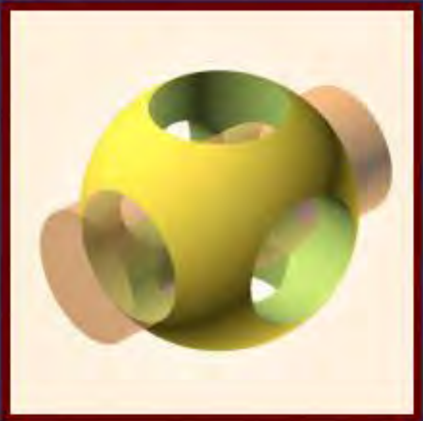
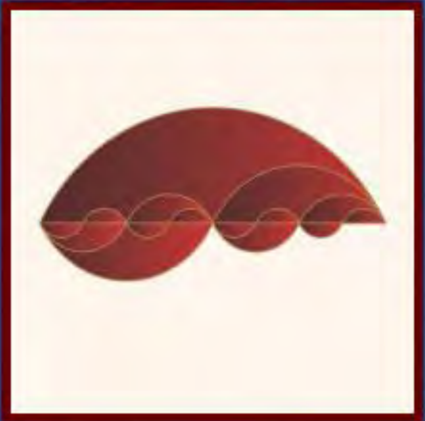
Breath Flute Developer's Guide



GPL 3 Free Software

GFDL



Free Cultural Works APPROVED FOR

Breath Flute Developer's Guide

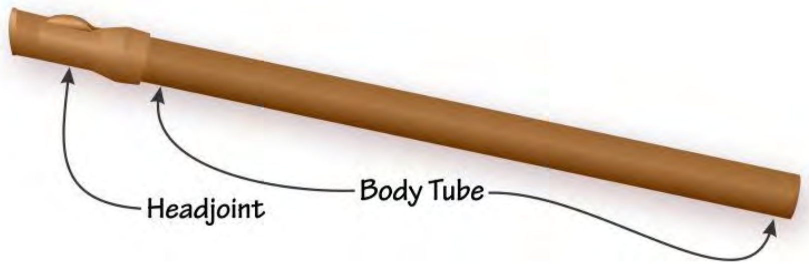
This document provides information and guidance for those who wish to fabricate Breath Flutes, as well as developers who wish to modify the design of the Breath Flute for their own purposes.

The Instrument

The Breath Flute is a wind instrument that lets players with little or no musical experience create deep, resonant music using only their breath. The design inherits from musical instruments of many cultures, including the Australian didgeridoo, overtone flutes from Slovakia, and the Native American flute.

It is ideal for community music gatherings since it can be played solo, in small groups, or as a large ensemble providing harmonic and rhythmic support for other musical instruments. It also has a role in music therapy, therapeutic music-making, and a wide range of personal breath practices such as yoga and meditation.

Breath Flutes are accessible to players with physical limitations, since they have no finger holes and require no finger dexterity. Most players with no musical experience can create music in the first few minutes and experience the physiological benefits associated with resonant breathing the first time they play the instrument.



Most Breath Flutes have a short headjoint section combined with a long body tube. The headjoint is about 5 in / 13 cm long and can be 3D-printed using free, open-source software. The body tube can be as simple as a length of standard pipe, or as complex as you wish to make it (decoration, curved designs, holes for tuning, etc.) The headjoint press-fits onto the body tube. It converts the player's breath into a resonant vibration in the body tube that we hear as sound.

The Project

The Breath Flute Project is a free, open-source software package that has all the files you need to 3D-print a Breath Flute headjoint. You can use the printed instruments yourself, for your music gatherings, and even offer them for sale without paying royalties.

The distribution package contains the source code as well as extensive documentation, letting you improve the design or adapt it for your needs. This can be done using free, open-source software.

You may download, use, modify, and distribute the Breath Flute Project software under a permissive license designed to keep the software free and open. The license encourages anyone to improve the design and share their improvements with the community. See the `License.txt` file in the distribution package for details.

— Clint Goss [clint@goss.com], as of 8/10/2018

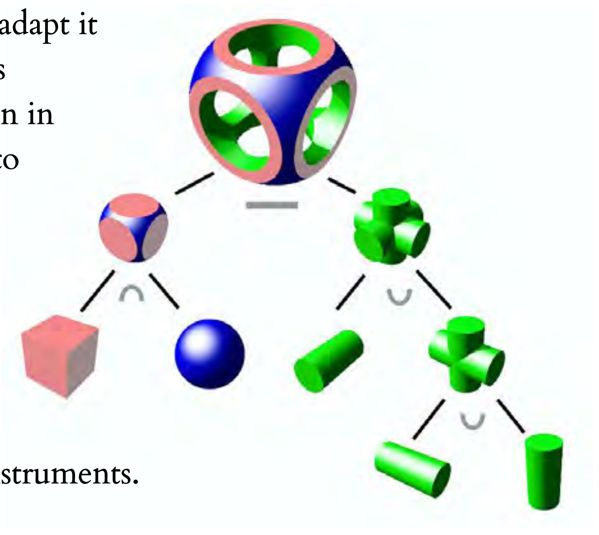
Acronyms and Abbreviations

CSG	Constructive Solid Geometry – the technique used in the Breath Flute for solid modelling.
FFF	"Fused Filament Fabrication" – the method used by many 3D printers. Note that this a copyrighted term of the RepRap project, licensed under *GPL*.
FDM	"Fused Deposition Modeling" – an alternate description of FFF. https://en.wikipedia.org/wiki/Fused_deposition_modeling
S3D	"Simplify3D" – slicer software.
SLA	"Stereolithography" – an optical fabrication resin printing technology for fabricating 3D models layer-by-layer using photopolymerization.

Overview

If you wish to print the Breath Flute Project design, you can use pre-compiled STL files from the distribution package. These files describe the shape – the "three-dimensional surface geometry" – of the Breath Flute.[1] You can use these STL files as the starting point for many of the 3D printing processes.[2] The typical process is to convert the STL files into specific machine instructions (typically G-code) that direct the actions of the hardware you are using to fabricate the Breath Flute.

If you wish to simply fabricate (3D-Print) Breath Flutes, you don't need to modify any code. The distribution package has pre-built STL files that you can use with your 3D printer.



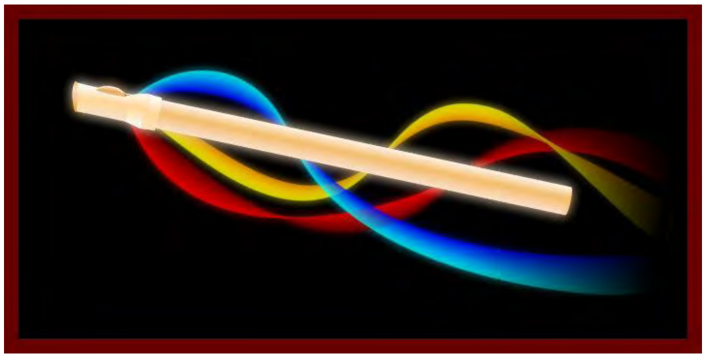
If you would like to improve the design of the Breath Flute or adapt it for your needs, the source code and extensive documentation is provided in the distribution package. The source code is written in OpenSCAD, which uses Constructive Solid Geometry (CSG) to create the geometric model of the Breath Flute. CSG is a technique that combines simple object – cubes, cylinders, prisms pyramids, spheres, cones, etc. – using Boolean operators – union, intersection, and difference – and transformations such as shift, rotate, and extrude. You would render your modified design directly in OpenSCAD to produce the STL files you need to fabricate your modified instruments.

If you do modify the design and fabricate flutes based on your modifications, you are encouraged to get your own Fabricator Code. These two characters printed on the side of the various components identify you and your design, and replace the default "BF" code that identifies "vanilla" flutes fabricated from STL files in a distribution package.

To register your own Fabricator Code, send me an email (clint@goss.com) with (a) your name, (b) your preferred two-letter code, and (c) your contact information, and I will add you to the published list (if possible).

[1] For an overview of STL files, see https://all3dp.com/what-is-stl-file-format-extension-3d-printing/.
[2] Extrusion technologies such as Fused Deposition Modeling, light polymerization systems such as Stereolithography, powder bed systems such as Selective Layer Sinstering, Laminated Object Manufacturing, or the metal alloy systems such as Electron Beam Fabrication. See https://en.wikipedia.org/wiki/3D_printing_processes for an overview.

The Breath Flute Project GPL License



See the ***About the Breath Flute Project GPL License*** section later in this document for information about the License and advice on how to handle common licensing scenarios.

Resources

You can access Web-based information on the Breath Flute as well as the latest distribution package at http://www.BreathFlute.com/. The latest version of this *Breath Flute Developer's Guide* is at:
http://www.BreathFlute.com/pdf/BreathFlute_DevelGuide.pdf

You may also be interested in these documents:

- *A Brief Introduction to the Breath Flute*:
  http://www.BreathFlute.com/pdf/BreathFlute_Introduction.pdf
- *Breath Flute History and Evolution*:
  http://www.BreathFlute.com/pdf/BreathFlute_History.pdf
- If you use Simplify3D as your slicer, you may get something from my *Simplify3D Settings Manual*, available at http://www.BreathFlute.com/pdf/S3D_SettingsCG.pdf
- If you use a Prusa i3 Mk3 printer with Simplify3D for a slicer, you might be interested in:
  - Profiles Comparison: http://www.BreathFlute.com/zip/S3D_Pi3Mk3_CoreProfiles_CG.zip
  - S3D Prusa Profile: http://www.BreathFlute.com/zip/S3D_Pi3Mk3_CG_BF_20180504.zip

Structure of the Distribution

This section outlines the directory structure of the distribution package.

Top-Level Directory

Files in plain text format that relate to the authorship, licensing, and history of development of the Breath Flute Project. A copy of *An Introduction to the Breath Flute* is also provided.

/Doc

Documentation. These are predominantly in PDF format, because of the value of including in-line images with the documentation. These files are licensed under the *GFDL*.

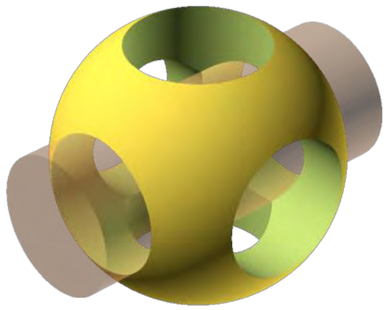
/Img

Digital image files that are part of the Breath Flute Project. Note that these images, and the designs they represent, are part of the distribution and are covered by the *Breath Flute Project GPL License*.

/Misc

Auxiliary files such as the `.log` file of the rendering of all `STL` files in the distribution package, a `PDF` file with templates for cutting out sandpaper for the sanding tools, and the zero-byte `Null.txt` file used to calibrate MD5 utilities.

/Src



Source code for the Breath Flute Project, including OpenSCAD files and batch files for command-line generation of `.stl` files. Third-party packages licensed under the *GPL* and other compatible licenses are contained in sub-directories.

/STL_Core and /STL

These two directories contain `STL` files (see https://en.wikipedia.org/wiki/STL_(file_format)) and related files. Each `STL` file describes the 3D geometry of a given Breath Flute component with a particular set of parameters specific to that component.

The `STL` files are in "stereolithography" format (a.k.a. "standard triangle language" or "standard tessellation language").

> The `/STL_Core` directory contains what might be called "starter" files – the three primary files you would consider printing to fabricate a Breath Flute with one set of options: a Large mouthpiece, no inlay channels, a typical expansion factor, and a ZAlign set to 5. `/STL_Core` also contains the two most useful sanding tools – `SandF` and `SandY` – with the parameters that are most useful.
>
> The `/STL` directory contains all other `STL` files, including Breath Flute components with other options and the files to fabricate the sanding tools that are part of this project.

The file name of each `STL` file indicates the OpenSCAD parameters used for the rendering of that file – see the ***STL File Names*** section for details.

> You will need some application to view the models in the `.stl` files. Most slicer applications (which you will need to 3D print a model) provide a way to view `.stl` files. In addition, I typically use the free STL viewer application provided by 3D-Tool (https://www.3d-tool.com/).

The `/STL` directory also contains `.log` files – plain text files that show the command-line output of the OpenSCAD renderings for the `.stl` files in this directory. The log files indicate the parameters used for each rendering and provide detailed information on the sizes of various components.

/TTF

TrueType font files for the ***Kurinto*** fonts used by the Breath Flute Project. If you are printing models from the `.stl` files in the distribution package, you do not need to install these fonts. However, for rendering new `.stl` files from the OpenSCAD source code, you will need to install the files. See the ***Fonts*** section below.

The Components

This section outlines each of the components that make up the Breath Flute. Most of them are generated directly from the OpenSCAD source code and are printable from `.STL` files provided in the distribution.

If you are printing a Breath Flute, you probably want to start with the `Foot` or `SpEdge` components. These are small, good for testing, and will help dial in which of the `MBIZE` variants you want. For a complete Breath Flute headjoint, you will need to print the `Bird`, `BodyProx`, and `BodyDist` components. You may also wish to print the sanding blocks – the various `Sand*` components – to assist in post-production.

I have included images of many of the components. They are from various past versions of the Breath Flute design, but they give you the general idea of the component.

For a description of the `MBIZE` field, see the ***MBIZE Variants*** section below.

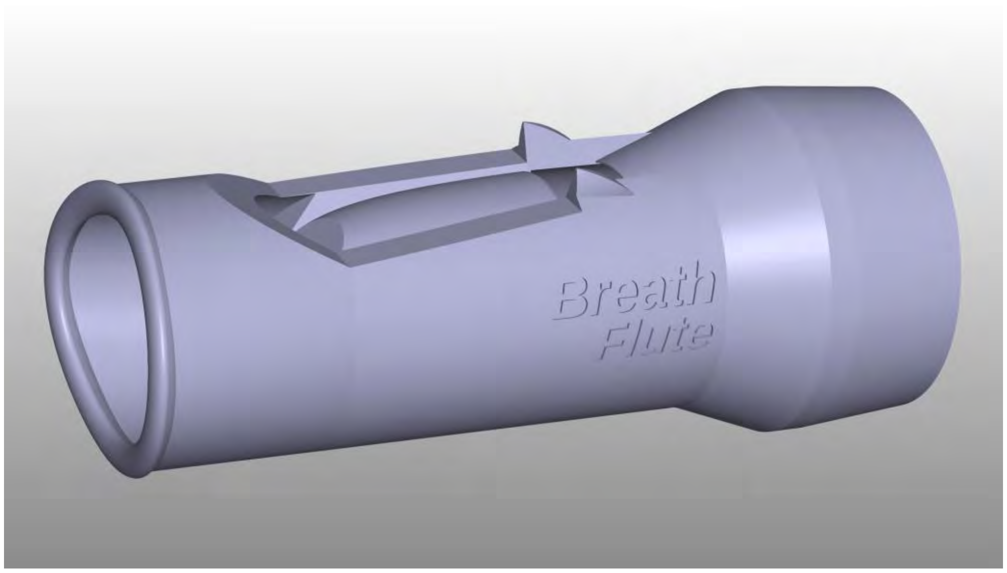
Sandpaper

Some of the components designed for sanding – those with a radius – assume a particular thickness of sandpaper: 0.0105″ or 0.2667 mm. This is the thickness I measured on November 29, 2017 of Mercer Abrasives Aluminum Oxide 220A Grit paper.

Composite Components

Components that may be useful for display rendering and demonstration. They are not typically practical for FDM printing because of: (a) bridging and overhang issues and (b) lack of access to the flue area for post-processing.

Code	Name	MBIZE	Description
HJ	`Headjoint`	M–IZE	The complete Breath Flute Headjoint in one model / print.
BY	`Body`	MBIZE	The `Headjoint` minus the `Bird`.
BB	`BPBird`	M–I––	The `BodyProx` plus the `Bird` (= the `Headjoint` minus the `BodyDist`).

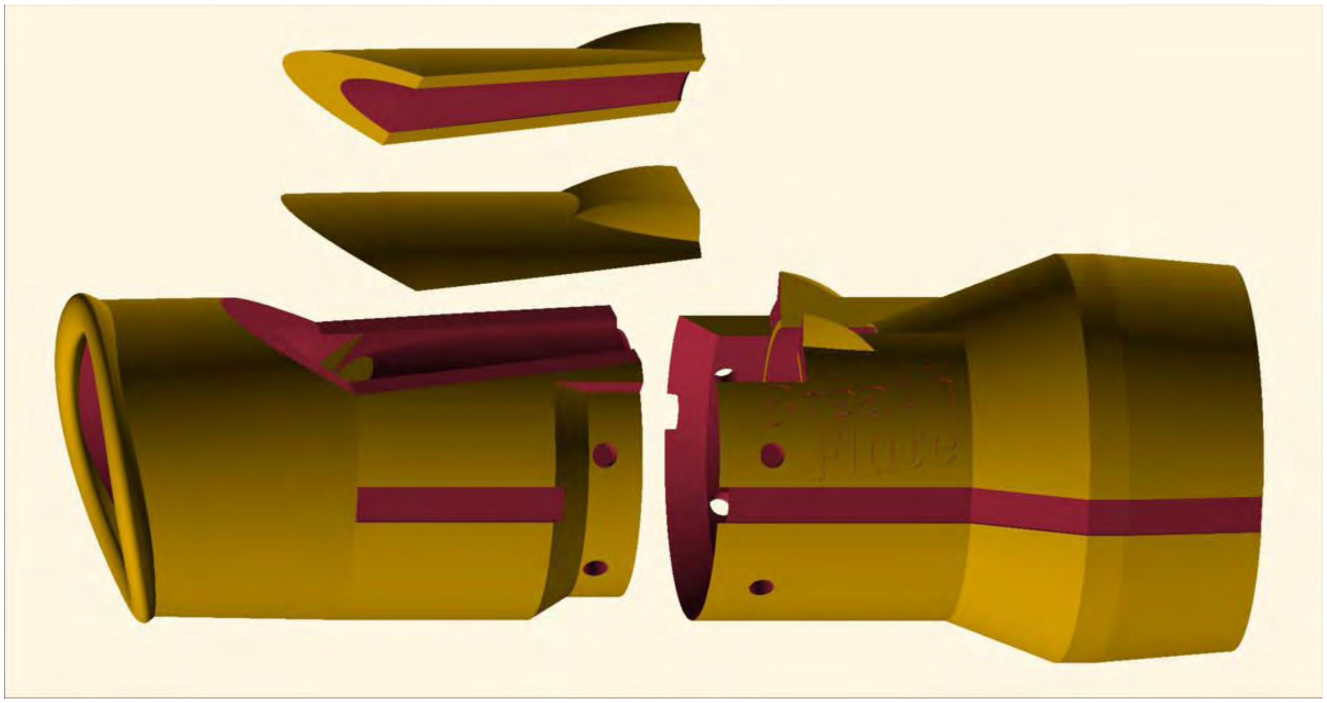


Body component from v61

Main (Split-headjoint) Components

The three components needed to make functional Breath Flutes.

