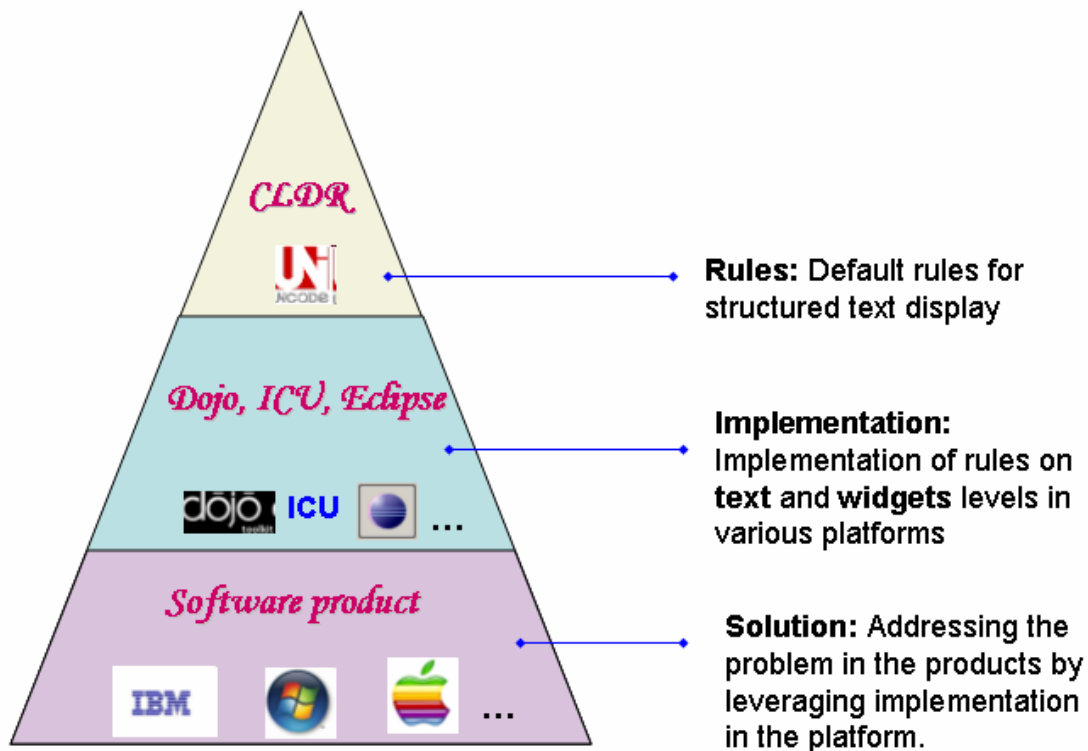
- SQL query
- Mathematical formula

### 3. Examples of expected and default display for selected types of STT<sup>1</sup>

STT type	Logical order (memory storage)	Expected display	Actual display provided by UBA
File path	c:\FOLDERA\FOLDERB\FOLDERutc	c:\AREDLOF\BREDLOF\REDLOFutc	c:\REDLOF\BREDLOF\AREDLOFutc
Concatenated text	PROJECT(3) – DESCRIPTION utc	TCEJORP(3) - NOITPIRCSED utc	NOITPIRCSED - (3) TCEJORP utc
Messages with placeholders	'*HELLO*' – 6 matches in (*TOMER*)	'*OLLEH*' - 6 matches in (*REMOT*)	'*6 - '*OLLEH matches in (*REMOT*)
URL	http://www.a.com/FOLDERB/FOLDERutc	http://www.a.com/BREDLOF/REDLOFutc	http://www.a.com/REDLOF/BREDLOFutc
SQL query	select ABC, DEFghi from SCHEMA, TABLEabc	select CBA, FEDghi from AMEHCS.ELBATabc	select FED, CBAghi from ELBAT.AMEHCSabc

### 4. What solution would we like to have?

General approach is depicted on the diagram below:



<sup>1</sup> A comprehensive and detailed report on all widely used types of STT accompanied with a lot of illustrations (showing the problem in real products) exist and can be provided upon request.

Current proposal focuses on CLDR (top of the pyramid above) layer. Any implementation of CLDR display rules for STT (pertaining to the middle layer in the pyramid above - ICU, Dojo, Eclipse etc.) should allow:

1. Preparation of string for display which will assure that its structure is preserved, This entails development of flexible set of parsers which will analyze the structure of text and will ensure proper order of tokens using available means (e.g. A) inject UCC (Unicode Control Characters) when appropriate. B) Use HTML 5 new attribute – bdi etc.). The framework should be extendable in order to allow adding new parsers (either custom or built-in) in the future.
2. Taking into consideration user preferences (default rules for STT handling): Arabic vs. Hebrew. Users in different geographies are accustomed to different rules for STT display. For example math formulas are always displayed from left-to-right for Hebrew users, while for Arabic users, they may be displayed from right-to-left.
3. Ability to leverage default rules for STT handling (defined in CLDR) and also customize them.

