

Converting PUA characters to Unicode

Peter S. Baker

November 9, 2025

I The scripts

Modern best practice, when typesetting documents or creating web pages, is to avoid the use of characters from the Unicode Private Use Area (PUA) whenever possible.¹ The Medieval Unicode Font Initiative (MUFI) defines more than 850 PUA characters, and Junicode includes all of them. Junicode also offers alternatives to all but a handful of these PUA characters, based on standard (non-PUA) Unicode with OpenType features applied.

OpenType is a powerful technology, but it is undeniably more difficult to work with than MUFI's PUA characters and entities. To copy any character (or its corresponding entity) from the MUFI website and paste it into your text is a straightforward procedure. By contrast, to work with OpenType one must locate the correct base character and the best OpenType feature, and then figure out how to include the needed markup, the syntax of which differs from one application to another.

For many users, a better procedure than struggling with OpenType tags may be to enter a PUA code or (better) an entity and let software handle the task of marking up the text appropriately. To help with this, the Junicode package includes a database that maps the relationships between MUFI's PUA characters and Junicode's OpenType features and a script that uses this database to convert any text's PUA characters to standard Unicode with OpenType. This script, `puazot`, comes in several flavors, all based on the script's base file, `puazot-common.js`: one is a web app, the second a script to embed in a web page, and a third a Node module. `puazot` works with html/css markup because it can easily be edited and can be imported into many applications.

¹For an explanation, see the *Junicode Manual*, § 4.1

Before you do anything else with **puazot**, copy the two font files **UnicodeVF-Roman.woff2** and **UnicodeVF-Italic.woff2** into the **puazot/fonts** directory in the unzipped Unicode file tree.

1.1 With Graphical User Interface

To use **puazot** with a GUI, find the file **puazot-gui.html** in the **puazot/gui** directory and double-click or drag it into an up-to-date web browser. The program will look like this:

MUFI PRIVATE USE TO OPENTYPE CONVERTER

The large text-box on the left (initially displaying an excerpt from the Middle English *Ancrene Wisse*) is the **Source Box**; it shows the text with PUA characters or entities. The text-box on the right is the **Destination Box**: it displays the result of the conversion. These two boxes should look the same: this sameness assures you that the conversion has gone well.

Immediately below the text boxes are options that affect their operation. To load a text or html file, click the “Choose File” button (“Browse” in some browsers) or drag the file into the Source Box. To edit the text, check the “Edit Source” box: you can type or paste in text when editing is enabled. To see which characters in the source are being replaced, check the “Mark Replaced Chars in Source” box.

To view the code that underlies the text displayed in the Destination Box, check the “Show Code” box. To export the converted text, click the “Download” or the “Copy” button.

The other options on the page affect the conversion. These are explained below, in the “Options” section.

1.2 Embedded in an HTML file

The second version of the script converts the <body> element of any HTML page before it is displayed, with the result that characters that could not be searched will become searchable, and the accessibility of the page will be improved. If the HTML file contains static text, make sure the script file, [pua2ot-embed-min.js](#), is in the same folder as your HTML file. Then include these lines at the very end of the <body>:

```
<script src="pua2ot-embed-min.js"></script>
<script class="noconv">
  // options.utagEntities = false;
  // options.utagLookup = UTAG_DICT;
  // options.nonWordTags = ["ss04", "ss05", "ss06", "pcap", "smcp", "hlig"];
  // options.keepUnicode = false;
  // options.methodPriority = "compact";
  // options.prefList = ["utag", "zwj", "otag", "entity"];
  // options.defaultTags = {"ss10": 1, "cv69": 7};
  // options.language = "en";
  // options.variantPreferences = [];
  convertAll();
</script>
```

If your web page loads text dynamically, it won’t work simply to include the command [convertAll\(\)](#) at the end of the HTML file, as it will likely be executed before the text is loaded. Instead, have the process that loads the text trigger an event when the loading is done and run [convertAll\(\)](#) from inside the event handler.

In the html file, include “noconv” in the class list of any element whose content you don’t want converted: the script will skip over such elements.

Other scripts may use this one: simply call the function [convert\(\)](#), passing a string as an argument; the function will return a converted string.

In the example above, the commented lines before the [convertAll\(\)](#) command are options, explained in a later section.

1.3 Node module

The file [pua2ot-common.js](#) includes instructions for using it as a Node module, but if you are not making changes in the programming you can use [node/pua2ot-node-min.js](#), which packages the [pua2ot](#) database and program code in a single minified file.

The file exports the object `options` (the options it supplies are listed in the next section) and two functions: `puazotNode.cleanup_string(s)`, which collapses all sequences of whitespace characters into single spaces, and `puazotNode.convert(text_buffer, repl_ent)`, which converts and returns the text `text_buffer`, first replacing any MUFI entities if `repl_ent` is true.

For an example of how the Node module can be used, consult and run the file `test2.js`, especially the `require` statement in line 3, the `options` object in line 13, and the invocation of `convert()` in line 15.

