MIDISYStemEXplorer

COLLABORATORS

	TITLE : MIDISYStemEXplorer	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		August 24, 2022	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1 MIDISYStemEXplorer **1**

1.1 MIDI SYStem EXplorer Documentation . . . 1
1.2 Introduction . . . 2
1.3 FAQ . . . 5
1.4 Demo Limitations . . . 8
1.5 Registering . . . 8
1.6 Development and Testing . . . 9
1.7 Installing . . . 9
1.8 Configuration . . . 10
1.9 ARexx . . . 12
1.10 IR Shell . . . 13
1.11 Problems . . . 14
1.12 Conventions . . . 14
1.13 Busy . . . 15
1.14 GetGad . . . 15
1.15 GetModule . . . 16
1.16 LoadPatch . . . 16
1.17 OpenModule . . . 17
1.18 Request . . . 17
1.19 SavePatch . . . 17
1.20 SetByte . . . 18
1.21 SetGad . . . 18
1.22 Quit . . . 19
1.23 Version . . . 19
1.24 Tutorial . . . 19
1.25 Menus . . . 23
1.26 Legalness . . . 24
1.27 Future Enhancements . . . 24
1.28 Known BUGS . . . 24
1.29 Getting Updates . . . 25
1.30 Locating The Author . . . 25
1.31 Previous versions . . . 25
1.32 New in this version . . . 28

Chapter 1

MIDISYStemEXplorer

1.1 MIDI SYStem EXplorer Documentation

MIDI SYStem EXplorer

Introduction

DEMO Limitations

FAQ

Registering

Development and Testing

Installing

Menus

Tutorial

Configuration

ARexx

Future Enhancements

Known BUGS

Legalness

New in this version

Getting Updates

Previous versions

Locating The Author

MIDI SYStem EXplorer

A utility for EXploring the world of MIDI SYStem EXclusive

1.2 Introduction

```
                Introduction
------------

NOTES
-----

For newcomers... be sure to have a look at the
                Tutorial
                 for a walk
through of the many featues of MSE.

If you've tried out MSE before... check out some of the new demo modules
included with this release... in particular:

One of the best and newest features of this MSE is automation.  To see
it in action load the TEST module and click on the automate gadget in
the upper left hand corner.  This gadget has been linked to an external
ARexx script which will do some fun things with the slider gadgets
below it.


FEATURES
--------

   - Fully automated control.

   - ARexx support.

   - Openable on any public screen.

   - Extensive customization allows the user to design standard
     interfaces for all MIDI products connected to the system.

   - Graphic interfaces are designed external to the main program.  This
     minimizes memory requirements and increases the interface speed.

   - Just about everything is customizable, including: layout, fonts,
     gadgets, colors, images, menus, MIDI events, etc.

   - Allows the user to enter MIDI information from a device's user manual
     and configure a custom interface to access parameters.

   - Supports ALL MIDI events.

   - Patch/Librarian data is stored in MIDI standard format, for easy
     importing to other software.

   - Has the ability to edit individual parameters on devices that only
     support patch dumps.

   - MIDI communication can operate with realtime feedback and/or through
```

user interaction.

- Edits can be made without connection to a MIDI device, if required.

- Includes lots of examples and common device interfaces, to get you started.

- Supports MIDI.library (PD), BlueRibbon.library (B&P), TriplePlayPlus, and CAMD.library (DeluxeMusic) allowing simultaneous use with other programs that also use these libraries.

ABOUT MSE

Many, so called, UNIVERSAL PATCH EDITORS have come and gone for the Amiga. One of our biggest complaints, and the reason why we are making MSE, is that these editors are not anywhere near UNIVERSAL. The approach others have taken is in requiring 'modules' to be purchased for each MIDI device to be controlled. This is fine as long as you can get a module for EVERY device you have. But we all know that, especially on the Amiga, this has not ever happened. Then of course, when the company goes out of business or discontinues the program, no further modules are ever made.

So, along come a few other public domain and shareware authors to try and fill in the hole. These programmers either chose to code individual editors or make an another attempt at a UNIVERSAL EDITOR of their own. These UNIVERSAL EDITORS require you to be able to program in C or Assembler in order to make your own editors. If you wanted to do that why would you bother with their patch editor at all?

Well, MIDI SYStem EXplorer has arrived... we do not claim that MSE is a UNIVERSAL PATCH EDITOR... we are not sure such a thing is even possible, but MSE does strive to solve a lot of these problems:

1) Development of other programs stopped because nobody was buying it, and no editor exists for my new one of a kind MIDI ZEN-ZAMBULATOR.

MSE is being developed for our own studio use, first! Even if nobody else wants to use it, we will continue to develop it. Its development will continue regardless of whether anyone buys it or not. We have been using our Amigas for over 10 years, and we do not see that stopping for a long time.

Even if we did stop developing MSE, you would very likely still be able to create new modules, because MSE already supports most MIDI formats.

2) You are a musician, you do not know how to program in C, or Assembler or any other language. And if you did you would rather be composing than programming modules!

MSE supports modules created by the MSE-Compiler. It converts text definition files into modules that MSE can use. The compiler uses an easy to learn language specifically written to be as easy to understand as possible. In many cases, you can simply modify an

existing definition file to work with another device.

