Introduction

Welcome to the world of Yerk, a programming environment designed especially for the Apple Macintosh computer.

In this introduction, we will introduce you to some basic concepts of Yerk and its terminology. In the process, you'll see what a Yerk program looks like, and how it differs from most other programming languages. We'll also take you on a guided tour to the contents of your Yerk disks and show you how to make working disks that you'll use for both the tutorial and program development.

Back Up Your Disk Now

Since you will be using your Yerk disk right away, you should make at least one backup copy of the Yerk disk in this package.

Place the original disk in a safe place. From now on, use only the new copy you made. From this copy, you will make working disks for the tutorial and future program development.

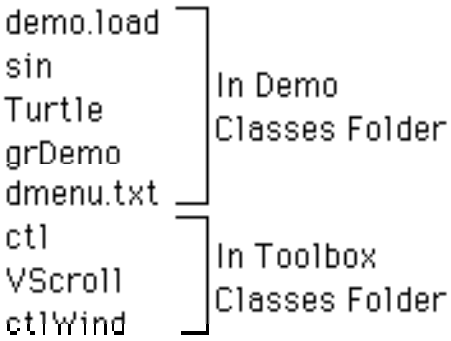
Making Working Copies

The Yerk disk as distributed will not boot by itself. If you have a hard disk, just copy the contents of the Yerk disk into a folder on your hard disk. If you have a two-drive system, you can put a system disk in the internal drive and a copy of your Yerk master in the external drive. This is your working disk. If you have a one-drive system, don't despair. Not all the files on the Yerk disk are needed to follow the tutorial or to develop Yerk programs. It is a good idea to make a working disk that offers lots of free space for you to save the programs you create. A working disk needs only the following files on it:

- System folder -- containing Finder, System, and other system files.
- Yerk -- the Yerk kernel (from the Yerk Folder).
- Yerk.com or YerkFP.com or another saved image -- the resident dictionary upon which you will build your application. Yerk.com is the Yerk dictionary without floating point support, and YerkFP.com is the dictionary with floating point support. Yerk.com is smaller, and will produce a smaller application.
- Yerk.rsrc -- the resource file used by Yerk for error messages and other resource data
- Modules -- the compiled modules (filenames ending in ".BIN") from your Yerk Folder.

- nMenu.txt -- a file containing the text of Yerk's menus.

For the tutorial, you'll also need some extra source code files on your disk. They are:



This still leaves plenty of room on the disk for your own source code files.

The Yerk Editor

Yerk does not have a resident editor, instead we recommend you use any one of many available desk accessory editors such as McSink, Vantage, MockWrite, etc. All you really need is a text editor available while running Yerk. With multifinder running, you may even use a word processor program, but you really don't need all that power.

Procedural Languages

Most of the common microcomputer programming languages -- BASIC, Pascal, C, and so on -- are sequential in nature. Programs written in these languages consist of long lists of precise instructions that the computer follows in strict order. Each instruction usually performs a single operation on one piece of data (e.g., assigning a number to a variable, retrieving a specific number from a long list of numbers in memory, and so on). Because the instructions are like steps of a procedure, these languages are called procedural languages. If you have used a procedural language and have a printout of a program, you can follow along the program's execution by tracing the single thread of execution from the beginning to the end.

Yerk: Objects and Messages

In contrast, a Yerk program listing does not consist solely of a precise sequence of events. Instead, large portions of a Yerk program define a framework within which a program's objects are to behave. A Yerk object is nothing more than a simulation of any real-world object you're familiar with: a rectangle, a Macintosh window, an artist's canvas, a bank account. When a Yerk program runs, a relatively small list of instructions route program execution through the framework, and the objects come to life: a rectangle draws itself on the screen; a window appears; a moused-controlled brush paints on an artist's canvas; a bank account monitors income and payments.

Yerk, itself, includes an extensive framework that gives you ready access to the resources inside the Macintosh. With this pre-existing framework, you can create complex objects as simply as typing two words. Let's do that right now, so you can get a taste of what Yerk has in store for you.

This is just a demonstration, not a lesson. So type along with us, and observe what happens without trying to remember each step.

