

The first bit of extra functionality that you should add to a simple slider, such as this, is a text field that shows the value of the slider. To do this, create a text field and place it on the Stage. You need to tell the marker sprite where this text sprite is located. An extra parameter should do it. Add a “pTextSprite” property to the property declarations at the top of the behavior. Add this to the *on getPropertyDescriptionList* handler:

```
addProp list, #pTextSprite, [#comment: "Text Sprite",\
    #format: #integer, #default: 0]
```

An “on setText” handler places the current value of the slider in this text field. Just in case no text sprite is used, this handler makes sure that the “pTextSprite” property is not 0. If it is, the “on setText” handler assumes that there is no text sprite and does not place the text:

```
-- this handler sets the text of the text sprite
on setText me
    if pTextSprite <> 0 then -- is there a text sprite?
        sprite(pTextSprite).member.text = string(pValue)
    end if
end
```

Calls to “on setText” need to be added immediately after the calls to “on setShadow” in both the *on beginSprite* and *on exitFrame* handlers.

The next elements that the slider needs are the buttons that enable the users to move the slider one value in either direction. You need both normal and down states for these buttons, as shown previously in Figure 15.8.

These buttons need their own behavior, but before you create the behavior, a handler needs to be added to the marker behavior. This handler enables the marker to be moved one value in either direction. It should take one parameter that is either a #left or a #right depending on which direction the slider should move. It also needs to perform the same boundary check that the “on moveMarker” handler does to make sure that the slider doesn’t go past its limits.

```
-- this handler moves the marker one value left or right
on moveMarkerOne me, direction
    if direction = #left then
        pValue = pValue - 1
    else if direction = #right then
        pValue = pValue + 1
    end if

    -- check to make sure it is within bounds
    if pValue > pMaximumValue then
        pValue = pMaximumValue
    else if pValue < pMinimumValue then
        pValue = pMinimumValue
    end if

    setMarker(me)
    setShadow(me)
    setText(me)
end
```

The “on moveMarkerOne” handler ends by calling the three handlers needed to update the slider. This makes it possible to call this one handler and have the slider changed and updated by the other three automatically.

A handler to take care of the two buttons is similar to any simple button handler. It mostly controls normal and down states for the button sprite. It also has to know where the marker sprite is so that it can send the “moveMarkerOne” message to it. A key parameter is “pArrowDirection”, which is set to #left or #right depending on which button is selected:

```
property pDownMember, pOrigMember -- down and normal states
property pPressed -- whether the sprite is being pressed
property pMarkerSprite -- the number of the marker sprite
property pArrowDirection -- 1 or -1 to add to slider
```

```
on getPropertyDescriptionList me
  list = [:]
  addProp list, #pMarkerSprite, [#comment: "Marker Sprite",\
    #format: #integer, #default: 0]
  addProp list, #pDownMember, [#comment: "Arrow Button Down Member",\
    #format: #bitmap, #default: ""]
  addProp list, #pArrowDirection, [#comment: "Arrow Direction",\
    #format: #symbol, #range: [#left,#right], #default: #right]
  return list
end
```

The rest of the slider button behavior handles the mouse clicking and calls the “moveMarkerOne” handler in the marker behavior once per frame loop while the button is pressed:

```
on beginSprite me
  pOrigMember = sprite(me.spriteNum).member
end

on mouseDown me
  pPressed = TRUE
  sprite(me.spriteNum).member = member pDownMember
end

on mouseUp me
  liftUp(me)
end

on mouseUpOutside me
  liftUp(me)
end

on liftUp me
  pPressed = FALSE
  sprite(me.spriteNum).member = member pOrigMember
end
```

```

on exitFrame me
  if pPressed then
    sendSprite(sprite pMarkerSprite, #moveMarkerOne, \
      pArrowDirection)
  end if
end

```

With eight members and two behaviors, you have all the functionality of the sliders used by other software programs. Even better, you can change the graphics used to anything you want to stylize your sliders to fit your design.

