Out Of This World

A VRML 2.0 Proposal
Submitted by Apple Computer, Inc.

1. Synopsis

This document presents Apple Computer's suggested enhancements for the VRML 2.0 format, titled "**Out Of This World**". Apple has chosen to submit this proposal in response to the VAG's call for technologies that can significantly enhance VRML, and we believe that the suggestions presented here truly enable VRML as a workable solution for the personal computer marketplace.

Out Of This World deals with issues that are of pertinence to the entire VRML community. We believe that the key to VRML's acceptance in the marketplace is the adoption of a foundation that offers mainstream developers both stability and the overall flexibility needed for innovation.

Apple Computer understands that the VRML 2.0 standard will be the synthesis of competing proposals. Rather than presenting an all-encompassing "magna carta", Apple Computer is offering **Out Of This World** as a foundation to coallesce the contributions of key members of the VRML community. We fully welcome comments and suggestions from all parties to enhance our proposal and we are committed to adjusting it based on this feedback. We do this in the spirit of consensus-building and creating the most robust, high-performance standard for VRML 2.0.

Similar to other VRML 2.0 proposals, **Out Of This World** offers insight on behavior implementation. The interested reader is referred to the "Adding Behaviors to 3DMF" section of this document.

Out Of This World's file format is based on the 3DMF (3D Metafile Format) specification used by QuickDraw 3D--Apple's cross-platform 3D graphics API. 3DMF does not require QuickDraw 3D to run. The file format is completely independent of the rendering API, and is available in ANSI C source code form free of charge, from the Apple Home page at http://www.info.apple.com/qd3d.

3DMF is a "container" file format that runs on Macintosh, Windows and UNIX systems. It encapsulates not only 3D information, but also multimedia attributes such as sound, Web URLs, etc. 3DMF supports all types of geometries and objects, plus textures, lights, shaders, cameras, active renderers, material properties, hierarchical information, and more.

There are five key areas that we believe differentiate our proposal in the file format area and make VRML a more conducive platform for personal computer developers:

1. 3DMF files are far smaller than both the VRML 1.0 or Moving Worlds formats--there is no need to GZIP a 3DMF file, which eliminates a performance bottleneck. Also, the 3DMF binary format can be parsed more easily, and with far less CPU load, than an ASCII format. While parsing speed has not been an issue when reading files over current internet technology, parsing speed is crucial for desktop authoring, DVD-based content, and future high-speed networks.

   3DMF is already a native file format for the Mac OS and is fully cross-platform for PCs and UNIX systems (we provide the source code for all platforms at no charge and license-free). 3DMF is already supported by over 60 developers on a cross-platform basis, including Adobe and Macromedia.

2. **Out Of This World** does not require that data be represented as a Scene-Graph. This avoids the high memory use and statically-routed dataflow typically associated with Scene-Graph architectures.

3. VRML browsers are actually scripts that are downloaded dynamically with the 3D world, using Java or another language (**Out Of This World** is API- and language-neutral), in effect creating the ultimate browser -- one that is always dynamically updated and never lacking in new features.

   Out Of This World creates a new paradigm of browser that takes advantage of the latest net-oriented technology and eliminates the need for the current "one size fits all" software model. Flexibility is assured for both developers and end-users.

4. Unlike Moving Worlds, Apple proposes a non-centralized database for the elements that make up a 3D world. This eliminates what we see as a fundamental issue in all current proposals: that the browser is solely responsible for parsing and communicating with the VRML world. We believe that our approach is more suited to the needs and systems that are on the majority of desktops worldwide.

5. 3DMF is backwards-compatible with VRML 1.0, by incorporating an extremely minor tweak into existing VRML worlds (adding three lines of code as a header and footer), which will enable VRML 1.0 files to be encapsulated by 3DMF.

This gives developers and end-users the best of both worlds (forgive the pun)--backwards compatibility with existing browsers and VRML 1.0, and high-performance and features for new **Out Of This World**/3DMF-based environments.

2. Request-for-Proposal Requirements

The VAG has issued a Request for Proposal (RFP) for VRML 2.0 specifying a number of requirements for proposal submittal. The Apple Computer **Out Of This World** proposal fulfills all requirements as indicated below.

