

books, eBooks, and digital learning

[Home](#) > [Articles](#) > [Programming](#)

## Browser BASIC (BBASIC): Adding an Easy-to-Use and Portable Language to a Web Page



By [Jeff Friesen](#)  
May 18, 2015

 [Print](#)  [Share This](#)


Page 1 of 1

With Browser BASIC (BBASIC), a JavaScript application that embeds a BASIC language interpreter in a web page, you can create and use programs written in a variation of BASIC. Jeff Friesen introduces BBASIC with a tutorial and tour of the BBASIC architecture.

### Like this article? We recommend



[Learning to Program](#)

[Learn More](#)  [Buy](#)

*Browser BASIC (BBASIC)* is a JavaScript application that embeds a BASIC language interpreter in a web page. BBASIC lets you load, enter, run, and save programs written in a variation of the BASIC language. BBASIC provides sophisticated control structures, includes a wide variety of functions, supports graphics, and offers other features.

### Related Resources

[Store](#)

[Articles](#)



**[Beginning Programming in 24 Hours, Sams Teach Yourself, 4th Edition](#)**

By [Greg Perry](#), [Dean Miller](#)

Book \$23.99



**[C++17 Fundamentals LiveLessons Part I \(Video Training\)](#)**

By [Paul J. Deitel](#)

Online Video \$119.99



**[Programming for Mixed Reality with Windows 10, Unity, Vuforia, and UrhoSharp](#)**

By [Dawid Borycki](#)


Book \$39.99

[See All Related Store Items](#)

### Like this article? We recommend



[Learning to Program](#)

[Learn More](#)  [Buy](#)

## NOTE

BBASIC is an experiment in creating an easy-to-use and portable browser language whose programs can run on desktops, tablets, and smartphones. More work remains to be done in order to achieve this goal.

I'll introduce BBASIC in this article by presenting an application-usage tutorial. (I've shown it in a Firefox browser context here, but I've also tested the software with other major desktop browsers.) We'll tour BBASIC's architecture by examining its HTML entry-point file, its three supporting JavaScript libraries, and its supporting PHP server code.

## Using BBASIC

This section provides a brief tutorial on using BBASIC. To follow along, point your desktop browser to [BBASIC's startup page](#). Figure 1 shows the startup screen in Firefox, revealing BBASIC 1.0.



**Figure 1** BBASIC's startup screen tells you how to obtain help.

## Getting Help

To get help on using the BBASIC application, enter the command HELP—or he1p, or any other combination of these four letters (BBASIC is not case-sensitive)—as indicated on the BBASIC startup screen.

BBASIC's help system offers many help screens. The first screen appears in response to your HELP command (see Figure 2). The next seven help screens require you to specify a digit for the screen number (2–8) after typing HELP. In addition to these initial eight help screens, individual commands, statements, and functions have their own help screens.

## Like this article? We recommend



[Learning to Program](#)

[Learn More](#)

[Buy](#)

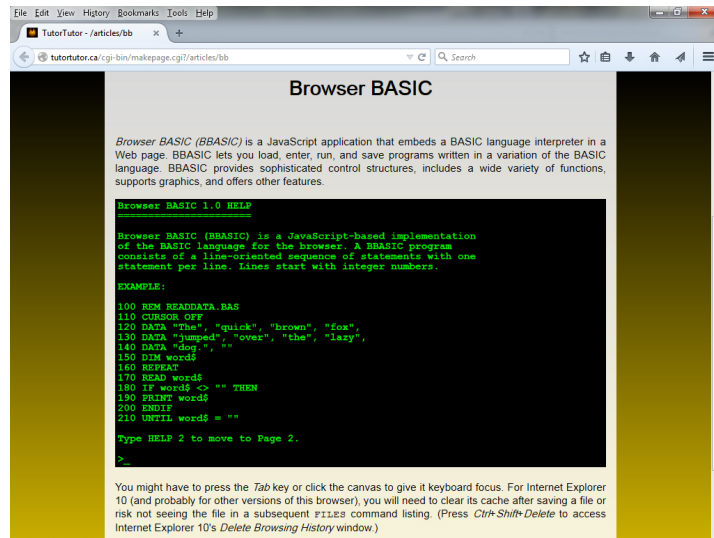


Figure 2 BBASIC's help is organized into multiple screens.

