MyEdit – BR Edition

A Multi-Language File Editor.
v3.x

User Guide

Table of Contents

1 Introduction

MyEdit - BR Edition (MyEditBR) is a multi-file editor that has been highly customized for use with the BusinessRules! (BR) programming language. This user guide will introduce you to some of the basic, as well as more interesting, features of the editor.

While MyEditBR is intended for use with the BR! Language it is intended for more general use as a notepad replacement for editing almost any text type of document. It has features that make it useful for any of the known supported language and file types.

While there are a number of new features and enhancements to existing features the biggest change to MyEditBR is the introduction of the paid support features. MyEditBR will continue to exist as a free application for anyone to use with certain base features. Many of the more advanced features now available in MyEditBR are only available with a paid support license.

I feel that, while this is a minor change in my policy, it is necessary for me to continue supporting the product for future releases.

MyEditBR will also be referenced as MyEditBR with in the context of this document.

Ryan Mills

Mills Enterprise Ltd.

2 Quick Overview

MyEditBR is a syntax highlighting editor that supports over 40 different source code file types.

Its easy to use tabbed interface, along with the split view functionality, allows editing multiple files at once.

With features that include spell checking, editor macros and customizable keyboard shortcuts for most of the standard commands this editor is flexible and powerful enough to be includeed in any programmers toolkit.

The small footprint of MyEditBR and it's file-based configuration files means that it can be moved from machine to machine with out the need for complex application installers. As such it can easily be installed upon a USB storage device which can then be used on any computer that supports those types of removable drives.

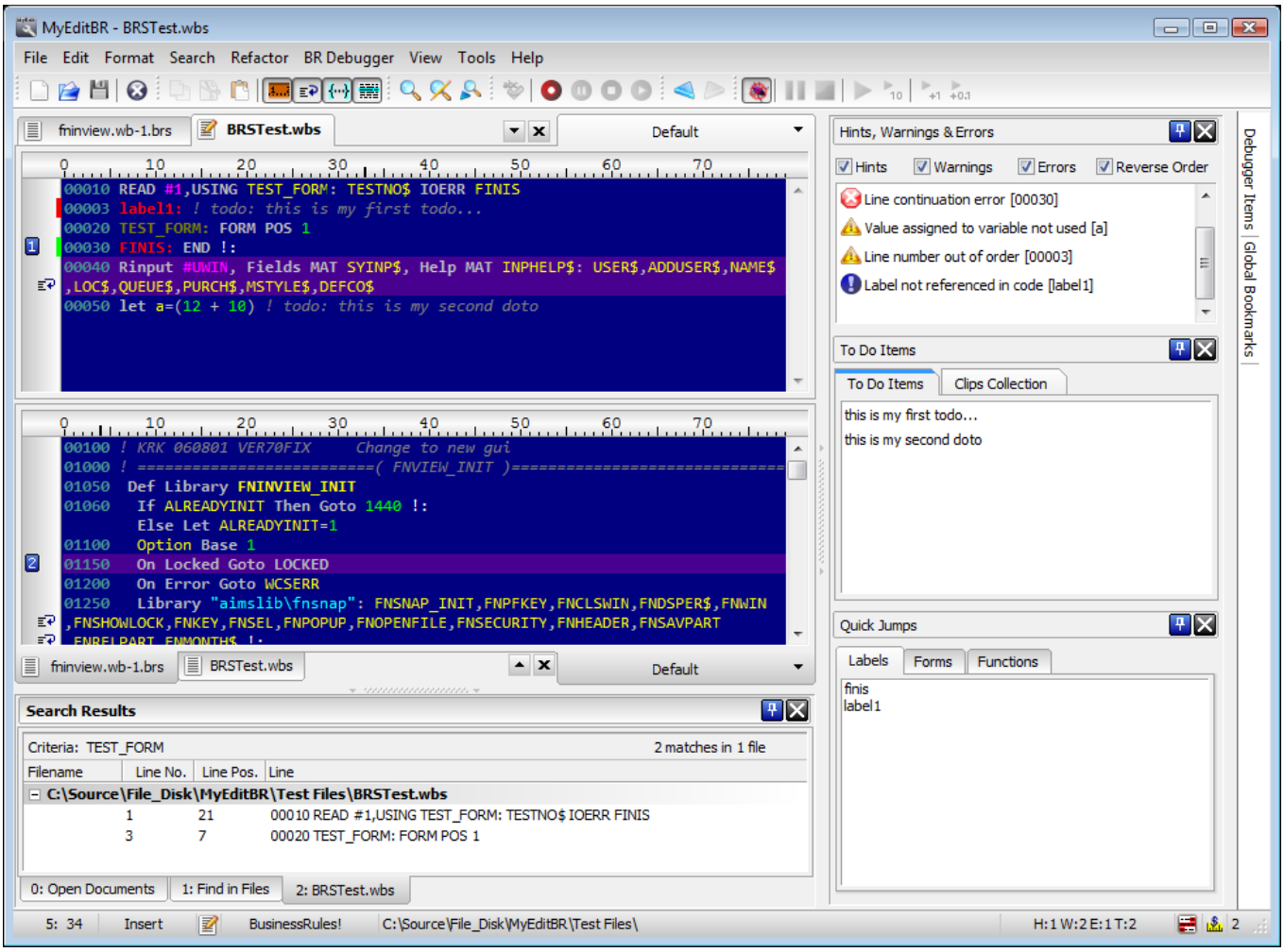
The configurable nature of MyEditBR extends from the UI right down to the individual languages it supports. With customizable Code Completion and Parameter Tool Tips this editor has the power to support custom language enhancements such as user defined libraries or scripts.

MyEditBR has extra support built in for the BusinessRules! language. These additional features include the ability to detect numerous coding situations (Hints, Warnings and Errors) to produce a list of messages for the user. Another aspect of the detection routines built into MyEditBR allows for the automatic detection of in-code methods and adding that information to the Code Completion and Parameter Tool Tips wizards.

With the advent of BR! V4.18 MyEditBR has been re-branded to MyEdit - BR Edition, or MyEditBR for short. Along with this renaming comes a host of new features including a visual debugger and on the fly BR! Syntax checking.

2.1 Editor

Fig. 1 – MyEditBR Main Window



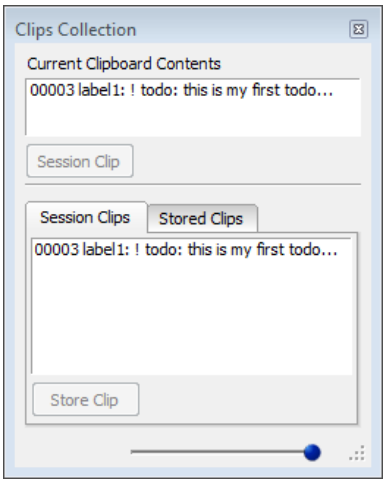
The image in Fig. 1 shows an example of how the editor could look as well as demonstrating a number of the features found with in the MyEditBR application.

2.2 Tool Windows

Tool Windows are sections of the application that reside with in individual windows. These windows can be used in one of two different configurations, docked or floating. Fig. 2 shows an example of what a floating tool window should look like.

When a tool window is floating it will behave like a standard windows form. If it is dragged over one of the three docking areas of the MyEditBR window then you will see the shape of the drag outline change to fit where the docking system will place the tool window if it is dropped over that location. In Fig. 1 you can see 4 docked tool windows: Search Results, Hints, Warnings & Errors, To Do Items and Quick Jumps. There are actually more docked windows than that but not all are being displayed. See **Tab Docks** for more information.

Fig. 2 – Clip Collection Tool Window (Floating)



2.3 Dock Drawers

Dock Drawers are areas of the docking system that can be shut or resized which will affect all docked tool windows. In Fig. 1 you can see two of three dock drawers. There is a grip that acts as a button to close the drawer (See Fig. 3). If the mouse is over the resize bar of the drawer the mouse cursor will change shape to show the resize direction it could take.

Fig. 3 – Bottom Dock Drawer Button

2.4 Tab Docks

Tab Docks are areas of the docking system that can slide a docked tool window into view when the mouse is over the tab of the same name. The tab docked window can then be used normally or can be made to stay permanently visible, this is done through the pinning feature of the tab dock control. Fig. 4 shows both the pinned and unpinned icon.

Fig. 4 – Pinned and Unpinned Icons

In Fig. 1 you can see two hidden tab docked tool windows: Debugger Items & Global Bookmarks.

2.5 Bookmarks

A bookmark is a user-defined place marker in the editor. The user can place a bookmark at the current cursor position and later when they need to can return the cursor to the exact location.

Through the use of the Global Bookmark Manager you can see every bookmark that has been made in any open file in the editor.

Fig. 5 - Bookmarks

2.6 Ruler

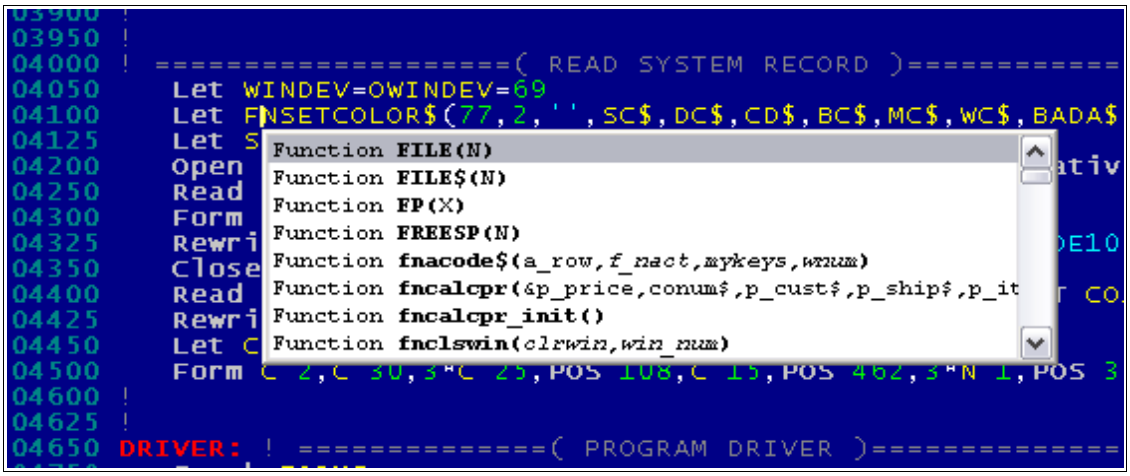
There is an optional ruler that can be displayed at the top of both editors to show the current cursor position as well as the size of the selection block horizontally. See the top of each editor view in Fig. 1.

2.7 Code Completion & Parameter Tool Tips

This feature is available for any language source, as long as an INI file for that language exists, in the Completion Code folder where the MyEditBR application is located.

When the Code Completion is activated, in Fig. 6, you will be presented with a list of items that either match up to what you've typed or everything that's available if there is a blank at the cursor.

Fig. 6 – Code Completion Window



In Fig. 6 then cursor is after the F in Let FNSETCOLOR$ so the Code Completion window provides you with a complete list of all variables and methods that start with the letter F. The list is filtered as you type so if the Second letter after the F is an I then the only items left would be the FILE(N) and FILE$(N) functions.

The Code Completion information is read out of an INI file with a specific name for each language supported by MyEdit. The name of the file must correspond to the same named file as found in Colors folder. The files in the Colors folder are automatically created the first time the Editor Options dialog is opened and closed, if they don't already exist.

With the Parameter Tool Tips optional parameters will appear in an italic font.

To force the Code Completion window open you can use **CTRL-Spacebar**.

To force the Parameter Lookup window open you can use **CTRL-Shift-Spacebar**

The options dialog now provides the ability to edit the Code Completion and Parameter Looks directly. See File Formats for more information regarding the layouts of these files.

2.8 Split View Editing

Commonly split view functionality provides the user the ability to view multiple parts of the same file or view multiple files at the same time. MyEditBR supports 2 different types of Split views, Horizontal and Vertical.

The split view editing functionality of MyEditBR allows for horizontal or vertical views of the current file list. By default, and without a support license, the second "editor" is a read-only concurrent view of any one of the open files during the editing session. The read-only editor will always be the bottom view if the editor is split horizontally and the right side view if the editor was split vertically. The tabs along the bottom of the secondary View allows the user to choose what file to view while working in the primary editor.

2.9 User Tools

User Tools are menu item commands that can be configured to run application or Windows commands with the option of passing in the current files full path and filename.

The **Ctrl-Alt-#** shortcut keys are automatically assigned by MyEdit. You are only allowed a maximum of 10 User Tools and the shortcut keys can not be changed, but the items can be re-ordered to change which keystrokes they are assigned.

2.10 Macros

MyEditBR has the ability to record keystrokes and certain functions and then play them back at a later time. Macros can be complex or very simple depending on the need.

2.11 Word Wrapping

The editor has the ability to wrap long lines to the edge of the editor window or if configured at a predetermined character position. See Editor Options for further information.

2.12 Searching

MyEditBR has multiple search features built-in. Each feature has a different way of looking at the data. The different search options are: Search & Replace, Regular Expressions (RegEx), Compound Items, Find in Files, Find Language Elements and Goto Line Reference

2.13 Menu / Tool bar Configurations

The menus and tool bars are both customizable. Menus can be customized for shortcut keys and tool bars for what buttons are actually displayed. The tool bars can also display large or small icons for the visible buttons.

2.14 Sessions

Sessions are groups of files that can be saved so that they can later be opened with a single command rather than having to open each individual file separately.

2.15 Support Licensed Options / Features

Starting with version 3 of MyEditBR support licensed features have been introduced. No features were taken away from the previously free versions of the application but newer, advanced, features were only available when a valid license has been installed in the application. The current list of licensed features include:

- Unlimited Open Files (Limited to 5 normally)
- Visible Line Status Information
- Language Support Import and Export
- Version Check
- Manual Logging (Support feature)
- Automated Crash Dump Logging (Support feature)
- BR Support (Open/Write .BR/.WB files, Syntax Checking)
- Suggestion Box
- Split View Editing (read/write)
- Conditional Breakpoints (BR Debugger)
- Refactoring (Labels, Named Forms, Variables)

2.16 BR Support

BR Support is accomplished through an external copy of BR running in the background that MyEditBR controls. This allows it to do things like Real-time syntax checking and conversions from compiled BR programs to source and back again.

2.17 BR Debugger

MyEditBR has the capabilities to perform as a visual debugger front end to the Business Rules! environment. This will be covered further in the Debugger section.

