Integrating Commercial off-the-shelf (COTS) Graphics and Extended Memory packages with CLIPS

Andres C. Callegari

Computer Sciences Corporation
16511 Space Center Blvd.
Houston, Texas 77508

Abstract

This paper addresses the question of how to mix CLIPS with graphics and how to overcome PC's memory limitations by using the extended memory available in the computer. By adding graphics and extended memory capabilities, CLIPS can be converted into a complete and powerful system development tool, on the most economical and popular computer platform. New models of Pcs have amazing processing capabilities and graphic resolutions that cannot be ignored and should be used to the fullest of their resources. CLIPS is a powerful expert system development tool, but it cannot be complete without the support of a graphics package needed to create user interfaces and general purpose graphics, or without enough memory to handle large knowledge bases. Now, a well known limitation on the PCs is the usage of real memory which limits CLIPS to use only 640 Kb of real memory, but now that problem can be solved by developing a version of CLIPS that uses extended memory. The user has access of up to 16 MB of memory on 80286 based computers and, practically, all the available memory (4 GB) on computers that use the 80386 processor. So if we give CLIPS a self-configuring graphics package that will automatically detect the graphics hardware and pointing device present in the computer, and we add the availability of the extended memory that exists in the computer (with no special hardware needed), the user will be able to create more powerful systems at a fraction of the cost and on the most popular, portable, and economic platform available such as the PC platform.

I. Introduction

Programmers who use CLIPS (C Language Integrated Production System) to design large PC applications with or without graphics have encountered the problem of being left with insufficient memory to run the application in a guaranteed and productive way.

This memory problem does not come as a surprise considering that DOS normally uses 640 KB of RAM to allocate the operating system, drivers, buffers, TSRs (Terminate and Stay Resident programs), and for loading and executing programs. DOS memory limitation constitutes a barrier that impedes applications to use the full potential of CLIPS and the standard features of the new generation of PCs such as: extended memory, higher resolution graphics cards and displays, etc. It is important to realize that graphics and image manipulation are usually memory intensive, and that CLIPS memory requirement varies according to the size of the knowledge base used.

Now, PC's are the most popular, portable, accessible, and every time more powerful computer platform available, and it will be a shame that having such excellent hardware power and software development tools such as CLIPS and high quality off-the-shelf software packages, there should still be problems using or developing large PC's applications.

Fortunately for us, two events have happened. The first event is that one of CLIPS blueprint goals was to create a highly portable and low-cost expert system tool that could be easily combined with external systems. This goal facilitates the integration of CLIPS with any external software package(s). The second event is the fact that the PC software market has been flooded with high quality off-the-shelf software packages. These software packages has been written for almost every need, and by using the right combination of software tools (off-the-shelf graphics packages and

DOS memory extenders packages), the problems of using and creating large applications utilizing CLIPS with or without graphics can be solved.

II. CLIPS Problem Areas

The solutions to graphics and memory problems that arise when developing large applications using CLIPS and applications mixing CLIPS with off-the-shelf graphics packages can be grouped into three main areas: CLIPS and extended memory, CLIPS and graphics, and CLIPS using graphics and extended memory. One of several ways to solve each of these problems will be analyzed next. All these solutions have been implemented, used, and tested in an application or demo.

1. CLIPS and Extended Memory

Many PC programmers using CLIPS to build their applications find that they run out of memory while designing, testing, or executing their programs. Once this problem occurs, the only thing left is to restructure the application in order to optimize memory usage. But, as painful as it sounds, sometimes there is no way around. Sometimes, the knowledge base becomes too big, and CLIPS will not have enough memory to operate. In most cases, the only solution is to have access to more memory, but, luckily for us, there are straight and easy solutions for these kind of problems.