If you can use a text editor and you understand X/Y coordinates you will likely learn MSE's definition language fast. See the definitions directory for samples of definition files. Once you get a hang of things, making modules is FAST. And once you have the editor done, you will be amazed at how much time is saved and how much more creative the process of sound generation becomes!

Definition files and modules will be freely available. So, if you are not interested in making your own modules you can still use modules made by us and others.

3) Other editors do not work while my sequencer or other program is running.

MSE will run in conjunction with programs that use the public domain MIDI.library, BlueRibbon.library or CAMD.library.

Imagine being able to edit patch data while having that MIDI information recorded directly into a sequencer. It is possible and opens up great creative possibilities.

5) Other software is too expensive!

We think so too.

WHAT WILL MSE DO?

MSE has been designed to control all types of MIDI data through a programmable graphic user interface or by external software control. MSE can be used as a simple MIDI Controller or a complex patch editing system.

Some possible applications:

- MIDI Volume/Pan Mixer (with automated fading)
- Patch Editor
- Patch Librarian

With the definition language you can design the entire user interface to meet your needs. Screen mode/size (Amiga/CyberGFX/etc.), window size and position, fonts, borders, colors, gadget placement, etc. can all be customized. As well, you can define the character set, MIDI port (Triple Play Plus is supported) and much more.

WHAT WON'T MSE DO?

- Waveform editing (samplers) is not supported. Another program WaveFormer (under development) is being designed for this.

- Probably some other things too, but that first one is the most commonly asked about.

```
REQUIREMENTS
------------
- WB 2.x+
- Text editor
- MIDI interface
- MIDI.library, BlueRibbon.library or CAMD.library (included).
- Some understanding of MIDI documentation to create custom modules


RECOMMENDATIONS
---------------
- Simple paint program for layout design
- WB 3.x
```

1.3 FAQ

```
FAQ: Frequently Asked Questions
-------------------------------

------------------------------------------------------------------------------
> Does MSE support all MIDI devices?

> I would LOVE to see support for Roland Super Jx-10, Jx-8p, Juno-1 and 2,
> Mks-50, Juno-106, and lot of other gear.
------------------------------------------------------------------------------

We have done our best to support most of the features that we have seen so
far.  However, not everything can be controlled by MSE.  It is possible to
make partial editors so if something has not been implemented yet, it can
be omitted and added later.  There are 1000s of device MIDI
implementations, and we have not seen them all.  Some things may not yet be
supported simply because we are not aware of them.

Currently, devices that require handshake communication are not supported.
Keep in mind that some devices support both handshake and one-way
communication so those may work.

We suggest that you send us a copy of the SYSEX docs for the units you want
to make modules for (check with us first to see if we already have the
info).  We will have a look at the documentation and confirm that MSE
supports everything required by that unit.  If it does not, we can try and
implement it at that time.  That way, you do not have to spend your money
until the program meets your needs.


We are currently making, or have completed, the following modules:

  Akai S612
  Boss SE-50
  Korg Wavestation
  Oberheim Matrix-1000
  Oberheim Matrix-6R
  Roland A-50
```

```
  Roland A-880
  Roland Juno-106
  Roland MKS-50
  Roland MSE-1
  Roland R8 / R8M
  Roland S-550
  Roland SDX-330
  Roland SRV-330
  Yamaha SPX-90II
  Yamaha TX802

Currently available third part modules:

  Roland MKS-7

The following third party modules are also expected:

  Blue Ribbon One Stop Music Shop
  Emu Proteus
  Korg 05DR
  Peavey DPM-V3
  Peavey Spectrum
  Yamaha SY77
  Yamaha TG-33
  Yamaha TG-100
  Yamaha YS-200

  more as we find out about them...

------------------------------------------------------------------------------
> Is it possible to manage banks of sounds?
------------------------------------------------------------------------------

Currently, no.  But it is in the plans.  We are waiting to see how this is
implemented on more devices, so that we can design the definition language
changes accordingly.

------------------------------------------------------------------------------
> Does MSE coexist with Music-X?
------------------------------------------------------------------------------

We are trying to obtain the necessary information to better integrate MSE
with Music-X.  If someone has this information and is willing to share it
with us, we will be happy to improve on this.

For now, you will have to disable use of the MIDI port by Music-X, while
using MSE.

We have learned that Music-X uses CAMD compatible drivers though it does
not actually use CAMD.  We are looking further into this.

------------------------------------------------------------------------------
> Does MSE support the CAMD.library?
------------------------------------------------------------------------------

Yes!  Currently MSE supports the CAMD.library, Midi.library and the
BlueRibbon.library (which allows simultaneous use with other software that
```

uses them). We will further improve on this support in future versions.

> Will MSE open on the Workbench screen?

YES! MSE can open on any public screen.

> MSE did not fill up my Multiscan 640x495 res screen.

NTSC 640x400 is the default screen size used by MSE. It can be changed with a tooltype.

Note that modules included with the demo are all designed for 640x400 screens. Even if you change the screenmode, these modules will still have windows defined for a 640x400 screen. However, this can be changed by editing definition files and recompiling them with the compiler included with the registered MSE. The MSE screen can open to any size supported by your hardware.

> I would like to know more about what is involved in writing a driver.
> I am NOT a programmer! (But I am somewhat familiar with sysex as I have
> to use it so send various commands from time to time).

