

**Re: Future Unicode Emoji Options**  
**From: Mark Davis, Emoji Subcommittee**  
**Date: 2021-04-20**

---

This document lists some problems with the current approach to encoding emoji, describes some alternative strategies that we could follow for future versions of Unicode Emoji, and makes a recommendation for choosing among those alternatives.

Note that all except for the [QID Emoji](#) option would have essentially no impact on end users of emoji. The QID approach would allow companies to encode additional valid (“standard-conformant”) emoji that do not have to be approved by the Unicode Consortium. There would still be a standard Unicode [RGI](#) set (Recommended for General Interchange) expected to be supported by all major platform companies.

## Problems

### 1. Issue 1: Lag Time

There is a long lag time between when we finalize emoji and when all companies can deploy them. While we have taken steps to line up better with the release schedules for companies, we still need to have a substantial lead time for emoji. A key issue is that new Unicode characters need to have the appropriate properties prepared as part of a Unicode release; otherwise the properties will be incorrect, and the characters will misbehave (for example, there can be line breaks in the middle of an emoji!).

Ideally, a company could release emoji simply by updating fonts and input (virtual keyboards and/or character pickers).

### 2. Issue 2: Single Source

Some people see the Unicode process as a barrier to getting new emoji. They are not satisfied with the alternatives, (GIFs, etc) and would like to see more flexibility in the process, adding the ability for third parties to supply additional emoji. The goal here would be a mechanism that would allow any company to use well-defined emoji without waiting for the Unicode Consortium.

## Recommendation

The ESC recommends that the UTC allow for a period of open discussion during the meeting about the different options, then the ESC modifies the proposal based on a consideration of the feedback, and presents a recommendation for a choice of strategy at the next UTC meeting.

## Strategies

We have discussed various options for dealing with the customization of emoji for many years, at least back to 2015. For more detailed history, see the documents in [References](#). The following are the chief contenders. Note that some of these options could be combined together.

In this context, it is important to remember that most emoji are not emoji *characters*, but rather emoji *sequences*. Two of the options replace all the encoding of new emoji characters, by always using

the encoding of new emoji sequences instead. There is of course the *status quo* option: We keep adding emoji characters with the new process (Encode ~30 emoji, focused on interoperability and broad, globally relevant concepts.)

---

There are two alternative options listed below (A1 and A2) for handling Issue 1: Lag Time. That is, either one can handle the issue, but you wouldn't need both.

### **A1. Reserved Emoji (RE)**

*This strategy is designed to deal with Issue 1: Lag Time. There is no change necessary to the Unicode emoji process.*

*There is little to no visible impact on end users of emoji.*

**The basic idea is that we define a certain set of *unassigned* characters to already behave as emoji, in terms of line break, other segmentation, etc. For any Unicode release, we only assign new emoji out of that set. That ensures that in Unicode version N, we only use characters that will work (in terms of processing) in implementations of Unicode N-1.**

In more detail, we take a set of 1024 reserved characters, and give them most default properties which are the same as the emoji for ELEPHANT (except they are still Cn (=unassigned) and have no name). We simplify the emoji EBNF so that any Emoji=true could be an Emoji\_Modifier\_Base.

We then put in place the following policies:

1. A release can only add emoji characters if they were Emoji=true in the ***previous release***.
2. Whenever we assign characters, we extend the Reserved Emoji code points as needed so that there is enough room for future versions.
3. Applications may need some modification so that a character with Emoji=true is parsed as emoji in line break and other segmentation, despite being unassigned.
4. The set of valid RGI modifier sequences is determined by the emoji sequence data files, instead of the Emoji\_Modifier\_Base property.

---

### **A2. Registered ID Emoji (RID Emoji)**

*This strategy is designed to deal with Issue 1: Lag Time. There is no change necessary to the Unicode emoji process.*

*There is little to no impact on end users of emoji.*

**The basic idea is that we never add new emoji characters. Instead, we add new emoji tag sequences. That ensures that in Unicode version N, we only use characters that will work in implementations on Unicode N-1: since there are no new emoji *characters*, we never worry about an implementation having the wrong properties. (Of course, we can also add ZWJ sequences and modifier sequences to the RGI set, as before.)**

In more detail, an RID is a form of Tag Sequence (see [UTS #51](#)). We define the format as follows:

<BASE><TAG-SEQ64><END>

The UTC would assign a TAG sequence using the BASE for the sequence according to the best fallback. The TAG-SEQ64 would be in ascending order for that BASE, using base 64 to get shorter tags (2 tags gets us 64 emoji per base, 3 tags get us an additional 4096 emoji per base, and so on). Thus a

wire-haired dachshund could use a DOG as a base, and a tag-seq64 sequence of TAG ZERO. In the notation of [UTS #51](#), this would be 🐕 0♦. if the next assigned RID with a base of DOG were great dane, it would be 🐕 1♦, and so on. (In some cases, there would be no obvious BASE; those would get a non-obvious base.)

We also change the EBNF for emoji so that Tag Sequences can be part of a ZWJ Sequence, and can act as Emoji\_Modifier\_Base characters. The set of valid RGI modifier sequences is determined by the emoji sequence data files, instead of the Emoji\_Modifier\_Base property. With these changes, the RID sequences behave like an emoji character as far as software is concerned.

New RID emoji would only be allocated if they are to be part of the RGI emoji, just like new emoji characters are currently allocated only if they are to be part of the RGI emoji.

