What's New?
Documentation
Products & APIs
Applets
For Developers
Java in the Real World
Business & Licensing
Support & Services
Marketing
Employment
Java Store™

Try the Applet Menu

# JAVAOS™: A STANDALONE JAVA™ ENVIRONMENT

by Peter Madany
contributions by Susan Keohan, Douglas Kramer and Tom Saulpaugh

## Contents





## Introduction

JavaOS™ is a new platform optimized to run Java™ on a variety of computing and consumer platforms. JavaOS provides a runtime specifically tuned to run Java applications directly on hardware platforms without requiring a host operating system. These Java applications are highly interactive, dynamic, secure, and portable.



Today many platforms exist, among them Microsoft Windows, Macintosh, OS/2, UNIX(r), Sun Solaris™, and NetWare(r). Currently, software must be compiled, tested, and packaged separately to run on each platform. In other words, the binary file for an application that runs on one platform cannot run on another platform, because the binary file is platform-specific.

The Java Platform is a new software platform for delivering and

running highly interactive, dynamic, and secure applets and applications on networked computer systems. The Java Platform sits on top of existing platforms and executes *bytecodes*, which are not specific to any physical machine, but are machine instructions for a *virtual* machine. A program written in the Java Language compiles to a bytecode file that can run wherever the Java Platform is present, on *any* underlying operating system. In other words, the same file can run on any operating system that is running the Java Platform. This portability is possible because at the core of the Java Platform is the Java Virtual Machine.

A Java Language development environment includes both the compile-time and runtime environments, as shown in Figure 1. The Java Platform is represented by the runtime environment. The developer writes Java Language source code (.java files) and compiles it to bytecodes (.class files). These bytecodes are instructions for the Java Virtual Machine.

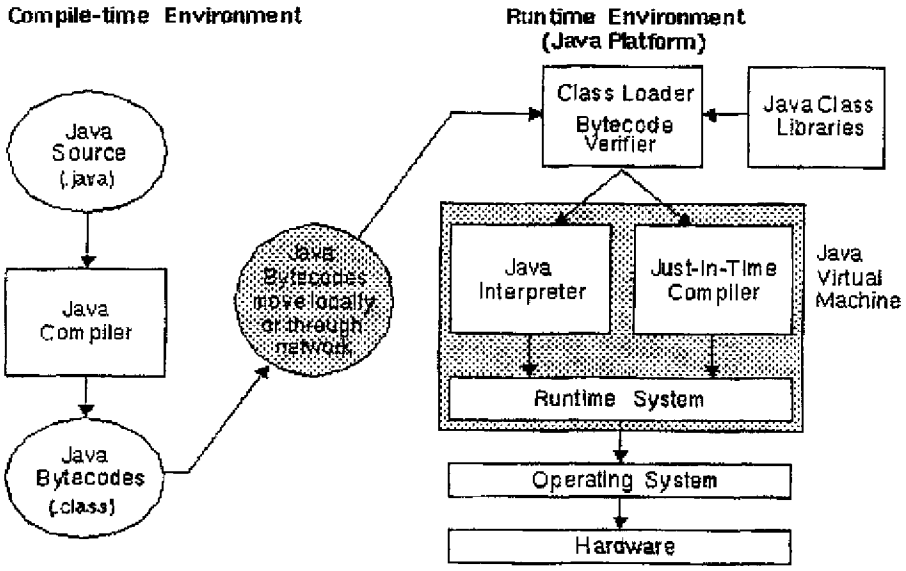


*Figure 1. Source code is compiled to bytecodes, which are executed at runtime.*

While each underlying platform has its own implementation of the Java Virtual Machine, there is only one virtual machine specification. Because of this, the Java Platform can provide a standard and uniform programming interface to applets and applications on any hardware. Therefore, the Java Platform is ideal for the Internet, where one program should be capable of running on any computer in the world. For more information about the features and architecture of the Java Platform and development environment, refer to the Developer's Corner at http://java.sun.com.

The Java API provides the specification of how the programmer writing an application or applet accesses the facilities of some object. The Java Development Kit (JDK) defines the Java API which

supports applications such as browsers and browser applets. The Java API is the same for all Java runtimes, including JavaOS, Microsoft Windows, UNIX, and Macintosh implementations.