Hopefully, we have developed a 'definition language' that is simple enough for almost anyone to use. Some knowledge of hexadecimal/binary number systems would be of help, but may not be required. Much of this depends on the MIDI documentation for your device. Some manufacturers have very easy MIDI implementation, others have very cryptic and confusing methods.

We would be happy to help with SYSEX 'programming'. It is likely that once we showed you how to implement a few parameters, you could probably finish the rest on your own. If we have a copy of the SYSEX info, it would certainly make things much easier for us to help you out.

> I would prefer more control over the program, like screenmode requester.

Screenmode, depth, etc are set with tooltypes. Currently, these can not be defined for each module. The thinking is that all modules should be setup the same to maintain consistancy. Of course, each user can decide what settings they want for all of their modules. There is not much that you can not control. Palettes can be defined within each module.

> I would like to select my own fonts, have draggable windows, etc.

Fonts and window options are set within each module definition. These settings as well as many others can be defined and then compiled into a module.

```
You can set the module fonts to anything or any size you want by
editing/creating the definition files and then compiling them with the
compiler.  The requesters, menus and title font can be set with a tooltype.

Draggable windows is currently not an option.  This will be added in a
future version if there is enough request for it.

------------------------------------------------------------------------------
> The .readme said new modules can be edited by users, but the examples
> are binaries.  Please explain.
------------------------------------------------------------------------------

Modules are NOT editable, they are created by MSE-Compiler (not included
with the demo).  Definition files, however, are simple text files.  The
demo includes sample definition files in the definitions directory.
```

1.4 Demo Limitations

```
Demo Limitations
----------------

- MIDI Channel is randomly chosen on startup.  You can see what it is by
  going to the MIDI Configuration menu, but you can't change it.

- Options for MIDI Handler, other than MIDI.library (serial.device), are
  disabled.  Shared access to the MIDI Port is disabled.

- Module menu functions are disabled, so MSE must be quit and restarted
  to load a different module.

- Can only load modules included with the demo.

- Ability to load/save and create default patches is disabled.

- Project (accept for QUIT), Prefs, ARexx and Utility menu functions
  disabled.

- Tooltypes disabled.

- Preferences loading/saving disabled.

- Some ARexx commands disabled.

- Module compiler not included with demo, but may be in a future release.

- Sending of MIDI messages limited to the first 10 gadgets in a
  module.
```

1.5 Registering

```
                Registering
-----------
```

For a limited time, MSE will be available for the low shareware
registration price of US$30 (CAN$40). After registering, any following
updates will be free to those with email access. This is your last
chance to get this program at this price. If you've been putting it
off... now's the time!

Print and complete the Registration.txt form and send it along with
payment to (Make checks or money orders payable to 'dhomas trenn'):

See:
Locating The Author
We suggest that you send us a copy of the SYSEX docs for the units ↩
you
want to make modules for (check with us first to see if we already have
the info). We will have a look at the documentation and confirm that
MSE supports everything required by that unit. If it does not, we can
try and implement it at that time. That way, you do not have to spend
your money until the program meets your needs.

Your shareware payment helps to insure continued development of MIDI
SYStem EXplorer.

See:
Future Enhancements

1.6 Development and Testing

Development and Testing

MIDI SYStem EXplorer is developed on an A3000T/060/50MHz with 2 Meg CHIP
/ 68 Meg FAST memory and Cybervision64/4 Meg at 1280x1024.

It is beta-tested on an A3000T with 2 Meg CHIP / 12 Meg FAST memory at
640x400, and an A600 with 2 Meg CHIP at 640x400.

MIDI SYStem EXplorer has been tested under Kickstarts 2.x and 3.x.

1.7 Installing

Installing

Requirements

This demo uses fonts included with WB: Garnet.16 and Helvetica.9.

Installing

Copy the contents of fonts/ to your FONTS: directory. The included
Topaz2, Topaz2p, and Topaz2N fonts are designed for use with the demo
modules. These modules have been defined to use these fonts, but they
are not necesarily required for custom modules you create. Topaz2 is a
nicer looking version of Topaz, Topaz2p is a proportional version of
Topaz2, Topaz2N is a narrow version of Topaz2.

Copy the contents of libs/ to your LIBS: directory.

1.8 Configuration

Configuration

Configuration is supported through use of Workbench ToolTypes or CLI
parameters. If any options are not included, the stated defaults will be
used.

Configuration options are not case sensitive.

Invalid options will be ignored. In this case, the default values will
be used.

The following configuration commands are supported:

  Depth
    Description: Sets the palette depth for the MSE screen. The number
                 of colors is equivalent to 2^depth.

                 Ignored if the 'PubScreen' tooltype is set.

    Valid values: >= 1

    Default: 2

    Example: Depth=4

  Font
    Description: Sets the font for the screen title and requesters. This
                 setting has no control over fonts defined within
                 modules. Can be enclosed within quotes, but is not
                 necessary.

