

Reference Manual

Volume III Interfaces Guide

Version 6.4.2

CLIPS Interfaces Guide
Version 6.4.2 January 16th 2025

CONTENTS

License Information.....	vi
Preface.....	vii
Section 1: Introduction.....	1
Section 2: CLIPS .NET IDE.....	2
2.1 The File Menu.....	3
2.1.1 Quit	3
2.2 The Edit Menu	3
2.2.1 Cut (Ctrl+X).....	3
2.2.2 Copy (Ctrl+C).....	3
2.2.3 Paste (Ctrl+V)	3
2.2.4 Set Dialog Font... (Ctrl+V)	4
2.2.5 Set Browser Font... (Ctrl+V)	4
2.3 The Environment Menu	4
2.3.1 Clear	4
2.3.2 Load Constructs... (Ctrl+L).....	4
2.3.3 Load Batch... (Ctrl+Shift+L)	4
2.3.4 Set Directory...	4
2.3.5 Reset (Ctrl+R).....	5
2.3.6 Run (Ctrl+Shift+R).....	5
2.3.7 Halt Rules (Ctrl+H)	5
2.3.8 Halt Execution (Ctrl+Shift+H)	5
2.3.9 Clear Scrollback.....	5
2.5 The Debug Menu	5
2.5.1 Watch Submenu	5
2.5.2 Agenda Browser.....	5
2.5.3 Fact Browser	6
2.5.4 Instance Browser.....	7
2.6 The Help Menu	8
2.6.1 CLIPS Home Page	8
2.6.2 Online Documentation	8
2.6.3 Online Examples	8
2.6.4 CLIPS Expert System Group	8
2.6.5 SourceForge Forums	9
2.6.6 Stack Overflow Q&A	9
2.6.7 About CLIPS IDE	9
2.8 Building the Windows Executables	9

2.8.1 Building CLIPSIDE Using Microsoft Visual Studio Community 2022.....	9
2.8.2 Building CLIPSDOS Using Microsoft Visual Studio Community 2022	9
Section 3: CLIPS macOS IDE	11
3.1 The CLIPS IDE Menu	13
3.1.1 About CLIPS.....	13
3.1.2 Preferences... (⌘,)	14
3.1.3 Quit CLIPS IDE (⌘Q)	15
3.2 The File Menu.....	15
3.2.1 New (⌘N)	15
3.2.2 Open... (⌘O)	16
3.2.3 Open Recent.....	16
3.2.4 Close (⌘W).....	16
3.2.5 Save (⌘S).....	16
3.2.6 Save As... (⇧⌘S)	16
3.2.7 Revert.....	16
3.2.8 Page Setup... (⇧⌘P).....	16
3.2.9 Print... (⌘P).....	17
3.3 The Edit Menu	17
3.3.1 Undo (⌘Z).....	17
3.3.2 Redo (⇧⌘Z).....	17
3.3.3 Cut (⌘X)	17
3.3.4 Copy (⌘C).....	17
3.3.5 Paste (⌘V).....	17
3.3.6 Delete	18
3.3.7 Select All (⌘A).....	18
3.3.8 Find Submenu	18
3.4 The Text Menu.....	19
3.4.1 Load Selection (⌘K).....	19
3.4.2 Batch Selection (⇧⌘K).....	19
3.4.3 Load Buffer.....	19
3.4.4 Balance (⌘B)	20
3.4.5 Comment.....	20
3.4.6 Uncomment.....	20
3.5 The Environment Menu	20
3.5.1 Clear.....	20
3.5.2 Load Constructs... (⌘L)	20
3.5.3 Load Batch... (⇧⌘L).....	20
3.5.4 Set Directory...	21
3.5.5 Reset (⌘R)	21
3.5.6 Run (⇧⌘R).....	21
3.5.7 Halt Rules (⌘.).....	21
3.5.8 Halt Execution (⇧⌘.).....	21

3.5.9 Clear Scrollback.....	21
3.6 The Debug Menu	21
3.6.1 Watch Submenu	21
3.6.2 Agenda Browser.....	22
3.6.3 Fact Browser	22
3.6.4 Instance Browser.....	24
3.6.5 Construct Inspector	25
3.7 The Window Menu	25
3.8 The Help Menu	25
3.8.1 CLIPS Home Page	25
3.8.2 Online Documentation	25
3.8.3 Online Examples.....	25
3.8.4 CLIPS Expert System Group	26
3.8.5 SourceForge Forums.....	26
3.8.6 Stack Overflow Q&A	26
3.9 Creating the macOS Executables.....	26
3.9.1 Building the CLIPS IDE Using Xcode 16.2	26
Section 4: CLIPS Swing IDE	27
4.2 The File Menu.....	28
4.2.1 New (^-N).....	28
4.2.2 Open... (^-O)	28
4.2.3 Save (^-S)	29
4.2.4 Save As... (^+⇧-S)	29
4.2.5 Page Setup.....	29
4.2.6 Print.....	29
4.2.7 Quit CLIPS IDE (^-Q)	29
4.3 The Edit Menu	29
4.3.1 Undo (^-Z).....	29
4.3.2 Redo (^+⇧-Z).....	29
4.3.3 Cut (^-X)	30
4.3.4 Copy (^-C).....	30
4.3.5 Paste (^-V).....	30
4.3.6 Set Font... ..	30
4.4 The Text Menu.....	30
4.4.1 Load Selection (^-K).....	30
4.4.2 Batch Selection (^+⇧-K).....	30
4.4.3 Load Buffer.....	31
4.4.4 Balance (^-B)	31
4.4.5 Comment.....	31
4.4.6 Uncomment.....	31
4.5 The Environment Menu	31
4.5.1 Clear.....	31

4.5.2 Load Constructs... (^-L)	31
4.5.3 Load Batch... (^+⇧-L).....	32
4.5.4 Set Directory...	32
4.5.5 Reset (^-R)	32
4.5.6 Run (^+⇧-R)	32
4.5.7 Halt Rules (^-.).....	32
4.5.8 Halt Execution (^+⇧-.).....	32
4.5.9 Clear Scrollback.....	32
4.6 The Debug Menu	33
4.6.1 Watch Submenu	33
4.6.2 Agenda Browser.....	33
4.6.3 Fact Browser	33
4.6.4 Instance Browser.....	34
4.6.5 Construct Inspector	36
4.7 The Window Menu	36
4.8 The Help Menu	36
4.8.1 CLIPS Home Page	36
4.8.2 Online Documentation.....	36
4.8.3 Online Examples.....	36
4.8.4 CLIPS Expert System Group.....	36
4.8.5 SourceForge Forums.....	37
4.8.6 Stack Overflow Q&A	37
4.8.7 About CLIPS IDE.....	37
4.9 Creating the Swing IDE Executable	37
Section 5: CLIPS DLL Interface	38
5.1 Installing the Source Code	38
5.2 Building the CLIPS Libraries	38
5.2.1 Building the Projects Using Microsoft Visual Studio Community 2022	39
5.3 Running the Library Examples	39
5.3.1 Building the Examples Using Microsoft Visual Studio Community 2022.....	39
Section 6: CLIPS .NET Interface	41
6.1 Installing the Source Code	41
6.2 Building the .NET Library and Example Executables.....	41
6.2.1 Building the Projects Using Microsoft Visual Studio Community 2022	42
6.3 Running the .NET Demo Programs	42
6.3.1 Wine Demo	42
6.3.2 Auto Demo.....	43
6.3.3 Animal Demo.....	43
6.3.4 Router Demo.....	44
6.4 CLIPS .NET Classes	44
6.4.1 The Environment Class.....	44

6.5.2 The PrimitiveValue Class and Subclasses	50
6.5.3 The CLIPSEException and CLIPSLoadException Classes	54
6.5.4 The Router Class.....	54
6.5.5 The UserFunction Class.....	56
6.5.6 Examples.....	57
Section 7: CLIPS Java Native Interface	62
7.1 CLIPSJNI Directory Structure.....	62
7.2 Issuing Commands from the Terminal	63
7.3 Running CLIPSJNI in Command Line Mode.....	64
7.4 Running the Swing Demo Programs.....	64
7.4.1 Sudoku Demo.....	65
7.4.2 Wine Demo	65
7.4.3 Auto Demo.....	66
7.4.4 Animal Demo.....	67
7.4.5 Router Demo.....	67
7.5 Creating the CLIPSJNI JAR File.....	68
7.6 Creating the CLIPSJNI Native Library.....	69
7.6.1 Creating the Native Library on macOS	69
7.6.2 Creating the Native Library on Windows 11	70
7.6.3 Creating the Native Library On Linux.....	70
7.7 Recompiling the Swing Demo Programs.....	70
7.7.1 Recompiling the Swing Demo Programs on macOS.....	70
7.7.2 Recompiling the Swing Demo Programs on Windows	71
7.7.3 Recompiling the Swing Demo Programs on Linux	71
7.8 Internationalizing the Swing Demo Programs.....	71
7.9 CLIPSJNI Classes.....	73
7.9.1 The Environment Class.....	73
7.9.2 The PrimitiveValue Class and Subclasses	79
7.9.3 The CLIPSEException and CLIPSLoadException Classes	82
7.9.4 The Router Interface	83
7.9.5 The UserFunction Interface	85
7.9.6 Examples.....	85
Appendix A: Support Information.....	91
A.1 Questions and Information.....	91
A.2 Documentation.....	91
A.3 CLIPS Source Code and Executables	91
Appendix B: Update Release Notes	92
Index.....	93

License Information

MIT No Attribution

Copyright 2023 Secret Society Software, LLC

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Preface

About CLIPS

Developed at NASA's Johnson Space Center from 1985 to 1996, the 'C' Language Integrated Production System (CLIPS) is a rule-based programming language useful for creating expert systems and other programs where a heuristic solution is easier to implement and maintain than an algorithmic solution. Written in C for portability, CLIPS can be installed and used on a wide variety of platforms. Since 1996, CLIPS has been available as public domain software.

CLIPS Version 6.4

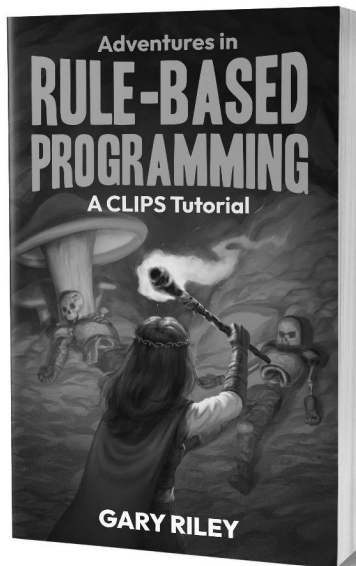
Version 6.4 of CLIPS includes three major enhancements: a redesigned C Application Programming Interface; wrapper classes and example programs for .NET and Java; and Integrated Development Environments with Unicode support for Windows and Java. For a detailed listing of differences between releases of CLIPS, refer to appendix B of the *Basic Programming Guide* and appendix B of the *Advanced Programming Guide*.

CLIPS Documentation

Two documents are provided with CLIPS.

- The *CLIPS Reference Manual* which is split into several volumes:
 - *Volume I - The Basic Programming Guide* provides information on the CLIPS programming language.
 - *Volume II - The Advanced Programming Guide* provides information on compiling CLIPS and use of the C Application Programming Interfaces.
 - *Volume III - The Interfaces Guide* provides information on the CLIPS Integrated Development Environments, wrapper classes, and example programs.
- The *CLIPS User's Guide* provides an introduction to CLIPS and rule-based programming.

Other Documentation



Adventures in Rule-Based Programming is a fun introduction to writing applications using CLIPS. In this tutorial you'll learn the basic concepts of rule-based programming, where rules are used to specify the logic of what must be accomplished, but an inference engine determines when rules are applied. You'll incrementally create a fully functional text adventure game, and in the process, learn how to write, organize, debug, test, and deploy CLIPS code. Visit clipsrules.net/airbp for more information.

Section 1:

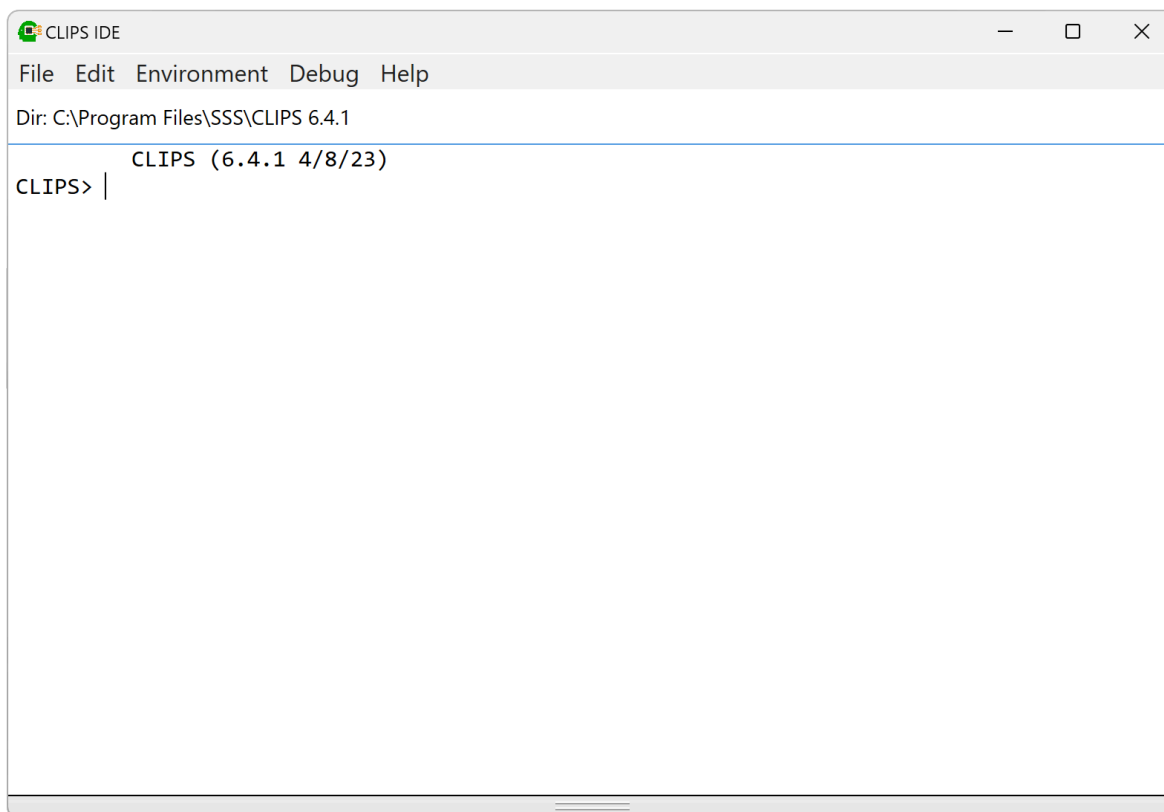
Introduction

This manual is the *Interfaces Guide* for CLIPS. It is intended for users interested in using the Integrated Development Environments (IDEs) for Windows, macOS, and Java; the wrapper classes for .NET and Java; and the example programs for .NET and Java. Section 2 of this manual describes the Windows IDE for CLIPS. Section 3 describes the macOS IDE for CLIPS. Section 4 describes the Java Swing IDE for CLIPS. Section 5 describes the CLIPS DLL Interface. Section 6 describes the CLIPS .NET Interface. Section 7 describes the CLIPS Java Native Interface.

Section 2:

CLIPS .NET IDE

This section provides a brief summary of the CLIPS 6.4 .NET Integrated Development Environment (IDE). The IDE provides a dialog pane that allows commands to be entered in a manner similar to the standard CLIPS command line interface. Any CLIPS I/O to standard input or standard output is directed to this dialog pane. In addition, the IDE also provides a browser pane for examining the current state of the CLIPS environment. When launched, the IDE displays a window containing a menu bar, a status bar, and a dialog pane:



The status bar is displayed beneath the menu bar. On the left side of the status bar is the current working directory. The splitter along the bottom edge of the dialog pane can be dragged to reveal or hide any browser tabs that are open in the browser pane.

Inline editing is supported in the dialog pane. The left and right arrow keys can be used to move the caret backwards and forwards through the current command. Pressing the delete key will delete the character to the left of the caret. Insertion of other characters or pasted text occurs at the caret. The esc key moves the caret to the end of the current command. The caret must be at the end of the current command in order for pressing the return key to execute the command.

A command history is also supported for the dialog pane. The up and down arrows allow you to cycle through the command history. The up arrow restores the previous command and the down arrow restores the next command. Holding the shift key down when the up or down arrow is pressed takes you respectively to the beginning or end of the command history.

From the CLIPS command prompt, the command **clear-window** (which takes no arguments) will also clear all of the text in the dialog pane.

Holding down the **control** key while pressing the period key will halt rule execution. The RHS actions of the currently executing rule will be allowed to complete before rule execution is halted. Holding down the **shift** key, the **control** key, and the H key will halt rule execution after the current RHS action. Remaining RHS actions will not be executed. This key combination can also be used to halt the execution of commands and functions that loop. The **Halt Rules** menu item can also be selected from the Environment menu during execution. Selecting this menu item is equivalent to holding down the **control** key while pressing the H key. The **Halt Execution** menu item can also be selected from the Environment menu during execution. Selecting this menu item is equivalent to holding down the **shift** key and the **control** key while pressing the H key.

2.1 The File Menu

2.1.1 Quit

This command exits CLIPS.

2.2 The Edit Menu

2.2.1 Cut (Ctrl+X)

This command removes selected text in the dialog pane and places it in the Clipboard. Only selected text from the current command being entered can be cut.

2.2.2 Copy (Ctrl+C)

This command copies selected text in the dialog pane and places it in the Clipboard.

2.2.3 Paste (Ctrl+V)

This command copies the contents of the Clipboard to the selection point or selected text in the dialog pane. Text can only be pasted into the current command being entered.

2.2.4 Set Dialog Font... (Ctrl+V)

This command allows the font used in the main dialog window to be changed.

2.2.5 Set Browser Font... (Ctrl+V)

This command allows the font used for displaying data in the browser tabs to be changed.

2.3 The Environment Menu

2.3.1 Clear

This command is equivalent to the CLIPS command (clear). When this command is chosen, the CLIPS command (clear) will be echoed to the dialog pane and executed. This command is not available when CLIPS is executing.

2.3.2 Load Constructs... (Ctrl+L)

This command displays a file selection dialog, allowing the user to select a text file containing constructs to be loaded into CLIPS. This command is equivalent to the CLIPS command (load <file-name>). When this command is chosen and a file is selected, the appropriate CLIPS load command will be echoed to the dialog pane and executed.

2.3.3 Load Batch... (Ctrl+Shift+L)

This command displays a file selection dialog, allowing the user to select a text file to be executed as a batch file. This command is equivalent to the CLIPS command (batch <file-name>). When this command is chosen and a file is selected, the appropriate CLIPS batch command will be echoed to the dialog pane and executed.

2.3.4 Set Directory...

This command displays a folder selection dialog, allowing the user to select the current directory associated with the CLIPS environment. File commands such as load, batch, and open use the current directory to determine the location where file operations should occur. The current directory for the dialog pane is displayed in the status bar.

2.3.5 Reset (Ctrl+R)

This command is equivalent to the CLIPS command (reset). When this command is chosen, the CLIPS command (reset) will be echoed to the dialog pane and executed.

2.3.6 Run (Ctrl+Shift+R)

This command is equivalent to the CLIPS command (run). When this command is chosen, the CLIPS command (run) will be echoed to the dialog pane and executed.

2.3.7 Halt Rules (Ctrl+H)

This command halts execution when the currently executing rule has finished executing all of its actions. This command has no effect if rules are not executing.

2.3.8 Halt Execution (Ctrl+Shift+H)

This command halts execution at the first available opportunity. If rules are executing, the currently executing rule may not complete all of its actions.

2.3.9 Clear Scrollback

This command clears all of the text in the dialog pane. From the CLIPS command prompt, the command **clear-window** (which takes no arguments) will also clear all of the text in the dialog pane.

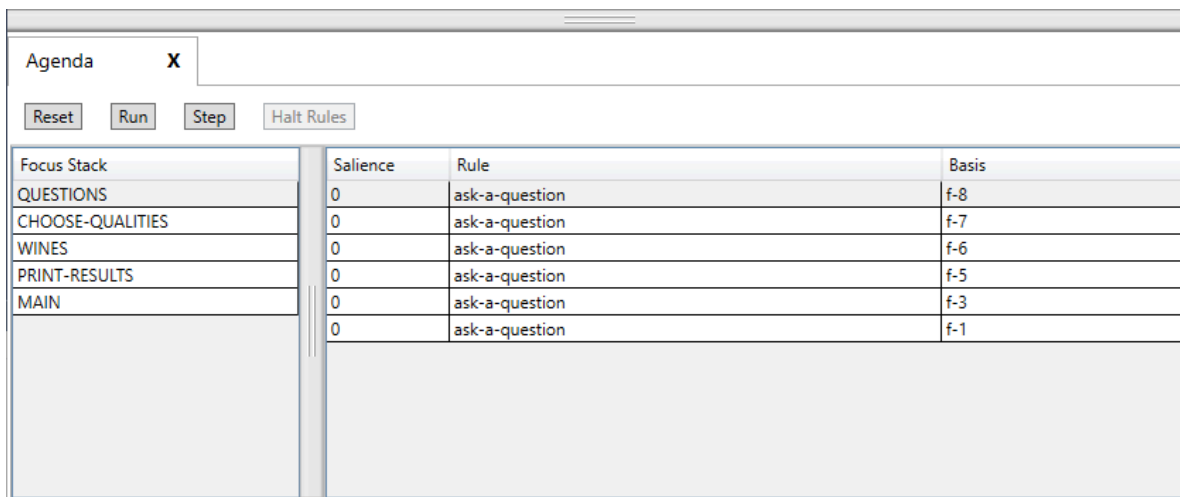
2.5 The Debug Menu

2.5.1 Watch Submenu

Watch items can be enabled or disabled by the appropriate menu item. Enabled watch items have a check to the left of the menu item. Disabled watch items have no check mark in their check box. Choosing the **All** menu item checks all of the watch items. Choosing the **None** menu item unchecks all of the watch items.

