Drill Designer 1.0: User's Manual

June 1991

The User's Manual and the accompanying Macintosh software were created in Fall 1990 by members of the Instructional Software Development Group (ISDG) at Weeg Computing Center, The University Of Iowa, in a collaborative effort with:

Stephen M. Alessi, Associate Professor,
Psychological And Quantitative Foundations

James P. Pusack, Associate Professor, German

Sue K. Otto, Director, Language Media Center

Members of ISDG are:

Stephen W. Wessels, Manager Software Development
Les J. Finken, Systems Analyst
Steven N. Bowers, Systems Analyst
Tom W. Mentz, Programmer/Analyst
Joe Halloum, Programmer/Analyst
Derek Siebert, Programmer/Analyst
Patti P. Santangelo, Programmer/Analyst

Participating ISDG members were:

Co-Project Leaders: Bowers, Finken
External Specifications: Bowers, Finken, Siebert
Internal Specifications: Siebert, Bowers
Implementation: Siebert
Testing: Siebert, Santangelo
Documentation: Siebert, Bowers

Credit for the instructional design theory upon which Drill Designer is based goes to Stephen Alessi and Stanley Trollip (refer to *Computer-Based Instruction: Methods and Development*, Prentice-Hall, 1991).

Further, it should be mentioned that another software package exists that is based upon these same instructional design theories. The 1985 publication *Drill Shell* (CONDUIT, The University Of Iowa) is a completely separate and distinct package for the Apple IIe containing importable BASIC source code pieces that aid drill development in Apple BASIC. This package provided much assistance in clarifying issues during the early design phases.

Table of Contents

Introduction to Drill Designer

Drill Designer is a set of commands and functions that allow easy and flexible drill handling for quizzing and testing applications. It facilitates the creation of a list of questions (or items) and the subsequent modification of that list based on correct and incorrect responses from a student or user. When these commands are included in a HyperCard stack, an Authorware course, or a C program, they remove the "busywork" of drill making and add features that can automatically provide for extra help on the items that are missed most often, thereby increasing the effectiveness of the drill.

The **future queue** is the list of questions to be given. It does not contain the question itself, only a question number that the author indexes with his/her list of questions. Initially, the future queue is created from the **long list**, a list of all the questions that can be presented. Additional questions can then be drawn from this list if desired. When a student successfully answers all occurrences of a given question in the future queue, that question is added to the **retired items list**.

Drill Designer uses "Variable Interval Performance Queueing" to provide additional practice on questions answered incorrectly. Basically this means that when a question is missed, all occurrences of this question are removed from the future queue. The question is then reinserted at positions chosen by the author. Generally, an author arranges these positions so that the distance between positions increases; thus, the question may be presented again fairly soon, then a bit later, and then quite a bit later. In this way the student will remember the answer in short-term memory and is gradually forced to commit it to long-term memory.

As the future queue gets smaller, it may not be long enough to allow the insertions to be made at the positions specified. In these instances, Drill Designer uses several methods to lengthen the future queue or modify the insert positions to allow them to fit. These vary depending on the **insertion technique** chosen by the author.

Drill authors should have a good working knowledge of HyperCard, Authorware, or C, whichever will be used to produce drills. To use Drill Designer effectively, authors should also have a good idea of how the Drill Designer commands operate. The following pages give a thorough presentation of Drill Designer and its abilities. It is suggested that authors read at least through the Basic Commands section before attempting to write an application that uses Drill Designer.

Hardware and Software Requirements

Hardware Requirements:

Macintosh Plus or higher

Drill Designer will not work with early Macintoshes that do not contain the 128K ROM version of the file manager.

Software Requirements:

HyperCard (version 1.2.5 or later)
or Authorware Professional (1.5.2 or later version)
or Think C (4.0 or later)

The version numbers above are recommendations. It is probable that Drill Designer will work on some earlier versions, but it is on those above that it has been tested.

Contents of the Distribution Disk

The distribution disk contains the following programs/files:

Drill XCMD is the file that contains the XCMD resource that can be used in a HyperCard stack or an Authorware course.

Drill Designer C.lib is the library that is used in your Think C project.
Drill Designer.h is the header file that is #included in the first line of your C program.

Inside the folder **Demo Programs**:
DD HyperCard Test is a test HyperCard stack.
DD Authorware Test is a test Authorware course.
These programs can be used to experiment with various drill formations and see exactly how queues are affected by various commands. They are also useful as a examples of how Drill Designer is incorporated into each of the formats.

Anatomy HC is a sample drill application in HyperCard.

Inside the folder **Recycle C Folder**:
Recycle C is a Think C application that uses Drill Designer to present a drill on recycling and the environment.
Recycle C.c is the source code for Recycle C drill; it is included to show how Drill Designer is used in a C program.
Recycle C Project is the Think C project for the Recycle C application. This shows how the Drill Designer C library is used for creating Think C projects.

Installing Drill Designer

To use the Drill Designer routines, they must be included in your C program or the Drill XCMD must be added to your HyperCard stack or Authorware course. The following describes how this is done in each case:

In HyperCard:

Using ResEdit (or any other application that can manipulate Macintosh resources), copy the XCMD "Drill" from the file "Drill XCMD" into the resource fork of your stack. Note: if the XCMD "Drill" is copied into the resource fork of HyperCard itself, then Drill Designer can be used by any of your stacks.

In Authorware:

From inside Authorware, open your course, look under *Variables* and select *Load Function*. Then use the dialog box to select the file "Drill XCMD", and to select the XCMD itself. Be sure to re-save your course after this or the XCMD will not be saved.

Drill Designer can only be used by the courses into which it is loaded.

In C:

As the first line of your program, use the following line exactly as it appears:

#include "Drill Designer.h"

Also, the library file **Drill Designer C.lib** must be added to your Think C project (along with the ANSI library). See the Think C manual for instructions on how to add libraries to your project.

Drill Author's Responsibilities

Drill authors are responsible for setting up two very important variables: the data handle and the path name. If the data handle variable is not declared, Drill Designer will not function at all. If the path name is not set, the Store and Restore functions will not be able to locate where to store and retrieve files and will use the current working directory.

The Data Handle

The data handle tells Drill Designer where to find information that it needs in order to execute. In HyperCard and Authorware, it must be declared as a global character variable and set to "0" (zero); in C, it is declared as a variable of type global_Ptr, and set to NULL.

In HyperCard:

Include this exact syntax in your stack (in the script of the stack, in your handler for the *openStack* message):

```
global DataHandle
put "0" into DataHandle
```

In Authorware:

Open your course and select *Variables* in the menu bar. Select *New* and a box should pop up. Type in DataHandle as the variable name and select *character* as the variable type. Then type in "0" (a zero) as the initial value.

In C:

Declare the following in your section of global variables:

```
global_ptr      data_Ptr = NULL;
```

In this case, you don't need to use the exact name "data_Ptr"; something shorter may be better, since you must pass this variable to every Drill Designer call from C. In the command syntax and examples contained in later pages, this variable is referred to as data_Ptr; simply substitute the name you have used if it is different.

global_ptr is a type declared in Drill Designer.h; this name must appear exactly the same in your code.

The Path Name

The path name is declared in a way similar to the data handle declaration. The explanation of <<path>> is on the following page.