3 Options – Program Configuration

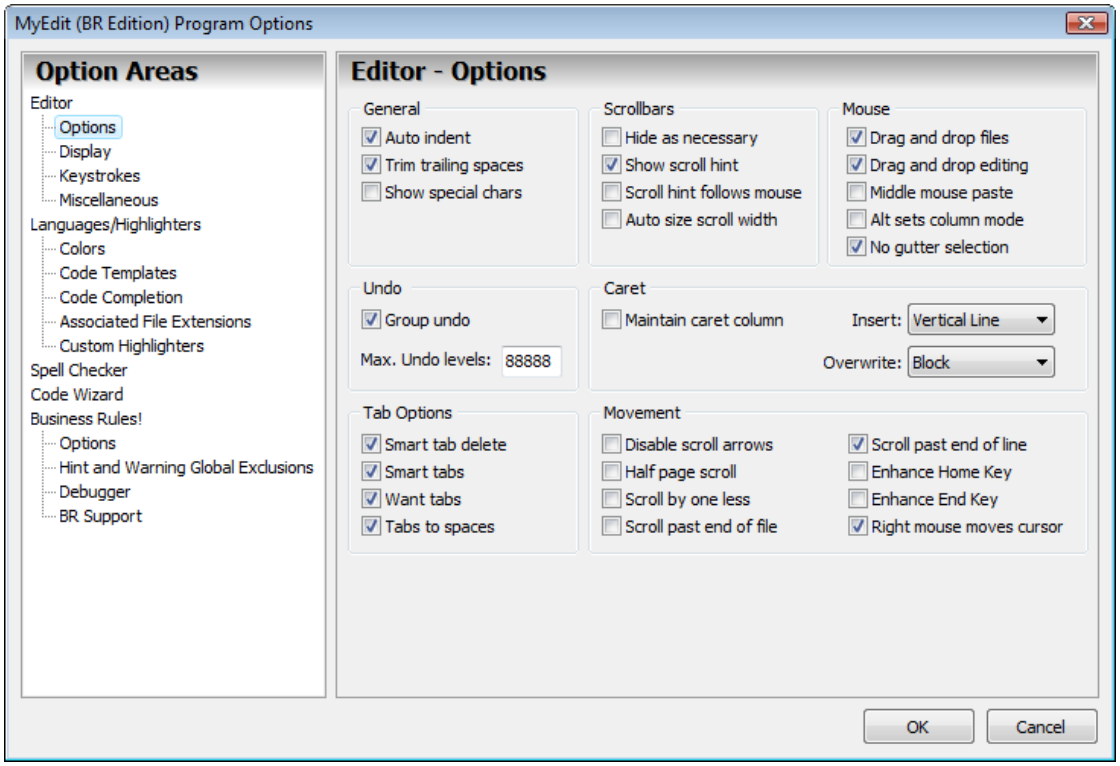
MyEditBR has an extensive set of configurable options. Most options are self explanatory and wont be covered in great depth, this document will try to present information on the rest.

All of the options presented will be broken into similar functional groups. Each group will then present the options in a way that further tries to group them along related operation boundaries. A good example of this is presented in Fig. 7.

The Options dialog is separated into two distinct areas. The area on the left is the navigation pane and provides the first level of functional grouping. The second area is the options pane and this is where all of the actual options are presented for user interaction.

3.1 Editor – Options

Fig. 7 – Options Dialog: Editor Options



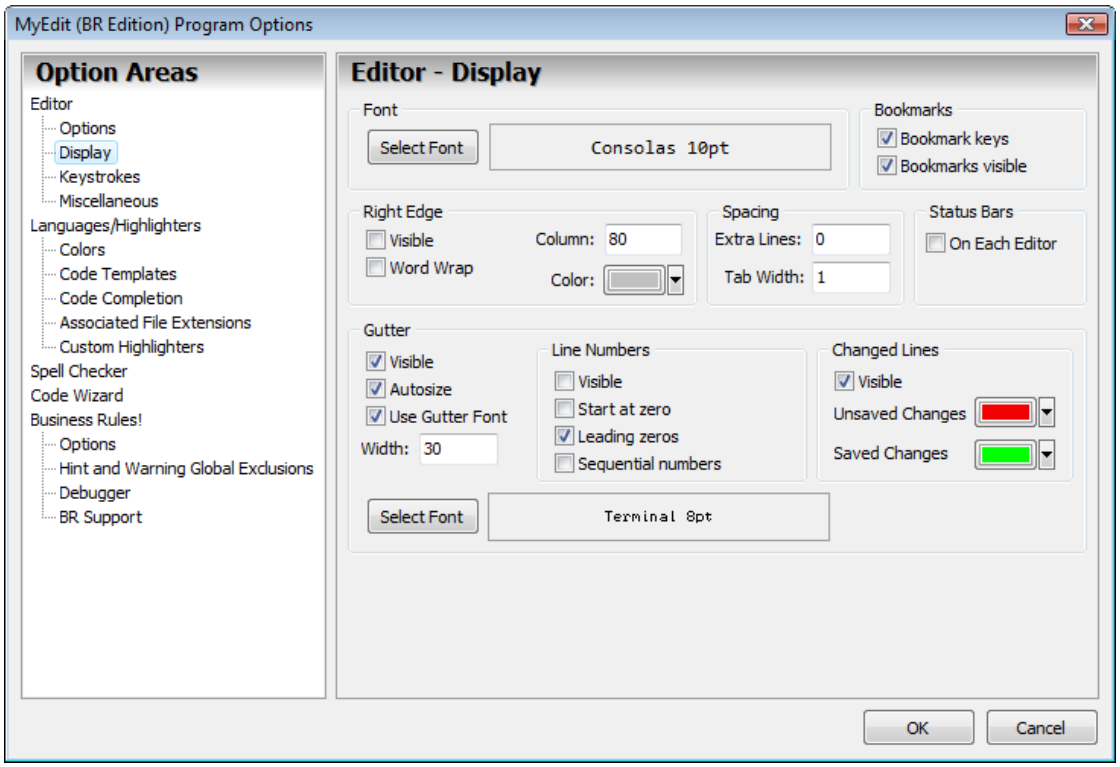
This area contains the general options for controlling how the items in the editor behaves and how certain items are displayed.

Things covered by this page include:

General (Indent, Special characters)	Scrollbars
Mouse (Drag and drop, selection)	Undo
Caret Styles	Tab Options
Movement Control	

3.2 Editor – Display

Fig. 8 – Options Dialog: Editor Display

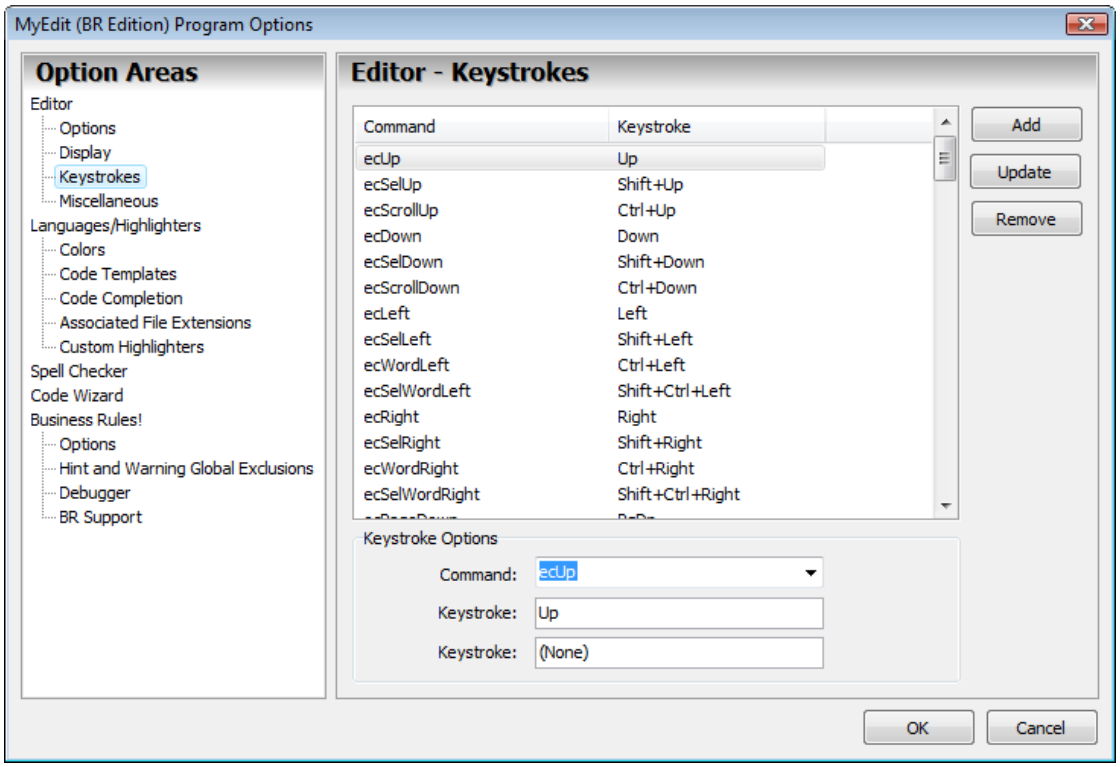


Things covered by this page are:

Font (Editor, Gutter)	Bookmarks
Right Edge	Line spaceing
Status Bars	Gutter
Line Numbers	Changed Line Indicators

3.3 Editor – Keystrokes

Fig. 9 – Options Dialog: Editor Keystrokes

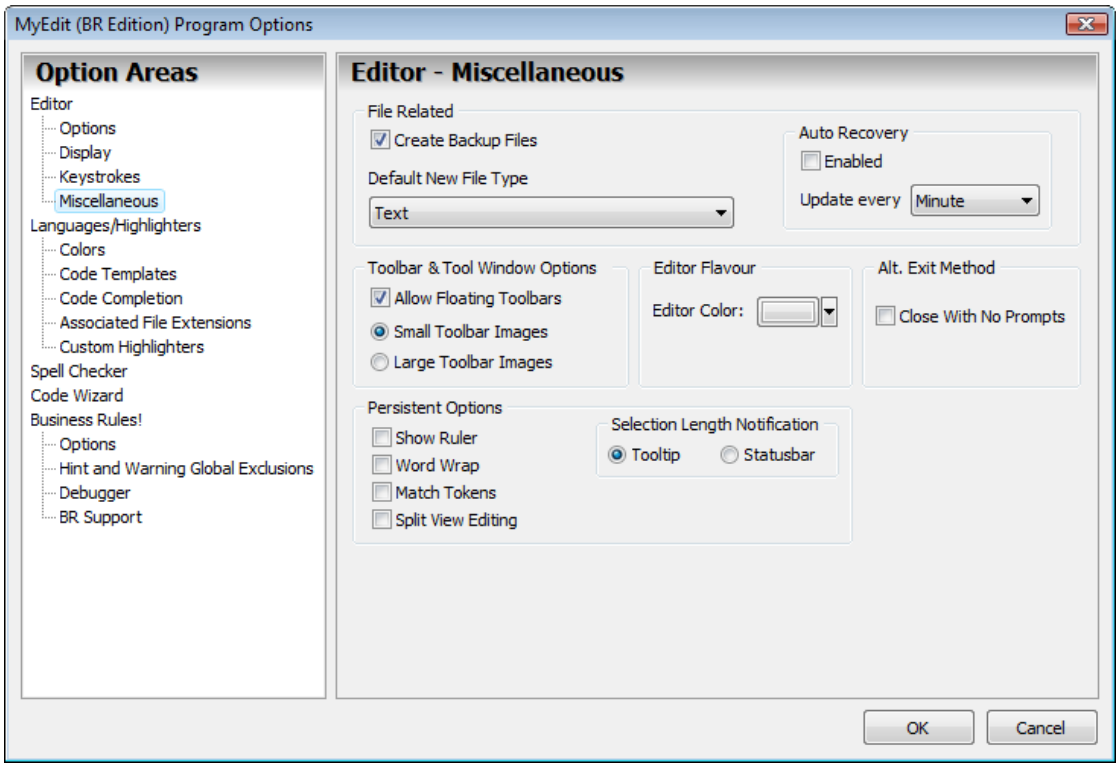


Here you can customize the basic editor keystrokes to match any configuration you want. You can make multi keystroke commands such as Wordstar type commands:

Start Block Select	Ctrl-K	B
End Block Select	Ctrl-K	K

3.4 Editor – Miscellaneous

Fig. 10 – Editor Options: Editor Miscellaneous

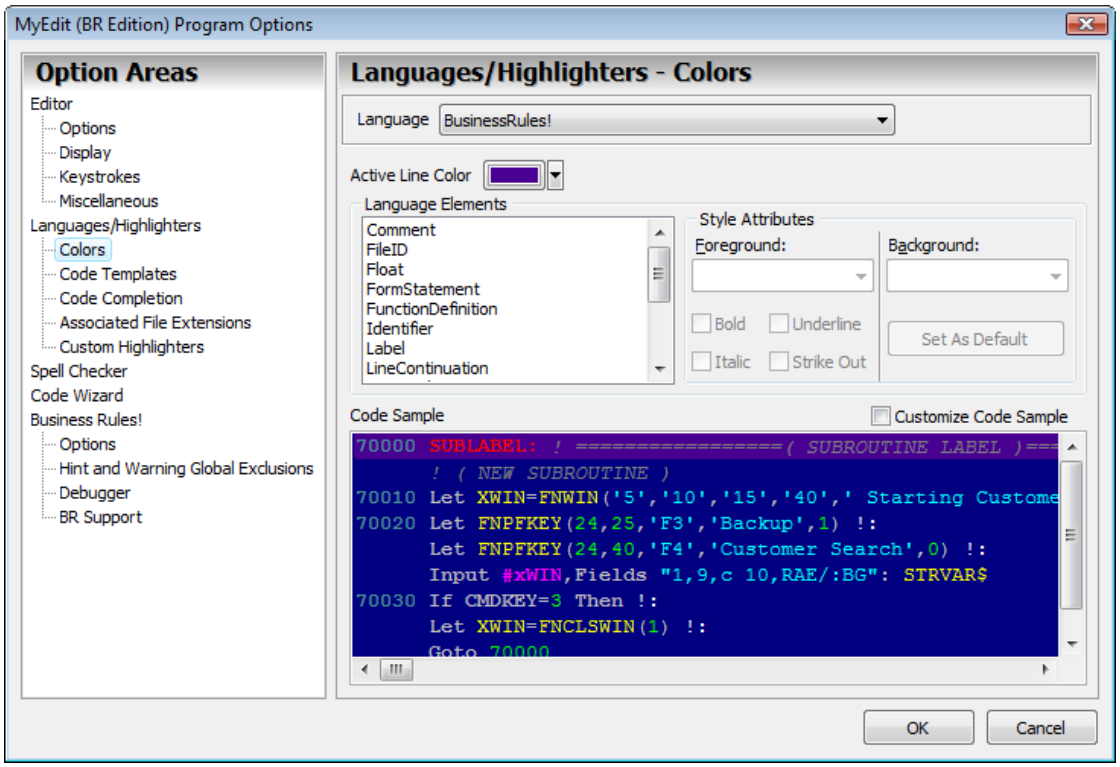


Items covered on this page include:

Backup Files
Default New File Type
Editor Color
Selected Length Notification
Auto Recovery
Tool bar / Tool windows
Persistent Options (Rulers, word wrap)

3.5 Language/Highlighter – Colors

Fig. 11 – Editor Options: Language/Highlighter Colors



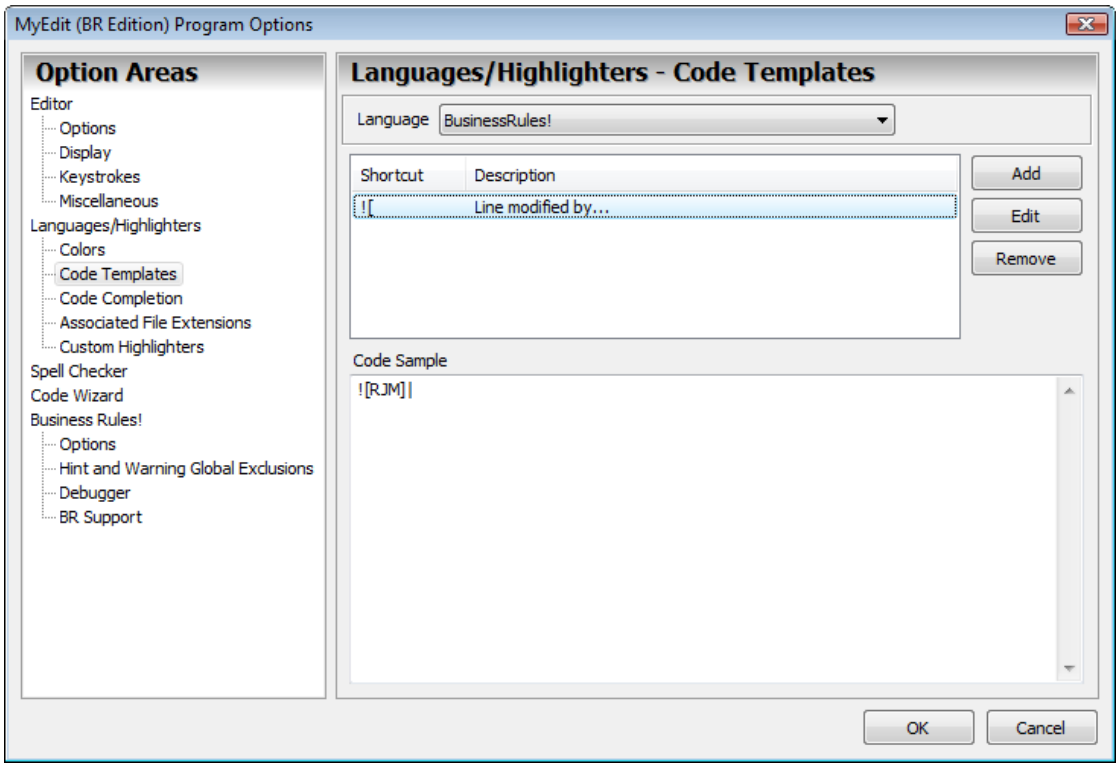
This page allows for the color customization of languages that are supported by MyEditBR.

In Fig. 11 we see what the different elements of the color customizer looks like.

Starting with version 3 of MyEditBR you can actually place specific code into the code sample window to see what effect the color changes will have on working code rather than this limited example code provided.

3.6 Language/Highlighter – Code Templates

Fig. 12 – Editor Options: Language/Highlighter Code Templates



Code templates are a way of abstracting chunks of commonly used text or code, depending on use. In the case of programming, the example shown in Fig. 12 displays a template for a code comment. The template can span multiple lines and has the ability to place the caret at a specific location once the replacement has been completed.

The data of the code templates are stored in a plain text file on the users system. Where this file is located depends upon how MyEditBR was installed and what OS it was installed on. If it was installed with the non-portable option then it will be located in the users Local Application Data folder under Mills Enterprise\MyEditBR\CodeTemplates.

Under Vista my local application data folder is (rmills is my login name):

c:\users\rmills\AppData\Roaming

Under WinXP my local application data folder is

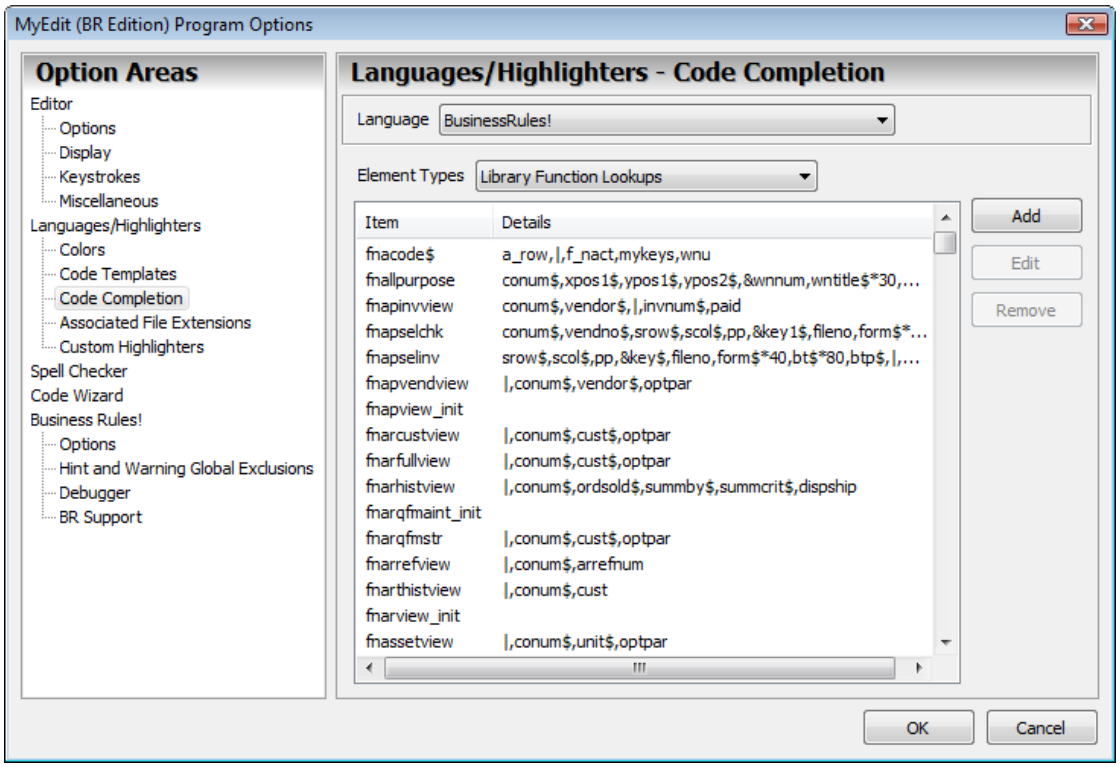
C:\Documents and Settings\rmills\Application Data

The files are identified by common file name extensions. So for example the BR file name would be: **syn_wb.ini**.

This naming convention is used for all language/highlighter data files.

3.7 Language/Highlighter – Code Completion

Fig. 13 – Editor Options: Language/Highlighter Code Completion



Code completion is a powerful tool for any programmer when a large number of methods are available in a programming language. It becomes even more powerful when it can be combined with a custom set of methods. In Fig. 6 you can see how the code completion window looks when it has been activated. The information presented in that window comes from the functions/procedures that have been entered for the given language on this screen.

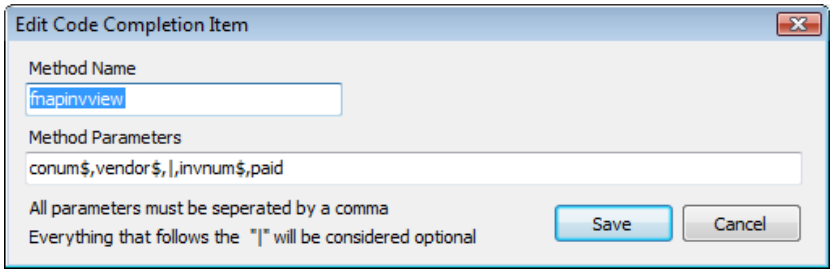
There are 3 different Code Completion areas or element types: Token Matches, Built-in functions and Library Function Lookups.

Token Matches can be setup to highlight common beginning and ending pairs of a given language such as IF/THEN or { }.

Built-In Functions and Library Function Lookups are both very similar but and both appear the same in the actual code completion windows. The difference is intended to provide a base level of functional reference for a language vs. custom library functions.

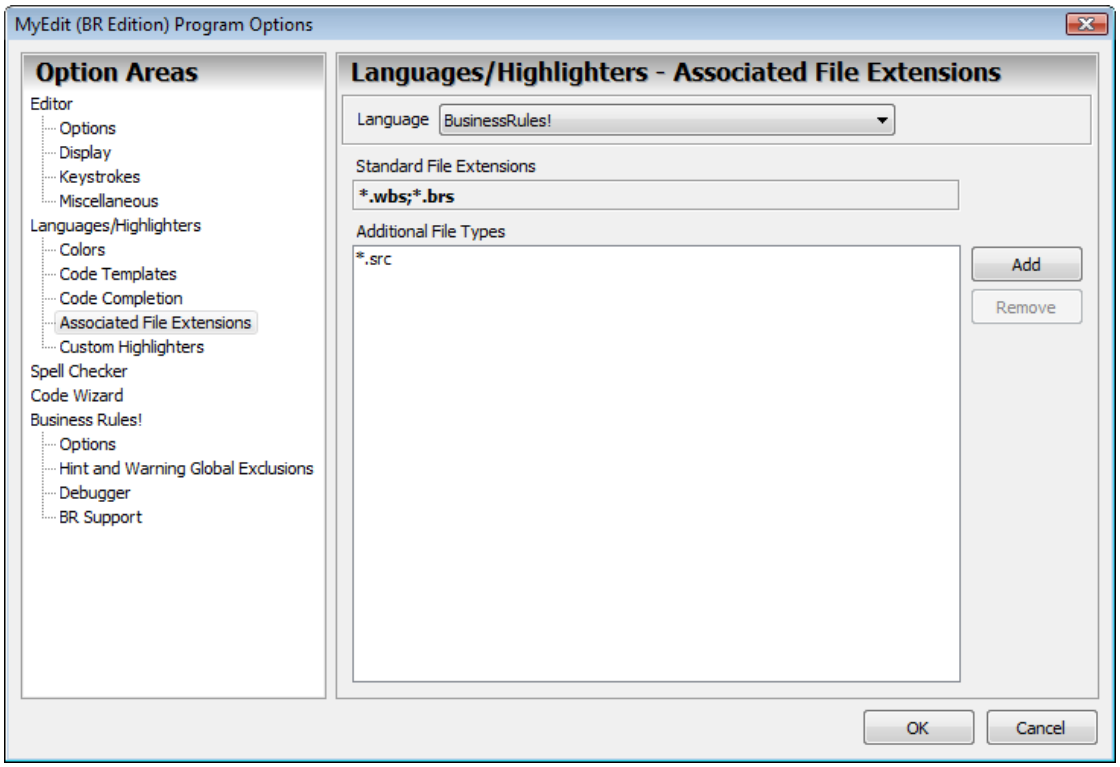
The built-in editor, See Fig. 14, provides details about how to format the function for use in the code completion window.

Fig. 14 – Code Completion Function Editor



3.8 Language/Highlighter – Associated File Extensions

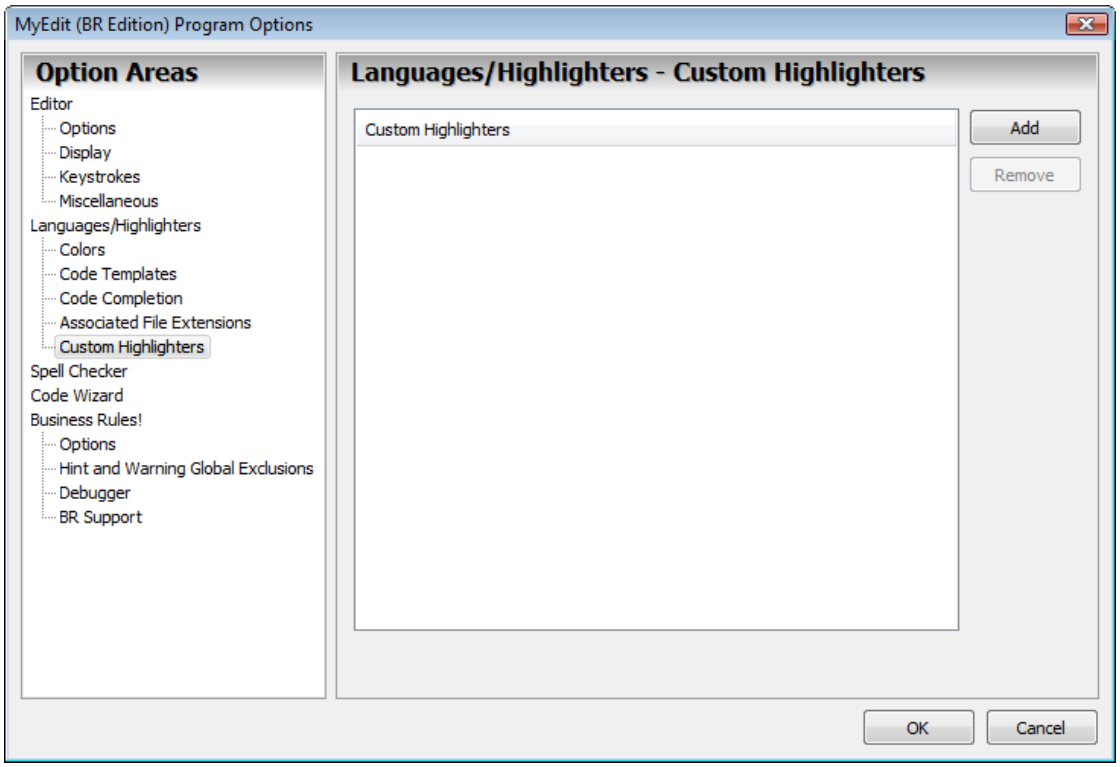
Fig. 15 – Editor Options: Language/Highlighter Associated File Extensions



This section allows the editor to internally associate new file extensions with a specific language highlighter. Each highlighter has a list of known extensions, as shown in the Standard File Extensions list, that it can work with.