You have a lot of control over how the slider looks and behaves. You can alter your code to place an extra word behind the number in the text field. So, rather than reading “50”, it can read “50%” or “50 widgets”. You can tweak the slider screen boundaries if the shadow sprite isn’t working the way you want. You can even remove the shadow sprite.

There is nothing to stop you from taking all the references to horizontal locations and boundaries and converting them to vertical locations and boundaries to make a vertical slider. This is a good example of how much control Lingo behaviors give you as opposed to the drag-and-drop interface elements in other authoring programs.

→ For an example of a slider, **see** “Creating Volume Controls,” **p. 595**

CREATING PROGRESS BARS

Using some of the same techniques used to make the slider bar, you can create a progress bar behavior. A *progress bar* is a rectangle that enlarges to fill a space as a process is completed. An example is the progress bar the Director displays every time you choose File, Save.

If you have a Lingo process that takes more than a fraction of a second to complete, you might want to display a progress bar so that the users know that the computer is not frozen, but simply processing their requests.

Figure 15.9 shows a simple progress bar. It shows two sprites: a hollow rectangle shape and a filled rectangle. To make the progress bar look like others used in various pieces of software, the fill of the rectangle is set to a pattern, rather than a solid.

Figure 15.9

A simple progress bar showing a task about one-half done.



To create a progress bar, you need a process to test it. Here are two movie handlers that work together to compute all the prime numbers between one and 1,000. (Recall that a prime number is a number divisible only by one and itself.)

```

on findPrimeNumbers
  list = []

```

```

repeat with i = 1 to 1000
  if isPrime(i) then add list, i
end repeat
return list
end

on isPrime n
  repeat with i = 2 to integer(sqrt(n))
    div = float(n)/float(i)
    if div = integer(div) then return FALSE
  end repeat
  return TRUE
end

```

The “on isPrime” function tries every number between two and the square root of the number to see whether it can find a case where there is no fractional remainder to the division. It compares the number to itself converted to an integer to see whether there is a remainder.

This process takes about a second to run on a PowerMac G3 with a 400MHz processor. It also does this calculation in 1,000 steps, so it’s a prime candidate for a progress bar.

Of the two sprites involved in the progress bar, only the filled portion needs a behavior. The other sprite is there for merely cosmetic purposes.

Like the slider marker behavior, this sprite needs to know the size of its final, full rectangle. To make that easy to determine, set the sprite up on the Stage to already be its full size. That way, the *on beginSprite* handler can get the full rectangle size by simply looking at the *rect* of the sprite when the behavior starts.

```

property pFullRect

on beginSprite me
  pFullRect = sprite(me.spriteNum).rect
end

```

The only other handler needed is the one that sets the progress bar when needed. It will be a custom handler that a process, such as “on findPrimeNumbers”, calls when it can.

```

on setProgress me, currentVal, highestVal
  -- get amount filled as a value between 0 and 1
  percentFilled = float(currentVal)/float(highestVal)

  -- convert to a pixel width
  pixelRange = pFullRect.right-pFullRect.left
  x = percentFilled*pixelRange

  -- set the rect of the sprite
  r = rect(pFullRect.left, pFullRect.top, \
    pFullRect.left + x, pFullRect.bottom)
  sprite(me.spriteNum).rect = r
end

```

The “on setProgress” handler takes two parameters. The first is the current value of the progress bar, and the second is the maximum value of the progress bar. It takes these two values and divides them to get the percentage of fill needed. It then determines the physical width of the progress bar area and determines the point to which the sprite should stretch. Finally, it builds a rectangle from this information and sets the sprite.

