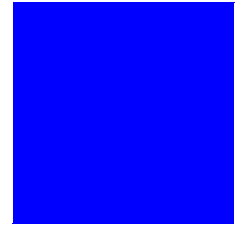


# PLATINUM



---

## WorldView for Developers

### User Guide

Version 2.1

**Title and Publication Number**

PLATINUM Publication Number: WVD-X-210-UG00-00

Printed: June 1, 1998

Information in this guide is subject to change without notice and does not constitute a commitment on the part of PLATINUM *technology, inc.* It is supplied on an "as is" basis without any warranty of any kind, either explicit or implied. Information may be changed or updated in this guide at any time.

**Copyright Information**

PLATINUM WorldView for DEvelopers is ©copyright 1997- 1998 by PLATINUM *technology, inc.* and its subsidiaries. This guide is ©copyright 1997 - 1998 by PLATINUM *technology, inc.*, and its subsidiaries and may not be reproduced in whole or in part, by any means, without the written permission of PLATINUM *technology, inc.* and its subsidiaries.

Names marked ™ or ® and other company and product names may be trademarks or registered trademarks of their respective vendors or organizations.

**Mailing Address**

PLATINUM *technology, inc.*  
1815 South Meyers Road  
Oakbrook Terrace, Illinois  
60181-5235

# Table of Contents



## Preface

|  |      |
|--|------|
| Philosophy .....                           | x    |
| Contacting Technical Support .....         | xi   |
| Technical Support Programs .....           | xi   |
| Complimentary Support .....                | xi   |
| Fee-Based Technical Support .....          | xii  |
| Before You Contact Technical Support ..... | xiii |
| Contacting PLATINUM .....                  | xvi  |
| About This Guide .....                     | xvii |
| Conventions .....                          | xix  |
| Related Publications .....                 | xx   |

## 1 • WorldView for Developers Features

|                                  |     |
|----------------------------------|-----|
| Features .....                   | 1-2 |
| WorldView Control .....          | 1-3 |
| The Power of VRML .....          | 1-4 |
| ActiveX Support .....            | 1-4 |
| Embedding Support .....          | 1-4 |
| The WorldView Product Line ..... | 1-5 |

## 2 • Getting Started

|   |     |
|---|-----|
| Installation .....                        | 2-2 |
| System Requirements .....                 | 2-2 |
| Software Requirements .....               | 2-2 |
| Installing WorldView for Developers ..... | 2-3 |
| Runtime Installation .....                | 2-5 |
| Uninstallation .....                      | 2-6 |

|  |      |
|--|------|
| <b>3 • The WorldView Browser</b>               |      |
| The WorldView Browser                          | 3-3  |
| Support for VRML 2.0                           | 3-3  |
| VRML 2.0 Nodes                                 | 3-4  |
| data: Protocol                                 | 3-11 |
| WorldView's Full Color Setting                 | 3-12 |
| Support for JavaScript                         | 3-13 |
| JavaScript and the VRML Console                | 3-13 |
| Variable Scoping                               | 3-13 |
| Type Conversion in JavaScript Expressions      | 3-13 |
| Unsupported Functions                          | 3-14 |
| Support for the Java EAI                       | 3-15 |
| Requirements for Internet Explorer             | 3-15 |
| Using WorldView's Java EAI in Web Browsers     | 3-15 |
| Support for Java in Script Nodes               | 3-17 |
| Java 1.1 Support                               | 3-17 |
| System.out and System.err                      | 3-17 |
| Security                                       | 3-17 |
| Using the EMBED tag in HTML documents          | 3-18 |
| WorldView EXTERNPROTO Extensions               | 3-19 |
| BillboardText                                  | 3-19 |
| BrowserSettings                                | 3-20 |
| PopupText                                      | 3-23 |
| StreamingAudioClip                             | 3-26 |
| DirectX Files                                  | 3-28 |
| <b>4 • WorldView for Developers Containers</b> |      |
| Introduction                                   | 4-3  |
| Macromedia Director Xtras                      | 4-3  |
| Microsoft Visual C++                           | 4-4  |
| Adding the Component                           | 4-4  |
| Using the Component in a Window                | 4-5  |
| Controlling the Component                      | 4-5  |
| Adding the Component to a Dialog Box           | 4-6  |
| Embedding in C++ at Runtime                    | 4-6  |

|   |      |
|---|------|
| Microsoft Java VM .....   | 4-7  |
| Embedding the WorldView Component .....                           | 4-8  |
| Accessing the COM API .....                                       | 4-9  |
| Using Standard Java EAI .....                                     | 4-10 |
| Microsoft Java and ActiveX Integration .....                      | 4-10 |
| Embedding in J++ at Runtime .....                                 | 4-11 |
| Samples .....   | 4-11 |
| Microsoft Visual Basic .....                                      | 4-12 |
| Adding the Component .....  | 4-12 |
| Positioning and Resizing the Control in a Visual Basic Form ..... | 4-13 |
| Getting a Reference to the Control .....                          | 4-13 |
| Getting a Reference to the VrmlBrowser Object .....               | 4-14 |
| Adding a VRML Primitive to the Scene .....                        | 4-14 |
| Removing a Node from the Scene .....                              | 4-16 |
| Changing the Fields of a Node in a VRML Scene .....               | 4-17 |
| Receiving Events from a VRML Scene .....                          | 4-18 |
| Passing Arrays to Methods .....                                   | 4-19 |
| Invoking an OCX from the Script node .....                        | 4-20 |
| Embedding in Visual Basic at Runtime .....                        | 4-21 |
| Samples .....   | 4-21 |
| Working with Web Pages: HTML and Java .....                       | 4-22 |
| Getting a Reference to the Control .....                          | 4-22 |
| Embedding a WorldView for Developers File in an HTML Page .....   | 4-23 |
| Getting a Reference to the VrmlBrowser Object .....               | 4-24 |
| Adding a VRML Primitive to the Scene .....                        | 4-26 |
| Removing a Node from the Scene .....                              | 4-27 |
| Changing the Fields of a Node in a VRML Scene .....               | 4-28 |
| Receiving Events from the VRML Scene .....                        | 4-29 |

- 5 • WorldView for Developers Runtimes**
  - Generating Runtime Applications ..... 5-2
  - Requirements for Runtime Applications ..... 5-2
  - Including your own Help files ..... 5-5
  - WorldView License Agreement ..... 5-5
  - Embedding WorldView for Developers at Runtime ..... 5-6
    - Embedding in C++ at Runtime ..... 5-6
    - Embedding in J++ at Runtime ..... 5-7
    - Embedding in Visual Basic at Runtime ..... 5-7
  
- 6 • The WorldView COM Object**
  - Introduction ..... 6-2
    - IWorldView Interface ..... 6-3
    - IWorldViewDeveloper Interface ..... 6-4
  
- 7 • WorldView OLE Automation Interface**
  - WorldView OLE Automation Interface ..... 7-2
  
- 8 • External Authoring Interface using COM**
  - Introduction ..... 8-2
  - Using WorldView's COM API from C++ ..... 8-3
  - Using COM from Java ..... 8-5
  - Using WorldView's COM API from other Languages ..... 8-6
  - WorldView for Developers COM API Library ..... 8-6

## 9 • WorldView for Developers Objects

|   |      |
|---|------|
| WorldView External Scripting Objects' Structure ..... | 9-9  |
| VrmlBaseNode Objects .....                            | 9-11 |
| VrmlBaseNode .....                                    | 9-12 |
| VrmlNode .....  | 9-13 |
| VrmlScriptNode .....                                  | 9-15 |
| VrmlBrowser Object .....                              | 9-17 |
| VrmlEvent Object .....                                | 9-20 |
| VrmlEventOutObserver Object .....                     | 9-21 |
| VrmlField Objects .....                               | 9-22 |
| VrmlField .....                                       | 9-23 |
| VrmlConstField .....                                  | 9-25 |
| VrmlConstMFColor .....                                | 9-26 |
| VrmlConstMFFloat .....                                | 9-28 |
| VrmlConstMField .....                                 | 9-29 |
| VrmlConstMFInt32 .....                                | 9-31 |
| VrmlConstMFNode .....                                 | 9-33 |
| VrmlConstMFRotation .....                             | 9-35 |
| VrmlConstMFString .....                               | 9-37 |
| VrmlConstMFTime .....                                 | 9-39 |
| VrmlConstMFVec2f .....                                | 9-41 |
| VrmlConstMFVec3f .....                                | 9-43 |
| VrmlConstSFBool .....                                 | 9-45 |
| VrmlConstSFColor .....                                | 9-47 |
| VrmlConstSFFloat .....                                | 9-49 |
| VrmlConstSFImage .....                                | 9-51 |
| VrmlConstSFInt32 .....                                | 9-53 |
| VrmlConstSFNode .....                                 | 9-54 |
| VrmlConstSFRotation .....                             | 9-56 |
| VrmlConstSFString .....                               | 9-58 |
| VrmlConstSFTime .....                                 | 9-59 |
| VrmlConstSFVec2f .....                                | 9-61 |
| VrmlConstSFVec3f .....                                | 9-63 |
| VrmlMFColor .....                                     | 9-65 |
| VrmlMFFloat .....                                     | 9-68 |

|   |       |
|---|-------|
| VrmIMField .....                        | 9-71  |
| VrmIMFInt32 .....                       | 9-73  |
| VrmIMFNode .....                        | 9-75  |
| VrmIMFRotation .....                    | 9-79  |
| VrmIMFString .....                      | 9-82  |
| VrmIMFTime .....                        | 9-85  |
| VrmIMFVec2f .....                       | 9-88  |
| VrmIMFVec3f .....                       | 9-91  |
| VrmISFBool .....                        | 9-94  |
| VrmISFColor .....                       | 9-96  |
| VrmISFFloat .....                       | 9-98  |
| VrmISFImage .....                       | 9-99  |
| VrmISFInt32 .....                       | 9-102 |
| VrmISFNode .....                        | 9-103 |
| VrmISFRotation .....                    | 9-105 |
| VrmISFString .....                      | 9-107 |
| VrmISFTime .....                        | 9-109 |
| VrmISFVec2f .....                       | 9-111 |
| VrmISFVec3f .....                       | 9-113 |
| VrmObjectFactory Interface .....        | 9-115 |
| VrmScriptImplementation Interface ..... | 9-125 |

**10 • Disabled Interfaces**

|                           |      |
|---------------------------|------|
| Disabled Interfaces ..... | 10-2 |
|---------------------------|------|

**11 • Error Handling**

|                       |      |
|-----------------------|------|
| Introduction .....    | 11-2 |
| Returned Errors ..... | 11-2 |



**A • Sample Applications**

|   |     |
|---|-----|
| Sample Applications .....   | A-2 |
| Invoking an OCX from a Script node: OCXDemo .....                   | A-3 |
| Software Requirements .....   | A-3 |
| Installation .....  | A-3 |
| Components .....  | A-3 |
| Instructions .....  | A-4 |
| Embedding WorldView in a C++ Application: Tiny3D .....              | A-4 |
| Software Requirements .....   | A-5 |
| Installation .....  | A-5 |
| Components .....  | A-5 |
| Instructions .....  | A-6 |
| Embedding WorldView in a Java Application: JWorldViewContainer .... | A-7 |
| Software Requirements .....   | A-7 |
| Installation .....  | A-7 |
| Components .....  | A-7 |
| Instructions .....  | A-8 |

**Index**





---

# Preface

Welcome to WorldView for Developers.

WorldView for Developers is an enhanced version of PLATINUM's industry-acclaimed VRML 2.0 browser: WorldView. WorldView for Developers offers you the ability to integrate WorldView's advanced 3D rendering technology into your own applications.

This manual is designed to acquaint you with the use of WorldView for Developers. It describes inserting the WorldView for Developers control into supported containers, generating runtime applications which embed the control, and lists and defines all objects, interfaces, properties, and methods available through the control. This manual also provides several working samples to demonstrate specific features and functions of WorldView for Developers, as well as a complete list of errors returned by the product.

## Philosophy

PLATINUM *technology, inc.* (PLATINUM), one of the top 20 software vendors worldwide, supports IT organizations by partnering with companies to manage and improve business software operations. PLATINUM's products and services help ensure that companies obtain a better return on their enterprise by maximizing internal IT service levels and focusing enterprise computing efforts on solutions that create a competitive advantage in the marketplace. PLATINUM provides integrated solutions for database management, systems management, business intelligence, application lifecycle, data warehousing, and Year 2000 initiatives.

PLATINUM's VRML software is one part of PLATINUM's ongoing, long-term strategy to help IT organizations gain better competitive advantage through the deployment of next-generation computing solutions and next-generation user interfaces. PLATINUM recognizes the value of VRML as a breakthrough technology and is committed to bringing the value of VRML to the mainstream, to developers, and to enterprise IT organizations. Current VRML product offerings are specifically designed to introduce individuals and companies to the power of VRML. PLATINUM's VRML services group partners with world-class, global organizations that wish to deploy sophisticated applications of VRML technology. PLATINUM is also integrating VRML into other PLATINUM products, services and technology offerings.

# Contacting Technical Support

## Technical Support Programs

PLATINUM *technology, inc.* has developed a mix of complimentary and fee-based technical support programs for WorldView for Developers customers in the U.S. and Canada. These programs have been designed to deliver fast, flexible, and comprehensive service to all WorldView for Developers users.

## Complimentary Support

**User Guide.** This manual should be considered the first step in troubleshooting any problems encountered using WorldView for Developers.

**On-line Help.** WorldView for Developers' on-line help replicates the information found in this User Guide. On-line help is provided simply for your convenience.

**ReadMe.** This file contains a list of known issues and solutions that were identified prior to this release. The ReadMe file is available from the WorldView for Developers' Program Menu.

**WorldView Help Index.** This file provides access to the WorldView browser help, including the User's Guide, Developer's Guide, ReadMe, and FAQ for that product. This file is installed with WorldView for Developers, in the Help directory, and should be consulted when browser specific problems are encountered.

**VRML 97 Specification.** We recommend that any developer using this product also consult the VRML 97 Specification, which may be found at <http://www.vrml.org/Specifications/VRML97>.

**Telephone Support.** Registered WorldView for Developers users receive 30 days of free telephone support. A valid WorldView for Developers registration number is required for this option. See the support document included in the WorldView for Developers package for additional information. This option is not available for the trial version.

**Email Support.** Registered WorldView for Developers users may submit support questions via email to the PLATINUM support Web site (<http://support.platinum.com>). A valid WorldView for Developers registration number is required for this option. See the support document included in the WorldView for Developers package for additional information. This option is not available for the trial version.

## **Fee-Based Technical Support**

Registered users of WorldView for Developers can take advantage of PLATINUM's fee-based support programs when their initial 30 days of free telephone support has expired. A valid WorldView for Developers registration number is required to use this option. See the support document provided in the WorldView for Developers package for additional information. Fee-based programs are not available for the trial version.

**Telephone Support.** \$40 (US) per 20-minute call via 900 number, \$2 (US) per minute thereafter. 8:00AM to 7:00PM CST. Monday-Friday.

All offerings, terms and prices are subject to change. For the latest information on fee-based technical support options, including contact information for WorldView for Developers technical support, refer to the Frequently Asked Questions (FAQ) pages for WorldView for Developers, which are available at the PLATINUM support Web site (<http://support.platinum.com>).

## Before You Contact Technical Support

### If you think you need technical support:

- Please read everything relevant to the problem in the User Guide or on-line help. Check the User Guide and on-line help indices for more references to the topic. More information on a procedure or feature may be found in a separate section.
- Refer to the WorldView for Developers ReadMe file for a list of known issues and solutions that were identified prior to WorldView for Developers' initial release.
- Refer to the WorldView browser help index for more information on browser functionality, including a list of know issues and solutions specific to the WorldView browser.
- Visit the Frequently Asked Questions (FAQ) pages on the PLATINUM *technology, inc.* support Web site (<http://support.platinum.com>) for a complete list of known issues and solutions identified after this WorldView for Developers' release.
- If something used to work and now no longer works, try to identify what may have changed. Perhaps you installed new software or changed some settings.
- Create a new file and try to reproduce the problem there. If the problem does not appear in the new file, compare the new file with the old file to identify and eliminate the differences.
- Copy the file, and begin deleting unrelated sections until the problem is solved to identify the exact location of the problem

---

**Note** • Many issues that you encounter may be solved by following the steps listed above.

---

**If you still need help:**

If you still need help at this point, a little preparation can save you time and money, and allow the support representative to help you more quickly. Please complete the following checklist before contacting technical support:

- Try to narrowly define the problem so that you can repeat the steps that lead to it and specifically identify when and how it occurred. The more clearly the problem is defined the better our support representative will be able to provide a solution.
- Be able to provide the following information:
  - Product name, version number, and registration number
  - Development environment name and version number
  - Type of computer, such as Pentium or Pentium Pro, local-bus or remote bus
  - Graphics card manufacturer, model, and driver version number
  - Sound card manufacturer and model name
  - Web browser name and version number
  - VRML browser name and version number
  - Operating system name and version number
  - Amount of memory installed
  - Amount of free hard disk space
  - Screen resolution (screen size in pixels, for example, 1024 by 768)
  - Screen color depth (for example, 16-bit, or thousands of colors)
  - A list of external devices connected to the computer
  - Brief description of the problem or error, and the specific text of any error messages



This information will help us pinpoint and solve your problem more quickly.

Please note that technical support can only answer installation, configuration, and other questions specific to WorldView for Developers. For questions concerning VRML, 3D graphics, or other software products, we recommend that you visit the appropriate vendor's support site.

**To send Email to PLATINUM Technical Support, use:**

Internet [support-worldview@platinum.com](mailto:support-worldview@platinum.com)

---

**Note** • WorldView for Developers users must preface the subject line of email messages to [support-worldview@platinum.com](mailto:support-worldview@platinum.com) with their product registration number. This number can be found on either the registration card or the license agreement, both of which are available in the WorldView for Developers package.

---

**To contact PLATINUM Technical Support, use:**

|                          |                              |
|--------------------------|------------------------------|
| USA or Canada, toll free | 800-655-9983 (first 30 days) |
| USA or CANADA, fee-based | 900-555-PLAT (after 30 days) |

---

**Note** • Telephone support is available only to registered WorldView for Developers users.

---

**For WorldView for Developers ordering information, call:**

|                          |              |
|--------------------------|--------------|
| USA or Canada, toll free | 800-373-7528 |
| Illinois                 | 630-620-5000 |

## Contacting PLATINUM

You can contact us with any questions or problems you have. You will be directed to an experienced software engineer familiar with *PLATINUM WorldView for Developers*.

### **For product assistance or information, contact:**

|                          |   |
|--------------------------|---|
| USA or Canada, toll free | 800-442-6861  |
| Illinois                 | 630-620-5000  |
| FAX                      | 630-691-0708 or 630-691-0406                                  |
| Internet                 | info@platinum.com   |
| World Wide Web           | <a href="http://www.platinum.com">http://www.platinum.com</a> |

### **For general information on PLATINUM *technology, inc.*, contact:**

|                          |   |
|--------------------------|---|
| USA or Canada, toll free | 800-442-6861  |
| Illinois                 | 630-620-5000  |
| FAX                      | 630-691-0708  |
| Internet                 | info@platinum.com   |
| World Wide Web           | <a href="http://www.platinum.com">http://www.platinum.com</a> |

### **Our Mailing Address is:**

PLATINUM *technology, inc.*  
1815 South Meyers Road  
Oakbrook Terrace, IL 60181-5235

## About This Guide

The *PLATINUM WorldView for Developers User Guide* explains features and interfaces available in WorldView for Developers.

This guide assumes that the appropriate *PLATINUM WorldView for Developers* components have been installed on your machine. The instructions for installing the product are in the Installation Guide.

| Ch. No. | Chapter Name                               | Content Description  |
|---------|--|--|
| 1       | <i>WorldView for Developers Features</i>   | Introduces the functions and features of WorldView for Developers.   |
| 2       | <i>Getting Started</i>                     | Outlines the installation and uninstallation process for both WorldView for Developers and its generated runtime applications. |
| 3       | <i>The WorldView Browser</i>               | Provides a general introduction to the requirements, properties, and extensions of the WorldView browser.                      |
| 4       | <i>WorldView for Developers Containers</i> | Describes loading the control, generating runtime applications, and techniques specific to supported development containers.   |
| 5       | <i>WorldView for Developers Runtimes</i>   | Lists requirements for runtime applications generated using WorldView for Developers.  |
| 6       | <i>The WorldView COM Object</i>            | Describes the WorldView COM object, and lists and defines its properties and methods.  |
| 7       | <i>WorldView OLE Automation Interface</i>  | Defines the OLE Automation Interface, a superset of the IWorldViewDeveloper interface visible to Visual Basic users.           |

| <b>Ch. No.</b> | <b>Chapter Name</b>                           | <b>Content Description</b>   |
|----------------|---|--|
| <b>8</b>       | <i>External Authoring Interface using COM</i> | Describes accessing WorldView's External Authoring Interface using Microsoft's Component Object Model (COM).       |
| <b>9</b>       | <i>WorldView for Developers Objects</i>       | Describes and defines the Objects available in WorldView for Developers External Authoring Interface.              |
| <b>10</b>      | <i>Disabled Interfaces</i>                    | Describes interfaces which you may be able to see in Visual Basic, but which are disabled, and will return errors. |
| <b>11</b>      | <i>Error Handling</i>                         | Lists and defines errors returned using WorldView for Developers.  |
| <b>A</b>       | <i>Sample Applications</i>                    | Offers several sample runtime application samples for your use.  |

---

## Conventions

Some or all of the following conventions appear in this guide:

| Symbol or Type Style   | Represents   | Example  |
|------------------------|--|--|
| <b>Bold</b>            | a new term   | ...called a <b>source object</b> .                 |
| <i>Alternate color</i> | (online only) hotlinked cross-references to other sections in this guide; if you are viewing this guide online in PDF format, you can click the cross-reference to jump directly to its location | ...see <i>Chapter 3, Data Migration</i> .          |
| <i>Italic</i>          | words that are emphasized  | ...the entry <i>after</i> the current entry...     |
|                        | the titles of other documents  | <i>PLATINUM General Facilities Reference Guide</i> |
|                        | syntax variables   | <i>COPY filename</i>                               |
| Monospace              | directories, file names, command names, computer code  | &HIGHLVL.SRCLIB                                    |
|                        | computer screen text, system responses, command line commands  | Copy file? Y/N                                     |
| Monospace bold         | what a user types  | ...enter RUN APP.EXE in the Application field      |
| < >                    | the name of a key on the keyboard  | Press <Enter>.                                     |
| ▶                      | choosing a command from a cascading menu   | File ▶ Import ▶ Object                             |

## **Related Publications**

As you use this *PLATINUM WorldView for Developers User Guide*, you might find it helpful to have these additional files available for reference:

- *PLATINUM WorldView for Developers Installation Guide*
- *PLATINUM WorldView browser help files*, available in HTML format from the *Help* directory of your WorldView for Developers installation.
- *The VRML 97 Specification*. The VRML specification is available from <http://www.vrml.org/Specifications/VRML97>.



---

# WorldView for Developers Features

|   |            |
|---|------------|
| <b>Features</b> .....                   | <b>1-2</b> |
| <b>WorldView Control</b> .....          | <b>1-3</b> |
| <b>The Power of VRML</b> .....          | <b>1-4</b> |
| <b>ActiveX Support</b> .....            | <b>1-4</b> |
| <b>Embedding Support</b> .....          | <b>1-4</b> |
| <b>The WorldView Product Line</b> ..... | <b>1-5</b> |

## Features

### **...INTERACTIVE 3D THROUGH ACTIVEX**

WorldView for Developers is an enhanced version of PLATINUM's industry-acclaimed VRML 2.0 browser: WorldView. WorldView for Developers offers you the ability to integrate WorldView's advanced 3D rendering technology into your own applications.

WorldView for Developers is a superset of the WorldView browser. Because WorldView for Developers is based on WorldView, all of the features that have made WorldView so popular are included. These include:

- Complete VRML 2.0 specification compliance
- Direct3D hardware acceleration
- Simple, intuitive user interface
- Windows 95 and Windows NT 4.0 compatibility
- Programmability via:
  - Java External Authoring Interface (EAI)
  - Java in Script nodes
  - JavaScript



## WorldView Control

Based on the successful WorldView browser, WorldView for Developers exposes a variety of properties, methods, and events to the developer so that they may fully control their application's behaviors.

**Navigation:** You may disable WorldView's built-in navigation system and programmatically set and manipulate the user's view of the VRML scene. Developers may also hide WorldView's toolbars and menus, and replace them with their own.

**External Authoring Interface (EAI):** WorldView for Developers recreates the EAI with ActiveX COM interfaces, so developers using any programming language, such as C++ or Visual Basic, may take advantage of its features. A draft standard for interfacing external Java applications to a VRML browser, EAI allows programs running outside of the browser to create and manipulate objects in the VRML scene interactively at runtime, and to interact with "scripts" in the VRML scene.

**Rendering Control:** WorldView for Developers allows you to alter the display of the scene, including switching between different color and shading modes. This switching may take place on the fly.

**Scene Loading:** You can load scenes into WorldView by supplying a file or an URL. Data may be loaded from a variety of sources, including databases and dynamically-created files.

## **The Power of VRML**

Just as HTML is the standard for “rich text” in Internet-aware applications, VRML is the standard for interactive 3D. VRML provides not only a definition for 3D geometry and lighting, but also enables integration of audio, images, and even Java and Javascript programs to control the “scene” and the user’s interaction with it. The VRML viewer provides a complete runtime environment for simulations. This power is provided transparently to the developer using WorldView. At a minimum, developers provide a pointer to a VRML file and WorldView does the rest. Many alternatives for exercising finer control are also provided.

## **ActiveX Support**

WorldView for Developers provides complete support for Microsoft’s ActiveX (OCX) standard for software components, enabling it to be used by applications developed in virtually any Windows development environment, including Microsoft’s Visual Basic and Visual C++. By supporting the full OCX interface (not simply the “lite” ActiveX interface used by some ActiveX controls) WorldView for Developers is able to integrate seamlessly with the user’s development environment. Both early (compile time) and late (runtime) binding is also supported.

## **Embedding Support**

Until now, writing a 3D graphics application meant using a low-level rendering library like OpenGL or Direct3D, or worse, “hand coding.” WorldView for Developers makes this unnecessary by doing the low-level 3D work for you. By embedding WorldView for Developers in your application, you can take advantage of WorldView’s advanced 3D technology. This saves you an enormous amount of development time, and lets you concentrate your resources where you most want to spend them: content.

## The WorldView Product Line

The WorldView Browser, WorldView for Developers, and WorldView for Developers runtime copies are three parts of the same product line, with the WorldView browser as its basis. Either WorldView for Developers or one of its runtime copies may be installed on a machine with the WorldView browser.

---

**Note** • It is important to note that installing a WorldView for Developers runtime application on your machine will overwrite your copy of WorldView for Developers, making it necessary for you to reinstall.

---

**WorldView** is a VRML browser plugin for Netscape Navigator and Internet Explorer. It is launched when a file with the model/vrml MIME type and/or the .wrl extension is loaded.

**WorldView for Developers** works in development environments such as Macromedia's Director, Visual Basic, and Visual C++ to allow authors to create applications using WorldView technology.

**WorldView for Developers runtimes** are distributed with applications created using WorldView for Developers. A user may have any number of applications that install the ActiveX control on one machine. Installing an application that uses the runtime will not install the functionality of the full WorldView for Developers SDK.

**WorldView Professional** is an extended version of the WorldView browser, which allows users to embed 3D VRML objects into Microsoft Office applications. Only one of these two applications (WorldView or WorldView Professional) may be present on a machine at any time. Both function as a VRML viewer from within an internet browser.

## ■ **WorldView for Developers Features**

---

### *The WorldView Product Line*

All WorldView products require that the following components be installed:

- dx5eng.exe (DirectX 5) 3.62 MB

<http://www.microsoft.com/msdownload/directx/dxf/enduser5.0/default.htm>

- dxmweb.exe (DirectShow) 3.83 MB

<http://www.microsoft.com/directx/resources/dx5mediaruntime.htm>

- msjavax86.exe (VM for Java) 3.02 MB

<http://www.microsoft.com/java/vm/vmdownload.htm>

---

# Getting Started

This chapter describes installing WorldView for Developers on your machine, and the components necessary to install the generated runtime applications on your customers' machines.

|   |            |
|---|------------|
| <b>Installation</b> .....                 | <b>2-2</b> |
| System Requirements .....                 | 2-2        |
| Software Requirements .....               | 2-2        |
| Installing WorldView for Developers ..... | 2-3        |
| Runtime Installation .....                | 2-5        |
| <b>Uninstallation</b> .....               | <b>2-6</b> |

## Installation

### System Requirements

- Windows 95 or NT 4.0
- 120 mhz Pentium PC (minimum)
- 32 MB RAM (minimum)
- 3D accelerator card

### Software Requirements

- Java VM (for including Java)
- DirectShow (for including Movie Texture)
- DirectX 5

---

**Note** • The Java VM is required for WorldView for Developers only if you are using Java in your application. If it is not yet installed on your machine, it may be obtained from: <http://www.microsoft.com/java/vm/vmdownload.htm>. If your application uses Java, the Java VM will also be required by your end users.

---

## Installing WorldView for Developers

To install WorldView for Developers,

- 1** exit all other Windows programs,
- 2** click on the `WVDev_Setup.exe` file, and
- 3** follow the installation Wizard. You will be prompted to enter your name and company in the provided fields, then select one of two options: No Password, or Password.
  - a** Select No Password to install an evaluation copy of WorldView for Developers. This will install a copy of WorldView for Developers which will expire 30 days after installation. (The time you have left will be shown in the About Box for your convenience.)
  - b** Select Password if you have purchased WorldView for Developers, and enter your password in the field provided. This will install a fully enabled version on your machine.
- 4** Follow the Install Wizard prompts to complete installation.

By default, the WorldView for Developers installer (`WVDev_Setup.exe`) copies the following files to your machine, in the locations listed:

```
WorldView for Developers 2.1\  
  axWorldView.zip  
  License.txt  
  LicenseEnabler.exe  
  MMAR.AX  
  MMVR.AX  
  ReadMe.txt  
  runtimeGenerator.exe  
  WorldView.lic  
  WorldView.ocx
```

```
WorldView for Developers 2.1\Distribution\  
  Help\ (WorldView browser Help files)  
  Axdist.exe (a third party installer)  
  WVLICENSE.txt
```

WorldView for Developers 2.1\Help\  
Images\ **(all files)**  
NETWH32.dll  
**WorldView browser HTML help files**  
WVforDev.cnt  
WVforDev.hlp  
WVforDev.pdf

WorldView for Developers 2.1\Include\  
WVInterface.h **(header file for C++ users)**

WorldView for Developers 2.1\Lib\  
WVGuid.lib **(lib file for C++ users)**

WorldView for Developers 2.1\Samples\  
JWorldViewContainer\ **(all files)**  
OCXDemo\ **(all files)**  
Tiny3D\ **(all files)**

The Compact installation does not include the Help directory and files, the Samples directory and files, or the ReadMe text in the root directory.

The `WorldView.lic` file is automatically generated during installation, and is used to control both the 30 day time limit on evaluation copies of WorldView for Developers, and to fully enable the software once it has been purchased. This file may not be redistributed.

The installation also creates these Start Menu shortcuts:

Start Menu ▶ PLATINUM WorldView for Developers 2.1 ▶  
Help (launches `WVforDev.hlp`)  
ReadMe (launches `ReadMe.txt`)  
BuyMe (launches `licenseEnabler.exe`)  
Copy Key to Clipboard (launches `runtimeGenerator.exe`)

The BuyMe item is used to enable a trial version of WorldView for Developers. After contacting our sales department, and receiving a password, simply select this item from the start menu and follow the directions to fully enable your copy of WorldView for Developers.



The Copy Key to Clipboard item is used to enable runtimes generated using C++ or J++. Simply select this item from the start menu to have the enabling string copied to the clipboard, then paste into your application.

This installer does not reboot.

## Runtime Installation

Applications created using WorldView for Developers require the same components as WorldView:

- DirectX 5,
- DirectShow (if the application contains Movie Textures), and the
- Java VM (if the application contains Java).

These components are not included in the WorldView for Developers installer, and should be installed by the application itself, if necessary. These components may be licensed from Microsoft for free. See:

<http://www.microsoft.com/java/vm/vmdownload.htm>,

<http://www.microsoft.com/directx/resources/endddl.htm>, and

<http://www.microsoft.com/java/vm/vmdownload.htm> for details.

Your application installer must also run the WorldView for Developers Runtime installer. Run `WVDevRt_Setup.exe` to install the OCX and DLLs required by WorldView.

## Uninstallation

To uninstall, simply select Add/Remove Programs from the Control Panels folder, select PLATINUM WorldView for Developers from the program list, and click Remove.

If you have used VB with WorldView for Developers, two files may have been generated in the WorldView for Developers folder:

WORLDVIEWLib.TWD, and

WORLDVIEW.oca.

If these files are present, you must manually delete the WorldView for Developers folder to complete uninstallation.

---

# The WorldView Browser

This chapter provides a general introduction to the requirements, properties, and extensions of the WorldView browser.

|  |             |
|--|-------------|
| <b>The WorldView Browser</b> .....               | <b>3-3</b>  |
| <b>Support for VRML 2.0</b> .....                | <b>3-3</b>  |
| VRML 2.0 Nodes .....                             | 3-4         |
| data: Protocol .....                             | 3-11        |
| WorldView's Full Color Setting .....             | 3-12        |
| <b>Support for JavaScript</b> .....              | <b>3-13</b> |
| JavaScript and the VRML Console .....            | 3-13        |
| Variable Scoping .....                           | 3-13        |
| Type Conversion in JavaScript Expressions .....  | 3-13        |
| Unsupported Functions .....                      | 3-14        |
| <b>Support for the Java EAI</b> .....            | <b>3-15</b> |
| Requirements for Internet Explorer .....         | 3-15        |
| Using WorldView's Java EAI in Web Browsers ..... | 3-15        |

|  |             |
|--|-------------|
| <b>Support for Java in Script Nodes</b> .....      | <b>3-17</b> |
| Java 1.1 Support .....                             | 3-17        |
| System.out and System.err .....                    | 3-17        |
| Security .....                                     | 3-17        |
| <b>Using the EMBED tag in HTML documents</b> ..... | <b>3-18</b> |
| <b>WorldView EXTERNPROTO Extensions</b> .....      | <b>3-19</b> |
| BillboardText .....                                | 3-19        |
| BrowserSettings .....                              | 3-20        |
| PopupText .....                                    | 3-23        |
| StreamingAudioClip .....                           | 3-26        |
| <b>DirectX Files</b> .....                         | <b>3-28</b> |

## **The WorldView Browser**

This chapter describes issues specific to the development of VRML content for use with the WorldView browser. It contains information for both WorldView for Netscape Navigator and WorldView for Internet Explorer.

Please be sure to consult the WorldView Release Notes for information about known bugs and important changes.

## **Support for VRML 2.0**

This section describes details of WorldView's VRML 2.0 implementation, including known bugs.

WorldView is fully compliant with the Minimum Conformance requirements in the ISO/IEC DIS 14772-1 VRML 2.0 specification, dated 4 April 1997. Fields considered optional according to the Minimum Conformance requirements are supported unless they are specifically mentioned below.

### VRML 2.0 Nodes

This section details WorldView's current implementation of the VRML 2.0 nodes. Nodes not specifically mentioned here are supported exactly as defined in the specification.

#### **Anchor**

Only the first URL specified is used.

In WorldView for Internet Explorer only, URLs that refer to files located on the local disk and contain viewpoint names do not work.