2.1 Contributors

Out Of This World is the result of contributions from several key individuals inside Apple Computer. While crediting everyone would amount to a long list of names, the main contributions have come from:

- **Michael Kelley** (Main Contact Person)
  Apple Computer, Inc.
  1 Infinite Loop, M/S 301-3K
  Cupertino, CA 95014
  Tel: (408)974-8535
  mwk@apple.com

- **Ronie Hecker**
  Apple Computer, Inc.

- **Dan Venolia**
  Apple Computer, Inc.

- **Kent Davidson**
  Apple Computer, Inc.

- **Pablo Fernicola**
  Apple Computer, Inc.

- **Fábio Pettinati**
  Apple Computer, Inc.

The following contributors are also available for additional information on the **Out Of This World** proposal, and to discuss the plans that Apple Computer has for 3D graphics both on and off the Web:

- **John Alfano** (Marketing Contact)
  Apple Computer, Inc.
  1 Infinite Loop, M/S 303-2E
  Cupertino, CA 95014
  Tel: (408)974-6495
  alfano1@applelink.apple.com

- **Shawn Hopwood** (Developer Evangelism Contact)
  Apple Computer, Inc.
  1 Infinite Loop, M/S 303-2EV
  Cupertino, CA 95014
  Tel: (408)974-4189
  s.hopwood.@applelink.apple.com

- **Jonathan Hirshon** (Public Relations Contact)
  Horizon Public Relations
  5201 Great America Pkwy, Suite 320
  Santa Clara, CA 95054
  Tel: (408)982-2555
  jh@horizonpr.com

2.2 Reference Implementation

The complete 3DMF Reference Specification is available online in HTML format. Source code for Version 3.0 of the cross-platform 3DMF parser is available in three formats for downloading:

1. Stuffed (.sea) Macintosh archive
2. Tar (.tar) Unix archive
3. Zip (.zip) Windows archive

Apple Computer is working on a reference implementation for "Adding Behaviors to 3DMF" as outlined elsewhere in this document.

Apple Computer is also working on a helper application and a plug-in that, working with Netscape Navigator, allow users to visualize the contents of 3DMF files through the World Wide Web. Pre-release versions of both components will be available during the first half of February'96 in the QuickDraw 3D and Netscape Web sites.

2.3 Examples

Examples of 3DMF files can be found throughout the online version of the 3DMF Reference Specification. A simple example conveying the structure of a 3DMF file is also available.

2.4 Historical Background

The roots of 3DMF come from several years of computer graphics research carried at different organizations inside Apple Computer. In 1990 Apple started the effort that eventually led to the development and deployment of QuickDraw 3D, and its native file format--3DMF.

QuickDraw 3D is a cross-platform application program interface (API) for creating and rendering real-time, workstation-class 3D graphics. It consists of a high-level modeling tool kit, a shading and rendering architecture, a device and acceleration manager for plug and play hardware acceleration, human interface guidelines and toolkit,and a cross-platform file format.

QuickDraw 3D is a developer-enabling technology, not an end-user product. End-users interact with QD3D by working with applications that incorporate the QuickDraw 3D libraries.

Apple Computer maintains a web site with copious information about QuickDraw 3D and 3DMF.

Since its inception, QuickDraw 3D has had four guiding principles that helped create a truly innovative 3D graphics architecture squarely aimed at fulfilling user needs: ease of use, price-performance flexibility, information exchange and reuse, and tailored solutions. Within the context of file format, the latter two principles are of paramount importance.

The ability to share and reuse information has been available to computer users everywhere for several years--until recently, most users of 3D graphics applications had limited access to such a feature. Feedback from over 300 developers indicated their discontent with the existing 3D file formats. These developers also indicated their interest in seeing Apple developing a truly ubiquitous response to their needs.

Listening to that feedback, Apple set out to create an object-oriented, extensible, cross-platform format that goes well beyond least-common denominator geometry information, to encompass all appearance and custom information.

The same developers confirmed the fact that the 3D graphics marketplace is not homogeneous. Although there is a common need for 3D objects to be created, manipulated, rendered, and reused, in reality, each market segment has very specific needs and requirements.

To address these diverse needs, Apple Computer designed 3DMF with extensibility in mind. The 3DMF allows for new custom (or application-specific) objects and attributes as part of the file format specification. A critical feature in 3DMF is that 3D applications supporting the file format specification be able to read files containing custom information. The cross-platform 3DMF parser, as well as the one part of QuickDraw 3D, take great care in preserving custom information throughout file access.