JavaOS implements the Java Platform for running Java powered applets and applications. As such, it implements the Java Virtual Machine and the underlying functionality for windowing, networking and file system, *without requiring the support of a host operating system.*

JavaOS is built from a combination of native code (instruction set and hardware platform specific) and Java code which is platform independent. JavaOS defines a *platform* as a CPU, physical memory, and any attached devices, buses, and slots. The platform independent component of the operating system is called the JavaOS *runtime*. The platform dependent portion of the operating system is referred to as the JavaOS *kernel*.

The JavaOS runtime is designed to run on a very hardware-limited platform. For example, the runtime doesn't require a Memory Management Unit (MMU) to map virtual addresses to physical memory addresses, nor does it require memory protection. The runtime lets the underlying kernel choose whether to use an MMU or enforce memory protection. For example, the JavaOS kernel bundled in JavaOS (*version 1*) uses an MMU to make several disjoint physical memory regions appear contiguous to the runtime; future kernels may manage the MMU in a more pro-active fashion.

Since the benefits of the Java Platform fit many of the goals of building simple, intelligent, and dynamic network devices, the challenge is providing the Java Platform for devices with limited hardware and software resources. One of the best ways of reducing a device's hardware requirements is to remove the overhead caused by requiring a general purpose operating system. JavaOS provides just enough operating system features to support the Java Platform, thus allowing developers to provide the benefits of the Java Platform on devices with limited hardware and software resources. By removing the dependence on a host operating system, developers can create code that supports many different kinds of devices without many of the constraints imposed by traditional operating systems.

To this end, JavaOS is constructed using a layered architecture. Each layer is designed to be maintained in an independent fashion. This architecture serves two purposes: product customization and a parallel operating system release model.

JavaOS based product customization is the assembling of JavaOS layers for the constraints and purpose of the product. For example, a smart phone running JavaOS might require a realtime kernel, the Java Virtual Machine, a minimum graphics capability, and some communication protocols. A network computer requires a more full featured kernel without the realtime constraints, but with the Java Virtual Machine, Abstract Window Toolkit (complete graphics libraries), and all the rest of the Java Developers Kit (JDK) Application Programming Interfaces (APIs), including the HotJava™ browser. This paper first describes how the Java Platform works with a host operating system, then explains how the Java Platform works

with JavaOS.

## Java with a Host Operating System

When the Java Platform is used with a host operating system, the Java Virtual Machine and foundation classes can either be imbedded within the operating system or within an application, such as a Web browser. To support the Java Platform, the host operating system provides some support for the Java Platform. Each of the major features of the Java Platform directly or indirectly places requirements on a host operating system. Many of the language and utility classes assume operating system services are available. Figure 2 illustrates the software architecture used when running the Java Platform on a host operating system. In Figure 2, everything above the Java API is platform independent.

It doesn't matter what the underlying operating system or hardware is, the Java API is the same on all platforms. The Java API is implemented by several classes written in the Java programming language, including the language and utility classes, the Abstract Window Toolkit, and the network and I/O classes. The Java runtime, which includes the interpreter and garbage collector, is largely written in platform-independent C code. When the runtime is ported to a new platform, some specific platform-dependent code must be ported; this is labeled Java runtime (platform dependent porting interface) in Figure 2).

