Think-aloud Study of MessageFormat 2

Shun Kashiwa*[1], Samantha Prestrelski†[1], and Michael Coblenz‡[1]

[1]UC San Diego, La Jolla, CA

January 22, 2025

Abstract

Internationalization and localization are critical for making software accessible to global audiences, yet their implementation often poses challenges due to linguistic diversity and complexity. MessageFormat 2 (MF2), a successor to MessageFormat 1.0, aims to overcome these challenges by offering enhanced expressivity and usability. This study evaluates MF2's usability and effectiveness through a think-aloud user study with software engineers and translators. Participants were tasked with comprehension, writing, and translation tasks using MF2, and we conducted a user study with ten participants. The results reveal that MF2's syntax is generally approachable for both software engineers and translators, though the two groups provided distinct feedback. Software engineers reported limitations in MF2's expressivity and spent time navigating its functions and parameters to compose messages. Translators, on the other hand, found the `.match` syntax challenging but faced no difficulty translating existing MF2 messages. In addition, MF2 exhibited shortcomings in handling certain locale-specific linguistic rules, such as Turkish suffixes and Danish ordinal forms. Overall, our findings suggest that MF2 holds promise as a standard framework for defining translatable messages. Expansion of the default function registry, comprehensive documentation, and the development of supporting tools such as GUI editors can foster its broader adoption.

1 Introduction

Internationalization (i18n) and localization (l10n) are critical processes in making software accessible and usable for people around the world. By enabling applications to adapt to diverse languages, cultures, and regions, i18n and l10n improve accessibility and ensure that software resonates with its users. However, implementing these processes effectively poses significant challenges, especially given the linguistic complexity of natural languages. Grammatical rules, such as pluralization and gender-specific constructs, vary widely and require precise handling to ensure correctness and inclusivity in translated messages.

To address these challenges and avoid re-implementing infrastructure for localization, standards have been developed to streamline the process. One of the most widely used is ICU (International Components for Unicode) by The Unicode Consortium [3], which provides libraries for i18n. A key component of ICU is *MessageFormat*, a syntax for writing localizable messages. Despite its popularity, MessageFormat 1.0 has faced criticism [7] for limitations in expressivity and usability. In response, the Unicode Consortium has been developing its successor – *MessageFormat 2* (MF2). As of December 7th, 2024, MF2 is included as a "final candidate" in the latest release of the Locale Data Markup Language (LDML) specification [6]. This means that the stability policy is not yet in place, allowing feedback to inform potential adjustments to various aspects of the language specification, with an emphasis on preserving backward compatibility whenever feasible.

Localization is particularly important on the web, where applications often need to serve diverse global audiences. Recognizing this, TC39 – the committee responsible for ECMAScript (JavaScript) – has been adding a number of APIs to the `Intl` namespace. The language now supports formatting

*skashiwa@ucsd.edu
†sprestrelski@ucsd.edu
‡mcoblenz@ucsd.edu

Task ID	Comprehension Tasks	Description
C01	Function Annotations - Number	Warm-up
C02	Function Annotations - Integer	Test behavior of integer function
C03	Function Annotations - Currency	Test behavior of currency function
C04	Valid Options	Test pattern recognition for options and functions
C05	Variables - Inputs	Test implicit parameters
C06	Variables - Declarations 1	Test local declaration overriding input parameters
C07	Variables - Declarations 2	Test duplicated declarations
C08	Matching - String	Test simple string matching
C09	Matching - String Numbers	Test matching with numbers and strings
C10	Matching - Ordinals	Test behavior of ordinal select

Table 1: Summary of Comprehension Tasks

numbers, dates, lists, etc., without external packages. Yet, the language lacks compiling dynamic messages, and there is a proposal to add `Intl.MessageFormat` for rendering MF2 messages [2]. This effort requires careful consideration: ECMA prioritizes backward compatibility, and introducing a new API entails significant responsibility. Once added, revising or removing it becomes difficult, if not impossible. To ensure that MF2 meets the needs of web developers and avoids potential pitfalls, TC39 has requested user studies.

In this project, we conducted a think-aloud user study to evaluate the design of MessageFormat 2 (MF2). Our study targeted two key user groups – software engineers and translators – and involved nine interviews. Section 2 outlines our evaluation methodology, including task design, and Section 3 presents the key findings of the study.