Code	Name	MBIZE	Description
BI	`Bird`	`MB---`	The Bird component – fixed to the `BodyProx` by a lanyard prior to v75 and then by set screws.
BP	`BodyProx`	`MBI--`	Proximal half of the `Body` component.
BD	`BodyDist`	`--IZE`	Distal half of the `Body` component.

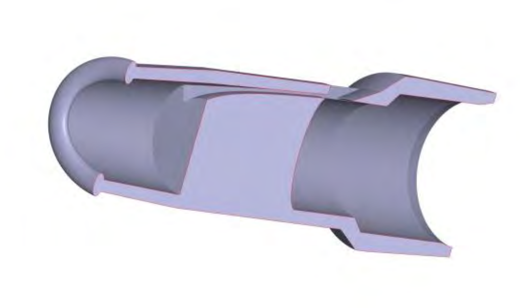


Bird, BodyProx, and BodyDist components from v70
(two Bird components are shown)

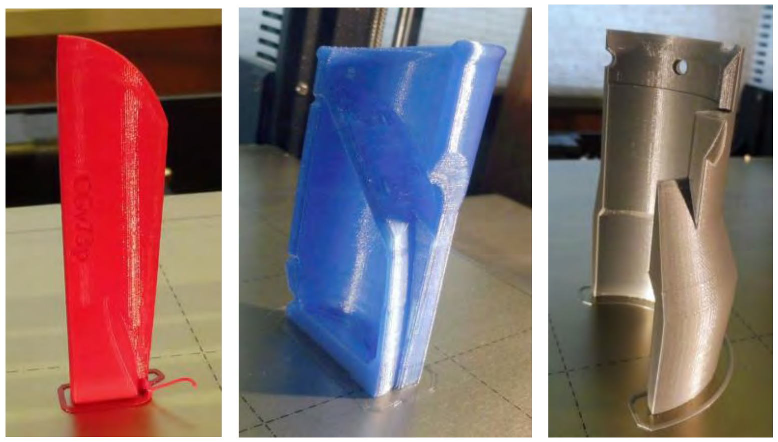
Half-components

Designed for a demo of the `Headjoint` *that is split lengthwise.*

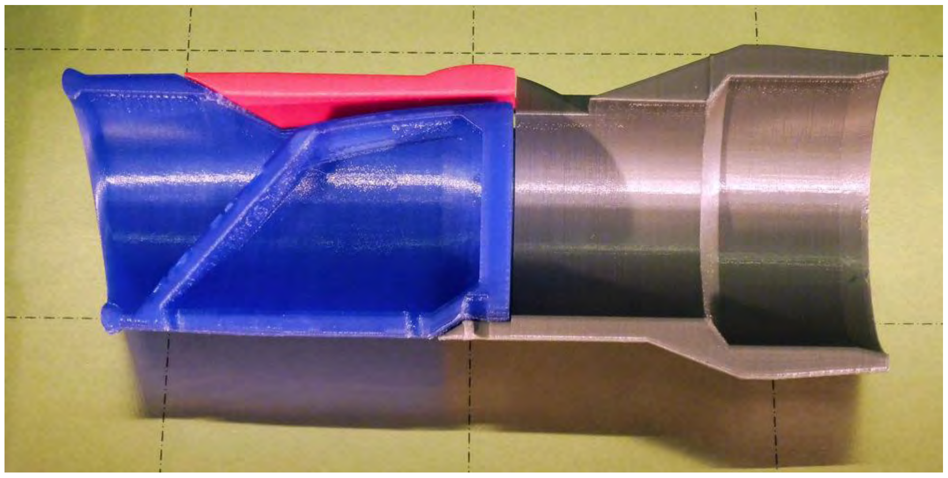
Code	Name	MBIZE	Description
H2	`HeadjointHalf`	M-IZE	Half of a `Headjoint`. Good for images, but not practical to FDM print.
B2	`BirdHalf`	MB---	Half of a `Bird`. Printable and can be assembled with `P2` and `D2`.
Y2	`BodyHalf`	MBIZE	Half of a `Body`. Good for images, but not practical to FDM print.
P2	`BodyProxHalf`	MBI--	Half of a `BodyProx`. Printable and can be assembled with `D2` and `B2`.
D2	`BodyDistHalf`	--IZE	Half of a `BodyDist`. Printable and can be assembled with `P2` and `B2`.



HeadjointHalf component from v40 (prior to the Bird)



v73 BirdHalf, BodyProxHalf, and BodyDistHalf (note the lack of a brim on the Bird!)



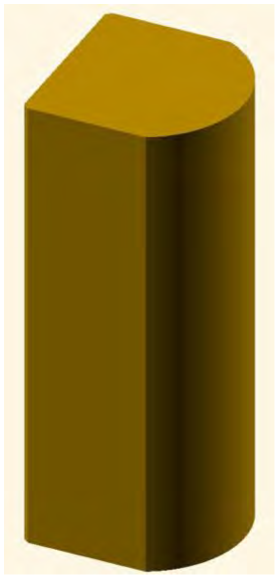
BirdHalf, BodyProxHalf, and BodyDistHalf components bolted together for demonstration.

Sanding Tools

Sanding Blocks, Wedges, and Dowels – useful for finishing printed flutes. Note that the names and codes are (mostly) in alphabetical order, for ease of sorting, and generally proceed from proximal to distal and from Bird to BodyProx to BodyDist.

Code	Name	Done?	MBIZE	Description
SB	`SandB`		-----	Bird Sanding Block Sanding block for the Bird Crown and Nest Rails. This is the inverse of `SandN`. This block matches the inflection angle of the base of the `Bird`.
SF	`SandF`		----E	Flue Sanding Block For the Flue on the underside of the Bird. This component has exactly the shape of the roof of the flue, allowing for a sheet of sandpaper. See the ***Sandpaper*** section above for the specifications of sandpaper thickness.
SG	`SandG`	X	-----	Sanding Guard – To protect the Grip at the bottom of Sanding Dowels in the jaws of a vice grip. This component is designed to crush before the Grip crushes, if too much force is applied to the vice grip.
SH	`SandH`	X	-----	Sanding Handle – A handle that can hold all Sanding Wedges and Dowels.
SN	`SandN`		-----	Nest Sanding Block Sanding block for the Crown and Nest Rails on the `BodyProx` or `Body` components. This is the inverse of `SandB`. This block matches the inflection angle of the base of the `Bird`.
SP	`SandP`		----E	Proximal Junction Sanding Block For the Proximal Junction of the `BodyProx` component.
SQ	`SandQ`		----E	Distal Junction Sanding Wedge For the Distal Junction at the proximal end of the `BodyDist` component.
SS	`SandS`		----E	Sound Chamber Sanding Dowel For the proximal end of the Sound Chamber in the `BodyDist` component. Two are needed to form a complete dowel.
ST	`SandT`		----E	Transition Sanding Wedge For the bevel transition from the Sound Chamber to the Mortice inside the the `BodyDist` component.
SV	`SandV`		-----	V Sanding Wedge A V-shaped wedge useful for sanding off the brim of `BodyDist`. It may be useful for other purposes.

`SY SandY`	`----E`	Mortice Sanding Dowel Semi-cylinder for the Mortice at the distal end of the `BodyDist` component that holds the Body Tube. Two are needed to form a complete dowel.
`SZ SandZ`	`----E`	Bottom Flare Sanding Wedge For the terminal flare at the foot of the entire Headjoint.



v70 SandB component

Service Components

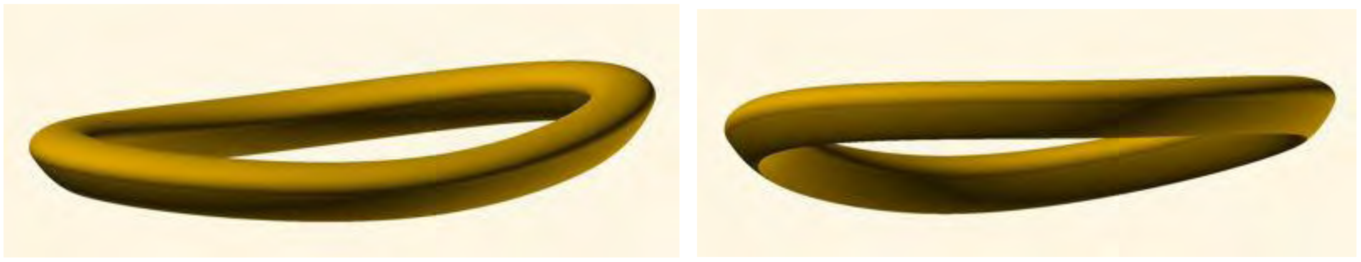
Useful for testing and calibration during fabrication.

Code	Name	MBIZE	Description
FT	Foot	----E	Foot end of the Headjoint – useful for selecting a scale factor.
SE	SpEdge	---Z-	Just the splitting edge area of the BodyDist component, for testing.

Developers' Components

Useful for development and testing. These may be useful for examining a shape, or for faster rendering of a section of the model, but are not intended for fabrication.

Code	Name	MBIZE	Description
LI	Lip	M----	Just the Lip at the top end of BodyProx – useful for development.
FL	Flue	M----	Shows the shape of the Flue – very useful for debugging.
BN	BodyProxN	MB---	BodyProx with no Lip – faster rendering during development.
TH	Throat	-B---	The throat area of the headjoint, optionally showing the pegs for the Bird component. This is useful during development to position elements in the throat area.
JP	JunctionProx	MBI--	The proximal part of the Junction Area – the area between the BodyProx and BodyDist components. Useful for development of Junction Pegs.
JD	JunctionDist	--IZ-	The distal part of the Junction Area. Useful for development of Junction Pegs.

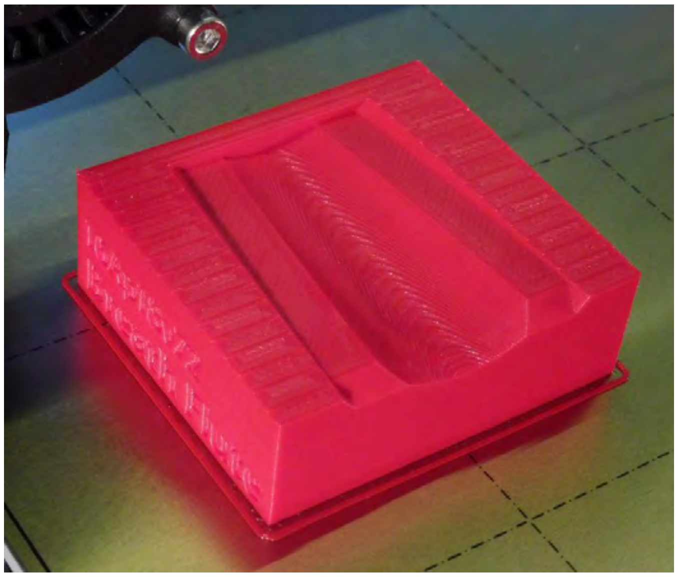


The v75 Lip component in isolation, viewed from above and below

Deprecated / Discontinued / Defunct Components

These components appear in the OpenSCAD code, but are no longer viable.

Code	Name	MBIZE	Description
SL	`Sleeve`	-----	From the ill-fated Sleeve / Insert design of the version 50 era.
IN	`Insert`	-----	From the ill-fated Sleeve / Insert design of the version 50 era.
SX	`SandX`	-----	The old sanding block for the rails of the nest area on the `BodyProx`. Retired in v77 with the Differential Flue Radius, which reduced the "bump" on the floor of the flue so it sits below the Nest rails. This component allowed sanding of the nest rails without affecting the floor of the flue. It is designed to have small strips of sandpaper fixed onto the rails. It also has an end-stop to prevent accidentally sanding the crown area. Note that this was actually called `SandN`, but the name was changed on 6/16/2018 to `SandX` to avoid conflict with the new `SandN` component.


v72 SandX component

Multi-Component Codes

Codes for multiple components. These codes are not printable, but are useful for photos and for `.gcode` *files with multiple components in a single print.*

Z0	A combination of `BodyProx` and `BodyDist`.
Z1	A combination of `BodyProx`, `BodyDist`, and `Bird`.
Z2	A combination of `BodyProxHalf`, `BodyDistHalf`, and `BirdHalf`.

ZZ	A combination of ***all*** components. This code is used, for example, when all components are rendered.

MBIZE Variants

There are five parameters that allow for variations of some of the Breath Flute components:

- The `Mouthpiece` parameter controls the size of the mouthpiece. See the `[M]` section in ***STL File Names*** (below).
- The `BirdStyle` parameter controls the size of the mouthpiece. See the `[bb]` section in ***STL File Names*** (below).
- The `Inlay` parameter controls the width of channels on the sides of some of the component, for decorative inlays. See the `[ii]` section in ***STL File Names*** (below).
- The `ZAlign` parameter controls the precise Z-axis alignment of the splitting edge of the flute. This is critical for getting the best sound from the fabricated instrument. See the `[zz]` section in ***STL File Names*** (below) as well as the ***Z Axis Synchronization*** section that follows.
- The `XYExpand` parameter controls the size of the Mortice at the foot end of the flute in the X-Y plane. This allows adjustments for different size body tubes. See the `[eeee]` section in ***STL File Names*** (below).

To keep track of which settings were used for a particular rendering, the setting used for each of these parameters is inscribed on each of the rendered parts that are affected by that parameter (unless, as with the `BirdStyle` parameter, it is obvious).

STL File Names

`STL` files in a Breath Flute Project distribution have a file names that help keep all the models and their variants organized. The file name, for example …

`BFlute_074a_BD_CG_f125i3z5e5_20180505_1234.stl`

… might seem cryptic. This section describes how to decipher those names, and also indicates the corresponding parameter in the OpenSCAD source code that relates to each of the variants.

The general format of an `STL` file name is:

`BFlute_VVVV_CC_FF_dsss[M][bb][ii][zz][eeee]_YYYYMMDD_hhmm.stl`

The sections in `[brackets]` are optional, and only appear for some of the `STL` files. The sections in colors (other than black) are:

`VVVV` The version of the Breath Flute Project OpenSCAD code that generated the STL file. This is always three digits followed by an optional single letter (e.g. "`074`" or "`129a`").

OpenSCAD parameter: `ProjectVersion` (except without any leading "`0`").
`render.bat` variables: `varProjectVersion24` and `varProjectVersion34`

`CC` A two-character code for the Component. See ***The Components*** above.

OpenSCAD parameter: `Component`
`render.bat` variables: `varComponent` and `varComponentFlag`

`FF` Fabricator Code: two upper-case letters that identify who has fabricated the instrument, by an FDM process or any other manufacturing process. This code is printed on the side of most of the components, as a way to identify who has produced the physical model. A list of fabricator codes currently in use is available at www.BreathFlute.com.

> Note that the special fabricator code "`BF`" is used for the STL files in the standard Breath Flute Project distribution package and may appear on printed models that have not been modified from that distribution.

If you wish to register your own Fabricator Code, send me an email (clint@goss.com) with:

- your name,
- preferred two-letter code, and
- contact information,

and I will add you to the published list (if possible).