**This proposal and PRI 185** (<http://www.unicode.org/review/pri185/>)

This proposal addresses all types of structured text while PRI 185 focuses only on IRI type of STT.

PRI 185 achieves proper display of STT by amending UBA itself including automatic identification of STT. As opposed to that this proposal does not seek automatism. It provides standard rules for expected display of STT. The display is assured by implementation of those rules on different platforms and by leveraging this implementation in the software products. UBA is not altered in any way.

It seems that automatic identification of STT type in general is practically impossible taking into consideration the following factors:

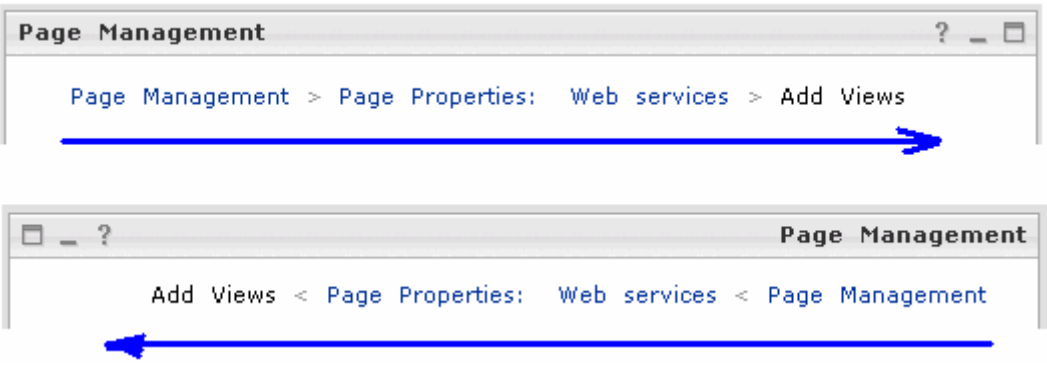
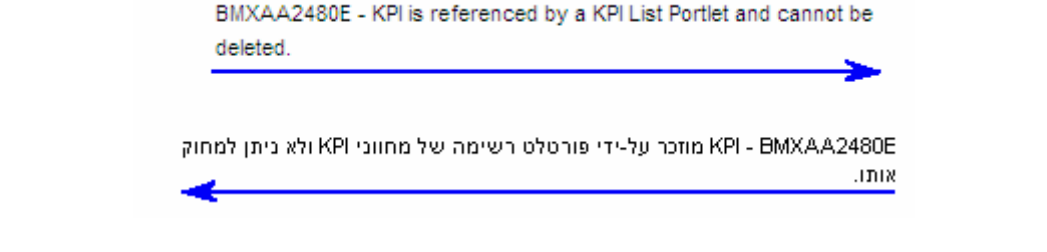
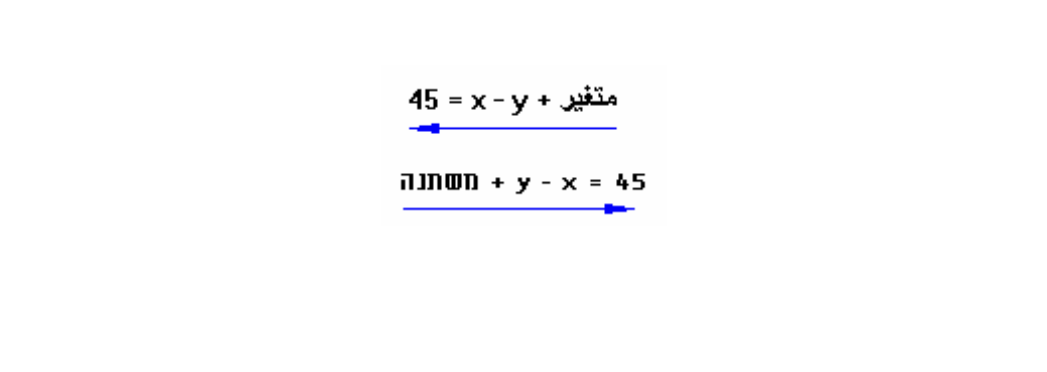
- STT can be nested (SQL query in Java code, message with file path replacing a placeholder etc.)
- Automatic identification of language to which message belongs
- Automatic identification of syntax for concatenated text

Addressing problem of STT display on the text level **only** (e.g. UBA) does not seem possible since information about factors affecting STT display (e.g. GUI direction, national preferences etc.) is not present on that level and thus can not be taken into account.

