

Working With Lists in Director

Introduction: What is a List?

One of the many capabilities of the powerful multimedia authoring tool Director is the ability to create a list. What is a list? A list is a type of variable. Director uses variables to store single pieces of information, but a list can store many pieces of information at once. For example if you wanted to store a student's last name, first name, student id number, and telephone number, you could use five individual variables or one list. Director treats the list and all its components as a single unit, and each piece of information on the list retains its individual identity. Items in a list may also be modified; items can be added or deleted, and the list still retains its identity. For example you could add the student's address to the list or change the student's telephone number, and the list would still be considered the same list.

Why Use a List?

Lists are a very efficient means of storing and accessing information. Large, overwhelming amounts of information can be organized and managed using a list. Lists can even contain other lists. For instance, you could create a database using one list of user information storing usernames, passwords, contact information, scores, etc, by using at least one sub-list for each user. Lists can also be used to track the steps of a user through a program and record the navigational sequence. This can help in scripting back and forward buttons so that the user can navigate the program with ease. Another common use for lists is to group multiple cast members or sprites together so that scripting in the program becomes easier. For instance, it is much easier to write a script that puts one list into a field rather than writing a script to put several individual cast members or sprites into that field. Lists are powerful tools that provide means to gain easy access to and convenient management of data. Overall the advantage of using a list in Director is efficiency; lists provide a workable organizational structure for pieces of information. Director can create two types of lists—linear lists and property lists.

Linear lists

Linear lists are comprised of single units of data that are linked in some way. The following are examples of linear lists

```
myFriends = ["Alan", "Nicole", "Teresa", "Brian"]
```

```
ingredients = ["sugar", "flour", "eggs", "vanilla", "butter"]
```

```
lessonsCompleted = [3, 4, 6, 7, 10, 11, 16, 20, 8, 22]
```

Lists are defined by using square brackets to hold the information. When the information is a text string rather than a number, it is placed in quotation marks. Each item on the list is separated by commas. Notice that the commas are placed after the quotation marks.

Working with Linear Lists—Capabilities and Lingo

Linear lists are particularly good for keeping track of order. They easily provide information about the position of the item in a list. Items can be added, deleted, replaced, and sorted. The position of each item can also be determined using lingo commands, and lingo commands will also identify the position of an item in a list. This becomes important when a program is being designed to keep track of sequence for the purpose of navigation. Positioning may also be important if orders are being placed and requests are processed in the order that they are received. The following is an example of how and why order may be tracked.

If a student is completing tutorial lessons you may want to keep track of which ones he has completed and in what order they were completed, a linear list would suit your purposes.

You want to start with an empty list to which the lessons will be added once they are completed. To begin a new list, name the list and define it as empty by typing an empty set of brackets.

```
lessonsCompleted = []
```

To add a lesson use the **add** command
add list, item

```
add lessonsCompleted, 3  
put lessonsCompleted  
-- [3]
```

Eventually your user has completed many lessons and you want to find out which ones and in what order.

```
lessonsCompleted = [3, 4, 6, 7, 10, 11, 16, 20, 8, 22]
```

You want to know which lesson the student completed third.

The **getaProp** command tells you the value of the item that appears in the position you name.

getaProp (list, position)

```
put getaProp(lessonsCompleted,3)  
-- 6
```

This says that lesson 6 was the third lesson completed.
You may also want to know the lesson that the student completed last.
The **getLast** command will give you that information.

getLast (list)

```
put getLast(lessonsCompleted)
-- 22
```

This shows that lesson 22 was the last lesson completed.

Next you want to know at what point the student completed lesson 16.
The **getPos** command tells you the position that an item holds in the list.

getPos (list, item)

```
put getPos(lessonsCompleted, 16)
-- 7
```

This shows that lesson 16 was the seventh lesson completed.

Linear lists can also sort information. If the order the lessons were completed is irrelevant, you may want to sort the list numerically by using the **sort** command.

sort list

```
sort lessonsCompleted
put lessonsCompleted
-- [3, 4, 6, 7, 8, 10, 11, 16, 20, 22]
```

Once the lesson has been checked by the instructor, you wish to delete it from the list.
The **deleteOne** command would be used here.

deleteOnce list, item

```
deleteOne lessonsCompleted, 3
put lessonsCompleted
-- [4, 6, 7, 8, 10, 11, 16, 20, 22]
```

For a more complete list of linear list lingo, please see the chart on page six of this paper.

While linear lists are useful when you know the position, what if the position is not known, or not important? What if the lessons above were Director lessons and you wanted to know if any of the lessons completed were on animation or on scripting? A linear list cannot tell you this information. What if you also wanted to store information regarding scores on the lessons, the date each was completed, and the subject of each? A linear list will not suffice when this much data must be compiled and organized. One must then use a property list.

