Beads Quickstart Guide / ver. 057

Overview

If you find any errors in this document or have any suggestions, please send them to `magicmouse94937@gmail.com` with the subject QUICKSTART so that your suggestion gets routed to the proper place.

Feel free to join the Beads discord group, with this invitation: `https://discord.gg/pTAdsSW`

Pre-installation Requirements

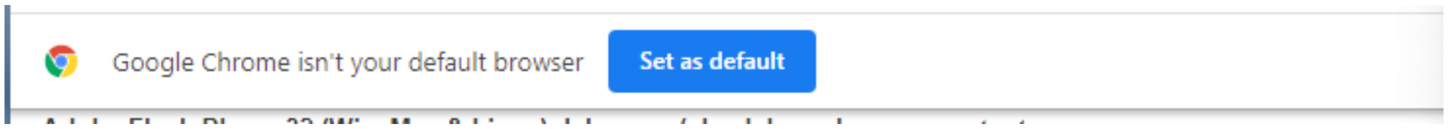
You will need to download the `beads.zip` file

The Beads SDK works on Macintosh OSX 10.12 or later, and Windows 7 or later.

Linux with the Wine Windows emulator, also works.

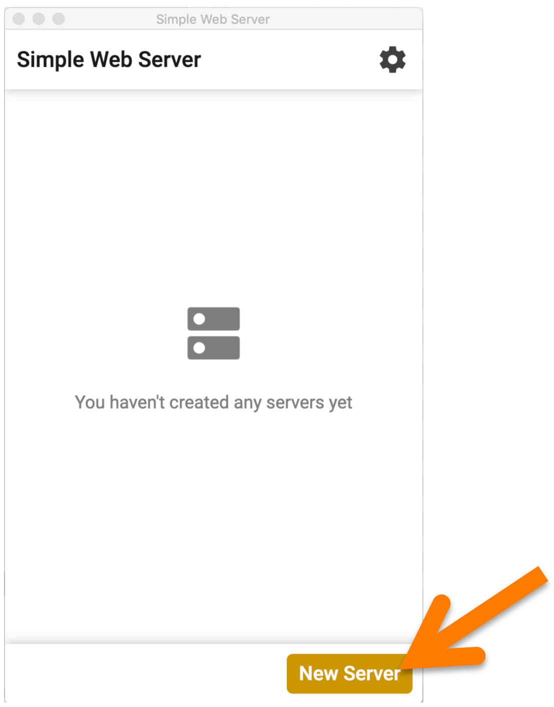
Special instructions for using Google Chrome debugger

When you first run Google Chrome, it will prompt you to allow it to be your default browser. Click the button that shows up at the top right if it isn't yet your default browser. Here is what the button looks like:

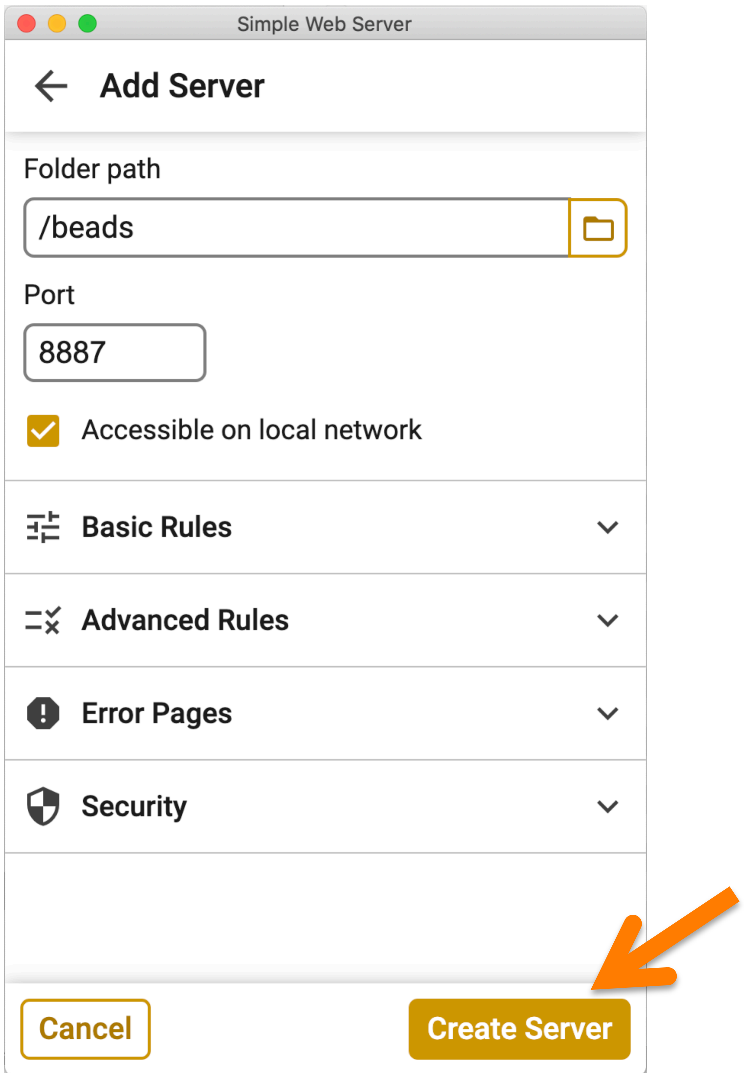


Google Chrome will not permit you to load local JS files. Firefox will typically let you do this, with some tweaks to the preferences, but the best method and the only practical way to get around Chrome's refusal to load local JS files, is to add a mini web server.

> Go to `https://simplewebserver.org/download.html` and download the executable for Windows or Macintosh or Linux.
>
> Once you run the web server you will see a config panel for the web server.



> Click NEW SERVER button to start up a new server.


Simple Web Server

Add Server

Folder path: /beads

Port: 8887

✓ Accessible on local network

Set the folder to /beads or /dev/beads (or wherever you put your beads files)
Set the port to be 8887, the default is 8080, but the compiler is set for 8887

Turn on "Accessible on local network" checkbox, so that mobile devices on your LAN can access the server

When this webserver is running, references to "Localhost:8887" will be redirected to that folder.

For WINDOWS:

A) Click the CHOOSE FOLDER button to set the folder being served to be your Beads development folder. In our case we had used C:\DEV\BEADS as our folder, but use the folder where you are storing Beads files. By setting the folder, then any `localhost:8887` URL accesses are relative to the `beads` folder, and you can load local resources without Chrome blocking it. You must select the `beads` folder that contains the `runtime` folder.

B) If you wish to allow other computers on your local network to test out your web app, turn on the "Accessible on local network" checkbox.

If when you run the "Hello World" application, if the `beads_std.js` file doesn't load, it means the mini webserver is not running.

Installation instructions for Macintosh

Preparing your browser for direct execution of Javascript/HTML files:

Apple Safari:

If you are using Safari on the Macintosh:
On the menu bar, under `Develop` turn on `Disable Local file restrictions`

`WARNING: Safari` before `14.0.3` has a **bug** which ignores the "Disable" feature. Please update your Safari to at least `14.0.3`. To get a later version of Safari, go to:
`https://developer.apple.com/safari/download/`

Mozilla Firefox:

1) go to `about:config` in the address bar and press return
2) click Show All
3) scroll down to `security.fileuri.strict_origin_policy` and toggle it to `false` by clicking the toggle icon to the right. ⇌ This will disable the overly strict rule that local files cannot be loaded.