In HyperCard:

Include the following in your stack's script (inside the *openStack* handler):

```
global PathName
put "<<path>>" into PathName
```

Or, use the following syntax instead to put your stored files in the same folder as your stack, making it portable to other volumes (diskettes, hard disks, etc.).

```
global PathName
put word 2 of the long name of me into PathName
delete char 1 of PathName
repeat until char length (PathName) of PathName = ":"
        delete char length (PathName) of PathName
end repeat
```

In Authorware:

Select *Variables*, then *New* as before with DataHandle. Type in PathName as the variable name and make it a character variable. Then for initial value, type in <<path>> as described on the next page.

Note: Setting the PathName equal to the Authorware system variable FileLocation (PathName := FileLocation) in an equation box will direct file operations to Authorware's defaults, thus making it portable to other volumes.

In C:

Put this declaration in your code (global area):

```
char * PATH_NAME = "<<path>>";
```

Or, use this to save files to the default volume:

```
/*  Declare these variables outside of main(), i.e., as global variables */
        char path_Buffer[256];
        char *PATH_NAME = (char *) path_Buffer;
        int     dummyInt;
/* Put the following section in the first lines of your main() code */
        GetVol ((StringPtr) PATH_NAME, &dummyInt);
        PtoCstr ( (char *) PATH_NAME);
```

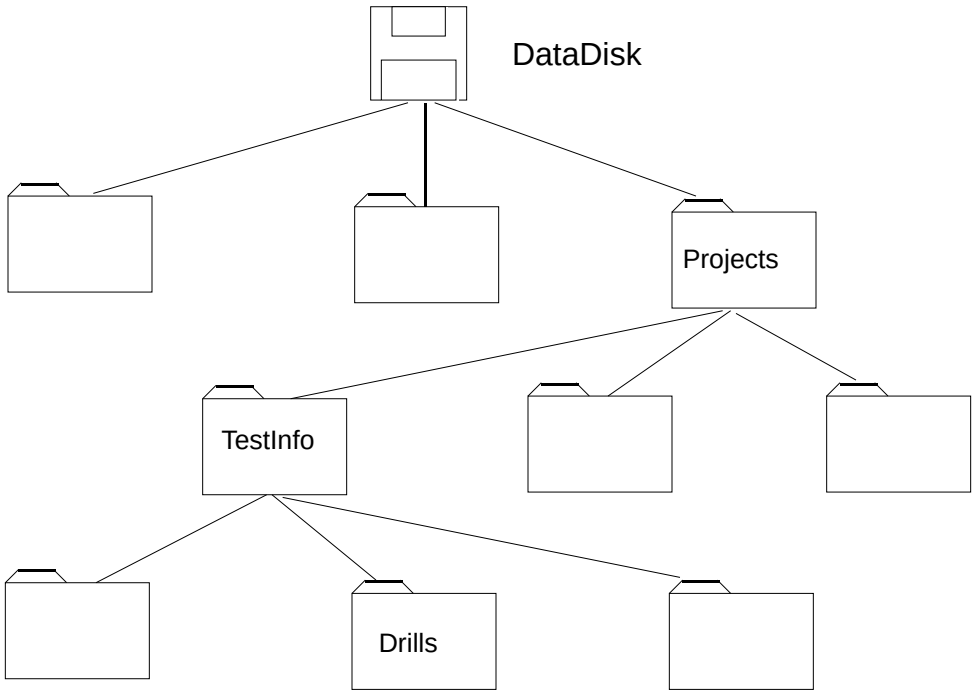
<<path>>

<<path>> is the full pathname where you want files saved to and restored from. This should consist of at least a volume name, and additionally any directories (folders) along the way to (and including) the folder you want the data stored into. When Drill files are stored, everything is contained in a "DrillData" folder; when you specify your path, you are specifying the complete path to the folder that will contain the DrillData folder after the Store command, and the folder in which Drill Designer looks for a DrillData folder when calling Restore (see Store, Restore and appendix B).

As per Macintosh specifications, directories in a full pathname are separated by colons (:). The directory name should be typed exactly as it appears on your display in order to be recognized.

A sample pathname appears below.

DataDisk:Projects:TestInfo:Drills



In this example, the "DrillData" folder will be located inside the folder called "Drills" above. See appendix B for information on the "DrillData" folder.

Simple Drill Program Flow

The commands CreateDrill, Dispose, GetErrorMessage, Next, Correct and Incorrect form the core of Drill Designer; with these six commands an author can create and process drills with much flexibility. Below is a general idea of how a program might flow using these commands to create and utilize a drill.

—*CreateDrill* is the command that sets everything up; thus it is usually the first Drill Designer command used in a drill application. In this call the author specifies many parameters used in the creation and upkeep of the drill.

—*GetErrorMessage* is used often while testing a drill application during its development. Following the CreateDrill call (and any subsequent calls) the result of GetErrorMessage can be tested and displayed by the author to see the error messages and warnings that result from the previous call. This will help the author avoid common pitfalls while making a drill application.

Then, the flow typically becomes a loop:

—*Next* returns the question number of the question to be given next. The author can use this number as an index to the actual written question, and then allow the student to respond in whatever way the author is having users respond. When *Next* returns 0, there are no more questions left in the queue.

The author determines whether the user provided the right answer, and depending on this, calls either:

—*Correct*, which handles correct responses by removing the front item from the queue (thereby readying the next question to be retrieved by *Next*).

—*Incorrect*, which handles incorrect responses by removing the item completely from the queue and reinserting it at the positions specified in the *CreateDrill* call. Again, the subsequent call to *Next* will provide the next question number to be used.

—*Dispose* frees the memory used by Drill Designer. When the future queue is exhausted, or the user wants to quit (or for any other reason the author sees fit to end the session), the author calls Dispose.

Notes on Parameter Formats

The following is a description of the parameter types shown in the syntax portion of the command descriptions. In each call, where parameters are necessary, a descriptor will be shown in the location where a parameter would be included. This descriptor is given a name that briefly summarizes its function and also tells what type of parameter is expected. The last segment of the descriptor name will be one of the following:

num: An integer or a variable containing an integer. In HyperCard and Authorware this can also be a string containing an integer.

str: A string or a variable containing a string. Strings are surrounded by double quotes ("This is a string"). If this is put into a C variable, it should be of type char *.

char: A single character or a variable containing a single character. In C, a single character is surrounded by single quotes, ('A', 'B', etc.). In HyperCard or Authorware, this simply means a string of length one, surrounded by double quotes ("A", "B", etc.) like any other string.

list: This specifies a certain type of string, a string of integers separated by spaces. For example, "3 6 9 12" and "17" are both lists, the latter being a list of one element. Again, in C, if this is passed by a variable, it should be of type char *.

When a command returns a value, included in the syntax listing is a generic method of storing the value. For example, in the HyperCard syntaxes, the phrase "put the result into YourStr" is actual HyperTalk script; *the result* is always the place where HyperCard stores what the XCMD returns. The other variable names do not have to be followed exactly. In the previous example, *YourStr* could be any variable name you want to use.

After the command syntax, each descriptor's meaning is explained as to its significance to the command. Some parameters have default values that are used in case of a bad or missing parameter; the default value, if there is one, is enclosed in brackets.

Following this is a listing and explanation of any variable parameters necessary (these are additional parameters needed for certain instances of a given call; usually there are none). When these are present, they follow the standard parameters in order and are used the same way syntactically.