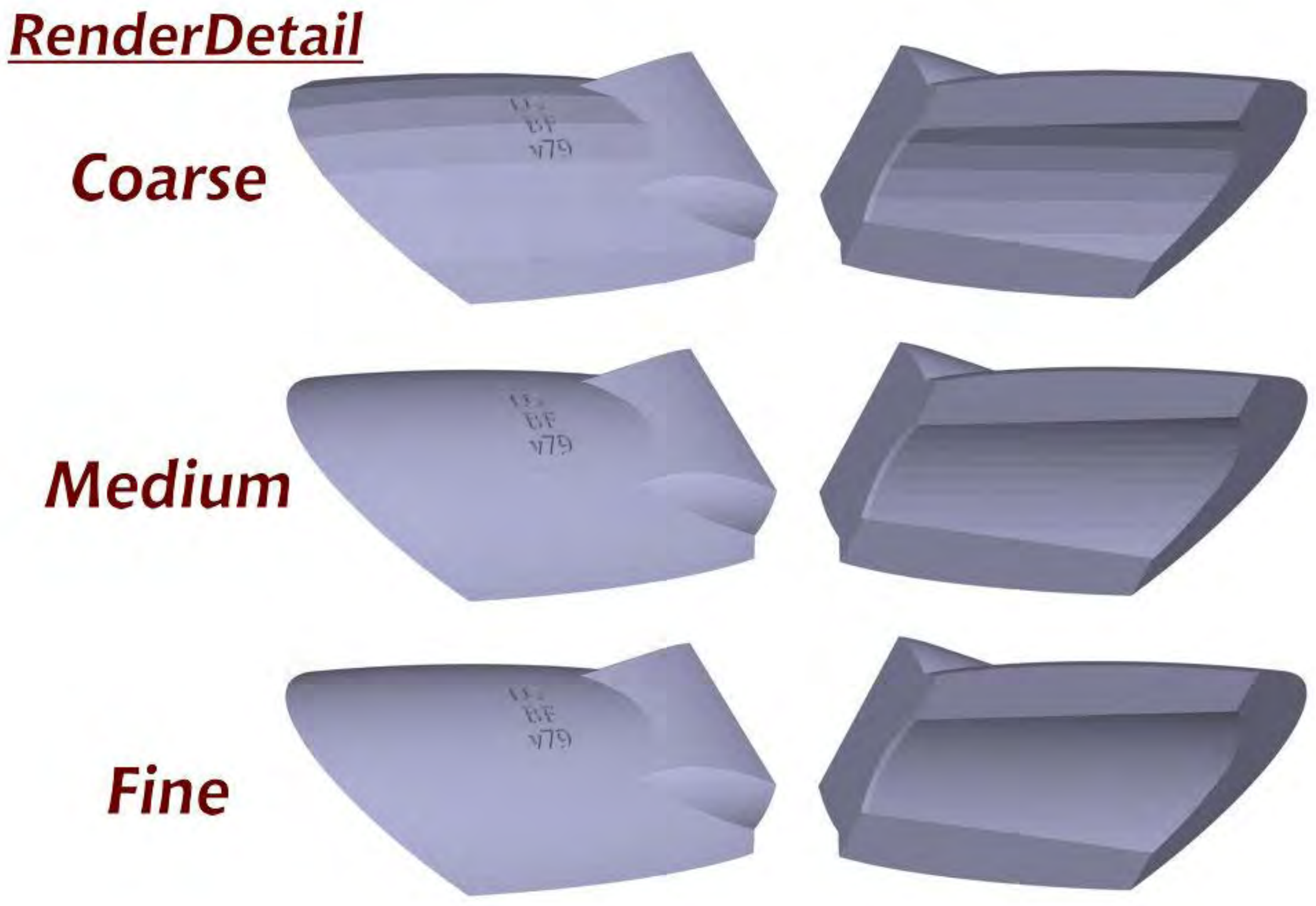
OpenSCAD parameter: `FabricatorCode`
`render.bat` variable: `varFabricatorCode`

d `RenderDetailFlag` – a parameter that controls the number of straight-line segments used to approximate curves. These single-letter flags that are used in `STL` file names run in tandem with the one-word `RenderDetail` values used in the both the OpenSCAD code and the `render.bat` script.

Values for the `RenderDetailFlag` and their corresponding `RenderDetail` values are "`c`" (`Coarse`), "`m`" (`Medium`), "`f`" (`Fine`), or "`s`" (`SuperFine`).

Developers will typically set the OpenSCAD parameter `RenderDetail` to "`Coarse`" during development. This allows them to quickly render and visualize their designs.

Note: the `RenderDetail` setting is not printed on the parts themselves. You will need to use the flag in the STL file name or look at the rendered component to determine what `RenderDetail` setting was used.



Various RenderDetail settings, using the Bird component from v79

For a rendering intended for fabrication, developers should change `RenderDetail` to “`Fine`” (`RenderDetailFlag` = “`f`”). The rendering will take significantly longer, but curves on the fabrication will be far more detailed. Most `STL` files in the distribution package are rendered with `RenderDetail` = “`Fine`”, although some `Coarse` and `Medium` renderings are included (in case you are using them on a machine with limited horsepower).

The difference in size between the different `RenderDetail` settings is dramatic: The size of the STL file increases from `Coarse` to `Medium` by about a factor of × 2 and from `Coarse` to `Fine` by about a factor of × 6–7.

For development work where a finer detail that coarse is needed, the `RenderDetail` setting “`Medium`” is available. This takes longer to render, but is useful for an F6 render, especially if images of the OpenSCAD rendering are being captured (e.g. for documentation).

Finally, the “`SuperFine`” setting affects only the rendering of the `Lip` component. I left “`SuperFine`” for the pre-v75 (needlessly high) `$fn` settings for `Lip` and changed “`Fine`” to one level down (half of `SuperFine`).

OpenSCAD parameter: `RenderDetail` – “`Coarse`”, “`Medium`”, “`Fine`”, or “`SuperFine`”
`render.bat` variables: `varRenderDetail` and `varRenderDetailFlag`

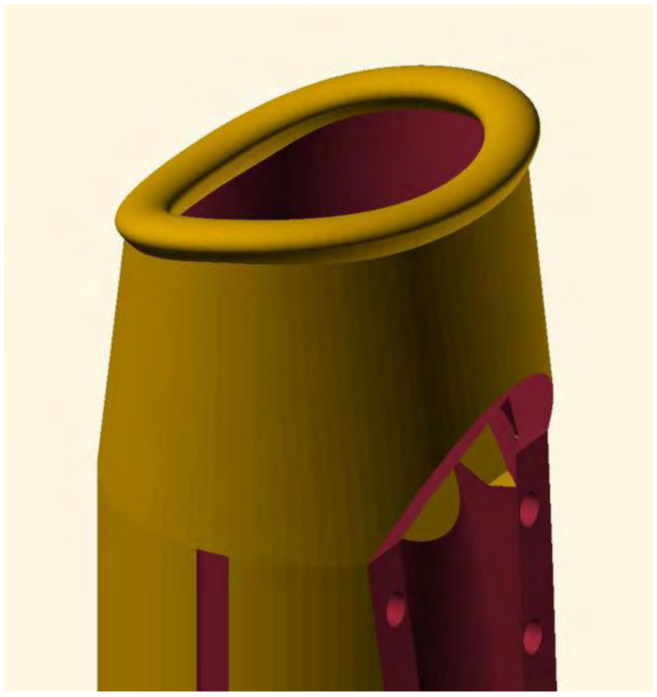
`sss` Size of the Body Tube. Currently supported values are:
`125` 1¼″ PVC tubing
More sizes are planned in the future …

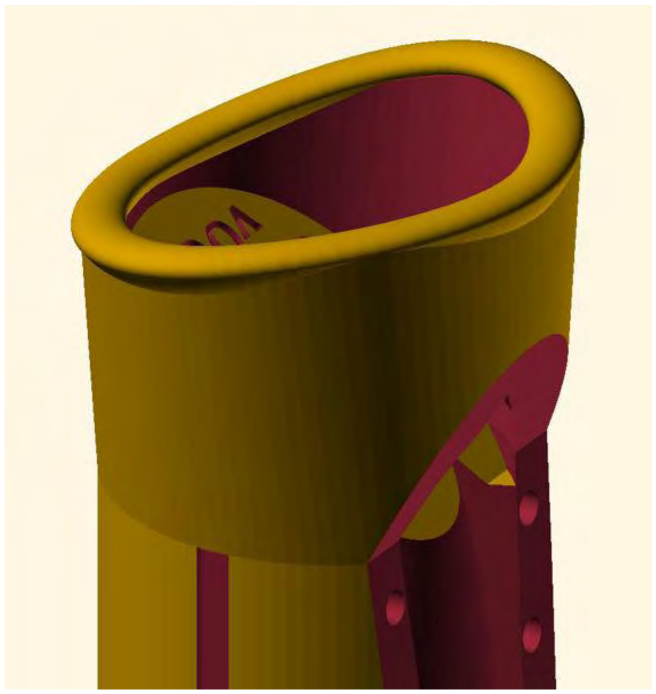
OpenSCAD parameter: `Pipe`
`render.bat` variable: `varPipe`

`[M]` Mouthpiece size (optional). A single, upper-case letter that indicates the size of the mouthpiece and lip plate at the proximal end of the headjoint:

- `S` Small
- `M` Medium
- `L` Large. This was the standard (and only) size prior to v76.
- `X` X-Large

Only present for the `Headjoint(HJ)`, `Body(BY)`, `BPBird(BB)`, `Bird(BI)`, `BodyProx(BP)`, `HeadjointHalf(H2)`, `BirdHalf(B2)`, `BodyHalf(Y2)`, `BodyProxHalf(P2)`, `Lip(LI)`, `Flue(FL)`, `BodyProxN(BN)`, and `JunctionProx(JP)` components.
Only multi-supplied for the `Bird(BI)`, `BodyProx(BP)`, and `Lip(LI)` components.
The `Flue` component is dependent on Mouthpiece size, but only for its shape in the SAC area. The shape of the `Flue` in the Throat area (i.e. underneath the `Bird`) does not change with Mouthpiece size.
Note: `Bird` components must be matched with a `BodyProx` component that has the same Mouthpiece size setting, because the shape at the proximal end of the `Bird` changes as the Mouthpiece size changes!
OpenSCAD parameter: `MouthpieceSize`
`render.bat` variable: `varMouthpieceSize`

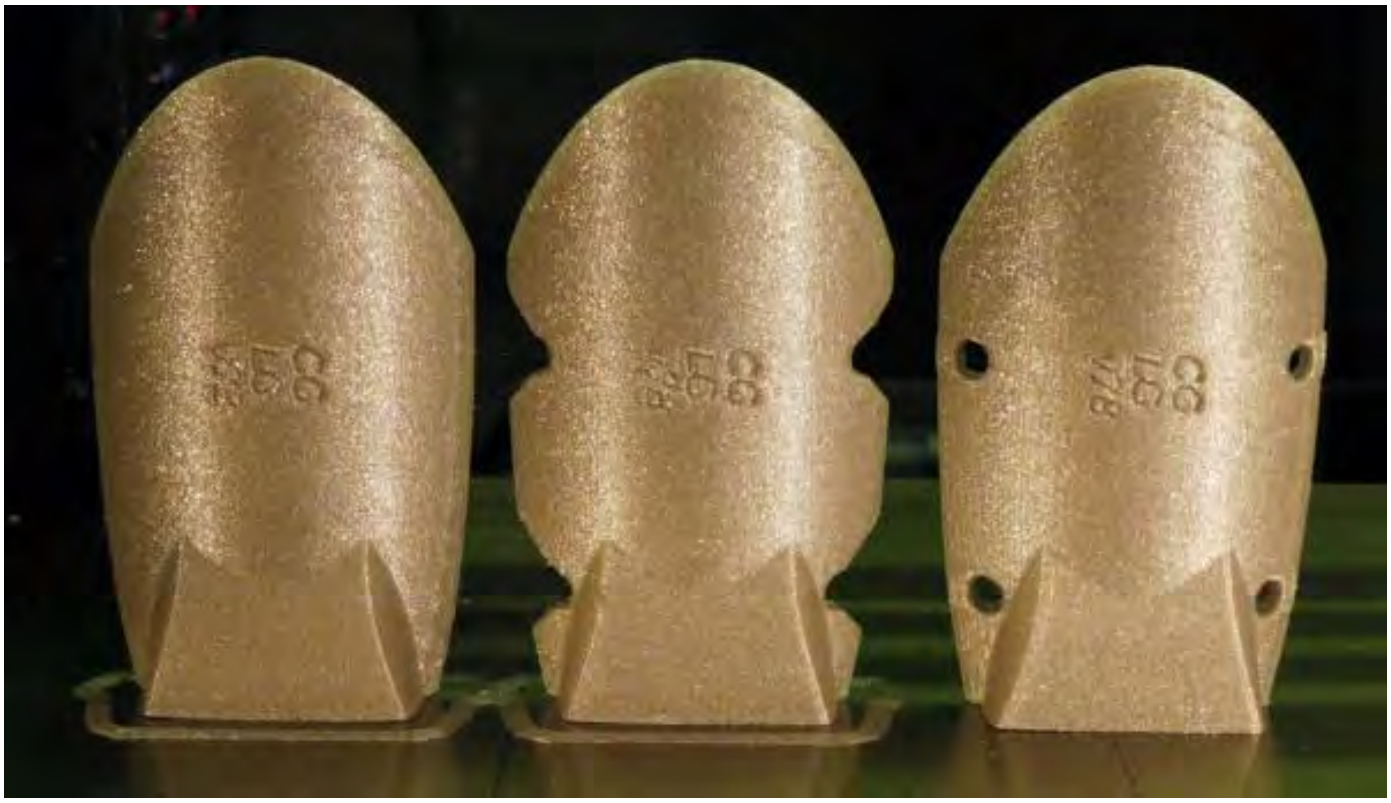

Small Mouthpiece


X-Large Mouthpiece

`[bb]` Bird Style (optional). Indicates which of these bird styles to use:

`bT`	Tied-on Bird – no peg holes for set screws
`bP`	Peg Holes – for using four 6-32 set screws to fix the `Bird` onto the `BodyProx` component
`bW`	Wedges – the same as Peg Holes with the addition of side wedges to avoid breaking the delicate outboard areas of the Bird when screwing in the set screws.

Only present for the `Body(BY)`, `Bird(BI)`, `BodyProx(BP)`, `BirdHalf(B2)`, `BodyHalf(Y2)`, `BodyProxHalf(P2)`, `BodyProxN(BN)`, `Throat(TH)`, and `JunctionProx(JP)` components.
Only multi-supplied for the `Bird(BI)` and `BodyProx(BP)` components.
OpenSCAD parameter: `BirdStyle`
`render.bat` variable: `varBirdStyle`


BirdStyle=T (Tied-on)   BirdStyle=W (Wedges)   BirdStyle=P (Peg Holes)

`[ii]` Inlay (optional). Indicates the size of any channels for decorative inlay on the sides of this model.

`i0`	No inlay channels
`i1`	1⁄16″ inlay channels
`i2`	2⁄16″ (1⁄8″) inlay channels
	. . .
`i8`	8⁄16″ (½”) inlay channels

Only present for the `Headjoint(HJ)`, `Body(BY)`, `BPBird(BB)`, `BodyProx(BP)`, `BodyDist(BD)`, `HeadjointHalf(H2)`, `BodyHalf(Y2)`, `BodyProxHalf(P2)`, `BodyDistHalf(D2)`, `JunctionProx(JP)`, and `JunctionDist(JD)` components.
Only multi-supplied for `BodyProx(BP)` and `BodyDist(BD)` components.
Note that the setting for this parameter is not imprinted on the sides of the instrument, since the width of the inlay can easily be measured on those components where it is important.
OpenSCAD parameter: `Inlay`
`render.bat` variable: `varInlay`

`[zz]` ZAlign (optional). Addresses the issues described in the ***Z Axis Synchronization*** section below.

`z0`	Layers are aligned to a 100-micron boundary.
`z1`	Layers are aligned to a `x.x10` mm boundary. This would occur, for example, if your Layer Height is 0.10 mm and your First Layer is set to 1.10mm (or 110% of the Layer Height).
`z2`	Layers are aligned to a `x.x20` mm boundary. This would occur, for example, if your Layer Height is 0.10 mm and your First Layer is set to 1.20mm (or 120% of the Layer Height).
	. . .
`z9`	Layers are aligned to a `x.x90` mm boundary. This would occur, for example, if your Layer Height is 0.10 mm and your First Layer is set to 1.90mm (or 190% of the Layer Height).

Only present for the `Headjoint(HJ)`, `Body(BY)`, `BodyDist(BD)`, `HeadjointHalf(H2)`, `BodyHalf(Y2)`, `BodyDistHalf(D2)`, `SpEdge(SE)`, and `JunctionDist(JD)` components.
Only multi-supplied for `BodyDist(BD)` and `SpEdge(SE)` components.
OpenSCAD parameter: `ZAlign`
`render.bat` variable: `varZAlign`

[eeee] XYExpand (optional) – XY Expansion Factor in per mille (‰ – 1 part per 1,000 … or ⅒%). This is used to scale the size of the Mortice at the foot end of the flute in the X-Y plane to allow adjusting for different size body tubes. This parameter replaces the prior use of slicer shrinkage / expansion factors, which would change the size of the entire headjoint. However, for the Mortice, this parameter is also largely deprecated in favor of tapping out the end of the Mortice using a reaming tool or sanding it with the SandY component.

This parameter can be used to scale (typically shrink) the SandF, SandS, and SandY components, to accommodate thicker sandpaper. It affects the diameter of the cylindrical part of these components.

The XYExpand parameter can be used to scale (typically enlarge) the SandP component, to accommodate thicker sandpaper. It affects the diameter of the cylindrical area of SandP.

For the SandQ, SandT, and SandZ components, the XYExpand parameter affects only the diameter of the bottom-most guide cylinder (and possibly the overall height of the sanding wedge). You can use this parameter to produce a part with a modified (typically smaller) guide cylinder.

The format for XYExpand is e###.

- Values of ### in the range 001–500 provide for X-Y expansion of 1–500‰ (0–50%) corresponding to an expansion multiplier of × 1.001 through × 1.500.
- Values of ### in the range 501–999 provide for X-Y ***shrinkage*** of 499–1‰ (49.9–0.1%). These values correspond to expansion multipliers in the range of × 0.501 through × 0.999.

The XYExpand value is inscribed on the model of all component that are affected by it. It is shown as “e###”. An XYExpand value of 000 is a special case: it is inscribed as “e0” rather than “e000”.

The shrinkage factors are particularly useful for the SandS and SandY components.

Only present for the Headjoint(HJ), Body(BY), BodyDist(BD), HeadjointHalf(H2), BodyHalf(Y2), BodyDistHalf(D2), SandF(SF), SandY(SP), SandQ(SQ), SandS(SS), SandT(ST), SandY(SY), SandZ(SZ), and Foot(FT) components.

Only multi-supplied for BodyDist(BD), SandS(SS), and SandY(SY), and Foot(FT) components.

OpenSCAD parameter: XYExpand
render.bat variable: varXYExpand

YYYY Date/time stamp of when the STL file was rendered from the OpenSCAD model.
MMDD The hh and mm use a 24-hour clock.
hhmm These are represented in the OpenSCAD parameters RenderADate, RenderODate, RenderS1Date, RenderS2Date, and RenderS3Date.

Variants in the Distribution Package

This section lists the variants of each component provided with the current distribution package. Each variant is available as a `.STL` file in either the `/STL` or the `/STL_Core` directory in the distribution package (the files that are in `/STL_Core` are noted below).