Reset the Macintosh (either by turning it off and then on again, or by pressing the RESET button on the programmer's switch) and insert the backup Yerk disk you just made. Locate the Yerk.com icon and double-click it. In a moment, the Yerk window appears. We'll

explain the window's contents in detail in Lesson 1 of tutorial, but for now, create a rectangle object -- called "Box" -- in memory by typing:

rect Box <RETURN>

We need to tell Box where on the screen it should appear, and how big it should be. The rectangle framework inside Yerk wants these instructions in the form of screen coordinates for two opposite corners, the top left and bottom right. We'll choose 250, 80 for the top left, and 450, 220 for the bottom right. Put these figures into Box's memory by typing the following line, making sure you observe the spacing between elements and the colon:

250 80 450 220 put: Box <RETURN>

This line is called a message, which we just sent to Box. Now we can send a message to Box to draw itself on the screen. Type:

draw: Box <RETURN>

Try typing these two messages that Box can accept and see what happens:

clear: Box (Erases Box)
paint: Box (Fills Box with black)

Box can also change its size if a message tells it how many points to inset the opposite corners. Type:

10 10 inset: Box <RETURN>
clear: Box <RETURN>

If you typed all the above messages, then your box should look like the one in Figure i.

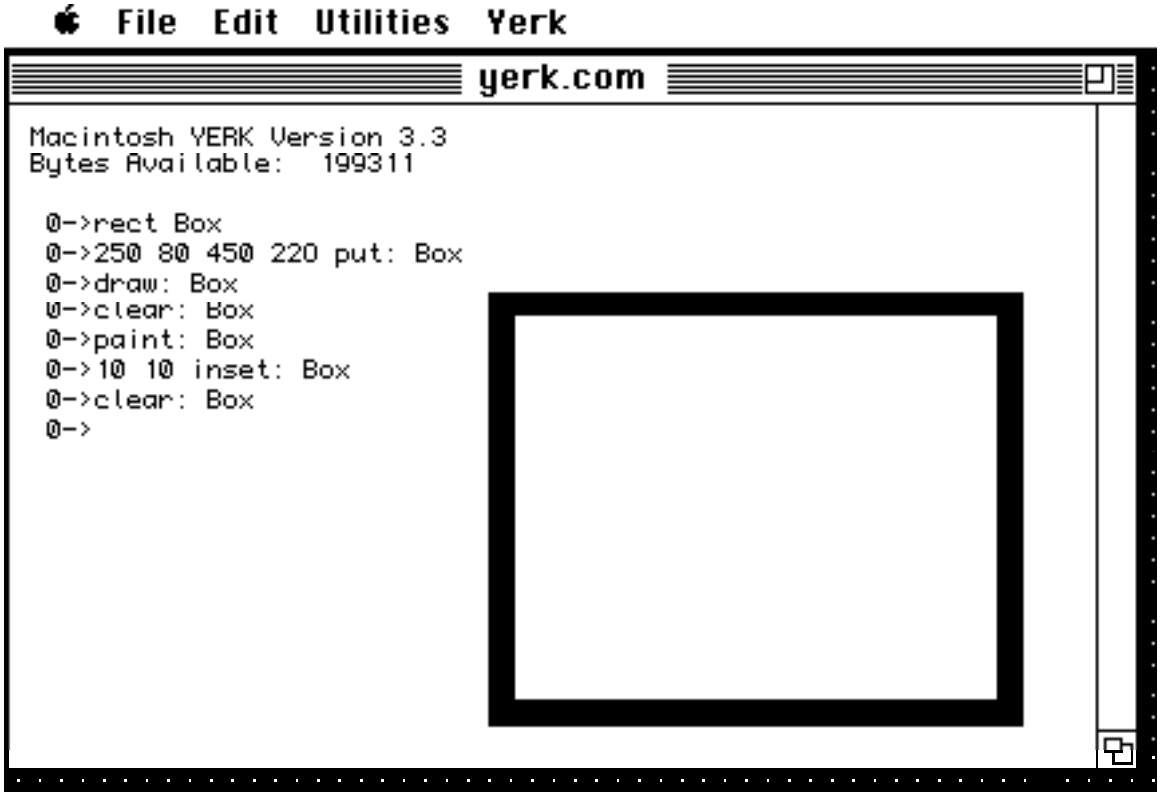


Figure i

Now, we'll expand on the size and location data that Box knows, and create a window object based on YERK's predefined window framework. First, clear the screen by typing:

cls <RETURN>

To set up a window object -- named "me" -- in memory, type:

window me <RETURN>

Macintosh windows need a lot of information before they can be placed on the screen, including data about Box (whose area defines the rectangular limits of the window), the title of the window, the type of window, whether it is to be visible, and whether it has a close box. Even then, the Macintosh Toolbox requires much more information, which Yerk automatically supplies. Some of the Yerk Classes, including Window, have example methods that display an instance of that class, with typical values. To see the window you just created, type the following message:

example: me <RETURN>

Your screen should look like the one in Figure ii.