CreateDrill

Syntax:

HyperCard:

Drill "CreateDrill", FQnum, REPnum, QPnum, ORDchar, TECHstr, INClist

Authorware:

Drill ("CreateDrill",FQnum, REPnum, QPnum, ORDchar, TECHstr, INClist)

C:

CreateDrill (**&**data_Ptr, FQnum, REPnum, QPnum, ORDchar, TECHstr, INClist);
/* note that in C the *address* of data_Ptr must be passed */

FQnum: the number of different questions to be in the future queue initially. [default = QPnum (if valid) or 20]

REPnum: the number of times to present the initial future queue (the queue is repeated based on this number) [1]

QPnum: the question pool size; this is the total number of questions in the drill. This should be greater than or equal to FQnum. [FQnum (if valid) or 20]

ORDchar: Determines the order (orientation) of the future queue and long list: sequential (S) or random (R). [S]

TECHstr: The insertion technique used for calls to Incorrect. Choices are: DII, RDII, RDIns, RDInt, RFII, and EC. [RFII]

INClist: The list of the insert positions used for calls to Incorrect. ["3 7 17"] (default in XCMD version only)

All parameters above must be included from C but are optional from HyperCard/Authorware (defaults are used).

Variable Parameters: Depending on the string passed in TECHstr, additional parameters will be necessary as follows:

TECHstr	extra parameters
RFII	(none)
DII, RDInt	SMLnum
RDIns	FEWnum
RDII	SMnum, FEWnum
EC	RETnum, RPLlist

SMLnum: The smallest insertion position allowed [default = 1]

FEWnum: The fewest insertions allowed [1]

RETnum: The number of items to retire before replenishing; cannot be greater than FQnum [1]

RPLlist: The list of insert positions to use when replenishing (see Correct) ["3 7 17"] (default in XCMD version only)

CreateDrill is used to create a drill from scratch; it sets up the initial future queue and long list and clears everything else to zero (or its default).

A call to either CreateDrill or Restore must be made before other operations can take place. When finished with a drill, Dispose should be called to clear memory.

CreateDrill first builds the long list using QPnum (question pool size) for the length. If ORDchar is "S", the list is sequentially ordered; with "R" (random), the list is randomized; it still contains the same numbers, though in a random order.

The future queue is constructed by pulling items from the long list (the amount specified in FQnum). These items are removed from the long list, leaving it shorter (and often empty; much of the time QPnum is chosen to be equal to FQnum). FQnum can be 0 (zero), which may be useful if an author wants to build a future queue completely from scratch using the Drill Modification Commands.

The future queue is then repeated based on REPnum. A value of 1 means the future queue is left as it is; 2 means it is doubled (i.e. repeated once), etc. Thus REPnum specifies the number of times each question exists in the future queue at the start of the drill.

TECHstr defines the insertion technique to be used: Diminishing Insertions and Intervals (DII), Resurrection with Diminishing Insertions and Intervals (RDII), Resurrection with Diminishing Insertions (RDIns), Resurrection with Diminishing Intervals (RDInt), Resurrection with Fixed Insertions and Intervals (RFII) or Endless Continuum (EC). Based on this parameter, others may be needed as shown on page 12. More detail about the insertion techniques and these parameters is contained under the description of the *Incorrect* command.

Unless the Endless Continuum (EC) insertion technique is being used, the long list is not normally a factor in the drill and in most cases FQnum and QPnum are equivalent. It is possible, though, to perform many operations on the long list regardless of the insertion technique being used; an author may want to start with a larger question pool while using DII, for example, and use the Drill Modification Commands to have control over replenishing.

Note that ORDchar (list order or orientation) affects the long list; the future queue is affected because the long list is used as the source for the initial future queue. If "R" is specified with QPnum at 40 and FQnum at 20, the initial future queue will not contain numbers 1 thru 20 in random order, but rather 20 numbers from 1 to 40 in a random order. To achieve the former, QPnum would have to be set to 20.

Examples:

HyperCard:

Drill "CreateDrill", 20, 2, 20, "S", "RFII", "3 7 17"

Drill "CreateDrill", 20, 2, 20, "S", "DII", "5 10 15", 5

Note the final parameter, the smallest allowed insert position.

Drill "CreateDrill", 20, 2, 20, "R", "RDInt", "2 8", 2

Note the final parameter (smallest position) and the random orientation

Authorware:

Drill ("CreateDrill", 20, 2, 20, "S", "RFII", "3 7 17")

Drill ("CreateDrill", 20, 2, 20, "S", "RDIns","1 4 8", 2)

Note the final parameter, the fewest allowable insertions

Drill ("CreateDrill",20,1,20,"S","RDII", "5 8 12", 4, 2)

Note the last two parameters, the smallest insert position allowed and the fewest insertions allowed

C:

CreateDrill (&data_Ptr, 20, 2, 20, 'S', "RFII", "3 7 17")

Note the single quotes surrounding the S (but the double quotes remain around the list.)

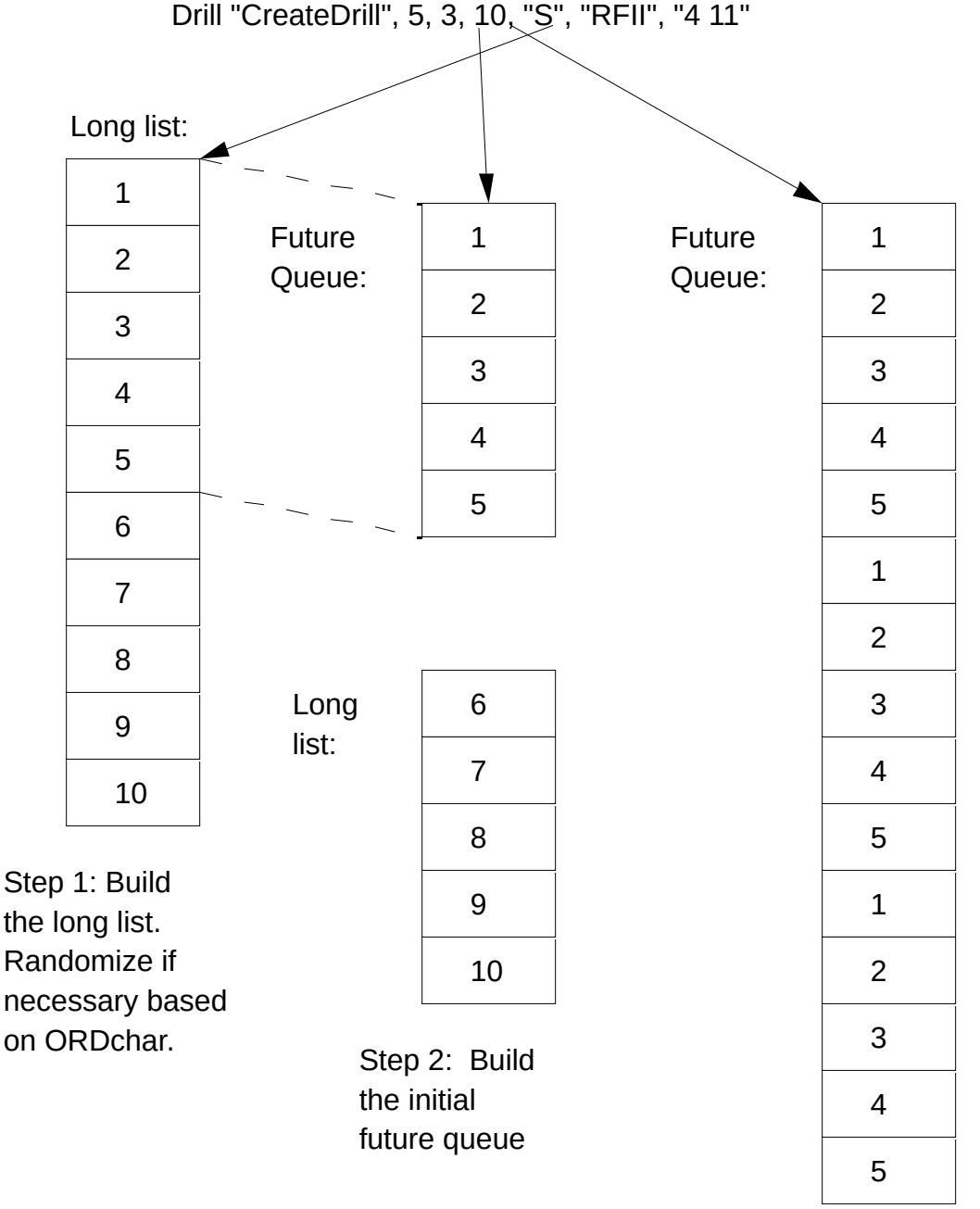
CreateDrill(&data_Ptr,20,2,80, 'S',"EC","3 7 17", 3, "4 8");