To use this behavior, you need to call it from the process taking place. In this case, the “on findPrimeNumbers” handler will use it.

```
on findPrimeNumbers
  list = []
  repeat with i = 1 to 1000
    sendSprite(sprite 2,#setProgress,i,1000)
    updateStage
    if isPrime(i) then add list, i
  end repeat
  return list
end
```

Notice that an *updateStage* is needed because the frame is not looping here. Director is caught inside the *repeat* loop and does not know to update the Stage without you telling it.

Check out the sample movie, 15progressbar.dir, which activates the “on findPrimeNumbers” handler with a button, shows the progress bar as the prime numbers are calculated, and then displays the list of prime numbers when it is done.

You can easily change the color and dimensions of a progress bar such as this. If you prefer a more stylized progress bar, you might want to convert this behavior to use three elements: a left end, a right end, and a stretchable middle piece. Doing so would require the same behavior to control two other sprites, possibly specified as parameters.

CREATING GRAPHICAL POP-UP MENUS

Like the check boxes, radio buttons, sliders, and progress bars, another item that can be created with bitmaps and a behavior is the pop-up menu.

Pop-up menus are the little menus that appear in windows and dialog boxes, as opposed to the main menu bar that appears at the top of a screen or window. In the Director authoring environment, you can see examples of these in the Score window and Sprite Inspector. As a matter of fact, just about every window in Director includes some type of pop-up menu.

Pop-up menus are useful when you want the users to choose from a list of items, but don't have the space in your interface to show all the choices. The users click to reveal the list of items, select one, and then the list goes away, leaving their choice in that space.

Creating pop-up menus as interface elements in your movies is possible with a behavior. First, create a series of bitmaps in the Cast to be used. Figure 15.10 shows the Cast window with some example bitmaps. A single bitmap represents the pop-up menu when it is not active, and then five other bitmaps that appear under it when it is pressed. All five of these bitmaps have a corresponding hilite state.

To create the pop-up on the Stage, place the sprites as you would want to see them when the pop-up is pressed and active. Never mind that the list under the “Choose One” graphic should not be present when the user first comes to the frame; the behavior will handle that.

For the behavior to do that, it needs to know where these sprites are. So, one property of this behavior must be a list of sprites. Another property will be the members that are initially in these sprites. The behavior itself will be attached to the pop-up menu button, which is the “Choose One” graphic in this case:

```
property pSpriteList, pMemberList, pPressed

on getPropertyDescriptionList me
    list = [:]
    addProp list, #pSpriteList, [#comment: "Sprite List", \
        #format: #list, #default: []]
    return list
end
```

To start, the behavior needs to record the members used by each of the sprites that are part of the pop-up. This example does not include the pop-up button.

```
on beginSprite me
    pMemberList = [:]
    repeat with i in pSpriteList
        addProp pMemberList, i, sprite(i).member.name
    end repeat
    hidePopup(me)
end
```

The call to “on hidePopup” removes the pop-up items before the user sees them. The best way to do this is to set the *memberNum* property of the sprites to 0:

```
on hidePopup me
    repeat with i = 1 to pSpriteList.count
        sprite(pSpriteList[i]).memberNum = 0
    end repeat
end
```

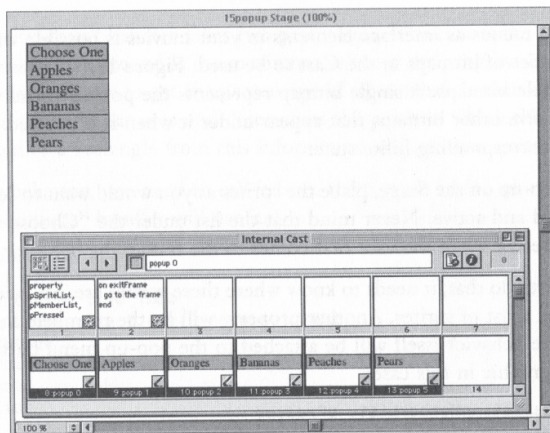
The mouse click handlers look similar to those used in earlier behaviors. Both *on mouseUp* and *on mouseUpOutside* call a custom handler that calls the “on hidePopup” again, as well as an “on select” handler:

```
on mouseDown me
    pPressed = TRUE
end

on mouseUp me
    liftUp(me)
end
```