#### **Collision**

When the Prevent Collisions user option is OFF, users will pass through geometry regardless of Collision nodes. This option is ON by default. To ensure that users will be blocked by geometry, use NavigationInfo type NONE.

#### **ColorInterpolator**

The interpolation of colors is performed in RGB color space, not HSV color space.

#### **DirectionalLight**

The effect of the color field is only visible when WorldView is in full color mode. (See *WorldView's Full Color Setting*, below.)

#### **ElevationGrid**

In full color mode, vertex colors are supported. Because an ElevationGrid with vertex colors is not affected by lights, the normal field will be ignored.

When full color is off, the vertex colors are averaged to compute face colors, which are affected by lights. With full color off, or colorPerVertex set to FALSE, the normal field will be used. (See *WorldView's Full Color Setting*, below.)

## Fog

The fogType is ignored and always assumed to be LINEAR. Fog is only visible when WorldView is in Full Color mode. (See [WorldView's Full Color Setting](#), below.)

## FontStyle

The family field may include the names of any installed Windows fonts, in addition to SERIF, SANS, and TYPEWRITER. If none of the specified fonts are available on the viewer's system, SERIF is used instead.

## IndexedFaceSet

In full color mode, vertex colors are supported. Because an IndexedFaceSet with vertex colors is not affected by lights, the normal field will be ignored.

When full color is off, the vertex colors are averaged to compute face colors, which are affected by lights. With full color off, or colorPerVertex set to FALSE, the normal field will be used. (See [WorldView's Full Color Setting](#), below.)

## IndexedLineSet

Fully supported in full color mode. With full color off, the color field is ignored, and the emissiveColor of the Material node is used to color the line set. (See [WorldView's Full Color Setting](#), below.)

### **ImageTexture**

Supported image file types are GIF, BMP, JPG, RAS, PPM and PNG.

The repeatS and repeatT fields are ignored.

Intermediate alpha opacities are not supported. Pixels with alpha less than 0.5 appear fully transparent, and pixels with alpha greater than or equal to 0.5 appear fully opaque.

### **Material**

The ambientIntensity field is not supported.

The appearance of transparent materials can be greatly affected by graphics accelerator cards.

### **MovieTexture**

File types are supported through DirectShow. Audio is supported in all formats.

The repeatS and repeatT fields are ignored. Negative values in the speed field are ignored.

Intermediate alpha opacities are not supported. Pixels with alpha less than 0.5 appear fully transparent, and pixels with alpha greater than or equal to 0.5 are opaque.

When using animated GIFs as textures, the pixel width and height must each be a power of two, for example: 256 x 128, or 64 x 64.



## NavigationInfo

The speed field is not supported.

The first value of the avatarSize field is used to determine the allowable distance between the user and geometry during collision detection. Other values in this field are ignored.

The NONE type disables both toolbars, the Navigation commands on the right mouse popup menu, and the ability to navigate using the mouse. The user may not change these settings.

The WALK, FLY, and EXAMINE types set the initial navigation mode when the world is loaded.

| Type    | Browser allows            |
|---------|---------------------------|
| WALK    | Walk and Turn             |
| FLY     | Walk, Turn, Pan, and Roll |
| EXAMINE | Study                     |

The "GoTo" function is allowed if the type includes WALK, FLY, or ANY. "Zoom Out," "Straighten Up," and viewpoint navigation are not allowed if the type is "NONE."

If the type is ANY, viewpoint transitions triggered by Anchor nodes or the loadURL() scripting method are animated and trigger sensors.

If Headlight is set to FALSE, the user may not enable the headlight.

## PixelTexture

The repeatS and repeatT fields are ignored.

Intermediate alpha opacities are not supported. Pixels with alpha less than 0.5 appear fully transparent, and pixels with alpha greater than or equal to 0.5 are opaque.

### PointLight

The effect of the color field is only visible when WorldView is in Full Color mode. (See *WorldView's Full Color Setting*, below.)

The radius field is emulated using quadratic attenuation. If the attenuation field is set to something other than the default (1 0 0), the radius field is ignored.

### PointSet

When Full Color is off, the color field is ignored, and the emissiveColor of the Material node is used to color the point set. (See *WorldView's Full Color Setting*, below.)

### Script

Script nodes may contain Java or Javascript. (VRMLScript is supported to the extent that it overlaps with JavaScript.) If the URL field does not specify a file, it must begin with one of the following:

“javascript:

“vrmlscript:

**Syntax:** url “javascript:<script>

**Or**

url “vrmlscript:<script>”

The Script node recognizes the “java:” protocol, which permits users to specify a Java class in a package directly in the VRML file.

**Syntax:** url “java:codebase:classname”

where codebase is the URL for the root of the Java class hierarchy, and classname is the fully-qualified name of the Java class, separated with periods.

The Script node also recognizes the “javabc: protocol.

**Syntax:** url “javabc:<base64 encoded \*.class file>

If there is an error in a Script node, the browser sends a notification to the VRML Console. Further events to the Script node are ignored.

A line number of "0" is incorrectly reported when an error occurs on the last line of a script contained within a Script node.

The loadURL scripting method uses only the first specified URL.

See also [Support for JavaScript](#) and [Support for Java in Script Nodes](#).

### **Sound**

The sound model is spherical rather than ellipsoidal. Values in the maxFront and maxBack fields are averaged to determine the radius of the outer sphere; minFront and minBack are averaged to determine the radius of the inner sphere.

### **SpotLight**

The effect of the color field is only visible when WorldView is in Full Color mode. (See [WorldView's Full Color Setting](#), below.)

### Text

WorldView allows the user to specify how text will be generated, and the resultant text image quality.

WorldView's default text is polygonal. Selecting the High Quality Text option, which is on by default in the WorldView Options... dialog, causes text to be generated as a polygonal mesh. WorldView extracts outline curve information from the TrueType font to produce this mesh.

When High Quality Text is not selected, text is generated by rendering a bitmap, which is then applied as a texture map to a simple rectangle.

Bitmapped text offers better performance when a long text string is imported. High quality, polygonal text offers better performance when shorter text strings are displayed, and better appearance overall. Polygonal text maintains its outline quality at any scale. Texture mapped text will become pixilated at close range.

The maxExtent field of the Text node is not supported when High Quality text is used.

See the WorldView Extensions [BrowserSettings](#) section for more information on setting Text options in the WorldView browser.

### Viewpoint

If a Viewpoint node has both

- jump set to FALSE, and
- a string in the description field

the user's position will change if they choose Viewpoint from the menu in the WorldView user interface. Changes to this Viewpoint that occur via any other mechanism will not cause the user's position to change, because jump is set to FALSE. To avoid this behavior, prevent the Viewpoint from appearing in the menu by not providing a description for it.

## data: Protocol

WorldView supports the `data:` protocol, as required by the VRML 2.0 specification, section 4.5.4. All file types, including scripts, movies, and audio, may be inlined in a VRML file using this feature. The `data:` URL may be used for in-line inclusion of base64 encoded data, such as JPEG, GIF, and PNG files.

The `data:` protocol may be used in any URL field with the syntax:

```
data:<mimetype>,base64;<base64-encoded data>
```

### Or

```
data:<mimetype>;<regular text with % used to do hex characters>
```

An example of the second, non base64 encoded type would read:

```
data:model/vrml;#VRML V2.0 utf8%0AShape {%0A geometry Box  
{}%0A}
```

### WorldView's Full Color Setting

There are several features in VRML that are only visible in WorldView when the Full Color setting is on. Full Color is off by default, but may be controlled on a per-session basis from the "Graphics" item on the right mouse button popup menu. The default may be changed permanently by choosing "Options..." from the right mouse button popup menu, and changing the setting of the "Full Color" check box in the Windows tabbed window.

The Full Color setting has a significant effect on navigation speed.

These VRML 2.0 nodes are affected when Full Color is OFF:

- The color field for PointLight, DirectionalLight, and SpotLight is ignored.
- Fog does not appear.
- IndexedFaceSet vertex colors are averaged to compute face colors, if colorPerVertex is set to TRUE.
- ElevationGrid vertex colors are averaged to compute face colors, if colorPerVertex is set to TRUE.
- The color field for IndexedLineSet is ignored, and the emissiveColor of the Material node is used to specify the line color.
- The color field for PointSet is ignored, and the emissiveColor of the Material node is used.

These nodes and fields are fully visible when Full Color is ON.

See the WorldView Extensions [BrowserSettings](#) section for more information on controlling the Full Color options within the WorldView browser.

## Support for JavaScript

WorldView's JavaScript implementation conforms to the JavaScript specification included in Annex C of the ISO/IEC DIS 14772-1 VRML 2.0 specification, dated 4 April 1997.

### JavaScript and the VRML Console

Javascript alert statements are routed to WorldView's VRML Console. (Select "Show VRML Console" from the right mouse Help menu.)

The Javascript `print()` method can be used to print information directly to WorldView's VRML Console. The `trace()` method, introduced in earlier versions of WorldView, works identically to `print()`. Although `trace()` remains supported, you should use `print()` to ensure interoperability with other browsers.

WorldView's JavaScript interpreter requires that all statements be terminated by semicolons.

### Variable Scoping

Variables declared with the `var` keyword are local to the function in which they are declared. Variables not declared with the `var` keyword are global to the Script node in which they are used, and are visible in all functions of the Script node, but not in functions of other script nodes. Script node fields and events are in the same name space as global variables.

### Type Conversion in JavaScript Expressions

JavaScript automatically converts between data types when necessary while evaluating expressions. The following table describes how WorldView's JavaScript implementation performs conversion from each data type to each other data type.

Find the source data type in the left column, then find the cell in that row whose column corresponds to the destination data type. "NS" indicates that the conversion is not currently supported by WorldView. Conversion of any non-object into an object is not supported.

| Data Type         | Function | Number                 | Boolean | String   |
|-------------------|----------|------------------------|---------|----------|
| function          |          | error                  | error   | NS       |
| object            | error    | error                  | true    | NS       |
| Null object       | NS       | 0                      | false   | NS       |
| number (non-zero) | error    |                        | true    | toString |
| number (0)        | error    |                        | false   | "0"      |
| Error (NaN)       | error    |                        | NS      | NS       |
| + infinity        | error    |                        | NS      | NS       |
| - infinity        | error    |                        | NS      | NS       |
| false             | error    | 0                      |         | "false"  |
| true              | error    | 1                      |         | "true"   |
| string (non-null) | NS       | numeric representation | true    |          |
| null string       | error    | error                  |         | false    |

---

## Unsupported Functions

The JavaScript eval() function is not supported.



## Support for the Java EAI

The Java EAI is fully supported according to the Proposal for a VRML 2.0 Informative Annex dated 21 January 1997. It is believed to be compatible with the EAI classes from other browsers.

WorldView uses the `vrml.external.*` naming convention, not the old `vrml.*` naming convention.

## Requirements for Internet Explorer

To access the Java EAI with WorldView for Internet Explorer, a Java applet must be considered a “trusted applet.” This can be done in two ways:

**Place the applet in a digitally signed cabinet file.** This can be done using the CabDevKit provided with the Microsoft Java SDK. The Microsoft Web site contains additional information on using signed CAB files to distribute EAI applets.

**Put the applet's class files in your CLASSPATH environment variable.** This is useful for development purposes. Microsoft Visual J++ does this automatically when debugging.

## Using WorldView's Java EAI in Web Browsers

The following example demonstrates using WorldView's Java EAI in Microsoft Internet Explorer or Netscape Navigator.

For Netscape Navigator, or Internet Explorer 4.0 or later, simply call `Browser.getBrowser` directly in the `init()` method, as described below.

```
*****
*****
HTML FILE:
  <HTML>
  <HEAD>
  <TITLE>WorldView ActiveX Control Example</TITLE>
  </HEAD>
  <BODY>
    <embed src="MyWorld.wrl" height=500 width=400>
    <applet code="MyApplet.class" height=140 width=700>
  </BODY>
</HTML>
*****
*****
JAVA FILE:
import java.applet.Applet;
import vrm1.external.*;
/*****/
public class MyClass extends Applet
{
public void init()
{
    setLayout(new FlowLayout());
    add(new Button("My Button"));
    //ok to get browser now
    Browser
    browser=Browser.getBrowser(this);
    Node node=browser.getNode();
}
}
public void run() {}
// Other Methods ..
}
```

## Support for Java in Script Nodes

The Java in Script nodes implementation is according to Annex B of the ISO/IEC DIS 14772-1 VRML 2.0 specification, dated 4 April 1997.

### Java 1.1 Support

WorldView's Java in Script nodes implementation uses the Microsoft Java VM, and supports the Java 1.1 standard to the same extent as Microsoft Internet Explorer 4.0.

### System.out and System.err

Java's System.out and System.err streams are routed to the WorldView VRML Console. (Select "Show VRML Console" from the right mouse Help menu.)

### Security

For Java in Script nodes, WorldView provides an environment similar to Netscape or Internet Explorer for running Java code. Script nodes have the following restrictions, which are similar to Java applets:

- Scripts cannot read, write or delete files.
- Scripts cannot execute programs (i.e. Runtime.exec).
- Scripts cannot create ServerSockets.
- Scripts cannot link to native code (i.e. System.loadLibrary).
- Scripts cannot create ClassLoaders.
- Scripts may bring up AWT top-level windows.
- Scripts may make socket connections only to their host of origin.

Java classes loaded from the CLASSPATH are considered "trusted code" and can perform operations ordinarily not allowed by the security manager, such as reading/writing files and making access to any network host.

## Using the EMBED tag in HTML documents

You may use the HTML EMBED tag to specify user interface settings that will be used in WorldView when a VRML world is embedded in an HTML document. The following tags are supported:

| Tag                                    | Setting   | Effect   |
|--|-----------|--|
| VRML_POPMENU,<br>SGI_POPMENU           | TRUE      | Enables the right mouse menu                     |
|  | FALSE     | Disables the right mouse menu                    |
| VRML_IMAGEQUALITY,<br>SGI_IMAGEQUALITY | BEST      | Sets image display to "Smooth Shading"           |
|  | SMOOTH    | Sets image to "Flat Shading"                     |
|  | SMOOTHEST | Sets image to "Wire Frame"                       |
| VRML_SPLASHSCREEN,<br>SGI_SPLASHSCREEN | TRUE      | Displays splash screen at startup                |
|  | FALSE     | Prevents splash screen from appearing at startup |
| VRML_DASHBOARD,<br>SGI_DASHBOARD       | TRUE      | Turns on horizontal and vertical toolbars        |
|  | FALSE     | Turns off both toolbars                          |
| VRML_FULLCOLOR                         | TRUE      | Turns on Full Color mode                         |
|  | FALSE     | Turns off Full Color mode                        |
| VRML_DITHERING                         | TRUE      | Turns on dithering                               |
|  | FALSE     | Turns off dithering                              |

## WorldView EXTERNPROTO Extensions

WorldView implements four new nodes using the standard EXTERNPROTO extension mechanism described in the VRML 2.0 specification: *BillboardText*, *BrowserSettings*, *PopupText*, and *StreamingAudioClip*.

### BillboardText

```
BillboardText {
  exposedField MFString string []
  exposedField SFNode fontStyle NULL
  exposedField MFFloat length [] # [0,inf)
  exposedField SFFloat maxExtent 0.0 # [0,inf)
} URN: "urn:inet:platinum.com:BillboardText"
```

The *BillboardText* node is a WorldView extension to VRML 2.0, which facilitates the display of efficient, high-quality text in the VRML scene. The text is billboarded so that it always faces the user. Unlike *PopupText*, text in a *BillboardText* node is scaled by the distance from the camera.

Example:

```
#VRML V2.0 utf8
EXTERNPROTO BillboardText [
  exposedField MFString string
  exposedField SFNode fontStyle
  exposedField MFFloat length
  exposedField SFFloat maxExtent
] "urn:inet:platinum.com:BillboardText"
Shape {
  appearance Appearance {
    material Material {
      diffuseColor 1 0 0
    }
  }
  geometry BillboardText {
    string "Hello, world!"
  }
}
```

### BrowserSettings

```
BrowserSettings {
  exposedField SFString quality "DEFAULT"
  exposedField SFString qualityWhenMoving "DEFAULT"
  exposedField SFString showNavigationBar "DEFAULT"
  exposedField SFString fullColor "DEFAULT"
  exposedField SFString dithering "DEFAULT"
  exposedField SFString rotateObjectsAutomatically "DEFAULT"
  exposedField SFString highQualityText "DEFAULT"
  exposedField SFString preventCollisions "DEFAULT"
  exposedField SFString popupMenu "DEFAULT"
}
```

The BrowserSettings node uses the standard EXTERNPROTO extension mechanism described in the VRML 2.0 specification, with the URN `urn:inet:platinum.com:BrowserSettings`.

The BrowserSettings node enables VRML authors to control WorldView's preference settings from within a VRML file. Each field in the BrowserSettings node controls a different WorldView preference. The initial value of every field is DEFAULT, which loads the user's default setting for the selected preference.

```
  exposedField SFString quality "DEFAULT"
  exposedField SFString qualityWhenMoving "DEFAULT"
```

Using this extension to control the preference settings simply defines the initial setting for each item. The user retains the ability to reset the preference through the Options... item in the right mouse menu.

**quality:** Sets the graphical quality to display the scene. The accepted values and the corresponding options on the right mouse button menu are:

“BEST” = Smooth Shading

“GOOD” = Flat Shading

“POOR” = Wire Frame

This setting can also be controlled with the `VRML_IMAGEQUALITY EMBED` tag.

**qualityWhenMoving:** Sets the graphical quality to display the scene when the user is navigating. The accepted values and the corresponding options on the “When Moving” submenu of the right mouse menu are:

“BEST” = Smooth Shading

“GOOD” = Flat Shading

“POOR” = Wire Frame

“NONE” = No Change

**showNavigationBar:** Shows or hides the navigation bar. Accepted values are `DEFAULT`, `TRUE`, and `FALSE`. This is equivalent to the “Show Navigation Bar” option on the right mouse menu. This setting can also be controlled with the `VRML_DASHBOARD EMBED` tag.

**fullColor:** Enables or disables Full Color mode. Accepted values are `DEFAULT`, `TRUE`, and `FALSE`. This is equivalent to the “Full Color” option on the right mouse menu. This setting can also be controlled with the `VRML_FULLCOLOR EMBED` tag.

**dithering:** Enables or disables dithering. Accepted values are `DEFAULT`, `TRUE` and `FALSE`. This is equivalent to the “Dithering” option on the right mouse menu. This setting can also be controlled with the `VRML_DITHERING EMBED` tag.

**rotateObjectsAutomatically:** Enables or disables automatic rotation of objects. Accepted values are DEFAULT, TRUE, and FALSE. This is equivalent to the “Rotate Objects Automatically” option in the WorldView Options dialog box.

**highQualityText:** Enables or disables high quality text. Accepted values are DEFAULT, TRUE, and FALSE. This is equivalent to the “High Quality Text” option on the right mouse menu.

**preventCollisions:** Enables or disables collision detection. Accepted values are DEFAULT, TRUE, and FALSE. This is equivalent to the “Prevent Collisions” option on the right mouse menu.

**popupMenu:** Enables or disables the right mouse menu. Accepted values are DEFAULT, TRUE, and FALSE. This setting can also be controlled with the VRML\_POPMENU EMBED tag.

Settings defined in the BrowserSettings node will be the world's default settings, but may be altered by the user through the right mouse menu.

BrowserSettings is a WorldView extension to VRML. To use it, you must declare it as an EXTERNPROTO, as shown below:

```
EXTERNPROTO BrowserSettings [  
    exposedField SFString quality  
    exposedField SFString qualityWhenMoving  
    exposedField SFString showNavigationBar  
    exposedField SFString fullColor  
    exposedField SFString dithering  
    exposedField SFString rotateObjectsAutomatically  
    exposedField SFString highQualityText  
    exposedField SFString preventCollisions  
    exposedField SFString popupMenu  
] "urn:inet:platinum.com:BrowserSettings"
```



## PopupText

```
PopupText {
  exposedField SFVec3f position 0 0 0
  exposedField MFString string []
  exposedField SFCOLOR textColor 1 1 1
  exposedField SFCOLOR backgroundColor 0 0 0
  exposedField SFCOLOR borderColor 1 1 1
  exposedField SFInt32 borderWidth 1
  exposedField SFString family [ "SERIF" ]
  exposedField SFString style "PLAIN"
  exposedField SFString anchor "CENTER"
  exposedField SFString justify "LEFT"
  exposedField SFInt32 pointSize 12
  exposedField SFBool transparent FALSE
  exposedField SFString language ""
  exposedField SFString positionType "LOCAL"
}
```

The `PopupText` node uses the standard EXTERNPROTO extension mechanism described in the VRML 2.0 specification, with the URN `urn:inet:platinum.com:PopupText`.

The `PopupText` node is a WorldView extension to VRML 2.0, which provides a way to overlay 2D text over the VRML scene. Possible applications include labels for charts, popup “tool tips” over objects, and score indicators.

`positionType`: Determines the position of the popup text. This field is interpreted according to the value of the `positionType` field. Possible values for the `positionType` field are `LOCAL`, `VIEWPORT` and `SCREEN`.

- `positionType LOCAL` sets the position of the popup text in the local coordinate system. This position is transformed into screen coordinates to place the text in the rendering window.
- `positionType VIEWPORT` sets the position of the popup text in the viewport. The X and Y values of the position field may range from 0 to 1, and the Z value is ignored. For X, 0 is the left side of the viewport and 1 is the right side. For Y, 0 is the bottom of the viewport and 1 is the top.

- `positionType SCREEN` sets the position of the popup text in screen coordinates. The X and Y values of the position field are specified in pixels, with 0 being the left side of the viewport for X and the top of the viewport for Y.

In all cases, the text is clipped to the rendering window, and is not obscured by the objects of the VRML scene.

The string field may contain multiple text strings specified using the UTF-8 encoding as specified by ISO 10646-1:1993. Each text string is displayed as a line of text.

`textColor`: Specifies the color of the text.

`transparent`: Specifies whether the text has a colored background rectangle behind it. If `transparent` is `FALSE`, the `backgroundColor` field specifies the background color of the rectangle.

`borderWidth`: Specifies the thickness of the border around the text. If `borderWidth` is 0, no border is displayed. The border is displayed in the color specified by the `borderColor` field.

Font attributes are defined with the `family`, `style` and `pointSize` fields. The `family` and `style` fields are interpreted in the same way as the fields with the same names in the `FontStyle` node. The `pointSize` field specifies the point size used to display the text.

`justify`: Determines the horizontal alignment of the text. Possible values are `LEFT` (default), `CENTER` and `RIGHT`. `LEFT` aligns the beginning of each string of text with the left side of the text rectangle. `RIGHT` aligns the end of each string with the right side of the text rectangle. `CENTER` centers each line of text in the text rectangle.

`anchor`: specifies which corner or side of the text rectangle is coincident with the 3D point specified by the position field. Valid values are `CENTER`, `N`, `S`, `E`, `W`, `NW`, `SW`, `NE`, and `SE`.

`language`: specifies the context of the language for the text string. This field is interpreted in the same way as the `language` field of the `FontStyle` node.

The following example is a frame-rate indicator which appears in the bottom right-hand corner of the viewing window.

```
#VRML V2.0 utf8
EXTERNPROTO PopupText [
  exposedField SFVec3f position
  exposedField MFString string
  exposedField SFCOLOR textColor
  exposedField SFCOLOR backgroundColor
  exposedField SFCOLOR borderColor
  exposedField SFInt32 borderWidth
  exposedField MFString family
  exposedField SFString style
  exposedField SFString anchor
  exposedField SFString justify
  exposedField SFInt32 pointSize
  exposedField SFBool transparent
  exposedField SFString language
  exposedField SFString positionType
] "urn:inet:platinum.com:PopupText"
DEF TEXT PopupText {
  family "TYPEWRITER"
  style "PLAIN"
  pointSize 12
  borderWidth 1
  textColor 1 1 1
  borderColor 0 0 0
  anchor "SE"
  positionType "VIEWPORT"
  position 1 0 0
  backgroundColor 0 0 0
}
DEF TIMER TimeSensor {
  loop TRUE
}
```

```
DEF SCRIPT Script {
  eventIn STime go
  eventOut MFString frameRate
  url "javascript: function go() {
    frameRate[0] =
    browser.getCurrentFrameRate();
  }"
}
ROUTE TIMER.time TO SCRIPT.go
ROUTE SCRIPT.frameRate TO TEXT.set_string
```

## StreamingAudioClip

```
StreamingAudioClip {
  eventIn MInt32 set_data
  exposedField SFString description ""
  exposedField SFloat pitch 1.0
  exposedField STime startTime 0
  exposedField STime stopTime 0
  eventOut SBool isActive
  eventOut SBool isReady
}
```

The StreamingAudioClip node uses the standard EXTERNPROTO extension mechanism described in the VRML 2.0 specification, with the URN `urn:inet:platinum.com:StreamingAudioClip`.

A StreamingAudioClip node is similar to the AudioClip node, and can be instantiated anywhere in the scene graph where an AudioClip node is expected.

StreamingAudioClip replaces AudioClip's url field with a set\_data eventIn, which allows dynamic feeding of audio data.

The isReady eventOut indicates when the node is ready to accept data. Its value is TRUE when the node has internal buffer space, and FALSE when the node is flooded.

The audio data is packed into an MFInt32 event. The first element of `set_data` (`set_data[0]`) contains the sequence number of the data chunk. The second element (`set_data[1]`) specifies the size of audio data in bytes. The size does not include the first two elements. Subsequent elements contain actual audio data.

A chunk with sequence number 0 contains a header specifying the format of the data to follow. A chunk with sequence number -1 indicates end of stream.

Audio data is packed into MFInt32 using little endian conventions, placing the least significant byte first.

`StreamingAudioClip` is a WorldView extension to VRML. To use it, you must declare it as an EXTERNPROTO, as shown below:

```
EXTERNPROTO StreamingAudioClip [  
  eventIn MFInt32 set_data  
  exposedField SFString description  
  exposedField SFFloat pitch  
  exposedField SFTIME startTime  
  exposedField SFTIME stopTime  
  eventOut SFBool isActive  
  eventOut SFBool isReady  
] "urn:inet:platinum.com:StreamingAudioClip"
```

## DirectX Files

Microsoft defines a file format to be used in conjunction with Direct3D. These files have a .X extension, and can contain both geometry and animations. With WorldView, .X files may be loaded as main worlds, or as Inlines. To be loaded as main worlds, the user must associate the .X extension with WorldView (standard Windows procedure). Then, double-clicking an DirectX file will launch WorldView and open the file.

DirectX files can also be specified as values of the url fields of Inline nodes.

```
#VRML V2.0 utf8
Inline {
  url "egg.x"
}
```

---

# WorldView for Developers Containers

This chapter describes issues and procedures specific to each of WorldView for Developers' containers.

|  |            |
|--|------------|
| <b>Introduction</b> .....                  | <b>4-3</b> |
| <b>Macromedia Director Xtras</b> .....     | <b>4-3</b> |
| <b>Microsoft Visual C++</b> .....          | <b>4-4</b> |
| Adding the Component .....                 | 4-4        |
| Using the Component in a Window .....      | 4-5        |
| Controlling the Component .....            | 4-5        |
| Adding the Component to a Dialog Box ..... | 4-6        |
| Embedding in C++ at Runtime .....          | 4-6        |

|   |             |
|---|-------------|
| <b>Microsoft Java VM</b> .....                                    | <b>4-7</b>  |
| Embedding the WorldView Component .....                           | 4-8         |
| Accessing the COM API .....                                       | 4-9         |
| Using Standard Java EAI .....                                     | 4-10        |
| Microsoft Java and ActiveX Integration .....                      | 4-10        |
| Embedding in J++ at Runtime .....                                 | 4-11        |
| Samples .....   | 4-11        |
| <b>Microsoft Visual Basic</b> .....                               | <b>4-12</b> |
| Adding the Component .....  | 4-12        |
| Positioning and Resizing the Control in a Visual Basic Form ..... | 4-13        |
| Getting a Reference to the Control .....                          | 4-13        |
| Getting a Reference to the VrmlBrowser Object .....               | 4-14        |
| Adding a VRML Primitive to the Scene .....                        | 4-14        |
| Removing a Node from the Scene .....                              | 4-16        |
| Changing the Fields of a Node in a VRML Scene .....               | 4-17        |
| Receiving Events from a VRML Scene .....                          | 4-18        |
| Passing Arrays to Methods .....                                   | 4-19        |
| Invoking an OCX from the Script node .....                        | 4-20        |
| Embedding in Visual Basic at Runtime .....                        | 4-21        |
| Samples .....   | 4-21        |
| <b>Working with Web Pages: HTML and Java</b> .....                | <b>4-22</b> |
| Getting a Reference to the Control .....                          | 4-22        |
| Embedding a WorldView for Developers File in an HTML Page .....   | 4-23        |
| Getting a Reference to the VrmlBrowser Object .....               | 4-24        |
| Adding a VRML Primitive to the Scene .....                        | 4-26        |
| Removing a Node from the Scene .....                              | 4-27        |
| Changing the Fields of a Node in a VRML Scene .....               | 4-28        |
| Receiving Events from the VRML Scene .....                        | 4-29        |



## Introduction

This chapter describes issues and procedures specific to using WorldView for Developers in each of its applicable containers. WorldView for Developers is compatible with:

- Macromedia Director Xtras
- Microsoft Visual C++
- Microsoft Java VM
- Microsoft Visual Basic
- Web Pages: HTML and Java

## Macromedia Director Xtras

Macromedia Control Xtra for ActiveX must be installed.

To place a WorldView control in your cast window:

- 1** Select Insert ▶ Control ▶ ActiveX... to invoke a dialog containing a list of all available ActiveX controls.
- 2** Select WorldView Control and click the OK button to invoke the ActiveX Control Properties - WorldView Control dialog.
- 3** Use this dialog to set control properties such as the VRML file to be displayed (the "World" property), the navigation mode, and the graphics display mode. Click OK to apply the selected properties, and exit the dialog.

The WorldView control now appears in your cast window. Drag it from the cast window onto the stage in order to place a running WorldView object into your presentation.

Refer to the Director manual and the tutorial for the ActiveX Xtra for more details on manipulating the control after it has been placed into a presentation.

## Microsoft Visual C++

Visual C++ 5.0 provides extensive support for ActiveX Controls, making it extremely straightforward to embed WorldView for Developers in your C++ application.

### Adding the Component

To add WorldView for Developers to your C++ project:

- 1** From the Project menu, select the Add To Project ► Components and Controls item. The Components and Controls dialog box will appear.
- 2** Select the folder “Registered ActiveX Controls.”
- 3** Double-click on the WorldView Control icon. When Visual C++ prompts you with the list of interfaces to include, press the OK button.

WorldView for Developers is now included in your project. Visual C++ will add two files, `worldview.h` and `worldview.cpp`, to your project, as well as other files that wrap WorldView for Developers COM objects.

## Using the Component in a Window

WorldView for Developers can now be created as a child window of one of your application's windows.

- 1 First, include the file `worldview.h` in the header file for the window you want to add WorldView to:

```
#include "worldview.h"
```

- 2 Declare a member variable for WorldView in your window class:

```
CWorldView* m_worldview;
```

- 3 Now add a call to `CWorldView::Create` to the initialization code for your window, e.g. the `OnCreate` method. For instance:

```
// Get the rectangle for the client area of the window:  
// WorldView will occupy the entire client area.  
RECT rect;  
GetClientRect(&rect);  
  
// Create WorldView  
BOOL bResult = m_worldview->Create(  
    NULL,  
    "WorldView",  
    WS_CHILD|WS_VISIBLE,  
    rect,  
    this,  
    0  
);
```

See also [Embedding WorldView in a C++ Application: Tiny3D](#).

## Controlling the Component

Once you have created an instance of WorldView for Developers, you can call methods and modify properties of the control. For example, to load the VRML file `cone.wrl`, set the `World` property of the control:

```
m_worldview->SetWorld("cone.wrl");
```

### Adding the Component to a Dialog Box

It is also possible to embed WorldView for Developers in a dialog box. After you have added the WorldView for Developers component to your C++ project, the WorldView icon will appear in the dialog box editor. Select it, and you can create a WorldView dialog box item just as you would create any other type of dialog box control.

By clicking the right-mouse button on the WorldView dialog, you can alter properties of the WorldView control, such as the World property.

To control the WorldView dialog box item from C++ code, add a member variable for it to your dialog box class. This will enable you to modify properties and call methods of the WorldView control.

- 1 From the ClassWizard, select your dialog box class and select the Member Variables property sheet.
- 2 Click on the ID of the WorldView dialog box item (e.g. IDC\_WORLDVIEW1) and click the "Add Variable..." button.

### Embedding in C++ at Runtime

In C++, the MFC framework permits you to specify a runtime license key when creating an instance of an ActiveX Control. First, paste the runtime license key into your application's source code:

```
#define RUNTIME_LICENSE_KEY L"<license key pasted from clipboard>"
```

To create the WorldView for Developers control, specify the license key as an argument to the Create method as follows:

```
BOOL bResult = m_pWorldView->Create(  
    NULL,  
    "WorldView",  
    WS_CHILD|WS_VISIBLE,  
    rect,  
    this,  
    0,  
    RUNTIME_LICENSE_KEY  
);
```

## **Microsoft Java VM**

The Microsoft Java VM shipped with Microsoft Internet Explorer 4.0 and distributed with Microsoft Java SDK 2.0 has the ability to host an ActiveX Control. This means that a Java application or applet can act as an ActiveX Container and embed WorldView for Developers.

Using Microsoft's Java-ActiveX integration, embedding WorldView for Developers can be accomplished with only a few lines of code. As the following examples demonstrate, the WorldView for Developers ActiveX Control can be added to a Java AWT panel or window just like any other AWT component.

### **Embedding the WorldView Component**

The following example is a standalone Java application that embeds WorldView:

```
import java.awt.*;
import com.ms.activeX.*;

public class JWorldViewContainer
    extends Frame implements ActiveXControlListener
{
    public static void main(String argv[])
    {
        new JWorldViewContainer();
    }

    public JWorldViewContainer()
    {
        super("JWorldViewContainer");

        // Create the WorldView for Developers ActiveX Control by CLSID.
        ActiveXControl c =
        new ActiveXControl("{0939A9F5-1788-11d2-8FDE-0060975B8649}");

        // Notify us when the control is loaded
        c.addActiveXControlListener(this);

        // Load a WRL file
        c.setProperty("World", "c:\\wrl\\mytest.wrl");

        // Add the WorldView ActiveX Control to this
        // AWT frame window
        add("Center", c);
        setSize(400, 400);
        show();
    }

    public void controlCreated(Object target)
    {
        // called when the ActiveX Control is done loading
    }
}
```

## Accessing the COM API

The previous example embeds WorldView for Developers inside a Java application and allows it to display a VRML file. To control the VRML scene from within the Java application, you must access the WorldView for Developers COM API. To do this, you will need to obtain the `IVrmlBrowser` interface.

- 1 First, add the following import lines to the previous example:

```
import intervista.javacom.worldview.IWorldView;  
import intervista.javacom.vrmlbrowser.IVrmlBrowser;  
import vrml.external.*;
```

- 2 Then add these lines to the `controlCreated` method:

```
IWorldView worldview = (IWorldView) c.getObject();  
IVrmlBrowser browser = worldview.GetBrowser();
```

The `controlCreated` method is invoked when the ActiveX Control is done loading. Because the `GetBrowser` method cannot be invoked until WorldView for Developers is completely loaded, this initialization code is placed in the `controlCreated` method.