Currently, all renderings in the distribution package use:

- a Detail of "`f`" (Fine),
- a pipe size set for a 1¼″ PVC pipe, and with
- the Fabricator code is set to "`BF`" (the Fabricator code for the pristine, un-modified `STL` files as distributed).

A † mark following a file indicates that the `STL` file is generated in the `Scope=Dev` setting in the `render.bat` script. The `Scope` variable is used to control how many renderings are done. Setting `Scope=All` generates all the files needed for a standard distribution package and takes quite a while.

Composite Components

These components are intended to be used for demonstration purposes only (e.g. in a display rendering for a presentation). They tend not to be practical for FDM printing, primarily because of overhand situations. The distribution package has only a few versions of each of them, with some typical parameters.

Headjoint – HJ

`BFlute_vvvv_HJ_FF_f125Li3z5e000_YYYYMMDD_hhmm.stl`

`BFlute_vvvv_HJ_FF_c125Li3z5e000_YYYYMMDD_hhmm.stl`

`BFlute_vvvv_HJ_FF_m125Li3z5e000_YYYYMMDD_hhmm.stl`

Body – BY

`BFlute_vvvv_BY_FF_f125Li3z5e000_YYYYMMDD_hhmm.stl`

`BFlute_vvvv_BY_FF_c125Li3z5e000_YYYYMMDD_hhmm.stl`

`BFlute_vvvv_BY_FF_m125Li3z5e000_YYYYMMDD_hhmm.stl`

BPBird – BB

`BFlute_vvvv_BB_FF_f125Li3_YYYYMMDD_hhmm.stl`

`BFlute_vvvv_BB_FF_c125Li3_YYYYMMDD_hhmm.stl`

`BFlute_vvvv_BB_FF_m125Li3_YYYYMMDD_hhmm.stl`

Main Components

Bird – BI

The Tied-on bird is the most viable version of the Bird design. Renderings for this component are provided in the distribution package for all size mouthpieces for a Tied-on Bird.

The Peg and Wedge variants are provided only for the Large mouthpiece.

`BFlute_vvvv_BI_FF_f125LbP_YYYYMMDD_hhmm.stl`

`BFlute_vvvv_BI_FF_f125LbT_YYYYMMDD_hhmm.stl` † *(located in /*`STL_Core`*)*

`BFlute_vvvv_BI_FF_c125LbT_YYYYMMDD_hhmm.stl` *(located in /*`STL_Core`*)*

`BFlute_vvvv_BI_FF_m125LbT_YYYYMMDD_hhmm.stl` *(located in /*`STL_Core`*)*

`BFlute_vvvv_BI_FF_f125LbW_YYYYMMDD_hhmm.stl`

`BFlute_vvvv_BI_FF_f125MbT_YYYYMMDD_hhmm.stl`

`BFlute_vvvv_BI_FF_f125SbT_YYYYMMDD_hhmm.stl`

`BFlute_vvvv_BI_FF_f125XbT_YYYYMMDD_hhmm.stl`

BodyProx – BP

The Tied-on bird is the most viable version of the Bird design. Renderings for this component are provided in the distribution package for all size mouthpieces for a Tied-on Bird with no inlays.

The Peg and Wedge variants are provided only for the Large mouthpiece with no inlay.

Inlay variants are also provided only for the Large mouthpiece with a tied-on bird.

Note that the `BirdStyle` variants of "P" (Pegs) and "W" (Wedge) should render identically.

`BFlute_vvvv_BP_FF_f125LbPi0_YYYYMMDD_hhmm.stl`

`BFlute_vvvv_BP_FF_f125LbTi0_YYYYMMDD_hhmm.stl` † *(located in /*`STL_Core`*)*

`BFlute_vvvv_BP_FF_c125LbTi0_YYYYMMDD_hhmm.stl` *(located in /*`STL_Core`*)*

`BFlute_vvvv_BP_FF_m125LbTi0_YYYYMMDD_hhmm.stl` *(located in /*`STL_Core`*)*

`BFlute_vvvv_BP_FF_f125LbTi2_YYYYMMDD_hhmm.stl`

`BFlute_vvvv_BP_FF_f125LbTi3_YYYYMMDD_hhmm.stl`

```
## Not provided – should be the same as _BP_FF_f125LbPi0_
## BFlute_vvvv_BP_FF_f125LbWi0_YYYYMMDD_hhmm.stl
```

`BFlute_vvvv_BP_FF_f125MbTi0_YYYYMMDD_hhmm.stl`

`BFlute_vvvv_BP_FF_f125SbTi0_YYYYMMDD_hhmm.stl`

`BFlute_vvvv_BP_FF_f125XbTi0_YYYYMMDD_hhmm.stl`

BodyDist – BD

The `e010` and `e020` variants increase the diameter of the Mortice area is increased by 1% and 2%, respectively. This can be used to compensate for shrinkage during fabrication or other artifacts that tend to make the junction between the headjoint and the body tube a very tight fit, requiring excessive post-processing with the `SandY` tool.

The version with no inlay is the primary option in the distribution package. These "no-inlay" renderings are provided for all combinations of:

- `ZAlign` values 0 and 5, and
- `XYExpand` values corresponding to expansions of the Mortice diameter that holds the Body Tube of 0.0%, 0.4%, 0.8%, 1.2%, 1.6%, and 2.0%.

In addition, these combinations are also provided:

- `ZAlign` values 2 and 7, and
- `XYExpand` values corresponding to expansions of the Mortice diameter that holds the Body Tube of 0.0% and 0.8%.

Two additional renderings are provided for inlay parameters of 2 and 3.

```
BFlute_vvvv_BD_FF_f125i0z0e000_YYYYMMDD_hhmm.stl
BFlute_vvvv_BD_FF_f125i0z0e004_YYYYMMDD_hhmm.stl
BFlute_vvvv_BD_FF_f125i0z0e008_YYYYMMDD_hhmm.stl        †
BFlute_vvvv_BD_FF_f125i0z0e012_YYYYMMDD_hhmm.stl
BFlute_vvvv_BD_FF_f125i0z0e016_YYYYMMDD_hhmm.stl
BFlute_vvvv_BD_FF_f125i0z0e020_YYYYMMDD_hhmm.stl

BFlute_vvvv_BD_FF_f125i0z2e000_YYYYMMDD_hhmm.stl
BFlute_vvvv_BD_FF_f125i0z2e008_YYYYMMDD_hhmm.stl

BFlute_vvvv_BD_FF_f125i0z5e000_YYYYMMDD_hhmm.stl
BFlute_vvvv_BD_FF_f125i0z5e004_YYYYMMDD_hhmm.stl
BFlute_vvvv_BD_FF_f125i0z5e008_YYYYMMDD_hhmm.stl        †    (located in /STL_Core)
BFlute_vvvv_BD_FF_c125i0z5e008_YYYYMMDD_hhmm.stl             (located in /STL_Core)
BFlute_vvvv_BD_FF_m125i0z5e008_YYYYMMDD_hhmm.stl             (located in /STL_Core)
BFlute_vvvv_BD_FF_f125i0z5e012_YYYYMMDD_hhmm.stl
```

`BFlute_vvvv_BD_FF_f125i0z5e016_YYYYMMDD_hhmm.stl`

`BFlute_vvvv_BD_FF_f125i0z5e020_YYYYMMDD_hhmm.stl`

`BFlute_vvvv_BD_FF_f125i0z7e000_YYYYMMDD_hhmm.stl`

`BFlute_vvvv_BD_FF_f125i0z7e008_YYYYMMDD_hhmm.stl`

`BFlute_vvvv_BD_FF_f125i2z0e000_YYYYMMDD_hhmm.stl`

`BFlute_vvvv_BD_FF_f125i3z0e000_YYYYMMDD_hhmm.stl`

Half-Components

These components are designed for display rendering for demonstration, but are not typically practical for FDM printing. The distribution package has only one version of each of them, with "typical" parameters.

HeadjointHalf – H2

`BFlute_vvvv_H2_FF_f125Li3z5e000_YYYYMMDD_hhmm.stl`

BirdHalf – B2

`BFlute_vvvv_B2_FF_f125LbP_YYYYMMDD_hhmm.stl`

`BFlute_vvvv_B2_FF_f125LbT_YYYYMMDD_hhmm.stl`

BodyHalf – Y2

`BFlute_vvvv_Y2_FF_f125LbPi3z5e000_YYYYMMDD_hhmm.stl`

`BFlute_vvvv_Y2_FF_f125LbTi3z5e000_YYYYMMDD_hhmm.stl`

BodyProxHalf – P2

`BFlute_vvvv_P2_FF_f125LbPi3_YYYYMMDD_hhmm.stl`

`BFlute_vvvv_P2_FF_f125LbTi3_YYYYMMDD_hhmm.stl`

BodyDistHalf – D2

`BFlute_vvvv_D2_FF_f125i3z5e000_YYYYMMDD_hhmm.stl`

Sanding Tools

Many (but not all) of the Sanding Tool components are configured by a the `XYExpand` parameter, which affects the diameter of particular diameters on each of the tools. The distribution package contains renderings for the sizes I have found most useful in finishing Breath Flutes.

SandB – SB – Bird Sanding Block

`BFlute_vvvv_SB_FF_f125_YYYYMMDD_hhmm.stl`

SandF – SF – Flue Sanding Block

`BFlute_vvvv_SF_FF_f125e000_YYYYMMDD_hhmm.stl` *(located in /`STL_Core`)*

`BFlute_vvvv_SF_FF_f125e960_YYYYMMDD_hhmm.stl`

The diameter of the cylindrical area is reduced by 4% to accommodate thicker sandpaper.

`BFlute_vvvv_SF_FF_f125e980_YYYYMMDD_hhmm.stl`

The diameter of the cylindrical area is reduced by 2% to accommodate thicker sandpaper.

SandG – SG

(this component is not yet implemented)

SandH – SH

(this component is not yet implemented)

SandN – SN

`BFlute_vvvv_SN_FF_f125_YYYYMMDD_hhmm.stl`

SandP – SP

`BFlute_vvvv_SP_FF_f125e000_YYYYMMDD_hhmm.stl`

`BFlute_vvvv_SP_FF_f125e020_YYYYMMDD_hhmm.stl`

The diameter of the cylindrical area is increased by 2% to accommodate thicker sandpaper.

`BFlute_vvvv_SP_FF_f125e040_YYYYMMDD_hhmm.stl`

The diameter of the cylindrical area is increased by 4% to accommodate thicker sandpaper.

SandQ – SQ

`BFlute_vvvv_SQ_FF_f125e000_YYYYMMDD_hhmm.stl`

`BFlute_vvvv_SQ_FF_f125e960_YYYYMMDD_hhmm.stl`

The diameter of the guide cylinder at the bottom is reduced by 4%.

`BFlute_vvvv_SQ_FF_f125e980_YYYYMMDD_hhmm.stl`

The diameter of the guide cylinder at the bottom is reduced by 2%.

SandS – SS

`BFlute_vvvv_SS_FF_f125e000_YYYYMMDD_hhmm.stl`

`BFlute_vvvv_SS_FF_f125e960_YYYYMMDD_hhmm.stl`
The diameter of the cylindrical area is reduced by 4% to accommodate thicker sandpaper.

`BFlute_vvvv_SS_FF_f125e980_YYYYMMDD_hhmm.stl`
The diameter of the cylindrical area is reduced by 2% to accommodate thicker sandpaper.

SandT – ST

`BFlute_vvvv_ST_FF_f125e000_YYYYMMDD_hhmm.stl`

`BFlute_vvvv_ST_FF_f125e960_YYYYMMDD_hhmm.stl`
The diameter of the guide cylinder at the bottom is reduced by 4%.

`BFlute_vvvv_ST_FF_f125e980_YYYYMMDD_hhmm.stl`
The diameter of the guide cylinder at the bottom is reduced by 2%.

SandV – SV

Note that `XYExpand` is not an official parameter for this component, but its value is inscribed on the side of the tool. Also, the angle of the sanding bevel of this tool – by default 45° – is also shown on the side of the tool. Changing this value requires editing the OpenSCAD code.

`BFlute_vvvv_SV_FF_f125_YYYYMMDD_hhmm.stl`

SandY – SY

`BFlute_vvvv_SY_FF_f125e000_YYYYMMDD_hhmm.stl`

`BFlute_vvvv_SY_FF_f125e960_YYYYMMDD_hhmm.stl`
The diameter of the cylindrical area is reduced by 4% to accommodate thicker sandpaper.

`BFlute_vvvv_SY_FF_f125e980_YYYYMMDD_hhmm.stl` *(located in /`STL_Core`)*
The diameter of the cylindrical area is reduced by 2% to accommodate thicker sandpaper.

SandZ – SZ

`BFlute_vvvv_SZ_FF_f125e000_YYYYMMDD_hhmm.stl`

`BFlute_vvvv_SZ_FF_f125e960_YYYYMMDD_hhmm.stl`
The diameter of the guide cylinder at the bottom is reduced by 4%.

`BFlute_vvvv_SZ_FF_f125e980_YYYYMMDD_hhmm.stl`

The diameter of the guide cylinder at the bottom is reduced by 2%.

Service Components

Foot – FT

The `e010` and `e020` variants increase the diameter of the Mortice area is increased by 1% and 2%, respectively. This can be used to compensate for shrinkage during fabrication or other artifacts that tend to make the junction between the headjoint and the body tube a very tight fit, requiring excessive post-processing with the `SandY` tool.

```
BFlute_vvvv_FT_FF_f125e000_YYYYMMDD_hhmm.stl
BFlute_vvvv_FT_FF_f125e004_YYYYMMDD_hhmm.stl
BFlute_vvvv_FT_FF_f125e008_YYYYMMDD_hhmm.stl
BFlute_vvvv_FT_FF_f125e010_YYYYMMDD_hhmm.stl
BFlute_vvvv_FT_FF_f125e012_YYYYMMDD_hhmm.stl
BFlute_vvvv_FT_FF_f125e016_YYYYMMDD_hhmm.stl
BFlute_vvvv_FT_FF_f125e020_YYYYMMDD_hhmm.stl
```

SpEdge – SE

These small components are designed to help determine the correct `ZAlign` parameter for a given fabrication setup. You could easily print all of them on a single print run.

```
BFlute_vvvv_SE_FF_f125z0_YYYYMMDD_hhmm.stl
BFlute_vvvv_SE_FF_f125z1_YYYYMMDD_hhmm.stl
BFlute_vvvv_SE_FF_f125z2_YYYYMMDD_hhmm.stl
BFlute_vvvv_SE_FF_f125z3_YYYYMMDD_hhmm.stl
BFlute_vvvv_SE_FF_f125z4_YYYYMMDD_hhmm.stl
BFlute_vvvv_SE_FF_f125z5_YYYYMMDD_hhmm.stl
BFlute_vvvv_SE_FF_f125z6_YYYYMMDD_hhmm.stl
BFlute_vvvv_SE_FF_f125z7_YYYYMMDD_hhmm.stl
BFlute_vvvv_SE_FF_f125z8_YYYYMMDD_hhmm.stl
BFlute_vvvv_SE_FF_f125z9_YYYYMMDD_hhmm.stl
```

Developers' Components

Lip – LI

This component might be useful to check out the shape of the just the `Lip` in its various sizes.

```
BFlute_vvvv_LI_FF_f125L_YYYYMMDD_hhmm.stl                †
BFlute_vvvv_LI_FF_f125M_YYYYMMDD_hhmm.stl
BFlute_vvvv_LI_FF_f125S_YYYYMMDD_hhmm.stl
BFlute_vvvv_LI_FF_f125X_YYYYMMDD_hhmm.stl
```

Flue – FL

The `Flue` component does change shape at the proximal end, based on the size of the Mouthpiece. However, since this component is not printed (and not even printable in many fabrication situations), we only provide the variant for a large mouthpiece.

```
BFlute_vvvv_FL_FF_f125L_YYYYMMDD_hhmm.stl                †
```

BodyProxN – BN

This component is only intended to provide a way to render faster when developing in OpenSCAD. It is not useful as a printed or even rendered model.

Throat – TH

This component shows just the area surrounding the Bird Pegs on the `BodyProx` component. It is useful during development when configuring the depth and angle of Bird Pegs, and it might be useful for demonstration purposes. Note that the "P" and "W" settings for `BirdStyle` should render identically.

```
BFlute_vvvv_TH_FF_f125bP_YYYYMMDD_hhmm.stl
BFlute_vvvv_TH_FF_f125bT_YYYYMMDD_hhmm.stl
```

JunctionProx – JP

A "typical" configuration for this development component.

```
BFlute_vvvv_JP_FF_f125LbTi3_YYYYMMDD_hhmm.stl            †
```

JunctionDist – JD

A "typical" configuration for this development component.

```
BFlute_vvvv_JD_FF_f125i3z5_YYYYMMDD_hhmm.stl             †
```

Deprecated / Discontinued / Defunct Components

Renderings are not provided in the distribution package.

Sleeve – SL

Insert – IN

SanxX – SX

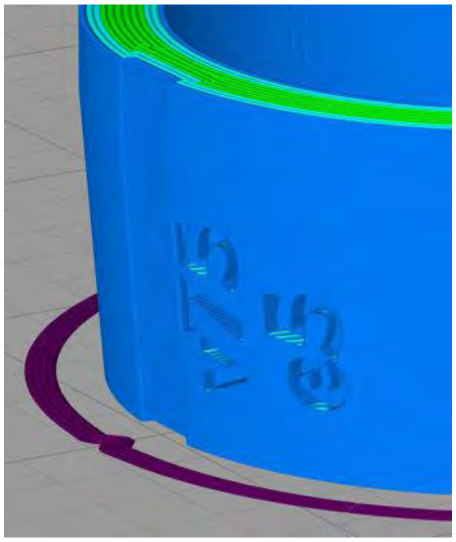
Identification

Many of the Breath Flute components have inscriptions that identify various parameters settings used for that component. These inscriptions help keep printed parts organized, and avoid mismatches that can occur if you try to combine components from different versions of the Breath Flute.