1.1 Using extended memory

On the PCs, CLIPS runs on computers using the old Intel 8086 chip as CPU or using a chip which can emulate the operation of this chip. When a program runs on a chip using this emulation mode, it is said that the program is running in **real mode**. Now, the new family of Intel chips (80286, 80386, and 80486 chips) were designed with two working modes (dual-mode chips). The first mode provided full compatibility with older chips so that existing programs will still run in the new computers. The second mode was designed to give on-chip memory management, task management, and protection tools to new and more powerful operating systems (multitasking and multiuser operating systems). When a program runs in the second mode, it is said that the program is running in **protected mode**. Rules for programs running in protected mode are more strict than those programs running in real mode. Protected mode was designed to support multitasking and multiuser systems, so direct access to the hardware and to the operating system has to be restricted in order to eliminate any possible interference with other running processes or with the operating system itself. A crucial advantage of a program running in protected mode is that it gains access to all the extended memory available in the computer.

Normally, when CLIPS runs in real mode, DOS will provide CLIPS with specific services: input/output, file system management, memory management, processor management, etc. In general, all programs will request any of these services from DOS or will bypass DOS and access the hardware directly.

Now, A.I. Architects, Inc. created a very interesting software package which provides to a program running in protected mode (and, therefore, able to access directly all the extended memory available in the computer) with all the services that DOS normally gives to a program running in real mode. This approach permits programs running on 80286 systems a direct memory addressing of 16 MB with 64 KB segments. On 80386 systems, the program can directly access up to 4 GB, with segment sizes as large as the memory installed in the computer.

1.2 Processing CLIPS

If the A.I. Architects package is installed in our compiler package (there is a large list of compilers and assemblers supported) and CLIPS source code is correctly processed, CLIPS will be able to run in protected mode. With CLIPS being able to run in protected mode, CLIPS will have access to all the extended memory available in the computer (15 MB on 80286 systems and 4 GB on 80386 systems). With access to extended memory, CLIPS will be able to handle large knowledge base systems; moreover, the size of the knowledge base that CLIPS could handle will depend on the amount of extended memory available in the computer.

In general, to make a C, assembly language, or FORTRAN program run in protected mode will normally imply the following steps: compiling or assembling the program (.OBJ), linking with the special patch libraries provided for each compiler brand, and maybe postprocessing it by using a special program which creates the final protected mode executable. After these steps are performed, one should load the kernel and load the program

into protected mode by using a special real-mode program called loader, which tells the kernel to manage and load the program into protected mode.

This enhanced version of CLIPS does not have the memory limitations and problems that CLIPS and PC users have suffered for so long. From now on, CLIPS will be able to fully use the extended memory normally available in the new powerful generation of machines found in today's market (machines based on Intels' 80386 and 80486 chips). Powerful and highly productive expert systems can be built at a very low cost, and they will be able to use all the graphics power, portability, low cost, and availability characteristic of PC platform's machines.

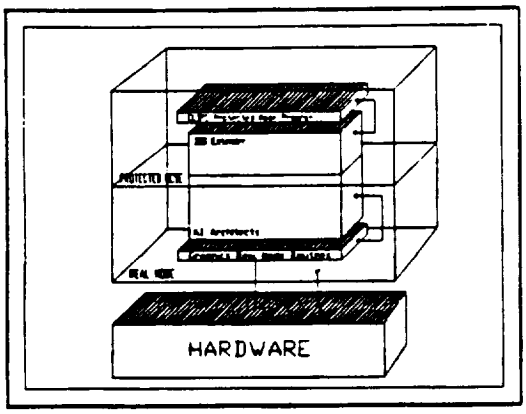

Figure 1. All graphics routines are run in real mode; CLIPS is run in protected mode, and the DOS extender provides the communication links between protected and real mode.

1.3 8086 Emulation

There is another solution which is not as complete as the one discussed before but is very simple to use and to implement. This option is only available for 80386 based systems, the 80386 chip has a virtual V86 mode, which emulates real-mode of an 8086 or 80286 in virtual address space. This emulation permits specially processed executables to run in virtual V86 mode and to use direct addressing in the device space. This approach gives the processed executable a total linear addressing space of 1 MB of RAM. Thus, if CLIPS is properly processed, it will have the capability to directly address up to 1 MB of memory.

One of the biggest advantages of this method is that the operating system, TSRs, and drivers will all run in real mode, so the application has a whole 1 MB linear address space for itself to use. Spawn processes don't take memory from the program, since each spawn process generates a new virtual V86 1 MB linear address space for the new process to use. Another advantage of this mode is that each process runs in a real-mode emulation, which means, they do not have the restrictions imposed by protected mode; they can bypass the operating system and access the hardware directly.