2.5 Legal Issues

Apple Computer's reference implementation for **Out Of This World** is based on the publicly available 3DMF (3D Metafile Format) parser. This parser is available in source code form, and may be reproduced and used without restriction or charge for educational, internal, and commercial or non-commercial use.

3. "Out Of This World" and VRML 2.0 requirements

The VRML 2.0 RFP contains a set of requirements for submitting proposals. The following sections outline how **Out Of This World** addresses these requirements:

- ***Performance***
  3DMF addresses the performance requirement on two different fronts: ease of parsing and small size. The syntax in 3DMF lends itself to files that can be easily parsed, even when encountering extensions to the basic format. On the size front, 3DMF files are very compact, regardless of whether they are stored in text or binary forms.

  For a wide range of 3D models and scenes, 3DMF files in text form are usually 70% smaller than their equivalent VRML 1.0 files. 3DMF files in binary form are highly compressed and normally smaller than equivalent VRML 1.0 files compressed with gzip. The fact that 3DMF files are so compact is an added bonus, since compression is not necessary--rendering can take place as soon as the necessary information becomes available on the client side.

- ***Scalability***
  3DMF imposes no size or complexity limits on 3D models. The ability to successfully parse file contents, and store the related information in data structures suitable for rendering depends only on available memory. Furthermore, the publicly available 3DMF parser is not dependent on any specific rendering technology. The only system requirement is the availability of an ANSI compliant C compiler.

- ***Composability***
  The syntax and semantics of 3DMF impose no restrictions on referencing externally located 3D models. Future versions of 3DMF could offer increased flexibility on internal and external file references.

- ***Authoring***
  Authoring has been a constant area for concern leading to the design of 3DMF. Apple Computer has worked closely with its developer base to help define a file format that would foster the creation of compelling 3D graphics content. 3DMF was the result of Apple's partnership with key industry independent software vendors and content providers. 3DMF has been extremely well received by the developer community at large: over 60 companies are actively developing tools and content based on the 3DMF specification.

  3DMF has been integrated within the Mac OS operating system, and offers users the ability to cut and paste 3D information across all applications. Apple is in the process of offering similar functionality to applications running on Windows 95 and Windows NT, as part of the QuickDraw 3D implementation for these platforms. It is important to re-emphasize that 3DMF does not require that QuickDraw 3D be ported to a given platform as a prerequisite for availability to all interested parties.

  One of the key advantages of 3DMF is the ability to bridge the gap between the web and the desktop, offering end users a much needed integration. A common VRML specification that allows 3D objects to be captured from the Web, transported to the desktop and later integrated with traditional media (i.e. printed documents) has become a critical end-user requirement. Likewise, users also demand the ability to take 3D

information from the desktop to the Web without resorting to saving a subset of their data just for the Web. 3DMF fulfills both requirements.

- ***Power***
  The power requirement embodies the need for seamless extensibility, an area that is integral to the 3DMF specification. 3DMF has been designed with extensibility in mind. In addition to the basic simple and compound data types, any 3DMF file can contain entirely new objects and attributes. The syntax in 3DMF allows for easy detection of "unknown" objects and attributes of any kind. These extensions allow any type of data to be encapsulated as a new object type, or attached as a custom attribute to existing objects.

  To illustrate how this might work, let us consider attaching sound information to objects in a metafile. Any application, even those which do not know how to handle sound information, can successfully parse that file's contents. Only applications that do understand the sound specification, however, would be able to manipulate that form of data.

  The key point to remember is the ability for 3DMF files to "travel" to platforms and applications that don't fully understand their content, and yet, everything works as expected. The publicly available 3DMF parser implements this philosophy, and any instance of custom data is preserved across application access.

- ***Interoperability***
  The syntax and semantics of 3DMF are orthogonal to network, graphics, and language standards. 3DMF is a container for information, whose interpretation takes place outside the realm of the file format. As such, 3DMF coexists with existing formats, regardless of their syntax and applicability.

  The 3DMF specification guarantees cross-platform readiness, and complete independence on endianess and byte ordering. Such independence applies to both text and binary formats. The only platform requirement is that floating point numbers be represented in the manner specified by the IEEE-754 floating-point standard.

