This is a manual for developers. There are two aspects to the code: the user-interface and the G-code translator. The UI is basically how Swing is used, plus the plumbing that holds the entire program together, while the translator is the central task the program is intended to accomplish.

1 User-interface and Plumbing

The entry point for the program is found in[1]

`vcnc.Main`

There's not a lot to say about this class: it merely kicks off the entire program by creating a

`vcnc.MainWindow`

This is where all the plumbing comes together and is the primary window for user activity. Most of what happens in the class is standard stuff: menus, windows, *etc.*

Alternatively, the `main()` method can invoke a series of units tests. Which of the two is done (ordinary GUI or unit test) depends on which of two methods is called by `main()`.[2] The unit tests are discussed further, below, with the translator.

This window puts each file in a tabbed frame. By default, Java's `JTabbedPane` doesn't have all the functionality desired, so there's a fair amount of UI code in the `vcnc.ui.TabbedPaneDnD` package, which is discussed further, below.

In addition, each tab shown has a type that corresponds to the contents of that tab. These types are managed by the `vcnc.ui.TabMgmt` package. What is does is simple.

`vncc.ui.TabMgmt.TabbedType`

is an `enum` listing the possible types of tab contents and

`vncc.ui.TabMgmt.TypedDisplayItem`

is an interface that each tab `Component` must implement to associate a type with its contents. Currently, all of these extend `JScrollPane`, but any number of other `awt.Component` objects would probably work as the base class.

Currently, there are only three of these types:

```
vncc.ui.TabMgmt.GInputTab
vncc.ui.TabMgmt.LexerTab
vncc.ui.TabMgmt.ParserTab
```

[1] "vcnc" stands for "Virtual CNC."

[2] It wouldn't be hard to the choice of which to run into a command-line argument, but it seems like an unnecessary complication.

The LexerTab and ParserTab are essentially identical, [3] and are used to display G-code output, which is not editable. The GInputTab is a bit more complicated because this code is editable and due to toggling line numbers.

The final class found in this package is

```
vncc.ui.TabMgmt.StaticWindow
```

which is used to convert a tab to an independant window whose contents can no longer be edited.[4]

The remaining files in the vcnc package are

```
vcnc.TextInputDialog
vcnc.TableMenu
```

The TextInputDialog class is currently used to set the material billet, but it's not being done in the way that it ultimately should be done.

vcnc.TableMain is leftover junk – test code for JTables. Get rid of it.

TALK ABOUT HOW THE MAIN WINDOW INVOKES THE UNDERLYING TRANSLATOR.

1.1 vcnc.ui.TabbedPaneDnD Package

Two features beyond those provided by JTabbedPane are added: an "X" to close the tabs, and the ability to drag and drop the tabs within each window or from one window to another. The close-box is managed (mostly) through ButtonTabComponent, and the remaining classes manage drag & drop, with TabbedpaneDnD being the main class of interest externally. See[5]

```
vcnc.ui.TabbedPaneDnD.ButtonTabComponent
vcnc.ui.TabbedPaneDnD.GhostGlassPane
vcnc.ui.TabbedPaneDnD.TabbedPaneDnD
vcnc.ui.TabbedPaneDnD.TabDragGestureListener
vcnc.ui.TabbedPaneDnD.TabDragSourceListener
vcnc.ui.TabbedPaneDnD.TabDropTargetListener
vcnc.ui.TabbedPaneDnD.TabTransferable
vcnc.ui.TabbedPaneDnD.TabTransferPacket
```

[3] Clean that up.

[4] This is also similar to LexerTab and ParserTab.

[5] I wrote this ages ago, and it seems to work, but it's not pretty. I need to rewrite it, make it more of a stand-alone thing that is useful more generally and provide better documentation for it. It's messy Swing, so I'm putting it off.

1.2 Miscellany in the `vcnc.util` Package

`vcnc.util.ChoiceDialogRadio`
`vcnc.util.ClickListener`
`vcnc.util.EmptyReadException`
`vcnc.util.FileIOUtil`
`vcnc.util.LoadOrSaveDialog`
`vcnc.util.StringUtil`

Not much to say about these...`ClickListener` may be trash, and they could all be tidied up with unused stuff taken out. They're mostly older code pulled from other projects.

