

Inline Emotes Ad-Hoc Session (UTC#172)

From: Sean Stewart (with significant review and edits by *Jennifer Daniel*) on behalf of the Emoji Subcommittee (ESC)

To: Unicode Technical Committee

Date: 2022-06-27

Background

Since 2016, the ESC has explored alternative means of interchanging arbitrary, emoji-like images that can be embedded directly within Unicode-encoded text. [UTS#51 \(Section 8\)](#) discusses the high-level goal of this exploration—that future “... *implementations [of emoji] should be to support embedded graphics, in addition to the emoji characters*”. The realization of such an implementation of course is not without its challenges, some of which are also identified by Section 8 and are further underlined by other documents referenced herein.

A number of documents have explored the future of emoji encoding including but not limited to:

- **Coded Hashes of Arbitrary Images ([L2/16-105R](#))**
 - Proposes the use of guaranteed unique, SHA-256 identifiers generated from raw image data serving as custom emoji.
- **Unicode-Specified Emoji Customizations ([L2/16-008R3](#))**
 - Proposes a reasonably extensible grammar consisting of key-value pairs that associate with specific presentation modification features, all of which are encoded in Emoji Tag characters.
- **Recent QID documents (chiefly [PRI#408](#), [L2/19-082](#))**
 - Proposes the delegation of generation of unique identifiers to the Wikimedia Foundation; these unique identifiers can serve as a discoverable key in a registry for not-yet-standardized emoji concepts.

However, at the time these forward-thinking thought experiments:

- Lacked a “complete story” detailing the end-to-end technological implementation and experience¹,
- Lacked sufficient solutions to inherent challenges,
- Or, lacked vendor interest/support.

The consistency at which this topic comes up indicates a need to evolve not just to meet the demands of the public but to create a long runway for the future.

1. Davis, Mark: L2/18-203 <https://www.unicode.org/L2/L2018/18203-coded-hashes.pdf>. 2018.

Introduction

Operating at the speed of language these days is a different beast than it was a thousand years ago or even thirty years ago. Over the past ten years many apps have employed popular custom-emote experiences (e.g. Discord, Slack, etc.) which has put ESC vendors responsible for the entire operating system experience in a position where they are unable to achieve interoperability for their customers. **To do this, ESC vendors are actively invested in finding ways to evolve the underlying technology within the near future that help solve open-ended challenges of this nature.**

In addition to vendor goals, the ever-increasing customer demand for *custom emotes* motivates this effort as well: language is a low bandwidth technology for us to express our seemingly infinite prismatic feelings. While emoji encoded into the Unicode Standard are intended for a global audience, the world is made up of subcultures with their own rules, inside jokes, rituals, clothing, and practices. Language is fluid and transient but emoji are “forever”. Reconciling the methodical formal process of the Unicode Technical Committee with the ever changing looseness that is human minds interacting with each other is increasingly important in a digital first world.

ESC vendors realize that encoding all possible, recognizable concepts as emoji within the closed-ended Unicode standard is infeasible. Instead, ESC vendors wish to establish an interchange where these “custom emoji” can be embedded directly within text—like the experience already afforded by numerous chat platforms from varying vendors and software publishers.

Therefore, the principal goal of this ad-hoc session is to discuss the prototyped candidate mechanisms that will enable this enhanced emoji experience. The ESC wishes to verbally review these mechanisms before vendor resources are allocated to continued prototyping. Ultimately, we wish to converge on a standardized, cross-platform, Unicode-compatible embedded image interchange to enable this enhanced emoji experience both **throughout** and **between** adopting vendor platforms.

Session Goals

1. Discuss new candidate mechanisms enabling custom emoji presentations
2. Aggregate feedback before embarking on drafting full-fledged proposals
3. Identify potential performance challenges in existing infrastructure impacted by candidate mechanisms
4. Identify suitable image encodings and size restrictions for interchange (PNG, SVG, native font-representations, etc.)
5. Identify appropriate compression if any (zlib, LZFS, etc.)