Just as obtaining the `vrml.external.Browser` object is the first step in a Java EAI program, obtaining the `IVrmlBrowser` interface is the first step in a program that uses the WorldView for Developers COM API. Using the `IVrmlBrowser` interface, you can acquire references to nodes in the VRML scene and control their behavior.

### **Using Standard Java EAI**

To use the standard Java EAI in a Java program that embeds WorldView for Developers, call the SetControl method of the class `vrml.external.WorldViewControl`, and pass it the `IWorldView` interface of the ActiveX Control.

The following example demonstrates how to obtain a standard Java EAI Browser object from the `IVrmlBrowser` interface.

```
// Acquire the IVrmlBrowser interface
IWorldView worldview = (IWorldView) c.getObject();
IVrmlBrowser browser = worldview.GetBrowser();

// Obtain the Java EAI Browser object
Applet dummyApplet = new Applet();
WorldViewControl.SetControl(c.getObject(), dummyApplet);
browser = Browser.getBrowser(dummyApplet);
```

### **Microsoft Java and ActiveX Integration**

Embedding WorldView for Developers in a Java program is made possible by Microsoft's integration of Java and ActiveX. For more information about Microsoft's Java/ActiveX integration, consult the documentation for the Microsoft Java SDK 2.0. The Microsoft Java SDK 2.0 may be downloaded from <http://www.microsoft.com/java/sdk>.



## Embedding in J++ at Runtime

- ▶ In J++, paste the runtime license key into your application's source code:

```
private static final String RUNTIME_LICENSE_KEY = "<license key>";
private static final String WORLDVIEW_CLSID = "{0939A9F5-1788-11d2-8FDE-0060975B8649}";
```

- ▶ To create the WorldView for Developers control, you will need to use the class `com.ms.com.LicenseMgr` which enables you to create COM objects using a license key.

```
import com.ms.com.*;
...
ILicenseMgr licenseMgr = (ILicenseMgr) new LicenseMgr();
Object control = licenseMgr.createWithLic(
    RUNTIME_LICENSE_KEY,
    WORLDVIEW_CLSID,
    null,
    ComContext.INPROC_SERVER
);
```

The `createWithLic` method returns a raw Java/COM object. You may wish to convert this to an instance of `com.ms.com.ActiveXControl` which can be treated as an AWT component and easily added to a window of your Java application.

```
ActiveXControl = new ActiveXControl((IUnknown)control);
```

## Samples

A more complex version of the `JWorldViewContainer` sample above is provided in Appendix A. See [Embedding WorldView in a Java Application: JWorldViewContainer](#).

## Microsoft Visual Basic

### Adding the Component

Install WorldView for Developers, and launch Visual Basic.

- 1** From the Visual Basic menu, select Project ▶ Components.
- 2** Select WorldView for Developers.
- 3** Press OK. The WorldView for Developers icon will appear in the Visual Basic Tool Palette.
- 4** Click on the WorldView for Developers icon and draw a rectangle with your mouse to place and give the control its initial size, or double click on the icon to place a default sized and positioned WorldView for Developers control on the form.

Once the control has been placed, it will behave as any Visual Basic component: its properties will appear on the Property Bar; double clicking it will invoke the component's code space; pressing F4 while the component has the focus will give you a popup property page, etc.

To set the control's properties manually, enter values in the property page. To set them programmatically, use the syntax described in this document.

It is possible to place multiple WorldView for Developers controls in the same form. They can be control arrays or not. Note that this may increase the hardware requirements for your application.

## Positioning and Resizing the Control in a Visual Basic Form

- 1 Click on the control on the form.
- 2 Drag the control around to move it. The control may also be moved by entering its desired position in pixels using the Top and Left properties of the control.
- 3 Place your mouse over the borders, indicated by squares on the sides of the control, and drag to resize the control as desired. Selecting and dragging a corner will resize the control's width and height simultaneously. The size may also be entered in pixels using the Height and Width properties of the control.

## Getting a Reference to the Control

By default, the control you place on your form will be named WorldView1. This can be changed to any desired name using Visual Basic's property page. The examples below assume the default name.

Referring to a control's properties always follows this syntax:

```
assignment:  
control.property = property value
```

```
reference:  
some variable = control.property
```

where control is the name of the WorldView for Developers control (in this case, WorldView1) and property is any of the valid properties of the control such as World, DashboardEnabled, or FullColor.

- ▶ To display the URL of the current VRML file loaded into the control in a popup message box:

```
MsgBox WorldView1.World
```

- ▶ To set the URL of the VRML file programmatically:

```
WorldView1.World = "C:\MyWorld.wrl"
```

### Getting a Reference to the VrmlBrowser Object

To be able to fully manipulate the VRML scene you must have a reference to the VrmlBrowser.

- ▶ Declare variables for the WorldView Control and Browser Objects (this is an example of early (Visual Basic compile time) binding):

```
Dim browser As VrmlBrowser
Dim factory as IVrmlObjectFactory

Set browser = WorldView1.GetBrowser
Set factory = browser
```

---

**Note** • You must declare an IVrmlObjectFactory because Visual Basic supports only one interface to an Object, and it will not cast one object to another type, as in Java or C++. IVrmlObjectFactory must be implemented to create new VRML data types such as ConstMFNode, SFString, etc.

---

Now you can get or set information about the browser, create VRML data types, and add nodes to the scene.

- ▶ To show the current frame rate:

```
Label1.Caption = "Current frame rate is " +
browser.GetCurrentFrameRate
```

### Adding a VRML Primitive to the Scene

Once you have a reference to the browser, use the External Authoring Interface for VRML that communicates with the scene via COM to add a VRML primitive to the scene.

There are several ways to add nodes to the scene, the simplest is to use CreateVrmlFromString. You can use the World property to load a preexisting VRML file.

- ▶ To add a blue cone to the scene (after declaring variables for the browser):

```

' declare VRML variables
Dim vrmlString as String
Dim addChildren as VrmIMFNode
Dim geom as VrmConstMFNode
Dim root as VrmNode
Dim coneNode as VrmBaseNode
Dim mfnode as VrmIMFNode

' load a blank world containing only a Transform DEF'd Root
' (refer to VRML Spec for more details on this command)
WorldView1.World = "c:\blankworld.wrl"

' Get a reference to the transform node in the loaded world
' Get a reference to the addChildren eventIn of the transform node
browser.GetNode("Root", root)
Set addChildren root.GetEventIn("addChildren")

' Create a VRML string for the cone
vrmlString = "Transform { translation 5 0 0 children [ "
vrmlString = vrmlString + " DEF touch TouchSensor{}"
vrmlString = vrmlString + " Shape { appearance Appearance "
vrmlString = vrmlString + " { material DEF mat Material { "
vrmlString = vrmlString + " diffuseColor 0 0 1 } } "
vrmlString = vrmlString + " geometry Cone {} ] } "

' Add the cone to the Root Transform in the scene
browser.CreateVrmIFromString(vrmlString, geom)
factory.CreateVrmIMFNode(mfnode)

' Get the first node (in this case, the Cone) and add it to the
' array of nodes which addChildren wants as a parameter
geom.GetIValue 0, coneNode
mfnode.AddValue geom.GetIValue(0)
addChildren.SetValueFromMFNode mfnode

```

If blankworld.wrl looked like this:

```

#VRML V2.0 utf8
DEF Root Transform { children [] }

```

A blue cone should appear in the scene 5 meters (VRML world meters) to the right of center.

## Removing a Node from the Scene

- ▶ Follow the same process as adding to a scene but declare another eventIn MFNode for removeChildren:

```
Dim removeChildren as VrmMFNode
```

```
root.GetEventIn("removeChildren", removeChildren)  
removeChildren.SetValue mfnode
```

Be certain that the values in MFNode are references to the nodes you want removed.

## Changing the Fields of a Node in a VRML Scene

- ▶ Generate a button on your VB form called rotateButton, which will rotate the cone clockwise along its X axis, and alternate its color from blue to red when selected by the user. (This example builds on the previous.)

```
' Get reference to the rotation exposedField and the
Dim coneRotation As VrmlSFRotation
Dim coneColor As VrmlSFColor
Dim coneMaterial As VrmlNode
Dim angle As Single
Dim red As Single

angle = 0
browser.GetNode("mat", coneMaterial)
set coneRotation=root.GetExposedField("rotation")
set coneColor=mat.GetExposedField("diffuseColor")

Sub rotateButton_Click()

    If angle >= 3.14 Then
        angle = 0
    End If

    angle = angle + .785
    coneRotation.SetValue(1, 0, 0, angle)
    red=coneColor.GetRed

    If red = 1 Then
        coneColor.SetValue 0, 0, 1
    Else
        coneColor.SetValue 1, 0, 0
    End If

End Sub
```

### Receiving Events from a VRML Scene

Make sure references to `IVrmlEvent`, `IVrmlField`, `IVrmlConstField` are set via the Project ► Reference menu in Visual Basic.

You must set up a call back in Visual Basic so that your program can be notified when an `eventOut` occurs from the VRML scene. The best way to do that is to create an `IVrmlEventOutObserver` class in Visual Basic.

#### To receive information when a user clicks on the cube

- 1 Create a new class and call it `ConeEventObserver`.
- 2 In the declaration section implement `IVrmlEventOutObserver`:

```
Implements IVrmlEventOutObserver
```

- 3 Then, implement the `Callback` method. Because it is an abstract class, you must implement all the methods declared as members of `IVrmlEventOutObserver`. The `Callback` method is called when the `WorldView` control generates an `eventOut` that the `IVrmlEventOutObserver` is observing.

This example method generates a popup message box saying “`eventOut caught`” whenever the user clicks the cone:

```
Private Sub Vrm1EventOutObserver_Callback(ByVal value As  
IVrmlConstField, ByVal timeStamp As Double, userData As Variant)  
    MsgBox “eventOut caught with tag = ” + format(userData)  
End Sub
```



- ▶ In the declaration section of your form enter:

```
Dim tag As Integer
Dim coneEvents As ConeEventObserver
Dim coneTouch As Vrm1Node
Dim coneTouchTime As Vrm1ConstSFTime

'Create an instance of ConeEventObserver
Set coneEvents = New ConeEventObserver

'Get a reference to the cone's touchSensor
Set coneTouch = browser.GetNode("touch")

set coneTouchTime=coneTouch.GetEventOut("touchTime")
tag = 1
coneTouch.Advise(coneEvents, tag)
```

When the user clicks on the cone, you should see a Visual Basic popup message box saying "eventOut caught with tag = 1."

## Passing Arrays to Methods

When a method requires an array as a parameter, you must pass the first element of the array. You must also specify the number of elements in the array being passed.

For example:

```
Dim SingleArray (0 to 5) As Single
Dim AnMFVec3f As Vrm1MFVec3f

For i = 0 to 5
    SingleArray(i) = i
Next i

AnMFVec3f.SetValue 6, SingleArray(0)
```

### Invoking an OCX from the Script node

The Script node is a powerful tool for programming behavior in a scene. However, the VRML 2.0 specification describes only the use of Java and JavaScript as scripting languages. As a result, the majority of VRML browsers support only these languages in the Script node. To enable you to write scripted behaviors in other programming languages, WorldView for Developers provides the ability to invoke an OCX from the Script node. Using this feature, you can write an OCX in any COM-capable language to control the VRML scene.

We strongly suggest that to create scripts written in Java, you use the VRML 2.0 Java in Script node interface, rather than the COM interface.

#### Supported protocol in the Script node's url field

An OCX is invoked from a Script node by using the clsid: protocol. Using the clsid: protocol, specify the GUID of your script OCX as follows:

```
Script {  
  url "clsid:62107a01-febf-11d0-bbc0-444553540000"  
}
```

The OCX must be registered on the user's computer before WorldView loads the VRML file. WorldView can not yet automatically download an OCX from the Internet and verify it using Microsoft Authenticode. Therefore, the OCX must be present on the user's system before the VRML file is loaded.

The clsid: protocol is a WorldView extension to the Script node, and is not currently supported by other VRML browsers. To ensure that your VRML file runs in other browsers, you must provide a "fallback" script which is executed if OCX functionality is not available. This example falls back on JavaScript:

```
Script {  
  url [ "clsid:62107a01-febf-11d0-bbc0-444553540000"  
        "javascript:  
          function initialize() { print('Could not load OCX'); }" ]  
}
```

The COM object implementing the script must support the IVrmlScriptImplementation interface or expose the methods of the interface through OLE Automation.

For a complete description of the IVrmlScriptImplementation Interface, see *Chapter 9, VrmlScriptImplementation Interface*.

For a complete description of the IVrmlScriptNode Interface, see *Chapter 9, VrmlBaseNode Objects*.

For a complete description of the IVrmlEvent Interface, see *Chapter 9, VrmlEvent Object*.

## **Embedding in Visual Basic at Runtime**

In Visual Basic, it is NOT necessary to explicitly acquire the runtime license key as in C++ and J++. Visual Basic will automatically embed the WorldView for Developers runtime license key in your .exe when you compile your Visual Basic program.

## **Samples**

For an example of an OCX that implements scripted behaviors in a VRML scene, see *Invoking an OCX from a Script node: OCXDemo*.

## Working with Web Pages: HTML and Java

### Getting a Reference to the Control

To set up a reference to the browser from an HTML page, embed a VBScript script, such as the sample script shown below, in the HTML page that will call a function in your Java class, then pass the WorldView control in the <OBJECT> tags to the Java class. (See [Chapter 3, The WorldView Browser](#) for more information.)

```
<script language="VBScript">  
  sub window_onLoad  
    document.WVforDevTest.SetControl WorldView  
  end sub  
</script>
```

## Embedding a WorldView for Developers File in an HTML Page

```
<html>
<head>
<title>WVforDev Test</title>
</head>

<body bgcolor=#>
<center>
<h3>W V f o r D e v   &nbsp; C O N T R O L   &nbsp; T E S T</h3>
</center>

<object id="WorldView" CLASSID="clsid:0939A9F5-1788-11d2-8FDE-
0060975B8649" height=250 width=100% align=center MAYSCRIPT>
<param name="World" value="output.wr1">
</object>

<p>
<applet id="WVforDevTest" code="WVControlTest.class" height=300
width=100%>
</applet>

<script language="VBScript">
  sub window_onLoad
    document.WVforDevTest.SetControl WorldView
  end sub
</script>

</body>
</html>
```

Once this VBScript has been embedded in your HTML page, references between the browser and the HTML page can be made.

### **Getting a Reference to the VrmlBrowser Object**

- 1** Import these java packages:

```
import axworldview.*;
import vrmlbrowser.*;
```

- 2** Declare variables for the control, browser, and factory objects:

```
IWorldView WorldViewControl = null;
IVrmlBrowser browser = null;
IVrmlObjectFactory factory = null;
```

- 3** Create the SetControl method that the VBScript on the HTML page will call when the page is loaded:

```
public void SetControl(Object o) {
    WorldViewControl=(IWorldView) o;
}
```

When the page is loaded, the VBScript described in the previous example calls this method, then passes a reference to the embedded WorldView for Developer object on the same HTML page. (The reference is passed by calling the method in the embedded Java applet.)

Once the reference to the WorldView object is set, it can be used in the rest of the code as follows (which shows how to get the reference to the VrmlBrowser object within the control):

- ▶ To run from the `init()` method of the applet

```
public void initVrml() {
    boolean testDone, worldDone, browserDone;
    testDone=worldDone=browserDone=false;
    while(true) {
        while (WorldViewControl==null) { }
        if (worldDone==false) worldDone=true;
        if (browser==null && !browserDone) {
            browser = (IVrmlBrowser)WorldViewControl.GetBrowser();
            browserDone=true;
        }
        if (browser!=null && WorldViewControl !=null && !testDone) {
            testDone=true;
        }
        if (testDone==true) { break; }
        try {
            Thread.sleep(200);
        }
        catch (InterruptedException e) {}
        if (testDone==true) break;
    }
}
```

This gives you access to the browser.

- ▶ To display browser info out to the Java console:

```
String strOut;
strOut = "Browser Info: " +
"Current World URL is " + ((IWorldViewDeveloper)
    WorldViewControl).getWorld() + "\n" +
"Current Viewpoint is " + ((IWorldViewDeveloper)
    WorldViewControl).getViewPoint() + "\n" +
getNavMode(((IWorldViewDeveloper)
    WorldViewControl).getNavigationMode()) + "\n";
System.out.println(strOut);
```

### Adding a VRML Primitive to the Scene

Once you have a reference to the browser, a VRML Primitive may be added to the scene using the External Authoring Interface for VRML that communicates with the scene via COM.

There are several ways to add nodes to the scene. The simplest is to use `CreateVrmlFromString`. You can use the `World` property to load a preexisting VRML file.

- ▶ To add a blue cone to the scene (after declaring variables for the browser), add these to the top of your Java program:

```
import platinum.javacom.vrmlbasenode.*;
import platinum.javacom.vrmlnode.*;
import platinum.javacom.vrmlmfnode.*;
import platinum.javacom.vrmlconstmfnode.*;

IVrmlNode root;
IVrmlConstMFNode geom;
IVrmlMFNode addChildren;
IVrmlMFNode mfnode;

String vrmlString;

root=browser.GetNode("Root");
addChildren = (IVrmlMFNode) root.GetEventIn("addChildren");
factory = (IVrmlObjectFactory) browser;

' Create a VRML string for the cone
vrmlString = "DEF Transform { translation 5 0 0 children [ " +
    " DEF touch TouchSensor{}" +
    " DEF Shape { appearance Appearance " +
    " { material DEF mat Material { " +
    " diffuseColor 0 0 1 } } " +
    " geometry Cone { } ] } ";

geom = CreateVrmlFromString(vrmlString);
mfnode.Set1Value(0, geom.Get1Value(0));
addChildren.SetValue(mfnode);
```



## Removing a Node from the Scene

- ▶ Use the same technique as adding to a scene, but declare another eventIn MFNode for removeChildren:

```
IVrmMFNode removeChildren;  
removeChildren = (IVrmMFNode) root.GetEventIn("removeChildren");  
removeChildren.SetValue(mfnode);
```

Be certain that the values in MFNode are the references to the node(s) you want to remove.

### Changing the Fields of a Node in a VRML Scene

- ▶ Generate a button in a Java applet called rotateButton, which will rotate the cone clockwise along its X axis, and alternate its color from blue to red when selected by the user: (This example builds on the previous.)

```
import vrmlsfcolor.*;
import vrmlsfrotation.*;
import vrmlnode.*;

IVrmlSFRotation coneRotation;
IVrmlSFColor coneColor;
IVrmlNode coneMaterial;

float angle;

angle = 0;
coneMaterial = root.GetNode("mat");
coneRotation = root.GetNode("rotation");
coneColor = mat.GetExposedField("diffuseColor");

public boolean action(Event e, Object what) {
    if (e.target==rotateButton) {

        if (angle >= 3.14) angle = 0;

        angle += .785;

        coneRotation.SetValue(1, 0, 0, angle);

        if (coneColor.GetRed() == 1) {
            coneColor.SetValue(0, 0, 1);
        }
        else {
            coneColor.SetValue(1, 0, 0);
        }
    }
}
```

## Receiving Events from the VRML Scene

Your applet must implement the `IVrmlEventOutObserver` interface.

The example applet below has its own `TouchTime` `EventOutObserver`:

- 1 Import `com.ms.com` to access COM components (such as `Variants`).

```
import com.ms.com.*;
```

- 2 These `WorldView for Developers` packages are also required:

```
import platinum.javacom.vrmlbrowser.*;
import platinum.javacom.vrmlconstsftime.*;
import platinum.javacom.vrmleventoutobserver.*;
import platinum.javacom.vrmlnode.*;
import platinum.javacom.worldview.*;
import platinum.javacom.vrmlconstfield.*;
```

```
import java.applet.*;
import java.awt.*;
```

```
public class YourClass extends Applet implements IVrmlEventOutObserver {

    private static TextField myTextArea=null;
    private IVrmlBrowser myBrowser=null;
    private IVrmlConstSFTIME myTouchTime=null;
    private TouchTimeObserver myTouchTimeObserver=null;
        int calledHowManyTimes=1;
```

- 3 As in previous examples, this method is called from a script on the HTML page that passes a reference to the embedded ActiveX control (`WorldView for Developers`) so the applet can get a reference to the `VrmlBrowser` object.

```
public void SetControl(Object control) {
    //Get reference to browser
    IWorldView worldViewControl=(IWorldView) control;
    myBrowser = worldViewControl.GetBrowser();
    initVrml();
}
```

- 4 Set the UI to show a scrolling text box so the callback method can print the number of times the object is touched:

```
public void init() {
    myTextArea = new TextField(30);
    myTextArea.setEditable(false);
    add(myTextArea);
}
```

- 5 Get references to the touchSensor node in the VRML scene (in this case, there was a touchSensor DEF'd to). Instance a new observer, and set the Advise method to notify the callback routine when an event has occurred.

```
public void initVrml() {
    //Get reference to touchSensor
    IVrmlNode touchSensorNode = myBrowser.GetNode("t");
    myTouchTime = (IVrmlConstSFTime)

    touchSensorNode.GetEventOut("touchTime");
    myTouchTimeObserver = new TouchTimeObserver(myTextArea);
    myTouchTime.Advise(this, new Variant(null));
}
```

- 6 The following must be implemented: the field is the VRML field that you wish to track events for; ts is the timestamp of the event; and userData is a Variant.

```
    public void Callback(IVrmlConstField field, double ts, Variant userData)
    {
        myTextArea.setText("touched"+String.valueOf(calledHowManyTimes++)
+ " times\n");
    }
}

class TouchTimeObserver implements IVrmlEventOutObserver {
    private TextField myTextArea=null;

    TouchTimeObserver(TextField ta) { myTextArea=ta; }

    public void Callback(IVrmlConstField f, double ts, Variant userData) {
        myTextArea.setText("callback called with user data " + userData);
    }
}
```

- ▶ The following sample VRML file describes a sphere that changes between blue and red each time it is touched.

```
#VRML V2.0 utf8

DEF Root Transform {
  children [
    DEF b Shape {
      appearance Appearance {
        material DEF mat Material {
          diffuseColor 1 0 0 }
      }
      geometry Sphere { radius 5 }
    }
    DEF t TouchSensor {}
    DEF s Script {
      eventIn SFFloat touched
      field SFFloat currColor 1 0 0
      eventOut SFFloat newColor
      url "javascript:
function touched() {
  if (currColor[0]==1) {
    currColor[0]=0; currColor[1]=0; currColor[2]=1;
  }
  else {
    currColor[0]=1; currColor[1]=0; currColor[2]=0;
  }
  newColor = currColor;
}"
    }
  ]
}

ROUTE t.touchTime TO s.touched
ROUTE s.newColor TO mat.diffuseColor
```

## ■ **WorldView for Developers Containers**

---

*Working with Web Pages: HTML and Java*

---

# WorldView for Developers Runtimes

This chapter describes the requirements necessary for runtime applications generated using WorldView for Developers.

|  |     |
|--|-----|
| <b>Generating Runtime Applications</b> .....               | 5-2 |
| <b>Requirements for Runtime Applications</b> .....         | 5-2 |
| <b>Including your own Help files</b> .....                 | 5-5 |
| <b>WorldView License Agreement</b> .....                   | 5-5 |
| <b>Embedding WorldView for Developers at Runtime</b> ..... | 5-6 |
| Embedding in C++ at Runtime .....                          | 5-6 |
| Embedding in J++ at Runtime .....                          | 5-7 |
| Embedding in Visual Basic at Runtime .....                 | 5-7 |

# Generating Runtime Applications

If you do not wish to generate your own installer, you may also use the WorldView for Developers 2.1 Runtime installer to install the WorldView OCX on your users' machines. This package is in a .zip archive available free from PLATINUM. (Please contact us at [support@intervista.com](mailto:support@intervista.com) for more information.) The installer was created using InstallShield5. Due to a known bug in InstallShield5, earlier versions of InstallShield, and other applications, may not be able to call the Setup.exe correctly.

The Runtime installer is a 4MB .zip archive, which includes:

- WorldView for Developers 2.1 CAB files,
- Setup.exe files (generated by InstallShield),
- InstallShield script, and
- support file.

# Requirements for Runtime Applications

Applications created using WorldView for Developers require the same components as WorldView:

- DirectX 5,
- DirectShow (if the application contains Movie Textures), and the
- Java VM (if the application contains Java).

These components are not included in the WorldView for Developers installer, and should be installed by the application itself, if necessary. These components may be licensed from Microsoft for free. See:

<http://www.microsoft.com/directx/resources/dx5mediaruntime.htm>,

<http://www.microsoft.com/directx/resources/enddl.htm>, and

<http://www.microsoft.com/java/vm/vmdownload.htm> for details.

Your application installer must also install the OCX and DLLs required by WorldView. All required files, with the exception of 3rd party installers, may be found on your machine in the directories listed below.



The WorldView OCX requires that the following installers be run to completion before registration:

- NT Service Pack 3 (for Windows NT)  
[http://support.microsoft.com/download/support/mslfiles/NT4SP3\\_I.EXE](http://support.microsoft.com/download/support/mslfiles/NT4SP3_I.EXE)

Requires a reboot before installation is complete.

- DirectX 5 (for Windows 95)  
<http://www.microsoft.com/directx/resources/enddl.htm>

May require a reboot before installation is complete.

- Microsoft VC++ Axdist.exe installer

- Microsoft VM for Java installer  
<http://www.microsoft.com/java/vm/vmdownload.htm>,

The WorldView OCX requires that several shared DLLs exist before registration. The following files must be installed in the same directory as the WorldView OCX, and are self-registering DLLs.

MMAR.AX  
MMVR.AX

The following files must be installed in the appropriate Windows system directory:

MSVCRT.DLL  
MFC42.DLL  
OpenGL32.DLL (for Windows 95)  
GLU32.DLL (for Windows 95)

The WorldView OCX requires that the following Java Class archive be installed in the same directory as the WorldView OCX:

axWorldView.zip

The following registry keys must exist before the WorldView OCX is registered:

```
HKEY_CURRENT_USER\Software\PLATINUM technology\WorldView for
DeveLopers 2.1\ActiveX\Main\Hardware Acceleration=1

HKEY_LOCAL_MACHINE\Software\PLATINUM technology\WorldView for
DeveLopers 2.1\ActiveX\Main\
  Support Directory=<path to WorldView OCX>
  Help Directory=<full path of help directory: same as WorldView
for Developers help>
```

You may use the WorldView HTML help files provided in the Distribution directory, or you may use your own help file. This help file will be displayed from the right mouse button popup menu Help item.

Self-registration of the WorldView OCX will prepend the Support Directory with axWorldView.zip to the HKEY\_LOCAL\_MACHINE\Software\Microsoft\JavaVM Classpath key. If this value is not the correct path to the axWorldView.zip, then Java will not function properly with the WorldView OCX.

Once all of the above conditions have been met, the WorldView OCX may be registered. WorldView.ocx is a self-registering OCX (ActiveX Control). If you are not using an installation kit, you should use regsvr32.exe to perform self-registration.

If your installer will install DirectX 5, you may set the following registry key to ensure that the WorldView OCX will be registered after any DirectX 5 files have been registered:

```
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnce
%REGISTER_WORLDVIEW%=regsvr32 /s "<path to WorldView
OCX>\WorldView.ocx"
```

---

**Note** • We do not recommend installing the NT Service Pack 3 from another installer. Users should be instructed to install NT Service Pack 3 before starting your installation.

---

## Including your own Help files

WorldView for Developers automatically includes the WorldView browser help files, in HTML format, in all runtime applications generated. These files are accessed from the right mouse menu item Help while the mouse is within the WorldView window in runtime mode. This help system originates with the file named `helpindex.html` in the `WorldView\Help` folder.

To include your own help files, or to amend and include ours as well, you may simply edit the included HTML files using any HTML editor.

You may also include help files using the `IWorldViewDeveloper` property: `UserHelpFile`. Your help system may be in either HTML or WinHelp format. The location of your file may be entered either as a full path, or as a relative path using the WorldView for Developers executable as its origin.

## WorldView License Agreement

The WorldView Browser License Agreement `WVLicense.txt`, must be installed with your runtime applications if no other license is to be included. Please include this file in the root directory of your installation.

This file is installed in the WorldView for Developers Distribution folder.

# Embedding WorldView for Developers at Runtime

Entering your Password enables WorldView for Developers. It does not affect your runtime applications, which must be enabled individually.

WorldView for Developers supports runtime license keys, which enables you to embed WorldView for Developers in your application and run on a user's machine without a .lic file present. This is done using the standard ActiveX mechanism for runtime licensing: WorldView for Developers gives you a runtime license key, which you embed in your application.

To acquire the runtime license key, select WorldView for Developers ▶ Copy Key To Clipboard from your computer's Start menu. This will copy the runtime license key to the Windows clipboard. You can now paste it into your application.

## Embedding in C++ at Runtime

In C++, the MFC framework permits you to specify a runtime license key when creating an instance of an ActiveX Control.

- ▶ First, paste the runtime license key into your application's source code:

```
#define RUNTIME_LICENSE_KEY L"<license key pasted from clipboard>"
```

- ▶ To create the WorldView for Developers control, specify the license key as an argument to the Create method as follows:

```
BOOL bResult = m_pWorldView->Create(  
    NULL,  
    "WorldView",  
    WS_CHILD|WS_VISIBLE,  
    rect,  
    this,  
    0,  
    RUNTIME_LICENSE_KEY  
);
```

## Embedding in J++ at Runtime

- ▶ In J++, paste the runtime license key into your application's source code:

```
private static final String RUNTIME_LICENSE_KEY = "<license key>";
private static final String
    WORLDVIEW_CLSID = "{0939A9F5-1788-11d2-8FDE-0060975B8649}";
```

- ▶ To create the WorldView for Developers control, you will need to use the class `com.ms.com.LicenseMgr` which enables you to create COM objects using a license key.

```
import com.ms.com.*;

...
ILicenseMgr licenseMgr = (ILicenseMgr) new LicenseMgr();
Object control = licenseMgr.createWithLic(
    RUNTIME_LICENSE_KEY,
    WORLDVIEW_CLSID,
    null,
    ComContext.INPROC_SERVER
);
```

The `createWithLic` method returns a raw Java/COM object. You may wish to convert this to an instance of `com.ms.com.ActiveXControl` which can be treated as an AWT component and easily added to a window of your Java application.

```
ActiveXControl = new ActiveXControl((IUnknown)control);
```

## Embedding in Visual Basic at Runtime

In Visual Basic, it is NOT necessary to explicitly acquire the runtime license key as in C++ and J++. Visual Basic will automatically embed the WorldView for Developers runtime license key in your .exe when you compile your Visual Basic program.

## ■ **WorldView for Developers Runtimes**

---

*Embedding WorldView for Developers at Runtime*

---

## The WorldView COM Object

This chapter describes the WorldView COM object, and lists and defines its properties and methods.

|  |                        |                   |
|--|------------------------|-------------------|
| <b>Introduction</b> .....                  | <b>6-2</b>             |                   |
| <b>IWorldView Interface</b> .....          | <b>6-3</b>             |                   |
| Properties                                 |                        |                   |
| World                                      |                        |                   |
| Methods                                    |                        |                   |
| GetBrowser                                 |                        |                   |
| <b>IWorldViewDeveloper Interface</b> ..... | <b>6-4</b>             |                   |
| Properties                                 |                        |                   |
| AutoRotate                                 | ConsoleVisible         | DashboardEnabled  |
| DashboardOn                                | Dithering              | FullColor         |
| GraphicsMode                               | GraphicsModeWhenMoving |                   |
| HeadlightOn                                | HighQualityText        | LoadTextures      |
| NavigationMode                             | NavigationSpeed        | PopupMenuEnabled  |
| PreventCollisions                          | SplashScreenEnabled    | UseAcceleration   |
| UserHelpFile                               | Viewpoint              | WebLinkEnabled    |
| World                                      |                        |                   |
| Methods                                    |                        |                   |
| GetBrowser                                 | NextViewpoint          | PreviousViewpoint |
| Reload                                     | RestoreView            | StraightenUp      |
| ViewAll                                    |                        |                   |

# Introduction

The WorldView COM object represents the WorldView ActiveX Control itself. This is the only object in the WorldView COM API which can be created by GUID, for example, via the CoCreateInstance function in the COM library.

The WorldView COM object supports two interfaces: IWorldView and IWorldViewDeveloper. The IWorldViewDeveloper interface is available only in WorldView for Developers. The IWorldView interface is available both in the WorldView browser and in WorldView for Developers. The IWorldViewDeveloper interface inherits from the IWorldView interface.

The GUID used to create the WorldView COM object differs in WorldView and WorldView for Developers:

WorldView {b0d7d800-4ebf-11d0-9490-00a02494d8a5}

WorldView for Developers {0939A9F5-1788-11d2-8FDE-0060975B8649}

This allows the two applications, as well as any runtime applications generated by WorldView for Developers, to be installed on a single machine without complications.



## IWorldView Interface

### Properties

#### World

**Prototype**(*Get*) World([out, retval] BSTR\* url);

**Prototype**(*Set*) World([in] BSTR url);

Returns the URL of the active VRML world, or replaces the active world with the VRML world located at the specified URL.

### Methods

#### GetBrowser

**Prototype** GetBrowser([out, retval] IVrmlBrowser\*\* browser);

Returns the IVrmlBrowser interface to the VrmlBrowser object of this browser instance. This interface can be used to control the VRML scene in a fashion similar to the External Authoring Interface (EAI). See *IVrmlBrowser Interface* for details.

# IWorldViewDeveloper Interface

The IWorldViewDeveloper interface inherits from the IWorldView interface. This interface is accessible only in WorldView for Developers, and not in the standard WorldView browser.

## Properties

### AutoRotate

**Prototype**(*Get*) AutoRotate([out, retval] boolean\* autoRotate);

**Prototype**(*Set*) AutoRotate([in] boolean autoRotate);

Returns or sets the state of auto-rotation: TRUE if auto-rotation is enabled; FALSE if disabled.

Controls whether WorldView automatically rotates objects. If auto-rotation is enabled, an object rotated in study mode will continue to spin after the mouse button is raised.

This setting can also be controlled using the Rotate Object Automatically check box in the Options dialog.

### ConsoleVisible

**Prototype**(*Get*) ConsoleVisible([out, retval] boolean\* consoleVisible);

**Prototype**(*Set*) ConsoleVisible([in] boolean consoleVisible);

Returns or sets whether the VRML console is currently visible: TRUE if visible; FALSE if hidden.

The VRML console window will be placed on the screen in the location last occupied in the most recent WorldView session.

## DashboardEnabled

**Prototype** (*Get*) DashboardEnabled([out, retval] boolean\* dashboardEnabled);

**Prototype** (*Set*) DashboardEnabled([in] boolean dashboardEnabled);

Controls whether WorldView's dashboard is enabled and available for use: TRUE if enabled; FALSE if disabled.

The dashboard is the blue bar along the left and bottom of the WorldView window which permits the user to control WorldView's navigation features. It may be desirable to turn off the dashboard in applications which do not require WorldView's navigation features.

If the dashboard is enabled, it may be selected from the right mouse menu, and its features may be used. Disabling the dashboard removes the item from the menu, and makes the dashboard itself unavailable for use in the application.

This setting can also be controlled using the Show Navigation Bar item on the browser's right-mouse menu.

## DashboardOn

**Prototype** (*Get*) DashboardOn([out, retval] boolean\* dashboardOn);

**Prototype** (*Set*) DashboardOn([in] boolean dashboardOn);

Controls whether WorldView's dashboard is visible by default when your application is first launched: TRUE if visible; FALSE if hidden.

DashboardEnabled must be set to TRUE for DashboardOn to have any visible effect.