    Default: Topaz2p.font

    Example: Font=helvetica.font

  FontSize
    Description: Sets the font size for the screen title and requesters.
                 This setting has no control over fonts defined within
                 modules.

```
  Default: 8

  Example: FontSize=11


Last
  Description: Opens the last opened module upon startup.  No file
               requester is presented unless the last used module can
               not be found.

  Valid values: Last

  Default: NO

  Example: Last


ModeID
  Description: Sets the ModeID of the MSE screen.  Value must be in
               decimal format.  If ModeID is not availabe, MSE use its
               default.

               Ignored if the 'PubScreen' tooltype is set.

  Default: 102404                   (NTSC: Hires Laced)

  Example: ModeID=1074921474        (CVision: 8bit 800 x 600)


Module
  Description: Opens the given module upon startup.  No file requester
               is present unless the module can not be found.

               The module name is referenced as the path and filename
               within the 'modules' directory.

  Default: NONE : FileRequester

  Example: Module=Roland/Juno-106.mse


OffsetX
  Description: If the 'PubScreen' tooltype is set, this sets the X
               offset of the main window on that screen.  All module
               windows are positioned relative to this position.

  Valid values: >= 0

  Default: 0

  Example: OffsetX=50


OffsetY
  Description: If the 'PubScreen' tooltype is set, this sets the Y
               offset of the main window on that screen.  All module
               windows are positioned relative to this position.
```

```
    Valid values: >= 0

    Default: 0

    Example: OffsetY=50


  PubScreen
    Description: Sets the publicscreen that MSE will open on.  Overrides
                 the 'ModeID' tooltype.

    Default: MSE opens on its own screen.

    Example: PubScreen="Workbench"


  ScreenH
    Description: Sets the height of the MSE screen.  Screens sizes that
                 are larger than the maximum display will be
                 auto-scrolling.

                 If the 'PubScreen' tooltype is set, this sets the
                 height of the main window.

    Valid values: >= 0

    Default: 400

    Example: ScreenH=600


  ScreenW
    Description: Sets the width of the MSE screen.  Screens sizes that
                 are larger than the maximum display will be
                 auto-scrolling.

                 If the 'PubScreen' tooltype is set, this sets the
                 width of the main window.

    Valid values: >= 0

    Default: 640

    Example: ScreenW=800
```

1.9 ARexx

```
                ARexx
-----


                 IR Shell

                 Problems
```

```
                Conventions
                 Commands

                Busy

                GetGad

                GetModule

                Request

                SetByte

                SetGad

                Version

                OpenModule

                LoadPatch

                SavePatch
               MSE's ARexx port name is "MSE.1".

The following result codes (RC) are set:

   0 - No Error
  10 - Unknown Option
  15 - Invalid Value
  20 - Syntax Error
  30 - Unknown Command


Unless otherwise specified, all values are returned in the RESULT variable.
```

1.10 IR Shell

```
IR Shell
--------

'IR Shell' is an interactive shell for testing MSE's ARexx commands and
functions.  It is an external ARexx program itself so it can be modified,
if you so wish.  It can be called from within MSE via the menus, or can
be launched externally.

It has been setup with some handy features to simplfy testing:

 -  automatic display of non-zero result codes (RC) upon completion of
    commands.

 -  '=<enter>' is provided as a shortcut for: say "RESULT = "RESULT

 - 'options results' is auto-activated.
```

```
* Multiple shells may be open simultaneously.

* Beware that some commands, such as LoadPatch and OpenModule currently
  cause display problems with the IR Shell window.

* QUIT is not possible if an IR Shell is active.
```

1.11 Problems

```
Problems
--------

Beware of potential quote problems.  ARexx will always strip quotes.  So
this works:

  Request '"Title Test" "Body text goes here."'

Because MSE receives this: (note the stripped '')

  Request "Title Test" "Body text goes here."

But this will not work:

  Request "Title Test" "Body text goes here."

Because MSE receives this:

  Request Title Test Body text goes here.
```

1.12 Conventions

```
Conventions
-----------

bold denotes required
plain denotes optional
bold+italic denotes optional-required.

[...] Denotes

<> Denotes required items.  The '<' and '>' characters should not be
   included.

[] Denotes optional items.  The '[' and ']' characters should not be
   included.

{} Denotes optional-required items.  Only one of these options may be used
   within a structure but one of them is required.  The '{' and '}'
   characters should not be included.

|  Denotes OR.  Only one of these options may be used with this item.
   The '|' character should not be included.
```

1.13 Busy

```
Busy
------

Busy <number> YES|NO
  Description: Sets the busy state.

        Where: ON    turns the busy state ON
               OFF   turns the busy state OFF

        Notes: For long/complex ARexx scripts it is best to set the busy
               state ON at the beginning and OFF at the end to avoid
               confusion.  If an error occurs during the execution of an
               ARexx script, the busy state will automatically be turned
               OFF.

               Never set Busy ON for scripts that are triggered from
               continuous type gadgets such as slider.  This will cause
               the program to dead lock.

      Example: Busy ON
```

1.14 GetGad

```
GetGad
------

GetGad <number> {RANGE} {TYPE} {VALUE}
  Description: Returns information about a Gadget.

        Where: <number> is the number of the gadget.
               [RANGE]  returns the range of valid values for the
                        gadget.  Ranges are returned as follows:

                        Button:  0 0
                         Cycle:  0 <MAX>
                            MX:  0 <MAX>
                        Slider:  <MIN> <MAX>
                        String:  0 <MaxChars>

               [TYPE]   returns the gadget type.
               [VALUE]  returns the gadget's current value.

       RESULT: Requested information separated by spaces and in the
               order listed.

        Notes: A returned VALUE for a gadget with type STRING may
               contain spaces.  Therefore, it may be necessary to avoid
               using VALUE in conjunction with other options for STRING
               gadgets.  Otherwise, you may not be able to properly
               parse the results.

               Gadgets do not need to be currently on screen.
```

```
      Example: GetGad 5 VALUE TYPE
       RESULT= 36 Slider
```

