CanDo

Power Steering For The Amiga

The CanDo© HowTo Book

For the first time you can harness the tremendous power already built into the Amiga.™

Doit

The
CanDo
HowTo
Book

CanDo's software and manual were written entirely by: Eddie Churchill, Cash Foley, Tom Hardison, Tim Martin, and Martin Murray. Special thanks to our families who missed seeing us for the past year. Manohman, this was alottawork.

For CanDo Help Call: 214-340-4992

INOVAtronics

INOVAtronics, Inc.
8499 Greenville Ave. Suite 209B
Dallas, Texas 75231 USA
214-340-4991

What's In The CanDo HowTo Book?

1 What Does CanDo Do?

2 You CanDo It

3 Decks & Cards

4 Objects

5 Script Editor

6 Commands

7 Appendices

1 What Does CanDo Do?

1 What Does CanDo Do?

CanDo is a revolutionary, Amiga specific, interactive software authoring system. Its unique purpose is to let you create real Amiga software without any programming experience. CanDo is extremely friendly to you and your Amiga. Its elegant design lets you take advantage of the Amiga's sophisticated operating system without any technical knowledge. CanDo makes it easy to use the things that other programs generate - pictures, sounds, animations, and text files. In a minimal amount of time, you can make programs that are specially suited to your needs. Equipped with CanDo, a paint program, a sound digitizer, and a little bit of imagination, you can produce standalone applications that rival commercial quality software. These applications may be given to friends or sold for profit without the need for licenses or fees.

Although CanDo is especially easy to use, it is extremely powerful and versatile. Its uses range from building simple slide shows and interactive presentations to animated multimedia productions, quality educational software, and even sophisticated control applications that communicate with external video and audio equipment. CanDo lets you build small programs and add features as you need them.

Since CanDo opens the world of software development to all Amiga users, it has the potential of making an endless number of applications available to every Amiga owner. We are very excited about this becoming a reality and hope to facilitate it with future enhancements and support.

Whatzindabox?

Along with this Manual, the package should contain Two Disks: CanDo and CanDoExtras. Also included in the box is a Registration Card that you should take the time to fill out and return to us. This will enable us to inform you of CanDo revisions and will help our technical support staff to help you.

Please take a look at the ReadMe files on each of the CanDo Disks. They contain last minute information that was not available when this Manual was printed. They may be viewed by double clicking their icons.

CanDo System Requirements

CanDo will work on any Amiga computer that has at least one (1) megabyte of memory. If you don't have a hard drive, it is recommended that you have two 3 1/2" floppy drives. Since CanDo makes it easy to work with sizable data like pictures and sounds, a hard drive and expanded memory will dramatically increase your efficiency and productivity. Be sure your Amiga is running WorkBench version 1.2 or greater.

This manual assumes you are familiar with the operation of your computer as explained in the manual that came with your Amiga. You should know how to use the mouse, start applications from WorkBench, and do simple WorkBench operations.

You CanDo a Backup

Please, before you use CanDo, make copies of all the original disks. Work with these copies and keep the originals in a safe place. For your convenience CanDo is not copy protected so spread the word not the disk.

To make copies of the CanDo disks:

1. Write-protect both CanDo disks by flipping up the write protect tab on the disks. This will protect them from accidental erasure.
2. Place the CanDo disk into drive DF0: and select the disk icon so that it is highlighted.
3. Select Duplicate from the WorkBench menu. Select Continue when asked to insert CanDo.
4. Insert a blank disk into drive DF0: when asked for the disk.
5. Exchange disks when requested and select Continue until the copy is complete.
6. Once you have a copy, rename it by highlighting the disk icon and selecting Rename from the WorkBench menu. Remove the "Copy of " text that was appended to CanDo. Be sure to delete any spaces that precede the name.
7. Repeat the procedure for the CanDoExtras disk.

CanDo on a Hard Disk or a Floppy Disk

CanDo on a Hard Disk or a Floppy Disk

CanDo is easy to install on your hard disk. You'll need at least one (1) megabyte free on your hard disk before copying CanDo and all the support files.

For floppy disk owners, CanDo will run directly from the working copy you made. You must start your Amiga with this CanDo disk and run CanDo from the icon. You will likely choose to install CanDo on your own WorkBench startup disk. This is done by using the CanDoInstall program found on the CanDoExtras disk.

To install CanDo on your WorkBench disk or your hard disk:

1. Start your computer normally and then insert the CanDoExtras disk into a drive.
2 Open the CanDoExtras disk by double clicking its icon on WorkBench. Open the Utilities drawer.
3. Now the Installer program may be started by double clicking on its icon.
4. This program will guide you through the installation process for both floppy drives and hard drives.
5. Once the installation is complete, you may remove the CanDoExtras disk and close its window.
6. CanDo may now be started from your floppy or hard disk by double clicking its icon.

A couple of questions you might have...

Why do I need to "Install" CanDo?

When CanDo is started, it looks in an Amiga system directory called LIBS: and another called L: for support files. These are particular directories that the Amiga uses to keep special files. The CanDoInstall program copies these files for you.

What am I creating when I use CanDo?

The applications you create with CanDo are called Decks. This is because they are comprised of Cards. In CanDo you work with Cards one at a time and they may have completely different environments and characteristics. When you save a Deck from CanDo, it is like an executable program that you may start by double clicking its icon on the WorkBench screen. These Decks are actually small files that use a "Library" that is installed on your system. All CanDo Decks and CanDo itself efficiently share the use of this library. This arrangement enables you to run many Decks at the same time and use the Amiga's memory and its resources to the full extent of its capabilites. Remember, the CanDo library is not a distributable portion of CanDo.

How do I make a distributable CanDo application?

When you are ready to make an executable version of your program, you can do so easily with the Binder. This utility can create a distributable version of your project in either of two ways:

1. **Independent Project:** A form of your Deck which will run on any Amiga, regardless of whether or not CanDo is also available on that Amiga. Programs produced in this way can be sold or distributed without a license or fees of any kind. These programs have the primary part of the CanDo support software built into them.
2. **Dependent Project:** A form in which the project will only run if you or your friends, or potential customers, have CanDo already. This type of bound project is significantly smaller in size than an Independent Project (it shares the same library of functions that CanDo uses), and is the perfect way to make a program which you intend to give or sell only to people who already own CanDo.

Continued next page...

To "Bind" a Deck, select the Deck's icon that you would like to make standalone and then, while holding down the shift key, double click on the Binder icon. It will guide you through the binding process and create a standalone application. Remember the icon for your new application will not be visible in a WorkBench window until it is re-opened. The standalone deck may now be distributed along with any graphics or sound files that it uses. Don't forget that these resource files must be found in the directories that were specified when the Deck was created.

Can I make room on my CanDo disk if I don't need all the features?

CanDo is very modular in design. If you don't need some of its features, you may remove them from your disk. If you don't use ARexx objects you may open the Objects drawer and discard it. Similarly, you may discard items in the EditorTools, Xtras, and HelpFiles drawers. Make sure you have a backup of these files in case you need these features in the future.

The modular design of CanDo also enables you to add features in the future when they become available.

How do I get help?

If you have a problem that this manual does not address, please call our Support Staff at: 214-340-4992 or if you have a modem, you can call our Product Support BBS at: 214-357-8511 (300/1200/2400 Baud)

2 You CanDo It Index

2 You CanDo It

There are two methods of starting CanDo.

First: CanDo has a Workbench Icon. Double-clicking on this Icon automatically launches the program.

Second: CanDo may be launched from a CLI or Shell by typing...

CanDo

and then pressing **Return**. CanDo will begin to load. *The CanDo Reporter*, a small message window, will appear and inform you that the files that makeup CanDo and its user interface are loading. The last message you will see is "All done, finishing up."

Now, at the bottom of the display, you will see CanDo's *Main Control Panel* and behind the panel is your default window.

You CanDo an Application.

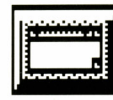
The first software project that you're going to create is a very simple one. When executed, this Program will open a Window, (or Card as we will now call them). All CanDo applications consist of Cards and Decks. Cards are the basic building blocks of any CanDo project. A Card contains a single Window and all its Objects and Attributes. A Deck consists of several Cards. Moving between Cards and Decks is a fundamental part of CanDo project design.

We will begin working on the first Card of this three Card project. It will have a full screen IFF Image displayed. On this Image, we are going to place two Buttons (or Gadgets in Amiga-ese), one with the words "Next Critter" written on it, and the other with the word "Quit." When the Button marked **Next Critter** is clicked, a second IFF Image will be shown. Clicking it again shows a third Image. Each Image is its own Card, with Buttons identical to the ones on the other Cards that you have created. When any of the Buttons marked **Quit** are clicked, the program will stop and the user will be returned to the CLI or Workbench.

Lets Do It

On the CanDo *Main Control Panel* there are three boxes: Status, Cards, and Objects. There are two Buttons in the Status box: Design and Browse. Selecting **Design** allows you to create your projects. You can select **Browse** to test or run your creations at any time, without exiting CanDo. Ready to start?

1. Click on the **Design** Button to begin working on your project.



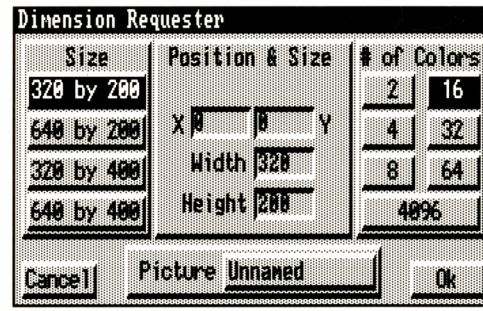
Window Button

Objects portion / Main Control Panel.

2. Click on the **Window** Button in the *Objects* portion of the *Main Control Panel.* This is the first step in the creation of a background image for your first Card. You'll be shown the *Window Editor.* This is how you'll describe your Window. You can type your Window's name in the Field under "Window Title" but for this example, we want to clear out the default name.

3. Click in the Field then press the **Right Amiga** Key... **X** which will clear the Field. This will produce a blank Title Bar above your Window, giving it a cleaner look.

4. Now click on the **Dimension** Button. The *Dimension Requester* appears.



Dimension Requester

5. Click on the Button to the right of "Picture". This brings up CanDo's *File Requester.* Use the requester to locate the file **Rabbit.Pic** that was included on your CanDoExtras disk. Refer to chapter 4-4 of the manual for information on how it operates.

6. Choose the **Rabbit.pic** file. The Path Field now shows the correct directory path and the File Field shows "Rabbit.pic."

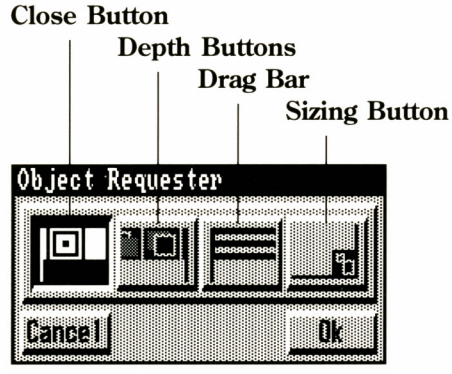
7. Now click the **Ok** Button. This will return you to the *Dimension Requester.* The Button next to "Picture" now shows, the highlighted Name **Rabbit.pic**, the File you chose. When your Window does open it will have the proper dimensions and resolution for the picture file you have just chosen, with the picture as it's background.

8. Again click the **Ok** Button. This returns you to the *Window Editor.* On the right, you'll see several Buttons. You've already used the first one "Dimension". For this application, you'll also use the bottom one, "Options."

9. Click on the **Options** Button. This will bring up the *Options Requester* which has several choices. For this example, you will need to change only one of the default settings.

10. Click on the first Button, **The window has visible borders.** The Button will change to **The window's borders are invisible**.

11. Now click on the **Ok** Button. This will return you to the *Window Editor.*



Window Editor.

12. Now click on **Objects**. This brings up the *Objects Requester.* Click on the **CloseButton** icon to deselect it, we won't need one. Click **Ok**.

13. Click on the *Window Editor's* **Ok** Button. There's your Rabbit. You've created your Window and you are back at the CanDo *Main Control Panel.* To see the the whole picture...

14. Click on the **Up/Down Arrow** Button on the right side of the panel. The panel will drop out of the way.

15. Click the **Up/Down Arrow** Button again and the panel returns the to its working position. You're now ready to add a title to this picture. To do this you need to write a Script.

Cards portion / Main Control Panel

16. Click on the **Edit Cards** Button in the Cards portion of the *Main Control Panel*. This will bring up the *Card System*. Here you will see a card Requester with one file name in it "Card#1". This is the Card you've been working on. Since it is already selected...

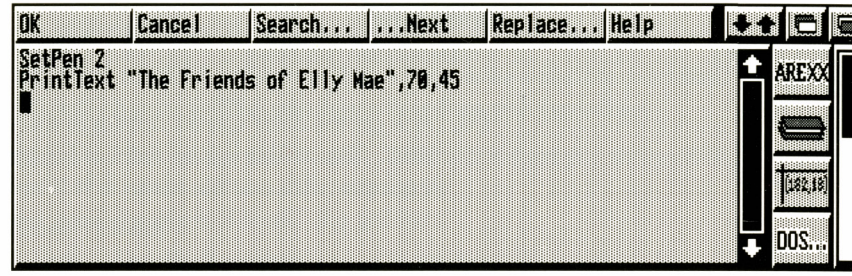
17. Click on the **Edit** Button. This will bring up the *Card Editor*. Here you see three Buttons that allow you to attach Scripts to your Card. But what is a Script anyway?

A script is a Command (or, more often, a series of them) to be executed under certain conditions in a CanDo project. A Script is often attached to an event in your application. In other words, events like clicking a Button, activating (clicking in) a Window, or going from one Card to another can cause a Script of Commands to be executed. Script Commands can be used to play a sound or sound sequence, run animations, move from one Card in a Deck to another Card, add text to a Window and many more complex tasks.

While this example project uses scripts in their basic, most straight for ward form, you may wish to take a moment to look at the complete list of Scripting Commands, together with descriptions, syntax, and examples, that appear in the Commands Section of this manual. This side trip will give you an idea of the power and simplicity of CanDo's Scripting Language.

18. Now back at The *Card Editor*, you will need to choose one of the three Buttons to add the title to your picture. Your choices are: at the Card's StartUp, AfterStartUp, or when Leaving the Card. You want the title to appear in your Window immediately after the Rabbit graphic is displayed.

Click the **AfterStartUp** Button.



Script Editor

19. In the *Script Editor*, type the following lines verbatim.

Setpen 2
PrintText "The Friends of Elly Mae",70 ,45

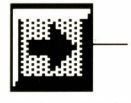
The Setpen Command sets the color of the text to be written from the Window's palette. The PrintText Command simply prints the text within the quotes at the coordinates specified (70 , 45). This means 70 pixels over and 45 lines down.

20. Click the **Ok** Button to add this Script to your project.

21. At the *Card Editor*, click **Ok**.

22. Now click **Exit** to leave the *Card Editing System*.
You're back at the *Main Control Panel.* You may want to go into Browse mode to check your creation so far.

23. Click **Browse** in the *Status Panel.*



Next Card

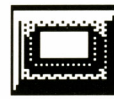
Cards portion / Main Control Panel

In the Cards portion of the *Main Control Panel* you see an Arrow Button pointing to the right, that's the NextCard Button. When the NextCard Button is clicked, the next Card in your Deck is rendered (or "started-up.") on Screen one after another. When you reach the last Card in your Deck and click NextCard again, CanDo loops back to the first Card in the Deck. Since we have only one Card so far in this example, clicking the NextCard Button will result in looping back to the same Card.

24. Now click the **NextCard** Button. Your Card, picture, title and all will be freshly rendered on your Screen. Not bad!

Now, we want to make this good looking window do something else, so let's Add some Buttons.

25. First, click on the **Add** Button in the *Objects Panel.*



Buttons Button

Objects portion / Main Control Panel

26. Then, click on the **Buttons** Button in the same panel. The *Button Object System Requester* will appear. This Requester asks you to locate the Origin Point of your new Button.

27. Click the **Ok** Button.
The *Edit Button Requester* will be lowered revealing your full Screen. Move your Mouse to position the Pointer on Screen where you want your Button. For this exercise move the Pointer to above the "T" in the word "The".

28. Now click and release the **Left Mouse** Button to set the Origin Point. Move your Mouse, notice that it now controls the shape and size of a rectangle. Move the Mouse in any direction you choose, adjusting the rectangle until you are satisfied with its size and shape.

If you don't like the Button for some reason, click the **Esc** Key and start the process over.

29. If the location and shape are to your liking, click the **Left Mouse** Button. The *Button Editor Requester* will be displayed and the location (origin) coordinates of the Button you've just created, in horizontal and vertical format, will have been entered for you automatically. You will also see the Button shape you have created positioned on the Screen. You may edit these by clicking in the field and typing new coordinates. Edit "Horiz" to be **50** and "Vert" to be **25**.

There are three types of Buttons in CanDo: Area, Text, Image. The Button we're making will be a Text Button.

30. Press the **Box** next to "Text." This brings up the *Font/Text Requester.* Here you will select your Font. The default setting of Topaz 8 Plain will work well here.

Notice, there are several style options available. This time choose **Shadowed**.

You'll also see three *Color Selectors*. Click the left one, it governs the color of the text on your Button.

Across the bottom of the Screen you will see a Color Palette. Click on a bright, light color. Your choice will be displayed in the Selector you clicked on.

You can also choose colors using the Arrows on the Selector. You may change the second Color Selector to suite your tastes. Try it.

31. Now, click in the "Text" Field and press the **Right Amiga** Key and **X** to clear the Field. Now type...

Quit

This text will become part of the Button.

32 Click the **Ok** Button. This will return you to the *Button Editor Requester.* To enable the "Quit" Button to actually end a project we must attach a Script to it. There are four Button Events which can trigger a Script. This time you want the Script executed when the Mouse Button is let up after being pressed. This is called a Release script.

33. Click on **Release** in the Scripts portion of the Requester. A *Script Editor Requester* will appear. Type:

quit

34. Click on the **Ok** Button.

35. At the *Edit Button Requester* , click on the **Ok** Button. Now you're back at the *Main Control Panel.*

The procedure for making the second Button is exactly the same as the first except that this Button's text and position will be different.

36. Click on **Add** and then on the **Button** Button in the *Main Control Panel.* Place this Button's Origin to the right of the "Quit Button" you just created and draw another small rectangle. You may adjust its coordinates to a horizontal of **175** and a vertical of **25**. This time, the text for the button will be "Next Critter" and its Release Script will be:

NextCard

When you have your new button finished you should click **Ok** at all Re questers until you're back at CanDo's *Main Control Panel.*

Now when you click "Next Critter" the project will go to its Next Card, which will also contain a Picture and some Buttons. What next Card you say? You can make one in less than thirty seconds.

37. Click on the Edit Cards Button in the Cards portion in the main panel. This brings the *Card System* back up.

Click on the **Duplicate** Button on the right of the Requester. A second Card will added to your project. This Card is a duplicate of your first Card. Since your first window was a Picture Window, the window in your second Card will be a Picture Window as well, complete with same graphic.

The new window also has the same Window Options and Window Objects as the first window.

The Buttons and scripts are carried over to the new Card.

38. This window will require changes to the script, so, here at the *Card Editor*, click on **AfterStartUp**.

When the *Script Editor* comes up, select the menu option **Clear** under the Scripts menu. Click **Yes** to verify that you really want to do this. Then click **Ok** to leave the *Script Editor.*

Click **Ok** at the *Card Editor.* Exit the Card Editing system. You are now working on a new Card. And since we don't really want two pictures of the same rabbit, let's change the graphic to a different critter.

39. Click the **Windows** Button on the *Main Control Panel.* In the *Window Editor*, click on **Dimensions** and then select **Picture** Button again to get the *File Requester.*

When it comes up, select the graphic file **Porcupine.pic**. Click **Ok**. Since the options of first window are carried over to this one, and we want to use them as they are, you can now click **Ok**. Your second Card is now finished and operational.

40. The third and final Card will be by far the easiest to create. We are going to clone the second Card, complete with graphic, buttons, and scripts. Then, we'll simply substitute a new critter pic.

41. Repeat step **37** and there's no need to edit the AfterStartUp Script.

42. Repeat step **38** and this time use the **Mink.pic** file. Return to the *Main Control Panel.* If you want, you can change the border style or text color of any button by clicking **Edit** and then on the actual button you would like to edit.

43. If you're not at the *Main Control Panel,* return to it for your final step in this project. Under the "Deck Menu," select **Save.** You will be prompted to give a filename and pathname. Now click **Ok**.
And that's it! You've just created a working program. Congratulations.

Second Project

Your second Project will be very different from the first. It will be an extremely useful and flexible Application, and in some ways, easier to create than your first one was.

We call this Application a "PowerPanel" and it puts almost any Amiga Command or Application just a click of the Mouse away.

You create it using CanDo's DOS Command. This Command allows you to run any Program as though it was a CLI Command. Because all Programs cannot be run through CLI, you should always consult the Program's Documentation before using it with CanDo's DOS Command.

The PowerPanel

A PowerPanel is (or at least can be) a Window that opens on the Workbench Screen with several Buttons on it. For example, there could be a Button that says "Paint" on it. When the Button is clicked, the PowerPanel could start your favorite paint program.

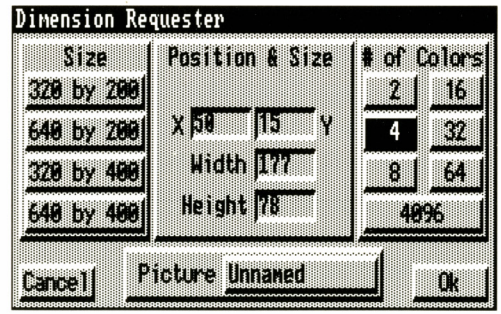
These PowerPanel Buttons can set up entire environments: change directories, make assignments... do almost anything.

This PowerPanel example will be designed to perform functions accessible to all Amiga/CanDo users:

- **Run CanDo**
- **Run Phone Index (a CanDo application)**
- **Open a new CLI**
- **Run the WorkBench Calculator Utility.**
- **Play some Sampled Sounds (just for grins)**

They're a breeze to create and run, so lets get started.

1. Click the **Design** Button.
2. Click on the **Windows** Button in *Objects*. The *Window Editor Requester* will appear.
3. Clear the "Window Title Field" and replace the title with... **PowerPanel**
4. Click on the **Dimension** Button.



Dimension Requester;

5. The *Dimension Requester*, appears. In the "Position and Size" portion set the Window's X coordinate to **50** and the Y coordinate to **15**.

6. Set the Window's Width to **177** and Height to **78**.

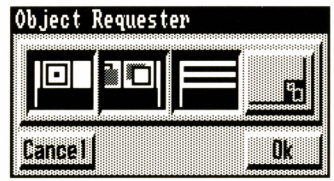
7. You'll use four colors (the Workbench colors), so click on **4** under "# of Colors".

8. Click **Ok** to return to the *Window Editor Requester* .

9. Click on the **Objects** Button to open the *Object Requester.*

   Here you see graphic representations of these familiar Amiga Window objects: Close Button, Depth Buttons, Drag Bar and Resizing Button. When added to CanDo Projects, all four Objects perform their standard functions.

   For example: a Window's Close Button will execute a Script when clicked which could quit the program, go to the next card, or perform any CanDo Command.



Object Requester.

   For this Project select the **Close Button**, **Depth Buttons**, and the **Drag Bar**. When selected, their Buttons will be Highlighted. Click **Ok** to record these choices.

10. At the *Window Editor Requester.* Click **Ok** You are back at the *Main Control Panel* and a smaller Window has been opened on your Workbench Screen. This is your PowerPanel Window and you're ready to create your PowerButtons.

    First, we'll Add the Button that starts CanDo. It will be a Text Button and will be created in the same manner as the Quit Button in the First Tutorial Project. Only the placement of the Button on the PowerPanel Screen and the Text on the Button will be different.

11. Click **Add** and the **Button** Button under Objects.

12. At the Add New Button Requester click **Ok**.

13. Move the Pointer to a position in the upper left corner of your PowerPanel Window. Click the **Left Mouse** Button to set the Origin Point. Size the Button rectangle as before and click the **Left Mouse** Button.

14. At the *Edit Button Requester* in the Origin section adjust the Button's Origin Point by changing the Horizontal and Vertical coordinates to **13** and **14**. The exact size you made this Button is unimportant because it will automatically adjust to the size of the Text that we put in it.

15. Now click on **Outline** in the Border section.

16. At the *Border Selector* click on the **Embossed** Button. Now choose the Border Colors you want using the *Color Selectors* and click **Ok**.

17. At the *Button Editor Requester* click on the **Box** next to "Text" under "Style." The *Text Button Definition Requester* will appear. In this Requester you will set these parameters:

Font: **Topaz**
Font Size: **8**
Font Style: **Plain**
Text Color: **(Your Choice)**
Text: **CanDo**

With these settings click **Ok**.

18. At the *Button Editor Requester* click the **Release** Button under "Scripts."

19. At the *Script Editor,* the far right hand slider controls a List of Editor Tool Icons. Locate and then Select the **DOS** Editor Tool.

DOS Editor Tool

DOS Editor Tool / Script Editor

20. At the *CanDo File Requester* use the File List on the right to find, then Select the **CanDo** Program File. Click **Ok**.

21. At the *Script Editor* you will see that the proper DOS Scripting Command has been automatically typed into the Editor. Click on **Ok**.

22. At the *Edit Button Requester* click **Ok**. The *Main Control Panel*, your PowerPanel Window and it's first Button will appear.

The remainder of the Buttons you will define for this Project are Text Buttons also. Retrace steps 11 through 22 which you used to create your first PowerButton but use these Button Definitions below. Other possible differences can be variations in Font choice, Text Color and Border Style. Use the *DOS Editor Tool* to locate the program each button will start.

Following is a description of each Button you need to define:

Phone Index Button

Button Co-ordinates:	**72, 14**
Border Style:	**Embossed**
Font:	**Topaz**
Font Size:	**8**
Font Style:	**Plain**
Text Color:	**(Your Choice)**
Text:	**Phone Index**
Program:	**CanDoExtras:Utilities/PhoneIndex**

Calculator Button

Button Co-ordinates:	**13, 32**
Border Style:	**Embossed**
Font:	**Topaz**
Font Size:	**8**
Font Style:	**Plain**
Text Color:	**(Your Choice)**
Text:	**Calculator**
Program:	**Sys:Utilities/Calculator**

New CLI Button

Button Co-ordinates:	**112, 32**
Button Style:	**Embossed**
Font:	**Topaz**
Font Size:	**8**
Font Style:	**Plain**
Text Color:	**(Your Choice)**
Text:	**NewCLI**
Program:	**c:NewCLI**

23. Return to the *Main Control Panel.* Now, let's add some text to the Window to explain that the Buttons at the bottom of the panel that we are about to add are for playing sounds. Click on the **Edit Cards** Button in the *Cards* box. The *Card System Requester* will appear. Press the **Edit** Button to edit the only card listed, "Card# 1" The C*ard Editor Requester* will appear. Press the **AfterStartUp** Button to add the Script that will be performed after the card's window has been opened and all the Buttons have been attached.

24. You want to print the text "Sound Samples" just below the Buttons you've already made. Type in the following Script:

SetPen 1
PrintText "Sound Samples",35,48
PrintText

25. Press **Ok** to save the Script. This will return you to the *Card Editor Requester* .

26. Press **Ok** again to return to the *Card System Requester.*

27. Now press **Exit** to return to the *Main Control Panel.*

28. Click on **Browse**, and then click the **NextCard** Button in the Cards panel. You'll see the text you just created appear in your Window.

29. Now click the **Design** Button. You're going to add a few simple unmarked rectangular Buttons. Each one, when clicked, will play a different sampled sound.

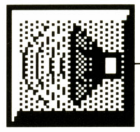
30. Click on **Edit** and then on the **Buttons** Button in the Objects panel.

31. At the *Button Editing System*, click **Add**. When prompted, place the Origin of the first of these sound buttons in the lower left corner of your Window. This Button is going to be an "Area Button." When the *Edit Button Requester* comes up, rename this Button "Sound1". Then, adjust the Button's co-ordinates to **13**, **62**.

32. Click on **Area** and the *Area Button Requester* will appear. Adjust the Button size to **25,10**, using the Width and Height Fields in the Requester. All of our "Sound Buttons" will be this size.

33. Click **Ok** to return to the *Edit Button Requester.* The Border and High light styles may be experimented with, but the defaults will work well for your purposes. Now, you'll want to add a Release Script, so click **Release**.

34. At the *Script Editor,* click the **Sound Editor Tool** on the right.



Sound Editor Tool

Sound Editor Tool / Script Editor

35. Select **Set The Filename** from the *Play a Sound Requester.* This will bring up CanDo's *File Requester.* Find the **DogBark.snd** file, select it and then click **Ok.**

36. Click **Ok** at the *Play a Sound Requester.* The PlaySound Command will have been typed into your script. Click **Ok** to save the script.

37. At the *Edit Button Requester.* click **Ok.** You are now back at the *Main Control Panel.*

There will be three more "sound buttons" like this one, except with different co-ordinates and different sounds attached.

Repeat Steps 29 through 36 using the Button definitions below:

Sound 2

Button Name:	**Sound2**
Co-ordinates:	**55,62**
Area Button Size:	**25,10**
Sound file:	**(Your Choice)**

Sound 3

Button Name:	**Sound3**
Co-ordinates:	**97,62**
Area Button Size:	**25,10**
Sound file:	**(Your Choice)**

Sound 4

Button Name:	**Sound4**
Co-ordinates:	**139,62**
Area Button Size:	**25,10**
Sound file:	**(Your Choice)**

Save your Deck now. Under the Decks Menu Select "Save". use CanDo's File Requester to set the Path and File Name for your Deck. Click **Ok** and you're done.

3 Decks & Cards Index

3 Decks & Cards

A CanDo project is comprised of a Deck of Cards. CanDo Cards are somewhat like Flash Cards in that only one is shown at a time. However, unlike Flash Cards that only present information, CanDo Cards can both present and receive information. Furthermore, you can make things "happen" on a CanDo Card. You do this by making Objects.

The Window Object allows you to specify the overall appearance of your Card. Buttons and Menus, for example, allow the user of your project to tell it to do something. Other Objects allow you to easily receive information or otherwise control what is happening.

By adding Objects to a Card, you design what it looks like and what it does. The Card is simply a container for the Objects you give it. Some applications only need a single Card, while others need many Cards to change the interface, appearance or activity. By simply changing Cards, you can cause dramatic or subtle differences in what is happening.

When you have designed a Deck, you can save it to disk. This allows you to later load that Deck back into CanDo for further editing. The Deck can be run separately by double clicking its icon from Workbench, or from a CLI using the CanDoRunner program.