2 Options

In addition to the checkboxes, menus, and text-entry blanks in the web app, the JavaScript program has an `options` object whose properties can be manipulated to affect the program's output.

2.1 More legible markup

Default: `options.methodPriority="compact"`. By default, `puazot` substitutes standard Unicode characters modified by Unicode tag characters (characters from the Unicode tag range) for MUFI PUA characters whenever possible. This method produces relatively compact and accessible web pages. However, Unicode tags are invisible, making them hard to edit, and Junicon is the only font that uses them in this way: the method is not exactly portable. If you check the “More legible markup” box or set `options.methodPriority="legible"`, `puazot` will instead use `` elements with inline style attributes that turn on OpenType features. The chief advantages of this method are that you can see and easily edit the markup, and HTML elements are more familiar to users than Unicode tags. However, the resulting files are a good bit bulkier than those that use Unicode tags. You can produce smaller and better organized files by running the output of this script through a utility (e.g. the [Javascript HTML Style Converter](#)) that converts inline styles to a CSS stylesheet.

2.2 Use entities for Unicode tags

Default: `options.utagEntities=false`. If `true`, `puazot` uses entities to represent Unicode tags. (These are not HTML entities, but unique to Junicon.) This makes the code produced by this script more legible, but the entities make the page somewhat larger

and can interfere with searching. It is probably best to use this feature to help you spot problems with your Unicode tags and turn it off before publishing your text.

2.3 Keep problematic Unicodes

Default: `options.keepUnicode=false`. “Problematic Unicodes” are characters that are standard Unicode (not PUA) and look like variants of common characters, but that many applications do not recognize as variants. Examples include dotless i (**ı**, U+0131) and the “insular” letter-shapes (e.g. **ð**, U+A77A). By default, this script replaces these characters with ones that can be searched as letters of the alphabet (e.g. **ð** as **d**). To retain the “problematic Unicodes,” likely simplifying your markup and making your file smaller, but making your text less accessible in some applications, check this box in the GUI version or set `options.keepUnicode=true`.

2.4 Alternate Unicodes

Default: `options.variantPreferences=[]`. As a large font with many OpenType features, Junicore often provides more than one way to achieve the outcome you want. **Pua2ot** operates by substituting a standard Unicode **base** character for a PUA character and then applying OpenType markup to that base: **pua2ot** options affect both the base and the markup. This option affects several classes of character (**insular** letters, Unicode-encoded **small caps**, **punctuation**, combining **marks**, and **currency** signs). Select one or more of the items in this list (use the Shift or Control/Command keys to make multiple selections) or assign an array to `options.variantPreferences` to alter the substitutions **pua2ot** makes for many characters in these classes. The default settings here are chosen to maximize accessibility and minimize the size of the output file, but you may have good reasons for preferring alternate choices. For example, small caps can by default be searched as their lowercase equivalents (e.g. as **a**), but if these small caps make a meaningful distinction in your text, you may wish readers to be able to search for them by their Unicode values instead (U+1D00 for **A**). Use this option to maintain that distinction.

2.5 replaceMUFIEntities

Default: `options.replaceMUFIEntities=true`. If **true**, MUFI’s entities (those that resolve to PUA code points) are replaced by those code points before any other processing takes place. Set this to **false** if the text you want to convert contains no entities

and you want to save a little time processing a large text. Not available in the GUI version, where entities are always converted.

2.6 Default features

Default: `options.defaultTags={"ss10": 1, "cv69": 7}`. You may wish some OpenType features to be on for the whole document. For example, `ss10` “Entities and Tags” must be on in any document that uses Junicode entities or Unicode tags. To manage your text’s OpenType features, this script needs to know about these default features; and when you download or copy your text, they will be recorded in an enclosing `<div>`. Note the format: this is a comma-separated list of key-value pairs, where the keys (four-letter tags identifying OpenType features) are in double quotation marks, followed by a colon and a number (1 means “on,” but some features (especially the `cvnn` “Character Variant” features) require a numerical index).

2.7 Non-word tags

Default: `options.nonWordTags=["sso4", "sso5", "sso6", "pcap", "smcp", "hlig"]`. When “More legible markup” is checked, the script applies most `` tags to whole words, but a few to individual characters. This distinction is based on an educated guess as to which OpenType features are appropriate to apply to whole words. Sometimes this guess will be wrong. For example, the word *rectificationem* (about halfway through the sample *Ancrene Wisse* text) contains two different styles of `t`, but when `cv40` (which transforms default `t` into insular `ṭ`) is applied to the whole word, that difference is erased. If you add `cv40` to the list in this box (remember that items in the list are enclosed in quotation marks and separated by commas), the feature `cv40` will always be applied to single letters rather than whole words. (An alternative to editing this list is to edit the markup for this word after you have exported your text.)

2.8 Language

Default: `options.language="en"`. It is important to set the language properly, both here and in your document, since this script and some of Junicode’s OpenType features behave differently depending on the language. If your language is not in the drop-down list (available only in the GUI version of this script), choose “Other” and type in the two-letter code for the language (you can look it up [here](#).)