- ***Multi-user Potential***
  Implementing multi-user scenarios is a very important future direction for VRML, and the 3DMF specification contains no restrictions or limitations to achieving that goal. All the multi-user issues described in the VRML 2.0 RFP can be implemented on top of the 3DMF specification. Likewise, the 3DMF specification can accommodate any number of multi-user scenarios.

- ***Open Standard***
  As stated elsewhere in this document, 3DMF is a standalone technology, not specifically tied to any rendering APIs or computer platforms. The fact that QuickDraw 3D, Apple's 3D rendering API, uses 3DMF as its file format should be seen as merely a convenience. Any rendering API other than QuickDraw 3D can be used without issue.

  The complete 3DMF Reference Specification is available online in HTML format. Source code for Version 3.0 of the cross-platform 3DMF parser is available in three formats for downloading:
  1. Stuffed (.sea) Macintosh archive
  2. Tar (.tar) Unix archive
  3. Zip (.zip) Windows archive

- ***Backward Compatibility***
  The syntax and semantics of 3DMF are backward compatible with VRML 1.0 by the use of adding header and footer lines to existing VRML 1.0 files that will enable 3DMF to recognize and encapsulate them. This represents an extremely minor change, easily added as part of VRML 1.1, which gives content designers the significant advantages of the 3DMF format. Here is an example:

  With the following three-line change to existing VRML files, 3DMF can be made backward compatible with VRML 1.0. These changes effectively transform a VRML file into a custom object according to the syntax and semantics of 3DMF.

```
3DMetafile ( 1 0 Normal tableofcontents0> )  # 1st line change
vrml (                                       # 2nd line change
  #  ******************************
  #  * existing VRML file contents *
  #  *        inserted here       *
  #  ******************************
)                                            # 3rd line change
```

  Existing VRML 1.0 content can also be automatically converted to 3DMF through the use of translators. Writing automated translators, while not a difficult task, might not be necessary since there is a growing number of QuickDraw 3D savvy applications that also open VRML 1.0 files. These applications can easily convert existing VRML 1.0 content to 3DMF.

  The next generation of easy-to-use authoring applications offer a far better interface to 3D content creation than creating files by hand with text editors. Being able to read VRML or 3DMF files is like being able to read Postscript files directly--it can be done, but there are far better ways to deal with that information.

  For VRML to become a truly pervasive standard, the availability of world-class authoring solutions is key to fostering the creation of compelling content. When the VRML 1.0 effort started, no such tools were available. Understandably, text editors filled that void, but there is no need to perpetuate this transitory solution except for those devoted programmers who demand ultimate flexibility.

- ***Completeness***
  Out Of This World is meant to stand on its own from the file format viewpoint. Apple Computer believes that achieving synergy among other competing proposals is the preferred way to arrive at the ideal VRML 2.0 standard.

  Apple Computer offers **Out Of This World** as part of achieving this synergy, and invites others to integrate their key ideas onto the 3DMF framework.

4. Adding Behaviors to 3DMF

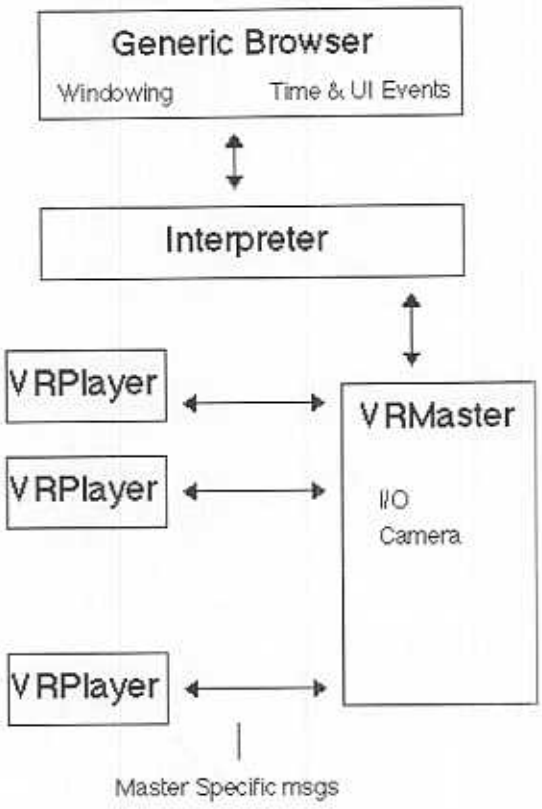
This section presents Apple Computer's contribution to adding behaviors to 3DMF.

