OctaMED

Ed Wiles

COLLABORATORS			
	TITLE : OctaMED		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Ed Wiles	August 7, 2022	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

OctaMED

1.1 Using ARexx with OctaMED / ARexx Tutorial

```
                SECTIONS

                 Introduction

                 An introduction to OctaMED commands

                 Simple commands and command groups

                 Parameters and command templates

                 Our first ARexx script

                 Entering a script

                 Executing a script

                 ARexx features

                 DO loops

                 IF...THEN statements

                 Writing a script
                OVERVIEW
Not strictly in the general format of the Features Guide, this
tutorial is taken from the OctaMED V6 manual. It provides a general
introduction to using OctaMED ARexx commands and writing ARexx
scripts for use with OctaMED.
```

1.2 ARexx Tutorial

```
             INTRODUCTION

ARexx is one of these 'buzzwords' used by so many Amiga people, but many
users seem to be in the dark as to what it actually is. Well, in relation
```

to OctaMED, it is both a way of adding new features to the program, and a time-saving alternative to mundane tasks.

You use ARexx by typing in an ARexx script, a set of commands (instructions) that ARexx is to carry out. If ARexx were to control a toaster, some of these commands could be 'start toasting', 'cancel toasting', 'set timer', 'switch on "frozen bread" indicator' and so on. But we'll be using ARexx to control OctaMED, so commands are more likely to be 'play song', 'open the Font window', or 'mark a range over track 3'.

Well, it would be nice if commands were so informal, but computers find informal English very difficult to understand! So commands are actually a lot more precise. For example, the instruction 'set instrument 04's default volume to 32' would in fact be performed by the two commands IN_SELECT 4 and IN_SETVOLUME 32.

Understanding how to use OctaMED's commands is vital for writing ARexx scripts. Let's look at them now: It's time for a tutorial! (Admittedly it's a very quick tour but hopefully you'll pick up the basics).

AN INTRODUCTION TO OCTAMED COMMANDS

SIMPLE COMMANDS AND COMMAND GROUPS

Close all windows but the main three (Main Control, Tracker editor, Information). Now select Project menu -> Command Shell. The window that appears is used for entering OctaMED commands.

If you enter a command that OctaMED doesn't recognize, or you make a typing mistake, you'll most likely get error code 25. Try this:

1) Type: fix the car (and press Return)

Not in OctaMED's vocabulary. Pity really. But here's one that is:

2) Type: ve_octamed (and press Return: do so after each command you type)

OctaMED displays the command result, the version of OctaMED you're using. All OctaMED commands consist of a two-letter group name (VE here), an underscore (_), and the command itself (OCTAMED here). Group names reveal a lot about a command's purpose; for instance, we know immediately that VE_OCTAMED is a version command. Other group names are as follows:

Group name	Purpose	Group name	Purpose
ED	Edit	SA	Sample editor
IN	Instrument	SG	Song
OP	Options	SY	Synth editor
PL	Playing	WI	Window
RN	Range		

(By the way, this manual will tend to show commands in capitals. Do feel free to enter them in lower-case, though, it looks a bit nicer!)

PARAMETERS AND COMMAND TEMPLATES

Armed with the knowledge that WI commands apply to windows, perhaps you can guess what the WI_OPEN command does.

3) Type: wi_open songoptions

Yes, the Song Options window opens. The WI_OPEN command can't be used on its own; it needs a parameter (in this case, SONGOPTIONS). Parameters are extra pieces of information that many commands need to function properly. There can be more than one of them, and they're all separated by spaces.

WI_OPEN's 'template' is WINDOW/A. This template describes WI_OPEN's parameter:

a) WINDOW is its keyword; b) /A is its specifier

Specifier /A means that the WINDOW parameter is necessary; without it, WI_OPEN won't work properly.

4) Close the Song Options window, then type: wi_open

Error code 10, because we didn't provide a window name. If you like, you can type the parameter's keyword (WINDOW in this case) before the parameter. This can serve as a reminder of what the parameter actually signifies.