## Checking Out the Demo Programs

After studying BBASIC's help screens, you'll probably want to create some programs. Start by checking out the [demo programs](#) that I've included with this application; they'll give you a sense of how a BBASIC program looks. You can obtain a list of demos by entering the FILES command. Figure 3 shows the resulting list of files.

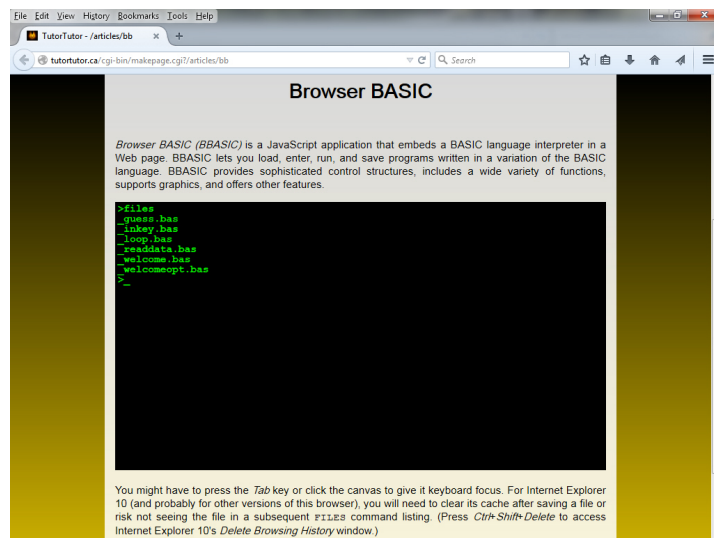


Figure 3 Files whose names begin with an underscore are read-only.

## Like this article? We recommend



[Learning to Program](#)

[Learn More](#)

[Buy](#)

The `_inkey.bas`, `_loop.bas`, and `_readdata.bas` files store short programs. For example, `_inkey.bas` prompts you to press a character key and displays the result. Figure 4 shows how to load, list, and run this program via the `LOAD`, `LIST`, and `RUN` commands, respectively.

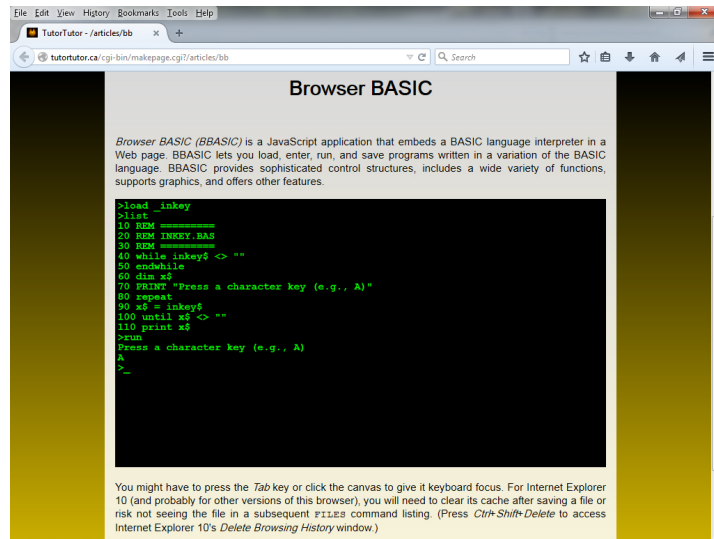


Figure 4 Loading, listing, and running `_inkey.bas`.

`_inkey.bas` starts with three `REM` (remark) statements that identify the program. Next, a `WHILE/ENDWHILE` loop construct and the `INKEY$` function drain the keyboard buffer before checking for a keystroke. Moving on, a `DIM` (dimension) statement defines a variable; in this case, a string variable (indicated by the `$` suffix) named `x`.

At this point, a `PRINT` statement outputs a message telling the user to press a character key, and a `REPEAT/UNTIL` loop construct repeatedly invokes `INKEY$` and assigns the result to `x$` until this variable contains something other than the empty string. Once a character key is pressed, this loop terminates and another `PRINT` statement prints its value.