1.15 GetModule

```
GetModule
---------

GetModule {DEVICE} {GADGETS} {MANUFACTURER}
  Description: Returns information about the current module.

        Where: [DEVICE]         returns the device name.
               [GADGETS]        returns the number of gadgets.
               [MANUFACTURER]   returns the manufacturer name.

       RESULT: Requested information separated by spaces and in the
               order listed.

        Notes: A returned name may contain spaces.  Therefore, it may
               be necessary to avoid using name options in conjunction
               with other options.  Otherwise, you may not be able to
               properly parse the results.

      Example: GetModule GADGETS MANUFACTURER DEVICE
       RESULT= 14 monkey volumizer
```

1.16 LoadPatch

```
LoadPatch
---------

LoadPatch [<patch>] [DEFAULT]
  Description: Loads a new patch into the current module.

        Where: [<patch>]  is the file name with full path.
               [DEFAULT]  indicates the default patch (.dm).

       RESULT: Loads the requested patch or sets RC='Invalid Value' if
               patch is not found or the module does not contain a
               'Loadable' Define Midi definition.

        Notes: If no parameters are included a file requester will be
               presented and the selected patch returned in RESULT.
               Although MSE will refuse to load a patch if it detects
               that it is not a patch for the current module, the invalid
               filename will still be returned in RESULT.

     Examples: LoadPatch PROGDIR:Modules/Roland/Juno-106.dm
               LoadPatch DEFAULT
               LoadPatch
```

1.17 OpenModule

```
OpenModule
---------

OpenModule [<name>] [patch]
  Description: Opens a new module and optional patch.

        Where: [<name>]  is the module name, referenced as the path and
                         filename within the 'modules' directory.
               [patch]   is the file name with full path.

       RESULT: Opens requested module and optional patch or sets
               RC='Invalid Value' if module or patch is not found.

        Notes: If no parameters are included a file requester will be
               presented and the selected module returned in RESULT.

     Examples: OpenModule "Roland/Juno-106.mse"
               OpenModule "Roland/Juno-106.mse" "PROGDIR:patches/j106.p"
               OpenModule
```

1.18 Request

```
                Request
-------

Request <title> <body>
  Description: Displays a system requester.

        Where: <title>  is the title for the requester.
               <body>   is the text to display in the body of the
                        requester.

       RESULT: NONE

        Notes: See:
                 Problems
                 for potential quote problems.

      Example: Request '"Automation Demo" "Waterfall"'
```

1.19 SavePatch

```
SavePatch
---------

SavePatch [<patch>] [DEFAULT]
  Description: Saves a patch.

        Where: [<patch>]  is the file name with full path.
               [DEFAULT]  indicates the default patch (.dm).
```

```
       RESULT: Saves the patch or sets RC='Invalid Value' if the module
               does not contain a 'Loadable' Define Midi definition.

        Notes: If no parameters are included a file requester will be
               presented and the patch name returned in RESULT.

     Examples: SavePatch PROGDIR:Modules/Roland/Juno-106.dm
               SavePatch DEFAULT
               SavePatch
```

1.20 SetByte

```
SetByte
------

SetByte <offset> <value>
  Description: Stores a value within a 'Loadable' MIDI message.

        Where: <offset> is the byte offset within the MIDI message.
               <value>  is the value to be stored.

       RESULT: Stores <value> at <offset> or sets RC='Invalid Value' if
               patch is not found, the module does not contain a
               'Loadable' Define Midi definition, or <offset> is greater
               than the message length.

        Notes: <value> must be a single byte value (0-255).  <offset>
               counts from 0.

      Example: SetByte 5 8
```

1.21 SetGad

```
SetGad
------

SetGad <number> <value>
  Description: Sets the value of a gadget.

        Where: <number> is the number of the gadget.
               <value>  is the value to set the gadget to.

       RESULT: NONE

        Notes: <value> must apply to the type of gadget being set.

               Gadgets do not need to be currently on screen to be set.

               Trying to set a Button type gadget will always generate
               an Invalid Value error.
```

```
      Example: SetGad 5 8
               SetGad 3 PatchName
```

1.22 Quit

```
Quit
----

Quit
  Description: Quits MSE.

        Notes: Same as selecting 'Quit' from the Project menu.  No
               saving is done.
```

1.23 Version

```
Version
-------

Version [MODULE]
  Description: Returns an MSE version string.

        Where: [MODULE] requests the version of the currently loaded
                        module.  Without this option the version of MSE
                        is returned.

       RESULT: MSE or current module version string.  If MODULE is used
               and no module is loaded an empty string is returned.
```

1.24 Tutorial

```
                Tutorial
--------

**********************************************************************

NOTES about the unregistered DEMO version of MSE:

- In order to load a different module, you have to first quit MSE and
  then restart it.

- In the demo version, only the first 10 gadgets of a module will
  transmit MIDI messages.

- To enable MIDI, go to the 'Prefs/MIDI Configuration' menu, select the
  MIDI handler and close this prefs window. The MIDI Channel is randomly
  selected on startup and can not be changed.  Look in the 'MIDI
  Configuration' prefs to see what channel is being used.  In the DEMO,
  this will have to be done everytime you start the program.
```

- Some parts of this tutorial are not possible in the DEMO. But, keep reading because other sections will work.