Note the final two parameters, the number to retire before replenishing and the positions to insert when replenishing. Note also the larger question pool size

Drill Creation Diagram

Example (HyperCard format):

Drill "CreateDrill", 5, 3, 10, "S", "RFII", "4 11"



Step 1: Build the long list. Randomize if necessary based on ORDchar.

Step 2: Build the initial future queue

Step 3: Repeat the future queue as per REPnum.

Dispose

Syntax:

HyperCard: Drill "Dispose"
Authorware: Drill ("Dispose")
C: Dispose (**&**data_Ptr);

Dispose frees the memory used by Drill Designer, and thus should be called before the author's application is exited. Failure to do so may result in memory fragmentation or storage problems. Dispose also sets the Authorware and HyperCard global DataHandle back to zero, or when in C, it sets data_Ptr to NULL.

Note that when used from C, the author must pass the address of data_Ptr (precede with the "&" character) to the Dispose function.

GetErrorMessage

Syntax:

```
HyperCard:      Drill "GetErrorMessage"
                put the result into YourStr
Authorware:  YourStr := Drill ("GetErrorMessage")
C:              YourStr = GetErrorMessage (data_Ptr);
```

GetErrorMessage returns a string containing the error messages that were generated by the previous Drill Designer call. Some error messages mean that the previous command could not be executed, while others are warnings signifying that the command was executed, but certain parameters were not specified appropriately so defaults were used. GetErrorMessage is very useful for the author while creating a drill program, course or stack to insure that all commands are recognized and that the parameters are appropriate.

When there is no error or warning from the previous call, GetErrorMessage returns a null string, and testing for a null string can tell whether there is any message. Remember that there are many messages that do not mean that the command has failed. See Appendix A for further information.

When used from HyperCard or Authorware, the maximum returned string length is 255; if the error message exceeds this length (which is unlikely if not impossible) it will be truncated.

When used from C, YourStr should be of type *char* * , not a character array .

Appendix A contains a list of all error messages and what they signify.

Next

Syntax:

HyperCard: Drill "Next"
put the result into YourNum

Authorware: YourNum := Drill ("Next")

C: YourNum = Next (data_Ptr);

A call to Next returns the question number of the first item in the future queue. This does not change the future queue in any way; repeated calls to Next would yield the same result. Next is used by the author to determine the next question to present to the student.

If the future queue is empty, Next returns a zero.

Correct

Syntax:

HyperCard: Drill "Correct"
Authorware: Drill ("Correct")
C: Correct (data_Ptr);

Correct is called by the author when the student correctly answers a question. That item is then removed from the head of the future queue. If there are no more occurrences of the item left in the future queue, the item is *retired*, i.e., it is placed into the retired items list.

It is possible that an item to be retired already exists in the retired items list; in this case, the item's up-to-date data is simply copied into the corresponding item in the retired items list. The position of the item in the retired items list does not change. There are never multiple occurrences of an item in the retired items list.

When the endless continuum (EC) insertion technique is being used and enough items have been retired, *replenishing* occurs. Replenishing refers to removing items from the front of the long list and inserting them into the future queue (at the positions specified by RPLlist (replenish positions) in the CreateDrill call). The number of items that must be retired to trigger replenishing is specified in the CreateDrill call (RETnum) and the count of items retired is reset to zero after each replenish.

Note that the replenish positions list (RPLlist) is a list of positions in which to insert a group of items (RETnum may be larger than one). For example, if five items are to be inserted into positions 3 and 6, then all five items will be inserted after position two, and then all five items will be inserted after position five. However, position five will have moved due to the first insertions. Therefore, the result of this will be that the items exist at positions 3, 4, 5, 6 and 7 and also at positions 10, 11, 12, 13 and 14. If the five items are taken as a "group" of one, then they are indeed at positions 3 and 6.

Incorrect

Syntax:

HyperCard: Drill "Incorrect"
Authorware: Drill ("Incorrect")
C: Incorrect (data_Ptr);

Incorrect is called when the student answers a question incorrectly. All occurrences of the item are removed from the future queue and then reinserted at the positions specified in the CreateDrill call (INClist).

Often the future queue is not long enough to accommodate the insertions at the positions specified. For example, if the specified positions are 3, 7 and 17, and the future queue contains only 13 elements, no insertion can be made at position 17. When this occurs, Drill Designer uses various methods to lengthen the future queue and/or decrease the specified positions in order to allow the insertions to be made. These methods vary depending on the "insertion technique" specified in the CreateDrill call:

—Resurrection (used by RDII, RDIns, RDInt, RFII and EC): This refers to the lengthening of the future queue by adding items from the retired items list to the end of the future queue. Items are resurrected in reverse order, meaning that items more recently retired (and therefore generally more difficult) are returned into the queue before the easier, earlier-retired items.

—Diminishing Intervals (used by DII, RDII and RDInt): This method seeks to reduce the size of the insert positions so that the insertions can be made. For example if the insert positions are 3, 7 and 17 and the length of the future queue is 10, then the 17 might be reduced to allow the insertions. This method cannot make a position lower than the "smallest position" (SMLnum) which is defined during CreateDrill.

—Diminishing Insertions (used by DII, RDII and RDIns): In this method the number of insert positions is reduced. For example if the insert positions are 3, 7 and 17 and the length of the future queue is 10, then the 17 might be deleted to allow the insertions. The number of insertions can not be less than "fewest positions" (FEWnum) defined during CreateDrill.

—Appending (used by RDII, RDInt and RDIns): This refers to the use of non-retired items to lengthen the future queue. In this case items are copied from the front of the future queue and padded to the end.

The insert techniques operate as follows:

—DII (Diminishing Insertions and Intervals):
Because this technique includes no methods for lengthening the future queue, it comes with a special case: if the length of the future queue is too small to allow the first insertion, then one insertion is made at the end of the future queue and the algorithm ends. Otherwise, this technique uses Diminishing Intervals and Diminishing Insertions (but it *does not* check "fewest positions" before deleting an insert position).

—RDII (Resurrection with Diminishing Insertions and Intervals):
This technique uses Resurrection, Diminishing Intervals, Diminishing Insertions, and Appending.

