L2/24-122

Linkification of URLs

Mark Davis, 2024-04-17

Working Document

This proposal is to add a section to UAX #31 Unicode Identifiers and Syntax with a specification for handling Unicode characters in linkification (aka link detection) of URLs and corresponding IRIs within plain text.

For example, with most email programs, when you paste in the plain text:

> The page https://ja.wikipedia.org/wiki/アルベルト・アインシュタイン contains information about Albert Einstein.

and send to someone else, they get:

> The page https://ja.wikipedia.org/wiki/アルベルト・アインシュタイン contains information about Albert Einstein.

Contents

Problem

Linkification (aka link detection) is the process of determining when a text substring represents a URL (more formally, an IRI, which allows for unescaped non-ASCII characters[1]). That substring can then have a link applied to it. Unfortunately, as yet there are no broadly observed specifications for linkification, perhaps because a good general solution requires expertise in Unicode characters and properties[2]. The specs for URL and IRI don't specify how to handle linkification, since they are only concerned with the structure in isolation, not when it is embedded within flowing text.

The linkification process for URLs is already fragmented, but it is amplified with the addition of non-ASCII characters added in IRIs, which often have very different behavior. That is, developers' lack of familiarity with the behavior of non-ASCII characters has caused the different implementations of linkification to splinter. Yet IRIs are very important for readability. People do not want to see the above IRI expressed in all ASCII:

- https://ja.wikipedia.org/wiki/%E3%82%A2%E3%83%AB%E3%83%99%E3%83%AB%E3%83%88%29%E3%82%A2%E3%82%A4%E3%83%B3%E3%82%B7%E3%83%A5%E3%82%BF%E3%82%A4%E3%83%B3

Thus the actual results of linkification vary wildly across implementations. For example, take the lists of links on List of articles every Wikipedia should have in different languages. When those are tested with major products, there are significant differences: any two implementations are likely to linkify those differently — terminating the linkification at different places. That makes it very difficult to exchange IRIs between products within plaintext, which is done surprisingly often — definitely causing problems for implementations that need predictable behavior.

Having consistent rules for linkification also has additional benefits, such as:

---

[1] Formally, a IRI is an extension of a URI, which encompasses both URLs and URNs. Here, we are focussed on those IRIs that correspond to URLs. Some people question the value of IRIs. But they are much more user-friendly than URLs, because they can contain readable characters in users' languages. It is far easier for the Japanese to read the IRI:

- https://ja.wikipedia.org/wiki/アルベルト・アインシュタイン

  than to read the corresponding URL:

- https://ja.wikipedia.org/wiki/%E3%82%A2%E3%83%AB%E3%83%99%E3%83%AB%E3%83%88%E3%83%BB%E3%82%A2%E3%82%A4%E3%83%B3%E3%82%B7%E3%83%A5%E3%82%BF%E3%82%A4%E3%83%B3.

  (which nobody can read).

For display of BIDI IRIs, see HL4 in UAX #9, Unicode Bidirectional Algorithm

[2] There are some guideline documents, such as UASG 010 Quick Guide to Linkification EN.

- If a system allows users to have their own user ids that end up in URLs, like https://www.linkedin.com/in/my.user.name , it can avoid user ids that have problematic linkification behavior, like trailing periods after path segments (actual problems).
- If linkification behavior becomes more predictable across platforms and applications, applications can use that knowledge to do minimal escaping[3]. For example:
    - Suppose that linkification is defined to break at unpaired parentheses. Then the following would not linkify past the):
        - (https://ja.wikipedia.org/wiki/アルベルト)アインシュタイン
    - Currently, because linkification cannot be predicted for IRIs, common practice is to exchange URLs, which gives unreadable results such as the , such as:
        - https://ja.wikipedia.org/wiki/%E3%82%A2%E3%83%AB%E3%83%99%E3%83%AB%E3%83%88%29%E3%82%A2%E3%82%A4%E3%83%B3%E3%82%B7%E3%83%A5%E3%82%BF%E3%82%A4%E3%83%B3
    - If linkification becomes more predictable, then for unusual cases it would only be necessary to escape just those characters that would need it, such as the %29:
        - https://ja.wikipedia.org/wiki/アルベルト%29アインシュタイン

The start of a URL/IRI is easy to determine when it starts with a known protocol (eg, https://).