2.5.2 Agenda Browser

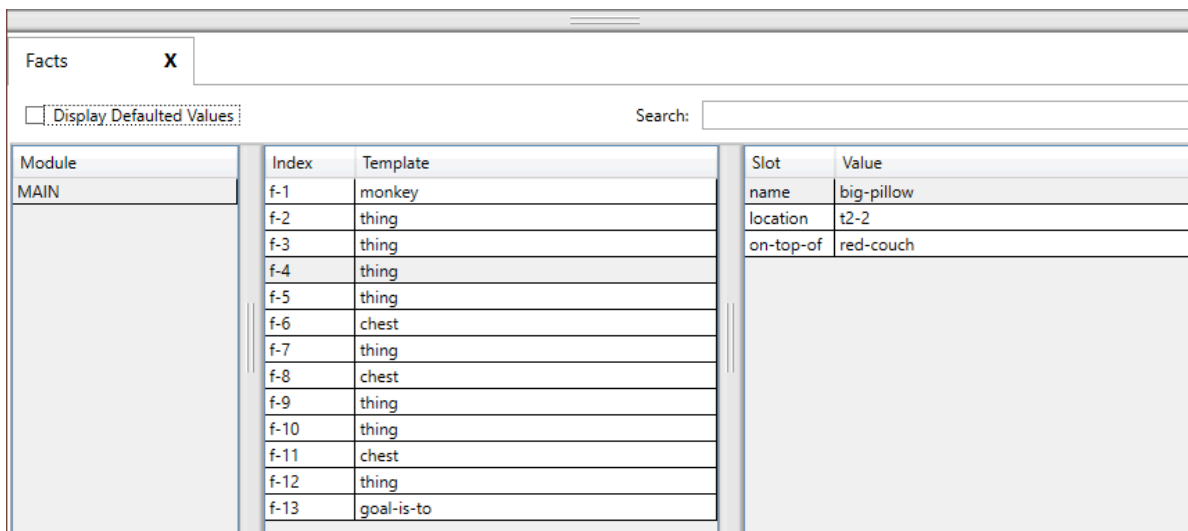
The **Agenda Browser** allows the activations on the agenda to be examined. The list on the left side of the browser shows the modules currently on the focus stack. The list of the right side of the browser shows the activations on the agenda of the selected module from the focus stack.



The **Reset** button sends a “(reset)” command to the dialog pane. The **Run** button sends a “(run)” command to the dialog pane. The **Step** button sends a “(run 1)” command to the dialog pane. Pressing the **Halt Rules** button when rules are executing will halt execution when the currently executing rule has finished all of its actions.

2.5.3 Fact Browser

The **Fact Browser** allows the facts in the fact list to be examined. The list on the left side of the browser shows the modules currently defined. The list in the middle of the browser shows the facts that are visible to the selected module from the module list. The list on the right side of the browser shows the slot values of the selected fact from the fact list.



The list of facts can be sorted based on either the fact index or the associated deftemplate name by clicking on either the **Index** or the **Template** column header. The list of slots can be sorted based on either the slot name or the slot value by clicking on either the **Slot** or the **Value** column header. If the **Display Defaulted Values** checkbox is enabled, then all of the slots of the selected fact will be displayed. If the checkbox is disabled, then only those slots that have a value different from their default slot value will be displayed.

The search text field can be used to filter the facts that are displayed in the fact list. When search text is entered and the return key is pressed each fact and its slots are examined to determine if the search text is found within one of the following templates:

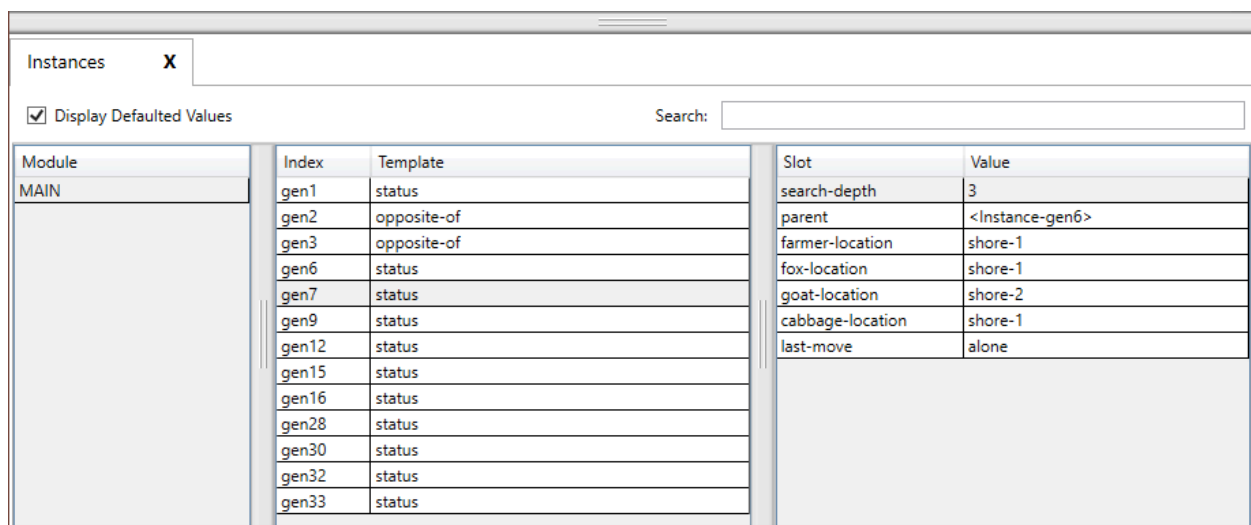
```
f-<index>
<deftemplate-name> <slot-name> <slot-value>
```

For example, if the fact associated with the deftemplate thing had a fact index of 4 and slots name with value big-pillow, location with value t2-2, and on-top-of with value red-couch, then the fact would be displayed in the fact list only if the search text was found in one of the following strings:

```
f-4
thing name big-pillow
thing location t2-2
thing on-top-of red-couch
```

2.5.4 Instance Browser

The **Instance Browser** allows the instances in the instance list to be examined. The list on the left side of the browser shows the modules currently defined. The list in the middle of the browser shows the instances that are visible to the selected module from the module list. The list on the right side of the browser shows the slot values of the selected instance from the instance list.



The list of instances can be sorted based on either the instance name or the associated defclass name by clicking on either the **Name** or the **Class** column header. The list of slots can be sorted based on either the slot name or the slot value by clicking on either the **Slot** or the **Value** column header. If the **Display Defaulted Values** checkbox is enabled, then all of the slots of the selected instance will be displayed. If the checkbox is disabled, then only those slots that have a value different from their default slot value will be displayed.

The search text field can be used to filter the instances that are displayed in the instance list. When search text is entered and the return key is pressed each instance and its slots are examined to determine if the search text is found within one of the following templates:

```
[<name>]
<defclass-name> <slot-name> <slot-value>
```

For example, if the instance associated with the defclass THING had the instance name [thing1] and slots name with value big-pillow, location with value t2-2, and on-top-of with value red-couch, then the instance would be displayed in the instance list only if the search text was found in one of the following strings:

```
[thing1]
THING name big-pillow
THING location t2-2
THING on-top-of red-couch
```

2.6 The Help Menu

2.6.1 CLIPS Home Page

Opens the CLIPS Home web page on SourceForge.

2.6.2 Online Documentation

Opens a web page with links to CLIPS Documentation including the CLIPS User's Guide, CLIPS Reference Manuals, and other Documentation.

2.6.3 Online Examples

Opens a web page with links to example programs.

2.6.4 CLIPS Expert System Group

Opens the CLIPS Expert System Group web page on Google Groups.

2.6.5 SourceForge Forums

Opens the CLIPS Discussion Forums web page on SourceForge.

2.6.6 Stack Overflow Q&A

Opens the Stack Overflow web page for the CLIPS question tag.

2.6.7 About CLIPS IDE

This command displays version information about the CLIPS IDE application.

2.8 Building the Windows Executables

In order to create the Windows executables, you must install the source code by downloading the *clips_windows_projects_642.zip* file (see appendix A for information on obtaining CLIPS). Once downloaded, you must then extract the contents of the file by right clicking on it and selecting the **Extract All...** menu item. Drag the *clips_windows_projects_642* directory into the directory you'll be using for development. In addition to the source code specific to the Windows projects, the core CLIPS source code is also included, so there is no need to download this code separately.

2.8.1 Building CLIPSIDE Using Microsoft Visual Studio Community 2022

Navigate to the *MVS_2022* directory. Open the file *CLIPS.sln* by double clicking on it or right click on it and select the **Open** menu item. After the file opens in Visual Studio, select **Configuration Manager...** from the **Build** menu. Select the **Release** configuration for CLIPSIDE, the appropriate platform (either **x64** for a 64-bit system or **x86** for a 32-bit system), and then click the **Close** button. Right click on the CLIPSIDE project name in the Solution Explorer and select the **Build** menu item. When compilation is complete, the CLIPSIDE executable will be in the corresponding <Platform>\<Configuration> directory of *MVS_2022\CLIPSIDE\bin*.

2.8.2 Building CLIPSDOS Using Microsoft Visual Studio Community 2022

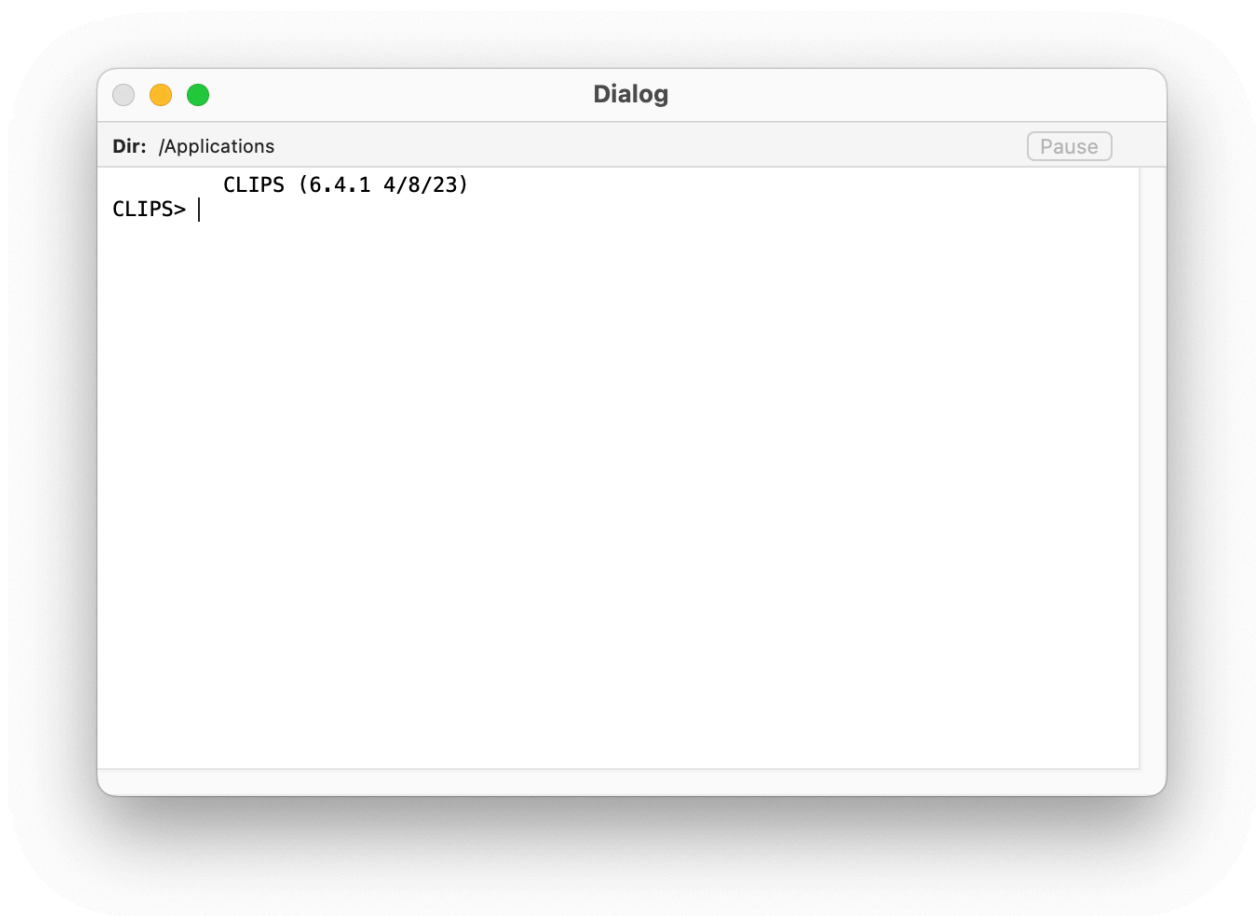
Navigate to the *MVS_2022* directory. Open the file *CLIPS.sln* by double clicking on it or right click on it and select the **Open** menu item. After the file opens in Visual Studio, select **Configuration Manager...** from the **Build** menu. Select the **Release** configuration for CLIPSDOS, the appropriate platform (either **x64** for a 64-bit system or **Win32** for a 32-bit system), and then click the **Close** button. Right click on the CLIPSDOS project name in the Solution Explorer and select the **Build** menu item. When compilation is complete, the CLIPSDOS

executable will be in the corresponding <Platform>\<Configuration> directory of the *MVS_2022\CLIPSDOS\Executables*.

Section 3:

CLIPS macOS IDE

This section provides a brief summary of the CLIPS 6.4 macOS Integrated Development Environment (IDE). The IDE provides a dialog window that allows commands to be entered in a manner similar to the standard CLIPS command line interface. Any CLIPS I/O to standard input or standard output is directed to this dialog window. In addition, the IDE also provides browser windows for examining the current state of the CLIPS environment. When launched, the IDE displays a dialog window:



A status bar is displayed beneath the title bar. On the left side of the status bar is the current working directory. A **Pause** button is on the right side of the status bar. The CLIPS IDE is multi-threaded and uses a separate thread to execute commands entered in the dialog window. Pressing the **Pause** button while a command is executing will suspend execution of the command thread. This is useful if you need to examine the output of the executing program. Pressing the **Pause** button a second time will resume execution of the command thread.

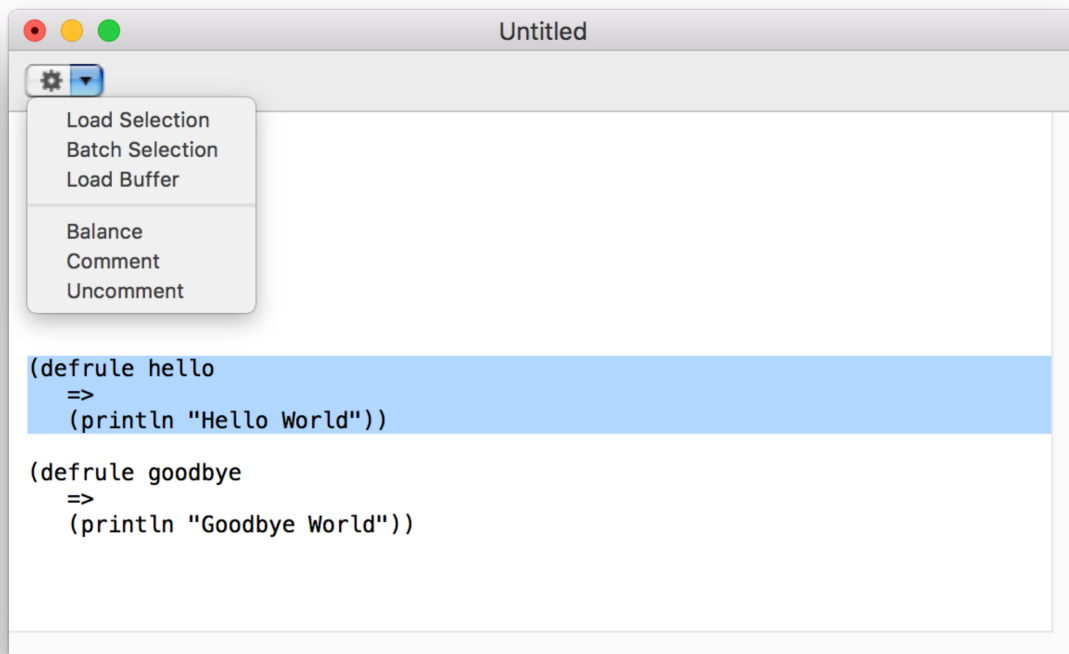
Inline editing is supported in the dialog window. The left and right arrow keys can be used to move the caret backwards and forwards through the current command. Pressing the delete key will delete the character to the left of the caret. Insertion of other characters or pasted text occurs at the caret. The esc key moves the caret to the end of the current command. The caret must be at the end of the current command in order for pressing the return key to execute the command.

A command history is also supported for the dialog window. The up and down arrows allow you to cycle through the command history. The up arrow restores the previous command and the down arrow restores the next command. Holding the shift key down when the up or down arrow is pressed takes you respectively to the beginning or end of the command history.

From the CLIPS command prompt, the command **clear-window** (which takes no arguments) will also clear all of the text in the dialog window.

Holding down the **command** key while pressing the period key will halt rule execution. The RHS actions of the currently executing rule will be allowed to complete before rule execution is halted. Holding down the **shift** key, the **command** key, and the period key will halt rule execution after the current RHS action. Remaining RHS actions will not be executed. This key combination can also be used to halt the execution of commands and functions that loop. The **Halt Rules** menu item can also be selected from the Environment menu during execution. Selecting this menu item is equivalent to holding down the **command** key while pressing the period key. The **Halt Execution** menu item can also be selected from the Environment menu during execution. Selecting this menu item is equivalent to holding down the **shift** key and the **command** key while pressing the period key.

The interface also provides a text editor for writing CLIPS programs. Editing windows contain a control strip with a drop-down menu and a content area for text:



Newly created editing windows begin with the word *Untitled* in their title bar. If an editing window is associated with a file, then the title bar will contain the file name. Beneath the title bar is a control strip. The drop-down menu on the left side of the strip provides access to the same menu items that are in the **Text** menu. In the window shown previously, selecting the **Load Selection** menu item (either from the action menu or the **Text** menu) would load the selection in the editing window in the *Dialog* window.

3.1 The CLIPS IDE Menu

3.1.1 About CLIPS

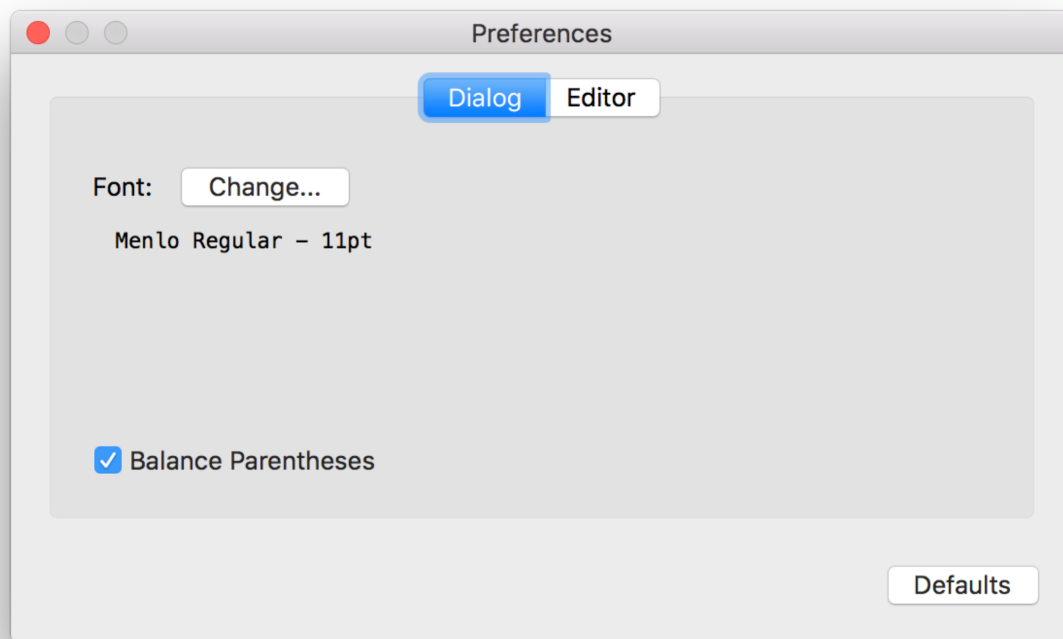
This command displays version information about the CLIPS IDE application.

3.1.2 Preferences... (⌘,)

This command displays a dialog box that allows the user to set the parameters for several options in the CLIPS MacOS IDE. With any tab selected, clicking the **Default** button restores the default settings for all tabs in the dialog.

3.1.2.1 Dialog Tab

The Dialog tab allows text options for the dialog window to be set.

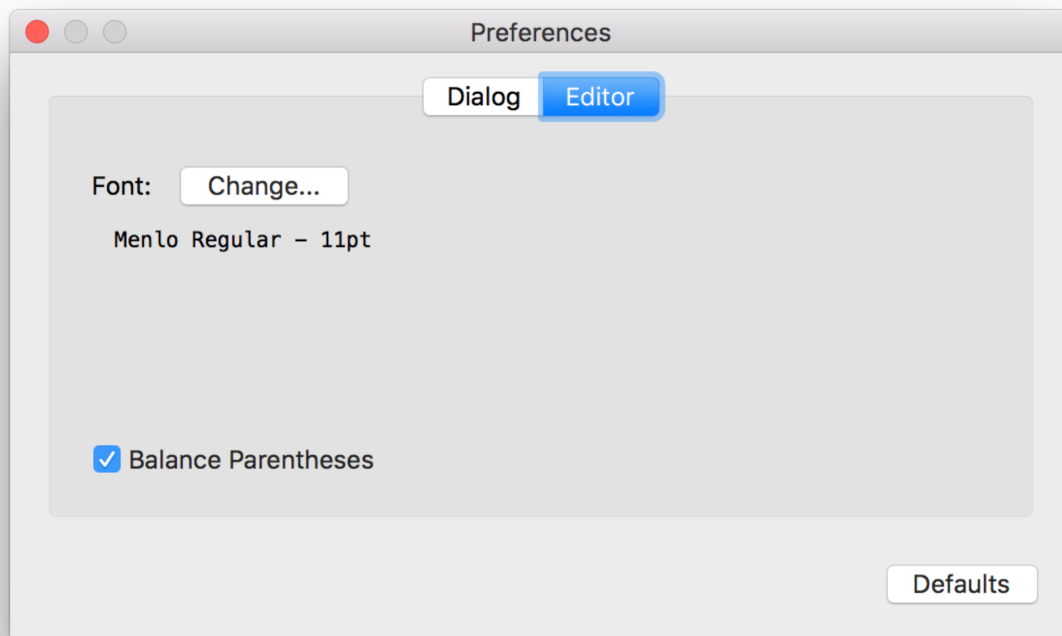


The **Change...** button allows the font used in the editing windows to be changed. When you click this button, a Fonts dialog will appear. Select a font or font size from the Font dialog and the text in the Dialog tab will change to reflect the new font or font size.