5) Type: wi_open window tempo

The Tempo window opens. We'll examine a new command now, SG_LOAD, and use it to load our demo song. Firstly we'll have a look at its template; and there's a quicker way than finding it in the manual...

6) Click inside the OctaMED Command Shell window, then type: sg_load ?

A ? beside any command returns its template. Examine the template:

a) NAME has no specifier: the parameter is a string. (A string is a series of any letters, numbers or symbols).

b) FORCE has specifier /S: the parameter is a switch. Type the keyword to switch FORCE on; leave the keyword out to switch FORCE off.

7) Insert a disk containing any song in any drive, and find out the full path and filename of one of the songs. Then type:
sg_load name <path and filename of song> (press Return as usual)

If you managed all that, your song should load. Let's examine what we've just typed. The path and filename is the NAME parameter, the name of the song. We could have left the NAME keyword out, as we've seen, but we left it in to make things clear.

Including the keyword FORCE switches the 'force loading' option on. This loads the song irrespective of whether or not the current song has been modified in any way. Omitting the FORCE keyword would switch the 'force loading' option off.

So we've seen parameters with no specifier, with specifier /A, and with

specifier /S. Let's look at the SG_SETTEMPO command (sets the song's tempo).

8) Type: sg_settempo ?

The /N specifier means that the parameter is a number. There are three keywords (BPM=SPD, TPL and LPB), each of which changes a different tempo setting. In fact, BPM=SPD is not a single keyword: it means you can use either BPM or SPD as a keyword for that parameter.

9) Type: sg_settempo spd 70

In the Tempo window, notice that the Tempo slider is now set to 70. (After entering these tempo-changing examples, feel free to use the commands PL_PLAYSONG and PL_STOP to try out the new playing speeds. No, don't reach for that mouse, you're not allowed to!)

10) Type: sg_settempo bpm 57

Again, the Tempo slider changes (to 57 now). So the two keywords SPD and BPM are interchangeable for that parameter. Now let's try the TPL slider.

11) Type: sg_settempo tpl 12

The slider becomes 12. You can also change both Tempo and TPL sliders at the same time.

12) Type: sg_settempo tpl 8 spd 25

So far, we've been preceding every parameter with its keyword. A while ago, I mentioned that this is not strictly necessary but it makes things clear. Well, using keywords is in fact necessary if a command's parameters are entered in a different order from that given by its template.

For example, the template for SG_SETTEMPO is BPM=SPD/N,TPL/N,LPB/N. So entering the command SG_SETTEMPO 25 10 4 (without keywords) sets the Tempo slider to 25, the TPL slider to 10, and LPB to 4: the order given by the template. If the desired effect, however, was to set LPB to 25, Tempo to 10, and TPL to 4, then you'd need to use keywords.

This is also true for changing a single slider. SG_SETTEMPO 17 (without keyword) will always set the Tempo slider to 17, as this is first in the template. If you actually wanted to set the TPL slider to 17, you'd have to use SG_SETTEMPO TPL 17 instead.

Perhaps entering these examples will clarify a bit. Notice the changes in slider values after entering each command.

13) Type: sg_settempo 25 10 4
    sg_settempo lpb 25 bpm 10 tpl 4
    sg_settempo 17
    sg_settempo tpl 17

Another occasion where you must use the keyword is if specifier /K is given. ED_GOTO's parameters use this specifier.

14) Close the Tempo window, then type: ed_goto ?

```
ed_goto block 4 track 2
```

Click on the Tracker editor's window depth gadget (far top right), and confirm that we're now in track 2 of block 4. Click the depth gadget again.

Did you understand ED_GOTO's template? All of its parameters have specifiers /K and /N, so they're all numbers and they all require their keyword.

The last specifier we'll examine is /F.

15) Type: wi_showstring ?
    wi_showstring message I'm sorry, I can't fix cars.