The user may show or hide the dashboard using the Show Navigation Bar item on the browser's right-mouse menu.

#### Dithering

**Prototype**(*Get*) Dithering([out, retval] boolean\* dithering);

**Prototype**(*Set*) Dithering([in] boolean dithering);

Returns or sets whether WorldView dithers while rendering: TRUE if dithering is active; FALSE if inactive.

Dithering improves rendering quality, but may lower performance.

This setting may also be controlled using the Graphics ► Dithering item of the browser's right-mouse menu.

#### FullColor

**Prototype**(*Get*) FullColor([out, retval] boolean\* fullColor);

**Prototype**(*Set*) FullColor([in] boolean fullColor);

Returns or sets WorldView's Full Color setting: TRUE if full color is active; FALSE if inactive. See also [WorldView's Full Color Setting](#).

This setting can also be controlled using the Graphics ► Full Color item of the browser's right-mouse menu.

#### GraphicsMode

**Prototype**(*Get*) GraphicsMode([out, retval] enumGraphicsMode graphicsMode);

**Prototype**(*Set*) GraphicsMode([in] enumGraphicsMode graphicsMode);

Returns or sets the current graphics mode. The graphics mode controls how the browser renders the VRML scene.

The current graphics mode is stored as type enumGraphicsMode and may take the following values:

|               |     |   |
|---------------|-----|---|
| SMOOTHSHADING | = 0 | (Slowest, Highest Resolution Rendering) |
| FLATSHADING   | = 1 |   |
| WIREFRAME     | = 2 | (Fastest, Lowest Resolution)            |

This setting can also be controlled using the Graphics popup menu on the browser's right mouse menu.

## GraphicsModeWhenMoving

**Prototype** (*Get*) GraphicsModeWhenMoving([out, retval] enumGraphicsMode\*graphicsMode);

**Prototype** (*Set*) GraphicsModeWhenMoving([in] enumGraphicsModegraphicsMode);

Contains the graphics mode used while the user is moving (navigating). This graphics mode is activated when navigation begins. When navigation ends, the normal graphics mode is restored. This property may be used to accelerate navigation in a large world by switching to a faster, lower-resolution graphics mode such as Wireframe while the user is moving through the space.

This setting can also be controlled using the Graphics ▶ When Moving item on the browser's right mouse menu.

The current graphics mode is stored as type enumGraphicsMode and may take the following values:

|               |     |   |
|---------------|-----|---|
| SMOOTHSHADING | = 0 | (Slowest, highest resolution)                   |
| FLATSHADING   | = 1 |   |
| WIREFRAME     | = 2 | (Fastest, lowest resolution)                    |
| NOCHANGE      | = 3 | (The graphics mode is not changed to navigate.) |

## HeadlightOn

**Prototype** (*Get*) HeadlightOn([out, retval] boolean\* headlightOn);

**Prototype** (*Set*) HeadlightOn([in] boolean headlightOn);

Returns or sets whether the headlight is enabled: TRUE if the headlight is on, and FALSE if the headlight is off.

The headlight is a directional light which points in the same direction as the 3D camera in the VRML scene. It is a simple means to illuminate a VRML scene.

The state of the headlight can also be controlled using the Graphics popup menu of the browser's right mouse menu.

#### HighQualityText

**Prototype**(*Get*) HighQualityText([out, retval] boolean\* highQualityText);

**Prototype**(*Set*) HighQualityText([in] boolean highQualityText);

Returns or sets whether WorldView uses high quality text: TRUE if high quality text is enabled; FALSE if disabled.

High quality text is constructed from polygons rather than bitmaps, and therefore scales better and is less pixilated. However, with long text strings, it is sometimes slower than “low-quality” text.

This setting can also be controlled using the High Quality Text check box in the Options dialog.

#### LoadTextures

**Prototype**(*Get*) LoadTextures([out, retval] boolean\* loadTextures);

**Prototype**(*Set*) LoadTextures([in] boolean loadTextures);

Controls whether WorldView loads textures: TRUE loads textures; FALSE does not.

Not loading textures improves load time and rendering speed, but may dramatically change the appearance of the VRML scene.

This setting can also be controlled using the Graphics ▶ Load Textures item of the browser’s right-mouse menu.

---

**Note** • Setting this property will not affect a VRML world that is already loaded, and will only affect worlds that are loaded after it is set.

---

## NavigationMode

**Prototype** (*Get*) `NavigationMode([out, retval] enumNavigationMode* navigationMode);`

**Prototype** (*Set*) `NavigationMode([in] enumNavigationMode navigationMode);`

Returns or sets the current navigation mode of the browser.

The current navigation mode is stored as the type `enumNavigationMode` which may take one of the following values:

```
DEFAULT = -1
WALK    = 0
PAN     = 1
TURN    = 2
ROLL    = 3
STUDY   = 4
JUMP    = 5
```

---

**Note** • The navigation mode enumeration value for GOTO is JUMP because GOTO is a reserved word in VB.

---

Setting the Navigation Mode to `DEFAULT` specifies that the container does not wish to define a mode, and will instead defer to that defined in the loaded content file itself.

The navigation mode may also be changed using the Movement popup menu of the browser's right-mouse menu.

#### NavigationSpeed

**Prototype** (*Get*) NavigationSpeed([out, retval] enumNavigationSpeed\*navigationSpeed);

**Prototype** (*Set*) NavigationSpeed([in] enumNavigationSpeed navigationSpeed);

Returns or sets the current navigation speed of the browser.

The current navigation speed is stored as the type enumNavigationSpeed which may take one of the following values:

```
VERYSLOW_SPEED = 0
SLOW_SPEED      = 1
MEDIUM_SPEED   = 2
FAST_SPEED      = 3
VERYFAST_SPEED  = 4
```

#### PopupMenuEnabled

**Prototype** (*Get*)PopupMenuEnabled([out, retval] boolean\* popupMenuEnabled);

**Prototype** (*Set*)PopupMenuEnabled([in] boolean popupMenuEnabled);

Returns or sets whether WorldView's popup (right-mouse button) menu is enabled: TRUE if the menu is enabled; FALSE if disabled.

#### PreventCollisions

**Prototype** (*Get*) PreventCollisions([out, retval]boolean\* preventCollisions);

**Prototype** (*Set*) PreventCollisions([in] boolean preventCollisions);

Controls whether the user is permitted to collide with and enter 3D geometry in the VRML scene, or if navigation stops when a collision occurs: TRUE if collisions are prevented; FALSE if collisions are permitted.

This setting can also be controlled using the Movement ► Prevent Collisions item of the browser's right-mouse menu.



## SplashScreenEnabled

**Prototype** (*Get*) SplashScreenEnabled([out, retval] boolean\* splashScreenEnabled);

**Prototype** (*Set*) SplashScreenEnabled([in] boolean splashScreenEnabled);

Controls whether WorldView displays its splash screen while loading a VRML world: TRUE if the splash screen is enabled; FALSE if disabled.

The splash screen will appear only the first time your application is run on any given machine.

## UseAcceleration

**Prototype** (*Get*) UseAcceleration([out, retval] boolean\* useAcceleration);

**Prototype** (*Set*) UseAcceleration([in] boolean useAcceleration);

Returns or sets whether WorldView uses available 3D hardware acceleration: TRUE if hardware acceleration is enabled; FALSE if disabled.

This setting can also be controlled using the Use Hardware Acceleration check box in the Options dialog.

## UserHelpFile

Gets or sets the name of the associated User Help file. This file is accessed in your final application from the right mouse menu item: Help. This file will be the WorldView browser's help file by default.

**Prototype** (*Get*) UserHelpFile([out, retval] BSTR \*userHelpFile);

**Prototype** (*Set*) UserHelpFile([in] BSTR userHelpFile);

#### Viewpoint

**Prototype**(*Get*) Viewpoint([out, retval] BSTR\* viewPoint);

**Prototype**(*Set*) Viewpoint([in] BSTR viewPoint);

Returns or sets the name of the currently bound Viewpoint in the VRML scene. This name is taken from the description field of the Viewpoint node. The name passed in the viewPoint parameter must match the description field of one of the Viewpoint nodes in the VRML scene.

#### WebLinkEnabled

**Prototype**(*Get*) WebLinkEnabled([out, retval] boolean \*WebLinkEnabled);

**Prototype**(*Set*) WebLinkEnabled([in] boolean WebLinkEnabled);

Returns or sets whether the web link is enabled: TRUE enables the link; FALSE disables it.

This link is designed so that clicking on the PLATINUM logo in the bottom right corner of the lower navigation bar launches the user's browser (if necessary), and jumps to the PLATINUM home page URL. Disabling this property prevents this link from occurring.

#### World

**Prototype**(*Get*) World([out, retval] BSTR\* url);

**Prototype**(*Set*) World([in] BSTR url);

Returns or sets the URL of the active VRML file.

## Methods

### GetBrowser

**Prototype** `GetBrowser([out, retval] IVrmlBrowser** browser);`

Returns the `IVrmlBrowser` interface to the `VrmlBrowser` object of this browser instance. This interface can be used to control the VRML scene in a fashion similar to the External Authoring Interface (EAI). See the *IVrmlBrowser Interface* for details. This method is inherited from the `IWorldView` interface.

### NextViewpoint

**Prototype** `NextViewpoint();`

Sets the currently bound viewpoint to be the next viewpoint in the viewpoint list. The viewpoint list is accessible from the Viewpoints popup menu of the browser's right-mouse menu.

### PreviousViewpoint

**Prototype** `PreviousViewpoint();`

Sets the currently bound viewpoint to be the previous viewpoint in the viewpoint list. The viewpoint list is accessible from the Viewpoints popup menu of the browser's right-mouse menu.

### Reload

**Prototype** `Reload();`

Reloads the current VRML scene. The current world is unloaded, and then reloaded from its original location. This can be useful to restart an animation, or if a new version of a world is known to be available.

### RestoreView

**Prototype** `RestoreView();`

Restores the viewpoint to its initial location and orientation.

#### **StraightenUp**


**Prototype** StraightenUp();

Adjusts the current viewpoint to align it with the Y axis of the world. This can be useful to realign the viewpoint with vertical after a series of navigation options have left the viewpoint tilted with respect to the world.

#### **ViewAll**

**Prototype** ViewAll();

Zooms out from the current viewpoint so that the entire VRML scene is visible. This action is identical to that of the Zoom Out button on the navigation bar.

 **7**

---

# WorldView OLE Automation Interface

**This chapter** describes the OLE Automation Interface, a superset of the IWorldViewDeveloper interface visible to Visual Basic users.

**WorldView OLE Automation Interface** ..... 7-2

## WorldView OLE Automation Interface

The WorldView OLE Automation interface is a superset of the IWorldViewDeveloper interface. It contains all its properties and methods, and uses the same prototypes as IWorldViewDeveloper.

It contains one additional property: AboutBox.

### **AboutBox**

**Prototype**      AboutBox();

Invokes the WorldView About Box, which displays the product name, copyright and version number. This method is inherited from the IWorldView interface.

---

# External Authoring Interface using COM

This chapter describes accessing WorldView's External Authoring Interface using Microsoft's Component Object Model (COM).

|   |            |
|---|------------|
| <b>Introduction</b> .....                                   | <b>8-2</b> |
| <b>Using WorldView's COM API from C++</b> .....             | <b>8-3</b> |
| <b>Using COM from Java</b> .....                            | <b>8-5</b> |
| <b>Using WorldView's COM API from other Languages</b> ..... | <b>8-6</b> |
| <b>WorldView for Developers COM API Library</b> .....       | <b>8-6</b> |

## Introduction

WorldView for Developers provides an extensive set of APIs through the Microsoft Component Object Model (COM). The key advantage of COM over traditional libraries is language independence. This means that you can control WorldView from a program written in any language that supports COM: C++, Java, Visual Basic, Delphi, and many others.

The most important concept in COM is the idea of an interface. An interface is a collection of operations, called methods. An object in COM is an entity which supports one or more interfaces. For example, in the WorldView COM API, there is a COM object which represents a VRML SFBool field. This object supports an interface called IVrmlSFBool, which has these methods:

    GetValue: Gets the current value of the SFBool field

    SetValue: Sets the current value of the SFBool field

    ToString: Returns a text string containing the SFBool field's value.

Object-oriented programming languages such as C++ or Java use a pointer or reference to an object to enable you to access the methods of that object. In contrast, COM uses a pointer to one or more of the object's interfaces, instead of a direct pointer to the object.

Several of the COM objects in the WorldView COM API support multiple interfaces. Each interface contains a group of related methods. This is designed to prevent enormous lists of methods under a single interface. Each interface supports a given functionality, which dictates the methods that are supported by the interface.

Given a pointer/reference to an interface of a WorldView COM object, you can access another interface of that object by querying for that interface. To query, specify the interface that you wish to find, and COM will return a pointer/reference to that interface, or an error if the COM object you are querying does not support the interface requested. The way to query for an interface will differ depending on the language used.

To learn more about COM, consult Microsoft's COM web site at <http://www.microsoft.com/com>.



## Using WorldView's COM API from C++

The C++ interface for the COM API is a group of .h files in the include subdirectory of the WorldView for Developers installation directory. To use this interface, add this line to each C++ source file that will be using COM API functionality:

```
#include "ComIfc.h"
```

Make sure that the WorldView for Developers include subdirectory is in your compiler's include path.

The use of COM from C++ is explained in detail in the COM Tutorial in the Microsoft Visual C++ Online Reference. In general, using the COM API is very similar to ordinary programming with C++ objects. A pointer to a COM interface looks like a regular C++ pointer to an object, and methods of the interface can be invoked much like C++ object methods.

Using the COM API differs from normal C++ programming in its use of reference counting. C++ is not a garbage-collected language like Java that transparently manages your memory use. You must inform COM when you intend to hold a reference (pointer) to an object, and you must also inform COM when you wish to release a reference on an object. COM maintains a reference count for each COM object and destroys an object when its reference count reaches zero. Every COM object has two methods used for reference counting:

**AddRef**: Adds a reference to the COM object.  
(Increases reference count by 1.)

**Release**: Releases a reference on the COM object.  
(Decreases reference count by 1.)

Also, given a pointer to a COM interface, you can "query" for another interface supported by the underlying COM object as described in the above section. To query, you use the method `QueryInterface` which is supported by every COM object.

## ■ External Authoring Interface using COM

---

### *Using WorldView's COM API from C++*

- ▶ To use QueryInterface to obtain a pointer to another interface:

```
IVrmIField* pVrmIField;
...
// Obtain the IVrmISFBool interface of the COM object
// from the IVrmIField interface
IVrmISFBool* pVrmISFBool = NULL;
HRESULT hr = pVrmIField->QueryInterface(IID_IVrmISFBool,
    (void**) &pVrmISFBool);
if (FAILED(hr)) {
    // This COM object does not support IVrmISFBool,
    // report an error.
    ...
}
// Got the IVrmISFBool interface, can call its methods now.
pVrmISFBool->SetValue(FALSE);
```

## Using COM from Java

Developers using the Microsoft Java VM and Java compiler can take advantage of WorldView's COM API by using Microsoft Java/COM. Java/COM is a Microsoft extension to Java which enables Java programs to transparently access and implement COM objects.

In Java/COM, COM objects are made available to Java by running the program `jactivex` on the COM object library's TLB files. This will generate Java class files that represent the COM objects. This program is distributed with the Microsoft Java SDK. The Java/COM class files are also distributed with WorldView, and installed on your system when WorldView is installed. These classes reside in the package `platinum.javacom`.

Unlike C++, Java is a garbage-collected language, so the reference counting required for C++ is not necessary in Java. Java will automatically detect when you are done using a particular COM object and destroy it.

COM interfaces are represented in Java by Java interfaces. Querying from one interface to another is as simple as type casting.

- ▶ For example, to query from an `IVrmlField` interface to `IVrmlSFBool`:

```
IVrmlField field;  
...  
IVrmlSFBool sfbool = (IVrmlSFBool) field;  
// Got the IVrmlSFBool interface, call a method of it.  
sfbool.SetValue(false);
```

Microsoft Java/COM is discussed in detail in the documentation for the Microsoft Java SDK, available from <http://www.microsoft.com/java/sdk>.

## **Using WorldView's COM API from other Languages**

It is possible to use the COM API from any programming language that supports Microsoft COM. Consult the documentation from the vendor of the programming language for specifics on that language's integration with COM.

## **WorldView for Developers COM API Library**

This is an external authoring interface, implemented via COM, which enables communication between the VRML scene and other desktop applications such as Visual Basic, Java, and Internet Explorer via Visual Basic and other COM automation compliant scripting languages. This documentation assumes the reader has VRML knowledge. (See The Virtual Reality Modeling Language Specification, V.2.0, ISO/IEC CD 14772.) WorldView external scripting classes are implemented as a set of COM objects.

---

## WorldView for Developers Objects

This chapter describes and defines the Objects available in WorldView for Developers External Authoring Interface.

|  |             |             |                 |
|--|-------------|-------------|-----------------|
| <b>WorldView External Scripting Objects' Structure</b> ..... | <b>9-8</b>  |             |                 |
| <b>VrmlBaseNode Objects</b> .....                            | <b>9-10</b> |             |                 |
| <b>VrmlBaseNode</b> .....                                    | <b>9-11</b> |             |                 |
| GetBrowser   | GetType     | ToString    |                 |
| <b>VrmlNode</b> .....  | <b>9-12</b> |             |                 |
| GetBrowser   | GetEventIn  | GetEventOut | GetExposedField |
| GetType  | ToString    |             |                 |
| <b>VrmlScriptNode</b> .....                                  | <b>9-14</b> |             |                 |
| GetBrowser   | GetEventIn  | GetEventOut | GetField        |
| GetType  | ToString    |             |                 |

|  |                      |                     |                 |  |
|--|----------------------|---------------------|-----------------|--|
| <b>VrmlBrowser Object</b> .....          | <b>9-16</b>          |                     |                 |  |
| AddRoute                                 | CreateVrmlFromString |                     |                 |  |
| CreateVrmlFromURL                        | DeleteRoute          | GetCurrentFrameRate | GetCurrentSpeed |  |
| GetName                                  | GetNode              | GetVersion          | GetWorldURL     |  |
| LoadURL                                  | ReplaceWorld         | SetDescription      |                 |  |
| <b>VrmlEvent Object</b> .....            | <b>9-19</b>          |                     |                 |  |
| Clone                                    | GetName              | GetTimeStamp        | GetValue        |  |
| ToString                                 |                      |                     |                 |  |
| <b>VrmlEventOutObserver Object</b> ..... | <b>9-20</b>          |                     |                 |  |
| Callback                                 |                      |                     |                 |  |
| <b>VrmlField Objects</b> .....           | <b>9-21</b>          |                     |                 |  |
| <b>VrmlField</b> .....                   | <b>9-22</b>          |                     |                 |  |
| Advise                                   | Clone                | GetType             | ToString        |  |
| Unadvise                                 |                      |                     |                 |  |
| <b>VrmlConstField</b> .....              | <b>9-24</b>          |                     |                 |  |
| Advise                                   | Clone                | GetType             | Unadvise        |  |
| <b>VrmlConstMFColor</b> .....            | <b>9-25</b>          |                     |                 |  |
| Advise                                   | Clone                | GetIValue           | GetSize         |  |
| GetType                                  | GetValue             | ToString            | Unadvise        |  |
| <b>VrmlConstMFFloat</b> .....            | <b>9-27</b>          |                     |                 |  |
| Advise                                   | Clone                | GetIValue           | GetSize         |  |
| GetType                                  | GetValue             | ToString            | Unadvise        |  |
| <b>VrmlConstMField</b> .....             | <b>9-28</b>          |                     |                 |  |
| Advise                                   | Clone                | GetSize             | GetType         |  |
| ToString                                 | Unadvise             |                     |                 |  |
| <b>VrmlConstMFInt32</b> .....            | <b>9-30</b>          |                     |                 |  |
| Advise                                   | Clone                | GetIValue           | GetSize         |  |
| GetType                                  | GetValue             | ToString            | Unadvise        |  |

|                                  |             |           |          |  |
|----------------------------------|-------------|-----------|----------|--|
| <b>VrmlConstMFNode</b> .....     | <b>9-32</b> |           |          |  |
| Advise                           | Clone       | Get1Value | GetSize  |  |
| GetType                          | GetValue    | ToString  | Unadvise |  |
| <b>VrmlConstMFRotation</b> ..... | <b>9-34</b> |           |          |  |
| Advise                           | Clone       | Get1Value | GetSize  |  |
| GetType                          | GetValue    | ToString  | Unadvise |  |
| <b>VrmlConstMFString</b> .....   | <b>9-36</b> |           |          |  |
| Advise                           | Clone       | Get1Value | GetSize  |  |
| GetType                          | GetValue    | ToString  | Unadvise |  |
| <b>VrmlConstMFTime</b> .....     | <b>9-38</b> |           |          |  |
| Advise                           | Clone       | Get1Value | GetSize  |  |
| GetType                          | GetValue    | ToString  | Unadvise |  |
| <b>VrmlConstMFVec2f</b> .....    | <b>9-40</b> |           |          |  |
| Advise                           | Clone       | Get1Value | GetSize  |  |
| GetType                          | GetValue    | ToString  | Unadvise |  |
| <b>VrmlConstMFVec3f</b> .....    | <b>9-42</b> |           |          |  |
| Advise                           | Clone       | Get1Value | GetSize  |  |
| GetType                          | GetValue    | ToString  | Unadvise |  |
| <b>VrmlConstSFBool</b> .....     | <b>9-44</b> |           |          |  |
| Advise                           | Clone       | GetType   | GetValue |  |
| ToString                         | Unadvise    |           |          |  |
| <b>VrmlConstSFColor</b> .....    | <b>9-46</b> |           |          |  |
| Advise                           | Clone       | GetType   | GetRed   |  |
| GetGreen                         | GetBlue     | GetValue  | ToString |  |
| Unadvise                         |             |           |          |  |
| <b>VrmlConstSFFloat</b> .....    | <b>9-48</b> |           |          |  |
| Advise                           | Clone       | GetType   | GetValue |  |
| ToString                         | Unadvise    |           |          |  |

|                                  |             |               |           |  |
|----------------------------------|-------------|---------------|-----------|--|
| <b>VrmlConstSFImage</b> .....    | <b>9-50</b> |               |           |  |
| Advise                           | Clone       | GetComponents | GetHeight |  |
| GetPixels                        | GetType     | GetWidth      | ToString  |  |
| Unadvise                         |             |               |           |  |
| <b>VrmlConstSFInt32</b> .....    | <b>9-52</b> |               |           |  |
| Advise                           | Clone       | GetType       | GetValue  |  |
| ToString                         | Unadvise    |               |           |  |
| <b>VrmlConstSFNode</b> .....     | <b>9-53</b> |               |           |  |
| Advise                           | Clone       | GetType       | GetValue  |  |
| ToString                         | Unadvise    |               |           |  |
| <b>VrmlConstSFRotation</b> ..... | <b>9-55</b> |               |           |  |
| Advise                           | Clone       | GetAngle      | GetValue  |  |
| GetX                             | GetY        | GetZ          | ToString  |  |
| Unadvise                         |             |               |           |  |
| <b>VrmlConstSFString</b> .....   | <b>9-57</b> |               |           |  |
| Advise                           | Clone       | GetType       | GetValue  |  |
| ToString                         | Unadvise    |               |           |  |
| <b>VrmlConstSFTime</b> .....     | <b>9-58</b> |               |           |  |
| Advise                           | Clone       | GetType       | GetValue  |  |
| ToString                         | Unadvise    |               |           |  |
| <b>VrmlConstSFVec2f</b> .....    | <b>9-60</b> |               |           |  |
| Advise                           | Clone       | GetType       | GetValue  |  |
| GetX                             | GetY        | ToString      | Unadvise  |  |
| <b>VrmlConstSFVec3f</b> .....    | <b>9-62</b> |               |           |  |
| Advise                           | Clone       | GetType       | GetValue  |  |
| GetX                             | GetY        | GetZ          | ToString  |  |
| Unadvise                         |             |               |           |  |
| <b>VrmlMFColor</b> .....         | <b>9-64</b> |               |           |  |
| AddValue                         | Advise      | Clear         | Clone     |  |
| Delete                           | GetIValue   | GetSize       | GetType   |  |
| GetValue                         | InsertValue | SetIValue     | SetValue  |  |
| ToString                         | Unadvise    |               |           |  |



|                             |             |                    |          |  |
|-----------------------------|-------------|--------------------|----------|--|
| <b>VrmlMFfloat</b> .....    | <b>9-67</b> |                    |          |  |
| AddValue                    | Advise      | Clear              | Clone    |  |
| Delete                      | Get1Value   | GetSize            | GetType  |  |
| GetValue                    | InsertValue | Set1Value          | SetValue |  |
| ToString                    | Unadvise    |                    |          |  |
| <b>VrmlMField</b> .....     | <b>9-70</b> |                    |          |  |
| Advise                      | Clear       | Clone              | Delete   |  |
| GetSize                     | GetType     | ToString           | Unadvise |  |
| <b>VrmlMFInt32</b> .....    | <b>9-72</b> |                    |          |  |
| AddValue                    | Advise      | Clear              | Clone    |  |
| Delete                      | Get1Value   | GetSize            | GetType  |  |
| GetValue                    | InsertValue | Set1Value          | SetValue |  |
| ToString                    | Unadvise    |                    |          |  |
| <b>VrmlMFNode</b> .....     | <b>9-74</b> |                    |          |  |
| AddValue                    | Advise      | Clear              | Clone    |  |
| Delete                      | Get1Value   | GetSize            | GetType  |  |
| GetValue                    | InsertValue | Set1Value          | SetValue |  |
| SetValueFromConstMFNode     |             | SetValueFromMFNode |          |  |
| ToString                    | Unadvise    |                    |          |  |
| <b>VrmlMFRotation</b> ..... | <b>9-78</b> |                    |          |  |
| AddValue                    | Advise      | Clear              | Clone    |  |
| Delete                      | Get1Value   | GetSize            | GetType  |  |
| GetValue                    | InsertValue | Set1Value          | SetValue |  |
| ToString                    | Unadvise    |                    |          |  |
| <b>VrmlMFString</b> .....   | <b>9-81</b> |                    |          |  |
| AddValue                    | Advise      | Clear              | Clone    |  |
| Delete                      | Get1Value   | GetSize            | GetType  |  |
| GetValue                    | InsertValue | Set1Value          | SetValue |  |
| ToString                    | Unadvise    |                    |          |  |
| <b>VrmlMFTime</b> .....     | <b>9-84</b> |                    |          |  |
| AddValue                    | Advise      | Clear              | Clone    |  |
| Delete                      | Get1Value   | GetSize            | GetType  |  |
| GetValue                    | InsertValue | Set1Value          | SetValue |  |
| ToString                    | Unadvise    |                    |          |  |

|                          |              |               |           |  |
|--------------------------|--------------|---------------|-----------|--|
| <b>VrmlMFVec2f</b> ..... | <b>9-87</b>  |               |           |  |
| AddValue                 | Advise       | Clear         | Clone     |  |
| Delete                   | Get1Value    | GetSize       | GetType   |  |
| GetValue                 | InsertValue  | Set1Value     | SetValue  |  |
| ToString                 | Unadvise     |               |           |  |
| <b>VrmlMFVec3f</b> ..... | <b>9-90</b>  |               |           |  |
| AddValue                 | Advise       | Clear         | Clone     |  |
| Delete                   | Get1Value    | GetSize       | GetType   |  |
| GetValue                 | InsertValue  | Set1Value     | SetValue  |  |
| ToString                 | Unadvise     |               |           |  |
| <b>VrmlSFBool</b> .....  | <b>9-93</b>  |               |           |  |
| Advise                   | Clone        | GetType       | GetValue  |  |
| SetValue                 | ToString     | Unadvise      |           |  |
| <b>VrmlSFColor</b> ..... | <b>9-95</b>  |               |           |  |
| Advise                   | Clone        | GetBlue       | GetGreen  |  |
| GetRed                   | GetType      | GetValue      | SetValue  |  |
| ToString                 | Unadvise     |               |           |  |
| <b>VrmlSFFloat</b> ..... | <b>9-97</b>  |               |           |  |
| Advise                   | Clone        | GetType       | GetValue  |  |
| SetValue                 | ToString     | Unadvise      |           |  |
| <b>VrmlSFImage</b> ..... | <b>9-98</b>  |               |           |  |
| Advise                   | Clone        | GetComponents | GetHeight |  |
| GetPixels                | GetType      | GetWidth      | SetValue  |  |
| ToString                 | Unadvise     |               |           |  |
| <b>VrmlSFInt32</b> ..... | <b>9-101</b> |               |           |  |
| Advise                   | Clone        | GetType       | GetValue  |  |
| SetValue                 | ToString     | Unadvise      |           |  |
| <b>VrmlSFNode</b> .....  | <b>9-102</b> |               |           |  |
| Advise                   | Clone        | GetType       | GetValue  |  |
| SetValue                 | ToString     | Unadvise      |           |  |

|                             |              |          |          |  |
|-----------------------------|--------------|----------|----------|--|
| <b>VrmISFRotation</b> ..... | <b>9-104</b> |          |          |  |
| Advise                      | Clone        | GetAngle | GetType  |  |
| GetValue                    | GetX         | GetY     | GetZ     |  |
| SetValue                    | ToString     | Unadvise |          |  |
| <b>VrmISFString</b> .....   | <b>9-106</b> |          |          |  |
| Advise                      | Clone        | GetType  | GetValue |  |
| SetValue                    | ToString     | Unadvise |          |  |
| <b>VrmISFTime</b> .....     | <b>9-108</b> |          |          |  |
| Advise                      | Clone        | GetType  | GetValue |  |
| SetValue                    | ToString     | Unadvise |          |  |
| <b>VrmISFVec2f</b> .....    | <b>9-110</b> |          |          |  |
| Advise                      | Clone        | GetType  | GetValue |  |
| GetX                        | GetY         | SetValue | ToString |  |
| Unadvise                    |              |          |          |  |
| <b>VrmISFVec3f</b> .....    | <b>9-112</b> |          |          |  |
| Advise                      | Clone        | GetType  | GetValue |  |
| GetX                        | GetY         | GetZ     | SetValue |  |
| ToString                    | Unadvise     |          |          |  |

**VrmlObjectFactory Interface .....9-114**

|                           |                           |
|---------------------------|---------------------------|
| CreateVrmlConstMFColor    | CreateVrmlConstMFFloat    |
| CreateVrmlConstMFInt32    | CreateVrmlConstMFNode     |
| CreateVrmlConstMFRotation | CreateVrmlConstMFString   |
| CreateVrmlConstMFTime     | CreateVrmlConstMFVec2f    |
| CreateVrmlConstMFVec3f    | CreateVrmlConstSFBool     |
| CreateVrmlConstSFColor    | CreateVrmlConstSFFloat    |
| CreateVrmlConstSFImage    | CreateVrmlConstSFInt32    |
| CreateVrmlConstSFNode     | CreateVrmlConstSFRotation |
| CreateVrmlConstSFString   | CreateVrmlConstSFTime     |
| CreateVrmlConstSFVec2f    | CreateVrmlConstSFVec3f    |
| CreateVrmlMFColor         | CreateVrmlMFFloat         |
| CreateVrmlMFInt32         | CreateVrmlMFNode          |
| CreateVrmlMFRotation      | CreateVrmlMFString        |
| CreateVrmlMFTime          | CreateVrmlMFVec2f         |
| CreateVrmlMFVec3f         | CreateVrmlSFBool          |
| CreateVrmlSFColor         | CreateVrmlSFFloat         |
| CreateVrmlSFImage         | CreateVrmlSFInt32         |
| CreateVrmlSFNode          | CreateVrmlSFRotation      |
| CreateVrmlSFString        | CreateVrmlSFTime          |
| CreateVrmlSFVec2f         | CreateVrmlSFVec3f         |

**VrmlScriptImplementation Interface .....9-124**

|                 |            |
|-----------------|------------|
| EventsProcessed | Initialize |
| ProcessEvent    | Shutdown   |

## WorldView External Scripting Objects' Structure

This chart is provided for your convenience, and describes the structure of the Objects available in WorldView's EAI. Objects are arranged alphabetically throughout the rest of the chapter, for your convenience.

```
automation
|
+- VrmIBaseNode
|   +- VrmINode
|   +- VrmIScriptNode
|
+- VrmIBrowser
+- VrmIEvent
+- VrmIEventOutObserver
+- VrmIObjectFactory
+- VrmIScriptImplementation
|
+- VrmIField
|   +- VrmISFBool
|   +- VrmISFColor
|   +- VrmISFFloat
|   +- VrmISFImage
|   +- VrmISFInt32
|   +- VrmISFNode
|   +- VrmISFRotation
|   +- VrmISFString
|   +- VrmISFTime
|   +- VrmISFVec2f
|   +- VrmISFVec3f
|   |
```

## ■ WorldView for Developers Objects

### WorldView External Scripting Objects' Structure

---

```
|
|
+- Vrm1Field (continued)
|   +- Vrm1ConstField
|       +- Vrm1ConstSFBool
|       +- Vrm1ConstSFColor
|       +- Vrm1ConstSFfloat
|       +- Vrm1ConstSFImage
|       +- Vrm1ConstSFInt32
|       +- Vrm1ConstSFNode
|       +- Vrm1ConstSFRotation
|       +- Vrm1ConstSFString
|       +- Vrm1ConstSFTime
|       +- Vrm1ConstSFVec2f
|       +- Vrm1ConstSFVec3f
|
|       +- Vrm1ConstMField
|           +- Vrm1ConstMFCOLOR
|           +- Vrm1ConstMFFloat
|           +- Vrm1ConstMFInt32
|           +- Vrm1ConstMFNode
|           +- Vrm1ConstMFRotation
|           +- Vrm1ConstMFString
|           +- Vrm1ConstMFTIME
|           +- Vrm1ConstMFVec2f
|           +- Vrm1ConstMFVec3f
|
+- Vrm1MField
|   +- Vrm1MFCOLOR
|   +- Vrm1MFFloat
|   +- Vrm1MFInt32
|   +- Vrm1MFNode
|   +- Vrm1MFRotation
|   +- Vrm1MFString
|   +- Vrm1MFTIME
|   +- Vrm1MFVec2f
|   +- Vrm1MFVec3f
```

## VrmlBaseNode Objects

The WorldView COM API distinguishes between “regular” nodes and Script nodes. Script nodes differ from regular nodes in several ways: they cannot have exposedFields, their fields can be read and written, and their eventOuts can be written to. Due to these differences, the WorldView COM API provides different interfaces for Script nodes and non-Script nodes. Script nodes are represented by the IVrmlScriptNode interface, and non-Script nodes by the IVrmlNode interface.

However, Script nodes and non-Script nodes have some properties in common. For this reason, the IVrmlScriptNode and IVrmlNode interfaces both inherit from a common superinterface: IVrmlBaseNode.

This distinction between Script and non-Script nodes is also made by the VRML Java Scripting Reference. See the VRML 97 specification Annex B.

## VrmlBaseNode

Base node for IVrmlNode and IVrmlScriptNode.

### IVrmlBaseNode Interface

#### GetBrowser

**Prototype** `GetBrowser([out, retval] IVrmlBrowser* browser);`

Returns the IVrmlBrowser interface for the VRML browser which contains the node. The IVrmlBrowser interface can be used to inquire about browser statistics such as frame rate, and to add and delete routes. See *IVrmlBrowser Interface* for more information.

#### GetType

**Prototype** `GetType([out, retval] BSTR *nodetype);`

Returns the type of the node as a string.

#### ToString

**Prototype** `ToString([out, retval] BSTR* string);`

Returns a text string representation of the node. The result is a legal VRML definition for the node.



