



November 3, 2019

Business Rules Dealers:

Contents:

- NEW CONSOLE
- Background
- Atom Fundamentals
- BR Language Support
- Support for Existing Custom Libraries
- Autocomplete Methodology
- Collaboration Features
- Additional Features
- Lexi Support
- Future Development
- The San Antonio BR Conference 2020
- BR Prices and Policies
- Dealer Discount Spreadsheet

NEW CONSOLE

For years now Gabriel has been talking about an advanced programming language editor that enables a very high level of programmer collaboration and steps a programmer through **context sensitive parameter definition**. Recently Chris developed a BR editor that achieves nearly all of Gabriel's stated objectives. Chris and Gabriel have refined that solution for Business Rules such that 1) we can presently use the editor's advanced capabilities, and 2) we can proceed to make this editor into a fully functional console.

In today's modern programming world the basic underlying language is not nearly as important as the custom tool set or library typically developed by you the programmer. Both the basic language and the library are important. But *libraries are getting more and more recognition as programming language extensions*.

Perhaps you remember when we moved the BR documentation from PDF to wiki format. People wondered at the time why such a move was necessary and if it was worth the trouble. Now most of us rely on the wiki nearly every day and wonder how we ever got along without it. Likewise once you are comfortable using the New Console you will wonder how you ever managed to write programs without it.

Background

During the last ten years editors have evolved with some very powerful features. While some of the early editors such as vi, PICO, Notepad++, EditPad, and EMACS have very powerful features, they often came in the form of cryptic abbreviations that took a while to learn and some were far from intuitive. While notepad++ and EditPad have evolved over time, more modern editors provide more useful features in the way of collaboration and context sensitive help. Additionally *they can be customized* by the user ***and those customizations are easily shared among users***. As a result editors have been evolving through open source contributions by many users.

One of the main platforms of such editors is Electron. Electron allows for the development of desktop GUI applications using web technologies (i.e. JavaScript, HTML, etc.). It includes the Chromium rendering engine and the Node.js runtime. Electron is the main GUI framework behind several notable open-source projects including Atom, GitHub Desktop, Light Table, **Visual Studio Code**, and WordPress Desktop.

Chris selected Atom as the foundation for the new BR console. Microsoft is migrating to the same technology (Electron) with their Visual Studio Code project. Chris and Gabriel have defined extensions to Atom that are specific to BR and placed them into the public domain on GitHub. Dozens of Atom extensions from other developers are readily available and easily installed. The BR extension is only one of many.

It is my intention to blend the Atom based editor with Business Rules in such a way that it fully replaces the existing console window for all activity other than low level debugging.

Atom Fundamentals

- A multitude of UI and Syntax themes (color schemes, etc.) are available from an integrated theme library. For an animated example see <ftp://brulescorp.com/atom/Theming.gif> (MS Word does not support animated gifs so we have provided links to the gifs.)
- Completely free and open-source
- Git and SVN integration
- Cross platform: MacOS, Windows, Linux
- Integrated file browser
- Multiple Cursors – support for changing many instances of a keyword simultaneously. For an animated example see ftp://brulescorp.com/atom/Multiple_Cursors.gif

BR Language Support

- Autocomplete for all BR statements and system functions. For animated examples see <ftp://brulescorp.com/atom/Autocomplete.gif> and <ftp://brulescorp.com/atom/Snippets.gif>
- Autocomplete snippets for Fileio and Screenio.
- Autocomplete for any libraries referenced by the source code being edited.
- Automatic compiling and de-compiling of BR code.
- Opening a '.br' file with Atom automatically extracts source for editing.
- Integrated compile-time error handling.
- Free and open-source mainly maintained by Chris and Gabriel at Sage but completely open to the community for enhancement through the Github project.

Support for Existing Custom Libraries

Context Sensitive Autocompletes for your libraries as they exist now (in addition to several other syntax trees). These look inside your libraries (via library statements), and even your file layouts, in order to provide autocompletes for everything that would make sense to type into the current program statement.