Figure 15.10

The Stage shows a pop-up menu as it appears during creation of the movie. The Cast window shows these members, as well as the hi-lite states.



```
on mouseUpOutside me
    liftUp(me)
end
```

```
on liftUp me
    pPressed = FALSE
    hidePopup(me)
    select(me)
end
```

In the *on exitFrame* handler, a variety of tasks are performed when the pop-up is pressed. First, it calls “on showPopup”, which returns the sprites to the members they used when you set up the movie. Then, it uses *the rollover* property to determine over which sprite the mouse is currently hovering. It uses *getOne* on the “pSpriteList” property to determine whether this is one of the pop-up items.

Note

The *getOne* property of a list enables you to find the position of an item in a list. If the item is there, it returns the number of the item, which can also be interpreted as *TRUE*. If the item is not in the list, a 0 is returned, which is always seen as *FALSE*.

If the mouse is over a sprite in the list, it then changes the member of that sprite to a member of the same name, except with the word “hilite” appended. This is just a convention that this behavior uses to figure out which highlighted members belong to which normal members. You could just as easily have placed all the highlighted members in the next member over from the normal members, and recorded the member numbers of the sprites, using a +1 to find the highlight member, which will be the very next member in the Cast.

```

on exitFrame me
  if pPressed then
    showPopup(me)
    s = the rollover
    if (pSpriteList.getOne(s)) then
      sprite(s).member = member (pMemberList.getProp(s)&&"hilite")
    end if
  end if
end

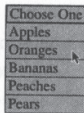
```

The result of this handler is that the pop-up items will be shown if the “pPressed” property is *TRUE*, and any item that the cursor is currently over will show up as the highlighted member.

Figure 15.11 shows this in action.

Figure 15.11

The pop-up menu has been selected and the cursor is over the second item.



The “on showPopup” handler is similar to the “on hidePopup” handler except that the “pMemberList” is used to assign the correct member to each sprite:

```

on showPopup me
  repeat with i in pSpriteList
    sprite(i).member = member pMemberList.getProp(i)
  end repeat
end

```

Finally, the “on select” handler is called when the user releases the mouse button. This handler cannot assume that the cursor is over an item in the pop-up list, so it must check in the same way that the *on exitFrame* handler checks:

```

on select me
  s = the rollover
  if (pSpriteList.getOne(s)) then
    alert pMemberList.getProp(s)
  end if
end

```

In this case, a simple *alert* box appears to signify that a choice has been made. However, in your program you will want to set a global, go to a frame, or perform some other action.

This pop-up menu behavior can be customized in a lot of ways. With a little more work, you can even have the selected item appear to replace the “Choose One” graphic, or have sounds play as the pop-up is used.

Notice that nothing in the script specifies that the items must be positioned directly below the original sprite. Why not have them line up to the right of the sprite? Or, have them appear above it? There are many possibilities.

→ For another way to make pop-up menus, see “Creating Text Pop-Up Menus,” p. 351

TROUBLESHOOTING GRAPHIC INTERFACE ELEMENTS

- Be sure you understand the difference between *the clickLoc* and *the mouseLoc*. The first is the exact location of the mouse when the user clicked. The second is the current location of the mouse, which might have changed since the user clicked.
- When performing calculations in which the result will be a number between 0 and 1, as in the slider behavior or the progress bar behavior, it is important to convert the numbers to floats before doing division. Otherwise, a calculation such as 3/6 will return a 0, rather than a 0.5.
- If you are using floating point numbers in a parameter and assigning a range to that parameter, you can control the number of decimal places used by setting *the floatPrecision* system property. The default is four digits after the decimal point.
- The *memberNum* property of a sprite enables you to set the member of a sprite by just referring to it by number. This property is good for setting a sprite to the next member or something similar, but should not be used otherwise. You should use the member property instead and set the sprite to a specific member by name.