`_loop.bas` demonstrates a `FOR/NEXT` loop, along with `CURSOR OFF` and `CURSOR ON` statements for hiding and showing the cursor (flashing underscore character). `_readdata.bas` demonstrates the `READ` and `DATA` statements for accessing and defining embedded data, the `IF/ENDIF` decision statement, `REPEAT/UNTIL`, and `CURSOR OFF/CURSOR ON`.

Finally, `_guess.bas` and `_welcome.bas` respectively present a number-guessing game (guess a number from 0–100, inclusive), and present a short demonstration of BBASIC's graphics capabilities. `_welcomeopt.bas` provides a slightly optimized version of `_welcome.bas`. Figure 5 shows the graphical screen for `_welcome/welcomeopt`.

## Like this article? We recommend



[Learning to Program](#)

[Learn More](#)

[Buy](#)



Figure 5 Graphics are rendered on the HTML5 canvas.

## Creating Your Own BBASIC Programs

After playing with the BBASIC demos, you'll have a better idea of what a BBASIC program looks like. Before creating your own program, enter the NEW command to ensure that no program is in memory. NEW erases a stored program and resets important program variables to their initial values.

Each statement begins with a line number, and the result is known as a *line*. The line number isn't used by the program; it simply serves to identify lines, so you can easily insert new lines between existing lines. For example, consider the short program in Listing 1 for printing "Hello, World" five times via a loop:

### Listing 1—hw.bas.

```
10 DIM i
20 FOR i = 1 TO 5
30 PRINT "Hello, World"
40 NEXT
```

Create this program by entering each line verbatim, starting with the line number. After you press Enter/Return at the end of each line, BBASIC stores the line in memory. After you've finished entering all of the lines, enter the RUN command (with no preceding line number). BBASIC executes the program and produces the output shown in Figure 6.

You can save this listing to a file by using the SAVE command. For example, SAVE hw saves Listing 1 to hw.bas. Because files whose names don't begin with an underscore character can be erased, someone else might overwrite your file unless you privately host BBASIC on your own server. (SAVE doesn't let you prefix a filename with an underscore.)

## Like this article? We recommend



[Learning to Program](#)

[Learn More](#)

[Buy](#)

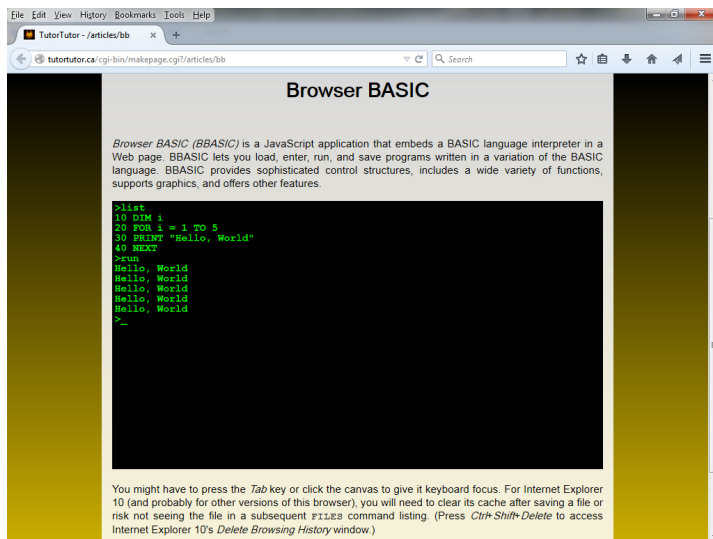


Figure 6 BBASIC outputs five copies of the "Hello, World" message to the console.

TIP

Use the RENUM command to renumber a listing. RENUM is handy when you need to insert a line between two existing lines with adjacent line numbers (for example, lines 7 and 8).

Touring the BBASIC Architecture

In this section, we'll tour BBASIC's architecture. This architecture consists of an HTML entry-point file that defines a canvas-based user interface, along with interpreter, console, and scanner JavaScript libraries. Some of BBASIC's commands also depend on PHP server scripts. The files in the following table contribute to BBASIC's architecture.