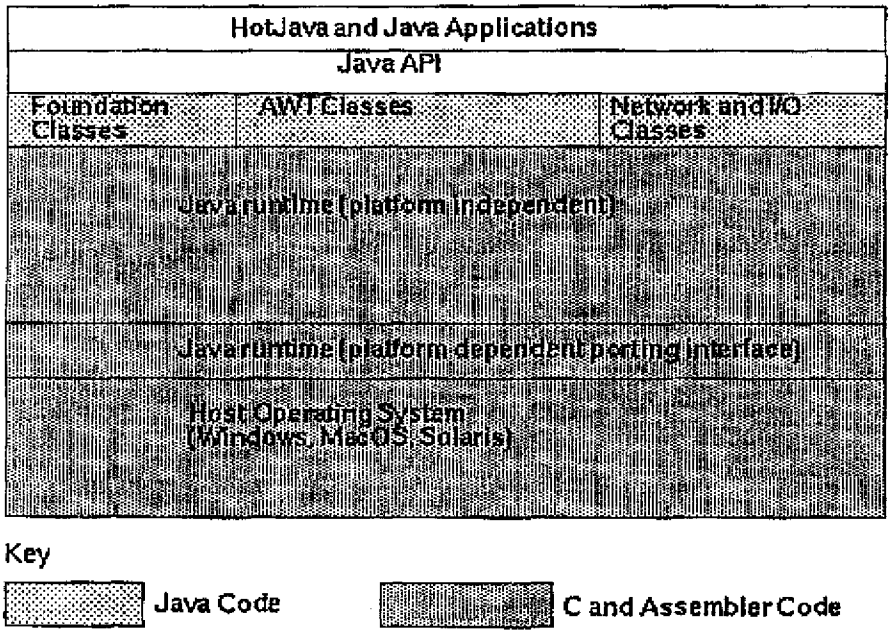


*Figure 2. Java on a Host Operating System*

The host operating system must provide the following:

- Multithreading support for the Java runtime; at least some primitive support for context-switching. If the host system provides better support for threads, that support may be used.
- Memory allocation. Although the Java runtime manages its own heap and includes garbage collection, it still needs a mechanism for allocating the memory that it will manage.
- Windowing and graphics support for the Abstract Window Toolkit which provides an abstract graphical user interfaces.
- Standard network protocols to support the Java networking classes.

When porting the Java Platform, the developer must do the following:

- Map the Abstract Window Toolkit (AWT) to the window and graphics subsystem provided by the host system.
- Map the networking classes to the native networking code on the system, which, for example, could have different system calls for operations on sockets.
- Map the file-related I/O classes to the host file system, which might use a different syntax for filenames.
- Port the platform-dependent part of the Java Virtual Machine to the particular system calls for memory allocation and thread management.

## Java without a Host Operating System

JavaOS provides a standalone Java environment. In other words, applications developed for the Java Platform using JavaOS can run on devices without depending on the support or existence of a host operating system. Also, applications written to run on machine without a host operating system may be run on machines that have a host operating system. To support the Java Platform, JavaOS:

- Supports the Java Virtual Machine using a kernel for Java.
- Supports AWT and the networking and file-related I/O classes.
- Provides the drivers for controlling a display, network interface, mouse, and keyboard.
- Supports the full Java API.

Figure 3 shows the software architecture used when running the Java Platform without a host operating system. As in Figure 2, programs above the Java API are platform-independent Java applications and applets.