When you are ready to make an executable version of your program, you can do so easily with the CanDo's Binder. It will create a new file that is a runnable program. However, the program created by the Binder can not be re-edited by CanDo. Make sure that you keep the original Deck saved by CanDo for later editing.

This Chapter tells you how to work with Decks and Cards. Chapter 4 describes the Objects you can add to a Card. Chapters 5 and 6 describe how to write the Scripts that make your application come to life.

Menus

When CanDo's Main Control Panel is the active window, you can access CanDo's Menus. They are not accessible when your application's Window is active. By clicking on any portion of the Main Control Panel, not necessarily on a Button, you activate the window. This gives you access to CanDo's Menus when you press the right mouse Button.

CanDo's menus are: Deck, Card, Objects, and Misc. The Objects Menu is described in Chapter 4.

Deck Menu

The Deck Menu has five items: New, Open, Save, About, and Quit.

New

New deletes all existing Cards and starts you up in the default configuration. (See the Advanced Topics Appendix for changing the default Deck.)

Open

Open replaces the current Deck with one loaded from a file. CanDo's File Requester allows you to locate the file. The file must be a Deck created with CanDo. You cannot load a Deck that has been made into a stand-alone program using CanDo's Binder.

Save

Save writes the current Deck to a file specified with CanDo's File Requester. A saved Deck can be loaded into CanDo using Open. It can also be run separately by double clicking its Icon, or bound into a stand-alone program using CanDo's Binder.

About

About simply displays a little information about INOVAtronics and CanDo.

Quit

Quit allows you to leave CanDo. A Requester will warn you if you have modified the current Deck.

Card Menu

The Card Menu has five items: Goto, First, Last, Previous, and Next.

Goto

Goto... allows you to go to a specific Card. A requester will show all the Card Names. Click on one of the names to highlight it. Clicking on the Goto Button or double clicking one of the entries will cause you to change Cards. When you want to stay on a Card, select the Exit Button.

First

First causes CanDo to go to the First Card in the Deck.

Last

Last causes CanDo to go the Last Card in the Deck.

Previous

Previous causes CanDo to move back one Card. If you are on the First Card, this will put you on the Last Card.

Next

Next causes CanDo to move forward one Card. If you are on the Last Card, this will put you on the First Card.

Misc. Menu

The Misc. Menu has two items: SuperBrowse and System Info.

SuperBrowse

SuperBrowse temporarily removes the CanDo interface to your application. The *Main Control Panel* will disappear, and your application will be able to interact a little faster. This mode allows you to see exactly how your application will run as a separate program.

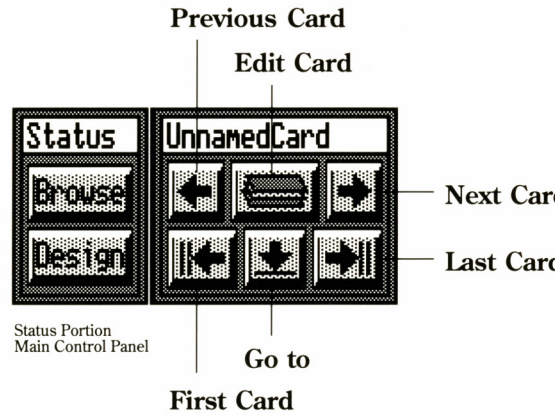
When you enter SuperBrowse Mode, CanDo will open a small Window on your Workbench Screen. When you double-click on the Window, CanDo's *Main Control Panel* will reappear and you will no longer be in SuperBrowse Mode.

System Info

System Info will display a requester displaying Available Memory, CanDo version information, and other useful statistics.

Status Panel

The Status Portion of the *Main Control Panel* contains the Browse and Design Buttons.



Browse

By selecting **Browse**, you can test your application. Because CanDo is monitoring your activity, your project will not be as responsive as when it is running in SuperBrowse Mode or as a separate application.

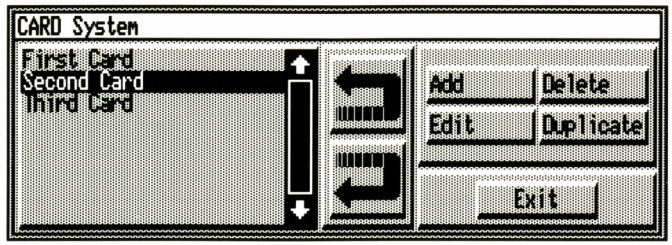
Design

When **Design** is selected, you can construct your application. Selecting any Button on the *Main Control Panel*, other than Browse, will automatically select Design.

The **First**, **Last**, **Previous**, **Next**, and **Goto** Buttons work the same as their Menu counterparts.

Edit Card

Selecting the **Edit** Card Button brings up the *Card List Requester.* From this Requester you can Add, Edit, Delete, Duplicate and Reorder Cards.



Card List Requester.

The list on the left side of the *Card List Requester* contains the names of all the Cards in the Deck. You can highlight a Card Name by clicking on it in the list. The Buttons on the right side of the Requester will work with the highlighted entry.

Add

The Add Button allows you to Add a new Card to the Deck. The New Card will not contain any Objects. When you click this Button, the *Card Editor Requester* will be displayed. This Requester is described later in the Chapter.

Edit

The Edit Button allows you to edit the currently highlighted Card. If you double-click a **Card Name**, it will also edit the Card. In either case, CanDo will go to the selected Card and display its *Card Editor Requester.*

Delete

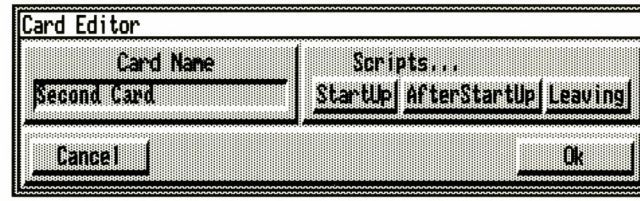
The Delete Button deletes the highlighted Card. A Requester will ask you if you really want to delete it. Be certain that you want to delete it. The Card and all of its Objects are disposed of completely.

Duplicate

The Duplicate Button makes a copy of the selected Card and Objects. *The Card Editor Requester* is then displayed for this Card. This allows you begin a new Card with a copy of the existing Objects.

The list shows the Cards in the current order. The two large **Arrow** Buttons allow you to reorder the Cards. By using these Buttons, you can move the highlighted Card up and down through the list.

The *Card Editor Requester* has a Field for the Card Name and three Buttons for Scripts. You can change the Card Name by simply changing the name in the field.



Card Editor Requester

The Card has three Scripts associated with it: Startup, AfterStartup, and Leaving.

StartUp

The StartUp Script is performed before your Card's window opens and before any other Objects on your Card are created. This is a good place from which to initialize variables, load files, etc. Because the window has not yet been opened, this Script cannot perform any sort of graphics commands.

AfterStartUp

In this Script, which is performed after your Card's window and other Objects (like Buttons) have been made, you may want to draw into your Card's window, activate a particular text field, start a BrushAnimation or sound playing, or perform other last-second steps before the user of your application begins to interact with it.

Leaving

The Leaving Script is performed anytime you switch Cards. This Script runs after your Card's Objects have been removed from the window. This is the proper place to save any files that were changed on the Card that is ending, before moving on to another Card or quitting entirely.

NOTE: You cannot put any Card Movement Commands in the Leaving Script.

4 Objects Index

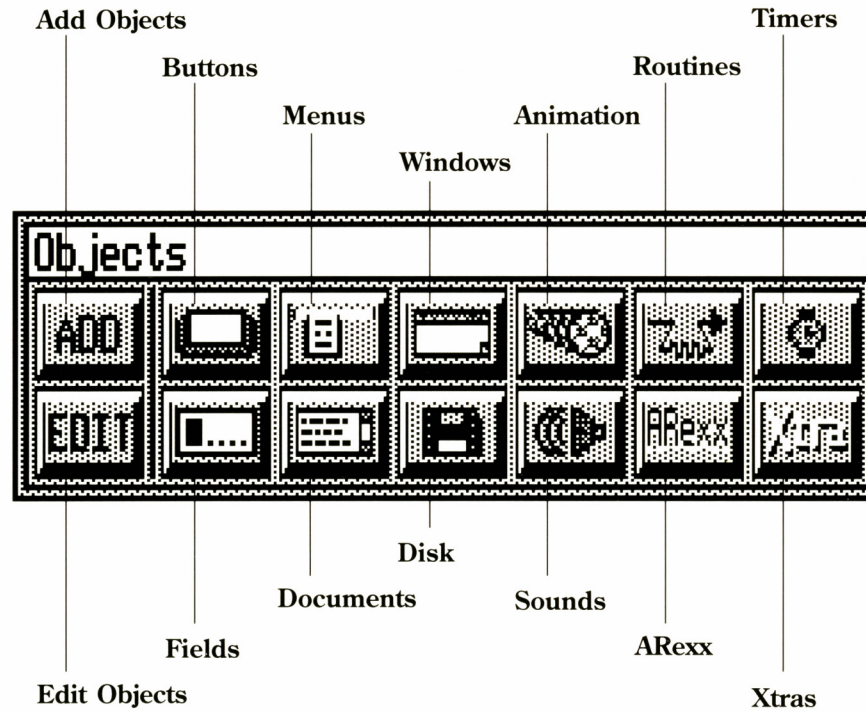
4 Objects

Lets start with some CanDo basics. Applications are comprised of a Deck of Cards. You can have as many Cards in a Deck as you wish. Each Card in the Deck has a Window. Like flash-cards, CanDo Cards are shown one at time.

It's up to you how each Card looks. They can look similar to each other or they can look completely different. CanDo gives you a lot of freedom using the Amiga graphics and sounds.

CanDo Objects, such as Buttons and Menus, are added to a Card allowing you to interact with your application. Other Objects, like Timers, allow you to set up events to control your application.

The Objects on a Card make things happen in your application. The Objects portion of the *Main Control Panel* allows you to Add and Edit the Objects.

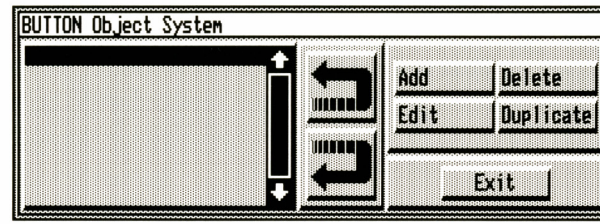


Objects Portion of the Main Control Panel

While in **Design** Mode, either the Add or Edit Button is highlighted. This indicates the selected method of working with Objects. When **Add** is selected, clicking one of the Object's Buttons allows you to add an Object.

Each Object has an *Editor Requester* which allows you to define how the Object appears and performs. While defining an Object, you give it a Name. CanDo uses the Name to identify the object. Each Object on a Card must have a unique Name. However, you can use the same Name for Objects on different Cards.

When **Edit** is selected on the *Main Control Panel*, clicking on a visible object (Buttons, Fields, Documents, Window Close Buttons or selecting a Menu) brings up the *Editor Requester* for the Object. Alternately, clicking one of the Object Buttons in the Control Panel brings up an *Edit List Requester* displaying the Names, of the selected Object Type, on the current Card. From this Requester you can Add, Edit, Delete, Duplicate and Reorder Objects.



Edit List Requester

The **Add** Button works just as Add from the control panel. Selecting the Add Button on the Control Panel is simply a shortcut to using the one on this Requester.

The **Edit** Button allows you to edit the currently highlighted Object in the list. Clicking this button brings up the *Editor Requester* for the Object Double clicking an entry automatically selects Edit.

The **Delete** Button deletes the highlighted Object. A *Requester* will ask you if you really want to delete the Object. Be certain that you want to delete it. The Object is disposed of completely. It does not put it in the Paste Buffer (see Object Menus).

The **Duplicate** Button makes an exact copy of the selected Object. The Object will be renamed in the same manner Workbench uses in duplicating files (ie. "copy of "). When you press the Duplicate Button, the *Editor Requester* will be displayed with the copy of the Object. This allows you to easily create similar Objects without starting from scratch each time.

You can also Reorder the Objects. Visible objects, such as Buttons, can be placed on top one another. The list shows the order in which Cando adds the Objects to the Card. Therefore, Objects that are lower in the list will appear on top in the window. The **Move Object Up** Button moves an object up in the list and the **Move Object Down** Button moves it down in the list.

Object Scripts

Objects make things happen by performing Scripts. Each Object has at least one Script that can be performed. Some can have more than one. The Object's *Editor Requester* contains either a button naming each Script or a Scripts Button that will display the available Scripts. You can tell if a Script already exists when the Button is black. Clicking the **Scripts** Button brings up the CanDo *Script Editor.*

Object Menus

Clicking on any area of the *Main Control Panel* (it does not have to be on a Button) allows you to access CanDo's menus using the Right Mouse Button. Under the Objects Menus, you can select from Browse, Add, Edit, Copy, and Paste. The Browse, Add, and Edit options perform in the same way as their corresponding Buttons on the *Main Control Panel.* The Menu equivalents provide convenience in that the Amiga Shortcut Keys of **Amiga**...**B**, **Amiga**...**A**, and **Amiga**...**E** can be used to select them.

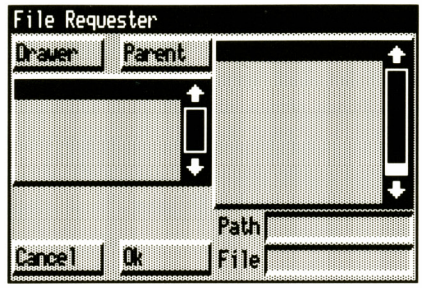
The remaining options are Copy and Paste. These Options allow you to Copy an Object into a Paste Buffer, and to Paste it onto another Card.

Selecting **Copy** allows you to copy an Object using two methods. First you can click on a visible Object on your card. After doing so, the previous Paste Buffer is replaced with the selected Object. Selecting one of your Menu Item copies all Menu Items and Subitems from the selected Menu into the Paste Buffer.

For the second method, after selecting Copy from the Objects Menu, you click on one of the Object Buttons in the Main Control Panel. This displays the list of Object Names of the selected Object Type. Double clicking an Object name, or clicking the Copy Button puts the Object into the Paste Buffer.

Selecting **Paste** from the Objects Menu adds the Object to the current Card. When Paste is selected, the Object's *Editor Requester* will be displayed. You can then modify the Object before selecting **Ok**, or select **Cancel** to abort the Paste operation.

At various times, CanDo will display its *File Requester* requiring you to specify the name and location of a file.



CanDo File Requester

A File specification has two parts: the Path and Filename. The Path Field shows where the file is, and the File Field shows the name of the file. You can type directly in these Fields or use the two Lists to set them.

The smaller List on the left sets the Path. You can use it to locate the Path using Drawers, Disks, Physical Devices, and Assignments. By default, the small list contains the list of Drawers or Sub-Directories in the directory indicated by the Path. Clicking one of its entries, sets the Path to the indicated directory.

Clicking the **Drawer** Button changes its name to Disks and the small list will display the available Disk Volumes. Clicking the **Disks** Button changes it to Physical. The list now displays the available physical devices. Not all of these can be used for accessing files! Clicking the **Physical** Button changes it to Assign. Now the list shows the available assignments. (See your Amiga documentation for setting these.) Finally, clicking the **Assign** Button sets the list back to Drawer.

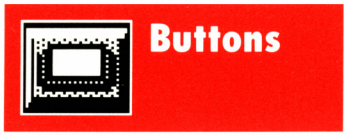
Clicking the **Parent** button changes the current Path to the parent directory if there is one.

The larger List on the right, allows you to select a File Name. Clicking an entry puts the name in the File field. Double-clicking an entry selects the name as though you clicked Ok.

Sometimes, the *File Requester* will have an additional preview button on it. For example: when you are suppose to find a picture file, there will be an additional button **Show It!**. Clicking it allows you to see the currently selected file. When finding a sound file, the **Hear It!** Button allows you listen to it.

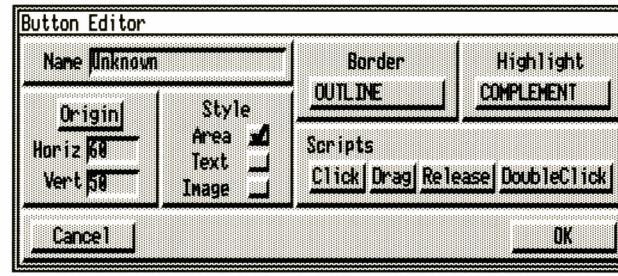
Clicking **Ok** selects the current file indicated in the Path and File Fields. If the File needs to be a specific type, CanDo will verify it and display an Error Requester when it is wrong.

While using the various Requesters within CanDo, you have the option of selecting Cancel or Ok. Ok accepts any changes you have made in the Requester. Cancel causes CanDo to forget any changes. Selecting either Cancel or Ok returns you to the previous Requester if there was one.



A Button is an area in a window that can be clicked with the Mouse Pointer. When clicked, specified actions can be performed. CanDo allows an unlimited number of Buttons to be defined on the Screen.

Selecting **Add** in the *Objects Portion of the Main Control Panel* allows Objects to be added to your application. If you click the **Button** Button on the same panel the *Add New Button Requester* will come up, directing you to locate the Origin Point of the New Button with your Mouse. Now click the **Ok** Button. The *Add New Button Requester* will be lowered revealing your full Screen. Next move your Mouse, positioning the Pointer at a location on the Screen where you want a Button. Now click the **Left Mouse** Button to set the Origin Point. Your Mouse now controls the shape and size of a rectangle representing a Button area. Move the Mouse in any direction you choose, adjusting the rectangle until you are satisfied with its size. If you don't like the Button for some reason, press the **Esc** key and start over. If the Button is to your liking, click the **Left Mouse** Button again and The *Button Editor Requester* will be displayed.



Button Editor Requester

The *Button Editor Requester* allows you to edit the unique information about each button. It displays the Button's Name, Origin, Button Style, Border and Highlight Styles, and the available Scripts. The origin coordinates of the Button, in X (horizontal) and Y (vertical) format, will have been entered for you automatically.

Name

The Name is a group of characters identifying the button. The same name can not be used for more than one Object on a card. The button is given a default name " Unknown ". However, you probably will want to name it something that is meaningful to you. The Name can include any character. It has a maximum length of 20 characters. When you press **Return**, CanDo will verify that the name is unique.

Origin

The Origin always indicates the location of the upper left corner of a Button. The Horizontal value indicates the distance in pixels from the left edge of the Window. The Vertical value indicates the number of pixels down from the top of the Window. The present values can be changed by clicking in these Fields and typing in new values with the keyboard.

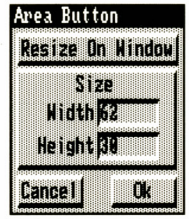
You can also set these values using the Mouse. First click the **Origin** Button, and the *Add Button Requester* will be lowered. Your Mouse now controls the position of a rectangle representing the Button Area. Now move the rectangle where you want the Button to be. If, however you choose not to change your present Button you can return to the *Button Definition Requester* by pressing the **Esc** Key. When the new location looks good just click the **Left Mouse** Button and the *Button Definition Requester* will reappear.

Style

CanDo has three Button styles: Area, Text and Image. When adding a Button, a rectangular area is defined. This area represents the "Hit Area" for an Area Button. Area Buttons do not have an image or text representing the Hit Area. However, they can be placed over images in the Window.

In addition to Area Buttons, CanDo allows you to use Text and Image Buttons for representing Image and Irregularly Shaped Buttons. These Buttons are described in greater detail in the Text and Image sections of this manual.

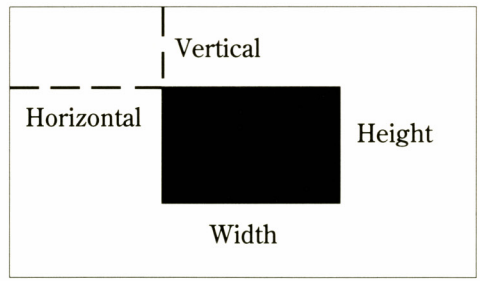
The check mark in the Box next to "Area" indicates the current Button is an Area Button. Each Button style has a Requester containing unique information for its particular requirements. Clicking the **Box** next to the indicated style, brings up the appropriate Requester and changes the Button style. Clicking **Cancel**, aborts this process.



Area Button Requestor

Clicking on the **Box** next to "Area" brings up a requester containing information about the size of the Area Button.

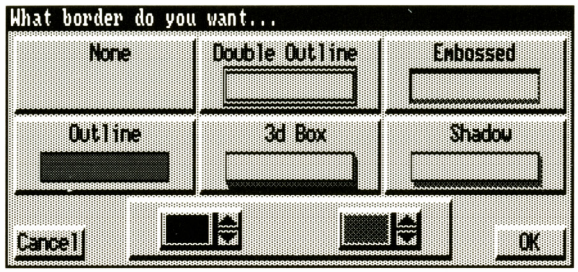
This requester allows you to view or change the Width and Height of the Area Button. These values can be changed using the Keyboard. However, they can also be set using the **Resize On Window** Button. Clicking on this Button lowers the *Main Control Panel.* Your Mouse is now controlling the lower right corner of a rectangular box. The upper left corner is stationary on the screen. This corner indicates the Origin of the Button. The rectangular box represents the Hit Area for the Area Button. By positioning the movable corner, and pressing the **Left Mouse** Button a new area for the button is defined. Pressing **Esc,** returns to the *Area Button Requestor* without modifying the Width and Height values.



Button Origin and Size

Border

An Area or Text Button can have a Border around its Hit Area. There are six Border Styles from which to choose. The default Border Style, Outline, is shown in the *Button Definition Requester.* By clicking on the **Border** Button, the *Border Selector,* will be displayed.



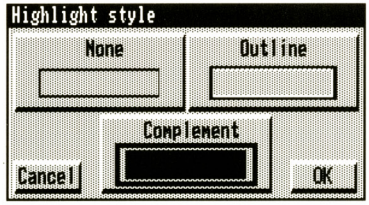
Border Selector

The *Border Selector* displays the border styles from which to select. The selected style will have a Black Hit Area. Clicking the Hit Area of the desired button selects the matching style.

The bottom row contains a Cancel Button, two *Color Selectors*, and an Ok Button. The two Color Selectors change the colors used in rendering the buttons border. The colors can be selected by using the two arrows on the Color Selectors or by clicking one of the available colors at the bottom of the Screen. The selected color is shown in the active Color Selector. Clicking on one of the Color Selectors makes it active. The sample buttons on the Border Selector depict the use of these colors.

Highlight

When you click on a Button that you have made with CanDo, the Button area can change to the existing color's Complementary color, or Outline the area, or do neither. The default Highlight Style, Complement, is shown in the *Add Button Requester.* By clicking on the **Highlight** Button the *Highlight Selector* will be shown.



Highlight Selector

The three Highlight Styles: None, Outline, and Complement are shown in this requester. Clicking on the associated button sets its Hit Area to Black.

Button Scripts

There are four types of events that can occur by using a Mouse Button. They are Click, Drag, Release, and Double Click. Each of these events can have a Script associated with them. A script is simply a set of instructions to perform.

Click

Click events happen when the **Left Mouse** Button is pressed while the mouse pointer is over the button. This type of event is useful for providing immediate response to the mouse.

Drag

Drag events occur as the **Left Mouse** Button is held down and moved over the button's Hit Area. This is an uncommon type of button usage. However, these movement events can be used for tracking mouse movements in applications such as paint programs.

Release

Release events occur when the **Left Mouse** Button is pressed and released while over a button. This is the most common button application. By delaying until the button is released, the user can decide to move the pointer off the button, thus avoiding an unwanted action.

Double-Click

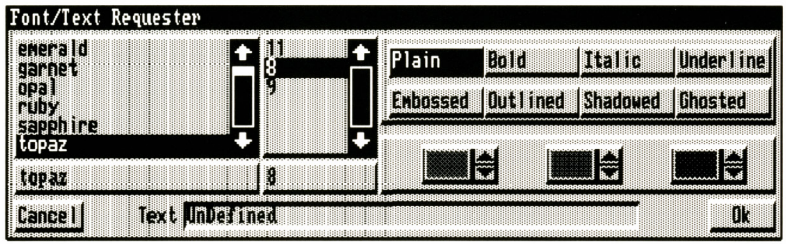
Double-Click events happen when the **Left Mouse** Button is clicked quickly two times. The Amiga Preferences allows a user to adjust the time delay used in determining a Double-Click. This type of event is used to insure that the user did not inadvertently press a button. The Icons on Workbench work in this manner.

Text Button

Like an Area Button, a Text Button has a rectangular Hit Area. However, CanDo will automatically display a text message in the hit area. You specify the text message, the font and point size, color, and style. The text is simply characters that you want to represent the action performed by the button. It can use any font available in your "Fonts:" directory and can also use a variety of Text Styles, to enhance its appearance. Like an Area Button, a Text Button can have a Border around its Hit Area.

An Area or Image Button can be changed to a Text Button by clicking in the **Box** next to "Text" in the *Button Definition Requester*. You can change the style of a Button as many times as you wish. When you have made your final choice there is one important thing to remember about this process: each Style has unique style information that must be saved. That information is saved only when you click **Ok** on the Button Definition Requester.

The *Text Button Definition Requester* is displayed when the **Box** next to "Text" is selected. A window on your screen displays the appearance of the text message.



Text Button Definition Requester

The *Text Button Definition Requester* allow you to select the font and point size, the Text Style, the colors used, and the text message to use in the button. Notice that you don't set the size of a text button. Its size is determined automatically by the size of the message in the selected font.

Selecting a Font and Point Size

The available fonts in your Fonts: directory are shown in a list. A font can be selected by clicking on its name with your mouse pointer. If you have more fonts than can be displayed in the list, the slider allows you to scroll through the list. The currently selected font is shown below the list. The point sizes available for that font are shown in the list next to the fonts. The selected size is shown below its list. When the font or size is selected, the sample text will be updated to show its appearance.

CanDo uses the current Fonts: directory. Some of you may have more than one Font directory. With applications such as Paint programs, you can change directories while working on a project. This is because the font is only used while you are rendering text to the screen. Afterwards, it is not needed anymore and you can change font directories.

Because CanDo uses the selected font when displaying a button, all fonts used by CanDo should be in the currently assigned Fonts: directory. If you change the fonts in this directory, remember to run FixFonts before using them. If the selected font can not be used, the system's default font will be used.

Selecting Text Style

The Amiga operating system supports Plain, Bold, Italic and Underline text styles. CanDo provides enhanced styles of Embossed, Outlined, Shadowed, and Ghosted.

Selecting Plain deselects all other Text Styles. Bold, Italic, and Underline can be used with each other along with one of CanDo's enhanced styles. Only one enhanced style can be used at a time. The Sample text shows the results of the selected styles.

Selecting Text Colors

The *Text Selector* has three *Color Selectors*. The first one is the primary color. It is used with all styles for depicting the text. The other two Color Selectors are used with the enhanced styles. The colors available in the Color Selectors are the ones currently used in your screen. Different color combinations provide different effects. Playing around with different styles and colors will give you the feel of how to use them.

Selecting Text

The Text defaults to the same name used for the button. However, it doesn't have to be the same. The text can be changed using the keyboard. Spaces at the beginning and the end of the text can be used to make the button larger.

Selecting **Cancel** does one of two things. If the Button was previously of a different Style the original Button Style will be restored. If it was already a Text Button, any changes made will be forgotten. Selecting **Ok** accepts any change.

Image Button

Image Buttons have a small picture for a Hit Area. A "DPaint" style Brush is needed to define the picture for an Image Button. Most paint packages allow a portion of a picture to be clipped and saved in a file. These small pictures are called Brushes. CanDo allows a Brush to be used as an Image Button. When a Brush is clipped from a picture, the background color is transparent. While the Brush is rectangular, the Image appears to exclude this one color. CanDo supports this technique in utilizing the Brush as a Button.

While the background color is drawn, it is not included in the Hit Area. This means that when the mouse is clicked on the background portion of the image, the button is not activated. This allows irregular shapes, such as arrows, to be used without including the area around the image as a Hit Area.

An Image Button is created by clicking in the **Box** next to "Image" in the *Button Definition Requester.* This will display the *Image Button Requester.*

Image Button Requester

The *Image Button Requester* allows you to select a "DPaint" style brush file for the image. The Image Name Button indicates the Name of the Brush File. By clicking the Button, you can locate the file using CanDo's *File Requester.*

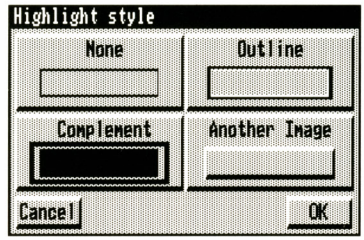
This Requester has a Button called "Show It!" The Show It Button loads the Brush File and displays it in a Window on your Screen. This allows you to see what the brush looks like. Don't be alarmed if a large brush is not completely visible in the Window. The purpose of this feature is for you to visually verify its appearance. If it is not a valid Brush File, CanDo will display an Error message. If the Brush does not use the same display mode or color palette as your screen, it might not appear as you expect. Keep in mind that the Amiga can use only one display mode and one color palette on a single screen. However, with some planning, you shouldn't have too much trouble getting predictable results.

CanDo allows you to use the shape of a brush as the hit area, without using it's image. Clicking the **Box** next to "Use Shape Only" toggles this feature. By selecting this feature, the non transparent colors in the brush image define the hit area of the button. However, the image of the brush is not rendered on the screen. This feature allows irregularly shaped buttons to be used without changing the image on the screen.

As with the *Area* and *Text Button Definition Requesters*, clicking **Cancel** aborts a style change or of any changes made in the *Image Button Definition*. Clicking **Ok** accepts changes and returns to the *Button Definition Requester*.

Alternate Image Button

An Image Button, that is not using the "Use Shape Only" feature, has an additional Highlight option. Clicking on the **Button** in the "Highlight" section of the *Button Definition Requestor*, displays the *Highlight Selector*.

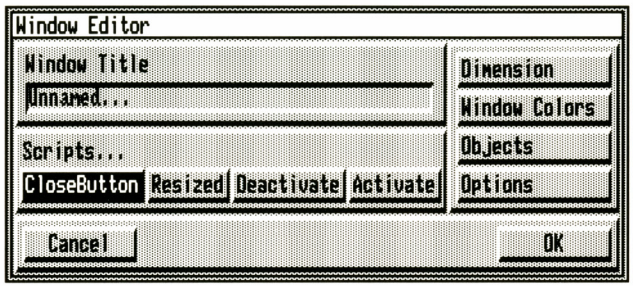


Highlight Selector with Alternate Image

However, this time the additional option, Alternate Image is available. Clicking on this Button displays CanDo's File Requester, allowing you to locate the Brush File to use as the Alternate Image. The Show It Button allows you to preview the Image. It is suggested that the Brush used for the Alternate Image be of the same dimensions as the Brush used for the Image Button. Otherwise, portions of an image will not be cleared when the image changes.

A Card always has a Window. The Window Object allows you to customize each Card's Window. You can change the resolution, number of colors, or provide a background image. You can also have close, resize and depth buttons. Several other options allow you to tailor the appearance of the Window.