The **Balance Parentheses** check box, if enabled, causes the matching left parenthesis to be momentarily highlighted whenever a right parenthesis is type or the cursor is moved immediately after a right parenthesis in the Dialog window.

3.1.2.2 Editor Tab

The Editor tab allows text options for editing windows to be set.



The **Change...** button allows the font used in the editing windows to be changed. When you click this button, a Fonts dialog will appear. Select a font or font size from the Font dialog and the text in the Editor tab will change to reflect the new font or font size.

The **Balance Parentheses** check box, if enabled, causes the matching left parenthesis to be momentarily highlighted whenever a right parenthesis is type or the cursor is moved immediately after a right parenthesis in an editing window.

3.1.3 Quit CLIPS IDE (⌘Q)

This command causes the CLIPS IDE to quit. The user will be prompted to save any files with unsaved changes.

3.2 The File Menu

3.2.1 New (⌘N)

This command opens a new buffer for editing with the window name Untitled.

3.2.2 Open... (⌘O)

This command displays the standard file selection dialog sheet, allowing the user to select a text file to be opened in a window for editing. More than one file can be opened at the same time, however, the same file cannot be opened more than once. As files are opened, they are automatically stacked.

3.2.3 Open Recent

This command displays a list of recently opened files, allowing the user to select a text file to be opened as a buffer for editing.

3.2.4 Close (⌘W)

This command closes the active window if it has red close button in its upper left corner. If the active window is an editing window that has been modified since it was last saved, an alert sheet will confirm whether the changes should be saved or discarded or whether the close command should be canceled.

3.2.5 Save (⌘S)

This command saves the file in the active edit window. If the file is untitled, a save file dialog sheet will prompt for a file name under which to save the file.

3.2.6 Save As... (⇧⌘S)

This command allows the active edit window to be saved under a new name. A save file dialog sheet will appear to prompt for the new file name. The name of the editing window will be changed to the new file name.

3.2.7 Revert

This command restores the active edit window to the last-saved version of the file in the buffer. Any changes made since the file was last saved will be discarded.

3.2.8 Page Setup... (⇧⌘P)

This command allows the user to specify information about the size of paper used by the printer.

3.2.9 Print... (⌘P)

This command allows the user to print the active edit window.

3.3 The Edit Menu

3.3.1 Undo (⌘Z)

This command allows you to undo your last editing operation. Typing, cut, copy, paste, and delete operations can all be undone. The **Undo** menu item will change in the **Edit** menu to reflect the last operation performed. For example, if a **Paste** command was just performed, the **Undo** menu item will read **Undo Paste**.

3.3.2 Redo (⇧⌘Z)

This command allows you to redo your last editing operation. Typing, cut, copy, paste, and delete operations can all be redone. The **Redo** menu item will change in the **Edit** menu to reflect the last operation performed. For example, if a **Paste** command was just performed, the **Redo** menu item will read **Undo Paste**.

3.3.3 Cut (⌘X)

This command removes selected text in the active edit window or the dialog window and places it in the Clipboard. In the dialog window, only selected text from the current command being entered can be cut.

3.3.4 Copy (⌘C)

This command copies selected text in the active edit window or the dialog window and places it in the Clipboard.

3.3.5 Paste (⌘V)

This command copies the contents of the Clipboard to the selection point in the active edit window or the dialog window. If the selected text is in the active edit window, it is replaced by the contents of the Clipboard. In the dialog window, text can only be pasted/replaced in the current command being entered.

3.3.6 Delete

This command removes selected text in the active edit window or the display window. The selected text is not placed in the Clipboard.

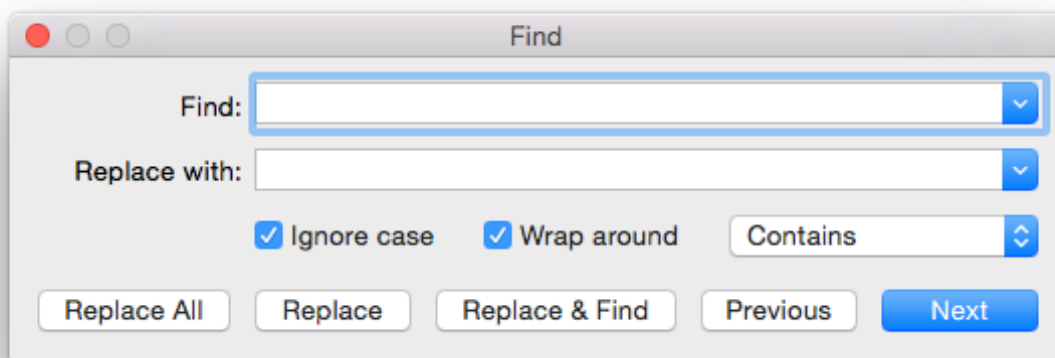
3.3.7 Select All (⌘A)

This command selects all of the text in the active edit or display window.

3.3.8 Find Submenu

3.3.8.1 Find... (⌘F)

This command displays a dialog box which allows the user to set parameters for text search and replacement operations. The dialog box that appears allows a search and replacement string to be specified.



Three other options can be set in the search dialog box. The **Ignore Case** option makes the string search operation case-insensitive for alphabetic characters; that is, the string “Upper” will match the string “uPPER”. The **Wrap Around** option determines whether the search is restarted at the top of the document when the bottom of the document is reached. The drop-down menu allows the match criterion to be set. If it is set to **Contains**, then the search matches any text containing the search string. If it is set to **Starts with**, then only whole words beginning with the search string will be matched. If it is set to **Full word**, then only whole words will be matched.

Once search options have been set, one of five search dialog buttons can be pressed. The **Replace All** button replaces all occurrences of the **Find** string with the **Replace With** string. The **Replace** button replaces the current selection with the **Replace With** string. The **Replace & Find** button

replaces the current selection with the **Replace With** string and finds and selects the next match for the **Find** string. The **Previous** button finds and selects the previous match for the **Find** string. The **Next** button finds and selects the next match for the **Find** string.

3.3.8.2 Find Next (⌘G)

This command finds and selects the next match for the **Find** string.

3.3.8.3 Find Previous (⇧⌘G)

This command finds and selects the previous match for the **Find** string.

3.3.8.4 Use Selection for Find (⌘E)

This command sets the **Find** string to the current selection.

3.3.8.5 Jump to Selection (⌘J)

This command brings the current selection into view.

3.4 The Text Menu

3.4.1 Load Selection (⌘K)

This command loads the constructs in the active edit window's current selection into CLIPS. Standard error detection and recovery routines used to load constructs from a file are also used when loading a selection (i.e., if a construct has an error in it, the rest of the construct will be skipped over until another construct to be loaded is found).

3.4.2 Batch Selection (⇧⌘K)

This command treats the active edit window's current selection as a batch file and executes it as a series of commands. Standard error detection and recovery routines used to load construct from a file are *not* used when batching a selection (i.e., if a construct has an error in it, a number of ancillary errors may be generated by subsequent parts of the same construct following the error).

3.4.3 Load Buffer

This command loads the constructs from the entire contents of active edit window into CLIPS. It is equivalent to selecting the entire buffer and executing a **Load Selection** command.

3.4.4 Balance (⌘B)

This command operates on the active edit window's current selection by expanding it until the selection begins and ends with parentheses and each parenthesis contained in the selection is properly nested (i.e. each left opening parenthesis has a properly nested right closing parenthesis and vice versa). Repeatedly using this command will select larger and larger selections of text until a balanced selection cannot be found. The balance command is a purely textual operation and does not ignore parentheses contained within CLIPS string values.

3.4.5 Comment

This command operates on the current selection in the active edit window by adding a semicolon to the beginning of each line contained in the selection.

3.4.6 Uncomment

This command operates on the current selection in the active edit window by removing a semicolon (if one exists) from the beginning of each line contained in the selection.

3.5 The Environment Menu

3.5.1 Clear

This command is equivalent to the CLIPS command (clear). When this command is chosen, the CLIPS command (clear) will be echoed to the dialog window and executed. This command is not available when CLIPS is executing.

3.5.2 Load Constructs... (⌘L)

This command displays the standard file selection dialog sheet, allowing the user to select a text file to be loaded into the knowledge base. This command is equivalent to the CLIPS command (load <file-name>). When this command is chosen and a file is selected, the appropriate CLIPS load command will be echoed to the environment window and executed.

3.5.3 Load Batch... (⇧⌘L)

This command displays the standard file selection dialog sheet, allowing the user to select a text file to be executed as a batch file. This command is equivalent to the CLIPS command (batch <file-name>). When this command is chosen and a file is selected, the appropriate CLIPS batch command will be echoed to the environment window and executed.

3.5.4 Set Directory...

This command displays the standard folder selection dialog sheet, allowing the user to select the current directory associated with the environment. CLIPS file commands such as load, batch, and open use the current directory to determine the location where file operations should occur. The current directory for an environment window is displayed in the status pane below the window title.

3.5.5 Reset (⌘R)

This command is equivalent to the CLIPS command (reset). When this command is chosen, the CLIPS command (reset) will be echoed to the dialog window and executed.

3.5.6 Run (⇧⌘R)

This command is equivalent to the CLIPS command (run). When this command is chosen, the CLIPS command (run) will be echoed to the dialog window and executed.

3.5.7 Halt Rules (⌘.)

This command halts execution when the currently executing rule has finished executing all of its actions. This command has no effect if rules are not executing.

3.5.8 Halt Execution (⇧⌘.)

This command halts execution at the first available opportunity. If rules are executing, the currently executing rule may not complete all of its actions.

3.5.9 Clear Scrollback

This command clears all of the text in the dialog window. From the CLIPS command prompt, the command **clear-window** (which takes no arguments) will also clear all of the text in the environment window.

3.6 The Debug Menu

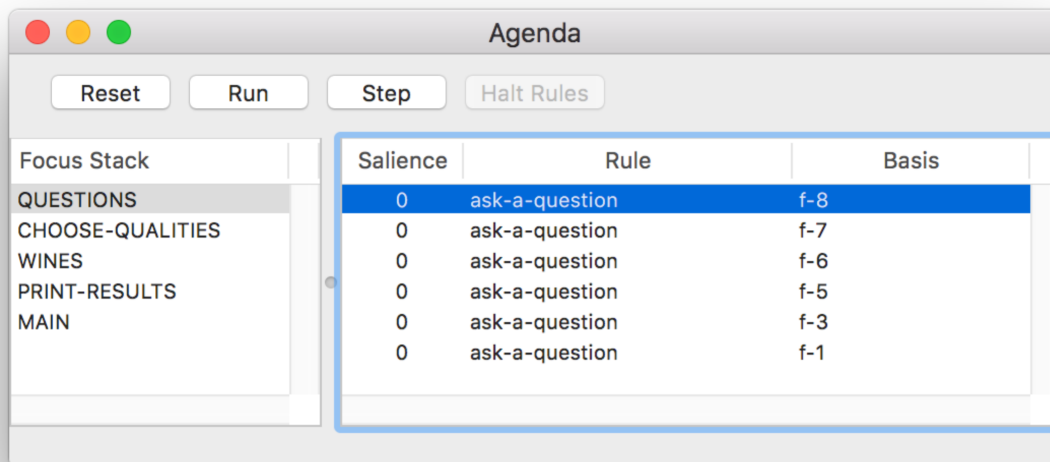
3.6.1 Watch Submenu

Watch items can be enabled or disabled by the appropriate menu item. Enabled watch items have a check to the left of the menu item. Disabled watch items have no check mark in their check box.

Choosing the **All** menu item checks all of the watch items. Choosing the **None** menu item unchecks all of the watch items.

3.6.2 Agenda Browser

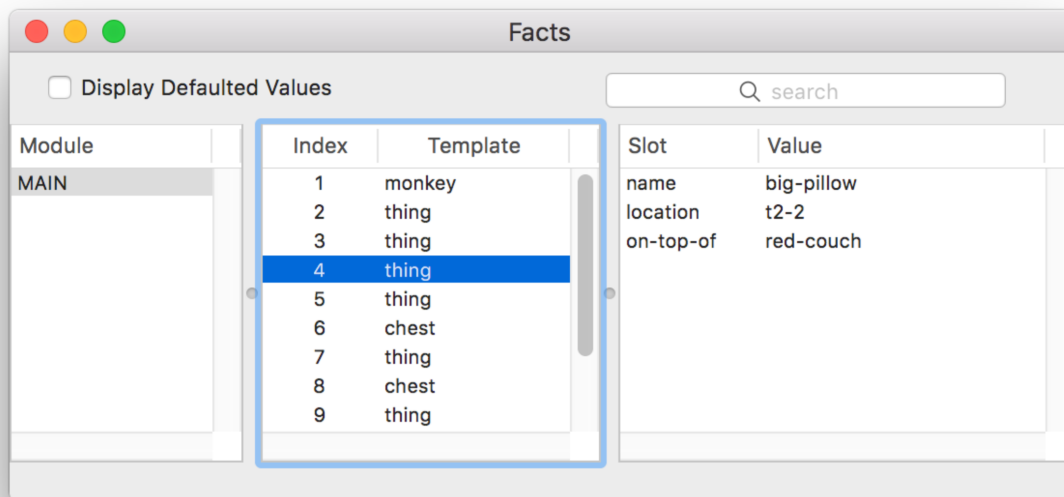
The **Agenda Browser** allows the activations on the agenda to be examined. The list on the left side of the window shows the modules currently on the focus stack. The list of the right side of the window shows the activations on the agenda of the selected module from the focus stack.



The **Reset** button sends a “(reset)” command to the dialog window. The **Run** button sends a “(run)” command to the dialog window. The **Step** button sends a “(run 1)” command to the dialog window. Pressing the **Halt Rules** button when rules are executing will halt execution when the currently executing rule has finished all of its actions.

3.6.3 Fact Browser

The **Fact Browser** allows the facts in the fact list to be examined. The list on the left side of the window shows the modules currently defined. The list in the middle of the window shows the facts that are visible to the selected module from the module list. The list on the right side of the window shows the slot values of the selected fact from the fact list.



The list of facts can be sorted based on either the fact index or the associated deftemplate name by clicking on either the **Index** or the **Template** column header. The list of slots can be sorted based on either the slot name or the slot value by clicking on either the **Slot** or the **Value** column header. If the **Display Defaulted Values** checkbox is enabled, then all of the slots of the selected fact will be displayed. If the checkbox is disabled, then only those slots that have a value different from their default slot value will be displayed.

The search text field can be used to filter the facts that are displayed in the fact list. When search text is entered and the return key is pressed each fact and its slots are examined to determine if the search text is found within one of the following templates:

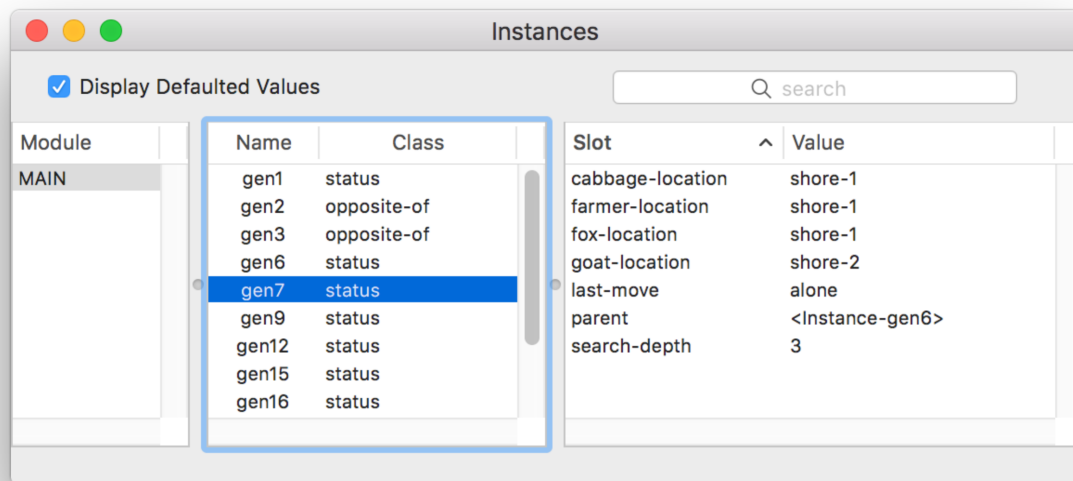
```
f-<index>
<deftemplate-name> <slot-name> <slot-value>
```

For example, if the fact associated with the deftemplate thing had a fact index of 4 and slots name with value big-pillow, location with value t2-2, and on-top-of with value red-couch, then the fact would be displayed in the fact list only if the search text was found in one of the following strings:

```
f-4
thing name big-pillow
thing location t2-2
thing on-top-of red-couch
```

3.6.4 Instance Browser

The **Instance Browser** allows the instances in the instance list to be examined. The list on the left side of the window shows the modules currently defined. The list in the middle of the window shows the instances that are visible to the selected module from the module list. The list on the right side of the window shows the slot values of the selected instance from the instance list.



The list of instances can be sorted based on either the instance name or the associated defclass name by clicking on either the **Name** or the **Class** column header. The list of slots can be sorted based on either the slot name or the slot value by clicking on either the **Slot** or the **Value** column header. If the **Display Defaulted Values** checkbox is enabled, then all of the slots of the selected instance will be displayed. If the checkbox is disabled, then only those slots that have a value different from their default slot value will be displayed.

The search text field can be used to filter the instances that are displayed in the instance list. When search text is entered and the return key is pressed each instance and its slots are examined to determine if the search text is found within one of the following templates:

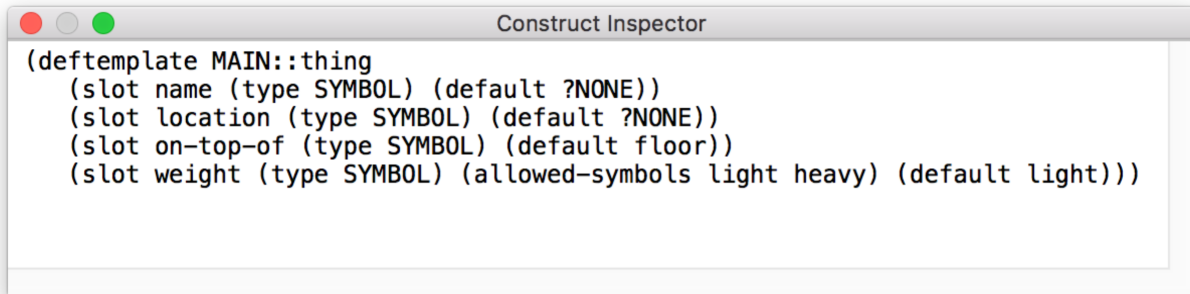
```
[<name>]
<defclass-name> <slot-name> <slot-value>
```

For example, if the instance associated with the defclass THING had the instance name [thing1] and slots name with value big-pillow, location with value t2-2, and on-top-of with value red-couch, then the instance would be displayed in the instance list only if the search text was found in one of the following strings:

```
[thing1]
THING name big-pillow
THING location t2-2
THING on-top-of red-couch
```

3.6.5 Construct Inspector

The **Construct Inspector** floats above the other CLIPS IDE windows and changes to show the text of the associated construct when one of the items from the browsers is selected.



3.7 The Window Menu

The bottom portion of the Window menu (everything below the other window management menu items) is a list of all windows associated with the CLIPS IDE. A check mark is placed by the window name to indicate that it is the frontmost window. A filled circle appears next to an edit window title that has changes that need to be saved (unless it is the frontmost window).

3.8 The Help Menu

3.8.1 CLIPS Home Page

Opens the CLIPS Home web page on SourceForge.

3.8.2 Online Documentation

Opens a web page with links to CLIPS Documentation including the CLIPS User's Guide, CLIPS Reference Manuals, and other Documentation.

3.8.3 Online Examples

Opens a web page with links to example programs.

3.8.4 CLIPS Expert System Group

Opens the CLIPS Expert System Group web page on Google Groups.

3.8.5 SourceForge Forums

Opens the CLIPS Discussion Forums web page on SourceForge.

3.8.6 Stack Overflow Q&A

Opens the Stack Overflow web page for the CLIPS question tag.

3.9 Creating the macOS Executables

In order to create the macOS executables, you must install the source code using the *clips_macos_project_642.dmg* disk image. This file can be downloaded from the SourceForge web site (see appendix A). Once downloaded, double click the file and then drag the *CLIPS Xcode Project* folder into the folder you'll be using for development. In addition to the source code specific to the macOS IDE, the core CLIPS source code is also included with the project, so there is no need to download this code separately.

3.9.1 Building the CLIPS IDE Using Xcode 16.2

Open the *CLIPS Xcode project* directory. Double click the *CLIPS.xcodeproj* file. After the file opens in the Xcode application, select the **Product** menu, then the **Scheme** submenu, and then select the **Edit Scheme...** menu item. On the **Info** tab, set the **Build Configuration** drop down menu to **Release** and the **Executable** drop down menu to **CLIPS IDE.app**. Select the **Build** menu item from the **Product** menu to create the CLIPS IDE executable. The generated executable can be found in the *:build:Release* folder.