3.9 Language/Highlighter – Custom Highlighters

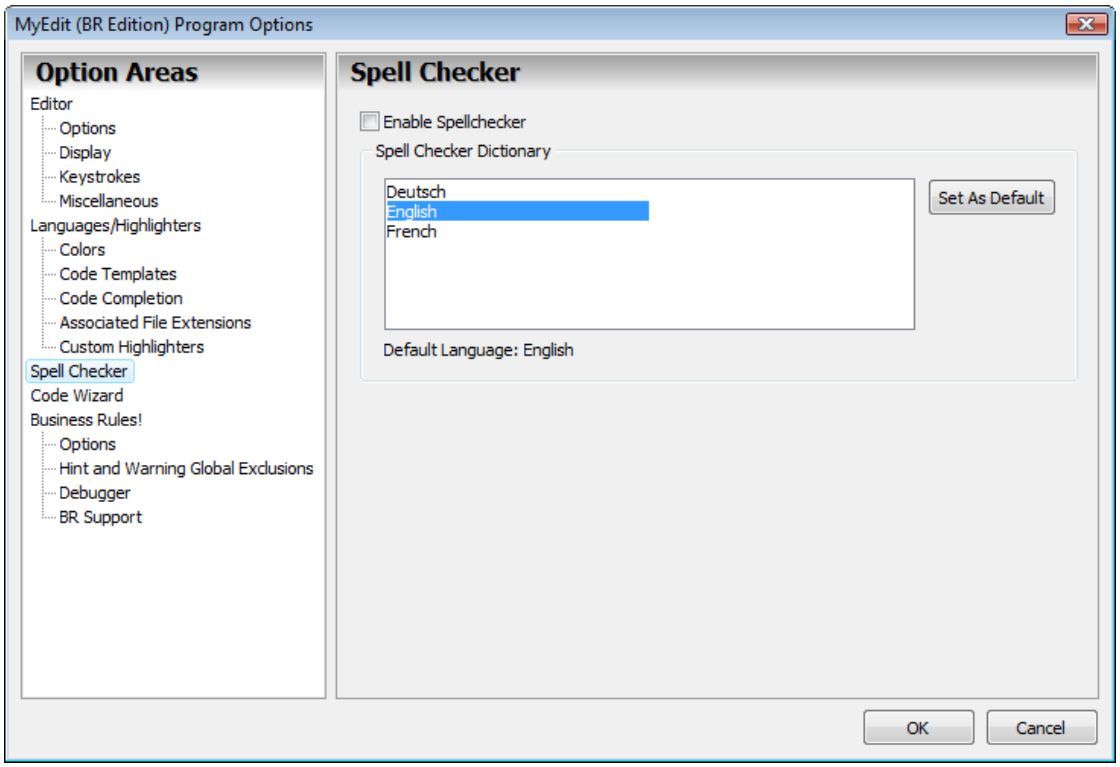
Fig. 16 - Editor Options: Language/Highlighter Custom Highlighters



This option page allows for the inclusion of addition customized highlighter scripts to MyEditBR. Once added, they appear in the rest of the Language/Highlighter pages and can be customized further. See the Custom Highlighter Appendix for details on how to create a custom highlighter.

3.10 Spell Checker

Fig. 17 – Editor Options: Spell Checker



The spell checker options page allows for the selection of the default language dictionary file as well as being able to turn the spell checking engine on or off.

When the spell checker is turned on, it can be configured to spell check specific areas of the current editor file. Each area can be turned on or off from the Tools|Spell Checker menu.

The three areas are:

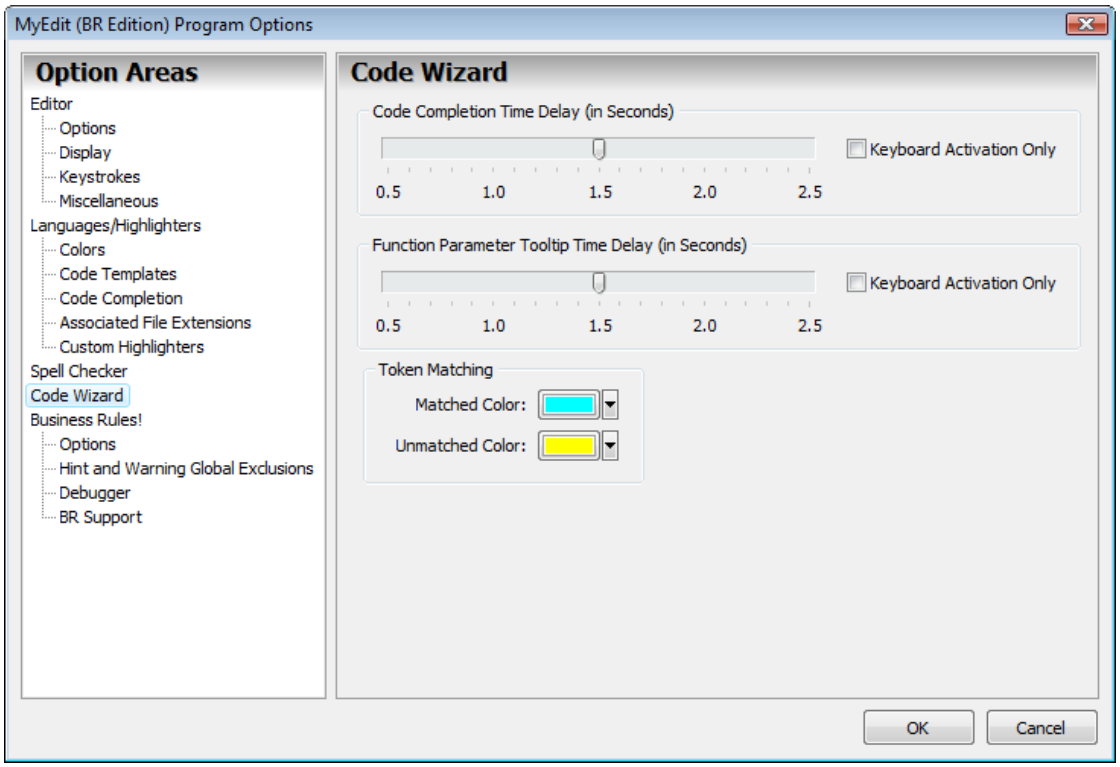
Strings – Strings are typically, in most programming languages, enclosed with either double or single quotes. Strings are usually what the end user sees.

Comments – Comments are usually only for internally documenting code in a program.

Identifiers – Identifiers are can usually be considered everything that does not fall into the other two categories. For spell checking plain text files this is the area that must be turned on.

3.11 Code Wizard

Fig. 18 – Editor Options: Code Wizard



The code wizard helps to control how and when the code completion and the function parameter tool tips are displayed. There are two ways to activate the code completion window pausing typing an Identifier for the selected time delay or by pressing the assigned keystroke.

There are two different keystrokes, one for the code completion window and one for the parameter tool tips.

CTRL-Space activates the code completion window

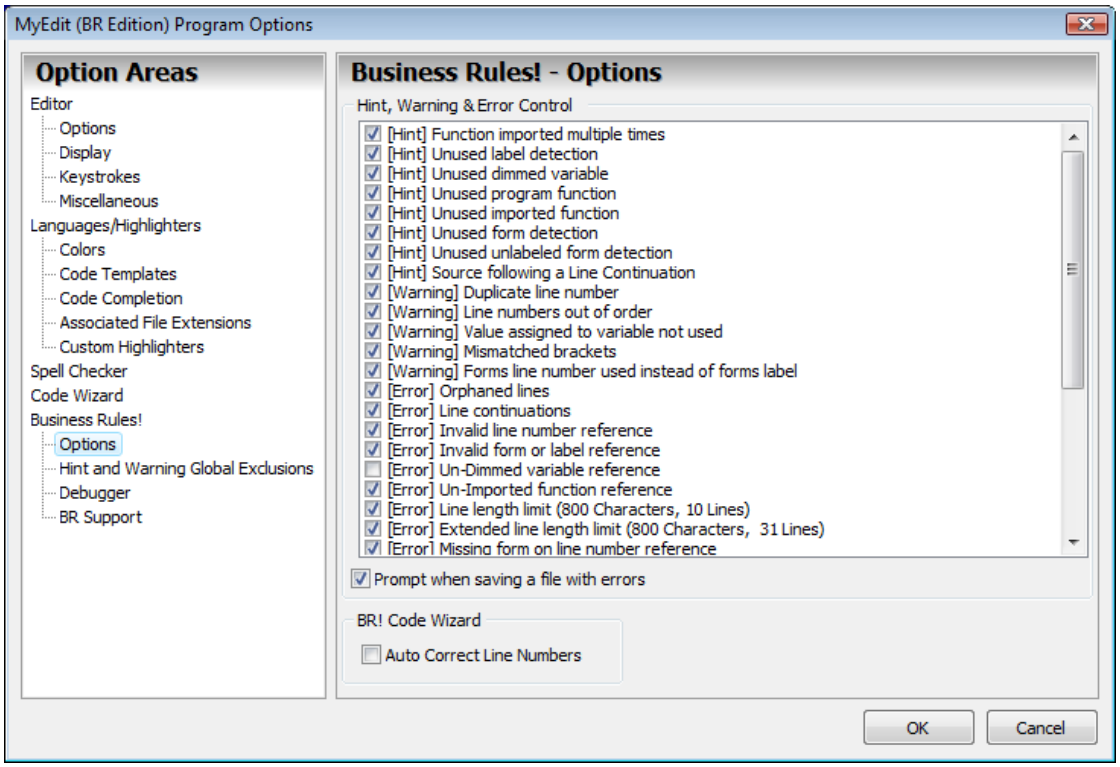
Shift-CTRL-Space activates the parameter tool tips.

The Parameter Tool Tips show what parameter your on for a given function and what else is expected, based upon the information given on the Code completion editor options page, see Fig. 13.

This page also provides access to the Token Matched/Unmatched color selection.

3.12 Business Rules! - Options

Fig. 19 – Editor Options: Business Rules! Options



Business Rules! (BR) is a programming language that doesn't have a good visual editor. MyEditBR has been built to try and fill this position. As such there are a number of features that are in the application that try to help the programmers write better and cleaner code.

Hints, Warnings & Errors (HWEs) are one major example of this. HWEs are a list of messages that MyEditBR can check for when working on a BR source file. HWEs will show up in the HWE tool window when one of the message items is found. By double clicking on the HWE in that tool window MyEditBR will move the caret to the position in the file that it thinks is causing the problem.

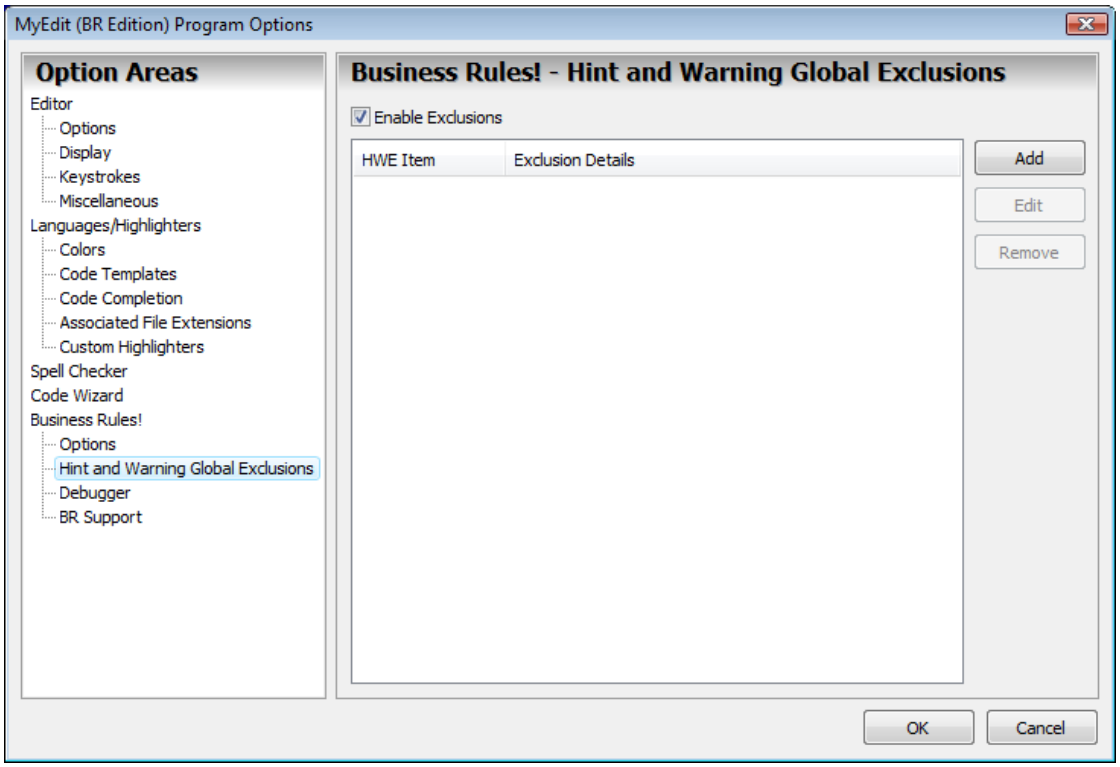
This page allows the programmer to disable certain messages from ever appearing. This is valid due to the fact that certain messages are only valid for certain versions of BR. Other messages such as the "Un-Dimmed variable reference" may not be valid if the program allows un-dimmed variables to be used.

There are two other features on this page to help BR programmers, **Prompt when saving a file with errors** is intended to alert programmers to errors that might cause BR to fail when returning from the EDIT command.

The other feature is the **Auto Correct Line Numbers**. Line numbers are five digits long but most programmers will enter only what's necessary to make a unique reference, so things like leading zeros are usually forgotten. MyEditBR will automatically provide the leading zeros when it detects an improperly formatted line number.

3.13 Business Rules! - Hint and Warning Global Exclusions

Fig. 20 - Editor Options: Business Rules! Hint and Warning Global Exclusions



The global exclusions options page provides another way of shutting off certain Hints and Warnings from being displayed while editing a BR source file. This method is not the same as unchecking an HWE in from the Business Rules! Options page, that is an all or nothing solution.

This allows you to direct the Hint or Warning to ignore very specific cases. For example, if you have a library that has a set of constants declared in it that aren't directly used in the library itself then you may want to include an exclusion for those specific constant declarations.