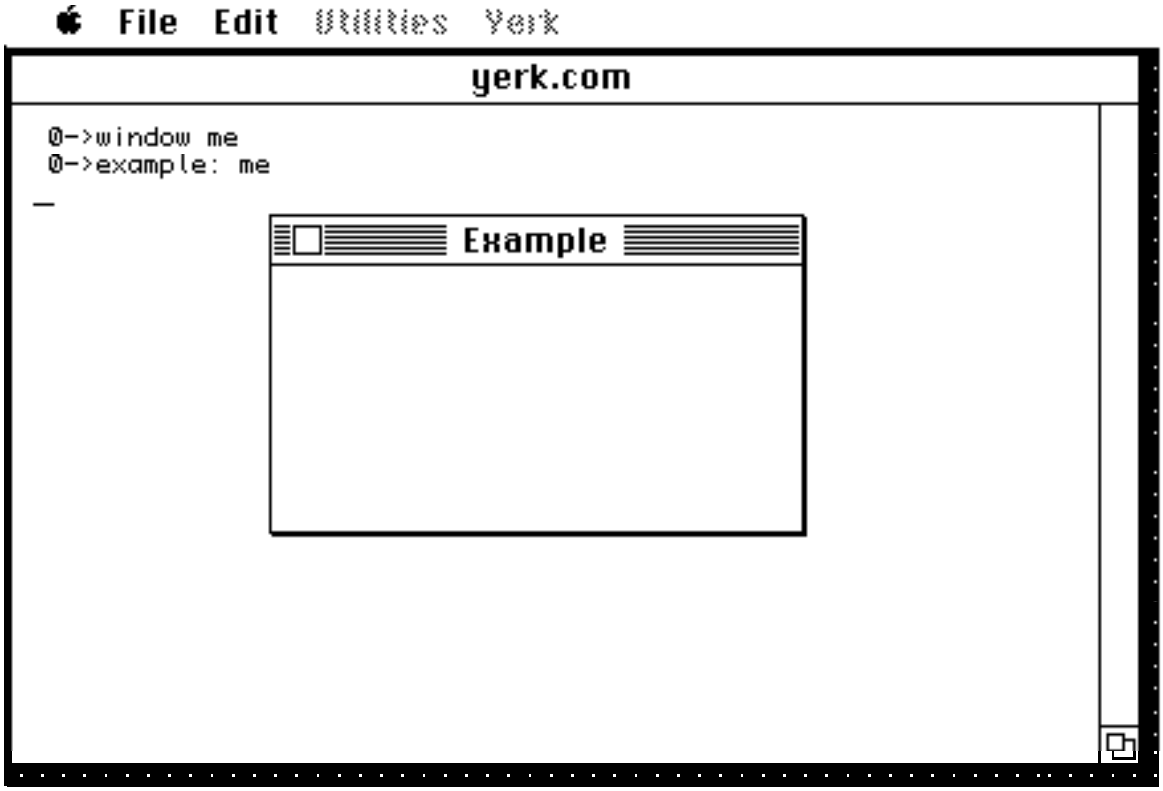


Figure ii

Press the Return key once. Notice that Example is now the active window and that the Yerk commands are processed in this window as long as it remains on top. Since each window has its own coordinate grid, we can make Box appear in the window, provided the coordinates fall within My Window's visible area. Type:

130 55 170 95 put: Box <RETURN>
draw: Box <RETURN>
paint: Box <RETURN>
7 7 inset: Box <RETURN>
clear: Box <RETURN>

Your screen should look like the one in Figure iii.