Selecting the **Window** object from the *Main Control Panel*, brings up the *Window Editor Requester.*



Window Editor Requester.

The Window Editor allows you to specify a Title for the Window. Along the right side of the *Window Editor Requester*, are four Buttons: Dimension, Window Colors, Objects, and Options. Each of these Buttons opens a Requester allowing you to control specific aspects of the Window. The Window can have four scripts associated with it: CloseButton, Resize, Deactivate, and Activate.

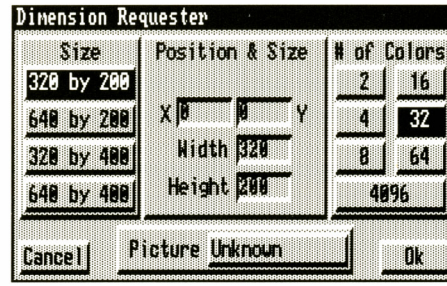
Window Title

This Field allows you to specify a Title for the Window Title Bar. If the Window Title is empty, (you can do this easily by clicking in the Field and pressing the **Amiga**...**X**) no Title will appear. In addition, if you do not specify a close button, depth buttons, and dragbar (see Window Objects) and the Window Title is empty, the Window will not have a Title Bar.

If you want to insure that you have a Title Bar, yet you don't want to have a Title, put a space in the Window Title Field.

Dimension

Clicking the **Dimension** Button on the *Window Editor*, brings up the *Dimension Requester.*



Dimension Requester

The *Dimension Requester* allows you to specify the Window size, and number of colors. Alternatively, you can specify a Background Image. In addition, you can specify an initial position.

Display Mode

When providing the Width and Height, CanDo automatically determines the Display Mode. The Amiga Display modes are Low-Resolution, Extra Half-Bright, Hold and Modify (HAM), and High-Resolution.

Widths up to 320 can be Low-Resolution, Extra Half-Bright, or HAM. The Display mode is determined by the number of colors. If you choose 32 colors or less you will be in Low-Resolution Mode. Choosing 64 colors produces Extra Half-Bright Mode. Finally, 4096 colors gives you HAM Mode.

Widths greater than 320 requires High-Resolution Mode and can have a maximum of 16 colors. Interlace is used with Heights greater than 200 (256 on PAL Amigas). On the left side of the *Dimension Requester*, four common screen Sizes can be selected. Clicking on the appropriate Button sets the Width and Height to the indicated values.

Position

The X and Y values are the initial position of the Window. Most application will use the default values of 0 (Zero). If the Window has a dragbar (see Window Objects), it can be repositioned with the mouse.

The X value is only used when the Window is opened on Workbench (See Window Options). It indicates the initial Horizontal position of the Window. The Y value indicates the initial vertical Position. If the Window is opened on Workbench, it will be used for the vertical position of the window. Otherwise, it will be the initial vertical position of the new screen.

Picture Window

Clicking the **Picture Window** Button brings up the CanDo's *File Requester*. The specified Image will be used as background image for the Window. The Window's size and number of colors will be determined by the picture. When a Picture Window has been selected, the Picture Window Button will be highlighted. Changing the Width or Height values, or Number of Colors will de-select the Picture Window.

Clicking **Ok** or **Cancel** on the *Dimension Requester* returns to the *Window Editor.*

Window Colors

Clicking the **Window Colors** Button on the *Window Editor* brings up the *Color Requester.* It allows you to change the color used for the Window's Background, Border and Text. These values can be set to the Color number you want to use.

Color Requester

Background Color

The Background Color is the initial color of the Window. If you specify a **Picture Window** in the *Window Dimension Requester*, this value will not be used.

Border Color

This is the color for drawing the Window's border and Title Bar. If the Window does not have a border or Title Bar (see Window Options), this value will not be used.

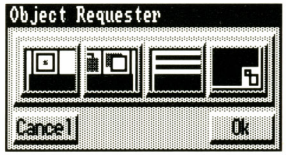
Text Color

This color is used for the Window's Title. If the Window does not have a Title Bar, it will not be used.

Clicking **Ok** or **Cancel** will return to the *Window Editor*

Window Objects

Clicking the **Objects Button** on the *Window Editor* brings up the *Object Requester.*



Object Requester

This Requester allows you to specify optional Window Objects. There are, from Left to Right: Close Button, Depth Buttons, Dragbar, and Resize Button. Click on the **Buttons** of your choice. The selected object will be added to your Window.

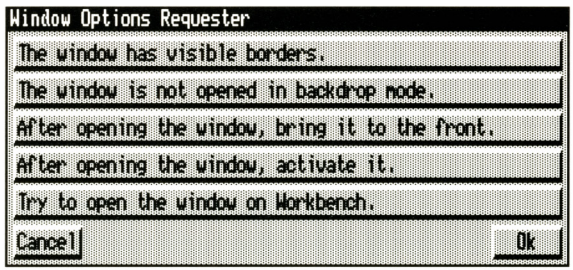
The Close Button object allows a Close Button script to be performed when clicked. The Depth Buttons allow the Window to be pushed to the back or brought to the front. The Dragbar allows the Window to be dragged. The Resize Button allows the size of the Window to be adjusted. Note: If a Resize Script exists, the Script is performed whenever the Window is resized.

The Depth Buttons, Dragbar and Resize Button are most useful when used with a Workbench Window. While they are not restricted to the Workbench Window, they are used for working with more than one Window on a screen.

Selecting **Ok** or **Cancel** returns to the *Window Editor.*

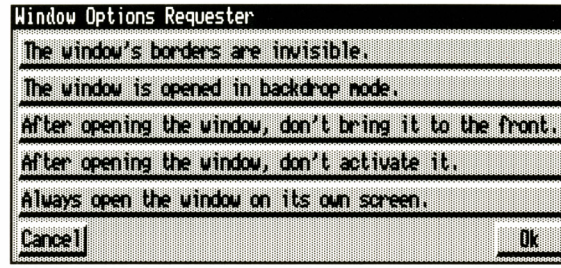
Window Options

Clicking the **Options** Button on the *Window Editor*, brings up the *Window Options Requester.* This Requester allows you to select various options effecting the Window. The default Options are shown below.



Window Options Requester

Clicking a Button changes the option. Each Button toggles between two choices. The following image shows the alternative to the default Options.



Alternate Window Options

Visible Borders

The Window has an optional border. When visible, a box is drawn around the Window using the color specified in Window Colors. If the border is invisible, the box is not drawn.

Backdrop Mode

A backdrop Window means the Window is attached to the screen. The window will not have a Window Title Bar and none of the Window Objects will be attached. If this is a Workbench Window, the window cannot be positioned on top of other windows nor can it be moved around. When the Window is opened on a Custom Screen, you will have access to the Screen Title Bar allowing the Screen to be lowered using the Mouse.

Bring Window to Front

This option effects whether the window is visible when it is opened. On a Workbench Window, the Window is allowed to be opened in front of, or behind any Windows on the screen. On a Custom Window, it affects whether the Screen is in front of or behind other Screens. You can use the WindowTo and ScreenTo Commands to change the position of the Window or Screen.

Window Active

The Amiga can only have one active window at a time. This Window receives keyboard input, and its menus are available to be used. The Window Activate option allows you to control whether your Window is the Active Window when it is first opened. Most of the time you will want it to be Active. However, you may wish to make a CanDo application that is started from a CLI and has a Status Window on Workbench. If you want to be able to continue typing in the CLI, without reactivating it, then you do not want your CanDo Window to become Active.

Workbench Window

This option tells CanDo to try to open the Window on Workbench. To do so, the Window must use 2 or 4 colors. In addition, if your Workbench Screen is non-interlace, the Window height can not be greater than 200 (256 on PAL Amigas). If the Window can not be opened on Workbench, CanDo will open a new screen.

Clicking **Ok** or **Cancel** returns to the *Window Editor.*

Window Scripts

The Window has four scripts that can be performed. The CloseButton, Resize, Deactivate, and Activate Scripts can be edited by clicking the appropriate Button on the *Window Editor.*

CloseButton

By enabling the CloseButton in Window Objects, the Window will have a CloseButton in the upper left corner. This script is performed when the it is clicked on.

Resize

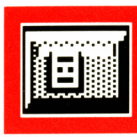
When the Window is resized, you may need to redraw your Window. Buttons, Fields, and Documents are redrawn automatically. However, all other imagery is not. This script can contain commands to redraw the imagery in the Window when it is resized.

Deactivate

When your Window is Active, and you click in any other Window, the Deactivate Script is performed.

Activate

When a user re-activates the Window this script is performed. The Deactivate and Activate scripts can be used to pause an activity while the another application is being used.

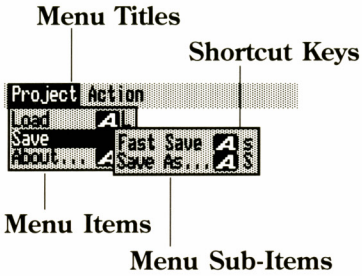


Menus

CanDo allows you to create menus for your applications. Most likely, you're familiar with using the menus provided in most of the software on your Amiga. CanDo lets you create Amiga menus with the features you have become accustomed to, and features such as Menu Images that even professional software seldom utilizes.

Features Provided in CanDo Menus

- Multiple Menu Titles.
- Menu Items and Subitems.
- Shortcut keys.
- Stylized Text Fonts.
- Menu Images.
- Alternate Text and Alternate Image Highlighting



Menu Features

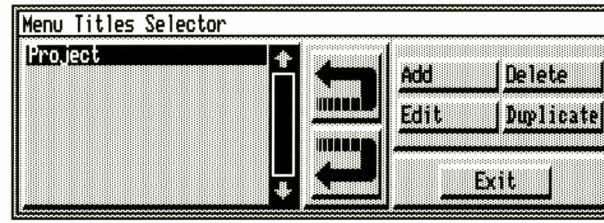
Menu Components

Menu Titles are visible on the Title Bar of a window when the right mouse button is pressed. They are text characters providing the context and location of the available menus. CanDo allows you to have as many Menu Titles as can be displayed on the Title Bar. When the Mouse Pointer is positioned over a Menu Title with the **Right Mouse** Button pressed, its Menu Items become visible.

Each Menu Title will have at least one Menu Item. The Menu Items are the selectable list of Text or Image entries associated with a Menu Title. A Menu Item can have Menu SubItems. They become visible when the mouse is positioned over the Menu Item. Like Menu Items, they can either be represented by Text or an Image. Each Menu Item and Subitem can have a script that is executed when it is selected. In addition, they can have a Shortcut Key that causes its script to be executed as though it had been selected using the mouse.

Adding and Changing Menus

Menus are added and edited through Requesters. Unlike Buttons and Fields, it makes no difference whether "Add"'or "Edit" is selected in the *Objects Control Panel.* When the **Menu** Button is selected from the Objects Control panel, the *Menu Titles Selector* is displayed.

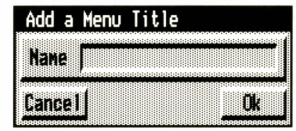


Menu Titles Selector

The *Menu Titles Selector* allows the creation, deletion, reordering, and renaming of the Menu Titles. It also provides access to the Menu Items for each Menu Title.

If you have two or more Menu Titles, the Movement Arrows allow you to change their order. The top entry will appear at the far left side of the *Menu Bar.* The bottom entry will appear on the right side.

Selecting **Add** or **Edit** brings up the *Menu Title Editor.* It contains the Menu Title. The Menu Title is the text that appears on the Title Bar.



Menu Title Editor

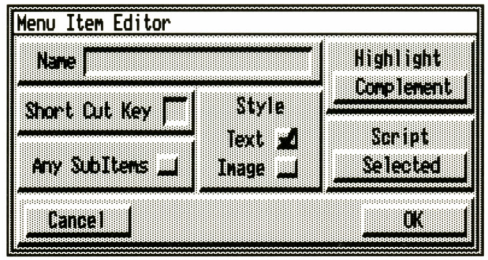
When **Edit** is selected the Menu Title Editor will contain the name of the selected Menu Title. If you want to re-name it, simply change the name in the Field.

Selecting **Cancel** returns to the *Menu Titles Selector.* Selecting **Ok** brings up the *Menu Item Editor* for the Menu Title.

Menu Items

The *Menu Items Selector* works in the same manner as the *Menu Titles Selector.* Menu Items are created, deleted, reordered using this Selector. A Menu Title must have at least one Menu Item. If you don't Add one before selecting **Exit** the Menu Title will not be created. Should you ever delete all Menu Items from a Menu Title, the Title will be removed.

Choosing **Edit** or **Add** brings up the *Menu Item Editor Requestor.* Add allows you to create a new Menu Item. Edit allows you to change an existing one.



Menu Item Editor Requester

The *Menu Item Editor Requester* allows you define and later modify the features of a Menu Item. From this Requester, you specify the Object Name, Highlighting Style, menu style, and a script to be performed when it is selected. You can optionally define a Shortcut Key and Menu Subitems.

Menu Item Name

As with other Objects, the Object Name here provides a unique group of characters identifying the Menu Item. It is the name that will be shown in the *Menu Items Selector* for the Menu Title.

Menu Style

Menu Items can be represented by an Image or Text. The check mark in the box next to 'Text" indicates it as the default style. Clicking on this box brings up the *Menu Text Definition Requester.* It works the same way as the *Text Button Definition Requester.* The Text field defaults to the Menu Item Name. However, it can be altered to whatever you like.

Clicking on the **Box** next to "Image" brings up CanDo's File Requester. It allows you to specify the "DPaint" style Brush you want to use as the Menu Image. Clicking the **Show It** Button will show you a representation of the Image on your Screen. As with all brush images, it uses the Screen's resolution and palette.

Menu Highlight

With the **Right Mouse** Button depressed and the Mouse Pointer positioned over a Menu Item, it will be Highlighted. The Highlighting style is indicated in the Button below "Highlight" in the *Menu Item Definition Requester.* Clicking the **Highlight** Button brings up the *Menu Highlighting Requester.* This Requester allows you to select from the available styles.

Both Text and Image items can have highlighting styles of None, Outline, and Complement. The Default style is Complement. These styles work the same way as their Button Object counterparts.

Text Menus have the additional Highlight Style of Alternate Text. Simply type the text into the field in the Alternate Text area of the Highlight requestor. The Alternate Text uses the same font and style as the Primary Text.

Image Menus have the additional Highlight Style of Alternate Image. Clicking on the **Alternate Image** Button brings up CanDo's *File Requester* allowing you to select a "DPaint" style brush file. While it is not necessary, it is suggested that the brush be of the same dimensions as the first Image used. If they are not, the images will not erase each other completely when highlighted and un-highlighted or selected and released.

Selected Script

A Menu Item has a single script that is performed when it is selected. Clicking on the **Selected** Button in the Script area brings up the *CanDo Editor* allowing you to edit the Script.

The Menu Item is selected by pressing the **Right Mouse** Button and positioning the Mouse Pointer over the Item then releasing the Button.

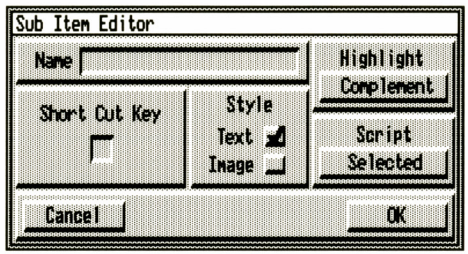
Shortcut Key

CanDo allows you to assign a **Shortcut** Key to a menu item. All you have to do is type a single character in the "Shortcut Key" Field. CanDo will display the Shortcut Key symbol on the right side of the Menu Item. Pressing the **Right Amiga** Button and the **Specified** Key, performs the Menu Item's selected script as though it had been selected using the Mouse.

Menu Sub-Items

A Menu Item can have a group of selectable Menu Sub-Items. By depressing the **Right Mouse** Button while the Mouse Pointer is positioned over the Menu Item, the Menu Sub-Items become visible. The Mouse Pointer can then be positioned over the Sub-Item you want. When the Right Mouse Button is released, the Sub-Item is Selected.

To Add or Edit an existing Menu Item's Sub-Items click on the **Box** next to "Sub-Item". This brings up the *Menu Sub-Item Selector.* It works identically to the *Menu Item Selector.* Selecting **Add**, brings up the *Menu Sub-Item Editor Requester.*



Menu Sub-Item Editor Requester

It allows you to define the Menu Sub-Item. This definition process is the same as the *Menu Item Editor Requester.* The only exception is that Menu Sub-Items can not have additional Sub-Items. This is a restriction enforced by the Amiga Operating System.

After a Menu Sub-Item is Added or Edited, clicking on **Ok** will return you to the *Menu Sub-Item Selector.* Exiting the Menu Sub-Item Selector returns to the *Menu Item Editor Requester.*

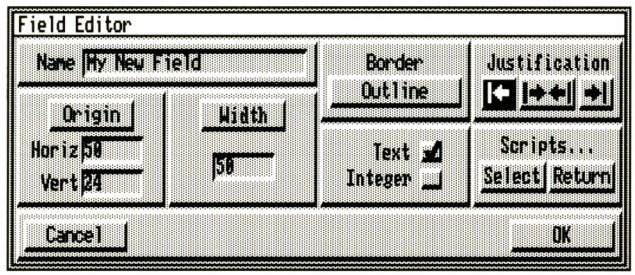
When a Menu Item has Menu Sub-Items, a check will appear in the Box next to Sub-Items. If all Sub-Items are deleted, the check will be removed.

One last thing to remember: when a Menu Item has Sub-Items, it can not be selected; only one of its Sub-Items can be selected This means that the Selected Script will not be performed when the mouse is released over the Menu Item.

Fields

A Field is an Area in which characters can be typed using the keyboard. CanDo provides two types of fields, Text and Integer. Text Fields allow any character, alphabetic or numeric, to be entered. Integer Fields are restricted to positive or negative numeric (integer) values.

Click the **Add** Button in the *Objects Panel* to tell CanDo you want to add an Object. Next, Click the **Field** Button, telling CanDo you want to add a Field. When the *Main Control Panel* is lowered, position the Mouse Pointer where you want the upper left corner of the Field. Click the **Left Mouse** Button. Now when you move the Mouse Pointer, CanDo will display a rectangular box representing the Field you are creating. Now click the Mouse Button again. The Screen will display the *Field Definition Requester.* Pressing the **Esc** key before you define the field area, returns to the *Main Control Panel* without adding a field.



Field Definition Requester

This requester contains the Field's Name, Origin, Width, Border Style, Justification, Type, and the available Scripts. The Name, Origin, and Border Style work the same as in the *Button Requester.*

The Name is a unique identifier for this Object. The Origin values can be altered directly with the keyboard or dynamically set using the Mouse after clicking on the Origin Button. As with Buttons, you can set the Field's Border by clicking on the Text Button showing the current Border Style.

The Font used within a Field is determined by a setting within your preferences. Most likely, you have selected 80 column text. If so, the fields will use Topaz 8. The characters in this Font have a height and width of 8 pixels. If you have 60 column text selected, the fields will use Topaz 9. Its characters have a height of 9 and width of 10 pixels.

Field Width

The Field's Width is shown in pixels. The number of displayable characters is this width divided by the character width of your system font. While it is not mandatory, it is best if the Field's Width is evenly divisible by the font's width.

Example:
If your Field is 64 pixels wide, devide 64 by 8 (64/8) = 8

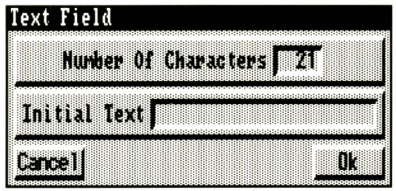
The Field's Width can be altered using the Keyboard, or resized dynamically on your screen using the Mouse. Clicking on the **Width** Button, lowers the *Control Panel.* The Mouse controls the width of a box representing the Field. The left side of the box is fixed at the Origin. The right side is adjusted with the Mouse. When the rectangle is the desired width, click the **Left Mouse** Button. If you don't want to resize the width, press the **Esc** Key.

Type Alignment

The information in a Field can be aligned Flush Left, Centered, or Flush Right**.** The Default Alignment is Flush Left. This can be altered by clicking on the button in the Alignment Area.

Text Field

The default Field type is Text. This is indicated by the check mark in the Box next to "Text". Clicking on this **Box** brings up the *Text Field Requester.* It allows you to define the maximum number of characters in a Field, and the *Initial Text.*



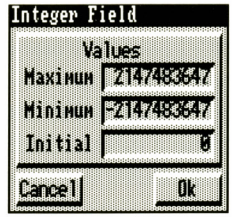
Text Field Requester

The number of visible characters is determined by the field's width. The Maximum Number Of Characters indicates the actual number of characters the field can contain. If it is greater than the number of visible characters, the contents will scroll as you type or use the arrow keys.

The *Initial Text* allows you to specify the characters in the field when it is created. However, a user can still alter the Field's contents.

Integer Field

From the *Field Definition Requester,* Clicking in the **Box** next to "Integer" will bring up the *Integer Field Requester.* It allows you to specify the Maximum, Minimum, and Initial values for the Field.



Integer Field Requester

The *Integer Field* allows a user to type any valid integer. You can automatically insure that the value is within a range by specifying the upper and lower boundaries in the Maximum and Minimum values.

Example:

If you set the Maximum Value to 1000 and the Minimum Value is 50, and the user enters a value of 10, when he presses **Return**, the value is changed to 50. If you entered 1500, it would be changed to 1000.

Using the Maximum and Minimum values, Scripts can safely assume a value is within the defined range. It also provides the user with immediate feed back of the value being used.

The *Initial Value* works similarly to the *Initial Text.* It provides a default value for the Integer Field. This value can be changed by the user if he so chooses.

Field Scripts

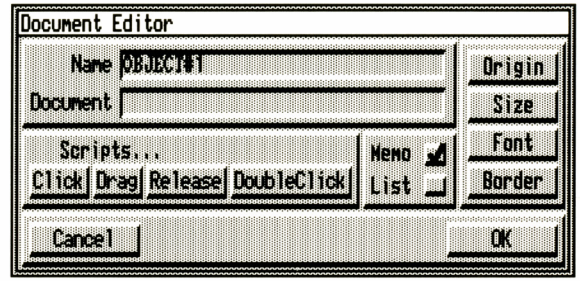
Fields have two types of scripts, Selected and Return. The Selected type of Script is executed when you click in the **Field**. The Return type of Script occurs when the user presses the **Return** Key.

CanDo's Document Object allows you to choose between a Memo editor and a List selector.

The Memo editor is a multi-line text editor with optional scroll bars. It provides for free form text input and display. The **No Typing** option allows for a non-editable display.

The List Selector displays a list of lines. The user of your application can choose from this list by clicking with the mouse. File Requestors use this technique to show a list of files.

Both the Memo editor and the List selector use CanDo Documents. The Document commands give you flexibility in the creation and manipulation of the Document's text. These commands are described in the Scripting Commands section.

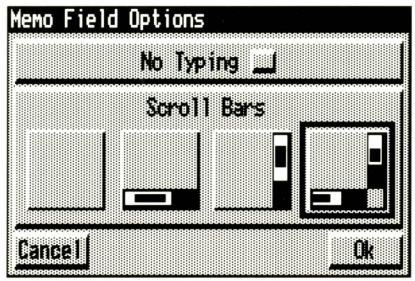


Document Editor

As with other Objects, the Name should be a unique name for an Object for the Card. The Document field indicates the "Document Name" to be used for the Object. You should read more about "Document Name"'s in the Document Scripting Commands. However, suffice it to say that this field indicates which "Document Name" to associate with this Document Object. If the "Document Name" has not already be created, CanDo will create a new one. When this is necessary, CanDo will look for a file using the "Document Name" as the file specification. If the file exists, it will automatically be loaded and displayed in the Document Object. Otherwise, an empty document will be created.

Document Type

The *Document Editor* allows you to choose between Memo and List. A check mark indicates the current selection. When you click on the **Box** next to Memo, the *Memo Field Options Requester* is displayed.



Memo Field Options Requester

The *Memo Field Options Requestor* allows you to choose between the Scroll Bar Options. It also allows you to select the No Typing option.

Clicking the **Box** next to No Typing enables and disables the option. A check mark indicates that the user of your application can not type into the Memo editor.

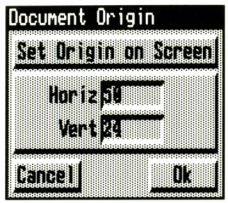
The Memo Field Editor has four Scroll Bar Options from which to choose. They are None, Horizonal Only, Vertical Only, and Horizonal & Vertical. Clicking on the appropriate Image chooses the option. Clicking the **Box** next to List does not display a Requester.

Document Definition

Four Buttons along the right side of the *Document Editor Requester* allow you to define the Origin, Size, Font, and Border for the Document Object.

Document Origin

Clicking the **Origin** Button brings up the *Document Origin Requester.*

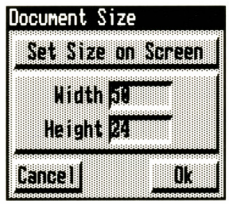


Document Origin Requestor

The Horizontal and Vertical Fields allow you to specify the location of the Document. The Horizontal Field indicates the number of pixels from the left edge of the window. The Vertical Field indicates the number of pixels down from the top of the window. You can reposition the document by clicking the Button **Set Origin On Screen**. Your mouse will move a rectangular box the size of your document. Position the box and click the mouse, or press **Esc** to abort. The Horizontal and Vertical Fields will be updated to the origin position.

Document Size

Clicking the **Size** Button displays the *Document Size Requester.*

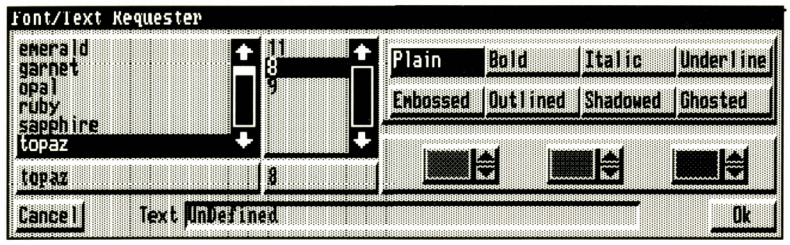


Document Size Requestor

The Width and Height Fields allow you to specify the document's dimensions. These Fields show the current values. You can modify them directly. In addition, clicking the Button **Set Size On Scree**n allows you to dynamically set them using the Mouse. The Mouse controls the corner opposite the origin point. Adjust the size of the box and click the mouse to set the values or press **Esc** to abort.

Document Font

Clicking the **Font** Button displays the *Document Font Requester.*



Document Font Requester

The *Document Font Requester* allows you to choose the Font and point size for the text to be displayed in the Document. You can also choose between Plain and a combination of Bold, Italic, and Underlined Text. Finally, you can choose the colors used for the text. The *Color Selector* on the left is for the Text and the one on the right is for the Background.

Document Border

Clicking the Border button displays the same Border Requester used for buttons.

Document Scripts

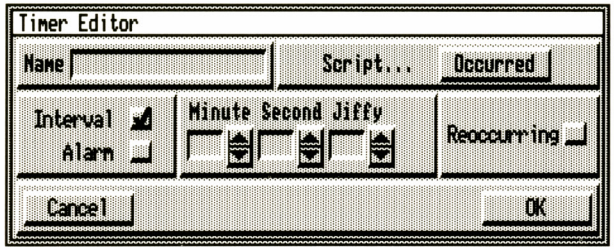
The Document Objects use the same Scripts as Buttons: Click, Drag, Release, and Double. They work in the same manner. When the user first presses the Left Mouse Button, the Click Script is performed. While they hold the Mouse Button down and move the pionter over the document, the Drag Script is performed. When they release the Mouse Button, the Reasease Script is performed. And finally, if they double click on the document, the Double Script is performed.

CanDo has two types of timers: Interval and Alarm. Interval timers go off after a length of time has elapsed. Alarm Timers go off at a specified time of day. When a Timer goes off, the Occurred Script is performed.

The *Timer Editor Requester* has two forms. Some of the titles and buttons are changed when selecting between Interval and Alarm Timer. Interval is the default setting. It is indicated by the check mark next to "Interval."

An Interval Timer can go off once after the elapsed time or it can be Reoccurring. This causes it to occur repetitively after the each interval. A Reoccurring Interval Timer, with a short elapsed time, can cause seemingly continuous activity while allowing other Object's Scripts to be performed if necessary.

When Interval is specified, the *Timer Editor Requester* allows you to specify the time interval and whether or not the timer is re-occurring.

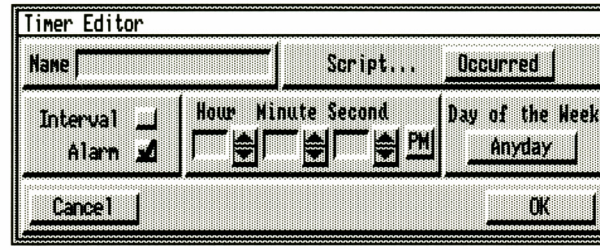


Timer Editor Requester

The time interval is specified in Minutes, Seconds, and Jiffys. Jiffys are fractions of a second. On NTSC Amigas, the U.S. standard, Jiffys are 1/60th of a second. On PAL Amigas, Jiffys are 1/50th of a second. The values can be set directly using the keyboard, or modified using the increment / decrement buttons next to each field.

Clicking the **Box** next to "Reoccurring", causes the Interval Timer to repeat continuously. For example: If the interval is set to 2 seconds, 00:02:00, the Occurred script will be performed every 2 seconds. Clicking the Box next to "Reoccurring" toggles it On and Off. A check in the Box indicates that it's on.

Clicking the **Box** next to "Alarm" indicates the Timer should go off at a specified time. The *Alarm Timer* can be set to occur every day or on a specific day of the week.

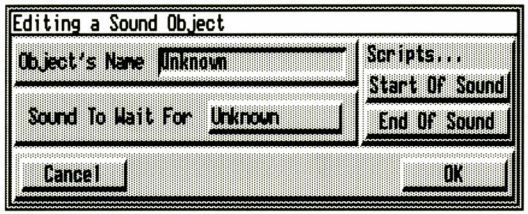


Timer Editor Requester

The Hour, Minute, and Second Fields let you indicate the Time for the Alarm. These times can be set using the keyboard, or they can be set using the Buttons next to each Field. The Alarm's Day Segment can be set by clicking the buttons **AM** or **PM** which toggle between the two.

Alarm Timers can be set for every day or for a specific day of the week. The default setting is Every Day. By clicking the **Day of the Week** Button, it cycles through each day of the week and Every Day.

The CanDo Sound Object allows you to synchronize other sounds and graphics with a sound. The Sound Object does NOT play the sound. It simply allow you to perform scripts when a specified sound starts or finishes playing. While it does not play the sound, it does load it into memory, if it is not already loaded there. When adding or editing a Sound Object, CanDo will display the *Sound Editor Requester.*



Sound Editor Requester

Clicking the **Button** next to "Sound To Wait For" brings up CanDo's *File Requester* allowing you to locate the sound associated with this object. It must be a valid 8SVX sound. 8SVX is the IFF sound standard supported by the Amiga.