2 G-code Translator

GER is similar to a (very simple) compiler or interpreter. It converts an input file of G-code to a simpler from. This simplification happens as the code passes through a series of layers, where each layer handles a particular aspect of the simplification. The code related to this translation process is all in `vcnc.tpile.*`, either in that package or in a sub-package.

The lowest layer is the lexical analyizer (or "lexer"). Because G-code is so simple, with very little context-dependence, the lexer is equally simple. It converts the incoming text file to a stream of `Token` objects. Each token represents one of the letter codes (G, M, I, J, *etc.*) and any associated value. GER allows the user to extend ordinary G-code with user-defined functions, and these functions are also converted to `Token` objects.

The `Token` objects are passed to the next layer, which is the parser. The parser assembles the tokens into `Statement` objects. These statements correspond to the individual conceptual steps of the program. The remaining layers convert these statements into an increasingly stripped-down subset of the possible G-codes, ultimately producing nothing but `G00, G01, G02, G03` codes for moving the cutter, plus a few M-codes and other codes that pass through the process untouched.

The remaining layers are where the meat of the simplification occurs. By layering the simplification process in this way, each layer is made easier to understand, although it does lead to a certain amount of repetative boilerplate. These layers are given numbers, starting with `00`.

2.1 Lexer

The code used for lexing is found in `vcnc.tpile.lex`. Classes outside this package – the next layer of the translator – need access to the `Lexer` and `Token` classes. The other classes here have default visibility, so are not accessible outside the package: there's a token buffer and some internally used exception classes.

2.2 Parser

The parser takes a stream of `Token` objects, generated by `Lexer`, and produces a series of `Statement` objects. This code is found in `vcnc.tpile.parse`. Most of the classes in this package extend `StatmentData`. Each type of `Statement` may need different data, one class for ciruclar interpolation, one for linear moves, *etc.* The names for these classes start with "`Data`."

2.3 Layer00

This layer eliminates all calls to subroutines. In particular, `M98` (call subprogram) and `M99` (return from subprogram) are replaced by the code of the corresponding subprogram(s).

IN ADDITION, this also eliminates certain items that serve no purpose in a simulator, like `M07, M09` and `M09` for coolant control, together with `M40` and `M41` for spindle high/low and `M48` and `M49` for feed and speed overrides.

For several other commands, it's not clear what the appropriate action should be, so they are treated as a "halt." These codes are `M00, M01` and `M02` (various forms of "stop"), along with `M47` (repeat program).

BUG: I suspect that some of these should pass all the way through the program, and only be dropped at the very end, when rendering.

2.4 Unit Tests

As noted above, the unit tests can be run from the `main()` method. The basic idea is that each layer of the translator, starting with the lexer, is able to produce output as text (a `String`). Each test is specified by an input file, which is run through the translator up to a certain level. The output from this run is compared to a static text file to see if they match.

3 Geometry

There are a few geometric problems that come up.

One problem is that we have two end-points of an arc and the radius, and we need to know the center. This arises in the `ArcCurve` class. Work in the xy-plane. Let the two end-points be (x_0, y_0) and (x_1, y_1), with radius r, and the center be at (c_x, c_y). We want to determine the center given the other values.

We have two equations and two unknowns:

$$
\begin{aligned}
r^2 &= (x_0 - c_x)^2 + (y_0 - c_y)^2 \\
r^2 &= (x_1 - c_x)^2 + (y_1 - c_y)^2
\end{aligned}
$$

Solve the first equation for c_x:

$$
\begin{aligned}
(x_0 - c_x)^2 &= (y_0 - c_y)^2 - r^2 \\
x_0 - c_x &= \pm\sqrt{(y_0 - c_y)^2 - r^2} \\
c_x &= x_0 \mp \sqrt{(y_0 - c_y)^2 - r^2}
\end{aligned}
$$