Section 4:

CLIPS Swing IDE

This section provides a brief summary of the CLIPS 6.4 Swing Integrated Development Environment (IDE). The IDE provides a dialog window that allows commands to be entered in a manner similar to the standard CLIPS command line interface. Any CLIPS I/O to standard input or standard output is directed to this dialog window. In addition, the IDE also provides browser windows for examining the current state of the CLIPS environment.

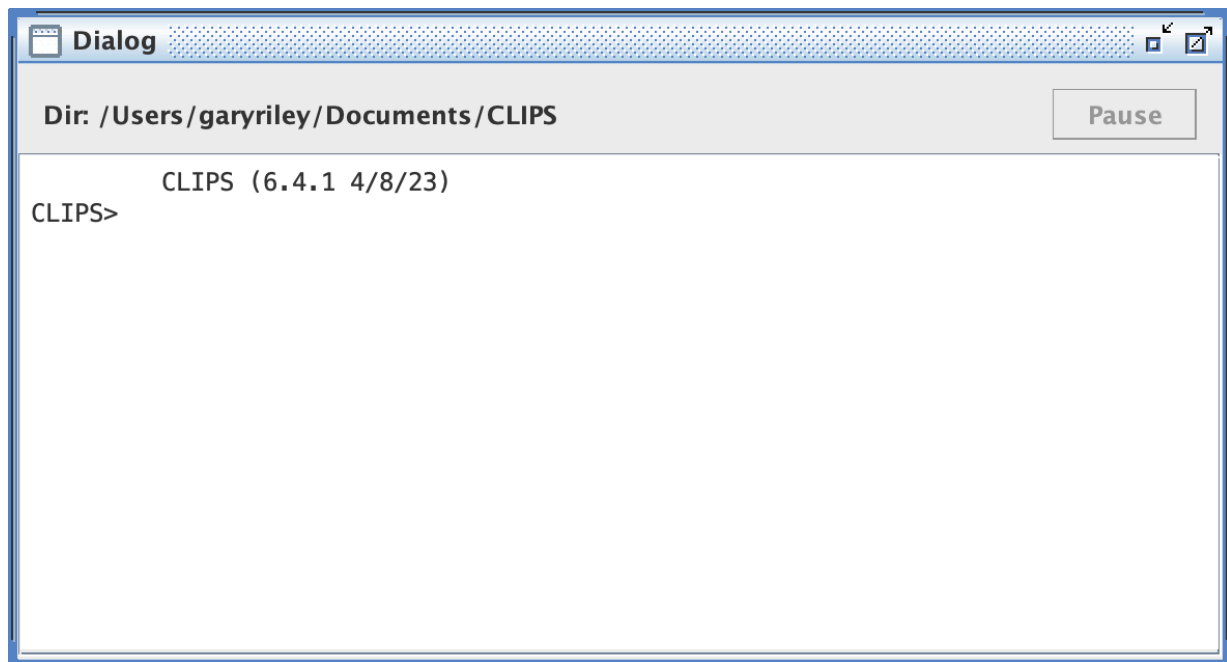
On Windows and macOS, enter the following command from the CLIPSJNI directory (see section 7.1) to launch the Swing IDE:

```
java -jar CLIPSIDE.jar
```

On Linux, you must first create the CLIPSJNI native library (see section 7.6.3). Once created, enter the following command from the CLIPSJNI directory:

```
java -Djava.library.path=. -jar CLIPSIDE.jar
```

When launched, the IDE displays a dialog window:



A status bar is displayed beneath the title bar. On the left side of the status bar is the current working directory. A **Pause** button is on the right side of the status bar. The CLIPS IDE is multi-

threaded and uses a separate thread to execute commands. Pressing the **Pause** button while a command is executing will suspend execution of the command thread. This is useful if you need to examine the output of the executing program. Pressing the **Pause** button a second time will resume execution of the command thread.

Inline editing is supported in the dialog window. The left and right arrow keys can be used to move the caret backwards and forwards through the current command. Pressing the delete key will delete the character to the left of the caret. Insertion of other characters or pasted text occurs at the caret. The esc key moves the caret to the end of the current command. The caret must be at the end of the current command in order for pressing the return key to execute the command.

A command history is also supported for the dialog window. The up and down arrows allow you to cycle through the command history. The up arrow restores the previous command and the down arrow restores the next command. Holding the shift key down when the up or down arrow is pressed takes you respectively to the beginning or end of the command history.

From the CLIPS command prompt, the command **clear-window** (which takes no arguments) will also clear all of the text in the dialog window.

Holding down the **control** key while pressing the period key will halt rule execution. The RHS actions of the currently executing rule will be allowed to complete before rule execution is halted. Holding down the **shift** key, the **control** key, and the period key will halt execution at the first possible opportunity. If rules are executing, this will typically occur after the current RHS action. Remaining RHS actions will not be executed. This key combination can also be used to halt the execution of commands and functions that loop.

4.2 The File Menu

4.2.1 New (^-N)

This command opens a new buffer for editing with the window name Untitled.

4.2.2 Open... (^-O)

This command displays the standard file selection dialog sheet, allowing the user to select a text file to be opened as a buffer for editing. More than one file can be opened at the same time, however, the same file cannot be opened more than once. As files are opened, they are automatically stacked.

4.2.3 Save (^-S)

This command saves the file in the active edit window. If the file is untitled, a save file dialog sheet will prompt for a file name under which to save the file.

4.2.4 Save As... (^+⇧-S)

This command allows the active edit window to be saved under a new name. A save file dialog sheet will appear to prompt for the new file name. The name of the editing window will be changed to the new file name.

4.2.5 Page Setup...

This command allows the user to specify information about the size of paper used by the printer.

4.2.6 Print...

This command allows the user to print the active edit window.

4.2.7 Quit CLIPS IDE (^-Q)

This command causes the CLIPS IDE to quit. The user will be prompted to save any files with unsaved changes.

4.3 The Edit Menu

4.3.1 Undo (^-Z)

This command allows you to undo your last editing operation. Typing, cut, copy, and paste operations can all be undone.

4.3.2 Redo (^+⇧-Z)

This command allows you to redo your last editing operation. Typing, cut, copy, and paste operations can all be redone.

4.3.3 Cut (^-X)

This command removes selected text in the active edit window or the dialog window and places it in the Clipboard. In the dialog window, only selected text from the current command being entered can be cut.

4.3.4 Copy (^-C)

This command copies selected text in the active edit window or the dialog window and places it in the Clipboard.

4.3.5 Paste (^-V)

This command copies the contents of the Clipboard to the selection point in the active edit window or the dialog window. If the selected text is in the active edit window, it is replaced by the contents of the Clipboard. In the dialog window, text can only be pasted/replaced in the current command being entered.

4.3.6 Set Font...

This command displays a dialog allowing the fonts to be changed for the dialog, browser, and editor windows.

4.4 The Text Menu

4.4.1 Load Selection (^-K)

This command loads the constructs in the active edit window's current selection into CLIPS. Standard error detection and recovery routines used to load constructs from a file are also used when loading a selection (i.e., if a construct has an error in it, the rest of the construct will be skipped over until another construct to be loaded is found).

4.4.2 Batch Selection (^+⇧-K)

This command treats the active edit window's current selection as a batch file and executes it as a series of commands. Standard error detection and recovery routines used to load construct from a file are *not* used when batching a selection (i.e., if a construct has an error in it, a number of ancillary errors may be generated by subsequent parts of the same construct following the error).

4.4.3 Load Buffer

This command loads the constructs from the entire contents of active edit window into CLIPS. It is equivalent to selecting the entire buffer and executing a **Load Selection** command.

4.4.4 Balance (^-B)

This command operates on the active edit window's current selection by expanding it until the selection begins and ends with parentheses and each parenthesis contained in the selection is properly nested (i.e. each left opening parenthesis has a properly nested right closing parenthesis and vice versa). Repeatedly using this command will select larger and larger selections of text until a balanced selection cannot be found. The balance command is a purely textual operation and does not ignore parentheses contained within CLIPS string values.

4.4.5 Comment

This command operates on the current selection in the active edit window by adding a semicolon to the beginning of each line contained in the selection.

4.4.6 Uncomment

This command operates on the current selection in the active edit window by removing a semicolon (if one exists) from the beginning of each line contained in the selection.

4.5 The Environment Menu

4.5.1 Clear

This command is equivalent to the CLIPS command (clear). When this command is chosen, the CLIPS command (clear) will be echoed to the dialog window and executed. This command is not available when CLIPS is executing.

4.5.2 Load Constructs... (^-L)

This command displays a file selection dialog, allowing the user to select a text file containing constructs to be loaded into CLIPS. This command is equivalent to the CLIPS command (load <file-name>). When this command is chosen and a file is selected, the appropriate CLIPS load command will be echoed to the dialog window and executed.

4.5.3 Load Batch... (^+⇧-L)

This command displays a file selection dialog, allowing the user to select a text file to be executed as a batch file. This command is equivalent to the CLIPS command (batch <file-name>). When this command is chosen and a file is selected, the appropriate CLIPS batch command will be echoed to the dialog window and executed.

4.5.4 Set Directory...

This command displays a folder selection dialog, allowing the user to select the current directory associated with the CLIPS environment. File commands such as load, batch, and open use the current directory to determine the location where file operations should occur. The current directory for the dialog window is displayed in the status pane below the window title.

4.5.5 Reset (^-R)

This command is equivalent to the CLIPS command (reset). When this command is chosen, the CLIPS command (reset) will be echoed to the dialog window and executed.

4.5.6 Run (^+⇧-R)

This command is equivalent to the CLIPS command (run). When this command is chosen, the CLIPS command (run) will be echoed to the dialog window and executed.

4.5.7 Halt Rules (^-.)

This command halts execution when the currently executing rule has finished executing all of its actions. This command has no effect if rules are not executing.

4.5.8 Halt Execution (^+⇧-.)

This command halts execution at the first available opportunity. If rules are executing, the currently executing rule may not complete all of its actions.

4.5.9 Clear Scrollback

This command clears all of the text in the dialog window. From the CLIPS command prompt, the command **clear-window** (which takes no arguments) will also clear all of the text in the dialog window.

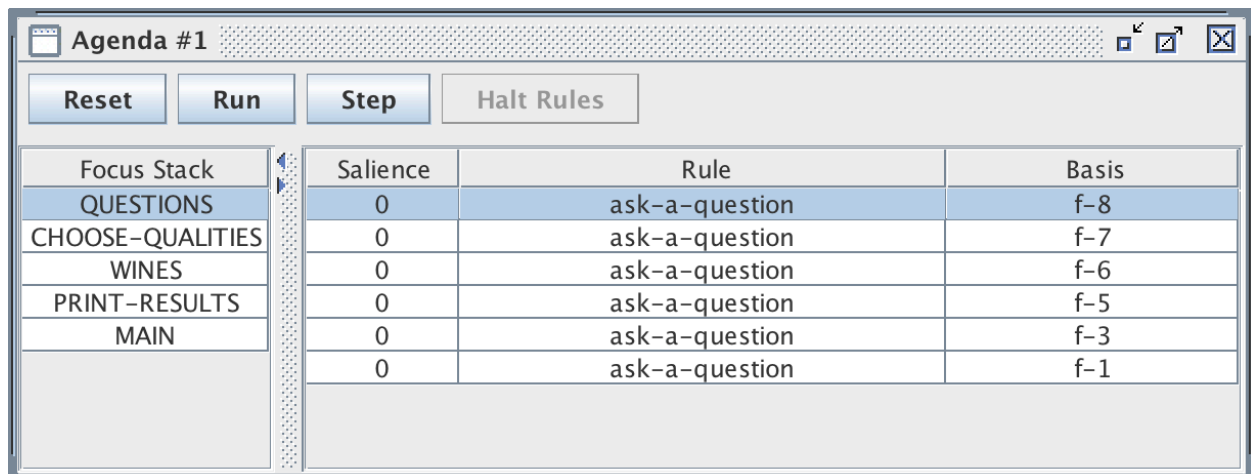
4.6 The Debug Menu

4.6.1 Watch Submenu

Watch items can be enabled or disabled by the appropriate menu item. Enabled watch items have a check to the left of the menu item. Disabled watch items have no check mark in their check box. Choosing the **All** menu item checks all of the watch items. Choosing the **None** menu item unchecks all of the watch items.

4.6.2 Agenda Browser

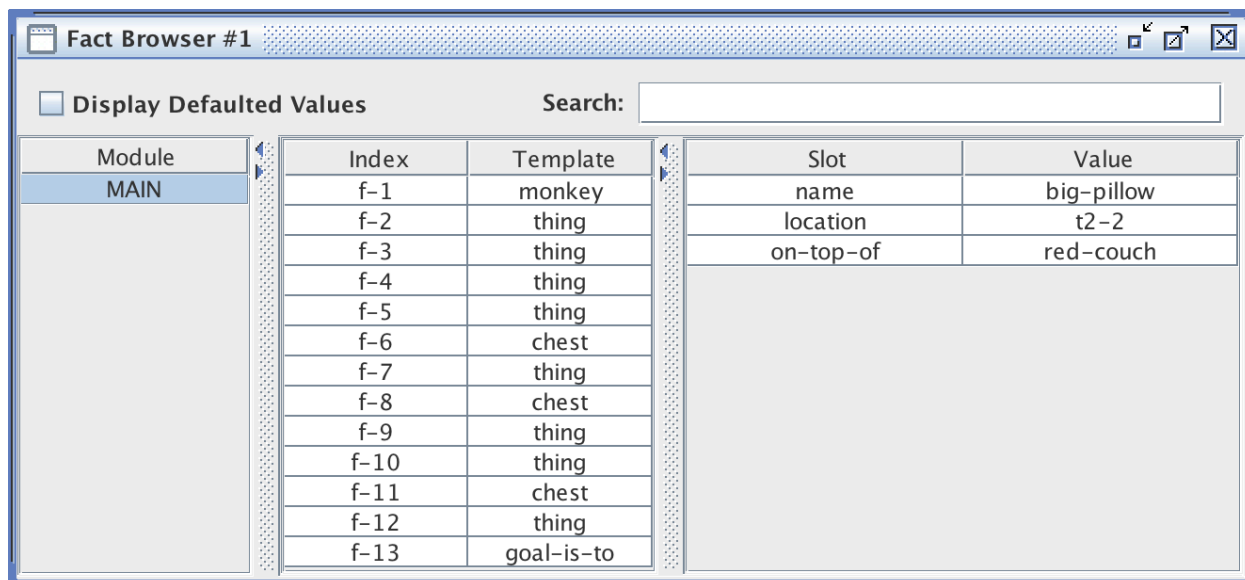
The **Agenda Browser** allows the activations on the agenda to be examined. The list on the left side of the window shows the modules currently on the focus stack. The list of the right side of the window shows the activations on the agenda of the selected module from the focus stack.



The **Reset** button sends a “(reset)” command to the dialog window. The **Run** button sends a “(run)” command to the dialog window. The **Step** button sends a “(run 1)” command to the dialog window. Pressing the **Halt Rules** button when rules are executing will halt execution when the currently executing rule has finished all of its actions.

4.6.3 Fact Browser

The **Fact Browser** allows the facts in the fact list to be examined. The list on the left side of the window shows the modules currently defined. The list in the middle of the window shows the facts that are visible to the selected module from the module list. The list on the right side of the window shows the slot values of the selected fact from the fact list.



The list of facts can be sorted based on either the fact index or the associated deftemplate name by clicking on either the **Index** or the **Template** column header. The list of slots can be sorted based on either the slot name or the slot value by clicking on either the **Slot** or the **Value** column header. If the **Display Defaulted Values** checkbox is enabled, then all of the slots of the selected fact will be displayed. If the checkbox is disabled, then only those slots that have a value different from their default slot value will be displayed.

The search text field can be used to filter the facts that are displayed in the fact list. When search text is entered and the return key is pressed each fact and its slots are examined to determine if the search text is found within one of the following templates:

```
f-<index>
<deftemplate-name> <slot-name> <slot-value>
```

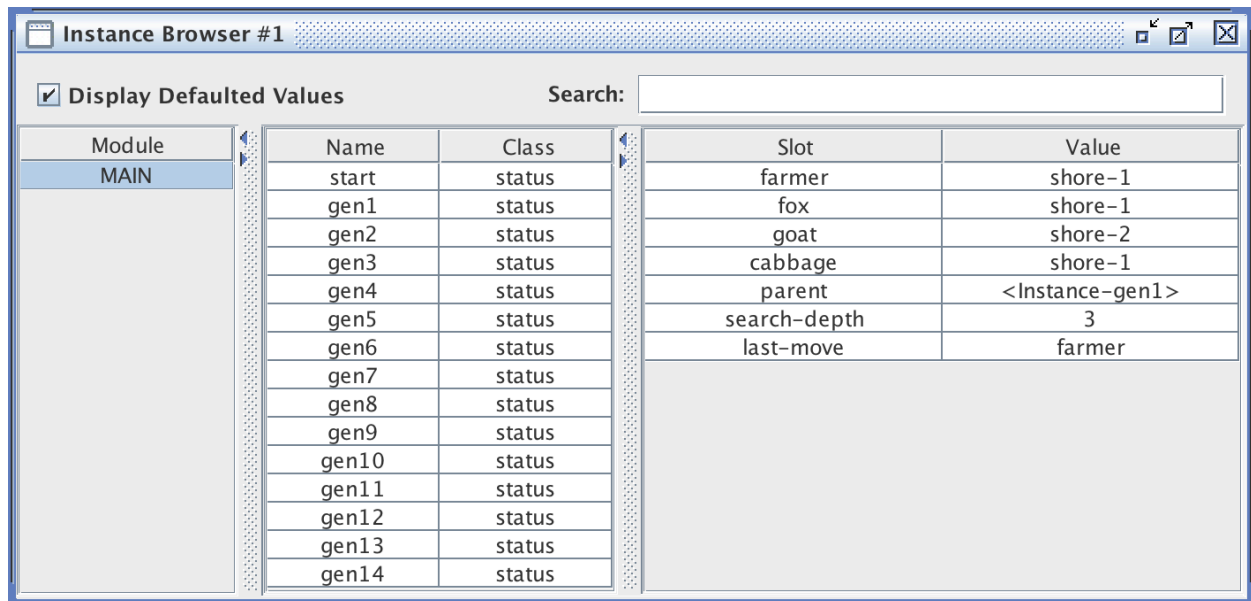
For example, if the fact associated with the deftemplate thing had a fact index of 4 and slots name with value big-pillow, location with value t2-2, and on-top-of with value red-couch, then the fact would be displayed in the fact list only if the search text was found in one of the following strings:

```
f-4
thing name big-pillow
thing location t2-2
thing on-top-of red-couch
```

4.6.4 Instance Browser

The **Instance Browser** allows the instances in the instance list to be examined. The list on the left side of the window shows the modules currently defined. The list in the middle of the window

shows the instances that are visible to the selected module from the module list. The list on the right side of the window shows the slot values of the selected instance from the instance list.



The list of instances can be sorted based on either the instance name or the associated defclass name by clicking on either the **Name** or the **Class** column header. The list of slots can be sorted based on either the slot name or the slot value by clicking on either the **Slot** or the **Value** column header. If the **Display Defaulted Values** checkbox is enabled, then all of the slots of the selected instance will be displayed. If the checkbox is disabled, then only those slots that have a value different from their default slot value will be displayed.

The search text field can be used to filter the instances that are displayed in the instance list. When search text is entered and the return key is pressed each instance and its slots are examined to determine if the search text is found within one of the following templates:

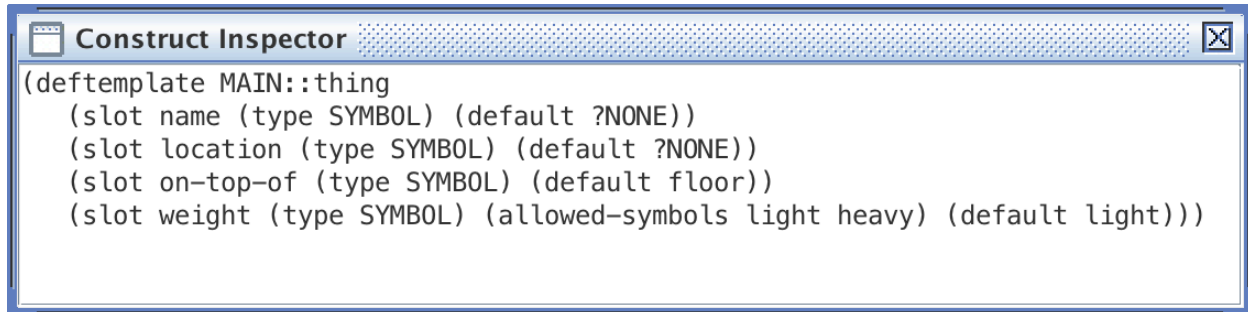
```
[<name>]
<defclass-name> <slot-name> <slot-value>
```

For example, if the instance associated with the defclass THING had the instance name [thing1] and slots name with value big-pillow, location with value t2-2, and on-top-of with value red-couch, then the instance would be displayed in the instance list only if the search text was found in one of the following strings:

```
[thing1]
THING name big-pillow
THING location t2-2
THING on-top-of red-couch
```

4.6.5 Construct Inspector

The **Construct Inspector** floats above the other CLIPS IDE windows and changes to show the text of the associated construct when one of the items from the browsers is selected.



4.7 The Window Menu

The Window menu is a list of all windows associated with the CLIPS IDE. A check mark is placed by the window name to indicate that it is the frontmost window.

4.8 The Help Menu

4.8.1 CLIPS Home Page

Opens the CLIPS Home web page.

4.8.2 Online Documentation

Opens a web page with links to CLIPS Documentation including the CLIPS User's Guide, CLIPS Reference Manuals, and other Documentation.

4.8.3 Online Examples

Opens a web page with links to example programs.

4.8.4 CLIPS Expert System Group

Opens the CLIPS Expert System Group web page on Google Groups.

4.8.5 SourceForge Forums

Opens the CLIPS Discussion Forums web page on SourceForge.

4.8.6 Stack Overflow Q&A

Opens the Stack Overflow web page for the CLIPS question tag.