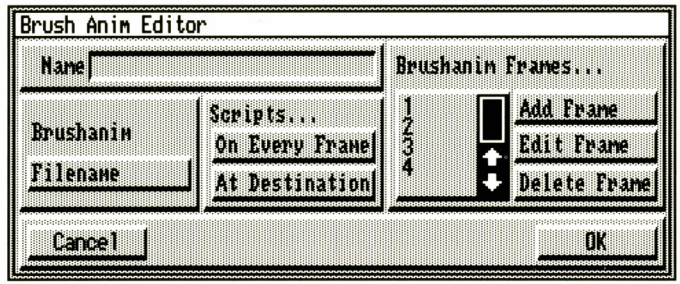
Also on the *File Requester,* you can press the **Hear It!** Button to preview the sound. After pressing **Ok**, the *Sound Editor Requester* will reappear and display the name portion in the "Sound To Wait For" Button.

The Sound Object waits for the specified sound to begin or finish playing. Any script on the current card can play the sound using the PlaySound or PlaySoundSequence Commands. However, they must use the exact same file specification as the "Sound Name". If the same sound is played using a different file specification or "Sound Name", the Sound Object's scripts will not be performed.

Scripts

A sound is played using the PlaySound and PlaySoundSequence Commands. (see the Sound Editor Tool and Sound Commands.) Whenever the specified sounds begins to play, the Start of Sound script is performed. When the sound is completed, the End of Sound script is performed.

CanDo allows you to display DPaint III style BrushAnims. The Animation Object does NOT display the BrushAnim. It simply allows you to perform scripts at specific points in the animation. This lets you coordinate other activities with the animation. Selecting the Animation Button form the Main Control Panel, brings up the BrushAnim Editor Requester.



BrushAnim Editor Requester

An Animation Object works with a single BrushAnim. The Animation Object's Scripts are triggered by the specified BrushAnim's activities. If you want Scripts associated with two BrushAnims, then you need to make two Animation Object's. Clicking on the **Filename** Button brings up CanDo's File Requester allowing you to locate a DPaint III style Brush Animation. Clicking the **Show It!** Button allows you to preview the animation.

Scripts

The Scripts for the specified BrushAnim can never be performed unless it is being displayed. The ShowBrushAnim Command can be performed in any Script except for a Card's StartUp Script. If you want it to be shown when you go to a Card you should put it the AfterStartUp Script. See BrushAnim Scripting Commands for more details.

The Animation Object allows you to perform a Script on each frame of an animation, on specific frames, or when a moving BrushAnim reaches a Destination.

On Every Frame ———

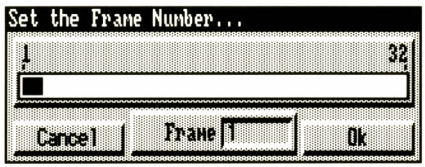
This Script is performed before each frame of the animation is shown. Depending on how long you make this Script, the animation may be slowed down dramatically.

On Destination ———

This Script is performed when the animation reaches the destination of a MoveBrushAnim or MoveBrushAnimTo Command.

BrushAnim Frames... ———

Individual Scripts can be performed before specific frames are shown. *The List Requester* shows the frame numbers for which there is a Script. The Buttons on the right hand side allow you to Add, Edit, and Delete Scripts from the list. Clicking **Add** displays the *BrushAnim Frame Requester.*



BrushAnim Frame Requester

It has a Slider allowing you to quickly choose the Frame Number. The selected Frame is shown in the Frame Field. You can also enter the Frame directly into the Field. If you click **Ok**, the *CanDo Script Editor* will be displayed. When you finish creating the Script, the *Brush Anim Editor* will be re-displayed.

CanDo provides a way of performing a Script when a diskette is removed from or inserted into the disk drive. This can be a fun way of playing different sounds when disks are removed and inserted. While it is not usually necessary, your application can monitor the available volumes. The Disk Object provides a way of performing these tasks.

Disk Object Editor

The Disk Object has two Scripts. Disk Removed and Disk Inserted. The scripts are performed whenever a diskette is removed or inserted from any drive.

Routines allow you to write a Script that can be performed by any Object. When creating the Routine, you give it a Name. The Script is performed using the Do Command with this name (see documentation on the Do Command for more details).

Unlike other Objects, the Routines Object is global to all Cards. This means that a Routine is accessible from every Card in your Deck.

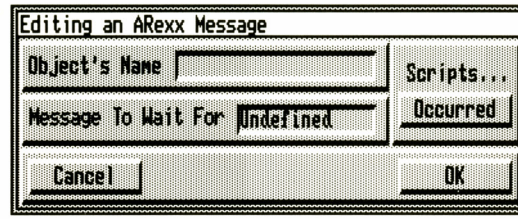
Routines keep you from having identical Scripts in different Objects. When you want multiple Objects to do the same thing, simply put the common scripting commands in a Routine. You can then access the Routine from any Object's Script using the Do Command.

Routine Editor

The *Routine Editor* has a field for the Routine's Name. The Do Command uses this Name to access the Script. When using the Do Command, put the Name in double "the name" quotes.

The Script Button allows you write the Routine's Script. The Script can contain Do Commands. While it is valid for a Routine to Do itself, you should not try this unless you are familiar with the advanced programming techniques. It is very easy to create an endless loop that uses all remaining memory.

Your application can listen to one ARexx port at a time. This port is specified using the ListenTo Command (see ARexx Commands for more information). Other Applications can send messages to this port. A message is simply an ASCII string. The first word of the message is the Command Word. When a message is received, CanDo uses the Command Word to see if you have ARexx Object that corresponds to it. If there is one, its Occurred Script is performed. Clicking the **ARexx** Button on the *Main Control Panel*, brings up the *ARexx Editor Requester*.



ARexx Editor Requester

The Object's Name Field is initially empty. If you do not fill it in, it will automatically be set to the word you put in Message Field when you press the Ok Button.

The Message Field should contain a single word identifying the message's Command Word. When a message is received, and its Command Word matches the word you put in this Field, the Occurred Script is performed. The script can then use the System Variable TheMessage which contains the complete text of the message just received.

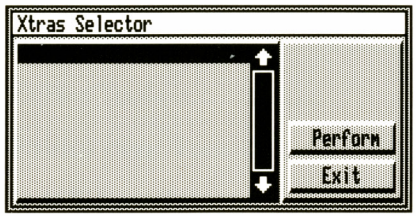
By making a series of ARexx objects, one for each Command Word that can be performed by your application, you can very easily create a complete ARexx server.

The Objects portion of the *Main Control Panel* contains the core group of Objects supported by CanDo. As more Objects are added, they will be accessible through the Xtra's Button.

In addition to Objects, the Xtras give you access to expanded operations. These Operations can also be accessed through the Xtra's Button.

Each CanDo Xtra has a file in the Xtras directory. As additional Xtras become available, you simply put its file in the Xtras directory. Because the Xtra Objects and operations are not individually documented in this manual, there will be documentation files on the disk.

Pressing the **Xtra** button on the *Main Control Panel* brings up the *Xtras Selector.*



Xtras Selector

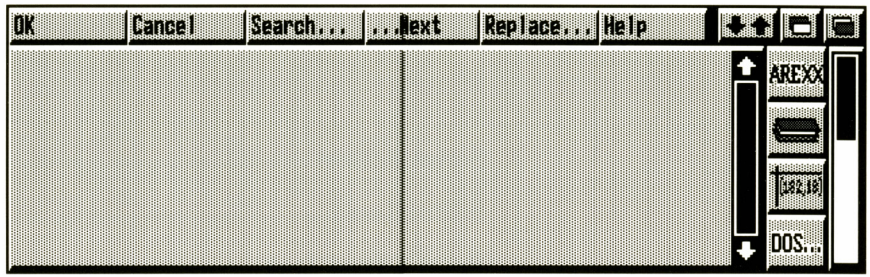
This Selector contains a list of the available Xtras. Simply **Double Click** an entry or click the **Perform** Button. When you do so, the selected operation will be performed. You should make sure you are familiar with what the Xtra is going to do by reading its documentation.

5 Script Editor Index

5 Script Editor

Every Object has at least one Script that can be performed. By clicking a button on the Object's Editor Requester, you can edit the Script using CanDo's Script Editor. The Script contains Commands telling CanDo what you want to happen when the Script is performed. CanDo's Scripting Commands are described in Chapter 6.

This Chapter describes how to use the Script Editor and Editor Tools.



CanDo Script Editor

Menus

The Script Editor works like most text editors. Using the keyboard and cursor keys you can type in your script. The vertical Slider on the right lets you scroll through a script.

The Editor has four Menus: Script, Text, Edit, and Misc. Some of their Menu Items have **Shortcut** Keys. These Menu functions can be invoked using the **Right Amiga** Key.

Script Menu

The Script Menu contains: Ok, Cancel, Verify, Clear, and Print.

Ok

Ok verifies the script and returns to the CanDo requester from which you invoked the Script Editor. If the script contains an Syntax Error, a requester will indicate nature of the error. When you select Continue, the cursor will be placed at the beginning of the problem line.

Cancel

Cancel returns to the previous requester without including any changes made to the script.

Verify

Verify checks the syntax of the script. Although this is done automatically when you select OK, this option allows you to verify the script without leaving the Scripting Editor. If an error is detected, a requester will indicate the error and your cursor will be move to the beginning of the line containing the error.

Clear

Clear erases all lines in the Script Editor. Should you inadvertently clear a script, select Cancel and re-invoke the Script Editor.

Print

Print sends the entire contents of the Script Editor to the Printer.

Text Menu

The Text Menu allows you to Load, Save, and Insert text.

Load

Load replaces the contents of the Script Editor with the contents of a File. CanDo's File Requester allows you to locate the file.

Save

Save writes the contents of the Script Editor to a file specified with CanDo's File Requester. If the File currently exists, it will be replaced.

Insert

Insert takes the contents of a file, specified with CanDo's File Requester, and inserts it into the Script at the current cursor position. Unlike Load, it does not first clear the existing script.

Edit Menu

The Edit Menu allows you to Search For and Replace text in the Script.

Search

Search... searches for the next occurrence of a text string. A requester allows you to specify the string to search for.

Search Next

Search Next searches for the next occurrence of the last text string specified with Search...

Replace

Replace... searches for the next occurrence of a text string and replaces it with another. A requester allows you to specify the Search string and the Replace String.

Replace Next

Replace Next repeats the last Replace... operation without bringing up the Search/Replace Requester.

Replace All

Replace All replaces all occurrences of a text string with another.

Misc. Menu

The Misc. Menu contains: Help, Delete Line, Delete to EOL, and Undelete Line.

Help

Help brings up the Scripting Help Requester. This Requester is described later in this Chapter.

Delete Line

Delete Line erases the current line. It can be restored using Undelete Line.

Delete to EOL

Delete to EOL erases all characters from the current cursor position to the end of the line. The characters can be restored using Undelete Line.

Undelete Line

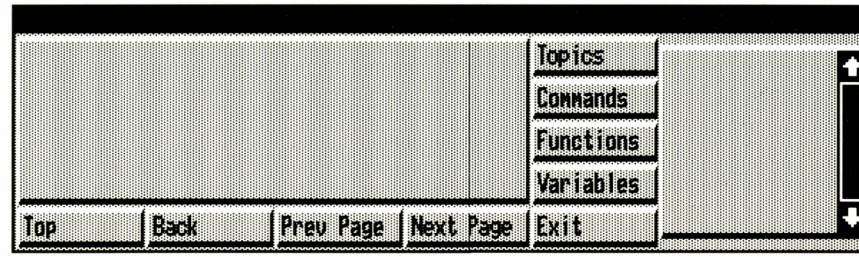
Undelete Line restores the characters erased by the last Delete Line and Delete to EOL.

Shortcut Buttons

Six of the Menu Items have a shortcut button located at the top of the Scripting Editor: Ok, Cancel, Search..., ...Next, Replace..., and Help. These buttons work the same way as their menu counterparts.

Scripting Help

You can get help on CanDo's Scripting Commands, Functions and System Variables through the Scripting Help Requester. It is invoked by selecting the Help Menu Item under Misc., by clicking the **Help** button, or double clicking a word in your script. CanDo will try to give you help for the word under the cursor or for the first word on the line.



Scripting Help Requester

The main area of the requester displays the help message. When there is more information than can be displayed in the message area, you can use the **Next Page** button to move forward. The **Prev Page** Button can then allow you to move backwards.

Some of the words in the message area can be clicked on for additional help. You can identify these words because they will be in Red. By clicking on a **Red Word**, a related help topic will be shown in place of the one which you are currently reviewing. When you want to return to the previous help topic, press the **Back** button. If you have moved several levels from your original Help Message, you can return to the first one by pressing the **Top** button.

The Topics, Commands, Functions, and Variables Buttons allow you to look up other Help Messages. By pressing one of these Buttons, the list on the right will contain a selection to choose from.

Pressing the **Topics** button displays a list of Topics such as Graphics and Card Movement. When you select one of the listed Topics, the lister will then display all of the Commands for the Topic and the **Commands** Button will be highlighted. Clicking on one of the Entries in the list, displays its Help Message.

By clicking the **Functions** Button, the list will display the Functions for the Topic. Clicking the **Variables** Button shows the System Variables.

When you are ready to return to the Script Editor, press the Exit Button.

Editor Tools

Along the right side of the Script Editor is a selection of Editor Tools. These Tools allow you to interactively create scripts.

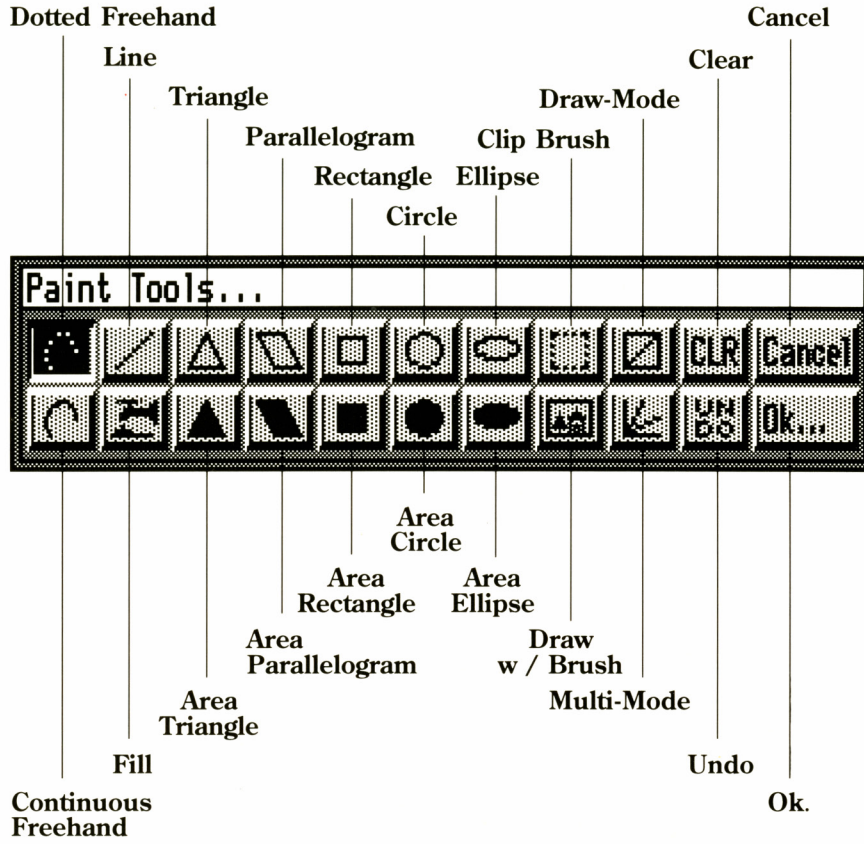
For beginners, this is an easy way to get started without spending too much time learning Scripting Commands. In fact, using the Editor Tools can be the easiest way to learn. By using them to create your own "examples", you can learn both specific Commands and the way CanDo's Scripting Commands work in general.

Advanced Users can create a "rough" script performing some of the tasks they want to accomplish. By changing a few constants to variables, and adding a loop, or other such modifications, it can be very easy to create sophisticated scripts.

Some Tools make it easy to access things that CanDo "knows" about. Card, Routine, and File Names, to name a few, are more easily and accurately identified using Editor Tools. Other Tools simply let you see the results before you try it out.

When you click on one of the Icons, a specialized requester will help you in creating your script. Feel free to just play around with the Tools. However, you should be familiar with CanDo's ARexx capabilities before exploring the ARexx Editor Tool.

The Paint Editor Tool works like a small paint program. By using its Tools, you can draw in the window and it will create the necessary Scripting Commands. By selecting the **Paint Editor Tool Icon**, CanDo will display the *Editor Tool Panel.* At the very bottom a color bar will display the current color palette. The Paint Editor Tool is designed to look and feel like a simple painting package. You simply select a drawing Tool, a color to use, and draw in your window.



While it may seem like a paint program, you need to be aware that you are creating a Script. Each action you take generates Scripting Commands. It is very easy to create large Scripts using this Tool.

Dotted and Continuous Freehand Drawing

These Tools allow you to draw in a series of dots or small connected lines. Simply select a color and draw using the **Left Mouse** Button. You should keep in mind that these Tools can easily create large Scripts.

Line

The Line Tool allows you to draw a line in the window. Simply position the mouse pointer, press and hold the **Left Mouse** Button, move the mouse pointer and release the Mouse Button. A line will be draw between the two points.

Flood Fill

The Flood Fill Tool allows you to fill an enclosed area with the selected color. Position the mouse pointer on your window and click the **Left Mouse** Button.

Triangle and Area Triangle

These Tools allow you to draw triangles. The Triangle Tool draws with lines, and the Area Triangle Tool draws filled Triangles. Simply define one side of the triangle in the same manner as drawing a line. When you release the mouse Button, the pointer will control the position of the third vertex. Position the pointer and click the **Left Mouse** Button.

Parallelogram and Area Parallelogram

These Tools allow you to draw parallelograms, a four sided polygon with parallel lines. The Parallelogram Tool draws with lines, and the Area Parallelogram Tool draws with a solid color. Draw a line by positioning the mouse pointer at the first vertex, press and hold the **Left Mouse** Button, and drag the mouse pointer to the second vertex and release the Mouse Button. The Mouse pointer will now control the three remaining sides. As you move the pointer, a parallelogram will be formed. When you press the mouse Button again, the final image will be displayed.

Rectangle and Area Rectangle

These Tools allow you to draw a rectangle with lines or a filled block. Position the Mouse pointer, press and hold the **Left Mouse** Button to define one of the corners. While holding down the Mouse Button, drag the pointer to the opposite corner and release the Mouse Button.

Circle and Area Circle

Using these Tools, you can draw filled or unfilled circles. These Tools do not automatically adjust the radius for all screen dimensions. However, you can use the ellipse Tool for these situations.

Position the mouse pointer then press and hold the **Left Mouse** Button. This defines the center of the circle. While holding down the Mouse Button, drag the pointer to define the size of the circle and release the Mouse Button.

Ellipse and Area Ellipse

These Tools allow you to draw filled or unfilled ellipses using the currently selected color. Position the Mouse pointer and press the **Left Mouse** Button. This defines the center of the ellipse. While holding the mouse Button down, drag the pointer to define the size and shape of the ellipse and release the Mouse Button.

Brush Clip

This Tool copies a rectangular area of the current window so it can be used by the Draw with Brush Tool. Simply position the mouse pointer to one of the corners and press the **Left Mouse** Button. While holding down the left mouse Button, drag the pointer to the opposite corner and release the mouse Button. The Draw with Brush Tool will automatically be selected.

Draw with Brush

This Tool draws with a clipped brush created using the Brush Clip Tool. CanDo will not allow you to select this Tool if you have not previously clipped a brush. When this Tool is selected, you will see the clipped image as you move the mouse pointer. Simply position the image and click the **Left Mouse** Button.

Draw-Mode

The Draw-Mode Button toggles between Normal and Complement.

Normal

Complement

While in Normal Mode, the drawing Tools will draw using the currently selected color. While in Complement Mode, the drawing Tools will complement the colors over which they are drawing. The advantage of Complement Mode is that when the Script is repeated, the drawing will Complement back to what it was.

Multi-Mode

The Multi-Mode Button toggles on and off. When it is off, drawing Tools will draw a single image. When it is on, the Line, Triangle, Parallelogram, Rectangle, Circle, and Ellipse Tools will draw multiple images.

Clear

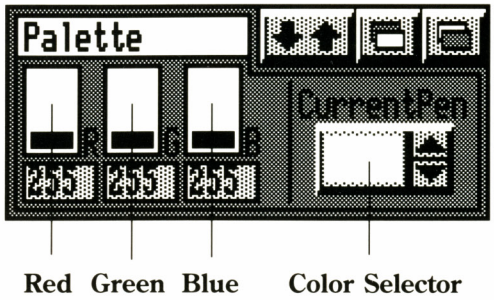
The Clear Button clears the window. Remember, this places a ClearWindow Command in the Script, and is not the same thing as starting over.

Undo

The Undo Button will undo previous operations. Each time you press the Undo Button, the last operation will be forgotten, and all remaining operations will be re-displayed. This can be repeated until all operations have been forgotten. This provides the unique feature of unlimited undo's. However, you need to keep in mind that you are removing Scripting Commands.

Selecting a Color

The *Color Selector* displays the current drawing color. You can use the up and down arrows to step through the available colors, or simply click on the color in the **Color Bar**. The **Palette Sliders** allow you to change a color.



Remember, changing a color generates a Scripting Command. It does not change the initial Palette. Rather, it changes the color when this Script is performed. NOTE: You should avoid changing the colors in a Workbench window.

The Text Editor Tool helps you print text in your window. When you select the **Text Editor Tool Icon**, the *Text Editor Tool Requester* will be displayed.

Selecting the **Set Text and Font...** Button brings up the *Text and Font Requester.*

Text and Font Requester.

This requester works in an identical fashion to the *Text Button Definition Requester* described on Page 4 - 10.

After setting the Text and Font, click the **Set Position...** Button. This will allow you to position a representation of the text using the mouse. Click the **Left Mouse** Button when it is positioned where you want it to be.

The Sound Editor Tool helps you play sounds. CanDo supports the playing of 8SVX digitized sound Files, the standard supported by the Amiga. This Tool helps you select a sound, volume and audio channel.

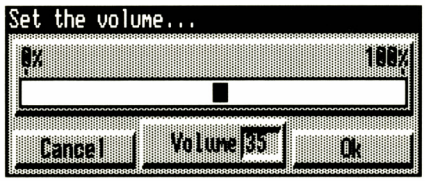
Selecting the **Sound Editor Tool Icon** brings up the *Sound Editor Tool Requester.*

Sound Editor Tool Requester.

This Requester allows you to choose from three options: selecting a sound File, changing the volume, and selecting an audio channel. You can choose one or more of these options. CanDo will produce the Scripting Commands for the options you choose. For example, you can select a sound File and a channel and not choose a volume.

Clicking the **Set the filename...** Button brings up CanDo's *File Requester.* You can use it to locate the sound you want to play. When you select **Ok**, CanDo verifies the File is an 8SVX digitized sound, and returns you to the *Sound Editor Tool Requester.*

Selecting the **Set the volume...** Button brings up the *Set Volume Requester.*

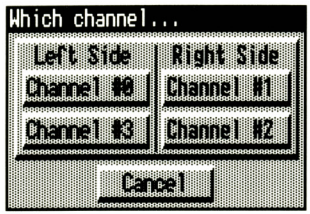


Set Volume Requester.

This Requester has a slider and a field you can use for selecting the volume level. The slider is scaled from 0% to 100%. As you move the slider, the value in the Volume Field will show the equivalent volume setting. This value ranges from 0 to 64. You can also set this value using the keyboard. When you select **Ok**, the *Sound Editor Tool Requester* is re-displayed.

NOTE: When you change the volume setting, all subsequent sounds will be played at the selected volume.

Selecting the **Set the channel...** Button from the *Sound Editor Tool Requester* brings up the *Set Channel Requester.*



Set Channel Requester.

This Requester allows you to specify a specific audio channel on which to play a sound. If you do not select a channel, the sound will play on the next available sound channel. By choosing a specific audio channel, you can control which speaker the sound is played on. Audio channels 0 and 3 are played on the left side, and channels 1 and 2 are played on the right side. However, if the specified audio channel is being used when the Script is performed, the sound will not be played.

When you select one of the four Buttons, it will become highlighted. When you select **Ok** from the *Set Channel Requester*, you will return to the *Sound Editor Tool Requester.*

Remember, whenever you are using the *Sound Editor Tool Requester*, you can select **Ok** and CanDo will return you to the *Script Editor.* CanDo will produce the Scripting Commands for the options you have selected up to that point.

The Picture Editor Tool helps you locate a Picture and creates the Command to show it. When you select the **Picture Editor Tool Icon**, CanDo's *File Requester* will allow you to find the picture file. When you select **Ok**, CanDo will create the Command to show the picture.

The DOS Editor Tool helps you Run another program. When you select the DOS Editor Tool Icon, CanDo's File Requester will appear. Locate the program you want to run, and press Ok. CanDo will verify it is an executable program.

This Tool creates a Dos Command. It simply tells the Amiga Operating System to execute the program within the quotes. The Editor Tool puts the word "c:Run" in front of the program you selected.

For example:

Dos "c:Run c:dir"

This allows your CanDo application to continue running after it starts the selected program. If you want your CanDo application to wait until the selected program is done, you can remove the word "c:Run" from the command.

Dos "c:dir"

Some programs allow parameters to be passed on the command line. You can do this by adding them after the selected program.

Dos "c:dir >ram:WorkFile"

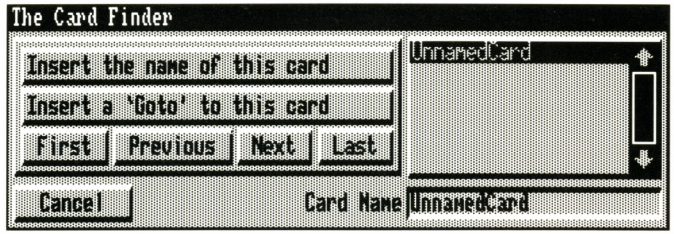
This final example would execute the Dir command in the c: directory. It does not have a "Run". Therefore, your CanDo application will wait until the command has completed before continuing. Finally, it tells the Dir command to save its output in a file called "ram:Workfile".

The File Editor Tool helps you to locate a file using CanDo's File Requester. It does not create a complete Scripting Command. It simply returns the file specification enclosed in double quotes. Many of CanDo's Commands use a file specification in this form. This way, you can use the File Editor Tool to locate the file for one of these Commands.

The Coordinates Editor Tool allows you to find the x,y coordinates of a location on your Window. When you select the Coordinates Editor Tool Icon, you should move the cross-hairs to the location you want and press the Left Mouse Button.

This Editor Tool does not create a complete Scripting Command. It simply returns the horizontal and vertical values for a single location. Many of CanDo's Scripting Commands use these x,y values. You can use this Editor Tool for finding the x,y coordinates for one of these Commands.

The Card **Finder** Editor **Tool** helps you with Card Movement Commands. Selecting its Icon displays the *Card Finder Requester.*



Card Finder Requester.

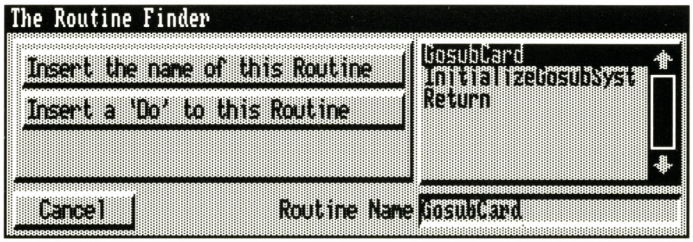
This Requester has Buttons on the left side for each of the options, and a list of all Card Names on the right side. This list works in conjunction with the first two options, inserting the Card Name and inserting a "Goto." By clicking on one of the entries, the Name is placed in the Card Name Field.

After selecting a Card Name, pressing the **Insert the Name of this Card** Button returns you to the Editor. The selected Name will be typed into your Script.

Selecting the **Insert a "Goto" to this Card** Button, will insert the necessary GotoCard instruction and return you to the Editor.

Selecting one of the four Buttons at the bottom, **First, Previous, Next, and Last**, inserts a single Scripting Command. These Commands do not use the selected Card Name.

The Routine Editor Tool displays a list of all the Routine names and allows you to insert the Name or a "Do" Command for the Routine. When you select the **Routine Editor Tool Icon**, CanDo will display the *Routine Finder Requester.*



Routine Finder Requester

The *Routine Finder Requester* displays a list of all currently defined Routines in a list on the right side of the Requester. By clicking on one of the entries, its Name will be placed in the Routine Name Field.

When you select the **Insert the Name of this Routine** Button, CanDo will put the Routine Name, enclosed in quotes, into your Script.

When you select the **Insert a "Do" to this Routine** Button, CanDo will put a Do Command into your Script.

The Field Editor Tool helps you with the Field Object. The Field Object allows the user of your application to enter or otherwise edit a single line of text or an integer number. You can set or retrieve the contents of a Field using Scripting Commands. This Editor Tool assists you in doing this.

Selecting the Field Editor Tool Icon displays the *Field Editor Tool Requester.*

Field Editor Tool Requester.

This Requester displays the Object Name for all Fields on the current card and gives you three Scripting options. You can simply insert the Object's Name, set the contents of the Field, or retrieve the contents of the Field.

Selecting a Name

Each Field is given a Name on the *Field Editor Requester.* The Names for all Field Objects are displayed in the list on the right side of the *Field Editor Tool Requester.* By clicking an entry, the Name will be put into the **Field Name** Field. You should select an entry by clicking one of the three Scripting options.

Insert Field's Name

Clicking the **Insert the Name of this Field** Button automatically types the selected Object Name with double quotes on each side. This is a simple way of finding a Field Name and using it with a Scripting Command. This option does not produce a complete Scripting Command.

Set Field's Contents

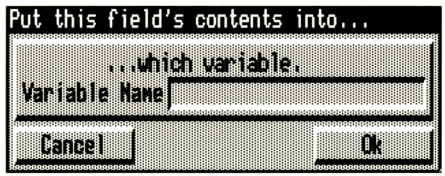
Clicking the **Set this field's contents to...** Button displays the *Set Field Requester.*

Set Field Requester.

The New Value Field allows you to type in a value to be put in the selected Field. If the selected Field is an Integer Field, you will be limited to entering a valid integer value. Otherwise, you can enter any characters you wish. Remember, if you want to include any double quotes, you should put two in a row (""). Selecting **Ok** creates the necessary Scripting Command and returns you to the Editor.