In a few words, a program running in real mode can have only 450 KB or less of free RAM memory left for execution. The operating system, TSRs, buffers, drivers, devices, etc. coexist in the same linear address, while a process running in V86 mode uses practically 1 MB of RAM exclusively for its execution and use.

In order for a program to be successfully processed, it must not used any unsupported DOS calls, and the programs should not be tied to specific physical addresses. The beauty of this solution is that the executable (.exe) can be processed, and there is no need to have the source code.

1.4 Performance

Now, the performance of a program running in a 80386 CPU in protected mode is faster than when it is run in real mode. In a 80286 based system, the performance is slightly slower because the 80286 chip needs to be reset (logic reset) every time it switches from real mode to protected mode, and it requires several overhead calls in order to return control to the running program (shutdown logic).

1.5 Restrictions

When a program runs in protected mode, it is subjected to more restrictions. First, the access to physical memory is no longer direct; in this case, indexes to descriptor tables are used instead of addresses. Access to the physical address is made through these descriptor tables when paging is not enable, and the segment register contains a symbolic representation of the address called selector.

A second difference is that memory can not be allocated in an arbitrary way. Third, one can not write to a code segment, and one can not write past the end of a segment. Fourth, a program can not interfere with the operating system. This protection is implemented to keep the operating system in optimal and healthy conditions at all times. These

restrictions are necessary because 80286, 80386, and 80486 chips are design to support multitasking and multiuser operating systems.

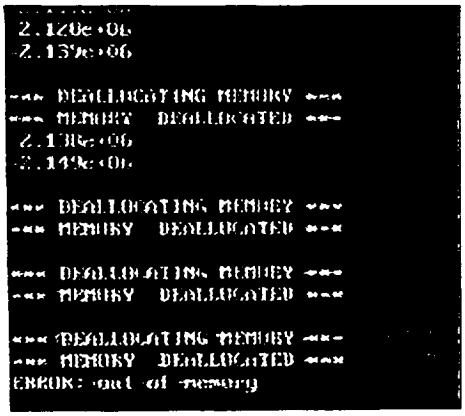

Figure 2. After running a memory exhaustive test program, CLIPS issued a memory allocation error message after using 2.1 MB. of extended memory.

In Figure 2, there is a picture of a CLIPS program processed so that it can run in protected mode. The CLIPS source program being run from the processed CLIPS executable has been designed to exhaust all the extended memory available in the computer. This test program continuously created CLIPS data forcing CLIPS to request more memory from the operating system until the system run out of memory. The picture shows that CLIPS requested 2.1 MB of memory from the operating system before the system run out of memory.

1.6 Limitations

The EMACS-style editor could not be used. It's code seems to violate some of the restrictions, discussed earlier, imposed over programs running in protected mode. However, one can create a user-defined function to call another editor until a cure is found. A redefinition of the "system" command is necessary. From now on, spawning is reserve for executable files only (.exe) not command files (.com). This means that in order clear the screen, one can not use the command [system "cls"] anymore. The solution is to create a small routine to clear the screen and added to the user defined functions. All of these problems can be fixed in the future, but it is very important to notice that unmodified CLIPS source code is being used and mixed with the A.I. Architects DOS memory extender package.

1.7 Conclusions

The ability of being able to run CLIPS in protected mode and being able to access all the extended memory available in the computer permits the application programmer to create large applications that can handle large knowledge bases. The new generation of PCs based on the Intel 80386 chips have processing speeds near the 8 MIPS mark, and computers based on the 80486 chip have speed around the 15 MIPS bench mark. With CLIPS breaking the PC DOS memory barrier which constrained CLIPS from being used to develop large PC applications, CLIPS will now be able to use all the power and portability of the new PC generations. Now, for example, powerful and complete expert systems can be run in a small but powerful laptop computer, which can be taken and run on practically any possible physical environment. This combination of performance, portability, graphics power, plus the intrinsic capabilities of CLIPS is what CLIPS programmers have been awaiting for. PCs are very powerful and fast but if the operating system can't give programs enough memory to work with, then all the good qualities and power of the PCs are useless. From now on, the situation is different; applications can use large amounts of memory and can use all the new features of the PC's (extended memory, higher resolution graphics cards, mass storage, etc.).