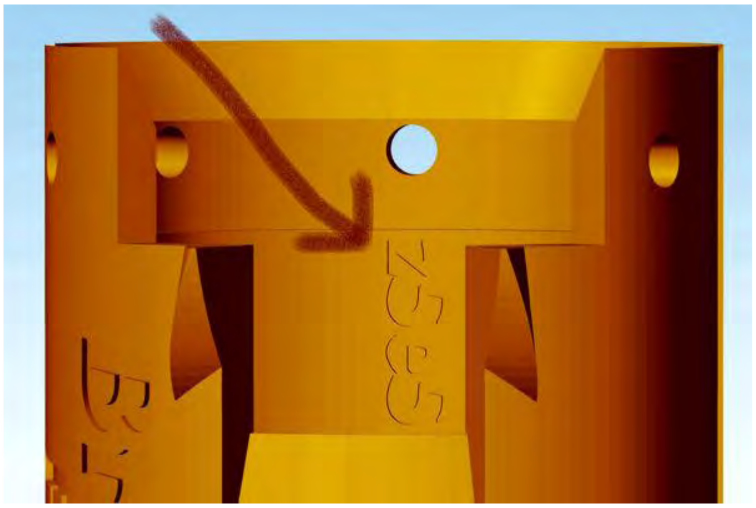
Here are some examples:

Breath
Flute
.com
11/17/17
v67

The `Foot` component is inscribed on the outside with the version and `XYExpand` parameter setting, since this is a test-only component. This is shown at the right for the v75 `Foot` component with an `XYExpand` setting of 005.



Settings for "production" components have the relevant settings inscribed on the inside. For example, the `BodyDist` component has an inscription on the inside wall of the sound chamber, visible by looking into the sound hole of the flute. It gives the values for the `ZAlign` and `XYExpand` setting for that render:



Fonts

The `/TTF` directory of the distribution package has TrueType font files for the ***Kurinto Sans*** fonts used by the Breath Flute Project. If you are printing models from the `.stl` files in the distribution package, you do not need to install these fonts.

However, if you are rendering the OpenSCAD source code into `.stl` files (whether or not you are modifying the code), you will need to install some fonts on your system so that they are accessible to OpenSCAD at render time.

> Note that, if you wish to get exactly the same font rendering as the `.stl` files in the distribution package, you should ***install the version of the fonts that came with that distribution package.***

For more information on the Kurinto fonts (as well as the latest version of the complete set of fonts), visit www.Kurinto.com. The Kurinto project provides a general-purpose set of fonts covering all the 136,755 characters in all the 139 scripts from the world's writing systems defined in the Unicode 10.0 standard. This document is primarily set in Kurinto Book.

For information on font licensing, see the ***How does the License apply to the fonts that are included in the distribution package?*** section below.

Installation

The fonts are provided as basic TrueType™ `.ttf` files. There are many excellent Web-based resources on how to install TrueType fonts on various systems.

Development Workflow

This section offers some suggestions on doing development work on the code, and provides a very brief outline about how I have been developing and make design changes to the Breath Flute model. If may be helpful if you wish to modify the design.

As a first step, please:

- Update the `Authors.txt` file with your name and contact information. You may enter your information in any of the section, including (if you will be doing modifications to the source code) the first section that lists copyright holders. You should ***not*** need to modify the `License.txt` files, since it refers to `Authors.txt`.
- Install the needed fonts, as described in the ***Fonts*** section (above).

Even if you don't plan to release your modifications or offer fabricated models to others, it's a good idea to register that yourself in the copyright list at the beginning.

Note that files with a `.txt` extension have CR-LF line terminators (the Windows standard). Please maintain that standard when modifying these files, to that downstream tools can reliably determine what changes have been made by contributors.

Throughout development and prior to release, please update the `ChangeLog.txt` file as appropriate. See https://en.wikipedia.org/wiki/Changelog for a description of the format. Here is a brief example (note that there are two spaces separating the date from the author and the author from the contact information):

```
2016-04-11  Jon Norris  <jon@jonnorrismusic.com>
            Clint Goss  <clint@goss.com>

  Prototypes v3-11 crafted April 11-16, 2016 at Jon's shop.

2017-02-03  Clint Goss  <clint@goss.com>

  Beginning of development of the v40 series in OpenSCAD.
```

You may also update the copyright year in any file you modify. Please see the section below on ***How should the year or years given in a copyright notice be set?***

Initial development work was done in 2017–2018 on *Beagle* – a 2015-vintage, Windows 7x64 PC with an Intel i7-4790, 3.6 GHz, 4-core, 8 processor CPU with 16 GB physical / 32 GB virtual memory. You may need a moderately hefty desktop with similar or better specs to render the model, especially components that use the compute-intensive and complex Lip at the top of the headjoint. (See ***Complexity of the Geometric Model*** below.)

My primary development tools for the Breath Flute Project are:

- OpenSCAD version [2015, 3, 0];
- 3D-Tool version 12 from 3D-Tool GmbH & Co to visualize `.stl` files;
- Simplify3D version 4.0.1 to "slice" the `.stl` files into `.gcode` files for my particular hardware setup (printer, filament, etc.);
- The fonts in this package were developed in High-Logic FontCreator 11.5.0.2427 Professional Edition.

For development work in OpenSCAD, I set `RenderDetail` to "Coarse" and use F5 to view my changes.

I use a script – `render.bat` – to generate all the needed components and variants of the model. While I am not typically a fan of Windows `BAT` files, this was the only command-line method I found that could execute OpenSCAD as a sub-process and not run into system limitations. Note that my typical run of `render.bat` typically takes about 4-5 hours to render all of the `.stl` files for a standard distribution.

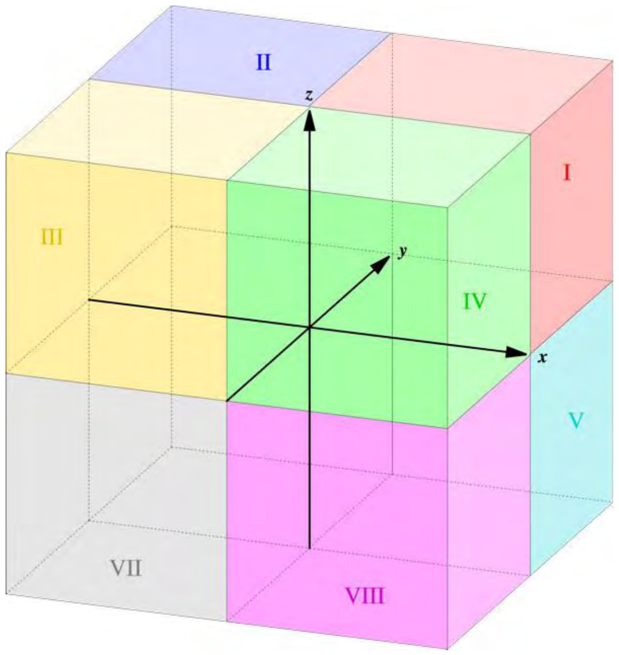
The documentation in `render.bat` outlines my process for bulk-rendering on my Win7x64 system. This includes the process for capturing the log output of the run. Feel free to develop your own process.

I use Simplify3D (currently version 4.0.1) to convert the needed `.stl` files into `.gcode` files for the printer I am using (a Prusa i3 Mk3 as of May 2018). See the ***Slicer Settings*** section (below) for suggestions on configuring your slicer.

I print it!

Units, Orientation, and Axes

This section provides background on how the Breath Flute code and model is oriented in X-Y-Z (Cartesian) as well as Radius-Angle (Polar) coordinate systems. The diagram at the right, from Wikimedia Commons, provides ***some*** assistance for the Cartesian Space axis, but we could really use a better one with a Breath Flute superimposed over the image …



Cartesian Space

Units in Cartesian space – distances, lengths, stations, diameters, etc. – are typically given in mm.

Longitudinal Axis

The longitudinal axis of the flute is oriented along the length of the instrument, with "length" and "station" locations defined along this axis.

The longitudinal axis corresponds to the Z axis of this CSG model, so "Length" and "Station" measurements along the longitudinal axis of the flute correspond to Z-axis values on this CSG model. The longitudinal axis is also called the "Centerline" of the CSG model.

The headjoint is printed standing on its distal end, so the most distal end of the headjoint sits on the X-Y plane with `Z=0` and Z values increase toward the proximal end of the headjoint. The entire headjoint has positive Z.

Vertical Axis

The vertical axis of the flute is oriented from the "front" or "top" of the instrument to the "back" or "bottom". The "front" of the flute is the side with the True Sound Hole and the Bird, and would typically have the finger holes on a Bansuri or a Native American flute.

The vertical axis corresponds to the Y axis of this CSG model. When viewed from the front, Y-axis values increase further from the observer.

The X-Z plane with `Y=0` intersects the centerline of the CSG model, which corresponds to the center of radius of many components such as the Sound Chamber and Mortice.

This gives elements in the front of the instrument – for example, the True Sound Hole and the Bird – negative Y values, and the drain holes on the back of the instrument positive Y values.

Lateral Axis

The lateral axis of the flute is often termed the "left side" and "right side" of the instrument, although with side is left changes depending on the vantage point (e.g. Player's perspective vs. a vantage point from the front side of the instrument).

The lateral axis corresponds to the X axis of this CSG model. When viewed from the player's perspective, X values increase from the player's right to the player's left. When viewed from the perspective of the front of the instrument, X values increase left to right.

As with the vertical axis, the Y-Z plane with `X=0` intersects the centerline of the CSG model.

Polar Coordinates

Angles are typically given in degrees. The reference axis for an angle – i.e. the axis that the angle is a measure from – varies based on context. (That's a nice way of saying "we're not consistant about angle measurement").

For most situations, the reference angle for an axis is fairly straightforward. However, for angles from the longitudinal axis, it might not be obvious. There are two possibilities:

Angle_fromLateral

A number of degrees from the X-Y plane, with an angle of 90 degrees being aligned with the longitudinal axis. From an FDM printing perspective, an `Angle_fromLateral` of 0 produces a flat overhang, and increasing angles become easier to print. The lowest practical `Angle_fromLateral` in this model is ...

```
Angle_fromLateral_MinPractical = 35;
```

Angle_fromCenterline

A number of degrees from the Longitudinal axis. Unsupported surfaces with higher values of `Angle_fromCenterline` become increasingly harder to print. An `Angle_from Centerline` of 90 degrees produces a flat overhang. The highest practical `Angle_fromCenterline` in this model is ...

```
Angle_fromCenterline_MaxPractical = 90 – Angle_fromLateral_MinPractical; // 55
degrees
```

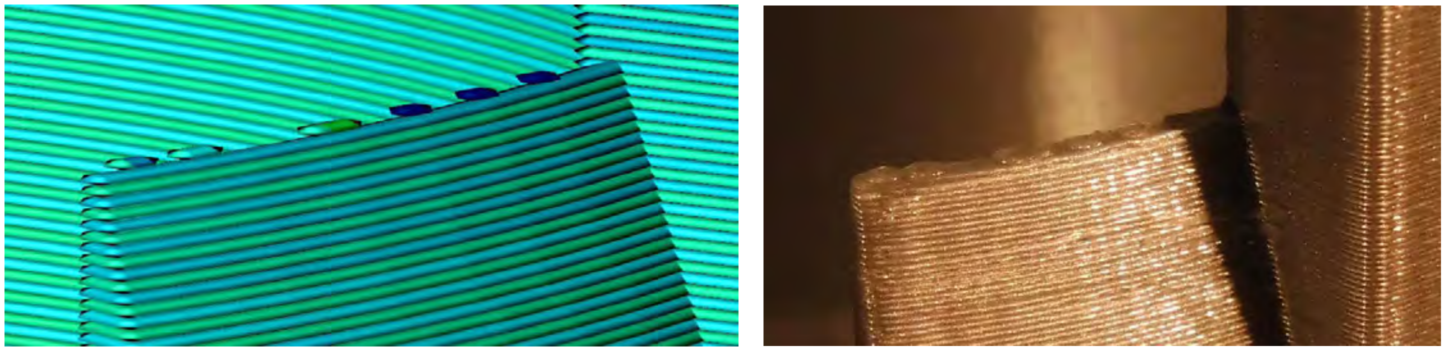
Starting with v77, key angles which were ambiguous were augmented with `_fromLateral` or `_fromCenterline`. The alternative – to modify all angles to the same standard – was too daunting and error-prone).

Z Axis Synchronization

This section describes an irksome little issue with precisely printing the splitting edge of the instrument.

Depending on your slicer settings for Layer Height, First Layer Height, and other Z-axis parameters, you may encounter issue with the integrity of the splitting edge – a critical area for producing the flute's sound.

Because of the curvature of the splitting edge, if the Z-axis position at the point where the splitting edge is printed is within certain bounds, some slicers will create G-code that fabricates as "grit" on the splitting edge. Here are some images from the rendering by Simplify3D and an actual print in PLA showing the "grit" scenario:



Splitting Edge of the v73 BodyDistHalf component, rendered by Simplify3D and printed in PLA.

I have addressed this issue with the `ZAlign` parameter in the OpenSCAD code. The `ZAlign` value (0 through 9) increases the Z-axis position of splitting edge in 10-micron steps. This control seems sufficient to allow all slicers I have tested to generate G-code for the sharp splitting edge that produces one single, continuous filament extrusion.

The `ZAlign` parameter comes into play for the `BodyDist` component, with is the primary component for the splitting edge.

The Trial-and-Error Method

So that you don't have to print entire `BodyDist` components to find the right `ZAlign` setting, there is a `SpEdge` component that prints just the area of the splitting edge. Ten `.stl` renderings of the `SpEdge` component are included in the distribution, and all ten of them can be printed in a few minutes.

This practical, "trial-and-error" method for finding the optimal `ZAlign` setting may be the most efficient for your situation.

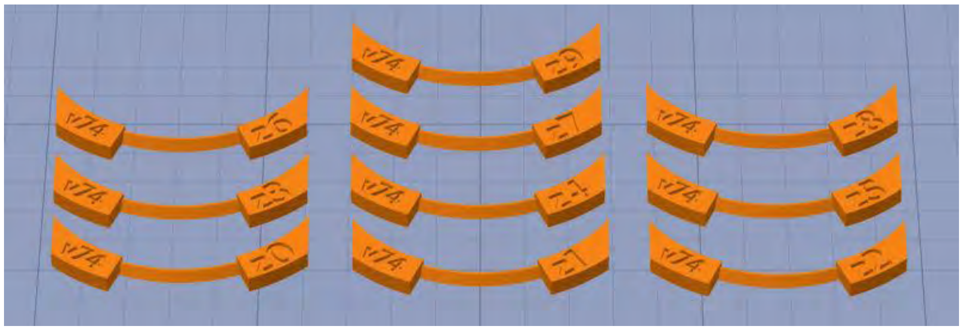
The Predictive Method

Alternately, you can try to predict the best `ZAlign` setting. The rationale goes something like this:

- If you are printing with a Layer Height of `0.10` mm (100 microns) and have a First Layer Height that is a multiple of `0.10` mm, then `ZAlign = 0` is likely to produce good results. The top of the splitting edge is exactly aligned on a `0.10` mm boundary, so the slicer should produce the top of the splitting edge in one, complete extrusion.

- If your combination of Layer Height and First Layer Height settings cause your splitting edge to line up on a boundary that is of the form `x.xZ0` mm, then `ZAlign` = `Z` is likely to produce good results.

Regardless of how you approach choosing your `ZAlign`, the setting is inscribed on all the components where it come into play – `BodyDist`, `SpEdge`, etc. This should avoid confusion and allow you to reliably choose the best `ZAlign` setting down the line.

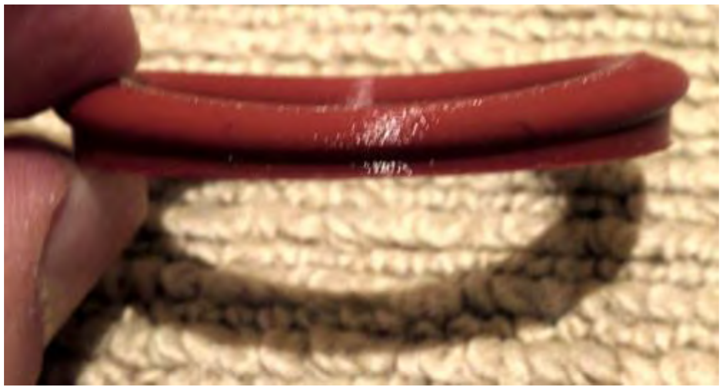


All ten SpEdge components from v74, rendered by Simplify3D.

Complexity of the Geometric Model

The complexity of the 3D geometry of the Breath Flute may cause performance or capacity issues on some systems. If you encounter limitations, here are some suggestions:

- Re-generate the STL files using the "Coarse" setting for `RenderDetail`, and see if the problems persist. Coarse mode uses a substantially lower `$fn` setting when rendering curves, and should produce `.stl` files that avoid capacity limitations.
- If that is successful, you can experiment with increasing `$fn` incrementally until you have reasonably smooth curves.
- The lip at the proximal end of the `BodyProx` component is particularly complex. You can experiment with controlling the `$fn` settings for just that portion of the model, independent of more general `$fn` settings.



A PLA print of just the Lip at the top of the BodyProx component, introduced in v46 using David Eccles's path_extrude() code.

Slicer Settings

Many fabrication processes use a "slicer" – a software component that converts the geometric model represented in a `.stl` file into a sequences of commands to direct the actions of a particular hardware device, such as a 3D printer or a CNC milling machine.

The process of slicing is outside the scope of the Breath Flute Project itself. However, the slicer you use and the settings you provide to it are critical to fabricating the Breath Flute model into a usable musical instrument.

During the development of the Breath Flute Project, I have encountered many issues that are addressed with slicer settings. This section offers some advice in this area. For more information, see these resources (which are not part of the Breath Flute Project, but can help you in setting up a slicer):

- S3D_SettingsCG.pdf
- Breath Flute History document

Z-axis Strength

One of the goals is to maximize the strength of the Breath Flute headjoint along the Z-axis. I anticipate that the maximum load forces will be applied along the Z-axis, and this tends to be the weakest axis in a fabrication process that uses FDM.