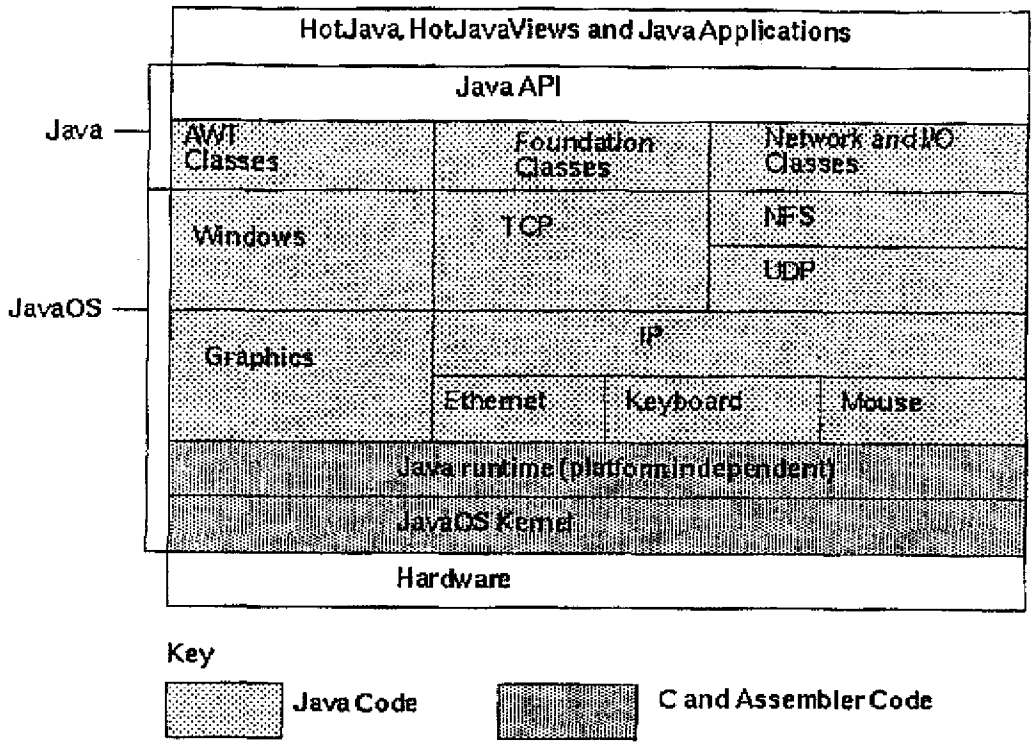


*Figure 3. Java Platform running on JavaOS*

**A Kernel for Java**

Within JavaOS, the Java runtime executes not only user-level applications, but also the system-level windows, graphics, networking, and driver code. The lowest layer of code handles the tasks often found in micro- or nano-kernels. The kernel for Java contains the low-level functions required by the Java Virtual Machine. This required functionality falls into the following categories:

- Booting
- Exceptions
- Threads
- Memory Management
- Monitors
- File System
- Timing
- Native Code Library Management
- Interrupts
- DMA

- Debugging
- Miscellaneous Platform Control

Boot systems for network, ROM, RAM, CD-ROM, floppy, and hard disks are all possible. While the bootstrap code is running, it allocates several memory regions, including one for the Java heap, and others for various I/O device registers and DMA regions. The bootstrap also handles mapping hardware devices that it has detected to their corresponding device drivers.

The main purpose of the trap and interrupt handling code is to service traps and interrupts and make the information accessible to the appropriate Java device driver.

The thread support code enables the Java Virtual Machine to switch contexts between the dozens of threads typically running in the system. Because of the protection provided by the Java programming language, JavaOS and all applications can run in a single address space. This simplifies and optimizes the context-switching code in JavaOS.

JavaOS does not require a Memory Management Unit, but it can use an MMU to make several disjoint ranges of physical memory appear contiguous, which simplifies memory allocation. The Java programming language eliminates the direct manipulation of memory by encapsulating all access into objects, classes, and automatic free memory collection. Eliminating pointers makes Java a more robust programming language than C and dramatically reduces the number of memory-related bugs.

JavaOS presents a portable, abstract memory model. The memory model is built upon the notion of *addressing*. Addressing is the process of identifying a memory location. The most fundamental unit of addressing in JavaOS is the *physical address*. A physical address identifies a unique memory location. A physical address is *not translated* by the MMU; rather it is a *raw* memory location, identifying what may be ROM, RAM, or I/O memory.

The JavaOS virtual address space is created by the JavaOS kernel layer. Note that the term virtual doesn't necessarily imply paging. Instead, the term *virtual means that this address space used by all software and may not resemble in any way the physical address space*. JavaOS does not assume a one to one correspondence between the physical and virtual address spaces. The use of an MMU to enhance the support of the virtual address space is not assumed either. Rather, MMU usage and page management is seen as a kernel implementation issue.

Unlike many operating systems in use today, JavaOS doesn't operate under the assumption of multiple virtual address spaces. Instead, JavaOS operates in a *single virtual address space*.

**Java Virtual Machine**

Within Java compatible systems like JavaOS, the Java Virtual Machine is obviously used to interpret Java bytecodes, but it is also

used as infrastructure for much of the rest of JavaOS. It executes the bytecodes in all classes within the system, handles exceptions, manages almost all of the RAM in the computer, and handles the simultaneous execution of multiple threads.

The implementation of the Java Virtual Machine that we use in JavaOS is very close to the standard one provided with Java Developer Kit, but we have tuned the memory allocation mechanisms, and added the reclamation of storage used by classes that are no longer needed by any objects in the system.

**Device Drivers**

All device drivers in JavaOS are written in the Java programming language. This is important for portability.

There are certain things almost every driver needs to do that cannot be done in pure Java code; these have been abstracted into two small support classes written in C. The Memory class enables drivers to access and modify specific bytes and words of storage. The Interrupt class handles interrupt dispatching. The methods of these classes are *not* made available to any Java applications.

Currently, several Java drivers exist for several different classes of devices and we are developing more that support both SPARC™ and X86 hardware. We are also in the process of defining a Java interface to enable third parties to build downloadable drivers for any device.