Now close the OctaMED Command Shell window, and look at the title bar. /F means that all information typed after the keyword will count as that one parameter. So if the command has other parameters (unlike WI_SHOWSTRING), make sure they come before the one with /F.

Also, parameters containing spaces are usually not allowed, but they are allowed with /F parameters. (Other parameters need to be surrounded by double quotes. For example, "This parameter contains spaces").

There is one other rarely-used specifier: /M (multiple parameters). It means you can enter the same parameter over and over again: different notes of a chord, for example. OP_MULTICMD uses /M.

OK, now that we have the basics of OctaMED commands under our belt, let's see how to string them all together by creating an ARexx script.

OUR FIRST AREXX SCRIPT

ENTERING A SCRIPT (using the Ed text editor)

Our first script will perform the simple (and admittedly not very useful) task of swapping track 0 with track 2.

You can't use OctaMED to enter an ARexx script: you need a text editor. Any text editor will do, and if you have a favourite by all means use it. I'll describe how to use Commodore's Ed text editor.

16) Switch to the Workbench screen using the screen depth gadget (far right of title bar), or if you're a keyboard fanatic, press Left Amiga-N

17) Click anywhere on the Workbench screen, then select Project menu -> Execute Command

18) In the text box that appears, type the following exactly:
    ed rexx:swaptracks.omed (and press Return)

If all has gone well, you'll be presented with an empty Ed window.

19) Now please type in the following script as accurately as you can, pressing Return at the end of every line. (' is on the key above Tab)

```
    /* SwapTracks.omed: Swaps tracks 0 and 2 */
```

```
    address octamed_rexx
    options results

    'ed_goto track 0'
    'rn_copy track'

    'ed_goto track 2'
    'rn_swap track'

    'ed_goto track 0'
    'rn_paste track'

    'wi_showstring Tracks 0 and 2 swapped.'
```

20) When you've finished, select Project menu -> Save

OK. Let's have a closer look at this script. The first three lines are very important:

i) /* ... */ A comment. You can type any text between the two symbols: a brief description of the script is a sensible move.

ii) address octamed_rexx Tells ARexx that it will be talking to OctaMED. OCTAMED_REXX is OctaMED's ARexx port.

iii) options results Necessary if OctaMED is to give ARexx information. Not really needed for this particular script, but best form habits.

These three lines, with perhaps a blank line in between lines 1 and 2 for neatness, must be at the start of every ARexx script you write for OctaMED. Yes, you can't even leave the comment out!

Now we progress to the main script. OctaMED is to move to track 0, select Track menu -> Copy, move to track 2, select Track menu -> Swap w/Buff, move to track 0, and finally select Track menu -> Paste. After all this, track 0 will have been swapped with track 2. Trust me!

Notice that all OctaMED commands are enclosed by single quotes ('). Make a habit of this, it's recommended for all commands that control programs. The quotes are actually necessary for some commands, and it also distinguishes OctaMED commands from other commands. Just use them, OK?

The script is rounded off nicely with a confirmation message that the tracks have been successfully swapped. Users like such messages.

21) Quit Ed by clicking its close gadget, then return to OctaMED's screen using the Workbench screen's depth gadget

EXECUTING A SCRIPT (using the Keyboard Shortcuts window)

OK. From within OctaMED, a good flexible way of executing (running) a script is by setting a keyboard shortcut for it. Our SwapTracks script will be executed by holding Shift and pressing F1.

22) Select Settings menu -> Keyboard Shortcuts. In the left-hand section,

click Ins. New. Now click inside the Name text box, delete <unnamed>, and type Shift-F1 (Swap tracks) (and press Return)

23) In the middle (Input) section, click once on the Shift cycle gadget (it should now show Either). In the Raw box, change the 0 to 80.