## VrmlNode

A node in the VRML scene.

### IVrmlNode Interface

#### GetBrowser

**Prototype** `GetBrowser([out, retval] IVrmlBrowser* browser);`

Returns the `IVrmlBrowser` interface for the VRML browser which contains the node. The `IVrmlBrowser` interface can be used to inquire about browser statistics such as frame rate, to add/delete routes, etc. See *IVrmlBrowser Interface* for more information.

#### GetEventIn

**Prototype** `GetEventIn([in] BSTR eventName,  
[out, retval] IVrmlField* field);`

Returns an `IVrmlField` interface referencing the node's eventIn whose name is `eventName`. The return value can be converted to the appropriate subinterface of the `IVrmlField` interface, such as `IVrmlSFBool`.

The returned interface is write-only.

#### GetEventOut

**Prototype** `GetEventOut([in] BSTR eventName,  
[out, retval] IVrmlConstField* field);`

Returns an `IVrmlField` interface referencing the node's eventOut whose name is `eventName`. The return value can be converted to the appropriate subinterface of the `IVrmlConstField` interface, such as `IVrmlConstSFBool`.

The returned interface is read-only.

### **GetExposedField**

**Prototype** `GetExposedField([in] BSTR fieldName,  
[out, retval] IVrmlField* field);`

Returns an IVrmlField interface referencing the node's exposedField whose name is fieldName. The return value can be queried to the appropriate subinterface of the IVrmlField interface, such as IVrmlSFBool.

The returned interface may be read and written.

### **GetType**

**Prototype** `String GetType();`

Returns the type of the node as a string.

### **ToString**

**Prototype** `ToString([out, retval] BSTR* string);`

Returns a text string representation of the node. The result is a legal VRML definition for the node.

## VrmlScriptNode

This node is similar to a VrmlNode. The only difference between them is that VrmlNode is a node in the scene, and VrmlScriptNode is a script that can control and/or respond to nodes within the scene.

The IVrmlScriptNode interface permits the script OCX to inquire about the eventIns, eventOuts, fields and containing VRML browser of the corresponding Script node. The IVrmlScriptNode interface supports the IVrmlBaseNode interface.

Note that the Java equivalents of the methods of IVrmlScriptNode reside in the Script base class. Thus, some methods of Script are implemented by the VRML browser, and some are overridden by user-created descendant classes of Script. In COM, this is not possible, so the functionality provided by WorldView resides in the IVrmlScriptNode interface, and the functionality written by the user resides in the IVrmlScriptImplementation interface.

## IVrmlScriptNode Interface

### GetBrowser

**Prototype** `GetBrowser([out, retval] IVrmlBrowser* browser);`

Returns the IVrmlBrowser interface for the VRML browser which contains the corresponding Script node. The IVrmlBrowser interface can be used to inquire about browser statistics such as frame rate, and to add and delete routes, etc. See [IVrmlBrowser Interface](#) for more information.

### GetEventIn

**Prototype** `GetEventIn([in] BSTR eventInName,  
[out, retval] IVrmlField* field);`

Returns an IVrmlField interface referencing the Script node's eventIn whose name is eventInName. The return value can be converted to the appropriate subinterface of the IVrmlField interface, for example IVrmlSFBool.

The returned interface is write-only.

#### **GetEventOut**

**Prototype** `GetEventOut([in] BSTR eventOutName,  
[out, retval] IVrmlField* field);`

Returns an `IVrmlField` interface referencing the Script node's eventOut whose name is `eventOutName`. The return value can be converted to the appropriate subinterface of the `IVrmlField` interface, for example `IVrmlSFBool`.

The returned interface may be read and written.

#### **GetField**

**Prototype** `GetField([in] BSTR fieldName,  
[out, retval] IVrmlField* field);`

Returns an `IVrmlField` interface referencing the Script node's field whose name is `fieldName`. The return value can be queried to the appropriate subinterface of the `IVrmlField` interface, for example `IVrmlSFBool`.

The returned interface may be read and written.

#### **GetType**

**Prototype** `GetType([out, retval] BSTR *type);`

Returns the type of the corresponding Script node. The return value will always be `Script`.

#### **ToString**

**Prototype** `ToString([out, retval] BSTR* string);`

Returns a text string representation of the Script node. The result will be a legal VRML definition for the corresponding Script node.

## VrmlBrowser Object

The VrmlBrowser object provides access to and control over the WorldView browser, and provides a means for creating new VRML data objects as needed.

### IVrmlBrowser Interface

#### AddRoute

**Prototype** `AddRoute([in]IVrmlBaseNode*fromNode,[in]BSTRfromEventOut,[in] IVrmlBaseNode* toNode, [in] BSTR toEventIn);`

Adds a route between the specified eventOut and eventIn of the given nodes.

#### CreateVrmlFromString

**Prototype** `CreateVrmlFromString([in] BSTR vrml,[out,retval] IVrmlConstMFNode** constmfNode);`

Parses a string into a VRML scene, and returns the nodes for the resulting scene.

#### CreateVrmlFromURL

**Prototype** `CreateVrmlFromURL([in] IVrmlMFString* url,[in]IVrmlBaseNode* baseNode, [in] BSTR event);`

Gets the VRML scene from the given URL or URLs. The browser will try the first URL, then the second, then the rest in order, until one loads or all URLs have been tried. When the scene is downloaded and parsed, it is sent to the IVrmlBaseNode node's IVrmlMFNode eventIn named by the event argument.

#### DeleteRoute

**Prototype** `DeleteRoute([in]VrmlBaseNode*fromNode[in]BSTRfromEventOut,[in] IVrmlBaseNode* toNode, [in] BSTR toEventIn);`

Deletes a route between the specified eventOut and eventIn of the given nodes.

#### **GetCurrentFrameRate**

**Prototype** `GetCurrentFrameRate([out, retval] float* rate);`

Gets the current frame rate of the browser.

#### **GetCurrentSpeed**

**Prototype** `GetCurrentSpeed([out, retval] float* speed);`

Gets the current velocity of the bound viewpoint in meters/second.

#### **GetName**

**Prototype** `GetName([out, retval] BSTR* name);`

Returns the name of the VRML Browser.

#### **GetNode**

**Prototype** `GetNode([in] BSTR name, [out,retval] IVrmlNode** node);`

Gets a DEF'd node by name. Nodes given names in the root scenegraph are available to this method. DEF'd nodes in Inlines, or those returned by `CreateVrmlFromString` or `CreateVrmlFromURL` are not available.

#### **GetVersion**

**Prototype** `GetVersion([out, retval] BSTR* version);`

Returns the version of the VRML browser.

#### **GetWorldURL**

**Prototype** `GetWorldURL([out, retval] BSTR* url);`

Gets the URL for the root of the current world, or an empty string if the URL is not available.

## LoadURL

**Prototype** `LoadURL([in] IVrmlMFString* url,  
[in] IVrmlMFString* parameter);`

Launches the specified VRML file. It requires 2 string parameters: the first is the URL of the VRML file to be loaded; the 2nd is a parameter which may be used in the same way as the Anchor node's parameter field (for example: parameter="target=\_top").

---

**Note** • A parameter such as the one used in this example will work only in a container that supports frames.

---

## ReplaceWorld

**Prototype** `ReplaceWorld([in] IVrmlMFNode* worldMFNode);`

Replaces the current world with the passed nodes.

## SetDescription

**Prototype** `SetDescription([in] BSTR description);`

Sets the description of the current world in a browser specific manner. (In WorldView, it prints the description to the status bar in Internet Explorer or Netscape Navigator.) To clear the description, pass an empty string as an argument.

## VrmlEvent Object

An eventIn or eventOut associated with a node, the VrmlEvent object provides information on communications data passed between nodes within a VRML scene.

The ProcessEvent method of IVrmlScriptImplementation is passed an IVrmlEvent interface which describes the event the Script node has just received. The IVrmlEvent interface is analogous to the Event class in the VRML Java Scripting Reference, Section B.9.2.1.

## IVrmlEvent Interface

### Clone

**Prototype** Clone([out, retval] IVrmlEvent\*\* event);

Duplicates this event into a new COM object and returns its VrmlEvent interface. This may be useful for saving an event as a variable in your script for later use.

### GetName

**Prototype** GetName([out, retval] BSTR\* name);

Returns the name of the eventIn at which the event arrived. This permits a Script with multiple eventIns to distinguish between events that arrived at different eventIns.

### GetTimeStamp

**Prototype** GetTimeStamp([out, retval] double\* time);

Returns the time the event arrived.



### GetValue

**Prototype** `GetValue([out, retval] IVrmlConstField** constfield);`

Returns an IVrmlConstField interface containing the value of the event. The IVrmlConstField interface can be queried to the interface corresponding to the eventIn's type, for instance, IVrmlConstSFBool, to get specific information about the event value.

### ToString

**Prototype** `ToString([out, retval] BSTR* string);`

Returns a text string representation of the event. The string will display the contents of the event's value field in a human-readable format. This may be useful for debugging purposes.

## VrmlEventOutObserver Object

An interface that must be implemented by the application, and is notified when an event on the field, for which it was registered or "advised," occurs.

## IVrmlEventOutObserver Interface

### Callback

**Prototype** `Callback([in] IVrmlConstField* value,  
[in] double timeStamp, [in] VARIANT* userData);`

The callback routine to enable the geometry to notify an external program when certain events occur. The field parameter is the node's field or exposed field that is being tracked. The timeStamp is a receive variable that is set when the event occurs, and userData is the data defined in the VrmlField Advise call used to register this callback.

## **VrmlField Objects**

The family of VrmlField COM objects represent all of the field data types available in VRML. There is a COM interface in the COM API corresponding to each of the field data types, and an additional COM interface which corresponds to the same data type but cannot be modified, called Const for constant. All of these COM interfaces inherit the IVrmlField interface, which provides functionality common to all field objects.

VrmlField objects (VrmlField and objects derived from it) define the data types used by a VRML scene. These objects provide access to and manipulation of the information within these data types.

The Const prefix indicates a read only field, usually associated with an eventOut. These classes support the getValue() method. Some classes support additional convenience methods to get value(s) from the object.

SF means single values. MF means multiple values. An MFField is an array of the base field type.

The InsertValue method, which applies to all the MF type classes, may only be used if the MFNode or MFField into which you wish to insert values is not empty: that is, if the Node or Field already contains at least one other object. If you insert an object into an empty array, there will be no error posted, but there will be no resultant action. You may not insert into a blank array.

An event is a message sent from one node to another as defined by a route. Events signal external stimuli, changes to field values, and interactions between nodes. An event consists of a time stamp and a field value.

An eventIn is a logical receptor attached to a node which receives events.

An eventOut is a logical output terminal attached to a node from which events are sent. The eventOut also stores the event most recently sent.

An eventOutObserver is an interface which must be implemented by the application, and is notified when an event on the field, for which it was registered or "advised," occurs.

## VrmlField

Base class for all fields (single/multi-valued, const/non-const).

### IVrmlField Interface

#### Advise

**Prototype** `Advise([in] IVrmlEventOutObserver* eventOutObserver, [in] VARIANT* userData);`

Permits a program using the COM API to receive notification when the value of this field changes. The parameter `eventOutObserver` must contain the `IVrmlEventOutObserver` interface of a COM object implemented by the user program. When the field's value changes, the `Callback` method of the `IVrmlEventOutObserver` interface will be invoked and passed the new value, the time at which the change occurred, and the `VARIANT` specified in the `userData` parameter.

#### Clone

**Prototype** `Clone([out, retval] IVrmlField** field);`

Creates a duplicate copy of the COM object and returns the `IVrmlField` interface of the newly created object.

If the field is an `eventIn`, `eventOut`, `field` or `exposedField`, the new COM object is identical to the old one, in that it points to the same `eventIn`, `eventOut`, `field` or `exposedField`. However if the field is "unattached," that is, created through the `IVrmlObjectFactory` interface, then a new, unattached field containing a copy of the field's data is returned.

### GetType

**Prototype** `GetType([out, retval] enumFieldType* type);`

Returns the type of the field. The type is returned as an integer; using the numeric values assigned to each type are taken in the Java External Authoring Interface (EAI) proposal.

| Field   | type value |
|---------|------------|
| SFBOOL  | 1          |
| SFIMAGE | 2          |
| SFTIME  | 3          |
| SFCOLOR | 4          |
| MFCOLOR | 5          |
| SFFLOAT | 6          |
| MFFLOAT | 7          |
| SFINT32 | 8          |
| MFINT32 | 9          |
| SFNODE  | 10         |

| Field      | type value |
|------------|------------|
| MFNODE     | 11         |
| SFROTATION | 12         |
| MFROTATION | 13         |
| SFSTRING   | 14         |
| MFSTRING   | 15         |
| SFVEC2F    | 16         |
| MFVEC2F    | 17         |
| SFVEC3F    | 18         |
| MFVEC3F    | 19         |
| MFTIME     | 20         |

### ToString

**Prototype** `ToString([out, retval] BSTR* s);`

Returns a text string representation of the field. The result will be a legal VRML definition for the field.

### Unadvise

**Prototype** `Unadvise([in] IVrmlEventOutObserver* eventOutObserver);`

Stops notifying the specified IVrmlEventOutObserver of changes in this field's value.

## VrmlConstField

Defines all VRML data types. These are read-only versions of the field type, and any object defined as VrmlConstField cannot be updated. These are usually the datatypes of eventOuts.

The IVrmlConstField interface is the base interface for all Const field types, such as IVrmlConstSFBool and IVrmlConstMFNode. It inherits the methods of the IVrmlField interface. It defines no new methods of its own.

To determine if an IVrmlField interface field is a Const field, query to the IVrmlConstField interface. If the query succeeds, the field must be Const.

## IVrmlConstField Interface

### Advise

**Prototype** `Advise([in] IVrmlEventOutObserver* eventOutObserver, [in] VARIANT* userData);`

Permits a program using the COM API to receive notification when the value of this field changes.

### Clone

**Prototype** `Clone([out, retval] IVrmlField** field);`

Creates a duplicate copy of the COM object and returns the IVrmlField interface of the newly created object.

### GetType

**Prototype** `GetType([out, retval] enumFieldType* type);`

Returns the type of the field.

### Unadvise

**Prototype** `Unadvise([in] IVrmlEventOutObserver* eventOutObserver);`

Stops notifying the specified IVrmlEventOutObserver of changes in this field's value.

# VrmlConstMFCOLOR

The VrmlConstMFCOLOR COM object represents a read-only MFCOLOR field in the VRML scene. An MFCOLOR field specifies zero or more RGB (red-green-blue) color triples. Each color triple consists of three floating point numbers in the range 0.0 to 1.0.

The VrmlConstMFCOLOR object supports the IVrmlConstMFCOLOR, IVrmlConstMField, IVrmlConstField and IVrmlField interfaces.

## IVrmlConstMFCOLOR Interface

### Advise

**Prototype** Advise([in] IVrmlEventOutObserver\* eventOutObserver, [in] VARIANT\* userData);

Starts notifying the specified IVrmlEventOutObserver interface of changes in this field's value. Note that although this field is read-only to COM API, it may be changed by other events in the VRML scene. This method is inherited from the *IVrmlField Interface*.

### Clone

**Prototype** Clone([out, retval] IVrmlField\*\* field);

Creates a duplicate copy of the COM object. This method is inherited from the *IVrmlField Interface*.

### Get1Value

**Prototype** Get1Value([in] int index, [out] float \*r, [out] float \*g, [out] float \*b);

Retrieves the index'th color triple and stores the red, green and blue values of the triple in the variables r, g, and b, respectively.

### GetSize

**Prototype** GetSize([out, retval] int\* size);

Returns the number of RGB color triples in the field. This method is inherited from the *IVrmlConstMField Interface*.

## GetType

**Prototype** GetType([out, retval] enumFieldType\* type);

Returns the type of this field, which will be MFCOLOR (5). This method is inherited from the *IVrmlField Interface*.

## GetValue

**Prototype** GetValue([out] float\* values);

Copies all color triples in the field into the values array. The first triple will be copied into values[0], values[1], and values[2] in R, G, B order; the second triple into values[3], values[4], values[5], and so on.

The values array must be large enough to accommodate the result or a crash may occur. The number of elements required is 3 times the result of the GetSize method.

## ToString

**Prototype** ToString([out, retval] BSTR\* s);

Returns a text string representation of the field. The result will be a legal VRML definition for the field.

## Unadvise

**Prototype** Unadvise([in] IVrmlEventOutObserver\* eventOutObserver);

Stops notifying the specified IVrmlEventOutObserver of changes in this field's value. This method is inherited from the *IVrmlMField Interface*.

## VrmlConstMFFloat

The VrmlConstMFFloat COM object represents a read-only MFFloat field in the VRML scene. An MFFloat field specifies zero or more single-precision floating point numbers.

The VrmlConstMFFloat object supports the IVrmlConstMFFloat, IVrmlConstMField, IVrmlConstField and IVrmlField interfaces.

### IVrmlConstMFFloat Interface

#### Advise

**Prototype** Advise([in] IVrmlEventOutObserver\* eventOutObserver, [in] VARIANT\* userData);

Starts notifying the specified IVrmlEventOutObserver interface of changes in this field's value. Note that although this field is read-only to COM API, it may be changed by other events in the VRML scene. This method is inherited from the *IVrmlField Interface*.

#### Clone

**Prototype** Clone([out, retval] IVrmlField\*\* field);

Creates a duplicate copy of the COM object. This method is inherited from the *IVrmlMField Interface*.

#### Get1Value

**Prototype** Get1Value([in] int index, [out, retval] float \*value);

Retrieves the index'th floating point number from the field and stores it in the value parameter.

#### GetSize

**Prototype** GetSize([out, retval] int\* size);

Returns the number of floating point numbers in the field. This method is inherited from the *IVrmlConstField Interface*.



## GetType

**Prototype** GetType([out, retval] enumFieldType\* type);

Returns the type of this field, which will be MFFLOAT (7). This method is inherited from the *IVrmlField Interface*.

## GetValue

**Prototype** GetValue([out] float\* values);

Copies all of the floating point numbers in the field into the values array.

The values array must be large enough to accommodate the result or a crash may occur. The number of elements required is returned by the GetSize method.

## ToString

**Prototype** ToString([out, retval] BSTR\* s);

Returns a text string representation of the field. The result will be a legal VRML definition for the field.

## Unadvise

**Prototype** Unadvise([in] IVrmlEventOutObserver\* eventOutObserver);

Stops notifying the specified IVrmlEventOutObserver of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

# VrmlConstMField

The VrmlConstMField family of objects represent multiple value, read-only fields in the VRML scene, for example VrmlConstMFColor. All objects in the VrmlConstMField family support the IVrmlConstMField, IVrmlConstField and IVrmlField interfaces.

### IVrmlConstMField Interface

#### Advise

**Prototype** Advise([in] IVrmlEventOutObserver\* eventOutObserver, [in] VARIANT\* userData);

Starts notifying the specified IVrmlEventOutObserver interface of changes in this field's value. Note that although this field is read-only to COM API, it may be changed by other events in the VRML scene. This method is inherited from the *IVrmlField Interface*.

#### Clone

**Prototype** Clone([out, retval] IVrmlField\*\* field);

Creates a duplicate copy of the COM object. This method is inherited from the *IVrmlField Interface*.

#### GetSize

**Prototype** GetSize([out, retval] int\* size);

Returns the number of elements in the field. The definition of an element will vary depending on the type of the field. See the description of this method for the specific field type.

#### GetType

**Prototype** GetType([out, retval] enumFieldType\* type);

Returns the type of this field. This method is inherited from the *IVrmlField Interface*.

#### ToString

**Prototype** ToString([out, retval] BSTR\* s);

Returns a text string representation of the field. The result will be a legal VRML definition for the field.

#### Unadvise

**Prototype** Unadvise([in] IVrmlEventOutObserver\* eventOutObserver);

Stops notifying the specified IVrmlEventOutObserver of changes in this field's value. Inherited from the *IVrmlField Interface*.

## VrmlConstMfInt32

The VrmlConstMfInt32 COM object represents a read-only MfInt32 field in the VRML scene. An MfInt32 field specifies zero or more 32-bit integers.

The VrmlConstMfInt32 object supports the IVrmlConstMfInt32, IVrmlConstMField, IVrmlConstField and IVrmlField interfaces.

## IVrmlConstMfInt32 Interface

### Advise

**Prototype** Advise([in] IVrmlEventOutObserver\* eventOutObserver, [in] VARIANT\* userData);

Starts notifying the specified IVrmlEventOutObserver interface of changes in this field's value. Note that although this field is read-only to COM API, it may be changed by other events in the VRML scene. This method is inherited from the *IVrmlField Interface*.

### Clone

**Prototype** Clone([out, retval] IVrmlField\*\* field);

Creates a duplicate copy of the COM object. This method is inherited from the *IVrmlField Interface*.

### Get1Value

**Prototype** Get1Value([in] int index, [out, retval] int \*value);

Retrieves the index'th integer from the field and stores it in the value parameter.

### GetSize

**Prototype** GetSize([out, retval] int\* size);

Returns the number of integers in the field. This method is inherited from the *IVrmlConstMField Interface*.

#### **GetType**

**Prototype** `GetType([out, retval] enumFieldType* type);`

Returns the type of this field, which will be MFINT32 (9). This method is inherited from the *IVrmlField Interface*.

#### **GetValue**

**Prototype** `GetValue([out] int* values);`

Copies all of the integers in the field into the values array.

The values array must be large enough to accommodate the result or a crash may occur. The number of elements required is returned by the `GetSize` method.

#### **ToString**

**Prototype** `ToString([out, retval] BSTR* s);`

Returns a text string representation of the field. The result will be a legal VRML definition for the field.

#### **Unadvise**

**Prototype** `Unadvise([in] IVrmlEventOutObserver* eventOutObserver);`

Stops notifying the specified `IVrmlEventOutObserver` of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

## VrmlConstMFNode

The VrmlConstMFNode COM object represents a read-only MFNode field in the VRML scene. An MFNode field specifies zero or more VRML nodes.

The VrmlConstMFNode object supports the IVrmlConstMFNode, IVrmlConstMField, IVrmlConstField and IVrmlField interfaces.

## IVrmlConstMFNode Interface

### Advise

**Prototype** Advise([in] IVrmlEventOutObserver\* eventOutObserver, [in] VARIANT\* userData);

Starts notifying the specified IVrmlEventOutObserver interface of changes in this field's value. Note that although this field is read-only to COM API, it may be changed by other events in the VRML scene. This method is inherited from the *IVrmlField Interface*.

### Clone

**Prototype** Clone([out, retval] IVrmlField\*\* field);

Creates a duplicate copy of the COM object. This method is inherited from the *IVrmlField Interface*.

### Get1Value

**Prototype** Get1Value([in] int index, [out, retval] IVrmlBaseNode \*\*value);

Retrieves the index'th node from the field and stores it in the value parameter. The node is represented by an IVrmlBaseNode interface.

### GetSize

**Prototype** GetSize([out, retval] int\* size);

Returns the number of nodes in the field. This method is inherited from the *IVrmlConstMField Interface*.

#### **GetType**

**Prototype** `GetType([out, retval] enumFieldType* type);`

Returns the type of this field, which will be MFNODE (11). This method is inherited from the *IVrmlField Interface*.

#### **GetValue**

**Prototype** `GetValue([out] IVrmlBaseNode** values);`

Copies all of the nodes in the field into the values array. Each node is represented by an IVrmlBaseNode interface.

The values array must be large enough to accommodate the result or a crash may occur. The number of elements required is returned by the GetSize method.

#### **ToString**

**Prototype** `ToString([out, retval] BSTR* s);`

Returns a text string representation of the field. The result will be a legal VRML definition for the field.

#### **Unadvise**

**Prototype** `Unadvise([in] IVrmlEventOutObserver* eventOutObserver);`

Stops notifying the specified IVrmlEventOutObserver of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

## VrmlConstMFRotation

The VrmlConstMFRotation COM object represents a read-only MFRotation field in the VRML scene. An MFRotation field specifies zero or more arbitrary rotations. Each rotation is specified by four floating point values. The first three values specify a normalized rotation axis vector about which the rotation takes place. The fourth value specifies the amount of right-handed rotation about that axis in radians.

The VrmlConstMFRotation object supports the IVrmlConstMFRotation, IVrmlConstMField, IVrmlConstField and IVrmlField interfaces.

### IVrmlConstMFRotation Interface

#### Advise

**Prototype** Advise([in] IVrmlEventOutObserver\* eventOutObserver, [in] VARIANT\* userData);

Starts notifying the specified IVrmlEventOutObserver interface of changes in this field's value. Note that although this field is read-only to COM API, it may be changed by other events in the VRML scene. This method is inherited from the *IVrmlField Interface*.

#### Clone

**Prototype** Clone([out, retval] IVrmlField\*\* field);

Creates a duplicate copy of the COM object. This method is inherited from the *IVrmlField Interface*.

#### Get1Value

**Prototype** Get1Value([in] int index, [out] float \*x, [out] float \*y, [out] float \*z, [out] float \*angle);

Retrieves the index'th rotation. The rotation axis is stored in the variables x, y, and z, respectively, and the angle of rotation is stored in the variable angle.

#### GetSize

**Prototype**    GetSize([out, retval] int\* size);

Returns the number of rotations in the field. This method is inherited from the *IVrmlConstMField Interface*.

#### GetType

**Prototype**    GetType([out, retval] enumFieldType\* type);

Returns the type of this field, which will be MFROTATION (13). This method is inherited from the *IVrmlField Interface*.

#### GetValue

**Prototype**    GetValue([out] float\* values);

Copies all of the rotations in the field into the values array. The first rotation will be copied into values[0], values[1], values[2], and values[3] in the order X, Y, Z, angle; the second triple into values[4...7], and so on.

The values array must be large enough to accommodate the result or a crash may occur. The number of elements required is 4 times the result of the GetSize method.

#### ToString

**Prototype**    ToString([out, retval] BSTR\* s);

Returns a text string representation of the field. The result will be a legal VRML definition for the field.

#### Unadvise

**Prototype**    Unadvise([in] IVrmlEventOutObserver\*  
                  eventOutObserver);

Stops notifying the specified IVrmlEventOutObserver of changes in this field's value. This method is inherited from the *IVrmlField Interface*.



## VrmlConstMFString

The VrmlConstMFString COM object represents a read-only MFString field in the VRML scene. An MFString field specifies zero or more strings.

The VrmlConstMFString object supports the IVrmlConstMFString, IVrmlConstMField, IVrmlConstField and IVrmlField interfaces.

### IVrmlConstMFString Interface

#### Advise

**Prototype** Advise([in] IVrmlEventOutObserver\* eventOutObserver, [in] VARIANT\* userData);

Starts notifying the specified IVrmlEventOutObserver interface of changes in this field's value. Note that although this field is read-only to COM API, it may be changed by other events in the VRML scene. This method is inherited from the *IVrmlField Interface*.

#### Clone

**Prototype** Clone([out, retval] IVrmlField\*\* field);

Creates a duplicate copy of the COM object. This method is inherited from the *IVrmlField Interface*.

#### Get1Value

**Prototype** Get1Value([in] int index, [out, retval] BSTR\* value);

Retrieves the index'th string from the field and stores it in the value parameter.

#### GetSize

**Prototype** GetSize([out, retval] int\* size);

Returns the number of strings in the field. This method is inherited from the *IVrmlConstMField Interface*.

#### **GetType**

**Prototype:** GetType([out, retval] enumFieldType\* type);

Returns the type of this field, which will be MFSTRING (15). This method is inherited from the *IVrmlField Interface*.

#### **GetValue**

**Prototype** GetValue([out] BSTR\* values);

Copies all of the strings in the field into the values array.

The values array must be large enough to accommodate the result or a crash may occur. The number of elements required is returned by the GetSize method.

#### **ToString**

**Prototype** ToString([out, retval] BSTR\* s);

Returns a text string representation of the field. The result will be a legal VRML definition for the field.

#### **Unadvise**

**Prototype** Unadvise([in] IVrmlEventOutObserver\* eventOutObserver);

Stops notifying the specified IVrmlEventOutObserver of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

## VrmlConstMTime

The VrmlConstMTime COM object represents a read-only MTime field in the VRML scene. An MTime field specifies zero or more time values, which are double-precision floating point numbers.

The VrmlConstMTime object supports the IVrmlConstMTime, IVrmlConstMField, IVrmlConstField and IVrmlField interfaces.

## IVrmlConstMTime Interface

### Advise

**Prototype** Advise([in] IVrmlEventOutObserver\* eventOutObserver, [in] VARIANT\* userData);

Starts notifying the specified IVrmlEventOutObserver interface of changes in this field's value. Note that although this field is read-only to COM API, it may be changed by other events in the VRML scene. This method is inherited from the *IVrmlField Interface*.

### Clone

**Prototype** :Clone([out, retval] IVrmlField\*\* field);

Creates a duplicate copy of the COM object. This method is inherited from the *IVrmlField Interface*.

### Get1Value

**Prototype** Get1Value([in] int index, [out, retval] double\* Time);

Retrieves the index'th time value from the field and stores it in the value parameter.

### GetSize

**Prototype** GetSize([out, retval] int\* size);

Returns the number of time values in the field. This method is inherited from the *IVrmlConstMField Interface*.

#### GetType

**Prototype** GetType([out, retval] enumFieldType\* type);

Returns the type of this field, which will be MFTIME (20). This method is inherited from the *IVrmlField Interface*.

#### GetValue

**Prototype** GetValue([out] double\* values);

Copies all of the time values in the field into the values array.

The values array must be large enough to accommodate the result or a crash may occur. The number of elements required is returned by the GetSize method.

#### ToString

**Prototype** ToString([out, retval] BSTR\* s);

Returns a text string representation of the field. The result will be a legal VRML definition for the field.

#### Unadvise

**Prototype** Unadvise([in] IVrmlEventOutObserver\* eventOutObserver);

Stops notifying the specified IVrmlEventOutObserver of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

## VrmlConstMFVec2f

The VrmlConstMFVec2f COM object represents a read-only MFVec2f field in the VRML scene. An MFVec2f field specifies zero or more 2D vectors. Each vector is a pair of floating point values.

The VrmlConstMFVec2f object supports the IVrmlConstMFVec2f, IVrmlConstMField, IVrmlConstField and IVrmlField interfaces.

### IVrmlConstMFVec2f Interface

#### Advise

**Prototype** Advise([in] IVrmlEventOutObserver\* eventOutObserver, [in] VARIANT\* userData);

Starts notifying the specified IVrmlEventOutObserver interface of changes in this field's value. Note that although this field is read-only to COM API, it may be changed by other events in the VRML scene. This method is inherited from the *IVrmlField Interface*.

#### Clone

**Prototype** Clone([out, retval] IVrmlField\*\* field);

Creates a duplicate copy of the COM object. This method is inherited from the *IVrmlField Interface*.

#### Get1Value

**Prototype** Get1Value([in] int index, [out] float \*x, [out] float \*y);

Retrieves the index'th 2D vector and stores the X and Y components of the vector in the variables x and y, respectively.

#### GetSize

**Prototype** GetSize([out, retval] int\* size);

Returns the number of 2D vectors in the field. This method is inherited from the *IVrmlConstMField Interface*.

#### GetType

**Prototype** GetType([out, retval] enumFieldType\* type);

Returns the type of this field, which will be MFVEC2F (17). This method is inherited from the *IVrmlField Interface*.

#### GetValue

**Prototype** GetValue([out] float\* values);

Copies all of the 2D vectors in the field into the values array. The first 2D vector will be copied into values[0] and values[1] in X, Y order; the second 2D vector into values[2] and values[3], and so on.

The values array must be large enough to accommodate the result or a crash may occur. The number of elements required is 2 times the result of the GetSize method.

#### ToString

**Prototype** ToString([out, retval] BSTR\* s);

Returns a text string representation of the field. The result will be a legal VRML definition for the field.

#### Unadvise

**Prototype** Unadvise([in] IVrmlEventOutObserver\* eventOutObserver);

Stops notifying the specified IVrmlEventOutObserver of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

## VrmlConstMFVec3f

The VrmlConstMFVec3f COM object represents a read-only MFVec3f field in the VRML scene. An MFVec3f field specifies zero or more 3D vectors. Each vector consists of three floating point values.

The VrmlConstMFVec3f object supports the IVrmlConstMFVec3f, IVrmlConstMField, IVrmlConstField and IVrmlField interfaces.

### IVrmlConstMFVec3f Interface

#### Advise

**Prototype** Advise([in] IVrmlEventOutObserver\* eventOutObserver, [in] VARIANT\* userData);

Starts notifying the specified IVrmlEventOutObserver interface of changes in this field's value. Note that although this field is read-only to COM API, it may be changed by other events in the VRML scene. This method is inherited from the *IVrmlField Interface*.

#### Clone

**Prototype** Clone([out, retval] IVrmlField\*\* field);

Creates a duplicate copy of the COM object. This method is inherited from the *IVrmlField Interface*.

#### Get1Value

**Prototype** Get1Value([in] int index, [out] float \*x, [out] float \*y);

Retrieves the index'th 3D vector and stores the X, Y and Z components of the vector in the variables x, y and z, respectively.

#### GetSize

**Prototype** GetSize([out, retval] int\* size);

Returns the number of 3D vectors in the field. This method is inherited from the *IVrmlConstMField Interface*.

#### **GetType**

**Prototype** `GetType([out, retval] enumFieldType* type);`

Returns the type of this field, which will be MFVEC3F (19). This method is inherited from the *IVrmlField Interface*.

#### **GetValue**

**Prototype** `GetValue([out] float* values);`

Copies all of the 3D vectors in the field into the values array. The first 3D vector will be copied into values[0], values[1] and values[2] in X, Y, Z order; the second 3D vector into values[3...5], and so on.

The values array must be large enough to accommodate the result or a crash may occur. The number of elements required is 3 times the result of the `GetSize` method.

#### **ToString**

**Prototype** `ToString([out, retval] BSTR* s);`

Returns a text string representation of the field. The result will be a legal VRML definition for the field.

#### **Unadvise**

**Prototype** `Unadvise([in] IVrmlEventOutObserver* eventOutObserver);`

Stops notifying the specified `IVrmlEventOutObserver` of changes in this field's value. This method is inherited from the *IVrmlField Interface*.



## VrmlConstSFBool

The VrmlConstSFBool COM object represents a read-only SFBool field in the VRML scene. An SFBool field contains a single boolean value, either TRUE or FALSE.

The VrmlConstSFBool object supports the IVrmlConstSFBool, IVrmlConstField and IVrmlField interfaces.

## IVrmlConstSFBool Interface

### Advise

**Prototype** Advise([in] IVrmlEventOutObserver\* eventOutObserver, [in] VARIANT\* userData);

Starts notifying the specified IVrmlEventOutObserver interface of changes in this field's value. Note that although this field is read-only to COM API, it may be changed by other events in the VRML scene. This method is inherited from the *IVrmlField Interface*.

### Clone

**Prototype** Clone([out, retval] IVrmlField\*\* field);

Creates a duplicate copy of the COM object. This method is inherited from the *IVrmlField Interface*.

### GetType

**Prototype** GetType([out, retval] enumFieldType\* type);

Returns the type of this field, which will be SFBOOL (1). This method is inherited from the *IVrmlField Interface*.

### GetValue

**Prototype** GetValue([out, retval] boolean\* value);

Returns the value of the field.

#### **ToString**

**Prototype** ToString([out, retval] BSTR\* s);

Returns a text string representation of the field. The result will be a legal VRML definition for the field.

