Awami Nastaliq - Developer information

Welcome font developers!

We welcome other developers who wish to get involved in supporting and enhancing these fonts or who want to modify them.

Permissions granted by the OFL

SIL's fonts are licensed according to the terms of the SIL Open Font License. The OFL allows the fonts to be used, studied, modified and redistributed freely as long as they are not sold by themselves. For details see the OFL.txt and OFL-FAQ.txt files in the package.

Building the fonts from source code

Font sources are published in a Github project. The build process requires smith and project build parameters are set in the wscript.

Font sources are in the UFO3 format with font family structures defined using designspace. There is no OpenType code in this font, but the font does use the Graphite smart font technology.

The fonts are built using a completely free and open source workflow using industry-standard tools (fonttools), a package of custom python scripts (pysilfont), and a build and packaging system (smith). The whole toolchain is available as a Docker container.

Full instructions for setting up the tools and building SIL fonts are available on a dedicated web site: Building and Modifying SIL Fonts.

Building

The Awami Nastliq project can be built from source using smith. This is done via the sequence:

```
    smith distclean
    smith configure
    smith build -v -j1
```

Because of the complex kerning and collision avoidance logic, builds can take up to 15 minutes or longer, depending on hardware.

Some useful `smith build` options

`-v` makes the output slightly more verbose, specifically including the "runner" information showing the actual commands smith is executing.

`-j` controls parallel processing. Depending on your machine's memory, `smith build` sometimes fails due to the intense computation requirements of the Awami build. If this occurs, `-j1` or `-j2` can be used to restrict parallel processing, although this slows the build process somewhat. The number after the `-j` indicates the number of tasks smith will try to do in parallel.

`-d` should normally be omitted when building the fonts. However, when developing/debugging a font using Graide, the `-d` must be used to prevent some optimizations that are incompatible with Graide.

`--regOnly` causes smith to build the Regular weight only. This is useful during development and debugging.

This would just build Regular:

```
smith distclean
smith configure
smith build -d --regOnly -v -j1
```

Modifying the font

Project documentation

A good deal of developer documentation for the Awami Nastaliq font can be found at the Awami GitHub repository.

Adding characters

Like most Nastaliq fonts, Awami takes a "decomposition" approach, where initial, medial, and final forms are constructed at rendering time from separate base glyphs and nuqtas and other inherent parts of characters. This means that adding a new character will likely require only adding the isolate form along with its USV encoding.

After adding new glyphs to the font, the shaping logic will need to be extended to handle them. The bulk of the code is found in: - `nastaliq_classes.gdh` - `nastaliq_cntxlClasses.gdh` - `nastaliq_shaping.gdh` - `nastaliq_rules.gdh`.

A helpful approach is to do a global search through the code for a character with similar behavior and add glyphs for the new character in all the corresponding places.

In addition, the following will need to be updated: - `glyph_data.csv` -- should include all glyphs in the font; it is used to set glyph order in the built font. - `nastaliq_complexShapes.gdh` -- needs to include any glyphs who shapes cannot be approximated by a simple polygon for the purposes of kerning, particularly those with concave portions. - "Octabox" data will need to be updated for all the weights of the font; see below. - `ftml_test_gen.py` -- should include the new character in order for it to be included in the automatically generated test files.

Generating octaboxes

"Octaboxes" are polygons that approximate the shape of the glyphs; these are used for kerning and fixing collisions. There is an octabox JSON file for each font weight. Whenever new glyphs are added or glyph shapes are signficantly modified, the octaboxes should be regenerated.

Before any octabox can be updated, the ttf file for the corresponding font must be in the `results/` folder, so you first need to build the fonts (see above). Then the command to update a single octabox is:

```
octalap -j 0 -q -o source/graphite/octabox_AwamiNastaliq-WEIGHT.json   results/AwamiNastaliq-WEIGHT.ttf
```

where WEIGHT is Regular, Bold, etc. As given above, the command must be run from the root of the project. The command must be executed for each weight, and each will take several minutes to execute.

Alternatively, there is a script in the `tools/` folder called `run_octalap` which, if run from the `tools/` folder, will update all the octaboxes:

```
cd tools
./run_octalap
```

In order to use the newly generated octaboxes the font must then be rebuilt.

Auto-generated test files

The project includes a Python program `tools/scripts/ftml_test_gen.py` that can generates test data in a form of XML called FTML (see below). A variety of test files can be built, covering various combinations of base characters and diacritics: - `basicforms` - only one character of each shape class is included, no diacritics - `allbasechars` - all base characters, no diacritics - `basic_somediac` - only one character of each

shape class with one lower and one upper diacritics - `basic_alldiac` - only one character of each shape class with all diacritics - `allbasecharforms` - all characters and diacritics are included (this creates a huge file!)

The test file builder is called as follows:

```
python3 tools/scripts/ftml_test_gen.py
```

which generates files for all five modes, or

```
python3 tools/scripts/ftml_test_gen.py -m MODE
```

These files are not automatically generated by the build.

Modifying `ftml_test_gen.py`

When new characters are added to the font, they should be added to `ftml_test_gen.py`. - Assign the character a short code and associated it with the USV the `_char_name_to_usv()` function. - Add details about the character to the `_group_name_format()` or `_diac_group_name_format()` function. - Add the code to the appropriate list in the `expand_sequences()` or `insert_diacritics()` function.

`tools/ftml.xsl` can be used to view ftml documents directly in Firefox (which supports both Graphite rendering).

Viewing FTML test files

The `ftml.xsl` file is used to view the FTML files in Firefox. Firefox is needed to both handle the XSL transforms as well as provide Graphite render.

However, in order for Firefox to access the .xsl file, you need to relax its "strict URI" policy by going to about:config and setting security.fileuri.strict_origin_policy to false.

Once you have this setting in effect, you can load the FTML documents directly into Firefox and see the built font rendered.

Contributing to the project

We warmly welcome contributions to the fonts, such as new glyphs, enhanced smart font code, or bug fixes. The brief overview of contributing changes is a good place to begin. The next step is to contact us by responding to an existing issue or creating an issue in the Github repository and expressing your interest. We can then work together to plan and integrate your contributions.

To enable us to accept contributions in a way that honors your contribution and respects your copyright while preserving long-term flexibility for open source licensing, you would also need to agree to the **SIL Global Contributor License Agreement for Font Software (v1.0)** prior to sending us your contribution. To read more about this requirement and find out how to submit the required form, please visit the CLA information page.

This guide is from the Awami Nastaliq project version 3.5 and is copyright © 2014-2026 SIL Global.