This can also be done with in a source file itself through the use of the exclusion line comment. An example of the command looks like this:

```
10 let x = 1: let y=2
20 !#exclude unused-value-assignment=x,y
```

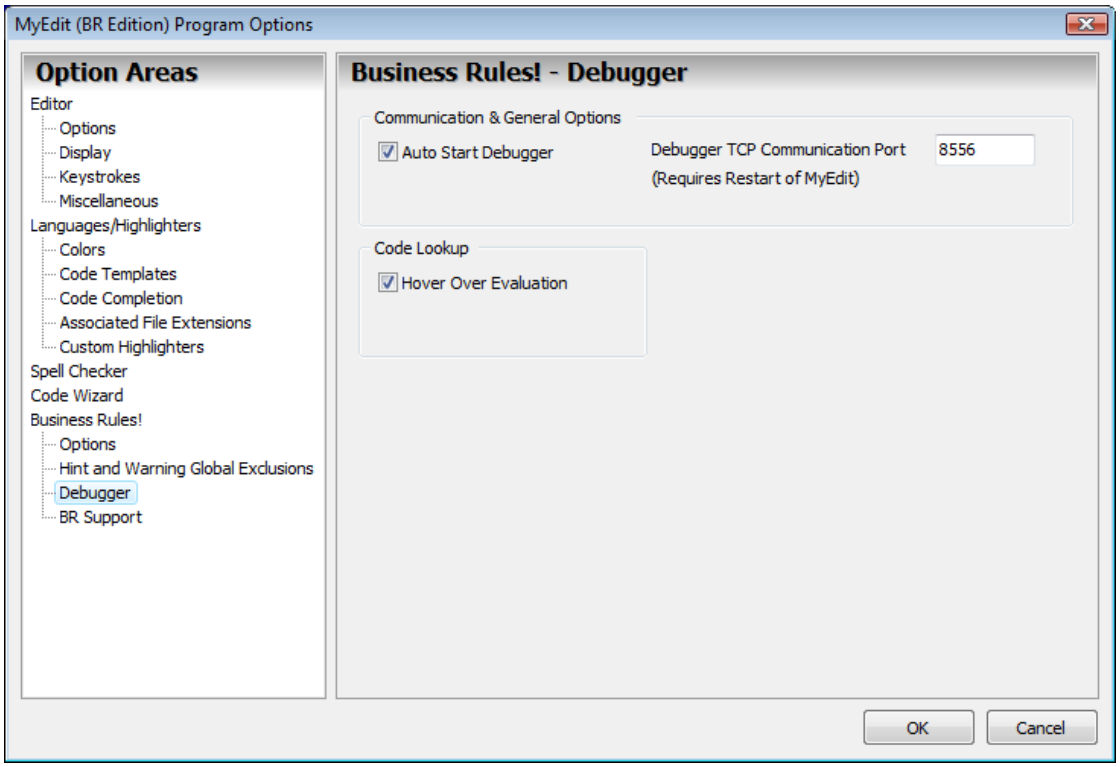
The above program demonstrates the in code solution to the unused value assignment warning for the variables x and y. This command only works if the **Enable Exclusions** check box has been checked. Since this is a per user configuration setting, each developer can decide whether or not to allow the program exclusion or have MyEditBR report it in the HWE tool window.

Here is a list of the valid exclusion commands:

unused-value-assignment	unused-variable
unused-label	unused-form
unused-program-function	unused-imported-function

3.14 Business Rules! Debugger

Fig. 21 – Editor Options: Business Rules! Debugger



Starting with version 4.18 of BR, MyEditBR has been able to act as a visual debugger for the BR language. It does this by connecting to BR through a dedicated TCP port and providing control over the connected BR session.

The available options are:

Auto Start Debugger – This means that when MyEditBR is started it automatically starts waiting for a BR session to connect to it for debugger purposes.

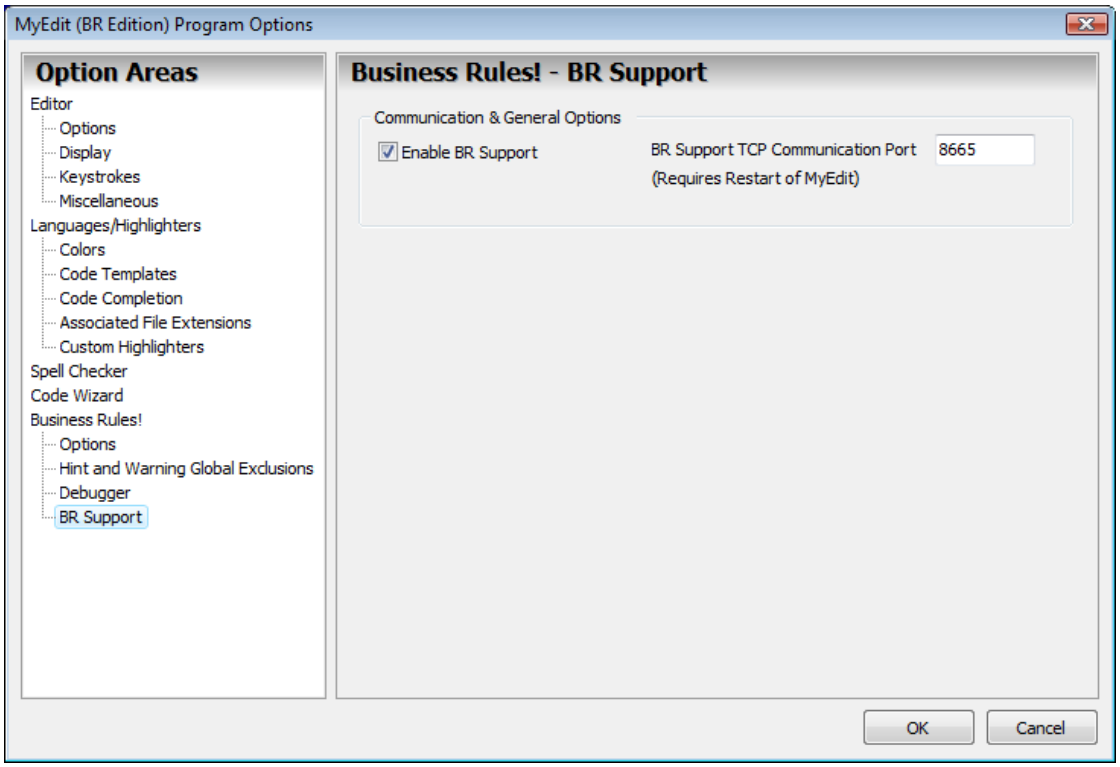
Debugger TCP Communication Port – by default this port number should be 8556 and should be left at that value as that is what BR will be looking to connect to.

Hover Over Evaluation – This is a feature of the debugger where when the program has been paused or has stopped running due to an error or a breakpoint you can place your mouse pointer over a variable and see what the value of the variable is.

See the Debugger section for more details on the debugger itself.

3.15 Business Rules! BR Support

Fig. 22 – Editor Options: Business Rules! BR Support



Also starting with version 4.18 of BR, MyEditBR has support built in to be able to use BR as a supporting program for features such as on the fly syntax checking, automatic conversion of source code to compiled BR programs and converting programs back to source.

If you disable this functionality MyEditBR will continue to run, just with out the listed features.

This program feature is a license supported feature. This means that you need to have purchased a valid support license from Mills Enterprise to get this functionality.

4 Standard Functionality

The standard functionality is generic and works for all programming languages and general text editing.

4.1 Code Completion & Parameter Tool Tips

This feature is available for any language source, as long as an INI file for that language exists, in the Completion Code folder where the MyEditBR application is located.

When the Code Completion is activated (Fig. 6) you will be presented with a list of items that either match up to what you've typed or everything if there is a blank at the cursor.

In Fig. 6 then cursor is after the F in Let FNSETCOLOR$ so the Code Completion window provides you with a complete list of all variables and methods that start with the letter F. The list is filtered as you type so if the Second letter after the F is an I then the only items left would be the FILE(N) and FILE$(N) functions.

The files in the Colors folder are automatically created the first time the Editor Options dialog is opened and closed, if they don't already exist.

There are three sections required in the Code Completion INI file those are:

- **Token Matches**
- **Built-ins**
- **Lookups**

The Code Completion feature and the following Parameter Tool Tips work off of the information in the Built-ins and Lookups sections.

The format of these two sections are identical. The Built-ins section is intended for built in language methods and the Lookups are for user defined methods. The format for the entries should look similar to the following:

```
[Built-ins]
Days=date,|,format$
ENV$=A$
PI=

[Lookups]
MyFunction1=|,isvisible:boolean
MyFunction2=x:integer,y:integer,data:string
```

The first entries list the built in functions of the language. Date has two parameters, date and format$, date is a required parameter while format$ is optional. The pipe (|) symbol tells MyEditBR to treat everything following it as an optional parameter. You can put any information you want in the parameters list as long as everything is comma separated.

The other part of the Code Completion INI file is the token matching section. This allows the user to define a set of rules that allows MyEditBR to highlight certain things in the editor for example (and).

There is a toggle button that can now be found on the Edit tool bar that can turn on or off the highlighting. You can also change the color of the highlights in the editor options dialog.

Here is an example for defining a Token Matching rule:

```
[tokenmatches]
token1=(,),symbol
token2=begin,end,key
```

The names (token1 and token2) are just to be unique and can be called anything you like. The starting item of the match, (or begin, must be seperated by a comma from the ending item of the match,) or end. The final part tells MyEditBR whether it is looking for a symbol (typically a character) or a keyword.

The tokens can be customized from the Editor options dialog on a per language basis.

With the Parameter Tool Tips optional parameters will appear in an italic font.

To force the Code Completion window open you can use **CTRL-Spacebar**.

To force the Parameter Lookup window open you can use **CTRL-Shift-Spacebar**

The options dialog now provides the ability to edit the Code Completion and Parameter Looks directly. As can be seen in Fig. 13

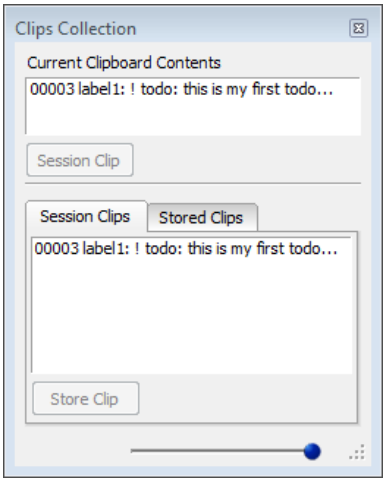
4.2 Clip Collection

This functionality works on the basis that people who use the clip board may want to hold multiple items on the clip board at any given time.

When ever something is placed on the clipboard (Copy or Cut) from with in MyEditBR it will automatically be added to the Session Clips section. This area can hold up to 25 different items with the oldest items being discarded as new items are added to the list.

The Current Clipboard Contents will always reflect the current contents of the clipboard of Windows. For anything that is added to the clipboard from outside of MyEditBR you can place it in the Session Clips section by clicking on the Session Clip button.

Fig. 23 – Clip Collection Tool Window



You can also store clips long term by adding it to the Stored Clips section. To do this you select the session clip and press the Store Clip button. You will then be prompted to provide a category name and description of the clip. The description should be meaning full since that will be how you know what the clip was.

To use either a session or a stored clip you double click on the item and it will appear on the clipboard, which can then be pasted into your document.

4.3 Selection Modes

There are three different types of block selection modes.

Normal – This allows the user to select a block that has offset starting and ending points. For example you may start selecting on char 2 line 3 and end on char 45 of line 12.

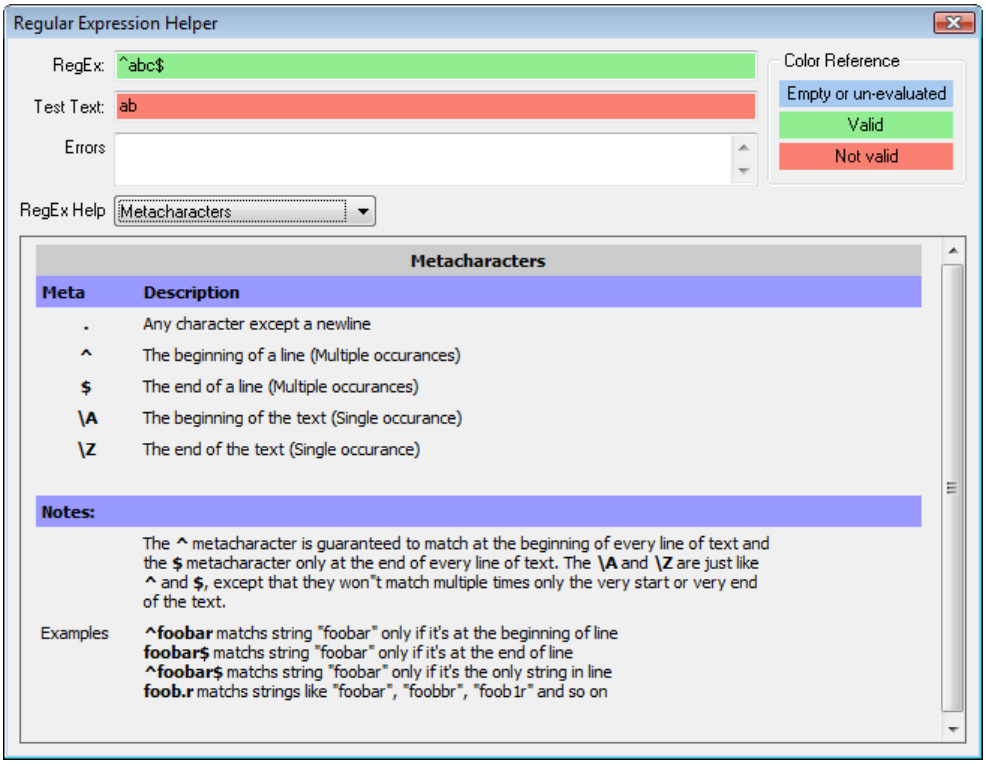
Line – This mode will always include the entire line no matter where the user starts or ends the selection.

Column – This mode allows for the user to select a custom block of text that forms a rectangle from the starting selection point to the ending selection point.

4.4 RegEx Helper

Regular Expressions (RegEx) are a powerful way to represent patterns of text using a type of symbolic language. The RegEx Helper is a tool that helps the user to develop regular expressions for use with in the search tool provided by MyEditBR.