#### **Unadvise**

**Prototype** Unadvise([in] IVrmlEventOutObserver\* eventOutObserver);

Stops notifying the specified IVrmlEventOutObserver of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

## VrmlConstSFCOLOR

The VrmlConstSFCOLOR COM object represents a read-only SFCOLOR field in the VRML scene. An SFCOLOR field contains one RGB (red-green-blue) color triple, specified by three floating point values.

The VrmlConstSFCOLOR object supports the IVrmlConstSFCOLOR, IVrmlConstField and IVrmlField interfaces.

### IVrmlConstSFCOLOR Interface

#### Advise

**Prototype** Advise([in] IVrmlEventOutObserver\* eventOutObserver, [in] VARIANT\* userData);

Starts notifying the specified IVrmlEventOutObserver interface of changes in this field's value. Note that although this field is read-only to COM API, it may be changed by other events in the VRML scene. This method is inherited from the *IVrmlField Interface*.

#### Clone

**Prototype** Clone([out, retval] IVrmlField\*\* field);

Creates a duplicate copy of the COM object. This method is inherited from the *IVrmlField Interface*.

#### GetType

**Prototype** GetType([out, retval] enumFieldType\* type);

Returns the type of this field, which will be SFCOLOR (4). This method is inherited from the *IVrmlField Interface*.

#### GetRed

**Prototype** GetRed([out, retval] float\* r);

Returns the red component of the field's RGB color triple.

#### **GetGreen**

**Prototype** `GetGreen([out, retval] float* g);`

Returns the green component of the field's RGB color triple.

#### **GetBlue**

**Prototype** `GetBlue([out, retval] float* b);`

Returns the blue component of the field's RGB color triple.

#### **GetValue**

**Prototype** `GetValue([out] float *values);`

Returns the value of the field in the values array. The red, green and blue components of the RGB color triple are returned in values[0], values[1] and values[2], respectively. The values array must be at least 3 elements in size or a crash may occur.

#### **ToString**

**Prototype** `ToString([out, retval] BSTR* s);`

Returns a text string representation of the field. The result will be a legal VRML definition for the field.

#### **Unadvise**

**Prototype** `Unadvise([in] IVrmlEventOutObserver* eventOutObserver);`

Stops notifying the specified IVrmlEventOutObserver of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

## VrmlConstSFFloat

The VrmlConstSFFloat COM object represents a read-only SFFloat field in the VRML scene. An SFFloat field contains one single-precision floating point number.

The VrmlConstSFFloat object supports the IVrmlConstSFFloat, IVrmlConstField and IVrmlField interfaces.

## IVrmlConstSFFloat Interface

### Advise

**Prototype** Advise([in] IVrmlEventOutObserver\* eventOutObserver, [in] VARIANT\* userData);

Starts notifying the specified IVrmlEventOutObserver interface of changes in this field's value. Note that although this field is read-only to COM API, it may be changed by other events in the VRML scene. This method is inherited from the *IVrmlField Interface*.

### Clone

**Prototype** Clone([out, retval] IVrmlField\*\* field);

Creates a duplicate copy of the COM object. This method is inherited from the *IVrmlField Interface*.

### GetType

**Prototype** GetType([out, retval] enumFieldType\* type);

Returns the type of this field, which will be SFFLOAT (6). This method is inherited from the *IVrmlField Interface*.

### GetValue

**Prototype** GetValue([out, retval] float \*value);

Returns the value of the field.

#### **ToString**

**Prototype** ToString([out, retval] BSTR\* s);

Returns a text string representation of the field. The result will be a legal VRML definition for the field.

#### **Unadvise**

**Prototype** Unadvise([in] IVrmlEventOutObserver\* eventOutObserver);

Stops notifying the specified IVrmlEventOutObserver of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

## VrmlConstSfImage

The VrmlConstSfImage COM object represents a read-only SfImage field in the VRML scene. An SfImage field defines an uncompressed two dimensional pixel image. See Section 5.5 of the VRML 2.0 Specification for a full explanation of the use of SfImage.

The VrmlConstSfImage object supports the IVrmlConstSfImage, IVrmlConstField and IVrmlField interfaces.

## IVrmlConstSfImage Interface

### Advise

**Prototype** Advise([in] IVrmlEventOutObserver\* eventOutObserver, [in] VARIANT\* userData);

Starts notifying the specified IVrmlEventOutObserver interface of changes in this field's value. Note that although this field is read-only to COM API, it may be changed by other events in the VRML scene. This method is inherited from the *IVrmlField Interface*.

### Clone

**Prototype** Clone([out, retval] IVrmlField\*\* field);

Creates a duplicate copy of the COM object. This method is inherited from the *IVrmlField Interface*.

### GetComponents

**Prototype** GetComponents([out, retval] int\* components);

Returns the number of components in the image. The number of components ranges from 1 to 4 and controls how pixel data is interpreted. See Section 5.5 of the VRML 2.0 Specification for a full explanation.

### GetHeight

**Prototype** GetHeight([out, retval] int\* height);

Returns the height of the image in pixels.

#### GetPixels

**Prototype** GetPixels([out] unsigned char \*pixels);

Copies the pixel data of the image into the pixels array. The pixels array must be at least width\*height\*components bytes in size or a crash may occur.

#### GetType

**Prototype** GetType([out, retval] enumFieldType\* type);

Returns the type of this field, which will be SFIMAGE (2). This method is inherited from the *IVrmlField Interface*.

#### GetWidth

**Prototype** GetWidth([out, retval] int\* width);

Returns the width of the image in pixels.

#### ToString

**Prototype** ToString([out, retval] BSTR\* s);

Returns a text string representation of the field. The result will be a legal VRML definition for the field.

#### Unadvise

**Prototype** Unadvise([in] IVrmlEventOutObserver\* eventOutObserver);

Stops notifying the specified IVrmlEventOutObserver of changes in this field's value. This method is inherited from the *IVrmlField Interface*.



## VrmlConstSFInt32

The VrmlConstSFInt32 COM object represents a read-only SFInt32 field in the VRML scene. An SFInt32 field contains a single 32-bit integer.

The VrmlConstSFInt32 object supports the IVrmlConstSFInt32, IVrmlConstField and IVrmlField interfaces.

### IVrmlConstSFInt32 Interface

#### Advise

**Prototype** Advise([in] IVrmlEventOutObserver\* eventOutObserver, [in] VARIANT\* userData);

Starts notifying the specified IVrmlEventOutObserver interface of changes in this field's value. Note that although this field is read-only to COM API, it may be changed by other events in the VRML scene. This method is inherited from the *IVrmlField Interface*.

#### Clone

**Prototype** Clone([out, retval] IVrmlField\*\* field);

Creates a duplicate copy of the COM object. This method is inherited from the *IVrmlField Interface*.

#### GetType

**Prototype** GetType([out, retval] enumFieldType\* type);

Returns the type of this field, which will be SFINT32 (8). This method is inherited from the *IVrmlField Interface*.

#### GetValue

**Prototype** GetValue([out, retval] int \*value);

Returns the value of the field.

#### Tostring

**Prototype** ToString([out, retval] BSTR\* s);

Returns a text string representation of the field. The result will be a legal VRML definition for the field.

#### Unadvise

**Prototype** Unadvise([in] IVrmlEventOutObserver\* eventOutObserver);

Stops notifying the specified IVrmlEventOutObserver of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

## VrmlConstSFNode

The VrmlConstSFNode COM object represents a read-only SFNode field in the VRML scene. An SFNode field specifies a single VRML node, or the value NULL to indicate that it is empty.

The VrmlConstSFNode object supports the IVrmlConstSFNode, IVrmlConstField and IVrmlField interfaces.

## IVrmlConstSFNode Interface

#### Advise

**Prototype** Advise([in] IVrmlEventOutObserver\* eventOutObserver, [in] VARIANT\* userData);

Starts notifying the specified IVrmlEventOutObserver interface of changes in this field's value. Note that although this field is read-only to COM API, it may be changed by other events in the VRML scene. This method is inherited from the *IVrmlField Interface*.

#### Clone

**Prototype** Clone([out, retval] IVrmlField\*\* field);

Creates a duplicate copy of the COM object. This method is inherited from the *IVrmlField Interface*.

## GetType

**Prototype** GetType([out, retval] enumFieldType\* type);

Returns the type of this field, which will be SFNODE (10). This method is inherited from the *IVrmlField Interface*.

## GetValue

**Prototype** GetValue([out, retval] IVrmlBaseNode \*\*value);

Returns the value of the field. The returned node is represented by an IVrmlBaseNode interface. The return value may also be NULL if the SFNode field is empty. (The representation of NULL will differ depending on the language you are using; consult the documentation from the vendor.)

## ToString

**Prototype** ToString([out, retval] BSTR\* s);

Returns a text string representation of the field. The result will be a legal VRML definition for the field.

## Unadvise

**Prototype** Unadvise([in] IVrmlEventOutObserver\* eventOutObserver);

Stops notifying the specified IVrmlEventOutObserver of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

## VrmlConstSFRotation

The VrmlConstSFRotation COM object represents a read-only SFRotation field in the VRML scene. An SFRotation field contains one arbitrary rotation, specified by four floating point values. The first three floating point values specify a normalized rotation axis vector about which the rotation takes place. The fourth value specifies the amount of right-handed rotation about that axis in radians.

The VrmlConstSFRotation object supports the IVrmlConstSFRotation, IVrmlConstField and IVrmlField interfaces.

### IVrmlConstSFRotation Interface

#### Advise

**Prototype** Advise([in] IVrmlEventOutObserver\* eventOutObserver, [in] VARIANT\* userData);

Starts notifying the specified IVrmlEventOutObserver interface of changes in this field's value. Note that although this field is read-only to COM API, it may be changed by other events in the VRML scene. This method is inherited from the *IVrmlField Interface*.

#### Clone

**Prototype** Clone([out, retval] IVrmlField\*\* field);

Creates a duplicate copy of the COM object. This method is inherited from the *IVrmlField Interface*.

#### GetAngle

**Prototype** GetAngle([out, retval] float\* angle);

Returns the field's angle of rotation in radians.

### GetValue

**Prototype** GetValue([out] float \*values);

Returns the value of the field in the values array. The X, Y and Z components of the field's rotation axis are returned in values[0], values[1] and values[2], respectively. The rotation angle in radians is returned in values[3]. The values array must be at least 4 elements in size or a crash may occur.

### GetX

**Prototype** GetX([out, retval] float\* x);

Returns the X component of the field's rotation axis.

### GetY

**Prototype** GetY([out, retval] float\* y);

Returns the Y component of the field's rotation axis.

### GetZ

**Prototype** GetZ([out, retval] float\* z);

Returns the Z component of the field's rotation axis.

### ToString

**Prototype** ToString([out, retval] BSTR\* s);

Returns a text string representation of the field. The result will be a legal VRML definition for the field.

### Unadvise

**Prototype** Unadvise([in] IVrmlEventOutObserver\* eventOutObserver);

Stops notifying the specified IVrmlEventOutObserver of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

## VrmlConstSFString

The VrmlConstSFString COM object represents a read-only SFString field in the VRML scene. An SFString field specifies a single string.

The VrmlConstSFString object supports the IVrmlConstSFString, IVrmlConstField and IVrmlField interfaces.

### IVrmlConstSFString Interface

#### Advise

**Prototype** Advise([in] IVrmlEventOutObserver\* eventOutObserver, [in] VARIANT\* userData);

Starts notifying the specified IVrmlEventOutObserver interface of changes in this field's value. Note that although this field is read-only to COM API, it may be changed by other events in the VRML scene. This method is inherited from the *IVrmlField Interface*.

#### Clone

**Prototype** Clone([out, retval] IVrmlField\*\* field);

Creates a duplicate copy of the COM object. This method is inherited from the *IVrmlField Interface*.

#### GetType

**Prototype** GetType([out, retval] enumFieldType\* type);

Returns the type of this field, which will be SFSTRING (14). This method is inherited from the *IVrmlField Interface*.

#### GetValue

**Prototype** GetValue([out, retval] BSTR\* value);

Returns the value of the field.

### ToString

**Prototype** ToString([out, retval] BSTR\* s);

Returns a text string representation of the field. The result will be a legal VRML definition for the field.

### Unadvise

**Prototype** Unadvise([in] IVrmlEventOutObserver\* eventOutObserver);

Stops notifying the specified IVrmlEventOutObserver of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

## VrmlConstSfTime

The VrmlConstSfTime COM object represents a read-only SfTime field in the VRML scene. An SfTime field contains a single time value, which is a double-precision floating point number.

The VrmlConstSfTime object supports the IVrmlConstSfTime, IVrmlConstField and IVrmlField interfaces.

## IVrmlConstSfTime Interface

### Advise

**Prototype** Advise([in] IVrmlEventOutObserver\* eventOutObserver, [in] VARIANT\* userData);

Starts notifying the specified IVrmlEventOutObserver interface of changes in this field's value. Note that although this field is read-only to COM API, it may be changed by other events in the VRML scene. This method is inherited from the *IVrmlField Interface*.

### Clone

**Prototype** Clone([out, retval] IVrmlField\*\* field);

Creates a duplicate copy of the COM object. This method is inherited from the *IVrmlField Interface*.

#### **GetType**

**Prototype** `GetType([out, retval] enumFieldType* type);`

Returns the type of this field, which will be SFTIME (3). This method is inherited from the *IVrmlField Interface*.

#### **GetValue**

**Prototype** `GetValue([out, retval] double *value);`

Returns the time value of the field as a double-precision floating point number.

#### **ToString**

**Prototype** `ToString([out, retval] BSTR* s);`

Returns a text string representation of the field. The result will be a legal VRML definition for the field.

#### **Unadvise**

**Prototype** `Unadvise([in] IVrmlEventOutObserver* eventOutObserver);`

Stops notifying the specified IVrmlEventOutObserver of changes in this field's value. This method is inherited from the *IVrmlField Interface*.



## VrmlConstSFVec2f

The VrmlConstSFVec2f COM object represents a read-only SFVec2f field in the VRML scene. An SFVec2f field contains one 2D vector, specified by two floating point values.

The VrmlConstSFVec2f object supports the IVrmlConstSFVec2f, IVrmlConstField and IVrmlField interfaces.

### IVrmlConstSFVec2f Interface

#### Advise

**Prototype** Advise([in] IVrmlEventOutObserver\* eventOutObserver, [in] VARIANT\* userData);

Starts notifying the specified IVrmlEventOutObserver interface of changes in this field's value. Note that although this field is read-only to COM API, it may be changed by other events in the VRML scene. This method is inherited from the *IVrmlField Interface*.

#### Clone

**Prototype** Clone([out, retval] IVrmlField\*\* field);

Creates a duplicate copy of the COM object. This method is inherited from the *IVrmlField Interface*.

#### GetType

**Prototype** GetType([out, retval] enumFieldType\* type);

Returns the type of this field, which will be SFVEC2F (16). This method is inherited from the *IVrmlField Interface*.

#### GetValue

**Prototype** GetValue([out] float \*values);

Returns the value of the field in the values array. The X and Y components of the field's 2D vector are returned in values[0] and values[1], respectively. The values array must be at least 2 elements in size or a crash may occur.

### *VrmlConstSFVec2f*

#### **GetX**

**Prototype** `GetX([out, retval] float* x);`

Returns the X component of the field's 2D vector.

#### **GetY**

**Prototype** `GetY([out, retval] float* y);`

Returns the Y component of the field's 2D vector.

#### **ToString**

**Prototype** `ToString([out, retval] BSTR* s);`

Returns a text string representation of the field. The result will be a legal VRML definition for the field.

#### **Unadvise**

**Prototype** `Unadvise([in] IVrmlEventOutObserver*  
eventOutObserver);`

Stops notifying the specified `IVrmlEventOutObserver` of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

## VrmlConstSFVec3f

The VrmlConstSFVec3f COM object represents a read-only SFVec3f field in the VRML scene. An SFVec3f field contains one 3D vector, specified by three floating point values.

The VrmlConstSFVec3f object supports the IVrmlConstSFVec3f, IVrmlConstField and IVrmlField interfaces.

### IVrmlConstSFVec3f Interface

#### Advise

**Prototype** Advise([in] IVrmlEventOutObserver\* eventOutObserver, [in] VARIANT\* userData);

Starts notifying the specified IVrmlEventOutObserver interface of changes in this field's value. Note that although this field is read-only to COM API, it may be changed by other events in the VRML scene. This method is inherited from the *IVrmlField Interface*.

#### Clone

**Prototype** Clone([out, retval] IVrmlField\*\* field);

Creates a duplicate copy of the COM object. This method is inherited from the *IVrmlField Interface*.

#### GetType

**Prototype** GetType([out, retval] enumFieldType\* type);

Returns the type of this field, which will be SFVEC3F (18). This method is inherited from the *IVrmlField Interface*.

#### GetValue

**Prototype** GetValue([out] float \*values);

Returns the value of the field in the values array. The X, Y, and Z components of the field's 3D vector are returned in values[0], values[1] and values[2], respectively. The values array must be at least 3 elements in size or a crash may occur.

#### **GetX**

**Prototype**    `GetX([out, retval] float* x);`

Returns the X component of the field's 3D vector.

#### **GetY**

**Prototype**    `GetY([out, retval] float* y);`

Returns the Y component of the field's 3D vector.

#### **GetZ**

**Prototype**    `GetZ([out, retval] float* z);`

Returns the Z component of the field's 3D vector.

#### **ToString**

**Prototype**    `ToString([out, retval] BSTR* s);`

Returns a text string representation of the field. The result will be a legal VRML definition for the field.

#### **Unadvise**

**Prototype**    `Unadvise([in] IVrmlEventOutObserver*  
eventOutObserver);`

Stops notifying the specified `IVrmlEventOutObserver` of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

## VrmlMFCOLOR

The VrmlMFCOLOR COM object represents a MFCOLOR field in the VRML scene. An MFCOLOR field specifies zero or more RGB (red-green-blue) color triples. Each color triple consists of three floating point numbers in the range 0.0 to 1.0.

The VrmlMFCOLOR object supports the IVrmlMFCOLOR, IVrmlMField and IVrmlField interfaces.

## IVrmlMFCOLOR Interface

### AddValue

**Prototype** AddValue([in] float r, [in] float g, [in] float b);

Adds a new color triple to the end of the field. The red, green and blue components of the new triple will be assigned the values r, g, and b, respectively.

### Advise

**Prototype** Advise([in] IVrmlEventOutObserver\* eventOutObserver, [in] VARIANT\* userData);

Starts notifying the specified IVrmlEventOutObserver interface of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

### Clear

**Prototype** Clear();

Clears the field by deleting all of the elements. This method is inherited from the *IVrmlMField Interface*.

### Clone

**Prototype** Clone([out, retval] IVrmlField\*\* field);

Creates a duplicate copy of the COM object. This method is inherited from the *IVrmlField Interface*.

#### Delete

**Prototype** Delete([in] int index);

Deletes the index'th color triple. This method is inherited from the *IVrmlMField Interface*.

#### Get1Value

**Prototype** Get1Value([in] int index, [out] float \*r, [out] float \*g, [out] float \*b);

Retrieves the index'th color triple and stores the red, green and blue values of the triple in the variables r, g, and b, respectively.

#### GetSize

**Prototype** GetSize([out, retval] int\* size);

Returns the number of RGB color triples in the field. This method is inherited from the *IVrmlMField Interface*.

#### GetType

**Prototype** GetType([out, retval] enumFieldType\* type);

Returns the type of this field, which will be MFCOLOR (5). This method is inherited from the *IVrmlField Interface*.

#### GetValue

**Prototype** GetValue([out] float\* values);

Copies all of the color triples in the field into the values array. The first triple will be copied into values[0], values[1], and values[2] in R,G,B order; the second triple into values[3], values[4], values[5], and so on.

The values array must be large enough to accommodate the result or a crash may occur. The number of elements required is 3 times the result of the GetSize method.

### InsertValue

**Prototype** `InsertValue([in] int index, [in] float r, [in] float g, [in] float b);`

Inserts a new color triple before the index'th color triple. The red, green and blue components of the new triple are assigned the values r, g, b, respectively. The field is expanded to accommodate the new color triple and the index'th color triple and all triples following it are shifted one slot to make space for the new element.

### Set1Value

**Prototype** `Set1Value([in] int index, [in] float r, [in] float g, [in] float b);`

Sets the index'th color triple in the field. The red, green and blue components of the RGB color triple are set to the values of the r, g, and b parameters, respectively.

### SetValue

**Prototype** `SetValue([in] int size, [in] float *values);`

Sets the value of the field. The values array should contain the color triples, in R,G,B order and packed one after the other. In VB, the size parameter should contain the number of color triples in the values array. Make sure the values array contains at least size\*3 elements or a crash may occur.

### ToString

**Prototype** `Tostring([out, retval] BSTR* s);`

Returns a text string representation of the field. The result will be a legal VRML definition for the field.

### Unadvise

**Prototype** `Unadvise([in] IVrmlEventOutObserver* eventOutObserver);`

Stops notifying the specified IVrmlEventOutObserver of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

# VrmlMFFloat

The VrmlMFFloat COM object represents a MFFloat field in the VRML scene. An MFFloat field specifies zero or more single-precision floating point numbers.

The VrmlMFFloat object supports the IVrmlMFFloat, IVrmlMField and IVrmlField interfaces.

## IVrmlMFFloat Interface

### AddValue

**Prototype** AddValue([in] float value);

Adds the value parameter as a new value at the end of the field. The field will be expanded to accommodate the new element.

### Advise

**Prototype** Advise([in] IVrmlEventOutObserver\* eventOutObserver, [in] VARIANT\* userData);

Starts notifying the specified IVrmlEventOutObserver interface of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

### Clear

**Prototype** Clear();

Clears the field by deleting all of the elements. This method is inherited from the *IVrmlMField Interface*.

### Clone

**Prototype** Clone([out, retval] IVrmlField\*\* field);

Creates a duplicate copy of the COM object. This method is inherited from the *IVrmlField Interface*.



## Delete

**Prototype** Delete([in] int index);

Deletes the index'th value. This method is inherited from the *IVrmlMField Interface*.

## Get1Value

**Prototype** Get1Value([in] int index, [out, retval] float \*value);

Retrieves the index'th floating point number from the field and stores it in the value parameter.

## GetSize

**Prototype** GetSize([out, retval] int\* size);

Returns the number of floating point numbers in the field. This method is inherited from the *IVrmlMField Interface*.

## GetType

**Prototype** GetType([out, retval] enumFieldType\* type);

Returns the type of this field, which will be MFFLOAT (7). This method is inherited from the *IVrmlField Interface*.

## GetValue

**Prototype** GetValue([out] float\* values);

Copies all of the floating point numbers in the field into the values array.

The values array must be large enough to accommodate the result or a crash may occur. The number of elements required is returned by the GetSize method.

## InsertValue

**Prototype** InsertValue([in] int index, [in] float value);

Inserts a new value before the index'th value. All values following the new value are shifted one slot to make space for the new element.

#### Set1Value

**Prototype** Set1Value([in] int index, [in] float value);

Sets the index'th value in the field to the value parameter.

#### SetValue

**Prototype** SetValue([in] int size, [in] float \*values);

Sets the value of the field. The size parameter should contain the number of values in the values array. Make sure the values array contains at least size elements or a crash may occur.

#### ToString

**Prototype** ToString([out, retval] BSTR\* s);

Returns a text string representation of the field. The result will be a legal VRML definition for the field.

#### Unadvise

**Prototype** Unadvise([in] IVrmlEventOutObserver\*  
eventOutObserver);

Stops notifying the specified IVrmlEventOutObserver of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

## VrmlMField

The VrmlMField family of objects represents multiple value fields in the VRML scene, for example VrmlMFColor. All objects in the VrmlMField family support the IVrmlMField and IVrmlField interfaces.

### IVrmlMField Interface

#### Advise

**Prototype** Advise([in] IVrmlEventOutObserver\* eventOutObserver, [in] VARIANT\* userData);

Starts notifying the specified IVrmlEventOutObserver interface of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

#### Clear

**Prototype** Clear();

Clears the field by deleting all of the elements. The field will be reset to zero length.

#### Clone

**Prototype** Clone([out, retval] IVrmlField\*\* field);

Creates a copy of the COM object. This method is inherited from the *IVrmlField Interface*.

#### Delete

**Prototype** Delete([in] int index);

Deletes the index'th element of the field. The gap created by deleting the element is filled by moving all of the elements after it up one slot. The field is resized to a capacity of one less element.

#### GetSize

**Prototype**    GetSize([out, retval] int\* size);

Returns the number of elements in the field. The definition of an element will vary depending on the type of field; see the description of this method for specific field types.

#### GetType

**Prototype**    GetType([out, retval] enumFieldType\* type);

Returns the type of this field. This method is inherited from the *IVrmlField Interface*.

#### ToString

**Prototype**    ToString([out, retval] BSTR\* s);

Returns a text string representation of the field. The result will be a legal VRML definition for the field.

#### Unadvise

**Prototype**    Unadvise([in] IVrmlEventOutObserver\*  
                  eventOutObserver);

Stops notifying the specified IVrmlEventOutObserver of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

## VrmlMFInt32

The VrmlMFInt32 COM object represents a MFInt32 field in the VRML scene. An MFInt32 field specifies zero or more 32-bit integers.

The VrmlMFInt32 object supports the IVrmlMFInt32, IVrmlMField and IVrmlField interfaces.

### IVrmlMFInt32 Interface

#### AddValue

**Prototype** AddValue([in] int value);

Adds the value parameter as a new value at the end of the field. The field will be expanded to accommodate the new element.

#### Advise

**Prototype** Advise([in] IVrmlEventOutObserver\* eventOutObserver, [in] VARIANT\* userData);

Starts notifying the specified IVrmlEventOutObserver interface of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

#### Clear

**Prototype** Clear();

Clears the field by deleting all of the elements. This method is inherited from the *IVrmlMField Interface*.

#### Clone

**Prototype** Clone([out, retval] IVrmlField\*\* field);

Creates a duplicate copy of the COM object. This method is inherited from the *IVrmlField Interface*.

#### Delete

**Prototype** Delete([in] int index);

Deletes the index'th value. This method is inherited from the *IVrmlMField Interface*.

#### Get1Value

**Prototype** Get1Value([in] int index, [out, retval] int \*value);

Retrieves the index'th integer from the field and stores it in the value parameter.

#### GetSize

**Prototype** GetSize([out, retval] int\* size);

Returns the number of integers in the field. This method is inherited from the *IVrmlMField Interface*.

#### GetType

**Prototype** GetType([out, retval] enumFieldType\* type);

Returns the type of this field, which will be MFINT32 (9). This method is inherited from the *IVrmlField Interface*.

#### GetValue

**Prototype** GetValue([out] int\* values);

Copies all of the integers in the field into the values array.

The values array must be large enough to accommodate the result or a crash may occur. The number of elements required is returned by the GetSize method.

#### InsertValue

**Prototype** InsertValue([in] int index, [in] int value);

Inserts a new value before the index'th value. All values following the new value are shifted one slot to make space for the new element.

### Set1Value

**Prototype** Set1Value([in] int index, [in] int value);

Sets the index'th value in the field to the value parameter.

### SetValue

**Prototype** SetValue([in] int size, [in] int \*values);

Sets the value of the field. The size parameter should contain the number of values in the values array. Make sure the values array contains at least size elements or a crash may occur.

### ToString

**Prototype** ToString([out, retval] BSTR\* s);

Returns a text string representation of the field. The result will be a legal VRML definition for the field.

### Unadvise

**Prototype** Unadvise([in] IVrmlEventOutObserver\* eventOutObserver);

Stops notifying the specified IVrmlEventOutObserver of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

## VrmlMFNode

The VrmlMFNode COM object represents a MFNode field in the VRML scene. An MFNode field specifies zero or more VRML nodes.

The VrmlMFNode object supports the IVrmlMFNode, IVrmlMField and IVrmlField interfaces.

### IVrmIMFNode Interface

#### AddValue

**Prototype** AddValue([in] IVrmIBaseNode\* value);

Adds the value parameter as a new value at the end of the field. The field will be expanded to accommodate the new element.

#### Advise

**Prototype** Advise([in] IVrmIEventOutObserver\* eventOutObserver, [in] VARIANT\* userData);

Starts notifying the specified IVrmIEventOutObserver interface of changes in this field's value. This method is inherited from the *IVrmIField Interface*.

#### Clear

**Prototype** Clear();

Clears the field by deleting all of the elements. This method is inherited from the *IVrmIField Interface*.

#### Clone

**Prototype** Clone([out, retval] IVrmIField\*\* field);

Creates a duplicate copy of the COM object. This method is inherited from the *IVrmIField Interface*.

#### Delete

**Prototype** Delete([in] int index);

Deletes the index'th node. This method is inherited from the *IVrmIField Interface*.



### Get1Value

**Prototype** `Get1Value([in] int index,  
[out, retval] IVrmlBaseNode **value);`

Retrieves the index'th node from the field and stores it in the value parameter. The node is represented by an IVrmlBaseNode interface.

### GetSize

**Prototype** `GetSize([out, retval] int* size);`

Returns the number of nodes in the field. This method is inherited from the *IVrmlMField Interface*.

### GetType

**Prototype** `GetType([out, retval] enumFieldType* type);`

Returns the type of this field, which will be MFNODE (11). This method is inherited from the *IVrmlField Interface*.

### GetValue

**Prototype** `GetValue([out] IVrmlBaseNode** values);`

Copies all of the nodes in the field into the values array. Each node is represented by an IVrmlBaseNode interface.

The values array must be large enough to accommodate the result or a crash may occur. The number of elements required is returned by the GetSize method.

### InsertValue

**Prototype** `InsertValue([in] int index, [in] IVrmlBaseNode* value);`

Inserts a new node before the index'th node. All nodes following the new value are shifted one slot to make space for the new element.

#### Set1Value

**Prototype** Set1Value([in] int index, [in] IVrmlBaseNode\* value);

Sets the index'th node in the field to the node in the value parameter.

#### SetValue

**Prototype** SetValue([in] int size, [in] IVrmlBaseNode \*\*values);

Sets the value of the field. The size parameter should contain the number of nodes in the values array. Make sure the values array contains at least size nodes or a crash may occur.

#### SetValueFromConstMFNode

**Prototype** SetValueFromConstMFNode([in]  
IVrmlConstMFNode\* mfnode);

Copies the values in the specified IVrmlConstMFNode field into this field.

#### SetValueFromMFNode

**Prototype** SetValueFromMFNode([in] IVrmlMFNode\* mfnode);

Copies the values in the specified IVrmlMFNode field into this field.

#### ToString

**Prototype** ToString([out, retval] BSTR\* s);

Returns a text string representation of the field. The result will be a legal VRML definition for the field.

#### Unadvise

**Prototype** Unadvise([in] IVrmlEventOutObserver\*  
eventOutObserver);

Stops notifying the specified IVrmlEventOutObserver of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

## VrmlMFRotation

The VrmlMFRotation COM object represents a MFRotation field in the VRML scene. An MFRotation field specifies zero or more arbitrary rotations. Each rotation is specified by four floating point values. The first three values specify a normalized rotation axis vector about which the rotation takes place. The fourth value specifies the amount of right-handed rotation about that axis in radians.

The VrmlMFRotation object supports the IVrmlMFRotation, IVrmlMField and IVrmlField interfaces.

## IVrmlMFRotation Interface

### AddValue

**Prototype** AddValue([in] float x, [in] float y, [in] float z, [in] float angle);

Adds a new rotation to the end of the field. The X, Y and Z components of the rotation axis will be assigned the values x, y, and z, respectively. The rotation angle will be assigned the value angle.

### Advise

**Prototype** Advise([in] IVrmlEventOutObserver\* eventOutObserver, [in] VARIANT\* userData);

Starts notifying the specified IVrmlEventOutObserver interface of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

### Clear

**Prototype** Clear();

Clears the field by deleting all of the elements. This method is inherited from the *IVrmlMField Interface*.

#### Clone

**Prototype** Clone([out, retval] IVrmlField\*\* field);

Creates a duplicate copy of the COM object. This method is inherited from the *IVrmlField Interface*.

#### Delete

**Prototype:** Delete([in] int index);

Deletes the index'th rotation. This method is inherited from the *IVrmlMField Interface*.

#### Get1Value

**Prototype** Get1Value([in] int index, [out] float \*x, [out] float \*y, [out] float \*z, [out] float \*angle);

Retrieves the index'th rotation. The rotation axis is stored in the variables x, y, and z, respectively, and the angle of rotation is stored in the variable angle.

#### GetSize

**Prototype** GetSize([out, retval] int\* size);

Returns the number of rotations in the field. This method is inherited from the *IVrmlMField Interface*.

#### GetType

**Prototype** GetType([out, retval] enumFieldType\* type);

Returns the type of this field, which will be MFROTATION (13). This method is inherited from the *IVrmlField Interface*.

## GetValue

**Prototype** GetValue([out] float\* values);

Copies all of the rotations in the field into the values array. The first rotation will be copied into values[0], values[1], values[2], and values[3] in the order X, Y, Z, angle; the second triple into values[4...7], and so on.

The values array must be large enough to accommodate the result or a crash may occur. The number of elements required is 4 times the result of the GetSize method.

## InsertValue

**Prototype** InsertValue([in] int index, [in] float x, [in] float y, [in] float z, [in] float angle);

Inserts a new rotation before the index'th rotation. The X, Y and Z components of the rotation axis are assigned the values x, y, and z, respectively. The rotation angle is assigned the value angle. The field is expanded to accommodate the new rotation and the index'th rotation and all rotations following it are shifted one slot to make space for the new element.

## Set1Value

**Prototype** Set1Value([in] int index, [in] float x, [in] float y, [in] float z, [in] float angle);

Sets the index'th rotation in the field. The X, Y and Z components of the rotation axis are assigned the values x, y, and z, respectively. The rotation angle is assigned the value angle.

## SetValue

**Prototype** SetValue([in] int size, [in] float \*values);

Sets the value of the field. The size parameter should contain the number of rotations in the values array. The values array should contain rotations in X, Y, Z, angle order and packed one after the other. Make sure the values array contains at least size\*4 elements or a crash may occur.

### VrmlMFString

#### Tostring

**Prototype** ToString([out, retval] BSTR\* s);

Returns a text string representation of the field. The result will be a legal VRML definition for the field.

#### Unadvise

**Prototype** Unadvise([in] IVrmlEventOutObserver\* eventOutObserver);

Stops notifying the specified IVrmlEventOutObserver of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

## VrmlMFString

The VrmlMFString COM object represents a MFString field in the VRML scene. An MFString field specifies zero or more strings.

The VrmlMFString object supports the IVrmlMFString, IVrmlMField and IVrmlField interfaces.

## IVrmlMFString Interface

#### AddValue

**Prototype** AddValue([in] BSTR value);

Adds the value parameter as a new string at the end of the field. The field will be expanded to accommodate the new element.

#### Advise

**Prototype** Advise([in] IVrmlEventOutObserver\* eventOutObserver, [in] VARIANT\* userData);

Starts notifying the specified IVrmlEventOutObserver interface of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

### Clear

**Prototype** Clear();

Clears the field by deleting all of the elements. This method is inherited from the *IVrmlMField Interface*.

### Clone

**Prototype** Clone([out, retval] IVrmlField\*\* field);

Creates a duplicate copy of the COM object. This method is inherited from the *IVrmlMField Interface*.

### Delete

**Prototype** Delete([in] int index);

Deletes the index'th string. This method is inherited from the *IVrmlMField Interface*.

### Get1Value

**Prototype** Get1Value([in] int index, [out, retval] BSTR\* value);

Retrieves the index'th string from the field and stores it in the value parameter.

### GetSize

**Prototype** GetSize([out, retval] int\* size);

Returns the number of strings in the field. This method is inherited from the *IVrmlMField Interface*.