---

## B. QID Emoji

*This strategy is designed to deal with both Issue 1: Lag Time and Issue 2: Single Source. There is no change necessary to the Unicode emoji process.*

*This option could have an impact on users. This option has no effect on the set of Unicode RGI (recommended for general interchange) emoji per year. But it would allow companies to encode additional emoji that do not have to be approved by the Unicode Consortium. The set of valid emoji expands to represent arbitrary entities, with new Wikidata QIDs are added over time. Thus companies could support a wider range of Unicode emoji than the Unicode RGI set, and interchange them with other platforms that supported the same set.*

*In essence, this would allow individual companies to quickly deploy and experiment with custom emoji characters, but in a manner that avoids encoding conflicts. Other companies can quickly jump on board if usage skyrockets. The resulting comparative usage data can provide strong evidence for integrating the emoji into the RGI set, making the most popular emoji more widely available.*

*However, whether or not this would have an impact on users would depend on whether significant companies deployed these additional QID emoji.*

**The basic idea is that we leverage the QIDs from Wikidata, where any QID tag sequence would be a valid emoji. So any entity (such as a [wire-haired dachshund](#)) that has a Wikidata QID corresponds to a valid tag sequence. To solve the Lag Time problem we either only have QID, or we also allow RE or RID. We can also add ZWJ sequences and modifier sequences to the RGI set, as before.**

The format becomes:

<QBASE><QID-SEQ><END>

QBASE is limited to a single emoji. It should be a new Emoji character that is reserved for use as a base, since then people would know if they saw the 'bare' character that it meant "missing emoji image"; perhaps something like ♦ (U+FFFD REPLACEMENT CHARACTER), but colorful. A fallback could be a very low-frequency existing emoji, like .

We would also change the EBNF for emoji so that Tag Sequences can be part of a ZWJ Sequence, and can act as Emoji\_Modifier\_Base characters. The set of valid RGI modifier sequences is determined by the emoji sequence data files, instead of the Emoji\_Modifier\_Base property. With these changes, the QID sequences behave like an emoji character as far as software is concerned.

The UTC would still determine which valid emoji were RGI (recommended for general interchange), and include QIDs in that set. Thus there are two levels here:

1. There is plain QID, which is similar to PUA but:

- a. a QID must be an emoji
    - i. an emoji appearance (square shape, colored independent of font color) and
    - ii. emoji behavior (in line-break, bidi, etc.)
  - b. its intended semantics are constrained by the associated Wikidata QID entry.
2. There is the possible addition of a QID to the RGI emoji set
    - a. This adds availability across major vendors, and supplies a representative glyph, and name (for accessibility), and search keywords.

There are two changes from [L2/19-082 QID Emoji Proposal](#), based on feedback

- Remove the line: “Alternatively, wide actual usage would be very strong evidence for encoding it as a regular Unicode emoji character. If we decide to provide for this, we would add an emoji data file with a mapping from defined emoji (or emoji sequences) to the corresponding original QID emoji sequence.”
- We remove the option of having multiple QBASE characters; there is only a single QBASE character. The “Appearance” section on the last page is also irrelevant.

Note: The long-term expectation for Emoji has been that they would eventually be replaced by direct exchange of images (GIFs, PNGs, etc.). That is, most interchange of text would be via interoperable protocols that would allow the inline attachment of images. See [Longer Term in UTS #51](#). The alternative to an approach like QID would be to accept that Unicode emoji would continue as a “single source”, and trust on the eventual replacement by rich protocols.

---

## References

The following are older documents submitted to the UTC that discuss various approaches for emoji customizations.

- Discussions of QID
  - [L2/19-082 QID Emoji Proposal](#) (2019)
  - <https://www.unicode.org/review/pri408/> (2019+) Collected public feedback
- Discussions of using hash codes sequences for images
  - [L2/18-251 Re: Coded Hashes of Arbitrary Images](#) (2) (2018)
  - [L2/18-203 Re: Coded Hashes of Arbitrary Images](#) (2018)
  - [L2/16-379 Feedback on Coded Hashes of Arbitrary Images](#) (2016)
  - [L2/16-105 Coded Hashes of Arbitrary Images \(revision\)](#) (2016)
  - [L2/16-105 Coded Hashes of Arbitrary Images](#) (2016)
- Discussions of customized emoji
  - [L2/16-010 Customized Emoji Tag Registration](#) (2016) discusses ideas for a future tag registry mechanism; this is the most speculative of the documents.
  - [L2/16-009 Unicode Customized Emoji Framework](#) (2016) describes the overall motivation, goals and framework for Unicode Customized Emoji (UCE).
  - [L2/16-008 Unicode Specified Emoji Customizations](#) (2016) describes three specific tag formats within that framework: U and V tags specified by the Unicode Consortium, and X tags for private use.
  - [L2/15-252 Unicode Customized Emoji \(UCE\) Proposal](#) (2015)
  - [L2/15-207 Customized Emoji](#) (2015)

---

## Image Transport

Just for completeness, it is worth mentioning that one of the early formats we considered for customizable emoji was simply encoding the emoji image inline using a sequence of special characters. It would work like TAG sequences, where a program that didn't understand the format would simply display a BASE character instead of the image. This is the "[8. Tagged Encapsulated Graphics](#)" option from back in 2015. It would require recipients that display emoji be upgraded to support the format, and we'd need to decide on the exact format (eg, SVG or PNG).

The advantage of this approach is that the image would be self-contained. Once components supported this format, someone could select a custom emoji in a virtual keyboard, causing it to enter into a message program as a sequence of Unicode characters, and be transported to a recipient, which could then display it. No font needed.

The big disadvantage would be the size. Any encoding that was using Unicode characters *and* was transparent to unaware implementations has a substantial increase in size over the base image size, since the format would need to be expressed with TAG characters or some new 4-byte Unicode characters. So simple emoji could be 10KB or so per image. For many purposes, this would be an extravagant use of memory. However, for common messaging programs, where people frequently attach images, memes, and even video, the memory consumption for such a custom emoji might be acceptable.