2 Method

2.1 Task Design

We are interested in two groups of individuals who work on internationalized web applications:

1. Software engineers: These are the individuals responsible for implementing the web UI. They write user-facing messages to match a specific design within the web application, typically in the primary language (likely English).

2. Translators: These individuals receive message templates from software engineers and translate them into other languages.

We aimed to evaluate each group's performance in reading and writing (including translating) MF2 messages. Both readability and ease of writing are critical aspects of the language. Readability is essential because individuals may encounter MF2 messages in existing projects without prior training, and they should be able to understand how these messages are rendered. The ability to write new messages effectively, on the other hand, ensures that people can efficiently create new messages with access to documentation. We designed the comprehension section and writing/translation sections to assess readability and ease of writing, respectively.

The comprehension section consists of ten multiple-choice questions asking participants to select one of three to five options. Participants are not allowed to use any external resources and are instructed to answer the questions to the best of their knowledge. Table 1 presents the list of comprehension tasks and their objectives.

To assess readability, we prepared two sets of tasks: one for software engineers and the other for translators. Software engineers' tasks (writing tasks) focus on writing MF2 messages from scratch. We gave participants specifications of the messages and asked them to write MF2 messages that satisfied them. We provided a test suite for each task with different input parameters, and the participants can move on to the next task once their solution passes all test cases. Figure 1 shows a screenshot of the interface provided to participants of this group, including the task description, input-output specifications, and the test suite results. Table 2 summarizes the writing tasks and features of the language required for completion. In this section, the participants were allowed to use messageformat.dev,

which is an unofficial documentation of MessageFormat 2[1]. We selected this as a reference as it covers all aspects of the language tested in this study and is more readable than the original specification. While this website is an unofficial website created by an individual, there are plans to transfer it to the Unicode Foundation as an official website. We believe that developers will refer to such a document when working on MessageFormat 2. We recommended the quick start page as a general reference. For tasks requiring features not covered in the quick start, we provided additional links to resources that explain those features.