4.1 Overview

The Apple proposal for behaviors in a VRML file is script-centered, with behaviors specified in code. Scripts interact among themselves, with predefined graphic data contained in the file, and with any graphic data that they have dynamically created. Scripts are in total control of the graphics - there is no centralized database of graphics, controlled by a browser.

Scripts have access to a large library of routines for manipulating their graphic data, controlling how and when drawing and picking is performed. No predefined static structures are imposed on what code can appear inside a script (as long as it conforms to the browser's security requirements), and who the recipient of a message sent from a script may be. Nor are there any predefined behaviors, or built-in fixed UI techniques for manipulating the camera. These are added as scripts.

To interact with VRML behavior files one needs a generic browser, such as Netscape's Navigator, and possibly an interpreter, if that is not built into the browser. Camera parameters, drawing, and picking are controlled within scripts.



We see two main authoring modes for behavior files. In the first, which is a special case of the second, a "web" of VRML files is authored by a single person. In this case, the generic browser and scripts are sufficient for playback: the scripts constitute a single program, distributed on the web, perhaps containing some graphic data.

In a multiperson authoring environment, there is a need to define how the "pieces" of the program fit together. One needs a script-to-script protocol: i.e. the messages allowed and their semantics.

A configuration that we think may have potential is the following: one script, which we call VRmaster, is a generalization of the 3D browser concept. It alone interacts with the generic browser - essentially fetching user, time and update events, and a port for drawing - and is free to interpret them as it wants. It "owns" the view: it is responsible for setting up lights, moving the camera, setting up for picking and drawing, and for interacting with all other scripts.

Each VRmaster publishes a public interface. Regular scripts - VRplayers - are written for a specific VRmaster (or a set of masters). This may allow for multiple types of behavioral environments to exist, without resorting to the lowest common denominator.

In a multi-player, game-oriented environment - which might be single or multi-authored - there is a need for another protocol supporting communications between two VRmasters. This too is published in the VRmaster public interface. It may contain both graphic and non graphic related messages.

4.2 Single Author Behavior VRML: a Sketch

We are assuming here that a developer is using Java scripts, although we are strongly recommending a non-specific behavior language, i.e. the system itself should support multiple scripting languages. In that case there will be an additional message dispatching routine for indirection. Both the syntax and the semantics are meant to give a sense of what we would like to have and this should be considered a "work in progress" -- Apple looks forward to receiving comments from VRML list members on how best to implement this area.

This VRML script implements a click loop: there is one geometry visible at any given time; every click on it changes the shape from a box to a cylinder and back to a box.

```
box "firstBox"            ( ... )     # given a name and
instantiated when parsing
box "secondBox"           ( ... )     # same for "secondBox"
cylinder "firstCylinder"  ( ... )     # same for "firstCylinder"
cylinder "secondCylinder" ( ... )     # same for "secondCylinder"

# A definition of a script called "simpleDemo"
# it takes two input parameters labeled
# "geom1" and "geom2"
# No instantiation when parsing this block

JavaScript "simpleDemo" {
    "geom1",
    "geom2",

    public class simpleDemo extends Applet3D
    {
        static View        view;
        static Camera    camera;
        Geometry            box, cylinder;
        int                which;

        static
        {
            // graphic library initialization,
            // view and lights creation

            Q3.Initialize ();
            view = new View ();
            camera = new Camera ();            // a default camera
            view.setCamera (camera);
            ...
        }

        public void init ()
        {
            // get a reference to the geometries using their names
            // passed in as parameters
            String boxName = getParameter (geom1)
            box = GetReferenceByName (boxName);

            String cylinderName = getParameter (geom2)
            cylinder = GetReferenceByName (cylinderName);

            which = 0;
        }

        public void paint (Graphics g)
        {
            int    status;

            view.startRendering ()
            do
            {
                this.submit ();
                status = view.endRendering ();
            }
            while (status == Q3.viewStatusRetraverse);
        }

        public void submit ()
        {
            if (which == 0)    box.submit ();
            else                    cylinder.submit ();
        }

        public mouseDown (Event e, int x, int y)
        {
            // similar to paint code: submit for picking.
            // reset 'which', if necessary
        }
    }
)            # of JavaScript

# Here are two different instantiations of "simpleDemo"
# These must appear after the data and script definition

Make "simpleDemo" {
    geom1     "firstBox",
    geom2     "firstCylinder"
}
```

```
Make "simpleDemo" {
    geom1     "secondBox",
    geom2     "secondCylinder"
}
```

In this example the camera is stationary. If browsing capability was required, the extra functionality should be coded into the script.