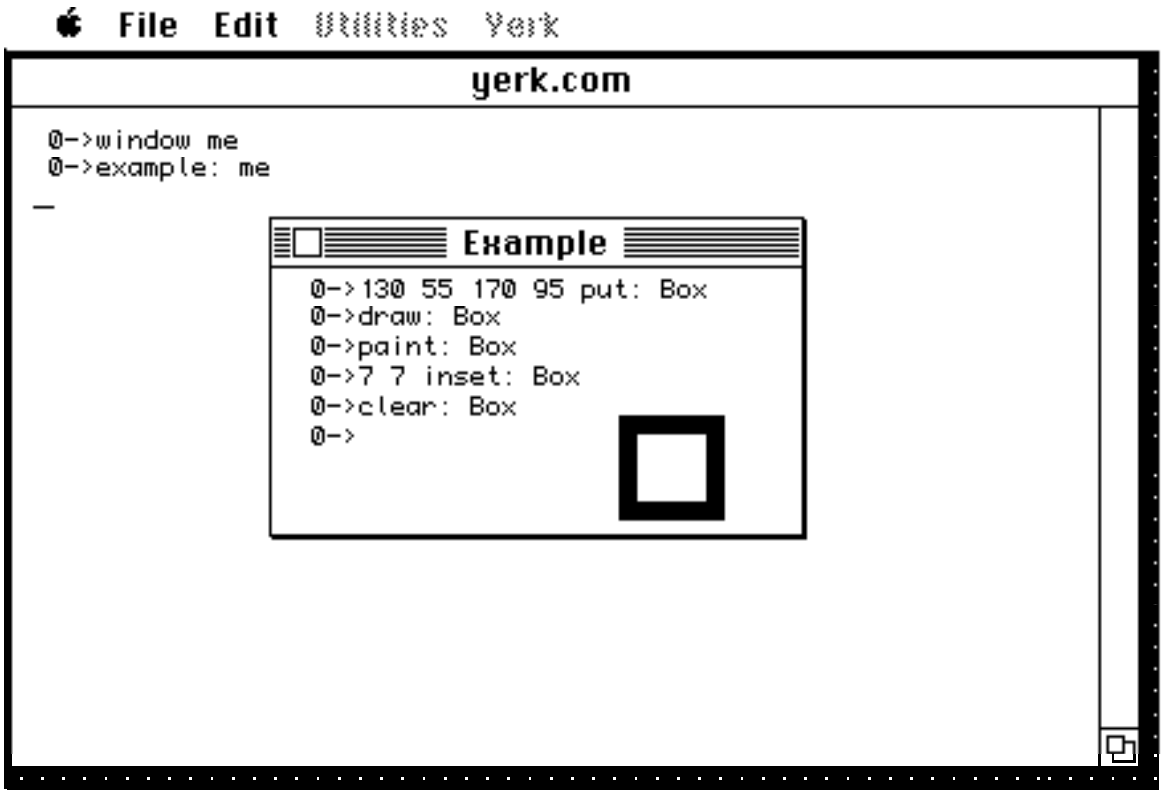


Figure iii

When you are finished experimenting, select Quit from the File menu, or type

bye <RETURN>

to return to the Desktop.

Private Data

The values you assigned to "Box" didn't just disappear after the rectangle appeared on the screen. A key element of Yerk is that the parameters you plug into an object (like the opposite corner points of Box) won't be disturbed, even if you were to create a new rectangle, called "square" and assign it entirely different values. You could create a hundred different rectangles with a hundred different names, locations, and sizes, and each rectangle object would hoard its own parameters as private data. Both an object and its private data stay in memory until called upon.

A Threaded Language

Even though a Yerk statement like "draw: Box" is rather simple, it is referencing several other statements (called definitions) that have been predefined for you in the Yerk dictionary (which is contained in Yerk.com). When the statement "draw: Box" is encountered, a chain reaction takes place inside the computer, as one definition reaches back for previous definitions, which, in turn, reach back for further definitions, until they reach YERK's

primeval definitions at the very core -- kernel -- of the language. And yet, far from being cumbersome and slow, Yerk is performing these self references in machine language -- the fastest possible method of accessing information stored in memory. This "reaching through" memory for all connections to a given definition is why Yerk is called a "threaded language."

The Yerk Dictionary

You can think of the "definitions" we've been discussing so far as if they were definitions in a dictionary of any language. The words are called Yerk words, and you can always look up their definitions in the Yerk Glossary in Part IV of this manual.

The Yerk language on the Yerk diskette is a complete, interactive dictionary of predefined Yerk words. When you double-click Yerk.com, the Yerk dictionary is loaded into memory. When you write a Yerk program, you essentially add definitions of your own Yerk words to the kernel (the new definitions apply only to the particular program you're writing, and don't affect the basic dictionary you'll use to write other programs).