File	Description
BBASIC.html	BBASIC HTML-based user interface and controlling JavaScript code
BASICI.js	BASIC interpreter JavaScript code
Console.js	Console management JavaScript code (used by BBASIC.html and BASICI.js)
Scanner.js	Scanner JavaScript code (used by BBASIC.html and BASICI.js)
BBASIC_files.php	Server support code for FILES command
BBASIC_kill.php	Server support code for KILL command

Like this article? We recommend



[Learning to Program](#)

[Learn More](#)

[!\[\]\(35dc653d59570f8f891c312eeece91a2\_img.jpg\) Buy.](#)

## BBASIC.html

The `BBASIC.html` file drives this application. The file's header imports `BASICI.js`, `Console.js`, and `Scanner.js`, and its body declares a canvas element and a script, which executes when this file is loaded. The script accomplishes the following tasks:

1. Declare string variables whose contents define the various help screens.
2. Initialize the console and echo startup text.
3. Initialize the BASIC interpreter.
4. Declare a `mode` variable and initialize it to `0`.
5. Declare a `tick()` function that runs the interpreter. Pass this function, along with `15`, to `setInterval()`, so that `tick()` is invoked every 15 milliseconds.

The real "meat" of `BBASIC.html`'s JavaScript code lies in the `tick()` function. This function gives the interpreter a time-slice before returning control to the browser, in order to prevent the browser from displaying an "unresponsive script" dialog box (which wouldn't impress the user).

`tick()`'s behavior depends on the value assigned to `mode`. When this value is `0`, `tick()` obtains the next line from the console by calling `Console`'s `getLine()` function. If `null` is returned, no new line (line content followed by a newline character) has been entered.

When a line *is* entered, the line is passed to the scanner for parsing. The scanner returns the first token, which is a command (`CLS`, `FILES`, `HELP`, `KILL`, `LIST`, `LOAD`, `NEW`, `RENUM`, `RUN`, or `SAVE`) or one of these:

- `EOLN` if a blank line was entered.
- `INTLIT` if the user is entering a line number followed by a statement.
- `ID` if an unknown command was entered.
- Or something else (garbage).

The `mode` variable remains initialized to `0` until you execute the `RUN`, `FILES`, `LOAD`, `KILL`, or `SAVE` command. When you execute `RUN`, `mode` is initialized to `1`. If you execute any of the other commands, `mode` is initialized to `2`. This is done because these commands may take a while to execute, and they must not delay the browser thread.

When the `mode` value is `1`, `tick()` knows that a program is being executed. Execution takes place by repeatedly calling `BASICI.run()` until this function returns false. All `tick()` has to do is invoke `BASICI.run()` again. When it detects that this function returns false, it shows the cursor (in case a program forgot to restore the cursor) and displays the `>` prompt.

When the `mode` value is `2`, `tick()` knows that an `Ajax` call is in progress. This call will perform file I/O with the server asynchronously, perhaps to obtain a list of stored files or to load a file into memory. When the I/O completes (successfully or otherwise), `mode` is restored to `0` so that a new command can be entered by the user.

## Supporting JavaScript Libraries

`BBASIC.html` depends on the supporting `BASICI.js`, `Console.js`, and `Scanner.js` JavaScript libraries. `BASICI.js` depends on `Console.js` and `Scanner.js`. This section reviews these supporting libraries, beginning with `Console`, continuing with `Scanner`, and concluding with `BASICI`.

### Console Library

`Console` is a JavaScript library for managing a text-oriented console. It relies on the HTML5 canvas element for displaying console output and obtaining console input. `BBASIC.html` defines the following canvas element for use by `Console`:

```
<canvas id="screen">
HTML5 canvas element not supported by this browser.
</canvas>
```

## Like this article? We recommend



[Learning to Program](#)

[Learn More](#)

[Buy](#)

In this code excerpt, the canvas is identified as `screen`. This identifier, along with the desired number of columns and rows, is passed to Console's `init()` function, which is used to initialize the console and connect it to the canvas:

```
Console.init("screen", 80, 26);
```

After the console initializes, introductory text is echoed to the console via Console's `echo()` function. Text is written to the cursor's current location, which advances to the right and down (scrolling canvas text vertically when necessary):

```
Console.echo("Browser BASIC 1.0\n");
Console.echo("=====\n\n");
Console.echo("Type 'help' (without the quotes) to obtain help.\n\n");
Console.echo(">");
```

I mentioned previously that the `getLine()` function is used for obtaining the next entered line of text. You can learn more about `getLine()` and the other Console functions in a two-part series that I wrote for [SitePoint](#):

- ["Add a Web Console to Your Toolbox, Part 1"](#)
- ["Add a Web Console to Your Toolbox, Part 2"](#)

After the SitePoint series was published, I upgraded Console to better support BBASIC. Specifically, I introduced several new functions, shown in the following table.