Substitute that into the second equation and solve for c_y:

$$
\begin{aligned}
r^2 &= (x_1 - c_x)^2 + (y_1 - c_y)^2 \\
r^2 &= \left(x_1 - \left(x_0 \mp \sqrt{(y_0 - c_y)^2 - r^2}\right)\right)^2 + (y_1 - c_y)^2 \\
r^2 &= \left(x_1 - x_0 \pm \sqrt{(y_0 - c_y)^2 - r^2}\right)^2 + (y_1 - c_y)^2 \\
r^2 &= (x_1 - x_0)^2 \pm 2(x_1 - x_0)\sqrt{(y_0 - c_y)^2 - r^2} + (y_0 - c_y)^2 - r^2 + (y_1 - c_y)^2
\end{aligned}
$$

Solving that for c_y might work, but it's sort of horrifying. Instead, use the fact that a line perpendicular to a chord of the arc must pass through the center. That is, there is a chord passing through (x_0, y_0) and (x_1, y_1). Let (m_x, m_y) be the mid-point of that chord. The slope of the chord is given by

$$
s = \frac{y_1 - y_0}{x_1 - x_0}
$$

(and the order of the x_i and y_i doesn't matter). The slope of the perpendicular to the chord is $-1/s$ or

$$
s = \frac{x_0 - x_1}{y_1 - y_0}
$$

and the line perpendicular to the chord is given by

$$
y - y_0 = \left(\frac{x_0 - x_1}{y_1 - y_0}\right)(x - x_0).
$$

In particular, the center of the circle lies on this line, so we must have

$$
c_y - y_0 = \left(\frac{x_0 - x_1}{y_1 - y_0}\right)(c_x - x_0).
$$

Let's simplify this mess...Define

$$\begin{aligned} m_x &= x_0 - x_1 \\ m_y &= y_0 - y_1. \end{aligned}$$

The slope of the chord is then $s = m_y/m_x$ and the slope of the perpendicular is $-1/s$ or $-m_x/m_y$. The equation of the perpendicular is

$$y - y_0 = (-m_x/m_y)(x - x_0),$$

and we must have

$$c_y - y_0 = (-m_x/m_y)(c_x - x_0),$$

or

$$c_y = y_0 - (m_x/m_y)(c_x - x_0).$$

Substitute that into our first equation and get

$$\begin{aligned} r^2 &= (x_0 - c_x)^2 + (y_0 - c_y)^2 \\ r^2 &= (x_0 - c_x)^2 + \left(y_0 - [y_0 - (m_x/m_y)(c_x - x_0)]\right)^2 \\ r^2 &= (x_0 - c_x)^2 + \left((m_x/m_y)(c_x - x_0)\right)^2 \\ r^2 &= (x_0 - c_x)^2 \left(1 + (m_x/m_y)^2\right) \\ r^2 &= (x_0 - c_x)^2 \left(\frac{m_x^2 + m_y^2}{m_y^2}\right) \end{aligned}$$

so that

$$\begin{aligned} (x_0 - c_x)^2 &= r^2 \left(\frac{m_y^2}{m_x^2 + m_y^2}\right) \\ x_0 - c_x &= \pm r \left(\frac{m_y}{\sqrt{m_x^2 + m_y^2}}\right) \end{aligned}$$

or

$$c_x = x_0 \mp r \left(\frac{m_y}{\sqrt{m_x^2 + m_y^2}}\right)$$

and plug that into the equation for c_y too:

$$\begin{aligned}
c_y &= y_0 - (m_x/m_y)(c_x - x_0) \\
&= y_0 - \frac{m_x}{m_y}\left(x_0 \mp r\left(\frac{m_y}{\sqrt{m_x^2 + m_y^2}}\right) - x_0\right) \\
&= y_0 \pm \frac{m_x}{m_y}\left(\frac{m_y r}{\sqrt{m_x^2 + m_y^2}}\right) \\
&= y_0 \pm \frac{m_x r}{\sqrt{m_x^2 + m_y^2}}
\end{aligned}$$