Fig. 24 – RegEx Helper

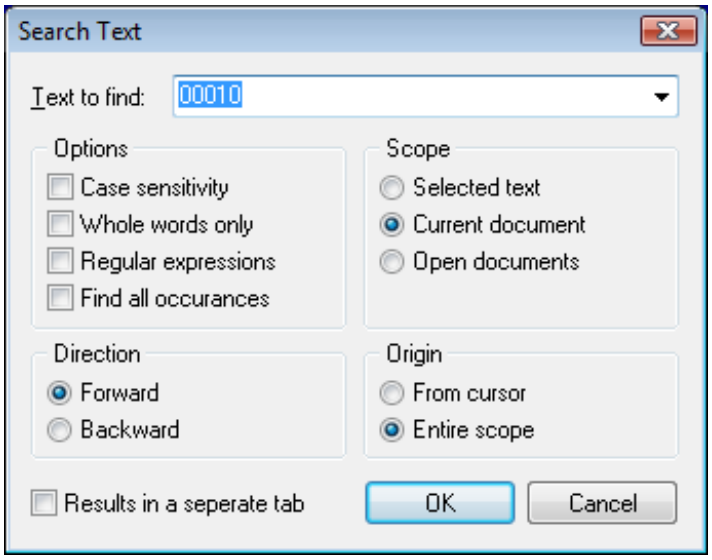


5 Searches

5.1 Standard Search

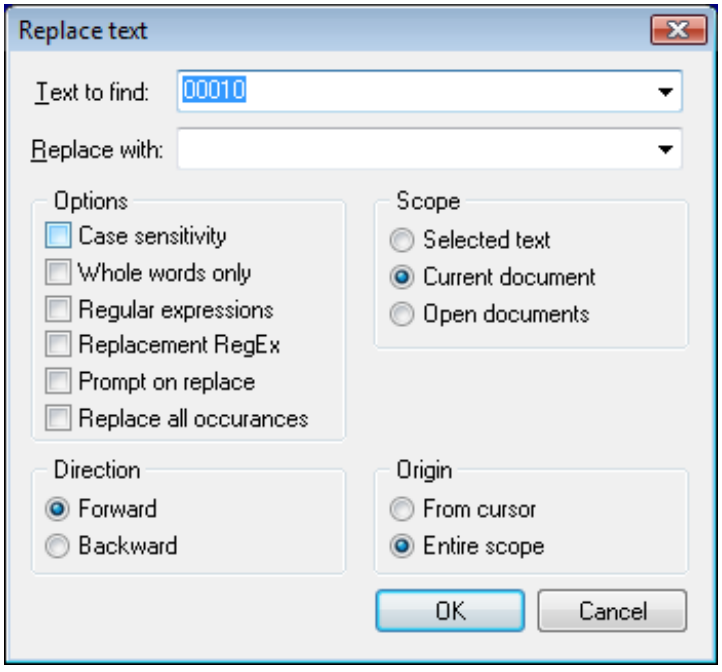
The standard search routines now include the ability to find all matching results in the current selection, current document or all open documents. When multiple search results are possible MyEditBR will automatically place the results in the improved search results section of the editor. Also the editor will allow at the same time the ability to place the results in a separate search results window.

Fig. 25 – Standard Search Dialog



5.2 Standard Replace

Fig. 26 - Replace Dialog



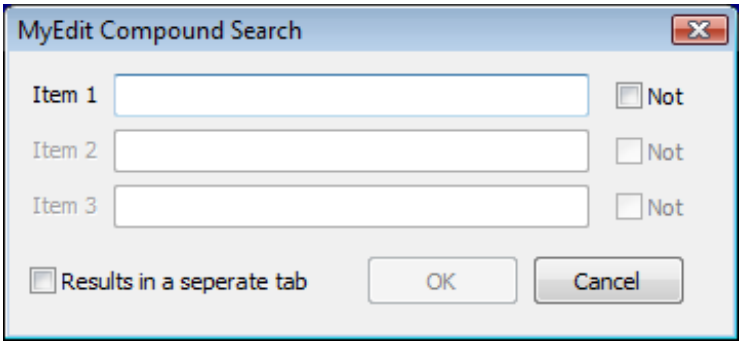
The Replace routine is very similar to the search and the dialog (fig. 26) looks very similar to the Search dialog window (fig. 25).

5.3 Compound Searches

The idea behind compound searches is that occasionally you want to be able to search on more than one condition at a time. This feature is very simple, it does not support Regular Expressions (RegEx), it does a simple item pattern match search.

The results of the Compound Search gets place in the Search Results tool window.

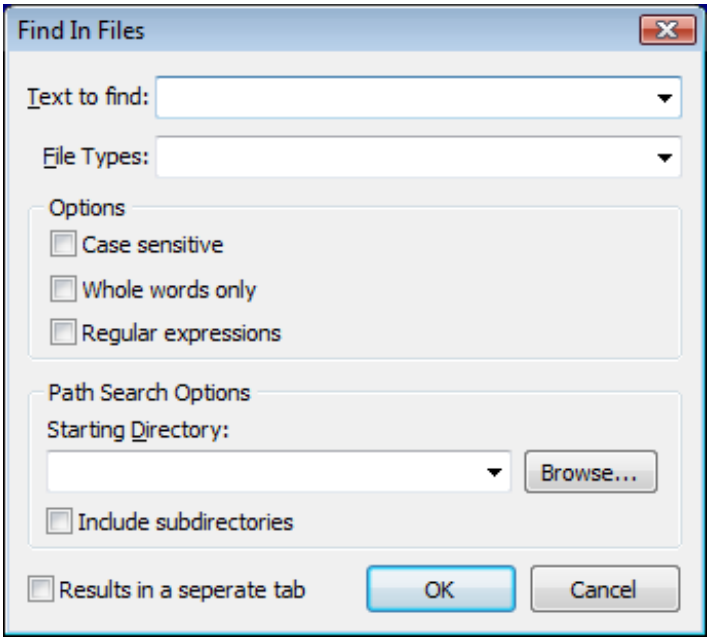
Fig. 27 – Compound Search Window



5.4 Find in Files

This routine now allows for the searching of a drive or computer network drive for a file that matches a pattern. The results will be placed in the search results tool window. The search can be conducted with Regular Expressions as well as a normal text search routine. It also has the ability to search the given folder recursively, meaning that any folder located in the specified folder will be searched as well.

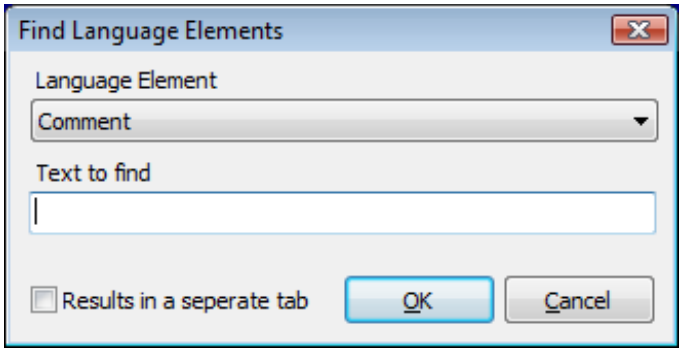
Fig. 28 – Find in Files Dialog.



5.5 Find Language Elements

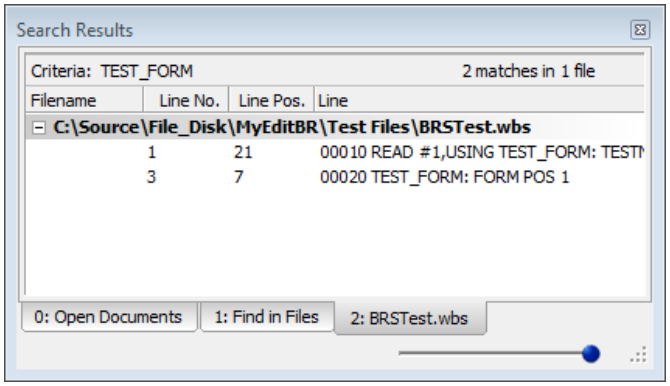
This is a new routine, not found in any other editor that we've ever seen, that works with the language parser to locate only specific items within the file and then to optionally search with in that item for a matching piece of text. For example, in any language that supports comments, you can now choose to search only comments and then find "Ryan" just with in those comments.

Fig. 29 - Find Language Elements Dialog



5.6 Search Results Window

Fig. 30 – Search Results Tool Window



In Fig. 30, you can see results of the search. All search routines that may return multiple results will now use this window to place the results in. You can see which document , the line and character position that the item was found at. You can double click on each of the items to have MyEditBR place the cursor on that line of code. If the document is not open when the item is double-clicked then MyEditBR will open the file in a new document tab and then proceed to locate the position in the file.

Each item will be placed under a collapsible node that displays the entire filename of where the match was located. This is the same for all items displayed in the search results window.

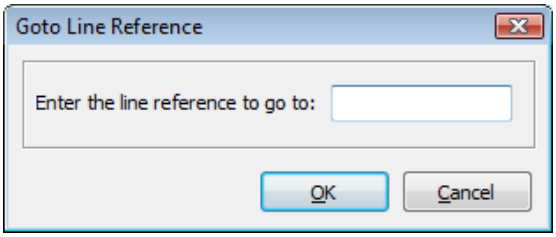
On the top right of the windows title bar you will notice a slider, this controls the windows transparency. If you float the window (undock it) and move the slider control to the left the window will become transparent allowing you to see the contents of the desktop under the search results window. You can use this if you have limited window space and you want to maximize the available area of the editor window.

5.7 Goto Line Reference

This dialog (see Fig. 31) allows you to jump to a specific text line number within the code. This feature also uses the Jump History.

See section 7.3 for BR specific functionality available for this dialog

Fig. 31 – Goto Line Reference Dialog



6 Tools

6.1 User Tools

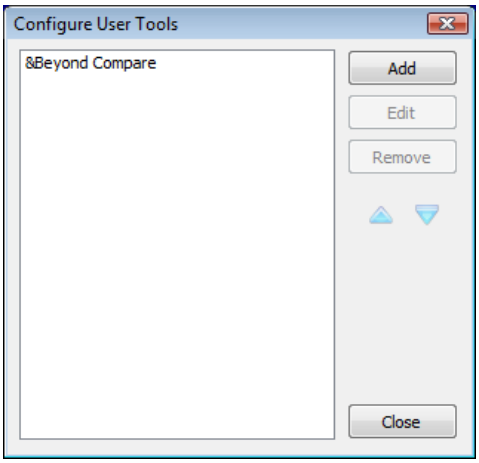
User Tools are menu item commands that can be configured to run application or Windows commands with the option of passing in the current files full path and filename.

The **Ctrl-Alt-#** shortcut keys are automatically assigned by MyEdit. You are only allowed a maximum of 10 User Tools and the shortcut keys can not be changed, but the items can be re-ordered to change which keystrokes they are assigned.

The command editing window (see Fig. 33) allows you to assign the name of the command as it will be seen on the menu.

(Note: To give the name an underlined character in the menu place an ampersand (&) character in front of the character to be underlined.)

Fig. 32 – User Tools Dialog



You can then choose the application you want to execute and place command line parameter for that application.

There are a number of replacement macros that are now available for different areas of the user tools command editing window. Each replacement macro will begin with **%%** and can be used to pull different pieces of information about the name and location of the file.

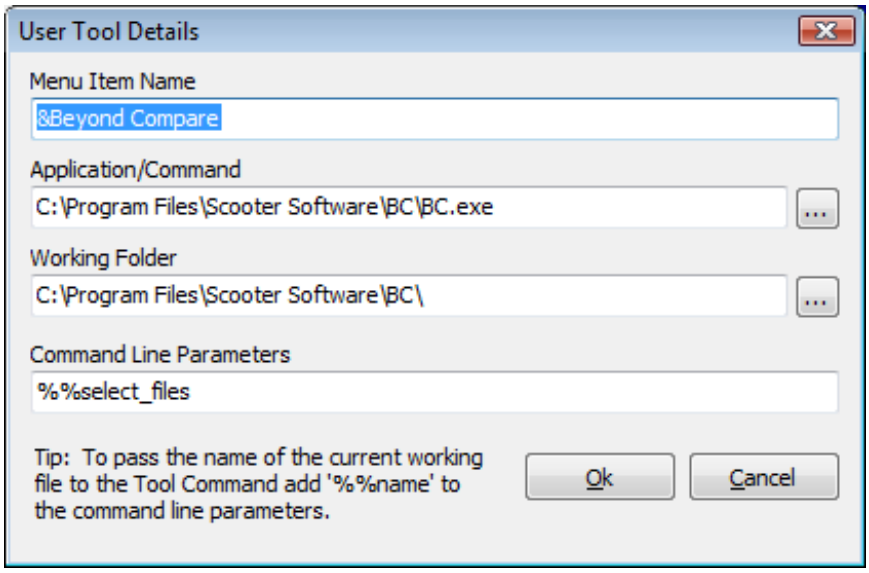
The following macros are only used in the Command Line Parameters:

%%name	provides the name and full path of the file
%%np_name	provides the name of the file (no path)
%%npne_name	provides the name of the file (no path, no extension)
%%folder	provides the path of the file (no filename)
%%select_files	prompt the user to select what files to work with.

The following macro is only available for the Working Folder:

%%folder	provides the path of the file (no filename)

Fig. 33 – Editing a User Tool Command



6.2 Editor Macros

This feature allows you to record large groups of scripted keystrokes and editor commands for later playback. So if you have a large number of repetitive keystrokes to do such as:

- Type an R
- Down Arrow
- Left Arrow
- Repeat

Then this is something that you should experiment with. MyEditBR supports the Saving,

Loading and Playback of recorded Macro scripts. Also note that similar types of search and replace operations can be done with the Search and Replace functionality. The Macros feature can be used for more than simple search and replace, with the correct usage editor commands such as Block Indent can be used with great success.

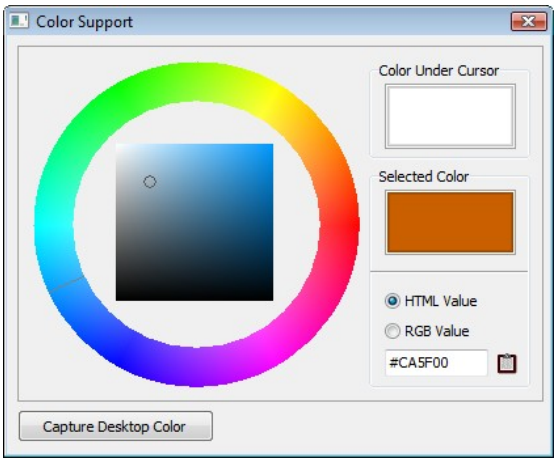
6.3 Plug-ins

MyEditBR has the ability to accept specially written plug-ins to add additional functionality. These features can be as simple as selecting a color from the screen to modifying text in the editor.

6.3.1 Color Support

The color support plug-in provides a dialog type window that allows the user to create a color and then put the color value into the editor has an HTML constant.