Function	Purpose
<code>arc()</code> , <code>bezierCurveTo()</code> , and other graphics functions	Support GRAPHICS statement
<code>hideCursor()</code> and <code>showCursor()</code> functions	Support CURSOR OFF and CURSOR ON statements
<code>setCursorPos()</code>	Supports LOCATE statement
<code>getNumCols()</code> and <code>getNumRows()</code> functions	Support SCREEN() function
<code>getHeight()</code> and <code>getWidth()</code> functions	Support SCREEN() function
<code>getKey()</code>	Supports INKEY\$() function
<code>isEsc()</code>	Used by <code>getLine()</code> and BASIC1's <code>run()</code> function to determine that the user wants to stop an executing program prematurely

I also upgraded `getLine()` to detect an Esc keypress, erasing the current console row in response.

### Scanner Library

Scanner is a JavaScript library for extracting tokens from a string. It's used by the parser to parse each entered command, or to parse each statement being executed.

Scanner provides a minimal API consisting of the following functions:

## Like this article? We recommend



[Learning to Program](#)

[Learn More](#)

[🛒 Buy](#)



- `init(text)` initializes the scanner. The text to be scanned is specified by `text`.
- `scan()` scans the next token. `Scanner.Token.type` records the token's type, and `Scanner.Token.lexeme` records the token's lexeme (its character sequence). `Scanner` provides predefined token types for language tokens; for example, `Scanner.Token.ABS`.

After obtaining a line from the console, `BBASIC.html` executes the following code sequence to initialize the scanner to the line that was entered, as well as scanning the first token:

```
Scanner.init(line = line.trim());
Scanner.scan();
```

**BASICI Library**

BASICI is a JavaScript library for interpreting a sequence of BASIC statements. Its public API consists of several functions. The following table describes the most interesting functions.

Function	Description
<code>init()</code>	Initializes the interpreter.
<code>list(begnum, endnum)</code>	Lists program lines ranging from <i>begnum</i> (beginning number) to <i>endnum</i> (ending number).
<code>new()</code>	Removes a program from memory.
<code>renum()</code>	Renumbers the stored program. The first line is assigned 10 and subsequent lines are assigned successive increments of 10.
<code>run()</code>	Runs the program.

Additionally, `addLine()`, `findLine()`, and `removeLine()` are used to store a new line or remove an existing line; and `getListing()` returns a stored program for saving to a file by the `SAVE` command.


After outputting `BBASIC`'s startup messages to the console, `BBASIC.html`'s JavaScript code invokes `BASICI.init()` to initialize the BASIC interpreter. The other functions mentioned earlier are invoked from within `tick()` as necessary.

`BASICI.js` is fairly easy to follow. However, you might find the JavaScript code that handles `BBASIC`'s `INPUT` statement to be somewhat confusing and deserving of explanation. Consider the following code excerpt:

```
parseInput: function()
{
    Scanner.scan();
    if (Scanner.Token.type != Scanner.Token.ID)
        BASICI.throwError("identifier expected");
    var var_ = BASICI.parseVar();
    if (BASICI.vars[var_.name] == undefined)
        BASICI.throwError("undefined var "+var_.name);
    if (var_.name.charAt(var_.name.length-1) != "$")
        BASICI.throwError("string var expected");
    if (Scanner.Token.type != Scanner.Token.EOLN)
        BASICI.throwError("extraneous text");
```

**Like this article? We recommend**



[Learning to Program](#)  
[Learn More](#)  [Buy](#)

```

        BASICI.input = true;
        BASICI.inputVar = var_;
    },

```

This excerpt specifies the private `parseInput()` function, which parses and partially executes the `INPUT` statement. Notice the final two lines, which initialize `BASICI`'s `input` and `inputVar` variables: `input` is a flag tested each time the `run()` function is called. When true, a line is read from the console via `Console.getLine()` and stored in the variable identified by `inputVar`. The variable is then read. If it doesn't contain null, a line was input into the variable and `input` is cleared to false, which completes the `INPUT` statement. Otherwise, the variable isn't cleared, and the console is reread the next time `run()` is invoked.