4.3 Parsing a Behavior File

Each VRML file is parsed by a VRmaster, with uses the graphic library API to do reads at the object level. The required VRmaster is downloaded, if not available locally, by the generic browser. Functionally, this will work like a content handler in HotJava - a script that is loaded based on the content type. In most browsers a mini-plug-in that loads an interpreter and dispatches the correct VRmaster is probably required.

A VRML file contains graphic data, code definition, and code instantiation. Both in the definition of code and in its instantiation block, references to graphic data and code defined in other files can appear. When such external references are encountered parsing of the external file is initiated but continued asynchronously to parsing of the original file. Except for script instancing, everything else is processed in the order it appears in the file.

Graphic data is instantiated as it is parsed, as in `box "firstBox" { ... }` which gives this instance the name "firstBox". Instantiation, though, does not imply rendering; it is a simple conversion from ASCII representation to an internal representation of the object.

The definition block of a script , as in

```
JavaScript "scriptName" {
    label1, label2, ... labelN      // names of input parameters
    theScript
}
```

does not instantiate the script. The script data is simply bound to its name and held for later use. Forward (named) references and references to objects in other files can appear inside the script. These are resolved at run time.

An instantiation of a script is specified with

```
Make "scriptName" {
    scriptURL   "some URL",         // optional
    label1      "param",
    ...             ...
    lableN      "paramN"
}
```

which loads the script if it is not loaded, creates an instance, and makes a list of string based arguments available to it. The arguments are of the form `label "parameter"`.

Inside a script, a parameter is retrieved by its label. Therefore, the order of parameters is unimportant. A `scriptName` label is mandatory and it specifies the script to instantiate. The optional `scriptURL` indicates the address of an externally defined script. Both scriptName and `scriptURL` are also passed in as parameters to the script. To implement this, aVRmaster is required to implement a

```
getParameter (char *label, char *value)
```

Classes may provide their own static readParameters method. If provided, it overrides the default. It may use are different convention for parsing input parameters in which case labels appearing in the script definition and instantiation may be unnecessary. The example below shows usage of a private read method.

Script instancing takes place after a file has been completely parsed. This applies to external references too: if parsing file a initiates parsing of file b then any scripts in file a are instanced only after file b has been completely parsed and its scripts instantiated. Immediately after instantiation, the VRmaster sends an `'init'` message to the new instance; other VRmaster specific messages are optional.

4.4 Accessing Graphic Data from a Script

>From within scripts objects are accessed by using their string based names. Any VRmaster has to be able to resolve a call of the form

```
object = GetReferenceByName ("URL", "objectName")
```

within a script to an object cookie, which can then be used to access the graphics library API. To do this the VRmaster keeps a list of objects that it had parsed with their name/scope data in the form

```
(objectCookie, URL, name, visibility)
```

In the above call, an empty `URL` denotes the file containing the caller.

The URL can specify a file that had not yet been parsed. In that case, the whole file is first parsed, according to the parsing rules stated above. This means that all data and script instantiations specified in the URL are invoked before resuming with the calling script.

4.5 Visibility Rules

Graphic objects and script definitions can be declared either private or public. As an example,

```
public mesh "table" { ... }
```

`Private` is the default. A `private` object is visible only within the file it is named. A `public` object is visible to any file that can be parsed with the same

VRmaster.

A script always has the option of creating graphic data on the fly, during the `'init'` call or anytime afterwards, without telling the VRmaster about it. Obviously, the visibility of this graphic by another script is governed by language rules regarding the variable it is stored in.

4.6 Instancing a Script within a Script

The `readParameters` method of a script can be used when invoking a script within another script. Instead of invoking the constructor of the class with a new call, one uses make, as in

```
Applet3D script = make (simpleDemo, "geom1", "firstBox", "geom2",
                        "secondBox");
```

As in the data-version of this call, the VRmaster calls the appropriate `readParameters` method and makes these parameters available to the script. One can always use

```
    Applet3D script = new simpleDemo;
```

with the consequence that `getParameter` will return null.