2. CLIPS and Graphics

Sometimes an application needs to express some or all of its output information in a graphical form or needs to have a specialized graphical user interface to interact with the user (icon menus, cascade menus, dialog windows, etc.).

In the following paragraphs, two ways of mixing CLIPS with graphics will be discussed. The first method is to mix CLIPS and two graphical packages. In this first case, a driving program controls the execution of the routines. The second method consists in embedding graphics package(s) into CLIPS and to define a complete set of user-defined graphics functions into CLIPS. That is, adding to the original CLIPS language a complete set of graphics commands so that any graphic output or image manipulation process can be performed by issuing commands from these extended language set. Each of this methods have their own advantages and disadvantages.

2.1 Embedding CLIPS (First method)

CLIPS was designed so that it can be embedded within other applications; therefore, when this happens, it needs a driving program which calls CLIPS as a subroutine. This driving program controls CLIPS activation and normally can control most of the graphics output of the application.

CLIPS can interact and interchange data with the driving program in many ways: declaring user-defined functions, passing variables from CLIPS into external functions, passing data from external functions to CLIPS, etc. It is very easy to integrate CLIPS with external functions, which gives CLIPS the capability to execute user-defined graphics commands (C language, etc.) whenever it is needed. In this way, both the controlling program and CLIPS will be able to process, modify, or send graphical information to the screen.

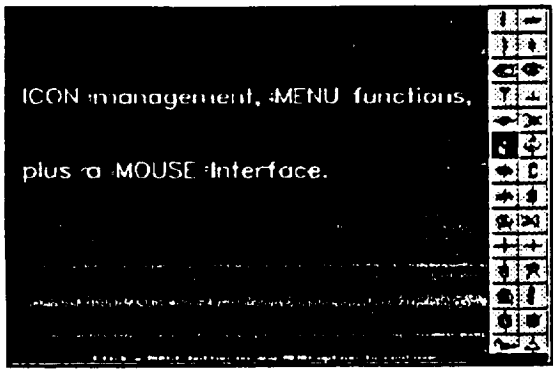

Figure. 3 This is a menu created by a extended graphics CLIPS command. It displays an icon menu activated by the mouse, and it uses all available extended memory.

2.2 Using off-the-shelf packages

There are many off-the-shelf graphics packages that can be used. Two of them (one from Metagraphics Software Corporation and one from Ithaca Street Software, Inc.) have excellent graphics packages that combined provide the following features: complete graphics environment support, a complete set of utilities for developing multi-window desktop applications, independence over graphics peripherals, icon manipulation routines, plus a complete and powerful set of graphics drawing functions. These features provide most of the tools needed to build any kind of graphical information, graphical objects, and complete user interfaces. These software provide most of the necessary routines needed to build higher user interface tools like pop-up menus, windows, image processing routines (frame animation, etc.), icon manipulation, automatic graphics hardware detection of graphics cards and mouse, etc.

The result of combining CLIPS with the graphics tools provided by these software packages is a **complete and powerful set of software development tools.** Computer Sciences Corporation created for NASA an application which mixes these graphics packages (from Metagraphics Software Corporation and Ithaca Street Software, Inc.) with CLIPS using Borland's C compiler as the blending environment.

Figure 4 shows a screen of the application which was develop using CLIPS 4.3 and the tools provided by the packages described above. A complete user interface (popup menus, Icon menus, help windows, etc.) and automatic hardware detection capabilities were created or provided by the former packages. In addition to this, a set of specialized graphics functions aimed to manipulate graphical objects on the screen were built too.

2.3 CLIPS Graphics Version (Second Method)

In the second method, CLIPS possesses all the graphical capabilities to create and manipulate (using its new set of graphical language commands) any graphical object on the screen: menus, image manipulation, icon manipulation, graphics functions, etc. In Figure 2, 5, and 6, there are examples of applications that use all the extended memory available in the computer and that use a mouse to activate the icon menus. These icon menus were created using the new set of CLIPS graphics commands (icon management and graphics environment provided by Metagraphics Software Corporation and Ithaca Street Software, Inc.). When an option is chosen, a fact specifying the chosen option will be asserted into the CLIPS fact list. Figure 5 gives a demonstration of text management, size, and the different kind of fonts available in the extended graphics CLIPS version.