—RDIns (Resurrection with Diminishing Insertions):
This technique uses Resurrection, Diminishing Insertions and Appending.

—RDInt (Resurrection with Diminishing Intervals):
This technique uses Resurrection, Diminishing Intervals and Appending.

—RFII (Resurrection with Fixed Insertions and Intervals):
This technique uses Resurrection and Appending until the insertions can be made in the specified positions.

—EC (Endless Continuum):
This technique works the same way as RFII above. The difference in this technique comes during calls to Correct where it replenishes from the long list (see Correct).

Store

Syntax:

HyperCard: Drill "Store", STUstr, DRLstr, AUTHstr
Authorware: Drill ("Store", STUstr, DRLstr, AUTHstr)
C: Store (data_Ptr, STUstr, DRLstr, AUTHstr)

STUstr: the student ID (<= 20 characters) [def. = null string]
DRLstr: the drill ID (<= 20 characters) [null string]
AUTHstr: the author variables, a string containing anything else the author wants to store to disk (this can be up to 255 characters). This parameter is optional from HyperCard and Authorware and from C, a null string ("") can be passed.

The Store command creates two files:

1) A performance file, containing information on how well the student did in the most recent session and on all sessions combined. This file lists in a short report format the number of questions answered, the number correct, and the percentage as well as the amount of time spent in the drill.

2) A state file, containing all the necessary information to allow the state to be restored (see Restore) to the point at which it was saved.

All files relating to a certain drill (specified by the drill ID) will be contained in a folder called DrillData_*DRLstr*(i.e. the word "DrillData_" with the drill ID appended to it). The Store command looks for this folder using the path name specified by the PathName global (in HyperCard and Authorware) or the PATH_NAME string in C (see Drill Author's Responsibilities). If this folder does not exist, Store will create it.

Inside the DrillData folder, Store will create the performance file, entitled *STUstr*.PER (the student ID that is passed, appended with ".PER"). Store also looks inside the DrillData folder for the states folder (and creates it if necessary), named States_*DRLstr* ("States_" appended with the drill ID passed). In this folder, Store creates the state file (*STUstr*.STA, the passed student ID with ".STA" appended).

There may be several student performance files inside a single DrillData folder, each of these students having taken the same drill. Each student would also have a corresponding state file inside the States folder (see appendix B).

Store writes over old files; therefore, if Student X is using Drill Y and Store is called, it will destroy any previous files generated by the same student using the same drill. Usually this makes no difference; if Restore was used, the old data is retained in cumulative form. However, authors should make duplicates of any stored files they wish to retain indefinitely.

Both the performance and the state files are standard text files. The state file should not be modified or an error could occur during the Restore command.

The format of the author variables string is up to the author; it can be any string, but most likely would be a list of numbers that the author has concatenated together (to be parsed after a call to Restore).

Examples:

HyperCard:

```
Drill "Store", "ST32", "Drill1", "23 13 7 late"
```

This command will store the drill under the name Drill1 with a student ID of ST32. The final string is the author variables string, which can contain anything and should have meaning to the drill author.

Authorware: To store the drill as Chem 101 with student ID 007, and with an author variables string containing the date the assignment was given, perhaps:

```
Drill ("Store", "007", "Chem 101", "1/31/91")
```

C:

```
Store (data_Ptr, "324-49-2316", "R3", "");
```

If no string of author variables is to be used from C, a null string must be specified.

Restore

Syntax:

HyperCard: Drill "Restore", STUstr, DRLstr
put the result into YourStr

Authorware YourStr:= Drill("Restore", STUstr, DRLstr)

C: YourStr = Restore (**&**data_Ptr, STUstr, DRLstr);

STUstr: the student ID (up to 20 characters) [def. = null string]
DRLstr: the drill ID (up to 20 characters) [null string]

Restore takes the place of the CreateDrill call and restores the state of the previously stored drill specified by the student ID and drill ID. It returns the string containing the author variables that were stored with the Store call (this can be up to 255 characters). This string must be converted back into whatever format the author's original information was in (Often the string is treated as a list of numbers and is parsed accordingly).

Restore uses the path specified by the PathName global (with HyperCard and Authorware) or by the PATH_NAME char * from C. There it looks for the DrillData folder named DrillData_*DRLstr*, inside which it looks for the States_*DRLstr* folder. There it locates the file *STUstr*.STA and reads the information.

C Notes: YourStr should be a char *, not an array. Also, it is important that the *address* of data_Ptr is passed (i.e., it must be preceded by the "&" character).

Insert

Syntax:

HyperCard: Drill "Insert", Qnum, INSlist, LSTchar
Authorware: Drill ("Insert", Qnum, INSlist, LSTchar)
C: Insert (data_Ptr, Qnum, INSlist, LSTchar);

Qnum: the question number to be inserted
INSlist: the positions at which to insert the question
LSTchar: Identifies list to insert into (F=future queue, L= long list; optional from HyperCard/Authorware) [default = F]

Insert allows the author to modify the future queue or the long list by inserting a question at the specified positions. To make the insertions correctly, INSlist should be specified in ascending order (e.g., "2 4 8"); otherwise, as insertions are made, previous insertions will be "pushed forward" and end up in the wrong place. Note that the insertions are made exactly where specified and no methods used in "Incorrect" are used; if the list isn't long enough, the insertions are made at the end.

Examples:

HyperCard: To insert item 5 into the long list at positions 3, 6 and 9:

Drill "Insert", 5, "3 6 9", "L"

Authorware: To insert item 15 into the future queue at positions 5 and 100 (note that if the future queue is less than 100 elements long the insertion will be made at the end).

Drill ("Insert", 15, "5 100")

C: Insert (data_Ptr, 10, "9 6 3", 'F');

The above example would insert item 10 into the Future Queue, but the resulting positions would be 11, 7 and 3 (the earlier insertions are pushed forward by the lower insertions).

Purge

Syntax:

HyperCard: Drill "Purge", PURlist, LSTchar
Authorware: Drill ("Purge", PURlist, LSTchar)
C: Purge (data_Ptr, PURlist, LSTchar);

PURlist: The list of items to purge **or** the list of positions to eliminate. If the first character in the string is a P, then the list will be treated as a list of positions.
LSTchar: The list to operate on (F= future queue, L= long list; optional from HyperCard/Authorware) [default = F]

Purge can be used in two ways. If a normal list of integers is passed as PURlist, then Purge will delete those items completely from the list specified by LSTchar. Those items will no longer be present in the list; no data associated with them (times answered, times missed) is retained, and the items will not be placed in the retired items list.

If a list of integers preceded by a "P" (P is the first character in the string) is passed, then Purge will delete the contents of those *positions* in the list.

Examples:

HyperCard: Starting with a future queue = 2,4,6,8,10,12,14,16,18,20:

Drill "Purge", "P 3 6 9"

The resultant future queue will be: 2,4,8,10,14,16,20 (positions 3,6 and 9 gone).

Authorware:

Drill ("Purge", "12 10", "L")

This will delete all occurrences of items 10 and 12 from the long list (the order of the items is not important here, but when deleting positions it is.)

C:

Purge (data_Ptr, "P1 2 3 4 5", "F");

Deletes the first five future queue positions.