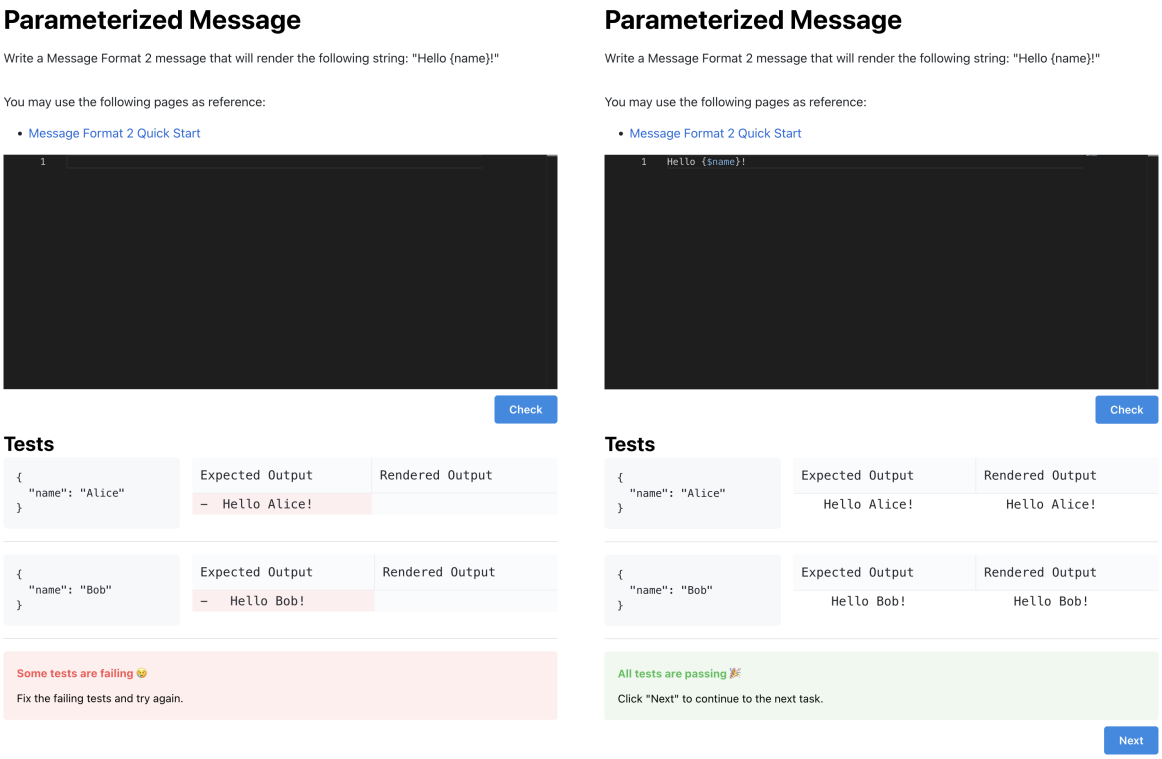

Figure 1: Sample failing and passing test suite

On the other hand, translators' tasks (translation tasks) focus on *translating* existing MF2 messages. We show the expected answers of the software engineers' tasks and ask translators to translate messages into their locale. Since there is no single translation and it is difficult to tell the correctness of the translation, we do not provide a test suite for translation tasks. Instead, we render the translated message with multiple input parameters and asked the participants to confirm their translation is reasonable before moving to the next tasks. Figure 2 shows a screenshot of the interface provided to participants in this group, including the MF2 message in English and previews.

[1]We deployed a fork of the website and used it to keep the content consistent throughout the study.