## 5. Factors affecting the display of STT<sup>2</sup>

There are different types of factors that can affect the proper display of structured text. Some types of STT, such as URLs or file paths, have a strong LTR directionality associated with them, while others have different display semantics depending on the directionality of the underlying GUI, or upon the content of the text itself. This is further complicated by the fact that the culturally accepted preferences for proper display of STT differ, even between Hebrew and Arabic users. Thus, we need to be able to define rules within CLDR to describe these factors and to be able to use that data in order to format STT properly in a given context.

In the table below you will find illustrations for some of the factors affecting STT display:

Factor	Illustrations	Comments
GUI direction		Breadcrumb runs in the same direction as the overall GUI direction. For a mirrored GUI, it runs from right to left, while for not mirrored GUI it runs from left to right.
Language of message content		Arabic / Hebrew messages should be displayed in RTL direction while English messages should be displayed in LTR direction.
National preference		Hebrew users expect mathematical formulas to always flow from left to right while Arabic users under certain conditions expect them to flow from right to left.

<sup>2</sup> A comprehensive and detailed report elaborating on all affecting factors and their impact on the expected STT display exists and can be provided upon request.

## 6. How the solution will be used ?

**Text level** (e.g. ICU case)

In most cases, developers using ICU will call an API similar to the following one:

```
prepareForDisplay(inputText, STT_Type, additional_parameters)
```

This API will inject UCC ( Unicode Control Characters ) into inputText according to STT\_Type and additional\_parameters and will return the result ready for display.

- inputText - structured text subject to display
- STT\_Type: file path, URL, email, Java etc.
- additional\_parameters - external parameters which might affect STT processing and which are not directly available to ICU (e.g. GUI direction etc.)

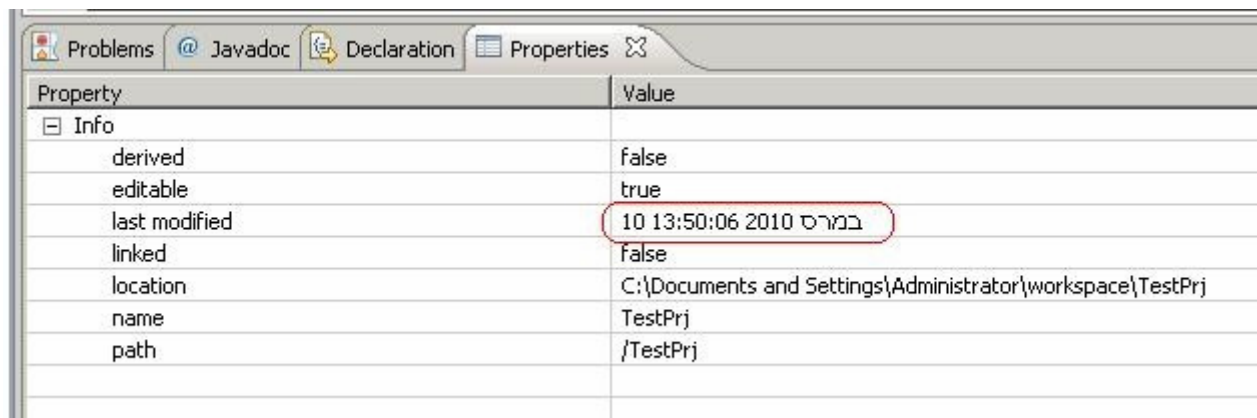
The result string will be embedded by the caller into a graphical element to render it on the screen.

ICU users will be able to either accept default STT rules (associated with locale and defined originally in CLDR) or customize them (via convenient API allowing setting / getting values for different aspects of such rules).

ICU users will be able to extend the list of parsers provided out of the box in order to address additional types of STT.

ICU will allow removal of UCCs from the text.

One additional usage of this functionality is in serialization / formatting dates / time stamps provided by ICU itself. A date / time stamp is considered structured text as well. To assure backward compatibility, ICU should not inject UCCs into date / time stamps by default. However, it can provide either a flag or an additional signature of the same function (responsible for date / time stamps generation) which will prepare them for display (by injection of UCC appropriately). This will assure proper display of date / time stamps generated by ICU. For example in Eclipse:



Property	Value
Info	
derived	false
editable	true
last modified	10 13:50:06 2010 במרחשון
linked	false
location	C:\Documents and Settings\Administrator\workspace\TestPrj
name	TestPrj
path	/TestPrj

### Widget level case (e.g. Dojo, Eclipse etc.)

At widget creation time, it should be possible to provide additional information about the type of structured text to be displayed in the widget. The following code snippet illustrates this concept:

```
Widget myWidget = new Widget(STT_type,...);
```

- STT\_Type: file path, URL, email, Java etc.