4.8.7 About CLIPS IDE

This command displays version information about the CLIPS IDE application.

4.9 Creating the Swing IDE Executable

See section 7 for details on creating the Swing IDE executable.

Section 5:

CLIPS DLL Interface

This section describes various techniques for integrating CLIPS and creating executables using Microsoft Windows. The examples in this section have been tested running Windows 10 Operating System with Visual Studio Community 2019.

5.1 Installing the Source Code

In order to run the integration examples, you must install the source code by downloading the *clips_windows_projects_642.zip* file (see appendix A for information on obtaining CLIPS). Once downloaded, you must then extract the contents of the file by right clicking on it and selecting the **Extract All...** menu item. Drag the *clips_windows_projects_642* directory into the directory you'll be using for development. In addition to the source code specific to the Windows projects, the core CLIPS source code is also included, so there is no need to download this code separately.

5.2 Building the CLIPS Libraries

The Visual Studio CLIPS solution file includes four projects for building libraries. They are:

- WrappedLib
- DLL
- WrappedDLL
- CLIPSJNI

WrappedLib is a starter project that demonstrates how to build a CLIPS C++ library that is statically linked with an executable. CLIPSJNI is a starter project that demonstrates how to build a CLIPS library for use with the Java Native Interface. DLL is a starter project that demonstrates how to build a CLIPS Dynamic Link Library (DLL) that is dynamically linked with an executable. WrappedDLL is a C++ “wrapper” library that simplifies the use of the CLIPS DLL.

Unless you want to make changes to the libraries, there is no need to build them. Windows executables are available through a separate installer and the precompiled libraries are available in the Libraries directory of the corresponding project directory.

5.2.1 Building the Projects Using Microsoft Visual Studio Community 2022

Navigate to the *Projects\MVS_2022* directory. Open the file *CLIPS.sln* by double clicking on it or right click on it and select the *Open* menu item. After the file opens in Visual Studio, select *Configuration Manager...* from the *Build* menu. Select the Configuration (Debug or Release) for the library project and then click the *Close* button. Right click on the library project name in the *Solution Explorer* pane and select the *Build* menu item. When compilation is complete, the example executable will be in the corresponding <Platform>\<Configuration> directory of the *Library* directory of the corresponding DLL, WrappedLib, or WrappedDLL directory.

The CLIPSJNI project assumes that Java SE Development Kit 11.0.17 is installed on your computer and that the Java header files are contained in the directories *C:\Program Files\Java\jdk-11.0.17\include* and *C:\Program Files\Java\jdk-11.0.17\include\win32*. To change the directory setting for the location of the headers files, right click on the CLIPSJNI project and select the *Properties* menu item. In the tree view control, open the *Configuration Properties* and *C/C++* branches, then select the *General* leaf item. Edit the value in the *Additional Include Directories* editable text box to include the appropriate directory for the Java include files.

5.3 Running the Library Examples

The Visual Studio CLIPS solution file includes three projects that demonstrate the use of the static and dynamic libraries from Section 5.2. They are:

- DLLExample
- WrappedLibExample
- WrappedDLLExample

The DLLExample project demonstrates how to statically load the CLIPS DLL. The example code links with the DLL import library (CLIPS.dll). The WrappedLibExample project demonstrates how to statically load the CLIPS Wrapped C++ library (WrappedLib.lib). The C++ class *CLIPSCPPEnv* is used to provide a C++ wrapper to the CLIPS API. The WrappedDLLExample project demonstrates the use of a C++ wrapper to simplify the use of the DLL. The example code used in this project is identical to the code used with the WrappedLibExample project.

5.3.1 Building the Examples Using Microsoft Visual Studio Community 2022

Navigate to the *MVS_2022* directory. Open the file *CLIPS.sln* by double clicking on it (or right click on it and select the **Open** menu item). After the file opens in Visual Studio, select **Configuration Manager...** from the **Build** menu. Select the Configuration (**Debug** or **Release**) for the example project, the appropriate platform (either **x64** for a 64-bit system or **Win32** for a 32-bit system), and then click the **Close** button. Note that the configuration chosen should match the configuration of the libraries/DLL projects (DLL, WrappedLib, and WrappedDLL). Right

click on the example project name in the Solution Explorer pane and select the **Build** menu item. When compilation is complete, the example executable will be in the corresponding <Platform>\<Configuration> directory of the *Executables* directory of the corresponding DLLExample, WrappedLibExample, or WrappedDLLExample directory.

Section 6:

CLIPS .NET Interface

This section describes various techniques for integrating CLIPS and creating executables when using Microsoft .NET. The examples in this section have been tested running on Windows 11 with Visual Studio Community 2022.

6.1 Installing the Source Code

In order to create the Windows .NET DLL and executables, you must install the source code by downloading the *clips_windows_projects_642.zip* file (see appendix A for information on obtaining CLIPS). Once downloaded, you must then extract the contents of the file by right clicking on it and selecting the **Extract All...** menu item. Drag the *clips_windows_projects_642* directory into the directory you'll be using for development. In addition to the source code specific to the Windows projects, the core CLIPS source code is also included, so there is no need to download this code separately.

6.2 Building the .NET Library and Example Executables

The Visual Studio CLIPS solution file includes nine .NET projects:

- AnimalFormsExample
- AnimalWPFExample
- AutoFormsExample
- AutoWPFExample
- CLIPSCLRWrapper
- RouterFormsExample
- RouterWPFExample
- WineFormsExample
- WineWPFExample

The CLIPSCLRWrapper project creates a .NET DLL using a Common Language Runtime wrapper around the native CLIPS code. There are four examples utilizing the DLL with each example implemented using a Windows Forms project and a Windows Presentation Foundation project (for a total of eight projects).

6.2.1 Building the Projects Using Microsoft Visual Studio Community 2022

Navigate to the *MVS_2022* directory and open the file *CLIPS.sln* by double clicking on it (or right click on it and select the **Open** menu item). After the file opens in Visual Studio, select **Configuration Manager...** from the **Build** menu. Select the Configuration (**Debug** or **Release**) and the Platform (**x86** or **x64**) for each project and then click the **Close** button. To compile projects individually, right click on the project name in the Solution Explorer pane and select the **Build** menu item. When compilation is complete, each example application will be in the `<Platform>\<Configuration>` subdirectory of the corresponding project *bin* directory and the .NET DLL files will be in the similar subdirectory of the *Libraries* directory of the *CLIPSCLRWrapper* project.

6.3 Running the .NET Demo Programs

The CLIPS .NET demonstration programs can be run on Windows by double clicking their executable. The *CLIPSCLRWrapper.dll* file must be in the same directory as the executable.

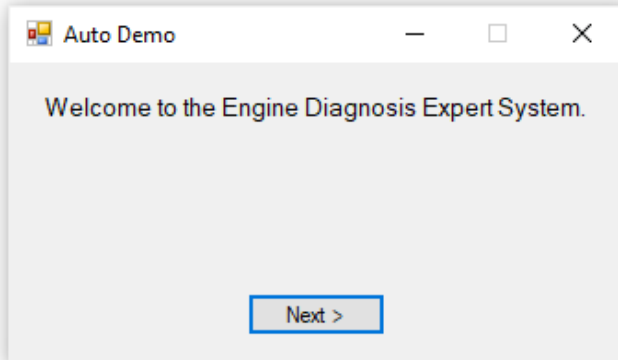
6.3.1 Wine Demo

When launched, the Wine Demo window should appear (WPF version pictured):

Wine	Recommendation Weight
Chardonnay	59%
Riesling	59%
Soave	36%
Chenin Blanc	36%
Gamay	36%
Cabernet Sauvignon	36%
Zinfandel	36%
Chablis	20%
Sauvignon Blanc	20%
Geverztraminer	20%
Valpolicella	20%
Pinot Noir	20%
Burgundy	20%

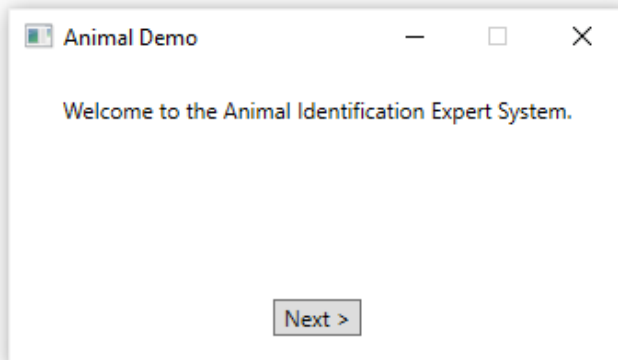
6.3.2 Auto Demo

When launched, the Auto Demo window should appear (Forms version pictured):



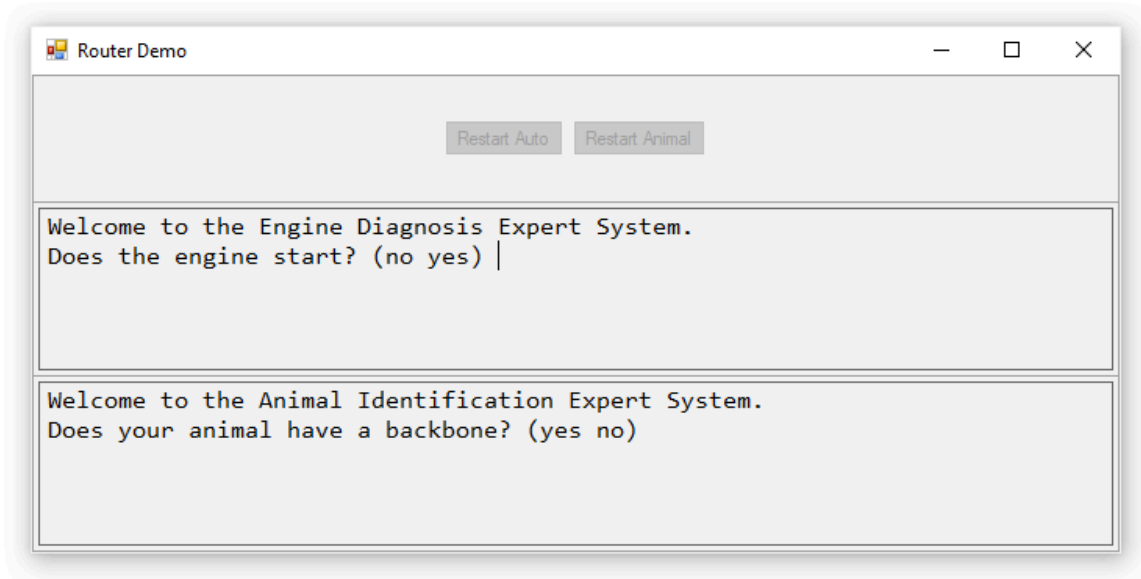
6.3.3 Animal Demo

When launched, the Animal Demo window should appear (WPF version pictured):



6.3.4 Router Demo

When launched, the Router Demo window should appear (Forms version pictured):



6.4 CLIPS .NET Classes

This section describes the classes and methods available in the CLIPSCLRWrapper.dll file for developing CLIPS .NET applications. These classes and methods reside in the CLIPSNET namespace.

6.4.1 The Environment Class

```
public class Environment
```

.NET programs interacting with CLIPS must create at least one instance of the **Environment** class.

6.4.1.1 Constructors

```
public Environment();
```

6.4.1.2 Clearing, Loading, and Creating Constructs

```
public void Clear();
```

```
public void Load(String fileName);
```

```
public void LoadFromString(String loadString);
```

```
public void LoadFromResource(String assemblyName, String resourceFile);

public void Build(String buildString);
```

The **Clear** method removes all constructs from an **Environment** instance. The **Load**, **LoadFromString**, and **LoadFromResource** methods load constructs into an **Environment** instance. The **fileName** parameter of the **Load** method specifies a file path to a text file containing constructs. The **loadString** parameter of the **LoadFromString** method is a string containing constructs. The **resourceFile** parameter of the **LoadFromResource** string specifies the resource path to a text file containing constructs and the **assemblyName** parameter specifies the assembly in which it's contained. The **Build** method loads a single construct into an **Environment** instance; it returns true if the construct was successfully loaded, otherwise it returns false.

A **CLIPSEException** is thrown by the **Clear** and **Build** methods if an error occurs. A **CLIPSLoadException** is thrown by the **Load**, **LoadFromString**, and **LoadFromResource** methods if an error occurs.

6.4.1.3 Executing Rules

```
public void Reset();

public long long Run(long long runLimit);

public long long Run();
```

The **Reset** method removes all fact and instances from an **Environment** instance and creates the facts and instances specified in deffacts and definstances constructs. The **Run** method executes the number of rules specified by the **runLimit** parameter or all rules if the **runLimit** parameter is unspecified. The **Run** method returns the number of rules executed (which may be less than the **runLimit** parameter value). A **CLIPSEException** is thrown by the **Reset** and **Run** methods if an error occurs.

6.4.1.4 Creating Facts and Instances

```
public FactAddressValue AssertString(String factString);

public InstanceAddressValue MakeInstance(String instanceString);
```

The **AssertString** method asserts a fact using the deftemplate and slot values specified by the **factString** parameter. The **MakeInstance** method creates an instance using the instance name, defclass, and slot values specified by the **instanceString** parameter. A **CLIPSEException** is thrown by the **AssertString** and **MakeInstance** methods if an error occurs.

6.5.1.5 Searching for Facts and Instances

```

public FactAddressValue FindFact(
    String deftemplate);

public FactAddressValue FindFact(
    String variable,
    String deftemplateName,
    String condition);

public List<FactAddressValue> FindAllFacts(
    String deftemplateName);

public List<FactAddressValue> FindAllFacts(
    String variable,
    String deftemplateName,
    String condition);

public InstanceAddressValue FindInstance(
    String defclassName);

public InstanceAddressValue FindInstance(
    String variable,
    String defclassName,
    String condition);

public List<InstanceAddressValue> FindAllInstances(
    String defclassName);

public List<InstanceAddressValue> FindAllInstances(
    String variable,
    String defclassName,
    String condition);

```

The **FindFact** methods return the first fact associated with the deftemplate construct specified by the **deftemplateName** parameter. The optional **variable** and **condition** parameters can be jointly specified to restrict the fact returned by specifying a CLIPS expression that must evaluate to a value other FALSE in order for the fact to be returned. Each fact of the specified deftemplate will be tested until a fact satisfying the condition is found. The fact being tested is assigned to the CLIPS variable specified by the **variable** parameter and may be referenced in the **condition** parameter. If no facts of the specified deftemplate exist, or no facts satisfy the condition, the value nullptr is returned. A **CLIPSException** is thrown if an error occurs.

The **FindAllFacts** methods returns the list of facts associated with the deftemplate construct specified by the **deftemplateName** parameter. The optional **variable** and **condition** parameters

can be jointly specified to restrict the facts returned by specifying a CLIPS expression that must evaluate to a value other FALSE in order for a fact to be returned. Each fact of the specified deftemplate will be tested to determine whether it will be added to the list. The fact being tested is assigned to the CLIPS variable specified by the **variable** parameter and may be referenced in the **condition** parameter. If no facts of the specified deftemplate exist, or no facts satisfy the condition, a list with no members is returned. A **CLIPSEException** is thrown if an error occurs.

The **FindInstance** methods return the first instance associated with the defclass construct specified by the **defclassName** parameter. The optional **variable** and **condition** parameters can be jointly specified to restrict the instance returned by specifying a CLIPS expression that must evaluate to a value other FALSE in order for the instance to be returned. Each instance of the specified defclass will be tested until an instance satisfying the condition is found. The instance being tested is assigned to the CLIPS variable specified by the **variable** parameter and may be referenced in the **condition** parameter. If no instances of the specified defclass exist, or no instances satisfy the condition, the value nullptr is returned. A **CLIPSEException** is thrown if an error occurs.

The **FindAllInstances** methods returns the list of instances associated with the defclass construct specified by the **defclassName** parameter. The optional **variable** and **condition** parameters can be jointly specified to restrict the instances returned by specifying a CLIPS expression that must evaluate to a value other FALSE in order for an instance to be returned. Each instance of the specified defclass will be tested to determine whether it will be added to the list. The instance being tested is assigned to the CLIPS variable specified by the **variable** parameter and may be referenced in the **condition** parameter. If no instances of the specified defclass exist, or no instances satisfy the condition, a list with no members is returned. A **CLIPSEException** is thrown if an error occurs.

6.5.1.5 Executing Functions and Commands

```
public PrimitiveValue Eval(String evalString);
```

The **Eval** method evaluates the command or function call specified by the **evalString** parameter and returns the result of the evaluation. A **CLIPSEException** is thrown if an error occurs.

6.5.1.6 Debugging

```
public void Watch(String watchItem);
```

```
public void Unwatch(String watchItem);
```

```
public bool GetWatchItem(String watchItem);
```

```
public void SetWatchItem(String watchItem, bool newValue);
```

The **watchItem** parameter should be one of the following static String values defined in the Environment class: FACTS, RULES, DEFFUNCTIONS, COMPILATIONS, INSTANCES,

SLOTS, ACTIVATIONS, STATISTICS, FOCUS, GENERIC_FUNCTIONS, METHODS, GLOBALS, MESSAGES, MESSAGE_HANDLERS, NONE, or ALL.

The **Watch** method enables the specified watch item and the **Unwatch** method disables the specified watch item. The **GetWatchItem** method returns the current state of the specified watch item. The **SetWatchItem** method enables the specified watch item if the **newValue** parameter is true and disables it if the **newValue** parameter is false.

6.5.1.7 Adding and Removing User Functions

```
public void AddUserFunction(
    String functionName,
    UserFunction callback);
```

```
public void AddUserFunction(
    String functionName,
    String returnTypes,
    unsigned short minArgs,
    unsigned short maxArgs,
    String argTypes,
    UserFunction callback);
```

```
public void RemoveUserFunction(
    String functionName);
```

The **AddUserFunction** method associates a CLIPS function name (specified by the **functionName** parameter) with an instance of a .NET class inheriting from the **UserFunction** class (specified by the **callback** parameter). This allows you to call .NET code from within CLIPS code. The optional parameters **returnTypes**, **minArg**, **maxArgs**, and **argTypes** can be used to specify the CLIPS primitive types returned by the function, the minimum and maximum number of arguments the function is expecting, and the primitive types allowed for each argument. The UNBOUNDED constant from the **UserFunction** class can be used for the **maxArgs** parameter to indicate that there is no upper limit on the number of arguments.

If the **returnTypes** parameter value is nullptr, then CLIPS assumes that the UDF can return any valid type. Specifying one or more type character codes, however, allows CLIPS to detect errors when the return value of a UDF is used as a parameter value to a function that specifies the types allowed for that parameter. The following codes are supported for return values and argument types:

Type Code	Type
b	Boolean
d	Double Precision Float
e	External Address
f	Fact Address
i	Instance Address
l	Long Long Integer
m	Multifield
n	Instance Name
s	String
y	Symbol
v	Void—No Return Value
*	Any Type

If the **argTypes** parameter value is null, then there are no argument type restrictions. One or more character argument types can also be specified, separated by semicolons. The first type specified is the default type (used when no other type is specified for an argument), followed by types for specific arguments. For example, "ld" indicates that the default argument type is an integer or float; "ld;s" indicates that the default argument type is an integer or float, and the first argument must be a string; "*,;m" indicates that the default argument type is any type, and the second argument must be a multifield; ";sy;ld" indicates that the default argument type is any type, the first argument must be a string or symbol; and the second argument type must be an integer or float.

The **AddUserFunction** method throws an **ArgumentException** if the association fails because one already exists for the **functionName** parameter.

The **RemoveUserFunction** method removes the association between a CLIPS function name and the user function code associated the function name. You can use this to remove a previously created associated (either to remove it altogether or to replace the old associated with a new one). The **RemoveUserFunction** method throws an **ArgumentException** if no association currently exists for the **functionName** parameter.

6.5.1.8 Managing Routers

```
public void AddRouter(
    Router theRouter);

public void DeleteRouter(
    Router theRouter);

public void ActivateRouter(
    Router theRouter);
```

```

public void DeactivateRouter(
    Router theRouter);

public void Write(
    String logicalName,
    String printString);

public void Write(
    String printString);

public void WriteLine(
    String logicalName,
    String printString);

public void WriteLine(
    String printString);

```

The **AddRouter** method adds an instance of a .NET class inheriting from the **Router** class to the list of routers checked by CLIPS for processing I/O requests. The **DeleteRouter** method removes a Router instance from the list of CLIPS routers. The methods **DeactivateRouter** and **ActivateRouter** allow a router to be disabled/enabled without removing it from the list of routers.

The **Write** and **WriteLine** methods output the string specified by the **printString** parameter through the router system. These methods direct the output to the logical name specified by the **logicalName** parameter. If the **logicalName** parameter is unspecified, output is directed to standard output. In addition, the **WriteLine** method appends a carriage return to the output.

6.5.1.9 Command Loop

```
public void CommandLoop();
```

The **CommandLoop** method starts the CLIPS Read-Eval-Print Loop (REPL) using the .NET standard input and output streams.

6.5.2 The PrimitiveValue Class and Subclasses

```

public class PrimitiveValue abstract

public class VoidValue : PrimitiveValue

public class NumberValue abstract : PrimitiveValue

public class FloatValue : NumberValue

public class IntegerValue : NumberValue

```

```

public class LexemeValue abstract : PrimitiveValue

public class SymbolValue : LexemeValue

public class StringValue : LexemeValue

public class InstanceNameValue : LexemeValue

public class MultifieldValue : PrimitiveValue , IEnumerable

public class FactAddressValue : PrimitiveValue

public class InstanceAddressValue : PrimitiveValue

public class ExternalAddressValue : PrimitiveValue

```

The **PrimitiveValue** class and its subclasses constitute the .NET representation of the CLIPS primitive data types. Several methods (such as **Eval** and **GetSlotValue**) return objects belonging to concrete subclasses of the **PrimitiveValue** class.