****

Double-click on the MSE (or MSE-Demo) icon.

When MSE starts, you will be prompted with a file requester to choose a module to load. Enter the Oberheim directory and select the Matrix-6R.mse module. This is a demo of an editor for the Matrix-6R, but does not require this device.

Click in the string gadget containing "--------" at the top middle of the screen. This is a gadget defined to contain the patch name for this patch. Because almost all patch names are required to be a constant length this gadget is forced to be in OVERWRITE mode and will not allow you to DEL characters. Try it, but don't press ENTER, yet.

Many manufacturers use different character sets for their devices. Some do not allow lowercase letters while others do. MSE allows you to define what the character set contains for every module you create. Curious about what characters are supported by this module? While in the patch name gadget press the HELP key (3.x only). A requester will pop up showing you what characters are supported. Notice that lowercase letters are not shown for this device. For your convenience, lowercase letters will be automatically converted to uppercase for this device. Also, you will not be allowed to enter any characters that are not allowed. Try it!

The cycle gadget in the upper left hand corner labled "DCO 1 / 2" is configured to flip through several pages of parameters. Try it! MSE uses system gadgets so its cycle gadgets are compatible with patches like CycleToMenu. This system patch, in particular, really enhances the speed of editing patches in MSE.

Note that you could, if you wanted to, redefine/reposition anything you see. Of course you will need the compiler included with the registered version of MSE to do it.

Play with the gadgets on these pages, change lots of values! Now, if you decided that the current values were what you wanted every new patch to default to you simply go to the PROJECT menu and select SAVE AS DEFAULT. Try it. Now whenever you restart this editor, these settings will be the default.

Let's load an existing patch... go to the PROJECT menu and select LOAD. When prompted with the file requester choose the PATCHES directory and select 'Matrix6R-Patch1'. This patch is loaded and returns to the first page of parameters. If this was not a Matrix6R patch, you would have been told so, and the patch would not have been loaded.

In a future version, MSE will automatically recognize the patch type and load the appropriate editor along with the patch! NEETO!

To start with a fresh patch, go to the PROJECT menu and select LOAD DEFAULT. Note that any time you load a patch, MSE automatically

performs a defined initialization routine. For each module you can define what the intialization includes. Normally this would include sending the patch to a predefined audition patch on the device. You can also manually send a patch by going to the PROJECT menu and selecting SEND.

Now let's check out the PREFS... goto the PREFS menu and select MIDI CONFIGURATION. These can be defined within each module. Check out the HANDLER gadget... MSE currently supports standard BlueRibbon.library use, the 3 outputs of the Triple Play Plus midi interface, the public domain MIDI.library, Commodores CAMD.library and a special FILE type. FILE is used for testing purposes and sends all MIDI output to a file. This makes things much easier when testing new modules. As well, you can override the defined MIDI channel here. Note that for this device channels 1-16 are supported. Some devices, such as the Akai S612 do not support all 16 channels, others support more. If that were the case you can configure this gadget to only allow the supported channels. Remember, this setting and all others can be stored with each module, so you don't have to change it each time you open a different module.

Reposition the MIDI CONFIGURATION window to some where else on the screen and then close it. Now, open it again. Notice that it remembers its new position. If you want MSE to remember this the next time you use it, goto the PREFS menu and select SAVE. SAVE stores the window configurations as well as directory paths and other preferences information.

Now, let's check out another module. Goto the MODULE menu and select OPEN. In the file requester, choose the TEST directory and select the 'Test.mse' module. This module is not an editor for a MIDI device. It just serves to demonstrate some of the different kinds of things you can define: Borderless and non-borderless windows, borders, different size fonts, etc. Note the custom slider gadget near the center, it is labeled "Special Gadget". It is similar to a cycle gadget in that it contains several different text values. Also the gadget below it, which defaults to '-6.9db'. This is another custom gadget which supports floating point values with any incremental value you want. MSE also supports several other custom gadgets that are designed specifically to meet the needs of MIDI devices.

The window that says 'Drag me!' has been defined as a draggable window. Use its drag bar to move it anywhere you want. This window also contains another custom gadget (the Note gadget). It is used for setting parameters such as note ranges. It has actually been defined to display the current value in two different ways, so you'll see two values updating as you change the value.

One of the best features of MSE is automation. To see it in action click on the 'Automate' gadget in the upper left hand corner. This gadget has been linked to an external ARexx script which will do some fun things with the gadgets in this module.

ARexx can also be used to set values of other gadgets or to perform complex tasks. Try changing the values of the first two slider gadgets below the word 'Calculate' (left-side middle). These two gadgets are linked to an external script which in real-time adds their current

values together and sets the third gadget to their sum. This simple example isn't very useful itself, but further demonstrates the power of MSE.

In a future version, popup windows will also be definable. These special windows will be triggered from button gadgets. For example, the Note gadget popup window will contain a graphic keyboard that you can select notes on. When you close the popup-window the value will be displayed as it is now. These kinds of popups will support many kinds of parameters: including envelopes, lists, etc. This support is in the works now and will be ready soon.