4.7 Multiple Scripting Languages

Support for multiple scripting languages is probably necessary. Furthermore, one should be able to specify a single behavior in a number of languages. This could be used because not all languages are universally available, or because of speed reasons. An example of the latter is Java and a PowerPC shared library. To implement this we can use an IO proxy: a group that instantiates only the first object that it recognizes from a list of objects.

```
GroupBegin (IOProxy)
    PPCMacOSScript(
        # script contents...
    )
    JavaScript (
        # script contents...
    )
GroupEnd
```

A flag tells IOProxy whether it should read in all objects (so as to write them out) or just read in the first recognizable one. Indirection, that is, sending messages, is also necessary to avoid reliance on a particular language.

4.8 Installing New Classes in the Hierarchy

It would be very powerful for an author of behavior files to have a base of classes to derive new classes from. This, of course, assumes an object oriented scripting language. Certainly, when adding new types of objects to the graphic hierarchy one would like to be able to override some of graphic's library internal calls - such as those dealing with duplicating, rendering and picking - so that the whole graphic hierarchy can be dealt with as a whole. A rendering call on the hierarchy would be able to draw the graph in its entirety, as an example.

In an object-oriented scripting language, it would be more convenient (and semantically clear) if the functionality of overriding internal graphic calls could be provided at the scripting language level. One would then subclass and override, all at the script side.

If the graphic library is internally object oriented, for subclassing to work, the internal object hierarchy and the external scripting language hierarchy must be maintained in synchronization. By synchronization we mean that the creation of new classes by subclassing a class that is derived from the graphics library base must be known to both sides. Similarly, for instances.

On the script side one simply subclasses. To convey this info to graphics library one can use one of three techniques:

- Data driven: the library's methods that need to be overridden are specified as entry points into a script class.
- API driven: call the library's `Class_Register`
- Automatic: Java seems to provide us with enough information to be able to call Class_Register during construction (this has yet to be proved). We can require that all loaders give us the class name, its place in the hierarchy, its methods, etc.

4.8 Example

Here we show the general subclassing approach and object instantiation with templates. We are assuming the file contains a number of scripting languages and the respective loaders can not provide us with the information necessary to register the methods on the library's side. If they do, this registration part and the call to Object_NewByTypeName would not be necessary.

This example also illustrates the use of a script's private read method. It overrides the default string based (label, parameter) read method.

```
script "Dog" (
    super "Geometry"        # name of superclass
    5                       # number of methods overridden within scripts

    groupBegin(IOProxy)

        # The first one of the following variations of the Script
        # object that we can execute on this machine is
        # instantiated.  The others are discarded. The chosen script
        # object is then pointed  to by all of the
        # objects within the "binding" container.
```