Get a Fields Contents

Selecting the P**ut this field's contents into...** Button displays the *Get Field Requester.*



Get Field Requester.

This Requester aids you in retrieving the contents of a Field and putting the value into a variable. You need to provide the Variable Name in the provided Field. Clicking **Ok** creates the necessary Scripting Command and returns you to the Field Editor Tool.

The ARexx Editor Tool helps you send and receive ARexx messages. If you are interested in doing this, it is recommended that you read both the ARexx Object and ARexx Commands sections before doing so. While this Editor Tool makes it easier to produce the ARexx Commands, you should first familiarize yourself with the ARexx concepts.

Before sending ARexx messages to an application, acquaint yourself with its ARexx capabilities. It should have documentation for its ARexx Port Name and the Commands it can receive.

After selecting the A**Rexx Editor Tool Icon,** CanDo will display the *ARexx Editor Tool Requester.*

ARexx Editor Tool Requester.

From this Requester you can select a Message Port to listen to or speak to, or send an ARexx Message to the current SpeakTo Port.

ListenTo

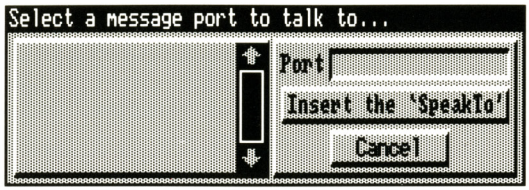
Selecting the **Select a Message Port to listen to...** Button, displays the *ListenTo Requester.*

ListenTo Requester.

This Requester allows you to specify the ARexx Port Name that your application uses to receive ARexx messages. It is recommended that you put this in the StartUp Script of your first Card. Selecting the **Insert the "ListenTo"** Button inserts the Command and returns you to the Editor.

SpeakTo

Selecting the **Select a message Port to speak to...** Button from the *ARexx Editor Tool Requester* displays the *SpeakTo Selector.*



SpeakTo Selector.

The *SpeakTo Selector* displays all of the Public Message Ports. While an ARexx Port is Public, not all Public Ports can receive ARexx messages. In fact, sending a message to some ports will cause the Amiga to crash. You should know the Name of the Port to which you wish to send a message. This Selector simply makes it easier for you to find the Name and insure that it is currently available.

Clicking an entry selects the Name and puts it into the Port Field. Selecting the **Insert the "SpeakTo"** Button or double clicking an entry inserts the Command and returns to the Editor.

Send Message

Selecting the **Send an ARexx message** Button from the *ARexx Editor Tool Requester* displays the *Send ARexx Message Requester.*

Send ARexx Message Requester.

This requester allows you to send a message to the current SpeakTo Port. Simply enter the message and click the **Insert the "Message"** Button. This will insert the SendMessage Command and return you to the Editor.

6 Commands Index

6 Commands

CanDo has over 200 scripting Commands and Functions. However, you shouldn't feel as though you need to know how to use all of them in order to make your application. In fact, you really only need to know a few to get started.

You should use the Editor Tools to get your feet wet. By looking at the Commands they produce, you can learn a great deal about the way the Commands work.

At first you will want to use this Chapter as a reference for the Commands you encounter. As your needs grow, you will want to read about a certain classes of Commands. This Chapter is organized for this purpose. Each section describes certain aspects or classes of Commands.

While you do not initially have to understand Expressions and Functions, you should eventually read the first two sections of this Chapter. By making use of Expressions and Functions, you can easily add a lot of sophistication to a simple application.

The FlowControl and CardMovement Commands are very powerful. They allow you change what is happening in your application. The CardMovement Commands, in particular, are very simple and you should learn to use them early on.

The Graphic, Screen and Window, Animation, and Audio Commands are the spice you use in your CanDo applications. However, these comprise the majority of the Commands and they may take little while to master. Keep in mind that you don't have to learn to use everything. If something seems complicated, come back to it later.

The Document Commands are unique. You can use them with the Document Objects or for internal manipulation of text. These Commands bring an additional level of power to the manipulation text. Depending on your needs, you may not want to work with these initially.

File I/O, Icons, ARexx, Object and Buffer Commands should be considered advanced. Power users can use these Commands to make applications that rival professional software.

Finally, Misc. Commands rounds off a few Commands that you may find useful in your application.

Command Symbols

There are several symbols used in describing the various Commands and Functions.

< > are used to indicate Integers. *

“ ” are used for Strings *

« » are used for Logicals. *

{ } are used to show optional parameters are Optional

, a Comma separates parameters

; a Semi-Colon can be used to add Comments to the end of a Command line.

KEYWORDS are always shown in all capital letters.

* You should read the Expression System section for an explanation of Integers, Strings and Logicals.

Expressions

An expression is a group of constants, variables, and functions combined with operators. You use expressions to tell CanDo's scripting commands what to do.

Most of the time, you will use constants such as 5, "Hello", and ON. However, sometimes you will want to do more sophisticated operations. Expressions allow you to describe a value that is determined at the time the command is performed.

CanDo's expression system has been designed to be as intuitive as possible. Numbers are represented as integer constants. Strings are contained in double "quotes". And logical values use names such as ON, OFF, YES, NO, TRUE, and FALSE. In addition, you can mix integer, string, and logical values within expressions, and CanDo will automatically convert the values as needed.

This section describes the details of CanDo's expression system. While it is not necessary for you to understand all aspects of this system, CanDo provides many sophisticated operations and functions. Depending upon your experience level, you may want to use this as a reference section or you may want to read it for a complete understanding of CanDo's expression system. Either way, you should at least scan it to get a grasp of its principles and abilities.

Most of CanDo's scripting commands use expressions to provide the information for the operation. For example, the LET command saves a value in a variable. The format for the LET command is:

LET VariableName = expression.

The results from evaluating the expression is saved in the indicated variable.

Examples:

Let X = 100

In this example, 100 is a simple expression containing a single integer constant. It is saved in the variable 'X'.

Let Y = X * -2

The expression in this example is 'X * -2'. The result, -200, is assigned to the variable 'Y'.

Let Z = 5 + Absolute (Y + 10)

This expression uses the ABSOLUTE function. A function takes one or more parameters within parenthesis, performs an operation, and returns a single value. Each parameter within a function can be an expression. The parameter for the ABSOLUTE function, in this example, is the expression 'Y + 10'. As you might expect, the ABSOLUTE function returns the absolute value of an expression. The absolute value of the expression '-200 + 10' is 190. 'Z' is assigned the value of '5 + 190', or 195.

An Expression is a combination of constants, variables, and functions with operators for computation. The evaluation of an expression results in a single value.

Within CanDo, expressions use three types of values: Integers, Strings and Logicals.

Integers

Integers are the basic numeric system for dealing with graphics and computer control. Integers are whole numbers that range from -2147483648 to 2147483647.

An integer constant is a series of digits. A minus sign preceding the digits indicates a negative integer. A plus sign is not necessary to indicate a positive integer. However, it can be used to provide clarity.

Examples:

105 -5000 +6

Strings

Strings are groups of characters. String constants are characters contained within double quotes. For example: "Fred is Here". However, the double quotes are not part of the string. They simply show where the string begins and ends.

Strings can be as large as available memory will allow. Or a string can be empty. An empty string is often referred to as a null string. CanDo recognizes a null string as two double quote marks "".

Strings can contain any character. These include many that are not visible or are considered part of the International character set.

For a string to contain a double quote, it must be typed twice within the string. CanDo will treat it as though only one is contained in the string. For example "Fred says ""Hi"" to everyone." contains the word "Hi" within the string.

Logicals

Logicals (known as Booleans) simply mean true or false. The words TRUE and FALSE are logical constants. Within CanDo, YES and ON also mean TRUE. Similarly, NO and OFF mean FALSE.

Variables

A variable is a name to be used in the place of an integer, string, or logical constant value. Variables can be thought of as storage locations for values. CanDo does not require you to 'declare' a variable.

Variable names may contain letters, digits, and underscore characters (_). The name must begin with a letter. The names can be any length. However, longer names use more memory and take longer to identify.

Any variable name can be used for integers, strings, or logicals.

LET

LET VariableName = expression

The Let command allows you to save a value in a variable. CanDo evaluates the expression and then saves the value in the variable. This value will be used in the evaluation of expressions containing the variable.

CanDo also provides system variables and functions such as 'MouseX' and 'Sign (expr)'. With the exception of these variables and functions, any variable name can be used.

Expressions

An expression contains constants, variables, and functions that can be combined with operators to form a new value.

An example of a simple expression is 'Count + 5'. The first value, 'Count', is a variable. The second value, '5' is an integer constant. The '+' represents the addition operator. This expression adds the contents of the variable 'Count' with 5.

Constants, variables, and functions can be used interchangeably as values within an expression.

An operator is a symbol that represents a process to be performed on one or two values.

CanDo allows any type of value to be used with any operator. Strings, integers, and logicals can be used interchangeably with any operation. Each is automatically converted to the type of data used by the operator.

Examples:

"-1000" + 5

The addition operator (+) adds two integer values. The first value, "-1000" is a string. It is automatically converted to an integer value of -1000 and is added to 5. The result of this expression is -995.

"-1000" || 5

The string concatenation operator (||) appends two strings together. The first value in the expression is a string "-1000". How ever, the second value is an integer constant 5. It is converted to a string constant of "5" and appended to "-1000" the result of this expression is "- 10005".

The default value for an unassigned variable is either an integer ZERO (0), a string NULL ("") or a logical FALSE. The default value used is determined by the type of operation being performed. For example, Repeat-Count will have default value of ZERO for the expression (5 + Repeat-Count).

The automatic conversion makes it easier to work with different kinds of data. It isn't necessary to keep track of different variable types or to use conversion functions. Most of the time, different data types can be used interchangeably without special consideration. The automatic conversion is described in the following sections so that predictable results can easily be achieved.

Order of Evaluation

Each operator has a "priority" that determines the order of computation. For example, multiplication is performed before addition. The priority scale ranges from 1 to 8, where operations with a priority of 8 are performed first and operations with a priority of 1 are last.

The priorities cause expressions to be evaluated in the standard order of algebraic rules. Operators with the highest priorities are evaluated first, followed by the next lower priority. Operators with the same priority are evaluated from left to right. Operations within parenthesis are evaluated before ones outside. All expressions allow the usage of parenthesis to change the order of evaluation or to clarify the intended usage.

Numeric Operations

These operators provide the basic arithmetic functions. They operate on integer values. If a value is a string or logical, it is converted into an integer.

Operation	Number of Values	Priority	Symbol
Unary Minus	1	8	-
Unary Plus	1	8	+
Multiply	2	6	*
Divide	2	6	/ or %
Modulo	2	6	MOD or //
Addition	2	5	+
Subtract	2	5	-

Unary Operators

As in algebra, + and - can be used to indicate positive or negative numbers. A unary operator can precede a single value. The value can be a constant, a variable or an expression within parentheses. Unary operators have the highest priority.

Examples:

- 8 + 56 - Count - (x + y)

Multiply, Divide, Modulo, Add, and Subtract Operators

These operations perform the basic algebraic operations.

The modulo operation returns the remainder of an integer divide. Both the symbol // or the word MOD can be used.

Expression	Result
4 + 7 - 2	9
5 +3 * 2	11
- 8 - 6 / 2	- 11
+ 6 - 11 mod 3	4
+ 6 - 11 // 3	4

Note: Currently you can use either / or % for Division. Future releases will use / for Floating Point Division amd % for Integer Division. Your current scripts will be fully compatable with future versions if you use the % operator.

String to Integer Conversion

If the first characters within the string represents an integer, the automatic conversion will recognize the value. Otherwise, an integer value of ZERO will be used.

String	Result
"104"	104
"14 Times"	14
"Five"	0
"- 86"	- 86
" -5 "	0
"Jacks"	0

Logical to Integer Conversion

Logicals are either TRUE or FALSE. TRUE converts to 1. FALSE converts to 0.

Logical	Result
TRUE	1
FALSE	0
ON	1
OFF	0
YES	1
NO	0
(5 = 6)	0
(5 < = 6)	1

String Concatenation Operations

The string concatenation operators append two strings together. The concatenate including space operator (III) appends the two strings with a space between them.

Operation	Number of Values	Priority	Symbol
Concatenate	2	4	II
Concatenate including Space	2	4	III

Expression	Result
"1234" II "5678"	"12345678"
"1234" III "5678"	"1234 5678"

Integer to String Conversion

If the value for a string operation is an integer, it will be converted to a string. The string equivalent will not contain leading spaces or zeros. If the integer is negative, it will contain a leading minus sign ("-").

Expression	Result
"Score: " || 250	"Score: 250"
"Ending Value: " ||| (15 - 25)	"Ending Value: - 10"

Logical to String

If a logical value is used in a string operation, it will be converted to either "TRUE" or "FALSE".

Expression	Result
"Value > 100 : " ||| (Value > 100)	"Value > 100 : TRUE"
TRUE ||| FALSE ||| YES ||| NO	"TRUE FALSE TRUE FALSE"

Relational

Relational operators compare two values to each other. These operations work with both integer and string values. The result of a relational comparison is a logical value.

Operation	Number of Values	Priority	Symbol
LessThan	2	3	<
GreaterThan	2	3	>
LessThan or Equal	2	3	<= =<
GreaterThan or Equal	2	3	>= =>
NotEqual	2	3	<> >< ~=
Equal	2	3	= == :=

If both values are integers, they are compared in the usual way. However, if either of the values are not integers, then both values are converted to strings.

The results of string comparisons are similar to the way a dictionary is ordered. However, upper and lower case letters are not the same. The order of the letters are based on the ASCII character set. (see ASCII appendix)

Expression	Result
7 + 2 < 6	FALSE
5 - 7 == - (12 / 6)	TRUE
"Five" <> "FIVE"	TRUE
"- 15" = 5 * - 3	TRUE (string comparison)

Boolean Operations

Boolean operators work with logical values. Often they are used with the results from relational comparisons.

Operation	Number of Values	Priority	Symbol
NOT	1	8	NOT ~
And	2	2	AND &
Or	2	1	OR |
Xor	2	1	XOR &&

The OR operation is used to determine if either of two conditions is TRUE. The AND operation determines if BOTH conditions are TRUE. The XOR is used to ascertain when one of the values is TRUE but not BOTH.

The NOT operation is similar to the unary minus. The unary minus changes the sign of a value. The NOT operation changes a TRUE value to FALSE, and a FALSE value to TRUE.

Expression	IntermediateResult	Result
3 < > 4 or 5 = 6	TRUE or FALSE	TRUE
5 = 3 + 2 and 7 < 2	TRUE and FALSE	FALSE
3 < = 7 and - 5 > = - 8	TRUE and TRUE	TRUE
NOT (5 = 5 and 6 < > 7)	NOT (TRUE)	FALSE

Integer to Logical Conversion

When the value for a logical operation is an integer, it will be converted to FALSE if the value is ZERO. Otherwise, it is converted to TRUE.

Expression	Result
0 or 1	TRUE
1 and 5	TRUE
NOT 5	FALSE

String to Logical Conversion

If the value for a logical operation is a string, it is converted to a TRUE when the STRING is "TRUE", "ON", or "YES". The identification is only on the leading characters of the string. If the leading characters do not match, it is converted to FALSE.

Expression	Result
"TRUE" or "FALSE"	TRUE
"TRUE" and "ON"	TRUE
"TRUE" and " ON"	FALSE
"TRUE" and "ONCE"	TRUE ("ON" is identified in "ONCE")

Functions

A function is an operation, which returns a single value, that can be used within an expression.

Some functions do not require parameters. These functions are called System Variables. This is because their usage resembles that of variables. MouseX is an example of a System Variable. It can be used in any expression just as though it was a variable. However, it is a read-only variable. This means you can not use the Let Command to change its value. It is always equal to the "current" value of the horizontal position of the mouse pointer.

Other functions require information, in the form of parameters, in order to perform its operation. A function can have one or more parameters, contained within parenthesis, and separated by commas.

Let X = Max { OldX , Y / 2 , 100 }

This assignment demonstrates the Max Function. The parameter list for the function is contained within parenthesis. Each parameter is an expression. This means it can contain constants, variables, operators, and even other functions. Multiple parameters are separated by commas.

As with operators, the parameters for functions have expected data types. The parameter is automatically converted to the required type.

The function returns a single result. This way, the function can be placed in an expression just as a variable or constant.

Conversion Functions

Within expressions, values are automatically converted to the type of data needed for an operation. However, the data type of the result is determined by last operation performed.

< integer > = Integer { expression }
"string" = String { expression }
« logical » = Logical { expression }

Example:

LET Count = "123" || "456"

The variable Count is assigned the string value of "123456". Count could be successfully used in arithmetic operations. However, each time it is used it is converted into an integer. IfCount is primar ily used as an integer, it would be more efficient to convert the value when it is assigned.

LET Count = Integer { "123" || "456" }

The Integer Function converts the value within the parenthesis into an integer. Likewise, the String and Logical Functions convert values into strings and logicals.

Integer Functions

The following functions allow you to work with in integer values.

Absolute

< integer > = Absolute (< value >)

The Absolute Function returns the absolute value of an integer. If the value is positive or ZERO, Absolute returns the same value. If the value is negative, it returns the positive value.

Expression	Result
Absolute (156)	156
Absolute (- 66)	66
Absolute (0)	0

Limit

< integer > = Limit (< limit1 > , < limit2 > , < test value >)

The Limit Function returns a value within a specified range. The first two values, < limit1 > and < limit2 > indicate the minimum and maximum values in the range. The third parameter is the value to test. If this value is between the two limits, the Limit Function returns the value unchanged. If it is less than the low limit, it returns the low limit value. If it is greater than the high limit, it returns the high limit value.

Expression	Result
Limit (0, 100, 89)	89
Limit (- 300, - 100, 45)	- 100
Limit (900, 400, 0)	400

Max

< integer > = Max (< val > { , < val > , ... })

The Max Function returns the value of the highest parameter. The Max Function can have one or more parameters.

Expression	Result
Max (5, - 1000, 66, - 5)	66
Max (- 10, - 5)	- 5
Max (6 - 5, 7 * 3 , - 1)	21

Min

< integer > = Min (< val > { , < val > , ... })

The Min Function returns the value of the lowest parameter. The Min Function can have one or more parameters.

Expression	Result
Min (100, 5000, 200)	100
Min (- 5 * 6, - 1)	- 30
Min (5, 7, - 3, 10, - 5, 88, 8)	- 5

Random

< integer > = Random (< Minimum >, < Maximum >)

The Random Function returns a random integer between and including the < Minimum > and < Maximum > values.

Expression	Result
Random (5, 20)	any value between 5 and 20.

Sign

< integer > = Sign (< value>)

The Sign Function returns the sign of the value. If the value is positive, it returns 1. If the value is ZERO (0), it returns 0. Otherwise, it returns - 1

Expression	Result
Sign (0)	0
Sign (- 56)	- 1
Sign (4 * 56)	1

String Functions

The string functions allow you to manipulate and work with strings.

ASCII

< integer > = ASCII ("String")

The ASCII Function returns the ASCII value of the first character in the supplied string. The ASCII value is a positive integer between 0 and 255.

Expression	Result
ASCII ("A")	65
ASCII ("m")	109
ASCII ("more")	109
ASCII ("<")	60

Char

"string" = Char (< integer >)

The Char Function returns a single character string corresponding to an ASCII integer. The ASCII values range from 0 to 255.

Expression	Result
Char (65)	" A "
Char (109)	" m "
Char (60)	" < "

NumberOfChars

< integer > = NumberOfChars ("string")

The NumberOfChars takes a string parameter, and returns the number of characters.

Expression	Result
NumberOfChars ("Hello!")	6
NumberOfChars ("Spanish Inquisition")	20
NumberOfChars (- 67)	3

LowerCase

"string" = LowerCase ("string")

The LowerCase Function converts uppercase characters within a string to lowercase. All other characters remain unchanged.

Expression	Result
LowerCase ("ABCdef123#$%")	"abcdef123#$%"

UpperCase

"string" = UpperCase ("string")

The UpperCase Function converts lowercase characters within a string to uppercase. All other characters remain unchanged.

Expression	Result
UpperCase ("The Input String")	"THE INPUT STRING"

DupeString

"string" = DupeString ("string" , < count >)

The DupeString duplicates a string a specified number of times. It returns a single string.

Expression	Result
DupeString (" * ", 10)	" ********** "
DupeString ("- ", 6)	" - - - - - - "
DupeString ("Hello! ", 3)	" Hello! Hello! Hello! "

TrimString

"string" = TrimString ("string")

The TrimString Function removes leading and trailing spaces and TAB characters from the source string. In addition, multiple spaces and TAB characters within the remaining string are replaced with a single space.

Expression	Result
TrimString (" This is a Test ")	"This is a Test"
TrimString ("Hello Out there! ")	"Hello Out there!"

InsertChars

"string" = InsertChars ("Source" ,"destination" ,< offset >)

The InsertChars Function inserts a source string into the destination string at the specified offset. If the offset is greater than the length of the destination string, the source string is appended to the end of the destination.

Expression	Result
InsertChars ("...", "123456", 4)	"1234...56"

RemoveChars

"string" = RemoveChars ("Source" ,< starting offset > ,< length >)

The RemoveChars Function returns a string with specified characters removed. The first parameter is the source string from which to remove the characters. The second parameter specifies the starting offset of the characters to be removed. The last parameter indicates the number of characters to remove.

Expression	Result
RemoveChars ("12345", 3, 2)	"125"
RemoveChars ("Hello!", 5, 1)	"Hell!"
RemoveChars ("Hello!", 5, - 1)	"Hello!"

FindChars

< integer > = FindChars ("Source" ,"Search" ,< starting offset >)

The FindChars Function searches the contents of a source string for a matching string.
The < starting offset > indicates the offset within the "Source" string to begin the search. The "Search" string must match identically. If the string is found, FindChars returns the offset within the "Source" string of the first character of the matching string. If the string is not found, FindChars returns a ZERO (0).

Expression	Result
FindChars ("Hello Bill!", "Bill" , 1)	7
FindChars ("Hello Bill!", "Fred" , 1)	0
FindChars ("This is it!", "is" , 1)	3
FindChars ("This is it!", "is" , 4)	5

GetChars

"string" = GetChars ("Source" ,< starting offset > ,< length >)

The GetChars Function returns a portion of a string. The < starting offset > indicates the starting character of the substring. The < length > indicates the number of characters to include.

Expression	Result
GetChars ("Brainpower",2,4)	"rain"
GetChars ("Hello!, 4, 10)	"lo!"
GetChars ("Jim Finney",20,1)	""

FindWord

< integer > = FindWord ("Source" ,"Search Word" { ,< StartWordNumber > { ,"WordDelimiters" } })

The FindWord searches a "Source" string for a matching "Search Word" and returns its word number. If the word is not found, it will return a ZERO (0). The optional < StartWordNumber > allows you to specify a starting word number to begin the search. If it is not specified, it defaults to 1. The "WordDelimiters" can contain characters indicating the characters that separate word.

Expression	Result
FindWord ("This is a sample sentence.","sample")	4

GetWord

"string" = GetWord ("Source" ,< WordNumber > { ,"WordDelimiters" })

The GetWord Function returns a specified word number from the "Source" string. The < WordNumber > indicates which word to return. A < WordNumber > of 1 returns the first word. A 2 returns the second word. By default, Words are separated by spaces. You can supply a list of characters in the optional parameter "WordDelimiters".

Expression	Result
GetWord ("The dog Smiled.",3)	"Smiled"
GetWord ("DF1:Sounds/Bird.snd",2,":/")	"Sounds"

PositionOfWord

< integer > = PositionOfWord ("Source", < WordNumber > , { ,"WordDelimiters" })

The PositionOfWord Function returns the offset into the "Source" string for a specified word < WordNumber >. If the < WordNumber > is greater than the number of words in the "Source" string, PositionOfWord returns a ZERO (0). The "WordDelimiters work in the same way as in GetWord allowing you to specify the characters separating words.

Expression	Result
PositionOfWord("Word1 Word2 Word3",2)	7

BumpRevision

"string" = BumpRevision ("Name")

The BumpRevision Function changes the revision of a "Name" in the same manner the Workbench Duplicate function does with a filename.

Expression	Result
BumpRevision ("Record")	"Copy of Record"
BumpRevision ("Copy of Record")	"Copy 2 of Record"
BumpRevision ("Copy 2 of Record")	"Copy 3 of Record"

EvaluateExpression

results = EvaluateExpression ("String")

The EvaluateExpression Function evaluates the string parameter. The string must contain a valid expression. EvaluateExpression returns the results of the evaluation. If there is any uncertainty as to the validity of the expression, a run time error can be avoided by using the VerifyExpression Function.

This function allows expressions to be created at run time. This can be quite powerful. However, it is error prone and it can make the script difficult to read.

Expression	Result
EvaluateExpression("5 + 6")	11
EvaluateExpression("123" || "-23")	100

VerifyExpression

« logical » = VerifyExpression ("String")

The VerifyExpression Function evaluates the string parameter. However, it does not return the resulting value of the evaluation. It returns a TRUE if the expression is successfully evaluated. Otherwise, it returns a FALSE. This function also allows an expression to be verified before using the EvaluteExpression.

Expression	Result
VerifyExpression ("5 + 6")	TRUE
VerifyExpression ("6 + * 7")	FALSE
VerifyExpression (" (5 + 6 ")	FALSE

System Variables

System Variable can be used in expressions as variables. Some of these variables return dynamic information that is updated by CanDo. Others have static values for common uses:

Boolean Values:

« logical » = FALSE - Boolean Value. (FALSE)

« logical » = TRUE - Boolean Value. (TRUE)

« logical » = NO - Boolean Value (FALSE)

« logical » = YES - Boolean Value (TRUE)

« logical » = OFF - Boolean Value (FALSE)

« logical » = ON - Boolean Value (TRUE)

Informational:

< integer > = AvailableChipMemory - available bytes in chip memory

< integer > = AvailableFastMemory - available bytes in fast memory

< integer > = AvailableMemory - available bytes in memory

< integer > = LargestChunkOfMemory - the largest continuous chunk of available memory

"string" = DeckName - the Name of the current application

"string" = CardName - the name of the current Card

"string" = ObjectName - the Name of the current running Object

< integer > = MaxInteger- returns 2,147,483,647

< integer > = MinInteger- returns -2,147,483,648

« logical » = Supervised- TRUE when running from CanDo

"string" = TheTime- returns the Current Time - **"HH MM SS"**

"string" = TheDate- returns the Current Date - **"YY MM DD"**

CanDo has a number of System variables that are closely related with a group of Commands. These System variables are documented in the sections with these Commands.

Flow Control Commands

The Flow Control commands allow you to change the order of execution within a script. Usually, commands execute one at a time from top to bottom. These Commands effect this flow of execution.

If... EndIf

If... EndIf

The If Command allows you to execute a group of instructions when a certain condition is TRUE. (See Logical Expressions)

If { logical expression }
...Commands...
EndIf

The commands between the If command and the EndIf command are performed only when the logical expression is true.

Example:

```
If NumberOfHits > 10
        ShowBrush "Brushes:Explode.br"
        PlaySound "sounds:bang.snd"
EndIf
```

If... Else... EndIf

If... Else... EndIf

The Else Command can be used with an If to perform an alternate set of commands when the logical expression is false.

Example:

```
If NumberOfHits > 10
        ShowBrush "Brushes:Explode.br"
        PlaySound "Sounds:Bang.snd"
        Let NumberOfHits = 0
Else
        PlaySound "sounds:Doink.snd"
        Let NumberOfHits = NumberOfHits + 1
EndIf
```

The commands between the If and the Else are performed when the value of NumberOfHits is greater than 10. Otherwise, the Commands between the Else and the EndIf are performed.

CanDo supports four looping combinations. These Commands allow you to repeat a group of commands given specified conditions.

While... EndLoop

While... EndLoop

CanDo supports four looping combinations. The While Command allows you to repeat a group of Commands while a condition is TRUE. The form for a While Command is:

```
While { logical expression }
        ...Commands...
EndLoop
```

The Commands between the While and EndLoop are performed while the logical expression is TRUE. When the While instruction is encountered, the logical expression is evaluated. If the condition is TRUE, the Commands between the While and the EndLoop are performed. When the EndLoop Command is encountered, execution is looped back to the While Command. This continues until the logical expression is TRUE.

When the logical expression is FALSE, all Commands between the While and EndLoop are skipped, and execution continues on the instruction following the EndLoop.

Example:

```
Let Xoffset = 20
While Xoffset <= 300
        ShowBrush "Brushes:Arrow.br",Xoffset,40
        Let Xoffset = Xoffset + 20
EndLoop
```

Loop... Until

Loop... Until

The typical format for the Until Looping Command is shown below.

```
Loop
        ...Commands...
Until { logical expression }
```

The Until works in a similar manner as the While Loop. The commands between the Loop and Until are performed until the logical expression is TRUE. The Until Loop is used when you want the Commands in the Loop to be performed once before evaluating the logical expression.

Example:

```
Let Xoffset = 20
Loop
        ShowBrush "Brushes:Arrow.br",Xoffset,40
        Let Xoffset = Xoffset + 20
Until Xoffset > 300
```

While... Until

While... Until

This Looping combination, while unusual, is valid. The format for the While and Until Looping Commands is shown below.

While { logical expression }
...Commands...
Until { logical expression }

The Commands between the While and Until are performed as long as the While's logical expression is TRUE and the Until's logical expression is FALSE.

Loop... EndLoop

Loop... EndLoop

The form for Loop and EndLoop Commands is shown below.

Loop
...Commands...
EndLoop

This is a simple looping system. All commands within the looped area are repeated until an 'ExitLoop' command is executed. You should be very careful insuring there is a way for the ExitLoop to be performed. It is usually within an If...EndIf placed in the Loop commands.

ExitLoop

ExitLoop

Exits the Highest Loop Level. This command is usually within an If...EndIf in the Loop commands. It can be used to exit any of the Loop combinations. If you have nested Loops (Loops within Loops) it exits the highest Loop level.

Note: Your Scripts will be more readable if you use the While and Until conditional Looping instead of ExitLoop.

Do "Routine Name" { , Argument 1... , Argument 10 }

The Do Command performs the routine created using the Routine Object. The "routine name" must match the Name used in the *Routine Editor Requester*. This is very similar to the GoSub used in other languages. The Commands within the routine will be performed, and execution will continue with the Command following the Do.

You can use the *Routine Editor Tool* for finding the available routine names.

Examples:

```
If value = 5
        Do "ShowExplosions"
        Do "SoundEffects"
EndIf
```

Optionally, you can pass a routine up to 10 arguments. An argument is simply a value that the calling script gives the routine.

```
Do "Draw Box" , 2 , 10 , 20
```

This example calls the Routine "Draw Box" and passes it three arguments: 2, 10, and 20. While this example uses constants, each argument can be an expression.