Requeue

Syntax:

HyperCard: Drill "Requeue", Qnum, REQlist, LSTchar
Authorware: Drill ("Requeue", Qnum, REQlist, LSTchar)
C: Requeue (data_Ptr, Qnum, REQlist, LSTchar);

Qnum: the question number to requeue
REQlist: the positions at which to re-insert the question
LSTchar: The list to be operated on (L = long list, F = future queue; optional from HyperCard/Authorware) [def.= F]

Requeue removes an item from a list and re-inserts it into that list at specified positions. Requeue can be thought of as a combination of Purge and Insert; it first purges an item from the list, then inserts it. In fact, Requeue will act exactly like Insert if the item specified is not in the list already. Requeue cannot be used to purge items, however; if it is unable a perform the insertions due to an invalid list, it will not perform the deletion.

Performing a Requeue does not change any of the data associated with an item, such as times missed and times answered.

Examples:

HyperCard: To remove item 10 from the future queue and re-insert it at positions 2,4,6 and 8:

```
Drill "Requeue", 10, "2 4 6 8", "F"
```

Authorware:

```
Drill ("Requeue", 333, "5 10", "L")
```

This operation will insert item 333 into the long list at positions 5 and 10, even if the item is not initially in the "question pool size" range (this will generate a warning in the next error message).

C: To make sure item 5 exists in the long list only at position 7:

```
Requeue (data_Ptr, 5, "7", 'L');
```

Retire

Syntax:

HyperCard: Drill "Retire", Qnum
Authorware: Drill ("Retire", Qnum)
C: Retire (data_Ptr, Qnum);

Qnum: The question number to retire

Retire removes all occurrences of an item from the future queue, then retires the item either by adding it to the retired items list, or if it already exists there, by updating the data associated with the item in the retired items list.

If the item is not present in the future queue, the Retire call does nothing.

Note that the Retire operation works only on the future queue; an item cannot be retired from the long list.

Examples:

HyperCard: To retire item 10:

Drill "Retire", 10

Authorware: Begin with retired items list = 1,2,3,4,5
and future queue = 3,7,10,4,7,8

Drill ("Retire",7)

This results in a retired items list of 1,2,3,4,5,7
and a future queue of 3,10,4,8

C: Assume question 5 is in both the retired items list and the future queue.

Retire (data_Ptr, 5);

This removes question 5 from the future queue; it remains in the same place in the retired items list, with its performance data updated.

Pad

Syntax:

HyperCard: Drill "Pad", PADlist, LSTchar
Authorware: Drill ("Pad", PADlist, LSTchar)
C: Pad (data_Ptr, PADlist, LSTchar);

PADlist: The list of items to pad.
LSTchar: The target of the pad operation (F = future queue, L = long list; optional from HyperCard/Authorware) [def.= F]

Pad adds the items in PADlist to the end of the list designated by LSTchar. If an item in PADlist is non-existent, i.e., the item number is larger than the question pool size specified in CreateDrill, Pad will still add the item but a warning will appear in the next call to GetErrorMessage (see appendix A).

Pad preserves the order of the items as they are passed to it. The rest of the list is unaffected by the addition of the new items. Performance data (times missed, times answered) stored with each item is kept consistent in cases where an item Padded to the end already exists in the list.

Examples:

HyperCard: To add items 2, 4 and 6 to the end of the future queue:

```
Drill "Pad", "2 4 6", "F"
```

Authorware: To add items 2, 4 and 6 to the end of the long list:

```
Drill ("Pad", "2 4 6", "L")
```

C: To add items 16, 3, 6 and 2 to the end of the future queue:

```
Pad (data_Ptr, "16 3 6 2", 'F');
```

RetrieveInfo

Syntax:

HyperCard: Drill "RetrieveInfo", INFOnum
put the result into YourNum

Authorware: YourNum := Drill ("RetrieveInfo", INFOnum)

C: YourNum = RetrieveInfo (data_Ptr, INFOnum);
/* In C, YourNum should be of type *long* */

INFOnum: An integer specifying the information requested.

Variable parameters: Depending on the value of *INFOnum*, additional parameters will be necessary as follows:

INFOnum	extra parameters
1-3, 10-15, 19-20	(none)
4-9	Qnum
16-18, 22	Pnum
21	HLLnum, FRMchar

Qnum: Specifies the item number for these requests
Pnum: Specifies the list position for these requests
HLLnum: Specifies how long the hardest items list is to be.
FRMchar: Determines how the hardest items list is to be constructed.
[default = #]

RetrieveInfo returns information requested by the author. The first number passed (INFOnum) indicates the information returned, as follows:

1) The length of the future queue is returned.
2) The length of the long list is returned.
3) The length of the retired items list is returned.
4) Returns the number of times an item (specified by Qnum) is in the future queue.
5) Returns the number of times an item (Qnum) is in the long list.
6) Returns the number of times an item (Qnum) is in the retired items list (this is either 0 or 1).
7) Returns the number of times an item (Qnum) has been missed.
8) Returns the number of times an item (Qnum) has been given.
9) Returns the percentage of times an item (Qnum) has been missed (rounds to nearest integer).

10) Returns the number of questions the user has answered for all sessions of the drill.
11) Returns the number of questions the user has answered for the current session only.
12) Returns the number of questions the user has answered correctly for all sessions of the drill.
13) Returns the number of questions the user has answered correctly for the current session only.
14) Returns the percentage of questions that the user has answered correctly for all sessions of the drill.
15) Returns the percentage of question that the user has answered correctly for the current session only.
16) Returns the item in position Pnum of the future queue.
17) Returns the item in position Pnum of the long list.
18) Returns the item in position Pnum of the retired items list.
19) Returns the time (in seconds) that the user has spent in the drill for all sessions. In C, this result should not be stored in a short integer as it may be too large.
20) Returns the time (in seconds) that the user has spent in the drill for the current session. In C, this result should not be stored in a short integer as it may be too large.
21) This option constructs a list of the hardest questions. The length of the list is specified by HLLnum. The format (method of construction) is specified by FRMchar; this can be either "#", which means the list is constructed by the number of times the question is missed, or "%", which signifies that the list is constructed based on percentage of times missed. In both cases, ties are referred to the other method; i.e., if "%" is used and two questions have been missed every time, one missed 10 out of 10 times will be listed before one missed 5 out of 5 times. If a question has not been missed, it will not be put into the hardest items list. Thus the resultant list will not contain HLLnum items if fewer questions have been missed; the returned value equals the length of the list that was constructed by the call.
22) Returns the item in position Pnum of the hardest items list. It will return 0 if the position doesn't exist or the hardest items list has not been created. Usually, a call is made to 21 immediately before this one to ensure that the list is correct; if calls have been made to Correct, Incorrect, Purge, etc. since the last call to 21 the hardest items list may no longer be up-to-date.

Examples:

HyperCard:

```
Drill "RetrieveInfo", 1
put the result into YourNum
```

YourNum now contains the length of the future queue.

```
Drill "RetrieveInfo", 17, 5
put the result into YourNum
```

YourNum now contains the 5th item in the long list

```
Drill "RetrieveInfo", 21, 10, "#"
put the result into YourNum
```

This constructs a list of the items that have been missed the most (length = YourNum, which is 10 or less).

Authorware:

```
YourNum := Drill ("RetrieveInfo", 21, 15, "%")
```

Constructs a list of the items that have been missed the highest percentage of times (the length of the list is 15 or less).

```
YourNum := Drill ("RetrieveInfo", 22, 5)
```

YourNum now contains the 5th item in the hardest items list

C:

```
YourNum = RetrieveInfo (data_Ptr, 8);
```

YourNum now contains the number of times that question eight has been answered.

```
YourNum = RetrieveInfo (data_Ptr, 12);
```

YourNum now contains the total number of questions the user has answered correctly for all sessions of the drill.