**Network Protocol Suite**

JavaOS includes a large suite of network protocols, all written in the Java programming language. These protocols include the basic transport and routing mechanisms specified by the TCP, UDP, IP and ICMP standards. JavaOS uses both DNS and NIS for looking up hostnames and supplying user names and passwords used during login.

JavaOS supports both Reverse ARP and DHCP for discovering the network address of a device. This enables the JavaOS machines to be installed with little or no per-machine administration.

A machine running JavaOS can access files as a client of a Network File System server and can be managed using the Simple Network Management Protocol. JavaOS machines can get the time of day from a network server, which also simplifies installation and administration.

**Window and Graphics**

Besides the network protocol suite, the largest piece of operating system functionality supplied by JavaOS is the windowing and graphics subsystem.

JavaOS uses the Tiny AWT library to provide Java based implementations of widgets such as buttons, menus, and scrollbars. Tiny AWT is not called "Tiny" because it is smaller than AWT, but because it enables AWT to place far fewer requirements on the

underlying window system.

We have developed a simple and memory-efficient window system in Java. We supply a system to manage the display of overlapping windows, a graphics package to draw and fill lines, arcs and polygons, and to render bitmapped fonts, and support for hardware video accelerators. The lowest levels of the graphics code need direct access to the frame buffer memory locations; therefore, they are written as native methods.

## HotJava and HotJava Views as the JavaOS Desktop Environment

HotJava™ is the first desktop user interface written in the Java programming language. As such, it can run on JavaOS. When combined with HotJava or HotJava Views™, JavaOS can function as a complete multitasking, graphical operating environment. HotJava can serve as the manager of the desktop metaphor, supporting multiple windows, each of which is capable of browsing HTML pages and running one or more Java applets. HotJava's customizability and extensibility make it an ideal framework for JavaOS applications.

JavaOS can also run other main programs besides HotJava and HotJavaViews.

## Is JavaOS an Operating System?

Is JavaOS really an operating system? Whether it is or isn't depends on one's perspective. JavaOS differs from conventional operating systems in several ways in that it does not:

- Need a file system.
- Need virtual memory.
- Need separate address spaces.
- Support more than one programming language.
- Have its own set of system calls.

JavaOS also resembles an operating system in several ways in that it:

- Is bootable.
- Supports a password-protected login feature.
- Safely runs several applets at a time.
- Includes several device drivers.
- Communicates using many standard network protocols.
- Has its own window system.
- Has an API.
- Will run the thousands of applets and applications that have been written.

## Performance

This section discusses some of the performance advantages you will see when you write applications for the Java Platform running on JavaOS.

### Speed

Currently JavaOS has had minimal performance tuning, has not used a "Just-In-Time" compiler to translate byte-codes into machine code, and makes minimal use of native methods. Therefore, one might expect JavaOS to perform poorly, but our measurements and benchmarks indicate that performance is not only better than expected but also better in some areas than some more mature systems written in C or C++.

The TCP/IP throughput on JavaOS is already double our original goal and is more than adequate for Web browsing. We have also run Pendragon Software's CaffeineMark benchmark on several systems to compare their performance when executing Java applets, and we have observed very encouraging results. The main reason for these good results is that JavaOS jettisons the many layers of software architecture that other systems have built up to make programming in languages like C safer and less platform-dependent. Ironically, while those layers do hurt performance, they do not come close to providing the safety and platform independence of Java!

### Space

How much memory is needed to support JavaOS? One could build a complete system with a total of 4MB of ROM and 4MB of RAM.

In the ROM would be all the code for JavaOS itself, including the kernel code, drivers, Java Virtual Machine and standard classes, plus the JavaOS windows, graphics, and networking components, plus the code for HotJava. The ROM could also include approximately 1MB of bitmaps for fonts with all the combinations of various types like serif, sans-serif, and typewriter, several point sizes, and various styles such as bold and italic.

Assuming that the ROM could be executed in place, then the system could use two and one-half MB of RAM for the dynamic requirements of JavaOS and HotJava and still have about one and one-half MB of RAM for downloaded HTML pages, applets, and images. Systems built using JavaOS that do not require windowing and HotJava code could run in less than half the space.

## Advantages of JavaOS

There are several advantages of using JavaOS to provide the Java Platform directly on hardware, including:

- JavaOS achieves the goal of eliminating the overhead of a host operating system. Because JavaOS contains no

extraneous features found in other operating systems, it allows smaller and simpler devices to be built that execute Java programs more efficiently than other systems.