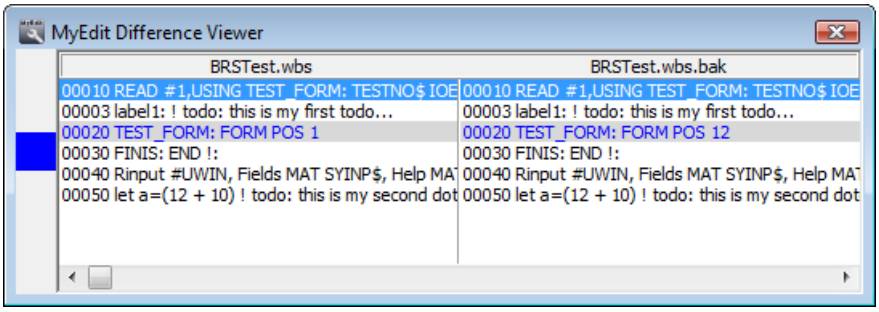
Fig. 34 – Color Support Plugin



6.3.2 Difference Viewer

The difference viewer is a simple way of seeing the difference between two copies of the same file.

Fig. 35 – Difference Viewer Plugin



In Fig. 35 you can see that a difference between the two files was found on line 00020.

There are three different kinds of differences that can be detected.

Blue means that a difference in the line was found.

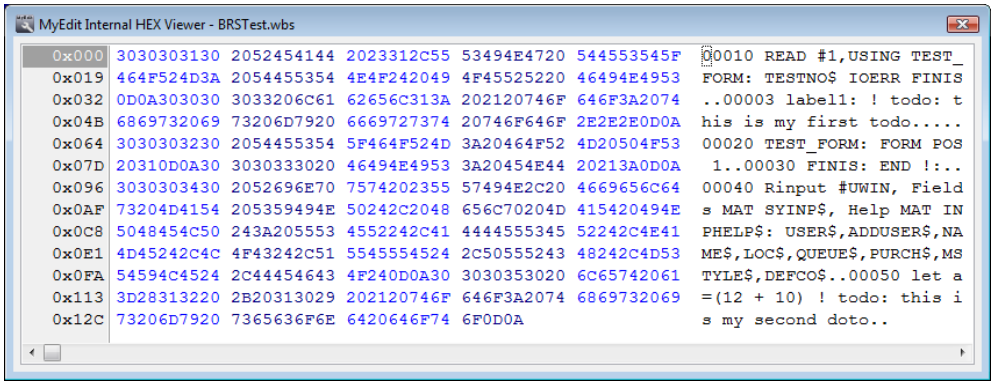
Green means that a block of text was inserted

Red means that a block of text was deleted.

6.3.3 Hexadecimal Viewer

A Hexadecimal, just hex, viewer is a tool commonly used by programmers to view binary files that can't normally be viewed in a standard text editor. It can also be useful in finding non-visible characters in a text document that are causing formatting problems.

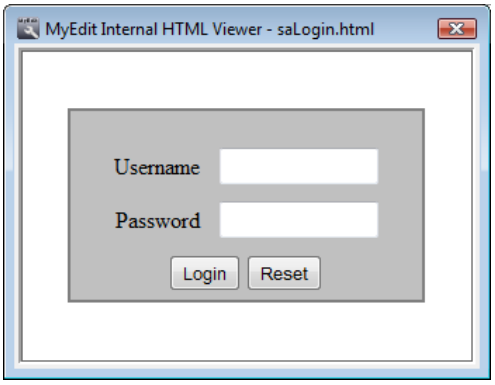
Fig. 36 – Hexadecimal Viewer



6.3.4 HTML Viewer

The HTML viewer plug-in provides a simple HTML rendering engine to quickly check how your HTML document should look like. It supports most CSS formatting options.

Fig. 37 – HTML Viewer Plugin

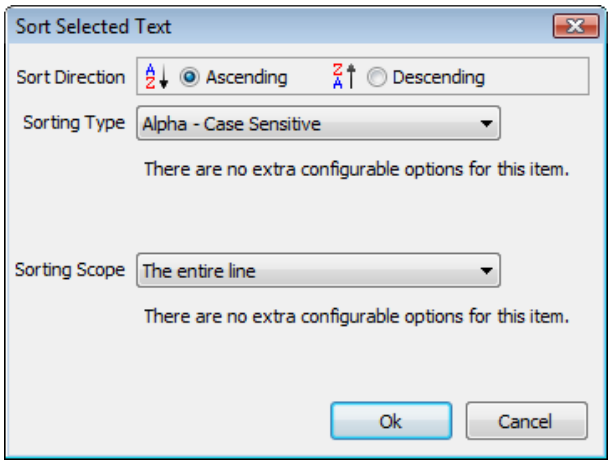


6.3.5 StrFX

The StrFX plug-in has many different features.

It has a sorting facility, Base64 encoder/decoder, a reverse selection command as well as an ASCII chart.

Fig. 38 – StrFX Sorting Dialog



The available sorting types are:

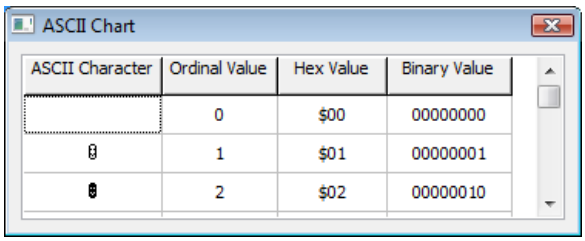
- Alpha – Case Sensitive
- Alpha – Case Insensitive
- Whole Numbers
- Real Numbers
- Formatted Date
- IP v4 Addresses

Scope can also be specified:

- The entire line
- The first word of the line
- The nth word of the line

This plug-in will generally place it's menu items under the **Format|Text Transforms** menu item.

Fig. 39 – StrFX Plugin ASCII Chart



6.3.6 Tidy HTML

The Tidy HTML plug-in uses an embedded copy of the open source LibTidy projects code. This library can be used to clean up non-standard HTML source as well as catching and cleaning up HTML coding errors.

6.4 Miscellaneous Items

This section will cover small little details of MyEditBR that probably wont be found any where else in this user guide.

- String Length Check – There are two ways to display the number of characters that have been selected. By default the tool tip display will be selected so that if you select a length of text in MyEditBR and then proceed to hold the mouse over the selection, the editor will display a tool tip showing you the number of characters selected. The other method is to show the selected length in the status bar.
- Current File Type – When a file is opened or created in MyEditBR the status bar will show the determined file type and what default language highlighter is being used for the current file.
- Editor Tab – The tab at the top right of the editor has a close button on it as well as a drop-down list of language highlighters. This drop-down list is the View File As section of the View Menu. This allows you to temporarily change the language highlighter used when viewing a known or unknown file type. This will not change the determined file type or access any of the language specific features built-in to MyEditBR, only opening the file and having MyEditBR determine the file type will do that for you.
- Tab Shifting – The file tabs that are displayed when a file is opened in MyEditBR can be rearranged in to a particular order by dragging and dropping the selected tab to the new position.
- Menu Shortcuts – This feature of MyEditBR allows you to customize the standard menu shortcuts to anything you want. For example by default the MyEditBR shortcut for Search and Replace is **Ctrl-R**, if you want to change that to **Ctrl-H** you just need to find it in the Menu Shortcuts editor and change the R to an H. (See Fig. 41 & 42)
- Active Line Highlight – This allows MyEditBR to change the color of the background that the cursor is on. If you choose the color, clNone, then MyEditBR will not display the highlight and will remove the toggle button from the tool bar. (See Fig. 11) This works on a per language highlighter basis.
- Automatic File Recovery – In the event that MyEditBR crashes during use it now has the ability to detect this and will attempt to find and recover the files from the last session. Any changes that had been made and saved to the internal temporary files prior to the crash can be recovered as long as those files exist. (See Fig. 43)
- File Associations – MyEditBR now has a File Associations dialog that allows it to configure windows to automatically open the file into the editor when the files icon is double clicked from the Windows Explorer. This same dialog can also be used to disable the associations. (See Fig. 40). File extensions that have been added to a language in the File Types page of the editor options dialog (See Fig. 15) will also be visible in this dialog window.

Fig. 40 – File Association Dialog

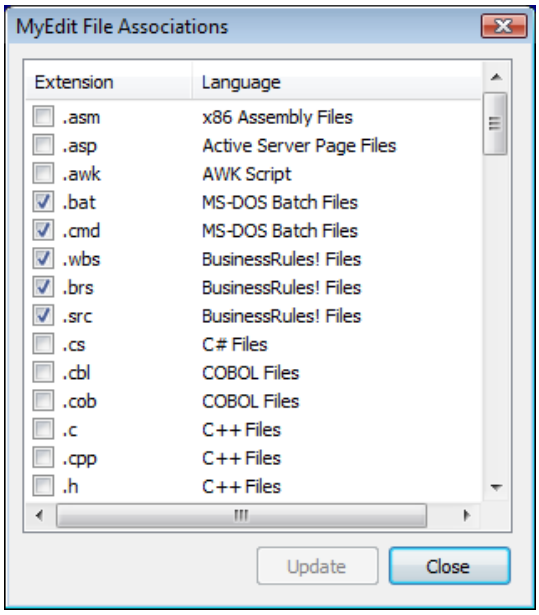


Fig. 41 – Tool bar Customization Dialog

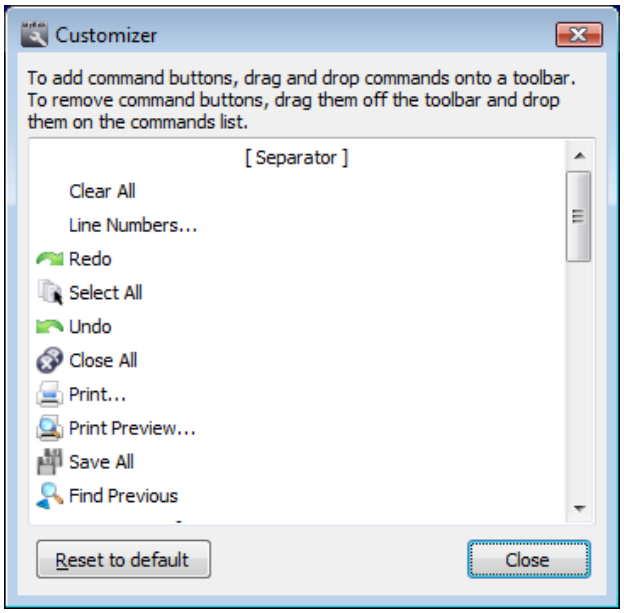


Fig. 42 - Shortcut Editor Dialog

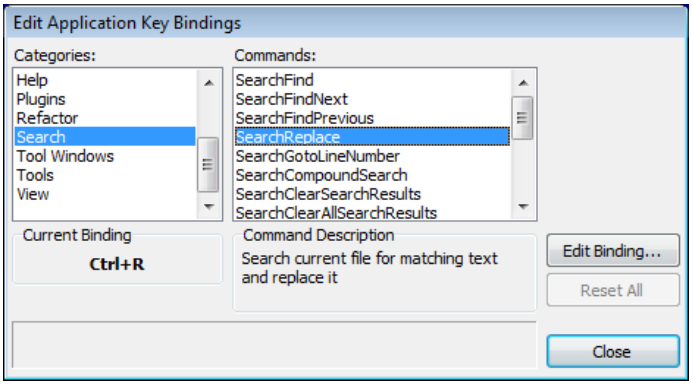
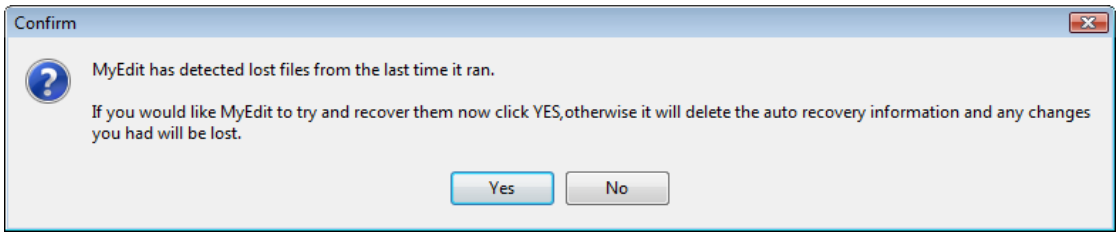


Fig. 43 - Auto-Recovery Message Dialog



7 Business Rules! Functionality

7.1 Hints, Warnings and Errors

There are, currently, thirty-three messages that can be displayed by MyEditBR.

Hints are items like unused variables, labels and forms statements.

Warnings are items that BR! is likely to allow you to type in but may not complain about. Such as line numbers out of order or duplicate line numbers.

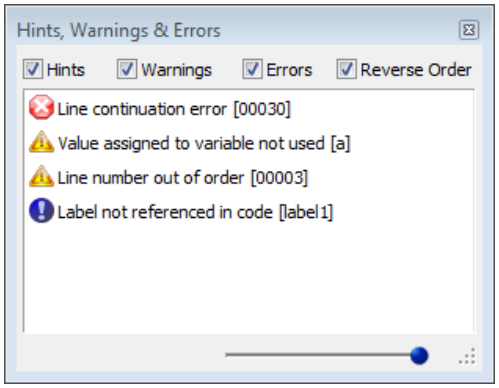
Errors are thing that will cause the BR! editor to not allow to be saved or run. These types of errors include orphaned lines and line continuation problems.

The Hints, Warnings and Errors tool window is able to take you to the location of point of the message when you double click on a specific item. If it can it will even take you to the exact point in the code that's causing the message to appear. This is extremely useful when trying to trouble shoot your code before returning it to the BR! Editor. (See Fig. 44)

There is an additional Status Bar panel that appears when editing a BR! source file, that will display the number of messages even if the BR! Drawer is closed. This new panel will appear at the far right on the Status Bar.

In the tool window there are four check boxes, three of which are tied to a group of messages. When checked the group of messages will be visible in the message box otherwise they will be hidden from view. The fourth check box allows for the HWE list to be reversed to display errors at the top of the list.

Fig. 44 - Hints, Warnings and Errors Tool Window



MyEditBR can now warn the user if they are trying to save a BR! Source file while is still has a detected Error in the source.

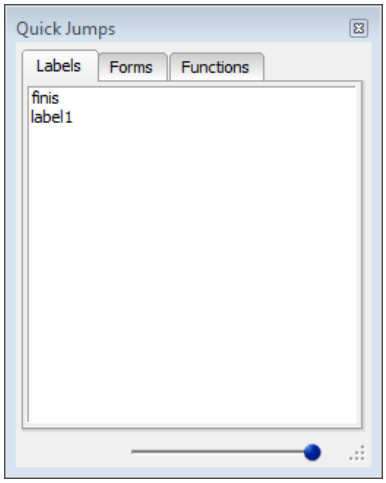
This option can be turned off if desired.

By default this option is enabled.