Clicking on the "More.." button on the autocompletes, further displays the documentation for the function, statement or command. See below for **a method provided for documenting each parameter of a function definition which is**

then presented to the user during autocomplete processing. This means *you can document your custom functions* in an easy format, *one parameter per line*, where the function is defined and that documentation will be presented via autocomplete whenever you attempt to specify that function, typically in another program.

Autocompletes are provided for all Statements, Internal Functions, Local Functions, Library Functions, and File Layout Subscripts. Variables that are listed for selection in autocompletes include those dimmed and those listed as parameters in the current function, or appear as local variables in the referenced function (unpassed optional parameters).

Autocomplete Methodology

All Autocompletes are context sensitive. The autocomplete engine builds a list of all entries that are applicable to the *current cursor position*. This list is then reduced by Atom based on what letters you've typed so far, in a fuzzy search. The fuzzy search means that it finds what you're looking for even if you misspell it slightly.

Context Sensitive Autocompletes are built from:

1. Any functions mentioned in your library statements:
 - a. If you're currently in a library statement - These include the function names without parameters for all library functions in the library.
 - b. Otherwise - These include the function names *with parameters* for all functions mentioned in library statements.
2. All of your local and global variables in your current program. Local variables are anything listed in the function you're currently editing. Globals include anything Dimmed in the current program.
3. Any file layouts that are opened in the current program, by examining the file layouts. Keep in mind this is open source so if you have custom dictionaries they can be integrated with this process.
4. All Internal Functions and Statements
5. FileIO and ScreenIO Libraries

Collaboration Features

A built in Chat Box is provided so you can write any questions you have while working, and all programmers using this BR Atom extension can see them. This is a way to ask general BR programming questions of the entire group

instantaneously, or answer them. This is like all of us sitting together in a virtual office, so we can work together and ask questions of each other while we each work on our separate projects.

Atom can also use a facility called Teletype, which allows you to invite another programmer to look at your code with you, and you each can have your own cursor (similar to Google Drive documents), so that you can both edit the code together. This makes programming by a two or more programmers convenient and encourages collaboration and group learning.

Additional Features

Full support of all MyEdit features.

Shift-Ctrl-Up and Shift-Ctrl-Down instantly search the document.

Tab and Shift Tab indent or outdent all selected text.

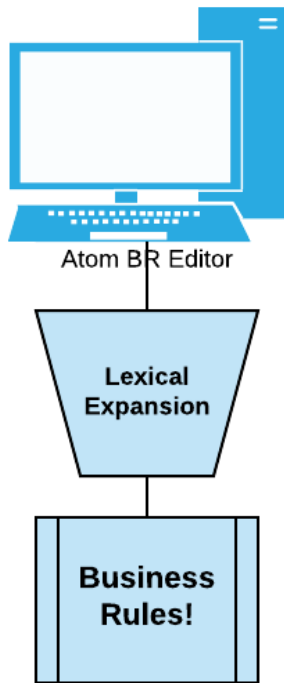
Fully customized and updated Syntax Coloring, that handles all BR statements.

Selecting any keyword and pressing F1 on it loads the wiki documentation for that keyword.

Autocompile on Save -- so you just press Ctrl-S and it saves your .brs file and compiles your .br file.

If you highlight a section of code and press CTRL-/, it automatically comments that whole block of code out.

Lexi Support



As you may know, there are a number of enhancements to BR syntax that are in the queue. But these have been deferred pending upgrades of internals such as wxWidgets and SSL. We also paused BR internal development when we upgraded the BR Web Server.

Thankfully that didn't stop Gabriel, (and now Chris). Gabriel, in addition to developing FileIO and ScreenIo, created Lexi, which provides much more than facilitating source code with no line numbers.