Notice that most of the menu selections under the PROJECT menu are now ghosted. That is because the TEST module does not contain a definition for patch data. MSE recognized this and disabled the menus that did not apply. NIFTY EH!

MSE also recognized that this module contains special USAGE information. Goto the MODULE menu and select USAGE. This can come in handy when you need to remember something important to use this editor. You can define whatever text you want to appear here. HANDY!

Okay, lets try one more module. Go to the MODULE menu and select OPEN. In the file requester, choose the 'Roland' directory, then choose the 'SDX-330' directory and select the 'Stereo8PhaseChorus.mse' module. Note that the MSE-Compiler automatically creates these directories for you, based on the modules you create. This helps keep things organized. If you wanted to, you could make a custom module to load modules. Then you could tell MSE to always load that specific module upon startup. MSE is very powerful indeed!

Okay, now that we have loaded the SDX-330 module let's see what other neet things MSE can do... Down near the bottom right hand corner, there is a cycle gadget marked 1-4. This is a special gadget that is defined to control multiple pages (much like the one you tried earlier). Cycle it and MSE jumps to the next page of Chorus parameters 5-8. Notice that only that particular area of the screen changed. There is another cycle page selector at the top left corner, second row, marked 'Chorus' (not the one marked 'Stereo 8 Phase Chorus', that one will be explained next.) Cycle it, and it flips pages too. Super Neeto! The cycle gadget above that one marked 'Stereo 8 Phase Chorus' is defined to select modules. Because the SDX-330 has many effects modes each effect type has been broken up into separate modules. This cuts down on memory requirements for this editor. You could make it all into one module if you wanted to, but it would have 100s of gadgets and many many pages. This is the recommended way of handling devices of this kind. Try it. Note that this module is incomplete, a couple of the modes are not included here.

Now, select 'Play Notes' under the 'Utilities' menu. Just like the Prefs window, you can move this one around and MSE will remember it's position for next time. This little utility allows you to send notes to a module while editing patches. Very handy! Close this window and lets move on...

That concludes the TUTORIAL. It only touchs on the many things you can do with MSE. And there is much more yet to come! We have included a few

```
other modules as well, try them out.

We have also included some definition files used to create the demo
modules in the definitions directory.  They are simple text files, have
a look!

So are you hooked yet?  Excited about the creative possibilities?

Consider
                registering
               .
```

1.25 Menus

```
Menus
=====

Project ============

  Load              - load new patch.
  Save              - save current patch with current name.
  Save As...        - save current patch with a new name.
  ---------------
  Load Default      - load default (.dm) patch.
  Save As Default   - save current patch as default (.dm).
  ---------------
  Send              - transmit current patch.
  ---------------
  Initialize        - performs the Initialize function if contained in
  ---------------     the current module.
  About...          - displays information about MSE and the author.
  ---------------
  Quit              - quits MSE.


Module =============

  Open...           - opens a new module.
  Usage...          - displays the Usage info if contained in the
  --------            current module.
  About...          - displays information about the current module.


ARexx ==============

  --------
  IR Shell          - opens an interactive ARexx shell for testing MSE's
                      ARexx functions and commands.

Prefs ==============

  MIDI Configuration  - MIDI configuration options.
  ------------------
x MIDI Enabled        - If enabled, activates MIDI transmission.
x Auto-Initialize     - If enabled, automatically performs the Initialize
```

```
  ------------------    function, if present, when a new module is opened.
  Save                - Saves the current prefs configuration as default.
                        Includes: Positions of 'MIDI Configuration' and
                        'Play Notes' windows, 'Play Notes Octave'
                        setting, patch path, prefs settings for 'Auto
                        Initialize' and 'MIDI Enabled'.

Utilities ==========

  Play Notes        - opens the midi 'piano' keyboard note sending window.
```

1.26 Legalness

```
                Legalness
---------

MIDI SYStem EXplorer and all of its components are copyright ©1996-1997
by dhomas trenn and young monkey. The files included with this
distribution may not be altered in any way. MIDI SYStem EXplorer is

                 Registering
                .

The software concept and design remain the property of the author,
dhomas trenn and young monkey.

It is strictly FORBIDDEN for the REGISTERED VERSION to be distributed or
to appear in any public domain software archives, on any commercial disks
or CDROMs, or in any other form.

The author is in no way liable for any damage resulting from the use of
this program.
```

1.27 Future Enhancements

```
Future Enhancements
-------------------

MIDI SYStem EXplorer is updated regularly.

There will be lots of other things added in, making MSE even more
useful.  As always, we are happy to listen to any suggestions you might
have.
```

1.28 Known BUGS

```
Known BUGS
----------

- For some unknown reason, after running the automation demo in the
```

Test/Test module several times, MSE's requesters no longer appear.

1.29 Getting Updates

Getting Updates

Updates are currently e-mailed to all registered users as they become available.

The latest versions of the demo, modules and other information can be found in the MSE support area of our website:

http://www.youngmonkey.ca

From the main page follow through the 'hands on' section, through 'Amiga Software' to the 'MIDI SYStem Explorer' support area.