Appendix A: Error Messages

The following is an alphabetical list and explanation of all error messages that Drill Designer can return to GetErrorMessage. The text explains which calls can yield these results, whether the command was executed (or continued from the point of error), and lists situations that may have caused the error itself.

Message:	Occurs in:	Executed?
Author variables list shortened	Store	Yes

Explanation: The string that contains the additional information that the author wants stored in the state file cannot be greater than 255 characters, or it will be truncated.

Message:	Occurs in:	Executed?
Bad path: default volume used	Store, Restore	Yes

Explanation: Either the volume in PathName (see Drill Author's Responsibilities) or one of the subdirectories specified was not found; if the volume was unrecognized, the default volume was used. If a subdirectory was not found but the volume was valid, then the DrillData folder is placed in that volume.

Message:	Occurs in:	Executed?
Can't make directory	Store	Yes

Explanation: After the DrillData or the States folder (directory) was not found, Store attempted to create the directory and could not; this may be due to a disk error or a lack of disk storage space.

Message:	Occurs in:	Executed?
Can't open state file	Restore	No

Explanation: Store attempted to read from a state file that doesn't exist; the name may have been incorrectly specified, the file might not be in the path specified by PathName, or it could be a disk error.

Message:	Occurs in:	Executed?
Can't write performance file	Store	???

Explanation: During a Store, the performance file could not be created. This may be due to a disk error or a lack of disk storage space. The state file might still be successfully written, however.

Message:	Occurs in:	Executed?
Can't write state file	Store	???

Explanation: During a Store, the state file could not be created. This may be due to a disk error or a lack of disk storage space. The performance file might have been successfully written, however.

Message:	Occurs in:	Executed?
COMMAND NOT EXECUTED	(HyperCard, Authorware)	No

Explanation: The command passed in the Drill call was not executed due to a shortage of passed parameters. From HyperCard and Authorware, some commands (CreateDrill, Store), will execute given no other parameters, setting defaults where necessary. All others require every non-optional parameter. See "Not enough parameters passed" and "Defaults set".

Message:	Occurs in:	Executed?
Defaults set	(HyperCard,Authorware)	Yes

Explanation: Like "COMMAND NOT EXECUTED", this command occurs when there are not enough parameters specified. In this case, however, the command was executed after defaults were set where necessary. See "COMMAND NOT EXECUTED" and "Not enough parameters passed".

Message:	Occurs in:	Executed?
DrillData folder not found[: created]	Restore, Store	Yes

Explanation: The DrillData folder was not found at the PathName specified. In Store, an attempt was made to create the folder. When using Store for the first time for a drill, this message will appear naturally since the folder will not exist until the first Store. However in Restore, and in subsequent Store calls, this signifies that the DrillData folder is missing, or it isn't in the correct path, or that the path itself or the drill name was incorrectly specified.

Message:	Occurs in:	Executed?
Drill ID shortened	Store, Restore	Yes

Explanation: The string passed as the drill ID (DRLstr) can be no longer than 20 characters; if it is longer, it is truncated.

Message:	Occurs in:	Executed?
Fewest insertions invalid, set to 1	CreateDrill	Yes

Explanation: The fewest insertions (passed as *FEWnum* when using the "RDII" and "RDIns" insertion techniques) cannot be less than zero nor greater than the number of insertions passed in *INClist* (the insert positions for Incorrect calls).

Message:	Occurs in:	Executed?
Future Queue empty	Correct, Incorrect	No

Explanation: Correct and Incorrect cannot function if the future queue is empty, since there is no first item to specify as correct or incorrect.

Message:	Occurs in:	Executed?
Hardest Items List form invalid, default set	RetrieveInfo(21)	Yes

Explanation: The character passed to specify the format of the hardest items list must be a "#" or a "%" character. The default is "#".

Message:	Occurs in:	Executed?
Hardest Items List position invalid	RetrieveInfo(22)	Yes

Explanation: The position specified does not exist for the hardest items list. Either the position was not a positive value or it was greater than the length of the hardest items list. A zero is returned by the function in this case.

Message:	Occurs in:	Executed?
Hardest Items List size invalid, default set	RetrieveInfo(21)	Yes

Explanation: The length specified for the hardest items list is either less than one or greater than the maximum integer size allowed. The default is 1.

Message:	Occurs in:	Executed?
Insert technique string invalid, set to RFII	CreateDrill	Yes

Explanation: The string that contains the insertion technique to be used could not be identified as one of the valid options.

Message:	Occurs in:	Executed?
Invalid item number	Retire,Insert,Requeue,RetrieveInfo	No

Explanation: The item number specified is either less than 1 or is greater than the maximum allowed integer size.

Message:	Occurs in:	Executed?
Invalid target list char, default set	Requeue,Purge,Insert,Pad	Yes

Explanation: The character that specifies the list to perform these operations on must be specified with an F or an L (F is the default).

Message:	Occurs in:	Executed?
Item larger than question pool size	Requeue, Insert, Pad	Yes

Explanation: The item specified to use in these actions is larger than the initial size of the long list; this means the item does not exist in the "question pool" as defined in CreateDrill, thus this warning is issued.

Message:	Occurs in:	Executed?
Item not in queue	Retire	No

Explanation: The item to be retired was not present in the future queue and thus cannot be retired.

Message:	Occurs in:	Executed?
List unusable for []	CreateDrill,Requeue,Purge,Insert,Pad	???

Explanation: A string that contains a list yielded unusable data. For the latter four commands above, the message contains the routine name at the end, and the command is not executed. For CreateDrill, the name of the unusable list (either "Incorrect" or "Replenish") follows, but the CreateDrill command is executed. An unusable list results from a null string ("") being passed, or any string that contains no numbers, such as "this is an invalid list string."

Message:	Occurs in:	Executed?
No insert positions list, can't execute	Requeue, Insert	No

Explanation: The insert positions list was invalid; Insert will not make any insertions, and Requeue will neither make insertions nor remove the item from the specified list in the first place (this prevents Requeue from purging an item accidentally).

Message:	Occurs in:	Executed?
Not enough parameters passed	HyperCard,Authorware	???

Explanation: Fewer parameters were passed than are specified in the command listings (omitting optional parameters will not generate this, but omitting necessary variable parameters will). After this message will be either "Defaults set", signifying that the command was executed, or "COMMAND NOT EXECUTED", signifying that it was not. From HyperCard and Authorware, some commands (CreateDrill, Store), will execute given no other parameters, setting defaults where necessary.

Message:	Occurs in:	Executed?
# of questions invalid, default set	CreateDrill	Yes

Explanation: The number of questions (*FQnum*) was either less than zero or greater than the maximum allowed integer. The default value is 20, unless the question pool size (*QPnum*) is less than 20, in which case *QPnum* is used.

Message:	Occurs in:	Executed?
# of repeats invalid, default set	CreateDrill	Yes

Explanation: The number of repeats (*REPnum*) specified must be greater than zero (remember, this specifies how many times the future queue is *presented*) and less than the maximum allowable integer.