DID YOU KNOW?

- In the *sendSprite* command, the word “sprite” is optional. So, you can write “sendSprite(7,#myHandler)” instead of “sendSprite(sprite 7, #myHandler)”.
- There are two forms of syntax for the keyword “rollover”. The function *rollover(x)* returns a *TRUE* if the cursor is over sprite *x*. The property *the rollover* returns the number of the sprite directly under the cursor.
- The *memberNum* property does not enable you to set the sprite to a member in another cast library. However, you can change the *castLibNum* to do that.
- The obsolete property *castNum* can still be used. It returns the same value as *memberNum* for members in the first cast library, but returns much higher values for members in other cast libraries. For instance, the first member in cast library 2 would have a *castNum* of 131073.

Y-Z

Yak Xtra, 866

year property (date object),
971

Zeus Productions Web site,
866

zLaunch Xtra, 866

Zoom pop-up menu, 33

Zoom tool, 87

zoomBox command, 971

zooming images

- on beginSprite handler,
582-583

- on exitFrame handler, 585

- on mouseDown handler,
583-584

- on mouseUp handler, 585

- on mouseUpOutside
handler, 585

- on mouseWithin handler,
584-585

- Zoom tool (Paint window),
87

zoomWindow handler, 484

zOpen Xtra, 866

zScript Xtra, 867

APPENDIX 1015B

LIBRARY | CATALOG

◀ < 1 of 1 > ▶

BOOK

Using Macromedia Director 8

[Full Record](#)[MARC Tags](#)

Personal name

[Rosenzweig, Gary.](#)

Main title

Using Macromedia Director 8 / Gary Rosenzweig.

Edition

Special ed.

Published/Created

Indianapolis, Ind. : Que, c2000.

[Request this Item](#)[LC Find It](#)

More Information

>

LCCN Permalink

<https://lccn.loc.gov/00100575>

Description

xvii, 1047 p. : ill. ; 23 cm. + 1 computer laser optical disc (4 3/4 in.)

ISBN

0789723344

LC classification

QA76.575 .R684 2000

LC Subjects

<https://catalog.loc.gov/vwebv/search?searchCode=LCCN&searchArg=00100575&searchType=1&permalink=y>

1/2

[Director \(Computer file\)](#)
[Multimedia systems.](#)

Browse by shelf order

[QA76.575](#)

LCCN

00100575

Type of material

Book

Item Availability



CALL NUMBER

[QA76.575 .R684 2000 FT MEADE](#)
Copy 1

Request in

Machine Readable Collections - STORED OFFSITE

Status

Not Charged

CALL NUMBER[QA76.575 .R684 2000](#)
Copy 2

Request in

See Reference Staff. By Appt in Jefferson Main RR (MRC)

Status

c.2 In Process 12-03-2001

LIBRARY | CATALOG

← 1 of 1 →

BOOK

Using Macromedia Director 8

[Full Record](#)[MARC Tags](#)

000	00839cam a22002414a 4500
001	11900104
005	20010531162824.0
008	000131s2000 inua 001 0 eng
906	— a 7 b cbc c orignew d 2 e epcn f 19 g y-gencatlg
925	0_ a acquire b 2 shelf copies x policy default
955	— a pn08 01-31-00; to ASCD pb11 02-21-01;jf00 02/23/01;jf05 (desc) 05/24/01; jf25 to Dewey 05-31-01
010	— a 00100575
020	— a 0789723344
040	— a DLC c DLC
042	— a pcc
050	00 a QA76.575 b .R684 2000
100	1_ a Rosenzweig, Gary.
245	10 a Using Macromedia Director 8 / c Gary Rosenzweig.
250	— a Special ed.
260	— a Indianapolis, Ind. : b Que, c c2000.
300	— a xvii, 1047 p. : b ill. ; c 23 cm. + e 1 computer laser optical disc (4 3/4 in.)
650	_0 a Multimedia systems.
630	00 a Director (Computer file)