Google Chrome:

Google Chrome has the best debugger, so it is the preferred browser at present. However, Chrome is quite annoying, in that it spits out CORS security errors when you try to load a JS file from your own hard drive, and the flags to disable this restriction have been discontinued. Luckily someone has built a tiny plug-in that is very convenient to use. Please install the Google Chrome simple web server utility as describe above, so that you can serve up the files on port 8887 without annoying Cross Origin errors.

Compiler Installation:

1. You unpack the ZIP file `beads.zip`, by double clicking, and then copy the contents into a folder called `beads` in your home folder.
   a. After copying to /beads, the folder structure should look like this:
      i. `/bin_OSX` -- where the compiler is stored for Macintosh OSX
      ii. `/bin_Windows` -- ignore these files they are for MS Windows

    iii. `/docs` -- for documentation
    iv. `/lib` -- contains the source and compiled header files for the `std` and `str` library modules
    v. `/projects` -- contains the master stub.html file which allows you to specify scrollbar style
        1. `/clock` - clock example, etc.
    vi. `/runtime_057` -- contains the runtime JS files that will be needed to publish web apps
    vii. `/runtime_057_min/` -- contains the minimized libs, which you can use to post to your web server. The non-minimized are only slightly larger, and a bit easier to debug.
    viii. `/runtime_057_node/` -- the runtime libraries recast for use with Node.JS

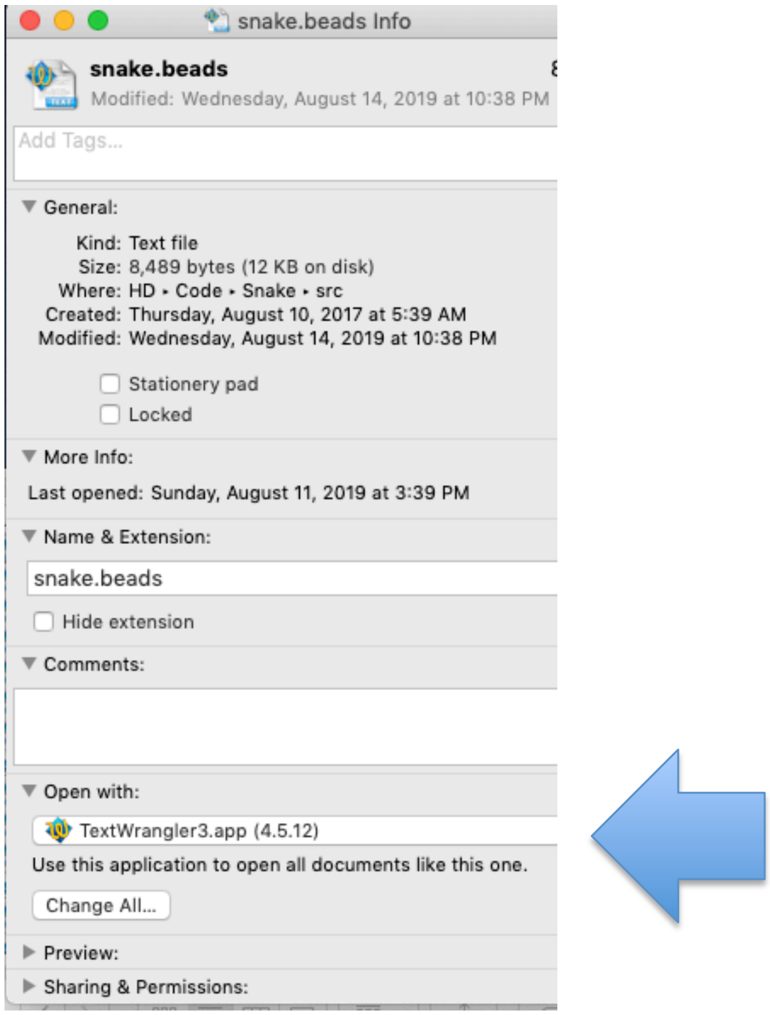
2. In the `beads/bin/OSX` folder, locate the `beads_nnn.app` (where *nnn* is the version number) program file and right click it, then select OPEN. Drag the application into your dock for convenient launching. Upon first open, it may prompt you to locate the `runtime` folder in your beads folder, if it is not in the default location.

3. All Macintosh computers come with the Safari browser built in. Unfortunately at present there is a bug in both Chrome and Firebox, and they refuse to allow local `.js` modules to be loaded and generate annoying cross-origin scripting errors. Only Safari as of October 2019 correctly allows local modules to be opened, because it has a handy option under the `Develop` menu bar item to disable cross-origin and local file restrictions. If you don't wish to use Safari, you can use Chrome, however, you will need to install the Chrome plug-in module that provides a mini web server. See the Windows instructions step #9 for how to get the Plug-in. There is a free app in the Chrome store that handles this function.

    a. You will need to set your default browser to Safari, which can be done inside Safari / Preferences / General, the very top item is a button to change your default browser.

4. There are many free code editors available, such as Atom (160MB), Brackets, TextWrangler3 (10MB), `BBedit` (16MB), `VSCode`, etc. `TextWrangler` is lightning fast and free, but is harder to find; `BBedit,` its sequel is available here:
    `https://www.barebones.com/products/bb/download.html`

5. We need to tell Mac OSX to associate the `.beads` filename suffix with your favorite code editor. So peek inside the `/beads/projects/hello` folder, and right click the `hello.beads` file, and select `Get Info`. You will then see a information panel on the file, where it mentions `"Opens with:"` In the pulldown menu, select your preferred editor, and click the `Change All…` button to make all `.beads` files automatically open via your preferred editor.



snake.beads Info

snake.beads
Modified: Wednesday, August 14, 2019 at 10:38 PM

Add Tags...

General:

Kind: Text file
Size: 8,489 bytes (12 KB on disk)
Where: HD ▸ Code ▸ Snake ▸ src
Created: Thursday, August 10, 2017 at 5:39 AM
Modified: Wednesday, August 14, 2019 at 10:38 PM

☐ Stationery pad
☐ Locked

More Info:

Last opened: Sunday, August 11, 2019 at 3:39 PM

Name & Extension:

snake.beads

☐ Hide extension

Comments:

Open with:

TextWrangler3.app (4.5.12)

Use this application to open all documents like this one.

7) Now locate the `/beads/projects/hello.html` file, and right click it and select `Get Info`, and assign Safari as the preferred program to open `HTML` files with, just like we did with the text editor above to link `.beads` with your preferred editor.

8) Proceed to the section "How to use the compiler"

Windows Installation:

Preparing your browser for direct execution of Javascript/HTML files:

Google Chrome

You will need to install the mini web server so you can load local .JS files (see above). Chrome will run with HTTP mode without any modifications.

Mozilla Firefox:

1) go to `about:config` in the address bar and press return
2) click Show All
3) scroll down to `security.fileuri.strict_origin_policy` and toggle it to `false` by clicking the toggle icon to the right. ⇌ This will disable the overly strict rule that local files cannot be loaded.

Note that in Firefox, the relative library load works fine, but Absolute file paths do not work under Windows.