### GetType

**Prototype** GetType([out, retval] enumFieldType\* type);

Returns the type of this field, which will be MFSTRING (15). This method is inherited from the *IVrmlMField Interface*.

#### GetValue

**Prototype** GetValue([out] BSTR\* values);

Copies all of the strings in the field into the values array. The values array must be large enough to accommodate the result or a crash may occur. The number of elements required is returned by the GetSize method.

#### InsertValue

**Prototype** InsertValue([in] int index, [in] BSTR value);

Inserts a new string before the index'th string. All strings following the new value are shifted one slot to make space for the new element.

#### Set1Value

**Prototype** Set1Value([in] int index, [in] BSTR value);

Sets the index'th string in the field to the string in the value parameter.

#### SetValue

**Prototype** SetValue([in] int size, [in] BSTR \*values);

Sets the value of the field. The size parameter should contain the number of strings in the values array. Make sure the values array contains at least size strings or a crash may occur.

#### ToString

**Prototype** ToString([out, retval] BSTR\* s);

Returns a text string representation of the field. The result will be a legal VRML definition for the field.

#### Unadvise

**Prototype** Unadvise([in] IVrmlEventOutObserver\* eventOutObserver);

Stops notifying the specified IVrmlEventOutObserver of changes in this field's value. Inherited from the *IVrmlField Interface*.



## VrmlMTime

The VrmlMTime COM object represents a MTime field in the VRML scene. An MTime field specifies zero or more time values, which are double-precision floating point numbers.

The VrmlMTime object supports the IVrmlMTime, IVrmlMField and IVrmlField interfaces.

## IVrmlMTime Interface

### AddValue

**Prototype** AddValue([in] double time);

Adds the value parameter as a new time value at the end of the field. The field will be expanded to accommodate the new element.

### Advise

**Prototype** Advise([in] IVrmlEventOutObserver\* eventOutObserver, [in] VARIANT\* userData);

Starts notifying the specified IVrmlEventOutObserver interface of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

### Clear

**Prototype** Clear();

Clears the field by deleting all of the elements. This method is inherited from the *IVrmlMField Interface*.

### Clone

**Prototype** Clone([out, retval] IVrmlField\*\* field);

Creates a duplicate copy of the COM object. This method is inherited from the *IVrmlField Interface*.

#### Delete

**Prototype** Delete([in] int index);

Deletes the index'th value. This method is inherited from the *IVrmlMField Interface*.

#### Get1Value

**Prototype** Get1Value([in] int index, [out, retval] double\* time);

Retrieves the index'th time value from the field and stores it in the value parameter.

#### GetSize

**Prototype** GetSize([out, retval] int\* size);

Returns the number of time values in the field. This method is inherited from the *IVrmlMField Interface*.

#### GetType

**Prototype** GetType([out, retval] enumFieldType\* type);

Returns the type of this field, which will be MFTIME (20). This method is inherited from the *IVrmlField Interface*.

#### GetValue

**Prototype** GetValue([out] double\* values);

Copies all of the time values in the field into the values array.

The values array must be large enough to accommodate the result or a crash may occur. The number of elements required is returned by the GetSize method.

#### InsertValue

**Prototype** InsertValue([in] int index, [in] double time);

Inserts a new time value before the index'th value. All values following the new value are shifted one slot to make space for the new element.

**Set1Value**

**Prototype** Set1Value([in] int index, [in] double time);

Sets the index'th time value in the field to the value parameter.

**SetValue**

**Prototype** SetValue([in] int size, [in] double \*values);

Sets the value of the field. The size parameter should contain the number of time values in the values array. Make sure the values array contains at least size elements or a crash may occur.

**ToString**

**Prototype** ToString([out, retval] BSTR\* s);

Returns a text string representation of the field. The result will be a legal VRML definition for the field.

**Unadvise**

**Prototype** Unadvise([in] IVrmlEventOutObserver\* eventOutObserver);

Stops notifying the specified IVrmlEventOutObserver of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

## VrmlMFVec2f

The VrmlMFVec2f COM object represents a MFVec2f field in the VRML scene. An MFVec2f field specifies zero or more 2D vectors. Each vector is a pair of floating point values.

The VrmlMFVec2f object supports the IVrmlMFVec2f, IVrmlMField and IVrmlField interfaces.

### IVrmlMFVec2f Interface

#### AddValue

**Prototype** AddValue([in] float x, [in] float y);

Adds a new 2D vector to the end of the field. The X and Y components of the new vector will be assigned the values x and y, respectively.

#### Advise

**Prototype** Advise([in] IVrmlEventOutObserver\* eventOutObserver, [in] VARIANT\* userData);

Starts notifying the specified IVrmlEventOutObserver interface of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

#### Clear

**Prototype** Clear();

Clears the field by deleting all of the elements. This method is inherited from the *IVrmlMField Interface*.

#### Clone

**Prototype** Clone([out, retval] IVrmlField\*\* field);

Creates a duplicate copy of the COM object. This method is inherited from the *IVrmlField Interface*.

## Delete

**Prototype** Delete([in] int index);

Deletes the index'th color triple. This method is inherited from the *IVrmlMField Interface*.

## Get1Value

**Prototype** Get1Value([in] int index, [out] float \*x,  
[out] float \*y);

Retrieves the index'th 2D vector and stores the X and Y components of the vector in the variables x and y, respectively.

## GetSize

**Prototype** GetSize([out, retval] int\* size);

Returns the number of 2D vectors in the field. This method is inherited from the *IVrmlMField Interface*.

## GetType

**Prototype** GetType([out, retval] enumFieldType\* type);

Returns the type of this field, which will be MFVEC2F (17). This method is inherited from the *IVrmlField Interface*.

## GetValue

**Prototype** GetValue([out] float\* values);

Copies all of the 2D vectors in the field into the values array. The first 2D vector will be copied into values[0] and values[1] in X, Y order; the second 2D vector into values[2] and values[3], and so on.

The values array must be large enough to accommodate the result or a crash may occur. The number of elements required is 2 times the result of the GetSize method.

#### InsertValue

**Prototype** InsertValue([in] int index, [in] float x, [in] float y);

Inserts a new 2D vector before the index'th vector. The X and Y components of the new vector are assigned the values x and y, respectively. The field is expanded to accommodate the new vector and the index'th vector and all vectors following it are shifted one slot to make space for the new element.

#### Set1Value

**Prototype** Set1Value([in] int index, [in] float x, [in] float y);

Sets the index'th 2D vector in the field. The X and Y components of the vector are assigned the values x and y, respectively.

#### SetValue

**Prototype** SetValue([in] int size, [in] float \*values);

Sets the value of the field. The size parameter should contain the number of 2D vectors in the values array. The values array should contain the vectors, in X,Y order and packed one after the other. Make sure the values array contains at least size\*2 elements or a crash may occur.

#### ToString

**Prototype** ToString([out, retval] BSTR\* s);

Returns a text string representation of the field. The result will be a legal VRML definition for the field.

#### Unadvise

**Prototype** Unadvise([in]  
IVrmlEventOutObserver\* eventOutObserver);

Stops notifying the specified IVrmlEventOutObserver of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

## VrmlMFVec3f

The VrmlMFVec3f COM object represents a MFVec3f field in the VRML scene. An MFVec3f field specifies zero or more 3D vectors. Each vector consists of three floating point values.

The VrmlMFVec3f object supports the IVrmlMFVec3f, IVrmlMField and IVrmlField interfaces.

### IVrmlMFVec3f Interface

#### AddValue

**Prototype** AddValue([in] float x, [in] float y, [in] float z);

Adds a new 3D vector to the end of the field. The X, Y and Z components of the new vector will be assigned the values x, y and z, respectively.

#### Advise

**Prototype** Advise([in] IVrmlEventOutObserver\* eventOutObserver, [in] VARIANT\* userData);

Starts notifying the specified IVrmlEventOutObserver interface of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

#### Clear

**Prototype** Clear();

Clears the field by deleting all of the elements. This method is inherited from the *IVrmlMField Interface*.

#### Clone

**Prototype** Clone([out, retval] IVrmlField\*\* field);

Creates a duplicate copy of the COM object. This method is inherited from the *IVrmlField Interface*.

#### Delete

**Prototype** Delete([in] int index);

Deletes the index'th color triple. This method is inherited from the *IVrmlMField Interface*.

#### Get1Value

**Prototype** Get1Value([in] int index, [out] float \*x, [out] float \*y, [out] float \*z);

Retrieves the index'th 3D vector and stores the X, Y and Z components of the vector in the variables x, y and z, respectively.

#### GetSize

**Prototype** GetSize([out, retval] int\* size);

Returns the number of 3D vectors in the field. This method is inherited from the *IVrmlMField Interface*.

#### GetType

**Prototype** GetType([out, retval] enumFieldType\* type);

Returns the type of this field, which will be MFVEC3F (19). This method is inherited from the *IVrmlField Interface*.

#### GetValue

**Prototype** GetValue([out] float\* values);

Copies all of the 3D vectors in the field into the values array. The first 3D vector will be copied into values[0], values[1] and values[2] in X, Y, Z order; the second 3D vector into values[3...5], and so on.

The values array must be large enough to accommodate the result or a crash may occur. The number of elements required is 3 times the result of the GetSize method.



### InsertValue

**Prototype** `InsertValue([in] int index, [in] float x, [in] float y, [in] float z);`

Inserts a new 3D vector before the index'th vector. The X, Y and Z components of the new vector are assigned the values x, y and z, respectively. The field is expanded to accommodate the new vector and the index'th vector and all vectors following it are shifted one slot to make space for the new element.

### Set1Value

**Prototype** `Set1Value([in] int index, [in] float x, [in] float y, [in] float z);`

Sets the index'th 3D vector in the field. The X, Y and Z components of the vector are assigned the values x, y and z, respectively.

### SetValue

**Prototype** `SetValue([in] int size, [in] float *values);`

Sets the value of the field. The size parameter should contain the number of 3D vectors in the values array. The values array should contain the vectors, in X,Y,Z order and packed one after the other. Make sure the values array contains at least size\*3 elements or a crash may occur.

### ToString

**Prototype** `ToString([out, retval] BSTR* s);`

Returns a text string representation of the field. The result will be a legal VRML definition for the field.

### Unadvise

**Prototype** `Unadvise([in] IVrmlEventOutObserver* eventOutObserver);`

Stops notifying the specified IVrmlEventOutObserver of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

# VrmlSFBool

The VrmlSFBool COM object represents an SFBool field in the VRML scene. An SFBool field contains a single boolean value, either TRUE or FALSE.

The VrmlSFBool object supports the IVrmlSFBool and IVrmlField interfaces.

## IVrmlSFBool Interface

### Advise

**Prototype** Advise([in] IVrmlEventOutObserver\* eventOutObserver, [in] VARIANT\* userData);

Starts notifying the specified IVrmlEventOutObserver interface of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

### Clone

**Prototype** Clone([out, retval] IVrmlField\*\* field);

Creates a duplicate copy of the COM object. This method is inherited from the *IVrmlField Interface*.

### GetType

**Prototype** GetType([out, retval] enumFieldType\* type);

Returns the type of this field, which will be SFBOOL (1). This method is inherited from the *IVrmlField Interface*.

### GetValue

**Prototype** GetValue([out] boolean\* value);

Returns the value of the field.

### SetValue

**Prototype** SetValue([in] boolean value);

Sets the value of the field.

### **ToString**

**Prototype** ToString([out, retval] BSTR\* s);

Returns a text string representation of the field. The result will be a legal VRML definition for the field.

### **Unadvise**

**Prototype** Unadvise([in] IVrmlEventOutObserver\* eventOutObserver);

Stops notifying the specified IVrmlEventOutObserver of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

# VrmlSFCOLOR

The VrmlSFCOLOR COM object represents an SFCOLOR field in the VRML scene. An SFCOLOR field contains one RGB (red-green-blue) color triple, specified by three floating point values.

The VrmlSFCOLOR object supports the IVrmlSFCOLOR and IVrmlField interfaces.

## IVrmlSFCOLOR Interface

### Advise

**Prototype** Advise([in] IVrmlEventOutObserver\* eventOutObserver, [in] VARIANT\* userData);

Starts notifying the specified IVrmlEventOutObserver interface of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

### Clone

**Prototype** Clone([out, retval] IVrmlField\*\* field);

Creates a duplicate copy of the COM object. This method is inherited from the *IVrmlField Interface*.

### GetBlue

**Prototype** GetBlue([out, retval] float\* b);

Returns the blue component of the field's RGB color triple.

### GetGreen

**Prototype** GetGreen([out, retval] float\* g);

Returns the green component of the field's RGB color triple.

### GetRed

**Prototype** GetRed([out, retval] float\* r);

Returns the red component of the field's RGB color triple.

## GetType

**Prototype** GetType([out, retval] enumFieldType\* type);

Returns the type of this field, which will be SFCOLOR (4). This method is inherited from the *IVrmlField Interface*.

## GetValue

**Prototype** GetValue([out] float \*values);

Returns the value of the field in the values array. The red, green and blue components of the RGB color triple are returned in values[0], values[1] and values[2], respectively. The values array must be at least 3 elements in size or a crash may occur.

## SetValue

**Prototype** SetValue([in] float r, [in] float g, [in] float b);

Sets the value of the field's RGB color triple. The red, green and blue components of the color are set to the values of the r, g and b parameters, respectively.

## ToString

**Prototype** ToString([out, retval] BSTR\* s);

Returns a text string representation of the field. The result will be a legal VRML definition for the field.

## Unadvise

**Prototype** Unadvise([in] IVrmlEventOutObserver\* eventOutObserver);

Stops notifying the specified IVrmlEventOutObserver of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

## VrmlSFFloat

The VrmlSFFloat COM object represents an SFFloat field in the VRML scene. An SFFloat field contains one single-precision floating point number.

The VrmlSFFloat object supports the IVrmlSFFloat and IVrmlField interfaces.

## IVrmlSFFloat Interface

### Advise

**Prototype** Advise([in] IVrmlEventOutObserver\* eventOutObserver, [in] VARIANT\* userData);

Starts notifying the specified IVrmlEventOutObserver interface of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

### Clone

**Prototype** Clone([out, retval] IVrmlField\*\* field);

Creates a duplicate copy of the COM object. This method is inherited from the *IVrmlField Interface*.

### GetType

**Prototype** GetType([out, retval] enumFieldType\* type);

Returns the type of this field, which will be SFFLOAT (6). This method is inherited from the *IVrmlField Interface*.

### GetValue

**Prototype** GetValue([out, retval] float \*value);

Returns the value of the field.

### SetValue

**Prototype** SetValue([in] float value);

Sets the value of the field.

### ToString

**Prototype** ToString([out, retval] BSTR\* s);

Returns a text string representation of the field. The result will be a legal VRML definition for the field.

### Unadvise

**Prototype** Unadvise([in] IVrmlEventOutObserver\* eventOutObserver);

Stops notifying the specified IVrmlEventOutObserver of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

## VrmlSfImage

The VrmlSfImage COM object represents an SfImage field in the VRML scene. An SfImage field defines an uncompressed two dimensional pixel image. See Section 5.5 of the VRML 2.0 Specification for a full explanation of the use of SfImage.

The VrmlSfImage object supports the IVrmlSfImage and IVrmlField interfaces.

## IVrmlSfImage Interface

### Advise

**Prototype** Advise([in] IVrmlEventOutObserver\* eventOutObserver, [in] VARIANT\* userData);

Starts notifying the specified IVrmlEventOutObserver interface of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

### Clone

**Prototype** Clone([out, retval] IVrmlField\*\* field);

Creates a duplicate copy of the COM object. This method is inherited from the *IVrmlField Interface*.

#### GetComponentents

**Prototype** GetComponentents([out, retval] int\* components);

Returns the number of components in the image. The number of components ranges from 1 to 4 and controls how pixel data is interpreted. See Section 5.5 of the VRML 2.0 Specification for a full explanation.

#### GetHeight

**Prototype** GetHeight([out, retval] int\* height);

Returns the height of the image in pixels.

#### GetPixels

**Prototype** GetPixels([out] unsigned char \*pixels);

Copies the pixel data of the image into the pixels array. The pixels array must be at least width\*height\*components bytes in size or a crash may occur.

The number of bytes occupied by each pixel in the pixels array is the field's components value, returned by the GetComponentents method. Pixels are packed one after the other without any empty space between them, starting with the bottom row and ending with the top row of the image.

#### GetType

**Prototype** GetType([out, retval] enumFieldType\* type);

Returns the type of this field, which will be SFIMAGE (2). This method is inherited from the *IVrmlField Interface*.

#### GetWidth

**Prototype** GetWidth([out, retval] int\* width);

Returns the width of the image in pixels.



### SetValue

**Prototype** SetValue([in] int width, [in] int height,  
[in] int components, [in] unsigned char \*pixels);

Sets the field's image. See the *GetPixels* method for an explanation of the pixels array.

### ToString

**Prototype** ToString([out, retval] BSTR\* s);

Returns a text string representation of the field. The result will be a legal VRML definition for the field.

### Unadvise

**Prototype** Unadvise([in] IVrmlEventOutObserver\*  
eventOutObserver);

Stops notifying the specified IVrmlEventOutObserver of changes in this field's value. Inherited from the *IVrmlField Interface*.

# VrmlSFInt32

The VrmlSFInt32 COM object represents an SFInt32 field in the VRML scene. An SFInt32 field contains a single 32-bit integer.

The VrmlSFInt32 object supports the IVrmlSFInt32 and IVrmlField interfaces.

## IVrmlSFInt32 Interface

### Advise

**Prototype** Advise([in] IVrmlEventOutObserver\* eventOutObserver, [in] VARIANT\* userData);

Starts notifying the specified IVrmlEventOutObserver interface if changes in this field's value. This method is inherited from the *IVrmlField Interface*.

### Clone

**Prototype** Clone([out, retval] IVrmlField\*\* field);

Creates a duplicate copy of the COM object. This method is inherited from the *IVrmlField Interface*.

### GetType

**Prototype** GetType([out, retval] enumFieldType\* type);

Returns the type of this field, which will be SFINT32 (8). This method is inherited from the *IVrmlField Interface*.

### GetValue

**Prototype** GetValue([out, retval] int \*value);

Returns the value of the field.

### SetValue

**Prototype** SetValue([in] int value);

Sets the value of the field.

### ToString

**Prototype** ToString([out, retval] BSTR\* s);

Returns a text string representation of the field. The result will be a legal VRML definition for the field.

### Unadvise

**Prototype** Unadvise([in] IVrmlEventOutObserver\* eventOutObserver);

Stops notifying the specified IVrmlEventOutObserver of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

## VrmlSFNode

The VrmlSFNode COM object represents an SFNode field in the VRML scene. An SFNode field specifies a single VRML node, or the value NULL to indicate that it is empty.

The VrmlSFNode object supports the IVrmlSFNode and IVrmlField interfaces.

## IVrmlSFNode Interface

### Advise

**Prototype** Advise([in] IVrmlEventOutObserver\* eventOutObserver, [in] VARIANT\* userData);

Starts notifying the specified IVrmlEventOutObserver interface of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

### Clone

**Prototype** Clone([out, retval] IVrmlField\*\* field);

Creates a duplicate copy of the COM object. This method is inherited from the *IVrmlField Interface*.

#### GetType

**Prototype** GetType([out, retval] enumFieldType\* type);

Returns the type of this field, which will be SFNODE (10). This method is inherited from the *IVrmlField Interface*.

#### GetValue

**Prototype** GetValue([out, retval] IVrmlBaseNode \*\*value);

Returns the value of the field. The returned node is represented by an IVrmlBaseNode interface. The return value may also be NULL if the SFNode field is empty.

---

**Note** • The representation of NULL will differ depending on the language you are using; consult the documentation from the vendor.

---

#### SetValue

**Prototype** SetValue([in] IVrmlBaseNode\* value);

Sets the value of the field. The parameter value should either be an IVrmlBaseNode interface, or NULL to set the field to be empty.

#### ToString

**Prototype** ToString([out, retval] BSTR\* s);

Returns a text string representation of the field. The result will be a legal VRML definition for the field.

#### Unadvise

**Prototype** Unadvise([in] IVrmlEventOutObserver\* eventOutObserver);

Stops notifying the specified IVrmlEventOutObserver of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

## VrmlSFRotation

The VrmlSFRotation COM object represents an SFRotation field in the VRML scene. An SFRotation field contains one arbitrary rotation, specified by four floating point values. The first three floating point values specify a normalized rotation axis vector about which the rotation takes place. The fourth value specifies the amount of right-handed rotation about that axis in radians.

The VrmlSFRotation object supports the IVrmlSFRotation and IVrmlField interfaces.

## IVrmlSFRotation Interface

### Advise

**Prototype** Advise([in] IVrmlEventOutObserver\* eventOutObserver, [in] VARIANT\* userData);

Starts notifying the specified IVrmlEventOutObserver interface of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

### Clone

**Prototype** Clone([out, retval] IVrmlField\*\* field);

Creates a duplicate copy of the COM object. This method is inherited from the *IVrmlField Interface*.

### GetAngle

**Prototype** GetAngle([out, retval] float\* angle);

Returns the field's angle of rotation in radians.

### GetType

**Prototype** GetType([out, retval] enumFieldType\* type);

Returns the type of this field, which will be SFROTATION (12). This method is inherited from the *IVrmlField Interface*.

#### **GetValue**

**Prototype** GetValue([out] float \*values);

Returns the value of the field in the values array. The X, Y and Z components of the field's rotation axis are returned in values[0], values[1] and values[2], respectively. The rotation angle in radians is returned in values[3]. The values array must be at least 4 elements in size or a crash may occur.

#### **GetX**

**Prototype** GetX([out, retval] float\* x);

Returns the X component of the field's rotation axis.

#### **GetY**

**Prototype** GetY([out, retval] float\* y);

Returns the Y component of the field's rotation axis.

#### **GetZ**

**Prototype** GetZ([out, retval] float\* z);

Returns the Z component of the field's rotation axis.

#### **SetValue**

**Prototype** SetValue([in] float x, [in] float y, [in] float z, [in] float angle);

Sets the value of the field. The X, Y and Z components of the rotation axis will be set to the values of the x, y and z parameters, respectively. The angle of rotation will be set to the value of angle.

#### **ToString**

**Prototype** ToString([out, retval] BSTR\* s);

Returns a text string representation of the field. The result will be a legal VRML definition for the field.

## Unadvise

**Prototype** `Unadvise([in] IVrmlEventOutObserver* eventOutObserver);`

Stops notifying the specified IVrmlEventOutObserver of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

# VrmlSFString

The VrmlSFString COM object represents an SFString field in the VRML scene. An SFString field specifies a single string.

The VrmlSFString object supports the IVrmlSFString and IVrmlField interfaces.

## IVrmlSFString Interface

### Advise

**Prototype** `Advise([in] IVrmlEventOutObserver* eventOutObserver, [in] VARIANT* userData);`

Starts notifying the specified IVrmlEventOutObserver interface of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

### Clone

**Prototype** `Clone([out, retval] IVrmlField** field);`

Creates a duplicate copy of the COM object. This method is inherited from the *IVrmlField Interface*.

### GetType

**Prototype** `GetType([out, retval] enumFieldType* type);`

Returns the type of this field, which will be SFSTRING (14). This method is inherited from the *IVrmlField Interface*.

#### **GetValue**

**Prototype** GetValue([out, retval] BSTR\* value);

Returns the value of the field.

#### **SetValue**

**Prototype** SetValue([in] BSTR value);

Sets the value of the field to the string in the value parameter.

#### **ToString**

**Prototype** ToString([out, retval] BSTR\* s);

Returns a text string representation of the field. The result will be a legal VRML definition for the field.

#### **Unadvise**

**Prototype** Unadvise([in] IVrmlEventOutObserver\* eventOutObserver);

Stops notifying the specified IVrmlEventOutObserver of changes in this field's value. This method is inherited from the *IVrmlField Interface*.



## VrmlSFTime

The VrmlSFTime COM object represents an SFTime field in the VRML scene. An SFTime field contains a single time value, which is a double-precision floating point number.

The VrmlSFTime object supports the IVrmlSFTime and IVrmlField interfaces.

## IVrmlSFTime Interface

### Advise

**Prototype** Advise([in] IVrmlEventOutObserver\* eventOutObserver, [in] VARIANT\* userData);

Starts notifying the specified IVrmlEventOutObserver interface of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

### Clone

**Prototype** Clone([out, retval] IVrmlField\*\* field);

Creates a duplicate copy of the COM object. This method is inherited from the *IVrmlField Interface*.

### GetType

**Prototype** GetType([out, retval] enumFieldType\* type);

Returns the type of this field, which will be SFTIME (3). This method is inherited from the *IVrmlField Interface*.

### GetValue

**Prototype** GetValue([out, retval] double \*time);

Returns the time value of the field as a double-precision floating point number.

#### SetValue

**Prototype** SetValue([in] double time);

Sets the time value of the field to the double-precision float point number in the value parameter.

#### ToString

**Prototype** ToString([out, retval] BSTR\* s);

Returns a text string representation of the field. The result will be a legal VRML definition for the field.

#### Unadvise

**Prototype** Unadvise([in] IVrmlEventOutObserver\*  
eventOutObserver);

Stops notifying the specified IVrmlEventOutObserver of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

## VrmlSFVec2f

The VrmlSFVec2f COM object represents an SFVec2f field in the VRML scene. An SFVec2f field contains one 2D vector, specified by two floating point values.

The VrmlSFVec2f object supports the IVrmlSFVec2f and IVrmlField interfaces.

## IVrmlSFVec2f Interface

### Advise

**Prototype** Advise([in] IVrmlEventOutObserver\* eventOutObserver, [in] VARIANT\* userData);

Starts notifying the specified IVrmlEventOutObserver interface of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

### Clone

**Prototype** Clone([out, retval] IVrmlField\*\* field);

Creates a duplicate copy of the COM object. This method is inherited from the *IVrmlField Interface*.

### GetType

**Prototype** GetType([out, retval] enumFieldType\* type);

Returns the type of this field, which will be SFVEC2F (16). This method is inherited from the *IVrmlField Interface*.

### GetValue

**Prototype** GetValue([out] float \*values);

Returns the value of the field in the values array. The X and Y components of the field's 2D vector are returned in values[0] and values[1], respectively. The values array must be at least 2 elements in size or a crash may occur.

#### **GetX**

**Prototype**    `GetX([out, retval] float* x);`

Returns the X component of the field's 2D vector.

#### **GetY**

**Prototype**    `GetY([out, retval] float* y);`

Returns the Y component of the field's 2D vector.

#### **SetValue**

**Prototype**    `SetValue([in] float x, [in] float y);`

Sets the X and Y components of the field's 2D vector to the values of the x and y parameters, respectively.

#### **ToString**

**Prototype**    `ToString([out, retval] BSTR* s);`

Returns a text string representation of the field. The result will be a legal VRML definition for the field.

#### **Unadvise**

**Prototype**    `Unadvise([in] IVrmlEventOutObserver*  
eventOutObserver);`

Stops notifying the specified `IVrmlEventOutObserver` of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

## VrmlSFVec3f

The VrmlSFVec3f COM object represents an SFVec3f field in the VRML scene. An SFVec3f field contains one 3D vector, specified by three floating point values.

The VrmlSFVec3f object supports the IVrmlSFVec3f and IVrmlField interfaces.

### IVrmlSFVec3f Interface

#### Advise

**Prototype** Advise([in] IVrmlEventOutObserver\* eventOutObserver, [in] VARIANT\* userData);

Starts notifying the specified IVrmlEventOutObserver interface of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

#### Clone

**Prototype** Clone([out, retval] IVrmlField\*\* field);

Creates a duplicate copy of the COM object. This method is inherited from the *IVrmlField Interface*.

#### GetType

**Prototype** GetType([out, retval] enumFieldType\* type);

Returns the type of this field, which will be SFVEC3F (18). This method is inherited from the *IVrmlField Interface*.

#### GetValue

**Prototype** GetValue([out] float \*values);

Returns the value of the field in the values array. The X, Y, and Z components of the field's 3D vector are returned in values[0], values[1] and values[2], respectively. The values array must be at least 3 elements in size or a crash may occur.

#### **GetX**

**Prototype**    GetX([out, retval] float\* x);

Returns the X component of the field's 3D vector.

#### **GetY**

**Prototype**    GetY([out, retval] float\* y);

Returns the Y component of the field's 3D vector.

#### **GetZ**

**Prototype**    GetZ([out, retval] float\* z);

Returns the Z component of the field's 3D vector.

#### **SetValue**

**Prototype**    SetValue([in] float x, [in] float y, [in] float z);

Sets the X, Y and Z components of the field's 3D vector to the values of the x, y and z parameters, respectively.

#### **ToString**

**Prototype**    ToString([out, retval] BSTR\* s);

Returns a text string representation of the field. The result will be a legal VRML definition for the field.

#### **Unadvise**

**Prototype**    Unadvise([in] IVrmlEventOutObserver\*  
                  eventOutObserver);

Stops notifying the specified IVrmlEventOutObserver of changes in this field's value. This method is inherited from the *IVrmlField Interface*.

## VrmlObjectFactory Interface

The IVrmlObjectFactory interface is an additional interface supported by the VrmlBrowser COM object. You can query for IVrmlObjectFactory from the IVrmlBrowser interface.

The IVrmlObjectFactory interface permits you to create COM API field objects that are “unattached,” that is, not representing a field or event of any node in the VRML scene. An unattached field can be used to store private data for your program, and can be passed to COM API methods as a parameter.

Note that the methods that create Const field objects also take parameters specifying initial values. This is provided because there is no way to modify the value of an unattached Const field once it has been created.

### CreateVrmlConstMFColor

**Prototype** `CreateVrmlConstMFColor([in] int size,  
[in] float* values,  
[out, retval] IVrmlConstMFColor** field);`

Creates a new VrmlConstMFFloat object containing the specified value and returns its IVrmlConstMFFloat interface in field. The size parameter specifies the number of color triples in the values array. The values array is an array of floating-point numbers containing RGB color triples, packed one after the other, in R, G, B order.

### CreateVrmlConstMFFloat

**Prototype** `CreateVrmlConstMFInt32([in] int size,  
[in] float* values,  
[out, retval] IVrmlConstMFFloat** field);`

Creates a new VrmlConstMFFloat object containing the specified value and returns its IVrmlConstMFFloat interface in field. The size parameter specifies the number of values in the values array. The values array specifies the floating-point values that the new field object will contain.

#### **CreateVrmlConstMFInt32**

**Prototype** `CreateVrmlConstMFInt32([in] int size, [in] int* values, [out, retval] IVrmlConstMFInt32** field);`

Creates a new VrmlConstMFInt32 object containing the specified value and returns its IVrmlConstMFInt32 interface in field. The size parameter specifies the number of integers in the values array. The values array specifies the integers that the new field object will contain.

#### **CreateVrmlConstMFNode**

**Prototype** `CreateVrmlConstMFNode([in] int size, [in] IVrmlBaseNode** values, [out, retval] IVrmlConstMFNode** field);`

Creates a new VrmlConstMFNode object containing the specified value and returns its IVrmlConstMFNode interface in field. The size parameter specifies the number of nodes in the values array. The values array specifies the nodes that the new field object will contain.

#### **CreateVrmlConstMFRotation**

**Prototype** `CreateVrmlConstMFRotation([in] int size, [in] float* values, [out, retval] IVrmlConstMFRotation** field);`

Creates a new VrmlConstMFRotation object containing the specified value and returns its IVrmlConstMFRotation interface in field. The size parameter specifies the number of rotations in the values array. The values array is an array of floating-point numbers containing rotations, packed one after the other, in X, Y, Z angle order.



### CreateVrmlConstMFString

**Prototype** `CreateVrmlConstMFString([in] int size,  
[in] BSTR* values,  
[out, retval] IVrmlConstMFString** field);`

Creates a new VrmlConstMFString object containing the specified value and returns its IVrmlConstMFString interface in field. The size parameter specifies the number of strings in the values array. The values array specifies the strings that the new field object will contain.

### CreateVrmlConstMFTime

**Prototype** `CreateVrmlConstMFTime([in] int size,  
[in] double* values,  
[out, retval] IVrmlConstMFTime** field);`

Creates a new VrmlConstMFTime object containing the specified value and returns its IVrmlConstMFTime interface in field. The size parameter specifies the number of time values in the values array. The values array specifies the time values that the new field object will contain.

### CreateVrmlConstMFVec2f

**Prototype** `CreateVrmlConstMFVec2f([in] int size,  
[in] float* values,  
[out, retval] IVrmlConstMFVec2f** field);`

Creates a new VrmlConstMFVec2f object containing the specified value and returns its IVrmlConstMFVec2f interface in field. The size parameter specifies the number of 2D vectors in the values array. The values array is an array of floating-point numbers containing 2D vectors, packed one after the other, in X, Y order.

#### **CreateVrmlConstMFVec3f**

**Prototype** `CreateVrmlConstMFVec3f([in] int size,  
[in] float* values,  
[out, retval] IVrmlConstMFVec3f** field);`

Creates a new VrmlConstMFVec3f object containing the specified value and returns its IVrmlConstMFVec3f interface in field. The size parameter specifies the number of 3D vectors in the values array. The values array is an array of floating-point numbers containing 3D vectors, packed one after the other, in X, Y, Z order.

#### **CreateVrmlConstSFBool**

**Prototype** `CreateVrmlConstSFBool([in] boolean value,  
[out, retval] IVrmlConstSFBool** field);`

Creates a new VrmlConstSFBool object containing the specified value and returns its IVrmlConstSFBool interface in field.

#### **CreateVrmlConstSFColor**

**Prototype** `CreateVrmlConstSFColor([in] float r, [in] float g,  
[in] float b,  
[out, retval] IVrmlConstSFColor** field);`

Creates a new VrmlConstSFColor object containing the specified value and returns its IVrmlConstSFColor interface in field. The red, green and blue components of the color are assigned the values r, g, and b, respectively.

#### **CreateVrmlConstSFFloat**

**Prototype** `CreateVrmlConstSFFloat([in] float value,  
[out, retval] IVrmlConstSFFloat** field);`

Creates a new VrmlConstSFFloat object containing the specified value and returns the new field object's IVrmlConstSFFloat interface in field.

### CreateVrmlConstSFImage

**Prototype** `CreateVrmlConstSFImage([in] int width, [in] int height, [in] int components, [in] unsigned char *pixels, [out, retval] IVrmlConstSFImage** field);`

Creates a new VrmlConstSFImage object containing the specified image and returns the new field object's IVrmlConstSFImage interface in field.

The image is created with the specified width, height and number of components as discussed in Section 5.5 of the VRML 2.0 Specification. In the pixels array, each pixel occupies components bytes. Pixels are packed one after the other, starting at the bottom row and ending with the top row of the image.

### CreateVrmlConstSFInt32

**Prototype** `CreateVrmlConstSFInt32([in] int value, [out, retval] IVrmlConstSFInt32** field);`

Creates a new VrmlConstSFInt32 object containing the specified value and returns its IVrmlConstSFInt32 interface in field.

### CreateVrmlConstSFNode

**Prototype** `CreateVrmlConstSFNode([in] IVrmlBaseNode *node, [out, retval] IVrmlConstSFNode** field);`

Creates a new VrmlConstSFNode object containing the specified node, and returns the new object's IVrmlConstSFNode interface in field. The specified node should be a IVrmlBaseNode interface representing the desired node, or NULL if the new SFNode object is to be empty.

#### **CreateVrmlConstSFRotation**

**Prototype** `CreateVrmlConstSFRotation([in] float x, [in] float y, [in] float z, [in] float angle, [out, retval] IVrmlConstSFRotation** field);`

Creates a new VrmlConstSFRotation object containing the specified value and returns its IVrmlConstSFRotation interface in field. The X, Y and Z components of the rotation axis are assigned the values x, y, and z respectively. The rotation angle is assigned the value angle, which is specified in radians.