Several methods are provided for determining the type of a **PrimitiveValue** object:

```

public CLIPSNetType CLIPSType();

public bool IsVoid();

public bool IsLexeme();

public bool IsSymbol();

public bool IsString();

public bool IsInstanceName();

public bool IsNumber();

public bool IsFloat();

public bool IsInteger();

public bool IsFactAddress();

public bool IsInstance();

public bool IsInstanceAddress();

public bool IsMultifield();

```

```
public bool IsExternalAddress();
```

The **CLIPSType** method returns one of the following **CLIPSNETType** enumerations: **FLOAT**, **INTEGER**, **SYMBOL**, **STRING**, **MULTIFIELD**, **EXTERNAL_ADDRESS**, **FACT_ADDRESS**, **INSTANCE_ADDRESS**, **INSTANCE_NAME**, or **VOID**.

Several methods are provided for creating objects belonging to the **FloatValue**, **IntegerValue**, **SymbolValue**, **StringValue**, **InstanceNameValue**, **MultifieldValue**, and **VoidValue** classes:

```
public FloatValue();

public FloatValue(long long value);

public FloatValue(double value);

public IntegerValue();

public IntegerValue(long long value);

public IntegerValue(double value);

public SymbolValue()

public SymbolValue(String value);

public StringValue();

public StringValue(String value);

public InstanceNameValue();

public InstanceNameValue(String value);

public MultifieldValue();

public MultifieldValue(List<PrimitiveValue> value);

public VoidValue();
```

The **AssertString** and **MakeInstance** methods of the **Environment** class can be used to create objects of the **FactAddressValue** and **InstanceAddressValue** classes respectively.

The following **NumberValue** operators are available for retrieving the underlying .NET value from **NumberValue** objects:

```
public static operator long long (NumberValue ^ val);
```

```
public static operator double (NumberValue ^ val);
```

The **Value** property is available for retrieving the underlying .NET value from **IntegerValue** objects:

```
property long long Value { get; }
```

The **Value** property is available for retrieving the underlying .NET value from **FloatValue** objects:

```
property double Value { get; }
```

The **Value** property is provided to retrieve the underlying Java value from **SymbolValue**, **StringValue**, and **InstanceNameValue** objects:

```
public property String ^ Value { get; }
```

The **InstanceNameValue** class also provides a method for converting an instance name to the corresponding instance address in a specified environment:

```
public InstanceAddressValue GetInstance (Environment theEnv);
```

The following **MultifieldValue** properties provide access to the list of **PrimitiveValue** objects contained in a **MultifieldValue** method:

```
property PrimitiveValue ^ default[int] { get; }
```

```
property List<PrimitiveValue ^> ^ Value { get; }
```

```
public property int Count { get; }
```

The following **FactAddressValue** methods and properties provide access to the slot values and fact index of the associated CLIPS fact:

```
public property PrimitiveValue default[String] { get; }
```

```
public PrimitiveValue GetSlotValue(String slotName) { get; }
```

```
public property long long FactIndex { get; }
```

The following **InstanceAddressValue** methods and properties provide access to the slot values and instance name of the associated CLIPS instance:

```
public property PrimitiveValue default[String] { get; }
```

```
public PrimitiveValue GetSlotValue(String slotName);
```

```
public property String InstanceName { get; }
```

Access to the underlying values of **ExternalAddressValue** objects is not currently supported.

6.5.3 The **CLIPSEException** and **CLIPSLoadException** Classes

```
public class CLIPSEException : Exception
```

```
public class CLIPSLoadException : CLIPSEException
```

CLIPS.NET provides two subclasses of the **Exception** class for methods generating errors: **CLIPSEException** and **CLIPSLoadException**.

6.5.3.1 **CLIPSLoadException** Properties

```
public property CLIPSLineError ^ default[int] { get; }
```

```
public property List<CLIPSLineError> LineErrors { get; }
```

```
public property int Count { get; }
```

```
public class CLIPSLineError;
```

Loading constructs can generate multiple errors, so the **LineErrors** property of the **CLIPSLoadException** class returns the list of **CLIPSLineError** objects detailing each error.

6.5.3.1.1 **CLIPSLineError** Properties

```
public property String FileName { get; }
```

```
public property long LineNumber { get; }
```

```
public property String Message { get; }
```

The **FileName**, **LineNumber**, and **Message** properties respectively return the file name, line number, and error message associated with a **CLIPSLineError** object.

6.5.4 The **Router** Class

```
public class Router
```

The **Router** class allows .NET objects to interact with the CLIPS I/O router system.

6.5.4.1 Required Properties and Methods

```
public property int Priority { get; set; }

public property String Name;

public virtual bool Query(String logicalName);

public virtual void Write(String logicalName,String writeString);

public virtual int Read(String logicalName);

public virtual int Unread(String logicalName,int theChar);

public void Exit(bool failure);
```

The **Priority** property is the integer priority of the router. Routers with higher priorities are queried before routers of lower priority to determine if they can process an I/O request. The **Name** property is the identifier associated with the router. The **Query** method is called to determine if the router handles I/O requests for the **logicalName** parameter. It should return true if the router can process the request, otherwise it should return false. The **Write** method is called to output the value specified by the **writeString** parameter to the **logicalName** parameter. The **Read** method returns an input character for the **logicalName** parameter. It should return -1 if no characters are available in the input queue. The **Read** method places the character specified by parameter **theChar** back on the input queue so that it is available for the next **Read** request. It returns the value of parameter **theChar** if successful, otherwise it returns -1. The **Exit** method is invoked when the CLIPS **exit** command is issued or an unrecoverable error occurs. The **failure** parameter will either be false for an exit command or true for an unrecoverable error.

6.5.4.2 Predefined Router Names

```
public static String STDIN;

public static String STDOUT;

public static String STDWRN;

public static String STDERR;
```

The String constants STDIN, STDOUT, STDWRN, and STDERR are the standard predefined logical names used by CLIPS.

6.5.4.3 The BaseRouter Class

```
public class BaseRouter : Router
```

The **BaseRouter** class is an implementation of the **Router** interface. Its **Write**, **Read**, **Unread**, and **Exit** methods are minimal implementations; the **Write** and **Exit** methods execute no statements and the **Read** and **Unread** methods always return -1. Subclasses can override these methods as needed to create functional routers.

6.5.4.3.1 Constructors

```
public BaseRouter(
    Environment env,
    String [] queryNames);

public BaseRouter(
    Environment env,
    String [] queryNames,
    int priority);

public BaseRouter(
    Environment env,
    String [] queryNames,
    int priority,
    String routerName);
```

The **BaseRouter** constructor requires the **env** and **queryName** parameters. Optionally, the **priority** parameter or the **priority** and **routerName** parameters can be supplied. The **env** parameter is the **Environment** object associated with the **Router** object. The **queryNames** parameter is an array of strings used by the **Query** method of the **BaseRouter** object to determine whether the router handles I/O for a specific logical name. The **priority** parameter is the priority of the router; if it is unspecified, it defaults to 0. The **routerName** parameter is the name that serves as an identifier for the **BaseRouter** object; if it is unspecified an identifier will be generated for the router.

6.5.5 The UserFunction Class

```
public class UserFunction
```

The **UserFunction** class provides a method for invoking a .NET method from CLIPS code.

6.5.5.1 Required Methods

```
public PrimitiveValue Evaluate(List<PrimitiveValue> arguments);
```

Once a linkage has been made between a CLIPS function name and an object implementing the **UserFunction** interface, the **Evaluate** method is invoked when the linked CLIPS function call is

executed. The **function** arguments are evaluated and passed to the evaluate method via the **arguments** parameter.

6.5.5.2 Constants

```
public static unsigned short UNBOUNDED;
```

The UNBOUNDED constant can be used for the **maxArgs** parameter of the **AddUserFunction** method of the **Environment** class to indicate that there is no upper limit on the number of arguments.

6.5.6 Examples

The following examples require a new **Console Application** project to be created in a solution containing the **CLIPSCLRWrapper** project.

To create a new project, right click on the solution in the **Solution Explorer** and select the **Add -> New Project...** menu item. In the **language** dropdown menu, select **C#**. In the **platform** dropdown menu, select **Windows**. In the **project types** dropdown, select **Console**. In the list of available projects, select **Console Application**. Click the **Next** button.

Enter Example as the content of the **Project name** text box and then click the **Next** button.

Select the appropriate Target Framework (or use the default framework) and then click the **Create** button to add the new project to the solution.

The new project must reference the **CLIPSCLRWrapper** project. To add a reference, right click on **Dependencies** in the **Example** project in the **Solution Explorer** and then select the **Add Project Reference...** menu item. In the left pane of the dialog that appears, select **Solution** under **Projects**. In the middle pane, check the box for the **CLIPSCLRWrapper** project. Finally, click the **OK** button.

6.5.6.1 Loading Constructs from an Embedded Resource file

This example demonstrates how to load a CLIPS source file that has been embedded in the application.

First, right click on the Example project, select **Add -> New Item...** menu item. Under **Visual C# Items -> General**, select **Text File**, change the name to **hello.clp**, and then click the **Add** button. Select the **hello.clp** file in the Solution Explorer and then change the **Build Action** in the Properties window to **Embedded Resource**.

Add the following content:

```
(defrule hello
=>
  (println "Hello World"))
```

Next replace the contents of the Program.cs file with the following code:

```
using CLIPSNET;

namespace Example
{
    class Program
    {
        static void Main(string[] args)
        {
            CLIPSNET.Environment clips = new CLIPSNET.Environment();

            clips.LoadFromResource("Example", "Example.hello.clp");
            clips.Watch(CLIPSNET.Environment.RULES);
            clips.Reset();
            clips.Run();
        }
    }
}
```

Finally, build and run the program:

```
FIRE    1 hello: *
Hello World
```

6.5.6.2 Fact Query

This example demonstrates how to query CLIPS to retrieve facts.

First, replace the contents of the Program.cs file with the following code:

```
using System;
using System.Collections.Generic;

using CLIPSNET;

namespace Example
{
    class Program
    {
        static void Main(string[] args)
        {
            CLIPSNET.Environment clips = new CLIPSNET.Environment();

            clips.Build("(deftemplate person (slot name) (slot age))");
            clips.AssertString("(person (name \"Fred Jones\") (age 17))");
        }
    }
}
```

```

        clips.AssertString("(person (name \"Sally Smith\") (age 23))");
        clips.AssertString("(person (name \"Wally North\") (age 35))");
        clips.AssertString("(person (name \"Jenny Wallis\") (age 11))");

        Console.WriteLine("All people:");

        List<FactAddressValue> people = clips.FindAllFacts("person");

        foreach (FactAddressValue p in people)
        { Console.WriteLine("    " + p["name"]); }

        Console.WriteLine("All adults:");

        people = clips.FindAllFacts("?f", "person", "(>= ?f:age 18)");

        foreach (FactAddressValue p in people)
        { Console.WriteLine("    " + p["name"]); }
    }
}

```

Next, build and run the program:

```

All people:
    "Fred Jones"
    "Sally Smith"
    "Wally North"
    "Jenny Wallis"
Adults:
    "Sally Smith"
    "Wally North"

```

6.5.6.3 Big Integer Multiplication User Function

This example demonstrates how to add a user function to multiply two numbers together using big integer math. It also demonstrates using the **Eval** method to evaluate a CLIPS function call.

Replace the contents of the Program.cs file with the following code:

```

using System;
using System.Collections.Generic;
using System.Numerics;

using CLIPSNET;

namespace Example
{
    public class BIM_UDF : UserFunction
    {
        public BIM_UDF()
        {
        }
    }
}

```

```

    public override PrimitiveValue Evaluate(List<PrimitiveValue> arguments)
    {
        LexemeValue lv = (LexemeValue) arguments[0];
        BigInteger rv = BigInteger.Parse(lv.Value);

        for (int i = 1; i < arguments.Count; i++)
        {
            lv = (LexemeValue) arguments[i];
            rv = BigInteger.Multiply(rv, BigInteger.Parse(lv.Value));
        }

        return new StringValue(rv.ToString());
    }
}

class Program
{
    static void Main(string[] args)
    {
        CLIPSNET.Environment clips = new CLIPSNET.Environment();

        clips.AddUserFunction("bi*", "s", 2, UserFunction.UNBOUNDED, "s",
            new BIM_UDF());

        Console.WriteLine("( * 9 8) = " +
            clips.Eval("( * 9 8)"));
        Console.WriteLine("(bi* \"9\" \"8\") = " +
            clips.Eval("(bi* \"9\" \"8\")"));
        Console.WriteLine("( * 4294967296 4294967296) = " +
            clips.Eval("( * 4294967296 4294967296)"));
        Console.WriteLine("(bi* \"4294967296\" \"4294967296\") = " +
            clips.Eval("(bi* \"4294967296\" \"4294967296\")"));
    }
}

```

Finally, build and run the program:

```

(* 9 8) = 72
(bi* "9" "8") = "72"
(* 4294967296 4294967296) = 0
(bi* "4294967296" "4294967296") = "18446744073709551616"
$

```

6.5.6.4 Get Properties User Function

This example demonstrates how to add a user function that returns a multifield value containing the list of environment variables.

First, replace the contents of the Program.cs file with the following code:

```

using System.Collections;
using System.Collections.Generic;

using CLIPSNET;

namespace Example
{
    public class GV_UDF : UserFunction
    {
        public GV_UDF()
        {
        }

        public override PrimitiveValue Evaluate(List<PrimitiveValue> arguments)
        {
            List<PrimitiveValue> values = new List<PrimitiveValue>();

            foreach (DictionaryEntry de in
                System.Environment.GetEnvironmentVariables())
            { values.Add(new SymbolValue(de.Key.ToString())); }

            return new MultifieldValue(values);
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            CLIPSNET.Environment clips = new CLIPSNET.Environment();

            clips.AddUserFunction("get-variables","m",0,0,null,new GV_UDF());
            clips.CommandLoop();
        }
    }
}

```

Next, build and run the program:

```

CLIPS (6.4.2 1/14/25)
CLIPS> (get-variables)
(HOMEPATH COMPUTERTNAME OneDrive VisualStudioEdition PROCESSOR_REVISION
VS100COMNTOOLS DNX_HOME PkgDefApplicationConfigFile PATHEXT SystemDrive TMP TEMP
LOCALAPPDATA PUBLIC USERDOMAIN Path PROCESSOR_LEVEL PROCESSOR_IDENTIFIER PROMPT
PSModulePath NUMBER_OF_PROCESSORS FPS_BROWSER_USER_PROFILE_STRING
CommonProgramFiles ProgramData ProgramFiles FP_NO_HOST_CHECK SystemRoot
SESSIONNAME VisualStudioVersion LOGONSERVER USERPROFILE
MSBuildLoadMicrosoftTargetsReadOnly VS140COMNTOOLS VSLANG
USERDOMAIN_ROAMINGPROFILE APPDATA HOMEDRIVE USERNAME
FPS_BROWSER_APP_PROFILE_STRING PROCESSOR_ARCHITECTURE OS ComSpec VisualStudioDir
windir ALLUSERSPROFILE)
CLIPS> (exit)

```

Section 7:

CLIPS Java Native Interface

This section describes the CLIPS Java Native Interface (CLIPSJNI) and the examples demonstrating the integration of CLIPS with a Swing interface. The examples have been tested with the following software environments:

- Windows 11 with JDK 11.0.17 and Visual Studio Community 2022
- MacOS 15.2 with JDK 18.0.1.1 and Xcode 16.2
- Linux: Ubuntu 22.04 LTS with OpenJDK 17.0.13, Debian 11.3 with OpenJDK 17.0.13, Fedora 36 with OpenJDK 17.0.7, CentOS 9 with OpenJDK 17.0.13, and Mint 20.3 with OpenJDK 17.0.13.

7.1 CLIPSJNI Directory Structure

In order to use CLIPSJNI, you must obtain the source code by downloading either the *clips_jni_642.zip* or *clips_jni_642.tar.gz* file from the Files page on the CLIPS SourceForge web page (see appendix A for the SourceForge URL). When uncompressed the *CLIPSJNI* directory contains the following structure:

```
CLIPSJNI
  bin
    animal
    auto
    clipsjni
    ide
    router
    sudoku
    wine
  java-src
  net
  sf
    clipsrules
    jni
      examples
      animal
      resources
```



```

    auto
      resources
    ide
      resources
    router
      resources
    sudoku
      resources
    wine
      resources
  library-src

```

If you are using the CLIPSJNI on Windows or macOS, then the native CLIPS library is already contained in the top-level *CLIPSJNI* directory.

On other systems or 32-bit systems, you must create a native library using the source files contained in the *library-src* directory before you can utilize the CLIPSJNI.

The *CLIPSJNI.jar* file is also contained in the top-level *CLIPSJNI* directory. The source files used to create the jar file are contained in the *java-src* directory.

7.2 Issuing Commands from the Terminal

As packaged, invoking and compiling various CLIPSJNI components requires that you enter commands from a terminal application.

On Windows 11, to run the precompiled Java applications, launch the Command Prompt application (enter ‘Command Prompt’ in the search field of the taskbar and then click on the Command Prompt app). To recompile the native library or use the provided makefiles to rebuild the Java source code, you must have Visual Studio installed. In this case, launch the Command Prompt application by selecting Start > All apps > Visual Studio 2022 > Developer Command Prompt for VS 2022. Using the *Developer Command Prompt for VS 2022* application sets the appropriate paths to use the Visual Studio compiler and make tools. Alternately *x86 Native Tools Command Prompt for VS 2022* or *x64 Native Tools Command Prompt for VS 2022* can be used to compile for a specific processor architecture.

On macOS, click the Spotlight icon in the menu bar, enter ‘Terminal’ in the search field, and then double click on Terminal.app in the search results to launch the application.

On Ubuntu, click on the “Search your computer” icon, enter ‘Terminal’ in the search field, and then click on Terminal in the search results to launch the application.

On Fedora and Debian, click on Activities in the menu bar, click the Show Applications icon, enter ‘Terminal’ in the search field, and then click on Terminal in the search results to launch the application.

On CentOS, click on Applications in the menu bar, click on Activities Overview, click the Show Applications icon, enter ‘Terminal’ in the search field, and then click on Terminal in the search results to launch the application.

On Mint, click on Menu in the lower toolbar, enter ‘Terminal’ in the search field, and then click Terminal in the search results to launch the application.

Once the terminal has been launched, set the directory to the CLIPSJNI top-level directory (using the `cd` command). Unless otherwise noted, all commands should be entered while in the CLIPSJNI directory.

7.3 Running CLIPSJNI in Command Line Mode

You can invoke the command line mode of CLIPS through CLIPSJNI to interactively enter commands while running within a Java environment.

On Windows and macOS, enter the following command from the CLIPSJNI directory:

```
java -jar CLIPSJNI.jar
```

On Linux, you must first create the CLIPSJNI native library (see section 7.6.3). Once created, enter the following command from the CLIPSJNI directory:

```
java -Djava.library.path=. -jar CLIPSJNI.jar
```

The CLIPS banner and command prompt should appear:

```
CLIPS (6.4.2 1/14/25)
CLIPS>
```

7.4 Running the Swing Demo Programs

The Swing CLIPSJNI demonstration programs can be run on Windows 11 or macOS using the precompiled native libraries in the CLIPSJNI top-level directory. On Linux and other systems, a CLIPSJNI native library must first be created before the programs can be run.