MS Internet Explorer:

IE has been replaced by MS by Edge, please download Edge as IE is hopelessly behind

MS Edge:

Edge is basically a Google Chrome clone, however the Google Chrome web server extension does not work under Edge, so if you want to test with Edge, you will have to run the mini webserver extension inside Chrome, so that port 8887 serves up your Beads folder.

Compiler Installation

1) Unpack the beads zip file into a folder where you are going to store your Beads project. It doesn't matter where, but keep the path short if possible. The folders should look like this:
   `\bin_Windows` -- where tools are stored
   `\docs` -- for documentation
   `\lib` -- contains the source and compiled header files for the `std` and `str` library modules
   `\projects` -- contains the master stub.html file which allows you to specify scrollbar style
      `\clock` - clock example, etc.
   `\runtime` -- contains the runtime JS files that will be needed to publish web apps

Verify the compiler is running by double clicking the `beads.exe` file in the `\bin_windows` folder. The system should automatically detect the runtime folders as they are up one folder from the executable, and down one.

To make a handy desktop shortcut, drag the EXE to your taskbar, or click "Create Shortcut" and drag that to your Desktop. That way you can launch the compiler shell conveniently.

If the compiler doesn't launch correctly, stop here, and send an email to `support@beadslang.com`, and we can set up a `Teamviewer` session and figure out why this isn't working.

---

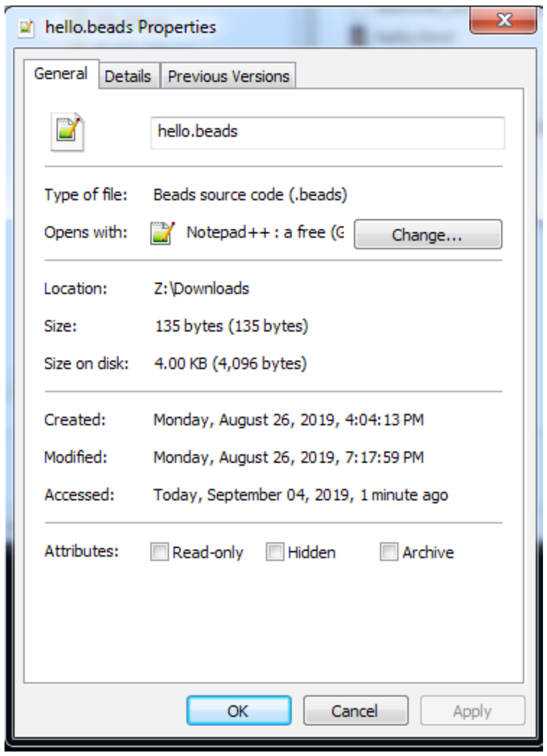
5) There are many free code editors available, such as Atom (130 MB), Brackets (76 MB), Notepad++ (tiny), VSCode, etc. Notepad++ is available here:

`https://notepad-plus-plus.org/downloads/`

Notepad++ is a fast and simple text editor. You can edit Beads code with any editor, and although we don't have syntax coloring plugins yet (if you know how to do this please contact us for a project), you can treat the file as a JavaScript file, and the comments and string literals will display mostly correctly.

---

6) Set Windows to associate the `.beads` filename suffix with your favorite code editor. So peek inside the `beads projects\hello` folder, and right click the `hello.beads` file, and select `Properties`. You will then see a Properties panel, where it mentions `"Opens with:…".` Click the `Change…` button to set the association with your favorite editor, which might be Notepad++ or VSCode or any of 100 other text editors available for windows.


hello.beads Properties

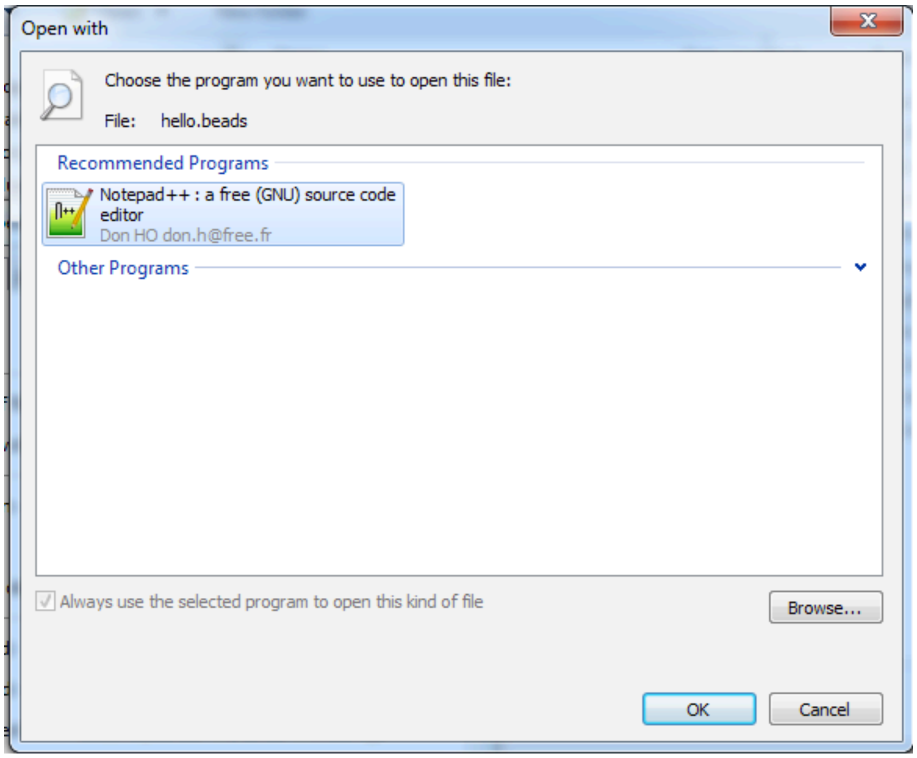
General | Details | Previous Versions

hello.beads

Type of file: Beads source code (.beads)
Opens with: Notepad++ : a free (G Change...
Location: Z:\Downloads
Size: 135 bytes (135 bytes)
Size on disk: 4.00 KB (4,096 bytes)
Created: Monday, August 26, 2019, 4:04:13 PM
Modified: Monday, August 26, 2019, 7:17:59 PM
Accessed: Today, September 04, 2019, 1 minute ago
Attributes: Read-only Hidden Archive

OK Cancel Apply

7) When you click the `Change…` button, it will present a picker window where you can select which programs should edit `.beads` source files. Select your favorite editor, or browse to the `.EXE` file of your editor if it isn't listed. Click OK to make the association:


Open with

Choose the program you want to use to open this file:

File: hello.beads

Recommended Programs
Notepad++ : a free (GNU) source code editor
Don HO don.h@free.fr
Other Programs
Always use the selected program to open this kind of file
Browse...
OK
Cancel

8) Double check the association is working by double clicking a `.beads` source code file. It should launch your favorite editor.

If you ever need to reset all the preferences for the Beads compiler, the file is located in:

```
username\AppData\Roaming\beads\Local Store\beads_prefs
```

where `username` is your user name. Note that `AppData` is an invisible folder for some unknown reason, and you will have to turn on the Windows file explorer option to see this folder.