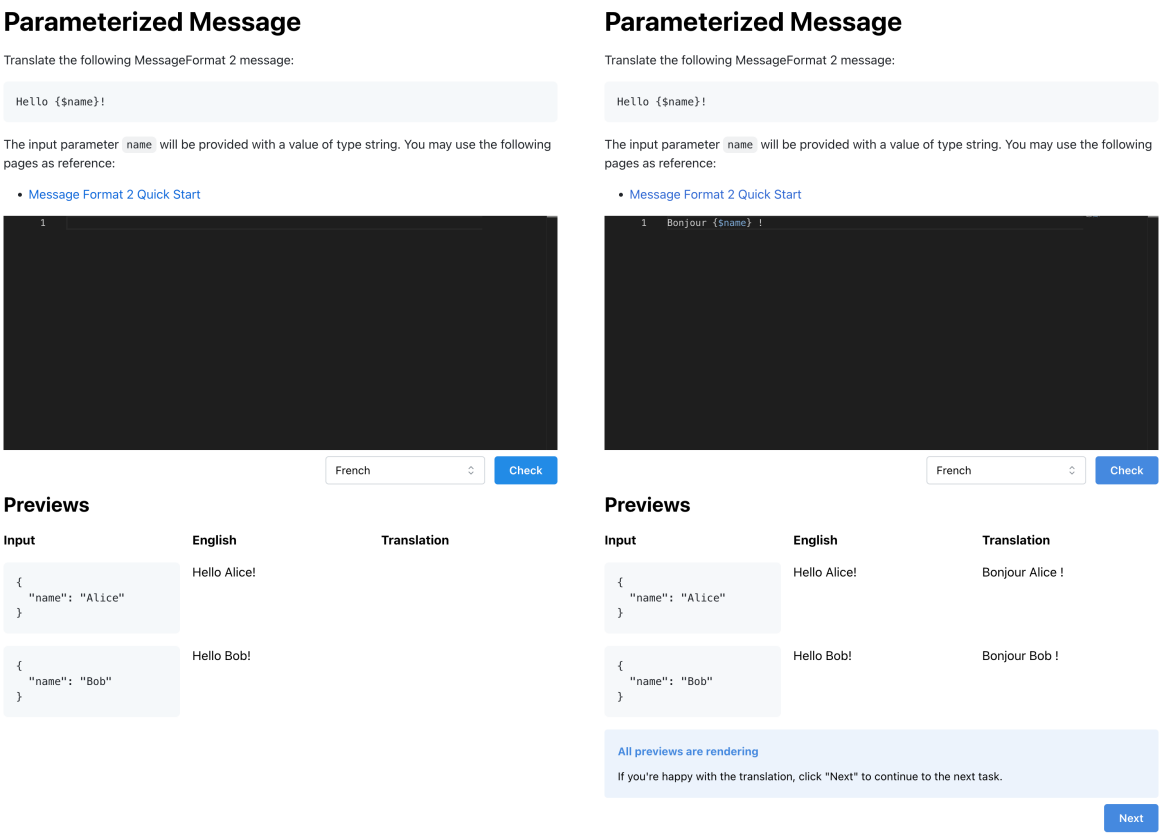

Figure 2: Sample translation render and confirmation

Task ID	Writing Tasks	Description
W01	MessageFormat 2 Message	Simple message
W02	Parameterized Message	Simple message with a placeholder
W03	Pluralization	Number with pluralization
W04	Number Formatting - Percentage	Format a number with 'percent' style
W05	Select - string	String matching on gender
W06	Select - ordinal	Number matching for ordinals
W07	Select - match at once	String matching on two variables
W08	String Literals	String matching with literals
W09	Formatting - Date	Test usability of date functionality
W10	Formatting - Timestamp	Test usability of datetime functionality
W11	Formatting - Match	Test date formatting in a non-traditional format

Table 2: Summary of Writing Tasks

After the writing/translation section, we concluded the study with the survey section. We asked the participants to rate the difficulty of the previous two sections and comment on the challenges and potential improvements to the language.

We developed *MessageFormat Arena* – a web application designed for participants to complete the specified tasks. The app runs in web browsers, enabling convenient remote interviews and allowing participants to use their own computers. For writing and translation tasks, MessageFormat Arena incorporates a simple text editor with syntax highlighting and error reporting features, powered by mf2-tools [1].

2.2 Recruitment

As a proxy for software engineers, we recruited six UC San Diego graduate students (SE participants). Each student was screened for experience writing programs with user-facing interfaces, such as web apps and smartphone apps, as well as previous experience with internationalization libraries or frameworks. While most participants have worked with web or mobile apps and some have worked on localization, none had experience with MF2. For anonymity, these participants are referred to as SE1 through SE6. Participants were given the option to choose between an in-person or virtual interview via Zoom. Three interviews were conducted in person, while the remaining three took place on Zoom. Graduate students received $20 gift cards as compensation for their time.

We interviewed four translators sourced from a contact at the Mozilla Foundation (TR participants). Each translator was screened for years of experience, the types of projects worked on, and previous experience with template languages. These translators have more than five years of experience in translating and worked on translating open-source Mozilla projects like Firefox and Thunderbird. None of the participants had experience with MF2. These participants are referred to as TR1 through TR4. Their working languages are Danish, Turkish, Traditional Chinese (Taiwan), and Portuguese (Brazil), respectively.

2.3 Interview Process

Each interview lasted approximately one hour. Participants first completed a consent form before being asked to open a link to MessageFormat Arena. Regardless of whether the interview was conducted in person or via Zoom, participants shared their screens on Zoom. They then followed the instructions provided on MessageFormat Arena and worked on a series of tasks. We asked the participants to verbalize their thinking throughout the study.

To ensure participants did not get stuck, we monitored their progress. If a participant had not made progress on a task for three minutes, we intervened with hints, typically directing them to the relevant section of the documentation.

3 Results

3.1 Comprehension

Translators found `.match` puzzling C08 asked the participants to predict the result of the following MF2 message when `val` is set to `"foo"`.

```
.input {$val :string}
.match $val
foo {{Foo}}
bar {{Bar}}
*   {{No match}}
```

The string `"foo"` matches the first branch, so the message will produce `"Foo"`. All six SE participants saw connections between the `.match` syntax and similar programming concepts, such as pattern matching in functional programming, switch statements, and regular expressions and selected the correct answer. In contrast, two translators (TR1 and TR2) struggled with this question. They didn't understand how the matching variable is used. TR1 guessed the correct answer but TR2 gave up and submitted a randomly selected answer. Fortunately, it didn't affect how they approached translation tasks – all translators correctly understood that they needed to translate each branch.