The routine "Draw Box" can use the arguments to perform its task. It does this using the System Variables Arg1 through Arg10.

```
SetPen Arg1
DrawRectangle Arg2 , Arg3 , 50 , 25
```

If the routine "Draw Box" contained this script, it could use three arguments to draw a box. Arg1 is used with the SetPen Com mand to set PenA. The previous example passed a value of 2 for the first argument. This would result in setting PenA to color 2. The DrawRectangle Command draws a box with a width of 50 and a height of 25. Its origin would be determined by the second and third arguments. Using the previous example, this would draw the box at 10, 20.

ExitScript

ExitScript

A Script usually exits after performing the last Command. An ExitScript exits as though the last Command was performed. If the ExitScript is performed from a Routine, execution will continue in the calling Script.

StopScript

StopScript

The StopScript command is similar to an ExitScript. However, if it is performed from a Routine, execution will not continue in the calling Script. It does not running your application, it simply terminates the execution of the current Script, and any scripts that may have called it using a Do Command.

Quit

Quit

This Command allows the user to exit the presently running Deck. While you are developing your application, CanDo will not allow you to exit using a Quit Command. However, when you run it as a separate application, it is important to provide a way to Quit.

Card Movement Commands

These Commands change Cards in the Deck. They correspond to the Card Movement Buttons on the *Main Control Panel.*

NextCard _____

NextCard goes to the next Card in the Deck. If you are currently on the last Card in the Deck, it will go to the first Card.

PreviousCard _____

PreviousCard goes to the previous Card in the Deck. If you are on the first Card, it will go to the last Card.

FirstCard _____

FirstCard goes to the first Card in the Deck.

LastCard _____

LastCard goes to the last Card in the Deck.

GotoCard _____

GotoCard "cardname"

You can use a Card's Name to move directly to a specific Card with the GotoCard Command. The name must be spelled exactly as it was spelled in the *Card Editor Requester.* The "cardname" is a string parameter.

Example:

GotoCard "Card #2"

Graphic Commands

The Graphic Commands change the images in your Card's Window. This can be done by showing pictures and brushes, by drawing images or displaying Text. In addition, there are variety of commands that control how other drawing commands are performed.

Picture and Brush Commands

CanDo's Picture and Brush Commands allow you to show images created with any Paint Program. These commands include "Clipping" allowing you to copy images from your window. These clipping images can then be displayed or saved to a file.

LoadPicture

LoadPicture "filename" { , "Name" { , loadflags } }

Before an image can be displayed, it must be loaded into memory. This can be done automatically with the ShowPicture Command. However, there will be a delay as the image is loaded. This may be Ok most of the time. However, it can be avoided by pre-loading the image using the LoadPicture Command. The LoadFlags are described in the LoadFlags Appendix.

"Filename"

The "Filename" is the file specification for the image being loaded. It must be a valid ILBM file type created by most Amiga Paint programs.

"Name"

The optional "Name" allows you to specify the name used by a ShowPicture Command. When the "Name" is not specified, the ShowPicture Command must use the exact same string used in the "filename".

Example:

```
LoadPicture "images:background.pic","background"
ShowPicture "background"
```

LoadBrush

LoadBrush "filename" {, "Name" { , loadflags } }

The LoadBrush Command works similarly to the LoadPicture Command, except it pre-loads a brush file created by a paint program. It can then be shown using the ShowBrush Command. The parameters are identical to LoadPicture's parameters.

Example:

```
LoadBrush "brushes:theleftarrow.br","LeftArrow"
ShowBrush "LeftArrow",10,190
```

ShowPicture

ShowPicture "Picture Name"

The ShowPicture Command is used to change the picture being displayed. CanDo automatically adjusts the screen resolution and color palette. Furthermore, Visible Objects are re-displayed on the new picture.

You should be careful when using pictures of different dimensions. Objects, such as Buttons, are displayed at specified Horizontal and Vertical locations. For Example: If the original window had a width of 640 and a Button is horizontally positioned at 500, it will not be visible if you show a picture having a width of 320. The safest approach is to use different cards with pictures having different dimensions.

"Picture Name"

The "Picture Name" indicates the file to be displayed. The name can be a file specification. If so, CanDo will automatically load it if it is not already in memory. It can also be a "Name" specified in a LoadPicture Command.

Examples:

```
ShowPicture "images:BigBird.pic"

LoadPicture "images:Background.pic","TheBackground"
ShowPicture "TheBackground"
```

ShowBrush

ShowBrush "Brush Name" , < x > , < y > { , BRUSHPALETTE }

This command shows a DPaint style brush at a specified location on the current window. Optionally, you can use the color palette saved with the brush. You will have more predictable results if the Brush uses the same resolution as the current picture.

BRUSHPALETTE indicates to use the palette saved with the brush.

Examples:

```
ShowBrush "Brushes:Bat.br" , 500 , 25

LoadBrush "brushes:helloworld.br" , "theimage"
ShowBrush "theimage", 50 , 50 , BRUSHPALETTE
```

Transparent

Transparent « logical expression »

When a brush is saved, the background color is called Transparent. This means instead of seeing the color, you can "see through" it. CanDo allows you to turn this ON or OFF before showing a brush.

The « logical expression » indicates whether to turn transparency on or off. When the expression is TRUE (or ON) you will be able to see through the transparent color. Otherwise, the background color will be shown. Most of the time you will want to use the constants ON or OFF.

Examples:

```
Transparent ON
ShowBrush "Brushes:Arrow.br",20,30

Transparent OFF
ShowBrush "Brushes:Dog.br",120,40
```

ClipPicture

ClipPicture "Picture Name" { ,CHIP }

ClipPicture allows you make a copy of the current card's window. The "Picture Name" names the image so you can use it later in a ShowPicture or SavePicture Command. If the "Name" is currently being used, it will be replaced with the clipped picture. The CHIP keyword indicates to save the image in the Amiga's Chip memory.

ClipBrush

ClipBrush < x > , < y > , < width > , < height > ,
"Brush Name" { ,CHIP }

ClipBrush lets you grab a portion of the current card's window and use it as a Brush. The < x > and < y > indicates the Origin, the location of the upper left corner, of the area to be clipped. The < width > and < height > indicates the number of pixels horizontally and vertically, from the Origin, to clip. The "Brush Name" gives it a name so you can use it in a ShowBrush or SaveBrush Command. If the "Brush Name" currently is being used, it will be replaced with the clipped brush. The CHIP keyword indicates to save the brush in the Amiga's Chip memory.

SetClipTransparentColor

SetClipTransparentColor < color >

The SetClipTransparentColor Command sets the background color used in a clipped brush. This color is transparent in a ShowBrush Command when Transparent is ON.

SavePicture

SavePicture "Picture Name" { , "Filename" }

The SavePicture Command lets you save a picture. The picture can be one that was loaded or created using the ClipPicture Command. It can either save it in the original file or create a new one. The "Picture Name" indicates the picture to save. The optional "filename" allows you to explicitly indicate the file name to use. If the "Filename" is not specified, CanDo will either save the file using the original file name, or use the "Picture Name" as the file specification if it was created using the ClipPicture Command.

SaveBrush

SaveBrush "Brush Name" { , "Filename" }

The SaveBrush Command allows you to save a brush. This command works similarly to the SavePicture Command, except the image is saved in the Brush format. The parameters work the same way as the SavePicture Commands parameters.

ShowPalette

ShowPalette "Name"

This command changes the palette to the one saved with a picture or brush. The "Name" is either the file specification or the name used with the LoadPicture or LoadBrush Commands. If the image is not currently in memory, the ShowPalette Command will use the "Name" as a file specification from which to read the palette from a file.

Draw Commands

CanDo uses the Amiga Operating System supplied graphics routines. They do not make color corrections for HAM. This means you may see some "jaggies" when drawing in HAM mode. This can be minimized or eliminated by only drawing on portions of the picture that use the 16 primary colors. (AmigaWorld's March 1989 issue has a good description of HAM.)

AreaCircle

AreaCircle < x > , < y > , < r >

Draws a filled circle with a center of < x >,< y > and a radius of < r >. The circle is filled with the color in PenA. The aspect ratio is not corrected for High-Resolution Screens.

Example:

```
AreaCircle 50 , 50 , 25
```

AreaEllipse

AreaEllipse < x > , < y > , < xr > , < yr >

Draw a filled ellipse with a center at < x >,< y >. The horizontal radius of the ellipse is < xr > and the vertical radius is < yr >. The ellipse is filled with the color in PenA.

Example:

```
AreaEllipse 50 , 50 , 25 , 50
```

AreaRectangle

AreaRectangle < x > , < y > , < w > , < h >

Draws a filled rectangle with an upper left coordinate at < x >,< y >. The width of the rectangle is < w > and the height is < h >. The rectangle is filled with the color in PenA.

Examples:

```
AreaRectangle 50 , 50 , 100 , 100
AreaRectangle 52 , 52 , 96 , 96
```

FloodFill

FloodFill < x > , < y >

The FloodFill Command fills a solid shape starting at < x >,< y >. It will fill using the color in PenA. The shape is determined by the color at location < x > , < y >.

Example:

```
SetPen 7
FloodFill 50 , 50
```

FillToBorder

FillToBorder < x > ,< y > , < BorderColor >

The FillToBorder fills a shape starting at < x >,< y > and is outined in the color specified by the < BorderColor >. Like FloodFill, it will fill using the color in PenA.

Example:

```
SetPen 3
DrawRectangle 10 , 10 , 100 , 100
SetPen 1
FillToBorder 20 , 20 , 3
```

DrawCircle

DrawCircle < x > , < y > , < r >

Draws a circle with a center of < x >,< y > and a radius of < r >. The aspect ratio is not corrected for High-Resolution Screens.

DrawEllipse

DrawEllipse < x > , < y > , < xr > , < yr >

Draws an ellipse with a center of < x > ,< y >. The horizontal radius of the ellipse is < xr > and it's vertical radius is < yr >.

Example:

```
DrawEllipse 50 , 50 , 25 , 50
```

DrawLine

DrawLine < x1 > , < y1 > , < x2 > , < y2 >

This command will draw a line from < x1 >,< y1 > to < x2 >,< y2 >. The current pen position will be set to < x2 >,< y2 >.

Examples:

```
DrawLine 50,50,100,50

DrawLine 50,52,100,52
```

DrawPixel

DrawPixel < x > , < y >

Draws a single pixel at a certain location. This command uses the present drawmode and will set the current pen position to that of the pixel drawn.

Parameters:

< x > The Number pixels from the windows right hand border.

< y > The Number pixels from the windows top border.

DrawRectangle

DrawRectangle < x > , < y > , < w > , < h >

Draw a rectangle with an upper left coordinate of < x >,< y >. The width of the rectangle is
< w > and the height is < h >. No aspect ratio computations are computed. The current pen position is set to the < x >,< y > of the rectangle.

Examples:

```
DrawRectangle 50,50,100,100

DrawRectangle 52,52,96,96
```

DrawTo

DrawTo < x > , < y >

This command is similar to the line Command. However, it uses the current pen position for the starting point for the line. It draws a line from the current pen position to the specified window coordinate. After drawing the line, the pen position will be set to the specified window coordinates.

Example:

```
Line 50 , 50 , 100 , 50

DrawTo 100 , 100
```

MovePen

MovePen < x > , < y >

Moves the pen to the specified window coordinate. This command does not draw any points in the window.

Example:

```
MovePen 50 , 50
```

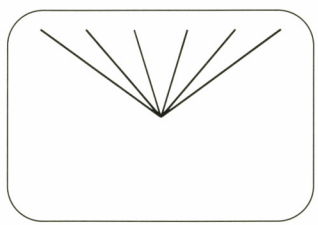
RayTo

RayTo < x > , < y >

This is similar to the DrawTo Command. With this command a line is drawn from the current pen position to the specified window coordinate. However, the current pen position is not modified.

Example:

```
Let XPos = 60
MovePen 160 , 100
While XPos <= 260
        RayTo XPos,10
        Let XPos = XPos + 40
EndLoop
```



ClearWindow

ClearWindow { < color > }

The ClearWindow Command clears the window to its inital state. This means it will redisplay the image if it is a Picture Window. If it is not a Picture Window, it will clear the window to the color specified in the Window Colors Requester or to the optional < color >. In either case, all visible objects will be redisplayed.

Examples:

ClearWindow

ClearWindow 5

Graphics Control

The Graphics Control Commands change various values that effect the other Graphic Commands.

GetRGB

GetRGB < col.reg >, < red var >, < green var >, < blue var >

Gets the RGB values from a specified color register. The RGB values range from 0 to 255. Remember that different view mode use the color registers very differently.

< col.reg >	the color register to read
< red var >	variable for the red part of the register
< green var >	variable for the green part of the register
< blue var >	variable for the blue part of the register

Example:

GetRGB 1, RedValue, GreenValue, BlueValue

SetAreaDrawMode

SetAreaDrawMode NORMAL or OUTLINE

The SetAreaDrawMode Command only effects the Area Commands: AreaCircle, AreaEllips, and AreaRectangle. The default mode is NORMAL. When it is NORMAL, the area is draw in a solid color using PenA. When it is OUTLINE, the Area is drawn using PenA outlined with the color in PenO. When this draw mode is set, all subsiquent Area Commands will use the specified Mode.

SetDrawMode

SetDrawMode { Normal } { Jam1 } { Jam2 }
{ Complement } { InverseVideo }

This command allows the user to specify the drawing style they wish to use. The keywords can be used with one another (i.e. SetDrawMode Normal InverseVideo).

{ Normal }	Draw with the primary pen
{ Jam1 }	Same as the Normal switch
{ Jam2 }	Draws with both primary and secondary pen
{ Complement }	Performs a logical XOR on the pixel's own color info.
{ InverseVideo }	Performs a logical NOT on the other draw modes

SetPen

SetPen < pena > { ,< penb > { ,< penO > } }

Set the primary, secondary and outline pen's color register. This command allows the user to specify which color to draw with. The PenB and penO are optional.

< PenA >	The color register for the primary drawing pen
< PenB >	The color register for the secondary drawing pen.
< penO >	The color register for the area outline pen.

Example:

```
SetPen 1

DrawPixel 10,14
```

SetRGB

SetRGB < col.reg >, < red >, < green >, < blue >

Set the RGB values for the specified color register. The RGB values range from 0 to 255. Remember that different view modes use the color registers very differently.

< col.reg >	The color register to be modified.
< red >	The Red Value.
< green >	The Green Value.
< blue >	The Blue Value.

Example:

```
SetRGB 3, RedValue, 4, BlueValue+BlueIncrement
```

CycleColors

CycleColors < From-Color > , < To-Color >
{ , FORWARD or BACKWARD }

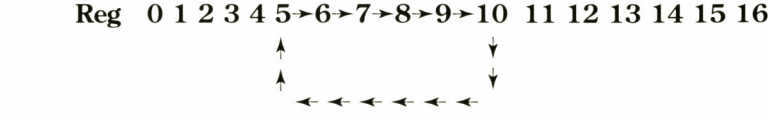
The CycleColors Command cycles the current color palette. The < From-Color > < To-Color > indicates the range. It does not make any difference if the < From-Color > is less than the < To-Color >.

The CycleColors Command rotates the colors once from the lower to the higher. The optional keywords FORWARD and BACKWARD specify the direction of the rotation. If neither is specified, it defaults to FORWARD. FORWARD indicates lower values are rotated upward. BACKWARD indicates the higher values are rotated downward.

The CycleColors Command can be placed in a reoccurring Timer Object to provide continuous cycling.

Example:

```
CycleColors 5 , 10 , FORWARD
```



Print Commands

The Print Commands allow you to print text messages to the window. You can use various fonts, sizes, colors, and enhanced print styles.

PrintText

PrintText < x > , < y > , "string"

Prints the "string" using the current pen colors, print style, and font. The x,y coordinate specifies the upper left corner of the text string.

Examples:

```
PrintText 50 , 50 , "Hello World"
```

SetPrintStyle

SetPrintStyle StandardFlags { ExtendedFlags }
{ , < ExtPen1 > { , < ExtPen2 > } }

The SetPrintStyle Command specifies the style of print for subsequent PrintText Commands. The StandardFlags are PLAIN, ITALIC, BOLD, and UNDERLINED. You can specify one or more of these. Optionally, you can specify one of the ExtendedFlags: SHADOW, OUTLINE, GHOSTED, or EMBOSSED. Make sure you do not seperate any of the flags with commas.

The Extended Pens, < ExtPen1 > and < ExtPen2 >, are used with the Extended styles. The PrintText command uses the color in PenA as the primary color for drawing the text. When using an Extended Style, the < ExtPen1 > is used as a secondary color. The EMBOSSED style uses the additional color < ExtPen2 >.

Examples:

```
SetPen 1
SetPrintStyle BOLD ITALIC
PrintText "This uses a single color. It is BOLD and
          Italic." , 20 , 20

SetPen 0
SetPrintStyle GHOSTED,5
PrintText "The Primary Color is 0 Ghosted in 5." , 20 , 80

SetPen 3
SetPrintStyle BOLD EMBOSSED,1,2
PrintText "The Center is Pen 3, with Color 1 and 2
          on each side." , 20 , 120
```

SetPrintFont

SetPrintFont "Fontname" , < Pointsize >

Set the print font for the PrintText Command. If the operating system cannot find the point size you are looking for, it will give you the next smallest size available. If the operating system cannot find the font you are looking for, it will give you the font Topaz 80.

Example:

```
SetPrintFont "Topaz" , 9
PrintText "Hello World", 50,50
```

GetTextDimensions

GetTextDimensions "Text", Width-Variable, Height-Variable

This command sets the contents of two variables to the width and Height of the indicated "Text" string. It evaluates the "Text" using the current font, point size, and Print style. The Width-Variable and Height-Variable must be valid variable names. The variables will be set to the corresponding values.

Example:

GetTextDimensions "Hello", Hello Width, Hello Height

Functions

This Function can be used in expressions to return information relating to Graphic Commands.

ColorOfPixel

< Integer > = ColorOfPixel (< x > , < y >) - This is the Color of the Pixel at the specified x , y location

System Variables

< Integer > = PenA - This is the Color Register for PenA.

< Integer > = PenB - This is the Color Register for PenB.

< Integer > = PenO - This is the Color Register for PenO.

< Integer > = ClipTransparentColor - This function returns the background color used when clipping brushes.

« logical » = TransparentStatus - This function returns TRUE if Transparent is enabled,and FALSE if not.

Screen and Window Commands

The *Window Object Editor* allows you to define the Window when it opens. The Window will open on either the Workbench, or a Custom Screen. If you are not familiar with the way the Amiga works, the differences between a Window and a Screen can be a little confusing. The Screen is the full-width area of the display which defines the display mode, resolution, and color palette. A Window is displayed on a Screen and can be its full size or smaller.

These Commands allow you to change the position and attributes of both the Screen and Window.

MoveScreen

MoveScreen < Delta - X > , < Delta - Y >

The MoveScreen Command moves the screen a specified number of pixels right or left, < Delta - X > , or up or down < Delta - Y >. Note that under Amiga operating systems 1.3 and before, it is not possible to move a screen horizontally. The delta integers make it easy to move the screen specific distances without reference to its current position.

Example:

MoveScreen 0 , - 10

This would move the screen up 10 pixels.

MoveScreen 0 , 10 - ScreenY

This example would move the screen to be 10 pixels from the top. Because ScreenY indicates the current position, 10-ScreenY will always position it 10 pixels from the top.

ScreenTo

ScreenTo FRONT or BACK

The ScreenTo Command moves the screen either to the back of the displayed screens or to the front of the displayed screens.

Examples:

ScreenTo BACK

ScreenTo FRONT

ScreenTitleBar

ScreenTitleBar « logical expression »

This command makes the screen (if it is your own, custom screen, and not Workbench's screen) have a visible title bar. By using this command and eliminating your window's title, and border, and all of your window's standard objects (dragbar, closebutton, etc.), you can use the screen's dragbar to move the screen up and down with the mouse.

SetScreenTitle

SetScreenTitle "Text"

This sets the title of the screen to be the indicated text when your window is the active window. If you have created a Workbench window, when your window is inactive, the screen title bar will reflect some other name (probably Workbench), and when your window is active the screen will show the title set with this command.

MoveWindow

MoveWindow < Delta - X > , < Delta - Y >

The MoveWindow Command moves the window a specified number of pixels right or left, < Delta - X >, or up or down < Delta - Y >. The delta integers make it easy to move the window specific distances without reference to its current position.

WindowTo

WindowTo FRONT or BACK

The WindowTo Command moves the window either to the back of the displayed windows or to the front of the displayed windows. Because a Custom Screen has only one Window, this command is only useful on a Workbench Window. If the window was created in Backdrop mode (see Window Options), it cannot be moved in front of other windows.

Examples:

WindowTo BACK

WindowTo FRONT

SetWindowTitle

SetWindowTitle "Text"

The SetWindowTitle Command sets the title of your window to the indicated text. If you have previously removed all of the standard window objects, border and title, this will only add a bar to the top of your window in which the title will be displayed. The Initial Window Title is set on the *Window Editor Requester.*

SetWindowLimits

SetWindowLimits < MinX > , < MinY > , < MaxX > , < MaxY >

The SetWindowLimits Command controls the minimum and maximum size of your window. With these limitations in place, your window cannot be resized by the user to any dimensions outside of these ranges. It is often convenient to use this command in the AfterStartup script of your card. In this way, immediately after the window Object has been opened, you can set the limitations on its size before the user has had a chance to resize you window.

Current Screen and Window Information

< Integer > = MouseX	- Horizontal position of the mouse
< Integer > = MouseY	- Vertical position of the mouse
< Integer > = WindowColors	- Number of Colors
< Integer > = WindowHeight	- Height of the window
< Integer > = WindowWidth	- Width of the window
"String" = WindowTitle	- Text in Window Title Bar
< Integer > = WindowX	- Horizontal offset of the window
< Integer > = WindowY	- Vertical offset of the window
< Integer > = ScreenColors	- Number of Colors (Same asWindowColors)
< Integer > = ScreenWidth	- Width of the screen
< Integer > = ScreenHeight	- Height of the screen
< Integer > = ScreenX	- Horizontal offset of the Screen
< Integer > = ScreenY	- Vertical offset of the Screen
« logical » = Interlace	- TRUE when the screen is Interlace
« logical » = Hires	- TRUE when the screen is High-Resolution
« logical » = NTSC	- TRUE when NTSC Amiga; FALSE for PAL

Brush Animation Commands

The Brush Animation Commands allow you to show, move, and control DPaint III BrushAnims.

You simply load the animation into a buffer using the LoadBrushAnim Command. The ShowBrushAnim Command begins the animation on the screen. You can specify a velocity for movement with the MoveBrushAnim Command, along with an acceleration rate. Or you can use the MoveBrushAnimTo Command, which moves the brush animation from it's current location to a destination with a specified velocity and duration.

Features

- Use of DPaintIII brush animations complete with the Forward, Reverse, PingPong and Duration parameters.
- Showing of multiple animations at the same time.
- Movement of animations in your window with full clipping.
- Real-time decompression in which each frame is decompressed into chip memory as needed, or
- One-time decompression in which all frames are decompressed into chip memory. This uses more memory but provides faster animations.
- Restoration of the background when the BrushAnims move.
- Support for Transparent brush animations.
- Sequenced animation that shows each frame in an animation before moving.

LoadBrushAnim

LoadBrushAnim "filename" { ,"BrushAnim Name" { ,loadflags } }

The LoadBrushAnim Command preloads a DPaint III style brush animation. The "filename" contains the file specification for the animation to be loaded. The optional "BrushAnim Name" allows you to refer to the brush animation by a name other than the "filename".

ShowBrushAnim

ShowBrushAnim "BrushAnim Name" ,< x > ,< y >

Adds the indicated brush animation to the window. The "BrushAnim Name" indicates the DPaint III style brush animation to show. The name can be a file specification or a name created using the LoadBrushAnim Command. If the file is not already in memory, it will be loaded. The < x > , < y > indicate the initial location of the brush animation. You must use the ShowBrushAnim Command, to display your brush animation, before you can move it using either the MoveBrushAnimTo or **MoveBrushAnim** Commands. In addition, a BrushAnim must be shown to allow Animation Object's Scripts to be performed.

MoveBrushAnimTo

MoveBrushAnimTo "BrushAnim Name" ,< x > ,< y > { ,ticks }

Moves the brush animation to the specified < x > ,< y > coordinates. If < ticks > are not provided, the BrushAnim will move instantly. Otherwise, movement will occur over the time specified in < ticks >. If you have an Animation Object watching this BrushAnim that has an OnDestination script defined, it will be performed when the BrushAnim arrives at the specified location.

MoveBrushAnim

MoveBrushAnim "BrushAnim Name" , < Xvel > , < Yvel > ,
{ < Xacc > , < Yacc > , { < ticks > } }

The MoveBrushAnim Command moves a currently displayed brush animation "BrushAnim Name" in a direction and acceleration indicated by the < Xvel > ,< Yvel > ,< Xacc > and < Yacc > values.

The < Xvel > ,< Yvel > are the velocity values are added to the x , y values to move the brush animation. If the brush animation has Linear Movement (default) then the velocity values are added after each frame. If the brush animation has Sequenced Movement, then the velocity values are added after each animation sequence.

The optional < Xacc > ,< Yacc > are the acceleration values that are added to the velocity values after each time they are added to the X , Y values. This will cause the movement of the brush animation to accelerate. If they are not specified, they default to 0. This causes the animation to move at a constant velocity.

The optional < ticks > value indicates the number of frames to move in the specified direction. After it has moved for the specified number of frames, it will stop. If you have an Animation Object watching this BrushAnim that has an OnDestination script defined, it will be performed when the BrushAnim arrives at the specified location.

If the < ticks > is not specified, the animation will move continuously in the indicated direction.

GetBrushAnimCoordinates

GetBrushAnimCoordinates "BrushAnim Name" ,Xvariable ,Yvariable

Returns the x , y coordinates of the brush animation in the indicated variables.

Example:

GetBrushAnimCoordinates "Helicopter" ,HeliX ,HeliY

This instruction will put the x location into the variable HeliX and the y location into the variable HeliY.

RemoveBrushAnim

RemoveBrushAnim "BrushAnim Name"

The RemoveBrushAnim Command stops animating the specified "BrushAnim Name". However, it does not erase it from the window and does not remove the BrushAnim from memory. The BrushAnim can be removed from memory with the Flush Command.

SetBrushAnimFlags

SetBrushAnimFlags "BrushAnim Name" ,< brushanimflags > [, { ticks }]

This Command is used to specify some options for the specified "BrushAnim Name". If the brush animation is not currently in memory, it will be loaded.

The brushanimflags are keywords that indicate specific options. Make sure that you do not separate the keywords with commas.

COMPRESSEDMODE or **DECOMPRESSEDMODE**

COMPRESSEDMODE or DECOMPRESSEDMODE indicates how the animation is kept in memory. COMPRESSEDMODE is the default mode; it takes less memory. However, it takes more time to animate because each frame must be created before it can be displayed.

RESTOREBACKGROUND or **LEAVEIMAGE**

RESTOREBACKGROUND or LEAVEIMAGE indicates whether CanDo should save and restore the background each time the animation is shown. When RESTOREBACKGROUND option is used, you can move the animation without leaving a trail. However, it can substantially slow down the animation. LEAVEIMAGE is the default mode.

USEMASK or **NOMASK**

USEMASK or NOMASK is the same thing as transparent mode for normal brushes. The background color saved with the Brush Animation is transparent when it is displayed. The USEMASK option increases the time required to display each animation frame, especially in COMPRESSEDMODE.

SEQUENCEDMOTION or **LINEARMOTION**

SEQUENCEDMOTION or LINEARMOTION specifies how the animation is moved. SEQUENCEDMOTION shows each frame of the animation before moving it. LINEARMOTION moves the animation on each frame. SEQUENCEDMOTION only effects the MoveBrushAnim Command and not the MoveBrushAnimTo Command.

FORWARD, BACKWARD and **PINGPONG**

FORWARD, BACKWARD and PINGPONG specify the order in which the animation's frames are shown. FORWARD plays the animation from first frame to last. BACKWARD plays the animation from last frame to first. PINGPONG switches between forward and backward motion each time the first or last frame of the animation is displayed.

NONE

NONE changes no flags. It simply allows you to specify the < ticks > without changing any flags.

< ticks >

The < ticks > value indicates how many ticks to remain on each frame of the animation.

BrushAnims

BrushAnims « logical expression »

When the « logical expression » is FALSE it stops all animations. When it is TRUE
it starts all animations.

Function

This Function can be used in expressions to return information relating to the Brush Animation Commands.

FrameOfAnimation

< intger > = FrameOfAnimation ("BrushAnim Name")

The FrameOfAnimation Function returns the current frame number of the specified "BrushAnim Name". If the BrushAnim has not been loaded, the FrameOfAnimation Function will return 0.

System Variable

« logical » = AnimationStatus - TRUE if animations are currently running in your window.

Audio Scripting Commands

The Audio Commands play Mono (non-stereo) digitized sounds. The Amiga IFF standard for these sounds is called 8SVX. CanDo loads these sounds and plays them through one of the four Audio Channels.

LoadSound

LoadSound "filename" { ,"Sound Name" }

The LoadSound Command loads an 8SVX sampled sound. The Optional "Sound Name" allows you to refer to it by a name other than the Filename. LoadSound pre-loads the sound before playing it. It can then be used in a PlaySound or PlaySoundSequence command. This Command is particularly useful for loading a sound during the Card's Startup Script. By preloading it, there will not be a delay the first time the sound is played.

Example:

```
LoadSound "Sounds:Laugh.snd"
PlaySound "Sounds:Laugh.snd"
LoadSound "Sounds:Bang.snd", "Bang"
PlaySound "Bang"
```

PlaySound

PlaySound "Sound Name" { , AudioFlags { ,< period > } }

The PlaySound Command plays a single 8SVX sound. The "Sound Name" can be a File specification or Name created using the LoadSound Command. The AudioFlags and Period are described below.

PlaySoundSequence

PlaySoundSequence "Document Name" { , AudioFlags { ,< period > } }

The PlaySoundSequence Command plays a list of sounds. The list is contained in a Document (See Document Commands). A Document is just like a text editor. The PlaySoundSequence plays each "Sound Name" in the Document one after another. Each "Sound Name" must be on a separate line. It can be a file specification or a Name created using the LoadSound Command.

Example:

```
MakeDocument "Shooting Sequence"
Type "Sounds:WatchOut.snd",NEWLINE
Type "Sounds:Bang.snd",NEWLINE
Type "Sounds:YouGotMe.snd",NEWLINE
PlaySoundSequence "Shooting Sequence"
```

ONCE

ONCE, which is the default, indicates the sound should only be played one time.

Example:

```
PlaySound "Bang", ONCE
```

CONTINUOUS

CONTINUOUS indicates the sound should be repeated until an Audio OFF Command is executed.

Example:

```
PlaySound "Music", CONTINUOUS , 600
```

WAIT

When WAIT is specified, the sound begins to play when an Audio ON is executed. This can be used to start multiple sounds at the same time.