24) In the right-hand (Action) section, click once on the cycle gadget (it should now show Execute ARexx File). In the Command box, type swaptracks (and press Return)

25) Save the shortcuts using Save (click OK in the requester), then close the window

26) Watching the Tracker editor carefully, press Shift-F1

Notice tracks 0 and 2 swapping? With a nice message afterwards? Very clever. If you're a perfectionist, you can speed things up a bit by adding a couple of commands to the script. Do steps 16 to 18 described earlier, then just after the options results line, type 'op_update off'. Then just before the wi_showstring line, type 'op_update on'. Save the script and quit Ed. Back in OctaMED, press Shift-F1 again and you should notice a difference.

You may have noticed that in step 24, we gave the script's filename as simply swaptracks (the full name is REXX:swaptracks.omed). This is possible as long as the file is in the REXX: directory and has the extension .omed.

AREXX FEATURES

We've seen how ARexx can be used to perform a series of OctaMED commands with a single keyboard shortcut, thus saving time. But ARexx can be far more powerful and time-saving than that, as I'll hint at in the next sections.

DO LOOPS (How to repeat commands)

One time-saving ARexx feature is its ability to repeat the same set of commands as many times as you like. This device is called a DO loop.

We'll now enter another script, which will change every second note in track 3 to D-3.

27) Switch to the Workbench screen again, and select Project menu -> Execute Command. In the text box, type ed rexx:track3d3.omed. Now enter the following script:

```
/* Track3D3.omed: Changes every 2nd note in track 3 to D-3 */

address octamed_rexx
options results

'ed_goto track 3'

do currline = 0 to 63 by 2
   'ed_goto line' currline
```

```
      'ed_setdata note 27'
    end

    'ed_goto line 0'
    'wi_showstring Notes changed.'
```

28) Save and quit in one go by selecting Project menu -> Save & Exit

29) Back in OctaMED, open the Keyboard Shortcuts window. Click Ins. New, and replace <unnamed> with Shift-F2 (Track 3 D-3). Cycle the Shift gadget to Either, and type 81 into Raw. Cycle the top-right gadget to Execute ARexx File, and type track3d3 into Command. Finally, save the shortcuts and close the window.

30) Move to one of blocks 0 to 12 in our demo song (C-3 should be on every second line in track 3). Now press Shift-F2 to execute our new script.

This time it really is slow! So perhaps you'd like to add 'op_update' commands as mentioned earlier. (These commands 'switch off' the Tracker editor and Information window for the duration of the script).

Right. Let's puzzle over this script. After the three very important lines - remember? - the cursor moves to track 3. Now we have our DO loop. The variable currline is set to represent the values 0 to 63 in steps of 2: 0, 2, 4, 6 and so on right up to 63. (Well, 62 because 63 is an odd number).

What does this mean? Well, currline is firstly given the value 0. From then on, everywhere that ARexx sees 'currline', it replaces it with 0. So 'ED_GOTO LINE' currline (the first line of the DO loop) is replaced with 'ED_GOTO LINE 0'. So OctaMED moves to line 0.

The next line is ED_SETDATA NOTE 27 which sets note D-3 (I'll explain later). Then ARexx reaches the end of the DO loop, marked by END. So it adds 2 to the value of currline (0), giving a new currline value of 2.

After checking that this new value is not more than 63, the DO loop starts again. 'ED_GOTO LINE' currline is replaced with 'ED_GOTO LINE 2'. OctaMED moves to line 2, and changes the note to D-3. ARexx reaches the end of the DO loop again, and adds 2 to currline, giving a new value of 4. And so on.

When currline eventually becomes 62, and the end of the DO loop is reached, ARexx adds 2 to currline as usual. This new value is 64, which is more than 63, so the DO loop finishes. ARexx continues with the line after END.