1.30 Locating The Author

Locating The Author

Through INTERNET EMAIL at:

MSE@youngmonkey.ca

Or by MAIL at:

young monkey studios
797 Mitchell Street
Fredericton, NB
CANADA E3B 3S8

If you are reporting bugs, be sure to include the version you are using. The version info can be found by selecting the Project/About menu item.

1.31 Previous versions

Previous versions

- Fixed a memory leak.

- Updated Test/Test.def to include new features.

- Updated Topaz2 and Topaz2p fonts and added 11 point versions.

- Now remembers the last path and module opened. The File requester defaults to these.

- Tooltype 'Last' added.

- All prefs files changed to non-binary format. Delete all old .prefs
  files.

- All prefs files are now created in one prefs directory.

- Now checks to see if a module needs to be recompiled before opening it.

- Fixed a bug in the ReadToolTypes routine that would cause an Enforcer
  hit when run from the CLI.

- Added camd.library support.

- All ToolTypes can now be used as parameters when used from the CLI.

- Implemented prefs/MSE.prefs file for configuration of memory usage.

- Improved MIDI Enable/Disable method and hopefully fixed the crashing
  bug related to this.

- Tooltype 'Module' added.

- Prefs are no longer stored in binary format. Delete the old file
  'MIDI-SYStemEXplorer.prefs' if it exists.

- AutoScrolling screens are now possible when the tooltypes ScreenW
  and/or ScreenH are set larger than the display screen size.

- Updated Test/Test.def to include new features.

- Modules directory is created if it does not exist.

- Fixed bug in WB ToolType processing.

- 'Last' tooltype usage modified.

- If a 'Modules' type Link is selected but the module does not exist, the
  previous module is reopened. Previously, the selection gadget would
  indicate that the selected module was the current one, even though it
  had not been opened.

- Fixed 'Mungwall' FreeMem hit on exit.

- Fixed bug in 'Link Type Modules' processing.

- Added ARexx port.

- Changed Public Screen Name to "MSE" and made it PUBLIC.

- Added 'IRShell.rexx' an interactive rexx shell for testing ARexx
  functions and commands.

- If tooltype 'ModeID' is unavailable MSE falls back to its default ID:
  HIRESLACE_KEY&.

- Added ARexx commands: 'OpenModule' and 'LoadPatch'.

- Documented Menus and 'IR Shell'.

- 'Link Type ARexx' events are no longer run as separate tasks. MSE's
  GUI is disabled until completion.

- You can no longer QUIT if an IR Shell is active.

- Improved string handling which could potentially have caused bugs.

- Fixed bugs which caused strings to become corrupt. This should solve
  the problem of running the Automation demo script numerous times in
  succession. It didn't really have anything to do with the ARexx
  implementation as I had thought. The demo just did things fast enough
  to cause the problem to happen.

- Updated the Automation demo (Test/Test.rexx).

- Checks to see if a module needs to be recompiled before opening it. It
  did this already but gets disabled when beta-testing. Forgot to enable
  it again in last release. This is now handled by the compiler so it's
  not necessary to remember anymore. :)

- 'Link Type Number' now displays "+" in front of values > 0 when
  'MinValue' < 0.

- Fixed slider value display bug under 3.0. 3.0 incorrectly calculates
  the default width of the value text, MSE now overrides this.

- Added Topaz2N/8, a narrow version of Topaz2/8 font.

- Updated Test/Test.def to look better on WB2.x which doesn't handle
  proportional fonts very well.

- Fixed a bug related to the ScreenW and ScreenH tooltypes.

- Added 'PubScreen' tooltype to allow MSE to open on public screens such
  as Workbench.

- Better handles Project icon invocations.

- Now properly locks directory when processing icon tooltypes.

- Fixed bugs in and improved creation of the Rexx port.

```
- Added 'OffsetX' and 'OffsetY' tooltypes for use with public screens.

- Fixed the bug that caused ARexx scripts to not run if Mungwall was
  running.

- Updated documentation.

- Added the camd.library to the distribution archive.

- Guides have icons now.


---------

- Prefs files no longer updated on error.

- Now exits properly if required midi library is not found.

- Updated the Automation demo (Test/Test.rexx).

- Improved rexx argument parsing.

- Added 'Play Notes' under the 'Utilities' menu for sending note on/off
  messages when testing sounds.

- Uses the screen's default font for menus when opened on a public
  screen.

- When dragging the main window on a public screen, all MSE's non-module
  windows are moved.

- Setting of the Handler in the MIDI Configuraton window is now not done
  until the window is closed.  So, cycling through each handler will no
  longer try to open each library.  This was implemented for those of you
  who are not using CycleToMenu.

97.Jul.10
---------

- Added ARexx command: 'SetByte'.  A good example of the use of this is
  the Akai-S612 editor which requires certain non-parameter bytes to be
  set to values calculated from the values of other gadgets.

- Added ARexx commands: 'Busy' and 'SavePatch'.

- Updated Test/Test.rexx to demonstrate the use of calculated values with
  ARexx.

- Properly handles 'Updates Final'.  Performs all display updates and
  internal processing.
```

1.32 New in this version

97.Oct.19

- Now checks to see if module requires more fonts or windows then defined in MSE.prefs before loading. Should also check the other values too, but does not, yet.

- Fixed bug that caused some windows to remain open when they should have been closed.

- Updated documentation.

- New 'useable' demo.