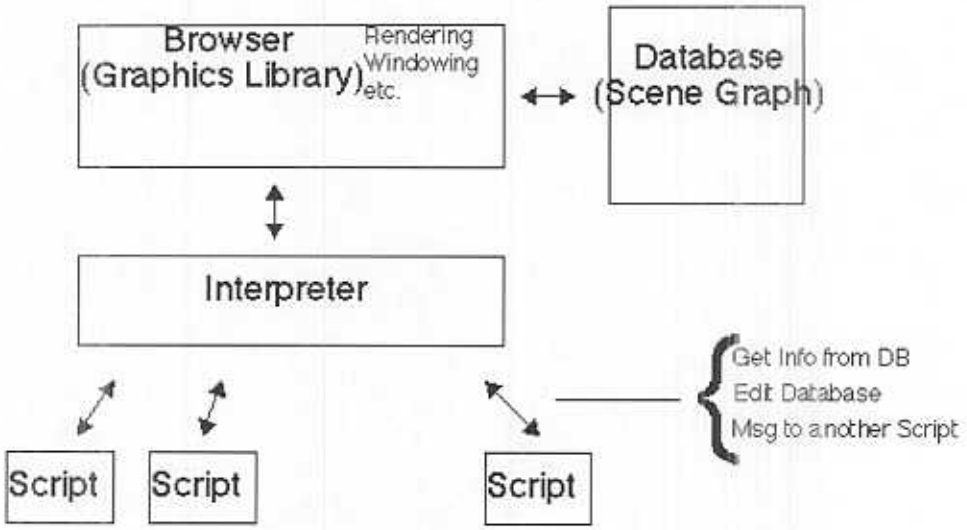
```
        PPCMacOSScript(
            # Per method, we have the method ID and, for this shared lib
script,
            # its entrypoint name as can be resolved by the Shared Library
            # Manager.

            kQ3MethodTypeObjectRead      "Dog::Read"    # a static method
            kQ3MethodTypeObjectSubmit    "Dog::Submit"  # submits this dog
object
            kQ3MethodTypeObjectClick     "Dog::Click"   # action on click
            'bark'                       "Dog::Bark"    # a custom method

            # note: we should define duplicate, dispose method, and perhaps
others
            <size of shared library data>
            <shared library data>
        )

        JavaByteCodeScript(
            # Per method, we have the method ID and, for this Java script, some
            # way of binding to a specific entrypoint.  Let's assume we
switch on
            # an int.

            kQ3MethodTypeObjectRead      1        # a static method
            kQ3MethodTypeObjectSubmit    2        # submits this dog object
            kQ3MethodTypeObjectClick     3        # action on click
            'bark'                       4        # a custom method
            <size of byte code data>
            <byte code data>
        )
    GroupEnd            # of I/O proxy
)                       # of script

mesh "sittingMesh" (...)    # instantiation of a mesh named "sittingMesh"
mesh "runningMesh" (...)    # instantiation of another mesh

Make "Dog" (        # "Dog" script object is instantiated.
    1               # parameter read in by the class' read method
)

Make "Dog" (        # Second "Dog" is instantiated with different parameter
    0
)
```

Here is the Java script for the bytecode above:

```
JavaByteCodeScript "Dog" (     # we assume the bytes are here

    public class Dog extends Geometry
    {
        // some constants:
        public final int            running = 0;
        public final int            walking = 1;

        private int         state;
        private Geometry    sittingMesh, runningMesh;
        private View        view;

        public Dog (int initialState)
        {
            Super (Q3.NewByTypeName("Dog"));    // tells the library to
                                                // create an instance

            initialize state;
            state = initialState;

            // get handles to the geometries
            sittingMesh = GetReferenceByName ("sittingMesh");
            runningMesh = GetReferenceByName ("runningMesh");
        }

        // Read method
        public void ReadParameters (File file)
        {
            int initialState = Q3.ReadInt (file);
            new Dog (initialState);
        }

        // Overrides the Geometry 'Submit'
        public int Submit (View view)
        {
            if (state == running)    runningMesh.Submit (view);
            else                            sittingMesh.Submit (view);
```

```
            return ....
        }

        // Overrides a browser supplied 'click' method
        public boolean mouseDown (Event e, int x int y)
        {
            if (state == running)    state = walking;
            else                     state = running;
        }
    }
}           # of JavaScript
```

4.9 Comparison with Moving Worlds



MW is a data-oriented description of a dynamic environment. The **Out Of This World** proposal is code oriented--like PostScript. It uses data to describe the initial configuration of the system, and code to describe the dynamic interactions. As an interesting example, our approach to detecting collisions, is to grant scripts access to library calls implementing that functionality, instead of specifying it in the scene graph.

The MW proposal includes a fixed, heavyweight 3D browser that is responsible for I/O, drawing, picking, and camera control. Apple suggests using environment-specific, lightweight 3D browsers, implemented as scripts that are invisibly downloaded on demand. With this approach, different sets of VRML files could use different types of browsers.

Our proposal does not require a centralized database. Scripts are responsible for figuring out important data relationships, and for scheduling and implementing redraws. In addition, there are no static routes which require special APIs to change on the fly. There is more control and inherent flexibility.

For subclassing, the Apple proposal suggests an object oriented pyramid of classes as a good starting base for writing scripts. For example, one solution is to use a hierarchy of Java classes with methods that can be overridden.

In general, the MW proposal seems geared to dynamic walkthrough environments. Apple's proposal is better suited for highly dynamic, game-like interfaces.

5. Change Log

- 01/08/96 ... Created initial document
- 01/15/96 ... Appended initial behavior specification
- 01/31/96 ... Revised behavior specification
- 02/02/96 ... Submission to VAG and VRML community