Using this information, the widget will assure proper display of structured text using various means (e.g. UCC injection).

**Important:** Injection of UCC is just one of possible approaches suggested in the previous subsection. This is almost the only way to affect UBA work when what we have in hand is text only. On the graphical library level (e.g. Dojo, SWT in Eclipse etc.) there might be more advanced techniques for achieving the same goal: proper enforcement of specific tokens' order on display level. For example:

- In Dojo we can leverage new HTML 5 attribute - <bdi>
- In Eclipse we can leverage BidiSegmentListener

## 7. What will happen during the migration period?

The migration period is the time during which some agents (e.g. programs, products) do support proper display of STT while others don't. During that period, people will be using a mixture of old and new applications. It is crucial to minimize the negative effects for interoperability of different displays of STT during this transition period.

As a matter of fact, to a certain extent we are already in the middle of this transition period. Different applications display the same STT differently. For example, MS Visual Studio based RTC client does not properly display code including bidi characters while Eclipse based RTC client (using Eclipse Java code editor) does. The same is true if we compare specific products / technologies which already have support for STT (e.g. Eclipse, Dojo) with products / technologies which don't (e.g. Swing, Flash etc.)

Since the problem is with **display only** and since resolution of the problem occurs on the glass without affecting text content, the presence of a solution on several agents has no impact on the rest of the agents. The only obvious result is that in some environments the text will be properly displayed while in others it won't. Despite that we will still be able to successfully exchange data between those agents without any effect on the data processing (e.g. Java code will compile in both environments despite the fact that it is not properly displayed in one of them).

The current situation is unbearable – the STT including bidi characters is simply incomprehensible due to distorted display. The only solution users have currently is to avoid usage of bidi characters, which is not always possible. A standardized approach and consistent platform level implementation seem to be a much preferable solution.

## 8. Proposed additions to CLDR

The following schema is proposed for addition into CLDR, most likely as sub-elements under the <layout> category. This structure defines the sets of rules in CLDR necessary to describe the proper display behavior for various types of structured text:

<!-- Top level element is associated with STT type: email, xml, java ...-->

```
<!ELEMENT sttDisplay (guiDir?)>
<!ATTLIST sttDisplay type (filepath | sql | xml | java | regexp | mathexp |
email | message | datetimestamp | concatenatedtext | breadcrumb) #REQUIRED >
```

<!-- Next level element is associated with GUI direction (or mirroring mode). -->

```
<!ELEMENT guiDir (textCondition?)>
<!ATTLIST guiDir type (ltr | rtl) #REQUIRED >
```

<!-- Next level element is associated with text content dependencies (if relevant). -->

<!-- All attributes are optional since not for all types of STT there is a dependency on the text content.-->

```
<!ELEMENT textCondition (sttDir, staticContextAlignment,
dynamicContextAlignment)>
<!ATTLIST textCondition messageLang (ar | he) #IMPLIED>
<!ATTLIST textCondition hasStrongRTLchars (yes | no) #IMPLIED>
<!ATTLIST textCondition hasArabicIndicDigits (yes | no) #IMPLIED>
<!ATTLIST textCondition hasArabicEuropeanDigits (yes | no) #IMPLIED>
<!ATTLIST textCondition firstStrongDirChar (none | ltr | rtl) #IMPLIED>
<!ATTLIST textCondition domainHasStrongRTLChar (none | yes) #IMPLIED>
```

<!-- On the lowest level we have only values derived from the conditions expressed by the higher level elements -->

```
<!ELEMENT sttDir (#PCDATA)>
<!ELEMENT staticContextAlignment (#PCDATA)>
<!ELEMENT dynamicContextAlignment (#PCDATA)>
```



## Sample data for various types of STT:

a). STT with strong directionality, such as filepath, would have the following rules in **root.xml**:

```
<sttDisplay type="filepath">
  <guiDir type = "ltr">
    <textCondition>
      <sttDir>LTR</sttDir>
      <staticContextAlignment>LEFT</staticContextAlignment>
      <dynamicContextAlignment>LEFT</dynamicContextAlignment>
    </textCondition>
  </guiDir>
  <guiDir type = "rtl">
    <textCondition>
      <sttDir>LTR</sttDir>
      <staticContextAlignment>RIGHT</staticContextAlignment>
      <dynamicContextAlignment>LEFT</dynamicContextAlignment>
    </textCondition>
  </guiDir>
</sttDisplay>
```

b). The "message" type has some variations based on the content, but the rules remain constant across locales, as follows:

**root.xml:**

```
<sttDisplay type="message">
  <guiDir type = "ltr">
    <textCondition>
      <sttDir>LTR</sttDir>
      <staticContextAlignment>LEFT</staticContextAlignment>
      <dynamicContextAlignment>LEFT</dynamicContextAlignment>
    </textCondition>
    <textCondition messageLang="he">
      <sttDir>RTL</sttDir>
      <staticContextAlignment>LEFT</staticContextAlignment>
      <dynamicContextAlignment>RIGHT</dynamicContextAlignment>
    </textCondition>
    <textCondition messageLang="ar">
      <sttDir>RTL</sttDir>
      <staticContextAlignment>LEFT</staticContextAlignment>
      <dynamicContextAlignment>RIGHT</dynamicContextAlignment>
    </textCondition>
  </guiDir>
  <guiDir type="rtl">
    <textCondition>
      <sttDir>LTR</sttDir>
      <staticContextAlignment>RIGHT</staticContextAlignment>
      <dynamicContextAlignment>LEFT</dynamicContextAlignment>
    </textCondition>
    <textCondition messageLang="he">
      <sttDir>RTL</sttDir>
      <staticContextAlignment>RIGHT</staticContextAlignment>
      <dynamicContextAlignment>RIGHT</dynamicContextAlignment>
    </textCondition>
    <textCondition messageLang="ar">
      <sttDir>RTL</sttDir>
      <staticContextAlignment>RIGHT</staticContextAlignment>
      <dynamicContextAlignment>RIGHT</dynamicContextAlignment>
    </textCondition>
  </guiDir>
</sttDisplay>
```

c). Rules for date and time stamps differ between Hebrew and Arabic, so we would have:

In **he.xml**:

```
<sttDisplay type="datetimestamp">
  <guiDir type = "ltr">
    <textCondition hasStrongRTLchars="yes">
      <sttDir>RTL</sttDir>
      <staticContextAlignment>LEFT</staticContextAlignment>
      <dynamicContextAlignment>RIGHT</dynamicContextAlignment>
    </textCondition>
    <textCondition hasArabicIndicDigits="yes">
      <sttDir>RTL</sttDir>
      <staticContextAlignment>LEFT</staticContextAlignment>
      <dynamicContextAlignment>RIGHT</dynamicContextAlignment>
    </textCondition>
    <textCondition hasStrongRTLchars="no" hasArabicIndicDigits="no">
      <sttDir>LTR</sttDir>
      <staticContextAlignment>LEFT</staticContextAlignment>
      <dynamicContextAlignment>LEFT</dynamicContextAlignment>
    </textCondition>
  </guiDir>
  <guiDir type = "rtl">
    <textCondition hasStrongRTLchars="yes">
      <sttDir>RTL</sttDir>
      <staticContextAlignment>RIGHT</staticContextAlignment>
      <dynamicContextAlignment>RIGHT</dynamicContextAlignment>
    </textCondition>
    <textCondition hasArabicIndicDigits="yes">
      <sttDir>RTL</sttDir>
      <staticContextAlignment>RIGHT</staticContextAlignment>
      <dynamicContextAlignment>RIGHT</dynamicContextAlignment>
    </textCondition>
    <textCondition hasStrongRTLchars="no" hasArabicIndicDigits="no">
      <sttDir>LTR</sttDir>
      <staticContextAlignment>RIGHT</staticContextAlignment>
      <dynamicContextAlignment>LEFT</dynamicContextAlignment>
    </textCondition>
  </guiDir>
</sttDisplay>
```

The following rules in **ar.xml** ( Arabic ):

```
<sttDisplay type="datetimestamp">
  <guiDir type = "ltr">
    <textCondition>
      <sttDir>LTR</sttDir>
      <staticContextAlignment>LEFT</staticContextAlignment>
      <dynamicContextAlignment>LEFT</dynamicContextAlignment>
    </textCondition>
  </guiDir>
  <guiDir type = "rtl">
    <textCondition>
      <sttDir>RTL</sttDir>
      <staticContextAlignment>RIGHT</staticContextAlignment>
      <dynamicContextAlignment>RIGHT</dynamicContextAlignment>
    </textCondition>
  </guiDir>
</sttDisplay>
```