The advantages of this method is that all programs will be written as part of an extended CLIPS language, they will run in interactive mode (easy to maintain, perform tests, or debug), and they will not need a driving program. The best part is that after modifying the code, there is no need of recompiling or relinking the program. This will give the expert system total and continuous control over the

process. Sometimes, if there is a driving program(s), information has to be passed to CLIPS to update any change in the state of the system that happened while the driving program was in control.

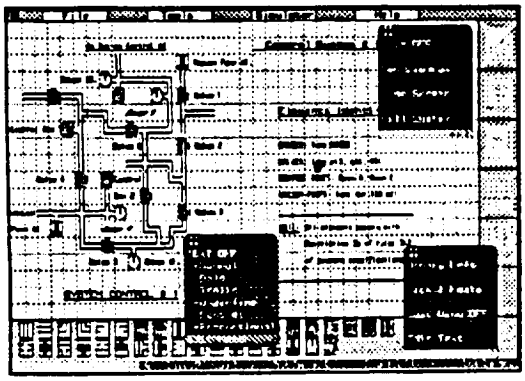

Figure 4. This figure shows the display of an ICAT system created by mixing CLIPS and graphics in a C environment.

Graphics commands behave and are issued exactly like any other CLIPS internal command, and rules containing graphics commands will behave like any other rule does.

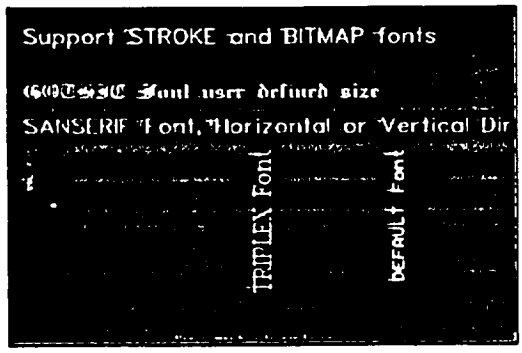

Figure 5. This figure shows font management and the available fonts (provided by Borlands C compiler) used in the extended memory/graphics CLIPS version.

2.4 Performance

The applications described above were tested on a PC running at 25 Mhz., 100 ns RAM memory, with coprocessor, and with a VGA card/monitor. The applications didn't have any problems in what speed pertains; CLIPS, the user interface, and the graphics responses run smoothly and pleasantly fast.

2.5 Memory Limitations

For systems that use CLIPS and a moderate amount of graphics (does not need a complete user interface or image manipulation routines), these graphics packages will provide the perfect development solution, and the application will almost have the same limitations as a normal CLIPS program (just a little less memory free for CLIPS).

The application in Figure 4 possesses a complete graphical user interface, works on graphical objects, and does a lot of image manipulation. Therefore, it is anticipated that after loading the program, there will not be much memory left for CLIPS to work with. This fact directly implies that there will be a strict limitation in the size of knowledge base that can be loaded and/or used by CLIPS.

If the knowledge base consumes most of the free memory left in the computer, then it will be very probable that CLIPS will run out of memory at run time. This is why an application using CLIPS and intensive graphics can not run in a guaranteed (knowledge base can grow and consume all the memory) and productive way (if the knowledge base is limited to a certain size, the application main goal will be restricted too).

2.6 Conclusions

Thanks to CLIPS special features and design, CLIPS can be easily integrated with off-the-shelf's graphics packages. These added graphics capabilities give CLIPS the power to express output in graphical form, which is needed in a large number of applications (simulations, training, charting, etc.), or in those applications that need a specialized graphical user interface (image manipulation, icon menus, etc.).

For large applications that use CLIPS and intensive graphics manipulations, a second package (DOS memory extender) has to be added. This package will permit CLIPS to run in protected mode and the graphics part to run in real mode. In this way, CLIPS knowledge base can grow as big as it needs and the graphics part of the application will have enough memory to operate without any problems.

