Turning `fontc_crater` into a generic A/B font testing tool

2025-08-12

Georg Seifert (Glyphs)
Martin Winter

Recap

`ttx_diff.py`

- one font file
- builds with `fontc` + `fontmake` (hard-coded)
- “default” or managed by Google Font Tools (“gftools”)
- normalizes and compares `ttx` files
- outputs diffs and failures to CLI or JSON

`fontc_crater`

- many font files
- by default, runs *two* `ttx_diff.py` builds for each font file:
  - “default”
  - “gftools”
- reads diffs and failures from JSON
- outputs changes to HTML (`fontc` + `fontmake` hard-coded)

Where we are now

`ttx_diff.py`

- adds Glyphs.app as additional tool (requires bundle paths)
- largely tool-agnostic
  - `tool_1` + `tool_2`
  - `Tool` class and subclasses
  - removes `compare` flag
- still includes tool-specific code
- can use same font cache as `fontc_crater`

`fontc_crater`

- similar to `ttx_diff.py`
- still supports presets (“mode”)
- `Tool` and `ToolManagement` enums, `ToolVersion` struct
- `RunConfiguration` struct converts CLI arguments (preset or arbitrary tools) to one or two `ToolPairs`
- separate results folder per tool pair recommended

Potential future

`ttx_diff.py`

- builds with *any* combination of two tools, not just `fontc`, `fontmake`, Glyphs.app
- main logic completely tool-agnostic
- tool configuration + custom run logic via plug-in files

`fontc_crater`

- support local font files (no repo)
- HTML output also tool-agnostic
  - solve hybrid runs (“default” + “gftools”)
- tool revisions not just for `fontc`
- extract from `fontc` repository
  - currently builds `fontc` tool
  - main external dependency: `google_fonts_sources`

Implementation

Finding the right abstractions without breaking existing functionality:

Tool	Management	Versioning
`fontc`	“default”, “gftools”	—
`fontmake`	“default”, “gftools”	—
Glyphs.app	— (“default”)	different builds

Implementation

Once appropriate abstractions were discovered, Rust made implementing them fairly easy:

- enums with associated values
- `match`
- `Result`

```
enum ToolManagement {
    Standalone,
    ManagedByGfTools,
}
struct ToolVersion {
    version_number: String,
    build_number: String,
}
enum Tool {
    Fontc(ToolManagement),
    Fontmake(ToolManagement),
    GlyphsApp(PathBuf, ToolVersion),
}
```

Implementation

Mode	Number of runs per font	Requires tool paths	Requires tool types
`GfTools` preset	2	—	—
`Default` preset	1	—	—
`GlyphsApp` preset	1	●	—
arbitrary tool pair	1	●	●

```
struct ToolPair {
    tool_1: Tool,
    tool_2: Tool,
}
struct RunConfiguration {
    tool_pairs: Vec<ToolPair>,
}
```

```rust
impl RunConfiguration {
    pub fn from_cli_args(args: &CiArgs) -> Result<Self, RunConfigurationError> {
        match (
            args.preset,
            args.tool_1_type, args.tool_1_path.as_ref(),
            args.tool_2_type, args.tool_2_path.as_ref()
        ) {
            // GFTools preset. Ignore tool types and paths.
            (Some(Preset::GfTools), _, _, _, _) => {
                Ok(RunConfiguration {
                    tool_pairs: vec![
                        ToolPair {
                            tool_1: Tool::from_cli_args(ToolTypeCli::StandaloneFontc, None)?,
                            tool_2: Tool::from_cli_args(ToolTypeCli::StandaloneFontmake, None)?,
                        },
                        ToolPair {
                            tool_1: Tool::from_cli_args(ToolTypeCli::GfToolsFontc, None)?,
                            tool_2: Tool::from_cli_args(ToolTypeCli::GfToolsFontmake, None)?,
                        },
                    ],
                })
            },
            ...
```

```
// Two tool types specified, no preset.
(None, Some(tool_1_type), tool_1_path, Some(tool_2_type), tool_2_path) => {
    let tool_1 = Tool::from_cli_args(tool_1_type, tool_1_path.map(|p| p.into()))?;
    let tool_2 = Tool::from_cli_args(tool_2_type, tool_2_path.map(|p| p.into()))?;

    // (The following logic is based on the `ttx_diff.py` script.)
    // Currently, having two tools of the same type is supported for Glyphs app
    // instances only. These must have at least different build numbers.
    if tool_1.tool_type() == tool_2.tool_type() {
        if tool_1.tool_type() != ToolType::GlyphsApp {
            return Err(RunConfigurationError::InvalidToolCombination(
                tool_1.tool_type(),
                tool_2.tool_type(),
            ));
        }

        // Both tools are Glyphs apps.
        if tool_1.version() == tool_2.version() {
            return Err(RunConfigurationError::SameToolBuilds);
        }
    }

    Ok(RunConfiguration {
        tool_pairs: vec![
            ToolPair {
                tool_1: tool_1,
                tool_2: tool_2,
            },
        ],
    })
},
```