7.4.1 Sudoku Demo

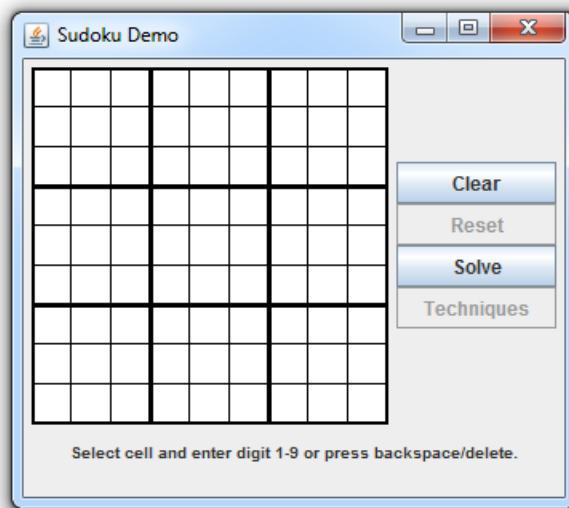
To run the Sudoku demo on Windows 11 or macOS, enter the following command:

```
java -jar SudokuDemo.jar
```

To run the Sudoku demo on Linux, enter the following command:

```
java -Djava.library.path=. -jar SudokuDemo.jar
```

The Sudoku Demo window should appear (Windows 11 pictured):



7.4.2 Wine Demo

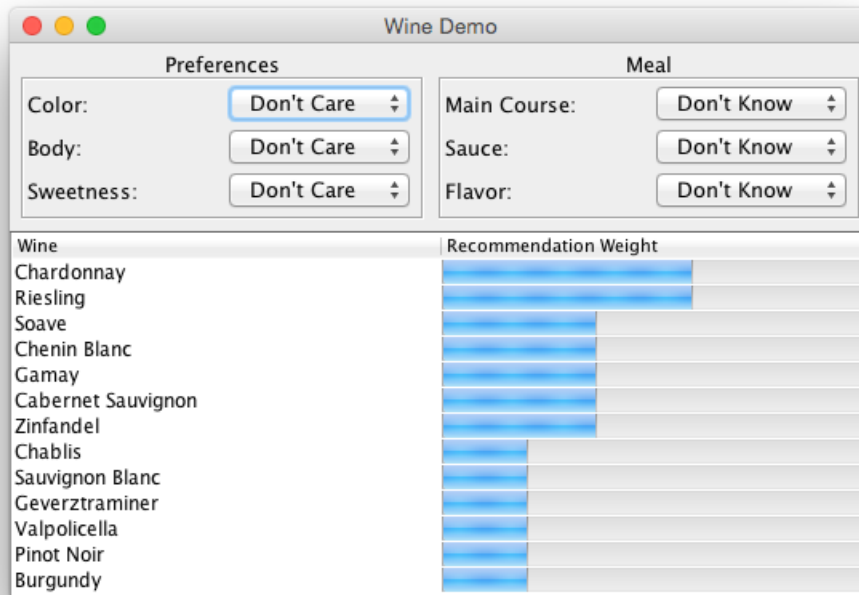
To run the Wine demo on Windows 11 or macOS, enter the following command:

```
java -jar WineDemo.jar
```

To run the Wine demo on Linux, enter the following command:

```
java -Djava.library.path=. -jar WineDemo.jar
```

The Wine Demo window should appear (macOS pictured):



7.4.3 Auto Demo

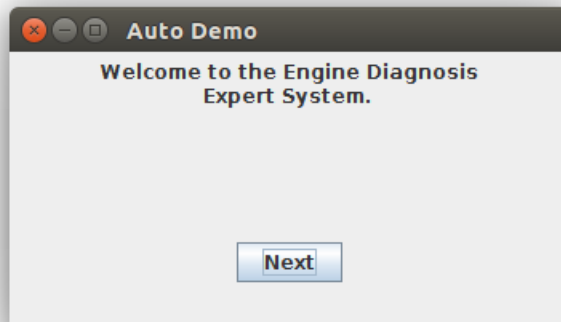
To run the Auto demo on Windows 11 or macOS, enter the following command:

```
java -jar AutoDemo.jar
```

To run the Auto demo on Linux, enter the following command:

```
java -Djava.library.path=. -jar AutoDemo.jar
```

The Auto Demo window should appear (Ubuntu pictured):



7.4.4 Animal Demo

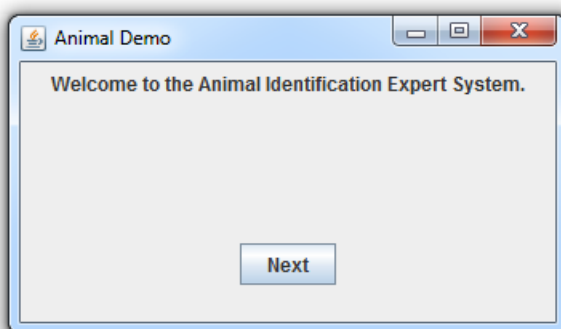
To run the Animal demo on Windows 11 or macOS, enter the following command:

```
java -jar AnimalDemo.jar
```

To run the Animal demo on Linux, enter the following command:

```
java -Djava.library.path=. -jar AnimalDemo.jar
```

The Animal Demo window should appear (Windows 11 pictured):



7.4.5 Router Demo

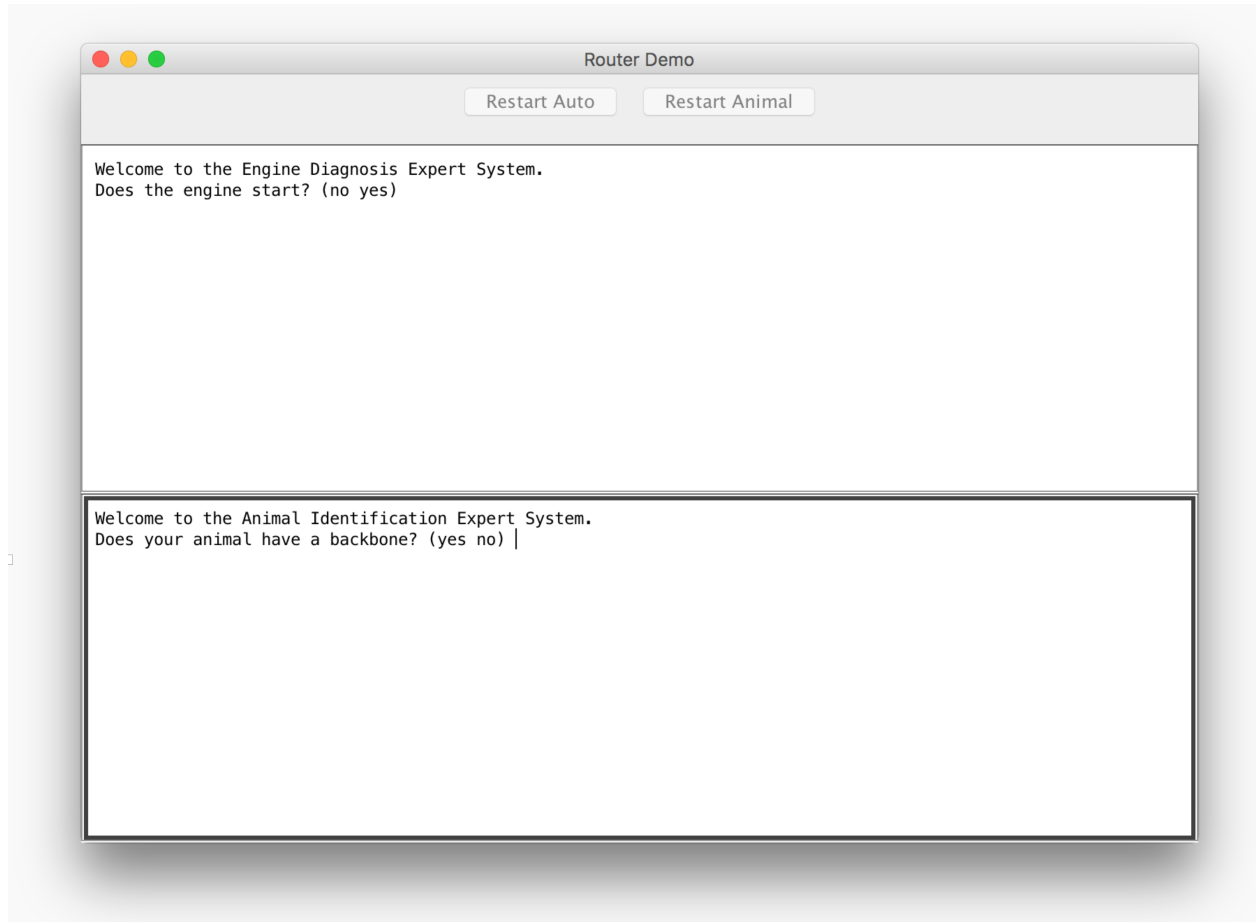
To run the Router demo on Windows 11 or macOS, enter the following command:

```
java -jar RouterDemo.jar
```

To run the Router demo on Linux, enter the following command:

```
java -Djava.library.path=. -jar RouterDemo.jar
```

The Router Demo window should appear (macOS pictured):



7.5 Creating the CLIPSJNI JAR File

If you wish to add new functionality to the CLIPSJNI package, it is necessary to recreate the CLIPSJNI jar file. The CLIPSJNI distribution already contains the precompiled CLIPSJNI jar file in the top-level CLIPSJNI directory, so if you are not adding new functionality to the CLIPSJNI package, you do not need to recreate the jar file (unless you want to create a jar file using a different version of Java).

If you are adding new native functions to the CLIPSJNI package, it is also necessary to create the JNI header file that is used to compile the native library. While you are in the CLIPSJNI directory, enter the following command:

```
javah -d library-src -classpath java-src -jni net.sf.clipsrules.jni.Environment
```

This command creates a file named `net_sf_clipsrules_jni_Environment.h` and places it in the `CLIPSJNI/library-src` directory.

On macOS, enter the following command to compile the CLIPSJNI java source and generate the JAR file:

```
make -f makefile.mac clipsjni
```

On Windows 11, enter the following command to compile the CLIPSJNI java source and generate the JAR file:

```
nmake -f makefile.win clipsjni
```

On Ubuntu, enter the following command to compile the CLIPSJNI java source and generate the JAR file:

```
make -f makefile.ubu clipsjni
```

7.6 Creating the CLIPSJNI Native Library

The CLIPSJNI distribution already contains a precompiled universal library for macOS, `libCLIPSJNI.jnilib`, and for Windows, `CLIPSJNI.dll`, in the top-level CLIPSJNI directory. It is necessary to create a native library only if you are using the CLIPSJNI with an operating system other than macOS or Windows. You must also create the native library if you want to add new functionality to the CLIPSJNI package by adding additional native functions. The steps for creating a native library varies between operating systems, so some research may be necessary to determine how to create one for your operating system.

7.6.1 Creating the Native Library on macOS

Launch the Terminal application (as described in section 7.2). Set the directory to the `CLIPSJNI/library-src` directory (using the `cd` command).

To create a universal native library that can run on both Intel and ARM 64 bit architectures, enter the following command:

```
make -f makefile.mac
```

Once you have create the native library, copy the `libCLIPSJNI.jnilib` file from the `CLIPSJNI/library-src` to the top-level CLIPSJNI directory.

7.6.2 Creating the Native Library on Windows 11

Launch the Terminal application (as described in section 7.2). Set the directory to the CLIPSJNI/library-src directory (using the `cd` command).

To create the native library DLL, enter the following command:

```
nmake -f makefile.win
```

Once you have create the native library, copy the CLIPSJNI.dll file from the CLIPSJNI/library-src to the top-level CLIPSJNI directory.

7.6.3 Creating the Native Library On Linux

Launch the Terminal application (as described in section 7.2). Set the directory to the CLIPSJNI/library-src directory (using the `cd` command).

To create a native library, enter the following command (where `<distribution>` is either `ubuntu`, `fedora`, `debian`, `mint`, or `centos`):

```
make -f makefile.lnx <distribution>
```

Once you have create the shared library, copy the `libCLIPSJNI.so` file from the CLIPSJNI/library-src to the top-level CLIPSJNI directory.

7.7 Recompiling the Swing Demo Programs

If you want to make modification to the Swing Demo programs, you can recompile them using the makefiles in the CLIPSJNI directory.

7.7.1 Recompiling the Swing Demo Programs on macOS

Use these commands to recompile the examples:

```
make -f makefile.mac sudoku
```

```
make -f makefile.mac wine
```

```
make -f makefile.mac auto
```

```
make -f makefile.mac animal
```

```
make -f makefile.mac router
```



```
make -f makefile.mac ide
```

7.7.2 Recompiling the Swing Demo Programs on Windows

Use these commands to recompile the examples:

```
nmake -f makefile.win sudoku
```

```
nmake -f makefile.win wine
```

```
nmake -f makefile.win auto
```

```
nmake -f makefile.win animal
```

```
nmake -f makefile.win router
```

```
nmake -f makefile.win ide
```

7.7.3 Recompiling the Swing Demo Programs on Linux

Use these commands to recompile the examples:

```
make -f makefile.lnx sudoku
```

```
make -f makefile.lnx wine
```

```
make -f makefile.lnx auto
```

```
make -f makefile.lnx animal
```

```
make -f makefile.lnx router
```

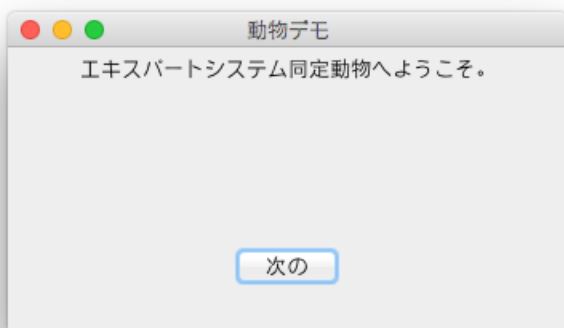
```
make -f makefile.lnx ide
```

7.8 Internationalizing the Swing Demo Programs

The Swing Demo Programs have been designed for internationalization. Several software generated example translations have been provided including Japanese (language code ja), Russian (language code ru), Spanish (language code es), and Arabic (language code ar). The Sudoku and Wine demos make use of translations just for the Swing Interface. The Auto and Animal demos also demonstrate the use of translation text from within CLIPS. To make use of one of the translations, specify the language code when starting the demonstration program. For example, to run the Animal Demo in Japanese on Mac OS X, use the following command:

```
java -Duser.language=ja -jar AnimalDemo.jar
```

The welcome screen for the program should appear in Japanese rather than English:



It may be necessary to install additional fonts to view some languages. On macOS, you can see which languages are supported by launching ‘System Preferences’ and clicking the ‘Language & Region’ icon. On Windows 10, you can see which languages are supported by launching Settings, selecting ‘Time and language,’ and then selecting ‘Region and language.’

To create translations for other languages, first determine the two-character language code for the target language. Make a copy in the resources directory of the ASCII English properties file for the demo program and save it as a UTF-8 encoded file including the language code in the name and using the .source extension. A list of language code is available at <http://www.mathguide.de/info/tools/languagecode.html>. For example, to create a Greek translation file for the Wine Demo, create the UTF-8 encoded WineResources_el.source file from the ASCII WineResources.properties file. Note that this step requires that you to do more than just duplicate the property file and rename it. You need to use a text editor that allows you to change the encoding from ASCII to UTF-8.

Once you’ve created the translation source file, edit the values for the properties keys and replaced the English text following each = symbol with the appropriate translation. When you have completed the translation, use the Java native2ascii utility to create an ASCII text file from the source file. For example, to create a Greek translation for the Wine Demo program, you’d use the following command:

```
native2ascii -encoding UTF-8 WineResources_el.source WineResources_el.properties
```

Note that the properties file for languages containing non-ASCII characters will contain Unicode escape sequences and is therefore more difficult to read (assuming of course that you can read the language in the original source file). This is the reason that two files are used for creating the translation. The UTF-8 source file is encoded so that you can read and edit the translation and the ASCII properties file is encoded in the format expected for use with Java internationalization features.

The CLIPS translation files stored in the resource directory (such as `animal_es.clp`) can be duplicated and edited to support new languages. The base name of each new file should end with the appropriate two-letter language code. There is no need to convert these UTF-8 files to another format as CLIPS can read these directly.

7.9 CLIPSJNI Classes

This section describes the classes and methods available in the `CLIPSJNI.jar` file for developing CLIPS applications in Java.

7.9.1 The Environment Class

```
public class Environment
```

Java programs interacting with CLIPS must create at least one instance of the **Environment** class.

7.9.1.1 Constructors

```
public Environment()
```

7.9.1.2 Clearing, Loading, and Creating Constructs

```
public void clear() throws CLIPSException
```

```
public void load(String fileName) throws CLIPSLoadException
```

```
public void loadFromString(String loadString) throws CLIPSLoadException
```

```
public void loadFromResource(String resourceFile) throws CLIPSLoadException
```

```
public void build(String buildString) throws CLIPSException
```

The **clear** method removes all constructs from an **Environment** instance. The **load**, **loadFromString**, and **loadFromResource** methods load constructs into an **Environment** instance. The **fileName** parameter of the **load** method specifies a file path to a text file containing constructs. The **loadString** parameter of the **loadFromString** method is a string containing constructs. The **resourceFile** parameter of the **loadFromResource** string specifies the resource path to a text file containing constructs. The **build** method loads a single construct into an **Environment** instance.

7.9.1.3 Executing Rules

```
public void reset() throws CLIPSException
```

```
public long run(long runLimit) throws CLIPSEException
```

```
public long run() throws CLIPSEException
```

The **reset** method removes all fact and instances from an **Environment** instance and creates the facts and instances specified in deffacts and definstances constructs. The **run** method executes the number of rules specified by the **runLimit** parameter or all rules if the **runLimit** parameter is unspecified. The **run** method returns the number of rules executed (which may be less than the **runLimit** parameter value).

7.9.1.4 Creating Facts and Instances

```
public FactAddressValue assertString(String factStr) throws CLIPSEException
```

```
public InstanceAddressValue makeInstance(String instanceStr) throws CLIPSEException
```

The **assertString** method asserts a fact using the deftemplate and slot values specified by the **factStr** parameter. The **makeInstance** method creates an instance using the instance name, defclass, and slot values specified by the **instanceStr** parameter.

7.9.1.5 Searching for Facts and Instances

```
public FactAddressValue findFact(
    String deftemplate) throws CLIPSEException
```

```
public FactAddressValue findFact(
    String variable,
    String deftemplateName,
    String condition) throws CLIPSEException
```

```
public List<FactAddressValue> findAllFacts(
    String deftemplateName) throws CLIPSEException
```

```
public List<FactAddressValue> findAllFacts(
    String variable,
    String deftemplateName,
    String condition) throws CLIPSEException
```

```
public InstanceAddressValue findInstance(
    String defclassName) throws CLIPSEException
```

```
public InstanceAddressValue findInstance(
    String variable,
    String defclassName,
    String condition) throws CLIPSEException
```

```
public List<InstanceAddressValue> findAllInstances(
    String defclassName) throws CLIPSException
```

```
public List<InstanceAddressValue> findAllInstances(
    String variable,
    String defclassName,
    String condition) throws CLIPSException
```

The **findFact** methods return the first fact associated with the deftemplate construct specified by the **deftemplateName** parameter. The optional **variable** and **condition** parameters can be jointly specified to restrict the fact returned by specifying a CLIPS expression that must evaluate to a value other FALSE in order for the fact to be returned. Each fact of the specified deftemplate will be tested until a fact satisfying the condition is found. The fact being tested is assigned to the CLIPS variable specified by the **variable** parameter and may be referenced in the **condition** parameter. If no facts of the specified deftemplate exist, or no facts satisfy the condition, the value null is returned.

The **findAllFacts** methods returns the list of facts associated with the deftemplate construct specified by the **deftemplateName** parameter. The optional **variable** and **condition** parameters can be jointly specified to restrict the facts returned by specifying a CLIPS expression that must evaluate to a value other FALSE in order for a fact to be returned. Each fact of the specified deftemplate will be tested to determine whether it will be added to the list. The fact being tested is assigned to the CLIPS variable specified by the **variable** parameter and may be referenced in the **condition** parameter. If no facts of the specified deftemplate exist, or no facts satisfy the condition, a list with no members is returned.

The **findInstance** methods return the first instance associated with the defclass construct specified by the **defclassName** parameter. The optional **variable** and **condition** parameters can be jointly specified to restrict the instance returned by specifying a CLIPS expression that must evaluate to a value other FALSE in order for the instance to be returned. Each instance of the specified defclass will be tested until an instance satisfying the condition is found. The instance being tested is assigned to the CLIPS variable specified by the **variable** parameter and may be referenced in the **condition** parameter. If no instances of the specified defclass exist, or no instances satisfy the condition, the value null is returned.

The **findAllInstances** methods returns the list of instances associated with the defclass construct specified by the **defclassName** parameter. The optional **variable** and **condition** parameters can be jointly specified to restrict the instances returned by specifying a CLIPS expression that must evaluate to a value other FALSE in order for an instance to be returned. Each instance of the specified defclass will be tested to determine whether it will be added to the list. The instance being tested is assigned to the CLIPS variable specified by the **variable** parameter and may be referenced in the **condition** parameter. If no instances of the specified defclass exist, or no instances satisfy the condition, a list with no members is returned.

7.9.1.5 Executing Functions and Commands

```
public PrimitiveValue eval(String evalStr) throws CLIPSException
```

The **eval** method evaluates the command or function call specified by the **evalStr** parameter and returns the result of the evaluation.

7.9.1.6 Debugging

```
public void watch(String watchItem)
```

```
public void unwatch(String watchItem)
```

```
public boolean getWatchItem(String watchItem)
```

```
public void setWatchItem(String watchItem,boolean newValue)
```

The **watchItem** parameter should be one of the following static String values defined in the Environment class: FACTS, RULES, DEFFUNCTIONS, COMPILATIONS, INSTANCES, SLOTS, ACTIVATIONS, STATISTICS, FOCUS, GENERIC_FUNCTIONS, METHODS, GLOBALS, MESSAGES, MESSAGE_HANDLERS, NONE, or ALL.

The **watch** method enables the specified watch item and the **unwatch** method disables the specified watch item. The **getWatchItem** method returns the current state of the specified watch item. The **setWatchItem** method enables the specified watch item if the **newValue** parameter is true and disables it if the **newValue** parameter is false.

7.9.1.7 Adding and Removing User Functions

```
public void addUserFunction(
    String functionName,
    UserFunction callback)
```

```
public void addUserFunction(
    String functionName,
    String returnTypes,
    int minArgs,
    int maxArgs,
    String argTypes,
    UserFunction callback)
```

```
public void removeUserFunction(
    String functionName)
```

The **addUserFunction** method associates a CLIPS function name (specified by the **functionName** parameter) with an instance of a Java class implementing the **UserFunction** interface (specified by the **callback** parameter). This allows you to call Java code from within CLIPS code. The optional parameters **returnTypes**, **minArg**, **maxArgs**, and **argTypes** can be used to specify the CLIPS primitive types returned by the function, the minimum and maximum number of arguments the function is expecting, and the primitive types allowed for each argument. The UNBOUNDED constant from the **UserFunction** interface can be used for the **maxArgs** parameter to indicate that there is no upper limit on the number of arguments.

If the **returnTypes** parameter value is null, then CLIPS assumes that the UDF can return any valid type. Specifying one or more type character codes, however, allows CLIPS to detect errors when the return value of a UDF is used as a parameter value to a function that specifies the types allowed for that parameter. The following codes are supported for return values and argument types:

Type Code	Type
b	Boolean
d	Double Precision Float
e	External Address
f	Fact Address
i	Instance Address
l	Long Long Integer
m	Multifield
n	Instance Name
s	String
y	Symbol
v	Void—No Return Value
*	Any Type

If the **argTypes** parameter value is null, then there are no argument type restrictions. One or more character argument types can also be specified, separated by semicolons. The first type specified is the default type (used when no other type is specified for an argument), followed by types for specific arguments. For example, "ld" indicates that the default argument type is an integer or float; "ld;s" indicates that the default argument type is an integer or float, and the first argument must be a string; "*,;m" indicates that the default argument type is any type, and the second argument must be a multifield; ";sy;ld" indicates that the default argument type is any type, the first argument must be a string or symbol; and the second argument type must be an integer or float.

The **addUserFunction** method throws an **IllegalArgumentException** if the association fails because one already exists for the **functionName** parameter.

The **removeUserFunction** method removes the association between a CLIPS function name and the user function code associated the function name. You can use this to remove a previously

created associated (either to remove it altogether or to replace the old associated with a new one). The **removeUserFunction** method throws an **IllegalArgumentException** if no association currently exists for the **functionName** parameter.