I typically print with a Layer Height of 0.10 mm (100μm) for good visual detail and precision on the critical airflow surfaces of the slow air chamber, flue, and splitting edge. However, the Yield Stress of an FDM part is improved 24% at extrusion diameters of 0.20mm ([3DMatter 2015]). As a compromise, I will print the outer perimeter (the "shells") at a Layer Height of 0.10 mm and print infill at 0.20 mm. In Simplify3D, this is done by setting the Infill → Combine parameter set to 2.

The [3DMatter 2015] study also showed that the greatest Yield Strength is attained by printing 3 perimeter shells and using an Infill percentage of 90%, which is how I set up my slicer.

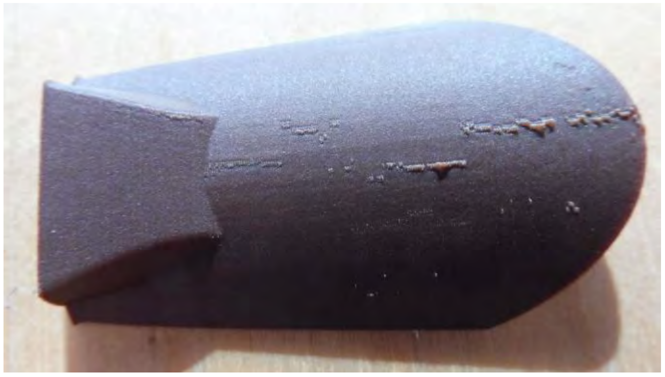
To further handle Z-axis stress forces, I use a rectilinear infill pattern (which appears to be stronger than the honeycomb patterns which are often recommended in popular articles). I also choose overlapping angles than are more in line with the expected direction of bending forces. As angles from the from the X-Z plane, I use: 0°, +50°, -25°, +25°, -50°, and 90° (and then back to the start of this list).

Brims

In order to keep parts stuck to the print bed, I will use a brim on some of the components. This is sometimes problematic on components, such as the `Bird`, where the first layer on the print bed has a critical shape (in this case, the chamfer on the front lip of the `Bird`).

I would recommend extreme care removing these brims. You can use a sharp knife, and carefully post-process the edge to preserve the chamfer angle.

Start / Stop Points



The outer perimeter of a 3D printed model is subject to artifacts at the point where the extruder begins a new layer. If the slicer generates code to begin each layer in the same or similar X-Y coordinates, you can wind up with a print that looks like the print of the `Bird` component at the right (from v60).

Many slicers let you control how start / stop points are chosen on the perimeter – you might select a "random" setting, or you might choose a particular location that is on the back or inside of the component.

Settings for Sanding Tools

For the various sanding tools, I tend to adjust the above settings by using a layer height of 0.20mm and 30% infill.

Post-Processing

Once you print a Breath Flute, you get to customize it in the post-processing phase. Rather than a cookbook How-To on post-processing, this section offers a smattering of useful things I have found along the way.

Sanding Tools

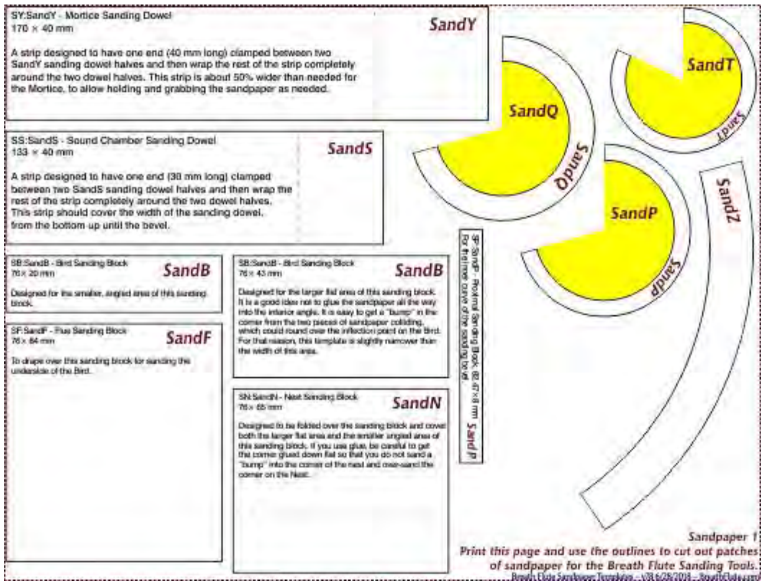
The distribution package for the Breath Flute provides `STL` files for an array of sanding tools. I have developed these tools on an as-needed basis while post-processing Breath Flutes (usually after getting mediocre results after 20 minutes of free-form hand-sanding). Each of the sanding tools is used for one very specific task on a part of a printed Breath Flute.

The distribution package has a `PDF` file with printable templates for cutting out sandpaper of the exact shape needed for the sanding tools.

I typically print the sanding tools in PLA with a 0.20 mm layer height, 30% infill, and 3 solid shells. Some things to consider:

Some of the tools (eg. `SandS` and `SandY`) use removable sandpaper. You probably only need one or two of these. However, for other sanding tools where the sandpaper needs to be glued onto, you may need to 3D print a number of copies.

Sanding Tool Templates


SY:SandY - Mortice Sanding Dowel
170 × 40 mm
A strip designed to have one end (40 mm long) clamped between two SandY sanding dowel halves and then wrap the rest of the strip completely around the two dowel halves. This strip is about 50% wider than needed for the Mortice, to allow holding and grabbing the sandpaper as needed.

SS:SandS - Sound Chamber Sanding Dowel
133 × 40 mm
A strip designed to have one end (30 mm long) clamped between two SandS sanding dowel halves and then wrap the rest of the strip completely around the two dowel halves. This strip should cover the width of the sanding dowel, from the bottom up until the bevel.

SB:SandB - Bird Sanding Block
76 × 20 mm
Designed for the smaller, angled area of this sanding block.

SF:SandF - Flue Sanding Block
76 × 64 mm
To drape over this sanding block for sanding the underside of the Bird.

SB:SandB - Bird Sanding Block
76 × 43 mm
Designed for the larger flat area of this sanding block. It is a good idea not to glue the sandpaper all the way into the interior angle. It is easy to get a "bump" in the corner from the two pieces of sandpaper colliding, which could round over the inflection point on the Bird. For that reason, this template is slightly narrower than the width of this area.

SN:SandN - Nest Sanding Block
76 × 65 mm
Designed to be folded over the sanding block and cover both the larger flat area and the smaller angled area of this sanding block. If you use glue, be careful to get the corner glued down flat so that you do not sand a "bump" into the corner of the nest and over-sand the corner on the Nest.

SP:SandP - Proximal Sanding Block, 82.47×8 mm
For the inner curve of the sanding bevel.

Sandpaper 1
Print this page and use the outlines to cut out patches of sandpaper for the Breath Flute Sanding Tools.
Breath Flute Sandpaper Templates - v8 6/28/2018 - BreathFlute.com

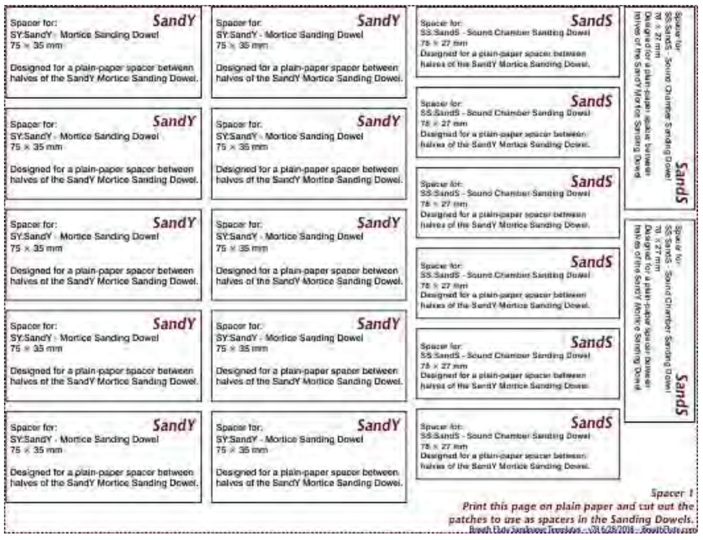
The distribution package has a template file – `BreathFlute_SandingToolTempaltes.pdf`. This document has templates that you can print and use to cut out patches of sandpaper that can be glued to the Sanding Tools used to finish a Breath Flute.

It also has templates for paper that can be used to adjust the diameter of the Sanding Dowels.

You can print this document on U.S. Letter size paper (8½″×11″) or A4 paper (210×297 mm). Each page has a ¼″ margin (in red) that should accommodate all the content. When you print:

1. Turn off all scaling options and print in landscape orientation at the original document size.
2. Measure your printout (with a ruler) to ensure the printed red border measures exactly 8″×10.5″ (203.2×266.7 mm).

The sandpaper shapes needed are either rectangles or frustums – the truncated cone shapes of Sanding Wedges that finish bevels on the Breath Flute. You can temporarily fix the printed template to the back of a sheet of sandpaper – I use binder clips – and cut along the solid lines of the template. Alternately, since there is a 4 mm space between the templates, you can cut between them and trim each one individually.



Spacer for: SY:SandY - Mortice Sanding Dowel 75 × 35 mm SandY
Designed for a plain-paper spacer between halves of the SandY Mortice Sanding Dowel.

Spacer for: SY:SandY - Mortice Sanding Dowel 75 × 35 mm SandY
Designed for a plain-paper spacer between halves of the SandY Mortice Sanding Dowel.

Spacer for: SS:SandS - Sound Chamber Sanding Dowel 78 × 27 mm SandS
Designed for a plain-paper spacer between halves of the SandY Mortice Sanding Dowel.

Spacer for: SS:SandS - Sound Chamber Sanding Dowel 78 × 27 mm SandS
Designed for a plain-paper spacer between halves of the SandY Mortice Sanding Dowel.

Spacer for: SY:SandY - Mortice Sanding Dowel 75 × 35 mm SandY
Designed for a plain-paper spacer between halves of the SandY Mortice Sanding Dowel.

Spacer for: SY:SandY - Mortice Sanding Dowel 75 × 35 mm SandY
Designed for a plain-paper spacer between halves of the SandY Mortice Sanding Dowel.

Spacer for: SS:SandS - Sound Chamber Sanding Dowel 78 × 27 mm SandS
Designed for a plain-paper spacer between halves of the SandY Mortice Sanding Dowel.

Spacer for: SS:SandS - Sound Chamber Sanding Dowel 78 × 27 mm SandS
Designed for a plain-paper spacer between halves of the SandY Mortice Sanding Dowel.

Spacer for: SY:SandY - Mortice Sanding Dowel 75 × 35 mm SandY
Designed for a plain-paper spacer between halves of the SandY Mortice Sanding Dowel.

Spacer for: SY:SandY - Mortice Sanding Dowel 75 × 35 mm SandY
Designed for a plain-paper spacer between halves of the SandY Mortice Sanding Dowel.

Spacer for: SS:SandS - Sound Chamber Sanding Dowel 78 × 27 mm SandS
Designed for a plain-paper spacer between halves of the SandY Mortice Sanding Dowel.

Spacer for: SS:SandS - Sound Chamber Sanding Dowel 78 × 27 mm SandS
Designed for a plain-paper spacer between halves of the SandY Mortice Sanding Dowel.

Spacer for: SY:SandY - Mortice Sanding Dowel 75 × 35 mm SandY
Designed for a plain-paper spacer between halves of the SandY Mortice Sanding Dowel.

Spacer for: SY:SandY - Mortice Sanding Dowel 75 × 35 mm SandY
Designed for a plain-paper spacer between halves of the SandY Mortice Sanding Dowel.

Spacer for: SS:SandS - Sound Chamber Sanding Dowel 78 × 27 mm SandS
Designed for a plain-paper spacer between halves of the SandY Mortice Sanding Dowel.

Spacer for: SS:SandS - Sound Chamber Sanding Dowel 78 × 27 mm SandS
Designed for a plain-paper spacer between halves of the SandY Mortice Sanding Dowel.

Spacer for: SY:SandY - Mortice Sanding Dowel 75 × 35 mm SandY
Designed for a plain-paper spacer between halves of the SandY Mortice Sanding Dowel.

Spacer for: SY:SandY - Mortice Sanding Dowel 75 × 35 mm SandY
Designed for a plain-paper spacer between halves of the SandY Mortice Sanding Dowel.

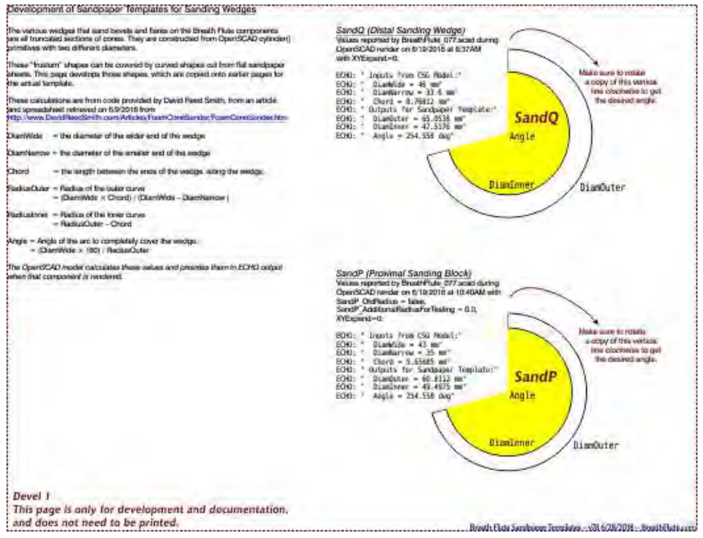
Spacer for: SS:SandS - Sound Chamber Sanding Dowel 78 × 27 mm SandS
Designed for a plain-paper spacer between halves of the SandY Mortice Sanding Dowel.

Spacer for: SS:SandS - Sound Chamber Sanding Dowel 78 × 27 mm SandS
Designed for a plain-paper spacer between halves of the SandY Mortice Sanding Dowel.

Spacer 1
Print this page on plain paper and cut out the patches to use as spacers in the Sanding Dowels.
Breath Flute Sandpaper Templates - v78 6/28/2018 - BreathFlute.com

The dashed purple lines are fold lines – the approximate location you will be folding that template after you cut it out.

If you fabricate Sanding Tools that are scaled to a different size – some of the Sanding Dowels support the `XYExpand` parameter for scaling the size of the dowel – these templates will no longer fit. You may be able to scale your printout to make these templates work. I have not worked out how the scale factors for the `XYExpand` parameter relate to scale factors for printing this document.



Development of Sandpaper Templates for Sanding Wedges

The various wedges that sand bevels and flanks on the Breath Flute components are all truncated sections of cones. They are constructed from OpenSCAD cylinder() primitives with two different diameters.

These "frustum" shapes can be covered by curved shapes cut from flat sandpaper sheets. This page develops those shapes, which are copied onto earlier pages for the actual template.

These calculations are from code provided by David Reed Smith, from an article and spreadsheet retrieved on 6/9/2018 from http://www.DavidReedSmith.com/Articles/FoamConeSander/FoamConeSander.htm

DiamWide = the diameter of the wider end of the wedge

DiamNarrow = the diameter of the smaller end of the wedge

Chord = the length between the ends of the wedge, along the wedge.

RadiusOuter = Radius of the outer curve
= (DiamWide × Chord) / (DiamWide – DiamNarrow)

RadiusInner = Radius of the inner curve
= RadiusOuter – Chord

Angle = Angle of the arc to completely cover the wedge.
= (DiamWide × 180) / RadiusOuter

The OpenSCAD model calculates these values and prints them in ECHO output when that component is rendered.

SandQ (Distal Sanding Wedge)
Values reported by BreathFlute_077.scad during OpenSCAD render on 6/10/2018 at 6:37AM with XYExpand=0.

```
ECHO: " Inputs from CSG Model:"
ECHO: "  DiamWide = 46 mm"
ECHO: "  DiamNarrow = 33.6 mm"
ECHO: "  Chord = 8.76812 mm"
ECHO: " Outputs for Sandpaper Template:"
ECHO: "  DiamOuter = 65.0538 mm"
ECHO: "  DiamInner = 47.5376 mm"
ECHO: "  Angle = 254.558 deg"
```

SandQ
Angle
DiamInner
DiamOuter
Make sure to rotate a copy of this vertical line clockwise to get the desired angle.

SandP (Proximal Sanding Block)
Values reported by BreathFlute_077.scad during OpenSCAD render on 6/10/2018 at 10:40AM with SandP_OldRadius = false, SandP_AdditionalRadiusForTesting = 0.0, XYExpand=0.

```
ECHO: " Inputs from CSG Model:"
ECHO: "  DiamWide = 43 mm"
ECHO: "  DiamNarrow = 35 mm"
ECHO: "  Chord = 5.65685 mm"
ECHO: " Outputs for Sandpaper Template:"
ECHO: "  DiamOuter = 60.8112 mm"
ECHO: "  DiamInner = 49.4975 mm"
ECHO: "  Angle = 254.558 deg"
```

SandP
Angle
DiamInner
DiamOuter
Make sure to rotate a copy of this vertical line clockwise to get the desired angle.

Devel 1
This page is only for development and documentation, and does not need to be printed.
Breath Flute Sandpaper Templates - v78 6/28/2018 - BreathFlute.com

This document also contains a page of templates that can be printed on plain paper and cut out to produce spacers for the Sanding Dowels. You can place one or more of these spacers between the two halves of a Sanding Dowel to make it slightly larger and increase the sanding pressure on the bore you are sanding.

Finally, there are several pages at the end that have the calculations and development of the flat arcs that create the frustum shapes.