Out of all this come three main points:

i) The DO loop sets up its variable (currline in this case) for the whole loop straight after the word DO. Here, the variable runs from 0 to 63 in steps of 2. If it were to run in steps of 1, you would use DO currline = 0 TO 63 BY 1, or more simply DO currline = 0 TO 63.

ii) Every occurrence of the variable's name in the DO loop is replaced by each value of the variable. Here, 'ED_GOTO LINE' currline is replaced by 'ED_GOTO LINE 0', 'ED_GOTO LINE 2', and so on. (Note the position of the quotes in the line 'ED_GOTO LINE' currline).

    Note that if the variable's name doesn't appear in the DO loop at

all, the variable is useless. For example, if you just wanted the same set of commands to be repeated 64 times, you need only use:

```
DO 64
<insert commands here>
END
```

(You may have noticed that this script only works for blocks 64 lines long. We'll adapt the script to work for any length of block later).

iii) Don't forget to mark the end of the DO loop with END! Also notice that the part between DO and END has been indented (moved to the right) by three spaces. I'd recommend this, it makes for an easier read.

Oh yes, I promised to explain the line 'ED_SETDATA NOTE 27'. It changes the current note to D-3: the number 27 represents the note D-3. This works by simply counting up from 1. 1 = C-1, 2 = C#1, 3 = D-2, .. 12 = B-1, 13 = C-2, 14 = C#2 and so on. Count up far enough and you'll reach 27 = D-3. (By the way, 0 = ---, the empty note).

IF...THEN STATEMENTS (How to choose how to act depending on a result)

ARexx, being a friendly system, doesn't always demand OctaMED to do things; it can also ask questions. In the midst of a script, you may need to know the current Tracker editor line, or the name of the current instrument, or whether or not the song has been edited since last saving. Indeed, the very first OctaMED command we entered asked the program its version number.

ARexx's IF and THEN devices enable you to choose a course of action depending on OctaMED's answer to a question. We'll look at how to use IF and THEN, and at how to ask the questions in the first place. Our final ARexx script in this brief tour will set all instruments' Hold value to 6, but it will ignore empty instrument slots.

31) Load the Ed text editor in the usual way. Call this script rexx:sethold.omed. Here it is:

```
/* SetHold.omed: Sets all instruments' hold values to 6 */
/*         (ignoring empty instruments) */

address octamed_rexx
options results

do inst = 1 to 63
   'in_select' inst

   'in_isslotused'
   if result = 1 then 'in_setholddecay hold 6'
end

'in_select 1'
'wi_showstring Hold values set.'
```

Save and quit when you've finished.

32) Using the Keyboard Shortcuts window, set Shift-F3 to execute sethold.
    (Set Raw to 82. Look back at steps 22 to 25 if you're unsure).

33) Now press Shift-F3. Open the Instrument Parameters window to check that
    all instruments' Hold values are now 6, apart from empty slots.

So how does this one work? A DO loop is set up, with variable inst running from 1 to 63 in steps of 1 (remember: if BY is left out, the step size is 1). Each instrument is selected in turn by replacing 'IN_SELECT' inst with 'IN_SELECT 1', 'IN_SELECT 2' and so on. (Note that IN_SELECT uses normal decimal values: IN_SELECT 10 selects instrument 0A rather than 10).

Now comes the new part. You can perhaps guess its meaning without further explanation. Is slot used? If result = 1, then set hold to 6. What actually happens is the following:

a) IN_ISSLOTUSED asks OctaMED if the current instrument slot has been used (i.e. isn't empty). OctaMED answers by giving the variable result a value. It becomes 1 if the slot is used, or 0 if the slot is empty.

b) The next line is straightforward. IF the variable result is 1, i.e. the slot is used, THEN set the current instrument's Hold to 6. This implies that if the result is 0, i.e. the slot is empty, then don't touch the Hold value and continue with the next instrument instead.

So there are two main points to be made:

i) OctaMED's 'question' commands, such as IN_ISSLOTUSED, return their answer in the variable result.