7.9.1.8 Managing Routers

```
public void addRouter(
    Router theRouter)

public void deleteRouter(
    Router theRouter)

public void activateRouter(
    Router theRouter)

public void deactivateRouter(
    Router theRouter)

public void print (
    String logicalName,
    String printString)

public void print(
    String printString)

public void println (
    String logicalName,
    String printString)

public void println(
    String printString)
```

The **addRouter** method adds an instance of a Java class implementing the **Router** interface to the list of routers checked by CLIPS for processing I/O requests. The **deleteRouter** method removes a Router instance from the list of CLIPS routers. The methods **deactivateRouter** and **activateRouter** allow a router to be disabled/enabled without removing it from the list of routers.

The **print** and **println** methods output the string specified by the **printString** parameter through the router system. These methods direct the output to the logical name specified by the **logicalName** parameter. If the **logicalName** parameter is unspecified, output is directed to standard output. In addition, the **println** method appends a carriage return to the output.

7.9.1.9 Command Loop

```
public void commandLoop()
```

The **commandLoop** method starts the CLIPS Read-Eval-Print Loop (REPL) using the Java standard input and output streams.

7.9.2 The PrimitiveValue Class and Subclasses

```
public abstract class PrimitiveValue

public class VoidValue extends PrimitiveValue

public abstract class NumberValue extends PrimitiveValue

public class FloatValue extends NumberValue

public class IntegerValue extends NumberValue

public abstract class LexemeValue extends PrimitiveValue

public class SymbolValue extends LexemeValue

public class StringValue extends LexemeValue

public class InstanceNameValue extends LexemeValue

public class MultifieldValue extends PrimitiveValue
    implements Iterable<PrimitiveValue>

public class FactAddressValue extends PrimitiveValue

public class InstanceAddressValue extends PrimitiveValue
```

The **PrimitiveValue** class and its subclasses constitute the Java representation of the CLIPS primitive data types. Several methods (such as **eval** and **getSlotValue**) return objects belonging to concrete subclasses of the **PrimitiveValue** class.

Several methods are provided for determining the type of a **PrimitiveValue** object:

```
public CLIPSType getCLIPSType()

public boolean isVoid()

public boolean isLexeme()
```

```

public boolean isSymbol()

public boolean isString()

public boolean isInstanceName()

public boolean isNumber()

public boolean isFloat()

public boolean isInteger()

public boolean isFactAddress()

public boolean isInstance()

public boolean isInstanceAddress()

public boolean isMultifield()

public boolean isExternalAddress()

```

The **getCLIPSType** method returns one of the following **CLIPSType** enumerations: **FLOAT**, **INTEGER**, **SYMBOL**, **STRING**, **MULTIFIELD**, **EXTERNAL_ADDRESS**, **FACT_ADDRESS**, **INSTANCE_ADDRESS**, **INSTANCE_NAME**, or **VOID**.

Several methods are provided for creating objects belonging to the **FloatValue**, **IntegerValue**, **SymbolValue**, **StringValue**, **InstanceNameValue**, **MultifieldValue**, and **VoidValue** classes:

```

public FloatValue()

public FloatValue(double value)

public FloatValue(Double value)

public IntegerValue()

public IntegerValue(long value)

public IntegerValue(Long value)

public SymbolValue()

public SymbolValue(String value)

public StringValue()

```

```

public StringValue(String value)

public InstanceNameValue()

public InstanceNameValue(String value)

public MultifieldValue()

public MultifieldValue(List<PrimitiveValue> value)

public VoidValue()

```

The **assertString** and **makeInstance** methods of the **Environment** class can be used to create objects of the **FactAddressValue** and **InstanceAddressValue** classes respectively.

The following **NumberValue** methods are available for retrieving the underlying Java value from **IntegerValue** and **FloatValue** objects:

```

public Number numberValue()

public int intValue()

public long longValue()

public float floatValue()

public double doubleValue()

```

The following **LexemeValue** method is provided to retrieve the underlying Java value from **SymbolValue**, **StringValue**, and **InstanceNameValue** objects:

```

public String lexemeValue()

```

The **InstanceNameValue** class also provides a method for converting an instance name to the corresponding instance address in a specified environment:

```

public InstanceAddressValue getInstance(Environment theEnv)

```

The following **MultifieldValue** methods provide access to the list of **PrimitiveValue** objects contained in a **MultifieldValue** method:

```

public List<PrimitiveValue> multifieldValue()

public int size()

public PrimitiveValue get(int index)

```

The following **FactAddressValue** methods provide access to the slot values and fact index of the associated CLIPS fact:

```
public PrimitiveValue getSlotValue(String slotName)

public long getFactIndex ()
```

The following **InstanceAddressValue** methods provide access to the slot values and instance name of the associated CLIPS instance:

```
public PrimitiveValue getSlotValue(String slotName)

public String getInstanceName()
```

The following **ExternalAddressValue** method provides access to the value of the associated CLIPS external address (a C pointer converted to a Java long integer):

```
public long getExternalAddress()
```

The **FactAddressValue**, **InstanceAddressValue**, and **ExternalAddressValue** classes provide the following reference count methods:

```
public void retain()

public void release()
```

Since objects of these classes retain pointers to C data structures, retaining the object prevents the C code from releasing these data structures while there are still outstanding references to them. Each call to the **retain** method increments the number of reference counts to the object and each call to the **release** method decrements the number of reference counts to the object.

7.9.3 The CLIPSEException and CLIPSLoadException Classes

```
public class CLIPSEException extends Exception

public class CLIPSLoadException extends CLIPSEException
```

CLIPSJNI provides two subclasses of the **Exception** class for methods generating errors: **CLIPSException** and **CLIPSLoadException**.

7.9.3.1 CLIPSLoadException Methods

```
public List<CLIPSLineError> getErrorList()
```

```
public class CLIPSLineError
```

Loading constructs can generate multiple errors, so the **getErrorList** method of the **CLIPSLoadException** class returns the list of **CLIPSLineError** objects detailing each error.

7.9.3.1.1 CLIPSLineError Methods

```
public String getFileName()
```

```
public long getLineNumber()
```

```
public String getMessage()
```

The **getFileName**, **getLineNumber**, and **getMessage** respectively return the file name, line number, and error message associated with a **CLIPSLineError** object.

7.9.4 The Router Interface

```
public interface Router
```

The **Router** interface allows Java objects implementing the interface to interact with the CLIPS I/O router system.

7.9.4.1 Required Methods

```
public int getPriority()
```

```
public String getName()
```

```
public boolean query(String logicalName)
```

```
public void write(String logicalName,String writeString)
```

```
public int read(String logicalName)
```

```
public int unread(String logicalName,int theChar)
```

```
public void exit(boolean failure)
```

The **getPriority** method returns the integer priority of the router. Routers with higher priorities are queried before routers of lower priority to determine if they can process an I/O request. The **getName** method returns the identifier associated with the router. The **query** method is called to determine if the router handles I/O requests for the **logicalName** parameter. It should return true if the router can process the request, otherwise it should return false. The **write** method is called to output the value specified by the **writeString** parameter to the **logicalName** parameter. The **read** method returns an input character for the **logicalName** parameter. It should return -1 if no

characters are available in the input queue. The **unread** method places the character specified by parameter **theChar** back on the input queue so that it is available for the next **read** request. It returns the value of parameter **theChar** if successful, otherwise it returns -1. The **exit** method is invoked when the CLIPS **exit** command is issued or an unrecoverable error occurs. The **failure** parameter will either be false for an exit command or true for an unrecoverable error.

7.9.4.2 Predefined Router Names

```
public static final String STDIN

public static final String STDOUT

public static final String STDWRN

public static final String STDERR
```

The String constants STDIN, STDOUT, STDWRN, and STDERR are the standard predefined logical names used by CLIPS.

7.9.4.3 The BaseRouter Class

```
public class BaseRouter implements Router
```

The **BaseRouter** class is an implementation of the **Router** interface. Its **write**, **read**, **unread**, and **exit** methods are minimal implementations; the **write** and **exit** methods execute no statements and the **read** and **unread** methods always return -1. Subclasses can override these methods as needed to create functional routers.

7.9.4.3.1 Constructors

```
public BaseRouter (
    Environment env,
    String [] queryNames)

public BaseRouter(
    Environment env,
    String [] queryNames,
    int priority)

public BaseRouter(
    Environment env,
    String [] queryNames,
    int priority,
    String routerName)
```

The **BaseRouter** constructor requires the **env** and **queryName** parameters. Optionally, the **priority** parameter or the **priority** and **routerName** parameters can be supplied. The **env** parameter is the **Environment** object associated with the **Router** object. The **queryNames** parameter is an Array of strings used by the **query** method of the **BaseRouter** object to determine whether the router handles I/O for a specific logical name. The **priority** parameter is the priority of the router; if it is unspecified, it defaults to 0. The **routerName** parameter is the name that serves as an identifier for the **BaseRouter** object; if it is unspecified an identifier will be generated for the router.

7.9.5 The UserFunction Interface

```
public interface UserFunction
```

The **UserFunction** interface provides a method for invoking a Java method from CLIPS code.

7.9.5.1 Required Methods

```
public PrimitiveValue evaluate (List<PrimitiveValue> arguments)
```

Once a linkage has been made between a CLIPS function name and an object implementing the **UserFunction** interface, the **evaluate** method is invoked when the linked CLIPS function call is executed. The **function** arguments are evaluated and passed to the **evaluate** method via the **arguments** parameter.

7.9.5.2 Constants

```
public static final int UNBOUNDED
```

The **UNBOUNDED** constant can be used for the **maxArgs** parameter of the **AddUserFunction** method of the **Environment** class to indicate that there is no upper limit on the number of arguments.

7.9.6 Examples

The following examples assume the example code is placed in the top-level CLIPSJNI directory. Additionally the native libraries must be built and present in the directory (either **libCLIPSJNI.jnilib** for macOS, **CLPSJNI.dll** for Windows, or **libCLIPSJNI.so** for Linux). The CLIPSJNI Java source files should also be compiled using the appropriate command for Windows, macOS, or Linux:

```
make -f makefile.win clipsjni
make -f makefile.mac clipsjni
make -f makefile.lnx clipsjni
```

7.9.6.1 Loading Constructs from a JAR file

This example demonstrates how to load a CLIPS source file that has been stored inside a JAR file.

First, create the source file `hello.clp` within the `CLIPSJNI` directory with the following content:

```
(defrule hello
  =>
  (println "Hello World"))
```

Next create the Java source file `Example.java` with the following content:

```
import net.sf.clipsrules.jni.*;

public class Example
{
    public static void main(String args[])
    {
        Environment clips;

        clips = new Environment();

        try
        {
            clips.loadFromResource("/hello.clp");
            clips.reset();
            clips.run();
        }
        catch (Exception e)
        { e.printStackTrace(); }
    }
}
```

Next, compile the Java source and create a jar file to contain the `Example` class, the `CLIPSJNI` classes, and the CLIPS construct file:

```
$ javac -cp CLIPSJNI.jar Example.java
$ jar -cfe Example.jar Example Example.class
$ jar -uf Example.jar -C bin/clipsjni net
$ jar -uf Example.jar hello.clp
$
```

Finally, run the program:

```
$ java -jar Example.jar
Hello World
$
```


7.9.6.2 Fact Query

This example demonstrates how to query CLIPS to retrieve facts.

First create the Java source file `Example.java` with the following content:

```
import net.sf.clipsrules.jni.*;

import java.util.List;

public class Example
{
    public static void main(String args[])
    {
        Environment clips;

        clips = new Environment();

        try
        {
            clips.build("(deftemplate person (slot name) (slot age))");

            clips.assertString("(person (name \"Fred Jones\") (age 17))");
            clips.assertString("(person (name \"Sally Smith\") (age 23))");
            clips.assertString("(person (name \"Wally North\") (age 35))");
            clips.assertString("(person (name \"Jenny Wallis\") (age 11))");

            System.out.println("All people:");

            List<FactAddressValue> people = clips.findAllFacts("person");

            for (FactAddressValue p : people)
            { System.out.println("    " + p.getSlotValue("name")); }

            System.out.println("Adults:");

            people = clips.findAllFacts("?f", "person", "(>= ?f:age 18)");

            for (FactAddressValue p : people)
            { System.out.println("    " + p.getSlotValue("name")); }

        }
        catch (Exception e)
        { e.printStackTrace(); }
    }
}
```

Next, compile the Java source and create a jar file to contain the `Example` and `CLIPSJNI` classes:

```
$ javac -cp CLIPSJNI.jar Example.java
$ jar -cfe Example.jar Example Example.class
$ jar -uf Example.jar -C bin/clipsjni net
$
```

Finally, run the program:

```
$ java -jar Example.jar
All people:
  "Fred Jones"
  "Sally Smith"
  "Wally North"
  "Jenny Wallis"
Adults:
  "Sally Smith"
  "Wally North"
$
```

7.9.6.3 Big Integer Multiplication User Function

This example demonstrates how to add a user function to multiply two numbers together using big integer math. It also demonstrates using the eval method to evaluate a CLIPS function call.

First create the Java source file Example.java with the following content:

```
import net.sf.clipsrules.jni.*;

import java.util.List;
import java.math.BigInteger;

public class Example
{
    public static void main(String args[])
    {
        Environment clips;

        clips = new Environment();

        clips.addUserFunction("bi*", "s", 2, Router.UNBOUNDED, "s",
            new UserFunction()
            {
                public PrimitiveValue evaluate(List<PrimitiveValue> arguments)
                {
                    LexemeValue lv = (LexemeValue) arguments.get(0);
                    BigInteger rv = new BigInteger(lv.lexemeValue());

                    for (int i = 1; i < arguments.size(); i++)
                    {
                        lv = (LexemeValue) arguments.get(i);
                        rv = rv.multiply(new BigInteger(lv.lexemeValue()));
                    }

                    return new StringValue(rv.toString());
                }
            });
    }
}
```

```

try
{
    System.out.println("( * 9 8) = " +
        clips.eval("( * 9 8)"));
    System.out.println("(bi* \"9\" \"8\") = " +
        clips.eval("(bi* \"9\" \"8\")"));
    System.out.println("( * 4294967296 4294967296) = " +
        clips.eval("( * 4294967296 4294967296)"));
    System.out.println("(bi* \"4294967296\" \"4294967296\") = " +
        clips.eval("(bi* \"4294967296\" \"4294967296\")"));
}
catch (Exception e)
{ e.printStackTrace(); }
}
}

```

Next, compile the Java source and create a jar file to contain the Example and CLIPSJNI classes:

```

$ javac -cp CLIPSJNI.jar Example.java
$ jar -cfe Example.jar Example Example*.class
$ jar -uf Example.jar -C bin/clipsjni net
$

```

Finally, run the program:

```

$ java -jar Example.jar
(* 9 8) = 72
(bi* "9" "8") = "72"
(* 4294967296 4294967296) = 0
(bi* "4294967296" "4294967296") = "18446744073709551616"
$

```

7.9.6.4 Get Properties User Function

This example demonstrates how to add a user function that returns a multifield value containing the list of system properties.

First create the Java source file Example.java with the following content:

```

import net.sf.clipsrules.jni.*;

import java.util.ArrayList;
import java.util.List;
import java.util.Properties;

public class Example
{
    public static void main(String args[])
    {
        Environment clips;

        clips = new Environment();
    }
}

```

```

        clips.addUserFunction("get-properties","m",0,0,null,
            new UserFunction()
            {
                public PrimitiveValue evaluate(List<PrimitiveValue> arguments)
                {
                    List<PrimitiveValue> values = new ArrayList<PrimitiveValue>();

                    Properties props = System.getProperties();
                    for (String key : props.stringPropertyNames())
                        { values.add(new SymbolValue(key)); }

                    return new MultifieldValue(values);
                }
            });

        clips.commandLoop();
    }
}

```

Next, compile the Java source and create a jar file to contain the Example and CLIPSJNI classes:

```

$ javac -cp CLIPSJNI.jar Example.java
$ jar -cfe Example.jar Example Example*.class
$ jar -uf Example.jar -C bin/clipsjni net
$

```

Finally, run the program:

```

$ java -jar Example.jar
      CLIPS (6.4.2 1/14/25)
CLIPS> (get-properties)
(java.runtime.name sun.boot.library.path java.vm.version gopherProxySet
java.vm.vendor java.vendor.url path.separator java.vm.name file.encoding.pkg
user.country sun.java.launcher sun.os.patch.level java.vm.specification.name
user.dir java.runtime.version java.awt.graphicsenv java.endorsed.dirs os.arch
java.io.tmpdir line.separator java.vm.specification.vendor os.name
sun.jnu.encoding java.library.path java.specification.name java.class.version
sun.management.compiler os.version http.nonProxyHosts user.home user.timezone
java.awt.printerjob file.encoding java.specification.version user.name
java.class.path java.vm.specification.version sun.arch.data.model java.home
sun.java.command java.specification.vendor user.language awt.toolkit java.vm.info
java.version java.ext.dirs sun.boot.class.path java.vendor file.separator
java.vendor.url.bug sun.cpu.endian sun.io.unicode.encoding socksNonProxyHosts
ftp.nonProxyHosts sun.cpu.isalist)
CLIPS> (exit)
$

```

Appendix A:

Support Information

A.1 Questions and Information

The URL for the CLIPS Web page is clipsrules.net.

Questions regarding CLIPS can be posted to one of several online forums including the CLIPS Expert System Group, groups.google.com/group/CLIPSESG/, the SourceForge CLIPS Forums, sourceforge.net/forum/?group_id=215471, and Stack Overflow, stackoverflow.com/questions/tagged/clips.

Inquiries related to the use or installation of CLIPS can be sent via electronic mail to support@clipsrules.net.

A.2 Documentation

The CLIPS Reference Manuals and other documentation are available at clipsrules.net/Documentation.html.

Adventures in Rule-Based Programming is a fun introduction to writing applications using CLIPS. In this tutorial you'll learn the basic concepts of rule-based programming, where rules are used to specify the logic of what must be accomplished, but an inference engine determines when rules are applied. You'll incrementally create a fully functional text adventure game, and in the process, learn how to write, organize, debug, test, and deploy CLIPS code. Visit clipsrules.net/airbp for more information.

Expert Systems: Principles and Programming, 4th Edition, by Giarratano and Riley comes with a CD-ROM containing CLIPS 6.22 executables (DOS, Windows XP, and Mac OS), documentation, and source code. The first half of the book is theory oriented and the second half covers rule-based, procedural, and object-oriented programming using CLIPS.

A.3 CLIPS Source Code and Executables

CLIPS executables and source code are available on the SourceForge web site at sourceforge.net/projects/clipsrules/files.

Appendix B:

Update Release Notes

The following changes were introduced in version 6.4.1 of CLIPS.

- **Compiler Support** - The following compilers are now supported.
 - Xcode 14.3.
 - Microsoft Visual Studio Community 2022.

The following changes were introduced in version 6.4.2 of CLIPS.

- **Compiler Support** - The following compilers are now supported.
 - Xcode 16.2.

Index

activateRouter	78	Exit	55
ActivateRouter	49	ExternalAddressValue	51
addRouter	78	FactAddressValue	51, 79
AddRouter	49	findAllFacts	74
addUserFunction	76	FindAllFacts	46
AddUserFunction	48	findAllInstances	75
Advanced Programming Guide	vii	FindAllInstances	46
AnimalFormsExample	41	findFact	74
AnimalWPFExample	41	FindFact	46
assertString	74	findInstance	74
AssertString	45	FindInstance	46
AutoFormsExample	41	FloatValue	50, 52, 79, 80
AutoWPFExample	41	getCLIPSType	79
BaseRouter	55, 84	getErrorList	82
Basic Programming Guide	vii	getExternalAddress	82
build	73	getFactIndex	82
Build	45	getFileName	83
clear	73	GetInstance	53
Clear	44	getInstanceName	82
CLIPSCLRWrapper	41	getLineNumber	83
CLIPSException	45, 54, 82	getMessage	83
CLIPSJNI	38	getName	83
CLIPSLoadException	45, 54, 82	getPriority	83
CLIPSType	51	getSlotValue	82
commandLoop	79	GetSlotValue	53
CommandLoop	50	getWatchItem	76
deactivateRouter	78	GetWatchItem	47
DeactivateRouter	50	InstanceAddressValue	51, 79
deleteRouter	78	InstanceNameValue	51, 52, 79, 81
DeleteRouter	49	IntegerValue	50, 52, 79, 80
DLL	38	Interfaces Guide	vii
DLLExample	39	isExternalAddress	80
Environment	44, 73	IsExternalAddress	52
eval	76	isFactAddress	80
Eval	47	IsFactAddress	51
evaluate	85	isFloat	80
exit	83	IsFloat	51

isInstance.....	80	Reference Manual	vii
IsInstance	51	removeUserFunction	76
isInstanceAddress.....	80	RemoveUserFunction.....	48
IsInstanceAddress	51	reset	73
isInstanceName	80	Reset	45
IsInstanceName	51	Router	54, 83
isInteger.....	80	RouterFormsExample.....	41
IsInteger	51	RouterWPFExample.....	41
isLexeme	79	run.....	74
IsLexeme	51	Run	45
isMultifield.....	80	setWatchItem.....	76
IsMultifield.....	51	SetWatchItem	47
isNumber	80	STDERR.....	55, 84
IsNumber.....	51	STDIN	55, 84
isString	80	STDOUT	55, 84
IsString	51	STDWRN	55, 84
isSymbol.....	80	StringValue.....	51, 52, 79, 80
IsSymbol	51	SymbolValue.....	51, 52, 79, 80
isVoid	79	UNBOUNDED.....	85
IsVoid.....	51	unread	83
LexemeValue	51, 79	Unread	55
load.....	73	unwatch	76
Load	44	Unwatch	47
loadFromResource	73	User's Guide.....	vii
LoadFromResource	45	UserFunction	56, 85
loadFromString	73	VoidValue	50, 52, 79, 81
LoadFromString	44	watch	76
makeInstance.....	74	Watch	47
MakeInstance	45	WineFormsExample.....	41
MultifieldValue	51, 52, 79, 81	WineWPFExample.....	41
NumberValue	50, 79	WrappedDLL	38
PrimitiveValue	50, 79	WrappedDLLExample	39
print	78	WrappedLib.....	38
println.....	78	WrappedLibExample	39
query.....	83	write.....	83
Query.....	55	Write.....	50, 55
read.....	83	WriteLine	50
Read	55		