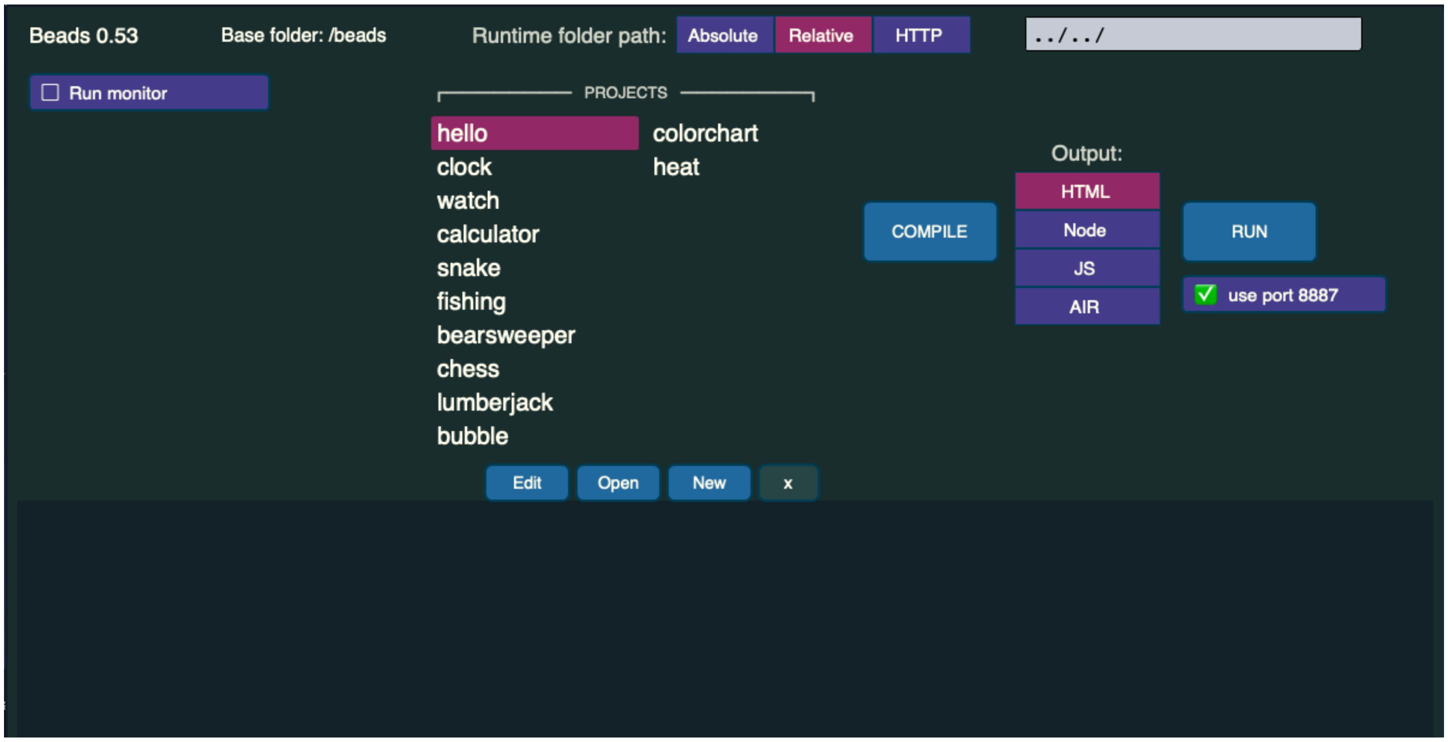
How to use the compiler

Normally one will use the launcher program to control the compiler. However, there is a command line interface, see next chapter.

For Macintosh: In the `beads/bin_OSX` folder there is a `Beads.app` program, which is the compiler. Double click it to launch the compiler

For Windows: In the `beads/bin_Windows` folder, you will see the `Beads.exe` program, which is the compiler. Double click it to launch the compiler.

The first time you run the compiler, it may ask you to find the folder containing the project and runtime folders.


Beads 0.53
Base folder: /beads
Runtime folder path: Absolute [✓ Relative] HTTP
../../
☐ Run monitor

PROJECTS

hello	colorchart
clock	heat
watch	
calculator	
snake	
fishing	
bearsweeper	
chess	
lumberjack	
bubble	

Output: [✓ HTML] Node JS AIR
✓ use port 8887

First select a project name. The current project will be highlighted in magenta.

To the right, you click the `COMPILE` button to run the compiler. The output target will usually be `HTML`. After compilation, you click `RUN` to launch the HTML file in your browser.
The 4 output targets:

`HTML` -- emits a `.html` file for your web app.
`NODE` -- emits a `.mjs` file for use with Node.JS (`.mjs` is the Node ES module suffix)
`JS` -- emits plain JavaScript module
`AIR` -- emits AS3 files for use with FlashBuilder/Adobe Animate/Haxe

Normally you will have the `use port 8887` option on, so that you can get around the annoying Chrome browser limitation which forbids for nonsense reasons loading JS files from your own computer. If you turn off port 8887 option, that would be used to launch the HTML page via port 80 (default) in Firefox or Safari, which do permit local file loading (with the appropriate option flags set).

The top line of the screen allows you to select how you want your emitted code to refer to the runtime library paths. The options are:

`Absolute` -- used to specify the exact folder on your local machine where the runtimes are found
`Relative` -- used to specify the relative folder to the project you are running, maybe 2 levels up (../../)
`HTTP` -- used to specify that the emitted code will reference the libraries on a remote server

If you are using Safari, we recommend you start with the RELATIVE option, because it will load faster by using the runtime on your own computer. Otherwise we recommend you use the HTTP option, because the default settings pull the libraries from the beadslang.com website, which we offer as a free service. If you are using the Chrome browser, you will need to use HTTP or install the local web server.

During production, if you keep the same relative folder layout, you can then upload the exact same files you run on your local computer to the remote server, and it will work, if the same relative position is set. This is the preferred method, as it doesn't depend on exact paths, just relative paths.

Start by clicking RUN on the `hello` project, and make sure your setup is running correctly. It should compile the code and launch your preferred browser and display "*Hello World*".

There are 16 program slots that can hold a beads project. We have preloaded the sample projects for your training. You click a folder icon to put a beads project file into the launcher's memory bank. You click the `EDIT` button to launch the text editor the `COMP` button to compile the code, and the `RUN` button compiles, and if successful runs the program. Most of the time your code will not compile, and the first few errors will be shown in the console area. As the compiler is not very bright about catching one error and ignoring the repercussions, I suggest you fix only the first or second error before trying compilation again.

If you Control-Click the `RUN` button, it will force the source to be recompiled, even if it thinks it is up to date. If you change external factors like the compiler version, you may need to manually force a rebuild.

Start by click the `RUN` button on the Hello project. It should launch in your browser. To see the output of your log statements, turn on the JavaScript console feature in the Safari `Develop` pulldown menu item named `Show Javascript Console`.

After familiarizing yourself with the sample projects, it is time to write your own programs! Please create a folder for each project in the `beads/projects` folder, and after writing some code in your text editor, go to the launcher program, click the Folder icon on the left of a slot, and navigate to your file. Then it is in the launcher's data bank, and you can quickly launch the editor and run the code. Remember that you have to save your work in your text editor after each change before you try to compile. Beads has no idea that your text editor has pending changes, so you have to be careful.