- JavaOS may be stored on ROM, enabling simple, low-cost systems that boot quickly.
- JavaOS is written in Java, thus, new components can be quickly developed because Java code is easier to debug, is inherently portable, and is dynamically extensible.
- JavaOS enables systems that are as easy to install and maintain as terminals, yet are nearly as powerful as traditional desktop machines. Perhaps the most expensive part of owning a computer is the cost of configuring and maintaining one. JavaOS can dramatically reduce that cost compared with a typical personal computer.

## Target Systems for JavaOS

JavaOS is ideal for several types of devices, including intranet computers, Internet computers, and embedded devices. As the name implies, JavaSoft develops software not hardware, and it supplies JavaOS to hardware companies to enable them to build intelligent and dynamic hardware devices.

### Intranet Computers

Intranet computers are computers connected to an enterprise's network infrastructure. Most companies can deliver Ethernet connections to each desktop machine, which provides plenty of performance for transferring Java classes and other data. Many companies also have higher speed backbone networks.

Servers are another key element of enterprise networks. They support centralized administration of intranet computers. For example, if a server implements the DHCP protocol, which supports dynamic allocation of IP addresses, no administration is required for installing an additional intranet computer.

The Java Platform is ideal for developing and deploying MIS applications, since applications can be automatically downloaded over a network and since it is so easy to write network-aware applications in the Java programming language.

### Internet Computers

The first Internet computers will have to work well with the relatively limited bandwidth provided by today's high-speed modems and ISDN. When cable modems become more prevalent, then home computers will have the network bandwidth common in today's local area networks. Since most people don't have sophisticated network servers at home, Internet Service Providers will have to provide the necessary infrastructure so that using an Intranet computer is as easy as using an appliance. For example, you just plug the computer in to an outlet and a phone jack, turn it on, and you are automatically on the Internet and

ready to go surf the World Wide Web.

**Embedded Devices**

What if a device has only 1MB or 2MB of RAM and 1MB or 2MB of ROM, and possibly no graphical display, yet you want it to be able to load and run Java applications? JavaOS can be tailored to fit particular devices like set-top boxes, PDAs, and electronic devices without any graphical display. Note this does not mean that one could subset the language itself or remove features from the language or utility classes. But, for example, if there is no display, one could remove not only AWT for that device, but also remove the window and graphics code from JavaOS.

Similarly if the device did not have a need for certain network protocols, they could be eliminated. In order to meet some embedded system requirements, we still have to tune the Java Virtual Machine and garbage collection to support some soft real-time capabilities.

We are working with software tool vendors to build a rich software development environment for JavaOS, including a remote debugging capability. The memory footprint for JavaOS in its smallest possible configuration will be about 128K of RAM and 512K of ROM. Note that this is the memory required for JavaOS itself; additional memory would be needed for applications.

## Availability

JavaOS currently runs on several platform. One set of platforms is based on SPARC microprocessors. Another set of platforms is based on microprocessors compatible with Intel's x86 instruction set. JavaOS has been ported to systems based on other microprocessor instruction sets. The current version of JavaOS is based on the 1.0 version of the JDK. Some of the key features we will be adding in future versions of JavaOS include:

- New Java Platform API's as they are added to the future versions of the JDK.
- The device driver interface that we are currently specifying.
- Enhanced window and graphics components with features like scalable fonts.
- An enhanced network protocol suite with PPP-related features.

## Summary

In summary, JavaOS is a new software platform that enables Java applications to run directly on hardware without requiring a host operating system.

JavaOS includes the Java Virtual Machine, the standard packages of classes, and just enough OS code to support them. The OS code includes low-level code written in C or assembly language, plus

device driver, networking, windowing, and graphics-rendering code written largely in the Java programming language.

The main advantage of JavaOS is that by eliminating the overhead and complexity of host operating systems, it enables new classes of simple, intelligent, and dynamic network devices that will be lower cost. JavaOS is targeted at systems such as intranet terminals for enterprise desktops, consumer Internet computers suitable for Web surfing, and embedded devices where hardware resources are even more restricted.

*This page was updated: 13-Oct-97*

---

FEEDBACK | SUPPORT & SERVICES | MAP
Search

---

For information, call:
(888) 843-5282 (North America)
(512) 434-1591 (Other locations)