The calculations for frustum shapes are from code were provided by David Reed Smith (David@DavidReedSmith.com), from an article and spreadsheet retrieved on 6/9/2018 from:

www.DavidReedSmith.com/Articles/FoamConeSander/FoamConeSander.htm

> Note: You need to use the version of the template that corresponds to the version of the Sanding Tools that you fabricated! Sizes and shapes do change from version to version, and these templates are updated to track those changes.

Sanding Blocks, Dowels, and Wedges

This section contains some notes on the use of the various sanding tools.

- Many of the Sanding Tools can be held by a table-mounted vice. I use a heavy drill-press vice that has a quick-set level. The vice is note even mounted on the table – it is heavy enough to provide a sufficient base for the tool.

- The vice-mounted tools do run the risk of being crushed in the vice. However, I have not had this issue in practice. There is a yet-to-be-designed `SandG` (Sanding Tool Guard) component to protect the vice-grip section these tools, but I have not found the need.
- Alternately you can simply hold the grip portion of the sanding tool in your hand. Gloves do help in this situation. Most sanding tools have eased edges – either small-radius curves or "micro-bevels" – to minimize the hand-trauma.
- The Sanding Dowels – `SandY` and `SandS` – need to be printed in pairs. You need two of them to make a complete sanding dowel. I have typically needed the full-size versions, plus the versions that are shrunk by 2% using the XYExpand parameter of 980 (for 98% size). I'll start with the 98% tools, adding paper spacers as I go from coarse to fine sandpaper, and then switch to the 100% size after I get to 5–8 paper spacers.

Sanding Sticks

Micro-Mesh Colored Sanding Sticks from Woodcraft.com – 5¾″ × ½″

Splitting Edge

NOTE that the curvature of the splitting edge is NOT the same as the tube adjacent to it! Don't just sand away blindly!!

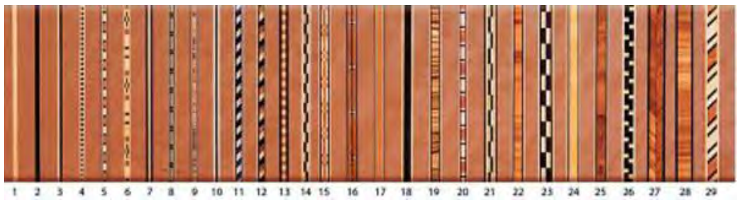
Peg Holes and Set Screws

The three main components of a complete Breath Flute are bolted together with set screws. There are four pairs of aligned "peg holes" on the `Bird` and `BodyProx` components and five pairs at the junction of the `BodyProx` and `BodyDist` components.

All the peg holes are designed for #6-32 set screws – ¼″ long for the five peg holes at the junction and ⅜″ long for the `Bird` peg holes. I use Fastenal SKU 25116 #6-32 × ¼″ Hex Drive Flat Point (i.e. Flat End) Black Oxide Finish Alloy Steel Socket set screws for the peg holes at the junction area. This particular hardware uses a ¹⁄₁₆″ hex drive Allen wrench.

It is possible to simply screw in the set screws into a freshly 3-D printed component. Alternately, you may wish to tap out the peg holes with a metal tap. I hand-drill a square-nose 6-32 tap into each of the holes. You may wish to pre-drill the hole with a drill (#35 or #36 drill bit sizes – approximately 7⁄64″ – are recommended), but this may not be necessary since the holes are already printed to size.

Inlays



The optional inlay channels on the side of the `BodyProx`, `BodyDist`, and other components are designed for decorative inlays that can be added after the model is printed. The `Inlay` parameter to the model determines the width of any inlay channel, and the `[ii]` component of the `.stl` file name indicates what `Inlay` value was used for that rendering.

I use inlays from Sauers & Company – see http://SVeneers.com/ and https://www.Woodcraft.com . Here are the specific Sauers inlay products that correspond to each of the possible Inlay values:

Inlay	Width	Sauers
0	*No inlay channels*	
1	1⁄16″ inlay	*No Sauers products*
2	1⁄8″ inlay	Sauers #1–4
3	3⁄16″ inlay	Sauers #5–10
4	1⁄4″ inlay	Sauers #11–18
5	5⁄16″ inlay	Sauers #19–22
6	3⁄8″ inlay	Sauers #23–26
7	7⁄16″ inlay	*No Sauers products*
8	1⁄2″ inlay	Sauers #27–29

The Body Tube

The mortice at the distal end of Breath Flutes headjoints are designed to fit pipes of standard "Schedule" sizes. The initial headjoint design is for a Schedule pipe size of (nominally) 1¼″, which has an outside diameter of 1.660″ and an inside diameter that depends on the wall thickness for that Schedule.

While you might start with PVC pipe from a local supply store, here are some special manufacturers of pipe that offer pipes that may be more suitable to a "finished" look that may be more appropriate for a musical instrument:

C&S Plastics: Winter Haven, FL. Specialists in PVC pipe extrusion and fittings. I visited their facility in July 2017 and used them for development of prototypes beginning with v79. They produce furniture-grade pipes in various specifications and colors with a glossy finish and no printing on the outside. They also fill orders for small quantities directly from their factory. http://candsplastics.com/

Dara Plastics: Lebanon, PA. Custom extrusion of pipes and tubing in a wide variety of materials. http://daraplastics.com/

Excaliber Extrusions: La Habra, CA. Now part of VIP Rubber. Plastic Pipe manufacturing. https://viprubber.com/

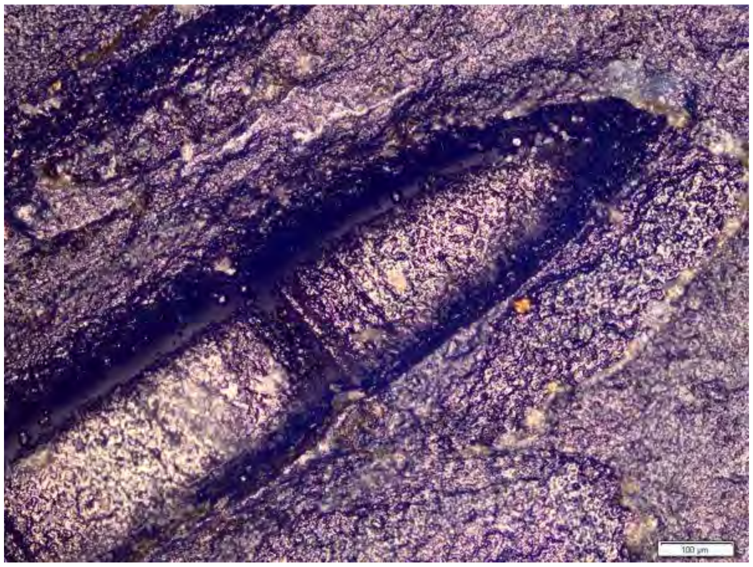
FlexPVC: Pahrump, NV. A supplier of a wide variety of PVC tubes and fittings. https://flexpvc.com/

FormUFit: Lenexa, KS. Furniture Grade PVC pipe, fittings, and accessories. https://formufit.com/

Filament Materials

There is a wide selection of 3D printing filaments in many materials and specifications, and the available offerings seem to be growing hourly. It is (well) beyond the scope of this Developer's Guide to survey 3D printing filaments. However, here are a few issues to consider when choosing a material:

- How easy / reliable is the material for 3D printing?
- Can 3D-printed components made from the material be chemically sanitized?
- At what temperature will 3D-printed components deform? Consider your prints on the dashboard of a car in the summer sun …
- Can 3D-printed components made from the material be sanitized in a dishwasher?
- Is it biodegradable?
- Is it food-safe? Some filaments are listed as safe or approved for food contact. However, please note that ***3D printing with a filament whose material is food-safe does not mean that the finished, 3D-printed component is safe for food contact.*** Here are some references on the topic:
    - Clinton Freeman, *Is 3D Printed PLA food safe?*, November 30, 2012. https://reprage.com/post/36869678168/is-3d-printed-pla-food-safe
    - *PLA and ABS vs food safety?* Discussion on RepRap.org. http://forums.reprap.org/read.php?1,164077,167669#msg-167669
    - R. E. Conn et al, *Safety assessment of polylactide (PLA) for use as a food-contact polymer*, Food and Chemical Toxicology, Volume 33, Issue 4, April 1995, pages 273–283. https://doi.org/10.1016/0278-6915(94)00145-E



One of the main issues with food safety seems to be the small pits and gouges that inevitably form on the surface of an FDM print. These pits catch food particles, which degrade and become host to nasty microbes.

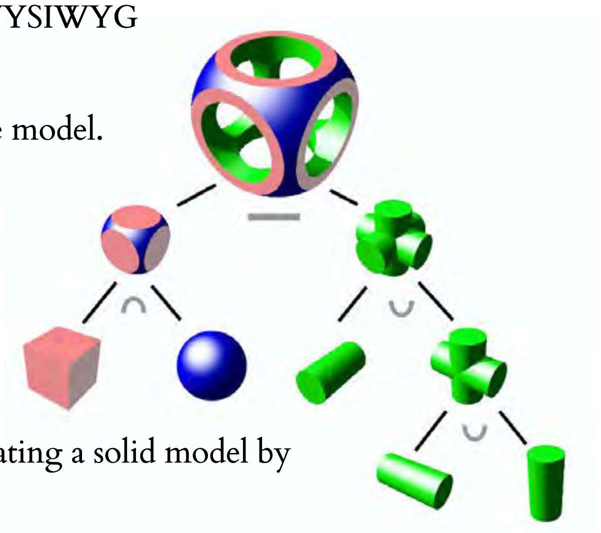
The image at the right was provided by user PomeroyB on the RepRap.org user forum. It shows the top infill from a Stratasys print using ABS. It clearly shows that the surface of the print is pitted. The infill wasn't perfect, so those huge gouges you see are the second-from-the-top layer showing through. The scale bar at the bottom is 100 micrometers.

FAQ

Why didn't I use a WYSIWYG tool for creating the solid geometry model for the Breath Flute?

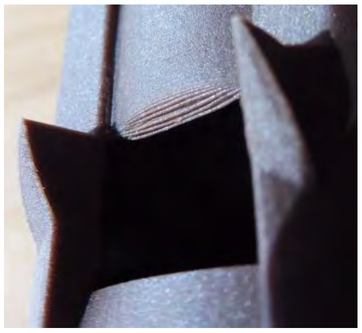
I began with Fusion 360 – a tool that does exactly that. The WYSIWYG approach fell short on a few fronts:

- It did not lend itself easily to very precise dimensions for the model.
- The complexity of the project – with more than a dozen components that were best auto-generated – lent itself to a programmatic tool.
- I'm a programmer by training and experience!



For these reasons, I chose OpenSCAD, which build a model using constructive solid geometry (CSG) – a technique for creating a solid model by combining simple objects using Boolean operators.

Why is the headjoint designed to print as three separate parts?



If the entire headjoint were printed at once several flat overhangs / bridging situations would invite sagging and breaking of the filament strands. See the image at the right, from v60 of the Breath Flute design that shows the distal end of the flue and the proximal end of the sound chamber.

Also, if the `Bird` component were integral with the rest of the headjoint, the flue area could not easily be sanded or cleaned.

Third-party Code

This section logs information about third-party code used in the Breath Flute Project.

NameOfThirdPartyPackage

The title of each section (`NameOfThirdPartyPackage`) names the package of imported third-party code. It also identifies the subdirectory of `/Src` that holds the code, license information, and related documentation for that package. The code and license files are typically kept in pristine form in the subdirectory (aside from modifications for ***Plain Text File Format***).

The code may be used directly (by an OpenSCAD `include <>` or `use <>` directive). Alternatively, the code may be copied and modified in module in `/Src`. To assist in locating instances of the code, you can search for the tag "`Third-party NameOfThirdPartyPackage`", which appears in comments associated with the copied code.

Author(s):	Open-source license under which the code was distributed. These authors are also listed in the `Authors.txt` file at the top level of the distribution package.
Contact:	Best known contact information for the author(s).
Directory:	`/Src/NameOfThirdPartyPackage` – the name of the `/Src` directory that holds the files.
License:	Open-source license under which the code was distributed.
Source:	Where the package came from, including a URL, if available.
Initial date:	The earliest known date that the package was publically made available.
File date:	The date stamp on the file(s) of the package. This may indicate a date of last modification, although it may not be reliable because these dates are sometimes altered by utilities that copies or agregates files. A notation in *(italics)* is used for cases where the file date was not preserved or is not useful.
Release date:	The date of release of the package as we are using it. In the case of GitHub repositories, this may be the date the file(s) were checked in.
Retrieval date:	The date this version of the file was retrieved (downloaded) from the Source.
Local cache:	Clint's working directory for this package. This typically contains all the available versions of the package and ancillary information that is available publicly. This directory is not in the distribution package.
Used:	Where / how the package is used.

Bioinfscripts

This software package contains a module that implements Path Extrusion, which is used to generate the complex curved shape of the lip at the proximal end of the headjoint.

Author(s):	David Eccles ("gringer" on GitHub and Thingiverse)
Contact:	David Eccles of Wellington, New Zealand is a member of GitHub and Thingiverse. He has a Web site at www.gringene.org and may be contacted at bioinformatics@gringene.org.
Directory:	`/Src/Bioinfscripts`
License:	GPLv3
Source:	GitHub – https://github.com/gringer/bioinfscripts
Initial date:	Initial commit of the bioinfscripts project on GitHub of version 1.83: July 1, 2013. Initial commit of `path_extrude.scad` on GitHub: January 5, 2016 @ 12:06 AM EDT.
File date:	*(not preserved)*
Release date:	The file we are using is the ***second*** commit of `path_extrude.scad` on GitHub: January 5, 2016 @ 3:17 PM EDT.
Retrieval date:	February 9, 2017
Local cache:	`/ThirdParty_Code/PathExtrude_DavidEccles`
Used:	Prior to v75, this code was incorporated in the `PathExtrude.scad` module in the main `/Src` directory. However, the only changes to the code were some comments explaining the source of the code. So, while setting up for distrubution, I returned to the initial pristine version of this code and moved the documentation to the `ReadMe.txt` file.

Hex

This software package has code for converting hexidecimal (base 16) numbers.

Author(s):	MichaelAtOz
Contact:	MichaelAtOz is a member of many on-line communities, such as SourceForge, GitHub, and Thingiverse, and may be contacted through those services.
Directory:	`/Src/Hex`
License:	Public Domain (based on a declaration in the source code)
Source:	A comment on the OpenSCAD language itself has example code that was extracted into its own `hex.scad` file. The original comment is at https://github.com/openscad/openscad/issues/877
Initial date:	July 16, 2014 @ 10:33 PM EDT

File date:	*(not applicable)*
Release date:	July 16, 2014 @ 10:33 PM EDT
Retrieval date:	May 24, 2018
Local cache:	`/ThirdParty_Code/HexToDecimal_MichaelAtOz`
Used:	Service routines that have been used at various times in the code.

SimpleSupport

This software package has service routines for generating support structures.

Author(s):	Brad J. Shannon
Contact:	Brad J. Shannon of Brooklyn, New York, is a member of many on-line communities, such as GitHub, Tumblr, and Thingiverse, and may be contacted through those services.
Directory:	`/Src/SimpleSupport`
License:	CC-BY
Source:	https://www.thingiverse.com/thing:1688476
Initial date:	July 25, 2016
File date:	*(not preserved)*
Release date:	July 25, 2016
Retrieval date:	February 11, 2017
Local cache:	`/ThirdParty_Code/SimpleSupport_BradShannon`
Used:	Service routines for generating support structures. The code in this package is not used directly, but is incorportated into `/Src/Support.scad`.

Wedge2D

This software package has service routines for generating two-dimensional wedges.

Author(s):	Nicholas Williams
Contact:	Nicholas Williams is a member of Thingiverse (user "WilliamsOb") and may be contacted through that service.
Directory:	`/Src/Wedge2D`
License:	CC-BY
Source:	http://www.thingiverse.com/thing:1843282
Initial date:	October 22, 2016

File date: *(not preserved)*

Release date: October 26, 2016

Retrieval date: February 7, 2017

Local cache: `/ThirdParty_Code/OpenSCAD_WedgeModule_NicholasWilliams`

Used: Service routines for generating two-dimensional wedges. The code in this package is not used directly, but is incorportated into `/Src/Wedges.scad`.

Wedge3D

This software package has service routines for generating three-dimensional wedges.

Author(s): Alex English

Contact: Alex English of Walla Walla, Washington is a member of Thingiverse (user "AlexEnglish") and may be contacted through that service. He also has a web site at www.AlexEnglish.info.

Directory: `/Src/Wedge3D`

License: CC-BY-SA (declared in the source code file and reported by the source page on Thingiverse as of 5/25/2018)

Source: https://www.thingiverse.com/thing:40245

Initial date: January 4, 2013

File date: *(not preserved)*

Release date: January 12, 2013

Retrieval date: February 7, 2017

Local cache: `/ThirdParty_Code/OpenSCAD_WedgeModule_AlexEnglish`

Used: Service routines for generating three-dimensional wedges. The code in this package is not used directly, but is incorportated into `/Src/Wedges.scad`.

Zazouck

This software package has routines for converting between various formats of numbers.

Author(s): Nathanaël Jourdane

Contact: Software dev. at IRAP (CNRS), Toulouse, France. Email: nathanael@jourdane.net

Directory: `/Src/Zazouck`

License: GPLv3

Source: https://github.com/roipoussiere/Zazouck

Initial date: Initial commit of the Zazouck project on GitHub: December 3, 2013.
Initial commit of `string.scad` on GitHub: January 5, 2014.

File date: January 11, 2014

Release date: January 11, 2014 @ 4:42 PM EDT

Retrieval date: May 24, 2018

Local cache: `/ThirdParty_Code/Zazouck_NathanaelJourdane`

Used: The routines in the `string.scad` module are used directly in various places. Prior to v75, this code was incorporated in the `BreathFluteLib.scad` module in the main `/Src` directory. However, while setting up for distrubution, I returned to the initial pristine version of this code and moved the documentation to the `ReadMe.txt` file.