[Request this Item](#)[LC Find It](#)

Item Availability

CALL NUMBER [QA76.575 .R684 2000 FT MEADE](#)
Copy 1

Request in Machine Readable Collections - STORED OFFSITE

Status Not Charged

CALL NUMBER [QA76.575 .R684 2000](#)
Copy 2

Request in See Reference Staff. By Appt in Jefferson Main RR (MRC)

Status c.2 In Process 12-03-2001

APPENDIX 1015C



Everything

▼ All Fields ▼

SEARCH

[Advanced Search](#)

[Ask-A-Librarian](#) [MSU Collection](#)



Using Macromedia Director 8 / Gary Rosenzweig.

Personal Author: Rosenzweig, Gary.

Edition: Special ed.

Publication: Indianapolis, Ind. : Que, [2000]

Copyright: ©2000

Publication Date: 2000

ISBN: 9780789723345

Content Type: text

Carrier Type: volume

Language: English

Subject: Multimedia systems.

Director (Computer file)

No. of Holds: 0

Select an Action

Place Hold

Cite This

PermaLink

Staff View

Available:1

Library	Call Number	Format	Item Notes	Status
MSU-Mitchell Memorial Library	QA76.575 .R684 2000	Book	CD in back of book under flyleaf	Circulating Collection

000 am4a0c
003 DLC
005 20010531162824.0
008 000131t20002000inua 001 0 eng
010 00100575
020 0789723344
035 ms 00100575
035 (Sirsi) AHH-5330
039 00 AHH-5330 LCMARC/AZT-3142/ZTAKACS/RSH1/jdg
035 AHH-5330
040 DLC DLC
042 pcc
049 MFMM
050 00 QA76.575 .R684 2000
100 1 Rosenzweig, Gary.
245 10 Using Macromedia Director 8 / Gary Rosenzweig.
250 Special ed.
264 1 Indianapolis, Ind. : Que, [2000]
264 4 ©2000
300 xvii, 1047 pages : illustrations ; 23 cm + 1 computer laser optical disc (4 3/4 in.)
336 text
337 unmediated
338 volume
596 MSU-MAIN
630 00 Director (Computer file)
650 0 Multimedia systems.
945 MARS
945 RDA ENRICHED

APPENDIX 1015D

libraries.org

A directory of libraries throughout the world

- [Home »](#)
- [Libraries »](#)
- [Guides](#)
- [Documents](#)
- [Vendors »](#)
- [Products »](#)
- [News](#)
- [Procurement](#)
- [Member Login »](#)

found 5 items where the item contains **Mississippi state university**. Showing item 1 of 5.

☐ list ☐ next >> search:

Mississippi State University Libraries

[Mississippi State University](#)

Address: 395 Hardy Road
Mail: PO Box 5408
[Mississippi State, Mississippi](#)
39762-5408
United States

County: [Oktibbeha](#)

Phone: 662-325-7668

Connect to: [Library Web Site](#) [Online Catalog](#)

Library details: Mississippi State University Libraries is an **Academic** library.

This library is affiliated with [Mississippi State University](#) ([view map](#)). The collection of the library contains **2,358,501** volumes. The library circulates **50,136** items per year. The library serves a population of **18,304** FTE Enrollment.

Other Info: Member of the Association of Southeastern Research Libraries (ASERL)

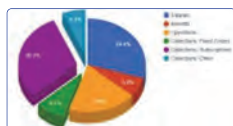
Permalink: <https://librarytechnology.org/library/1151>
(Use this link to refer back to this listing.)

Administration: The director of the library is **Frances Coleman**.

Organizational structure: The library is part of a **publicly-funded** organization. It operates on a **Non-profit** financial model.