Property Lists

Property lists differ from linear lists in that the information contained in the list can be organized by category and they can contain other lists within the list. Property lists contain pairs of information that set a property and a value. The property designates the category and the value is the specific information associated with it. The following are sample property lists.

```
ingredients = [#sugar: "1 c.", #flour:"1 c.", #eggs:2, #vanilla: "1/2  
tsp. ", #butter: "4 tbsp."]
```

```
multimedia = [#authoring:"Director", #sound:"Peak", #video"Premiere",  
#distribution:"Shockwave"]
```

```
lessonsCompleted = [#linearAnimation:3, #transitions:4, #filmLoop:6,  
#keyframes:7, #behaviors:10, #shockwave:11, #markers:16, #navigation:20,  
#video:8, #lingo:22]
```

Properties are defined using symbols. A symbol may only contain words, numbers, and the underscore. It is designated as a symbol by placing the pound sign in front of it. A symbol differs from a variable in that it does not store a value, it is merely a way of recalling a value. In the case of a property list, it serves as a tag or label.

Working with Property Lists—Capabilities and Lingo

Property lists may be used like linear lists in that items may be added, deleted, and replaced. They may also be used to retrieve information about position. However, now there is an additional piece of information so that items in the list may be retrieved by category instead of position in the list.

To add an item to the list, you may use the add command as in the linear list, but there is the danger of duplicating a property. You may use the **setaProp** command to avoid duplicating properties. For instance you could decide your recipe needs salt.

setaProp(list, property, newValue)

```
setaProp(ingredients, #salt, "1/4 tsp.")  
put ingredients  
-- [#sugar: "1 c.", #flour: "1 c.", #eggs: 2, #vanilla: "1/2 tsp. ",  
#butter: "4 tbsp.", #salt: "1/4 tsp."]
```

You may replace a property using the **setProp** command to replace the value of a property that already exists. For instance if you decide your recipe needs more sugar.

setProp(list, property, newValue)

```
setProp(ingredients, #sugar, "2 c.")  
put ingredients  
-- [#sugar: "2 c.", #flour: "1 c.", #eggs: 2, #vanilla: "1/2 tsp. ",  
#butter: "4 tbsp.", #salt: "1/4 tsp."]
```

If you wish to retrieve the value of a property, use either the **getProp** or **getProp** commands. For example you may want to know how many eggs are required by the recipe.

getProp(list, property)

```
put getProp(ingredients, #eggs)
-- 2
```

For a more complete list of linear list lingo, please see the chart on page six of this document.

Lists Within Lists

One of the main advantages of using lists is that they can contain other lists. For instance if a student completed multiple lessons on the same topic in director, the list might look like this

```
-- [#linearAnimation: [2, 3], #transitions: 4, #filmLoop: 6, #keyframes:
7, #behaviors: [9, 10, 12], #Shockwave: 11, #markers: 16, #navigation:
20, #video: 8, #lingo: [22, 23, 24]]
```

Notice how the properties linearAnimation, behaviors, and lingo indicate that several lessons have been completed on these topics. These sub-lists are only linear lists, though. Property lists may be sub-lists as well. The list studentInfo provides an example of how a list can organize and track several piece of info per student on the list.

```
studentInfo=[#Freddy:[#idNum:123456789, #year:2004,
#grades:[#assign1:80, #assingn2:84, #assign3:90]],
#Penny:[#idNum:987654321, #year:2003, #grades:[#assign1:86, #assign2:90,
#assign3:91]]]
```

This example shows only two properties in the list studentInfo (Freddy and Penny), but the values of those properties are other lists that provide info on these two students. This capability of incorporating several lists makes it possible to organize large amounts of data.

List Lingo

The chart below contains the most typical list lingo commands. This chart is a resource from the site <http://www.director-online.com/accessArticle.cfm?id=293>

Linear List (*item1, item2, ...*)

To: And you Use the command:
know
the:

Add

add an item at end of an unsorted list or in order in a sorted list	--	add <i>list, item</i>
add an item at end regardless of whether the list is sorted	--	append <i>list, item</i>
add an item in a specific position	position	addAt <i>list, position, item</i>

Replace

replace an item	position	setAt <i>list, position, newItem</i>
-----------------	----------	--------------------------------------

Delete

delete an item	item	deleteOne <i>list, item</i>
delete an item	position	deleteAt <i>list, position</i> deleteProp <i>list, position</i>
delete all items	--	deleteAll <i>list</i>

Get

get an item	position	getaProp (<i>list, position</i>) getAt (<i>list, position</i>) <i>list[position]</i>
get the last item	--	getLast (<i>list</i>)
get the greatest item	--	max (<i>list</i>)

get the least item	--	min (<i>list</i>)
get a position	item	getOne (<i>list, item</i>) getPos (<i>list, item</i>)
get the position of the closest match	item	findPosNear (<i>list, item</i>) ¹

Sort

sort the list	--	sort <i>list</i>
---------------	----	------------------

¹ The list must be sorted to use this command.