If you ever want to trash the `beads_launcher_prefs` file, it is located:
Win: `C:\users\MyUserName\AppData\Roaming\beads\Local Store\beads_prefs`
Mac: `MyUserName/Library/Application Support/beads/Local Store/beads_prefs`

The sample projects are as follows, in order of complexity:

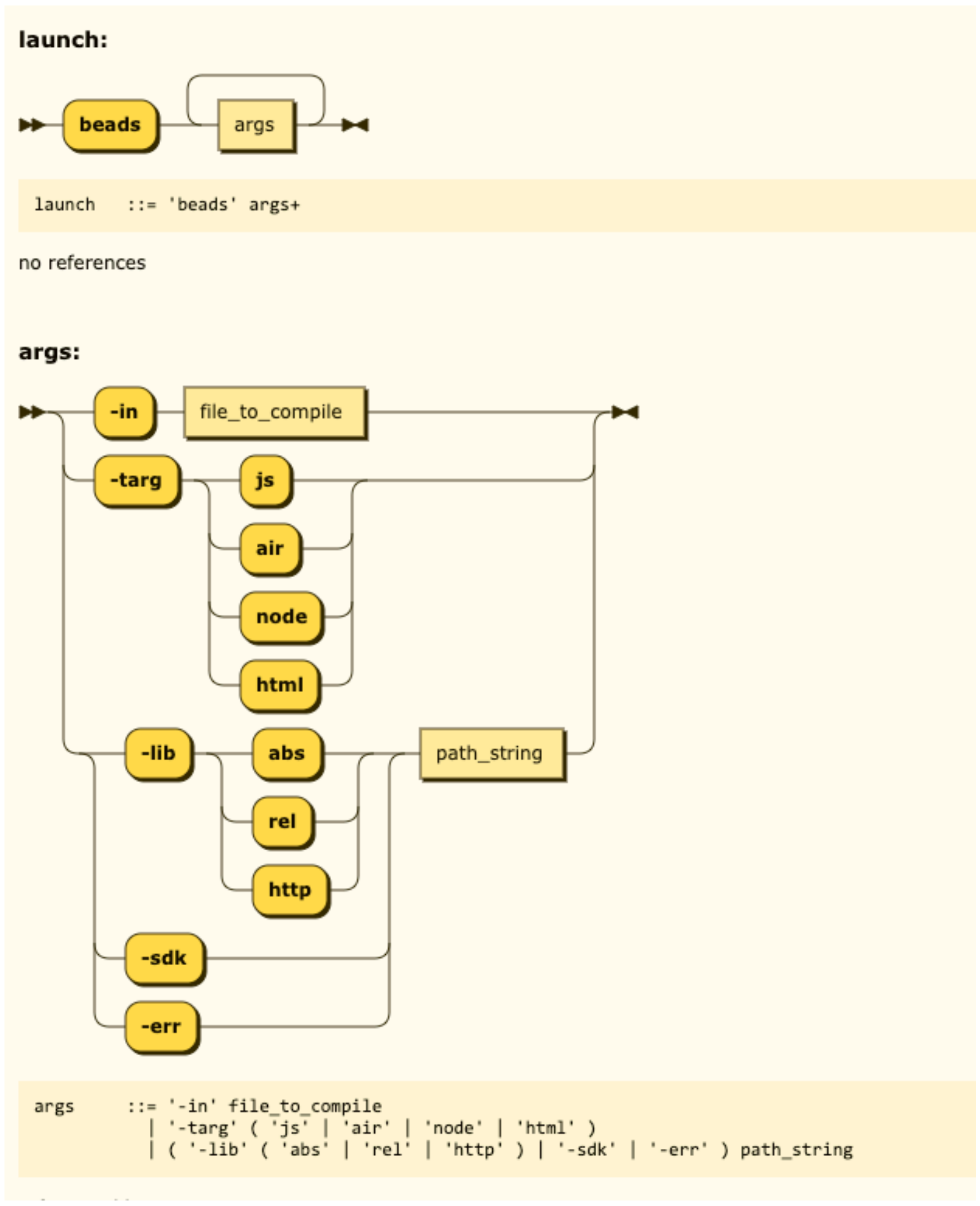
`hello`	-- a simple hello world
`clock`	-- a simple analog clock
`watch`	-- a bitmap simulation of a wristwatch
`calculator`	-- a copy of the Apple calculator in basic mode (without some of the Apple bugs!)

`snake`	-- the classic Nokia snake game
`tictactoe`	-- the classic Children's game
`chess`	-- a nice version of chess
`form1`	-- a data entry form that rearranges its layout depending on the screen size

There is also a module `buttons` that contains components you can copy and paste into your project, which draws one-shot buttons, mutually exclusive choice buttons, and toggle buttons.

How to run the compiler by command line

The syntax for using command line parameters is:



launch:

```
launch   ::= 'beads' args+
```

no references

args:

```
args     ::= '-in' file_to_compile
           | '-targ' ( 'js' | 'air' | 'node' | 'html' )
           | ( '-lib' ( 'abs' | 'rel' | 'http' ) | '-sdk' | '-err' ) path_string
```

ADDENDUM: in version 048 we have added `-ofold` parameter to allow an override of the default output folder.

The order of parameters is not important. All parameters are lower case. The default target is `html,` the default library method used for the includes in the emitted JS files is relative, and will go up 2 folders `("../..")` in Unix folder notation`)`.

The only required parameters are the `sdk` location, the error file path, and the input file to compile. The default error output file is `~/err.txt` (`~` means your home folder). Please note that the path string for `-lib` and `-sdk` is a folder name, but the path name for the `-err` output file is a full path including filename.

So an example command line would be:

```
beads -in "C:\stuff\beads\projects\myproj\myproj.beads" -sdk "C:\stuff\beads"
```

The error file with start with 2 digits containing the number of syntax errors. If it is `00` then the program compiled perfectly. The error text will follow. An example error output file:

```
00errs
compiled hello.beads at 12…
```

The following are the predefined meta-folders that you can use on command line parameters:

`~appres/`	--- [APP]/Contents/Resources
`~applib/`	--- /Library/Application Support/[APP]/Local Store
`~docs/`	--- same as ~/Documents
`~desk/`	--- same as ~/Desktop/
`~/`	--- /Users/[USERNAME], the current user's home folder

How to run Client/Server programs in Beads

In the Beads example folder, we have a `RobinHoody` folder, which shows a real-time stock ticker program. You can run this as a client/server program, provided you install Node.JS on your server.

Step 0) Make sure Node.JS has all the modules it needs to run, and patch the runtime system to match your file locations.

Install the ws module if it is not already installed:

```
sudo npm install ws -g
```

You will also need to add the graceful file system module (`graceful-fs` module), which is needed for all client/server work in Node. It is not a standard built-in module, and you have to download and install it with NPM (the node package manager).