"Ordinal" is confusing C10 asked participants to predict the result of the following MF2 message when `num` is set to `3` in English. This message involves ordinal categories (e.g., 1st, 2nd, 3rd, 33rd, 111th) in combination with the `.match` syntax.

```
.input {$num :number select=ordinal}
.match $num
one  {{You are {$num}st}}
```

```
two  {{You are {$num}nd}}
few  {{You are {$num}rd}}
*    {{You are {$num}th}}
```

The number `3` belongs to the `few` category in English, so the third branch will render "You are 3rd." Multiple SE participants commented that they did not know what the ordinal keyword meant and were not sure what the output would be. The ordinal selector for the number function uses the `few` key to denote 3rd, which SE2 and SE3 commented on. SE2 stated "I assume this is meant to print third, but I'm not sure how much 'few' exactly is." SE6 and TR3 assumed the provided number 3 would not match the `few` case and fall back to the catch-all (`*`) case. During W06, the writing task for ordinals, SE6 noted the difference in endings for numbers like 1st versus 111th and was surprised that the ordinal selector handled that case: "I don't think this will capture all cases, because at some point it will wrap around and start with 'nd' again."

3.2 Writing Tasks

Violating the "Don't Repeat Yourself" principle Matchers allow you to group together different variants of a message, where one variant is chosen based on runtime data. Each key is followed by a separate quoted pattern. While this functionality is useful for translators, SE participants commented on how repetitive match tasks were (e.g., W06, W07). Figure 3 shows a screenshot of a repetitive solution. Nested matching was intentionally excluded from the language, citing feature coverage by "the combination of message references and top-level selection features, which together provide a sufficient feature set at a lower cost to the ecosystem than nested selectors would do."[4] SE1 indicated they liked the syntax but noted a code versus readability tradeoff: "If you could have match inside match, it might have been easier. It might not look as clean but it might reduce the amount of code." SE3 attempted to assign a `.match` to a `.local`. SE6 pointed out that their code violated the "Don't Repeat Yourself" principle and proposed a switch statement construction of putting "a literal string in a switch statement before and after it just to match the part that I need." None of the TR participants commented on the verbosity of matchers.

```
.input {$count :integer}
.input {$gender :string}
.match $count $gender
one female {{She has {$count} item in her cart}}
one male   {{He has {$count} item in his cart}}
one *      {{They have {$count} item in their cart}}
*   female {{She has {$count} items in her cart}}
*   male   {{He has {$count} items in his cart}}
*   *      {{They have {$count} items in their cart}}
```

Figure 3: Sample solution for W07

Unique use of the pipe symbol (`|`) for literals W08 tested string matching on a non-alphanumeric character, which requires the pipe symbol to quote the string. This differs from other programming languages where single quotation marks, double quotation marks, or backticks are commonly used. This difference was an intentional design choice in MF2, as one of the goals of the syntax is that messages should be "easily embeddable inside many container formats."[5] Three of the SE participants attempted to use double quotes for W08 before checking the documentation on how to write a literal. SE1 attempted to use curly braces, similar to how values in `.match` statements are surrounded. SE4 tried to escape the space in between `"New\ York"`. SE6 noted that "every language I know except for sed uses double quotes." Although nonintuitive at first, no further issues with literals arose after participants discovered the pipe syntax.

Feedback on Syntax When asked about challenging aspects of the tasks during the post-task survey, SE participants felt that the syntax itself was not difficult to learn, but figuring out which parameters and functions to use was. SE1 stated that "If I knew how everything worked, it would

be easier to write the messages." SE2 stated that the most challenging part of the writing tasks was "figuring out which parameters were required and which were optional."

3.3 Translation Tasks

Translation takes less time than writing Compared to writing tasks for software engineers, translation tasks took less time to complete. Figure 4 shows the average time spent on each writing/translation task per participant group. On average, translation tasks for software engineers required 32 minutes and 58 seconds, whereas translation tasks for translators only required 13 minutes and 23 seconds. While we do not make any conclusive claims given the small sample size, these numbers suggest translation is easier than writing messages from scratch, which is desirable as each message can be translated to many locales. It is worth noting that none of the translators referred to the documentation and worked directly on translating the given English MF2 messages.