   In fact, they can be returned in any variable you wish, with a little more effort. Remember our second script, Track3D3? I mentioned at the time that the script only worked for 64-line blocks. Well, try making these two alterations:

I) Before DO currline =..., insert: 'ed_getnumlines var numlines'
   II) Change TO 63 to TO numlines - 1

   The command in step I returns the number of lines in the current block in variable numlines (notice the use of VAR). In step II, currline now runs from 0 to the number of lines in the block minus 1. (Why minus 1? In a 64-line block, the last line number is 63. So...)

   By using the same technique, you can get our most recent script to stop when it reaches the last used instrument, instead of carrying right on to instrument 63. See if you can work out how!
   (Hint: The commands IN_SELECT LASTUSED and IN_GETNUMBER VAR last will return the last used instrument in variable last)

ii) The IF and THEN devices can be used to act in various ways depending on the result of an OctaMED command. Try replacing the IF...THEN line in our script with the following:

```
if result = 1 then do
   'in_setholddecay 6'
   'in_setvolume 32'
end
```

```
  else 'in_setname Empty'
```

Now, if result is 1 then Hold will be set to 6 and the default volume will be set to 32. (Notice the use of DO and END: this is necessary if OctaMED should carry out more than one command. It isn't a DO loop!).

Also, if result is NOT 1 (i.e. if it's 0, or if it's anything else for that matter), then the instrument's name will be changed to Empty. So ELSE tells ARexx what to do if the required condition is not met.

Note that you can use symbols other than '=' with IF...THEN. Try < for less than, > for greater than, <= for less than or equal to, >= for greater than or equal to, or ~= for not equal to. Also use & for and and | for or. For example, IF result = 1 & last < 15 THEN... .

WRITING A SCRIPT

So you've followed all the examples, and you've (at least, sort of) understood how they work. The next obvious step is to start writing your own scripts. But where on earth do you start?

The answer is quite simple: with an idea. You are composing in OctaMED, and suddenly you think, "Wouldn't it be nice if I could write a script to...". Presto, you have an idea for a script. You're a quarter of the way there already.

But the idea's kind of vague. Next you need to form a precise idea. For example, my original idea for the previous example script was,

1) IDEA: "How about a script that sets all instruments' Hold values to 6?"

But then I thought, "Hold on a minute, what if the instrument is empty? Changing an empty instrument's Hold value is no good to anyone!" The key words there are "what if". Think of all the possibilities. All the things that could go wrong. So I came up with:

2) PRECISE IDEA: "How about a script that sets all instruments' Hold values to 6, but ignores empty instruments?"

Now it's time to translate informal English into more formal pseudo-code. This is the transitory stage between your idea and the final script.

3) PSEUDO CODE:

1. Let variable inst run from the first to the last instrument
2. Select the 'inst' instrument number
3. Is the current slot used?
4. If so, set the current instrument's Hold value to 6
5. If not, continue

Finally think, "What commands does OctaMED and ARexx provide to do this?" And come up with your AREXX CODE (step 4).

Of course, it's quite rare that scripts work first time (unless they're very simple). Errors do tend to creep in... Any errors in your script will be displayed within OctaMED in a window entitled ARexx Output.

```
And that's it! I hope this tutorial's been of some use. A couple of notes
mentioned in the original tutorial:

1) You might find the ARexx command EXIT useful when dealing with things
   that may go wrong. For example, if the script is to quit if no range is
   marked:

   'RN_ISRANGED'
   IF result = 0 THEN DO
      'WI_SHOWSTRING Please select range first.'
      EXIT
   END

2) A quick word about quotes: If the variable newname contains a string,
   use the following command to set the current block's name to newname:

  'ED_SETBLOCKNAME "'newname'"'

   Don't ask why!
```

1.3 Using ARexx with OctaMED / ARexx Features

```
                HOW TO...

                     Use Action sections

                 Use the ARexx command shell

                   Execute any ARexx script
                OVERVIEW
The main way of using ARexx with OctaMED is through 'Action'
sections, available in three different parts of the program. The
Keyboard Shortcuts, Input Map Editor, and ARexx Trigger Setup
windows all have Action sections.

The ARexx command shell is a way of testing OctaMED's ARexx
support. Through the Execute ARexx Script file requester, any ARexx
script can be loaded and run.
```