Example:

```
PlaySound "Sound1", WAIT
PlaySound "Sound2", WAIT
PlaySound "Sound3", WAIT
Audio ON
```

QUEUE

Normally, a sound will only play when an Audio Channel is available at the time the PlaySound Command is performed.

QUEUE is like telling the sound to wait in line until an Audio Channel is available. As soon as one is, the sound will begin to play. There is no limit to the number of sounds that can be queued.

QUEUEPREVIOUS

QUEUEPREVIOUS is similar to QUEUE. Instead of playing on any channel, if one is available, it will play on the same channel as the most previous sound.

Example:

```
PlaySound "Sounds:WatchOut.snd"
PlaySound "Sounds:Bang.snd", QUEUEPREVIOUS
PlaySound "Sounds:YouGotMe.snd", QUEUEPREVIOUS
```

< period >

The optional period value overrides the sound's natural period/rate saved within the 8SVX sound file. This value can range from 124, being the fastest, to 65535 being the slowest.

SetVolume

SetVolume < volume > { ,< channel > }

The SetVolume Command sets the volume for subsequently played sounds. It does not change the volume of currently playing sounds.

The Volume is an integer ranging from 0, being no volume, to 64 being full volume.

The Channel number is optional. When it is not specified, the indicated volume is used for all channels. On the other hand, specifying a channel between 0 and 3 sets the volume for a single channel.

Example:

SetVolume 64
SetVolume 32,3

SetChannel

SetChannel < channel >

The SetChannel allows you to specify the channel that is used for the next PlaySound or PlaySoundSequence command. Usually, these commands look for any available channel. If one is available, it plays the sound.

NOTE:

If you are running another application that is using the specified channel, the sound will not be played.

Audio

Audio « logical expression »

This Command turns all Audio Channels On or Off. The logical expression is evaluated to True or False. (On = True and Off = False). When the expression is True (ON), all PlaySounds using the WAIT keyword are started. When the expression is False, all sounds (regardless of the WAIT usage) will be terminated.

Document Commands

A Document is simply a data area that contains text. Using CanDo's Memo Object, you can read from, and type text into a Document. The List Object allows you to select lines from a Document.

This section describes the CanDo scripting Commands that work with Documents. These Commands allow you to work with a Document in the same manner as you would from a Text Editor.

Commands such as Type, work as though you were typing characters from a keyboard. 'Type "Hello World"' enters the text "Hello World" into a Document. A cursor position is maintained for each Document. Typing text into the document occurs at the cursor position.

Cursor movement Commands, such as 'MoveCursor LEFT 5', moves the cursor in the document. The Delete Command remove characters at the cursor position.

The Commands LoadDocument and SaveDocument allow you to load and save text files.

The Memo and List objects show the contents of a single Document. The Document is specified in the Document Definition Requestor in the "Document Name" field. This makes the contents of the Document Visible in the Object. However, you can create Documents that are not visible. These can be particularly useful in working with text "behind the scenes". For example, some applications may copy portions of an invisible Document into visible Documents.

While CanDo provides a number of Commands that work with documents, not all are necessary for use in simple applications. This section is organized with the most common and necessary Commands listed first. The last part of this section describes the CanDo Variables that give information about the current document.

First, it is necessary to create an empty document. This is done using the MakeDocument Command.

MakeDocument

MakeDocument "Document Name"

This Command creates an empty document that is identified the "Document Name" string. The new Document becomes the Current Document. This means Document Commands, such as Type, work with the new document.

CanDo applications can have more than one Document. However, the Document Commands only work with one Document at a time. The Command WorkWithDocument Command specifies the new Current Document.

WorkWithDocument

WorkWithDocument "Document Name"

This Command changes the current Document. All subsequent Document Commands will use the indicated Document. To work with a Document that is visible through a Memo or List Object, you should specify the name indicted in the "Document Name" field of the Document Definition Requestor.

If the Document specified with the "Document Name" does not currently exist, CanDo will make a new Document with the specified name. Furthermore, it will attempt to automatically load a file using the "Document Name" as a file specification. If the file does not exist, it will create an empty document. This allows you to easily create a document and load it with a file if it exists. However, if you do not want CanDo to attempt to load a file, you must first create the document using the MakeDocument Command.

LoadDocument

LoadDocument "filename" { , "Document Name" }

Reads the contents of the specified "filename" into a "Document Name". If a Document with a name of "Document Name" currently exists, it will clear the document before loading the document.

SaveDocument

SaveDocument "Document Name" { ,"filename" }

The SaveDocument Command saves the specified "Document Name". When the "filename" is not specified, CanDo will replace the file that was originally loaded or use the "Document Name" as a filename if the Document was not loaded. If the "filename" is specified, it will use it as the filename.

Type

Type "String" { ,NewLine }

This Command "Type's" the string into the current Document. The string can contain imbedded LF's for RETURNs, DELETE's and BACKSPACE's. These characters can be created using the Char Function.

Example:

Type "Hello" || World

This example concatenates the string constant "Hello" with the contents of the variable World, and types it into the current Document.

SplitLine

SplitLine { < count > }

This Command is the same as a RETURN key. The optional < count > specifies the number of times to repeat the Command.

Examples:

SplitLine
SplitLine 5

NewLine

NewLine

This Command is the same as moving the cursor to the end of the line and pressing the RETURN key. It does not have any parameters. It is a useful Command for creating a new line without any concern whether the cursor is currently in the middle of a line.

MoveCursor

MoveCursor direction { ,< Count > }

Moves the cursor in the specified direction. If no count is specified, the cursor moves once. Otherwise, it moves the direction the number of times specified in the count. If the count is negative, the count moves in the opposite direction.

direction: You can only use one of the direction key words.

UP

Moves the cursor Up. This Command has no effect when the cursor reaches the first line in the Document.

DOWN

Moves the cursor Down. This Command has no effect when the cursor reaches the last line in the Document.

LEFT

Moves the cursor Left. If the cursor is at the beginning of a line (and it is not the first line in the Document), it moves to end of the previous line.

RIGHT

Moves the cursor right. If the cursor is at the end of a line (and it is not the last line in the Document), it moves to the beginning of the Next line.

Examples:

MoveCursor UP
MoveCursor RIGHT , 10
MoveCursor RIGHT , RepeatCount + 5

MoveCursorTo

MoveCursorTo location area

This Command moves the cursor to the start of or to the end of the current Document, line, word, previous word, or next word.

location - You can only use one of the direction key words.

STARTOF
Indicates to move the cursor to the beginning of the specified Area.

ENDOF
Indicates to move the cursor to the end of the specified Area.

area - You can only use one of the direction key words.

DOCUMENT
Moves the cursor to the specified Location within the current Document.

LINE
Moves the cursor to the specified Location on the current line.

NEXTWORD
Moves the cursor to the specified Location on the word after the current word.

THISWORD
Moves the cursor to the specified Location on the current word. If the cursor is not on a word, this Command will not have any effect.

PREVIOUSWORD
Moves the cursor to the specified Location on the word before the current word.

Examples:

MoveCursorTo STARTOF DOCUMENT
MoveCursorTo ENDOF LINE
MoveCursorTo STARTOF NEXTWORD

PositionOnLine

PositionOnLine < line >

Moves the cursor to the specified Line. If the specified line number is greater than the number of lines in the Document, the cursor is placed on the last line. The cursor offset is not effected.

Clear

Clear LINE or DOCUMENT

This Command clears either the current line or the current Document.

LINE

Indicates to clear the current line. This is different than deleting the line. All characters are erased, but the line remains with the cursor in the first column.

DOCUMENT

Indicates to clear the current Document. This is different than deleting the Document. The Document still exists with all the characters erased.

Examples:

ClearText LINE
ClearText DOCUMENT

Delete

Delete KeyWord { ,< count > }

This Command deletes specified text from the current Document. An optional count indicates the number of times to repeat the action. (While the syntax allows it, a count used with ToStartOfLine and ToEndOfLine has no effect). Only one Keyword may be specified.

KeyWord - You can only use one of the direction key words.

LINE

Deletes the Current Line. This removes the line completely, positioning the cursor at the beginning of the next line.

TOSTARTOFLINE

Deletes all characters before the cursor to the beginning of the line. (Count has no effect.)

TOENDOFLINE

Deletes all characters from the cursor to the end of the line. This includes the current character.

CHARACTER

If no count is specified, a single character is deleted to the right of the cursor location. Otherwise, the count specifies the number of characters to delete. If the count is positive, then charac ters are deleted to the right of the cursor. When the count is nega tive, characters are deleted to the left of the cursor. This is the same as a Backspace. When the count is greater than the number of characters remaining on the line, the current line is concatenated with the next line (previous line in the case of a backspace). This concatenation will count as a deleted character.

Examples:

Delete LINE	;Deletes a single line
Delete LINE 5	;Deletes 5 lines
Delete TOSTARTOFLINE	;Deletes to the beginning of a line
Delete CHARACTER	;Deletes a single Character
Delete CHARACTER 5	;Deletes 5 characters
Delete CHARACTER -Repeat	;Backspaces < Repeat > times

SearchFor

SearchFor "text" { , qualifiers }

Searches for the specified string. The qualifiers allow searching for complete words and for ignoring case differences.

qualifiers - You can use one or both BYWORD and NOCASE.

BYWORD

This qualifier allows searching for complete words. The search will not match a string if it is a portion of a word.

NOCASE

This qualifier allows searching for a string ignoring case differences between the search string and the potential target string.

Examples:

SearchFor "FooMan Chew"
SearchFor "is", BYWORD
SearchFor "bald", NOCASE BYWORD

Replace

Replace "fromtext","totext" { , { GLOBAL } { BYWORD } { NOCASE } }

The Replace Command replaces the "fromtext" with the "totext". It will search forward from the current cursor location for the "fromtext". If the "fromtext" is not found, nothing will happen. If it is found, the cursor will be moved to the character following the new "totext".

qualifiers - You can use one or more qualifiers with the Replace Command.

GLOBAL

GLOBAL replace all remaining matches of the "fromtext" with the "totext" string. When it is not specified, only one replace can potentially be replaced.

BYWORD

BYWORD indicates the "fromtext" will not match if it is contained within a word.

NOCASE

NOCASE indicates the "fromtext" will match even if some of the characters are not the same. If NOCASE is not specified, it must match exactly.

InsertDocument

InsertDocument "Document Name"
{ , < Start Line > { , < LineCount > } }

The InsertDocument document Command copies the text from the specified "Document Name" into the current document at the current cursor position. The optional < Start Line > specifies the first line, from the "Document Name" to copy. When it is not specified, it starts on line 1. The optional < LineCount > indicates the number of lines to copy. If it is not specified, all lines after the < Start Line > are copied.

If the "Document Name" does not exist, CanDo will attempt to automatically load it using the "Document Name" as the file specification.

Examples:

InsertDocument "Work Document"
InsertDocument "S:startup-sequence"
InsertDocument "My Resume",5
InsertDocument "DF1:WorkFile.TXT",25,10

SetWordDelimiters "delimiterlist"

This Command sets the word delimiter list. The word delimiters define the characters that separate words. The default delimiters are , ( ) ! @ # $ % " & * - = + \ | < > ? / in addition to the TAB characters and Line Delimiters. A word is defined to be characters that have one or more of the word delimiters on both sides of the word.

The "delimiterlist" string contains the delimitating characters for words. This string can be any size. The beginning and ending of a line are always word delimiters. As such, they are not explicitly in the list, but are always implicitly delimiters.

The following list of CanDo variables return information about the Current Document. These are write only variables that can be used in any expression.

< **Integer** > = **TheLineNumber** - line number the cursor is on.

< **Integer** > = **TheColumnNumber** - column number the cursor is on.

" **String** " = **TheLine** - the character string for the current line.

" **String** " = **TheCharacter** - the character the cursor is on ("" if at the end of line)

" **String** " = **TheWord** - the current word the cursor is on. This value is based on the word delimiters.

" **String** " = **CharsToBegOfLine** - the characters before the cursor on the current line.

" **String** " = **CharsToEndOfLine** - the characters from the cursor to the inc of the current line.

< **Integer** > = **LinesInDocument** - the number of lines in the current Document.

< **Integer** > = **LengthOfLine** - the length of the current line

< **Integer** > = **SizeOfDocument** - the number of bytes in the current Document.

" **String** " = **DocumentName** - the name of the current document.

< **Integer** > = **SizeOfDocument** - number of characters in Document.

" **String** " = **TheWordDelimiters** - the characters separating words in the current Document.

File I/O Commands

The File I/O Commands provide an easy way to write and read data from files. CanDo uses the Amiga's Buffered I/O system for implementing these commands. While the commands are not complicated, they are intended for those who are familiar with File I/O.

If you are mostly interested in working with text files, you might want to consider using the Document commands.

OpenFile

OpenFile "filename" , "Buffer Name" , IOFlags , AccessFlags

Opens a file for input/output purposes. Memory is the only limit to the number of opened files that can be open at one time. The "filename" specifies the file to be opened. The "Buffer Name" is a string used to identify the buffer associated with this file access.

The IOFlags are READONLY or WRITEONLY. READONLY indicates that you can only read ASCII data from the file. WRITEONLY means that you can only write ASCII data to the file.

The AccessFlags are NEWFILE, OLDFILE or APPEND. NEWFILE indicates to create a new file. If the specified file already exists, then it is deleted. OLDFILE indicates to open an existing file. If it does not currently exist, it is created. This is the DEFAULT operation. APPEND is similar to OLDFILE, except it will position all writes to the end of the file.

Example:

```
OpenFile "t:Savedata.txt" , "Data" , READONLY , OLDFILE
```

FileWriteLine

FileWriteLine "Buffer Name" , "String"

This command will output the "string" to the file referenced by the "Buffer Name". An important note about this command is that it will output a linefeed at the end of each write. The file associated with the "Buffer Name" must have been opened in WRITEONLY mode for this command to work.

Examples:

```
FileWriteLine "Data" , "This is a Data Line."

Let OutputData = Outputdata || "additional data"
FileWriteLine "Data" , OutputData
```

FileReadLine

FileReadLine "Buffer Name" , VariableName

The FileReadLine Command reads a line from a previously opened file. It creates a string and saves it in the specified VariableName. A line from the file is defined to be a string terminated with a line feed.

Example:

```
FileReadLine "Data File" , DataLine
```

FileReadChars

FileReadChars "Buffer Name" , VariableName , < NumberOfChars >

This command reads a specified number of characters, <NumberOfChars>, from a previously opened file, "Buffer Name". These characters are stored in the VariableName.

Example:

```
FileReadChars "Input File" , FiveChars , 5
```

FileWriteChars

FileWriteChars "Buffer Name" , "String" { , < length > }

This command will output the characters contained in "string" to the file associated with the "Buffer Name". This command differs from the FileWriteLine in that this command will NOT output a linefeed after each write. The referenced file must have been previously opened in WRITEONLY. There is an optional argument that specifies how many characters to output. If the length specified is greater than the length of "expression" then only what is contained in "expression" is written. If the "expression" is shorter than the specified < length >, spaces are NOT concatenated to the string.

Example:

```
FileWriteChars "Data",Outdata || spaces, 20
```

SetFileBufferSize

SetFileBufferSize < sizeinkilos >

This command sets the file input/output buffer size in kilobytes. The default for the buffer size is four (4) kilobytes. A value of 8, allocates 8192 byte buffer. The buffer size is set on a file by file basis. This command only changes the buffer size of files opened after the command is issued.

Example:

```
SetFileBufferSize 2
OpenFile "t:WorkFile.dat" , "WorkData" ,
        READONLY , OLDFILE
```

Close

Close "Buffer Name"

This command closes the file associated with the specified Data Name. The 'Flush' command performs the same function when used with a "Buffer Name". Any buffered changes that you have made to the file will be written to the file at this time.

Icon Commands

Most application programs on the Amiga have a file that goes with them called an icon file. All such icon files end with the letters ".info". These files contain imagery that Workbench uses to display a visual representation of the application on the Workbench screen or in various Workbench windows. Most diskettes and many data files also have icon files. With CanDo, you can easily create, examine, and modify icons without knowing any of the details of the icon file internal format.

LoadIcon

LoadIcon "FileName" { ,"Name" { , < load flags > } }

Loads the icon for the "FileName" into memory. The extension of ".info" is automatically appended to the filename. Therefore, to load the icon for CanDo you would type, LoadIcon "CanDo". Any valid Amiga icon can be loaded with this command, although only Project and Tool icons can be created using the MakeIcon Command.

SaveIcon

SaveIcon "Icon Name" { ,"FileName" }

This command saves the icon buffer "Icon Name" as the icon for the "FileName" indicated. The extension of ".info" is automatically appended to the filename. This command will overwrite any icon that already exists for the "FileName". If the "FileName" parameter is not included, CanDo will save the file using the original file name or use the "Icon Name" as the file specification if it was created using the MakeIcon command.

If the icon "Icon Name" is not already in memory, it will be loaded and then saved, so you can easily copy an icon from disk to a new name without first loading it.

Example:

```
SaveIcon "fred","ram:Jack"
Flush "fred"
```

This script would load the icon file "fred.info" from disk (assuming it was not already in memory) and save it into a file called "ram:Jack.info". The "fred" icon buffer in memory would then be flushed.

MakeIcon

MakeIcon "Icon Name", PROJECT or TOOL,"ImageName"
{ ,"AltImageName" }

This command creates an icon buffer with the given "Icon Name". You must specify whether this icon is to be a PROJECT or a TOOL icon (explained below), and you must indicate a Brush to use for the image of the icon. In addition, you may specify a Brush buffer to use for the highlight of the icon when it is clicked on with the mouse on Workbench. If you do not specify an AltImage Brush, the icon will complement when clicked. If either the "ImageName" or the "AltImageName" brushes are not in memory currently, they will be loaded from disk and buffers for them will be created using the indicated names.

Once an icon has been created in this way, it can be saved to disk with the SaveIcon Command.

Examples:

MakeIcon "MyProject", PROJECT, "Brushes:WorkIcon.br",

MakeIcon "MyTool", TOOL, "Brushes:Tool1.br",
"Brushes:Tool2.br"

SetDefaultTool

SetDefaultTool "Icon Name" , "DefaultTool"

This command sets the Default Tool of the icon "Icon Name". If "Icon Name" is not already in memory, it will be loaded. The Default Tool of an icon is the application that will be run if the icon is double-clicked on Workbench and is itself not an application. For example, many paint programs make icons for the pictures and brushes that they save. When the icon is double-clicked the paint program that made the icon is launched and it then loads the picture.

By using the Info menu command from the Workbench, you can see the various default tools specified by icons on your Amiga.

When choosing a default tool for icons you create or modify with CanDo, it is important to be sure that those applications properly deal with being run from the Workbench environment. CanDo and all of your applications made with CanDo can run equally well from CLI and Workbench.

SetIconImage

SetIconImage "Icon Name" , "ImageName" { ,"AltImageName" }

SetIconImage changes the image for "Icon Name" to the Brush "ImageName", and, if the "AltImageName" is supplied, will also set the highlight of the icon to that image. If the "AltImageName" is not supplied, the current highlight mode for the icon will not be changed. If the "Icon Name", or either of the Brushes, is not in memory it will be loaded into memory and will be given the indicated names.

Note: that the size of the "hit area" of the icon will be changed to reflect the size of the "ImageName" used in this command.

InsertToolTypeList

InsertToolTypeList "Icon Name"

This command TYPEs into the current Document all of the ToolTypes for the "Icon Name" (see Document commands). A ToolType is any ASCII string that contains information about any subject you like. By using the Info command from Workbench on the CanDo icon, you will see that it uses many ToolTypes to control the default configuration for CanDo. (These ToolTypes can also be set through the Workbench Info command.)

By performing this command when your CanDo application starts, you can let your user specify information which your application can use while running. For example, you could create a single-card application called "Viewer" that performed the following script in its AfterStartup script:

```
LoadIcon TheOriginDirectory || "Viewer"   ;load our own icon
MakeDocument "pictures"
InsertToolTypeList                        ;get the filenames
MoveCursorTo StartOf Document

While TheLine < > " "                     ;continue until a blank line
        If FileType( TheLine ) = "Picture"  ;make sure it's OK
                ShowPicture TheLine       ;show it if so
                Delay 0 ,10 ,0            ;wait for a sec
        EndIf

        MoveCursor Down                   ;go to the next line
EndLoop
```

This is a slide-show program created with CanDo in the amount of time it takes you to type in the above script. The user just puts the names of the picture files to show into the icon using theWorkbench Info command and then runs your program.

SetToolTypeList

SetToolTypeList "Icon Name" , "DocumentName"

This command takes each line from the indicated Document and makes it part of the "Icon Name"'s ToolType list. See InsertToolTypeList above for a discussion of the function and merits of ToolTypes.

Icon Functions

These Functions can be used in expressions to return information relating to the Icon Commands.

DefaultTool

"String" = DefaultTool ("Icon Name")

The DefaultTool Function returns the default tool for the icon buffer. See the SetDefaultTool Command for more information about DefaultTools.

IconType

"String" = IconType ("Icon Name")

The IconType return the icon type for the "Icon Name". It returns a string indicating the Type. It returns one of the following: "Project", "Disk", "Drawer", "Tool", "NDOS", "Device", "Kickstart", and "Unknown".

ARexx Commands

ARexx is a high-level language developed by William Hawes. Applications supporting ARexx offer standardized communications using Amiga Public Message Ports. An application can create a Public Message Port for receiving ARexx messages. A Port created for this purpose is referred to as an ARexx Port.

CanDo does not internally support ARexx's interpreted language; CanDo supports ARexx communications. While the ARexx language itself can be a powerful enhancement to your Amiga, it is not required for your CanDo application to utilize ARexx communication.

Your application can send or receive ARexx messages, or do both. To receive messages, you need to create an ARexx Port using the ListenTo command. By putting this command in the Startup script of your first Card, you can be sure you are ready to receive messages in the Port. The CanDo's ARexx Objects allow you to look for specific messages received by the Port. After receiving a message, a script can use the System Variable TheMessage to examine the full text of the message.

CanDo can also send messages to other applications that support ARexx. The SpeakTo Command tells CanDo the name of the ARexx Port belonging to the application with which you want to communicate. The SendMessage Command sends a message to the application through this Port. You can optionally receive Results from the application. When sending a message to another application, you should consult its documentation regarding its ARexx Port name and valid message commands. Of course, you can design your own applications using CanDo that communicate using ARexx.

SpeakTo

SpeakTo "PortName"

This Command tells CanDo the name of the ARexx Port to send messages to. All subsequent SendMessage Commands will send messages to the specified "PortName".

SendMessage

SendMessage "MessageText" (, NORESULTS)

The SendMessage Command sends the "MessageText" to the "PortName" specified in the most previously executed SpeakTo Command. The optional keyword, NORESULTS, indicates that the application does not return a text message. If NORESULTS is not specified, the System Variable MessageReturned will contain the returned message.

The System Variable MessageErrorCode will contain an error code returned by the application. By convention, a ZERO indicates it completed the operation. A non-zero value is an application specified error code.

Example:

```
SpeakTo "Widget"
SendMessage "Do Your Job"
if MessageErrorCode < > 0
        GoToCard "Error Card"
endif
```

This example first tells CanDo to send messages to an application that has an ARexx Port with a name of "Widget". It then sends the message "Do Your Job". The System Variable will not be equal to 0 (ZERO) if the application returned an error condition. This script would cause your CanDo application to go to a Card called "Error Card".

ListenTo

ListenTo "PortName"

This command specifies the name of the "PortName" for receiving messages. This is the Public Message Port through which your ARexx Objects receive messages. If you have not previously specified a ListenTo Port, CanDo will create a Public Message Port by the specified name. If you are changing the ListenTo "PortName", the old "PortName" will no longer exist. It is considered good practice for your application to have only one "PortName" for receiving messages.

InsertMessagePortList

InsertMessagePortList

This command TYPE's the list of all Public Message Ports into the current Document (see Document Commands). Each Port name will be on its own line. Because there are Public Message Ports that do not support ARexx messages, you should not haphazardly send messages to just any port in the list. Sending a message to a non-ARexx Port can cause the Amiga to crash.

"String" = CurrentListenTo	- "PortName" of most recent ListenTo.
"String" = CurrentSpeakTo	- "PortName" of most recent SpeakTo.
< integer > = MessageErrorCode	- Error Code from previous Return message.
"String" = MessageReturned	- Most recent Return message.
"String" = TheMessage	- Last message received through ListenTo.

Object Commands

The Object Commands effect various Objects created with CanDo. Many of these commands use an "Object Name" as a parameter. The "Object Name" should coorespond to the Name specified in an *Object's Editor* (See Objects Documentation for more information).

DisableObject

DisableObject "Object Name"

The "Object Name" identifies a Button, Field, or Document Object. This command will make the Object inaccessable to the user of your application and the Object will be displayed as ghosted.

EnableObject

EnableObject "Object Name"

This command will make the Object accessable to the user of your application and will display the Object as non-ghosted. Note that simply enabling certain types of Objects is not enough to ensure that their appearance is the way you want it to be. This can be especially true for Fields and Area Buttons. In these cases, a ClearWindow Command may be appropriate to use after the EnableObject Command.

SetObjectState

SetObjectState "ObjectName" , « logical expression »

This Command can be used to deselect or select a Button, Field, or a Memo Document. When the « logical expression » is TRUE, it will select the Object. When it is FALSE it will delselect the Object. For Buttons, selection means that the highlighted form of the Button's display is shown in the window. For Fields and Memos, selection means that this Object will receive keyboard input and a cursor will appear in the Object's hit area.

FastFeedBack

FastFeedBack « logical expression »

The FastFeedBack Command enables or disables fast feedback of mouse movement for an Object's Drag Script. The Drag Script is performed while the left mouse button is pressed and moved over a selected Object. Each time the mouse is moved, the Drag Script is performed. It is possible for the mouse to move several times while the Drag Script is being performed.

When FastFeedBack is disabled, the Drag script is performed for every mouse movement, including the movements that could not be performed while a script is being performed. This has the benefit of having the script performed at each point in a path. However, this will cause the execution of a script to fall behind the current location of the mouse pointer.

When FastFeedBack is enabled, the Drag script will skip up to 20 mouse movements that occurr while a script is being performed. This allows the script execution to keep up with mouse movement. However, this means it will potentially miss some of the points along the path.

If the « logical expression » is TRUE, then fast feed back is enabled. Otherwise, it is disabled.

MoveObject

MoveObject "Object Name" , < x > , < y >

The MoveObject Command moves a visible Object in your window. You can use the MoveObject Command to move Buttons, Fields, and Documents. The "Object Name" is the Name specified in the Object's Editor. The < x > , < y > values are the horizontal and vertical coordinates where you want to move it to. NOTE: it is possible to move the object off the visible portion of your window.

SetInteger

SetInteger "Object Name" , < value >

The SetInteger Command sets the value displayed in the Integer Field indicated by the "Object Name". The "Object Name" is the Name specified in the *Field Editor Requester.* The < value > is the integer value to put into the field. The Minimum and Maximum range for this value will be controlled by the Limits as Defined in the Field Editor Requester for this field.

SetText

SetText "Object Name" , "Text"

The SetText Command sets the string displayed in the String Field indicated by the "Object Name". The "Object Name" is the Name specified in the *Field Editor Requester.* The "Text" is the string value to put into the field. No more than the Maximum Number Of Characters (as defined in the Field Editor Requester) will be copied into the field.

IntegerFrom

< integer > = IntegerFrom ("Object Name")

The IntegerFrom Function returns the integer currently displayed in an Integer Field. The "Object Name" is the Name specified in the *Field Editor Requester.* The < integer > value is guaranteed to be between the Maximum and Minimum values specified in the *Integer Field Requester.*

TextFrom

"Text" = TextFrom ("Object Name")

The TextFrom Function returns the string currently displayed in a Text Field. The "Object Name" is the Name specified in the *Field Editor Requester.*

ObjectState

« logical » = ObjectState ("Object Name")

Buffer Commands

The data CanDo uses is stored in area of memory called a Buffer. Every Picture, Brush, BrushAnim, Document, Sound, File I/O, and Icon, is kept in a Buffer. CanDo "manages" these buffers for you automatically. This automatic system is designed to relieve the novice user from concern about what can be a complicated problem. For this reason, it is not necessary for all CanDo users to concern themselves with using the Buffer Commands. However, these commands are provided for the users who may want to change the way the automatic system works.

Each Buffer has a unique Name. The Name is simply a "String" identifying the Buffer. Most of the time, the Name is the file specification from which the data was loaded. This is the case when a ShowBrush Command needs to load the image before showing it. However, the LoadBrush Command allows you to provide some other Buffer Name. (Note: most of the documentation avoids the terminology "Buffer" because it would make it seem unnecessarily complicated.)

CanDo keeps track of each Buffer's Name, Data, the file specification if the Data came from a file, the Buffer Type, whether it is currently being used and if the data has been modified. The purpose of this is to manage the memory used by a buffer.

Memory is a scarce resource that is in constant demand. The types of operations made easy by CanDo can be a nightmare of details for a programmer in another language. For example: when a picture is to be displayed, memory must be allocated from a "Pool" that is available to all programs. The Picture is loaded from a file that was created in a standard format known as IFF. The IFF data is loaded from the file and converted to a form that the Amiga Hardware can use for displaying the image. For the image to be displayed, it must be put into the Amiga's Chip memory. There is a separate Pool for Chip memory. When the picture is done being displayed, its memory is given back to the Pool.

Some of the difficulty occurs when there is not enough memory in one of the Pools. There are ways to tell the Amiga Operating System give back memory that it is not using any more. If there still is not enough memory, the program needs to look for memory it has allocated from the Pool and is not currently being used.

When CanDo can not get enough memory from a Pool to complete an operation, it looks through the Buffers and returns as much memory as it can. If the Buffer's Data is not currently being used and it has not been modified, CanDo returns the memory being used by the Buffer's Data. However, it keeps all of the other information about the Buffer.

By keeping the Buffer information and not its Data, CanDo can automatically reload the data from the file if it is ever needed again.

SetAutoLoadFlags

SetAutoLoadFlags (CHIP and ASNEEDED) or NONE

The LoadFlags Appendix describes how you can control on a file by file basis how CanDo keeps the file in memory. However, many of the files are automatically loaded without a load command. This is done with commands such as ShowBrush and PlaySound. Files are also loaded by some of CanDo's Objects such as Image Buttons and Picture Windows.

The SetAutoLoadFlags Command allows you to use the loadflags ASNEEDED and CHIP for files that are automatically loaded. When ASNEEDED is indicated, CanDo automatically returns the Buffer's Data as soon as it is not being used. The CHIP keyword indicates the file should be loaded directly into the Amiga's Chip memory. While having the Data in Chip memory can speed up some operations, it increases the chances of CanDo running completely out of memory. By Default the AutoLoadFlags are NONE.