Implementation

Tools have been encapsulated in Python classes, but the management dimension is currently not separated out. This may not be needed since `ttx_diff.py` is by its nature closer to the CLI:

```
# Note:  This is currently a flat list like the `ToolTypeCli` enum in
# `fontc_crater/src/args.rs`. It does not separate the concepts of tool type,
# tool management and tool instance.
class ToolType(StrEnum):
    # Standalone versions
    FONTC = "fontc"
    FONTMAKE = "fontmake"

    # Managed by GFTools
    FONTC_GFTOOLS = "fontc_gftools"
    FONTMAKE_GFTOOLS = "fontmake_gftools"

    GLYPHSAPP = "glyphsapp"
```

```python
@dataclass
class Tool(ABC):
    source: Path

    # Subclasses should return their specific tool type.
    @property
    @abstractmethod
    def type(self) -> ToolType:
        pass

    # Subclasses may override the tool name, e. g. by appending a version.
    @property
    def name(self) -> str:
        return self.type

    # Subclasses may override the font file extension.
    @property
    def font_file_extension(self) -> str:
        return "ttf"
```

```
    # Subclasses should return their specific build action as a closure.
    @abstractmethod
    def build_action(self, build_dir: Path, font_file_name: str = None) -> Callable[[], None]:
        pass

    # Central method for building and exception handling (internal).
    def _build(self, build_action: Callable[[], None]) -> Optional[dict]:
        try:
            build_action()
        except BuildFail as e:
            return {
                "command": " ".join(e.command),
                "stderr": e.msg[-MAX_ERR_LEN:],
            }

        return None

    # Call this method to generate the specific build action and start a build.
    def run(self, build_dir: Path, font_file_name: str = None) -> Optional[dict]:
        action = self.build_action(build_dir, font_file_name)
        return self._build(action)
```

```python
@dataclass
class FontcTool(Tool):
    root: Path
    fontc_path: str = None

    _fontc_bin: Path = field(init=False, repr=False)

    def __post_init__(self):
        self._fontc_bin = get_crate_path(self.fontc_path, self.root, FONTC_NAME)
        assert self._fontc_bin.is_file(), f"fontc path '{self._fontc_bin}' does not exist"

    # Created once, read-only.
    @property
    def fontc_bin(self):
        return self._fontc_bin


@dataclass(unsafe_hash=True)
class StandaloneFontcTool(FontcTool):

    @property
    def type(self) -> ToolType:
        return ToolType.FONTC

    def build_action(self, build_dir: Path, font_file_name: str = None) -> Callable[[], None]:
        def action():
            build_fontc(self.source, self.fontc_bin, build_dir, font_file_name)
        return action
```

```
@dataclass(unsafe_hash=True)
class GlyphsAppTool(Tool):
    bundle_path: str = None

    # Created once, internal.
    _glyphsapp_proxy: NSDistantObject = field(init=False, repr=False)

    _version: str = None
    _build_number: str = None

    def __post_init__(self):
        self._glyphsapp_proxy, self._version, self._build_number = application(self.bundle_path)
        if self._glyphsapp_proxy is None:
            sys.exit(f"No JSTalk connection to Glyphs app at {self.bundle_path}")

    @property
    def type(self) -> ToolType:
        return ToolType.GLYPHSAPP

    @property
    def name(self) -> str:
        underscored_version = self._version.replace(".", "_")
        return f"{self.type}_{underscored_version}_{self._build_number}"

    @property
    def font_file_extension(self) -> str:
        return "otf"

    def build_action(self, build_dir: Path, font_file_name: str = None) -> Callable[[], None]:
        def action():
            exportFirstInstance(self._glyphsapp_proxy, self.source, build_dir, font_file_name)
            self._glyphsapp_proxy = None
        return action
```

```
# Ensure that required Glyphs app bundle paths are set.
if tool_1_type == ToolType.GLYPHSAPP and FLAGS.tool_1_path is None:
    sys.exit("Must specify path to Glyphs app bundle used as tool 1")
if tool_2_type == ToolType.GLYPHSAPP and FLAGS.tool_2_path is None:
    sys.exit("Must specify path to Glyphs app bundle used as tool 2")

# Currently, having two tools of the same type is supported for Glyphs app
# instances only. These must have at least different build numbers.
if tool_1_type == tool_2_type:
    if tool_1_type != ToolType.GLYPHSAPP:
        sys.exit("Must specify two different tools")

    # Both tools are Glyphs apps.
    version_1, build_number_1 = app_bundle_version(FLAGS.tool_1_path)
    version_2, build_number_2 = app_bundle_version(FLAGS.tool_2_path)
    if version_1 == version_2 and build_number_1 == build_number_2:
        sys.exit("Must specify two different Glyphs app builds")

tool_1 = None;
if tool_1_type == ToolType.FONTC:
    tool_1 = StandaloneFontcTool(source, root, FLAGS.tool_1_path)
elif tool_1_type == ToolType.FONTMAKE:
    tool_1 = StandaloneFontmakeTool(source)
elif tool_1_type == ToolType.FONTC_GFTOOLS:
    tool_1 = GfToolsFontcTool(source, root, FLAGS.tool_1_path)
elif tool_1_type == ToolType.FONTMAKE_GFTOOLS:
    tool_1 = GfToolsFontmakeTool(source)
elif tool_1_type == ToolType.GLYPHSAPP:
    tool_1 = GlyphsAppTool(source, FLAGS.tool_1_path)
```