6. Identify reasonable fall-back and “exception” experiences (emails, URLs, file names, etc.)
 7. Discuss any alternative mechanisms, improvements, or concerns not listed
-

High-Level Breakdown of Previous Investigations

Each of the previous investigations into mechanisms that enable “custom emoji” both expressed appealing, forward-thinking directions while also uncovered certain challenges including but not limited to extensibility, interoperability, stability, or fall-back experience to name a few. This section breaks down the high-level benefits and challenges belonging to the three discrete investigations that have already been published and reviewed:

1. Unicode-Specified Emoji Customizations ([L2/16-008R3](#))

Overview: Establish a relatively extensible, light-weight, key-value-based grammar using a reserved block of SMP tag characters that will supercede the existing mechanisms present in UTS#51 for gender, skintone, and hair color presentation modification.

a. Benefits

- i. Supports a moderately extensible key/value syntax
- ii. Encodes presentation customizations via Unicode-compatible tag characters reserved specifically for the emoji feature
- iii. Enables ESC to iterate on some emoji presentation features without necessarily needing additional codepoint allocations from UTC
- iv. Touches on then bleeding-edge compositional font features

b. Challenges

- i. Sequence length currently limited to 16 tag characters
- ii. Potentially confusing end-user fall-back experience when certain key-values are not supported within a platform
- iii. Some proposed keys still quantize presentation modification of identity into gradated values (e.g. 6 hair styles) to save font space with respect to glyph mappings
- iv. Certain keys and values can be associated with only certain bases, adding complexity around which emoji support which customizations
- v. Still technically requires a base “recommended general interchange” and a registry of which presentation features are supported by which vendors

2. Coded Hashes of Arbitrary Images ([L2/15-105R](#))

Overview: Establish a cryptographically-secure, SHA-256-hash-based, unforgeable, global unique identifier representing an arbitrary, emoji-sized image alongside its associated metadata that can be interchanged using a sequence of 32 SMP tag characters.

a. Benefits

- i. Enables fully open-ended emoji customization by relying on (bitmap) image technologies
- ii. Reasonable fall-back experience using existing emoji characters as bases

b. Challenges

- i. Raw image data is always in a separate location from the base emoji
- ii. Requires external registry of image mappings, pushing emoji selection challenges to a third party entity or otherwise requires vendor interoperability agreements
- iii. Provides no accessibility guidelines for reading modified emoji sequences
- iv. Provides no “full picture” of how vendors should implement the technology end-to-end

3. QID Emoji ([PRI#408](#))

Overview: Establish a reasonably static unique identifier based on keys consisting of numeric values allocated to unique concepts captured in the semantic ontologies managed by Wikidata. These identifiers (called [QIDs](#)) can be interchanged via SMP tag characters as a means to represent unique emoji concepts that remain unstandardized by Unicode.

a. Benefits

- i. Enables a discoverable means of representing numerous concepts that may never become standardized emoji
- ii. Leverages pre-existing semantic databases that assigns reasonably stable, reasonably sized globally unique identifiers that also avoids collision

- iii. Provides potential solution to challenge of enforcing appropriate base character fallback associations by citing “first-mover” advantage (passes the onus to vendors)

b. Challenges

- i. Delegates ID-to-concept mapping to the Wikimedia organization
- ii. Demonstrates ID stability concerns
- iii. Defines complexity around “visually despicable entities” (i.e. some concepts make for good emoji; others may not)
- iv. Lacks a means to modify presentation of generic concepts (i.e. a QID may exist for a generic “teacher” profession but a specific QID for a “male teacher” may not exist.)
- v. Proposes potentially complex migration from open-ended QID representation to Unicode-standardized RGI
- vi. Provides several options for fall-back experience; several of which are confusing to the end-user

Breakdown of New Candidate Mechanisms

The following mechanisms attempt to cherry-pick the most appealing aspects of these prior investigations while also addressing some of their inherent issues by establishing two discrete “custom emoji” representations: one for interchange, the other serves as a lighter-weight reference.

The candidate mechanisms are still not without their own challenges however. One major challenge remains related to the impact on string length resulting from embedding raw image data. Without establishing a third-party image registrar, it appears the only way forward to a future with “custom emoji” is to establish a transport/encoding that enables the embedding of raw image data. To limit this impact, an ancillary goal of these candidate mechanisms is to provide a secondary, more preferred, lighter-weight, reference-based variant that platforms can use in place of the full-fledged representation once the platform encounters a custom emoji.

1. External Representation of Emoji Customizations

Overview: When a document (or plainly, *text*) containing in-line custom emoji is serialized (i.e. when text must be committed to storage or when text is reasonably expected to escape a user’s device), this mechanism is used. The Base64-encoded, raw image data is serialized by means of sequences of SMP tag characters alongside a related base emoji and other necessary metadata. This data is wholly encoded directly in the plain text, Unicode-encoded string. This mechanism is *not intended* to be used everywhere a custom emoji is shown: it is instead designed to be used as a platform-to-platform interchange or during serialization only.