Academic Level: The library Serves an educational institution at the **University** level. The Carnegie Foundation designates this institution as: **Doctoral Universities: Very High Research Activity** (level 15).

Statistics Academic		
Enrollment	18,304	FTE Students
Collection size	2,358,501	volumes
Annual Circulation	50,136	transactions



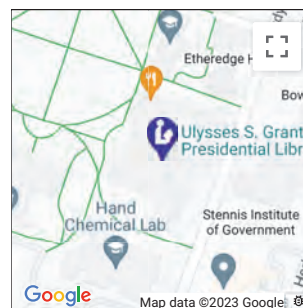
[View Detailed Statistics](#)

Wireless: The library offers **wireless** access to the Internet.

[Technology Profile](#)



Mississippi State University Libraries



Related Libraries

- Other libraries with Carnegie class: **Doctoral Universities: Very High Research Activity**
- Other members of [Mississippi Library Partnership](#)
- Libraries located in [Mississippi State](#) (Mississippi)
- Libraries located in [Oktibbeha](#) county (Mississippi)
- [View map of libraries in Oktibbeha County](#)
- [all Academic libs in Mississippi](#)

About libraries.org

libraries.org is a free directory of libraries throughout the world.

This directory is a component of [Library Technology Guides](#).

Please [contact Marshall Breeding](#) to report corrections about information listed for this

	Product Name	Year Contracted
Current Automation System	Symphony	2002
Previous Automation System	DRA	1993
Previous Automation System	CLSI	1986
Discovery Service (w/index)	EBSCO Discovery Service	2010
OpenURL Link resolver	LinkSource	2010
Previous Link resolver	360 Link	2005
Electronic Resource Management	EBSCO ERM Essentials	
Institutional Repository	DSpace	
Digital Asset Management	CONTENTdm	
(* Older versions of SirsiDynix Symphony were called Unicorn)		
The library participates in the shared automation system provided by the Mississippi Library Partnership (view map)		

[View Privacy and Security Report](#)

Identifiers	
libraries.org ID	1151
OCLC Symbol	MFM
WorldCat Registry ID	5204
NCES LIBID	176080

Record History: This listing was created on Jun 15, 1997 and was last modified on Jan 6, 2023.

Updates: Corrections or Updates? Registered members of Library Technology Guides can submit updates to library listings in libraries.org. [Registration](#) is free and easy. Already registered? [login](#). Or, you can report corrections just by sending a message to [Marshall Breeding](#).

[libraries.org](#) [Update this Entry](#) [Library Technology Guides](#)

Maintained by [Marshall Breeding](#)

[terms of service](#)
Copyright 1994-2022

library.

Follow Marshall on [Twitter](#).

← → ↻ https://web.archive.org/web/20010405184544/http://library.msstate.edu:80/collections/galaxy.asp 150% ☆

http://library.msstate.edu/collections/galaxy.asp Go FEB APR JUN 05 2000 2001 2002 About this capture

54 captures
17 Oct 2000 – 3 Feb 2013

MISSISSIPPI STATE UNIVERSITY LIBRARIES

Galaxy ONLINE CATALOG MITCHELL MEMORIAL LIBRARY ARCHITECTURE BRANCH LIBRARY COLLEGE OF VETERINARY MEDICINE BRANCH LIBRARY

Sections
About the Libraries
Collections
Of special interest
Online resources
Services

Shortcuts
Online catalog
Indexes/databases
Ask a Librarian
Calendar of events
Departments
Home page
Hours
John Grisham Room
Online request forms
Site index
Workshops/seminars
Print version 
myLibrary

Search the online catalog for titles held in the Mississippi State University Libraries.