Parts of a URL/IRI

Protocol	***Domain***	***Path***	***Query***	***Fragment***
https://	docs.foobar.com	/knowledge/area/	?name=article&topic=seo	#top

Implementations have also developed heuristics for determining the start of the IRI when the protocol is elided, taking advantage of the fact that there are relatively few top-level domains. And those techniques can be easily applied to internationalized domain names, which still have strong limitations on the valid characters. So the end of the domain name is also relatively easy to determine.

The real issues are when finding the *end* of the text past the domain name. The important parts are thus the path, query, and fragment, and can contain most Unicode characters. So the proposal focuses on those three parts.

The key is to be able to determine, given a Part in the IRI BNF (such as a ***Query***), when a sequence of characters that should terminate the IRI in linkification, even though they are valid in the specification.

---

[3] Additional characters can be escaped to reduce confusability, especially when they are confusable with URL syntax characters, such as a ? character in a path. For security implications of IRIs, see UTS #39: Unicode Security Mechanisms. For related issues, see UTS #55 Unicode Source Code Handling.

It is impossible for a linkification algorithm to match user expectations in all circumstances, given the variation in usage of various characters both within and across languages. So the goal is to cover use cases as broadly as possible, recognizing that it will sometimes not match user expectations in certain cases.

Proposal

Here is an initial proposal for discussion and refinement. At a high level, it consists of three features:

1. A way to identify when to terminate the scope for linkification based on a property that defines candidates and contexts for terminating the parsing of a URL.
    - This addresses the question, for example, when a trailing period should be counted as part of a link or not.
2. A way to identify balanced quotes and parens that enclose a URL
    - This addresses the distinction, for example, of enclosing the entire URL in parens, vs. URLs that contain a part that is enclosed in parens, etc.
3. An algorithm for doing the above, together with an enumerated property and a mapping.

One of the goals is also predictability; it should be relatively easy for users to understand the linkification behavior at a high level.

Properties

LinkTermination Property

We add an enumerated property of characters with 5 enum values: {**none, hard**, **soft**, **closing**, **opening**}

Value	**Description / Examples**
none	There is no stop before the character; it is included in the link.
	Example — letters 1. https://ja.wikipedia.org/wiki/アルベルト・アインシュタイン
hard	The IRI/URL terminates before this character.
	Example — a space 1. Go to https://ja.wikipedia.org/wiki/アルベルト・アインシュタイン to find the material.

soft	The IRI terminates before this character, if it is followed by /soft* (hard | endOfText)/
	Example — a question mark 1. https://ja.wikipedia.org/wiki/アルベルト・アインシュタイン?abc 2. https://ja.wikipedia.org/wiki/アルベルト・アインシュタイン? abc 3. https://ja.wikipedia.org/wiki/アルベルト・アインシュタイン?
closing	If the character is paired with a previous character *in the same part* (path, query, fragment), it is treated as **none**. Otherwise it is treated as **hard**.
	Example — an end parenthesis 1. https://ja.wikipedia.org/wiki/(アルベルト)アインシュタイン 2. (https://ja.wikipedia.org/wiki/アルベルト)アインシュタイン 3. (https://ja.wikipedia.org/wiki/アルベルトアインシュタイン
opening	Used to match **closing** characters.
	Example — same as under **closing**

LinkPairedOpeners Property

We also add a property that for each character with LinkTermination=closing, returns a character with LinkTermination=opening. (Also see the Issues.)