Unlike most procedural languages, which lock you into its vocabulary of step-by-step instructions, Yerk lets you define one word to do the work of dozens of words. The ability to create new commands as you create your program is called extensibility, and you will find that many of the Yerk words you define in your first few programs will be reusable in other programs. This will help you reduce development time for succeeding projects and free up time for further exploration of the vast richness of the Macintosh environment.

Yerk comes with a large number of finished building blocks, called predefined classes. You have already used two of these classes: Class Rect and Class Window. You will be using these and other predefined classes while learning Yerk and later in developing commercial-quality programs. Most of these classes have been designed to quicken the learning process by giving you simple access to the most common Macintosh Toolbox routines.

Developing Commercial Software

Yerk is capable of creating commercial-quality software. The programs will contain the Yerk dictionary, plus the definitions you add to shape the program into whatever your design is. But you won't necessarily be giving Yerk away with each copy of your program.

Software written in Yerk for stand-alone applications is installed in a way that "seals off" Yerk from the user of your software. Your targeted users won't be able to "break out" of your program and gain access to Yerk. Instructions for this procedure are detailed in Part II.

Many programs don't need to be "stand-alone", but in fact, may allow access to the Yerk dictionary and interpreter.

What Your Yerk Disk Contains

To keep all master Yerk disk files on one convenient disk, and to prevent the accidental use of your master disk, the Yerk disk in this package does not contain any Macintosh System Folder files. In other words, you cannot start up your Mac with the Yerk distribution disk or the backup copy you just made. Working disks, as described in the next section, will have room for a full System Folder and be "bootable."

Files on the Yerk disk are divided into folders. They are:

System source	--
	Source code listings of basic Yerk classes.
Toolbox classes	--
	Source code listings of Yerk classes that interact with the Mac Toolbox.
Module source	--
	Source code listings for Yerk utilities and other program segments that are loaded into memory only as needed.
Yerk folder	--
	Essential files including the Yerk kernel (Yerk), an image of a Yerk dictionary with most of the predefined classes already loaded (Yerk.com), and the compressed, binary code files of the modules that are loaded when needed.
Asm source	--
	Source code listings of the Yerk assembler.
Float source	--
	Source code listings of the Yerk floating point package that is part of YerkFP.com
Supplement	--
	Source files of demonstration programs used in the tutorial; other example code, optional code and utilities.
nuc	--
	Source files of the Yerk nucleus (Yerk itself). The listing and macros are in MacAsm (Mainstay) format.

Most of the source code files in System, Toolbox, and Demo folders are provided not only for added documentation, but also if you want to recompile a modified version of Yerk. A study of that code, along with the tutorial, will help you master the powers of Yerk.

Yerk.com (or YerkFP.com) is the predominant file you will be opening, just as you did earlier in this introduction. It contains the majority of the Yerk words and predefined classes on which you will build programs.

The Bomb Box

In the course of your experimenting with Yerk, you will inevitably -- and perhaps unknowingly -- issue a command that the Mac doesn't understand. While many such errors will bring an error message to the screen (each is explained in Appendix A), other will summon the system error box, sometimes called the "Bomb Box" because of the icon it displays. In almost all cases, you will be able to click the Resume button, and continue your experimentation.

On some occasions, however, only the Restart button will respond to the mouse click. This completely resets the computer, erasing from memory whatever you were working on. It is a good idea, therefore, to save your work often -- a guideline you should follow in all your computer work.

There will be other occasions when the computer will appear to "lock up." It will be unresponsive to your keyboard or mouse actions. For this reason, it is suggested you install the Programmers Switch, which was packaged with your Macintosh (installation instructions are in the Mac Owner's Manual). If the computer appears frozen, press the Interrupt button on the Programmers Switch. Chances are you will be able to unlock the problem this way and see the Bomb Box on the screen. You may then be able to Resume normal operation. If the Interrupt button elicits no response, then press the Reset button, which functions the same as turning your Mac off and on again.

If you have Macsbug or some other debugger, you may use it as an added development tool. If you pop into the debugger, you may warm start back to Yerk by starting at 86 bytes offset from register A3.

If you get the Bomb Box on the screen or lock up the computer, don't despair. It happens to the most experienced Macintosh programmers. Do your best to remember what you did to cause the problem and try to determine how you can avoid it again. Only by making mistakes will you learn the right way.

With that, get ready to learn how to program your Macintosh in Yerk.