Property List (*property1:value1, property2:value2, ...*)

(The word *item* refers to a *property:value* pair.)

To: **And you** **Use the command:**
know
the:

Add

add an item at end of an unsorted list or in order in a sorted list	--	<code>addProp list, property, value</code>
---	----	--

Replace

replace a value	property	<code>setaProp list, property, newValue²</code> <code>list[property] = newValue²</code> <code>setProp list, property, newValue²</code> <code>list.property = newValue^{2,4}</code>
replace a value	position	<code>setAt list, position, newValue</code> <code>list[position] = newValue</code>

Delete

delete an item	property	<code>deleteProp list, property</code>
delete an item	value	<code>deleteOne list, value</code>
delete an item	position	<code>deleteAt list, position</code>
delete all items	--	<code>deleteAll list</code>

Get

get a property	value	<code>getOne (list, value)</code>
get a property	position	<code>getPropAt (list, position)</code>
get a value	property	<code>getaProp (list, property)³</code> <code>list[property]³</code> <code>getProp (list, property)³</code> <code>list.property^{3,4}</code>
get a value	position	<code>getAt (list, position)</code> <code>list[position]</code>
get the last value	--	<code>getLast (list)</code>
get the greatest value	--	<code>max (list)</code>
get the least value	--	<code>min (list)</code>
get a position	property	<code>findPos (list, property)</code>
get a position	value	<code>getPos (list, value)</code>
get the position of	property	<code>findPosNear (list, property)⁵</code>

the closest match

Sort

sort the list -- sort *list*

² *setaProp* and the [] syntax add the item if the property doesn't already exist. *setProp* and the . (dot) syntax return an error if the property doesn't exist.

³ *getaProp* and the [] syntax do not return an error if the property is not found. *getProp* and the . (dot) syntax return an error if the property is not found.

⁴ In the property, don't use the # character if this is part of your property name.

⁵ The list must be sorted to use this command.

Summary

Lists are powerful tools for storing and accessing information. They can collect and sort large amounts of information. Linear lists contain values that make it convenient to track position and sequence items. Property lists contain a property:value pair using a symbol as a property to act as an identification tag for the value being stored. Some of the most common uses of lists include tracking navigation, making scripting easier, and creating databases. What your list includes and how you use it is up to you.

References

1. *Director 8 Demystified* by Phil Gross and Jason Roberts
Information regarding this book can be found at <http://www.demystified.com>
2. "Scripting examples for working with lists in Director 7 and later" from Macromedia
http://www.macromedia.com/support/director/ts/documents/d7_list_syntax_examples.htm
3. "Why Use Lists" by Judi Taugher
<http://www.director-online.com/accessArticle.cfm?id=293>
4. "Using Property Lists" by Zac Belado
<http://www.director-online.com/accessArticle.cfm?id=820>
5. "Why and How to Use Linear Lists"

<http://www.macromedia.com/support/director/how/expert/lists/lists.html/lists02.html>

6. "Director FAQ" from Director Web. Select option "12. Lists"

<http://www.mcli.dist.maricopa.edu/director/faq/index.htm>

7. "Tips and Scripts" from Director Web. Select options "Converting Lists to Fields," "Making Copies of Lists,"

<http://www.mcli.dist.maricopa.edu/director/tips.html>

8. "Director 5 List Commands: Syntax and Usage" provides a chart with typical list lingo commands.

<http://www.updatestage.com/idlt05.html>

9. "The Power of Lists" provides a tutorial in working with lists.

<http://www.fbe.unsw.edu.au/learning/Director/Scripting/tut16.htm>

10. Lingo Lounge: "Using Director with Databases" by Gary Rosenweig Parts 1-4
These articles provide a tutorial in creating an html database using Director.

Part 1: <http://www.director-online/access/Article.cfm?id=953>

Part 2: <http://www.director-online/access/Article.cfm?id=956>

Part 3: <http://www.director-online/access/Article.cfm?id=959>

Part 4: <http://www.director-online/access/Article.cfm?id=965>

11. Director Web provides a host of resources and links related to lists and pretty much any Director function.

<http://www.mcli.dist.maricopa.edu/director/>

This paper is written by Jody Seelinger for the course EDC385G Multimedia Authoring
at the University of Texas—Austin.