a. Benefits

- i. Raw image data for custom emoji always accompanies base emoji
- ii. Specification could still allow for URIs to remote resources if deemed necessary
- iii. Alleviates the need to establish third-party ID-to-image mappings
- iv. Alleviates stability challenges inherent to delegating the generation of a concept’s unique identifier to a third party
- v. Demonstrates a reasonable fall-back mechanism by requiring reasonably related emoji (sequences) used as base
- vi. Has good fallback for accessibility technologies, by requiring reasonably related emoji (sequences) used as base
- vii. Enables open-ended customization through bitmaps, not requiring presentation features be exhaustively added to Unicode

b. Challenges

- i. Impact to string length resulting from embedded bitmap images can be considerable as compared to outgoing emoji technologies
- ii. Adds complexity around converting between “external” (more expensive) and “internal” (light-weight) custom emoji representations
- iii. Stresses the notion of “plain text”

Encoding

If a modification to definition [ED-14a emoji tag-sequence \(ETS\)](#) allows for ZWJ sequences present in the RGI, embedded customizations to most emoji tokens would be possible. More encoding definitions used below are listed in [UTS#51](#). It’s also reasonable to discuss allowing customization of non-RGI ZWJ sequences, striking a

balance of open-ended customization while also being able to rely on standardized characters for accessibility and fall-back support.

Example

```
raw_image_data := [\x{E0020}-\x{E007E}]+
custom_emoji_sequence := emoji_character | emoji_modifier_sequence |
emoji_presentation_sequence | emoji_zwj_sequence raw_image_data \x{E007F}
```

Prototype Information

Custom emoji PNG image Resolution: 250px by 250px Size: 10,653 bytes (~12KB) <i>Serialized directly within the string using tag characters.</i>	
Base “toy” emoji-like character	
External representation length	Approx. 12KB ⚠️

** This information was obtained through the analysis of a working prototype that implements the External Representation. This specific toy example was created due to image licensing concerns and the author’s inability to design good iconography.*

2. Internal Representation of Emoji Customizations

Overview: Once a custom emoji has entered a user’s device by means of instant message, email, or general document deserialization, the raw image data representing custom emoji present in the document’s text are sent to an on-demand emoji image cache maintained by the platform’s operating system. A lighter-weight, time-based GUID is then assigned to the custom emoji, and in place of the raw image data, this GUID can subsequently be used as a reference to the cached custom emoji. The GUID, similarly, is represented by sequences of SMP tag characters overall limiting the footprint of custom emojis by treating their raw image data as “singleton” resources throughout the operating system. This mechanism can only be used within the confines of a user’s device: once a custom emoji is expected to leave the device, the more expensive, external representation mechanism must be used.

a. Benefits

- i. *Mirrors all benefits of the External Representation above*
- ii. Presents a smaller footprint by using shorter identifiers that reference singleton instances of the raw image data as opposed to embedding the raw image data everywhere
- iii. Slightly relieves the stress on the notion of “plain text” inherent to the “external” representation

b. Challenges

- i. Adds complexity around converting between “external” (more expensive) and “internal” (light-weight) custom emoji representations
- ii. Requires a novel service maintained by platform that caches emoji image content and assigns unique identifiers

Encoding

Similar to the “External Representation” above, this mechanism will make use of emoji tag-sequences ([ED-14a](#)), but instead these tag sequences will encode the 128-bit (16-byte) time-based GUID associated with the singleton raw image data stored in a device-specific cache. Instead of the raw image data accompanying the base emoji sequence, a “pointer” (the GUID) is used instead to minimize the impact of exchanging raw image data.

Example

```
image_uuid_pointer := \xE0003 [\xE0020]-\xE007E]{32}
custom_emoji_sequence := emoji_character | emoji_modifier_sequence |
emoji_presentation_sequence | image_uuid_pointer \xE007F}
```

The example will require more control / signal characters to distinguish the raw image data from a UUID reference (as examples). So more thought must be given to the example; it however suffices for the ad-hoc discussion to use U+E0003 to delimit the beginning of a reference.

Prototype Information

Custom emoji PNG image Resolution: 250px by 250px Size: 10,653 bytes (~12KB) <i>Stored in some sort of highly-available image cache in the platform.</i>	
Base “toy” emoji-like character	
External representation length	Approx. 144 bytes (36 UTF-32 chars) ✓

Other Requirements

Custom Emote Image Cache

In order to alleviate the size-impact of transmitting embedded image data within text, some sort of operating system service (daemon) that would ideally cache the image data for these emotes. This same service would assign guaranteed unique identifiers that serve as references (or pointers) to this singleton data, and when queried would provide the original raw image data so that operating systems can render the customizations.

This type of service is a new infrastructure that would likely not currently exist in Unicode-compliant platforms.