Example

1. LinkPairedOpeners(**}**) == **{**

Algorithm

This algorithm processes each final part [path, query, fragment] of the IRI in turn. It stops when it encounters a code point that meets one of the terminating conditions and reports the last location in the current part that is still safely considered part of the link. Special processing prevents paired enclosing punctuation — in the same part — from triggering a terminating condition. More formally:

Process each of the parts [path, query, fragment] as follows, where cp[i] refers to the ith code point in the part, where i is from 0 to n-1.

1. Set lastSafe to zero — this marks the last code point that is definitely included in the linkification.
2. Set closingStack to empty
3. Set the current code point position i to 0
4. Loop from *i* = 0 to *n*
    a. Set LT to LinkTermination(cp[*i*])
    b. If LT = none, set lastSafe to be *i*+1, continue loop
    c. If LT = soft, continue loop
    d. If LT = hard, stop linkification and return lastSafe
    e. If LT = opening, push cp[*i*] onto closingStack
    f. If LT = closing, set *open* to the pop of closingStack, or 0 if the closingStack is empty
        i. If LinkPairedOpeners(cp[i]) == *open*, set lastSafe to be *i*+1, continue loop.
        ii. Otherwise, stop linkification and return lastSafe
5. If lastSafe == *n*+1, then the entire part is safe; continue to the next part
6. Otherwise, stop linkification and return lastSafe

This can be optimized in various ways, of course.

TBD: rework the algorithm to cover the boundary conditions between Parts (? for query and # for fragment).

Draft Property Assignments

The following are initial assignments of properties; they should be reviewed to see where they need enhancement.

LinkTermination=Hard

Whitespace, controls, unassigned,...

- [[\p{whitespace}\p{NChar}\p{C}]-\p{Cf}]

Linkification=Soft

Termination characters and quotation marks:

- \p{Term}
- ['‘-‘ < > "“-” « »']

Linkification=Opening, Linkification=Closing

Derived from LinkPairedOpeners property

Linkification=None

Any other code point

LinkPairedOpeners

1. LinkPairedOpeners(cp) == if BidiPairedBracketType(cp) != Open then \x{0} else BidPairedBracket(cp)
    a. Bidi_Paired_Bracket

Issues

Scripts sans spaces

For scripts that don't need spaces between words, it is a bit tricky to linkify within sentences.

For example, take:

1. https://ja.wikipedia.org/wiki/アルベルト・アインシュタイン is an important page.

The URL is set off from the rest of the text. But then look at in the equivalent Japanese (TBD get example from native speaker):

2. https://ja.wikipedia.org/wiki/アルベルト・アインシュタインは重要なページです

That would not maintain a separation between the text if simply substituted for x in a phrase like "xは重要なページです" — so the linkification would go too far. One would need some kind of separator character to separate the text. That can be done with Hard characters (eg, space):

3. https://ja.wikipedia.org/wiki/アルベルト・アインシュタイン は重要なページです

Or with closing characters, such as:

4. 『https://ja.wikipedia.org/wiki/アルベルト・アインシュタイン』は重要なページです

An alternative would be having a termination between non-spacing scripts and spacing scripts. That wouldn't help with the above examples, but would help with cases like:

5. https://en.wikipedia.org/wiki/Albert_Einsteinは重要なページです

However, that complicates the behavior for little overall benefit. So the proposal does not include this.

Quotation Marks

We could add quotation marks as opening/closing, but that makes the algorithm more complicated, because the pairings are not 1:1 in natural languages. The simplest and most predictable solution is to have them be Soft.

Note also that some quotation marks appear in non-paired usage, such as RIGHT SINGLE QUOTATION MARK or APOSTROPHE, but also QUOTATION MARK as an alternative to HEBREW PUNCTUATION GERSHAYIM.

Examples:

Open(s)	Close
"	"
”	“
“ ” ”	”
❮	❯
❯	❮
«	»
»	«

Notes