Lexi is the key to expanding BR syntax without changing BR itself. Lexi supports the following capabilities (4 – 6 are new):

- 1) Elimination of line numbers during editing. For those of us who presently depend on line numbers, *this is a scary thought*. But

line numbers are really unneeded when you use a modern editor. And the freedom of cutting and pasting text and using various custom indentation schemes without concern for line numbers is delightful.

One of the conundrums associated with support for source files with no line numbers occurs when you are debugging programs which are either saved without source or running with a *run-only* version of BR. When an error is reported at a particular line number, or when stepping through a program that does not display source, it is generally necessary to have a local copy of the program with the same line numbers in order to know what the remote program is doing.

By using **autonumber** statements in your program, you can maintain preassigned line number ranges for standard program sections, which greatly reduces the program version coordination required for debugging.

```
! #Autonumber# 43900,10
```

One significant aspect of using the extended syntax that Lexi supports is that *you will want to keep your program masters in source only form*. There are several aspects of program formatting that are not supported by the present BR console such as retaining original upper and lower case in names, as well as compression techniques like substitution strings.

- 2) Substitution strings. In the following example, wherever `[[ScreenControls]]` appears in the program source, the long list of parameter values are substituted:

```
!. "#Define# [[ScreenControls]] = "mat ControlName$, mat
FieldName$, mat Description$, mat VPosition, mat HPosition,
mat FieldType$"
    def fnEditScreen(fScreenIO, fScreenFld, ScreenName$, mat
ScreenIO$, mat ScreenIO, [[ScreenControls]]
```

- 3) We have implemented SELECT CASE as a precompiler directive. This means, when editing your programs, you can write SELECT CASE statements and they will automatically be converted into IF THEN - ELSE IF THEN statements when the program is compiled. See Lexi on the wiki for details.
- 4) Multiline (section) comments. Whole sections can be commented by using the `/* ... */` method of commenting.
- 5) Interpolated strings. This feature facilitates making long strings consisting of concatenated values more readable (**see example below**). When Lexi encounters a backtick (```), also known as a grave accent, (located just left of the top row of numbers on most keyboards – looks like **a reverse apostrophe**) it regards it as the start of a multiline string definition – which ends with another backtick.

Any characters between the backticks are combined into a single string at compile time. **Carriage returns** are replaced with the text `"&hex$("0d0a")&"` so that Carriage Returns in your interpolated string operate as carriage returns in the output.

Additionally, within that string any values enclosed in double braces are considered to be live independent values. In the example below **name\$** and **year** are variables to be concatenated into a string delineated by backticks. This syntax `{{ }}` is more readable than the “& ... &” syntax used by BR. The BR concatenation syntax is inserted in place of the braces at compile time. This automatic concatenation of embedded terms is what is meant by interpolation. Note that since *year* is numeric, it is automatically enveloped by a `STR$()` function.

```

2  |
3  | let Name$="Chris"
4  | let Year=2019
5  |
6  | ! #Autonumber# 100,10
7  | let msgbox( !
8  | ..`This is a multiline
9  | ..String handled by "Lexi"
10 | ..These were invented by {{Name$}} in {{Year}}.
11 | ..`)
12 |
13 | ../* This is a multiline comment in Lexi
14 | ..Handled like a c++
15 | ..multiline comment */
16 |
17 | ..def library fnCsvExport(Layout$*64; !_ The file layout to export
18 | .....DialogType, !_ Dialog Option Specs for Layout
19 | .....Filename$*300, !_ Filename to export to
20 | .....IncludeRecNums) ! Boolean flag to export Rec Num
21 |
22 | .....! Some code
23 | .....! Some more code
24 |
25 | ..fend
26 |

```

- 6) Parameter list continuation. A new symbol “!_” (note the underscore after the !) is used to comment parameters in function calls. Each parameter of a function call can appear on it’s own line of source code. The comments that follow the parameter are made available during autocomplete prompting. This enables you 1) to document your custom function parameters in the library where they are defined, and 2) to have that documentation

automatically presented to you while you are coding references to that function.