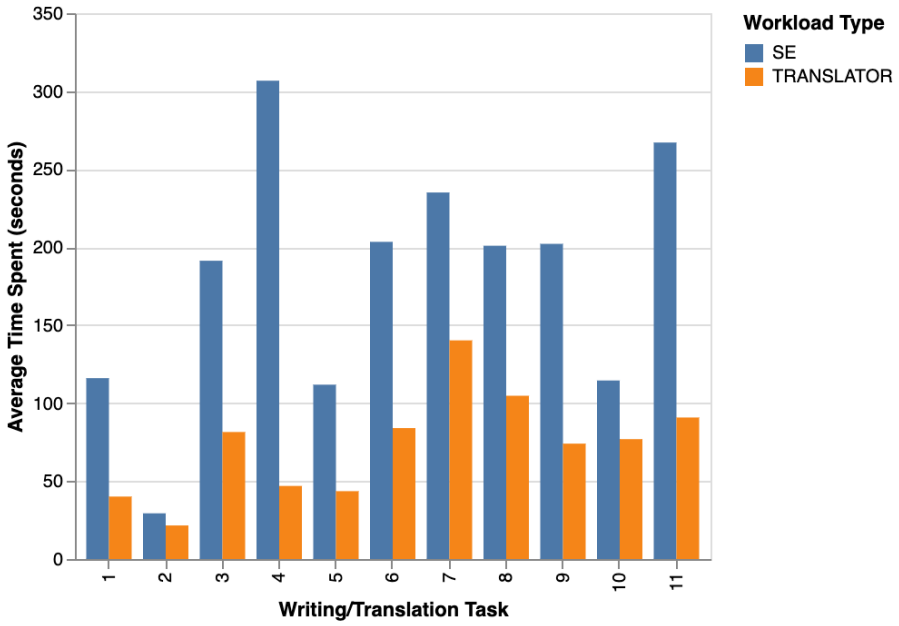

Figure 4: Average Time Spent for Writing/Translation Tasks

Translation is still challenging MF2 offers features designed to simplify internationalization; however, translators still encountered challenges and had to make compromises. While they indicated that their translations effectively conveyed the intended message, some of the rendered messages differed from how they would have translated specific, concrete messages. We report two instances in which MF2 failed to adequately support localization.

- In Turkish, certain words require suffixes that depend on the pronunciation of the final syllable. W10 involved formatting a timestamp in a "long" style, which concludes with the timezone in the format `GMT+n`, where n is a number. The suffix added to n varies based on its pronunciation (e.g., 1'de, 3'te, 6'da). The current design of MF2 does not support encoding this linguistic rule.
- In Danish, ordinals are written as words depending on the context (e.g., første, anden, tredje, etc.). In MF2, there is no built-in way to derive these word forms from numbers, other than explicitly matching the number and hardcoding cases for each one.

It is worth noting that, although these issues exist in the current design of MF2, they can be addressed by introducing new functions. A significant limitation of MF1 was its lack of extension points [7]. Learning from this, MF2 includes a general "function" mechanism that allows for future extensions, either through the "default registry" or user-defined functions. Both of the issues mentioned above can be resolved by adding appropriate functions.

4 Limitations

The sample size in this study was small for both groups, which may limit the generalizability of our results to a larger population. Though we were recommended to test Polish, Japanese, Turkish, French, and Arabic for their unique localization differences, we were unable to test all of these languages due to the availability of only four translators.

Additionally, the MF2 registry is still changing. One common application of localization is currency, where the currency and exchange value are indicated for specific locales. However, as this function had not been merged to the released version of the library, we had to exclude it from the study. Testing this function would also require knowledge of Unicode Currency Identifiers and other Unicode documentation, which do not affect the syntax of MF2. Converting datetimes between timezones was similarly excluded for this reason.

This study focuses on the current design of the specification, leaving other topics beyond its scope. For instance, TR1 mentioned a broader problem of translating words with different genders. While match statements might be able to handle this, but they would require developers to deliberately account for it when writing MF2 messages. Such topics are deferred for future studies.

5 Discussion

SE participants did not find it difficult to learn the MF2 syntax. A repeatedly stated challenge was needing to know the correct functions and options for the tasks. This may be an issue for software engineers at first, but these options would become more familiar over time for those who frequently need to implement localized strings. All SE participants were able to complete the tasks in the allotted time with the MF2 documentation. Thus, software engineers who wanted to use MF2 could ostensibly figure out the options for their intended use case. If MF2 was integrated into a larger codebase, developers could look at previous examples to learn unusual syntax, like pipes surrounding quoted literals, without having to look at the MF2 documentation. The study results suggest that syntax itself is not a barrier to software engineers' adoption, but quality resources are essential for them to learn how to write messages in a localization-friendly way.