#### **CreateVrmlConstSFString**

**Prototype** `CreateVrmlConstSFString([in] BSTR value, [out, retval] IVrmlConstSFString** field);`

Creates a new VrmlConstSFString object containing the specified value and returns its IVrmlConstSFString interface in field.

#### **CreateVrmlConstSFTime**

**Prototype** `CreateVrmlConstSFTime([in] double value, [out, retval] IVrmlConstSFTime** field);`

Creates a new VrmlConstSFTime object containing the specified value and returns its IVrmlConstSFTime interface in field.

#### **CreateVrmlConstSFVec2f**

**Prototype** `CreateVrmlConstSFVec2f([in] float x, [in] float y, [out, retval] IVrmlConstSFVec2f** field);`

Creates a new VrmlConstSFVec2f object containing the specified value and returns its IVrmlConstSFVec2f interface in field. The X and Y components of the vector are assigned the values x and y, respectively.

**CreateVrmlConstSFVec3f**

**Prototype** `CreateVrmlConstSFVec3f([in] float x, [in] float y, [in] float z, [out, retval] IVrmlConstSFVec3f** field);`

Creates a new `VrmlConstSFVec3f` object containing the specified value and returns its `IVrmlConstSFVec3f` interface in `field`. The X, Y and Z components of the vector are assigned the values `x`, `y` and `z`, respectively.

**CreateVrmlMFColor**

**Prototype** `CreateVrmlMFColor([out, retval] IVrmlMFColor** field);`

Creates a new `VrmlMFColor` object and returns its `IVrmlMFColor` interface. The new field is empty; that is, its value is `[]`.

**CreateVrmlMFFloat**

**Prototype** `CreateVrmlMFFloat([out, retval] IVrmlMFFloat** field);`

Creates a new `VrmlMFFloat` object and returns its `IVrmlMFFloat` interface. The new field is empty; that is, its value is `[]`.

**CreateVrmlMFInt32**

**Prototype** `CreateVrmlMFInt32([out, retval] IVrmlMFInt32** field);`

Creates a new `VrmlMFInt32` object and returns its `IVrmlMFInt32` interface. The new field is empty; that is, its value is `[]`.

**CreateVrmlMFNode**

**Prototype** `CreateVrmlMFNode([out, retval] IVrmlMFNode** field);`

Creates a new `VrmlMFNode` object and returns its `IVrmlMFNode` interface. The new field is empty; that is, its value is `[]`.

#### **CreateVrmlMFRotation**

**Prototype** `CreateVrmlMFRotation([out, retval]  
IVrmlMFRotation** field);`

Creates a new VrmlMFRotation object and returns its IVrmlMFRotation interface. The new field is empty; that is, its value is [].

#### **CreateVrmlMFString**

**Prototype** `CreateVrmlMFString([out, retval]  
IVrmlMFString** field);`

Creates a new VrmlMFString object and returns its IVrmlMFString interface. The new field is empty; that is, its value is [].

#### **CreateVrmlMFTime**

**Prototype** `CreateVrmlMFTime([out, retval] IVrmlMFTime** field);`

Creates a new VrmlMFTime object and returns its IVrmlMFTime interface. The new field is empty; that is, its value is [].

#### **CreateVrmlMFVec2f**

**Prototype** `CreateVrmlMFVec2f([out, retval] IVrmlMFVec2f** field);`

Creates a new VrmlMFVec2f object and returns its IVrmlMFVec2f interface. The new field is empty; that is, its value is [].

#### **CreateVrmlMFVec3f**

**Prototype** `CreateVrmlMFVec3f([out, retval] IVrmlMFVec3f** field);`

Creates a new VrmlMFVec3f object and returns its IVrmlMFVec3f interface. The new field is empty; that is, its value is [].

#### **CreateVrmlSFBool**

**Prototype** `CreateVrmlSFBool([out, retval] IVrmlSFBool** field);`

Creates a new VrmlSFBool object and returns its IVrmlSFBool interface. The value of the new field is FALSE.

**CreateVrmlSFColor**

**Prototype** `CreateVrmlSFColor([out, retval] IVrmlSFColor** field);`

Creates a new VrmlSFColor object and returns its IVrmlSFColor interface. The value of the new field is 0 0 0.

**CreateVrmlSFFloat**

**Prototype** `CreateVrmlSFFloat([out, retval] IVrmlSFFloat** field);`

Creates a new VrmlSFFloat object and returns its IVrmlSFFloat interface. The value of the new field is 0.

**CreateVrmlSFImage**

**Prototype** `CreateVrmlSFImage([out, retval] IVrmlSFImage** field);`

Creates a new VrmlSFImage object and returns its IVrmlSFImage interface. The value of the new field is 0 0 0 (an empty image).

**CreateVrmlSFInt32**

**Prototype** `CreateVrmlSFInt32([out, retval] IVrmlSFInt32** field);`

Creates a new VrmlSFInt32 object and returns its IVrmlSFInt32 interface. The value of the new field is 0.

**CreateVrmlSFNode**

**Prototype** `CreateVrmlSFNode([out, retval] IVrmlSFNode** field);`

Creates a new VrmlSFNode object and returns its IVrmlSFNode interface. The value of the new field is NULL.

**CreateVrmlSFRotation**

**Prototype** `CreateVrmlSFRotation([out, retval]  
IVrmlSFRotation** field);`

Creates a new VrmlSFRotation object and returns its IVrmlSFRotation interface. The value of the new field is 0 0 1 0.

#### **CreateVrmlSFString**

**Prototype** `CreateVrmlSFString([out, retval]  
IVrmlSFString** field);`

Creates a new VrmlSFString object and returns its IVrmlSFString interface. The value of the new field is “ ”, the empty string.

#### **CreateVrmlSFTime**

**Prototype** `CreateVrmlSFTime([out, retval] IVrmlSFTime** field);`

Creates a new VrmlSFTime object and returns its IVrmlSFTime interface. The value of the new field is -1.

#### **CreateVrmlSFVec2f**

**Prototype** `CreateVrmlSFVec2f([out, retval] IVrmlSFVec2f** field);`

Creates a new VrmlSFVec2f object and returns its IVrmlSFVec2f interface. The value of the new field is 0 0.

#### **CreateVrmlSFVec3f**

**Prototype** `CreateVrmlSFVec3f([out, retval] IVrmlSFVec3f** field);`

Creates a new VrmlSFVec3f object and returns its IVrmlSFVec3f interface. The value of the new field is 0 0 0.



## VrmlScriptImplementation Interface

This is a WorldView feature. Standard VRML allows you to use javascript or Java to implement a VrmlScript. VrmlScriptImplementation is an interface which allows any language that supports COM to implement a VrmlScript.

We strongly discourage developers from using any blocking operations, such as putting up dialog boxes, in script nodes, as this will cause crashes.

The OCX must implement the IVrmlScriptImplementation interface. IVrmlScriptImplementation is analogous to the Script abstract base class in the VRML Java Scripting Reference, Section B.9.2.3. The methods in IVrmlScriptImplementation are also described in the VRML 2.0 Specification, Section 4.12.

## IVrmlScriptImplementation Interface

### EventsProcessed

**Prototype** EventsProcessed();

Called after a series of events are received. This method allows a script that does not rely on the order of events received to generate fewer events than a Script which generates events each time an event is received.

#### **Initialize**

**Prototype** Initialize([in] IVrmlScriptNode\* scriptNode);

This method allows the OCX to perform initialization tasks. It is invoked before any events are processed by any node in the VRML file containing the Script node invoking the OCX, including the Script node itself. No other methods of IVrmlScriptImplementation will be invoked before the Initialize method is called.

The scriptNode parameter is an IVrmlScriptNode interface which enables the OCX to obtain information about the eventIns, eventOuts and fields of the Script node in which it is embedded. It is often useful to save the pointer to the IVrmlScriptNode interface for later use. See *IVrmlScriptNode Interface* for more information.

#### **ProcessEvent**

**Prototype** ProcessEvent([in] IVrmlEvent\* anEvent);

Called each time an event is received by one of the eventIns of the Script node. The received event is described by the event parameter. See *IVrmlEvent Interface* for more information.

#### **Shutdown**

**Prototype** Shutdown();

Invoked when the Script node is deleted, or when the world containing the Script node is unloaded or replaced by another world. This enables the OCX to perform any cleanup tasks that it requires, such as freeing memory or closing files. No other methods of IVrmlScriptImplementation will be called after this method is invoked.

# ■ 10

---

## Disabled Interfaces

Describes interfaces which you may be able to see in Visual Basic, but which are disabled, and will return errors.

**Disabled Interfaces** ..... 10-2

## **Disabled Interfaces**

WorldView for Developers includes several interfaces which are disabled and unavailable for use at this time. These interfaces may be visible in your development environment, but are not documented in this manual, and will return errors when queried.

---

# Error Handling

This chapter describes errors returned using WorldView for Developers.

|                              |      |
|------------------------------|------|
| <b>Introduction</b> .....    | 11-2 |
| <b>Returned Errors</b> ..... | 11-2 |

## Introduction

All of the methods in the WorldView for Developers COM API return standard Windows HRESULT error codes. The way these error codes are reported will depend on the language used to access the COM API. In C++, for instance, the return type of every COM API method is HRESULT. In Java, however, an exception of the class `com.ms.com.ComException` containing the HRESULT error code is thrown.

WorldView for Developers includes several interfaces which are disabled and unavailable for use at this time. These interfaces may be visible in your development environment, but are not documented in this manual, and will return errors when queried.

## Returned Errors

If no error occurs during the execution of a method, the return code will be `S_OK`. If an error does occur, the return code will be one of the following values:

---

|  |  |
|--|--|
| <code>VRMLERR_INVALIDEVENTIN</code>      | The specified <code>eventIn</code> does not exist. This is returned by the <code>GetEventIn</code> method of the <code>IVrmlNode</code> and <code>IVrmlScriptNode</code> interfaces. |
| <code>VRMLERR_INVALIDEVENTOUT</code>     | The specified <code>eventOut</code> does not exist. This is returned by the <code>GetEventOut</code> method of the <code>IVrmlScriptNode</code> interface.                           |
| <code>VRMLERR_INVALIDEXPOSEDFIELD</code> | The specified <code>exposedField</code> does not exist. This is returned by the <code>GetExposedField</code> method of the <code>IVrmlNode</code> interface.                         |
| <code>VRMLERR_INVALIDFIELD</code>        | The specified field does not exist. This is returned by the <code>GetField</code> method of the <code>IVrmlScriptNode</code> interface.  |

---

---

|                               |  |
|-------------------------------|--|
| VRMLERR_INVALIDFIELDCHANGE    | <p>This error may result from a variety of invalid field changes, such as:</p> <ul style="list-style-type: none"><li>■ Adding a node from one world as the child of a node in another world.</li><li>■ Creating a circularity in the scene graph.</li><li>■ Setting an invalid string on enumerated fields, such as the fogType field of the Fog node.</li><li>■ Calling the SetValue, AddValue, InsertValue or Delete methods on an IVrmlField interface obtained by the GetEventIn method.</li></ul> |
| VRMLERR_INVALIDROUTE          | <p>The argument is invalid. This is returned when the AddRoute or DeleteRoute methods of the IVrmlBrowser interface are called and one or more of the arguments is invalid.</p>  |
| VRMLERR_INVALIDVRML           | <p>The VRML being parsed contains a syntax error. This is returned by the CreateVrmlFromString, CreateVrmlFromURL, and LoadURL methods of the IVrmlBrowser interface.</p>  |
| VRMLERR_INVALIDVRMLSYNTAX     | <p>The VRML being parsed contains a syntax error. This is returned by the CreateVrmlFromString, CreateVrmlFromURL, and LoadURL methods of the IVrmlBrowser interface.</p>  |
| VRMLERR_INVALIDNODE           | <p>The specified node does not exist. This is returned by the GetNode method of the IVrmlBrowser interface.</p>  |
| VRMLERR_ARRAYINDEXOUTOFBOUNDS | <p>The array index is out of bounds. This is returned when an array index passed into the COM API is out of the bounds of the object being read or modified.</p>   |
| VRMLERR_ILLEGALARGUMENT       | <p>The argument is illegal. This is returned when one or more of the arguments being passed into the COM API are illegal.</p>  |

---

## ■ **Error Handling**

---

### *Returned Errors*





---

# Sample Applications

|   |            |
|---|------------|
| <b>Sample Applications</b> .....  | <b>A-2</b> |
| <b>Invoking an OCX from a Script node: OCXDemo</b> .....                    | <b>A-3</b> |
| Software Requirements .....   | A-3        |
| Installation .....  | A-3        |
| Components .....  | A-3        |
| Instructions .....  | A-4        |
| <b>Embedding WorldView in a C++ Application: Tiny3D</b> .....               | <b>A-4</b> |
| Software Requirements .....   | A-5        |
| Installation .....  | A-5        |
| Components .....  | A-5        |
| Instructions .....  | A-6        |
| <b>Embedding WorldView in a Java Application: JWorldViewContainer</b> ..... | <b>A-7</b> |
| Software Requirements .....   | A-7        |
| Installation .....  | A-7        |
| Components .....  | A-7        |
| Instructions .....  | A-8        |

## Sample Applications

These samples are provided to illustrate how WorldView for Developers may be used to create applications, and to demonstrate some of its features.

---

**Note** • There are delays placed in the sample source code to allow some time for the VRML file to load before trying to get references to the nodes within them. These time delays are arbitrary, and their length will depend on the machines on which they are running. A future release of WorldView for Developers will include a method that will allow the programmer to determine if the VRML file has finished loading within the control.

---

## Invoking an OCX from a Script node: OCXDemo

OCXDemo demonstrates how to use WorldView for Developers to invoke an OCX from a Script node. It displays a scrolling marquee of text in the VRML window. OCXDemo is a small ActiveX Control written in C++ invoked from a VRML Script node using WorldView's clsid: protocol.

### Software Requirements

- WorldView for Developers (to run the program)
- Visual C++ 5.0 or above (to edit or recompile the source code)

### Installation

This application will be installed in the WorldView for Developers Samples folder.

- 1 Register the OCXDemo DLL by typing

```
regsvr32 ocxdemo.dll
```

---

**Note** • The program regsvr32 may not be in your PATH. It is located in your Windows system directory.

---

- 2 In Internet Explorer, open the file OCXDemo.HTML.
- 3 In Visual C++, open the workspace OCXDemo.dsw to examine the source code.

### Components

OCXDemo.WRL : Main VRML file  
OCXDemo.DLL : OCXDemo DLL

### Instructions

When OCXDemo is started, it will display a marquee. Text will scroll across the screen from right to left: new letters are added to the right, while old characters are pushed off to the left side.

The displayed text is read from the file `OCXDEMO.TXT`. To insert text of your own, simply modify this file.

This is not an interactive demo. To exit, simply close the Internet Explorer window.

## Embedding WorldView in a C++ Application: Tiny3D

Tiny3D is an example of a simple 3D authoring tool written on top of WorldView for Developers. Tiny3D is a Windows application, written in C++ and using the Microsoft Foundation Classes (MFC) application framework. It is an ActiveX Container which embeds the WorldView OCX and controls it via WorldView's COM API.

Tiny3D demonstrates how WorldView for Developers enables you to create powerful 3D graphics applications with a small amount of code. Tiny3D creates four types of objects: boxes, cones, cylinders and spheres. It can create multiple objects in a scene, and can independently position, rotate, scale and change the color of objects. While Tiny3D's functionality is fairly "tiny," it is also a tiny program: the executable is a mere 37K.

Tiny3D is based on a Java applet by the same name written by Chris Marrin at Silicon Graphics, Inc. This Java applet communicated with the VRML browser using the Java External Authoring Interface (EAI). However, the applet ran parallel to the VRML browser on the same HTML page, rather than embedding the browser within it, and both had to be run inside a Web browser.

## Software Requirements

- WorldView for Developers (to run the program)
- Visual C++ 5.0 or above (to edit or recompile the source code)

## Installation

This application will be installed in the WorldView for Developers Samples folder.

- 1 Launch `tiny3d.exe` to run the application.
- 2 In Visual C++, open the workspace `tiny3d.dsw` to examine the source code.

## Components

TINY3D.EXE : Tiny3D Executable  
ROOT.WRL : Main VRML file

### Instructions

- 1 Launch Tiny3D. It will open with a blank window and a control panel window.
- 2 Create an object. Select Box, Cone, Sphere or Cylinder from the Type combo box, and press the Create button. The object will appear in the VRML scene and will be listed in the Object List list box. Tiny3D generates a name for the object by appending the number of objects in the scene to the type of object created. For example, you might create Box1, Cone2, then Sphere3.

Once you have created an object, you can control its translation, rotation, scale and color attributes using the scroll bars. The three scroll bars associated with Translation control the object's X, Y and Z translation components. The scroll bars for Scale control the object's X, Y and Z scaling components, and the scroll bars for Color control the object's R (Red), G (Green) and B (Blue) components.

The object's rotation is controlled using three scroll bars labeled X, Y and Z. In Tiny3D, an object's rotation is calculated by rotating about the X axis, then the Y axis, then the Z axis. The X, Y and Z scroll bars control the rotation about each of the axes. (This method of determining a rotation is called Euler angles, and differs from the usual VRML concept of a rotation axis and angle.)

You can create multiple objects and switch between them by selecting the object's name in the Object List list box. The scroll bars will update to show the attributes for the currently selected object. Pressing the Delete Object button will delete the selected object.

## Embedding WorldView in a Java Application: JWorldViewContainer

JWorldViewContainer is a simple example of embedding WorldView for Developers in a Java application with the Microsoft Java VM. This sample also demonstrates how to use the Java EAI in a standalone Java application that embeds WorldView for Developers.

### Software Requirements

- WorldView for Developers (to run the program)
- Microsoft Java VM
- Visual J++ 1.1 or above (to edit or recompile the source code)

### Installation

This application will be installed in the WorldView for Developers Samples folder.

- 1 To run the application, enter:

```
view JWorldViewContainer
```

- 2 In Visual J++, open the file JWorldViewContainer.java to examine the source code.

### Components

```
WorldViewContainer.class : Java class file  
WorldViewContainer.wrl  : Main VRML file
```

## ■ Sample Applications

---

*Embedding WorldView in a Java Application: JWorldViewContainer*

### Instructions

When JWorldViewContainer is started, it will display a Java frame window containing a VRML scene and three buttons labeled Red, Green and Blue. The VRML scene contains a sphere. Pressing the Red, Green or Blue button will change the color of the sphere.





---

# Index

## A

- AboutBox [7-2](#)
- accessing the COM API
  - Microsoft Java VM [4-9](#)
- ActiveX Support [1-4](#)
- adding a VRML primitive
  - Microsoft Visual Basic [4-14](#)
  - Web pages [4-26](#)
- adding the component to a dialog box
  - Microsoft Visual C++ [4-6](#)
- adding the control
  - Microsoft Visual Basic [4-12](#)
  - Microsoft Visual C++ [4-4](#)
- Anchor node [3-4](#)
- AutoRotate [6-4](#)

## B

- BillboardText EXTERNPROTO [3-19](#)
- BrowserSettings EXTERNPROTO [3-20](#)

## C

- C++ [4-4](#)
  - adding the component to a dialog box [4-6](#)
  - adding the control [4-4](#)
  - controlling the component [4-5](#)
  - Runtime applications [5-6](#)
  - using the COM API [8-3](#)
  - using the component in a window [4-5](#)
- changing the fields of a node
  - Microsoft Visual Basic [4-17](#)
  - Web pages [4-28](#)
- Collision node [3-4](#)
- ColorInterpolator node [3-4](#)
- COM (Component Object Model)
  - introduction [8-2](#)
- COM API
  - accessing in Java [4-9](#)
  - using from C++ [8-3](#)
  - using from other languages [8-6](#)
  - using in Java [8-5](#)
- COM API Library [8-6](#)
- COM object [6-2](#)

ConsoleVisible 6-4  
Contacting PLATINUM *xvi*  
containers  
    Web pages 4-22  
controlling the component  
    Microsoft Visual C++ 4-5

## D

DashboardEnabled 6-5  
DashboardOn 6-5  
data protocol 3-11  
DirectionalLight node 3-4  
DirectX files 3-28  
Dithering 6-6

## E

EAI (External Authoring Interface)  
    structure 9-9  
ElevationGrid node 3-4  
EMBED tag 3-18  
embedding at Runtime  
    Microsoft Java VM 4-11  
    Microsoft Visual Basic 4-21  
Embedding Support 1-4  
embedding the control  
    at Runtime 5-6  
    Microsoft Java VM 4-8  
    Web pages 4-23

## Error Handling

    introduction 11-2  
    VRMLERR\_ARRAYINDEXOUTOFBOUNDS 11-3  
    VRMLERR\_ILLEGALARGUMENT 11-3  
    VRMLERR\_INVALIDEVENTIN 11-2  
    VRMLERR\_INVALIDEVENTOUT 11-2  
    VRMLERR\_INVALIDEXPOSEDFIELD 11-2  
    VRMLERR\_INVALIDFIELD 11-2  
    VRMLERR\_INVALIDFIELDCHANGE 11-3  
    VRMLERR\_INVALIDNODE 11-3  
    VRMLERR\_INVALIDROUTE 11-3  
    VRMLERR\_INVALIDVRML 11-3  
    VRMLERR\_INVALIDVRMLSYNTAX 11-3

## Extensions

    BillboardText 3-19  
    BrowserSettings 3-20  
    PopupText 3-23  
    StreamingAudioClip 3-26  
EXTERNPROTO extensions  
    BillboardText 3-19  
    BrowserSettings 3-20  
    Introduction 3-19  
    PopupText 3-23  
    StreamingAudioClip 3-26

## F

Features 1-2  
Fog node 3-5  
FontStyle node 3-5  
Frequently Asked Questions *xiii*  
Full Color setting 3-12  
FullColor 6-6

**G**

- generating Runtime applications 5-2
- GetBrowser 6-3, 6-13
- getting a reference to the control
  - Microsoft Visual Basic 4-13
  - Web pages 4-22
- getting a reference to the VrmlBrowser
  - Object
    - Microsoft Visual Basic 4-14
    - Web pages 4-24
- GraphicsMode 6-6
- GraphicsModeWhenMoving 6-7
- GUID
  - WorldView 6-2
  - WorldView for Developers 6-2

**H**

- HeadlightOn 6-7
- help
  - WorldView browser xx
- Help files
  - including 5-5
- HighQualityText 6-8
- HTML
  - EMBED tag 3-18

**I**

- ImageTexture node 3-6
- IndexedFaceSet node 3-5
- IndexedLineSet node 3-5
- Installing WorldView for Developers 2-2, 2-3
- Internet Explorer
  - Java EAI requirements 3-15
  - using the Java EAI 3-15
- invoking an OCX from the Script node
  - Microsoft Visual Basic 4-20
- IWorldView 6-3
  - Methods 6-3
  - Properties 6-3
- IWorldView interface 6-3
- IWorldView methods
  - GetBrowser 6-3
- IWorldView properties
  - World 6-3
- IWorldViewDeveloper
  - interface 6-4
  - properties 6-4
- IWorldViewDeveloper interface
  - methods 6-13
- IWorldViewDeveloper methods
  - GetBrowser 6-13
  - NextViewpoint 6-13
  - PreviousViewpoint 6-13
  - Reload 6-13
  - RestoreView 6-13
  - StraightenUp 6-14
  - ViewAll 6-14

**IWorldViewDeveloper properties**

- AutoRotate 6-4
- ConsoleVisible 6-4
- DashboardEnabled 6-5
- DashboardOn 6-5
- Dithering 6-6
- FullColor 6-6
- GraphicsMode 6-6
- GraphicsModeWhenMoving 6-7
- HeadlightOn 6-7
- HighQualityText 6-8
- LoadTextures 6-8
- NavigationMode 6-9
- NavigationSpeed 6-10
- PopupMenuEnabled 6-10
- PreventCollisions 6-10
- SplashScreenEnabled 6-11
- UseAcceleration 6-11
- UserHelpFile 6-11
- Viewpoint 6-12
- WebLinkEnabled 6-12
- World 6-12

**J****Java**

- Runtime applications 5-7
- using the COM API 8-5

**Java 1.1 support 3-17****Java EAI**

- requirements for Internet Explorer 3-15
- support 3-15

**Java in Script Nodes**

- Java 1.1 support 3-17
- Security 3-17
- Support 3-17
- System.out and System.err 3-17

**Java VM**

- accessing the COM API 4-9
- embedding at Runtime 4-11
- embedding the control 4-8
- Microsoft Java and ActiveX integration 4-10
- samples 4-11
- using Standard Java EAI 4-10

**JavaScript and the VRML Console 3-13****JavaScript support 3-13**

- Variable Scoping 3-13
- VRML Console 3-13

**L****LoadTextures 6-8****M****Material node 3-6****Methods**

- IWorldView 6-3
- IWorldViewDeveloper 6-13

**Microsoft Component Object Model**

- introduction 8-2

**Microsoft Java and ActiveX integration 4-10****Microsoft Java VM**

- accessing the COM API 4-9
- embedding at Runtime 4-11
- embedding the control 4-8
- Microsoft Java and ActiveX integration 4-10
- using Standard Java EAI 4-10

- Microsoft Visual Basic
  - adding a VRML primitive 4-14
  - adding the control 4-12
  - changing the fields of a node 4-17
  - embedding at Runtime 4-21
  - getting a reference to the control 4-13
  - getting a reference to the VrmlBrowser Object 4-14
  - invoking an OCX from the Script node 4-20
  - passing arrays to methods 4-19
  - positioning and resizing the control 4-13
  - receiving events 4-18
  - removing a node 4-16
  - Runtime applications 5-7
  - samples 4-21
- Microsoft Visual C++
  - adding the component to a dialog box 4-6
  - adding the control 4-4
  - controlling the component 4-5
  - Runtime applications 5-6
  - using the COM API 8-3
  - using the component in a window 4-5
- Microsoft Visual J++
  - Runtime applications 5-7
  - using the COM API 8-5
- MovieTexture node 3-6

**N**

- NavigationInfo node 3-7
- NavigationMode 6-9
- NavigationSpeed 6-10
- Netscape Navigator
  - using the Java EAI 3-15
- NextViewpoint 6-13

**O**

- OCX
  - invoking from the Script node 4-20
- OLE Automation interface 7-2
  - AboutBox 7-2
- on-line help xi, xiii

**P**

- passing arrays to methods
  - Microsoft Visual Basic 4-19
- PixelTexture node 3-7
- WorldView browser help files xx
- PLATINUM, contacting xvi
- PointLight node 3-8
- PointSet node 3-8
- PopupMenuEnabled 6-10
- PopupText EXTERNPROTO 3-23
- positioning and resizing the control
  - Microsoft Visual Basic 4-13
- PreventCollisions 6-10
- PreviousViewpoint 6-13

**R**

- ReadMe file xi, xiii
- receiving events
  - Microsoft Visual Basic 4-18
  - Web pages 4-29
- Reload 6-13
- removing a node
  - Microsoft Visual Basic 4-16
  - Web pages 4-27
- Requirements
  - software 2-2
  - system 2-2
- Requirements for Internet Explorer 3-15
- RestoreView 6-13
- Runtime 2-5

## Runtime applications

- C++ 5-6
- component requirements 2-5
- generating 5-2
- Help files 5-5
- J++ 5-7
- License Agreement 5-5
- License keys 5-6
- Requirements 5-2
- Visual Basic 5-7

**S**

## samples

- embedding in a C++ application (Tiny3D) A-4
- embedding in a Java application (JWorldViewContainer) A-7
- introduction A-2
- invoking an OCX from a Script node (OCXDemo) A-3
- Java 4-11
- Microsoft Visual Basic 4-21

## Script node 3-8

## Security

- Java In Script Nodes 3-17

## Software Requirements 2-2

## Sound node 3-9

## SplashScreenEnabled 6-11

## SpotLight node 3-9

## StraightenUp 6-14

## StreamingAudioClip EXTERNPROTO 3-26

## Support for JavaScript 3-13

- Variable Scoping 3-13

## VRML Console 3-13

## Support for the Java EAI 3-15

## System Requirements 2-2

## System.out and System.err 3-17

**T**

## technical support xi

- by email xii, xv

- by phone xii, xv

- preparing to contact xiv

## telephone support xi

## Text node 3-10

**U**

## Uninstalling WorldView for Developers 2-6

## UseAcceleration 6-11

## UserHelpFile 6-11

## using Standard Java EAI

- Microsoft Java VM 4-10

## using the component in a window

- Microsoft Visual C++ 4-5

## Using the EMBED tag in HTML

- documents 3-18

## Using the Java EAI in Netscape Navigator or Internet Explorer 3-15

**V**

- Variable Scoping 3-13
- ViewAll 6-14
- Viewpoint 6-12
- Viewpoint node 3-10
- Visual Basic
  - adding a VRML primitive 4-14
  - adding the control 4-12
  - changing the fields of a node 4-17
  - embedding at Runtime 4-21
  - getting a reference to the control 4-13
  - getting a reference to the VrmlBrowser Object 4-14
  - invoking an OCX from the Script node 4-20
  - passing arrays to methods 4-19
  - positioning and resizing the control 4-13
  - receiving events 4-18
  - removing a node 4-16
  - Runtime applications 5-7
  - samples 4-21
- VRML 2.0 nodes 3-4
  - Anchor 3-4
  - Collision 3-4
  - ColorInterpolator 3-4
  - DirectionalLight 3-4
  - ElevationGrid 3-4
  - Fog 3-5
  - FontStyle 3-5
  - ImageTexture 3-6
  - IndexedFaceSet 3-5
  - IndexedLineSet 3-5
  - Material 3-6
  - MovieTexture 3-6
  - NavigationInfo 3-7
  - PixelTexture 3-7
  - PointLight 3-8
  - PointSet 3-8
  - Script 3-8
  - Sound 3-9
  - SpotLight 3-9
  - Text 3-10
  - Viewpoint 3-10
- VRML 2.0 support 3-3
- VRML 97 Specification xi, xx
- VrmlBaseNode Objects
  - introduction 9-11
  - VrmlBaseNode 9-12
  - VrmlNode 9-13
  - VrmlScriptNode 9-15
- VrmlBrowser Object
  - introduction 9-17
  - VrmlBrowser 9-17
- VrmlConstField 9-25
- VrmlConstMFCColor 9-26
- VrmlConstMFFloat 9-28
- VrmlConstMField 9-29
- VrmlConstMFIInt32 9-31
- VrmlConstMFNode 9-33
- VrmlConstMFRotation 9-35
- VrmlConstMFString 9-37
- VrmlConstMFTime 9-39
- VrmlConstMFVec2f 9-41
- VrmlConstMFVec3f 9-43
- VrmlConstSFBool 9-45
- VrmlConstSFColor 9-47
- VrmlConstSFFloat 9-49
- VrmlConstSFImage 9-51
- VrmlConstSFInt32 9-53
- VrmlConstSFNode 9-54
- VrmlConstSFRotation 9-56
- VrmlConstSFString 9-58
- VrmlConstSFTime 9-59

- VrmlConstSFVec2f 9-61
- VrmlConstSFVec3f 9-63
- VRMLERR\_ARRAYINDEXOUTOFBOUND  
DS 11-3
- VRMLERR\_ILLEGALARGUMENT 11-3
- VRMLERR\_INVALIDEVENTIN 11-2
- VRMLERR\_INVALIDEVENTOUT 11-2
- VRMLERR\_INVALIDEXPOSEDFIELD 11-  
2
- VRMLERR\_INVALIDFIELD 11-2
- VRMLERR\_INVALIDFIELDCHANGE 11-3
- VRMLERR\_INVALIDNODE 11-3
- VRMLERR\_INVALIDROUTE 11-3
- VRMLERR\_INVALIDVRML 11-3
- VRMLERR\_INVALIDVRMLSYNTAX 11-3
- VrmlEvent Object 9-20
- VrmlEventOutObserver Object 9-21
- VrmlField 9-23
- VrmlField Objects
  - Introduction 9-22
  - VrmlConstField 9-25
  - VrmlConstMFColor 9-26
  - VrmlConstMFFloat 9-28
  - VrmlConstMField 9-29
  - VrmlConstMFInt32 9-31
  - VrmlConstMFNode 9-33
  - VrmlConstMFRotation 9-35
  - VrmlConstMFString 9-37
  - VrmlConstMFTime 9-39
  - VrmlConstMFVec2f 9-41
  - VrmlConstMFVec3f 9-43
  - VrmlConstSFBool 9-45
  - VrmlConstSFColor 9-47
  - VrmlConstSFFloat 9-49
  - VrmlConstSFImage 9-51
  - VrmlConstSFInt32 9-53
  - VrmlConstSFNode 9-54
  - VrmlConstSFRotation 9-56
  - VrmlConstSFString 9-58
  - VrmlConstSFTime 9-59
  - VrmlConstSFVec2f 9-61
  - VrmlConstSFVec3f 9-63
- VrmlField 9-23
- VrmlMFColor 9-65
- VrmlMFFloat 9-68
- VrmlMField 9-71
- VrmlMFInt32 9-73
- VrmlMFNode 9-75
- VrmlMFRotation 9-79
- VrmlMFString 9-82
- VrmlMFTime 9-85
- VrmlMFVec2f 9-88
- VrmlMFVec3f 9-91
- VrmlObjectFactory 9-115
- VrmlSFBool 9-94
- VrmlSFColor 9-96
- VrmlSFFloat 9-98
- VrmlSFImage 9-99
- VrmlSFInt32 9-102
- VrmlSFNode 9-103
- VrmlSFRotation 9-105
- VrmlSFString 9-107
- VrmlSFTime 9-109
- VrmlSFVec2f 9-111
- VrmlSFVec3f 9-113
- VrmlMFColor 9-65
- VrmlMFFloat 9-68
- VrmlMField 9-71
- VrmlMFInt32 9-73
- VrmlMFNode 9-75
- VrmlMFRotation 9-79
- VrmlMFString 9-82
- VrmlMFTime 9-85
- VrmlMFVec2f 9-88



VrmlMFVec3f 9-91  
VrmlNode 9-13  
VrmlObjectFactory 9-115  
VrmlScriptImplementation 9-125  
VrmlScriptNode 9-15  
VrmlSFBool 9-94  
VrmlSFColor 9-96  
VrmlSFFloat 9-98  
VrmlSFImage 9-99  
VrmlSFInt32 9-102  
VrmlSFNode 9-103  
VrmlSFRotation 9-105  
VrmlSFString 9-107  
VrmlSFTime 9-109  
VrmlSFVec2f 9-111  
VrmlSFVec3f 9-113

## W

### Web pages

- adding a VRML primitive 4-26
- changing the fields of a node 4-28
- embedding the control 4-23
- getting a reference to the control 4-22
- getting a reference to the VrmlBrowser  
Object 4-24
- receiving events 4-29
- removing a node 4-27
- working with 4-22

WebLinkEnabled 6-12

### World

- IWorldView 6-3
- IWorldViewDeveloper 6-12

WorldView browser 3-3  
described 1-5

WorldView COM object 6-2

WorldView control features 1-3

WorldView EXTERNPROTO extensions  
3-19

### WorldView for Developers

- described 1-5
- Installation 2-2, 2-3
- uninstalling 2-6

WorldView for Developers runtimes  
described 1-5

WorldView Help Index xi, xiii

WorldView License Agreement 5-5

WorldView OLE Automation interface 7-2

### WorldView Professional

- described 1-5

WorldView's Full Color setting 3-12