Plain Text File Format

One might hope (even "expect") that structure of files containing "plain text" – 8-bit, ASCII-encoded characters with no formatting, fonts, images, colors, etc. – would be straightforward. Well, we're not quite there yet. For various arcane technical and historical reasons (see https://en.wikipedia.org/wiki/Text_file and https://en.wikipedia.org/wiki/Newline), there is an array of conventions for the structure of plain text files across different system.

The different formats create angst and confusion – walls of run-on text with missing line breaks, mal-functioning applications, and many other horror stories abound. This issue also affects the value of checksums on files (e.g. the MD5 hash that we use to verify that a file has not been altered), since any change in the bits of a file completely alters the checksum, even if the semantic meaning of the text of the file has not changed.

The predominant plain-text standards today are Unix and DOS. There is also the older (pre-OS X) Apple standard, which I'll call "Mac".

One area of confusion are the control character(s) used to separate lines of text, which are different in these three standards: Unix (LF), DOS (CR+LF), and Mac (CR).

Another issue is the meaning of a newline character(s) in the file. Newlines either separate lines of text, or they terminate lines of text. From the Wikipedia Newline text:

> If a newline is considered a separator, there will be no newline after the last line of a file. Some programs have problems processing the last line of a file if it is not terminated by a newline. On the other hand, programs that expect newline to be used as a separator will interpret a final newline as starting a new (empty) line. Conversely, if a newline is considered a terminator, all text lines including the last are expected to be terminated by a newline. If the final character sequence in a text file is not a newline, the final line of the file

may be considered to be an improper or incomplete text line, or the file may be considered to be improperly truncated.

DOS-based systems (including Microsoft Windows) consider the CR+LF newline sequence to be a separator, and it is common for the last line of text not to be terminated with a CR+LF marker. Many DOS-based text editors (including *Notepad* and *Wordpad*) do not automatically insert a newline sequence on the last line. The Unix standard is to terminate all lines with a newline sequence (a single LF character). I do not know the Mac standard.

Some DOS-based tools – *Notepad* and *Internet Explorer*, in particular – display plain text from a file that uses the Unix standard as a "wall of text" with no line breaks. Some Unix-based tools (*vi*, for example) report errors when the final line of a plain-text file has no newline sequence.

Before v76 of the Breath Flute Project, multiple versions of plain-text files were maintained in parallel. This caused heartburn because (a) two files need to be maintained, (b) the name of the Unix-flavor plain text file had no "`.txt`" extension, causing issues, (c) it was not clear which is the original / master and they could get out of step.

For these reasons, as of v76:

All plain-text files in the Breath Flute Project are provided in a single version, with `Name.txt` as the naming convention, using the DOS convention (CR+LF) for the newlines sequence, and using the newline sequence as a line terminator (the Unix convention). This means that, without knowing the source of the file, it is ambiguous whether a file ending in **`text`** followed by CR+LF represents an original file with an empty final line (i.e. it came from a DOS-based system) or whether it represents a file where the final line was **`text`**.

Note that the zero-byte `/Misc/Null.txt` file (used for calibrating MD5 utilities) has no lines of text, therefore has no newline sequences.

All plain-text files are converted from their original source to follow these conventions, and any checksums provided are on this converted file format.

About the Breath Flute Project GPL License

This section provides some of the rationale and motivation for the way the *Breath Flute Project GPL License* ("the License") is written. It also provides some advice on how to handle common licensing issues.

However, this section does not provide authoritative legal advice, nor does it modify or amend the terms of the License.

Motivation

The intent of this open-source approach is to foster a community of ongoing development and refinement, and to allow those who fabricate Breath Flutes to benefit commercially. The Breath Flute Project was built on many instances of this open-source approach: Linux, Apache, Firefox, WordPress, BIND, LibreOffice, Audacity, the open STL file format, OpenSCAD, fonts licensed under SIL OFL, and RepRap. There is also a predominant philosophy among the Native American flute community of open sharing of ideas, designs, techniques, and resources.

> *The more people that you have access to, the better the selection of ideas at your disposal, and the more you can recombine into something brand new. This is the key to cultural evolution.*
>
> *Someone thinks of something. Someone else adapts it to a brand-new environment. Someone incrementally improves on it. Copying, recombining, and learning from each other.*
>
> *This is what humans are good at. And this is the key to our intelligence.*
>
> — Michael Muthukrishna, London School of Economics

To promote ongoing development and collaboration, the Breath Flute Package is copyrighted and licensed under the GPL and the GFDL, which contain key permissions that promote sharing and continued development. This license is a Free Cultural license (www.FreedomDefined.org), and contains "Attribution" and "ShareAlike" conditions.

When distributing a release package (original or modified), physical objects generated using code from a release package, or any item that is covered by the License, the license requires you to give appropriate credit ("Attribution") and provide a link to the license. You may do this by simply prominently displaying the text contained in the `License.txt` file, including the link to www.BreathFlute.com. You must also describe any modifications that have been made to the release package. You may do so in any reasonable manner, but not in any way that suggests that any of the authors (see the `Authors.txt` file) endorse you or your use.

If you modify or build upon the material in the Breath Flute Package, you must distribute your contributions under the same license as the original ("ShareAlike"). Also, you may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.

What am I allowed to do with the Breath Flute Project?

Briefly, here are a few things that are allowed (and encouraged) under the License:

- You are free to download the software, use it to fabricate Breath Flutes, and even offer them for sale.
- You are free to modify the software or the design of the Breath Flute, fabricate your modified design, and offer them for sale.
- You are not required to release any modifications you make. However, ***if*** you release the modified version to the public in some way (e.g. by offering fabricated instruments based on your modified design for sale), the *GPL* requires you to make the modified ***source code*** available to the program's users, under the *GPL*. This promotes the continued, open development of the design by the community, a core intention of this project.
- If you release your modified version of the Breath Flute Project, you are free (and encouraged) to add your name to the `Authors.txt` file. Listing your name in the first ("`### AUTHORS`") section adds you to the copyright-holders of the project and offers you protections provided by the License. You do not need to modify the copyright notices in the `License.txt`, source code files, or documentation files, since the copyright notices in those files refer to the `Authors.txt` file.
- If you include code from third-party sources (see the ***What are the license terms for modifications that I make?*** section (below) on when that is allowed), then you need to conspicuously identify the complete details of its source and of any license or other restriction (including, but not limited to, related patents, trademarks, and license agreements) of which you are personally aware. To satisfy any "attribution" clause, you may modify the comments in the third-party code.
- You may also wish to add the names of others who have contributed to the Breath Flute Project, as described in the sections marked "`### CONTRIBUTORS`", "`### ADVISERS`", or "`### APPRECIATION`" in the `Authors.txt` file. ***Please obtain their permission prior to doing so.*** Also, please keep the list of Contributors in alphabetical order of their project title, and Advisers in alphabetical order by name. In the case of code from third-party sources, the authors of that code are typically listed in the "`### CONTRIBUTORS`" section, unless they wrote code specifically for the Breath Flute Project.
- If you are adding a new file in your release, please include a header, a copyright notice, and a license statement near the beginning of the file. Here is an example of a typical header in the OpenSCAD files of the project:

```
// The Breath Flute Project – XXX Module – FFFF.scad
//
// The Breath Flute is a wind instrument that lets players with limited musical
// experience create music using only their breath. The Breath Flute Project
// generates a Constructed Solid Geometry model in OpenSCAD for fabricating a
// Breath Flute headjoint, typically by 3D printing. The headjoint is combined
// with a body tube (e.g. 1.25" PVC) to make a complete instrument. See
// ReadMe.txt and www.BreathFlute.com for more information.
//
// This module implements ...
```

Please use the text below for the copyright notice and license statement. You may change the text to set the appropriate year(s) for the copyright (see the section below on ***How should the year or years given in a copyright notice be set?***) and adjust the text for the comment conventions of the programming language for that file, but the remaining text should remain unchanged:

```
// Copyright 2016–2018 the Breath Flute Project Authors (see Authors.txt).
//
// This software is part of the Breath Flute Project (www.BreathFlute.com).
// You may copy, use, modify, and distribute this software under the terms of
// the GNU General Public License version 3 (“the GPL”) or any later version,
// as published by the Free Software Foundation. See the License.txt and
// GPL.txt files for details.
//
// This software is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
// or FITNESS FOR A PARTICULAR PURPOSE. See the GPL for more details.
//
// The GPL is distributed in plain text with this software. It is also
// available at <http://www.GNU.org/>.
```

- If you are adding a new documentation file, please copy the copyright statement and license in `BreathFlute_DevGuide.pdf`.

Some important things you are ***not*** allowed to do under the License:

- You may ***not*** distribute (free or commercially) fabricated versions of modified Breath Flutes without releasing the source code under the same license.
- You may ***not*** strip out the copyright notices in the source code of versions you distribute.
- You may ***not*** remove `Authors.txt`, `ChangeLog.txt`, `License.txt`, `ReadMe.txt`, or other files from the top-level directory of the distribution.
- You may ***not*** remove ***or modify*** the `GPL.txt` or `GFDL.txt` files.
- You may ***not*** remove ***or modify*** the license files in any subdirectory of `/Src` – for example, the `/Src/Bioinfscripts/License.txt` file.
- You may ***not*** apply additional restrictions when you redistribute original or modified versions of the software. One example of an additional restriction would be to provide the software to others under a non-disclosure agreement.

To confirm that the static files in a given distribution have not been modified, first confirm that your MD5 utility provides these checksum values on the `/Misc/Null.txt` file in the distribution package. You can use either the Hex or Base64 versions of the checksum:

`Null.txt`	Size:	`0`
	MD5 Hex:	`d41d8cd98f00b204e9800998ecf8427e`
	Base64:	`1B2M2Y8AsgTpgAmY7PhCfg`

Then check the MD5 checksums on these static files:

`GPL.txt`	Size:	`35,823`
	MD5 Hex:	`e62637ea8a114355b985fd86c9ffbd6e`
	Base64:	`5iY36ooRQ1W5hf2Gyf+9bg`
`GFDL.txt`	Size:	`23,415`
	MD5 Hex:	`a5a587b3f95d049d6df40d1e30206baf`
	Base64:	`paWHs/ldBJ1t9A0eMCBrrw`

For more details and examples, see *GPL Frequently Asked Questions* at https://www.gnu.org/licenses/gpl-faq.en.html.

What are the license terms for modifications that I make?

You may submit modifications to the Breath Flute Project that satisfy any combination of one or more of these conditions:

- Modifications that you personally author, as long as you agree with the terms of the License (see ***The Breath Flute Project GPL License*** section above or `License.txt`).
- Modifications from other sources that are openly available under licensing terms that are compatible with the License. Some examples that ***are compatible*** with the License are (as long as any licensing terms, such as "attribution", are met):
  - Apache 2.0
  - Artistic License 2.0
  - BSD "Modified" or "Three-clause"
  - CC-0 or other statement placing the code in the public domain
  - CC-BY
  - FreeBSD (also known as the "BSD Two-clause")
  - GPLv3
  - GPLv2 with an "or later version" clause,
  - MPL 2.0
  - X11
  - XFree86 1.1

  Some examples that ***are not compatible*** with the License are:
  - No license. Most source code that does not carry a written license is granted a restrictive copyright by default. There are specific exemptions to this rule, but the rules are complex. It is best to avoid incorporating such code.
  - AGPL 1.0
  - Apache 1.0

  - Apache 1.1
  - Artistic License 1.0
  - BSD "Original" or "Four-clause"
  - CC-BY-NC
  - CC-BY-ND
  - GPLv2 without an "or later version" clause
  - The JSON license. This license has a clause mandating "The Software shall be used for Good, not Evil.", which makes it incompatible with GPLv3. We could, as some others have (eg. IBM), obtain a special exemption allowing us to "use JSON for evil", but that's probably not a good idea.
  - MPL 1.1
  - Open Public License
  - Reciprocal Public License

  See https://www.gnu.org/licenses/license-list.en.html for additional examples and a discussion of license compatibility.

If I change just the Fabricator Code and print a flute, does that mean I need to release the source code?

No. Changing the fabricator code (the two letters printed on the side of the flute that indicates who fabricated that instrument) can be done by executing OpenSCAD from the command line and providing an appropriate override for the `FabricatorCode` variable. There is no change to the code in this case, so this situation, in and of itself, does not trigger the *GPL* requirement to release your source code.

Why does the free, open-source license for the Breath Flute Project have a copyright statement?

A central concept of the Open Source movement is how it uses existing copyright law to guarantee the free and open access to and redistribution of software. This may seem counterintuitive, since copyrights have traditionally been used to restrict the rights of access and redistribution.

The Breath Flute Project provides access to the software to anyone, in exchange for abiding by the terms of the *GPL*. The *GPL* uses copyright law to guarantee that no party may restrict or curtail anyone else's right to copy, use, modify, and redistribute the software. A party that violates the terms of the *GPL* (for example, by restricting the rights of others) loses their rights under the License, places them in violation of copyright law, and opens the possibility of prosecution.

To further promote open collaboration and development, the guarantees of the *GPL* extend to the source code of the software.

Why isn't this software simply released into the Public Domain?

While it might appear that releasing software into the public domain would be a more noble approach, this path tends not to serve the community as well as the *GPL*.

- The concept of public domain is not universally recognized.

- The mechanism of making a public domain declaration is problematic in many situations, and even illegal in some countries.
- The authors cannot apply a warranty clause and open themselves to litigation by making the code available under a Public Domain declaration.
- Public domain places no restrictions on use. This freedom allows anyone to apply legal restrictions that prevent others (even the original authors) from using the software. Anyone may even claim authorship of the software, making the history of its development unclear. There are many notable cases of these perverse situations involving public domain content.
- Many commercial enterprises avoid using public domain content, because it opens ***them*** to subsequent litigation by others who have subsequently applied copyright and trademark restrictions.

For these reasons, I am using the well-established tradition of the *GPL* to promote continued, open development of the Breath Flute Project.

Why is version 3 of the GPL used?

Version 3 is the most recent and robust of the *GPL* licenses, providing protection from such as tivoization, digital rights management, and restrictive patent agreement attacks on *GPL*-licensed code that have become part of case law since *GPL* version 2 was drafted. I use a "version 3 or later" style of license, to allow combination of this software with other software that may be licensed under a later version of the *GPL.*

Note, however, code licensed under *GPL* version 2 only (rather than "GPL version 2 or later") is incompatible with the Breath Flute Project License and cannot be combined with this software.

Why is the documentation under a different license?

The *GNU Free Documentation License* (the *GFDL*) shares the basic intent and design as the *GPL*, but is specifically designed to cover documentation.

How does the License address issues of patent law?

The Breath Flute Project makes every effort not to use patented technology.

Also, the open publication of the Breath Flute design and software establishes a "prior art" defense against anyone who might endeavor to restrict access using patent law. If you modify the design of the Breath Flute (by modifying the code of the Breath Flute Project, or by other means), we encourage you to openly publish all those ideas to prevent their being patented by others.

To further promote the goals of open source software, you may consider the Open Invention Network (OIN) – a group of individuals, organizations, and companies to free existing patents and to prevent known or obvious ideas from being patented. You may also be interested in the concept of *Free Patents* and the efforts of the *Ask Patents* initiative (http://patents.stackexchange.com/).

How should the year or years given in a copyright notice be set?

Based on legal advice form Epiphany Law, we set the Year component of the copyright notice for a work (a document or file) using the format "`StartYear–EndYear`". `StartYear` is the year portion of the date of the oldest content still contained in the work and `EndYear` is the last year that the work was revised. If `StartYear` is the same as `EndYear`, then the format is simply "`Year`".

How does the License apply to the fonts that are included in the distribution package?

Licensing of open source fonts is often handled under a different license from other software components. The *GPL* does provide an optional Font Exception (https://www.gnu.org/licenses/gpl-faq.en.html#FontException). This allows people to embed *GPL*-licensed fonts in documents (such as PDF files), without the fonts themselves causing the whole document to be covered under the GPL.

However, we have chosen ***not*** to include the Font Exception in the *Breath Flute Project GPL License*, because:

- The only place that the fonts are uses is on the Breath Flute models themselves, which are already covered by the *GPL*.
- A *GPL + Font Exception* license does not address the complex issues of font licensing as comprehensively as other open-source licenses that have designed specifically for the complex issues of font licensing. See, for example, the *SIL Open Font License version 1.1* (the "*SIL OFL*" – see http://scripts.sil.org/ofl_web).
- The **Kurinto Sans** fonts used by the Breath Flute Project is licensed under *SIL OFL*, which allows are extremely limited in coverage of character sets, and of little use outside of inscribing text on FDM models.

The SIL OFL 1.1 license allows inclusion of the ***Kurinto Sans*** fonts in this GPL-licensed package, according to this *OFL-FAQ web version (1.1-update5)* published by SIL International. From http://scripts.sil.org/ofl-faq_web, retrieved June 8, 2018:

> Question: 1.2 Can the fonts be included with Free/Libre and Open Source Software collections such as GNU/Linux and BSD distributions and repositories?
>
> Answer: Yes! Fonts licensed under the OFL can be freely included alongside other software under FLOSS (Free/Libre and Open Source Software) licenses. Since fonts are typically aggregated with, not merged into, existing software, there is little need to be concerned about incompatibility with existing software licenses. You may also repackage the fonts and the accompanying components in a .rpm or .deb package (or other similar packaging formats or installers) and include them in distribution CD/DVDs and online repositories. (Also see section 5.9 about rebuilding from source.)

Resources

In addition to the documentation included with the release of the Breath Flute Project, here are additional resources that are not part of the release:

- The web site http://www.BreathFlute.com/
- Videos on YouTube and Vimeo. You can search for "Breath Flute" and find a number of playlists that you can browse.

References

[3DMatter 2015] 3D Matter, *What is the influence of infill %, layer height and infill pattern on my 3D prints?*, retrieved 4/28/2018 from http://my3dmatter.com/influence-infill-layer-height-pattern/.