After downloading the `graceful-fs` module, you will need to patch the runtime file for node which is stored in the beads development folder in `/beads/runtime_0xx_node/node_std.mjs`
Locate the lines around line 99 where it includes the node module source files:

```
import ws from "/usr/local/lib/node_modules/ws/index.js";

import fs from "/usr/local/lib/node_modules/npm/node_modules/graceful-
fs/graceful-fs.js";
```

You will need to patch these file entries to match your file system location where the node system is stored. If you don't patch the runtime file, then when you try to run Node.JS on the server it will barf and say it can't locate the file, in a module not found error message that looks like this:

```
internal/modules/esm/default_resolve.js:84 let url = moduleWrapResolve(specifier,
parentURL);
^ Error: Cannot find module /usr/local/lib/node_modules/ws/index.js imported from
/Users/xyz/beads/runtime_041_node/node_std.mjs at Loader.resolve [as _resolve]
(internal/modules/esm/default_resolve.js:84:13) at Loader.resolve
(internal/modules/esm/loader.js:73:33) at Loader.getModuleJob
(internal/modules/esm/loader.js:147:40) at ModuleWrap.<anonymous>
(internal/modules/esm/module_job.js:41:40) at link
(internal/modules/esm/module_job.js:40:36) { code: 'ERR_MODULE_NOT_FOUND'
```

Step 1)
add the `hoody_server` program to the compiler projects list by clicking the folder icon next to an empty slot, and then picking the `hooder_server.beads` file.

Step 2)
set the compiler for `Node` output mode so we emit `.MJS` files, and runtime at `RELATIVE`

Step 3)

click the COMP button to compile the server. The output file should be a `hoody_server.mjs` file. Put that file on your server folder. Note that the runtime folders are considered to be 2 levels up. (if you are using RELATIVE mode).

Step 4)

run the server under Node.JS on your local machine by doing the following in your command line:

```
node --experimental_modules hoody_server.mjs
```

You should see it updating prices every second. Press `CTRL-C` to stop the Node program. We will relaunch it later.

Step 5)

add the `hoody_client` program to the compiler projects list by clicking the folder icon next to an empty slot, and then picking the `hoody_client.beads` file.

Step 6)

set the compiler for HTML output mode.

Step 7)

compile the `hoody_client` program.

Step 8)

run the `hoody_client` program. You should see the a black screen with the top line showing it is ready to try and connect.

Step 9)

restart the server program now that the client is ready to go

```
node --experimental_modules hoody_server.mjs
```

Step 10)

Click the `CONNECT` button in the client, and hopefully the server will answer.

The layout of a Beads source code file

The first line

Every Beads program has a required first line that specifies the level of the language that it was written against, and the kind of project you are building such as a library or standalone program, and the program name. Most of the time you will be writing a standalone program, so it resembles this:

```
beads 1 program chess
```

Module imports

The next section of the program lists all the modules that need to be imported from other parts of your program. You can choose to import all the export symbols of a module, or may select a specific set of the symbols to import. By default, the two main system library modules `std` (the standard library module containing the most frequently used library functions) and the `str` module (the string manipulation function library) are included by default. Many programs will not use any imports

Art asset imports

The next section of the program lists all the art assets and resources that are to be included into the program. Most graphical interactive programs incorporate many art resources. You can import files one at a time, or can import folders of art resources. The types of resources you can import are:

Still images like JPEG files
Sounds like MP3 files
Movies like M4A files
Fonts like TTF files

Example:

```
assets local:"art/" remote:"http://magicmouse.com/beads/examples/chess/"
	file: "check.mp3" label:CHECK_SOUND
	file: "basso.mp3" label:BEEP
	file: "checkmate.mp3" label:CHECKMATE_SOUND
	file: "stalemate.mp3" label:STALEMATE_SOUND
```

Definitions

The next section of the program lists the user-defined enumerated values, units and unit families and records that will be used in your program. These are specific to your program.

Records

Since Beads supports composable record types, you can build up your model data structures with record definitions.

Constants

The next section defines the constants that you will use in your program. A constant is a variable whose value is set once and never modified. You also will define any enums (some languages call them symbols), which are effectively numeric constants where you don't care what the number is, as long as they don't match any other number in the system.

Variables

Many programs will have one big tree variable, which stores the state of the program. This is the state that is referenced to draw the screen, and any changes in value of the state will cause the refresh of any draw block that used that state variable.

Each chunk of code, which in some languages are called functions or subroutines, has a prefix identifying what kind of code it contains and a name. So for example this chunk of code is function that is going to calculate something:

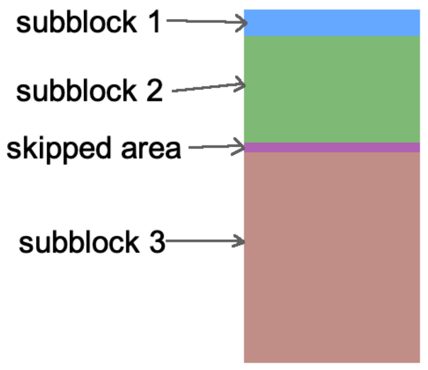
```
calc half(
     arg : num
     ) : num
     return arg / 2
```

The `calc` prefix indicates that this function is going to calculate something. A `calc` function cannot call drawing routines.   A draw function indicates that the block can access the screen. Implied in every drawing type of block there is an implied record variable `b` that contains fields like `box` which is a rectangle field that describes the local coordinates of the bounding box of the space that it is to draw inside. There are many other kinds of drawing blocks, the most common of which is the `horz slice` block, which takes the input area, and subdivides it into slices. Inside a slice block, you define the widths of the slice, and by calling the add statement, you can add more slices, thus further subdividing the rectangle. Note that the add statement can be inside an if statement, and loops, so you can dynamically control how the screen is subdivided based on whatever factors you wish, such as the aspect ratio of the screen. Unlike CSS which is by design a static system with no IF or LOOP statements, `Beads` is a completely dynamic layout system.

For example, if I have the following block:

```
vert slice myblock
     add 30 pt subblock1
     add 20 al subblock2
     skip 10 px
     add 40 al subblock3
```

If supplied from the parent block a total area of 400 x 800 pixels, the commands above will create a drawing like this:



In this example we have subdivided the original rectangle into 4 sections. 3 of them will draw something via subroutines, and one of the slices is dead space (the `skip` slice creates dead space). The block dimensions can be given in physical units based on the screen resolution like `pt` for point (1/72 of an inch) or `pc` for pica (1/6 of an inch), or device dependent resolution like `px` for pixels, or it could be specified based on proportional units (`al` for aliquots, a term from chemistry). In this example the first slice is 30 points high, which at 144 dpi would measure out at 60 pixels high. So if the original box was 400 x 800 pixels in dimensions, the system will first subtract the fixed size items of 60 pixels and 10 pixels, leaving 800 - 60 - 10 = 730 pixels remaining. The proportional units are added, and the 20 + 40 aliquots adds up to 60 aliquots, and slice subblock2 will get 20/60th of the remaining space (1/3rd) and subblock3 will receive 40/60th (2/3rd) of the remaining space.

Thus the final slice dimensions are:

`subblock1`	60 pixels
`subblock2`	243.3 pixels
skipped area	20 pixels
`subblock3`	486.7 pixels

which adds up to a total of exactly 800 pixels. These complicated calculations are so common they are built into the language, and by using the proportion system instead of absolute amounts, your products can scale nicely to fit the available screen space. In the modern world, you cannot predict the size of the screen you are going to be on, and pixel densities range from 100 dpi to 500 dpi, a huge range which makes any product that uses fixed pixel dimensions ugly on some target devices. We encourage you to always use physical measurements like points or aliquots.