1.4 ARexx Features / How To Use Action Sections

```
        HOW TO USE ACTION SECTIONS
INTRODUCTION
OctaMED supports ARexx in three different windows using the same
method. The Keyboard Shortcuts window, Input Map Editor and ARexx
Trigger Setup window all support ARexx, and all have a section of
their window labelled 'Action'.

Using an Action section, OctaMED can perform an OctaMED ARexx
```

command, execute an ARexx script, send an ARexx command to another program, and start another program running. The circumstances in which these things are done depends on the window. In the Keyboard Shortcuts window, for example, pressing a particular key combination does one of the above things.

An Action section consists of a cycle gadget, a Command text box and an ARexx Port text box. The cycle gadget's options are as follows:

* OctaMED Command: The option you'll use most often: it executes an OctaMED ARexx command. Type the command into the Command box, together with any required command parameters. To execute more than one command in this way, use the OP_MULTICMD command.

* Execute ARexx File: Runs an ARexx file from disk. Type its name into the Command box; unless the file is in PROGDIR: or REXX:, specify the full path name. The filename should end in .omed, but this isn't obligatory.

* Ext. ARexx Command: Sends an ARexx command to another program. Type the comand into the Command box, and the program's ARexx port name into ARexx Port.

* Launch Program: Loads and runs a program from disk. Really, the program is 'launched' so that OctaMED can continue operating while the program is running. Type its filename into the Command box.

NOTES
* To use Execute ARexx File or Ext. ARexx Command successfully, make sure you've run the RexxMast program.

* The maximum length of the Command box is 255 characters.

* If an ARexx script produces any output or reports an error, OctaMED opens an ARexx Output window to display the text.

ALSO SEE
How To Set A New Keyboard Shortcut
How To Set Up And Use An Input Map
How To Set A Player Command To Perform An ARexx Function

1.5 ARexx Features / How To Use The ARexx Command Shell

HOW TO USE THE AREXX COMMAND SHELL
INTRODUCTION
Using the ARexx command shell, OctaMED's ARexx commands can be tried out before they are used in an ARexx script. Also, the 'template' - accepted parameters - of any command can be displayed.

STEPS
1) Open a command shell. Several can be open simultaneously.
> Project menu -> Command Shell

```
2) Enter OctaMED ARexx commands. Follow the command name with a ?
   to display the command's template.

3) If necessary, clear the window.
   > Type clear and press Return OR
   > Type cls and press Return

4) When you've finished with the window, close it.
   > Click its close gadget OR
   > Type bye and press Return OR
   > Press Ctrl-\
```

1.6 ARexx Features / How To Execute Any Arexx Script

```
      HOW TO EXECUTE ANY AREXX SCRIPT
INTRODUCTION
As an alternative to an Action section, an ARexx script can be
executed through a file requester, opened through the Project menu.
It's especially useful for quickly running an ARexx script without
having to set up an Action section.

By default, the REXX: directory is loaded into the file requester.
If you have a favourite directory in which you keep ARexx scripts,
the file requester can load that instead. Type the directory path
into the ARexx Scripts text box in the Default Directories window.  Where?

STEPS
1) Open the Execute ARexx Script file requester.
   > Select Project menu -> ARexx Script

2) Select a script from the requester.             How?

NOTES
* There is one restriction on the name of the ARexx script: There
  must not be a space in either the path name or the file name.
  This has the unfortunate consequence that scripts in RAM: can't
  immediately be exectued, for RAM: has the volume name "Ram Disk"
  which contains a space!

  To get around this, relabel RAM: with a name not containing a
  space. For example:
      relabel ram: RamDisk

  You could open an AmigaDOS shell to do this, through the Project
  menu.
```