Examples:

```
SetAutoLoadFlags ASNEEDED
SetAutoLoadFlags CHIP
SetAutoLoadFlags CHIP ASNEEDED
SetAutoLoadFlags NONE
```

Flush

Flush "Buffer Name"

The Flush Command frees all memory used by the specified "Buffer Name". This causes CanDo to forget everything about the Buffer as though it was never created. If you Flush a buffer that is currently being used, CanDo will Flush it as soon as it can.

Example:

```
LoadBrush "brushes:theimage.grab","arrow"
ShowBrush "arrow",50,50
Flush "arrow"
```

FlushAll

FlushAll

This command Flushes all Buffers. Buffers that are currently being used will be flushed as soon as they can be safely flushed.

RamScram

RamScram

This causes CanDo to free all Buffer Data that is not being used and has not been modified. Unlike Flushing a buffer, it does not cause CanDo to forget the buffer completely. This way the Data can be automatically loaded if it is needed again.

InsertChangedBufferList

InsertChangedBufferList

This command TYPE's the name of all modified buffers into the current Document (See Document Commands). Each Buffer Name will be on a separate line.

SaveAllChangedBuffers

SaveAllChangedBuffers

This causes CanDo to save all buffers that have been modified. If the Data was originally loaded from a file, it will be saved using the original file specification. Otherwise, it will use the Buffer Name as the file specification.

GetBufferInfo

GetBufferInfo "Buffer Name", Var1 { ,Var2 { ,Var3 { ,Var4 } } }

This command returns information about the specified "Buffer Name". This Command only supports the Buffer Types of Picture, Brush, BrushAnim, Sound, and Documents. The Var1 through Var4 specifies the variable names that should be set to the buffer information. Each of the valid Buffer Types sets the variables to a specialized piece of information for the specific Type. These are outlined below:

Buffer Type	Var1	Var2	Var3	Var4
Picture	Width	Height	Depth	
Brush	Width	Height	Depth	
BrushAnim	Width	Height	Depth	# of Frames
Sound	Samples	Seconds	Sample Rate	
Documents	Lines	Longest Line	Size	

Examples:

```
GetBufferInfo"Images:Man.pic",PicWidth,PicHeight,
        NbrOfBitPlanes

GetBufferInfo"Sounds:Howl.snd",Samples,Time,Rate
```

Functions

This Function can be used in expressions to return information relating to the Buffer Commands.

BufferType

"string" = BufferType("Buffer Name")

This function returns the Buffer Type for the specified "Buffer Name". The Buffer Types are: "Brush", "BrushAnim", "Document", "Icon", "Picture", "Read File", "Write File", and "Sound". If the "Buffer Name" is not found, BufferType will return a NULL ("") string.

Sytem Variables

« **logical** » = **AnyChangedBuffers** - TRUE if there are any modified Buffers.

Misc Commands

This last section contains various other commands relating to overall control of the Amiga and of your application.

Pointer

Pointer « logical expression »

This command turns on or off the mouse pointer. When the « logical expression » value is TRUE, the image is turned on. When it is False, it is turned off.

Example:

Pointer ON

SetPointer

SetPointer "Brush Name" [,< x > ,< y >]

This command allows you to use a DPaint style brush as the mouse pointer. It can be up to 16 pixels wide, and as tall as 127 pixels. If it is larger than these dimensions, only the leftmost and topmost part of the image will be used. The Optional < x > ,< y > offset specifies the "hotspot" for the pointer. Otherwise, it will assume it is located at 0 ,0.

Color Registers 17 ,18 , and 19 are used both for displaying an image and for the mouse pointer.

Examples:

SetPointer "Brushes:HandImage.br" ,5 ,5

Dos

Dos "DOS Command"

The Dos Command executes the string specified in "DOS Command" as an AmigaDOS command. This works as though you typed the command from a CLI Window.

Example:

Dos "run c:dir >Ram:Temp.txt"

SetCurrentDirectory

SetCurrentDirectory "directory path"

This Command sets your application's default directory. This command affects file specifications that do not include "device:" portions. It also sets the initial default directory for applications invoked using the Dos Command. Many applications use this default directory for identifying their environment.

Delay

Delay < mm > ,< ss > ,< jj >

The Delay Command causes a delay in the execution of the script. The values indicate the number of minutes, seconds, and jiffies to delay. The script continues to execute after the delay.

Note: while a script is being executed, no other object's scripts can be processed. This prevents any Buttons, Menus, etc... from executing a script. Because the delay cannot be interrupted, an excessively long delay will prevent you from exiting a script until the delay has elapsed.

The < mm > value indicates the number of minutes, the < ss > indicates the number of seconds, and the < jj > indicates the number of jiffies to delay. (U.S. Amigas use 1/60 of a second for jiffies, and PAL Amigas use 1/50 of a second.)

Examples:

```
PrintText 50 ,50 ,"Taking a 6 second breather."
Delay 0 ,6 ,0

PrintText 50 ,50 ,"Taking a One minute break."
Delay 1 ,0 ,0

PrintText 50 ,50 ,"Hold for a half a second."
Delay 0 ,0 ,30
```

Echo

Echo "Text" { , NOLINE }

The Echo Command prints a text message if CanDo or your application was started from a CLI. This can be useful for making CLI applications or displaying debugging information.

Example:

```
Echo "Print on my CLI Window"
Echo "Variable Count = " || Count
```

InsertDeviceList

InsertDeviceList LOGICAL PHYSICAL ASSIGNS ALL

This command TYPE's the list of Devices into the current Document. The keyword LOGICAL returns the logical names of all mounted devices. PHYSICAL returns the physical device names (i.e. DF0:,DF1:). ASSIGN returns all system assignments. ALL returns all LOGICALS, PHYSICALS, and ASSIGNMENTS.

Examples:

```
MakeDocument "System Assignments"
InsertDeviceList ASSIGNS

MakeDocument "All Devices"
InsertDeviceList ALL
```

InsertDirectoryList

InsertDirectoryList { FILESONLY or DIRECTORIESONLY }

This command TYPE's the Directory listing for the Current Directory (see SetCurrentDirectory). The optional keyword FILESONLY excludes directories from the directory listing. DIRECTORIESONLY returns only the sub-directories in the Current Directory.

Examples:

```
MakeDocument "System Directory"
SetCurrentDirectory "SYS:"
InsertDirectoryList

MakeDocument "Font List"
SetCurrentDirectory "Fonts:"
InsertDirectoryList DIRECTORYIESONLY
```

InsertStartingMessage

InsertStartingMessage

The InsertStartingMessage Command is used to get the parameters used for starting an application. If it was started from Workbench using an Icon, it will TYPE the full pathnames of all other icons that were selected at the time your application was started into the current Document. (A user of your program could select five icons and then start your program. With this command, you would get the names of the five files referred to by those five icons.) If your application was run from CLI, this command TYPEs each of the parameters on the command line. Any parameters enclosed in double-quotes will be treated as single parameters. They will have the double-quotes removed and internal any doubled double-quotes (" ") will be reduced to single double-quotes.

NOTE: When developing your application from CanDo, this command will insert the starting message for CanDo. After all, CanDo started your program after itself having been started. When running your applications separately, however, InsertStartingMessage will insert the starting message to your application.

« **logical** » = StartedFromWorkbench - TRUE when started from Workbench

"string" = TheCommandLine - Returns the command line if started from CLI, otherwise a NULL string

"string" = TheCurrentDirectory - Returns your application's current directory

"string" = TheOriginDirectory - This is the directory your program started running from.

7 Appendices Index

Commands Index

Page		Command
6 - 52	" String " =	**DocumentName**
6 - 66		**Dos "DOS Command"**
6 - 29		**DrawCircle < x >,< y >,< r >**
6 -2 9		**DrawEllipse < x >,< y >,< xr >,< yr >**
6 - 29		**DrawLine < x1 >,< y1 >,< x2 >,< y2 >**
6 - 29		**DrawPixel < x >,< y >**
6 - 30		**DrawRectangle < x >,< y >,< w >,< h >**
6 - 30		**DrawTo < x >,< y >**
6 - 13	" String " =	**DupeString (" String " , < count >)**
6 - 67		**Echo "Text" { , NOLINE }**
6 - 60		**EnableObject "Object Name"**
6 - 16	results =	**EvaluateExpression (" String ")**
6 - 20		**ExitLoop**
6 - 22		**ExitScript**
6 - 17	« Logical » =	**FALSE**
6 - 60		**FastFeedBack « Logical Expression »**
6 - 54		**FileReadChars "Buffer Name", VariableName, < NumberOfChars >**
6 - 53		**FileReadLine "Buffer Name",VariableName**
6 - 54		**FileWriteChars "Buffer Name" , " String " { , < length > }**
6 - 53		**FileWriteLine "Buffer Name","String"**
6 - 29		**FillToBorder < x >,< y >,< BorderColor >**
6 - 14	< Integer > =	**FindChars ("Source" , "Search" , < starting offset >)**
6 - 15	< Integer > =	**FindWord("Source","Search Word"{ ,< StartWordNumber > { ,"WordDelimiters"} })**
6 - 23		**FirstCard**
6 - 28		**FloodFill < x >,< y >**
6 - 63		**Flush "Buffer Name"**
6 - 63		**FlushAll**
6 - 41	< Integer > =	**FrameOfAnimation ("BrushAnim Name")**
6 - 39		**GetBrushAnimCoordinates "BrushAnim Name", Xvariable,Yvariable**
6 - 64		**GetBufferInfo "Buffer Name", Var1 { ,Var2 { ,Var3 { ,Var4} } }**
6 - 15	" String " =	**GetChars ("Source" , < starting offset > , < length >)**
6 - 31		**GetRGB < col.reg >, RedVar, GreenVar, BlueVar**
6 - 34		**GetTextDimensions "Text",Width-Variable, Height-Variable**
6 - 15	" String " =	**GetWord ("Source", < WordNumber > { ,"WordDelimiters" })**
6 - 23		**GotoCard "cardname"**
6 - 37	« Logical » =	**Hires**
6 - 57	" String " =	**IconType("Icon Name")**
6 - 18		**If « Logical Expression » ... Else ... EndIf**
6 - 64		**InsertChangedBufferList**
6 - 14	" String " =	**InsertChars ("source" , "destination" , < offset >)**
6 - 67		**InsertDeviceList LOGICAL PHYSICAL ASSIGNS ALL**

Page		Command
6 - 68		**InsertDirectoryList { FILESONLY or DIRECTORIESONLY }**
6 - 51		**InsertDocument "Document Name" { , < Start Line > { , < LineCount > } }**
6 - 59		**InsertMessagePortList**
6 - 68		**InsertStartingMessage**
6 - 57		**InsertToolTypeList "Icon Name"**
6 - 10	< Integer > =	**Integer (Expression)**
6 - 61	< Integer > =	**IntegerFrom ("Object Name")**
6 - 37	« Logical » =	**Interlace**
6 - 17	< Integer > =	**LargestChunkOfMemory**
6 - 23		**LastCard**
6 - 52	< Integer > =	**LengthOfLine**
6 - 5		**Let VariableName = Expression**
6 - 11	< Integer > =	**Limit (< limit1 >, < limit2 >, < test value >)**
6 - 52	< Integer > =	**LinesInDocument**
6 - 59		**ListenTo "PortName"**
6 - 24		**LoadBrush "filename" (, "Name" (, loadflags))**
6 - 38		**LoadBrushAnim "filename" { ,"BrushAnim Name" { ,loadflags } }**
6 - 46		**LoadDocument "filename" { , "Document Name" }**
6 - 55		**LoadIcon "filename" { ,"Name" { ,< load flags > } }**
6 - 24		**LoadPicture "filename" (, "Name" (, loadflags))**
6 - 42		**LoadSound "filename" { ,"Sound Name"}**
6 - 10	« Logical » =	**Logical (Expression)**
6 - 19		**Loop ... Until « Logical Expression »**
6 - 20		**Loop ... EndLoop**
6 - 13	" String " =	**LowerCase (" String ")**
6 - 45		**MakeDocument "Document Name"**
6 - 56		**MakeIcon "Icon Name",PROJECT or TOOL,"ImageName" { ,"AltImageName" }**
6 - 11	< Integer > =	**Max (< val > , < val > , ...)**
6 - 17	< Integer > =	**MaxInteger**
6 - 59	< Integer > =	**MessageErrorCode**
6 - 59	" String " =	**MessageReturned**
6 - 11	< Integer > =	**Min (< val > , < val > , ...)**
6 - 17	< Integer > =	**MinInteger**
6 - 37	< Integer > =	**MouseX**
6 - 37	< Integer > =	**MouseY**
6 - 39		**MoveBrushAnim "BrushAnim Name",< Xvel >,< Yvel >, { < Xacc >,< Yacc >, { < ticks > } }**
6 - 38		**MoveBrushAnimTo "BrushAnim Name", < x >,< y > { ,ticks }**
6 - 47		**MoveCursor UP DOWN LEFT or RIGHT { ,< Count > }**
6 - 48		**MoveCursorTo Location Area** **Location: STARTOF or ENDOF** **Area: DOCUMENT, LINE, NEXTWORD, PREVIOUSWORD, or THISWORD**

Page		Command
6 - 61		**MoveObject “Object Name”,< x >,< y >**
6 - 30		**MovePen < x >,< y >**
6 - 35		**MoveScreen < Delta-X >,< Delta-Y >**
6 - 36		**MoveWindow < Delta-X >,< Delta-Y >**
6 - 47		**NewLine**
6 - 23		**NextCard**
6 - 17	« Logical » =	**NO**
6 - 37	« Logical » =	**NTSC**
6 - 13	< Integer > =	**NumberOfChars (“ String ”)**
6 - 17	“ String ” =	**ObjectName**
6 - 61	« Logical » =	**ObjectState (“Object Name”)**
6 - 17	« Logical » =	**OFF**
6 - 17	« Logical » =	**ON**
6 - 53		**OpenFile “filename”,“Buffer Name”, IOFlags, AccessFlags** **IOFlags: READONLY or WRITEONLY.** **AccessFlags: NEWFILE, OLDFILE or APPEND.**
6 - 34	< Integer > =	**PenA**
6 - 34	< Integer > =	**PenB**
6 - 34	< Integer > =	**PenO**
6 - 42		**PlaySound “Sound Name” { ,AudioFlags ,< period > }**
6 -4 2		**PlaySoundSequence “Document Name” { ,AudioFlags ,< period > }**
6 - 66		**Pointer « Logical Expression »**
6 - 16	< Integer > =	**PositionOfWord (“Source”, < WordNumber >, { ,“WordDelimiters” })**
6 - 48		**PositionOnLine < line >**
6 - 23		**PreviousCard**
6 - 33		**PrintText < x >, < y >, “ String ”**
6 - 22		**Quit**
6 - 63		**RamScram**
6 - 12	< Integer > =	**Random (< Minimum >, < maximum >)**
6 - 30		**RayTo < x >,< y >**
6 - 39		**RemoveBrushAnim “BrushAnim Name”**
6 - 14	“ String ” =	**RemoveChars (“source” , < starting offset > , < length >)**
6 - 51		**Replace “fromtext”,“totext” { ,[GLOBAL or ONCE] BYWORD NOCASE }**
6 - 64		**SaveAllChangedBuffers**
6 - 27		**SaveBrush “Brush Name” (, “filename”)**
6 - 46		**SaveDocument “Document Name” { ,“filename” }**
6 - 55		**SaveIcon “Icon Name” { ,“FileName”}**
6 - 27		**SavePicture “Picture Name” (, “filename”)**
6 - 37	< Integer > =	**ScreenColors**

Page		Command
6 - 37	< Integer > =	**ScreenHeight**
6 - 35		**ScreenTitleBar « Logical Expression »**
6 - 35		**ScreenTo FRONT or BACK**
6 - 37	< Integer > =	**ScreenWidth**
6 - 37	< Integer > =	**ScreenX**
6 - 37	< Integer > =	**ScreenY**
6 - 50		**SearchFor "text" { , BYWORD and NOCASE }**
6 - 59		**SendMessage "MessageText" { , NORESULTS }**
6 - 31		**SetAreaDrawMode NORMAL or OUTLINE**
6 - 63		**SetAutoLoadFlags [CHIP and ASNEEDED] or NONE**
6 - 40		**SetBrushAnimFlags "BrushAnim Name",< brushanimflags > { ,{ ticks } }** **COMPRESSEDMODE or DECOMPRESSEDMODE** **RESTOREBACKGROUND or LEAVEIMAGE** **USEMASK or NOMASK** **SEQUENCEDMOTION or LINEARMOTION** **FORWARD, BACKWARD and PINGPONG** **NONE**
6 - 44		**SetChannel < channel >**
6 - 26		**SetClipTransparentColor < color >**
6 - 66		**SetCurrentDirectory "directory path"**
6 - 56		**SetDefaultTool "Icon Name","DefaultTool"**
6 - 31		**SetDrawMode Normal JAM1 JAM2 COMPLEMENT INVERSEVIDEO**
6 - 54		**SetFileBufferSize < sizeinkilos >**
6 - 56		**SetIconImage "Icon Name","ImageName" { ,"AltImageName" }**
6 - 61		**SetInteger "Object Name",< value >**
6 - 60		**SetObjectState "ObjectName", « Logical Expression »**
6 - 32		**SetPen < pena > { ,< penb > { ,< penO > } }**
6 - 66		**SetPointer "Brush Name" { ,< x >,< y > }**
6 - 33		**SetPrintFont "fontname", < pointsize >**
6 - 33		**SetPrintStyle StandardFlags { ExtendedFlags { , < ExtPen1 > { , < ExtPen2 > } } }** **StandardFlags: PLAIN ITALIC BOLD and UNDERLINED.** **ExtendedFlags: SHADOW OUTLINE GHOSTED or EMBOSSED.**
6 - 32		**SetRGB < col.reg >, < red >, < green >, < blue >**
6 - 35		**SetScreenTitle "Text"**
6 - 61		**SetText "Object Name","Text"**
6 - 57		**SetToolTypeList "Icon Name","DocumentName"**
6 - 44		**SetVolume <volume> { ,< channel > }**
6 - 36		**SetWindowLimits < MinX >,< MinY >,< MaxX >,< MaxY >**
6 - 36		**SetWindowTitle "Text"**
6 - 52		**SetWordDelimiters "delimiterlist"**
6 - 25		**ShowBrush "Brush Name",< x >,< y > { , BRUSHPALETTE }**

Page		Command
6 - 38		**ShowBrushAnim "BrushAnim Name", < x >,< y >**
6 - 27		**ShowPalette "file specification"**
6 - 25		**ShowPicture "Picture Name"**
6 - 12	< Integer > =	**Sign (< value >)**
6 - 52	< Integer > =	**SizeOfDocument**
6 - 58		**SpeakTo "PortName"**
6 - 47		**SplitLine { < count > }**
6 - 68	« Logical » =	**StartedFromWorkbench**
6 - 22		**StopScript**
6 - 10	" String " =	**String (Expression)**
6 - 17	« Logical » =	**Supervised**
6 - 61	" String " =	**TextFrom ("Object Name")**
6 - 52	" String " =	**TheCharacter**
6 - 52	< Integer > =	**TheColumnNumber**
6 - 68	" String " =	**TheCommandLine**
6 - 68	" String " =	**TheCurrentDirectory**
6 - 17	" String " =	**TheDate**
6-52	" String " =	**TheLine**
6-52	< Integer > =	**TheLineNumber**
6-59	" String " =	**TheMessage**
6-68	" String " =	**TheOriginDirectory**
6-17	" String " =	**TheTime**
6-52	" String " =	**TheWord**
6-52	" String " =	**TheWordDelimiters**
6-26		**Transparent « Logical Expression »**
6-34	« Logical » =	**TransparentStatus**
6-13	" String " =	**TrimString (" String ")**
6-17	« Logical » =	**TRUE**
6-46		**Type " String " { ,NewLine }**
6-13	" String " =	**UpperCase (" String ")**
6-17	« Logical » =	**VerifyExpression (" String ")**
6-20		**While « Logical Expression » ... Until « Logical Expression »**
6-19		**While « Logical Expression » ... EndLoop**
6-37	< Integer > =	**WindowColors**
6-37	< Integer > =	**WindowHeight**
6-37	" String " =	**WindowTitle**
6-36		**WindowTo FRONT or BACK**
6-37	< Integer > =	**WindowWidth**
6-37	< Integer > =	**WindowX**
6-37	< Integer > =	**WindowY**
6-46		**WorkWithDocument "Document Name"**
6-17	« Logical » =	**YES**

LoadFlags Appendix

LoadFlags

It is not necessary to use or understand the LoadFlags. However, they give you some options on how data is loaded and kept in memory. The LoadFlags are KEYWORDS that can be added to the end of any load Command. These Commands are: LoadPicture, LoadBrush, LoadSound, LoadBrushAnim, and LoadDocument.

The LoadFlags are: OVERWRITE CHIP (DELAYLOAD or ASNEEDED).

You do not have to specify any of the loadflags or you can use one or more of them. If you use more than one, do not separate them with commas.

OVERWRITE

OVERWRITE causes the file to be re-loaded even if it has already been loaded into memory. Normally, a load Command will not re-load the file if it is currently in memory.

CHIP

CHIP causes the file to be loaded into the Amiga's CHIP memory. Loading a file into CHIP memory causes an image to be displayed slightly faster. However, CHIP memory is a valuable resource. The CHIP loadflag is only relevant when used with the LoadPicture and LoadBrush Commands.

DELAYLOAD or **ASNEEDED**

DELAYLOAD or ASNEEDED are used to alter when the file is loaded. By Default, a file is loaded if it is not already in memory. You can specify either DELAYLOAD or ASNEEDED, but not both.

DELAYLOAD

DELAYLOAD indicates not to load the file until a Show or Play Command is used. Also, using the DELAYLOAD fla allows you to specify a "Name" other than the "filename" and still have it automatically loaded. When a Show Command uses "Name", it will load the "filename" specified in the Load Command.

ASNEEDED

ASNEEDED does the same thing as DELAYLOAD except it automatically flushes the loaded file from memory when it is not being used. Normally, the file remains in memory until it is flushed with a Flush Command or a memory panic occurs. A memory panic occurs when there is not enough memory to complete an operation. When this happens, CanDo automatically flushes ALL data not being used. Even so, running out of memory is a dangerous condition. ASNEEDED is very useful when there is not a lot of memory and a delay is acceptable each time the file is used.

Examples:

LoadPicture "Images:Background.pic", "Background" ,DELAYLOAD

LoadPicture "Images:Background.pic", "Background" ,CHIP

LoadPicture "Images:Background.pic", "Background" ,ASNEEDED CHIP

LoadPicture "Images:Background.pic", "Background" ,OVERWRITE

Advanced Features

CanDo has many features that you may customize to suit your tastes. These may be changed by editing the Tool Types in CanDo's icon. Do this by selecting the CanDo icon and then selecting Info in the WorkBench menu. Edit each Tool Type and press Return. An alternate method of setting CanDo's defaults is to create a text file named CanDo.defaults and save it to your S: directory. This file should contain the same text as was contained in the Tool Types, one Tool Type per line. Here are the Tool Types and an explanation of each. If the Tool Type is not defined, it will default to the value shown in brackets ([ ]).

EDITORTOOLS = "Path" ["EditorTools"]

The name of the directory where the EditorTool files reside.

XTRAS = "Path" ["XtraTools"]

The name of the directory where the Xtra Object files reside.

OBJECTS = "Path" ["ObjectTools"]

The name of the directory where the Object files reside.

HELPFILES = "Path" ["CanDoExtras:HelpFiles"]

The name of the directory where the Help files reside.

SOUNDS = "Path" ["CanDoExtras:Sounds"]

This is the default directory in which CanDo looks for your sounds.

IMAGES = "Path" ["CanDoExtras:Images"]

This is the default directory in which CanDo looks for your images.

BRUSHES = "Path" ["CanDoExtras:Brushes"]

The default directory in which CanDo looks for your brushes or clipped graphics.

BRUSHANIMS = "Path" ["CanDoExtras:BrushAnims"]

The default directory in which CanDo looks for your DeluxePaint III BrushAnims.

DOCUMENTS = "Path" ["CanDoExtras:Documents"]

The default directory in which CanDo looks for your text documents.

DECKS = "Path" ["CanDoExtras:Decks"]

The default directory for saving and loading your decks.

DEFAULTDECK = "PathFile"

You may set the Deck to be loaded when CanDo is started or when you select New from CanDo's Deck Menu.

SOUNDEFFECTS = On or Off [On]

This turns all CanDo's sound effects On or Off.

SOUNDVOLUME = 0 to 64 [64]

This is the volume setting for CanDo's sound effects. It can range from 0 (quiet) to 64 (loud).

SCROLLSPEED = 1 to 10 [5]

This controls how fast CanDo's screen scrolls up and down. It can be a value of 1 to 10 where 1 is the fastest.

INHERITWINDOW = On or Off [On]

When CanDo makes a new card it inherits the current card's window (On) or it uses the default window (Off).

COORDINATEDRAG = On or Off [Off]

When making a box (button,field,etc...) to define an object, this flag controls the method of interaction:

On = Position, Click, Drag, Release
Off= Position, Click, Release, Move, Click, Release

CANDOFONT = "fontname" ["topaz"]

This defines the font to be used by CanDo. This font must have an eight point non-proportional render.

AUTOREQUESTERS = On or Off [On]

This can be used to turn off (skip showing) intermediate requesters.

CRASHFILE = "PathFile" ["CanDoExtras:Decks/CanDo.CrashDeck"]

If CanDo crashes, it can store the current Deck to a file. This must be a valid path and filename. If the name is a Null ("") then CanDo does not store the current Deck.

ICONS = On or Off [On]

This informs CanDo whether or not to create an icon for Decks that are saved.

ICONDEFAULTTOOL = "Default Tool" ["C:CanDoRunner"]

When CanDo creates an icon for a Deck, this is the default tool for the icon. Normally, this should be CanDo.runtime, which is the small run time module that uses the CanDo.library in your LIBS: directory. To make a distributable application be sure to use the CanDoBinder.

Remember to look on your CanDo and CanDoExtras disk for the ReadMe files. These files contain last minute information that was not included in this manual.

There are many subtle, and not so subtle, methods of easily doing seemingly complex operations in CanDo's scripting language. Please look through the example Decks that are included on your CanDoExtras disk. These Decks have many different scripts that are good examples of scripting. The best way of learning the ins and outs is to experiment and try new things.

Error Messages

Syntax Errors

Unknown Command

The first word on the line is not a valid Command. You probably misspelled the Command or you forgot to type LET for an assignment.

Too many parameters

The Command line has too many parameters. It's also possible you provided a parameter for a Command that does not use any.

Not enough parameters

The Command needs more parameters than you provided.

Unrecognized KEYWORD

You provided an unrecognized word where a KEYWORD was expected. It is possible you tried to use a variable name in its place.

Conflicting switches specified

You indicated two or more KEYWORDS that cannot be used together.

Too many IFs

You have too many IF/ENDIF combinations in your script.

IF without matching ENDIF

You do not have an ENDIF for every IF Command in your script.

Unexpected ELSE or ENDIF

You don't have an IF before every ELSE or ENDIF Command.

LOOP/WHILE without matching ENDLOOP/UNTIL

Your script does not have an ENDLOOP or UNTIL for every LOOP or WHILE Command in your script.

Unexpected EXITLOOP/ENDLOOP/UNTIL

You don't have a LOOP or WHILE before every EXITLOOP, ENDLOOP, or UNTIL Command.

Too many LOOPs

You have too many LOOP/WHILE ... ENDLOOP/UNTIL combinations in your script.

Misplaced operator

An expression contains a misplaced operator. An operator was found in a place where a variable or constant was expected.

Mismatched parenthesis

An expression does not have matching parenthesis.

Expression is too complicated

An expression has too many parenthesis and operations. Simplify the expression by removing a portion of it and assigning it to a variable. You can then use the variable in place of this portion.

Invalid expression

CanDo could not figure out an expression.

Error Messages

Run Time Errors

Named routine not found

A Do Command attempted to run a routine that does not exist.

Card not found

A GotoCard Command attempted to go to a card that does not exist.

Named Object not found

An Object Command referenced an Object that does not exist.

Named Object is wrong type

The referenced Object can not be used in this context.

Addressed port not found

The ARexx port specified in an ListenTo Command does not exist.

Named buffer not found

The "Buffer Name" specified in a Buffer Command does not exist.

Named buffer is wrong type

The "Buffer Name" can not be used in this context.

Stack overflow

You have too many nested Do Commands. This probably is because a Routine is calling itself indefinitely.

Command not allowed in current mode

This Command cannot be used from within CanDo but it can be used when your project is running by itself
The System Variable SUPERVISED is true when your project is running from CanDo.

No Document selected

You attempted to use a Document Command without first making a Document.

Name evaluated to a NULL string

A string evaluated to a NULL ("") when a valid name was required.

Division by ZERO

An expression contained a division by ZERO (0).

Invalid variable name

The Command contained an invalid variable name, or an attempt to set the value of a System Variable or Function.
The Command required a variable name.

Run Time Errors

Screen open error

Not enough chip memory was available in your Amiga to open a screen the size that is required by your card's window.

Window open error

Not enough chip memory was available in your Amiga to open your card's window.

Unknown IFF FORM type

The IFF file referenced is not of a type CanDo knows how to deal with. Alternately, it might be a corrupt IFF file.

Brush file has no mask stored with it

The brush file was saved in a format that does not have a mask, and one cannot be computed.
The mask is used for Image buttons and ShowBrush with Transparent enabled.

Premature EOF

A file referenced in your script, or used by one of your Objects, was not complete.

Not enough memory

Not enough memory was available in your Amiga to perform the current operation. CanDo will first try to release any memory is not using at the moment before resorting to this error.

Object in use

A file (or device) that your script referenced is in use by another program in your Amiga. Your program must wait until that DOS object is free.

Directory not found

Your script tried to run a SetCurrentDirectory Command into a directory that could not be found. Check your script for accuracy.

Object not found

A file (or device) could not be located using the name as it appeared in your script. Check the filename for accuracy and correct it.

Wrong file type

A file that you tried to load (probably as an IFF file) is not of the correct type.

Disk not validated

Your script tried to write to a disk or volume that has not been validated by AmigaDOS.

Disk write-protected

Your script tried to write to a disk that has been write-protected.

Device not mounted

A file was referenced on a volume that could not be found.

Disk full

The disk that your script was writing to has become full. Often, this can be corrected without interrupting your script by deleting unneeded files from the full disk.

File delete-protected

Your script attempted to save a buffer on top of a file already on the disk that has been protected from deletion.

File write-protected

Your script opened a WriteFile buffer using a file that is write-protected.

Not DOS disk

A disk in your system, referenced by your script, is not a standard AmigaDOS disk. Disks of this sort cannot be used with CanDo.

Doit

INOVAtronics

CanDo is a product of INOVAtronics, Inc.
8499 Greenville Avenue
Suite 209B
Dallas, Texas 75231
214-430-4991