If you have something that needs to be a fixed size, and you want to absorb leftover space, you can put in one aliquot slice, which has the effect of absorbing all leftover space. The aliquot system is not the same as CSS percentage system, and although it takes a little time to get used to, it produces superior results.

In addition to vertical slicing, there is horizontal slicing, scrolling, two-dimensional grid draw blocks, and a very powerful report system, which allows a classic styled computer printout report with nice alignment, subtotal areas, etc. See the reference manual for more details on the drawing system.

The initialization entry point

The runtime system will call the function with the special name `main_init` once at the beginning of execution. This is where you typically initialize your main data structures, read from preference files, etc. All declared variables start out at the value U, so you only need to initialize values that should be something other than U.

The main drawing function

After the `main_init` function is called, the runtime will then calculate the total screen area available to your program, and call the optional resize event tracker of the `main_draw` function. Then the runtime calls the `main_draw` function, which handles the drawing of the screen. This draw block typically subdivides the screen into smaller blocks. All draw blocks proceed from the main drawing function, as it creates the tree of sub-blocks. The runtime system keeps track of which data is used by each draw block, and if the underlying data changes, it is smart enough to deduce that that block needs to be redrawn. Thus it does an optimized refresh process without any attention from the programmer.

Calculation functions

Functions that calculate a value or change the state of the model are typically put at the end of the program, as they are typically pure functions that do some repetitive computation. A calculation function does not have a draw context, and cannot touch the screen.

Getting started with the language

Primitive data types

The primitive types of Beads are:

`num`	-- a number, stored as 64 bit floating point
`str`	-- a Unicode character string (internally UTF16 in all current runtimes)
`yesno`	-- either yes (`Y`), no (`N`), `U`, or `ERR`, the equivalent of a Boolean type
`color`	-- a color
`regexp`	-- a regular expression pattern
`changelist`	-- an array of find/replace operations (similar to Unix `sed` command)
`meas`	-- a physical unit of measure
`ptr`	-- the address of a node in a tree (not a RAM pointer, a path to the node)
`image`	-- a bitmap image
`sound`	-- sound
`tree`	-- functions as an array, a record, a graph, the Swiss army knife of data
`video`	-- a movie
`bits`	-- a bit string (for low level work)
`bytes`	-- a byte string (for low level work)
`any`	-- a catch-all container that can hold the other types

Every variable has a fixed data declared at compile time. If the type can be inferred from the initialization expression, you don't have to bother to specify it. If you declare a variable with no type specified and no expression from which to infer the type, the variable will be considered to be a `num`, which is the most commonly used data type.

In addition to the primitive data types, the standard library includes some essential record types (which are of data type `tree`):

`a_xy`	-- a 2D point with fields `x, y`
`a_xyz`	-- a 3D point with fields `x, y, z`
`a_rect`	-- rectangle with fields `left, top, width, height`
`a_gradient`	-- a gradient definition, with the color bands, etc. specified

The tree data type is a hybrid between a graph and a tree, something we call a Supertree, as it has properties above and beyond a regular computer tree that people are used to from S-expressions and JSON. It is a graph structure that allows you to store values, records, arrays, arrays of records, records with array fields, and relationships between nodes of a tree. Mastery of the Beads languages revolves almost entirely on learning how to structure your model in a simple, clean manner using trees. See the reference manual for more information.

Base value of U

Beads has a universal null value similar to JavaScript's `undefined`, the value `U`. All variables are set to `U` at the beginning. The value of `U` is compatible with any other data type, including strings, bitmap images, sounds, etc. It is a universe base value. There is a second value that is compatible with all data types, and that is the error value `ERR`, which occurs very rarely, when one does some invalid operation, like trying to take the square root of a negative number.

Indenting is significant

Beads is similar to Python, in that it uses indenting as syntactically significant. Instead of using braces for the IF…THEN…ELSE clauses, just as in Python, you indent to indicate a code glob that will be treated as a unit. However, notice that the colons required in Python are not present in Beads:

```
if condition
      a = 3
else
      a = 4
```

In the example above, if the condition evaluates to YES, then the variable a will be set to 3, otherwise it will be set to 4. This statement can be recast in a more compact form using what in JS and C they call the ternary operator, but it has a different syntax (following Python's model):

```
a = 3 if condition else 4
```

The `yesno` type is used in place of Boolean

Beads does not use term Boolean or BOOL like C for true/false values but instead uses the data type `yesno`. Because all datatypes are compatible with `U` and `ERR`, it means that unlike C a variable of type `yesno` can have four values at any moment. It starts out at `U`, might be set to `Y` (equivalent to yes, true, 1) or `N` (equivalent to no, false 0), and could become `ERR` in very unusual circumstances. The single biggest adjustment compared to previous languages you may have used, is one must be careful to consider the four possibilities. Unlike C, which allows any value but 0 to be considered "truthy", in Beads a condition is not true unless it evaluates to `Y`. The statement:

```
if val
   log "condition was evaluated"
```

The message would be printed only if `val` has the value `Y`. If could be any other value like `N`, `U`, or `ERR` the condition will not be executed.

Line comments

Beads allow three character patterns to indicate the start of a line comment:

```
// line comment
-- line comment
==== line comment
```

This makes it much easier to make pretty boxes around function names:

```
============================
//    CHESS
============================
```

In addition there are inline comments which are ignored by the compiler are intended for computer generated code comments for hooking up to code generation programs which need to inject markers in the middle of a line. Backticks which are used in JavaScript signal an interpreted string are instead reserved in Beads for meta programming, and the tokens inside the backticks will be treated as a comment and ignored by the compiler.

```
a = `this is a comment` 3  //  equivalent of a = 3
```

Block comments

Beads allows two different block comment patterns:

`/* …. */`   and   `(* ….. *)`

The second one makes it easy to comment out blocks of code that already have comments.

Line continuation

In an indent-significant language there is the problem of lines that are too long to comfortably fit in the edit window. Although there is no line length limitation, it is considered good practice not to go past 120 characters, so it will occasionally be necessary to continue a statement on the next line. If you end a line in a comma, the compiler will automatically continue to the next line, but if you have a long series of data items, to save the trouble of having to mark each line as continuing, we use a marker on the first line, and the line will be continued until the end mark appears:

```
var A = [1, 2, 3,                    -- ended with a comma so it automatically continues
 4, 5, 6]

var B = myfunction( @+               -- uses the begin continued line block signal (@+)
     1234 * 1353 - 1234
     *3 - 7 * 1234 - 8234
     + 1234 )
     @-                          -- ends the block of continued lines with the @-
```

Function calls

Beads has a very flexible function call syntax. It supports recursion, named fields with default values, variable number of parameters, and other advanced features. One unusual feature is that if the function being called has no parameters, you can skip the empty parentheses, which saves typing:

```
myfunc()
myfunc     -- abbreviated form of a call to a function with no parameters
```

Most of the system library functions like draw_str, have one or two required parameters, and then use optional named parameters that have default values:

```
draw_str("hello", color:GREEN, opacity:0.3)
```

-- in the above call, the first parameter is a required positional parameter, the other parameters are named as they are optional.