3. CLIPS, Graphics, and Extended Memory

3.1 Problems

Developing an application that uses CLIPS in extended memory and graphics involves a deeper understanding of how real mode and protected mode work. First, off-the-shelf graphics packages provide only libraries and object files. Most packages do not provide the source code; therefore, it will not be possible to process the code so that it can run in protected mode. Second, there are software whose code access directly the hardware (direct screen write, etc.); protected mode will not let these programs access the hardware directly.

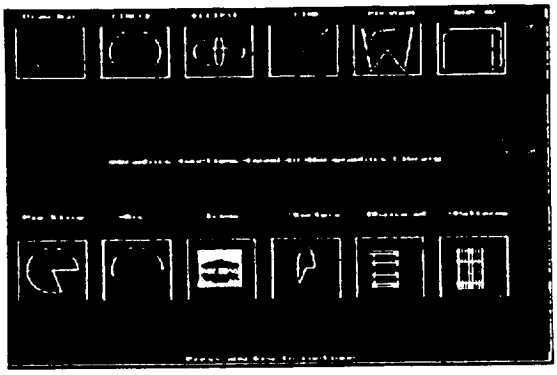

Figure 6. This is a sample of some graphics features available in the extended memory/graphics version of CLIPS.

3.2 Solution

The solution consists in running the hardware dependent routines in real mode, where they can access the hardware directly, and to run all non hardware dependent code in protected mode. A.I. Architects developed mechanisms for interprocessor communication. A routine running in protected mode can pass data to a routine in real mode which will process the data and will return data to the protected mode application. There are two more ways how a real procedure can communicate with a protected-mode application. The real procedure can signal a protected-mode handler, or one can make use of interrupts.

If the application is going to use graphics (graphics run faster if the graphics routines can access the hardware directly), the best solution will be to run all the graphics routines or hardware dependent routines in real mode. The required communication links will be established with the protected mode application so that the application can issue any graphics command. Figures 3, 5, and 6, show screens of an application created by the extended graphics version of CLIPS. This version of CLIPS runs graphics in real mode and runs all other CLIPS routines in protected mode. When CLIPS needs to issue a graphics command, it will make a call to the real procedure and will pass the needed data so that the real procedure can execute the graphics commands. This extended CLIPS version that includes graphics and extended memory was built using packages from Borland, A.I. Architects Inc., Metagraphics Software Corporation, and from Ithaca Street Software Inc.

3.3 Limitations

The transaction buffer size is 4 KB. This buffer is used to pass data when the protected mode application calls a real-mode procedure. One can get around this problem by using interrupts or real procedure signals. Second, the number of real procedures can not exceed 32. Considering that DOS only uses 640 KB of memory, 32 real procedures will be sufficient for most purposes. If the real procedure is called as an overlay, then the CS:IP in the EXE header is needed; therefore, the executable must be an .EXE file not a .COM file.

3.3 Conclusions

In appendix A, there is a list of graphics commands used by the extended version of CLIPS. This graphics version of CLIPS uses extended memory and was created using the packages mentioned in section 3.2. Appendix B contains two rules that cut a portion of an image on the screen and will slide it randmoly around the screen. One excellent feature provided by the off-the-shelf packages is the ability to detect the graphics hardware and pointing device present in the computer. The PC platform has a wide variety of hardware and is difficult to keep track of all the different brands and models. This feature frees the user from the trouble of configuring the application for the particular system in which it will run.

Applications created using this method has the advantage that all programs are written in the extended CLIPS language (expert system, graphics output, and user interface). In order to run different applications, the only executable needed is the extended version of CLIPS. If the application

needs to be modified, only the application code needs to be changed without needing to compile or link the source code again. Moreover, all the tools like fonts, drivers, and graphics routines will altogether coxist in one CLIPS executable package. Applications will only consist of source code and data; thus, large amounts of storage space will be saved permiting even PCs with small storage devices to store complete applications.