The reason for this strange behavior is that each call to `Console.getLine()` processes only a single keystroke in the key buffer, and it returns null until an entire line has been entered (as signified by the newline keystroke). `getLine()` cannot wait until an entire line has been entered, because it could delay the browser thread excessively, leading to some kind of "unresponsive script" message—and an annoyed user.

## Supporting PHP Server Code

BBASIC's `FILES`, `KILL`, `LOAD`, and `SAVE` commands access files stored on the server. Because this file access is slow, they use Ajax's `XMLHttpRequest` object to communicate with the server. Unlike `LOAD`, whose `XMLHttpRequest` call returns the specified file's contents, `FILES`, `KILL`, and `SAVE` need PHP scripts to help them perform their tasks.

Listing 2 presents the PHP script that satisfies the `FILES` command.

### *Listing 2—BBASIC\_files.php.*

```

#!/usr/bin/php
<?php
$files = scandir('/home/jfriesen/public_html/articles/bb/programs');
foreach ($files as $file)
{
    if ($file != '.' && $file != '..')
        echo "$file\n";
}
?>

```

Listing 2 obtains the sorted names of all files in the following directory:

```

/home/jfriesen/public_html/articles/bb/programs

```

Then it echoes back to BBASIC the name of each file in this directory (except for the special `.` and `..` files).

Listing 3 presents the PHP script that satisfies the `KILL` command.

### *Listing 3—BBASIC\_kill.php.*

```

#!/usr/bin/php
<?php
echo "/home/jfriesen/public_html/articles/bb/programs/" . $_GET["name"];
if (unlink("/home/jfriesen/public_html/articles/bb/programs/" . $_GET["name"]) == FALSE)
    echo "unable to delete file\n";
?>

```

## Like this article? We recommend



[Learning to Program](#)

[Learn More](#)

[Buy.](#)

Listing 3 obtains the name of the file to be killed; then it invokes `unlink()` with the file's path and name to kill the file. It returns an error message when it cannot kill the file.

Listing 4 presents the PHP script that satisfies the `SAVE` command.

**Listing 4—BBASIC\_save.php.**

```
#!/usr/bin/php
<?php
$name = $_POST["name"];
$listing = $_POST["listing"];
$fh = fopen("/home/jfriesen/public_html/articles/bb/programs/".$name, 'wb');
if ($fh == FALSE)
    echo "unable to save to file\n";
else
{
    fwrite($fh, $listing);
    fclose($fh);
    echo "file saved\n";
}
?>
```

Listing 4 obtains the name of the file to be written and the listing to be stored in the file, and then it attempts to create this file. If successful, the listing is saved. Whether successful or not, a suitable message is echoed back to BBASIC.

## Conclusion

More work is needed to turn BBASIC into an easy-to-use and portable browser language for use on all kinds of devices. For one thing, BBASIC's production version probably wouldn't feature a command-oriented console, but would access a BBASIC program, like accessing a JavaScript script.

More importantly, BBASIC's performance needs to be improved before this application can be taken seriously. Performance is limited in part by having to reparse each statement before execution. Also, the constant need to return control to the browser thread greatly limits performance; this issue might be overcome by using web workers.

At some point, I might create a compiler to compile BBASIC listings into object files, and then create a virtual machine that leverages web workers to run those object files. Because the compiler can be embedded in a browser, this approach should solve the performance issue and help to obfuscate BBASIC code. Until then, I hope you have fun with the current BBASIC.

[+ Share This](#) [📌 Save To Your Account](#)

Page 1 of 1

## Like this article? We recommend



[Learning to Program](#)

[Learn More](#)

[🛒 Buy.](#)