There are a few functions that are used so frequently that one should memorize their names and basic options. You will use them constantly:

```
draw_rect -- draw a rectangle
draw_oval -- draw a circle or oval
draw_str -- draw some text
draw_line -- draw a line segment
draw_input -- draw a text entry field

solve_rect -- solve for a rectangle given a series of constraints
```

String substitutions

In JavaScript one uses backticks with a `` `value=${expression}` `` syntax to indicate an embedded expression inside a string. In Beads, you use regular double quotes, and can skip the $ in front of the braces. Our statistics show that one uses embedded values 10 times more often than the plain brace, so to get a plain brace use the backslash to escape the brace.

```
ss = "total={mytotal}"

tt = "a string with a brace: \{ in it"
```

String concatenation

String concatenation is denoted by `&`. The `+` operator is only used to mean numerical addition. There is no operator overloading allowed in Beads (although there are user defined operators available).

```
msg = "part one" & "\npart two"   -- concatenate two strings together
```

Arithmetic operators

The classic +, -, *, / operators are used for addition, subtraction, multiplication and division

Unlike JS, which has no exponentiation operator, in Beads you use the caret (^) for exponentiation. The compiler will only allow integer exponents with this operator, as units of physical measurement require integer exponents:

```
val = 2^3 -- two to the third power, 8

val = 16^1|2 -- sixteen to the ½ power, which is 4.
```

Note that the vertical bar character is used to mark fractional powers not the slash which means division, so that the expression `16^2/3` would mean 16 squared, divided by 3. To calculate 16 to the 2/3rds power, you would write that `16^2|3.` To raise a number to a non-integer power, use the `pow()` library function.

Comparison operators

The comparison operators are:

`<`			`less than`
`<=`	`≤`	`U+2264`	`less than or equal`
`==`			`equal`
`<>`	`≠`	`U+2260`	`not equal`
`>=`	`≥`	`U+2265`	`greater than or equal`
`>`			`greater than`

Note that because the `!` character is used for other purposes, Beads uses `<>` instead of `!=` like most other languages. In your source code, the Unicode forms of less than equal, not equal, and greater than or equal are equivalent to the 2 character forms.

Logical operators

Unlike JavaScript which uses `!` for logical NOT, `&&` for logical AND, and `||` for logical OR, and `^` for XOR, Beads uses the keywords: `and`, `or`, `xor`, and `not`.

Single value assignment operations

Beads can be written with assignments to the left or two the right. We encourage people to use the left to right form whenever practical, because it is more readable.

```
len = 3  -- left assignment of 3 into the variable len

3 => len   -- right assignment of 3 into variable len
```

When copying entire records, which in the graph database world of Beads a subtree, you use the long form of the assignment operator, which means copy the entire record, not just one value:

```
location <=== { x:12, y:14 }  -- left assignment of a tree into the variable location

{ x:12, y:14 } ===> location   -- right assignment form
```

There are shortcuts to add or subtract 1 from a number, or to toggle a `yesno` value, or to swap a value:

```
inc val   -- same as  val = val + 1
dec val  -- same as  val = val - 1
toggle val  -- flip the Y or N value of the yesno variable
swap a <=> b   -- exchangle the values
```

Subtree operations

Because Beads entirely revolves around the graph database structure we call a Supertree (because it is superior to a plain tree), there is a series of operations that affect the tree in various ways. If you are treating the tree as an array, you might use prepend to put a value in front, append to add a value to the rear of the array. You can merge a record with another, or copy which has the effect of erasing the old values and putting in the new. See the user manual for a more complete explanation of all the details of the operations.

```
trunc dest                -- truncate the tree at the destination, leave the destination value alone
clear dest                -- truncate the tree, and clear the destination value to U
clear field1, field2  in dest  -- clear selected fields in a record
renum dest                -- renumber the subtree starting at 1
touch dest                -- mark the tree as being modified, but don't change any values
append src => dest index:i    -- append a value to an array at the end
prepend src => dest index:I    -- add a value in the front of the array
merge src ===> dest    -- copy the source record fields, but leave other fields alone in the destination
move src ===> dest    -- copy the source to the destination, then erase the source
swap src <===> dest    -- swap two subtrees with each other
join  a <===> b       -- create a relationship between two different subtrees
```

Flow of control

Conditional statements

```
if condition1
     …statements…
elif condition2
     …statements…
else
     …statements…

case value
| 1
     …statements…
| 2
     …statements…
else
     …statements…
```

Looping

All looping is done with one loop statement. It has many options, which allow you to iterate forwards, reverse, or traverse a subtree in various orders like depth-first, or breadth-first. You can iterate based on a sorted array or by indices. You can ask for temporary loop variables to be set which allow you to interrogate the index of the loop, or the value of the array at that node, or how many times the loop has been executed. See the user manual for the list all the different loop options.

```
loop reps:5              -- loop 5 times
loop from:2 to:8 by:2 -- loop from 2, 4, 6, 8
loop from:1 to:5 rev:Y -- loop from 1 to 5 in reverse (5, 4, 3, 2, 1)
loop array:[2,3,5,7]   -- loop across an array, which will generate the sequence of: 2, 3. 5, 7
loop top_down:mytree -- traverse a tree, going from the top downward
loop while:total<5   -- iterate while a condition stays Y
```

Misc. control statements

`exit` -- breaks out of the current level of a loop
`exit` *label* -- breaks out of a labeled loop

`continue` -- jumps to the bottom of the current level of a loop
`continue` *label* -- jumps to the bottom of a labeled loop

`return` -- simple return from a function
`return` *val* -- return a value

`nop` -- no operation, sometimes needed as a placeholder in preprocessor IF statements to guarantee some code in a conditional clause

Advanced features not covered in this guide

Client/Server communication

You can subscribe a client to a server, and attach to a subset of a tree, and receive atomic updates in the background as the data on the server changes.

Once attached to a server, the client can call functions with parameters passed that exist on the remote server, which can update the data structures that are subscribed to.

Preprocessor

There are preprocessor directives which allow execution at compile time, and special meta-character sequences that permit computer generated code to be conveniently included without worrying about tab indent matching.

Internationalization

There is a syntax for marking string literals that are to be translated, as well as accessing strings that change based on quantity. External tools can scan a project for strings, and build string tables for translation.

User defined operators

There are 5 user defined binary operators, which allow the programmer to extend the arithmetic capabilities of the system.

Defining your own physical units of measurement

You can create your own physical units in an existing family like defining fathoms equal to 6 feet, and you can also create your own unit families with new fundamental units.

Finite state machines

There is a data type in the language of a finite state machine. You define how the machine operates, and in your program you can create a machine, and update the state with operations, and the system will track each instance of the machine using the logic you specify.

Derivation functions

In the Excel spreadsheet program, when you put a formula into a cell, when the inputs to the formula change, that cell is automatically recalculated and redrawn. This feature of automatic natural order of recalculation has an analog in Beads, whereby you can specify parts of a record that are derived from other fields, and a derivation function specifies how to calculate the derived quantity in general for any record containing that field, and any reference to the derived field will execute the formula to generate it as needed, a kind of lazy evaluation.

Bits and bytes

There are data types allowing byte string and bit string operations, which are useful in interfacing with low level external systems.