When you are coding and you type FN a search is performed of the current program and all libraries referenced by it to identify all functions available to the program. Then as each character in the function name is typed the autocomplete dropdown list is reduced accordingly allowing you to select a function name from the list. Then as each parameter is typed the original definition of that parameter is displayed along with the text following the !_ sequence.

Future Development

- 1) The ability to step through a program using the debugger, similar to MyEditBR needs to be provided.
- 2) In general I would like to tightly link BR to AtomBR so that this new editor effectively replaces the existing console, except for showing traces and data during debugging. In other words most commands and responses will come through the Atom window, but when a program is running, the results of BREAK and DISPLAY commands (including step by step statement display) will appear on the old console.
- 3) The *automatic startup of the Atom console when BR starts* will lead new users to do their coding in Atom and this will become the standard method of using Business Rules!

This is where you come in. **We skipped billing maintenance for last year but we need to finance these developments. So I am including half of last year's maintenance in this year's maintenance invoice. So the invoice is for support for 2 years but last year is at half price.** I am excited and grateful for your support and for the good faith Chris and Gabriel have shown in producing this wonderful facility.

All of the revenues for annual maintenance will directly be used to finance these changes before any leftover is applied to other expenses.

The San Antonio BR Conference 2020

This conference is going to include a full explanation and demonstration of the New Console, and some items following up from last year concerning the web server. If you really want to see how all of this works, come on down!

BR Prices and Policies

(Nothing has changed here from the previous letter.)

AS OF JULY 1, 2013 BR VERSIONS PRIOR TO 4.2 HAVE NOT BEEN ELIGIBLE FOR TRADE-IN (this was announced February 2012). Some exceptions to this policy have been made available to dealers who have upgraded their entire customer base. The reason for this withdrawal of support for prior editions is to reinforce the importance of moving your customer base to new platforms and capabilities. Change can be uncomfortable, but we must change with the computer industry or we will be left behind and our businesses will experience a slow death competitively. On the other hand, legacy BR applications perform quite well using the new BR platforms.

Business Rules new product prices have remained the same since the inception of the WB product in 1984. The price for new licenses of BR will remain the same, including the volume discount rate structure. The dealer discount is based on a ten year purchase history.

Upgrades

The cost for upgrades from one major release to another continues to be \$50 per user increment per level for the dealers on maintenance. This upgrade charge has been uniform since the release of BR version 3.8. It will continue to remain in effect. In other words the charge to upgrade from 4.1 to 4.2 is the same as the cost to upgrade from 4.2 to 4.3, etc. User increments are defined in the “new product” price list as follows:

Increment	Users	Increment	Users	Increment	Users
1	1	6	20-26	11	65-76
2	2-4	7	27-34	12	77-89
3	5-8	8	35-43	13	90-103
4	9-13	9	44-53	14	104-118
5	14-19	10	54-64	15	119-134

Upgrades, for dealers who do not have an annual maintenance contract, are *double the amount for order processing and user increment per level for upgrades*.

Please remember that some operating systems were not available when you bought your BR, therefore it is unwise to expect older versions of BR to run with advanced versions of Windows.

Annual Support

Annual support is indexed by your discount level as follows:

Discount Percent	Annual Maintenance	ODBC
30	700	400
40	850	500
49	1000	600
57	1200	700
64	1500	850

If you wish to reduce your discount level and therefore lower your annual maintenance fee, that is your option. If you feel that you need special terms call me personally.

Each fall we review each dealer's purchase history and make adjustments to the discount levels accordingly. Your annual maintenance invoice will indicate your purchase volume according to our records. And it will indicate your corresponding discount level. (See the underlined announcement at the beginning of this section.)

Dealer Discount Spreadsheet

We have posted a spreadsheet on the FTP site under prices which will tell you exactly how much it costs for any number of users on any platform. Just plug in your discount level and save it for reference.

It is at **FTP://Brulescorp.com/Dll_Distr/prices.**

Gordon Dye