Multiple SE participants compared MF2 to Python formatting strings and commented on the flexibility. P2 stated that MF2 "seems like a beefed up version with more customization for different types." P6 stated that "I can see myself using this because there are repetitive tasks that occur in any programming language, like ordinal" and that "it might be simpler to use this than create extra Python code for a trivial task." There exist implementations to use MF2 messages in JavaScript/TypeScript, Java, and C++, but the positive feedback from this study encourages further adoption of MF2 support in other languages.

The TR participants experienced no difficulty translating source strings into different languages. They quickly identified the parts of the source strings requiring translation and completed the task efficiently. Because the process did not involve adding language constructs, the translation proceeded smoothly, with only a few exceptions discussed in Section 3.3. All translators noted that they primarily use web-based GUIs for translation tasks and rarely engage directly with the underlying syntax. The successful completion of the translations without a GUI indicates that no significant issues are anticipated when using a GUI.

6 Future Work

Project Integration Since this study specifically focused on the MF2 format and not the integration, we did not address differences in how a project may be structured when using MF2 versus an existing framework. Extensions of this study could compare MF2 to popular localization and internationalization frameworks, for example i18next and formatJS for JavaScript, and evaluate message length, feature availability, and ease of integration.

Feature Testing Some features are present in MF2 but were not tested by this study. For example, MF2 allows users to define custom functions such as applying text transformations. This might be a

useful feature for software engineers who want finer control over messages. Future work could explore challenges in using and implementing custom functions.

Nested Matching MF2 does not allow nested matching, which leads to repetitive values in long match statements. The MF2 Working Group cited "the use of them has not been evaluated in production localization systems to date"[4] as a reason for not incorporating them into the language. It is worth noting that nested matching is likely to complicate translation as translators would have to deal with the inner match expression to fit the grammar of the target language. Future work could compare the readability of messages with nested matches versus the current implementation.

MF2 Ecosystem All four translators noted that they primarily rely on graphical user interface (GUI) tools for their translation work, highlighting the critical role of user-friendly tooling in localization workflows. However, as a newly introduced standard, MF2 currently lacks a mature ecosystem of tools to support its adoption and practical use. To ensure its success and widespread adoption, it is essential to prioritize the development of a robust ecosystem, including GUIs tailored to translators, developer-focused integrations, and comprehensive documentation.

7 Conclusion

This report describes the results of a think-aloud user study evaluating MessageFormat 2, a Unicode standard for localizable dynamic message strings. We conducted a think-aloud user study on software engineers and translators to understand if MF2 meets the needs of web developers and what challenges they face while writing MF2 messages. We observed that both software developers and translators did not face major issues with the core MF2 syntax, while we found minor obstacles the participants faced, such as the `.match` syntax, the "ordinal" select style, the use of the pipe symbol, etc. We believe that these issues can be resolved with proper documentation, further refinements to the default function registry, and the development of supporting tools such as GUI editors.

References

[1] Luca Casonato. *mf2-tools*. 2024. URL: https://github.com/lucacasonato/mf2-tools.

[2] TC39. *Intl.MessageFormat Proposal*. en. 2022. URL: https://github.com/tc39/proposal-intl-messageformat.

[3] Unicode. *International Components for Unicode*. URL: https://icu.unicode.org/.

[4] Unicode. *Message Format Consensus Decisions*. URL: https://github.com/unicode-org/message-format-wg/blob/main/docs/consensus_decisions.md.

[5] Unicode. *Message Format Syntax Specification*. URL: https://github.com/unicode-org/message-format-wg/blob/main/spec/syntax.md.

[6] Unicode. *Unicode Locale Data markup language (LDML)*. Nov. 2024. URL: https://www.unicode.org/reports/tr35/.

[7] Unicode. *Why MessageFormat needs a successor*. URL: https://github.com/unicode-org/message-format-wg/blob/7c1f1a4af4aead387ca6aec48fffc2c6e2191c4d/docs/why_mf_next.md.