7.2 Quick Jumps

Quick Jumps are items that, when double clicked on, will reposition the editors cursor to that exact location in the current file. The three current quick jump items are Labels, Forms and Functions. These are read from the file and are updated as the user is editing the file. While these are similar to bookmarks, they are a little more flexible as they will appear and be available to the programmer with no extra work on the programmers part. (See Fig. 45)

Fig. 45 - Quick Jumps Tool Window



There is now a jump history that uses the Back Jump and Forward Jump buttons. They work similar to the back and forward buttons of most web browsers and allow for rapid movement across the history of the jumps. The history is built using Quick Jumps, the

7.3 Go To Line Reference

When working on a BR! File this feature is to allow the programmer to jump to a specific line number, label, form or function in the current file with out spending a lot of time looking for it. This feature also uses the Jump History. (See Fig. 31)

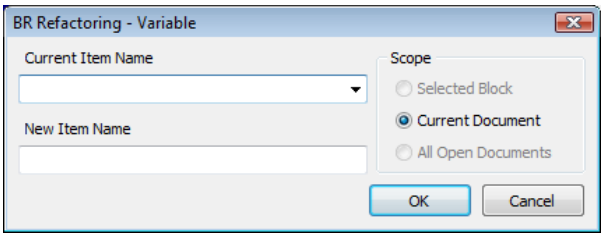
7.4 Auto Correct Line Numbers

To make some of the magic of MyEditBR work line numbers must be the full five digit width (ie 00001). With this option enabled if you type a line number such as 120 and continue on with the rest of the line, MyEditBR will correct the line number to read 00120 while you type. If it detects any other line numbers that are not properly formatted it will update them on the fly as well. (See Fig. 19)

7.5 Refactor

Starting in v3 MyEditBR has the ability to go through a BR source file and help the developer to clean up, a process sometimes known a refactoring, the source code. MyEditBR now supports a number of refactoring procedures. The screen presented in Fig. 46 is the variable refactoring dialog. All of the refactoring dialogs, with the exception of the Renum dialog, use the same basic design.

Fig. 46 – MyEditBR Variable Refactoring



7.5.1 Variable

When started the refactoring dialog will appear and will have a drop down list populated with what it believes are the appropriate information. In this case it will present you with a list of all detected variables. You choose one and then provide the new desired name.

The scope is very important as this will determine the area affected by the refactoring procedures.

7.5.2 Label

Labels work the same way as variables do. The drop down list contains labels.

7.5.3 Named Form

Named forms work the same way as variables do. The drop down list contains form names.

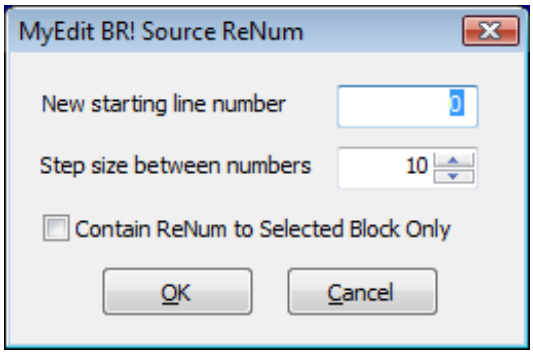
7.5.4 Line Numbers

BR! has a feature called renum that, when used properly, can renumber a range of program lines and update all the references of the changed lines.

To make this feature work you have to select a block of source you wish to renumber. Execute the ReNum command and you'll be presented with the dialog found in Fig. 47.

So for example, if you wanted to renumber the entire program, do a Select All and then do the ReNum.

Fig. 47 – MyEditBR Source ReNum Dialog



The check box on the dialog allows for the ReNum to only affect the selected block of text and not the rest of the program. This is useful when pasting in code from another BR program. When this is checked if there are line references that can not be resolved because they haven't been found with in the selected block you will receive a message telling you how many unresolved line references MyEditBR found.

8 BR Debugger

Starting with Business Rules! version 4.18 MyEditBR has been developed to help provide a better development experience by integrating with BR to provide end users with a visual debugger. This debugger works externally to BR by controlling the debugging session.

8.1 Activating / Deactivating

MyEditBR is setup not to activate the Debugger engine. Visually you can tell the state of the debugger engine by finding the debugger icon on the tool bars.

Fig. 48 – Debugger Engine Icon

When the tool bar button with the icon from Fig. 48, is pushed down then the debugger engine is active and waiting for a debug session to begin.

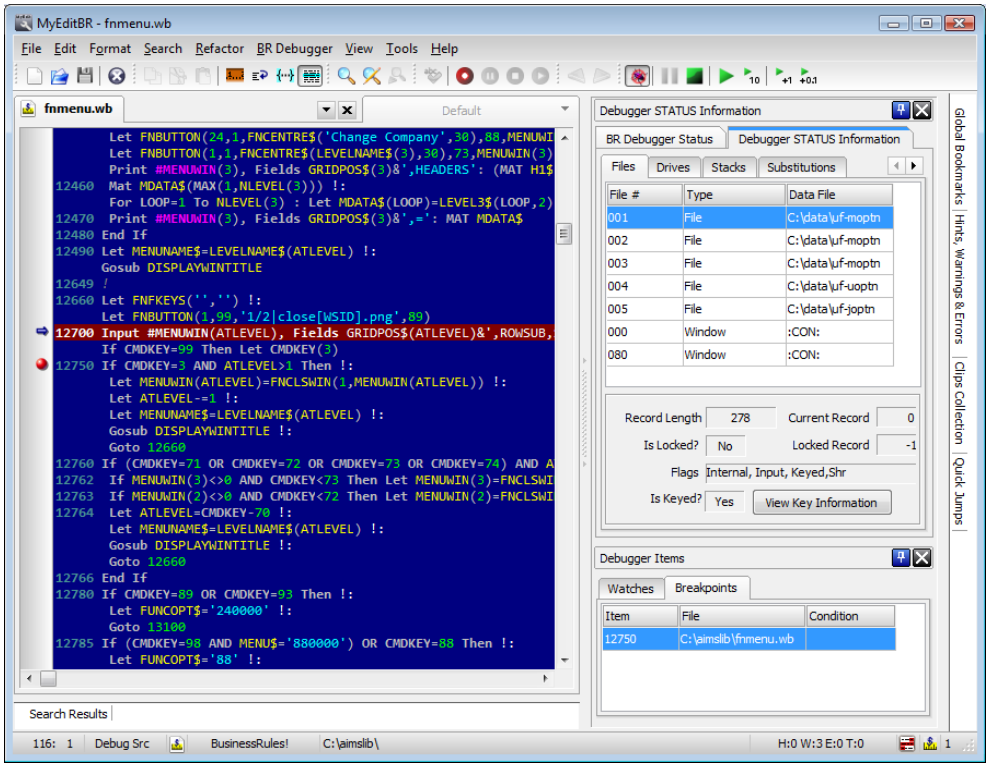
If you are in an active debug session and you press this same button then MyEditBR will try and disconnect the session. Once the session has been disconnected then pushing the button a second time will shut the debugger engine off.

8.2 Starting A Debug Session

With the debugger engine active, with no active debug session, MyEditBR is waiting for a connection command from the BR program. To get BR to initiate a debug session you must issue the following console command:

```
debug connect [ip address:port]
```

Fig. 49 – Example MyEditBR Debug Session



By default the IP address will be 127.0.0.1, or the local machine, and the port number will be 8556. Normally you wont have to change either so just use **debug connect**.

Once connected you will lose the BR console window and MyEditBR will look similar to Fig. 49.

The blue arrow in the gutter bar and the red highlighted line indicates the current program line if the current BR state is ATTN or ERROR. If BR is in the READY state then you will not see either the blue arrow or the red line.

You will see the Green VCR type command buttons on the debugger tool bar light up giving you the ability to do certain things line RUN, Step (Into and Over) and STOP for example.

8.3 Breakpoints

Breakpoints are a way of controlling the flow of a program. A breakpoint can be placed to cause BR to stop program execution and enter the ATTN state. This then allows you to do things like evaluate variables, as an example. Starting with v4.18 of BR, breakpoints can now be set outside of a running program and for programs or libraries that haven't been loaded yet. This means that you can open a library file and set a breakpoint in the library file and anytime that breakpoint is encountered during a program BR will stop program execution and enter the ATTN state, thus giving control back to MyEditBR. See section 8.x for information on how to open program files during a debug session.

There is a support licensed feature called a conditional breakpoint.

A conditional breakpoint is a standard breakpoint that can have a logic rule applied to it so that the debugger will ignore the breakpoint until the logic rule evaluates to true.

A example of this would be to set a breakpoint on the first line of a LOOP clause and then set the condition so that it's only to break if the loop variable equals 5. This allows the program to run until the breakpoint is hit and then only stop program execution if the loop variable is equal to 5.

Breakpoints can be viewed in two different ways. The first is through the Debugger Items tool window on the breakpoints tab. This is a list of all breakpoints including the line, file and any conditional information about the breakpoint.

The other way is the little orange dot in the gutter of the program file. See Fig. 49 for an example of both.

8.4 Watches

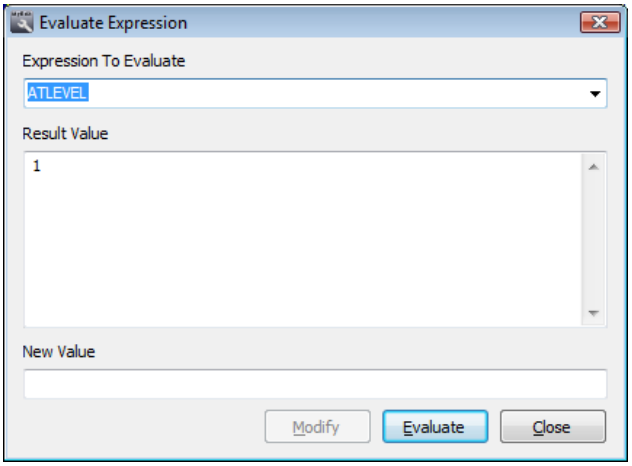
A watch is a way of watching a variables value after every executed step of the debugger. For example, if you set a watch on a loop variable and have a breakpoint set on every iteration of the loop, the watch would show the correct current value every time MyEditBR regained control over the debug session. It would work the same way after every executed step command.

Watches are displayed in the watch tab of the Debugger Items tool window.

8.5 Evaluate / Modify

The evaluate / modify window is the one dialog you should become very familiar with. This window allows you to see the value of a specific variable, statement or operation and possibly modify that value, depending on what you were evaluating.

Fig. 50 – Evaluate/Modify Window



As can be seen in Fig. 50 the window allows for variables to be evaluated and have new values assigned to them.

Please Note: Do not evaluate a function or something that permanently sets a value. This does actually execute code!!!

8.6 Stepping (Into/Over)

Control of a debug session is done through means of running the program. There are many ways of controlling the program. The most useful in terms of the debugger are the step commands. These work the same way as the traditional BR step commands, except that everything is designed to allow you to see what code your currently executing and visually follow along.

There are 4 stepping commands:

Step Into Line – Processes up to the next line number encountered.

Step Into Clause – Processes only the next clause

Step Over Line – Processes the next line plus all routines called by it

Step Over Clause - Processes the next clause plus all routines called by it

These commands are probably the 4 most useful commands in controlling the debug session.

8.7 Retry

Standard BR command. Executes exactly as if you had typed it on the command line.

8.8 GO to current line

Standard BR command. Executes exactly as if you had typed GO xxx on the command line. (Where xxx is a program line number)

8.9 Console Commands

For those who wish to have more control MyEditBR provides you with the ability to execute console commands directly. This should not be done lightly.

8.10 STATUS Information

MyEditBR has taken most if not all of the standard STATUS commands and has placed them into a easy to use format. If Fig. 49 you can see the, docked, STATUS Information tool window.

This window provides STATUS information for the following commands:

- Files
- Drives
- Stacks
- Substitutions
- Config
- Environment
- Attributes

Each command is updated after every debugger command has been executed. Most of the pages have a refresh button on them in the event that the information doesn't appear to have refreshed properly.

9 Regular Expressions

Regular expressions can be a very powerful tool when used correctly. Now that may seem like a very strange opening sentence for this chapter, but for as powerful as Regular Expressions (RegEx) are they can cause all sorts of unexpected problems when they are used incorrectly. This section will attempt to provide you with a very basic guide to writing regular expressions. As there are entire books dedicated to this matter, I'll leave it up to you to learn more about the wonderful world regular expressions.

MyEditBR supports the use of RegEx statements in both the search and replace functions. Using them for searching is probably the safest way to experiment with writing regular expressions but once you've mastered them using RegEx patterns for replacing portions of text in your files can be the quickest way to do certain tasks.

A regular expression is composed of a sequence of sub-expressions, each of the form in the operators table below. The entire expression may be preceded by ^ to indicate that the expression is only matched at the start of a line, or ended by $ to indicate that the expression can only exist at the end of a line.

9.1 Regular Expression Operators

Regular Expression Meta Characters

****	Meta characters escape	**\w**	A single character, one of [a-zA-Z0-9_]
\t	The tab character	**\W**	Any single character not matching \w
\n	The newline character	**\d**	A single character [0-9]
\r	The return character	**\D**	A single character not matching \d
\f	The formfeed character	**\s**	A whitespace character [\t\r\n\f\b\ ]
\b	The backspace character	**\S**	A single character not matching \s
\xNN	The hexadecimal character NN	**\o000**	The octal character ooo
.	Any character except a newline	*****	0 or more of the previous character
?	0 or 1 of the previous character	**+**	1 or more of the previous character
^	The beginning of a line	**$**	The end of a line
( )	Compound statement markers	**{ }**	Occurrences value markers
[ ]	Single character set markers	**[^]**	Single character exclusion set marker
|	Logical OR	**" "**	Exact pattern markers

9.2 Simple Examples

Expression	Matches	Does not match
"this" | "that"	this that	This That
\d{2}\.\d{2}	22.45 03.22	2.4 0.1
[a-zA-Z_]\w*	Identifier	2Identifiers
\(*[x01-\x7F]+*\)	(* a c-style comment *)	(not a comment *)
th[ae]n	then THEN Then than THAN Than	thin THIN Thin thaen THAEN Thaen

9.3 Complex Examples

1. Find Internet References

Expression:

("http://"|"mailto:"|"ftp://")[^ \n\r\"\<\\]+

Meaning:

Find all occurrences of text that start with 'http://', 'mailto:' or 'ftp://' and are followed by at least one character that is not one of a space (\s), a newline(\n), a carriage return(\r), a quote(\"), a bracket (\<), or a slash (\\)

2. Find all compiler directives in an Object Pascal program

Expression:

"{$"[a-zA-Z_]+[\s\t\r\n]*[^\}]*"}"

Meaning:

Find all occurrences of text that starts with a curly brace and a dollar ("{$") and is followed by at least one letter or underscore([a-zA-Z_]+) optionally followed by some whitespace ([\s\t\r\n]*) optionally followed by any number of anything except a close curly brace([^\}]*) and terminated by a close curly brace "}"