- [Search Mitchell Memorial Library holdings](#)
- [Search Architecture Branch Library holdings](#)
- [Search College of Veterinary Medicine Branch Library holdings](#)
- [Telnet to the text version of the catalog of collections.](#) (best for checking journal listings)
[username=libserv]
- [Information about Electronic Reserves](#)

The Wayback Machine - <https://web.archive.org/web/20010405184544/http://library.msstate.edu:80/colle...>



Sections

About the Libraries
Collections
Of special interest
Online resources
Services

Shortcuts

Online catalog
Indexes/databases
Ask a Librarian
Calendar of events
Departments
Home page
Hours
John Grisham Room
Online request forms
Site index
Workshops/seminars

Print version 

myLibrary

login

Search the online catalog for titles held in the Mississippi State University Libraries.

- [Search Mitchell Memorial Library holdings](#)
- [Search Architecture Branch Library holdings](#)
- [Search College of Veterinary Medicine Branch Library holdings](#)
- [Telnet to the text version of the catalog of collections.](#) (best for checking journal listings)
[username=libserv]
- [Information about Electronic Reserves](#)



The Library provides copying services for a small fee.



APPENDIX 1015E

Copyright

United States Copyright Office

Try the [Copyright Public Records System \(CPRS\)](#) pilot with enhanced search features and filters.

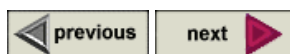
[Help](#)[Search](#)[History](#)[Titles](#)[Start Over](#)

Public Catalog

Copyright Catalog (1978 to present)

Search Request: Left Anchored Title = Using Macromedia Director

Search Results: Displaying 1 of 2 entries

[Labeled View](#)

Using Macromedia Director 8 / Gary Rosenzweig.

Type of Work: Text

Registration Number / Date: TX0005349335 / 2000-07-26

Application Title: Special edition Using Macromedia Director 8

Title: Using Macromedia Director 8 / Gary Rosenzweig.

Edition: Special ed.

Imprint: Indianapolis : Que, c2000.

Description: 1047 + CD-ROM.

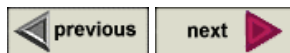
Copyright Claimant: Que Corporation

Date of Creation: 2000

Date of Publication: 2000-07-19

Copyright Note: C.O. correspondence.

Names: [Rosenzweig, Gary](#)
[Que Corporation](#)



Save, Print and Email ([Help Page](#))

Select Download Format Full Record

Enter your email address:

[Help](#) [Search](#) [History](#) [Titles](#) [Start Over](#)

[Contact Us](#) | [Request Copies](#) | [Get a Search Estimate](#) | [Frequently Asked Questions \(FAQs\) about Copyright](#) | [Copyright Office Home Page](#) | [Library of Congress Home Page](#)

https://cocatalog.loc.gov/cgi-bin/Pwebrecon.cgi?v1=1&ti=1,1&Search_Arg=Using Macromedia Director&Search_Code=TALL&CNT=25&PID=-z6hgBGJsINmQ9c... 1/1

APPENDIX 1015F

Citations to Rosenzweig

Source Document

Rosenzweig, G., (2000, January), *Using Macromedia Director 8*, Special Edition, Indianapolis, Ind.: Que.

Early Citations: 2001 (in chronological order)

Sod, D. M. (2001, May). *Design and Implementation of a Modular Preschool Multimedia Program* (Doctoral dissertation, University of Cincinnati).

Bean, J. C., Groves, J. F., Kansari, N., Appleyard, M., Lehmbeck, C., Wayne, T., & Brittingham, M. (2001, June). The creation of microelectronics-based visualizations to enhance science education and literacy. In *Proceedings of the Fourteenth Biennial University/Government/Industry Microelectronics Symposium (Cat. No. 01CH37197)* (pp. 110-113). IEEE.

Signalness, J. (2001, July). *From the Honors Program Grand Forks, North Dakota July, 2001* (Doctoral dissertation, University of North Dakota).

Ilya Jehan, M. (2001). *Nombor & objek pintar/Ilya Jehan Mazlan* (Doctoral dissertation, University of Malaya).