III. Concluding Remarks

Section 1 shows a way overcome CLIPS memory problems by using extended memory; section 2 shows a way to mix graphics with CLIPS, and section 3 shows a way to use CLIPS in extended memory and how to mix it with graphics. Again, this is only one way to solve these problems, but if you are designing a large application on a PC, you will surely have one of the problems discussed in this paper, so if you have any of these problems and don't have a solution, or you are thinking in designing a large application, this paper provides you with the information needed to solve that problem.

The best outcome of the whole process is that PC's applications, built using CLIPS and graphics, can overcome the 640K memory limitation imposed by DOS, and applications using CLIPS will be able to handle large knowledge bases. This capability allows developers to use the PC platform as an application development and delivery platform, and will permit users to enjoy the power, low cost, portability, and accessibility of the new generation of PCs.

APPENDIX A

Extended CLIPS version (Graphics and Extended Memory)

List of Graphics Commands:

initialize-graphics	draw-sector	write-grstringxy	erase-mode
draw-bar	draw-linerel	write-grstring	set-palette
memory-left	draw-line	set-textstyle	draw-icon-bar
change-bkcolor-to	draw-polygon	set-textjustify	free-mouse-draw
set-viewport	draw-lineto	hide-mouse	cursor-shape
draw-rectangle	pause	show-mouse	draw-menu
clear-viewport	draw-bar3d	mouse-waitchange	save-image-file
clear-screen	status-message	set-usercharsize	read-image-file
draw-ellipse	pause-keyortime	mouse-waitrelease	write-image
draw-circle	fill-pattern	mouse-waitpress	read-image
draw-pieslice	status-message-top	mouse-RightPressed	grid-pick
draw-point	set-linestyle	mouse-LeftPressed	release-image
mouse-in-rectangle	move-mouse	mouse-AnyPressed	set-rectangle
pt-in-rectangle	random-number	mouse-MiddlePressed	close-graphics
set-active-page	move-cursorrel	mouse-positiony	set-nosound
limit-mouse	move-cursor	mouse-positionx	set-sound
set-draw-mode	set-visual-page	max-screenx	spawn-process
key-pressed	get-scankey	clear-text	draw-textbox
max-screeny	Initialize-Animation	close-animation	bell
ed	rectangle-animation	text-attribute	text-xy
draw-arc	set-usercolorpattern	mouse-visible	set-delay
set-timer			

APPENDIX B

Extended CLIPS version (Sample Source Code)

These are two rules that when fire, they will cut an image from the screen and will move it randomly around the screen. The move will be performed in the specified number of steps. Before running the program, the command "**initialize-graphics**" has to be run interactively from CLIPS or added to the CLIPS initialization rule.

```
;
;    This Rule Reads an image from the screen and initializes the animation procedure.
;

(defrule copy-rectangle
  (initial-fact)
 =>
  (limit-mouse 0 0 639 479)                    ; Limit mouse movements to specified box
  (read-image 251 251 349 349 1)               ; Read image in specified box (cut)
  (clear-screen)                               ; Clear the graphics screen
  (write-image 255 255 1)                      ; Write image to specified position (overwrite)
  (release-image)                              ; Free image resources
  (Initialize-Animation 251 251 349 349)       ; Initialize animation for given box.
  (clear-screen)
  (assert(count 80)
         (animation-start)
         (do animation)
         (coord 251 251)
         (rectangle-start)))

;
;    This rule moves the image ramdomly around the screen 80 times.
;

(defrule do-animation
  ?one<-(do animation)                         ; Begin process
  ?two<-(coord ?a ?b)                          ; Retrieve actual coordinates
  ?three<-(count ?cc)                          ; How many times more do we need to fire this rule
  (test (> ?cc  0))                            ; Stop moving..?
  (test (key-pressed))                         ; Test if key was pressed, if pressed don't fire rule.
 =>
  (retract ?one ?two ?three)
  (bind ?x (random-number 539))
  (bind ?y (random-number 379))                ; Create random numbers within the screen size
  (bind ?w (+ ?a 98))
  (bind ?z (+ ?b 98))
  (bind ?cc (- ?cc 1))                         ; Working variables.
  (rectangle-animation ?a ?b ?w ?z 5 ?x ?y 0)  ; Move image using given arguments
  (assert (do animation)
  (coord ?x ?y)(count ?cc)))                   ; Repeat until count is reached.
```