Examples: ttx_diff.py

```
# Before:
$ python resources/scripts/ttx_diff.py font/repo/url
      --compare default

# After (mode replaced with tool types):
$ python resources/scripts/ttx_diff.py font/repo/url
      --tool_1_type fontc --tool_2_type fontmake
```

Examples: ttx_diff.py

```
# Before:
$ python resources/scripts/ttx_diff.py font/repo/url
      --compare gftools
      --config path/to/yaml

# After (mode replaced with tool types, new cache parameter):
$ python resources/scripts/ttx_diff.py font/repo/url
      --tool_1_type fontc_gftools --tool_2_type fontmake_gftools
      --config path/to/yaml
      --cache_path path/to/fontc_crater_cache
```

Examples: ttx_diff.py

```
# Two Glyphs builds:
$ python resources/scripts/ttx_diff.py font/repo/url
      --tool_1_type glyphsapp
      --tool_1_path '/Applications/Glyphs 3.4 (3422).app'
      --tool_2_type glyphsapp
      --tool_2_path '/Applications/Glyphs 4.0a (3838).app'

# Glyphs and `fontc`:
$ python resources/scripts/ttx_diff.py font/repo/url
      --tool_1_type glyphsapp
      --tool_1_path '/Applications/Glyphs 4.0a (3838).app'
      --tool_2_type fontc
```

Examples: ttx_diff.py

```
# Specify different versions of _any_ tool.
# Besides Glyphs, this already works for `fontc`,
# with `tool_x_path` replacing `fontc_path`.
# The `fontmake` version could be specified with a
# virtual environment or a Python version:
$ python resources/scripts/ttx_diff.py font/repo/url
      --tool_1_type fontc
      --tool_1_path path/to/fontc/binary
      --tool_2_type fontmake
      --tool_2_path path/to/fontmake/venv
```

Examples: fontc_crater

```
# Before (disable `gftools` if running `default` only):
$ cargo run --release -p fontc_crater -- ci
      path/to/google-fonts-sources.json
      --out path/to/gftools/results
      --gftools false

# After (explicit preset):
$ cargo run --release -p fontc_crater -- ci
      path/to/google-fonts-sources.json
      --out path/to/results
      --preset default
```

Examples: fontc_crater

```
# Glyphs app preset (requires tool paths):
$ cargo run --release -p fontc_crater -- ci
      path/to/google-fonts-sources.json
      --out path/to/glyphsapp/results
      --preset glyphsapp
      --tool-1-path '/Applications/Glyphs 3.4 (3422).app'
      --tool-2-path '/Applications/Glyphs 4.0a (3838).app'
```

Examples: fontc_crater

```
# Glyphs and `fontc`, also use existing cache argument:
$ cargo run --release -p fontc_crater -- ci
      path/to/google-fonts-sources.json
      --cache path/to/fontc_crater_cache
      --out path/to/glyphsapp/results
      --tool-1-type glyphsapp
      --tool-1-path '/Applications/Glyphs 3.4 (3422).app'
      --tool-2-type fontc
```

Questions?

- Virtual sources

Thank you!