Message:	Occurs in:	Executed?
Number to retire invalid, set to 1	CreateDrill	Yes

Explanation: The number to retire before replenishing (RPLnum), a variable parameter used with the "EC" insertion technique, must be greater than zero but cannot be greater than *FQnum*.

Message:	Occurs in:	Executed?
Orientation invalid, set to S	CreateDrill	Yes

Explanation: The character (*ORDchar*) that determines whether the long list is to be sequential (S) or random (R) did not contain a valid character.

Message:	Occurs in:	Executed?
Out of source elements	Incorrect	Yes

Explanation: This message may occur when Incorrect is using Resurrection or Appending methods to lengthen the future queue. For Resurrection, if the retired items list is empty, and for Appending, if the future queue is empty, this message will appear.

Message:	Occurs in:	Executed?
Purge position doesn't exist	Purge	No

Explanation: The purge position specified is greater than the length of the list specified.

Message:	Occurs in:	Executed?
Question pool size reset to # of questions	CreateDrill	Yes

Explanation: The question pool size cannot be less than the number of questions in the initial future queue, nor can it be greater than the maximum allowed integer. If the number of questions (*FQnum*) is also invalid, both are set to 20.

Message:	Occurs in:	Executed?
Ran out of memory	CreateDrill,Correct,Incorrect,Requeue, Retire,Insert,Pad,Correct,Purge	varies

Explanation: At some point in the routine, there was not sufficient memory to allocate memory for an item. The command may have been partially completed (in CreateDrill, this may occur if allocating a huge long list and future queue) or not at all. In any case, this error generally signifies that the queues are too large for the memory available on your system. Note that depending on the circumstances, there may not even be enough memory available to retrieve this message, nor execute any other command, so try to avoid running out of memory.

Message:	Occurs in:	Executed?
Request not within valid parameters	RetrieveInfo	No

Explanation: The integer that specifies the information requested must be greater than zero but not greater than the highest numbered request (22 as of this version).

Message:	Occurs in:	Executed?
Smallest position invalid, set to 1	CreateDrill	Yes

Explanation: The *smallest position* (SMLnum) specified when using "DII", "RDII", and "RDInt" insertion techniques cannot be less than one, nor can it be greater than any of the positions specified in the *incorrect positions* list.

Message:	Occurs in:	Executed?
States folder not found[: created]	Restore, Store	Yes

Explanation: The States folder was not found at the PathName specified. In Store, an attempt was made to create the folder. When using Store for the first time for a drill, this message will appear naturally since the folder will not exist until the first Store. However in Restore, and in subsequent Store calls, this signifies that the States folder is missing, or that it isn't in the correct path, or that the path itself or the drill name was incorrectly specified.

Message:	Occurs in:	Executed?
Student ID shortened	Store, Restore	Yes

Explanation: The string passed as the student ID (STUstr) can be no longer than 20 characters; if it is longer, it is truncated.

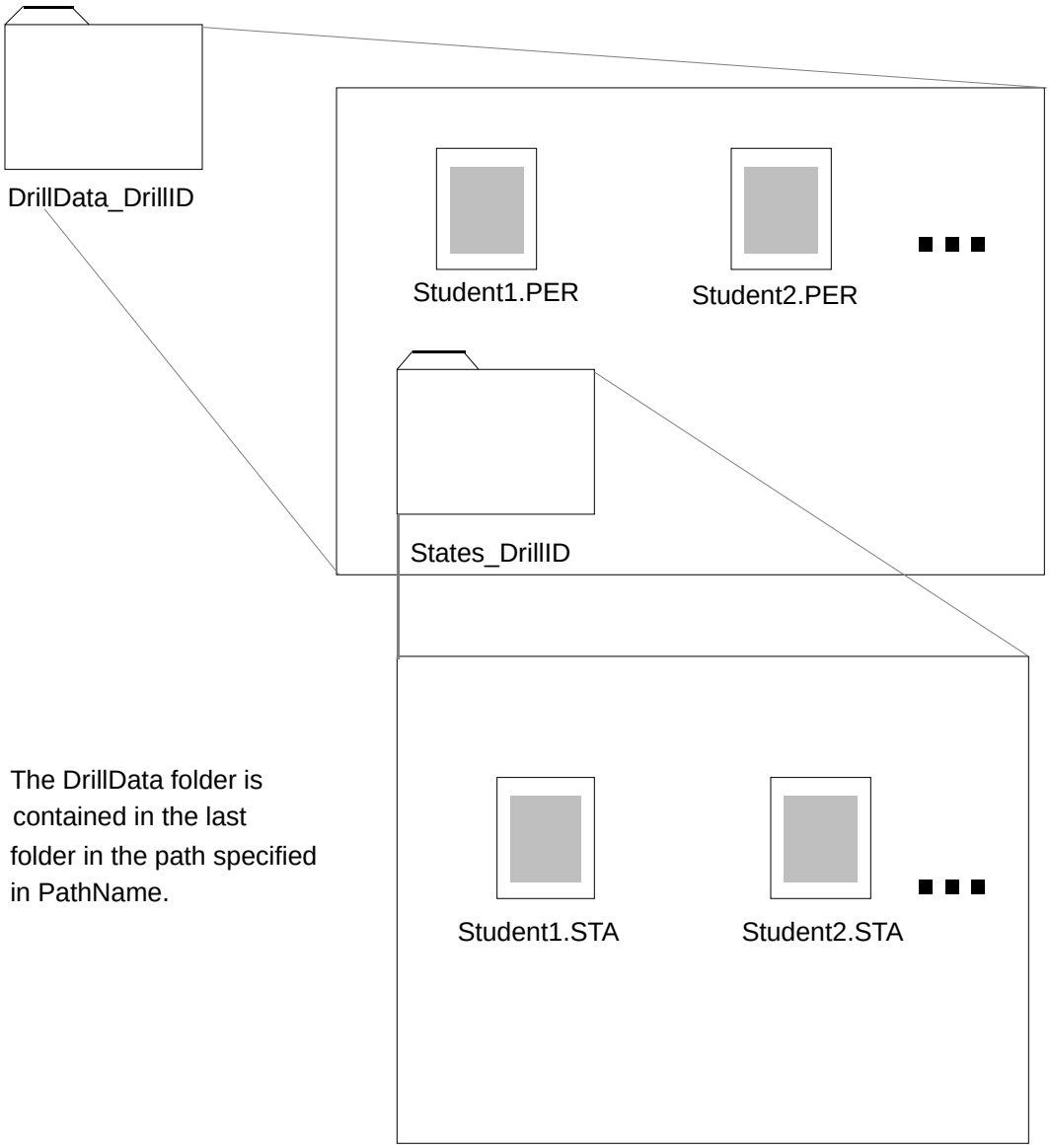
Message:	Occurs in:	Executed?
Unrecognized command	HyperCard,Authorware	No

Explanation: The string that contains the command name could not be identified as a valid command; thus no action is taken.

Message:	Occurs in:	Executed?
Warning: No active drill	HyperCard,Authorware	usually not

Explanation: This means that, currently, no drill exists. This message always appears (in HyperCard or Authorware) after a call to Dispose. When no drill exists, every command except for CreateDrill and Restore will yield this message. If a call to CreateDrill or Restore yields this message, it should be treated as an "Unrecognized command" message. In addition, if Restore is called when no drill exists and this message results, it may indicate that no student ID or drill ID parameters were passed.

Appendix B: Store and Restore File Map



The DrillData folder is contained in the last folder in the path specified in PathName.